СОДЕРЖАНИЕ

Об авторе

Джеймс Форшоу – известный специалист по компьютерной безопасности из команды Google Project Zero с более чем десятилетним опытом анализа и эксплуатации уязвимостей в сетевых протоколах прикладного уровня. Его навыки варьируются от взлома игровых консолей до выявления сложных проблем проектирования в операционных системах, особенно в Microsoft Windows, что принесло ему награду в размере 100 000 долларов и позволило занять первое место в списке Microsoft Security Response Center (MSRC). Он создал Canape, инструмент для анализа сетевых протоколов, который он разработал, будучи специалистом с многолетним опытом работы в этой области, а также был приглашен принять участие в глобальных конференциях по безопасности, таких как BlackHat, CanSecWest и Chaos Computer Congress, где он представил свои новаторские исследования.

О рецензенте

С первых дней существования Commodore PET и VIC-20 технологии были постоянным спутником (а иногда и навязчивой идеей!) Клиффа Янзена. Клифф обнаружил в себе страсть к этой профессии, когда в 2008 г. после десяти лет работы в ИТ перешел работать в сферу информационной безопасности. С тех пор ему посчастливилось сотрудничать с лучшими специалистами этой отрасли и учиться у них, включая мистера Форшоу и сотрудников из No Starch во время создания этой книги. Он работает консультантом по вопросам безопасности, занимаясь всем – от анализа политик до тестов на проникновение. Ему повезло, что у него есть карьера, которая вместе с тем является его любимым хобби, и жена, которая его поддерживает.

ПРЕДИСЛОВИЕ

Когда я впервые познакомилась с Джеймсом Форшоу, я занималась тем, что в 2007 г. журнал Popular Science описал как одну из десяти худших профессий Microsoft Security Grunt. Это ярлык, который журнал использовал для всех, кто работал в Microsoft Security Response Center (MSRC). Это позиционировало нашу работу хуже, чем «исследователь китовых фекалий», но немного лучше, чем «вазэктомист, лечащий слонов» в этом списке профессий (настолько известном среди тех из нас, кто страдал в Редмонде, штат Вашингтон, что мы сделали футболки), так это непрекращающийся шквал отчетов об ошибках в системе безопасности в продуктах Microsoft.

Именно здесь, в MSRC, Джеймс, с его острым и творческим взглядом на необычное и упускаемое из виду, впервые привлек мое внимание в качестве стратега по безопасности. Джеймс был автором некоторых самых интересных отчетов об ошибках безопасности. Это был немалый подвиг, учитывая, что MSRC получал более 200 000 отчетов об ошибках безопасности в год от исследователей в области ИБ. Джеймс обнаруживал не только простые ошибки – в платформе .NET Framework он нашел проблемы на уровне архитектуры. Хотя их было труднее исправить с помощью простого патча, они были гораздо более ценными для Microsoft и ее клиентов.

Перенесемся к первой программе Bug Bounty от корпорации Microsoft, которую я создала в компании в июне 2013 года. Первоначально у нас было три программы – программы, которые обещали платить исследователям безопасности, таким как Джеймс, наличными в обмен на сообщение о наиболее серьезных ошибках в Microsoft. Я знала: для того чтобы эти программы доказали свою эффективность, нужно было исправлять серъезные ошибки.

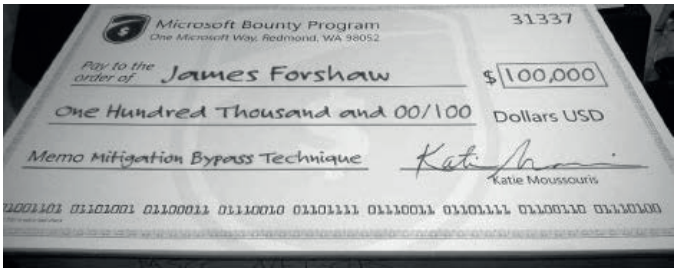
Если бы мы создали ее, не было никакой гарантии, что к нам приедут специалисты по поиску ошибок. Мы знали, что соревнуемся за право стать одними из самых высококвалифицированных специалистов по поиску ошибок в мире. Было доступно множество других денежных вознаграждений, но не все вознаграждения назначались именно за защиту. У отдельных государств и преступников был хорошо развитый рынок ошибок и эксплойтов, и Microsoft полагалась на специалистов, которые уже предоставляли по 200 000 отчетов об ошибках в год бесплатно. Награды должны были привлечь внимание этих дружелюбных, альтруистических охотников за ошибками, в устранении которых Microsoft нуждалась больше всего.

Поэтому я, конечно же, позвонила Джеймсу и другим, потому что рассчитывала, что они займутся этим. Мы, специалисты по безопасности из MSRC, действительно хотели получить уязвимости для бета-версии Internet Explorer (IE) 11, и нам нужно было нечто, за что ни один поставщик программного обеспечения никогда не пытался назначить вознаграждение: мы хотели узнать о новых техниках эксплуатации. Эта награда была известна как Mitigation Bypass Bounty и в то время составляла 100 000 долларов. Я помню, как сидела с Джеймсом за кружкой пива в Лондоне, пытаясь увлечь его поиском ошибок в IE, когда он объяснил, что раньше никогда особо не интересовался безопасностью браузера, и предупредил меня, чтобы я не ожидала от него слишком многого.

Тем не менее Джеймс создал четыре уникальных варианта побега из песочницы бета-версии IE 11 Четыре. Они находились в тех областях кода IE, которые наши внутренние команды и добросовестные внешние специалисты по тестированию на проникновение пропустили. Побеги из песочницы были необходимы для более надежной эксплуатации других ошибок. Джеймс получил награды за все четыре ошибки, за которые заплатила сама команда IE, плюс дополнительный бонус в размере 5000 долларов из моего бюджета. Оглядываясь назад, я, наверное, должна была дать ему лишние 50 000 долларов. Потому что это супер! Неплохо для охотника за ошибками, который никогда раньше не интересовался безопасностью веб-браузеров.

Всего несколько месяцев спустя я позвонила Джеймсу, находясь рядом с кафетерием Microsoft прохладным осенним днем, совершенно запыхавшись, чтобы сказать ему, что он только что вошел в историю: его заявка на участие в одной из других программ Bu Bount, Microsoft – Mitigation Bypass Bounty, на сумму 100 000 долларов, была принята. Джеймс Форшоу нашел новый уникальный способ обойти все средства защиты платформы, используя недостатки архитектурного уровня в последней версии операционной системы, и выиграл самую первую награду в размере 100 000 долларов от Microsoft.

Во время того телефонного разговора, насколько я помню, он сказал, что представил, как я вручаю ему до смешного огромный чек на сцене во время внутренней конференции Microsoft BlueHat. После этого звонка я отправила в отдел маркетинга записку, и в одно мгновение «Джеймс и гигантский чек» навсегда вошли в историю Microsoft и интернета.



Я уверена, что читатели узнают на страницах этой книги о несравненной гениальности Джеймса – той же гениальности, которую я увидела в одном или четырех отчетах об ошибках много лет назад. Существует очень мало исследователей безопасности, которые могут найти ошибки в одной передовой технологии, и еще меньше тех, кто может последовательно находить их в нескольких технологиях. Кроме того, есть такие люди, как Джеймс Форшоу, которые могут сосредоточиться на более глубоких архитектурных проблемах с точностью хирурга. Я надеюсь, что те, кто читает эту книгу и будет читать все последующие книги Джеймса, воспримут ее как практическое руководство, которое поможет им пробудить ту же гениальность и творческий потенциал в своей работе.

На собрании по Bug Bounty в Microsoft, когда члены команды IE качали головами, гадая, как они могли пропустить эти ошибки, о которых сообщил Джеймс, я просто сказала: «Джеймс видит Женщину в Красном платье, а также код, который ее визуализировал, в Матрице». Все, кто сидел за столом, приняли это объяснение того, как работал у Джеймса ум. Ему все было по плечу; и, изучая его работы, если вы не грешите предвзятостью, то тоже сможете стать такими же.

Всем искателям ошибок в мире – вот ваша планка, и она высока. Несмотря на неисчислимое количество специалистов по безопасности, пусть все ваши отчеты об ошибках будут такими же интересными и ценными, как и те, что предоставлены единственным и неповторимым Джеймсом Форшоу.

Кэти Муссурис,
основатель и генеральный директор Luta Security
Октябрь 2017 г.

ВВЕДЕНИЕ

Когда впервые была представлена технология, позволявшая устройствам подключаться к сети, она была эксклюзивной для крупных компаний и правительств. Сегодня большинство людей носят с собой полностью подключенные к сети вычислительные устройства, а с развитием интернета вещей (IoT) можно добавить в этот взаимосвязанный мир такие устройства, как холодильник и домашняя система безопасности. Безопасность этих устройств становится все более важной. Хотя вы, возможно, и не слишком беспокоитесь о том, что кто-то раскроет подробности того, сколько йогуртов вы покупаете, если ваш смартфон будет взломан в той же сети, что и ваш холодильник, вы можете лишиться всей своей личной и финансовой информации – она будет доступна злоумышленнику.

Эта книга называется «*Атака сетей на уровне протоколов*», потому что для обнаружения уязвимостей в устройстве, подключенном к сети, необходимо принять образ мыслей злоумышленника, который хочет использовать эти слабые места. Сетевые протоколы обмениваются данными с другими устройствами в сети, и поскольку эти протоколы должны быть доступны в открытой сети и нечасто подвергаются такому же уровню проверки, как другие компоненты устройства, они являются очевидной целью атаки.

Зачем читать эту книгу?

Во многих книгах обсуждается захват сетевого трафика с целью диагностики и базового анализа сети, но в них не говорится об аспектах безопасности протоколов. Эту книгу от других отличает тот факт, что она фокусируется на анализе пользовательских протоколов для поиска уязвимостей в системе безопасности.

Она для тех, кто интересуется анализом и атаками сетей на уровне протоколов, но не знает, с чего начать. Здесь вы познакомитесь с методами обучения захвату сетевого трафика, выполнением анализа протоколов, а также обнаружением и эксплуатацией уязвимостей в системе безопасности. В книге представлена справочная информация о сетях и сетевой безопасности, а также практические примеры протоколов для анализа.

Хотите ли вы атаковать сеть, чтобы сообщить об уязвимостях поставщику приложения, или просто хотите узнать, как ваше IoT-устройство обменивается данными, вы найдете здесь интересующие вас темы.

Что есть в этой книге?

Эта книга состоит из теоретических и практических глав. Для практических глав я разработал и сделал доступной сетевую библиотеку Canape Core, которую можно использовать для создания собственных инструментов для анализа и эксплуатации уязвимостей протоколов. Я также представил образец сетевого приложения под названием SuperFunkyChat, которое реализует протокол чата между пользователями. Следуя обсуждениям в главах, вы можете использовать это приложение, чтобы изучить навыки анализа протоколов и атаковать образцы сетевых протоколов. Ниже приводится краткое описание каждой главы.

Глава 1. Знакомство с основами

В этой главе описываются основы компьютерных сетей и особый акцент делается на стеке TCP/IP, который составляет основу сетевых протоколов прикладного уровня. В следующих главах предполагается, что вы хорошо разбираетесь в основах построения сетей. В этой главе также представлен подход, который я использую для моделирования протоколов приложений. Эта модель разбивает протокол приложения на гибкие уровни и абстрагирует сложные технические детали, позволяя вам сосредоточиться на отдельных частях протокола, который вы анализируете.

Глава 2. Перехват трафика

В этой главе представлены концепции пассивного и активного перехватов сетевого трафика, и это первая глава, в которой сетевые библиотеки Canape Core используются для практических задач.

Глава 3. Структуры сетевых протоколов

В этой главе содержится подробная информация о внутренних структурах, которые распространены в сетевых протоколах, таких как представление чисел или удобочитаемый текст. Когда вы анализируете перехваченный сетевой трафик, то можете использовать эти знания, чтобы быстро определить распространенные структуры, ускоряя тем самым анализ.

Глава 4. Расширенный перехват трафика

В этой главе исследуется ряд более продвинутых методов перехвата, которые дополняют примеры из главы 2. Методы расширенного перехвата включают в себя настройку механизма NAT для перенаправления интересующего вас трафика и спуфинга протокола APR.

Глава 5. Анализ на практике

В этой главе представлены методы анализа перехваченного сетевого трафика с использованием пассивных и активных методов перехвата, описанных в главе 2. Здесь мы используем приложение SuperFunkyChat для генерации трафика.

Глава 6. Обратная разработка

В этой главе описываются методы обратного проектирования программ, подключенных к сети. Обратная разработка позволяет анализировать протокол без необходимости захватывать трафик. Эти методы также помогают определить, как реализовано пользовательское шифрование или запутывание кода, чтобы можно было лучше анализировать перехваченный трафик.

Глава 7. Безопасность сетевого протокола

В этой главе представлена справочная информация о методах и криптографических алгоритмах, используемых для защиты сетевых протоколов. Защита содержимого сетевого трафика от раскрытия или подделки при его передаче по общедоступным сетям имеет первостепенное значение для безопасности сетевых протоколов.

Глава 8. Реализация сетевого протокола

В этой главе объясняются методы реализации сетевого протокола в вашем собственном коде, чтобы вы могли протестировать его поведение и найти слабые места.

Глава 9. Основные причины уязвимостей

В этой главе описаны распространенные уязвимости, с которыми вы можете столкнуться в сетевом протоколе. Когда вы поймете коренные причины уязвимостей, вам будет легче идентифицировать их во время анализа.

Глава 10. Поиск и эксплуатация уязвимостей

В этой главе описываются процессы поиска уязвимостей на базе основных причин, указанных в главе 9, и демонстрируется ряд способов их эксплуатации, включая разработку собственного шелл-кода и обход средств защиты от эксплойтов посредством возвратно-ориентированного программирования.

Приложение. Набор инструментов для анализа сетевых протоколов

В этом приложении вы найдете описания инструментов, которые я обычно использую при выполнении анализа сетевых протоколов. Многие инструменты также кратко описаны в основной части книги.

Как пользоваться этой книгой

Если вы хотите освежить в памяти основы, прочтите сначала главу 1. Когда вы ознакомитесь с основами, переходите к главам 2, 3 и 5, чтобы получить практический опыт в перехвате сетевого трафика и изучить процесс анализа сетевых протоколов.

Зная принципы перехвата сетевого трафика и его анализа, можно перейти к главам с 7 по 10 для получения практической информации о том, как находить и эксплуатировать уязвимости в этих протоколах.

В главах 4 и 6 содержится более подробная информация о дополнительных методах перехвата и обратном проектировании приложений, поэтому если хотите, то можете прочитать их после того, как ознакомитесь с другими главами.

Для выполнения практических примеров вам потребуется установить .NET Core (*https://www.microsoft.com/net/core/*), кросс-платформенную версию среды выполнения .NET от корпорации Microsoft, которая работает в Windows, Linux и macOS. Затем вы можете скачать выпуски для Canape Core на странице *https://github.com/tyranid/CANAPE.Core/releases/* и SuperFunkyChat на странице *https://github.com/tyranid/ExampleChatApplication/releases/*. Они используют .NET Core в качестве среды выполнения. Ссылки на каждый сайт доступны в ресурсах книги на странице *https://www.nostarch.com/networkprotocols/*.

Чтобы выполнить пример сценария Canape Core, необходимо использовать приложение *CANAPE.Cli*, которое будет находиться в пакете релиза, загруженном из репозитория Canape Core на Github. Выполните следующий код в командной строке, заменив *script.csx* именем сценария, который вы хотите выполнить.

```
dotnet exec CANAPE.Cli.dll script.csx
```

Все примеры листингов для практических глав, а также захват пакетов доступны на странице книги по адресу *https://www.nostarch.com/networkprotocols/*.

Лучше всего загрузить эти примеры перед тем, как вы приступите, чтобы можно было следовать за главами без необходимости вводить большой объем исходного кода вручную.

Как связаться со мной

Мне всегда интересно получать как положительные, так и отрицательные отзывы о моей работе, и эта книга не исключение. Вы можете написать мне по адресу *attacking.network.protocols@gmail.com*, а также подписаться на меня в Twitter – *@tiraniddo* – или подписаться на мой блог на странице *https://tyranidslair.blogspot.com/*, где я публикую некоторые из своих последних передовых исследований в области безопасности.

1

ЗНАКОМСТВО С ОСНОВАМИ

Чтобы атаковать сеть на уровне протоколов, необходимо знать основы. Чем лучше вы понимаете, как устроены и функционируют обычные сети, тем проще будет применить эти знания для перехвата трафика, его анализа и эксплуатации уязвимостей.

В этой главе я познакомлю вас с основными концепциями, с которыми вы сталкиваетесь каждый день при анализе сетевых протоколов. А также заложу основы для их понимания, что упростит поиск ранее неизвестных проблем безопасности во время анализа.

Сетевая архитектура и протоколы

Начнем с обзора базовой терминологии и зададим себе главный вопрос: что такое сеть? *Сеть* – это два или более компьютеров, соединенных между собой для обмена информацией. Обычно каждое подключенное устройство называют *узлом*, чтобы это описание можно было применить к более широкому кругу устройств. На рис. 1.1 приведен очень простой пример.

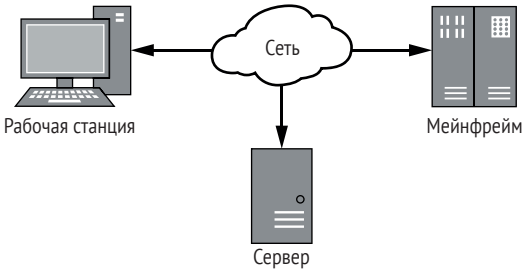


Рис. 1.1. Простая сеть из трех узлов

Здесь показаны три узла, подключенных к общей сети. У каждого узла может быть своя операционная система или оборудование. Но пока каждый узел следует набору правил, или *сетевому протоколу*, он может взаимодействовать с другими узлами в сети. Чтобы обмен данными осуществлялся надлежащим образом, все узлы в сети должны понимать один и тот же сетевой протокол.

Сетевой протокол выполняет множество функций, включая одну или несколько из перечисленных ниже:

Поддержание состояния сеанса – обычно протоколы реализуют механизмы для создания новых подключений и завершения уже существующих.

Идентификация узлов посредством адресации – данные должны передаваться на правильный узел. Некоторые протоколы реализуют механизм адресации для идентификации конкретных узлов или групп узлов.

Управление потоком – объем данных, передаваемых по сети, ограничен. Протоколы могут реализовывать способы управления потоком данных для увеличения пропускной способности и уменьшения задержки.

Гарантия порядка передачи данных – многие сети не гарантируют, что порядок отправки данных будет соответствовать порядку, в котором они будут получены. Протокол может изменить порядок данных, чтобы убедиться, что они будут доставлены правильно.

Обнаружение и исправление ошибок – многие сети не являются надежными на 100 %, и данные могут быть повреждены. Важно обнаружить повреждение и, в идеале, исправить его.

Форматирование и кодирование данных – данные не всегда находятся в формате, подходящем для передачи их по сети. Протокол может указывать способы кодирования данных, например кодирование текста на английском языке в двоичные значения.

Набор интернет-протоколов

TCP/IP – это протокол де-факто, используемый современными сетями. Хотя можно рассматривать TCP/IP как единый протокол, на са-

мом деле это комбинация двух протоколов: *протокола управления передачей* (TCP) и *интернет-протокола* (IP). Оба они являются частью набора интернет-протоколов (IPS), концептуальной модели того, как сетевые протоколы отправляют сетевой трафик через интернет. Таким образом, обмен данными можно разделить на четыре уровня, как показано на рис. 1.2.

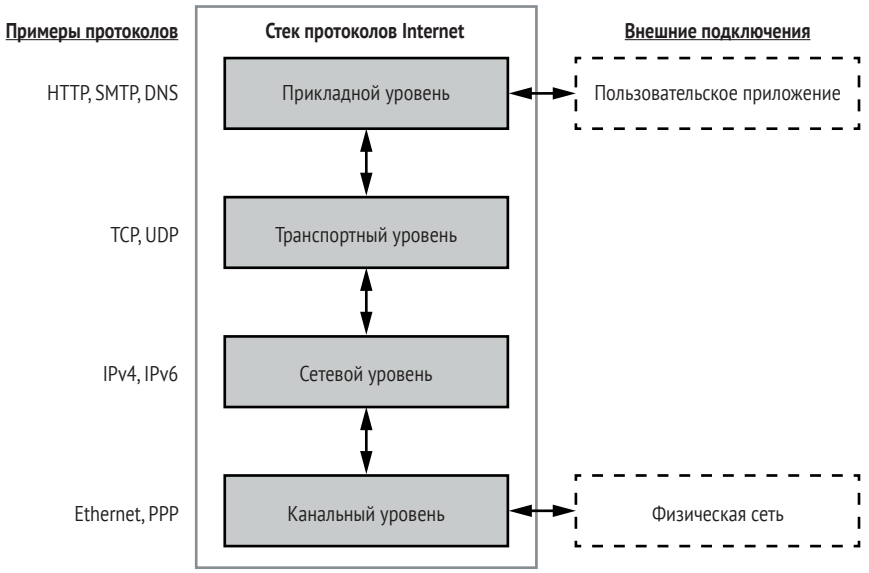


Рис. 1.2. Уровни набора интернет-протоколов

Эти четыре уровня образуют *стек протоколов*. В следующем списке приводится объяснение каждого из этих уровней.

Канальный уровень (**уровень 1**) – является самым низким уровнем и описывает физические механизмы, используемые для передачи информации между узлами в локальной сети. Хорошо известные примеры включают Ethernet (проводной и беспроводной) и протокол Point-to-Point (PPP).

Сетевой уровень (**уровень 2**) – предоставляет механизмы для определения пути передачи данных. В отличие от уровня 1, узлы не обязательно должны находиться в локальной сети. Этот уровень содержит IP; в современных сетях фактический используемый протокол может быть либо версией 4 (IPv4), либо версией 6 (IPv6).

Транспортный уровень (**уровень 3**) – отвечает за соединения между клиентами и серверами, иногда обеспечивая правильный порядок пакетов и предоставляя мультиплексирование сервисов. Мультиплексирование сервисов позволяет одному узлу поддерживать несколько различных сервисов, присваивая каждому сервису разные номера; этот номер называется *портом*. На этом уровне работают протоколы TCP и UDP.

Прикладной уровень (уровень 4) – содержит сетевые протоколы, такие как *протокол передачи гипертекста* (*HTTP*), который передает содержимое веб-страниц; *простой протокол передачи почты* (*SMTP*), передающий электронную почту; и *протокол системы доменных имен* (*DNS*), который преобразует имя в узел в сети. В этой книге мы сосредоточимся главным образом на этом уровне.

Каждый уровень взаимодействует только с уровнем, который располагается выше и ниже него, но должны осуществляться и внешние взаимодействия со стеком. На рис. 1.2 показаны два внешних соединения. Канальный уровень взаимодействует с физическим сетевым соединением, передавая данные в физической среде, например электрические импульсы или импульсы света. Прикладной уровень взаимодействует с пользовательским приложением: *приложение* представляет собой набор связанных функций, которые предоставляют сервис пользователю. На рис. 1.3 показан пример приложения, обрабатывающего электронную почту. Сервис, предоставляемый почтовым приложением, – это отправка и получение сообщений по сети.

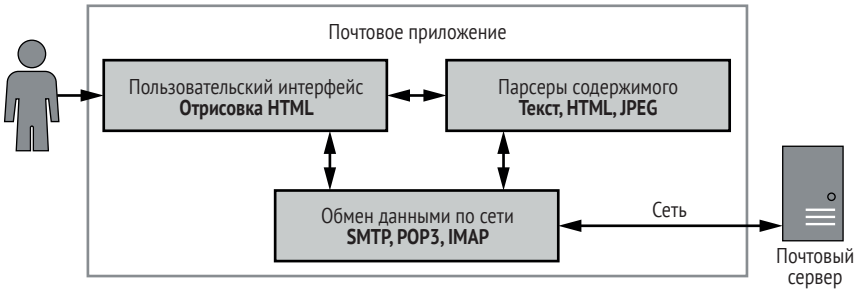


Рис. 1.3. Пример почтового приложения

Обычно приложения содержат следующие компоненты:

Обмен данными по сети – этот компонент обменивается данными по сети и обрабатывает входящие и исходящие данные. Для почтового приложения обмен данными по сети, скорее всего, является стандартным протоколом, таким как SMTP или POP3.

Парсеры содержимого – данные, передаваемые по сети, обычно включают в себя содержимое, которое необходимо извлечь и обработать. Это могут быть текстовые данные, такие как тело электронного письма, или изображения или видео.

Пользовательский интерфейс – позволяет пользователю просматривать полученные электронные письма и создавать новые письма для передачи. В почтовом приложении пользовательский интерфейс может отображать электронные письма с использованием HTML в веб-браузере.

Обратите внимание, что пользователь, взаимодействующий с пользовательским интерфейсом, не обязательно должен быть человеком.

Это может быть другое приложение, автоматизирующее отправку и получение писем посредством инструмента командной строки.

Инкапсуляция данных

Каждый уровень в IPS построен на уровне, находящемся ниже, и каждый уровень может инкапсулировать данные, полученные от вышестоящего уровня, чтобы те могли перемещаться между уровнями. Данные, передаваемые каждым уровнем, называются *блоком данных протокола* (PDU).

Заголовки, концевики и адреса

На каждом уровне блок данных протокола содержит передаваемые полезные данные. Обычно к полезным данным добавляется *заголовок*, содержащий необходимую информацию для этих передаваемых данных, такую как *адреса* узлов источника и назначения в сети. Иногда у PDU также есть *концевик*, который добавляется к полезным данным и содержит значения, необходимые для обеспечения правильной передачи, например информацию для проверки ошибок. На рис. 1.4 показано, как блоки данных протокола размещаются в IPS.

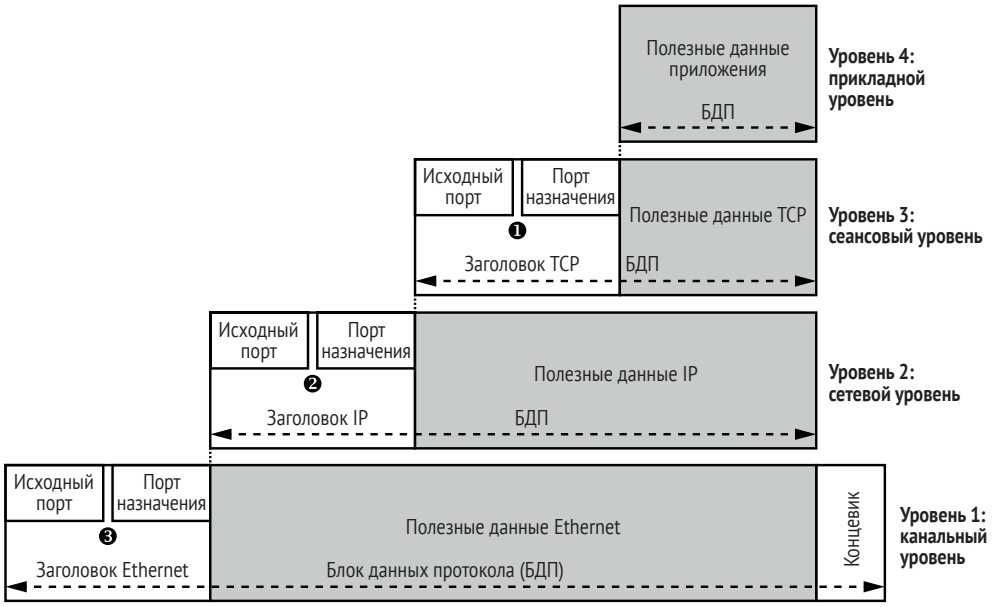


Рис. 1.4. Инкапсуляция данных IPS

Заголовок TCP содержит номер исходного порта и порта назначения ❶. Благодаря этим номерам у одного узла может быть несколько уникальных сетевых соединений. Номера портов для протокола TCP (и UDP) находятся в диапазоне от 0 до 65 535.

Большинство номеров портов присваиваются новым соединениям по мере необходимости, но есть и особые случаи, например порт 80 для HTTP. (В большинстве Unix-подобных операционных систем текущий список присвоенных номеров портов можно найти в файле */etc/services*.) Полезные данные и заголовок TCP обычно называются *сегментом*, тогда как полезные данные и заголовок UDP – *дейтаграммой*.

Протокол IP использует адрес источника и адрес назначения ❷. *Адрес назначения* позволяет отправлять данные на определенный узел в сети. *Адрес источника* позволяет получателю данных узнать, какой узел отправил данные, и дает возможность получателю ответить отправителю.

IPv4 использует 32-битные адреса, которые обычно записываются в виде четырех чисел, разделенных точками, например 192.168.10.1. IPv6 использует 128-битные адреса, потому что 32-битных адресов недостаточно для количества узлов в современных сетях. Адреса IPv6 обычно записываются в виде шестнадцатеричных чисел, разделенных двоеточиями, например fe80:0000:0000:0000:897b:581e:44b0:2057. Длинные строки с 0000 можно записывать с использованием знака двойного двоеточия. Например, предыдущий IPv6-адрес также можно записать как fe80::897b:581e:44b0:2057. Полезные данные и заголовок протокола IP обычно называются *пакетом*.

Ethernet также содержит адреса источника и назначения ❸. Ethernet использует 64-битное значение, которое называют *MAC-адрес*. Как правило, MAC-адрес устанавливается при изготовлении адаптера Ethernet. Обычно эти адреса записываются в виде серии шестнадцатеричных чисел, разделенных дефисом или двоеточием, например 0A-00-27-00-00-0E. Полезные данные Ethernet, включая заголовок и концевик, обычно называются *кадром*.

Передача данных

Вкратце рассмотрим, как данные передаются от одного узла к другому с помощью модели инкапсуляции данных IPS. На рис. 1.5 показана простая сеть Ethernet с тремя узлами.

В данном примере узел с IP-адресом 192.1.1.101 ❶ хочет отправить данные по протоколу IP на узел ❷ с IP-адресом 192.1.1.50. (*Коммутатор* ❸ пересылает кадры Ethernet между всеми узлами в сети. Коммутатору не нужен IP-адрес, потому что он работает только на канальном уровне.) Вот что происходит при передаче данных между двумя узлами.

1. Узел сетевого стека операционной системы ❶ инкапсулирует данные прикладного и транспортного уровней и создает IP-пакет с адресом отправителя 192.1.1.101 и адресом назначения 192.1.1.50.
2. На данном этапе операционная система может инкапсулировать IP-данные как кадр Ethernet, но она может не знать MAC-адрес

целевого узла. Она может запросить MAC-адрес для определенного IP-адреса с помощью протокола ARP, который отправляет запрос всем узлам в сети, чтобы найти MAC-адрес для IP-адреса назначения.

3. Как только узел ❶ получает ответ от ARP, он может построить кадр, задав в качестве адреса отправителя локальный MAC-адрес 00-11-22-33-44-55 и адрес назначения 66-77-88-99-AA-BB. Новый кадр передается по сети и принимается коммутатором ❸.
4. Коммутатор пересылает кадр на узел назначения, который распаковывает IP-пакет и проверяет соответствие IP-адреса назначения. Затем полезные данные IP извлекаются и передаются вверх по стеку, чтобы их получило приложение.

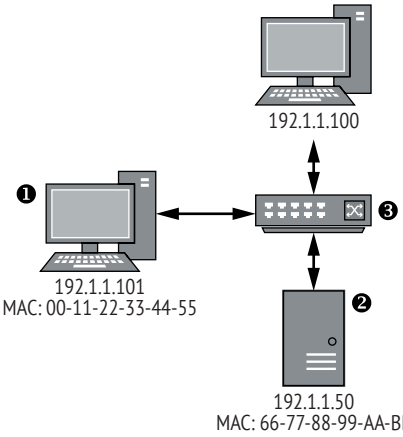


Рис. 1.5. Простая сеть Ethernet

Сетевая маршрутизация

Ethernet требует, чтобы все узлы были подключены напрямую к одной локальной сети. Данное требование является серьезным ограничением для по-настоящему глобальной сети, поскольку физически соединять узлы друг с другом не представляется возможным. Вместо того чтобы требовать прямого подключения всех узлов, адреса отправителя и получателя позволяют *маршрутизировать* данные по разным сетям до тех пор, пока те не достигнут нужного узла назначения, как показано на рис. 1.6.

На рисунке видны две сети Ethernet, каждая из которых имеет отдельные диапазоны IP-адресов. Следующее описание объясняет, как IP использует эту модель для отправки данных от узла ❶ в сети 1 к узлу ❷ в сети 2.

1. Узел сетевого стека операционной системы ❶ инкапсулирует данные прикладного и транспортного уровней и создает IP-пакет с адресом отправителя 192.1.1.101 и адресом получателя 200.0.1.50.

2. Сетевому стеку необходимо отправить кадр Ethernet, но поскольку IP-адрес назначения не существует ни в одной сети Ethernet, к которой подключен узел, стек обращается к *таблице маршрутизации* операционной системы. В этом примере таблица маршрутизации содержит запись для IP-адреса 200.0.1.50. Запись указывает на то, что маршрутизатор ❸ на IP-адресе 192.1.1.1 знает, как добраться до этого адреса назначения.
3. Операционная система использует протокол ARP для поиска MAC-адреса маршрутизатора по адресу 192.1.1.1, а исходный IP-пакет инкапсулируется в кадр Ethernet с этим MAC-адресом.
4. Маршрутизатор получает кадр Ethernet и распаковывает IP-пакет. Когда маршрутизатор проверяет IP-адрес назначения, он определяет, что IP-пакет предназначен не для маршрутизатора, а для другого узла в другой подключенной сети. Маршрутизатор ищет MAC-адрес 200.0.1.50, инкапсулирует исходный IP-пакет в новый кадр Ethernet и отправляет его в сеть ❷.
5. Узел назначения получает кадр Ethernet, распаковывает IP-пакет и обрабатывает его содержимое.

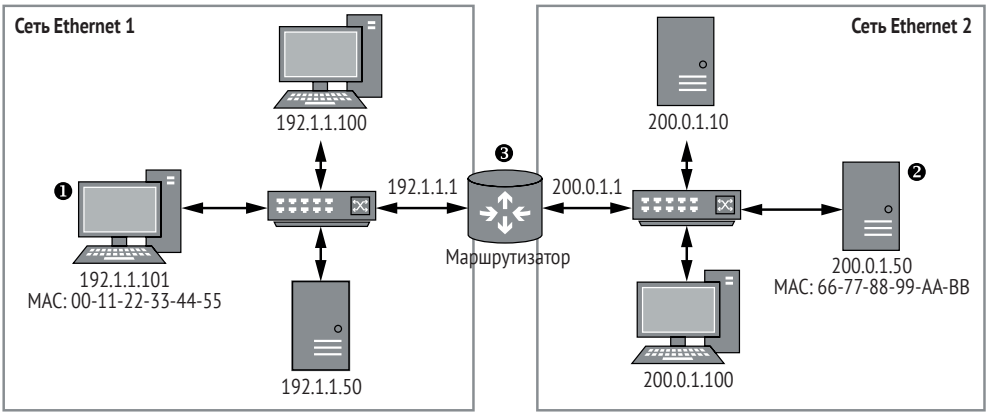


Рис. 1.6. Пример маршрутизируемой сети, соединяющей две сети Ethernet

Данный процесс маршрутизации может повторяться несколько раз. Например, если маршрутизатор не был бы подключен к сети, содержащей узел 200.0.1.50 напрямую, он сверился бы со своей таблицей маршрутизации и определил бы следующий маршрутизатор, которому он мог бы отправить IP-пакет.

Очевидно, что для каждого узла сети было бы непрактично выяснять, как добраться до другого узла в интернете. Если для пункта назначения нет явной записи маршрутизации, операционная система предоставляет запись в таблице маршрутизации по умолчанию, называемую *шлюзом по умолчанию*. Она содержит IP-адрес маршрутизатора, который может пересылать IP-пакеты по назначению.

Моя модель для анализа сетевых протоколов

IPS описывает, как работает обмен данными по сети; однако для анализа бóльшая часть этой модели не актуальна. Проще использовать мою модель, чтобы понять, как ведет себя сетевой протокол прикладного уровня. Эта модель содержит три уровня, как показано на рис. 1.7, где видно, как я буду анализировать HTTP-запрос.

Вот три уровня моей модели:

- **уровень содержимого** – обеспечивает смысл того, что передается. Как видно на рис. 1.7, смысл состоит в том, чтобы выполнить запрос файла *image.jpg*. по протоколу HTTP;
- **уровень кодирования** – предоставляет правила, определяющие, как вы представляете содержимое. В данном примере HTTP-запрос кодируется как запрос по протоколу HTTP с использованием метода GET, который указывает файл, который нужно получить;
- **транспортный уровень** – предоставляет правила для управления передачей данных между узлами. В нашем примере запрос по протоколу HTTP с использованием метода GET отправляется через TCP/IP-соединение на порт 80 на удаленном узле.

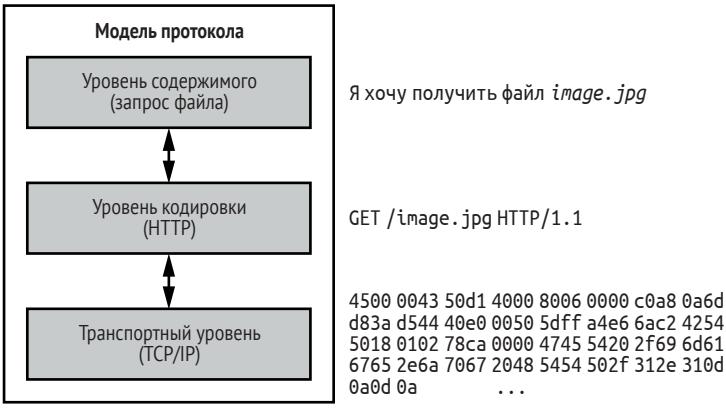


Рис. 1.7. Моя концептуальная модель протокола

Такое разделение модели снижает сложность работы с протоколами прикладного уровня, поскольку позволяет отфильтровывать те детали сетевого протокола, которые не имеют значения. Например, поскольку нам все равно, как данные TCP/IP отправляются на удаленный узел (то, что они каким-то образом туда попадают, мы считаем само собой разумеющимся), мы просто рассматриваем их как данные, передающиеся в двоичном режиме, который просто работает.

Чтобы понять, почему подобная модель протокола полезна, рассмотрим такой пример: представьте, что вы проверяете сетевой тра-

фик вредоносного ПО. Вы обнаруживаете, что вредоносная программа использует протокол HTTP для получения команд от оператора через сервер. Например, оператор может попросить вредоносную программу перечислить все файлы на жестком диске зараженного компьютера. Список файлов можно отправить обратно на сервер, после чего оператор может запросить загрузку определенного файла.

Если проанализировать протокол с точки зрения того, как оператор будет взаимодействовать с вредоносным ПО, например запрашивая файл для загрузки, новый протокол разбивается на уровни, показанные на рис. 1.8.

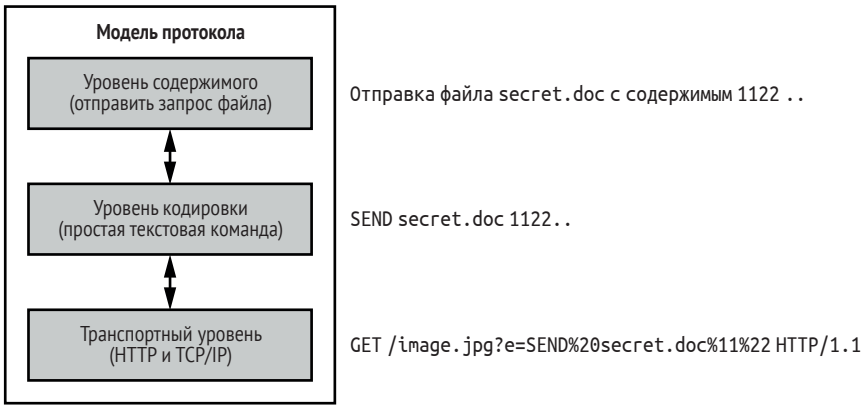


Рис. 1.8. Концептуальная модель протокола вредоносного ПО, использующего протокол HTTP

В следующем списке приводится объяснение каждого уровня новой модели протокола:

- **уровень содержимого** – вредоносное приложение отправляет украденный файл *secret.doc* на сервер;
- **уровень кодирования** – кодировка команды для отправки украденного файла представляет собой простую текстовую строку с командой SEND, за которой следует имя и данные файла;
- **транспортный уровень** – протокол использует параметр HTTP-запроса для передачи команды. Он использует стандартный механизм процентного кодирования, чтобы сделать этот запрос допустимым.

Обратите внимание, что в этом примере мы не рассматриваем отправку HTTP-запроса через TCP/IP; мы объединили кодирование и транспортный уровень на рис. 1.7 в транспортный уровень, показанный на рис. 1.8. Хотя вредоносное ПО по-прежнему использует протоколы более низкого уровня, такие как TCP/IP, эти протоколы не важны при анализе команды вредоносной программы на отправку файла. Причина, по которой это не важно, заключается в том, что

можно рассматривать отправку HTTP-запроса через TCP/IP как единый транспортный уровень, который просто работает, и сосредоточиться конкретно на уникальных командах вредоносного ПО.

Сузив рамки до уровней протокола, которые нужны нам для анализа, мы избегаем лишней работы и концентрируемся на уникальных аспектах протокола. С другой стороны, если бы мы проанализировали этот протокол, используя уровни, изображенные на рис. 1.7, то можно было бы предположить, что вредоносная программа просто запрашивает файл *image.jpg*, потому что казалось бы, что это все, что делает HTTP-запрос.

Заключительное слово

В этой главе был представлен краткий обзор основ построения сетей. Мы обсудили IPS, включая протоколы, с которыми вы столкнетесь в реальных сетях, и увидели, как данные передаются между узлами в локальной сети, а также в удаленных сетях посредством маршрутизации. Кроме того, я описал, как рассматривать сетевые протоколы прикладного уровня, чтобы вам было проще сосредоточиться на уникальных функциях протокола и тем самым ускорить его анализ.

В главе 2 мы будем использовать эти основы, что поможет нам при перехвате сетевого трафика, который мы будем анализировать. Цель перехвата сетевого трафика – получить доступ к данным, необходимым для запуска процесса анализа, определить, какие протоколы используются, и в конечном итоге обнаружить проблемы безопасности, которые можно применять для компрометации приложений, использующих эти протоколы.