A Web Application Hacker’s Handbook

A Web Application Hacker’s Methodology

Эта глава содержит подробную пошаговую методологию, которой вы можете следовать при атаке веб-приложения. Он охватывает все категории уязвимостей и методов атаки, описанных в этой книге. Выполнение всех шагов в этой методологии не гарантирует, что вы обнаружите все уязвимости в данном приложении. Тем не менее, он обеспечит вам хорошую уверенность в том, что вы исследовали все необходимые области поверхности атаки приложения и обнаружили как можно больше проблем, учитывая имеющиеся у вас ресурсы.

Рисунок 21-1 иллюстрирует основные области работы, которые описывает эта методология. Мы углубимся в эту диаграмму и проиллюстрируем разделение задач, которые включает в себя каждая область. Числа на диаграммах соответствуют иерархическим
нумерованный список, используемый в методологии, так что вы можете легко перейти к действиям, связанным с областью спецификации.
Методология представлена как множество задач, которые организованы и упорядочены в соответствии с логическими взаимозависимостями между ними. Насколько это возможно, эти взаимозависимости выделены в описаниях задач. Однако на практике вам часто нужно мыслить творчески о направлении
в котором ваша деятельность должна идти и позволять им руководствоваться тем, что вы узнайте о приложении, на которое вы нападаете. Например:

- Информация, собранная на одном этапе, может позволить вам вернуться к более раннему этапу и сформулировать более целенаправленные атаки. Например, ошибка контроля доступа, которая позволяет вам получить список всех пользователей, может позволить вам выполнить более эффективную атаку, угадывающую пароль, на функцию аутентификации.
- Обнаружение ключевой уязвимости в одной области приложения может позволить вам сократить часть работы в других областях. Например, уязвимость раскрытия информации может позволить вам выполнять проверку кода ключевых функций приложения, а не проверять их исключительно black-box способом.
- Результаты вашего тестирования в некоторых областях могут выделить шаблоны повторяющихся уязвимостей, которые вы можете сразу исследовать в других областях. Например, общий дефект во входных фильтрах проверки приложения может позволить вам быстро найти обход его защиты от нескольких различных категорий атаки.

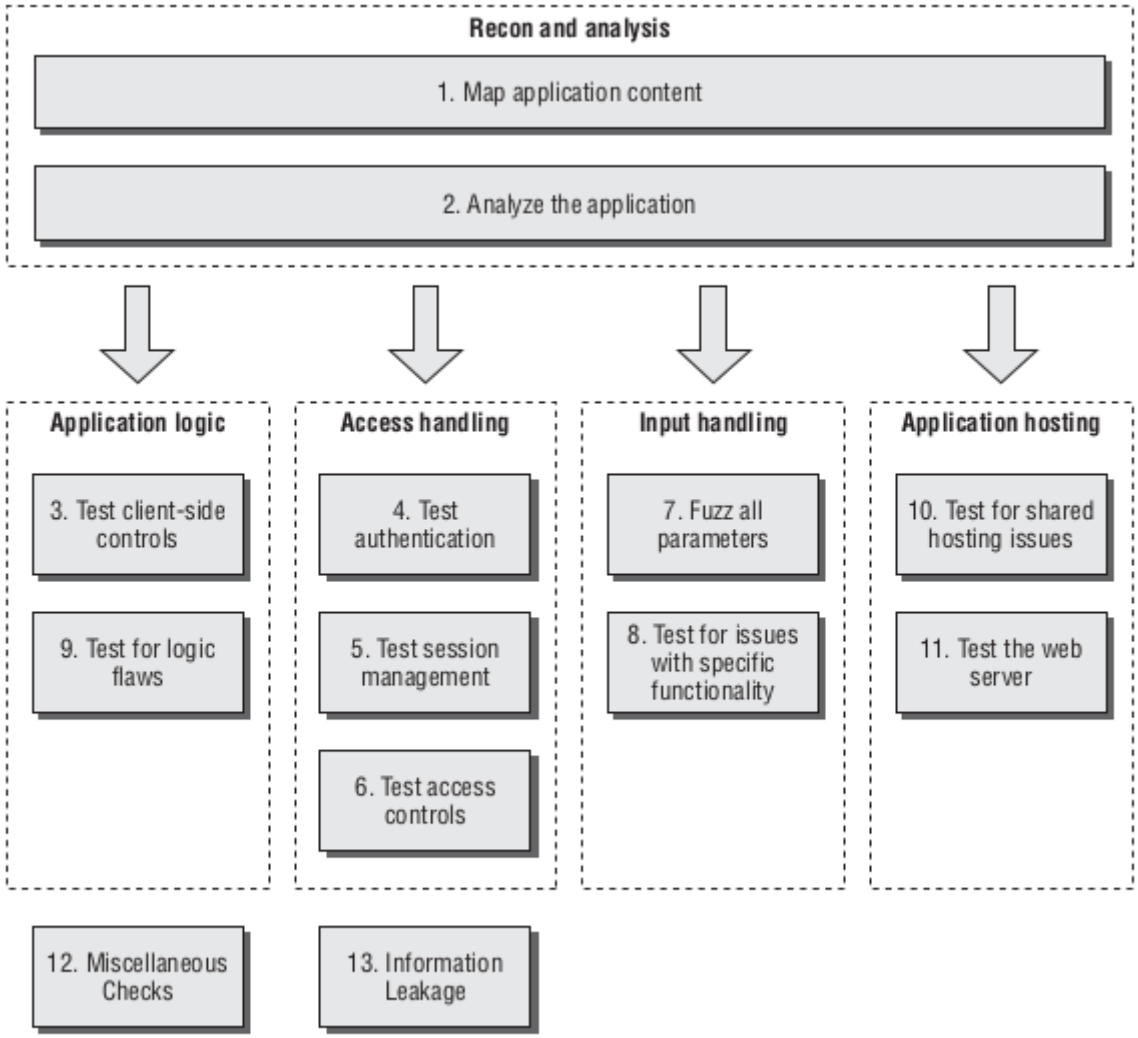

Figure 21-1: The main areas of work involved in the methodology

Используйте шаги в этой методологии, чтобы направлять вашу работу, и в качестве контрольного списка, чтобы избежать упущений, но не чувствуйте себя обязанным придерживаться их слишком жестко. Помните следующее: задачи, которые мы описываем, в основном стандартные и ортодоксальные; самые впечатляющие атаки на веб-приложения всегда включают в себя мышление за их пределами.

Общие рекомендации

При выполнении подробных задач, связанных с атакой веб-приложения, всегда следует учитывать некоторые общие соображения. Они могут применяться ко всем различным областям, которые вам необходимо изучить, и методам, которые вам необходимо выполнить.

- Помните, что несколько символов имеют особое значение в разных частях HTTP-запроса. Когда вы изменяете данные в запросах, вы должны URL-кодировать эти символы, чтобы убедиться, что они интерпретируются так, как вы намереваетесь :

- & используется для разделения параметров в строке запроса URL и теле сообщения. Чтобы вставить буквенный и символьный, вы должны кодировать его как% 26 .

- = используется для разделения имени и значения каждого параметра в строке запроса URL и теле сообщения. Чтобы вставить буквенный символ =, вы должны кодировать его как % 3d .

- ? используется для обозначения начала строки запроса URL ? Чтобы вставить буквальный . персонаж, вы должны кодировать это как % 3f

- space используется для обозначения конца URL в первой строке запросов и может указывать конец значения cookie в заголовке Cookie. Чтобы вставить буквальное пространство, вы должны кодировать его как% 20 или + .

- Поскольку + представляет закодированное space, чтобы вставить буквальный символ +, вы должны кодировать его как %2b

- ; используется для разделения отдельных файлов cookie в заголовке Cookie. Вставить буквальный ; персонаж, вы должны закодировать это как % 3b .

- # используется для пометки идентификатора фрагмента в URL. Если вы введете этот символ в URL в вашем браузере, он эффективно усекает URL, который отправляется на сервер. Чтобы вставить буквальный символ #, вы должны закодировать его как% 23

- % используется в качестве префикса в схеме кодирования URL. Чтобы вставить буквальный символ %, вы должны кодировать его как% 25 .

- Любые непечатаемые символы, такие как нулевые байты и новые строки, должны, конечно, кодироваться URL с использованием их кода символов ASCII - в данном случае, как % 00 и %0a соответственно.

- Кроме того, обратите внимание, что ввод данных в кодировке URL в форму обычно заставляет ваш браузер выполнять другой уровень кодирования. Например, отправка %00 в форме, вероятно, приведет к отправке значения %2500 на сервер. По этой причине обычно лучше всего наблюдать окончательный запрос в перехватывающем прокси.

- Многие тесты на наличие общих уязвимостей веб-приложений включают отправку различных созданных входных строк и мониторинг ответов приложения на аномалии, которые указывают на наличие уязвимости. В некоторых случаях ответ приложения на конкретный запрос содержит подпись конкретной уязвимости, независимо от того, был ли подан триггер для этой уязвимости. В любом случае, когда конкретный созданный ввод приводит к поведению, связанному с уязвимостью (например, конкретным сообщением об ошибке), вам следует дважды проверить, вызывает ли отправка доброкачественного ввода в соответствующий параметр такое же поведение. Если это произойдет, ваш предварительный результат, вероятно, является ложным положительным.

- Приложения обычно накапливают количество состояний по предыдущим запросам, что влияет на то, как они реагируют на дальнейшие запросы. Иногда, когда вы пытаетесь исследовать предварительную уязвимость и выделить точную причину определенного фрагмента аномального поведения, вы должны устранить последствия любого накопленного состояния. Для этого обычно достаточно начать новый сеанс с новым процессом браузера, перейти к месту наблюдаемой аномалии, используя только доброкачественные запросы, а затем повторно отправить ваш созданный ввод. Вы

можете часто копировать эту меру, настраивая части ваших запросов, содержащие файлы cookie и кэширующую информацию. Кроме того, вы можете использовать такой инструмент, как Burp Repeater, чтобы изолировать запрос, внести в него конкретные изменения и переиздать его столько раз, сколько вам потребуется.

- Некоторые приложения используют конфигурацию с балансировкой нагрузки, в которой последовательные HTTP-запросы могут обрабатываться различными внутренними серверами в Интернете, презентациях, данных или других уровнях. Различные серверы могут иметь небольшие различия в конфигурации, которые влияют на ваши результаты. Кроме того, некоторые успешные атаки приведут к изменению состояния конкретного сервера, который обрабатывает ваши запросы, например, к созданию нового файла в корневом веб-файле. Чтобы изолировать эффекты конкретных действий, может потребоваться выполнить несколько идентичных запросов подряд, проверяя результат каждого, пока ваш запрос не будет обработан соответствующим сервером.

Предполагая, что вы внедряете эту методологию как часть консультационного задания, Вы всегда должны обязательно выполнять обычное упражнение по определению, чтобы точно согласовать, какие имена хостов, URL, и функциональность должна быть включена, и существуют ли какие-либо ограничения на типы тестирования, которые вам разрешено выполнять. Вы должны информировать владельца приложения о внутренних рисках, связанных с проведением любого вида тестирования на проникновение
цель черного ящика. Посоветуйте владельцу сделать резервную копию любых важных данных перед вами
начать свою работу.

1 Map the Application’s Content

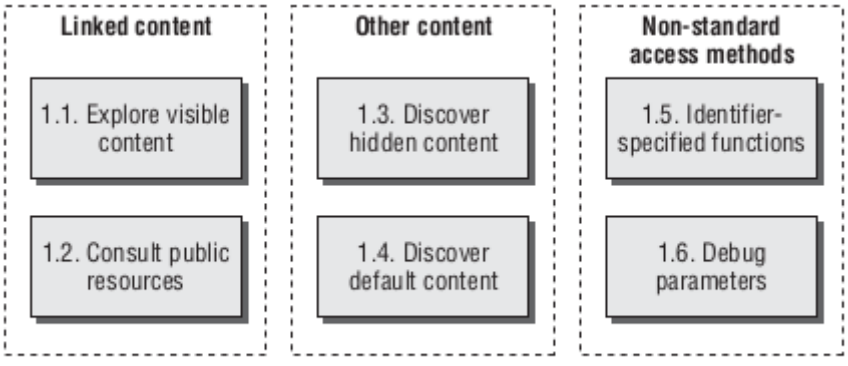

Figure 21-2: Mapping the application’s content

1.1 Explore Visible Content

1.1.1 Настройте свой браузер, чтобы использовать ваш любимый интегрированный инструмент прокси. И Burp, и WebScarab могут использоваться для пассивного spidering сайта путем мониторинга и анализа веб-контента, обрабатываемого прокси-сервером.

1.1.2 Если вы сочтете это полезным, настройте свой браузер на использование расширения, такого как IEWatch, для мониторинга и анализа содержимого HTTP и HTML, обрабатываемого браузером.

1.1.3 пройти через приложение обычным способом, посещая каждую ссылку и URL, отправляя каждую форму и проходя через все многошаговые функции до завершения. Попробуйте просмотреть с включенным и отключенным JavaScript
и с включенными и отключенными файлами cookie. Многие приложения могут обрабатывать различные конфигурации браузера, и вы можете получить различный контент
и пути кода в приложении.

1.1.4 Если приложение использует аутентификацию, и у вас есть или вы можете создать учетную запись для входа, используйте ее для доступа к защищенной функциональности.

1.1.5 При просмотре отслеживайте запросы и ответы, проходящие через ваш перехватчик, чтобы получить представление о видах данных
отправка и способы, которыми клиент используется для контроля поведения приложения на стороне сервера.

1.1.6 Просмотрите карту сайта, сгенерированную пассивной прокладкой, и определите любой контент или функции, которые вы не прошли через свой браузер. Из результатов паука установите, где был обнаружен каждый элемент (например, в Burp Spider проверьте детали Linked From). Получите доступ к каждому элементу с помощью браузера, чтобы паук анализировал ответ с сервера, чтобы идентифицировать любой дополнительный контент. Продолжайте этот шаг рекурсивно, пока не будет идентифицирован дальнейший контент или функциональность.

1.1.7 Когда вы закончите просмотр вручную и пассивно, вы можете использовать своего паука для активного сканирования приложения, используя набор обнаруженных URL-адресов в качестве семян. Иногда это может раскрыть дополнительный контент, который вы упустили при работе вручную. Перед автоматическим сканированием сначала определите любые URL-адреса, которые опасны или могут нарушить сеанс приложения, а затем настройте паука, чтобы исключить их из сферы действия.

1.2 Consult Public Resources

1.2.1 Использовать поисковые системы и архивы в Интернете (например, Wayback Machine) определить, какой контент они проиндексировали и сохранили для вашего целевого приложения.

1.2.2 Используйте расширенные параметры поиска, чтобы повысить эффективность вашего исследования.
Например, в Google вы можете использовать site: для извлечения всего контента для вашего целевого сайта и ссылки: для извлечения других сайтов, которые ссылаются на него. Если ваш поиск идентифицирует контент, который больше не присутствует в реальном приложении, вы все равно сможете просмотреть его из кэша поисковой системы. Этот

старый контент может содержать ссылки на дополнительные ресурсы, которые еще не были удалены.

1.2.3 Выполнять поиск по любым именам и адресам электронной почты, которые вы обнаружили в содержимом приложения, например по контактной информации. Включите элементы, не представленные на экране, такие как комментарии HTML. Помимо веб-поиска, выполняйте новости и групповые поиски. Ищите любые технические подробности, размещенные на интернет-форумах относительно целевого приложения и его поддерживающей инфраструктуры.

1.2.4 Просмотрите все опубликованные файлы WSDL, чтобы создать список имен функций и значений параметров, потенциально используемых приложением

1.3 Discover Hidden Content

1.3.1 Подтвердите, как приложение обрабатывает запросы на несуществующие элементы. Сделайте несколько ручных запросов на известные действительные и недействительные ресурсы и сравните ответы сервера, чтобы установить простой способ определить, когда элемент не существует.

1.3.2 Получить списки общих имен файлов и каталогов и общих расширений файлов. Добавьте к этим спискам все элементы, фактически наблюдаемые в приложениях, а также элементы, выведенные из них. Постарайтесь понять соглашения об именах, используемые разработчиками приложений. Например, если есть страницы, называемые *AddDocument.jsp* и *ViewDocument.jsp*, также могут быть страницы, называемые *EditDocument.jsp* и *RemoveDocument.jsp* .

1.3.3 Просмотрите весь код на стороне клиента, чтобы определить любые подсказки о скрытом контенте на стороне сервера, включая комментарии HTML и отключенные элементы формы.

1.3.4 Используя методы автоматизации, описанные в главе 14, делайте большое количество запросов на основе вашего каталога, имени файла и списков расширений файлов. Мониторинг ответов сервера на вопрос, какие элементы присутствуют и доступны.

1.3.5 Выполняйте эти упражнения по обнаружению контента рекурсивно, используя новый перечисленный контент и шаблоны в качестве основы для дальнейшего дифференцирования, направленного пользователем, и дальнейшего автоматического обнаружения.

1.4 Discover Default Content

1.4.1 Запустите Nikto с веб-сервера, чтобы обнаружить любой контент по умолчанию или известный контент, который присутствует. Используйте опции Nikto, чтобы максимизировать его эффективность. Например, вы можете использовать опцию –root, чтобы указать каталог для проверки содержимого по умолчанию, или -404, чтобы указать строку, которая идентифицирует пользовательскую страницу «Файл не найден».

1.4.2. Проверять любые потенциально интересные результаты вручную, чтобы устранить любые ложные срабатывания в результатах.

1.4.3 Запросите корневой каталог сервера, указав IP-адрес в заголовке хоста, и определите, отвечает ли приложение другим содержимым. Если это так, запустите сканирование Nikto по IP-адресу, а также по имени сервера.

1.4.4 Сделать запрос в корневой каталог сервера, указав диапазон заголовков *User-Agent*, как показано на сайте *www.useragentstring.com/pages/useragentstring.php* .

1.5 Enumerate Identifi er-Specifi ed Functions

1.5.1. Определить любые случаи, когда к конкретным функциям приложения обращаются, передавая идентификатор функции в параметре запроса (например, */admin.jsp?action = editUser* или */main.php?func = A21*).

1.5.2. Применять методы обнаружения контента, используемые на шаге 1.3, к механизму, используемому для доступа к отдельным функциям. Например, если приложение использует параметр, содержащий имя функции, сначала определите его поведение, когда указана недопустимая функция, и попытайтесь установить простой способ идентификации, когда запрашивается допустимая функция. Скомпилируйте список общих имен функций или цикл через синтаксический диапазон идентификаторов, которые, как ожидается, используются. Автоматизируйте упражнение, чтобы перечислить действительные функции как можно быстрее и проще.

1.5.3 Если применимо, скомпилируйте карту содержимого приложения на основе функциональных путей, а не URL-адресов, показывая все перечисленные функции и логические пути и зависимости между ними. (См. Главу 4 для примера.)

1.6 Test for Debug Parameters

1.6.1 Выберите одну или несколько страниц приложения или функций, в которых могут быть реализованы скрытые параметры отладки (например, debug = true). Скорее всего, они будут отображаться в ключевых функциях, таких как логин, поиск и загрузка или загрузка файлов.

1.6.2 Использовать списки общих имен параметров debug (таких как debug, test, hide и source) и общие значения (такие как true, yes, on и 1). Итерация во всех их перестановках, отправка каждой пары name / value каждой целевой функции. Для запросов POST укажите параметр как в строке запроса URL, так и в теле запроса. Используйте методы, описанные в главе 14, чтобы автоматизировать это упражнение. Например, вы можете использовать тип атаки cluster bomb в Burp Intruder, чтобы объединить все перестановки двух списков полезных нагрузок.

1.6.3. Просмотрите ответы приложения на любые аномалии, которые могут указывать на то, что добавленный параметр оказал влияние на обработку приложения.

2 Analyze the Application

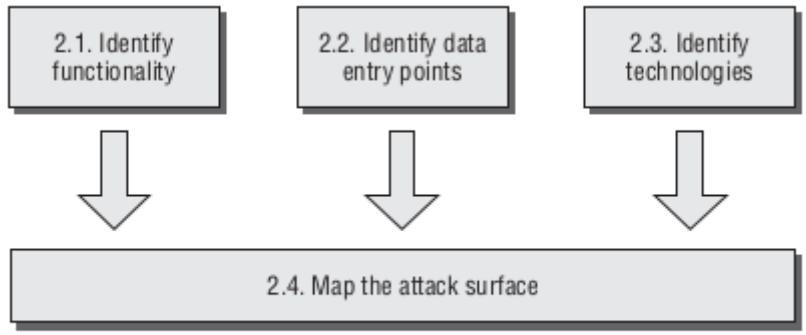

Figure 21-3: Analyzing the application

2.1 Identify Functionality

2.1.1. Определите основные функции, для которых было создано приложение, и действия, для которых каждая функция предназначена для выполнения при использовании по назначению.

2.1.2. Определить основные механизмы безопасности, используемые приложением, и то, как они работают. В частности, понять ключевые механизмы, которые обрабатывают аутентификацию, управление сеансами и контроль доступа, а также функции, которые их поддерживают, такие как регистрация пользователей и восстановление учетной записи.

2.1.3. Определить все более периферийные функции и поведение, такие как использование перенаправлений, внешних ссылок, сообщений об ошибках, а также административных функций и функций ведения журнала.

2.1.4. Определить любую функциональность, которая отличается от стандартного внешнего вида графического интерфейса, именования параметров или навигационного механизма, используемого в других местах приложения, и выделить его для углубленного тестирования

2.2 Identify Data Entry Points

2.2.1. Определить все различные точки входа, которые существуют для введения пользовательского ввода в обработку приложения, включая URL-адреса, параметры строки запроса, данные POST, файлы cookie и другие заголовки HTTP, обрабатываемые приложением.

2.2.2 Изучите любые настраиваемые механизмы передачи данных или кодирования, используемые приложением, такие как нестандартный формат строки запроса. Поймите, инкапсулируют ли представляемые данные имена и значения параметров, или используется ли альтернативное средство представления.

2.2.3. Определить любые out of band каналы, по которым управляемые пользователем или другие сторонние данные вводятся в обработку приложения. Примером является веб-почта-приложение, которое обрабатывает и отображает сообщения, полученные через SMTP

2.3 Identify the Technologies Used

2.3.1. Определите каждую из различных технологий, используемых на стороне клиента, таких как формы, scripts, файлы cookie, апплеты Java, элементы управления ActiveX и объекты Flash

2.3.2 Насколько это возможно, установите, какие технологии используются на стороне сервера, включая языки сценариев, платформы приложений и взаимодействие с внутренними компонентами, такими как базы данных и системы электронной почты.

2.3.3 Проверьте заголовок HTTP-сервера, возвращаемый в ответах приложения, а также проверьте наличие других идентификаторов программного обеспечения, содержащихся в пользовательских заголовках HTTP или комментариях исходного кода HTML. Обратите внимание, что в некоторых случаях различные области приложения обрабатываются разными внутренними компонентами, поэтому могут быть получены разные баннеры.

2.3.4 Запустите инструмент Httprint, чтобы снять отпечатки с веб-сервера.

2.3.5 Просмотрите результаты ваших упражнений по отображению контента, чтобы определить любые интересные расширения файлов, каталоги или другие последующие URL-адреса, которые могут дать подсказки о технологиях, используемых на сервере. Просмотрите имена любых сеансовых токенов и других выпущенных файлов cookie. Используйте Google для поиска технологий, связанных с этими элементами.

2.3.6. Определить любые интересные имена сценариев и параметры строк запросов, которые могут принадлежать сторонним компонентам кода. Ищите их в Google, используя квалификатор inurl: для поиска любых других приложений, использующих те же сценарии и параметры, которые, следовательно, могут использовать те же сторонние компоненты. Выполните неинвазивный обзор этих сайтов, потому что это может раскрыть дополнительный контент и функциональность, которые явно не связаны с приложением, на которое вы атакуете.

2.4 Map the Attack Surface

2.4.1 Постарайтесь выяснить вероятную внутреннюю структуру и функциональность серверного приложения и механизмы, которые оно использует за кулисами, чтобы обеспечить поведение, которое видно с точки зрения клиента. Например, функция получения заказов клиентов, вероятно, будет взаимодействовать с базой данных.

2.4.2 Для каждого элемента функциональности определите виды распространенных уязвимостей, которые часто связаны с ним. Например, функции загрузки файлов могут быть уязвимы для path traversal, межпользовательские сообщения могут быть уязвимы для XSS, а функции «Свяжитесь с нами» могут быть уязвимы для инъекции SMTP. См. Главу 4 для примеров уязвимостей, обычно связанных с конкретными функциями и технологиями.

2.4.3 Сформулируйте план атаки, расставив приоритеты в наиболее интересной функциональности и наиболее серьезных потенциальных уязвимостях, связанных с ней. Используйте свой план, чтобы определить количество времени и усилий, которые вы посвящаете каждой из оставшихся областей этой методологии.

3 Test Client-Side Controls

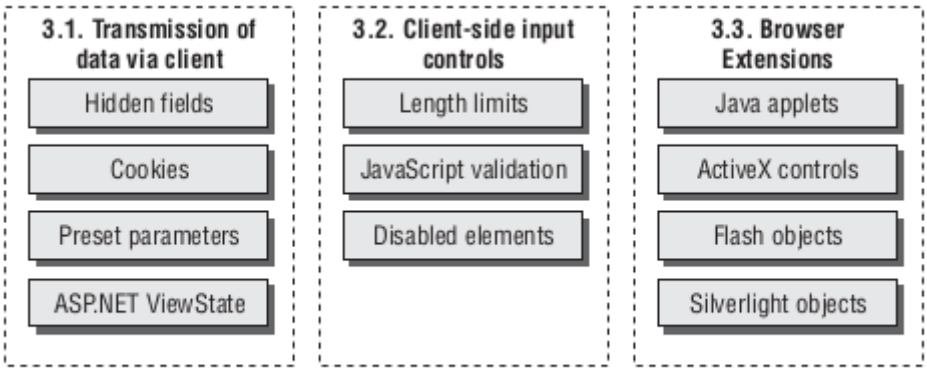

Figure 21-4: Testing client-side controls

3.1 Test Transmission of Data Via the Client

3.1.1 Найдите все экземпляры в приложении, где скрытые поля формы, файлы cookie и параметры URL, по-видимому, используются для передачи данных через клиента.

3.1.2 Попытка определить цель, которую элемент играет в логике приложения, на основе контекста, в котором он появляется, и его имени и значения.

3.1.3 Изменить ценность элемента способами, которые имеют отношение к его роли в функциональности приложения. Определите, обрабатывает ли приложение произвольные значения, представленные в поле, и может ли этот факт использоваться для вмешательства в логику приложения или для подрыва любых мер безопасности.

3.1.4 Если приложение передает непрозрачные данные через клиента, вы можете атаковать их различными способами. Если элемент запутан, вы можете расшифровать алгоритм запутывания и, следовательно, представить произвольные данные в непрозрачном элементе. Даже если он надежно зашифрован, вы сможете воспроизвести
элемент в других контекстах, чтобы мешать логике приложения. См. Главу 5 для получения более подробной информации об этих и других атаках.

3.1.5 Если приложение использует ASP.NET ViewState, проверьте, может ли это быть подделано или содержит какую-либо конфиденциальную информацию. Обратите внимание, что ViewState может использоваться по-разному на разных страницах приложения.

3.1.5.1 Используйте анализатор ViewState в Burp Suite, чтобы определить, включена ли опция EnableViewStateMac, что означает, что содержимое ViewState не может быть изменено.

3.1.5.2 Просмотрите декодированный ViewState, чтобы определить любые конфиденциальные данные, которые он содержит.

3.1.5.3 Изменить одно из декодированных значений параметров и перекодировать и отправить ViewState . Если приложение принимает измененное значение, вы должны рассматривать ViewState как входной канал для ввода произвольных данных в обработку приложения. Выполните то же тестирование на данных, которые он содержит, как и для любых других параметров запроса.

3.2 Test Client-Side Controls Over User Input

3.2.1. Определить любые случаи, когда элементы управления на стороне клиента, такие как ограничения длины и проверки JavaScript, используются для проверки ввода пользователем перед его отправкой на сервер. Эти элементы управления можно легко обойти, поскольку вы можете отправлять произвольные запросы на сервер. Например:

<form action="order.asp" onsubmit="return Validate(this)">
<input maxlength="3" name="quantity">
...

3.2.2 Проверяйте каждое затронутое поле ввода по очереди, отправляя входные данные, которые обычно блокируются элементами управления на стороне клиента, чтобы проверить, реплицируются ли они на сервере.

3.2.3 Возможность обхода проверки на стороне клиента не обязательно представляет какую-либо уязвимость. Тем не менее, вы должны внимательно изучить, что проверка выполняется. Подтвердите, полагается ли приложение на элементы управления на стороне клиента, чтобы защитить себя от искаженного ввода. Также подтвердите, существуют ли какие-либо эксплуатационные условия, которые могут быть инициированы
таким входом.

3.2.4 Просмотрите каждую форму HTML, чтобы идентифицировать любые отключенные элементы, такие как серые кнопки отправки. Например:

<input disabled="true" name="product">

Если вы найдете их, отправьте их на сервер вместе с другими параметрами формы. Посмотрите, влияет ли параметр на обработку сервера, которую вы можете использовать при атаке. В качестве альтернативы используйте правило автоматического прокси-сервера для автоматического включения отключенных полей, таких как правила «HTML-модификации» Burp Proxy.

4 Test the Authentication Mechanism

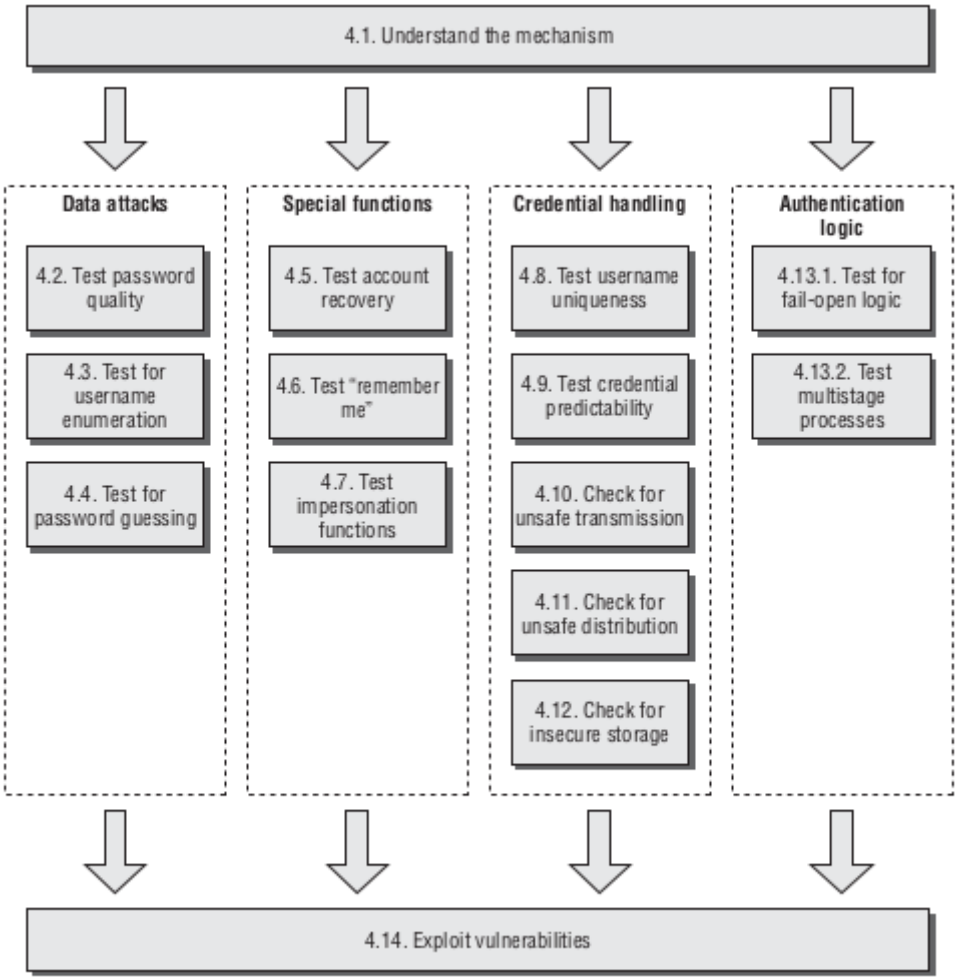

Figure 21-5: Testing the authentication mechanism

4.1 Understand the Mechanism

4.1.1 Установить используемые технологии аутентификации (например, формы, сертификаты или мультифактор).

4.1.2 Найдите все функции, связанные с аутентификацией (включая вход в систему, регистрацию, восстановление учетной записи и т. Д.).

4.1.3 Если приложение не реализует автоматизированный механизм саморегистрации, определите, существуют ли какие-либо другие средства для получения нескольких учетных записей пользователей.

4.2 Test Password Quality

4.2.1. Просмотрите приложение для любого описания минимальных правил качества, применяемых в паролях пользователей.

4.2.2 Попытка установить различные виды слабых паролей, используя любые функции саморегистрации или изменения пароля, чтобы установить фактически применяемые правила.
Попробуйте короткие пароли, только буквенные символы, однократные символы только словарные слова и текущее имя пользователя.

4.2.3 Тест на неполную проверку учетных данных. Установите надежный и сложный пароль (например, 12 символов со смешанными буквами, цифрами и типографскими символами). Попытайтесь войти в систему, используя различные варианты этого пароля, удалив последний символ, изменив регистр символа и удалив любые специальные символы. Если какая-либо из этих попыток входа в систему является успешной, продолжайте систематически экспериментировать, чтобы определить, какая проверка действительно выполняется.

4.2.4 Установив минимальные правила качества пароля и степень проверки пароля, определить диапазон значений, которые пароль-угадывание атаки нужно будет использовать, чтобы иметь хорошую вероятность успеха. Попытка найти любые встроенные учетные записи, которые, возможно, не подпадали под действие стандартных требований к сложности пароля.

4.3 Test for Username Enumeration

4.3.1. Определить каждое местоположение в различных функциях аутентификации где имя пользователя отправлено, в том числе через экранное поле ввода скрытое поле формы или cookie. Общие местоположения включают первичные вход в систему, саморегистрация, изменение пароля, выход из системы и восстановление учетной записи.

4.3.2 Для каждого местоположения отправьте два запроса, содержащие действительное и недопустимое имя пользователя. Просмотрите каждую деталь ответов сервера на каждую пару
запросы, включая код состояния HTTP, любые перенаправления, информацию отображаются на экране любые различия, скрытые в источнике HTML-страницы и время, необходимое для ответа сервера. Обратите внимание, что некоторые различия может быть тонким (например, одно и то же сообщение об ошибке может содержать незначительное
типографские различия). Вы можете использовать функцию истории вашего

перехват прокси для проверки всех трафика с на сервер и с сервера. WebScarab имеет функцию сравнения двух ответов, чтобы быстро выделить любые различия между ними.

4.3.3 Если вы заметили какие-либо различия между ответами, в которых представлено действительное и недопустимое имя пользователя, повторите тест с другой парой значения и подтвердите, что существует систематическая разница, которая может обеспечить основа для автоматического подсчета имени пользователя.

4.3.4 Проверьте наличие других источников утечки информации в приложении, которые могут позволить вам составить список действительных имен пользователей. Примерами являются функции регистрации, фактические списки зарегистрированных пользователей и прямое упоминание имен или адресов электронной почты в комментариях к исходному коду.

4.3.5 Найдите любую вспомогательную аутентификацию, которая принимает имя пользователя, и определите, можно ли ее использовать для подсчета имени пользователя. Обратите особое внимание на страницу регистрации, которая позволяет указывать имя пользователя.

4.4 Test Resilience to Password Guessing

4.4.1. Определить каждое место в приложении, в котором представлены учетные данные пользователя. Двумя основными экземплярами обычно являются основная функция входа в систему и функция изменения пароля. Последний обычно является действительной целью для атак с угадыванием пароля, только если может быть предоставлено произвольное имя пользователя.

4.4.2 В каждом месте, используя учетную запись, которой вы управляете, вручную отправляйте несколько запросов, содержащих действительное имя пользователя, но другие недействительные учетные данные. Мониторинг ответов приложения, чтобы выявить любые различия. После примерно 10 неудачных входов в систему, если приложение не вернуло сообщение о блокировке учетной записи, отправьте запрос, содержащий действительные учетные данные. Если этот запрос будет успешным, политика блокировки учетной записи, вероятно, не вступит в силу.

4.4.3 Если вы не контролируете какие-либо учетные записи, попытайтесь перечислить или угадать действительное имя пользователя и выполните несколько неверных запросов, используя это предположение, отслеживая любые сообщения об ошибках, связанных с блокировкой учетной записи. Конечно, вы должны знать, что этот тест может привести к приостановке или отключению учетной записи, принадлежащей другому пользователю.

4.5 Test Any Account Recovery Function

4.5.1. Определить, содержит ли приложение какое-либо средство для пользователей, чтобы восстановить контроль над своей учетной записью, если они забыли свои учетные данные. На это часто указывает ссылка «Забыл пароль» рядом с основной функцией входа в систему.

4.5.2. Установить, как работает функция восстановления учетной записи, выполнив полный процесс восстановления с помощью учетной записи, которую вы контролируете.

4.5.3 Если функция использует вызов, такой как секретный вопрос, определите, могут ли пользователи установить или выбрать свой собственный вызов во время регистрации. Если это так, используйте список перечисленных или общих имен пользователей, чтобы собрать список проблем, и просмотрите его для тех, которые кажутся легко угадываемыми.

4.5.4 Если функция использует подсказку пароля, выполните то же упражнение, чтобы собрать список подсказок пароля и идентифицировать любые, которые кажутся легко угадываемыми.

4.5.5 Выполните те же тесты для любых задач по восстановлению учетной записи, которые вы выполняли в основной функции входа, чтобы оценить уязвимость к атакам автоматического угадывания.

4.5.6 Если функция включает отправку электронного письма пользователю для завершения процесса восстановления, найдите любые недостатки, которые могут позволить вам взять под контроль учетные записи других пользователей. Определите, возможно ли это контролировать адрес, на который отправляется электронное письмо. Если сообщение содержит уникальный URL-адрес для восстановления, получите несколько сообщений с использованием адреса электронной почты, которым вы управляете, и попытайтесь определить любые шаблоны, которые могут позволить вам предсказать URL-адреса, выданные другим пользователям. Примените методологию, описанную на этапе 5.3, для определения любых предсказуемых последовательностей.

4.6 Test Any Remember Me Function

4.6.1 Если основная функция входа или поддерживающая логика содержит Remember Я функционирую, активирую это и проверяю его эффекты. Если эта функция позволяет пользователю входить в систему в последующих случаях без ввода каких-либо учетных данных, вам следует внимательно изучить ее на наличие каких-либо уязвимостей.

4.6.2. Внимательно проверяйте все постоянные файлы cookie, которые устанавливаются при активации функции «Запомнить меня». Ищите любые данные, которые идентифицируют пользователя явно или, по-видимому, содержат некоторые предсказуемые идентификаторы пользователя.

4.6.3 Даже там, где сохраненные данные, по-видимому, сильно закодированы или запутаны, внимательно изучите их и сравните результаты запоминания нескольких очень похожих имен пользователей и / или паролей, чтобы определить любые возможности для реинжиниринга исходных данных. Применяйте методологию, описанную на этапе 5.2, для выявления любых значимых данных.

4.6.4 В зависимости от ваших результатов, измените содержимое вашего файла cookie подходящими способами, пытаясь маскироваться под других пользователей приложения.

4.7 Test Any Impersonation Function

4.7.1 Если приложение содержит какие-либо явные функции, которые позволяют одному пользователю выдавать себя за другого, внимательно изучите это на наличие любых уязвимостей, которые могут позволить вам выдавать себя за произвольных пользователей без надлежащей авторизации.

4.7.2 Ищите любые предоставленные пользователем данные, которые используются для определения цели олицетворения. Попытайтесь манипулировать этим, чтобы выдать себя за других пользователей, особенно административных, что может позволить вам увеличить привилегии.

4.7.3 Если вы выполняете какие-либо автоматические атаки для угадывания паролей на другие учетные записи пользователей, ищите учетные записи, которые имеют более одного действительного пароля, или несколько учетных записей, которые, по-видимому, имеют один и тот же пароль. Это может указывать на наличие бэкдора, который администраторы могут использовать для доступа к приложению как любой пользователь.

4.8 Test Username Uniqueness

4.8.1 Если приложение имеет функцию саморегистрации, которая позволяет вам указать желаемое имя пользователя, попробуйте дважды зарегистрировать одно и то же имя пользователя разные пароли.

4.8.2 Если приложение блокирует вторую попытку регистрации, вы можете использовать это поведение для перечисления зарегистрированных имен пользователей.

4.8.3 Если приложение регистрирует обе учетные записи, дополнительно проверьте его поведение при столкновении имени пользователя и пароля. Попытка изменить пароль одной из учетных записей, чтобы он соответствовал другому. Также попробуйте зарегистрировать две учетные записи с одинаковыми именами пользователей и паролями.

4.8.4 Если приложение предупреждает вас или генерирует ошибку при столкновении имени пользователя и пароля, вы, вероятно, можете использовать это для выполнения автоматической атаки на угадывание, чтобы обнаружить пароль другого пользователя. Направьте на перечисленное или предполагаемое имя пользователя и попытайтесь создать учетные записи, которые имеют это имя пользователя и разные пароли. Когда приложение отклоняет определенный пароль, вы, вероятно, нашли существующий пароль для целевой учетной записи.

4.8.5 Если приложение, по-видимому, допускает столкновение имени пользователя и пароля без ошибки, войдите в систему, используя учетные данные столкновения. Определите, что происходит и можно ли использовать поведение приложения
получить несанкционированный доступ к учетным записям других пользователей.

4.9 Test Predictability of Autogenerated Credentials

4.9.1 Если приложение автоматически генерирует имена пользователей или пароли, попробуйте быстро получить несколько значений и определить любые обнаруживаемые последовательности или шаблоны.

4.9.2 Если имена пользователей создаются предсказуемым образом, экстраполируйте назад, чтобы получить список возможных действительных имен пользователей. Вы можете использовать это в качестве основы для автоматического угадывания паролей и других атак.

4.9.3 Если пароли создаются предсказуемым образом, экстраполируйте шаблон для получения списка возможных паролей, выданных другим пользователям приложения. Это можно комбинировать с любыми списками имен пользователей, которые вы получаете для выполнения атаки с угадыванием пароля.

4.10 Check for Unsafe Transmission of Credentials

4.10.1. Проходить через все функции, связанные с аутентификацией, которые включают передачу учетных данных, включая основной вход в систему, регистрацию учетной записи изменение пароля и любой страницы, которая позволяет просматривать или обновлять информацию профиля пользователя. Контролируйте весь трафик, проходящий в обоих направлениях между клиентом и сервером, используя ваш перехватчик.

4.10.2. Определить каждый случай, когда учетные данные передаются в любом направлении. Вы можете установить правила перехвата в своем прокси-сервере, чтобы пометить сообщения, содержащие определенные строки.

4.10.3 Если учетные данные когда-либо передаются в строке запроса URL, они потенциально уязвимы для раскрытия в истории браузера, на экране, в журналах сервера и в заголовке Referer при отслеживании сторонних ссылок.

4.10.4 Если учетные данные когда-либо хранятся в cookie-файле, они потенциально уязвимы для раскрытия посредством атак XSS или локальных атак конфиденциальности.

4.10.5 Если учетные данные когда-либо передаются с сервера клиенту, они могут быть скомпрометированы с помощью любых уязвимостей в управлении сеансом или контроле доступа, или в атаке XSS.

4.10.6 Если учетные данные когда-либо передаются по незашифрованному соединению, они уязвимы для перехвата подслушивающим устройством.

4.10.7 Если учетные данные отправляются с использованием HTTPS, но сама форма входа загружается с использованием HTTP, приложение уязвимо для атаки «человек посередине», которая может использоваться для захвата учетных данных.

4.11 Check for Unsafe Distribution of Credentials

4.11.1 Если учетные записи создаются по какому-либо out of band каналу или приложение имеет функцию саморегистрации, которая сама не определяет все начальные учетные данные пользователя, установите средства, с помощью которых учетные данные распространяются среди новых пользователей. Общие методы включают отправку сообщения на адрес электронной почты или почтовый адрес.

4.11.2 Если приложение генерирует URL-адреса активации учетной записи, которые распространяются out of band, попробуйте зарегистрировать несколько новых учетных записей в тесной последовательности и определить любую последовательность в URL-адресах, которые вы получаете. Если шаблон может определите, попытайтесь предсказать URL-адреса, отправленные недавним и будущим пользователям, и попытайтесь использовать эти URL-адреса, чтобы взять на себя ответственность за их учетные записи.

4.11.3 Попробуйте повторно использовать один URL-адрес активации несколько раз и посмотрите, позволяет ли это приложение. Если этого не произойдет, попробуйте заблокировать целевую учетную запись перед повторным использованием URL-адреса и посмотреть, работает ли URL-адрес. Определите, позволяет ли это установить новый пароль в активной учетной записи.

4.12 Test for Insecure Storage

4.12.1 Если вы получаете доступ к хэшированным паролям, проверьте учетные записи, которые имеют одинаковое значение хэшированного пароля. Попробуйте войти с общими паролями для наиболее распространенного значения хеширования.

4.12.2 Используйте автономную радужную таблицу для рассматриваемого алгоритма хеширования, чтобы восстановить значение четкого текста.

4.13 Test for Logic Flaws
4.13.1 Test for Fail-Open Conditions

4.13.1.1 Для каждой функции, в которой приложение проверяет учетные данные пользователя, включая функции входа и изменения пароля, проходите процесс обычным способом, используя учетную запись, которую вы контролируете. Обратите внимание на каждый параметр запроса, представленный в приложение.

4.13.1.2 Повторяйте процесс много раз, изменяя каждый параметр по очереди различными неожиданными способами, предназначенными для вмешательства в логику приложения. Для каждого параметра включите следующие изменения:

- Отправьте пустую строку в качестве значения.
- Удалить пару имя / значение.
- Подать очень длинные и очень короткие значения.
- Отправляйте строки вместо цифр и наоборот.
- Отправьте один и тот же именованный параметр несколько раз, с одинаковыми и разными значениями.

4.13.1.3 Внимательно изучите ответы заявки на предыдущие запросы. Если возникают какие-либо неожиданные расхождения с базовым случаем, отправьте это наблюдение обратно в кадрирование дальнейших тестовых случаев. Если одна модификация вызывает изменение в поведении, попробуйте объединить это с другими изменениями, чтобы расширить логику приложения до предела.

4.13.2 Test Any Multistage Mechanisms

4.13.2.1 Если какая-либо связанная с аутентификацией функция включает отправку учетных данных в серии различных запросов, определите очевидную цель каждого отдельного этапа и запишите параметры, представленные на каждом этапе.

4.13.2.2 Повторите процесс много раз, изменяя последовательность запросов способами, разработанными для вмешательства в логику приложения, в том числе
следующие тесты:

- Пройдите все этапы, но в последовательности, отличной от намеченной.
- Перейдите непосредственно к каждому этапу по очереди и продолжайте нормальную последовательность оттуда.
- Пройдите через нормальную последовательность несколько раз, пропуская каждый этап по очереди и продолжая нормальную последовательность со следующего этапа.
- На основе ваших наблюдений и очевидной цели каждого этапа механизма, попробуйте придумать дальнейшие способы изменения последовательности и получить доступ к различным этапам, которые разработчики, возможно, не ожидали.

4.13.2.3 Определите, передается ли какая-либо отдельная информация (например, имя пользователя) на более чем одном этапе, либо потому, что она перехвачена более одного раза у пользователя, либо потому, что она передается через клиент в поле скрытой формы, cookie, или предустановленный параметр строки запроса. Если это так, попробуйте представить разные значения на разных этапах (как действительных, так и недействительных) и наблюдать за эффектом. Постарайтесь определить, является ли представленный элемент иногда излишним, или проверен на одном этапе, а затем доверен впоследствии, или проверен на разных этапах с помощью разных проверок.
Попробуйте использовать поведение приложения, чтобы получить несанкционированный доступ или снизить эффективность средств контроля, налагаемых механизмом.

4.13.2.4 Ищите любые данные, которые передаются через клиент, который не был получен от пользователя в любой момент. Если скрытые параметры используются для отслеживания

состояния процесса на последовательных этапах, может быть возможно помешать логике приложения, изменив эти параметры искусными способами.

4.13.2.5 Если какая-либо часть процесса связана с представлением приложения случайным образом изменяющейся задачи, проверьте на наличие двух распространенных дефектов:

- Если параметр, указывающий вызов, представлен вместе с ответом пользователя, определите, сможете ли вы эффективно выбрать свой собственный вызов, изменив это значение.
- Попробуйте перейти к изменяющемуся вызову несколько раз с одним и тем же именем пользователя и определите, представлен ли другой вызов. Если это так, вы можете эффективно выбрать свой собственный вызов, неоднократно переходя к этому этапу, пока не будет представлен желаемый вызов.

4.14 Exploit Any Vulnerabilities to Gain Unauthorized Access

4.14.1. Просмотрите все уязвимости, которые вы идентифицировали в различных функциях аутентификации, и определите все, что вы можете использовать для достижения своих целей при атаке приложения. Обычно это включает попытку аутентификации как другого пользователя - если это возможно, пользователя с административными привилегиями.

4.14.2 Перед установкой любого вида автоматической атаки запишите все защитные средства блокировки учетной записи, которые вы определили. Например, при выполнении подсчета имени пользователя по функции входа в систему, отправляйте общий пароль с каждым запросом, а не совершенно произвольное значение, чтобы не тратить неудачную попытку входа в систему на каждое обнаруженное имя пользователя.
Точно так же выполняйте любые атаки с угадыванием пароля на основе широты, а не глубины. Запустите свой список слов с наиболее распространенными слабыми паролями и просмотрите этот список, примеряя каждый элемент с каждым перечисленным именем пользователя.

4.14.3. При построении списков слов для использования в любом пароле учитывайте правила качества пароля и полноту проверки пароля
угадывание атаки, чтобы избежать невозможных или лишних тестовых случаев.

4.14.4 Используйте методы, описанные в главе 14, чтобы автоматизировать как можно больше работы и максимизировать скорость и эффективность ваших атак.

5 Test the Session Management Mechanism

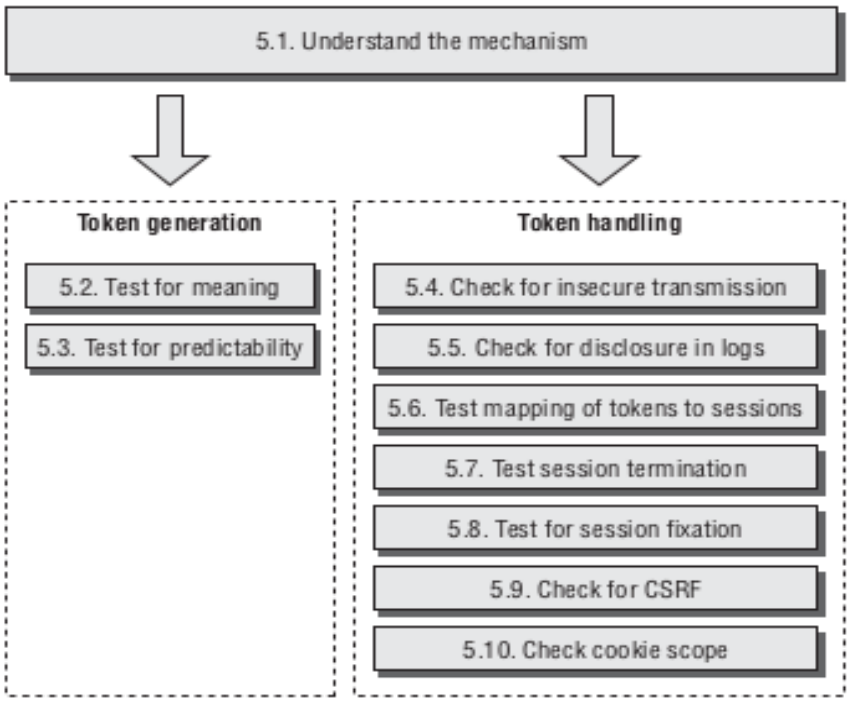

Figure 21-6: Testing the session management mechanism

5.1 Understand the Mechanism

5.1.1 Анализировать механизм, используемый для управления сеансами и состояний. Установите, использует ли приложение токены сеанса или какой-либо другой метод обработки серии запросов, полученных от каждого пользователя. Обратите внимание, что некоторые технологии аутентификации (такие как аутентификация HTTP) могут не требовать полного механизма сеанса для повторной идентификации пользователей после аутентификации.

Кроме того, некоторые приложения используют механизм состояния без сеанса, в котором вся информация о состоянии передается через клиента, обычно в зашифрованном или запутанном виде.

5.1.2 Если приложение использует токены сеанса, точно подтвердите, какие фрагменты данных фактически используются для идентификации пользователей. Элементы, которые могут использоваться для передачи токенов, включают файлы cookie HTTP, параметры строки запроса и поля скрытой формы. Несколько различных частей данных могут использоваться совместно для повторной идентификации пользователя, а разные элементы могут использоваться разными внутренними компонентами. Часто элементы, которые выглядят как токены сеанса, на самом деле не могут использоваться приложением как таковые, например cookie-файл по умолчанию, сгенерированный веб-сервером.

5.1.3 Чтобы проверить, какие элементы фактически используются в качестве токенов сеанса, найдите страницу или функцию, которая, безусловно, зависит от сеанса (например, пользовательская спецификация с My Details page). Затем сделайте несколько запросов на это, систематически удаляя каждый элемент, который, как вы подозреваете, используется в

качестве токена сеанса. Если удаление элемента останавливает возврат страницы, зависящей от сеанса, это может подтвердить, что элемент является токеном сеанса. Burp Repeater - полезный инструмент для выполнения этих тестов.

5.1.4 Установив, какие элементы данных фактически используются для повторной идентификации пользователей, для каждого токена подтвердите, проверяется ли он полностью, или же некоторые подкомпоненты токена игнорируются.
Измените значение токена на 1 байт за раз и проверьте, все ли измененное значение все еще принято. Если вы обнаружите, что определенные части токена фактически не используются для поддержания состояния сеанса, вы можете исключить их
из дальнейшего анализа.

5.2 Test Tokens for Meaning

5.2.1 Войдите в систему как несколько разных пользователей в разное время и запишите токены, полученные с сервера. Если доступна саморегистрация, и вы можете выбрать свое имя пользователя, войдите в систему с помощью ряда похожих имен пользователей, которые имеют небольшие вариации, таких как A, AA, AAA, AAAA, AAAB, AAAC, AABA и так далее. Если другие специфичные для пользователя данные передаются в логине или хранятся в профилях пользователей (например, адрес электронной почты), выполните аналогичное упражнение, чтобы систематически изменять эти данные и захватывать полученные токены.

5.2.2 Анализируйте полученные токены для любых корреляций, которые, по-видимому, связаны с именем пользователя и другими контролируемыми пользователем данными.

5.2.3 Анализировать токены для любой обнаруживаемой кодировки или запутывания. Ищите корреляцию между длиной имени пользователя и длиной токена, что сильно указывает на то, что используется какая-то запутывание или кодирование. Если имя пользователя содержит последовательность одного и того же символа, найдите соответствующую последовательность символов в токене, которая может указывать на использование запутывания XOR. Ищите последовательности в токене, которые содержат только шестнадцатеричные символы, которые могут указывать шестнадцатеричное кодирование строки ASCII или другой информации. Ищите последовательности, оканчивающиеся на знак равенства и / или содержащие только другие
действительные символы Base64: от a до z, от A до Z, от 0 до 9, + и /.

5.2.4 Если вы можете идентифицировать какие-либо значимые данные в своем образце токенов сеанса, подумайте, достаточно ли этого для создания атаки, которая пытается угадать токены, недавно выпущенные другим пользователям приложения. Найти страница приложения, которая зависит от сеанса и использует методы, описанные в главе 14, для автоматизации задачи генерации и тестирования
возможные токены.

5.3 Test Tokens for Predictability

5.3.1 Быстро генерируйте и захватывайте большое количество токенов сеанса, используя запрос, который заставляет сервер возвращать новый токен (например, успешный запрос входа в систему).

5.3.2 Попытка идентифицировать любые шаблоны в вашем образце токенов. Во всех случаях вам следует использовать Burp Sequencer, как описано в главе 7, для проведения подробных статистических тестов свойств случайности токенов приложения. В зависимости от результатов, также может быть полезно выполнить следующий ручной анализ:

- Примените свое понимание того, какие токены и последовательности приложение фактически использует для повторной идентификации пользователей. Игнорируйте любые данные, которые не используются таким образом, даже если они варьируются между образцами.

- Если неясно, какой тип данных содержится в токене или в каком-либо отдельном его компоненте, попробуйте применить различные декодирования (например, Base64), чтобы увидеть, появляются ли какие-либо более значимые данные. Может потребоваться применить несколько декодирований по последовательности.

- Попробуйте идентифицировать любые шаблоны в последовательностях значений, содержащихся в каждом декодированном токене или компоненте. Рассчитайте различия между последовательными значениями. Даже если они кажутся хаотичными, может существовать фиксированный набор наблюдаемых различий, который значительно сужает масштабы любой атаки грубой силы

- Получите подобный образец токенов после ожидания в течение нескольких минут и повторите тот же анализ. Постарайтесь определить, зависит ли какое-либо содержимое токенов от времени.

5.3.3 Если вы идентифицируете какие-либо шаблоны, захватите второй образец токенов, используя другой IP-адрес и другое имя пользователя. Это поможет вам определить, обнаружен ли тот же шаблон и могут ли токены, полученные в первом упражнении, быть экстраполированы для угадывания токенов, полученных во втором.

5.3.4 Если вы можете определить какие-либо используемые последовательности или зависимости времени, подумайте, достаточно ли этого для создания атаки, которая пытается угадать токены, недавно выпущенные другим пользователям приложения. Используйте методы, описанные в главе 14, чтобы автоматизировать задачу генерации и тестирования возможных токенов. За исключением простейших последовательностей, вполне вероятно, что ваша атака должна включать какой-то настраиваемый скрипт.

5.3.5 Если идентификатор сеанса отображается на заказ, используйте источник полезной нагрузки «bit flopper» в Burp Intruder, чтобы последовательно изменять каждый бит в токене сеанса по очереди. Привязать строку в ответе, которая указывает, не привело ли изменение токена к недопустимому сеансу, и принадлежит ли сеанс другому пользователю.

5.4 Check for Insecure Transmission of Tokens

5.4.1 Просмотрите приложение как обычно, начиная с не аутентифицированного содержимого по начальному URL-адресу, проходя через процесс входа в систему, а затем просматривая все функции приложения. Запишите каждый случай, когда выдается новый токен сеанса, и какие части ваших сообщений используют HTTP, а какие используют HTTPS Вы можете использовать функцию регистрации вашего перехватывающего прокси-сервера для записи этой информации.

5.4.2 Если файлы cookie HTTP используются в качестве механизма передачи для токенов сеанса, проверьте, установлен ли защищенный флаг, не позволяя им когда-либо передаваться по соединениям HTTP.

5.4.3. Определить, передаются ли когда-либо при обычном использовании приложения токены сеанса через соединение HTTP. Если это так, они уязвимы для перехвата.

5.4.4 В случаях, когда приложение использует HTTP для неаутентифицированных областей и переключается на HTTPS для входа и / или аутентифицированных областей приложения, проверить, выпущен ли новый токен для части HTTPS сообщений, или же токен, выпущенный на этапе HTTP, остается активным, когда приложение переключается на HTTPS.
Если токен, выпущенный на этапе HTTP, остается активным, токен уязвим для перехвата.

5.4.5 Если область HTTPS приложения содержит какие-либо ссылки на URL-адреса HTTP, следуйте им и проверьте, представлен ли токен сеанса. Если это так
определить, продолжает ли он быть действительным или немедленно прекращается

5.5 Check for Disclosure of Tokens in Logs

5.5.1 Если в ваших упражнениях по картированию приложений были определены какие-либо функции ведения журнала, мониторинга или диагностики, внимательно изучите эти функции, чтобы определить, раскрываются ли в них какие-либо токены сеанса. Подтвердите, кто обычно имеет право доступа к этим функциям. Если они предназначены только для администраторов, определите, существуют ли какие-либо другие уязвимости, которые могли бы позволить пользователю с более низкими привилегиями получить к ним доступ.

5.5.2. Определить случаи, когда токены сеанса передаются в URL-адресе. Может случиться так, что токены обычно передаются более безопасным способом, но разработчики использовали URL-адрес в определенных случаях для решения конкретной проблемы. Если это так, они могут передаваться в заголовке Referer, когда пользователи переходят по любым сторонним ссылкам. Проверьте любую функциональность, которая позволяет вводить произвольные ссылки за пределами сайта на страницы, просматриваемые другими пользователями.

5.6 Check Mapping of Tokens to Sessions

5.6.1. Войдите в приложение дважды, используя одну и ту же учетную запись пользователя, либо из разных процессов браузера, либо с разных компьютеров. Определите, остаются ли обе сессии активными одновременно. Если они это делают, приложение поддерживает параллельные сеансы, позволяя злоумышленнику, который скомпрометировал учетные данные другого пользователя, использовать их без риска обнаружения.

5.6.2 Войдите и выходите из системы несколько раз, используя одну и ту же учетную запись пользователя, либо из разных процессов браузера, либо с разных компьютеров. Определите, выдается ли новый токен сеанса каждый раз, или же один и тот же токен выдается каждый раз, когда один и тот же аккаунт входит в систему. Если последнее происходит, приложение на самом деле не использует правильные токены сеанса, но использует уникальные постоянные строки для повторной идентификации каждого пользователя. В этой ситуации нет способа защитить от одновременных входов в систему или должным образом обеспечить тайм-аут сеанса.

5.6.3 Если токены содержат какую-либо структуру и значение, попытайтесь отделить компоненты, которые могут идентифицировать пользователя, от компонентов, которые кажутся непостижимыми. Попробуйте изменить любые связанные с пользователем компоненты токена, чтобы они ссылались на других известных пользователей приложения. Проверьте, принимает ли приложение полученный токен и позволяет ли оно маскироваться под этого пользователя. См. Главу 7 для примеров такой тонкой уязвимости.

5.7 Test Session Termination

5.7.1 При тестировании на недостатки тайм-аута сеанса и выхода из системы фокусируйтесь исключительно на обработке сеансов и токенов сервером, а не на любых событиях, происходящих на клиенте. С точки зрения завершения сеанса, ничего особенного не зависит от того, что происходит с токеном в браузере клиента.

5.7.2 Проверьте, не реализован ли срок действия сеанса на сервере :

- Войдите в приложение, чтобы получить действительный токен сеанса.

- Подождите некоторое время без использования этого токена, а затем отправьте запрос на защищенную страницу (например, Мои данные) с помощью токена.
- Если страница отображается нормально, токен все еще активен.
- Используйте пробный и ошибочный, чтобы определить, как долго истекает срок действия сеанса, или же токен можно использовать через несколько дней после предыдущего запроса, который его использовал. Burp Intruder может быть настроен на увеличение временного интервала между последовательными запросами для автоматизации этой задачи.

5.7.3 Проверьте, существует ли функция выхода из системы. Если это произойдет, проверьте, действительно ли это лишает законной силы сеанс пользователя на сервере. После выхода из системы попытайтесь повторно использовать старый токен и определите, является ли он все еще действительным, запросив защищенную страницу с помощью токена. Если сеанс все

еще активен, пользователи остаются уязвимыми для некоторых атак захвата сеанса даже после того, как они «вышли из системы».«Вы можете использовать Burp Repeater, чтобы продолжать отправлять определенный запрос из истории прокси-сервера, чтобы увидеть, реагирует ли приложение по-разному после выхода из системы.

5.8 Check for Session Fixation

5.8.1 Если приложение выдает токены сеанса не аутентифицированным пользователям, получите токен и выполните вход в систему. Если приложение не выдает новый токен после успешного входа в систему, оно уязвимо для фиксации сеанса.

5.8.2 Даже если приложение не выдает токены сеанса не аутентифицированным пользователям, получите токен, войдя в систему, а затем вернитесь на страницу входа. Если приложение готово вернуть эту страницу, даже если вы уже прошли аутентификацию, отправьте другой логин как другой пользователь, используя тот же
жетон. Если приложение не выдает новый токен после второго входа в систему, оно уязвимо для фиксации сеанса.

5.8.3. Определите формат токенов сеанса, который использует приложение. Измените свой токен на изобретенное значение, которое правильно сформировано, и попытайтесь войти в систему. Если приложение позволяет вам создать аутентифицированный сеанс с использованием изобретенного токена, оно уязвимо для фиксации сеанса.

5.8.4 Если приложение не поддерживает логин, но обрабатывает конфиденциальную информацию о пользователе (например, личные данные и данные платежа) и позволяет отображать ее после отправки (например, на странице «Проверка моего заказа»), выполните предыдущие три теста в отношение к страницам, отображающим конфиденциальные данные. Если набор токенов во время анонимного использования приложения может позже использоваться для получения конфиденциальной пользовательской информации, приложение уязвимо для фиксации сеанса.

5.9 Check for CSRF

5.9.1 Если приложение использует исключительно файлы cookie HTTP в качестве метода передачи токенов сеанса, оно может быть уязвимо для подделки межсайтовых запросов атаки.

5.9.2. Изучите ключевые функции приложения и определите конкретные запросы, которые используются для выполнения чувствительных действий. Если злоумышленник может заранее определить параметры для любого из этих запросов (то есть они не содержат токенов сеанса, непредсказуемых данных или других секретов), приложение почти наверняка уязвимо.

5.9.3 Создание HTML-страницы, которая выдаст нужный запрос без какого-либо взаимодействия с пользователем. Для запросов GET, Вы можете поместить тег <img> с параметром src, установленным на уязвимый URL. Для запросов POST, Вы можете создать

форму, которая содержит скрытые поля для всех соответствующих параметров, необходимых для атаки, и цель которой установлена на уязвимый URL. Вы можете использовать JavaScript для автоматической отправки формы, как только страница загрузится. При входе в приложение используйте тот же браузер для загрузки HTML-страницы. Убедитесь, что желаемое действие выполняется в приложении.

5.9.4 Если приложение использует дополнительные токены в запросах в попытке предотвратить атаки CSRF, проверьте их надежность так же, как и для токенов сеанса. Также проверьте, уязвима ли заявка к атакам возмещения пользовательского интерфейса, чтобы победить защиту против CSRF (см. Главу 13 для более подробной информации).

5.10 Check Cookie Scope

5.10.1 Если приложение использует файлы cookie HTTP для передачи токенов сеанса (или любых других конфиденциальных данных), просмотрите соответствующие заголовки Set-Cookie и проверьте наличие каких-либо атрибутов домена или пути, используемых для контроля объема файлов cookie.

5.10.2 Если приложение явно либерализует область действия своих файлов cookie для родительского домена или родительского каталога, оно может оставаться уязвимым для атак через другие веб-приложения, которые размещены в родительском домене или каталоге.

5.10.3 Если приложение устанавливает область действия своего домена cookie на свое собственное доменное имя (или не указывает атрибут домена), оно все равно может подвергаться атакам через любые приложения, размещенные в поддоменах. Это является следствием того, как работает поиск файлов cookie. Этого нельзя избежать, кроме как не размещать какие-либо другие приложения в поддомене чувствительного к безопасности приложения.

5.10.4. Определить любую зависимость от сегрегации по пути, например /site/main и /site/demo, которые могут быть подорваны в случае атаки сценариев между сайтами.

6 Test Access Controls

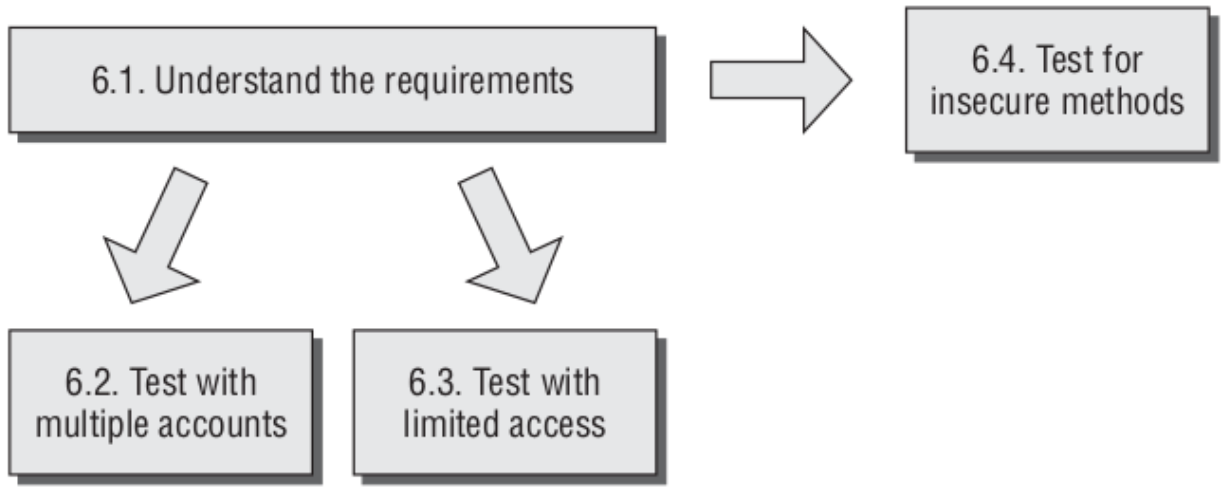

Figure 21-7: Testing access controls

6.1 Understand the Access Control Requirements

6.1.1 На основе основных функций, реализованных в приложении, понять широкие требования к контролю доступа с точки зрения вертикальной сегрегации (разные уровни пользователей имеют доступ к различным типам функций) и горизонтальной сегрегации (пользователи с одной и той же привилегией) уровень имеют доступ к различным подмножествам данных). Часто присутствуют оба типа сегрегации. Например, обычные пользователи могут иметь доступ к своим собственным данным, а администраторы могут получить доступ к данным каждого.

6.1.2 Просмотрите результаты картирования приложений, чтобы определить области функциональности и типы ресурсов данных, которые представляют собой наиболее плодотворные цели для атак по эскалации привилегий.

6.1.3 Чтобы выполнить наиболее эффективное тестирование на уязвимости контроля доступа, в идеале вы должны получить несколько различных учетных записей с различными вертикальными и горизонтальными привилегиями. Если возможна саморегистрация, вы, вероятно, можете получить ее непосредственно из приложения. Чтобы получить первое, вам, вероятно, понадобится сотрудничество владельца приложения (или вам нужно использовать некоторую уязвимость, чтобы получить доступ к высокопривилегированной учетной записи). Наличие различных типов учетных записей повлияет на типы тестирования, которые вы можете выполнить, как описано ниже.

6.2 Test with Multiple Accounts

6.2.1 Если приложение обеспечивает вертикальную сегрегацию привилегий, сначала используйте а мощный аккаунт, чтобы найти все функции, к которым он может получить доступ. Затем
используйте менее привилегированную учетную запись и попытайтесь получить доступ к каждому элементу этого функциональность.

6.2.1.1 Используя Burp, просмотрите все содержимое приложения в одном пользовательском контексте.

6.2.1.2 Просмотрите содержимое карты сайта Burp, чтобы убедиться, что вы определили все функции, которые вы хотите протестировать. Затем выйдите из приложения и войдите в систему, используя другой пользовательский контекст.
Используйте контекстное меню, чтобы выбрать функцию «сравнить карты сайтов», чтобы определить, какие запросы с высокой привилегированностью могут быть доступны для пользователя с более низкой привилегией. См. Главу 8 для более подробной информации об этой технике.

6.2.2 Если приложение обеспечивает горизонтальную сегрегацию привилегий, выполните эквивалентный тест, используя две разные учетные записи с одной и той же привилегией уровень, пытаясь использовать одну учетную запись для доступа к данным, принадлежащим другой учетной записи. Обычно это включает замену идентификатора (такого как идентификатор документа) в запросе, чтобы указать ресурс, принадлежащий другому пользователю.

6.2.3 Выполнить ручную проверку логики управления доступом к ключам.

6.2.3.1 Для каждой привилегии пользователя просмотрите ресурсы, доступные пользователю. Попытайтесь получить доступ к этим ресурсам из несанкционированной учетной записи пользователя, повторив запрос с помощью токена сеанса неавторизованного пользователя.

6.2.4 Когда вы выполняете любой вид теста контроля доступа, Обязательно протестируйте каждый шаг многоступенчатых функций по отдельности, чтобы подтвердить, были ли элементы управления доступом должным образом реализованы на каждом этапе, или предполагает ли приложение, что пользователи, которые получают доступ к более позднему этапу, должны были пройти проверки безопасности, реализованные на более ранних этапах. Например, если административная страница, содержащая форму, должным образом защищена, проверьте, подвергается ли фактическая подача формы надлежащему контролю доступа.

6.3 Test with Limited Access

6.3.1 Если у вас нет предварительного доступа к учетным записям на разных уровнях привилегий или к нескольким учетным записям с доступом к различным данным, тестирование на сломанные элементы управления доступом не так просто.
Многие распространенные уязвимости будет гораздо сложнее найти, потому что вы не знаете имен URL-адресов, идентификаторов и параметров, которые необходимы для использования слабых сторон.

6.3.2 В своих упражнениях по картированию приложений, в которых используется низкопривилегированная учетная запись, вы, возможно, определили URL-адреса для

привилегированных функций, таких как административные интерфейсы. Если они не защищены должным образом, вы, вероятно, уже знаете об этом.

6.3.3. Разберите все присутствующие скомпилированные клиенты и извлеките любые ссылки на функции сервера.

6.3.4 Большинство данных, на которые распространяются элементы управления горизонтальным доступом, доступны с использованием идентификатора, такого как номер учетной записи или ссылка на заказ. Тестировать Независимо от того, эффективны ли средства управления доступом с использованием только одной учетной записи, вы должны попытаться угадать или обнаружить идентификаторы, связанные с другими пользователями данные. Если возможно, создайте серию идентификаторов в быстрой последовательности (например, создав несколько новых заказов). Попытайтесь определить любые шаблоны, которые могут позволить вам предсказать идентификаторы, выданные другим пользователям. Если нет способа генерировать новые идентификаторы, вы, вероятно, ограничены анализом тех, которые у вас уже есть, и догадками на этой основе.

6.3.5 Если вы нашли способ предсказать идентификаторы, выданные другим пользователям, используйте методы, описанные в главе 14, для создания автоматической атаки для сбора интересных данных, принадлежащих другим пользователям. Используйте функцию Extract Grep в Burp Intruder, чтобы получить соответствующую информацию из ответов приложения.

6.4 Test for Insecure Access Control Methods

6.4.1 Некоторые приложения реализуют средства контроля доступа на основе параметров запроса небезопасным образом. Ищите такие параметры, как edit=false или access=read в любых ключевых запросах, и изменяйте их в соответствии с их очевидной ролью, чтобы попытаться помешать логике управления доступом приложения.

6.4.2 Некоторые приложения основывают решения по управлению доступом на заголовке HTTP Referer. Например, приложение может надлежащим образом контролировать доступ к /admin.jsp и принимать любой запрос, показывающий это как его реферер . Чтобы проверить это поведение, попытайтесь выполнить некоторые привилегированные действия, на которые вы авторизованы, и отправьте отсутствующий или измененный заголовок Referer. Если это изменение заставляет приложение блокировать ваш запрос, оно может использовать заголовок Referer небезопасным способом. Попробуйте выполнить то же действие, что и неавторизованный пользователь, но предоставьте исходный заголовок Referer и посмотрите, удастся ли это действие.

6.4.3 Если HEAD является разрешенным методом на площадке, проверьте наличие небезопасного контейнера
управляемый контроль доступа к URL. Сделайте запрос, используя метод HEAD, чтобы определить, позволяет ли приложение это.

7 Test for Input-Based Vulnerabilities

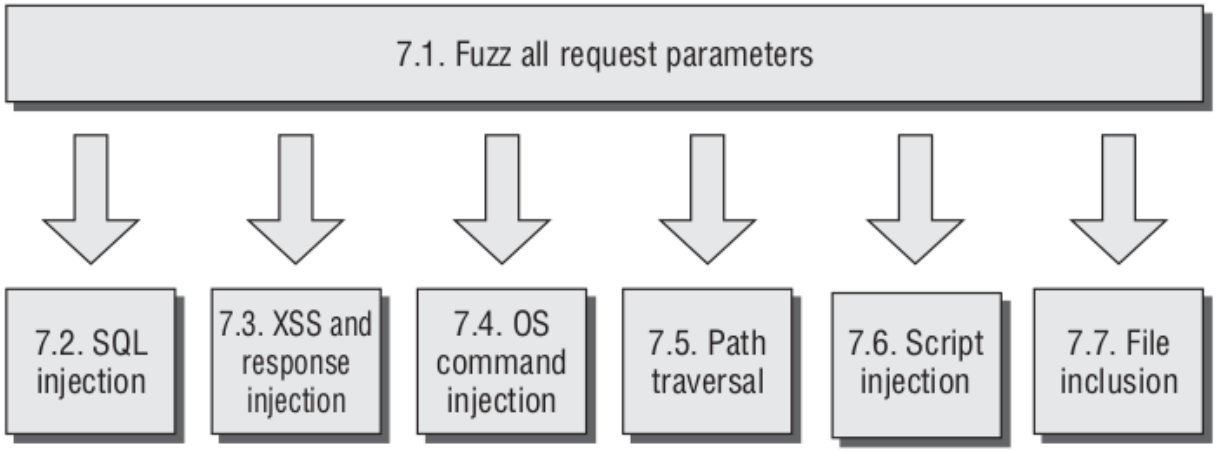

Figure 21-8: Testing for input-based vulnerabilities

7.1 Fuzz All Request Parameters

7.1.1 Просмотрите результаты ваших упражнений по картированию приложений и определите каждый отдельный клиентский запрос, который отправляет параметры, которые обрабатывает приложение на стороне сервера. Соответствующие параметры включают элементы в строке запроса URL, параметры в теле запроса и файлы cookie HTTP. Также включите любые другие элементы пользовательского ввода, которые, как было отмечено, влияют на поведение приложения, такие как заголовки Referer или User-Agent.

7.1.2 Чтобы fuzz параметры, вы можете использовать свои собственные сценарии или готовый инструмент для fuzzing. Например, чтобы использовать Burp Intruder, загружайте каждый запрос по очереди в инструмент. Простой способ сделать это - перехватить запрос в Burp Proxy и выбрать действие «Отправить в Intruder» или щелкнуть правой кнопкой мыши элемент в истории Burp Proxy и выбрать этот параметр. Используя эту опцию, можно связать Burp Intruder с содержимым запроса, а также с правильным целевым хостом и портом. Он также автоматически отмечает значения всех параметров запроса как позиции полезной нагрузки, готовые к размыванию.

7.1.3 Используя вкладку полезных нагрузок, настройте подходящий набор полезных нагрузок для атаки, чтобы проверить наличие уязвимостей в приложении. Вы можете вводить полезные нагрузки вручную, загружать их из файла или выбирать один из предварительно установленных списков полезных нагрузок.
Обнаружение каждого параметра запроса в приложении обычно влечет за собой выдачу большого количества запросов и проверку результатов на наличие аномалий.
Если ваш набор строк атаки слишком велик, это может быть контрпродуктивным и генерировать чрезмерно большой объем вывода для просмотра.
Следовательно, разумный подход заключается в нацеливании на ряд распространенных уязвимостей, которые часто можно легко обнаружить в аномальных ответах на конкретные созданные исходные данные и которые часто проявляются где-либо внутри приложения, а не в рамках определенных типов функций. Вот подходящий набор полезных нагрузок, которые вы можете использовать для проверки некоторых распространенных категорий уязвимостей:

SQL Injection
‘
‘--
‘; waitfor delay ‘0:30:0’--
1; waitfor delay ‘0:30:0’--

XSS and Header Injection
xsstest
“><script>alert(‘xss’)</script>

OS Command Injection
|| ping -i 30 127.0.0.1 ; x || ping -n 30 127.0.0.1 &
| ping –i 30 127.0.0.1 |
| ping –n 30 127.0.0.1 |
& ping –i 30 127.0.0.1 &
& ping –n 30 127.0.0.1 &
; ping 127.0.0.1 ;
%0a ping –i 30 127.0.0.1 %0a
` ping 127.0.0.1 `

Path Traversal
../../../../../../../../../etc/passwd
../../../../../../../../../../boot.ini
..\..\..\..\..\..\..\..\..\..\etc\passwd
..\..\..\..\..\..\..\..\..\..\boot.ini

Script Injection
;echo 111111
echo 111111
response.write 111111
:response.write 111111

File Inclusion
http://<your server name>/
http://<nonexistent IP address>/

7.1.4 Все предыдущие полезные нагрузки показаны в буквальном виде. Персонажи ? , ; , &, +, =, и пространство должно быть закодировано URL, потому что они имеют особое значение в HTTP-запросах. По умолчанию Burp Intruder выполняет необходимую кодировку этих символов, поэтому убедитесь, что эта опция не была отключена. (Чтобы восстановить все параметры по умолчанию после более ранней настройки, выберите Burp ,Восстановить по умолчанию.)

7.1.5 В функции Grep Burp Intruder настройте подходящий набор строк, чтобы пометить некоторые распространенные сообщения об ошибках в ответах. Например:

error
exception
illegal

invalid
fail
stack
access
directory
file
not found
varchar
ODBC
SQL
SELECT
111111

Обратите внимание, что строка 111111 включена для проверки на успешные атаки скрипта. Полезные нагрузки на шаге 7.1.3 включают запись этого значения в ответ сервера.

7.1.6 Также выберите опцию Payload Grep, чтобы flag-ответы, содержащие саму полезную нагрузку, указывая на потенциальную уязвимость XSS или впрыска заголовка.

7.1.7 Настройка веб-сервера или слушателя netcat на хосте, который вы указали в первой полезной нагрузке включения файла. Это поможет вам отслеживать попытки подключения, полученные с сервера в результате успешной атаки на удаленное включение файла.

7.1.8 Запустите атаку. Когда он завершится, просмотрите результаты для аномальных ответов, указывающих на наличие уязвимостей. Проверьте расхождения в коде состояния HTTP, длине ответа, времени отклика, появлении ваших настроенных выражений и появлении самой полезной нагрузки. Вы можете щелкнуть каждый заголовок столбца в таблице результатов, чтобы отсортировать результаты по значениям в этом столбце (и щелкнуть Shift, чтобы изменить сортировку результатов). Это позволяет быстро выявлять любые аномалии, которые выделяются из других результатов.

7.1.9 Для каждой потенциальной уязвимости, указанной в результатах вашего тестирования на fuzz, обратитесь к следующим разделам этой методологии. Они описывают подробные шаги, которые вы должны предпринять в отношении каждой категории проблем, чтобы проверить наличие уязвимости и успешно ее использовать.

7.1.10 После того, как вы настроили Burp Intruder для выполнения теста fuzz одного запроса, вы можете быстро повторить тот же тест для других запросов внутри приложение. Просто выберите каждый целевой запрос в Burp Proxy и выберите опцию «Отправить в Intruder». Затем немедленно запустите атаку в Intruder, используя существующую конфигурацию атаки. Таким образом, вы можете запускать большое количество тестов одновременно в отдельных окнах и вручную просматривать результаты, когда каждый тест завершает свою работу.

7.1.11 Если в ваших упражнениях по картографированию были обнаружены какие-либо внеполосные входные каналы, посредством которых управляемый пользователем ввод может быть введен в обработку приложения, вам следует выполнить аналогичное упражнение по fuzzing на этих входных каналах. Отправляйте различные созданные данные, предназначенные для запуска распространенных уязвимостей при обработке в веб-приложении. В зависимости от характера входного канала вам может потребоваться создать для этой цели собственный скрипт или другой жгут.

7.1.12 В дополнение к вашему собственному fuzzing запросов приложений, если у вас есть доступ к автоматизированному сканеру уязвимостей веб-приложений, вы должны запустить его с целевым приложением, чтобы обеспечить основу для сравнения с вашими собственными выводами.

7.2 Test for SQL Injection

7.2.1 Если строки атаки SQL, перечисленные в шаге 7.1.3, приводят к каким-либо аномальным ответам, вручную проверьте обработку соответствующего параметра приложением, чтобы определить, присутствует ли уязвимость инъекции SQL.

7.2.2 Если были возвращены какие-либо сообщения об ошибках в базе данных, выясните их значение.
Используйте раздел «Синтаксис и ссылка на ошибки SQL» в главе 9, чтобы помочь интерпретировать сообщения об ошибках на некоторых распространенных платформах базы данных.

7.2.3 Если отправка одной кавычки в параметре вызывает ошибку или другое аномальное поведение, отправьте две отдельные кавычки. Если этот ввод вызывает исчезновение ошибки или аномального поведения, приложение, вероятно, уязвимо для SQL-инъекции.

7.2.4 Попробуйте использовать общие функции конкатенатора строк SQL для создания строки, эквивалентной некоторому доброкачественному вводу. Если это вызывает тот же ответ, что и исходный доброкачественный ввод, приложение, вероятно, уязвимо. Например, если исходным входом является выражение FOO, вы можете выполнить этот тест, используя следующие элементы (в третьем примере обратите внимание на пространство между двумя кавычками):

‘||’FOO
‘+’FOO
‘ ‘FOO

Как всегда, обязательно используйте символы URL-кода, такие как + и пробел, которые имеют особое значение в HTTP-запросах.

7.2.5 Если исходный ввод числовой, попробуйте использовать математическое выражение, эквивалентное исходному значению. Например, если исходное значение было 2, попробуйте отправить 1 + 1 или 3–1. Если приложение реагирует одинаково, оно может быть уязвимым, особенно если значение числового выражения систематически влияет на поведение приложения.

7.2.6 Если предыдущий тест прошел успешно, вы можете получить дополнительную уверенность в том, что
уязвимость внедрения SQL связана с использованием
математических выражений, специфичных для SQL, для построения определенного значения. Если
логикой приложения можно систематически манипулировать таким образом, оно почти наверняка
уязвимо для внедрения SQL. Например, оба следующих элемента

эквивалентны числу 2:

67-ASCII(‘A’)
51-ASCII(1)

7.2.7 Если какой-либо из тестовых случаев fuzz с использованием команды waitfor привел к аномальной задержке времени до ответа приложения, это является сильным показателем того, что тип базы данных MS-SQL и приложение уязвимо для внедрения SQL. Повторите тест вручную, указав различные значения в параметре waitfor, и определите, систематически ли изменяется время, необходимое для ответа, в зависимости от этого значения. Обратите внимание, что полезная нагрузка вашей атаки может быть вставлена в несколько SQL-запросов, поэтому наблюдаемая временная задержка может быть кратной указанному значению.

7.2.8 Если приложение уязвимо для внедрения SQL, подумайте, какие виды атак возможны и, вероятно, помогут вам достичь ваших целей.
Подробные инструкции, необходимые для выполнения любой из следующих атак, см. в главе 9:

- Измените условия в предложении WHERE, чтобы изменить логику приложения (например, путем введения или 1=1 - для обхода входа в систему).
- Используйте оператор UNION, чтобы ввести произвольный запрос SELECT и объединить результаты с результатами исходного запроса приложения.
- Определите тип базы данных с помощью специфичного для базы данных синтаксиса SQL.
- Если тип базы данных MS-SQL и приложение возвращает сообщения об ошибках ODBC в своих ответах, используйте их для перечисления структуры базы данных и извлечения произвольных данных.
- Если вы не можете найти способ прямого извлечения результатов произвольного введенного запроса, используйте следующие расширенные методы для извлечения данных:
- Извлекайте строковые данные в числовой форме по одному байту за раз.
- Use an out-of-band channel.
- Если вы можете вызвать разные ответы приложения на основе одного произвольного условия, используйте Абсент для извлечения произвольных данных по одному биту за раз.
- Если вы можете вызвать временные задержки на основе одного произвольного условия, используйте их для извлечения данных по одному биту за раз.
- Если приложение блокирует определенные символы или выражения, необходимые для выполнения определенной атаки, попробуйте различные методы обхода, описанные в главе 9, чтобы обойти входной фильтр.
- Если возможно, усиливайте атаку на базу данных и
- подчиненный сервер, используя любые уязвимости или мощные функциив базе данных.

7.3 Test for XSS and Other Response Injection
7.3.1 Identify Refl ected Request Parameters

7.3.1.1 Отсортируйте результаты вашего тестирования fuzz, щелкнув столбец Grep полезной нагрузки, и определите любые совпадения, соответствующие полезным нагрузкам XSS, перечисленным в шаге 7.1.3. Это случаи, когда строки теста XSS были возвращены без изменений в ответах приложения.

7.3.1.2 В каждом из этих случаев просмотрите ответ приложения, чтобы найти местоположение предоставленного ввода. Если это появляется в теле ответа, проверьте наличие уязвимостей XSS. Если входные данные отображаются в любом HTTP-заголовке, проверьте наличие уязвимостей для внедрения заголовка. Если он используется в заголовке местоположения ответа 302 или используется для указания перенаправления каким-либо другим способом, проверьте наличие уязвимостей перенаправления. Обратите внимание, что один и тот же ввод может быть скопирован в несколько мест в ответе и что может присутствовать более одного типа отраженной уязвимости.

7.3.2 Test for Refl ected XSS

7.3.2.1 Для каждого места в теле ответа, где отображается значение параметра запроса, просмотрите окружающий HTML, чтобы определить возможные способы обработки ваших входных данных для выполнения произвольного JavaScript.
Например, вы можете ввести теги <script>, внедрить их в существующий сценарий или поместить созданное значение в атрибут тега.

7.3.2.2 Использовать различные методы обхода фильтров на основе сигнатур, описанные в Глава 12 в качестве ссылки на различные способы, с помощью которых созданный ввод может быть использован для выполнения JavaScript.

7.3.2.3 Попробуйте отправить в приложение различные возможные эксплойты и отслеживать его ответы, чтобы определить, выполняется ли какая-либо фильтрация или очистка входных данных. Если ваша строка атаки возвращается неизмененной, используйте браузер, чтобы окончательно убедиться, что вам удалось выполнить произвольный JavaScript (например, создав диалоговое окно с предупреждением).

7.3.2.4 Если вы обнаружите, что приложение блокирует ввод, содержащий определенные символы или выражения, которые вам необходимо использовать, или кодирует определенные символы в формате HTML, попробуйте различные способы обхода фильтра, описанные в главе 12.

7.3.2.5 Если вы обнаружите уязвимость XSS в запросе POST, ее все равно можно использовать через вредоносный веб-сайт, содержащий форму с требуемыми параметрами и скрипт для автоматической отправки формы. Тем не менее, доступен более широкий спектр механизмов доставки атак, если эксплойт может быть доставлен с помощью запроса GET. Попробуйте отправить те же параметры в запросе GET и посмотрите, будет ли атака по-прежнему успешной. Вы можете использовать действие Изменить метод запроса в прокси-сервере Burp, чтобы преобразовать запрос для вас.

7.3.3 Test for HTTP Header Injection

7.3.3.1 Для каждого места в заголовках ответов, где отображается значение параметра запроса, проверьте, принимает ли приложение данные, содержащие кодированные URL-адресом символы возврата каретки (%0d) и перевода строки (%0a) , и возвращаются ли они в своем ответе без изменений. (Обратите внимание, что вы ищете сами символы новой строки, которые будут отображаться в ответе сервера, а не их эквиваленты в кодировке URL.)

7.3.3.2 Если в заголовках ответов сервера появляется новая строка, когда вы добавляете обработанный ввод, приложение уязвимо для ввода HTTP-заголовка . Это может быть использовано для выполнения различных атак, как описано в Глава 13.

7.3.3.3 Если вы обнаружите, что в ответах сервера возвращается только один из двух символов новой строки, все еще может быть возможно создать рабочий эксплойт, в зависимости от контекста и браузера целевого пользователя.

7.3.3.4 Если вы обнаружите, что приложение блокирует ввод, содержащий символы новой строки, или удаляет эти символы в своем ответе, попробуйте следующие элементы ввода, чтобы проверить эффективность фильтра:

foo%00%0d%0abar
foo%250d%250abar
foo%%0d0d%%0a0abar

7.3.4 Test for Open Redirection

7.3.4.1 Если отраженный ввод используется для указания цели какого-либо перенаправления , проверьте, возможно ли предоставить обработанный ввод, который приводит к произвольному перенаправлению на внешний веб-сайт. Если это так, то такое поведение может быть использовано для придания достоверности атаке в стиле фишинга.

7.3.4.2 Если приложение обычно передает абсолютный URL-адрес в качестве значения параметра, измените имя домена в URL-адресе и проверьте , перенаправляет ли приложение вас на другой домен.

7.3.4.3 Если параметр обычно содержит относительный URL-адрес, измените его на абсолютный URL-адрес для другого домена и проверьте , перенаправляет ли приложение вас на этот домен.

7.3.4.4 Если приложение выполняет некоторую проверку параметра перед выполнением перенаправления, чтобы предотвратить внешнее перенаправление, это часто уязвимо для обходов. Попробуйте различные атаки, описанные в главе 13, чтобы проверить надежность фильтров.

7.3.5 Test for Stored Attacks

7.3.5.1 Если приложение хранит элементы ввода, предоставленные пользователем, и позже отображает их на экране, после того, как вы размыли все приложение, вы можете заметить , что некоторые из ваших строк атаки возвращаются в ответ на запросы, которые сами по себе не содержат этих строк. Обратите внимание на все случаи, когда это происходит, и определите исходную точку входа для сохраняемых данных.

7.3.5.2 В некоторых случаях предоставленные пользователем данные успешно сохраняются только в том случае, если вы выполняете многоступенчатый процесс, который не выполняется при базовом тестировании на нечеткость. Если ваши упражнения по сопоставлению приложений выявили какие-либо функции такого рода, вручную выполните соответствующий процесс и проверьте сохраненные данные на наличие уязвимостей XSS.

7.3.5.3 Если у вас есть достаточный доступ для его тестирования, внимательно изучите любую административную функциональность, в которой данные, полученные от пользователей с низкими привилегиями, в конечном итоге отображаются на экране в сеансе более привилегированных пользователей.
Любые сохраненные уязвимости XSS в функциях такого рода обычно приводят непосредственно к повышению привилегий.

7.3.5.4 Проверять каждый экземпляр, в котором хранятся и отображаются предоставленные пользователем данные для пользователей. Проверьте их на наличие XSS и других атак с ответной инъекцией , описанных ранее.

7.3.5.5 Если вы обнаружите уязвимость, при которой ввод, предоставленный одним пользователем, отображается другим пользователям, определите наиболее эффективную полезную нагрузку атаки, с помощью которой вы можете достичь своих целей, таких как захват сеанса или подделка запроса. Если сохраненные данные отображаются только тому же пользователю, от которого они были получены, попробуйте найти способы связать воедино любые другие обнаруженные вами уязвимости (например, нарушенные средства контроля доступа), чтобы внедрить атаку в сеансы других пользователей.

7.3.5.6 Если приложение позволяет загружать и скачивать файлы, всегда проверяйте эту функциональность на предмет сохраненных атак XSS. Если приложение разрешает файлы HTML, JAR или текстовые файлы и не проверяет или не очищает их содержимое, оно почти наверняка уязвимо. Если он разрешает файлы JPEG и не проверяет, содержат ли они действительные изображения, он, вероятно, уязвим для атак против Пользователи Internet Explorer. Проверьте, как приложение обрабатывает каждый тип файлов, который оно поддерживает, и убедитесь, как браузеры обрабатывают ответы, содержащие HTML вместо обычного типа контента.

7.3.5.7 В каждом месте, где данные, отправленные одним пользователем, отображаются другим пользователям, но где фильтры приложения предотвращают выполнение сохраненной атаки XSS, проверьте, делает ли поведение приложения уязвимым для подделки запросов на месте.

7.4 Test for OS Command Injection

7.4.1 Если какая-либо из строк атаки с внедрением команд, перечисленных в шаге 7.1.3, привела к аномальной задержке времени до того, как приложение отреагировало, это является сильным показателем того, что приложение уязвимо для внедрения команд операционной системы. Повторите тест, вручную указав различные значения в -i или - параметр n и определите, систематически ли изменяется время, необходимое для ответа , в зависимости от этого значения.

7.4.2 Используя любую из строк ввода, которая была признана успешной, попробуйте ввести более интересную команду (например, ls или dir) и определите , можете ли вы получить результаты команды в свой браузер.

7.4.3 Если вы не можете получить результаты напрямую, для вас открыты другие варианты:

- Вы можете попытаться открыть внеполосный канал обратно на свой компьютер. Попробуйте использовать TFTP для копирования инструментов на сервер, используя telnet или netcat для создания обратной оболочки обратно на ваш компьютер и используя команду mail для отправки выходных данных команды через SMTP.
- Вы можете перенаправить результаты своих команд в файл вкорневом каталоге веб-сайта, который затем можно получить непосредственно с помощью браузера. Например: dir > c:\inetpub\wwwroot\foo.txt

7.4.4 Если вы найдете способ вводить команды и получать результаты, вам следует определить свой уровень привилегий (используя whoami или аналогичную команду или пытаясь записать безопасный файл в защищенный каталог).
Затем вы можете попытаться повысить привилегии, получить доступ через черный ход к конфиденциальным данным приложения или атаковать другие хосты, до которых можно добраться с взломанного сервера.

7.4.5 Если вы считаете, что ваш ввод передается какой-либо команде операционной системы, но перечисленные строки атаки неудачны, посмотрите, можете ли вы использовать символ <или> для направления содержимого файла на вход команды или для направления вывода команды в файл. Это может позволить вам читать или записывать произвольное содержимое файла. Если вы знаете или можете догадаться о фактической выполняемой команде, попробуйте ввести параметры командной строки , связанные с этой командой, чтобы изменить ее поведение полезными способами (например, указав файл вывода в корневом каталоге веб-сайта).

7.4.6 Если вы обнаружите, что приложение экранирует определенные ключевые символы, необходимые для выполнения атаки с использованием командной строки, попробуйте поместить escape-символ перед каждым таким символом. Если приложение не избегает самого экранирующего символа, это обычно приводит к обходу этой защитной меры.
Если вы обнаружите, что пробелы заблокированы или удалены, вы можете использовать $IFS вместо пробелов на платформах на базе UNIX.

7.5 Test for Path Traversal

7.5.1 Для каждого выполненного вами теста на нечеткость просмотрите результаты, сгенерированные строками атаки с обходом пути, перечисленными в шаге 7.1.3. Вы можете щелкнуть верхнюю часть столбца полезной нагрузки в Burp Intruder, чтобы отсортировать таблицу результатов по полезной нагрузке и сгруппировать результаты для этих строк. В любых случаях, когда было получено необычное сообщение об ошибке или ответ с ненормальной длиной , просмотрите ответ вручную, чтобы определить, содержит ли он содержимое указанного файла или другие доказательства того, что аномальный произошла файловая операция.

7.5.2 При отображении области атаки приложения вы должны были отметить любую функциональность, которая конкретно поддерживает чтение и запись файлов на основе введенных пользователем данных. В дополнение к общему размытию всех параметров, вы должны вручную очень тщательно протестировать эту функциональность, чтобы выявить любые существующие уязвимости при обходе пути.

7.5.3 Если параметр содержит имя файла, часть имени файла или каталог, измените существующее значение параметра, чтобы вставить произвольный подкаталог и одну последовательность обхода. Например, если приложение отправляет этот параметр:
file=foo/file1.txt
попробуйте отправить это значение:
file=foo/bar/../file1.txt
Если поведение приложения в обоих случаях идентично, оно может быть уязвимым, и вам следует перейти к следующему шагу. Если поведение отличается тем, что приложение может блокировать, удалять или очищать последовательности обхода, что приводит к недопустимому пути к файлу. Попробуйте использовать кодировку и другие атаки, описанные в главе 10, в попытке обойти фильтры.

7.5.4 Если предыдущая проверка использования последовательностей обхода в базовом каталоге прошла успешно, попробуйте использовать дополнительные последовательности, чтобы подняться над базовым каталогом и получить доступ к известным файлам в операционной системе сервера . Если эти попытки не увенчаются успехом, приложение может вводить различные фильтры или проверки перед предоставлением доступа к файлам. Вам следует продолжить исследование, чтобы понять, какие элементы управления реализованы и существуют ли какие-либо обходы.

7.5.5 Приложение может проверять запрашиваемое расширение файла и разрешать доступ только к определенным видам файлов. Попробуйте использовать атаку с нулевым байтом или новой строкой вместе с известным принятым расширением файла в попытке обойти фильтр. Например:
../../../../../boot.ini%00.jpg
../../../../../etc/passwd%0a.jpg

7.5.6 Приложение может проверять, что путь к файлу, предоставленный пользователем, начинается с определенного каталога или стержня. Попробуйте добавить последовательности обхода после известного принятого стержня в попытке обойти фильтр. Например:
/изображения/../../../../../../../etc/passwd

7.5.7 Если эти атаки не увенчались успехом, попробуйте объединить несколько обходов, работая изначально полностью в базовом каталоге, пытаясь понять существующие фильтры и способы, которыми приложение обрабатывает неожиданный ввод.

7.5.8 Если вам удастся получить доступ на чтение к произвольным файлам на сервере, попытайтесь получить любой из следующих файлов, которые могут позволить вам усилить атаку:

- Файлы паролей для операционной системы и приложения
- Файлы конфигурации сервера и приложений, чтобы обнаружить другие возможности или точно настроить другую атаку
- Включать файлы, которые могут содержать учетные данные базы данных
- Источники данных, используемые приложением, такие как файлы базы данных MySQL или XML-файлы
- Исходный код для исполняемых страниц сервера, чтобы выполнить проверку кода в поисках ошибок
- Файлы журнала приложений, которые могут содержать такую информацию, как имена пользователей и маркеры сеанса

7.5.9 Если вам удастся получить доступ на запись к произвольным файлам на сервере, проверьте, возможны ли какие-либо из следующих атак, чтобы усилить вашу атаку:

- Создание сценариев в папках запуска пользователей
- Изменение файлов, in.ftpd, для выполнения произвольных команд при следующем подключении пользователя
- Написание сценариев в веб-каталог с разрешениями на выполнение и вызов их из вашего браузера

7.6 Test for Script Injection

7.6.1 Для каждого выполненного вами теста на нечеткость просмотрите результаты для строки 111111 сам по себе (то есть, ему не предшествует остальная часть тестовой строки). Вы можете быстро определить их в нарушителе отрыжки, щелкнув по заголовку строки Grep 111111, чтобы сгруппировать все результаты, содержащие эту строку. Найдите все, у которых нет проверки в столбце Grep полезной нагрузки. Любые выявленные случаи, вероятно, будут уязвимы для внедрения команд сценариев.

7.6.2 Просмотрите все тестовые случаи, в которых использовались строки ввода скрипта, и определите любые содержащие сообщения об ошибках сценариев, которые могут указывать на то, что ваш ввод выполняется, но вызвал ошибку. Возможно, их потребуется настроить для успешного внедрения сценария.

7.6.3 Если приложение кажется уязвимым, проверьте это, введя дополнительные команды, указанные в используемой платформе сценариев. Например, вы можете использовать полезные нагрузки атаки, аналогичные тем, которые используются при фаззинге для внедрения команд операционной системы:

system(‘ping%20127.0.0.1’)

7.7 Test for File Inclusion

7.7.1 Если вы получили какие-либо входящие HTTP-соединения от инфраструктуры целевого приложения во время обработки, приложение почти наверняка уязвимо для удаленного включения файлов. Повторите соответствующие тесты однопоточным и регулируемым по времени способом, чтобы точно определить, какие параметры заставляют приложение выдавать HTTP-запросы.

7.7.2 Просмотрите результаты тестовых случаев включения файлов и определите все, что вызвало аномальную задержку в ответе приложения. В этих случаях может оказаться, что само приложение уязвимо, но что в результате Время ожидания HTTP-запросов истекает из-за фильтров сетевого уровня.

7.7.3 Если вы обнаружили уязвимость удаленного включения файлов, разверните веб-сервер , содержащий вредоносный сценарий, специфичный для языка, на который вы ориентируетесь, и используйте команды, подобные тем, которые использовались для проверки внедрения сценария, чтобы убедиться , что ваш сценарий выполняется.

8 Test for Function-Specifi c Input Vulnerabilities

В дополнение к атакам на основе ввода, нацеленным на предыдущем шаге, ряд уязвимостей обычно проявляется только в определенных видах функциональности. Прежде чем перейти к отдельным шагам, описанным в этом разделе, вы должны пересмотреть свою оценку поверхности атаки приложения, чтобы определить конкретные функции приложения, в которых могут возникнуть эти дефекты, и сосредоточить свое тестирование на них.

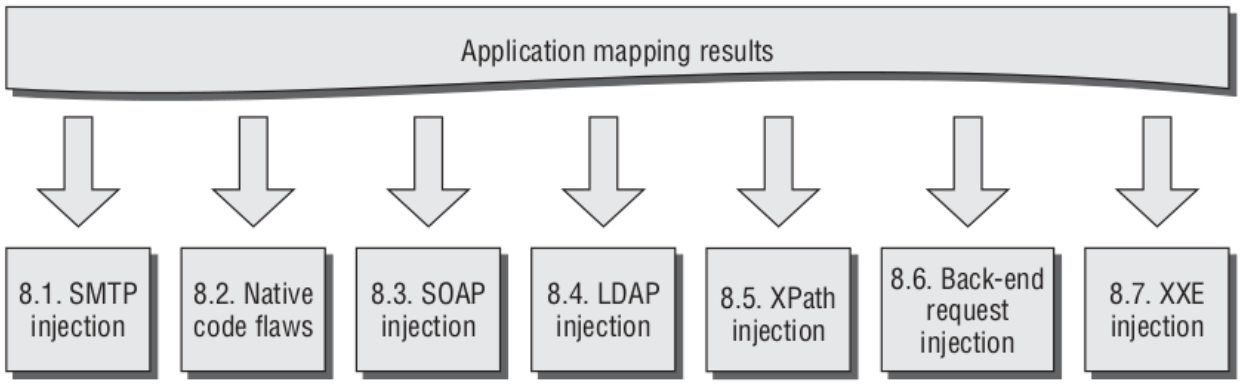

Figure 21-9: Testing for functionality-specific input vulnerabilities

8.1 Test for SMTP Injection

8.1.1 Для каждого запроса, используемого в функциях, связанных с электронной почтой, отправьте каждую из следующих тестовых строк в качестве каждого параметра по очереди, указав свой собственный адрес электронной почты в соответствующей позиции. Вы можете использовать нарушителя отрыжки для автоматизации этого, как описано в шаге 7.1 для

общего размытия. Эти тестовые строки уже содержат специальные символы, закодированные в URL, поэтому не применяйте к ним никакой дополнительной кодировки.

<youremail>%0aCc:<youremail>
<youremail>%0d%0aCc:<youremail>
<youremail>%0aBcc:<youremail>
<youremail>%0d%0aBcc:<youremail>
%0aDATA%0afoo%0a%2e%0aMAIL+FROM:+<youremail>%0aRCPT+TO:+<youremail>
%0aDATA%0aFrom:+<youremail>%0aTo:+<youremail>%0aSubject:+test%0afoo
%0a%2e%0a
%0d%0aDATA%0d%0afoo%0d%0a%2e%0d%0aMAIL+FROM:+<youremail>%0d%0aRCPT
+TO:+
<youremail>%0d%0aDATA%0d%0aFrom:+<youremail>%0d%0aTo:+<youremail>
%0d%0aSubject:+test%0d%0afoo%0d%0a%2e%0d%0a

8.1.2 Просмотрите результаты, чтобы определить любые сообщения об ошибках, возвращаемые приложением. Если они, по-видимому, связаны с какой-либо проблемой в функции электронной почты, выясните, нужно ли вам настроить ввод данных для использования уязвимости.

8.1.3 Следите за указанным вами адресом электронной почты, чтобы узнать, получены ли какие-либо сообщения электронной почты.

8.1.4 Внимательно просмотрите HTML-форму, которая генерирует соответствующий запрос. Он может содержать подсказки относительно используемого программного обеспечения на стороне сервера. Он также может содержать скрытое или отключенное поле, которое используется для указания На адрес электронной почты, который вы можете изменить напрямую.

8.2 Test for Native Software Vulnerabilities

8.2.1.1 Для каждого объекта данных, на который нацелена цель, отправьте ряд длинных строк с длина несколько длиннее, чем у обычных буферных размеров. Целевой один элемент данные за раз, чтобы максимизировать охват кодовых путей в приложении. Вы можете использовать источник полезной нагрузки блоков символов в Burp Intruder to автоматически генерировать полезные нагрузки различных размеров. Следующий буфер размеры подходят для тестирования:

1100
4200
33000

8.2.1.2 Мониторинг ответов приложения для выявления любых аномалий.
Неконтролируемый переполнение почти наверняка вызовет исключение в приложении, хотя диагностика характера проблемы может быть удаленной
сложно. Ищите любую из следующих аномалий:

- Код состояния HTTP 500 или сообщение об ошибке, где другие искаженные (но не слишком длинные) данные не имеют такого же эффект

- Информационное сообщение, указывающее, что произошел сбой в каком -либо внешнем компоненте машинного кода
- Неполный или искаженный ответ, полученный от сервера
- TCP-соединение с сервером внезапно закрывается без ответа
- Все веб-приложение больше не отвечает
- Неожиданные данные, возвращаемые приложением, возможно, указывают на то, что строка в памяти потеряла свой нулевой терминатор

8.2.2 Test for Integer Vulnerabilities

8.2.2.1 При работе с компонентами машинного кода идентифицируйте любые данные на основе целых чисел, в частности индикаторы длины, которые могут использоваться для запуска уязвимостей целых чисел.

8.2.2.2 В каждом целевом элементе отправляйте подходящие полезные нагрузки, предназначенные для запуска любых уязвимостей. Для каждого целевого элемента данных по очереди отправляйте ряд различных значений, представляющих граничные случаи для подписанных и неподписанных версий разных размеров целого числа. Например:

- 0x7f и 0x80 (127 и 128)
- 0xff и 0x100 (255 и 256)
- 0x7ffff и 0x8000 (32767 и 32768)
- 0xffff и 0x10000 (65535 и 65536)
- 0x7fffffff и 0x80000000 (2147483647 и 2147483648)
- 0xffffffff and 0x0 (4294967295 and 0)

8.2.2.3 Когда изменяемые данные представлены в шестнадцатеричной форме, отправляйте версии каждого теста как с малым, так и с большим порядком, такие как ff7f и 7fff. Если шестнадцатеричные числа представлены в форме ASCII, используйте тот же регистр, что и само приложение для буквенных символов, чтобы убедиться, что они правильно декодированы.

8.2.2.4 Отслеживать ответы приложения на аномальные события, как описано в шаге 8.2.1.2.

8.2.3 Test for Format String Vulnerabilities

Ориентируясь на каждый параметр по очереди, отправляйте строки, содержащие длинные последовательности различных спецификаторов формата. Например:

%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n
%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s
%1!n!%2!n!%3!n!%4!n!%5!n!%6!n!%7!n!%8!n!%9!n!%10!n! etc...
%1!s!%2!s!%3!s!%4!s!%5!s!%6!s!%7!s!%8!s!%9!s!%10!s! Etc…

Не забудьте кодировать символ % в URL-адресе как %25.

8.2.3.2 Monitor the application’s responses for anomalous events, as described in step 8.2.1.2.

8.3 Test for SOAP Injection

8.3.1 Нацеливайте по очереди каждый параметр, который, как вы подозреваете, обрабатывается с помощью сообщения SOAP. Отправьте неверный закрывающий тег XML, например </foo>. Если ошибка не возникает, ваш ввод, вероятно, не вставляется в сообщение SOAP или каким-либо образом очищается.

8.3.2 Если была получена ошибка, отправьте вместо нее допустимую пару открывающих и закрывающих тегов, например <foo></foo>. Если это приведет к исчезновению ошибки, приложение может быть уязвимым.

8.3.3 Если отправленный вами элемент копируется обратно в ответы приложения, отправьте по очереди следующие два значения. Если вы обнаружите, что один из элементов возвращается как другой или просто как тест, вы можете быть уверены, что ваши входные данные вставляются в сообщение на основе XML.

test<foo/>
test<foo></foo>

8.3.4 Если HTTP-запрос содержит несколько параметров, которые могут быть помещены в сообщение SOAP, попробуйте вставить символ открывающего комментария <!-- в один параметр и символ закрывающего комментария!--> в другой параметр. Затем переключите их (потому что у вас нет возможности узнать, в каком порядке отображаются параметры). Это может привести к комментированию части сообщения SOAP сервера, что может изменить логику приложения или привести к другому состоянию ошибки, которое может привести к разглашению информации.

8.4 Test for LDAP Injection

8.4.1 В любой функциональности, где данные, предоставленные пользователем, используются для извлечения информации из службы каталогов, поочередно настраивайте каждый параметр, чтобы проверить возможность внедрения в запрос LDAP.

8.4.2 Отправьте символ *. Если возвращается большое количество результатов, это хороший показатель того, что вы имеете дело с запросом LDAP.

8.4.3 Попробуйте ввести несколько закрывающих скобок:

))))))))))

Этот ввод делает синтаксис запроса недействительным, поэтому в случае ошибки или другого аномального поведения приложение может быть уязвимым (хотя многие другие функции приложения и ситуации внедрения могут вести себя аналогичным образом).

8.4.4 Попробуйте ввести различные выражения, предназначенные для взаимодействия с различными типами запросов, и посмотрите, позволяют ли они вам влиять на возвращаемые

результаты. Атрибут cn поддерживается всеми реализациями LDAP и полезен, если вы не знаете никаких подробностей о запрашиваемом каталоге:

)(cn=*
))(|(cn=
*))%00

8.4.5 Попробуйте добавить дополнительные атрибуты в конец ввода, используя запятые для разделения каждого элемента. Проверьте каждый атрибут по очереди. Ошибка указывает на то, что атрибут недопустим в данном контексте. Следующие атрибуты обычно используются в каталогах, запрашиваемых LDAP:

cn
c
mail
givenname
o
ou
dc
l
uid
objectclass
postaladdress
dn
sn

8.5 Test for XPath Injection

8.5.1 Попробуйте ввести следующие значения и определите, приводят ли они к другому поведению приложения, не вызывая ошибки:

‘ or count(parent::*[position()=1])=0 or ‘a’=’b
‘ or count(parent::*[position()=1])>0 or ‘a’=’b

8.5.2 Если параметр числовой, также попробуйте выполнить следующие тестовые строки:

or count(parent::*[position()=1])=0
1 or count(parent::*[position()=1])>0

8.5.3 Если какая-либо из предыдущих строк вызывает различное поведение в приложении, не вызывая ошибки, вполне вероятно, что вы можете извлечь произвольные данные, создав условия тестирования для извлечения 1 байта информации за раз. Используйте ряд условий со следующей формой, чтобы определить имя родительского узла текущего узла:

substring(name(parent::*[position()=1]),1,1)=’a’

8.5.4 После извлечения имени родительского узла используйте ряд условий со следующей формой для извлечения всех данных в дереве XML:

substring(//parentnodename[position()=1]/child::node()[position()=1]/text(),1,1)=’a’

8.6 Test for Back-End Request Injection

8.6.1 Найдите любой экземпляр, в котором в параметре указано имя внутреннего сервера или IP-адрес. Отправьте произвольный сервер и порт и следите за временем ожидания приложения. Также отправьте localhost и, наконец, свой собственный IP-адрес, отслеживая входящие соединения на указанном порту

8.6.2 Выберите параметр запроса, который возвращает определенную страницу для определенного значения, и попытайтесь добавить новый введенный параметр, используя различный синтаксис, в том числе следующий:

%26foo%3dbar (URL-encoded &foo=bar)
%3bfoo%3dbar (URL-encoded ;foo=bar)
%2526foo%253dbar (Double URL-encoded &foo=bar)

Если приложение ведет себя так, как если бы исходный параметр не был изменен, существует вероятность уязвимости при внедрении параметров HTTP. Попытайтесь атаковать серверный запрос, введя известные пары имя/значение параметра, которые могут изменить логику серверной части, как описано в главе 10.

8.7 Test for XXE Injection

8.7.1 Если пользователи отправляют XML на сервер, может быть возможна атака с внедрением внешних сущностей. Если известно поле, которое возвращается пользователю, попытайтесь указать внешнюю сущность, как в следующем примере:

POST /search/128/AjaxSearch.ashx HTTP/1.1
Host: mdsec.net
Content-Type: text/xml; charset=UTF-8
Content-Length: 115
<!DOCTYPE foo [<!ENTITY xxe SYSTEM “file:///windows/win.ini” >]>
<Search><SearchTerm>&xxe;</SearchTerm></Search>

Если не удается найти ни одного известного поля, укажите внешний объект “http://192.168.1.1:25 ”и следите за временем отклика страницы. Если возврат страницы занимает значительно больше времени или время ожидания, она может быть уязвимой.

9 Test for Logic Flaws

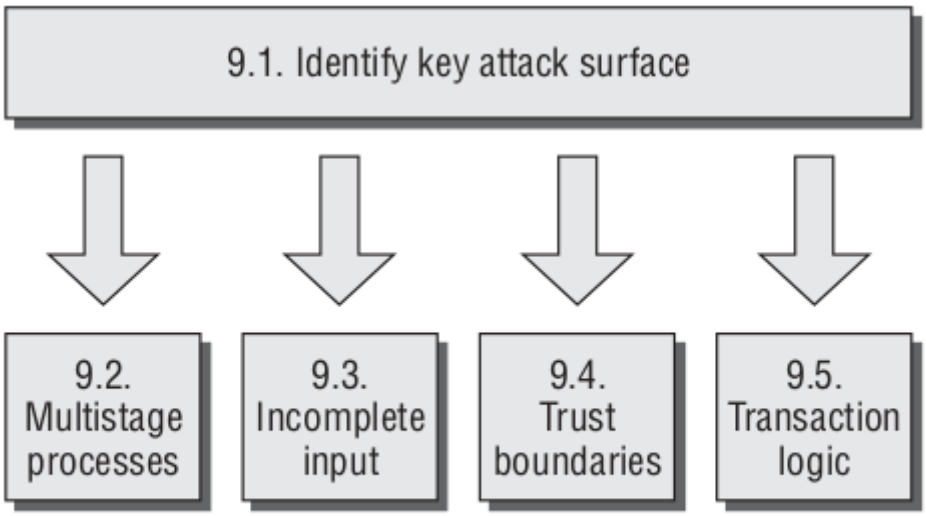

Figure 21-10: Testing for logic flaws

9.1 Identify the Key Attack Surface

9.1.1 Логические недостатки могут принимать самые разнообразные формы и существовать в любом аспекте функциональности приложения. Чтобы убедиться, что поиск логических ошибок возможен, вы должны сначала сузить область атаки до разумной области для ручного тестирования.

9.1.2 Просмотрите результаты ваших упражнений по сопоставлению приложений и определите любые примеры следующих функций:

- Многоступенчатые процессы
- Важные функции безопасности, такие как вход в систему
- Переходы через границы доверия (например, переход от анонимности к самостоятельной регистрации и входу в систему)
- Контекстно-ориентированная функциональность, предоставляемая пользователю
- Проверки и корректировки, внесенные в цены или количества транзакций

9.2 Test Multistage Processes

9.2.1 Если многоступенчатый процесс включает определенную последовательность запросов, попытайтесь отправить эти запросы вне ожидаемой последовательности. Попробуйте пропустить определенные этапы, получить доступ к одному этапу более одного раза и получить доступ к более ранним этапам после более поздних.

99.2.2 Последовательность этапов может быть доступна через серию запросов GET или POST для разных URL-адресов, или они могут включать отправку разных наборов параметров на один и тот же URL-адрес. Вы можете указать запрашиваемый этап, отправив имя функции или индекс в параметре запроса. Убедитесь, что вы полностью понимаете механизмы, которые использует приложение для предоставления доступа к различным этапам.

9.2.3 В дополнение к вмешательству в последовательность шагов, попробуйте взять параметры, представленные на одном этапе процесса, и представить их на другом этапе. Если соответствующие элементы данных обновляются в состоянии приложения, вам следует изучить, можно ли использовать это поведение для вмешательства в логику приложения.

9.2.4 Если в многоступенчатом процессе разные пользователи выполняют операции с одним и тем же набором данных, попробуйте взять каждый параметр, представленный одним пользователем, и отправить его как другой. Если они приняты и обработаны как этот пользователь, изучите последствия такого поведения, как описано ранее.

9.2.5 Исходя из контекста реализованной функциональности, попытайтесь понять, какие предположения могли сделать разработчики и где лежит поверхность атаки. Попытайтесь определить способы нарушения этих предположений, чтобы вызвать нежелательное поведение в приложении.

9.2.6 При неупорядоченном доступе к многоступенчатым функциям в приложении часто возникают различные аномальные условия, такие как переменные с нулевыми или неинициализированными значениями, частично определенное или несогласованное состояние и другое непредсказуемое поведение. Ищите интересные сообщения об ошибках и выходные данные отладки, которые вы можете использовать, чтобы лучше понять внутреннюю работу приложения и тем самым точно настроить текущую или другую атаку.

9.3 Test Handling of Incomplete Input

9.3.1 Для критически важных функций безопасности в приложении, которые включают обработку нескольких элементов пользовательского ввода и принятие на их основе решения, проверьте устойчивость приложения к запросам, содержащим неполный ввод.

9.3.2 Для каждого параметра по очереди удалите из запроса как имя, так и значение параметра. Отслеживайте ответы приложения на любые отклонения в его поведении и любые сообщения об ошибках, которые проливают свет на выполняемую логику.

9.3.3 Если запрос, которым вы манипулируете, является частью многоступенчатого процесса, выполните его до конца, поскольку приложение может хранить данные, отправленные на более ранних этапах сеанса, а затем обрабатывать их на более позднем этапе.

9.4 Test Trust Boundaries

9.4.1 Проверьте, как приложение обрабатывает переходы между различными типами доверия пользователя. Ищите функциональность, в которой пользователь с заданным статусом доверия может накапливать информацию о состоянии, относящуюся к его личности. Например, анонимный пользователь может предоставить личную информацию во время самостоятельной регистрации или пройти часть процесса восстановления учетной записи, предназначенного для установления его личности.

9.4.2 Попытайтесь найти способы совершения неправильных переходов через границы доверия путем накопления соответствующего состояния в одной области, а затем

переключения в другую область способом, который обычно не происходит. Например, выполнив часть процесса восстановления учетной записи, попытайтесь переключиться на страницу, указанную пользователем, прошедшим проверку подлинности. Проверьте, не назначает ли приложение вам неподходящий уровень доверия при переходе таким образом.

9.4.3 Попытайтесь определить, можете ли вы использовать какую-либо функцию с более высокими привилегиями прямо или косвенно для доступа или вывода информации.

9.5 Test Transaction Logic

9.5.1 В случаях, когда приложение устанавливает ограничения на транзакции, проверьте последствия предоставления отрицательных значений. Если они будут приняты, возможно, удастся преодолеть ограничения, совершая крупные транзакции в противоположном направлении.

9.5.2 Проверьте, можете ли вы использовать серию последовательных транзакций для создания состояния, которое вы можете использовать в полезных целях. Например, вы можете выполнить несколько переводов с низкой стоимостью между счетами, чтобы накопить большой баланс, который логика приложения должна была предотвратить.

9.5.3 Если приложение корректирует цены или другие чувствительные значения на основе критериев, которые определяются контролируемыми пользователем данными или действиями, сначала поймите алгоритмы, используемые приложением, и точку в его логике, в которой производятся корректировки. Определите, вносятся ли эти корректировки разово или они пересматриваются в ответ на дальнейшие действия, выполняемые пользователем.

9.5.4 Попытаться найти способы манипулировать поведением приложения, чтобы привести его в состояние, в котором примененные им корректировки не соответствуют первоначальным критериям, предусмотренным его разработчиками.

10 Test for Shared Hosting Vulnerabilities

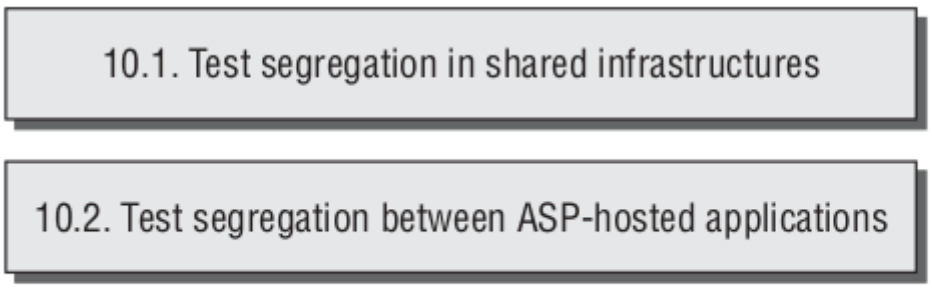

Figure 21-11: Testing for shared hosting vulnerabilities

g

10.1 Test Segregation in Shared Infrastructures

10.1.1 Если приложение размещено в общей инфраструктуре, изучите механизмы доступа, предоставляемые клиентам общей среды, для обновления и управления их контентом и функциональностью. Рассмотрим следующие вопросы:

- Использует ли средство удаленного доступа безопасный протокол и соответствующую защищенную инфраструктуру?
- Могут ли клиенты получить доступ к файлам, данным и другим ресурсам, доступ к которым им не требуется на законных основаниях?
- Могут ли клиенты получить интерактивную оболочку в среде хостинга и выполнять произвольные команды?

10.1.2 Если проприетарное приложение используется для того, чтобы позволить клиентам настраивать и настраивать общую среду, рассмотрите возможность использования этого приложения как способа поставить под угрозу саму среду и отдельные приложения, работающие в ней.

10.1.3 Если вы можете добиться выполнения команд, внедрения SQL или произвольного доступа к файлам в одном приложении, тщательно изучите, обеспечивает ли это какой-либо способ эскалации вашей атаки на другие приложения.

10.2 Test Segregation Between ASP-Hosted Applications

10.2.1 Если приложение принадлежит службе, размещенной на ASP, состоящей из набора общих и настраиваемых компонентов, определите любые общие компоненты, такие как механизмы ведения журнала, административные функции и компоненты кода базы данных. Попытайтесь использовать их, чтобы скомпрометировать общую часть приложения и тем самым атаковать другие отдельные приложения.

10.2.2 Если в какой-либо общей среде используется общая база данных, выполните всесторонний аудит конфигурации базы данных, уровня исправлений, структуры таблиц и разрешений с помощью инструмента сканирования базы данных, такого как NGSSquirrel. Любые дефекты в модели безопасности базы данных могут привести к эскалации атаки из одного приложения в другое.

11 Test for Application Server Vulnerabilities

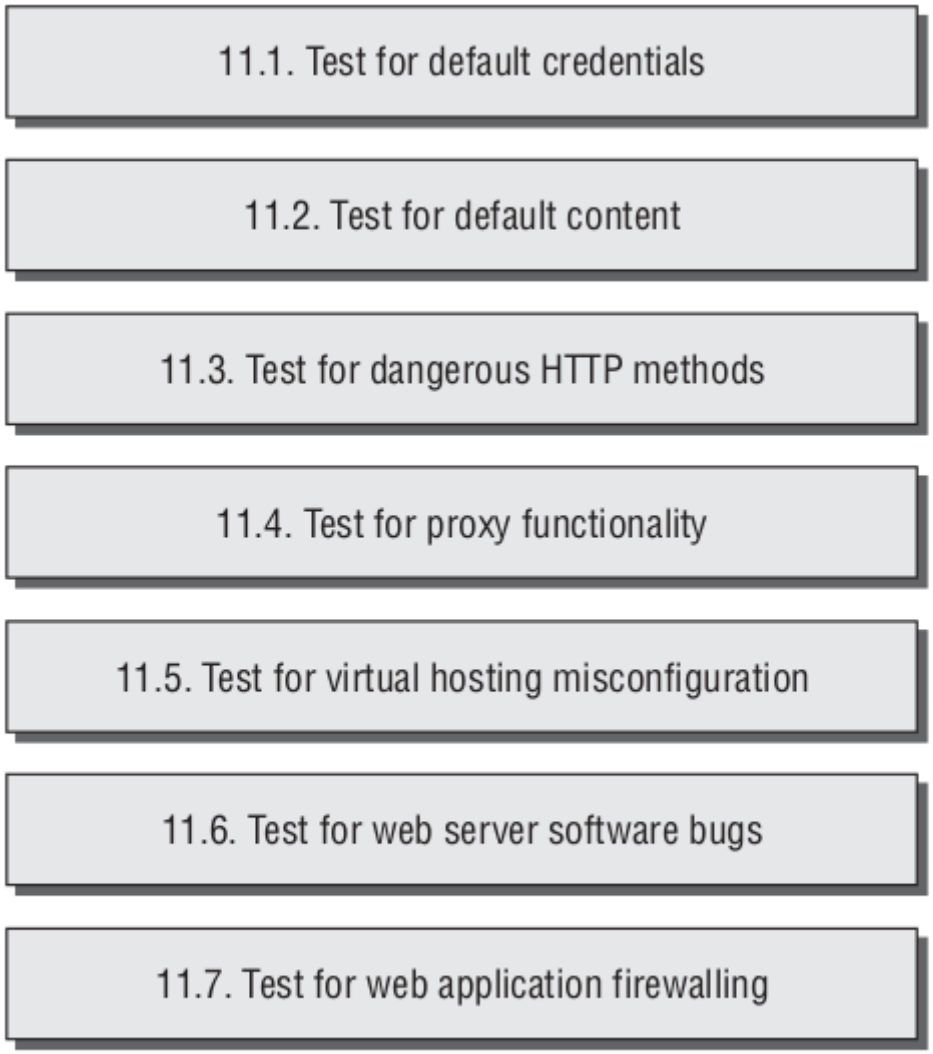

Figure 21-12: Testing for web server vulnerabilities

11.1 Test for Default Credentials

11.1.1 Просмотрите результаты ваших упражнений по сопоставлению приложений, чтобы определить веб-сервер и другие используемые технологии, которые могут содержать доступные административные интерфейсы.

11.1.2 Выполните сканирование портов веб-сервера, чтобы определить любые административные интерфейсы, работающие на другом порту, чем основное целевое приложение.

11.1.3 Для любых идентифицированных интерфейсов обратитесь к документации производителя и спискам общих паролей по умолчанию, чтобы получить учетные данные по умолчанию.

11.1.4 Если учетные данные по умолчанию не работают, выполните действия, перечисленные в разделе 4, чтобы попытаться угадать действительные учетные данные.

11.1.5 Если вы получаете доступ к административному интерфейсу, просмотрите доступные функциональные возможности и определите, можно ли их использовать для дальнейшей компрометации хоста и атаки на основное приложение.

11.2 Test for Default Content

11.2.1 Просмотрите результаты сканирования Nikto (шаг 1.4.1), чтобы определить любой контент по умолчанию, который может присутствовать на сервере, но не является неотъемлемой частью приложения.

11.2.2 Использовать поисковые системы и другие ресурсы, такие как www.exploit-db.com и www.osvdb.org для определения содержимого и функций по умолчанию, включенных в технологии, которые, как вам известно, используются. Если это возможно, выполните их локальную установку и проверьте их на наличие любых функций по умолчанию, которые вы можете использовать в своей атаке.

11.2.3 Проверьте содержимое по умолчанию на наличие любых функций или уязвимостей, которые вы можете использовать для атаки на сервер или приложение.

11.3 Test for Dangerous HTTP Methods

11.3.1 Используйте метод OPTIONS для перечисления HTTP-методов, доступных в состояниях сервера. Обратите внимание, что в разных каталогах могут быть включены разные методы. Для выполнения этой проверки вы можете выполнить сканирование уязвимостей в Paros.

11.3.2 Попробуйте каждый описанный метод вручную, чтобы подтвердить, действительно ли он может быть использован.

11.3.3 Если вы обнаружите, что некоторые методы WebDAV включены, используйте клиент с поддержкой WebDAV для дальнейшего изучения, например Microsoft FrontPage или опцию Открыть как веб-папку в Internet Explorer.

11.4 Test for Proxy Functionality

11.4.1 Используя запросы GET и CONNECT, попробуйте использовать веб-сервер в качестве прокси-сервера для подключения к другим серверам в Интернете и получения контента с них.

11.4.2 Используя запросы GET и CONNECT, попытайтесь подключиться к различным IP-адресам и портам в инфраструктуре хостинга.

11.4.3 Используя запросы GET и CONNECT, попытайтесь подключиться к общим номерам портов на самом веб-сервере, указав 127.0.0.1 в качестве целевого хоста в запросе.

11.5 Test for Virtual Hosting Misconfiguration

11.5.1 Отправлять запросы GET в корневой каталог, используя следующее:
- Правильный заголовок хоста
- Поддельный заголовок хоста
- IP-адрес сервера в заголовке хоста
- Нет заголовка хоста (используйте только HTTP/1.0)

11.5.2 Сравните ответы на эти запросы. Общим результатом является то, что списки каталогов получаются, когда IP-адрес сервера используется в заголовке хоста. Вы также можете обнаружить, что доступно другое содержимое по умолчанию.

11.5.3 Если вы наблюдаете другое поведение, повторите упражнения по сопоставлению приложений, описанные в разделе 1, используя имя хоста, которое дало другие результаты. Обязательно выполните сканирование Nikto с помощью параметра -vhost, чтобы определить любое содержимое по умолчанию, которое могло быть пропущено при первоначальном сопоставлении

11.6 Test for Web Server Software Bugs

11.6.1 Запустите Nessus и любые другие подобные сканеры, которые у вас есть, для выявления любых известных уязвимостей в программном обеспечении веб-сервера, которое вы атакуете.

11.6.2 Просмотрите такие ресурсы, как Security Focus, Bugtraq и Полное раскрытие, чтобы найти подробную информацию о любых недавно обнаруженных уязвимостях, которые, возможно, не были исправлены в вашей цели.

11.6.3 Если приложение было разработано третьей стороной, выясните, поставляется ли оно с собственным веб-сервером (часто сервером с открытым исходным кодом). Если это произойдет, проверьте это на наличие каких-либо уязвимостей. Имейте в виду, что в этом случае стандартный баннер сервера, возможно, был изменен.

11.6.4 Если возможно, рассмотрите возможность локальной установки программного обеспечения, которое вы атакуете, и проведите собственное тестирование, чтобы найти новые уязвимости, которые не были обнаружены или широко распространены.

11.7 Test for Web Application Firewalling

11.7.1 Отправьте произвольное имя параметра приложению с четкой полезной нагрузкой атаки в значении, в идеале где-нибудь приложение включает имя и/или значение в ответ. Если приложение блокирует атаку, это, скорее всего, связано с внешней защитой.

11.7.2 Если может быть передана переменная, возвращаемая в ответе сервера, отправьте диапазон нечетких строк и закодированных вариантов, чтобы определить поведение защиты приложения для ввода пользователем.

11.7.3 Подтвердите это поведение, выполнив те же атаки на переменные в приложении.

11.7.4 Для всех строк и запросов размытия используйте строки полезной нагрузки, которые вряд ли существуют в стандартной базе данных сигнатур. Хотя приводить примеры из них по определению невозможно, избегайте использования /etc/passwd или /windows/system32/config/sam в качестве полезной нагрузки для извлечения файлов. Также избегайте использования таких терминов, как <script> в атаке XSS, и использования alert() или xss в качестве полезных нагрузок XSS.

11.7.5 Если конкретный запрос заблокирован, попробуйте отправить тот же параметр в другом месте или контексте. Например, отправьте один и тот же параметр в URL-адресе в запросе GET, в теле запроса POST и в URL-адресе в запросе POST.11.7.5 Если конкретный запрос заблокирован, попробуйте отправить тот же параметр в другом месте или контексте. Например, отправьте один и тот же параметр в URL-адресе в запросе GET, в теле запроса POST и в URL-адресе в запросе POST.

11.7.6 Вкл. ASP.NET, также попробуйте отправить параметр в виде файла cookie. Запрос API.Параметры[“foo”] извлекут значение файла cookie с именем foo, если параметр foo не найден в строке запроса или теле сообщения.

11.7.7 Просмотрите все другие методы ввода пользовательских данных, представленные в главе 4, выбрав любые, которые не защищены.

11.7.8 Определите места, где вводимые пользователем данные (или могут быть) представлены в нестандартном формате, таком как сериализация или кодирование. Если таковая недоступна, создайте строку атаки путем объединения и/или путем ее охвата несколькими переменными. (Обратите внимание, что если цель ASP.NET , вы можете использовать HPP для объединения атаки с использованием нескольких спецификаций одной и той
же переменной.)

12 Miscellaneous Checks

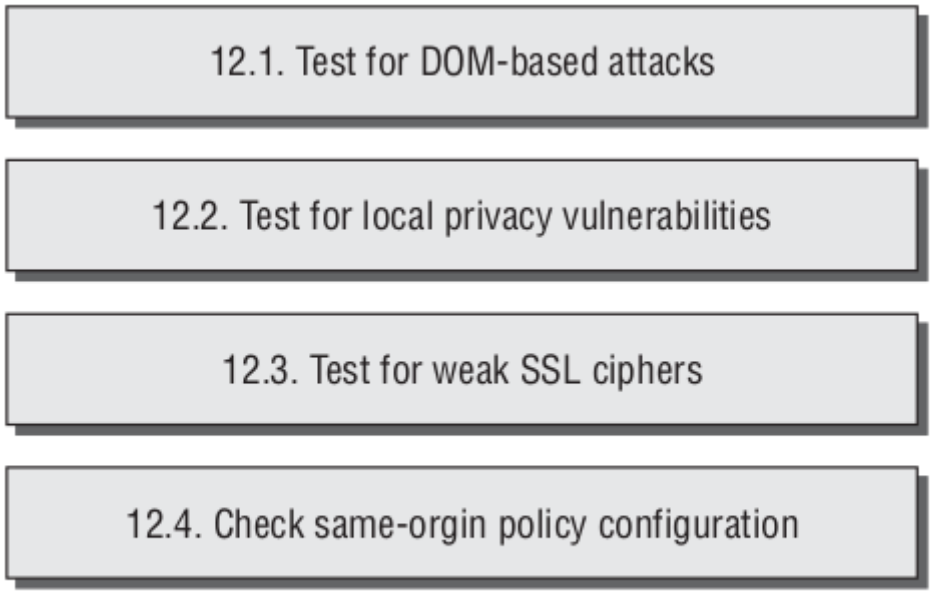

Figure 21-13: Miscellaneous checks

12.1 Check for DOM-Based Attacks

12.1.1 Выполните краткий обзор кода каждого фрагмента JavaScript, полученного из приложения. Определите любые уязвимости XSS или перенаправления, которые могут быть вызваны использованием созданного URL-адреса для внесения вредоносных данных в DOM соответствующей страницы. Включите все автономные файлы JavaScript и сценарии, содержащиеся на HTML-страницах (как статические, так и динамически создаваемые).

12.1.2 Определите все виды использования следующих API, которые могут использоваться для доступа к данным DOM, которыми можно управлять с помощью созданного URL-адреса:

document.location
document.URL
document.URLUnencoded
document.referrer
window.location

12.1.3 Отслеживать соответствующие данные с помощью кода, чтобы определить, какие действия с ним выполняются. Если данные (или их измененная форма) передаются в один из следующих API-интерфейсов, приложение может быть уязвимо для XSS:

document.write()
document.writeln()
document.body.innerHtml
eval()
window.execScript()
window.setInterval()
window.setTimeout()

12.1.4 Если данные передаются в один из следующих API-интерфейсов, приложение может быть уязвимо для атаки перенаправления:

document.location
document.URL
document.open()
window.location.href
window.navigate()
window.open()

12.2 Check for Local Privacy Vulnerabilities

12.2.1 Просмотрите журналы, созданные вашим перехватывающим прокси-сервером, чтобы идентифицировать все директивы Set-Cookie, полученные от приложения во время тестирования. Если какой-либо из них содержит атрибут expires с датой, которая наступит в будущем, файлы cookie будут храниться браузерами пользователей до этой даты. Просмотрите содержимое любых постоянных файлов cookie на предмет наличия конфиденциальных данных.

12.2.2 Если установлен постоянный файл cookie, содержащий какие-либо конфиденциальные данные, локальный злоумышленник может захватить эти данные. Даже если данные зашифрованы, злоумышленник, который их захватит, сможет повторно отправить файл cookie в приложение и получить доступ к любым данным или функциям, которые это позволяет.

12.2.3 Если доступ к каким-либо страницам приложения, содержащим конфиденциальные данные, осуществляется по протоколу HTTP, найдите любые директивы кэша в ответах сервера. Если какая-либо из следующих директив не существует (либо в заголовках HTTP, либо в метатегах HTML), соответствующая страница может быть кэширована одним или несколькими браузерами:

Expires: 0
Cache-control: no-cache
Pragma: no-cache

12.2.4 Идентифицировать любые экземпляры в приложении, в которых конфиденциальные данные передаются с помощью параметра URL. Если какие-либо случаи существуют, проверьте браузер

12.2.5 Для всех форм, которые используются для сбора конфиденциальных данных пользователя (таких как данные кредитной карты), просмотрите HTML-источник формы. Если атрибутautocomplete=off . не установлен, то в теге формы или теге для отдельного поля ввода введенные данные сохраняются в браузерах, поддерживающих автозаполнение, при условии, что пользователь не отключил эту функцию.

12.2.6 Проверьте наличие локального хранилища, специфичного для конкретной технологии.

12.2.6.2 Проверьте любое изолированное хранилище Silverlight в этом каталоге:

C:\Users\{username}\AppData\LocalLow\Microsoft\Silverlight\

12.2.6.3 Проверьте любое использование локального хранилища HTML5.

12.3 Check for Weak SSL Ciphers

12.3.1 Если приложение использует SSL для какой-либо из своих коммуникаций, используйте инструмент THCSSLCheck, чтобы перечислить поддерживаемые шифры и протоколы.

12.3.2 Если поддерживаются какие-либо слабые или устаревшие шифры и протоколы, злоумышленник, находящийся в подходящем положении, может выполнить атаку, чтобы понизить или расшифровать SSL-соединения пользователя приложения, получив доступ к его конфиденциальным данным.

12.3.3 Некоторые веб-серверы рекламируют определенные слабые шифры и протоколы как поддерживаемые, но отказываются фактически выполнять рукопожатие с их использованием, если клиент запрашивает их. Это может привести к ложным срабатываниям при использовании инструмента THCSSLCheck. Вы можете использовать браузер Opera, чтобы попытаться выполнить полное рукопожатие, используя определенные слабые протоколы, чтобы подтвердить, действительно ли они могут быть использованы для доступа к приложению.

12.4 Check Same-Origin Policy Configuration

12.4.1 Проверьте наличие /crossdomain.xml фи ле. Если приложение разрешает неограниченный
доступ (указав <allow-access-from domain=”*” />), Flash-объекты с любого другого сайта могут осуществлять двустороннее взаимодействие, используя сеансы пользователей приложения. Это позволило бы извлекать все данные и выполнять любые действия пользователя любым другим доменом.

с любого другого сайта можно осуществлять двустороннее взаимодействие, используя сеансы пользователей приложения. Это позволило бы извлекать все данные и выполнять любые действия пользователя любым другим доменом.

12.4.2 Проверьте наличие /clientaccesspolicy.xml фи ле. Аналогично Flash, если конфигурация <cross-domain-access> слишком разрешительна, другие сайты
могут выполнять двустороннее взаимодействие с оцениваемым сайтом.

12.4.3 Протестируйте обработку приложением междоменных запросов с помощью XMLHttpRequest, добавив заголовок источника, указывающий другой домен, и изучив все возвращаемые заголовки управления доступом.
Последствия для безопасности, связанные с предоставлением двустороннего доступа из любого домена или из указанных других доменов, такие же, как и те, которые описаны для междоменной политики Flash.

13 Follow Up Any Information Leakage

13.1 Во время всех ваших исследований целевого приложения следите за его ответами на сообщения об ошибках, которые могут содержать полезную информацию о причине ошибки, используемых технологиях, а также внутренней структуре и функциональности приложения.

13.2 Если вы получаете какие-либо необычные сообщения об ошибках, изучите их с помощью стандартных поисковых систем. Вы можете использовать различные расширенные функции поиска, чтобы сузить область поиска. Например:

“unable to retrieve” filetype:php

13.3 Просмотрите результаты поиска, ища как любое обсуждение сообщения об ошибке, так и любые другие веб-сайты, на которых появилось такое же сообщение. Другие приложения могут выдавать то же сообщение в более подробном контексте, позволяя вам лучше понять, какие условия приводят к ошибке. Используйте кэш поисковой системы для извлечения примеров сообщений об ошибках, которые больше не отображаются в реальном приложении.

13.4 Используйте поиск кода Google, чтобы найти любой общедоступный код, который может быть ответственен за конкретное сообщение об ошибке. Найдите фрагменты сообщений об ошибках, которые могут быть жестко закодированы в исходном коде приложения.
Вы также можете использовать различные функции расширенного поиска, чтобы указать язык кода и другие сведения, если они известны. Например:

unable\ to\ retrieve lang:php package:mail

13.5 If you receive error messages with stack traces containing the names of library components and third-party code, search for these names in both types of search engine.