Глава 7 Атака на управление сессиями

Глава 7

Атака на управление сессиями

Механизм управления сессиями — это фундаментальный компонент безопасности в большинстве веб-приложений. Именно он позволяет приложению уникально идентифицировать конкретного пользователя в рамках множества различных запросов и обрабатывать данные, которые оно накапливает о состоянии взаимодействия этого пользователя с приложением. Там, где приложение реализует функциональность входа, управление сессиями имеет особое значение, поскольку именно оно позволяет приложению сохранять уверенность в личности любого пользователя после того запроса, в котором он предоставил свои учетные данные.

Из-за ключевой роли, которую играют механизмы управления сессиями, они являются основной целью для вредоносных атак на приложение. Если злоумышленник сможет взломать управление сессиями приложения, он сможет эффективно обойти его средства контроля аутентификации и выдавать себя за других пользователей (masquerade), не зная их учетных данных. Если злоумышленник таким образом скомпрометирует пользователя-администратора, он сможет завладеть всем приложением.

Как и в механизмах аутентификации, в функциях управления сессиями часто можно обнаружить широкий спектр дефектов. В наиболее уязвимых случаях злоумышленнику достаточно просто инкрементировать значение токена, выданного ему приложением, чтобы переключить свой контекст на контекст другого пользователя. В этой ситуации приложение полностью открыто для доступа любого желающего во все его области. С другой стороны, в самых сложных случаях злоумышленнику, возможно, придется приложить огромные усилия, расшифровывая несколько слоев обфускации и разрабатывая сложную автоматизированную атаку, прежде чем он найдет трещину в броне приложения.

В этой главе рассматриваются все типы уязвимостей, с которыми авторы сталкивались в реальных веб-приложениях. В ней подробно излагаются практические шаги, которые вам необходимо предпринять, чтобы находить и эксплуатировать эти дефекты. Наконец, в ней описываются защитные меры, которые приложения должны предпринимать для защиты от этих атак.

> **РАСПРОСТРАНЕННЫЙ МИФ** *«Мы используем смарт-карты для аутентификации, и сессии пользователей не могут быть скомпрометированы без них».*

Каким бы надежным ни был механизм аутентификации приложения, последующие запросы от пользователей связываются с этой аутентификацией только через итоговую сессию. Если управление сессиями в приложении имеет изъяны, злоумышленник может обойти надежную аутентификацию и все равно скомпрометировать пользователей.

Потребность в состоянии

Протокол HTTP по своей сути не имеет состояния (stateless). Он основан на простой модели «запрос-ответ», в которой каждая пара сообщений представляет собой независимую транзакцию. Сам протокол не содержит механизма для связывания серии запросов, сделанных конкретным пользователем, и отличения их от всех других запросов, получаемых веб-сервером. На заре веба в таком механизме не было необходимости: веб-сайты использовались для публикации статичных HTML-страниц, доступных для просмотра любому желающему. Сегодня все совсем иначе.

Большинство веб-«сайтов» на самом деле являются веб-приложениями. Они позволяют вам регистрироваться и входить в систему. Они дают вам возможность покупать и продавать товары. Они запоминают ваши предпочтения к вашему следующему визиту. Они предоставляют богатый мультимедийный опыт с контентом, создаваемым динамически на основе того, что вы нажимаете и вводите. Для реализации любой из этих функций веб-приложениям необходимо использовать концепцию сессии.

Самое очевидное применение сессий — в приложениях, поддерживающих вход в систему (логин). После ввода имени пользователя и пароля вы можете использовать приложение от имени этого пользователя до тех пор, пока не выйдете из системы или сессия не истечет из-за неактивности. Без сессии пользователю пришлось бы повторно вводить свой пароль на каждой странице приложения. Поэтому, аутентифицировав пользователя один раз, приложение создает для него сессию и рассматривает все запросы, принадлежащие этой сессии, как исходящие от этого пользователя.

Приложения, не имеющие функции входа, также обычно нуждаются в использовании сессий. Многие сайты, продающие товары, не требуют от клиентов создания учетных записей. Однако они позволяют пользователям просматривать каталог, добавлять товары в корзину для покупок, указывать детали доставки и производить оплату. В этом сценарии нет необходимости аутентифицировать личность пользователя: большую часть его визита приложение не знает и не интересуется, кто этот пользователь. Но чтобы вести с ним дела, ему необходимо знать, какая серия полученных запросов исходит от одного и того же пользователя.

Самый простой и до сих пор самый распространенный способ реализации сессий — это выдать каждому пользователю уникальный токен сессии или идентификатор. При каждом последующем запросе к приложению пользователь повторно отправляет этот токен, позволяя приложению определить, к какой последовательности предыдущих запросов относится текущий запрос.

В большинстве случаев приложения используют HTTP-cookie в качестве механизма передачи этих токенов сессий между сервером и клиентом. Первый ответ сервера новому клиенту содержит HTTP-заголовок следующего вида:

Set-Cookie: ASP.NET_SessionId=mza2ji454s04cwbgwb2ttj55

а последующие запросы от клиента содержат такой заголовок:

Cookie: ASP.NET_SessionId=mza2ji454s04cwbgwb2ttj55

Этот стандартный механизм управления сессиями по своей сути уязвим для различных категорий атак. Основная цель злоумышленника, атакующего этот механизм, — каким-либо образом перехватить сессию (hijack the session) легитимного пользователя и таким образом выдавать себя (masquerade) за него. Если пользователь был аутентифицирован в приложении, злоумышленник может получить доступ к личным данным, принадлежащим этому пользователю, или выполнить несанкционированные действия от его имени. Даже если пользователь не аутентифицирован, злоумышленник все равно может просматривать конфиденциальную информацию, отправленную пользователем во время его сессии.

Как и в предыдущем примере с сервером Microsoft IIS под управлением ASP.NET, большинство коммерческих веб-серверов и платформ для веб-приложений реализуют свое собственное готовое («коробочное») решение для управления сессиями на основе HTTP-cookie. Они предоставляют API, которые разработчики веб-приложений могут использовать для интеграции своей собственной, зависящей от сессии, функциональности с этим решением.

Было обнаружено, что некоторые готовые («коробочные») реализации управления сессиями уязвимы для различных атак, что приводит к компрометации сессий пользователей (это обсуждается далее в этой главе). Кроме того, некоторые разработчики обнаруживают, что им нужен более тонкий (детальный) контроль над поведением сессий, чем тот, что предоставляют встроенные решения, или они хотят избежать некоторых уязвимостей, присущих решениям на основе cookie. По этим причинам довольно часто можно встретить самописные (bespoke) и/или не основанные на cookie механизмы управления сессиями, используемые в критичных к безопасности приложениях, таких как онлайн-банкинг.

Уязвимости, существующие в механизмах управления сессиями, в основном делятся на две категории:

- Уязвимости в генерации токенов сессий
- Уязвимости в обработке токенов сессий на протяжении всего их жизненного цикла

Мы поочередно рассмотрим каждую из этих областей, описывая различные типы дефектов, которые часто встречаются в реальных механизмах управления сессиями, и практические методики для их обнаружения и эксплуатации. Наконец, мы опишем меры, которые приложения могут предпринять для защиты от этих атак.

ПРАКТИЧЕСКИЕ ШАГИ

Во многих приложениях, использующих стандартный механизм cookie для передачи токенов сессий, определить, какой именно элемент данных содержит токен, довольно просто. Однако в других случаях это может потребовать некоторой детективной работы:

1. Приложение может часто использовать несколько различных элементов данных совместно в качестве токена, включая cookie, параметры URL и скрытые поля формы. Некоторые из этих элементов могут использоваться для поддержания состояния сессии в разных бэкенд-компонентах. Не предполагайте, что определенный параметр является токеном сессии, не доказав этого, или что сессии отслеживаются с помощью только одного элемента.

2. Иногда элементы, которые выглядят как токен сессии приложения, таковыми не являются. В частности, стандартный cookie сессии, генерируемый веб-сервером или платформой приложения, может присутствовать, но фактически не использоваться приложением.

3. Наблюдайте, какие новые элементы передаются в браузер после аутентификации. Часто новые токены сессий создаются после того, как пользователь аутентифицирует себя.

4. Чтобы проверить, какие элементы действительно используются в качестве токенов, найдите страницу, которая определенно зависит от сессии (например, страница «мои данные», специфичная для пользователя). Сделайте к ней несколько запросов, систематически удаляя каждый элемент, который, по вашему подозрению, используется в качестве токена. Если удаление элемента приводит к тому, что зависимая от сессии страница не возвращается, это может подтвердить, что данный элемент является токеном сессии. Burp Repeater — полезный инструмент для выполнения этих тестов.

Альтернативы сессиям

Не каждое веб-приложение использует сессии, и некоторые критичные к безопасности приложения, содержащие механизмы аутентификации и сложную функциональность, выбирают другие методики для управления состоянием. Вы, скорее всего, столкнетесь с двумя возможными альтернативами:

- **HTTP-аутентификация** — Приложения, использующие различные технологии аутентификации на основе HTTP (basic, digest, NTLM), иногда избегают необходимости использовать сессии. При HTTP-аутентификации клиентский компонент взаимодействует с механизмом аутентификации напрямую через браузер, используя HTTP-заголовки, а не через специфичный для приложения код

на отдельной странице. После того как пользователь вводит свои учетные данные в диалоговое окно браузера, браузер фактически повторно отправляет эти учетные данные (или заново выполняет необходимое «рукопожатие») с каждым последующим запросом к тому же серверу. Это эквивалентно приложению, которое использует аутентификацию на основе HTML-форм и размещает форму входа на каждой странице, требуя от пользователей повторно аутентифицироваться при каждом действии. Таким образом, при использовании HTTP-аутентификации приложение может повторно идентифицировать пользователя в нескольких запросах без использования сессий. Однако HTTP-аутентификация редко используется в интернет-приложениях любой сложности, а другие разнообразные преимущества, которые предлагают полноценные сессионные механизмы, означают, что практически все веб-приложения на самом деле их и используют.

- **Бессессионные механизмы состояния** — Некоторые приложения не выдают токены сессий для управления состоянием взаимодействия пользователя с приложением. Вместо этого они передают все необходимые для управления этим состоянием данные через клиент, обычно в cookie или скрытом поле формы. По сути, этот механизм использует бессессионное состояние во многом так же, как это делает ViewState в ASP.NET. Чтобы этот тип механизма был безопасным, данные, передаваемые через клиент, должны быть должным образом защищены. Обычно это включает в себя создание двоичного блока данных (binary blob), содержащего всю информацию о состоянии, и его шифрование или подписание с использованием признанного алгоритма. Достаточный контекст должен быть включен в данные, чтобы злоумышленник не мог собрать объект состояния в одном месте приложения и отправить его в другое место, вызвав нежелательное поведение. Приложение также может включать время истечения срока действия в данные объекта для реализации аналога тайм-аутов сессии. В Главе 5 более подробно описаны безопасные механизмы передачи данных через клиент.

ПРАКТИЧЕСКИЕ ШАГИ

1. Если используется HTTP-аутентификация, возможно, что механизм управления сессиями не реализован. Используйте описанные ранее методы, чтобы изучить роль, которую играют любые токеноподобные элементы данных.
2. Если приложение использует бессессионный механизм состояния, передавая все необходимые для поддержания состояния данные через клиент, иногда это может быть трудно определить с уверенностью, но следующие признаки являются вескими индикаторами использования такого механизма:
    - Токеноподобные элементы данных, выдаваемые клиенту, довольно длинные (100 байт или более).
    - Приложение выдает новый токеноподобный элемент в ответ на

каждый запрос.

- Данные в элементе выглядят зашифрованными (и поэтому не имеют различимой структуры) или подписанными (и поэтому имеют осмысленную структуру, сопровождаемую несколькими байтами бессмысленных двоичных данных).

- Приложение может отклонять попытки отправить один и тот же элемент с более чем одним запросом.

3. Если доказательства убедительно свидетельствуют о том, что приложение не использует токены сессий для управления состоянием, маловероятно, что какие-либо из атак, описанных в этой главе, к чему-либо приведут. Ваше время, вероятно, будет лучше потрачено на поиск других серьезных проблем, таких как нарушения контроля доступа или **внедрение кода** (code injection).

Уязвимости в генерации токенов

Механизмы управления сессиями часто уязвимы для атак, потому что токены генерируются небезопасным образом, что позволяет злоумышленнику определять значения токенов, которые были выданы другим пользователям.

> **ПРИМЕЧАНИЕ** *Существует множество мест, где безопасность приложения зависит от непредсказуемости генерируемых им токенов. Вот несколько примеров:*
>
> - *Токены для восстановления пароля, отправляемые на зарегистрированный адрес электронной почты пользователя.*
> - *Токены в скрытых полях формы для предотвращения атак межсайтовой подделки запроса (CSRF) (см. Главу 13).*
> - *Токены для предоставления одноразового доступа к защищенным ресурсам.*
> - *Постоянные токены (persistent tokens), используемые в функциях «Запомнить меня».*
> - *Токены, позволяющие клиентам приложения для покупок, которое не использует аутентификацию, получать текущий статус существующего заказа.*
>
> *Рассмотрения в этой главе, касающиеся уязвимостей в генерации токенов, применимы ко всем этим случаям. Фактически, поскольку многие современные приложения полагаются на зрелые платформенные механизмы для генерации токенов сессий, именно в этих других областях функциональности часто и обнаруживаются эксплуатируемые уязвимости в генерации токенов.*

Осмысленные токены

Некоторые токены сессий создаются путем преобразования имени пользователя, его адреса электронной почты или другой связанной с ним информации. Эта информация может быть каким-либо образом закодирована или обфусцирована (запутана) и объединена с другими данными.

Например, следующий токен поначалу может показаться длинной случайной строкой:

757365723d6461663b6170703d61646d696e3b646174653d30312f31322f3131

Однако при ближайшем рассмотрении видно, что он содержит только шестнадцатеричные символы. Предположив, что эта строка на самом деле является шестнадцатеричным представлением строки символов ASCII, ее можно пропустить через декодер и получить следующее:

user=daf;app=admin;date=10/09/1

Злоумышленники могут проэксплуатировать значимую информацию внутри этого токена сессии, чтобы попытаться угадать текущие сессии других пользователей приложения. Используя список перечисленных или распространенных имен пользователей, они могут быстро сгенерировать большое количество потенциально действительных токенов и проверить их, чтобы подтвердить, какие из них являются валидными.

Токены, содержащие значимые данные, часто имеют структуру. Другими словами, они содержат несколько компонентов, часто разделенных разделителем, которые можно извлечь и проанализировать по отдельности, чтобы злоумышленник мог понять их назначение и способ генерации. Вот некоторые компоненты, которые могут встречаться в структурированных токенах:

- Имя пользователя учетной записи.
- Числовой идентификатор, который приложение использует для различения учетных записей.
- Имя и фамилия пользователя.
- Адрес электронной почты пользователя.
- Группа или роль пользователя в приложении.
- Отметка даты и времени.
- Инкрементное или предсказуемое число.
- IP-адрес клиента.

Каждый отдельный компонент в структурированном токене, или даже весь токен целиком, может быть закодирован различными способами. Это может быть преднамеренной мерой для обфускации их содержимого, или это может просто обеспечивать безопасную передачу двоичных данных по HTTP. Часто встречающиеся схемы кодирования включают XOR, Base64 и шестнадцатеричное представление с использованием символов ASCII (см. Главу 3). Может потребоваться протестировать различные способы декодирования для каждого компонента структурированного токена, чтобы распаковать его до исходной формы.

ПРИМЕЧАНИЕ *Когда приложение обрабатывает запрос, содержащий структурированный токен, оно может на самом деле не обрабатывать каждый компонент токена или все данные, содержащиеся в каждом компоненте. В предыдущем примере приложение может декодировать токен из Base64, а затем обработать только компоненты «user» и «date». В случаях, когда токен содержит блок двоичных данных, большая часть этих данных может быть* ***заполнителем*** *(padding). Только небольшая его часть может быть действительно важна для проверки, которую сервер выполняет над токеном. Сужение круга фактически необходимых частей токена часто может значительно уменьшить кажущуюся энтропию и сложность, которые содержит токен.*

ПРАКТИЧЕСКИЕ ШАГИ

1. Получите один токен от приложения и систематически изменяйте его, чтобы определить, проверяется ли весь токен целиком или некоторые его подкомпоненты игнорируются. Попробуйте изменять значение токена по одному байту за раз (или даже по одному биту за раз) и повторно отправлять измененный токен, чтобы определить, будет ли он по-прежнему принят. Если вы обнаружите, что определенные части токена на самом деле не обязаны быть корректными, вы можете исключить их из дальнейшего анализа, потенциально сократив объем необходимой работы. Вы можете использовать тип полезной нагрузки «char frobber» (посимвольный перебор) в Burp Intruder, чтобы изменять значение токена по одной символьной позиции за раз, что поможет в этой задаче.

2. Войдите в систему под несколькими разными пользователями в разное время и записывайте токены, полученные от сервера. Если доступна самостоятельная регистрация и вы можете выбирать имя пользователя, войдите в систему с серией похожих имен пользователей с небольшими вариациями между ними, такими как A, AA, AAA, AAAA, AAAB, AAAC, AABA и так далее. Если при входе отправляются или хранятся в профилях другие специфичные для пользователя данные (например, адрес электронной почты), проделайте аналогичное упражнение, чтобы систематически варьировать эти данные, и записывайте полученные после входа токены.

3. Проанализируйте токены на предмет любых корреляций, которые, по-видимому, связаны с именем пользователя и другими контролируемыми пользователем данными.

4. Проанализируйте токены на предмет любого обнаружимого кодирования или обфускации. Если имя пользователя содержит последовательность одинаковых символов, ищите соответствующую последовательность символов в токене, что может указывать на использование обфускации с помощью XOR. Ищите в токене последовательности, содержащие только шестнадцатеричные символы, что может указывать на шестнадцатеричное кодирование строки ASCII или другой информации. Ищите

последовательности, которые заканчиваются знаком равенства и/или содержат только другие допустимые символы Base64: от a до z, от A до Z, от 0 до 9, + и /.

5. Если какой-либо смысл удалось восстановить (реверс-инжиниринг) из выборки токенов сессий, оцените, достаточно ли у вас информации, чтобы попытаться угадать токены, недавно выданные другим пользователям приложения. Найдите страницу приложения, которая зависит от сессии, например, ту, что возвращает сообщение об ошибке или перенаправляет в другое место при доступе без действительной сессии. Затем используйте такой инструмент, как Burp Intruder, чтобы сделать большое количество запросов к этой странице, используя угаданные токены. Отслеживайте результаты на предмет любых случаев, когда страница загружается корректно, что указывает на действительный токен сессии.

ПОПРОБУЙТЕ!

http://mdsec.net/auth/321/
http://mdsec.net/auth/329/
http://mdsec.net/auth/331/

Предсказуемые токены

Некоторые токены сессий не содержат никаких осмысленных данных, связывающих их с конкретным пользователем. Тем не менее, их можно угадать, потому что они содержат последовательности или шаблоны, которые позволяют злоумышленнику, экстраполируя на основе выборки токенов, находить другие действительные токены, недавно выданные приложением. Даже если экстраполяция включает в себя некоторый метод проб и ошибок (например, одно верное предположение на 1000 попыток), это все равно позволит автоматизированной атаке выявить большое количество действительных токенов за относительно короткий промежуток времени.

Уязвимости, связанные с предсказуемой генерацией токенов, могут быть гораздо легче обнаружены в коммерческих реализациях управления сессиями, таких как веб-серверы или платформы для веб-приложений, чем в самописных приложениях. Когда вы удаленно атакуете самописный механизм управления сессиями, ваша выборка выданных токенов может быть ограничена производительностью сервера, активностью других пользователей, вашей пропускной способностью, сетевыми задержками и так далее. Однако в лабораторных условиях вы можете быстро создать миллионы образцов токенов,

все они будут точно упорядочены и снабжены временными метками, и вы сможете исключить помехи, вызванные другими пользователями.

В самых простых и вопиюще уязвимых случаях приложение может использовать простое последовательное число в качестве токена сессии. В этом случае вам нужно получить всего лишь два или три токена, прежде чем запустить атаку, которая быстро захватит 100% всех действующих в данный момент сессий.

На Рисунке 7-1 показано, как Burp Intruder используется для перебора последних двух цифр последовательного токена сессии, чтобы найти значения, где сессия все еще активна и может быть перехвачена. Здесь длина ответа сервера является надежным индикатором того, что найдена действительная сессия. Также была использована функция «extract grep» для отображения имени вошедшего в систему пользователя для каждой сессии.

В других случаях токены приложения могут содержать более сложные последовательности, для обнаружения которых требуются определенные усилия. Типы потенциальных вариаций, с которыми вы можете здесь столкнуться, безграничны, но опыт авторов в этой области показывает, что предсказуемые токены сессий обычно возникают из трех различных источников:

- Скрытые последовательности
- Зависимость от времени
- Слабая генерация случайных чисел

Мы поочередно рассмотрим каждую из этих областей.

Скрытые последовательности

Часто встречаются токены сессий, которые трудно предсказать при анализе в их необработанном (сыром) виде, но которые содержат последовательности, проявляющиеся, когда токены соответствующим образом декодируются или распаковываются.

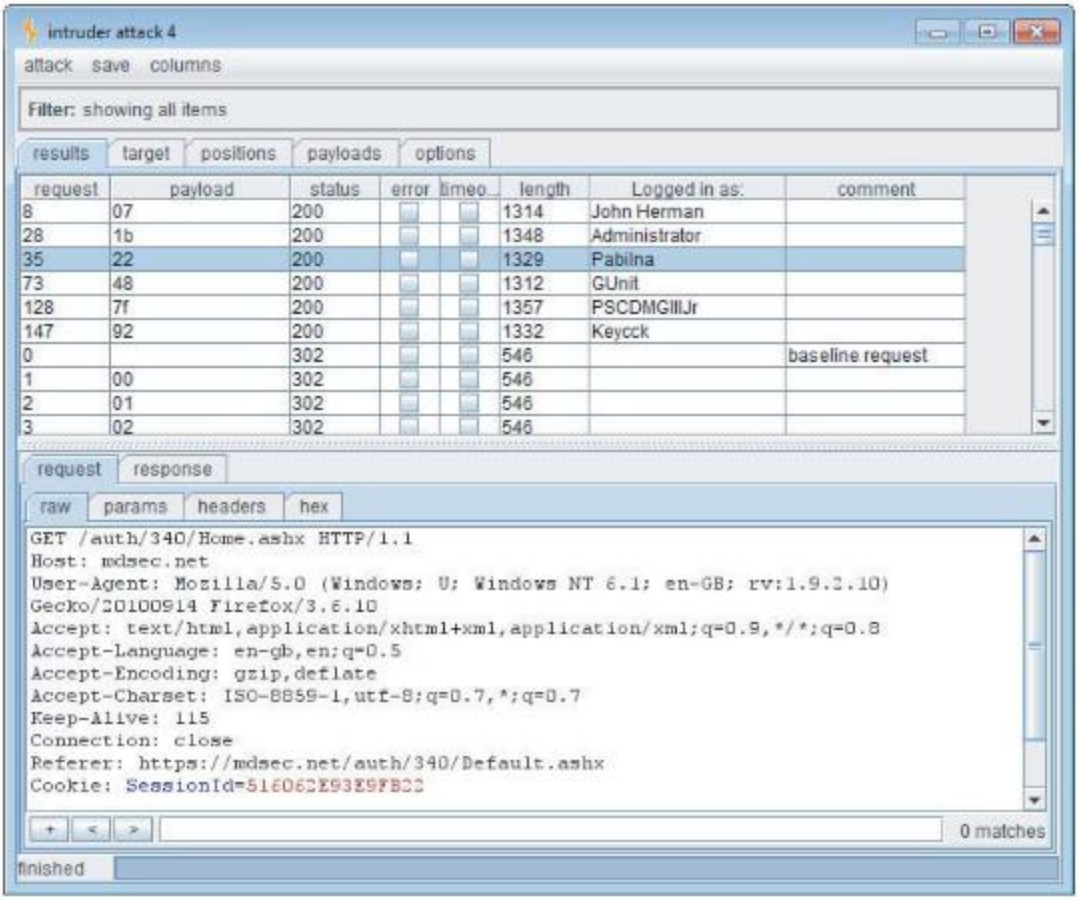


Рисунок 7-1: Атака для обнаружения действительных сессий в случае, когда токен сессии предсказуем

Рассмотрим следующую серию значений, которая является одним из компонентов структурированного токена сессии:

lwjVJA
Ls3Ajg
xpKr+A
XleXYg
9hyCzA
jeFuNg
JaZZoA

Сразу никакого шаблона не видно; однако беглый осмотр показывает, что токены могут содержать данные в кодировке Base64. Помимо буквенных символов в разном регистре и цифр, присутствует символ +, который также является допустимым в строке Base64.

Пропустив токены через декодер Base64, мы получим следующее:

--Õ$
.ÍÀŽ
Æ'«ø
^W-b
ö,Ì
?án6
%¦Y

Эти строки выглядят как бессмыслица (gibberish), а также содержат непечатаемые символы. Обычно это указывает на то, что вы имеете дело с двоичными данными (binary data), а не с текстом ASCII. Отображение декодированных данных в виде шестнадцатеричных чисел дает следующее:

9708D524
2ECDC08E
C692ABF8
5E579762
F61C82CC
8DE16E36
25A659A0

Видимого шаблона по-прежнему нет. Однако, если вычесть каждое число из предыдущего, вы получите следующее:

FF97C4EB6A
97C4EB6A
FF97C4EB6A
97C4EB6A
FF97C4EB6A
FF97C4EB6A

что немедленно раскрывает скрытый шаблон. Алгоритм, используемый для генерации токенов, добавляет 0x97C4EB6A к предыдущему значению, обрезает результат до 32-битного числа и кодирует в Base64 эти двоичные данные, чтобы их можно было передавать с помощью текстового протокола HTTP. Используя это знание, вы можете легко написать скрипт для воспроизведения серии токенов, которые сервер сгенерирует в будущем, и серии, которую он сгенерировал до захваченной выборки.

Зависимость от времени

Некоторые веб-серверы и приложения используют алгоритмы для генерации токенов сессий, которые используют время генерации в качестве входных данных для значения токена. Если в алгоритм заложено недостаточно другой энтропии (случайности), вы, возможно, сможете предсказать токены других пользователей. Хотя любая отдельно

взятая последовательность токенов может показаться случайной, та же последовательность в сочетании с информацией о времени генерации каждого токена может содержать различимый шаблон. В загруженном приложении, где каждую секунду создается большое количество сессий, скриптовая атака может успешно выявить большое количество токенов других пользователей.

При тестировании веб-приложения одного онлайн-ритейлера авторы столкнулись со следующей последовательностью токенов сессий:

3124538-1172764258718
3124539-1172764259062
3124540-1172764259281
3124541-1172764259734
3124542-1172764260046
3124543-1172764260156
3124544-1172764260296
3124545-1172764260421
3124546-1172764260812
3124547-1172764260890

Каждый токен явно состоит из двух отдельных числовых компонентов. Первое число следует простой возрастающей последовательности, и его легко предсказать. Второе число каждый раз увеличивается на переменную величину. Вычисление разницы между его значениями в каждом последующем токене показывает следующее:

344
219
453
312
110
140
125
391
78

Эта последовательность, по-видимому, не содержит надежно предсказуемого шаблона. Однако, очевидно, было бы возможно подобрать перебором (брутфорсом) соответствующий диапазон чисел в ходе автоматизированной атаки, чтобы обнаружить действительные значения в этой последовательности. Прежде чем, однако, приступать к этой атаке, мы подождем несколько минут и соберем еще одну последовательность токенов:

3124553-1172764800468
3124554-1172764800609
3124555-1172764801109

3124556-1172764801406
3124557-1172764801703
3124558-1172764802125
3124559-1172764802500
3124560-1172764802656
3124561-1172764803125
3124562-1172764803562

При сравнении этой второй последовательности токенов с первой сразу становятся очевидны два момента:

- Первая числовая последовательность продолжает инкрементно увеличиваться; однако пять значений были пропущены с момента окончания первой последовательности. Предположительно, это связано с тем, что пропущенные значения были выданы другим пользователям, которые входили в приложение в промежутке между двумя тестами.

- Вторая числовая последовательность продолжает увеличиваться на похожие интервалы, как и раньше; однако первое полученное нами значение на огромную величину в 539 578 больше предыдущего значения.

Это второе наблюдение немедленно указывает нам на роль, которую играет время в генерации токенов сессий. Наиболее вероятное объяснение заключается в том, что второе число зависит от времени и, вероятно, является простым счетчиком миллисекунд.

Действительно, наша догадка верна. На последующем этапе нашего тестирования мы проводим анализ кода, который раскрывает следующий алгоритм генерации токенов:

```
String sessId = Integer.toString(s_SessionIndex++) + "-" + System.currentTimeMillis();
```

Учитывая наш анализ того, как создаются токены, несложно построить скриптовую атаку для сбора токенов сессий, которые приложение выдает другим пользователям:

- Мы постоянно опрашиваем (поллим) сервер, чтобы получать новые токены сессий подряд за короткое время.

- Мы отслеживаем приращения в первом числе. Когда оно увеличивается более чем на 1, мы знаем, что токен был выдан другому пользователю.

- Когда токен выдан другому пользователю, мы знаем верхнюю и нижнюю границы второго числа, которое было ему выдано, поскольку у нас есть токены, выданные непосредственно до и после него. Так как мы часто получаем новые токены сессий, диапазон между этими границами обычно будет состоять всего из нескольких сотен значений.

- Каждый раз, когда токен выдается другому пользователю, мы запускаем атаку полного перебора (брутфорс) для итерации по каждому числу в этом диапазоне, добавляя его к пропущенному инкрементному числу, которое, как мы знаем, было выдано другому пользователю. Мы пытаемся получить доступ к защищенной странице, используя каждый созданный нами токен, пока попытка не увенчается успехом и мы не скомпрометируем сессию пользователя.

- Постоянный запуск этой скриптовой атаки позволит нам перехватить токен сессии каждого другого пользователя приложения. Когда войдет администратор, мы полностью скомпрометируем все приложение.

ПОПРОБУЙТЕ!

http://mdsec.net/auth/339/
http://mdsec.net/auth/340/
http://mdsec.net/auth/347/
http://mdsec.net/auth/351/

Слабая генерация случайных чисел

Очень немногие процессы внутри компьютера являются по-настоящему случайными. Поэтому, когда для какой-либо цели требуется случайность, программное обеспечение использует различные методики для генерации чисел псевдослучайным образом. Некоторые из используемых алгоритмов создают последовательности, которые кажутся стохастическими (случайными) и демонстрируют равномерное распределение по всему диапазону возможных значений. Тем не менее, они могут быть экстраполированы вперед или назад с идеальной точностью любым, кто получит небольшой образец значений.

Когда для создания токенов сессий используется предсказуемый **генератор псевдослучайных чисел** (ГПСЧ), итоговые токены уязвимы для восстановления последовательности злоумышленником. Jetty — это популярный веб-сервер, на 100% написанный на Java, который предоставляет механизм управления сессиями для работающих на нем приложений. В 2006 году Крис Энли (Chris Anley) из NGSSoftware обнаружил, что этот механизм был уязвим для атаки по предсказанию токенов сессий. Сервер использовал Java API java.util.Random для генерации токенов. Он реализует «линейный конгруэнтный генератор» («linear congruential generator»), который генерирует следующее число в последовательности следующим образом:

```
synchronized protected int next(int bits) {
    seed = (seed * 0x5DEECE66DL + 0xBL) & ((1L << 48) - 1);
    return (int)(seed >>> (48 - bits));
}
```

Этот алгоритм берет последнее сгенерированное число, умножает его на константу и добавляет другую константу, чтобы получить следующее число. Число обрезается до 48 бит, и алгоритм сдвигает результат, чтобы вернуть определенное количество бит, запрошенное вызывающей стороной.

Зная этот алгоритм и одно-единственное сгенерированное им число, мы можем легко вывести последовательность чисел, которую алгоритм сгенерирует в будущем. Применив немного теории чисел, мы также можем вывести последовательность, которую он сгенерировал ранее. Это означает, что злоумышленник, получивший от сервера один-единственный токен сессии, может получить токены всех текущих и будущих сессий.

> **ПРИМЕЧАНИЕ** *Иногда, когда токены создаются на основе вывода генератора псевдослучайных чисел, разработчики решают конструировать каждый токен путем объединения (конкатенации) нескольких последовательных выводов из генератора. Предполагаемое обоснование этого — создание более длинного, а следовательно, «более стойкого» токена. Однако эта тактика обычно является ошибкой. Если злоумышленник сможет получить несколько последовательных выводов из генератора, это может позволить ему получить некоторую информацию о его внутреннем состоянии. Фактически, злоумышленнику может быть даже легче экстраполировать последовательность вывода генератора, как вперед, так и назад.*

Другие готовые фреймворки приложений используют удивительно простые или предсказуемые источники энтропии при генерации токенов сессий, большая часть которых детерминирована. Например, в фреймворках PHP версии 5.3.2 и ранее токен сессии генерируется на основе IP-адреса клиента, Unix-время (epoch time) в момент создания токена, микросекунд в момент создания и линейного конгруэнтного генератора. Хотя здесь есть несколько неизвестных значений, некоторые приложения могут раскрывать информацию, позволяющую их предположить. Сайт социальной сети может раскрывать время входа и IP-адрес пользователей. Кроме того, начальное значение (seed), используемое в этом генераторе, — это время запуска процесса PHP, которое злоумышленник, наблюдающий за сервером, может определить в рамках небольшого диапазона.

> **ПРИМЕЧАНИЕ** *Это постоянно развивающаяся область исследований. На уязвимости в генерации токенов сессий PHP было указано в списке рассылки Full Disclosure в 2001 году, но не было продемонстрировано, что они действительно эксплуатируемы. Теория 2001 года была наконец реализована на практике Сэми Камкаром (Samy Kamkar) с помощью инструмента phpwn в 2010 году.*

Тестирование качества случайности

В некоторых случаях вы можете выявить шаблоны в серии токенов путем визуального осмотра или с помощью небольшого ручного анализа. Однако, в общем, вам необходимо использовать более строгий подход к тестированию качества случайности в токенах приложения.

Стандартный подход к этой задаче применяет принципы статистической проверки гипотез и использует различные хорошо документированные тесты, которые ищут признаки неслучайности в выборке токенов. Высокоуровневые шаги этого процесса следующие:

1. Начните с гипотезы, что токены генерируются случайным образом.

2. Примените серию тестов, каждый из которых наблюдает за специфическими свойствами выборки, которые, вероятно, будут иметь определенные характеристики, если токены генерируются случайно.

3. Для каждого теста вычислите вероятность появления наблюдаемых характеристик, исходя из предположения, что гипотеза верна.

4. Если эта вероятность падает ниже определенного уровня («уровня значимости», the «significance level»), отвергните гипотезу и сделайте вывод, что токены не являются случайно сгенерированными.

Хорошая новость в том, что вам не нужно делать все это вручную! Лучший инструмент, доступный на данный момент для тестирования случайности токенов веб-приложений, — это Burp Sequencer. Этот инструмент гибко применяет несколько стандартных тестов и выдает понятные результаты, которые легко интерпретировать.

Чтобы использовать Burp Sequencer, вам нужно найти ответ от приложения, который выдает тестируемый токен, например, ответ на запрос входа, который устанавливает новый cookie с токеном сессии. Выберите опцию «send to sequencer» (отправить в Sequencer) из контекстного меню Burp и в конфигурации Sequencer укажите расположение токена в ответе, как показано на Рисунке 7-2. Вы также можете настроить различные опции, влияющие на сбор токенов, а затем нажать кнопку начала сбора, чтобы начать их захват. Если вы уже получили подходящую выборку токенов другим способом (например, сохранив результаты атаки в Burp Intruder), вы можете использовать вкладку ручной загрузки (manual load), чтобы пропустить сбор токенов и перейти непосредственно к статистическому анализу.

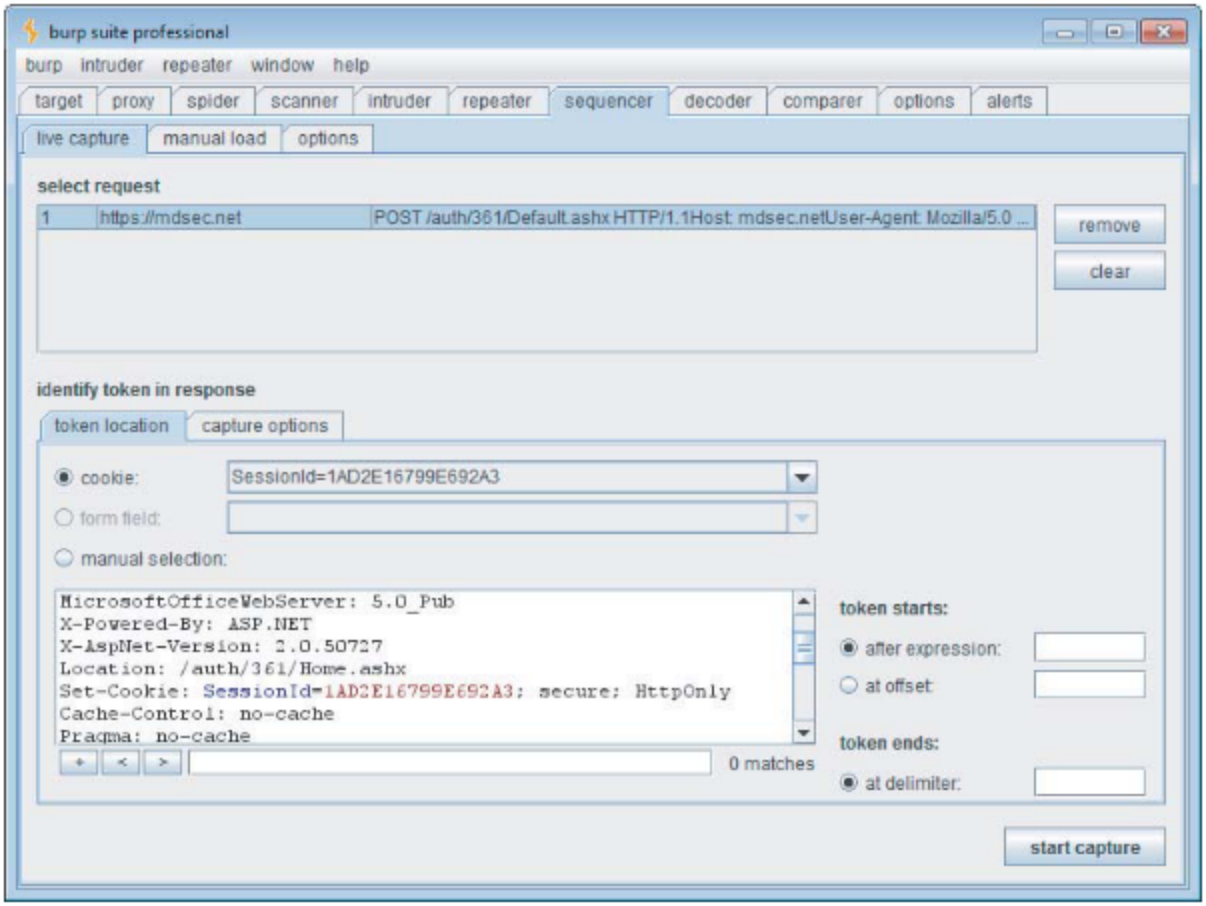


Рисунок 7-2: Настройка Burp Sequencer для тестирования случайности токена сессии

Когда вы получили подходящую выборку токенов, вы можете выполнить статистический анализ этой выборки. Вы также можете выполнять промежуточный анализ, пока выборка еще собирается. В целом, получение большей выборки повышает надежность анализа. Минимальный размер выборки, требуемый Burp, — 100 токенов, но в идеале вам следует получить выборку гораздо большего размера. Если анализ нескольких сотен токенов убедительно показывает, что токены не проходят тесты на случайность, вы можете резонно решить, что дальнейший сбор токенов не требуется. В противном случае вам следует продолжать сбор токенов и периодически повторно выполнять анализ. Если вы соберете 5 000 токенов, которые пройдут тесты на случайность, вы можете решить, что этого достаточно. Однако для соответствия формальным тестам FIPS на случайность вам необходимо получить выборку в 20 000 токенов. Это самый большой размер выборки, который поддерживает Burp.

Burp Sequencer выполняет статистические тесты на уровне символов и на уровне битов. Результаты всех тестов объединяются, чтобы дать общую оценку количества бит эффективной энтропии в токене; это и есть ключевой результат, который следует рассматривать. Однако вы также можете углубиться в результаты каждого теста, чтобы точно понять, как и почему различные части токена прошли или не прошли каждый тест,

как показано на Рисунке 7-3. Методология, используемая для каждого типа теста, описана под результатами теста.

Рисунок 7-3: Анализ результатов Burp Sequencer для понимания свойств протестированных токенов

Обратите внимание, что Burp выполняет все тесты индивидуально для каждого символа и каждого бита данных внутри токена. Во многих случаях вы обнаружите, что большие части структурированного токена не являются случайными; само по себе это может не представлять никакой уязвимости. Что имеет значение, так это чтобы токен содержал достаточное количество бит, которые действительно проходят тесты на случайность.

Например, если большой токен содержит 1000 бит информации, и только 50 из этих бит проходят тесты на случайность, токен в целом является не менее надежным, чем 50-битный токен, который полностью проходит тесты.

> **ПРИМЕЧАНИЕ** *При выполнении статистических тестов на случайность следует помнить о двух важных оговорках (caveats). Эти оговорки влияют на правильную интерпретацию результатов тестов и их последствия для общего уровня безопасности приложения. Во-первых, токены, сгенерированные полностью детерминированным способом, могут проходить статистические тесты на случайность. Например, линейный конгруэнтный генератор псевдослучайных чисел или алгоритм, вычисляющий хэш от последовательного числа, могут производить вывод, который проходит тесты. Тем не менее, злоумышленник, знающий алгоритм и внутреннее состояние генератора, может экстраполировать его вывод с полной надежностью как вперед, так и назад.*
>
> *Во-вторых, токены, которые не проходят статистические тесты на случайность, могут на самом деле не быть предсказуемыми в какой-либо практической ситуации. Если данный бит токена не проходит тесты, это означает лишь, что последовательность наблюдаемых битов в этой позиции содержит характеристики, которые маловероятны в действительно случайном токене. Но попытка предсказать значение этого бита в следующем токене, основываясь на наблюдаемых характеристиках, может быть ненамного надежнее, чем слепое угадывание. Умножение этой ненадежности на большое количество бит, которые необходимо предсказать одновременно, может означать, что вероятность сделать правильное предсказание чрезвычайно низка.*

ПРАКТИЧЕСКИЕ ШАГИ

1. Определите, когда и как выдаются токены сессий, пройдя по приложению от первой страницы до любых функций входа. Распространены два варианта поведения:
    - Приложение создает новую сессию каждый раз, когда получает запрос без токена.
    - Приложение создает новую сессию после успешного входа в систему.

    Чтобы автоматизированно собрать большое количество токенов, в идеале найдите один-единственный запрос (обычно GET / или отправка формы входа), который вызывает выдачу нового токена.
2. В Burp Suite отправьте запрос, создающий новую сессию, в Burp Sequencer и настройте расположение токена. Затем запустите сбор в реальном времени (live capture), чтобы собрать как можно больше токенов. Если используется

самописный механизм управления сессиями, и у вас есть только удаленный доступ к приложению, собирайте токены как можно быстрее, чтобы минимизировать потерю токенов, выданных другим пользователям, и уменьшить влияние любой зависимости от времени.

3. Если используется коммерческий механизм управления сессиями и/или у вас есть локальный доступ к приложению, вы можете получать бесконечно длинные последовательности токенов сессий в контролируемых условиях.

4. Во время сбора токенов в Burp Sequencer включите настройку «auto analyse» (автоматический анализ), чтобы Burp периодически автоматически выполнял статистический анализ. Соберите не менее 500 токенов, прежде чем детально изучать результаты. Если достаточное количество бит в токене прошло тесты, продолжайте собирать токены так долго, как это возможно, просматривая результаты анализа по мере их поступления.

5. Если токены не проходят тесты на случайность и, по-видимому, содержат шаблоны, которые можно использовать для предсказания будущих токенов, повторите упражнение с другого IP-адреса и (если применимо) с другим именем пользователя. Это поможет вам определить, обнаруживается ли тот же самый шаблон, и можно ли экстраполировать токены, полученные в первом упражнении, для определения токенов, полученных во втором. Иногда последовательность токенов, собранная одним пользователем, демонстрирует шаблон. Но это не позволит напрямую экстраполировать его на токены, выданные другим пользователям, потому что такая информация, как IP-адрес источника, используется в качестве источника энтропии (например, как начальное значение (seed) для генератора случайных чисел).

6. Если вы считаете, что достаточно разобрались в алгоритме генерации токенов, чтобы провести автоматизированную атаку на сессии других пользователей, скорее всего, лучший способ достичь этого — использовать собственный (кастомный) скрипт. Он сможет генерировать токены, используя обнаруженные вами специфические шаблоны, и применять любое необходимое кодирование. В Главе 14 описаны некоторые общие методики применения автоматизации для такого рода задач.

7. Если доступен исходный код, внимательно изучите код, отвечающий за генерацию токенов сессий, чтобы понять используемый механизм и определить, уязвим ли он для предсказания. Если энтропия извлекается из данных, которые можно определить в приложении в пределах диапазона, поддающегося перебору, оцените практическое количество запросов, которое потребуется для подбора токена приложения.

ПОПРОБУЙТЕ!

http://mdsec.net/auth/361/

Зашифрованные токены

Некоторые приложения используют токены, содержащие значимую информацию о пользователе, и стремятся избежать очевидных проблем, которые это влечет за собой, шифруя токены перед их выдачей пользователям. Поскольку токены шифруются с использованием секретного ключа, неизвестного пользователям, это кажется надежным подходом, так как пользователи не смогут расшифровать токены и вмешаться в их содержимое.

Однако в некоторых ситуациях, в зависимости от используемого алгоритма шифрования и способа, которым приложение обрабатывает токены, для пользователей тем не менее может оказаться возможным вмешиваться в значимое содержимое токенов, фактически не расшифровывая их. Как бы странно это ни звучало, это на самом деле реализуемые атаки, которые иногда легко осуществить, и многочисленные реальные приложения оказались уязвимы для них. Виды применимых атак зависят от конкретного криптографического алгоритма, который используется.

Шифры ECB (режим электронной кодовой книги)

Приложения, использующие зашифрованные токены, применяют симметричный алгоритм шифрования, чтобы токены, полученные от пользователей, можно было расшифровать для восстановления их значимого содержимого. Некоторые симметричные алгоритмы шифрования используют шифр в режиме «электронной кодовой книги» (ECB). Этот тип шифра делит открытый текст (plaintext) на блоки одинакового размера (например, по 8 байт каждый) и шифрует каждый блок с помощью секретного ключа. Во время расшифровки каждый блок шифротекста (ciphertext) расшифровывается тем же ключом для восстановления исходного блока открытого текста. Особенность этого метода в том, что шаблоны в открытом тексте могут приводить к появлению шаблонов в шифротексте, поскольку идентичные блоки открытого текста будут зашифрованы в идентичные блоки шифротекста. Для некоторых типов данных, например, растровых изображений (bitmap), это означает, что значимую информацию из открытого текста можно различить в шифротексте, как показано на Рисунке 7-4.

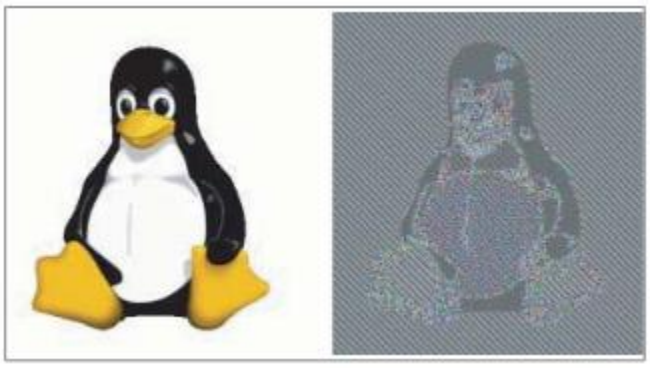


Рисунок 7-4: Шаблоны в открытом тексте (plaintext), зашифрованном с помощью шифра ECB, могут быть видны в итоговом шифротексте (ciphertext)

Несмотря на этот недостаток ECB, эти шифры часто используются для шифрования информации в веб-приложениях. Даже в ситуациях, когда проблема шаблонов в открытом тексте не возникает, уязвимости все равно могут существовать. Это связано с поведением шифра, заключающимся в шифровании идентичных блоков открытого текста в идентичные блоки шифротекста.

Рассмотрим приложение, чьи токены содержат несколько различных значимых компонентов, включая числовой идентификатор пользователя:

rnd=2458992;app=iTradeEUR_1;uid=218;username=dafydd;time=634430423694715000;

Когда этот токен зашифрован, он, по-видимому, бессмыслен и, скорее всего, пройдет все стандартные статистические тесты на случайность:

68BAC980742B9EF80A27CBBBC0618E3876FF3D6C6E6A7B9CB8FCA486F9E11922776
F0307329140AABD223F003A8309DDB6B970C47BA2E249A0670592D74BCD07D51A3E1
50EFC2E69885A5C8131E4210F

Используемый шифр ECB работает с 8-байтными блоками данных, и блоки открытого текста (plaintext) сопоставляются с соответствующими блоками шифротекста (ciphertext) следующим образом:

rnd=2458	*68BAC980742B9EF8*
992;app=	*0A27CBBBC0618E38*
iTradeEU	*76FF3D6C6E6A7B9C*
R_1;uid=	*B8FCA486F9E11922*
218;user	*776F0307329140AA*
name=daf	*BD223F003A8309DD*
ydd;time	*B6B970C47BA2E249*
=6344304	*A0670592D74BCD07*
23694715	*D51A3E150EFC2E69*
000;	*885A5C8131E4210F*

Поскольку каждый блок шифротекста (ciphertext) всегда будет расшифровываться в один и тот же блок открытого текста (plaintext), злоумышленник может манипулировать последовательностью блоков шифротекста, чтобы осмысленно изменить соответствующий открытый текст. В зависимости от того, как именно приложение обрабатывает итоговый расшифрованный токен, это может позволить злоумышленнику переключиться на другого пользователя или повысить привилегии.

Например, если продублировать второй блок после четвертого, последовательность блоков будет следующей:

rnd=2458	*68BAC980742B9EF8*
992;app=	*0A27CBBBC0618E38*
iTradeEU	*76FF3D6C6E6A7B9C*
R_1;uid=	*B8FCA486F9E11922*
992;app=	***0A27CBBBC0618E38***
218;user	*776F0307329140AA*
name=daf	*BD223F003A8309DD*
ydd;time	*B6B970C47BA2E249*
=6344304	*A0670592D74BCD07*
23694715	*D51A3E150EFC2E69*
000;	*885A5C8131E4210F*

Расшифрованный токен теперь содержит измененное значение uid, а также дублированное значение app. Что именно произойдет, зависит от того, как приложение обрабатывает расшифрованный токен. Часто приложения, использующие токены таким образом, проверяют только определенные части расшифрованного токена, например, идентификатор пользователя. Если приложение ведет себя так, то оно обработает запрос в контексте пользователя с uid, равным 992, а не первоначального 218.

Только что описанная атака зависела бы от получения подходящего значения rnd, которое соответствовало бы действительному значению uid после манипуляции с блоками. Альтернативная и более надежная атака заключалась бы в регистрации имени пользователя, содержащего числовое значение в нужном смещении, и дублировании этого блока с целью замены существующего значения uid. Предположим, вы регистрируете имя пользователя daf1 и получаете следующий токен:

9A5A47BF9B3B6603708F9DEAD67C7F4C76FF3D6C6E6A7B9CB8FCA486F9E11922A5B
C430A73B38C14BD223F003A8309DDF29A5A6F0DC06C53905B5366F5F4684C0D2BBB
B08BD834BBADEBC07FFE87819D

Блоки открытого текста (plaintext) и шифротекста (ciphertext) для этого токена следующие:

rnd=9224	*9A5A47BF9B3B6603*
856;app=	*708F9DEAD67C7F4C*
iTradeEU	*76FF3D6C6E6A7B9C*
R_1;uid=	*B8FCA486F9E11922*
219;user	*A5BC430A73B38C14*
name=daf	*BD223F003A8309DD*
1;time=6	*F29A5A6F0DC06C53*
34430503	*905B5366F5F4684C*
61065250	*0D2BBBB08BD834BB*
0;	*ADEBC07FFE87819D*

Если затем продублировать седьмой блок после четвертого, ваш расшифрованный токен будет содержать значение uid, равное 1:

rnd=9224	*9A5A47BF9B3B6603*
856;app=	*708F9DEAD67C7F4C*
iTradeEU	*76FF3D6C6E6A7B9C*
R_1;uid=	*B8FCA486F9E11922*
1;time=6	***F29A5A6F0DC06C53***
219;user	*A5BC430A73B38C14*
name=daf	*BD223F003A8309DD*
1;time=6	*F29A5A6F0DC06C53*
34430503	*905B5366F5F4684C*
61065250	*0D2BBBB08BD834BB*
0;	*ADEBC07FFE87819D*

Путем регистрации подходящего диапазона имен пользователей и повторного проведения этой атаки вы потенциально сможете перебрать весь диапазон действительных значений uid и таким образом выдавать себя за каждого пользователя приложения.

ПОПРОБУЙТЕ!

http://mdsec.net/auth/363/

Шифры CBC (режим сцепления блоков шифротекста)

Недостатки шифров ECB привели к разработке шифров в режиме сцепления блоков шифротекста (CBC). В шифре CBC перед шифрованием каждого блока открытого текста (plaintext) выполняется его операция XOR с предыдущим блоком шифротекста (ciphertext), как показано на Рисунке 7-5. Это предотвращает шифрование идентичных блоков открытого текста в идентичные блоки шифротекста. Во время расшифровки операция XOR

применяется в обратном порядке, и каждый расшифрованный блок подвергается операции XOR с предыдущим блоком шифротекста для восстановления исходного открытого текста.

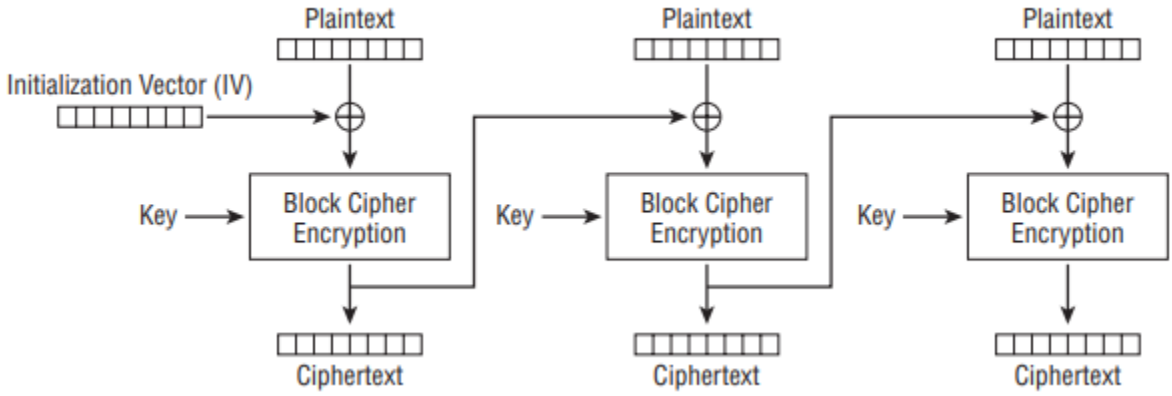


Рисунок 7-5: В шифре CBC каждый блок открытого текста (plaintext) подвергается операции XOR с предыдущим блоком шифротекста (ciphertext) перед шифрованием

Поскольку шифры CBC позволяют избежать некоторых проблем, присущих шифрам ECB, стандартные симметричные алгоритмы шифрования, такие как DES и AES, часто используются в режиме CBC. Однако то, как токены, зашифрованные в режиме CBC, часто применяются в веб-приложениях, означает, что злоумышленник может оказаться в состоянии манипулировать частями расшифрованных токенов, не зная секретного ключа.

Рассмотрим вариацию предыдущего приложения, чьи токены содержат несколько различных значимых компонентов, включая числовой идентификатор пользователя:

rnd=191432758301;app=eBankProdTC;uid=216;time=6343303;

Как и ранее, когда эта информация зашифрована, получается, на первый взгляд, бессмысленный токен:

0FB1F1AFB4C874E695AAFC9AA4C2269D3E8E66BBA9B2829B173F255D447C51321586257C6E459A93635636F45D7B1A43163201477

Поскольку этот токен зашифрован с использованием шифра CBC, при его расшифровке результат дешифрования каждого блока шифротекста подвергается операции XOR с предыдущим блоком шифротекста для получения открытого текста. Теперь, если злоумышленник изменяет части шифротекста (полученного им токена), это приводит к тому, что соответствующий блок расшифровывается в «мусор». Однако это также приводит к тому, что следующий блок открытого текста подвергается операции XOR с измененным значением, в результате чего получается измененный, но все еще осмысленный открытый текст. Другими словами, манипулируя одним отдельным блоком токена, злоумышленник может систематически изменять расшифрованное содержимое следующего за ним блока. В зависимости от того, как приложение обрабатывает итоговый

расшифрованный токен, это может позволить злоумышленнику переключиться на другого пользователя или повысить привилегии.

Посмотрим, как это работает. В описанном примере злоумышленник проходит по зашифрованному токену, произвольным образом изменяя по одному символу за раз и отправляя каждый измененный токен приложению. Это включает в себя большое количество запросов. Ниже приведена подборка значений, которые получаются, когда приложение расшифровывает каждый измененный токен:

????????32858301;app=eBankProdTC;uid=216;time=6343303;
????????32758321;app=eBankProdTC;uid=216;time=6343303;
rnd=1914????????;aqp=eBankProdTC;uid=216;time=6343303;
rnd=1914????????;app=eAankProdTC;uid=216;time=6343303;
rnd=191432758301????????nkPqodTC;uid=216;time=6343303;
rnd=191432758301????????nkProdUC;uid=216;time=6343303;
rnd=191432758301;app=eBa????????;uie=216;time=6343303;
rnd=191432758301;app=eBa????????;uid=226;time=6343303;
rnd=191432758301;app=eBankProdTC????????;timd=6343303;
rnd=191432758301;app=eBankProdTC????????;time=6343503;

В каждом случае блок, который изменил злоумышленник, как и ожидалось, расшифровывается в бессмыслицу (обозначено как ????????). Однако следующий блок расшифровывается в осмысленный текст, который незначительно отличается от оригинального токена. Как уже было описано, это различие возникает потому, что расшифрованный текст подвергается операции XOR с предыдущим блоком шифротекста, который злоумышленник немного изменил.

Хотя злоумышленник и не видит расшифрованных значений, приложение пытается их обработать, и злоумышленник видит результаты в ответах приложения. Что именно произойдет, зависит от того, как приложение обрабатывает поврежденную часть расшифрованного токена. Если приложение отклоняет токены, содержащие любые недействительные данные, атака проваливается. Однако часто приложения, использующие токены таким образом, проверяют только определенные части расшифрованного токена, например, идентификатор пользователя. Если приложение ведет себя так, то восьмой пример из предыдущего списка оказывается успешным, и приложение обрабатывает запрос в контексте пользователя с uid равным 226, а не первоначальным 216.

Вы можете легко тестировать приложения на эту уязвимость, используя тип полезной нагрузки «bit flipper» (побитовое изменение) в Burp Intruder. Сначала вам нужно войти в приложение под своей учетной записью. Затем вы находите страницу приложения, которая зависит от залогиненной сессии и показывает идентификатор вошедшего пользователя в ответе. Обычно для этой цели служит домашняя страница пользователя или страница с деталями учетной записи. На Рисунке 7-6 показан Burp Intruder, настроенный на атаку

домашней страницы пользователя, где зашифрованный токен сессии отмечен как позиция для полезной нагрузки (payload).

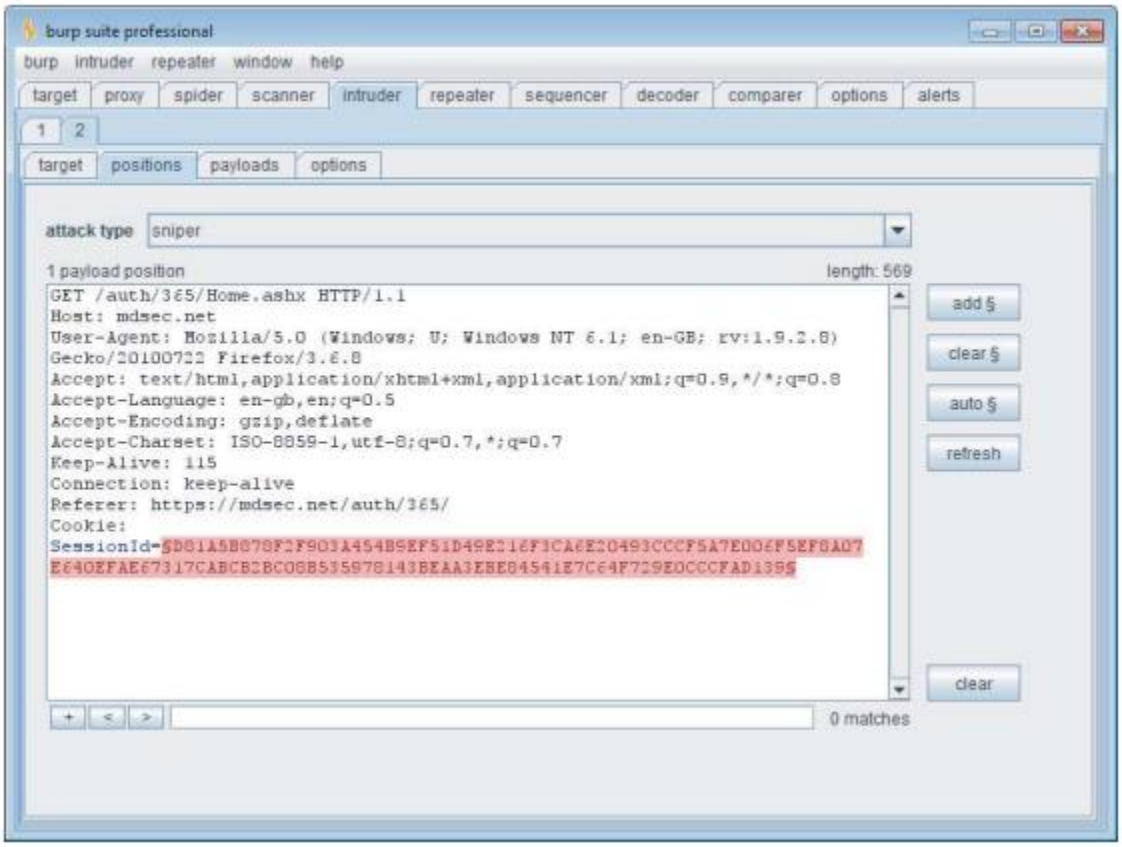


Рисунок 7-6: Настройка Burp Intruder для изменения зашифрованного токена сессии

На Рисунке 7-7 показана необходимая конфигурация полезной нагрузки (payload). Она указывает Burp работать с исходным значением токена, рассматривая его как ASCII-представление шестнадцатеричных данных, и инвертировать (flip) каждый бит в каждой символьной позиции. Этот подход идеален, потому что он требует относительно небольшого количества запросов (восемь запросов на каждый байт данных в токене) и почти всегда определяет, уязвимо ли приложение. Это позволяет вам использовать более сфокусированную атаку для проведения реальной эксплуатации.

При выполнении атаки начальные запросы не вызывают заметных изменений в ответах приложения, и сессия пользователя остается нетронутой. Это интересно само по себе, поскольку указывает на то, что первая часть токена не используется для идентификации вошедшего пользователя. Многие из последующих запросов в ходе атаки вызывают перенаправление на страницу входа, что указывает на то, что изменение сделало токен недействительным. Что особенно важно, есть также серия запросов, где ответ, по-видимому, является частью действительной сессии, но не связан с личностью исходного пользователя. Это соответствует блоку токена, который содержит значение uid. В некоторых случаях приложение просто отображает «неизвестный пользователь»,

указывая, что измененный uid не соответствует реальному пользователю, и атака не удалась. В других случаях оно показывает имя другого зарегистрированного пользователя приложения, что окончательно доказывает успех атаки. На Рисунке 7-8 показаны результаты атаки. Здесь мы определили столбец «extract grep», чтобы отображать идентификатор вошедшего пользователя, и установили фильтр для скрытия ответов, представляющих собой перенаправление на страницу входа.

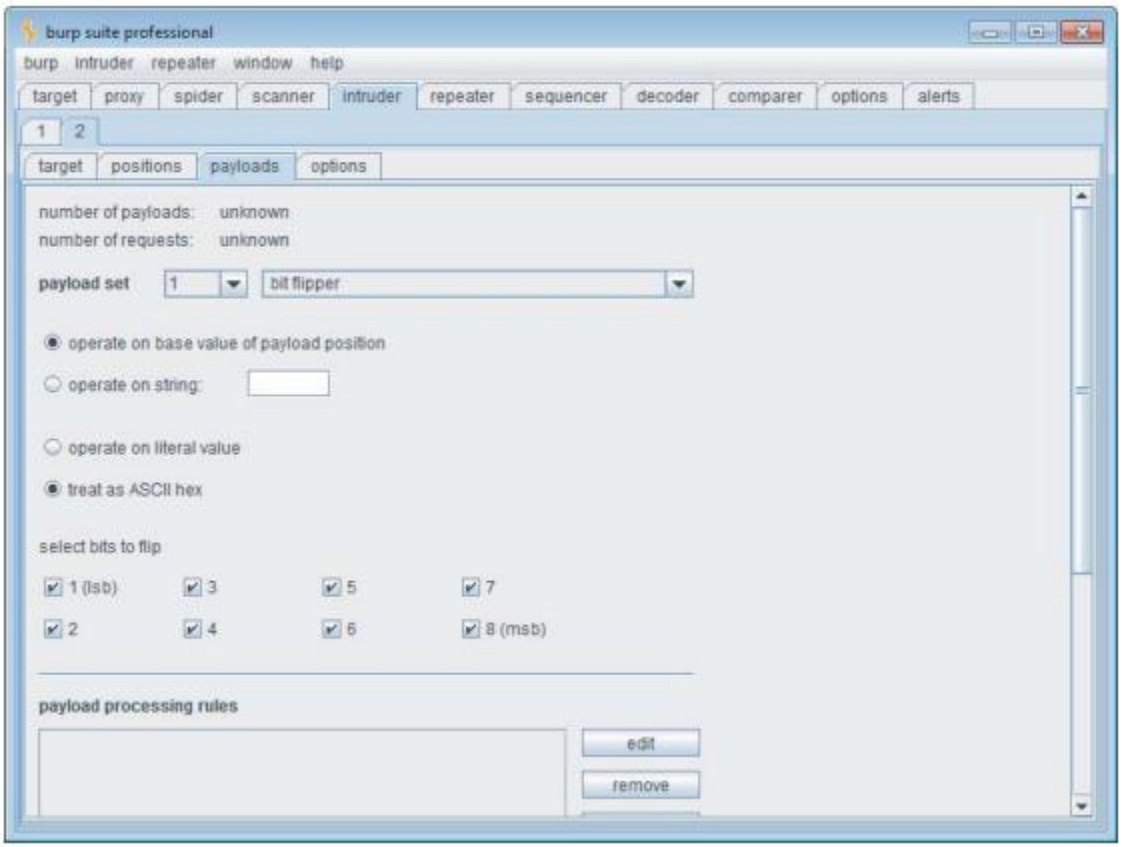


Рисунок 7-7: Настройка Burp Intruder для инвертирования (flipping) каждого бита в зашифрованном токене

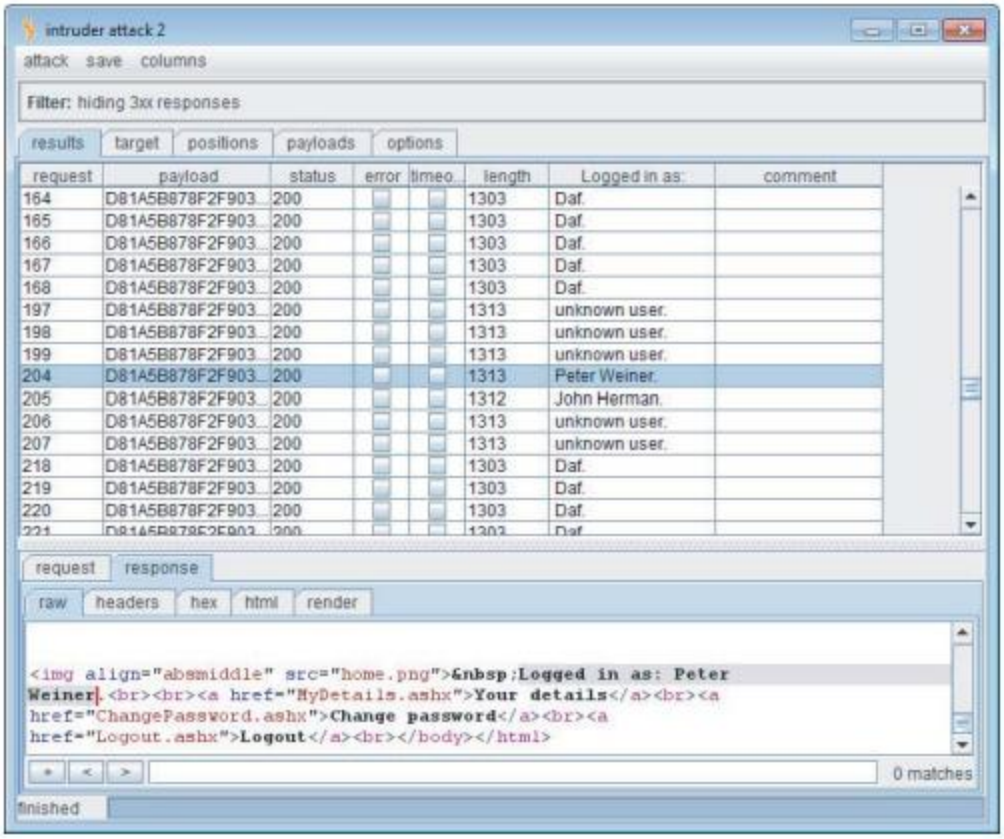


Рисунок 7-8: Успешная атака с инвертированием битов (bit flipping) на зашифрованный токен

Обнаружив уязвимость, вы можете перейти к ее эксплуатации с помощью более сфокусированной атаки. Для этого вам нужно по результатам определить, какой именно блок зашифрованного токена подвергается изменениям, когда меняется контекст пользователя. Затем вы бы провели атаку, которая тестирует множество других значений именно в этом блоке. Для этого вы можете использовать тип полезной нагрузки «numbers» (числа) в Burp Intruder.

ПОПРОБУЙТЕ!

http://mdsec.net/auth/365/

ПРИМЕЧАНИЕ *Некоторые приложения используют технику шифрования значимых данных в параметрах запроса в более общем смысле, в попытке предотвратить несанкционированное изменение (tampering) данных, например, цен на товары в магазине. В любом месте, где вы видите данные, которые выглядят зашифрованными и играют ключевую роль в функциональности приложения, вам следует попробовать технику*

инвертирования битов (bit-flipping), чтобы проверить, можете ли вы осмысленно манипулировать зашифрованной информацией с целью вмешательства в логику приложения.

При попытке проэксплуатировать описанную в этом разделе уязвимость вашей целью, конечно же, будет выдать себя за других пользователей приложения — в идеале, за администратора с более высокими привилегиями. Если вы ограничены слепым манипулированием частями зашифрованного токена, это может потребовать определенной доли удачи. Однако в некоторых случаях приложение может оказать вам большую помощь. Когда приложение применяет симметричное шифрование для защиты данных от вмешательства со стороны пользователей, часто во всем приложении используются один и тот же алгоритм и ключ шифрования. В этой ситуации, если какая-либо функция приложения раскрывает пользователю расшифрованное значение произвольной зашифрованной строки, это можно использовать для полной расшифровки любого элемента защищенной информации.

Одно из приложений, которое наблюдали авторы, содержало функцию загрузки/скачивания файлов. После загрузки файла пользователи получали ссылку для скачивания, содержащую параметр с именем файла. Чтобы предотвратить различные атаки, связанные с манипулированием путями к файлам, приложение шифровало имя файла в этом параметре. Однако, если пользователь запрашивал удаленный файл, приложение отображало сообщение об ошибке, показывающее расшифрованное имя запрошенного файла. Это поведение можно было использовать, чтобы узнать незашифрованное (plaintext) значение любой зашифрованной строки, используемой в приложении, включая значения токенов сессий. Было обнаружено, что токены сессий содержат различные значимые значения в структурированном формате, уязвимом для атак, описанных в этом разделе. Поскольку эти значения включали текстовые имена пользователей и роли в приложении, а не числовые идентификаторы, было бы чрезвычайно сложно провести успешный эксплойт, используя только слепое инвертирование битов. Однако, используя функцию-дешифратор имен файлов, можно было систематически манипулировать битами токена, наблюдая при этом за результатами. Это позволило сконструировать токен, который при расшифровке указывал на действительного пользователя и административную роль, что дало полный контроль над приложением.

ПРИМЕЧАНИЕ *Другие методики могут позволить вам расшифровать зашифрованные данные, используемые приложением. «Раскрывающий» криптографический оракул (reveal encryption oracle) можно использовать для получения незашифрованного значения зашифрованного токена. Хотя это может быть серьезной уязвимостью при расшифровке пароля, расшифровка токена сессии не дает немедленной возможности скомпрометировать сессии других пользователей. Тем не менее, расшифрованный токен дает полезное представление о структуре открытого текста, что полезно при проведении целенаправленной атаки с инвертированием битов (bit-flipping). Подробнее об атаках с использованием «раскрывающего» криптографического оракула см. в Главе 11.*

Атаки по побочным каналам (Side channel attacks) на оракулы дополнения (padding oracles) могут использоваться для компрометации зашифрованных токенов. Подробнее см. в Главе 18.

ПРАКТИЧЕСКИЕ ШАГИ

Во многих ситуациях, где используются зашифрованные токены, реальная возможность эксплуатации может зависеть от различных факторов, включая смещения границ блоков относительно данных, которые вам нужно атаковать, и терпимость приложения к изменениям, которые вы вызываете в окружающей структуре открытого текста. Работая вслепую, может показаться сложным построить эффективную атаку, однако во многих ситуациях это на самом деле возможно.

1. Если токен сессии не является очевидно осмысленным или последовательным сам по себе, всегда рассматривайте возможность, что он может быть зашифрован. Вы часто можете определить, что используется блочный шифр, регистрируя несколько разных имен пользователей и каждый раз добавляя по одному символу в длину. Если вы обнаружите, что добавление одного символа приводит к скачку длины токена сессии на 8 или 16 байт, то, вероятно, используется блочный шифр. Вы можете подтвердить это, продолжая добавлять байты к вашему имени пользователя и наблюдая за таким же скачком через 8 или 16 байт.

2. Уязвимости, связанные с манипуляцией шифром ECB, обычно трудно выявить и проэксплуатировать в контексте чистого «черного ящика». Вы можете попробовать вслепую дублировать и перемещать блоки шифротекста внутри вашего токена и проверять, остаетесь ли вы залогинены в приложении в своем собственном контексте, в контексте другого пользователя или вообще ни в каком.

3. Вы можете тестировать уязвимости манипуляции шифром CBC, запустив атаку в Burp Intruder по всему токену, используя источник полезной нагрузки «bit flipping» (инвертирование битов). Если атака с инвертированием битов выявит уязвимый участок в токене, манипуляция которым позволяет вам оставаться в действительной сессии, но под видом другого или несуществующего пользователя, проведите более сфокусированную атаку только на этот участок, пробуя более широкий диапазон значений в каждой позиции.

4. Во время обеих атак отслеживайте ответы приложения, чтобы определить пользователя, с которым связана ваша сессия после каждого запроса, и пытайтесь проэксплуатировать любые возможности для повышения привилегий, которые могут возникнуть.

5. Если ваши атаки не увенчались успехом, но из шага 1 следует, что контролируемый вами ввод переменной длины включается в токен, вам следует попробовать сгенерировать серию токенов, добавляя по одному символу за раз, по крайней мере, до размера используемого блока. Для

каждого полученного токена следует повторить шаги 2 и 3. Это повысит шанс того, что данные, которые вам нужно изменить, будут правильно выровнены по границам блоков для успеха вашей атаки.

Уязвимости в обработке токенов сессий

Неважно, насколько эффективно приложение гарантирует, что генерируемые им токены сессий не содержат никакой значимой информации и не поддаются анализу или предсказанию — его механизм сессий будет полностью открыт для атаки, если с этими токенами не обращаются осторожно после их генерации. Например, если токены каким-либо образом раскрываются злоумышленнику, он может перехватывать сессии пользователей, даже если предсказать токены невозможно.

Небезопасная обработка токенов приложением может сделать его уязвимым для атак несколькими способами.

> **РАСПРОСТРАНЕННЫЙ МИФ** *«Наш токен защищен от раскрытия третьим сторонам, потому что мы используем SSL».*
>
> *Правильное использование SSL, безусловно, помогает защитить токены сессий от перехвата. Но различные ошибки все равно могут привести к тому, что токены будут передаваться в открытом виде (cleartext), даже при наличии SSL. И для получения их токенов могут быть использованы различные прямые атаки на конечных пользователей.*

> **РАСПРОСТРАНЕННЫЙ МИФ** *«Наш токен генерируется платформой с использованием зрелых, криптографически стойких технологий, поэтому он не уязвим для компрометации».*
>
> *Поведение сервера приложений по умолчанию часто заключается в создании cookie сессии при первом посещении сайта пользователем и сохранении его доступности на протяжении всего взаимодействия пользователя с сайтом. Как будет описано в следующих разделах, это может привести к различным уязвимостям безопасности в том, как обрабатывается токен.*

Раскрытие токенов в сети

Эта область уязвимостей возникает, когда токен сессии передается по сети в незашифрованном виде, что позволяет перехватчику (eavesdropper), находящемуся в подходящей точке, получить токен и, таким образом, выдавать себя за легитимного пользователя. Подходящие места для перехвата включают локальную сеть пользователя, IT-отдел пользователя, интернет-провайдера пользователя, магистральные каналы Интернета, интернет-провайдера, предоставляющего хостинг для приложения, и IT-отдел организации, хостящей приложение. В каждом случае это включает как авторизованный

персонал соответствующей организации, так и любых внешних злоумышленников, скомпрометировавших соответствующую инфраструктуру.

В простейшем случае, когда приложение использует незашифрованное HTTP-соединение для коммуникаций, злоумышленник может перехватить все передаваемые данные между клиентом и сервером, включая учетные данные для входа, личную информацию, платежные реквизиты и так далее. В этой ситуации атака на сессию пользователя часто не является необходимой, поскольку злоумышленник и так может просматривать привилегированную информацию и может войти в систему, используя перехваченные учетные данные, для выполнения других вредоносных действий. Однако все же могут быть случаи, когда сессия пользователя является основной целью. Например, если перехваченных учетных данных недостаточно для повторного входа (скажем, в банковском приложении они могут включать число, отображаемое на изменяющемся физическом токене, или определенные цифры из PIN-кода пользователя), злоумышленнику может потребоваться перехватить подслушанную сессию, чтобы выполнить произвольные действия. Или, если входы в систему тщательно аудируются, и пользователь уведомляется о каждом успешном входе, злоумышленник может захотеть избежать выполнения собственного входа, чтобы быть как можно более незаметным.

В других случаях приложение может использовать HTTPS для защиты ключевых коммуникаций между клиентом и сервером, но все равно быть уязвимым для перехвата токенов сессий в сети. Эта уязвимость может проявляться различными способами, многие из которых возникают именно при использовании HTTP-cookie в качестве механизма передачи токенов сессий:

- Некоторые приложения предпочитают использовать HTTPS для защиты учетных данных пользователя во время входа, но затем возвращаются к HTTP на протяжении оставшейся части сессии. Так ведут себя многие приложения веб-почты. В этой ситуации перехватчик не может перехватить учетные данные пользователя, но все равно может захватить токен сессии. Инструмент Firesheep, выпущенный как плагин для Firefox, делает этот процесс простым.

- Некоторые приложения используют HTTP для областей сайта, не требующих аутентификации, таких как главная страница, но переключаются на HTTPS, начиная со страницы входа. Однако во многих случаях пользователю выдается токен сессии на первой же посещенной странице, и этот токен не изменяется, когда пользователь входит в систему. Сессия пользователя, изначально неаутентифицированная, «повышается» до аутентифицированной сессии после входа. В этой ситуации перехватчик может перехватить токен пользователя до входа, дождаться, пока коммуникации пользователя переключатся на HTTPS (что указывает на то, что пользователь входит в систему), а затем попытаться получить доступ к защищенной странице (например, «Мой аккаунт»), используя этот токен.

- Даже если приложение выдает новый токен после успешного входа и использует HTTPS, начиная со страницы входа, токен аутентифицированной сессии

пользователя все равно может быть раскрыт. Это может произойти, если пользователь повторно посетит страницу, не требующую аутентификации (например, «Помощь» или «О нас»), перейдя по ссылкам из защищенной области, используя кнопку «назад» или введя URL напрямую.

- В вариации предыдущего случая приложение может пытаться переключиться на HTTPS, когда пользователь нажимает ссылку «Вход». Однако оно все еще может принять вход по HTTP, если пользователь соответствующим образом изменит URL. В этой ситуации злоумышленник, находящийся в подходящей точке сети, может изменить страницы, возвращаемые в неаутентифицированных областях сайта, так, чтобы ссылка «Вход» вела на HTTP-версию. Даже если приложение выдает новый токен сессии после успешного входа, злоумышленник все равно сможет перехватить этот токен, если ему удастся понизить уровень соединения пользователя до HTTP.

- Некоторые приложения используют HTTP для всего статического контента, такого как изображения, скрипты, таблицы стилей и шаблоны страниц. На такое поведение часто указывает предупреждение в браузере пользователя, как показано на Рисунке 7-9. Когда браузер показывает это предупреждение, он уже получил соответствующий элемент по HTTP, поэтому токен сессии уже был передан. Цель предупреждения браузера — позволить пользователю отказаться от обработки данных ответа, полученных по HTTP, которые, следовательно, могут быть «заражены» (tainted). Как описывалось ранее, злоумышленник может перехватить токен сессии пользователя, когда браузер обращается к ресурсу по HTTP, и использовать этот токен для доступа к защищенным, нестатическим областям сайта по HTTPS.

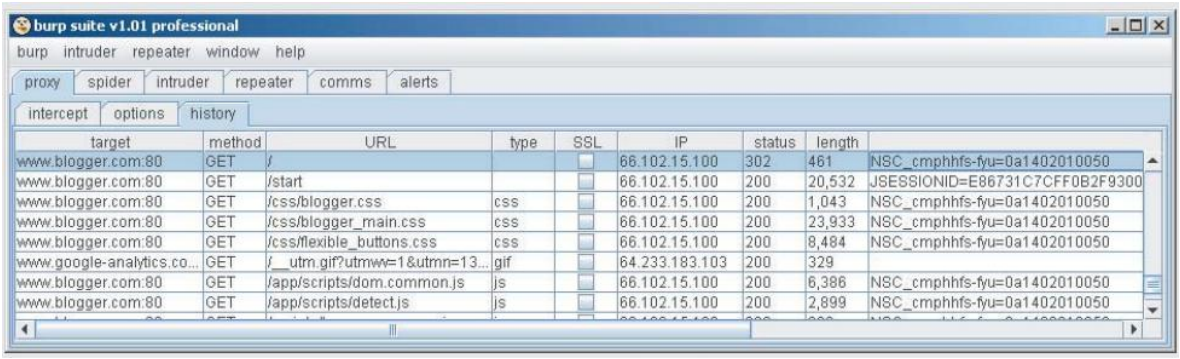
Рисунок 7-9: Браузеры показывают предупреждение, когда страница, доступ к которой получен по HTTPS, содержит элементы, доступ к которым получен по HTTP

- Даже если приложение использует HTTPS для каждой страницы, включая неаутентифицированные области и статический контент, все еще могут существовать обстоятельства, при которых токены пользователей передаются по HTTP. Если злоумышленник сможет каким-либо образом побудить пользователя сделать запрос по HTTP (либо к HTTP-сервису на том же сервере, если он запущен, либо к http://server:443/), его токен может быть отправлен. Способы, которыми злоумышленник может этого добиться, включают отправку пользователю URL в

электронном письме или мгновенном сообщении, размещение автозагружаемых ссылок на контролируемом им веб-сайте или использование кликабельных рекламных баннеров. (Подробнее о такого рода методиках для проведения атак на других пользователей см. в Главах 12 и 13.)

ПРАКТИЧЕСКИЕ ШАГИ

1. Пройдите по приложению обычным путем от первого входа (стартового URL), через процесс логина, а затем по всей функциональности приложения. Ведите запись каждого посещенного URL и отмечайте каждый случай получения нового токена сессии. Обращайте особое внимание на функции входа и на переходы между HTTP и HTTPS. Это можно сделать вручную с помощью сетевого сниффера, такого как Wireshark, или частично автоматизировать с помощью функций журналирования вашего перехватывающего прокси, как показано на Рисунке 7-10.



target	method	URL	type	SSL	IP	status	length	
www.blogger.com:80	GET	/			66.102.15.100	302	461	NSC_cmphhfs-fyu=0a1402010050
www.blogger.com:80	GET	/start			66.102.15.100	200	20,532	JSESSIONID=E86731C7CFF0B2F9300
www.blogger.com:80	GET	/css/blogger.css	css		66.102.15.100	200	1,043	NSC_cmphhfs-fyu=0a1402010050
www.blogger.com:80	GET	/css/blogger_main.css	css		66.102.15.100	200	23,933	NSC_cmphhfs-fyu=0a1402010050
www.blogger.com:80	GET	/css/flexible_buttons.css	css		66.102.15.100	200	8,484	NSC_cmphhfs-fyu=0a1402010050
www.google-analytics.co...	GET	/__utm.gif?utmwv=1&utmn=13...	gif		64.233.183.103	200	329	
www.blogger.com:80	GET	/app/scripts/dom.common.js	js		66.102.15.100	200	6,386	NSC_cmphhfs-fyu=0a1402010050
www.blogger.com:80	GET	/app/scripts/detect.js	js		66.102.15.100	200	2,899	NSC_cmphhfs-fyu=0a1402010050

Рисунок 7-10: Обход приложения для определения мест, где получаются новые токены сессий

2. Если для передачи токенов сессий используются HTTP-cookie, проверьте, установлен ли флаг secure, который запрещает их передачу по незашифрованным соединениям.

3. Определите, передаются ли когда-либо токены сессий по незашифрованному соединению при обычном использовании приложения. Если да, их следует считать уязвимыми для перехвата.

4. В случаях, когда стартовая страница использует HTTP, а приложение переключается на HTTPS для входа и аутентифицированных областей, проверьте, выдается ли новый токен после входа или токен, переданный на этапе HTTP, все еще используется для отслеживания аутентифицированной сессии пользователя. Также проверьте, примет ли приложение вход по HTTP, если URL входа будет соответствующим образом изменен.

5. Даже если приложение использует HTTPS для каждой страницы, проверьте, прослушивает ли сервер также порт 80, запуская какой-либо сервис или контент. Если да, посетите любой URL по HTTP напрямую из

аутентифицированной сессии и проверьте, передается ли токен сессии.

6. В случаях, когда токен для аутентифицированной сессии передается на сервер по HTTP, проверьте, остается ли этот токен действительным или немедленно аннулируется сервером.

ПОПРОБУЙТЕ!

http://mdsec.net/auth/369/
http://mdsec.net/auth/372/
http://mdsec.net/auth/374/

Раскрытие токенов в логах

Помимо передачи токенов сессий в открытом виде при сетевом обмене, самым распространенным местом, где токены просто раскрываются для несанкционированного просмотра, являются различного рода системные журналы (логи). Хотя это случается реже, последствия такого рода раскрытия обычно более серьезны. Эти логи может просматривать гораздо более широкий круг потенциальных злоумышленников, а не только тот, кто находится в подходящей точке для перехвата сетевого трафика.

Многие приложения предоставляют администраторам и другому персоналу поддержки функциональность для мониторинга и управления аспектами состояния приложения во время выполнения, включая сессии пользователей. Например, сотрудник службы поддержки, помогающий пользователю с проблемами, может спросить его имя, найти его текущую сессию через список или функцию поиска и просмотреть соответствующие детали сессии. Или администратор может изучать лог недавних сессий в ходе расследования инцидента безопасности. Часто такая функциональность мониторинга и контроля раскрывает фактический токен сессии, связанный с каждой сессией. И часто эта функциональность плохо защищена, что позволяет неуполномоченным пользователям получать доступ к списку текущих токенов сессий и, таким образом, перехватывать сессии всех пользователей приложения.

Другая основная причина появления токенов сессий в системных логах — это когда приложение использует строку запроса URL в качестве механизма для их передачи, вместо использования HTTP-cookie или тела POST-запросов. Например, поиск в Google по запросу inurl:jsessionid выявляет тысячи приложений, которые передают токен сессии платформы Java (называемый jsessionid) в URL-адресе:

http://www.webjunction.org/do/Navigation;jsessionid=F27ED2A6AAE4C6DA409A3044E79B8B48?category=327

Когда приложения передают свои токены сессий таким способом, весьма вероятно, что эти токены появятся в различных системных логах (журналах), к которым могут иметь доступ неуполномоченные стороны:

- Логи (журналы) браузеров пользователей
- Логи веб-сервера
- Логи корпоративных или провайдерских прокси-серверов
- Логи любых обратных прокси-серверов, используемых в среде хостинга приложения
- Логи Referer любых серверов, которые пользователи приложения посещают, переходя по внешним ссылкам, как показано на Рисунке 7-11

Некоторые из этих уязвимостей возникают, даже если во всем приложении используется HTTPS.

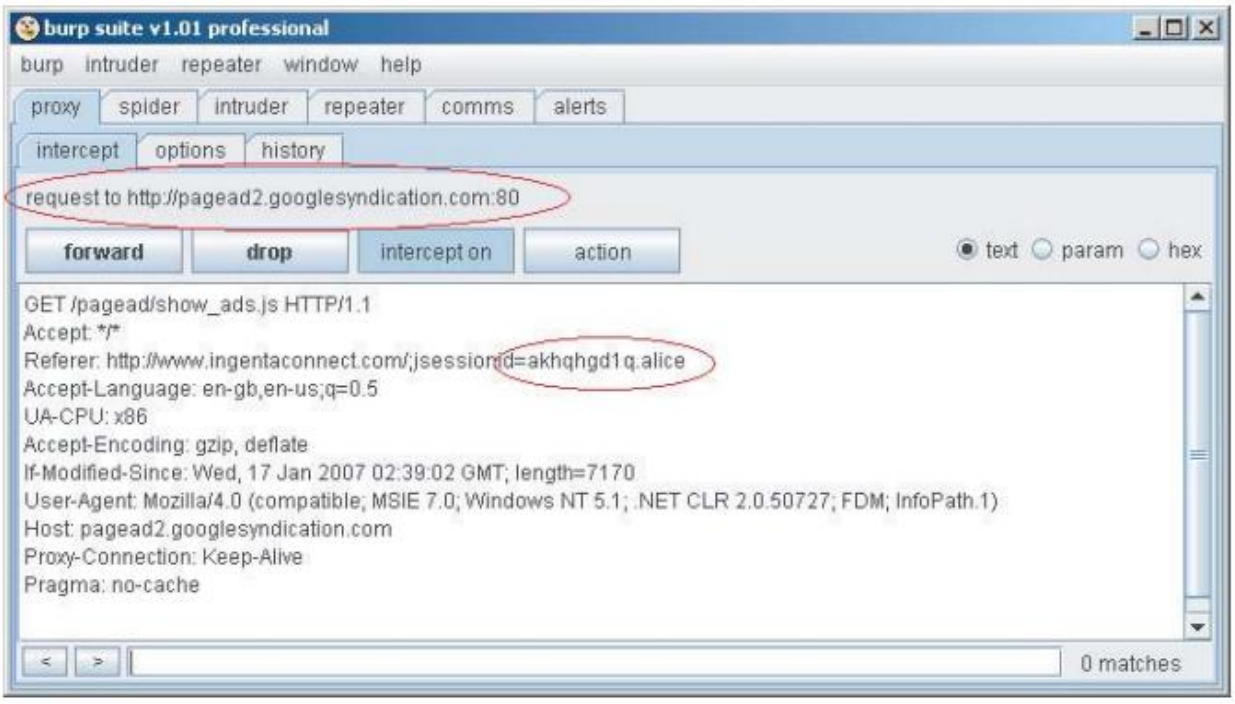


Рисунок 7-11: Когда токены сессий находятся в URL-адресах, они передаются в заголовке Referer, когда пользователи переходят по внешней ссылке или их браузер загружает внешний ресурс

Последний из только что описанных случаев предоставляет злоумышленнику высокоэффективное средство для перехвата токенов сессий в некоторых приложениях.

Например, если приложение веб-почты передает токены сессий в URL-адресе, злоумышленник может отправлять пользователям этого приложения электронные письма, содержащие ссылку на контролируемый им веб-сервер. Если какой-либо пользователь переходит по ссылке (потому что он нажимает на нее или потому что его браузер загружает изображения, содержащиеся в электронном письме формата HTML), злоумышленник получает в реальном времени токен сессии пользователя. Злоумышленник может запустить на своем сервере простой скрипт для перехвата сессии каждого полученного токена и выполнения какого-либо вредоносного действия, такого как рассылка спама, сбор личной информации или смена паролей.

ПРИМЕЧАНИЕ *Текущие версии Internet Explorer не включают заголовок Referer при переходе по внешним ссылкам со страницы, доступ к которой был получен по HTTPS. В этой же ситуации Firefox включает заголовок Referer, при условии, что доступ к внешней ссылке также осуществляется по HTTPS, даже если она ведет на другой домен. Следовательно, конфиденциальная информация, размещенная в URL-адресах, уязвима к утечке в логи Referer, даже при использовании SSL.*

ПРАКТИЧЕСКИЕ ШАГИ

1. Определите всю функциональность приложения и найдите любые функции журналирования или мониторинга, где можно просматривать токены сессий. Проверьте, кто имеет доступ к этой функциональности — например, администраторы, любой аутентифицированный пользователь или любой анонимный пользователь. Методики обнаружения скрытого контента, на который нет прямых ссылок из основного приложения, см. в Главе 4.

2. Определите любые случаи в приложении, когда токены сессий передаются в URL-адресе. Возможно, в целом токены передаются более безопасным способом, но в конкретных случаях разработчики использовали URL для обхода определенных трудностей. Например, такое поведение часто наблюдается, когда веб-приложение взаимодействует с внешней системой.

3. Если токены сессий передаются в URL-адресах, попытайтесь найти любую функциональность приложения, которая позволяет вам внедрять произвольные внешние ссылки на страницы, просматриваемые другими пользователями. Примеры включают функциональность доски объявлений, обратной связи с сайтом, вопросов и ответов и так далее. Если такая возможность есть, отправьте ссылки на контролируемый вами веб-сервер и ждите, появятся ли в ваших логах Referer токены сессий каких-либо пользователей.

4. Если какие-либо токены сессий перехвачены, попытайтесь перехватить сессии пользователей, используя приложение в обычном режиме, но подставив перехваченный токен вместо своего. Вы можете сделать это, перехватив следующий ответ от сервера и добавив свой собственный

заголовок Set-Cookie с перехваченным значением cookie. В Burp вы можете применить единую для всего пакета конфигурацию, которая будет устанавливать определенный cookie во всех запросах к целевому приложению, что позволяет легко переключаться между различными контекстами сессий во время тестирования.

5. Если перехвачено большое количество токенов, и перехват сессий позволяет вам получить доступ к конфиденциальным данным, таким как личные данные, платежная информация или пароли пользователей, вы можете использовать автоматизированные методики, описанные в Главе 14, для сбора всех желаемых данных, принадлежащих другим пользователям приложения.

ПОПРОБУЙТЕ!

http://mdsec.net/auth/379/

Уязвимое сопоставление токенов с сессиями

Различные распространенные уязвимости в механизмах управления сессиями возникают из-за слабых мест в том, как приложение сопоставляет создание и обработку токенов сессий с сессиями отдельных пользователей.

Самая простая уязвимость — это разрешение одновременного присвоения нескольких действительных токенов (concurrent sessions) одной и той же учетной записи пользователя. Практически в любом приложении нет законной причины, по которой у пользователя должна быть активна более чем одна сессия одновременно. Конечно, довольно часто пользователь может бросить активную сессию и начать новую — например, потому что он закрыл окно браузера или перешел на другой компьютер. Но если кажется, что пользователь использует две разные сессии одновременно, это обычно указывает на то, что произошла компрометация безопасности: либо пользователь раскрыл свои учетные данные другой стороне, либо злоумышленник получил его учетные данные каким-то другим способом. В обоих случаях разрешение одновременных сессий нежелательно, потому что это позволяет пользователям упорствовать в нежелательных практиках без неудобств и потому что это позволяет злоумышленнику использовать захваченные учетные данные без риска быть обнаруженным.

Связанная, но отличающаяся уязвимость — это использование приложениями «статичных» токенов. Они выглядят как токены сессий и поначалу могут показаться таковыми, но на самом деле ими не являются. В этих приложениях каждому пользователю

присваивается токен, и этот же самый токен повторно выдается пользователю каждый раз, когда он входит в систему. Приложение всегда принимает этот токен как действительный, независимо от того, входил ли пользователь в систему недавно и был ли ему выдан этот токен. Подобные приложения на самом деле демонстрируют недопонимание всей концепции сессии и преимуществ, которые она дает для управления и контроля доступа. Иногда приложения работают так, реализуя плохо спроектированную функциональность «Запомнить меня», и статичный токен, соответственно, хранится в постоянном cookie (см. Главу 6). Иногда сами токены уязвимы для атак на предсказание, что делает уязвимость гораздо более серьезной. Вместо компрометации сессий вошедших в данный момент пользователей, успешная атака компрометирует навсегда учетные записи всех зарегистрированных пользователей.

Иногда также наблюдаются другие виды странного поведения приложения, которые демонстрируют фундаментальный дефект в связи между токенами и сессиями. Один из примеров — когда осмысленный токен конструируется на основе имени пользователя и случайного компонента. Например, рассмотрим токен:

dXNlcj1kYWY7cjE9MTMwOTQxODEyMTM0NTkwMTI=

который после декодирования из Base64 превращается в:

user=daf;r1=13094181213459012

После тщательного анализа компонента r1 мы можем прийти к выводу, что его невозможно предсказать на основе выборки значений. Однако, если логика обработки сессий в приложении нарушена, может оказаться, что злоумышленнику достаточно отправить любое действительное значение r1 и любое действительное значение user, чтобы получить доступ к сессии в контексте безопасности указанного пользователя. По сути, это уязвимость контроля доступа, поскольку решения о доступе принимаются на основе предоставленных пользователем данных, находящихся вне самой сессии (см. Главу 8). Она возникает потому, что приложение фактически использует токены сессий, чтобы обозначить, что запрашивающий установил некую действительную сессию с приложением. Однако контекст пользователя, в котором обрабатывается эта сессия, не является неотъемлемым свойством самой сессии, а определяется для каждого запроса с помощью других средств. В данном случае эти средства могут напрямую контролироваться запрашивающим.

ПРАКТИЧЕСКИЕ ШАГИ

1. Войдите в приложение дважды под одной и той же учетной записью, либо из разных процессов браузера, либо с разных компьютеров. Определите, остаются ли обе сессии активными одновременно. Если да, приложение поддерживает одновременные сессии, что позволяет злоумышленнику, скомпрометировавшему учетные данные другого пользователя, использовать

их без риска быть обнаруженным.

2. Войдите в систему и выйдите из нее несколько раз, используя одну и ту же учетную запись, либо из разных процессов браузера, либо с разных компьютеров. Определите, выдается ли каждый раз новый токен сессии или один и тот же токен при каждом входе. Если происходит второе, приложение на самом деле не использует полноценные сессии.

3. Если токены, по-видимому, имеют какую-либо структуру и смысл, попытайтесь отделить компоненты, которые могут идентифицировать пользователя, от тех, что кажутся непостижимыми (случайными). Попробуйте изменить любые компоненты, связанные с пользователем, в токене так, чтобы они ссылались на других известных пользователей приложения, и проверьте, будет ли полученный токен принят приложением и позволит ли он вам выдавать себя за этого пользователя.

ПОПРОБУЙТЕ!

http://mdsec.net/auth/382/
http://mdsec.net/auth/385/

Уязвимое завершение сессии

Правильное завершение сессий важно по двум причинам. Во-первых, поддержание жизненного цикла сессии настолько коротким, насколько это необходимо, сокращает окно возможностей, в течение которого злоумышленник может перехватить, угадать или неправомерно использовать действительный токен сессии. Во-вторых, это предоставляет пользователям средство для аннулирования существующей сессии, когда они в ней больше не нуждаются. Это позволяет им еще больше сократить это окно и взять на себя часть ответственности за безопасность своей сессии в общей компьютерной среде. Основные уязвимости в функциях завершения сессии связаны с невыполнением этих двух ключевых задач.

Некоторые приложения не обеспечивают эффективного истечения срока действия сессии. Однажды созданная сессия может оставаться действительной в течение многих дней после получения последнего запроса, прежде чем сервер в конечном итоге ее прекратит. Если токены уязвимы для какого-либо вида атаки на последовательность, которую особенно трудно проэксплуатировать (например, 100 000 предположений для каждого найденного действительного токена), злоумышленник все еще может оказаться в

состоянии захватить токены каждого пользователя, который обращался к приложению в недавнем прошлом.

Некоторые приложения не предоставляют эффективной функциональности выхода из системы:

- В некоторых случаях функция выхода просто не реализована. У пользователей нет способа заставить приложение аннулировать их сессию.

- В некоторых случаях функция выхода на самом деле не заставляет сервер аннулировать сессию. Сервер удаляет токен из браузера пользователя (например, выдав инструкцию Set-Cookie для его очистки). Однако, если пользователь продолжит отправлять этот токен, сервер все равно его примет.

- В худших случаях, когда пользователь нажимает «Выход», этот факт не передается на сервер, поэтому сервер не выполняет никаких действий. Вместо этого выполняется клиентский скрипт, который очищает cookie пользователя, что означает, что при последующих запросах пользователь возвращается на страницу входа. Злоумышленник, получивший доступ к этому cookie, мог бы использовать сессию так, как будто пользователь никогда не выходил из системы.

Некоторые приложения, не использующие аутентификацию, все равно содержат функциональность, которая позволяет пользователям накапливать конфиденциальные данные в своей сессии (например, приложение для покупок). Тем не менее, они обычно не предоставляют никакого эквивалента функции выхода для завершения сессии пользователями.

ПРАКТИЧЕСКИЕ ШАГИ

1. Не попадайтесь в ловушку анализа действий, которые приложение выполняет с клиентским токеном (таких как аннулирование cookie через новую инструкцию Set-Cookie, клиентский скрипт или атрибут времени истечения). С точки зрения завершения сессии, почти ничего не зависит от того, что происходит с токеном в браузере. Вместо этого выясните, реализовано ли истечение срока действия сессии на серверной стороне:

   a. Войдите в приложение, чтобы получить действительный токен сессии.

   b. Подождите некоторое время, не используя этот токен, а затем отправьте запрос к защищенной странице (например, «мои данные»), используя этот токен.

   c. Если страница отображается как обычно, токен все еще активен.

   d. Используйте метод проб и ошибок, чтобы определить, какова продолжительность тайм-аута истечения сессии, или можно ли использовать

токен спустя несколько дней после последнего запроса с ним. Burp Intruder можно настроить для автоматизации этой задачи, увеличивая временной интервал между последовательными запросами.

2. Определите, существует ли функция выхода и находится ли она на видном месте и доступна для пользователей. Если нет, пользователи более уязвимы, поскольку у них нет способа заставить приложение аннулировать их сессию.

3. Если функция выхода предоставлена, проверьте ее эффективность. После выхода из системы попытайтесь повторно использовать старый токен и определите, действителен ли он еще. Если да, пользователи остаются уязвимыми для некоторых атак по перехвату сессии, даже после того, как они «вышли из системы». Вы можете проверить это с помощью Burp Suite, выбрав недавний зависимый от сессии запрос из истории прокси и отправив его в Burp Repeater для повторной отправки после того, как вы вышли из приложения.

ПОПРОБУЙТЕ!

http://mdsec.net/auth/423/
http://mdsec.net/auth/439/
http://mdsec.net/auth/447/
http://mdsec.net/auth/452/
http://mdsec.net/auth/457/

Уязвимость клиента к перехвату токенов

Злоумышленник может атаковать других пользователей приложения в попытке перехватить или неправомерно использовать токен сессии жертвы различными способами:

- Очевидная полезная нагрузка для атак межсайтового скриптинга (XSS) — это запросить cookie пользователя, чтобы получить его токен сессии, который затем можно передать на произвольный сервер, контролируемый злоумышленником. Все различные вариации этой атаки подробно описаны в Главе 12.

- Различные другие атаки на пользователей могут быть использованы для перехвата сессии разными способами. При уязвимостях фиксации сессии (session fixation) злоумышленник подсовывает известный токен сессии пользователю, ждет, пока тот войдет в систему, а затем перехватывает его сессию. При атаках межсайтовой подделки запроса (CSRF) злоумышленник создает специальный запрос к приложению с контролируемого им веб-сайта и эксплуатирует тот факт, что браузер пользователя автоматически отправляет с этим запросом его текущий cookie. Эти атаки также описаны в Главе 12.

ПРАКТИЧЕСКИЕ ШАГИ

1. Определите любые уязвимости межсайтового скриптинга (XSS) в приложении и выясните, можно ли их проэксплуатировать для перехвата токенов сессий других пользователей (см. Главу 12).

2. Если приложение выдает токены сессий неаутентифицированным пользователям, получите токен и выполните вход. Если приложение не выдает новый токен после успешного входа, оно уязвимо для фиксации сессии (session fixation).

3. Даже если приложение не выдает токены сессий неаутентифицированным пользователям, получите токен, войдя в систему, а затем вернитесь на страницу входа. Если приложение готово вернуть эту страницу, несмотря на то, что вы уже аутентифицированы, отправьте данные для входа под другим пользователем, используя тот же самый токен. Если приложение не выдает новый токен после второго входа, оно уязвимо для фиксации сессии.

4. Определите формат токенов сессий, используемых приложением. Измените ваш токен на выдуманное, но корректно отформатированное значение и попытайтесь войти. Если приложение позволяет вам создать аутентифицированную сессию, используя выдуманный токен, оно уязвимо для фиксации сессии.

5. Если приложение не поддерживает вход в систему, но обрабатывает

конфиденциальную информацию пользователя (например, личные и платежные данные) и позволяет отображать ее после отправки (например, на странице «проверить мой заказ»), проведите предыдущие три теста по отношению к страницам, отображающим конфиденциальные данные. Если токен, установленный во время анонимного использования приложения, может быть впоследствии использован для получения конфиденциальной информации пользователя, приложение уязвимо для фиксации сессии.

6. Если приложение использует HTTP-cookie для передачи токенов сессий, оно вполне может быть уязвимо для межсайтовой подделки запроса (XSRF). Сначала войдите в приложение. Затем убедитесь, что запрос к приложению, исходящий со страницы другого приложения, приводит к отправке токена пользователя. (Эта отправка должна быть выполнена из окна того же процесса браузера, который использовался для входа в целевое приложение.) Попытайтесь выявить любые чувствительные функции приложения, параметры которых злоумышленник может определить заранее, и проэксплуатируйте это для выполнения несанкционированных действий в контексте безопасности целевого пользователя. Подробнее о том, как выполнять атаки XSRF, см. в Главе 13.

Слишком широкая область видимости cookie

Обычное простое описание того, как работают cookie, заключается в том, что сервер выдает cookie с помощью заголовка HTTP-ответа Set-cookie, а браузер затем повторно отправляет этот cookie в последующих запросах к тому же серверу с помощью заголовка Cookie. На самом деле, все обстоит несколько тоньше.

Механизм cookie позволяет серверу указывать как домен, так и путь URL, для которых каждый cookie будет повторно отправляться. Для этого используются атрибуты domain и path, которые могут быть включены в инструкцию Set-cookie.

Ограничения домена для cookie

Когда приложение, расположенное на foo.wahh-app.com, устанавливает cookie, браузер по умолчанию повторно отправляет этот cookie во всех последующих запросах на foo.wahh-app.com, а также на любые его поддомены, такие как admin.foo.wahh-app.com. Он не отправляет cookie на другие домены, включая родительский домен wahh-app.com и любые другие поддомены родительского домена, такие как bar.wahh-app.com.

Сервер может переопределить это поведение по умолчанию, включив атрибут domain в инструкцию Set-cookie. Например, предположим, что приложение на foo.wahh-app.com возвращает следующий HTTP-заголовок:

Set-cookie: sessionId=19284710; domain=wahh-app.com;

В этом случае браузер будет повторно отправлять этот cookie на все поддомены wahh-app.com, включая bar.wahh-app.com.

> **ПРИМЕЧАНИЕ** *Сервер не может указать любой произвольный домен с помощью этого атрибута. Во-первых, указанный домен должен быть либо тем же самым доменом, на котором работает приложение, либо его родительским доменом (непосредственным или более высокого уровня). Во-вторых, указанный домен не может быть доменом верхнего уровня, таким как .com или .co.uk, поскольку это позволило бы вредоносному серверу устанавливать произвольные cookie для любого другого домена. Если сервер нарушает одно из этих правил, браузер просто игнорирует инструкцию Set-cookie.*

Если приложение устанавливает излишне широкую область видимости домена для cookie, это может сделать приложение уязвимым для различных атак.

Например, рассмотрим приложение для ведения блогов, которое позволяет пользователям регистрироваться, входить в систему, писать посты и читать блоги других людей. Основное приложение расположено на домене wahh-blogs.com. Когда пользователи входят в приложение, они получают токен сессии в cookie, область видимости которого ограничена этим доменом. Каждый пользователь может создавать блоги, доступ к которым осуществляется через новый поддомен, начинающийся с его имени пользователя:

herman.wahh-blogs.com
solero.wahh-blogs.com

Поскольку cookie автоматически отправляются на каждый поддомен в пределах их области видимости, когда залогиненный пользователь просматривает блоги других, его токен сессии отправляется вместе с его запросами. Если авторам блогов разрешено размещать произвольный JavaScript в своих блогах (что обычно и происходит в реальных блог-платформах), злонамеренный блогер может украсть токены сессий других пользователей так же, как это делается при хранимой атаке межсайтового скриптинга (XSS) (см. Главу 12).

Проблема возникает потому, что создаваемые пользователями блоги размещаются на поддоменах основного приложения, которое обрабатывает аутентификацию и управление сессиями. В механизме HTTP-cookie нет возможности запретить отправку cookie, выданных основным доменом, на его поддомены.

Решение состоит в том, чтобы использовать другое доменное имя для основного приложения (например, www.wahh-blogs.com) и ограничить область видимости домена его cookie с токеном сессии этим полным доменным именем. В этом случае cookie сессии не будет отправляться, когда залогиненный пользователь просматривает блоги других.

Другая версия этой уязвимости возникает, когда приложение в явном виде устанавливает область видимости домена для своих cookie на родительский домен. Например, предположим, что критичное к безопасности приложение расположено на домене sensitiveapp.wahh-organization.com. При установке cookie оно в явном виде расширяет их область видимости домена следующим образом:

Set-cookie: sessionId=12df098ad809a5219; domain=wahh-organization.com

Следствием этого является то, что cookie с токеном сессии от чувствительного приложения будут отправляться, когда пользователь посещает каждый поддомен, используемый wahh-organization.com, включая:

www.wahh-organization.com
testapp.wahh-organization.com

Хотя эти другие приложения могут принадлежать той же организации, что и чувствительное приложение, нежелательно, чтобы cookie от чувствительного приложения отправлялись на другие приложения по нескольким причинам:

- Персонал, ответственный за другие приложения, может иметь другой уровень доверия, чем персонал, ответственный за чувствительное приложение.

- Другие приложения могут содержать функциональность, которая позволяет третьим сторонам получать значения cookie, отправленных в приложение, как в предыдущем примере с блогом.

- Другие приложения могли не подвергаться тем же стандартам безопасности или тестированию, что и чувствительное приложение (потому что они менее важны, не обрабатывают конфиденциальные данные или были созданы только для целей тестирования). Многие виды уязвимостей, которые могут существовать в этих приложениях (например, уязвимости межсайтового скриптинга), могут быть не важны для уровня безопасности этих приложений. Но они могут позволить внешнему злоумышленнику использовать небезопасное приложение для перехвата токенов сессий, созданных чувствительным приложением.

ПРИМЕЧАНИЕ *Разделение cookie на основе домена не такое строгое, как политика одинакового происхождения (same-origin policy) в целом (см. Главу 3). В дополнение к уже описанным проблемам при обработке имен хостов, браузеры игнорируют как протокол, так и номер порта при определении области видимости cookie. Если приложение использует один и тот же хост с недоверенным приложением и полагается на различие в протоколе или номере порта для их разделения, более свободная обработка cookie может подорвать это разделение. Любые cookie, выданные приложением, будут доступны недоверенному приложению, использующему тот же хост.*

ПРАКТИЧЕСКИЕ ШАГИ

Проанализируйте все cookie, выдаваемые приложением, и проверьте наличие любых атрибутов domain, используемых для контроля области видимости cookie.

1. Если приложение в явном виде расширяет область видимости своих cookie до родительского домена, оно может становиться уязвимым для атак через другие веб-приложения.

2. Если приложение устанавливает область видимости своих cookie на собственное доменное имя (или не указывает атрибут domain), оно все равно может быть уязвимо для атак со стороны приложений или функциональности, доступных через поддомены.

Определите все возможные доменные имена, которые будут получать cookie, выданные приложением. Установите, доступно ли через эти доменные имена какое-либо другое веб-приложение или функциональность, которую вы, возможно, сможете использовать для получения cookie, выданных пользователям целевого приложения.

Ограничения пути для cookie

Когда приложение, расположенное по пути /apps/secure/foo-app/index.jsp, устанавливает cookie, браузер по умолчанию повторно отправляет этот cookie во всех последующих запросах к пути /apps/secure/foo-app/, а также к любым его подкаталогам. Он не отправляет cookie в родительский каталог или на любые другие пути каталогов, существующие на сервере.

Как и в случае с ограничениями на основе домена, сервер может переопределить это поведение по умолчанию, включив атрибут path в инструкцию Set-cookie. Например, если приложение возвращает следующий HTTP-заголовок:

Set-cookie: sessionId=187ab023e09c00a881a; path=/apps/;

то браузер будет повторно отправлять этот cookie во все подкаталоги пути /apps/.

В отличие от ограничения области видимости по домену, это ограничение на основе пути гораздо строже, чем то, что накладывает политика одинакового происхождения (same-origin policy). Как таковое, оно почти полностью неэффективно, если используется в качестве механизма безопасности для защиты от недоверенных приложений, размещенных на том же домене. Клиентский код, выполняющийся на одном пути, может открыть окно или iframe, нацеленный на другой путь на том же домене, и может читать из этого окна и писать в него без каких-либо ограничений. Следовательно, получение cookie,

область видимости которого ограничена другим путем на том же домене, является относительно простой задачей. Подробнее см. в статье Амита Кляйна (Amit Klein):

http://lists.webappsec.org/pipermail/websecurity_lists.webappsec.org/2006-March/000843.html

Обеспечение безопасности управления сессиями

Защитные меры, которые должны принимать веб-приложения для предотвращения атак на свои механизмы управления сессиями, соответствуют двум широким категориям уязвимостей, затрагивающих эти механизмы. Для безопасного управления сессиями приложение должно надежным образом генерировать свои токены и защищать их на протяжении всего их жизненного цикла от создания до удаления.

Генерируйте надежные (стойкие) токены

Токены, используемые для повторной идентификации пользователя между последовательными запросами, должны генерироваться таким образом, чтобы не давать злоумышленнику, получившему большую выборку токенов, никакой возможности предсказывать или экстраполировать токены, выданные другим пользователям.

Наиболее эффективные механизмы генерации токенов — это те, которые:

- Используют чрезвычайно большой набор возможных значений.
- Содержат надежный источник псевдослучайности, обеспечивающий равномерное и непредсказуемое распределение токенов по всему диапазону возможных значений.

В принципе, любой элемент произвольной длины и сложности может быть угадан с помощью полного перебора (брутфорса) при наличии достаточного времени и ресурсов. Цель проектирования механизма для генерации надежных токенов заключается в том, чтобы сделать чрезвычайно маловероятным успех решительного злоумышленника с большими ресурсами пропускной способности и вычислительной мощности в угадывании хотя бы одного действительного токена в течение срока его действия.

Токены не должны представлять собой ничего, кроме идентификатора, используемого сервером для нахождения соответствующего объекта сессии, который будет использоваться для обработки запроса пользователя. Токен не должен содержать никакого смысла или структуры, ни в явном виде, ни в «обертке» из слоев кодирования или обфускации. Все данные о владельце и состоянии сессии должны храниться на сервере в объекте сессии, которому соответствует токен.

Будьте осторожны при выборе источника случайности. Разработчики должны знать, что различные доступные им источники, скорее всего, будут значительно отличаться по своей стойкости. Некоторые, как, например, java.util.Random, отлично подходят для многих целей, где требуется источник изменяющихся данных. Но их можно экстраполировать как вперед, так и назад с абсолютной уверенностью на основе одного-единственного элемента вывода. Разработчикам следует изучать математические свойства реальных алгоритмов, используемых в различных доступных источниках случайности, и читать соответствующую

документацию о рекомендуемом использовании различных API. В общем, если алгоритм не описан в явном виде как криптографически стойкий, следует считать его предсказуемым.

> **ПРИМЕЧАНИЕ** *Некоторым источникам высокой стойкости случайности требуется некоторое время, чтобы вернуть следующее значение в своей выходной последовательности, из-за шагов, которые они предпринимают для получения достаточной энтропии (например, из системных событий). Поэтому они могут не выдавать значения достаточно быстро для генерации токенов в некоторых высоконагруженных приложениях.*

Помимо выбора наиболее надежного из доступных источников случайности, хорошей практикой является добавление в качестве источника энтропии некоторой информации об индивидуальном запросе, для которого генерируется токен. Эта информация может и не быть уникальной для данного запроса, но она может быть эффективна для смягчения любых слабостей в используемом основном генераторе псевдослучайных чисел. Вот несколько примеров информации, которую можно включить:

- IP-адрес и номер порта источника, от которого был получен запрос.
- Заголовок User-Agent в запросе.
- Время запроса в миллисекундах.

Высокоэффективная формула для включения этой энтропии — это создание строки, которая объединяет (конкатенирует) псевдослучайное число, разнообразные специфичные для запроса данные, как перечислено выше, и секретную строку, известную только серверу и генерируемую заново при каждой его перезагрузке. Затем от этой строки берется подходящий хэш (например, SHA-256 на момент написания этой книги), чтобы получить управляемую строку фиксированной длины, которую можно использовать в качестве токена. (Размещение наиболее изменчивых элементов в начале входных данных для хэширования максимизирует «лавинный эффект» в алгоритме хэширования.)

> **ПОДСКАЗКА** *Выбрав алгоритм для генерации токенов сессий, полезно провести «мысленный эксперимент»: представьте, что ваш источник псевдослучайности сломан и всегда возвращает одно и то же значение. В этом случае сможет ли злоумышленник, получивший большую выборку токенов от приложения, экстраполировать токены, выданные другим пользователям? При использовании описанной здесь формулы в целом это крайне маловероятно, даже при полном знании используемого алгоритма. IP-адрес и номер порта источника, заголовок User-Agent и время запроса вместе создают огромное количество энтропии. И даже при полном знании этих данных злоумышленник не сможет воспроизвести соответствующий токен, не зная секретной строки, используемой сервером.*

Защищайте токены на протяжении всего их жизненного цикла

Теперь, когда вы создали надежный токен, значение которого невозможно предсказать, этот токен необходимо защищать на протяжении всего его жизненного цикла от создания до удаления, чтобы гарантировать, что он не будет раскрыт никому, кроме пользователя, которому он выдан:

- Токен должен передаваться только по HTTPS. Любой токен, переданный в открытом виде, следует считать «зараженным» (tainted), то есть не обеспечивающим уверенности в личности пользователя. Если для передачи токенов используются HTTP-cookie, они должны быть помечены флагом secure, чтобы браузер пользователя никогда не передавал их по HTTP. Если это возможно, HTTPS следует использовать для каждой страницы приложения, включая статический контент, такой как страницы помощи, изображения и так далее. Если это нежелательно и HTTP-сервис все же реализован, приложение должно перенаправлять любые запросы к конфиденциальному контенту (включая страницу входа) на HTTPS-сервис. Статические ресурсы, такие как страницы помощи, обычно не являются конфиденциальными, и к ним можно обращаться без аутентифицированной сессии. Следовательно, использование безопасных cookie можно подкрепить инструкциями по области видимости cookie, чтобы предотвратить их отправку в запросах к этим ресурсам.

- Токены сессий никогда не должны передаваться в URL-адресе, поскольку это предоставляет простой способ для атак фиксации сессии и приводит к появлению токенов в многочисленных механизмах журналирования. В некоторых случаях разработчики используют этот метод для реализации сессий в браузерах с отключенными cookie. Однако лучшим способом достижения этого является использование POST-запросов для всей навигации и хранение токенов в скрытом поле HTML-формы.

- Должна быть реализована функциональность выхода. Она должна уничтожать все ресурсы сессии, хранящиеся на сервере, и аннулировать токен сессии.

- Должно быть реализовано истечение срока действия сессии после подходящего периода неактивности (например, 10 минут). Это должно приводить к тому же поведению, как если бы пользователь явно вышел из системы.

- Одновременные входы должны быть запрещены. Каждый раз, когда пользователь входит в систему, должен выдаваться другой токен сессии, а любая существующая сессия, принадлежащая этому пользователю, должна быть уничтожена, как если бы он из нее вышел. При этом старый токен может храниться в течение некоторого времени. Любые последующие запросы, полученные с использованием старого токена, должны возвращать пользователю предупреждение о безопасности, гласящее, что сессия была прекращена, потому что он вошел в систему с другого места.

- Если приложение содержит какую-либо административную или диагностическую функциональность, позволяющую просматривать токены сессий, эта функциональность должна быть надежно защищена от несанкционированного доступа. В большинстве случаев нет необходимости, чтобы эта функциональность отображала фактический токен сессии. Вместо этого она должна содержать достаточно деталей о владельце сессии для выполнения любых задач поддержки и диагностики, не разглашая токен сессии, который пользователь отправляет для идентификации.

- Область видимости домена и пути для cookie сессий приложения должна быть установлена настолько ограниченно, насколько это возможно. Cookie со слишком широкой областью видимости часто генерируются плохо настроенными платформами веб-приложений или веб-серверами, а не самими разработчиками. Никакие другие веб-приложения или недоверенная функциональность не должны быть доступны через доменные имена или пути URL, которые включены в область видимости cookie приложения. Особое внимание следует уделить любым существующим поддоменам того доменного имени, которое используется для доступа к приложению. В некоторых случаях, чтобы гарантировать отсутствие этой уязвимости, может потребоваться изменить схему именования доменов и путей, используемую различными приложениями в организации.

Следует принимать специальные меры для защиты механизма управления сессиями от разнообразных атак, целью которых могут стать пользователи приложения:

- Кодовая база приложения должна проходить тщательный аудит для выявления и устранения любых уязвимостей межсайтового скриптинга (XSS) (см. Главу 12). Большинство таких уязвимостей могут быть использованы для атаки на механизмы управления сессиями. В частности, хранимые (или второго порядка) XSS-атаки обычно могут быть использованы для обхода всех мыслимых защит от неправомерного использования и перехвата сессий.

- Произвольные токены, отправленные пользователями, которых сервер не распознает, не должны приниматься. Токен должен быть немедленно аннулирован в браузере, а пользователь — возвращен на стартовую страницу приложения.

- Атаки межсайтовой подделки запроса и другие сессионные атаки можно усложнить, требуя двухэтапного подтверждения и/или повторной аутентификации перед выполнением критически важных действий, таких как перевод средств.

- От атак межсайтовой подделки запроса можно защититься, не полагаясь исключительно на HTTP-cookie для передачи токенов сессий. Уязвимость существует потому, что cookie автоматически отправляются браузером независимо от того, что вызвало запрос. Если токены всегда передаются в скрытом поле HTML-формы, злоумышленник не сможет создать форму, отправка которой вызовет несанкционированное действие, если он уже не знает значение токена. В этом случае он может просто провести легкую атаку по перехвату сессии. Токены для

каждой страницы также могут помочь предотвратить эти атаки (см. следующий раздел).

- Новая сессия всегда должна создаваться после успешной аутентификации, чтобы смягчить последствия атак фиксации сессии. В случаях, когда приложение не использует аутентификацию, но позволяет отправлять конфиденциальные данные, угрозу, исходящую от атак фиксации, устранить сложнее. Один из возможных подходов — сделать последовательность страниц, где отправляются конфиденциальные данные, как можно короче. Затем вы можете создавать новую сессию на первой странице этой последовательности (при необходимости копируя из существующей сессии нужные данные, например, содержимое корзины). Или вы могли бы использовать токены для каждой страницы (описано в следующем разделе), чтобы помешать злоумышленнику, знающему токен, использованный на первой странице, получить доступ к последующим. За исключением случаев крайней необходимости, личные данные не должны отображаться обратно пользователю. Даже там, где это требуется (например, на странице «подтвердить заказ», где показываются адреса), конфиденциальные элементы, такие как номера кредитных карт и пароли, никогда не должны отображаться обратно пользователю и всегда должны быть замаскированы в исходном коде ответа приложения.

Токены для каждой страницы (постраничные токены)

Более детального (гранулярного) контроля над сессиями можно достичь, а многие виды сессионных атак — усложнить или сделать невозможными, используя токены для каждой страницы в дополнение к токенам сессии. В этом случае новый токен страницы создается каждый раз, когда пользователь запрашивает страницу приложения (в отличие, например, от изображения) и передается клиенту в cookie или скрытом поле HTML-формы.

Каждый раз, когда пользователь делает запрос, токен страницы проверяется на соответствие последнему выданному значению, в дополнение к обычной проверке основного токена сессии. В случае несоответствия вся сессия прекращается. Многие из самых критичных к безопасности веб-приложений в Интернете, такие как онлайн-банки, применяют токены для каждой страницы для обеспечения повышенной защиты своего механизма управления сессиями, как показано на Рисунке 7-12.

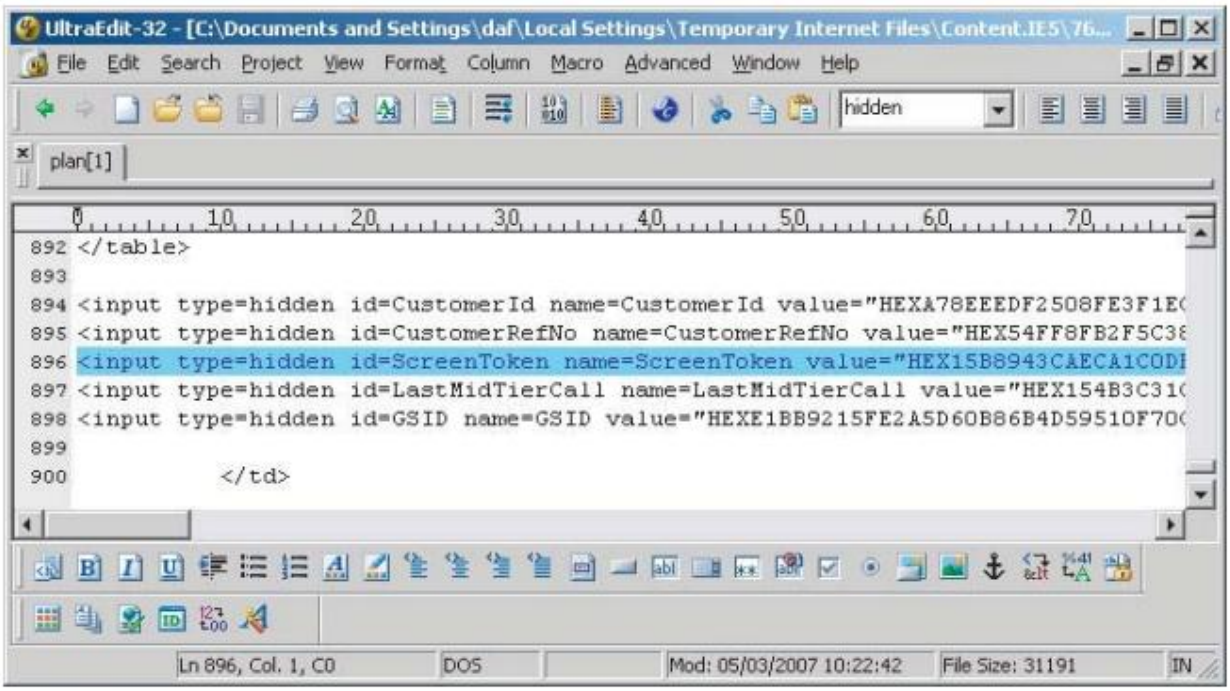


Рисунок 7-12: Токены для каждой страницы (постраничные токены), используемые в банковском приложении

Использование токенов для каждой страницы действительно накладывает некоторые ограничения на навигацию (например, на использование кнопок «назад» и «вперед» и просмотр в нескольких окнах). Однако это эффективно предотвращает атаки фиксации сессии и гарантирует, что одновременное использование перехваченной сессии легитимным пользователем и злоумышленником будет быстро заблокировано после того, как оба сделают по одному запросу. Токены для каждой страницы также могут быть использованы для отслеживания местоположения и перемещений пользователя по приложению. Их также можно использовать для обнаружения попыток доступа к функциям не в определенной последовательности, что помогает защититься от определенных дефектов контроля доступа (см. Главу 8).

Журналируйте, отслеживайте и оповещайте

Функциональность управления сессиями приложения должна быть тесно интегрирована с его механизмами журналирования, мониторинга и оповещения, чтобы предоставлять надлежащие записи об аномальной активности и позволять администраторам предпринимать защитные действия при необходимости:

- Приложение должно отслеживать запросы, содержащие недействительные токены. За исключением самых предсказуемых случаев, успешная атака по подбору токенов, выданных другим пользователям, обычно включает в себя отправку

большого количества запросов с недействительными токенами, что оставляет заметный след в логах приложения.

- Атаки полного перебора (брутфорс) на токены сессий трудно заблокировать полностью, поскольку для прекращения атаки нельзя отключить какую-либо конкретную учетную запись или сессию. Одно из возможных действий — блокировать IP-адреса источников на определенное время при получении некоторого количества запросов с недействительными токенами. Однако это может быть неэффективно, когда запросы одного пользователя исходят с нескольких IP-адресов (например, у пользователей AOL) или когда запросы нескольких пользователей исходят с одного IP-адреса (например, у пользователей за прокси-сервером или брандмауэром, выполняющим трансляцию сетевых адресов).

- Даже если атаки перебором на сессии невозможно эффективно предотвратить в реальном времени, ведение подробных логов и оповещение администраторов позволяет им расследовать атаку и предпринимать соответствующие действия, когда это возможно.

- Где это возможно, пользователей следует оповещать об аномальных событиях, связанных с их сессией, таких как одновременные входы или очевидный перехват (обнаруженный с помощью токенов для каждой страницы). Даже если компрометация уже произошла, это позволяет пользователю проверить, не были ли совершены какие-либо несанкционированные действия, например, перевод средств.

Реактивное завершение сессии

Механизм управления сессиями можно использовать как высокоэффективную защиту от многих других видов атак на приложение. Некоторые критичные к безопасности приложения, такие как онлайн-банки, крайне агрессивно завершают сессию пользователя каждый раз, когда он отправляет аномальный запрос. Примерами являются любой запрос, содержащий измененное скрытое поле HTML-формы или параметр строки запроса URL, любой запрос, содержащий строки, связанные с SQL-инъекциями или атаками межсайтового скриптинга, и любой ввод пользователя, который обычно был бы заблокирован проверками на стороне клиента, такими как ограничения длины.

Конечно, любые реальные уязвимости, которые могут быть проэксплуатированы с помощью таких запросов, необходимо устранять в их источнике. Но принуждение пользователей к повторной аутентификации при каждой отправке недействительного запроса может замедлить процесс исследования приложения на предмет уязвимостей на много порядков, даже при использовании автоматизированных методик. Если остаточные уязвимости все еще существуют, их обнаружение кем-либо в реальных условиях становится гораздо менее вероятным.

В тех случаях, когда реализуется такого рода защита, также рекомендуется, чтобы ее можно было легко отключать для целей тестирования. Если легитимный тест на проникновение замедляется так же, как и реальный злоумышленник, его эффективность резко снижается. Кроме того, весьма вероятно, что наличие этого механизма приведет к тому, что в рабочем коде останется больше уязвимостей, чем если бы он отсутствовал.

ПРАКТИЧЕСКИЕ ШАГИ

Если атакуемое вами приложение использует такого рода защитную меру, вы можете обнаружить, что исследование приложения на предмет многих распространенных уязвимостей становится чрезвычайно трудоемким. Утомительная необходимость входить в систему после каждого неудачного теста и заново переходить к исследуемой части приложения быстро заставит вас сдаться.

В этой ситуации вы часто можете использовать автоматизацию для решения проблемы. При использовании Burp Intruder для проведения атаки вы можете использовать функцию «Obtain Cookie» (Получить Cookie), чтобы выполнять новый вход в систему перед отправкой каждого тестового случая и использовать новый токен сессии (при условии, что вход одноэтапный). При ручном просмотре и исследовании приложения вы можете использовать возможности расширения Burp Proxy через интерфейс IBurpExtender. Вы можете создать расширение, которое обнаруживает, когда приложение выполнило принудительный выход, автоматически входит в систему заново и возвращает новую сессию и страницу в браузер, опционально с всплывающим сообщением, чтобы сообщить вам о произошедшем. Хотя это ни в коем случае не устраняет проблему, в определенных случаях это может существенно ее смягчить.

Заключение

Механизм управления сессиями предоставляет богатый источник потенциальных уязвимостей для ваших атак на приложение. Из-за его фундаментальной роли в идентификации одного и того же пользователя между несколькими запросами, взломанная функция управления сессиями обычно предоставляет ключи от королевства. Запрыгнуть в сессии других пользователей — это хорошо. Перехват сессии администратора — еще лучше; обычно это позволяет вам скомпрометировать все приложение целиком.

Вы можете ожидать встретить широкий спектр дефектов в реальных функциях управления сессиями. Когда применяются самописные механизмы, возможные уязвимости и векторы атак могут показаться бесконечными. Самый важный урок, который следует извлечь из этой темы, — быть терпеливым и настойчивым. Довольно много механизмов управления сессиями, которые кажутся надежными при первом осмотре, оказываются несостоятельными при более внимательном анализе. Расшифровка метода, который приложение использует для генерации своей последовательности кажущихся случайными токенов, может потребовать времени и изобретательности. Но, учитывая награду, это обычно инвестиция, которая того стоит.