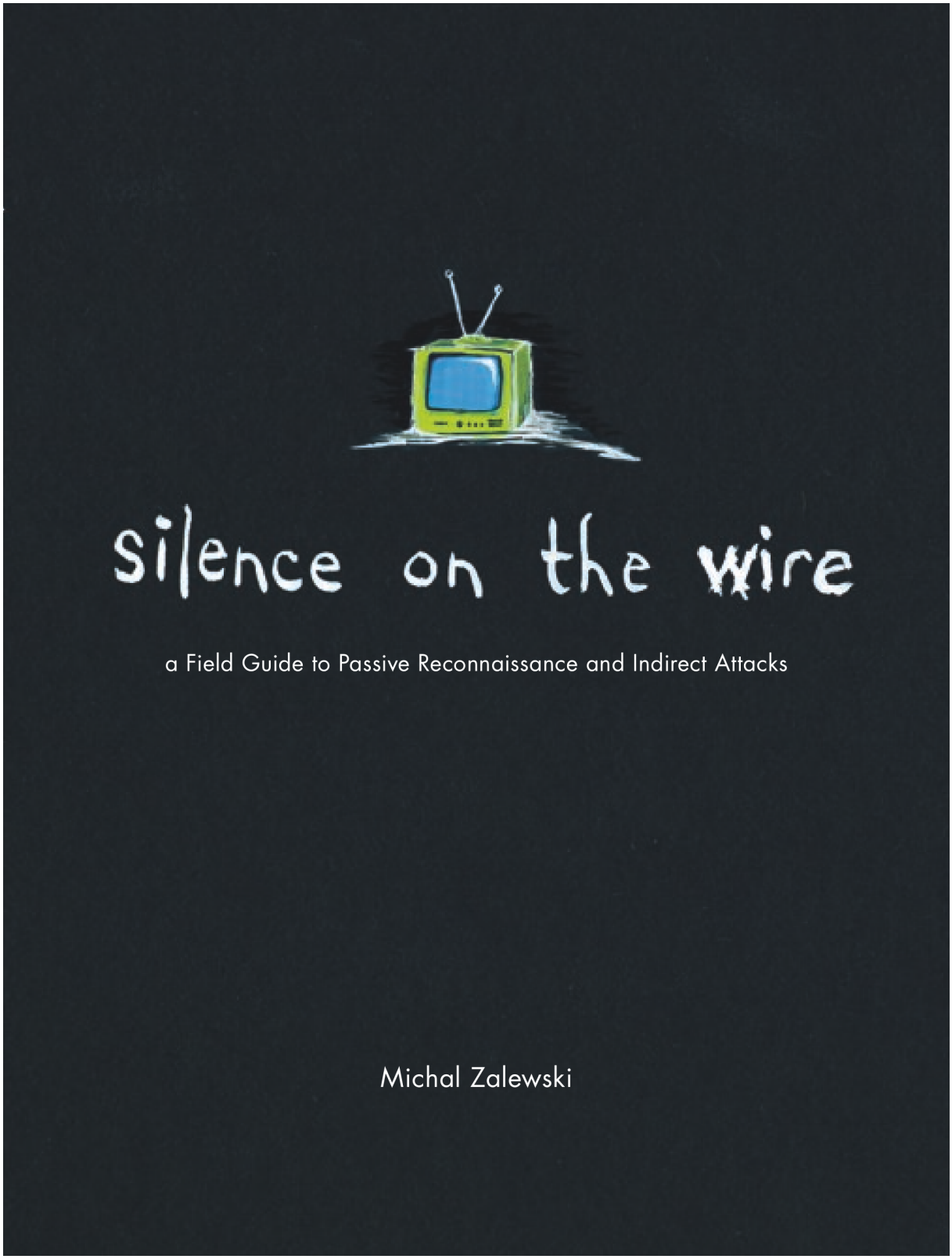
Тишина в проводах. Практическое руководство по пассивной разведке и косвенным атакам

Русский перевод практического руководства Михала Залевского по пассивной разведке, побочным каналам и косвенным атакам.

Больше крутых материалов в моём телеграм-канале @grogrigs

Тишина в проводах



Практическое руководство по пассивной разведке и косвенным атакам

Михал Залевски

Сан-Франциско

Сведения об издании

15 14 13 12 6 7 8 9

ISBN-10: 1-59327-046-1
ISBN-13: 978-1-59327-046-9

• Издатель: William Pollock
• Ответственный редактор: Karol Jurado
• Руководитель производства: Susan Berge
• Дизайн обложки и внутреннего оформления: Octopod Studios
• Редакторы-разработчики: William Pollock и John Mark Walker
• Технический рецензент: Solar Designer
• Литературный редактор: Pat Coleman
• Верстальщик: Riley Hoffman
• Корректор: Stephanie Provines
• Составитель указателя: Ted Laux

За сведениями о распространителях книги или переводах обращайтесь непосредственно в No Starch Press, Inc.:

No Starch Press, Inc.
555 De Haro Street, Suite 250, San Francisco, CA 94107
телефон: 415.863.9900; факс: 415.863.9950; info@nostarch.com; nostarch.com

Каталожные данные Библиотеки Конгресса

Zalewski, Michal.

Silence on the wire: a field guide to passive reconnaissance and indirect attacks / Michal Zalewski.

p. cm.
Включает предметный указатель.
ISBN 1-59327-046-1

1. Компьютерные сети - меры безопасности. I. Название.
TK5105.59.Z35 2005
005.8-dc22
2004009744

права.

Посвящение

Майе.

Об авторе

Михал Залевски - исследователь информационной безопасности, освоивший эту область самостоятельно. Он занимался темами от принципов проектирования аппаратного обеспечения и операционных систем до компьютерных сетей. С середины 1990-х годов он активно искал ошибки, часто публиковался в Bugtraq и создал популярные инструменты безопасности, в том числе p0f - средство пассивного определения операционной системы. Он также опубликовал ряд высоко оценённых исследовательских работ по безопасности. Михал работал экспертом по безопасности в нескольких известных компаниях как в родной Польше, так и в США, включая двух крупных операторов связи. Будучи увлечённым исследователем и время от времени программистом, Михал также пробует силы в искусственном интеллекте, прикладной математике и электронике, а ещё занимается любительской фотографией.

Предисловие

Что нужно, чтобы написать по-настоящему новую книгу о компьютерной безопасности? Или, точнее, что нужно, чтобы написать новеллу о современных вычислениях?

Молодой, но уже чрезвычайно опытный автор, одарённый во многих областях, среди которых многочисленные аспекты вычислительной техники, математика и электроника - и, быть может, увлечение робототехникой, - а также имеющий другие, на первый взгляд никак с ними не связанные интересы - скажем, фаталистическую эротическую фотографию, - и, разумеется, талант и желание писать.

Давным-давно в тёмном и почти неисследованном лесу волшебная химия деревьев - то есть клеток мозга - породила бит информации, лишь затем, чтобы отпустить его в плавание по стремительной реке, в бескрайнее море - Интернет, - где он в конце концов обретёт новый дом, могилу или, возможно, место в музее.

Так начинается эта история. Добрый наш маленький бит или злой, ещё в юном возрасте он попадёт в поток, текущий к сияющему замку из фольги светлого цвета, который многие, впрочем, считают чёрным ящиком. Он пройдёт через вход и подойдёт к стойке регистрации. Не будь он так наивен и близорук, он мог бы заметить группу зловещих битов: те издали следят за стойкой и отмечают время прибытия и отбытия других битов. Но выбора у него всё равно нет: придётся зарегистрироваться.

Отдохнув, нашему герою, возможно, предложат объединиться с братьями и сёстрами или присоединиться к группе других битов и битесс, после чего все они плотно улягутся в подержанную надувную лодку. Внимательный бит может заметить в лодке кусочки мусора - или это не мусор? - вероятно, оставленные предыдущей группой.

Следя за сигналами светофоров и протискиваясь сквозь пробки, наши биты входят в безопасную гавань и плывут к причалу. Заметят ли их из соседних замков и с маяков? Станет ли кто-нибудь следить за переключением светофоров, чтобы точно установить время отплытия нашей группы? Включит ли кто-нибудь свет на причале и сделает снимки? Присвоят ли те злые биты личность наших героев и не выйдут ли в море первыми? Наши биты этого не знают. И вот на причале они пересаживаются в другую лодку и выходят в море...

Путешествие наших любимых битов продолжается, а впереди их ждёт ещё немало опасностей.

Нет, книга Михала не прячет технические подробности за сказкой, как это сделал я. Напротив: читать её очень интересно, однако все факты изложены точно, а ответы на большинство задач, поставленных в начале каждой главы, даются без промедления.

Silence on the Wire уникальна во многих отношениях, но особенно выделяются два. Во-первых, она подробно рассматривает почти все важнейшие этапы обработки данных, благодаря которым сегодня возможно «объединение сетей»: от нажатия клавиши до предполагаемого конечного результата этого нажатия. Во-вторых, она описывает в значительной мере игнорируемые, малоизученные и неотъемлемые проблемы безопасности, связанные с каждым этапом работы сети и со всем процессом в целом. Рассмотренные проблемы прекрасно показывают искусство исследования уязвимостей как с точки зрения нападающего, так и с точки зрения защитника и побуждают читателя продолжить исследования самостоятельно.

Очевидно, книга о компьютерной безопасности не может быть всеобъемлющей. В SotW Михал сделал провокационный выбор и не стал рассматривать все широко известные, но чрезвычайно опасные и распространённые уязвимости и атаки, которые сегодня обсуждает и исследует большая часть сообщества информационной безопасности. Он расскажет вам о тонких временных атаках по нажатиям клавиш, но не станет напоминать, что программные «троянские кони» с возможностью

регистрации нажатий сейчас и распространённее, и проще в применении, чем могла бы оказаться любая такая атака.

Зачем говорить о времени нажатий, но не упоминать трояны? Потому что временные атаки в значительной степени недооценены и неверно поняты даже специалистами по информационной безопасности, тогда как трояны - широко известная и очевидная угроза. Уязвимость для временных атак является свойством конструкции множества задействованных компонентов, а для внедрения трояна необходима либо ошибка в программе, либо ошибка конечного пользователя.

Точно так же, за редкими исключениями, в SotW вы почти не встретите упоминаний об активно эксплуатируемых программных ошибках и даже об общих классах таких ошибок, например о «переполнениях буфера». Если вы ещё не знакомы с распространёнными угрозами компьютерной безопасности и хотите получить эти знания, путешествие по этой книге придётся дополнить чтением менее увлекательных материалов из Интернета и других книг, в особенности материалов, относящихся к конкретным операционным системам, которыми вы пользуетесь.

Вы можете спросить: зачем изучать тишину, разве она не есть ничто? В некотором смысле - да. Ноль в этом смысле тоже ничто. Но вместе с тем это число, понятие, без которого мы едва ли смогли бы по-настоящему понять мир.

Наслаждайтесь тишиной - насколько это возможно.

Alexander Peslyak
основатель и технический директор
Openwall, Inc.
более известный как
Solar Designer
руководитель проекта Openwall
январь 2005 года

Введение

Несколько слов обо мне

Похоже, компьютерщиком я родился, но моё знакомство с сетевой безопасностью началось лишь по воле случая. Я всегда любил экспериментировать, исследовать новые идеи и решать задачи, которые кажутся чётко сформулированными, но всё же ускользают и требуют изобретательного, творческого подхода, - пусть даже результатом окажется лишь неудачная попытка. В юности я почти всё время посвящал порой рискованным и зачастую глупым попыткам исследовать мир химии, математики, электроники и, наконец, вычислительной техники, вместо того чтобы целыми днями кататься на велосипеде вокруг квартала. (Вероятно, я немного преувеличиваю, но мама, кажется, всегда за меня тревожилась.)

Вскоре после первого знакомства с Интернетом - это было в середине 1990-х, примерно через восемь лет после того, как я написал первую программу «Hello world» на любимой 8-битной машине, - я получил необычное предложение. В спам-письме, как ни трудно поверить, меня и ещё пару тысяч человек приглашали вступить в подпольную группу предположительно злонамеренных хакеров в чёрных шляпах. В подполье я не ушёл - возможно, благодаря сильному инстинкту самосохранения, в некоторых кругах известному как трусость, - однако письмо каким-то образом стало хорошим побуждением подробнее изучить компьютерную безопасность. За плечами у меня уже было немало любительского программирования, и меня увлекла возможность взглянуть на код с другой стороны, попытаться заставить алгоритм сделать нечто сверх положенного. Интернет казался великолепным источником желанных задач: огромная сложная система, подчинённая единственному руководящему принципу - по-настоящему доверять нельзя никому. Так всё и началось.

У меня нет той подготовки, какую можно было бы ожидать от типичного специалиста по компьютерной безопасности, профессия которого сегодня становится всё более привычной. Я никогда не получал формального образования в области информатики и не обладаю внушительно звучащим набором сертификатов. Безопасность всегда была одной из главных моих страстей, а теперь стала ещё и работой. Я не стереотипный компьютерщик: время от времени я всё-таки встаю, чтобы посмотреть на свою работу со здравого расстояния или вообще отойти от компьютеров.

Хорошо это или плохо, всё сказанное повлияло на форму и идею этой книги. Я хочу показать другим, как сам вижу компьютерную безопасность, а не то, как её обычно преподают. Для меня безопасность - не отдельная задача, которую нужно решить, и не простой процесс, которому нужно следовать. Она не сводится к профессионализму в какой-то одной области. Это упражнение в способности увидеть всю экосистему и понять каждый её компонент.

Об этой книге

Даже в тусклом свете мониторов мы остаёмся всего лишь людьми. Нас учили доверять окружающим, а становиться чрезмерно подозрительными нам не хочется. Чтобы жить удобно, приходится искать разумный компромисс между безопасностью и продуктивностью.

И всё же Интернет отличается от общества реального мира. Соблюдение правил не приносит здесь общей выгоды, а раскаяние за виртуальные проступки встречается редко. Мы не можем

просто доверять системе, и попытки вывести единое правило, применимое ко всем проблемам, закончатся жалким провалом. Инстинктивно мы проводим прямую черту между «нами» и «ими», а собственный остров объявляем безопасным. Затем начинаем высматривать на горизонте пиратские корабли. Вскоре проблемы безопасности представляются нам локальными отклонениями, которые легко определить, диагностировать и устранить. С этой точки зрения кажется, будто нападающими движут ясные мотивы и, если сохранять бдительность, их можно заметить и остановить на подходе.

Однако виртуальный мир устроен совсем иначе: безопасность не означает отсутствия ошибок, а защищённость не достигается одной лишь недосягаемостью для нападающих. Практически любой процесс, связанный с информацией, несёт неотъемлемые последствия для безопасности; они становятся видны, как только мы смотрим дальше непосредственной цели этого процесса. Искусство понимать безопасность - это всего лишь искусство переступать черту и смотреть с другой стороны.

Надеюсь, перед вами необычная книга. Это не собрание проблем и не руководство по защите ваших систем. Она начинается с попытки проследить историю частицы информации от мгновения, когда ваши руки касаются клавиатуры, до удалённой стороны на другом конце провода. В ней рассматриваются технологии и их последствия для безопасности, причём основное внимание уделяется проблемам, которые нельзя назвать программными ошибками: в них может не быть нападающего, изъяна, который следовало бы проанализировать и исправить, или обнаружимой атаки - по крайней мере такой, которую можно отличить от законной деятельности. Цель книги - показать, что понять Интернет можно лишь набравшись смелости выйти за пределы спецификаций или прочесть то, что скрывается между строк.

Как подсказывает подзаголовок, книга посвящена проблемам конфиденциальности и безопасности, неотделимым от повседневной связи и вычислений. Одни из них имеют глубокие последствия, другие просто любопытны и заставляют задуматься. Ни одна не нанесёт вашей среде немедленного ущерба и не уничтожит данные на диске. Приведённая здесь информация полезна и ценна для специалистов по информационным технологиям и опытных любителей, желающих дать своему уму непростую работу и узнать о неочевидных последствиях проектных решений. Эта книга написана для тех, кто хочет научиться пользоваться подобными тонкостями, чтобы управлять собственной средой и получить преимущество перед внешним миром.

Книга разделена на четыре части. Первые три посвящены этапам движения данных и применяемым на них технологиям. Последняя рассматривает сеть в целом. В каждой главе описываются относящиеся к соответствующему этапу элементы технологии обработки данных, обсуждаются последствия для безопасности, демонстрируются побочные эффекты, предлагаются способы решения проблем, если это возможно, и даются рекомендации по дальнейшему изучению темы. Я всеми силами стараюсь обходиться без схем, таблиц, многостраничных спецификаций и тому подобного, хотя многочисленные сноски вам всё же встретятся. Поскольку в сети легко найти множество хороших справочных материалов, я прежде всего стремился сделать эту книгу просто увлекательной.

Начнём?

Часть I. Источник

О проблемах, которые возникают задолго до отправки какой-либо информации по сети.

Глава 1. Я слышу, как вы печатаете

Где мы исследуем, как за нажатиями ваших клавиш можно следить издалека, очень издалека.

С момента первого нажатия клавиши отправляемая вами информация начинает долгое путешествие по виртуальному миру. За микросекунды до того, как пакеты понесутся по оптоволоконным линиям и станут отражаться от спутниковых приёмопередатчиков, частица информации проделывает немалый путь через удивительный лабиринт электрических цепей. Прежде чем нажатия клавиш примет операционная система и работающие под её управлением приложения, в дело вступает множество точных и тонких низкоуровневых механизмов. Этот процесс интересует самых разных хакеров и, как оказалось, имеет большое значение для специалистов по безопасности. На пути в пространство пользователя скрывается немало сюрпризов.

Эта глава посвящена ранним этапам передачи данных и возможностям, которые позволяют другим пользователям - возможно, не самым добропорядочным - узнать непозволительно много о том, чем вы занимаетесь, удобно устроившись за собственным терминалом.

Яркий пример потенциальной утечки, связанной со способом обработки пользовательского ввода компьютером, относится к теме, которая на первый взгляд кажется в лучшем случае никак с ним не связанной: к трудной задаче получения случайных чисел на машине, ведущей себя совершенно предсказуемо. Менее очевидную связь трудно себе представить, но описываемая проблема вполне реальна и может позволить скрытному наблюдателю восстановить значительную часть действий пользователя - от вводимых паролей до личного электронного письма.

Потребность в случайности

Компьютеры полностью детерминированы. Они обрабатывают данные по чётко определённому своду законов. Инженеры всеми силами компенсируют недостатки производственного процесса и свойства самих электронных компонентов - помехи, тепловой шум и тому подобное, - чтобы системы всегда следовали одной и той же логике и работали правильно. Когда со временем и под нагрузкой компоненты перестают вести себя ожидаемым образом, мы считаем компьютер неисправным.

Способность машин достигать подобной последовательности в сочетании с их замечательными вычислительными возможностями и делает компьютеры великолепным инструментом для тех, кому удаётся ими овладеть и управлять. Разумеется, нужно сделать одну оговорку: не всё так радужно, и жалующиеся на ненадёжность компьютеров не совсем неправы. Несмотря на безупречную работу оборудования, сами программы время от времени ведут себя неверно. Дело в том, что хотя аппаратное обеспечение может быть - и часто бывает - последовательным и надёжным, обычно невозможно надолго предсказать поведение достаточно сложной программы, не говоря уже о сложной совокупности взаимозависимых программ, такой как типичная операционная система. Поэтому проверка программы крайне трудна, даже если предположить, что мы способны построить подробную, достаточно строгую и при этом безошибочную гипотетическую модель того, как программа должна работать. Почему? В 1936 году Алан Тьюринг, отец

современных вычислений, методом *reductio ad absurdum* - сведением к абсурду - доказал отсутствие общего способа за конечное время определить результат любой компьютерной процедуры, или алгоритма, хотя для некоторых алгоритмов частные способы существовать могут.[1]

На практике это означает следующее. Нельзя ожидать, что операционная система или текстовый редактор всегда будут вести себя в точности так, как задумали вы или автор программы. Однако вполне разумно ожидать, что два экземпляра текстового редактора на системах с одинаковым оборудованием при одинаковом вводе будут вести себя последовательно и одинаково - если, конечно, один экземпляр не раздавит падающее пианино или на него не повлияет иное досадное внешнее событие. Для производителей программ это прекрасная новость, однако в некоторых случаях нам, специалистам по безопасности, хотелось бы, чтобы компьютер был немного менее детерминированным. Не обязательно в поведении, но хотя бы в результатах, которые он способен породить.

Возьмём шифрование данных, особенно этого загадочного зверя - криптографию с открытым ключом. Эта новая и гениальная разновидность шифрования, причём не только шифрования, впервые предложенная в 1970-х годах Уитфилдом Диффи и Мартином Хеллманом, а вскоре превращённая Роном Ривестом, Ади Шамиром и Леном Адлеманом в полноценную криптосистему, основана на простой мысли: одни задачи труднее других. Конечно, это кажется очевидным, но добавьте несколько понятий высшей математики - и революционное изобретение готово.

Традиционная симметричная криптография требовала передать всем участникам секретного обмена одинаковое общее «секретное» значение - ключ. Ключ необходим и достаточен, чтобы зашифровать, а затем расшифровать передаваемую информацию, поэтому сторонний наблюдатель, знающий метод шифрования, всё равно не способен понять сообщение. Потребность в общем секрете делала весь подход не всегда практичным для компьютерной связи, главным образом потому, что до начала общения сторонам требовалось создать защищённый канал обмена: передача секрета по незащищённому потоку позволила бы расшифровать сообщения. В компьютерном мире мы часто общаемся с системами или людьми, которых никогда прежде не видели и с которыми у нас нет иного доступного и безопасного канала связи.

Криптография с открытым ключом, напротив, не основана на общем секрете. У каждой стороны есть две порции информации: одна - открытый ключ - полезна для создания зашифрованного сообщения, но почти бесполезна для расшифровки; другая - закрытый ключ - полезна для расшифровки ранее зашифрованного сообщения. Теперь стороны могут обменяться открытыми ключами по незащищённому каналу, даже если за ним следят. Они предоставляют друг другу сведения, бессмысленные для наблюдателя, которые необходимы для шифрования сообщений между ними, но держат в тайне ту часть, которая позволяет получить доступ к зашифрованным данным. Так безопасная связь между совершенно незнакомыми сторонами - например, покупателем на диване в собственной квартире и сервером интернет-магазина - стала гораздо ближе к реальности.

В основе исходной криптосистемы с открытым ключом RSA - Rivest, Shamir и Adleman - лежит наблюдение, что вычислительная сложность умножения двух сколь угодно больших чисел сравнительно невелика: она прямо пропорциональна количеству умножаемых цифр. С другой стороны, сложность поиска множителей большого числа - факторизации - значительно выше, если только вы не легендарный криптографический гений из Агентства национальной безопасности. Сначала алгоритм RSA выбирает два произвольных очень больших простых числа[1], `p` и `q`, и перемножает их. Затем он использует произведение вместе с числом `(p - 1)(q - 1)`, взаимно простым[2] с другим параметром, чтобы построить открытый ключ. Этим ключом можно зашифровать информацию, но одного его недостаточно для расшифровки без факторизации.

Вот в чём хитрость: факторизация произведения двух больших простых чисел зачастую практически невыполнима, и такие атаки срываются. Самому быстрому универсальному алгоритму факторизации целых чисел на традиционных компьютерах, методу решета числового поля общего вида, GNFS, потребовалось бы более тысячи лет для нахождения множителей такого 1024-битного целого числа при скорости миллион проверок в секунду. Между тем обычный персональный компьютер находит два простых числа с произведением такого размера за несколько секунд.

Как уже говорилось, в RSA вместе с открытым ключом создаётся и закрытый. Закрытый ключ содержит дополнительные сведения о простых числах, позволяющие расшифровать любую информацию, зашифрованную открытым ключом. Этот трюк возможен благодаря китайской теореме об остатках, теореме Эйлера и другим несколько пугающим, но увлекательным математическим понятиям, которые особенно любознательный читатель, возможно, захочет изучить самостоятельно.[2]

Позднее появились и другие криптосистемы с открытым ключом, основанные на иных трудных математических задачах, включая системы на эллиптических кривых и так далее, но все они используют основную идею открытого и закрытого ключей. Метод оказался практичным для защиты электронной почты, веб-транзакций и многого другого, даже если две стороны никогда не общались и до установления соединения у них не было безопасного канала для обмена дополнительными сведениями.[3] Почти все повседневные схемы шифрования - от Secure Shell, SSH, и Secure Sockets Layer, SSL, до обновлений с цифровой подписью и смарт-карт - существуют благодаря вкладу Диффи, Хеллмана, Ривеста, Шамира и Адлемана.

Автоматическая генерация случайных чисел

Есть лишь одна проблема: при реализации RSA на детерминированной машине сначала нужно получить два очень больших простых числа, p и q. Компьютеру нетрудно найти большое простое число, но остаётся небольшая загвоздка: другие не должны иметь возможности угадать эти числа, и на разных машинах они не могут быть одинаковыми. В противном случае для атаки на алгоритм не понадобилась бы факторизация, а p и q были бы известны любому владельцу похожего компьютера.

За последние годы разработано множество алгоритмов, которые быстро находят кандидаты в простые числа - псевдопростые числа - и выполняют быстрые предварительные проверки простоты, применяемые для проверки таких кандидатов.[3] Но для получения действительно непредсказуемого простого числа нужна хорошая порция энтропии, или случайности: тогда можно вслепую выбрать одно из простых чисел заданного диапазона либо начать со случайного места и взять первое встреченное простое число.

Хотя при создании ключа без некоторой случайности не обойтись, на этом потребность в ней не заканчивается. Криптография с открытым ключом опирается на довольно сложные вычисления и потому работает относительно медленно, особенно в сравнении с традиционной симметричной криптографией, где используются короткие общие ключи и набор операций, которые машины, как известно, выполняют очень быстро.

Чтобы реализовать, например, SSH с приемлемой производительностью, разумнее установить первоначальную связь и выполнить базовую проверку с помощью алгоритмов с открытым ключом, создав тем самым безопасный канал. Затем стороны обмениваются компактным симметричным ключом шифрования, скажем 128-битным, переключаются на старую симметричную криптографию и продолжают общение. Главный недостаток симметричной криптографии устраняется благодаря первоначальному - медленному - защищённому потоку для обмена общим секретом; затем используются более быстрые алгоритмы, и пользователь получает высокую производительность, не жертвуя безопасностью. Но для разумного применения симметричной криптографии всё равно

требуется определённая энтропия, позволяющая создавать непредсказуемый симметричный сеансовый ключ для каждого защищённого сеанса связи.

Безопасность генераторов случайных чисел

Программисты придумали немало способов заставить компьютер создавать числа, кажущиеся случайными; общее название таких алгоритмов - генераторы псевдослучайных чисел, PRNG.

PRNG достаточно для простых задач, например для создания «случайных» событий в компьютерных играх или бессмысленных тем особенно назойливых массовых спам-рассылок. Возьмём линейный конгруэнтный генератор, также называемый генератором степенных вычетов,[4] - классический пример такого алгоритма. Несмотря на непонятное название, при каждом получении «случайного» результата этот генератор выполняет последовательность простых операций: умножение, сложение и взятие остатка[4]. Для вычисления следующего значения `r(t+1)` он использует предыдущий результат `r(t)`, где `t` обозначает время:

```
r(t+1) = (a * r(t) + c) mod M
```

Операция взятия остатка ограничивает диапазон и предотвращает переполнение - ситуацию, когда результат на некотором этапе выходит за заранее определённые границы значений. Если `r(0)`, `a`, `M` и `c` - набор управляющих переменных генератора - являются положительными целыми числами, все результаты уравнения лежат в диапазоне от 0 до `M - 1`.

Однако, хотя после некоторой настройки результаты алгоритма могут обладать статистическими свойствами, подходящими для создания подобий случайных чисел, в самих его действиях нет ничего по-настоящему непредсказуемого. В этом и заключается проблема: нападающий легко может создать собственную копию генератора и с её помощью определить любое количество результатов, которые выдаст наш генератор. Даже если исходное состояние генератора, `r(0)`, нападающему неизвестно, зачастую он способен успешно вывести важные свойства этого значения, наблюдая за последующими результатами генератора жертвы, а затем использовать полученные сведения, чтобы настроить свою версию и заставить её подражать нашей. В действительности общий метод восстановления и предсказания всех полиномиальных конгруэнтных генераторов был разработан более десяти лет назад,[5] и игнорировать эту небольшую, пусть и несколько неудобную подробность было бы крайне неразумно: при использовании алгоритма в критически важных задачах она оставляет зияющую брешь.

Со временем мы поняли, что единственный разумный способ для компьютера получить практически непредсказуемые данные - если не считать крупного сбоя памяти или расплавления процессора - состоит в том, чтобы собрать как можно больше практически непредсказуемой информации из физического окружения, а затем передавать это значение любому приложению, которому нужна хорошая случайность. Проблема в том, что у обычного компьютера нет «органов чувств», которыми он мог бы исследовать среду в поисках внешних сигналов, кажущихся случайными. И всё же нам известен довольно хороший способ обойти это неудобство.

Энтропия ввода-вывода: это говорит ваша мышь

Почти в любой компьютерной системе внешние устройства сообщают о значимых асинхронных событиях - например, о появлении сведений от сетевой платы или клавиатуры - с помощью механизма аппаратных прерываний. Каждому устройству назначен номер аппаратного прерывания, IRQ, и оно сообщает о важных событиях, изменяя напряжение на выделенной аппаратной линии внутри компьютера, соответствующей этому IRQ. Изменение интерпретирует устройство, называемое программируемым контроллером прерываний, PIC, которое служит личным дворецким

главного процессора или процессоров.

Получив указания от CPU, PIC решает, следует ли, когда, каким образом и с каким приоритетом передать главному устройству запросы от внешних устройств; благодаря этому процессору проще эффективно и надёжно обрабатывать события. Приняв сигнал от PIC, процессор откладывает текущую задачу, если только CPU не решил в данный момент игнорировать все запросы прерываний, поскольку очень занят. Затем он вызывает код, назначенный операционной системой для обработки обратной связи от данного устройства или группы устройств. Когда программа обработает событие, CPU восстанавливает исходный процесс и его контекст - сведения о состоянии окружения в момент прерывания - и продолжает работу, словно ничего не произошло.

Доставка прерываний: практический пример

На практике между обнаружением внешнего условия и созданием и приёмом IRQ выполняется множество дополнительных действий. Например, на рисунке 1-1 показана последовательность событий, которую запускает нажатие или отпускание клавиши. Ещё до того, как вы коснётесь первой клавиши, крошечная микросхема микроконтроллера внутри клавиатуры, выполняющая роль контроллера клавиатуры, непрерывно обследует её в поисках изменений состояния.

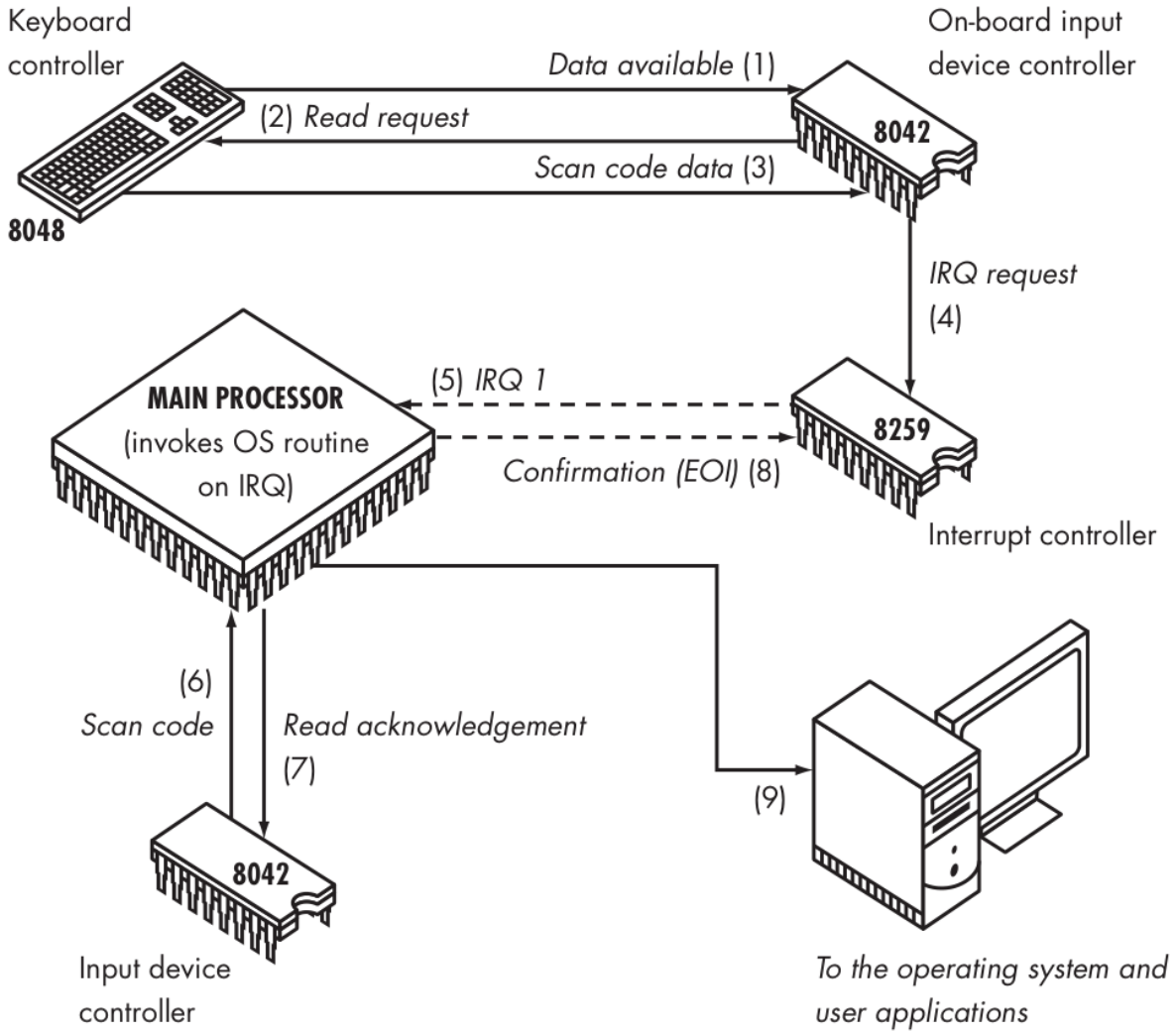


Рисунок 1-1. Обмен данными между клавиатурой и компьютером

Схема содержит следующие элементы и этапы: контроллер клавиатуры 8048 сообщает о доступности данных (1); контроллер устройства ввода 8042 отправляет запрос чтения (2); передаются данные скан-кода (3); встроенный контроллер устройства ввода отправляет запрос IRQ (4); контроллер прерываний 8259 передаёт IRQ 1 (5) главному процессору, который по IRQ вызывает процедуру операционной системы; далее следуют скан-код (6), подтверждение чтения (7), подтверждение окончания прерывания EOI (8) и передача скан-кода операционной системе и пользовательским приложениям (9).

Клавиатура устроена как решётка горизонтальных и вертикальных проводников. В месте пересечения каждой строки и столбца установлена клавиша - микропереключатель или чувствительный к давлению мембранный переключатель. Контроллер по отдельности и с очень высокой скоростью проверяет каждую строку и каждый столбец.

Если, например, во время проверки строки 3 и столбца 5 контроллер клавиатуры обнаруживает замкнутую цепь - на это указывает низкое сопротивление при подаче напряжения на соответствующие линии, - он заключает, что клавиша в данной позиции, `J`, нажата. Обнаружив изменение, контроллер преобразует координаты строки и столбца в скан-код - значение, однозначно определяющее клавишу. Скан-код помещается в очередь внутреннего буфера микросхемы, после чего та сообщает CPU о новых данных и возвращается к своим делам.

Микросхема контроллера ввода на материнской плате - партнёр контроллера клавиатуры. Обычно контроллер ввода обслуживает все основные устройства ввода, например мышь и клавиатуру. Он принимает от клавиатурной микросхемы один скан-код и передаёт соответствующее прерывание дворецкому CPU - контроллеру PIC. Как только PIC определяет, что это IRQ можно доставить, он передаёт сигнал процессору. Обычно процессор прерывает текущую задачу и вызывает обработчик прерывания, установленный операционной системой. Обработчик должен прочитать данные и сообщить микросхеме об успешном чтении скан-кода. Затем контроллер ввода возвращается к обычной работе и со временем считывает с клавиатуры ещё один скан-код, если в буфере есть данные.[5]

Эта схема важна для генерации случайных чисел, хотя связь здесь косвенная. Через механизм уведомления об асинхронных событиях - прерывания - компьютер почти мгновенно и точно получает обратную связь о действиях пользователя; пожалуй, особенно интересны аккуратно измеренные задержки между нажатиями клавиш. Эти сведения не всегда непредсказуемы, но, возможно, представляют собой лучший доступный машине внешний измеримый и в некоторой степени недетерминированный сигнал. Поэтому, чтобы обойти детерминированную природу компьютера и внести случайность в вычисления, авторы защищённых реализаций PRNG собирают энтропию из в целом непредсказуемого поведения определённых устройств: мыши, клавиатуры, сетевых интерфейсов, а иногда и дисков. Для этого они добавляют в обработчик прерываний операционной системы дополнительный код, который записывает некоторые параметры каждого подходящего события.

Можно возразить, что ни один из этих источников не всегда даёт по-настоящему случайную обратную связь: например, после того как пользователь набрал `aardva`, следующими двумя символами, скорее всего, окажутся `rk`. Но некоторые стороны поведения - в частности, то, что мне вообще пришло в голову подумать о трубкозубах, - с практической точки зрения действительно весьма непредсказуемы, если не углубляться в академическую дискуссию о свободе воли и детерминированных вселенных. Этот способ добавления энтропии работает достаточно хорошо, поскольку объединяет несколько факторов, которые нападающий не способен разумным образом учесть, отследить или предсказать, сохранив при этом рассудок. Если долго собирать данные из всех этих источников, законы вероятности говорят, что мы накопим некоторое количество энтропии. Собирая данные в буфере, мы создаём пул энтропии, который может наполняться или истощаться в зависимости от спроса и предложения непредсказуемых данных. К сожалению, небольшие

фрагменты случайности в этом пуле - те случаи, когда на наш набор повлияли космические события, - всё ещё смешаны с большим объёмом легко предсказуемых данных, а потому не могут немедленно использоваться для получения случайных чисел.

Чтобы фактическая энтропия, собранная при поддержании и пополнении пула, равномерно распределилась по всем выходным битам PRNG и все непредсказуемые данные были израсходованы, пул нужно хешировать: тщательно перемешать так, чтобы ни одну часть данных нельзя было предсказать легче другой. Каждый выходной бит должен нетривиальным образом в равной степени зависеть от всех входных битов. Добиться этого, не зная, какие сведения предсказуемы, а какие нет - а компьютер, следящий за нажатиями клавиш и движениями мыши, такой информацией напрямую не располагает, - бывает трудно.

Односторонние функции быстрого преобразования

К счастью, безопасные односторонние хеш-функции, или функции получения дайджеста сообщения, - передовое достижение современной криптографии - помогают перемешать данные так, чтобы каждый выходной бит получил максимум энтропии независимо от неравномерности входа. Такие функции создают отпечаток фиксированной длины - уникальный идентификатор произвольного блока входных данных. Но это ещё не всё.

У всех односторонних хеш-функций есть два важных свойства:

• Вычислить отпечаток легко, но по результату невозможно восстановить исходное сообщение или какое-либо его свойство. Любое конкретное изменение сообщения с той же вероятностью повлияет на все свойства результата, что и любое другое изменение.
• Вероятность совпадения отпечатков двух различных сообщений определяется лишь размером отпечатка. Если он достаточно велик, чтобы полный перебор был непрактичен - сегодня это примерно 128-160 бит, или от приблизительно 3,4E+38 до 1,46E+48 комбинаций, - найти два сообщения с одинаковым отпечатком невозможно.

Таким образом, функции получения отпечатка позволяют равномерно распределить энтропию входных данных по выходным. Это решает проблему источников энтропии, которые случайны в целом, но предсказуемы локально: мы собираем из окружения приблизительное количество энтропии, смешанной или не смешанной с предсказуемыми данными, и можем получить отпечаток, гарантированно столь же непредсказуемый, как первоначально собранная энтропия, независимо от её распределения во входных данных.

Как работают функции получения отпечатка? Некоторые снова опираются на математические задачи, которые, насколько нам известно, решить очень трудно. Фактически любой безопасный алгоритм симметричной криптографии или криптографии с открытым ключом легко превратить в безопасную хеш-функцию. Пока человечество не найдёт действительно хитроумного решения какой-либо из этих задач, полагаться на такой подход вполне допустимо.

Однако, выкатывая тяжёлую артиллерию, мы получаем медленные и чрезмерно сложные средства вычисления отпечатков, часто непрактичные для компактных реализаций, особенно при интеграции решения с операционной системой. Альтернатива - обрабатывать данные так, чтобы взаимозависимость всех входных и выходных битов оказалась достаточно сложной для полного сокрытия входного сообщения, и надеяться, что этого «достаточно» для противодействия известным методам криптоанализа. Поскольку выражение «будем надеяться, достаточно» фактически служит девизом немалой части информатики, мы охотно принимаем такой подход как разумный.

Преимущество последней группы алгоритмов, включающей популярные функции MD2, MD4, MD5 и SHA-1, состоит в том, что обычно они гораздо быстрее и проще в применении, чем аналоги,

основанные на трудных математических задачах, а при хорошем проектировании не поддаются стандартным приёмам криптоанализа. Их слабость в недоказуемой безопасности: ни один из них не сводится к классической трудноразрешимой задаче. И действительно, у некоторых были доказаны конкретные слабости.[6]

Как уже отмечалось, важнейшая услуга функций получения отпечатка генераторам псевдослучайных чисел состоит в следующем. Можно обработать участок данных с n случайными битами и любым количеством предсказуемых битов и получить отпечаток, который равномерно распределит n битов энтропии по всем своим битам благодаря двум рассмотренным выше основным свойствам односторонних функций. Таким образом, функция получения отпечатка становится удобным экстрактором энтропии. Обработав ею достаточный объём данных, собранных в целом непредсказуемым обработчиком прерываний, мы получаем случайные числа, не раскрывая ценных сведений о точной форме исходной информации и не рискуя, что несовершенный ввод сколько-нибудь существенно повлияет на результат. Нужно лишь удостовериться, что в порции данных прерываний собрано и передано функции достаточное количество энтропии, иначе под угрозой окажется вся схема. Если нападающий способен предсказать значительную часть данных, используемых для получения случайных чисел, а для оставшейся части существует всего несколько возможных комбинаций, он может успешно атаковать нашу реализацию полным перебором, просто проверив все возможные значения. Например, если функция создаёт 128-битные дайджесты, то независимо от фактического объёма собранных данных - будь то 200 байт или 2 мегабайта стука по клавишам - до хеширования нужно убедиться, что как минимум 128 входных битов непредсказуемы для нападающего.

Как важно быть педантичным

Рассмотрим пример, в котором всё может пойти не так. Пользователь решает написать сценарий оболочки, когда системный пул энтропии пуст - возможно, некоторое время назад была выполнена операция, поглотившая множество случайных чисел. Нападающий замечает, что пользователь пишет сценарий, поскольку выполняется команда `vi delallusers.sh`, и предполагает, что сценарий, вероятно, начинается примерно с `#!/bin/sh`. Точно знать продолжение он не может, однако разумно ожидать, что в начале сценария будет вызвана команда оболочки, а безвкусное стихотворение о трубкозубах несколько менее вероятно.

В этот момент некая программа шифрования внезапно запрашивает у системы 128-битное случайное число, которое послужит сеансовым ключом для защиты связи. Однако система неверно оценивает объём энтропии в буфере, записавшем процесс набора первых строк сценария, и задача нападающего становится лёгкой. Компьютер не располагает сведениями о том, предсказуемо ли для окружающих конкретное действие, которое пользователь выполняет именно сейчас. Он может лишь предполагать - опираясь на допущения программиста, - что за несколько минут или часов действия пользователей сложатся в нечто, чего нельзя было точно предсказать, и что в среднем именно такая часть ввода действительно зависит от факторов, непредсказуемых для нападающего.

Теперь нападающий знает большую часть содержимого пула энтропии, а для неизвестной части остаётся едва ли несколько тысяч вариантов, хотя операционная система убеждена, что энтропии в буфере гораздо больше. Для человека, вооружённого компьютером, несколько тысяч вариантов - совсем небольшое препятствие. В результате ничего не подозревающая криптографическая программа вместо 128-битного случайного числа с 39-значным количеством комбинаций получает число, входные данные которого могли принимать лишь одно из нескольких тысяч значений. Нападающий легко проверит их методом проб и ошибок и вскоре расшифрует сведения, которые должны были остаться защищёнными.

Энтропией нельзя разбрасываться

Поскольку точно предсказать объём энтропии, собранной у пользователя за короткое время, практически невозможно, все реализации для защиты от описанной выше проблемы предсказуемого результата PRNG включают в процесс получения нового результата предыдущий отпечаток или внутреннее состояние PRNG. Предыдущий результат становится частью уравнения, по которому вычисляется следующее значение PRNG.

При такой конструкции после первоначального накопления в системе достаточной энтропии последние данные, пополняющие пул, не обязаны постоянно быть полностью случайными, чтобы обеспечивать базовую безопасность.

Но возникает другая проблема. Если реализация долго работает на старой, унаследованной энтропии, лишь снова и снова хешируемой с помощью MD5 или SHA-1, она становится полностью зависимой от безопасности алгоритма получения отпечатка, которому нельзя безоговорочно доверять из-за рассмотренного выше компромисса между производительностью и безопасностью. Более того, компетентные криптографы могли не провести надлежащую оценку пригодности хеш-функций именно для такого применения. Теперь реализация опирается уже не просто на свойства перемешивания битов функцией отпечатка, а полностью зависит от её неуязвимости для взлома. Если с каждым последующим шагом раскрывается небольшая часть сведений о внутреннем состоянии генератора, а новые непредсказуемые данные в пул не добавляются, со временем накопленной информации может хватить, чтобы с достаточной уверенностью восстановить или угадать внутреннее состояние и получить возможность предсказывать дальнейшее поведение устройства. С другой стороны, если новые случайные данные добавляются со скоростью, которая хотя бы статистически предотвращает существенное повторное использование внутреннего состояния, атака становится гораздо менее осуществимой даже при фундаментальной уязвимости хеш-функции.

Многие специалисты считают, что в самых требовательных применениях хеш-функции нельзя оказывать такое доверие и настолько на неё полагаться. Поэтому реализация должна учитывать оценочный объём энтропии, собранной в системе: даже если мгновенная оценка неточна, она отражает общую статистическую тенденцию, ожидаемую от используемых источников. Небольшие краткосрочные колебания доступности внешней энтропии, подобные примеру с редактированием сценария, допустимы и компенсируются алгоритмом повторного использования результата. Но точные долгосрочные прогнозы всё же необходимы: они обеспечивают частое пополнение внутреннего пула энтропии и сокращают риск на случай, если хеш-функция со временем начнёт раскрывать внутреннее состояние. Таким образом, реализация должна учитывать всю энтропию, израсходованную на данные для пользовательских процессов, и отказываться выдавать новые случайные числа, пока достаточный объём энтропии не станет доступен.

Хороший пример правильной реализации PRNG, учитывающей всё сказанное, - превосходная система, разработанная и реализованная в 1994 году Теодором Цо из Массачусетского технологического института. Его механизм `/dev/random` впервые появился в Linux, а позднее был перенесён в FreeBSD, NetBSD и HP/UX. Механизм Цо наблюдает за рядом системных событий ввода-вывода, измеряя интервалы времени и другие важные характеристики прерываний. Он также сохраняет пул энтропии на диск при завершении работы, чтобы после загрузки система не оказалась в полностью предсказуемом состоянии; это ещё больше затрудняет атаку.

Атака: последствия внезапной смены парадигмы

В чём может состоять проблема этой на вид безошибочной схемы, которая предоставляет непредсказуемые случайные числа требовательным приложениям? Ни в чём - по крайней мере не

там, где её можно было бы ожидать. Полученные числа действительно трудно предсказывать.

Однако в рассуждениях автора технологии есть одна небольшая, но гибельная ошибка. Конструкция господина Цо предполагает, что нападающий хочет предсказывать случайные числа по имеющимся у него сведениям о машине и её окружении. Но что, если нападающему нужно прямо противоположное?

Нападающий с учётной записью на машине, не имея непосредственного доступа к вводимой пользователем информации, способен определить точный момент активности ввода. Для этого он опустошает пул энтропии - достаточно запросить у системы случайные данные и выбросить их, - а затем следит за доступностью результатов PRNG. При отсутствии активности ввода-вывода новых данных у PRNG не появится, поскольку оценка энтропии не изменится. Если клавишу нажать или отпустить, нападающему станет доступен небольшой объём информации, и он сможет заключить, что произошло нажатие или отпускание.

Другие события, например дисковая активность, тоже создают некоторый результат PRNG, но объём и временные шаблоны собранной таким способом энтропии отличаются от характеристик данных клавиатурных прерываний. Поэтому по объёму данных, доступных в каждый момент, события легко различить. Данные от нажатий будут выглядеть иначе, чем данные дисковой активности.

В конечном счёте механизм, призванный обеспечить максимально возможную безопасность создания случайных чисел, снижает конфиденциальность пользователя. Возможностью оценивать объём энтропии, полученной от внешнего источника, можно злоупотребить и следить за некоторыми аспектами системного ввода. Нападающий не видит, что именно набирается, однако набор разных слов на клавиатуре создаёт устойчивые временные шаблоны, особенно при наличии точных сведений о нажатиях и отпусканиях, как в данном случае. Анализируя эти шаблоны, нападающий способен восстановить фактический ввод или, по крайней мере, существенно облегчить его угадывание.

Более пристальный взгляд на временные шаблоны ввода

Углублённый анализ, выполненный группой исследователей Калифорнийского университета,[7] показывает, что одни лишь интервалы между нажатиями позволяют определить некоторые свойства пользовательского ввода и даже полностью восстановить данные. Исследователи пришли к выводу, что при плавном наборе опытным пользователем интервалы могут несколько различаться, однако преобладающие временные шаблоны каждого перехода от клавиши к клавише видны отчётливо.

Причина в определённом положении рук на клавиатуре и в том, что расположение клавиши влияет на скорость, с которой палец может до неё добраться. Например, интервал между нажатиями `e` и `n` обычно отличается от интервала между `m` и `l`. В первом случае одна рука обслуживает левую сторону клавиатуры, а другая - правую, как показано на рисунке 1-2, поэтому набор обоих символов почти не требует движения, и клавиши нажимаются практически одновременно, с интервалом менее 100 миллисекунд. Набор `m` и `l` требует довольно неудобной работы пальцами и занимает гораздо больше времени.

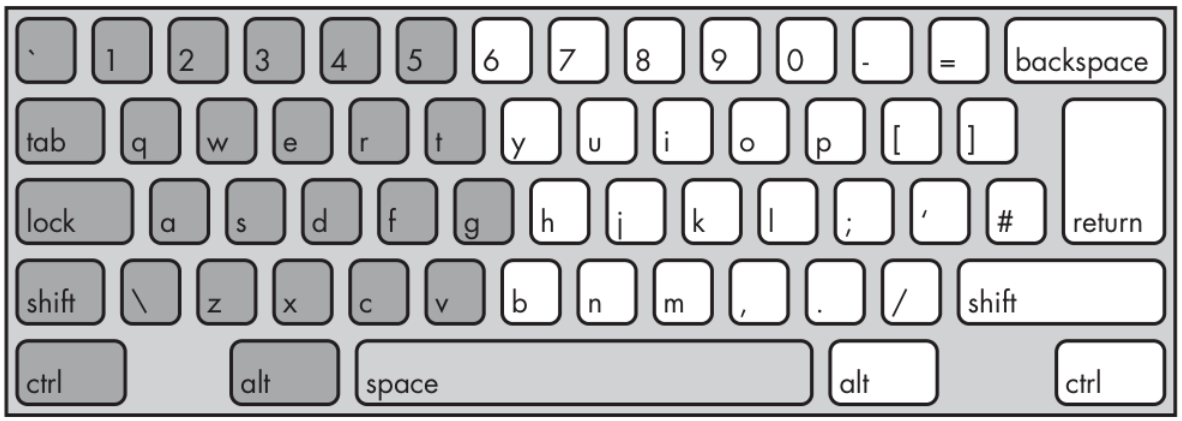


Рисунок 1-2. Обычная область каждой руки. Тёмно-серыми клавишами обычно управляет левая рука, белыми - правая.

Проанализировав несколько образцов, авторы исследования подсчитали, что из временных данных можно извлечь приблизительно 1,2 бита информации на каждую нажатую клавишу. Наблюдая последовательность задержек, можно определить набор вариантов клавиатурного ввода, с наибольшей вероятностью создающих такой шаблон, а значит, упростить угадывание точной последовательности клавиш. Подсчёт долей бита может показаться нелепым, но в действительности это означает, что число вариантов для каждой клавиши сокращается в `2^1,2`, или примерно в 2,40 раза. Для одного обычного нажатия, которое изначально несёт не более 6 битов случайности, множество вариантов сокращается приблизительно с 64 до 26 элементов.

Итоговый эффект - сокращение пространства поиска. Мы видим способ ограничить количество вариантов при попытке угадать набираемые клавиши. Само по себе такое сокращение, возможно, не особенно впечатляет, но следует учесть, что вводимые с клавиатуры данные изначально вряд ли представляют собой случайный мусор. Энтропия английского текста оценивается всего в 0,6-1,3 бита на символ,[8] а значит, для успешного предсказания следующего символа в среднем требуется примерно 1,5-2,5 попытки. Дополнительное сокращение пространства поиска позволяет найти однозначные совпадения со словарными словами почти для всех входных данных.

Чтобы проверить оценки и продемонстрировать проблему на практике, исследователи угадывали нажатия с помощью скрытой марковской модели и алгоритма Витерби. Марковская модель - способ описания дискретной системы, в которой следующее значение зависит только от текущего состояния, а не от предыдущих значений; такую систему называют цепью Маркова. Скрытая марковская модель - разновидность, позволяющая описать систему, где каждое внутреннее состояние порождает наблюдение, но само фактическое состояние неизвестно. Эта модель широко применяется, например, для распознавания речи, где требуется получить чистые данные - текстовое представление произнесённых слов - из их конкретного проявления, дискретизированной формы сигнала.

Авторы заключили, что скрытая марковская модель применима к анализу нажатий. Внутренним состоянием системы они считают сведения о нажатых клавишах, а наблюдением - интервалы между нажатиями.

Можно возразить, что это чрезмерное упрощение, поскольку, особенно в ситуации на рисунке 1-3, зависимость может оказаться глубже.

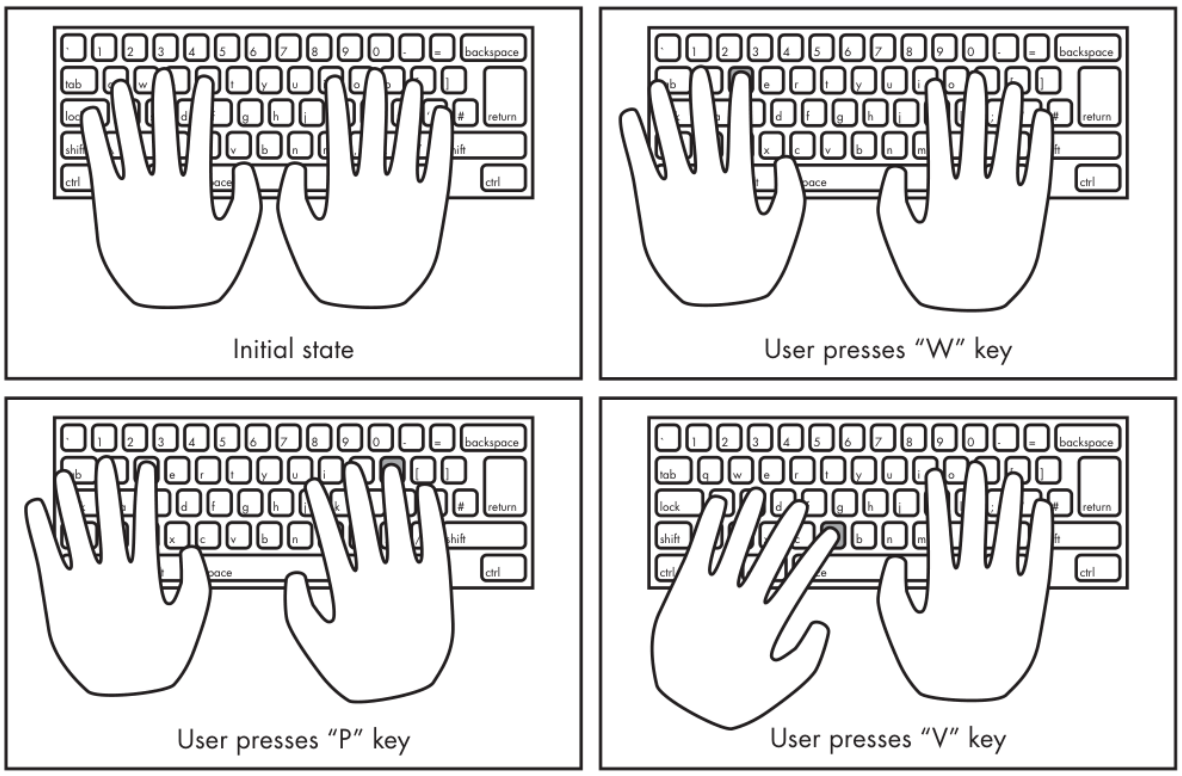


Рисунок 1-3. Необходимость на предыдущем шаге переместить левую руку в другое положение влияет на время перехода `P-V`*. Марковская модель не способна учитывать предыдущее положение руки в сценариях смены рук. Схема показывает исходное состояние, затем нажатие клавиши* `W`*, после него -* `P` *и, наконец,* `V`*.*

Алгоритм Витерби - один из способов решения задач скрытой марковской модели. По последовательности наблюдений он позволяет найти наиболее вероятную последовательность внутренних состояний. В нашем случае по последовательности временных интервалов определяется наиболее вероятная последовательность символов.

Применение алгоритма Витерби в итоге сокращает пространство поиска несловарных восьмисимвольных паролей в 50 раз. При восстановлении набранного английского текста из словарных слов коэффициент, вероятно, будет значительно выше.

Теперь вернёмся к наблюдению за прерываниями. Только что описанное исследование было посвящено неполным сведениям, доступным при слежении за шаблонами трафика Secure Shell, SSH. При наблюдении за прерываниями в распоряжении нападающего оказывается гораздо больше информации. Во-первых, наряду с интервалами между нажатиями доступна длительность каждого нажатия, которая зависит от использованного пальца. Например, указательный палец обычно касается клавиши меньше всего, безымянный, вероятно, медленнее всех и так далее. Это ценная информация, существенно упрощающая приблизительное определение области клавиатуры.

Во-вторых, данные позволяют следить за переходами между руками - моментами, когда первый символ набирается левой рукой, а второй правой, или наоборот. Каждая рука управляется своим полушарием мозга, поэтому при смене рук почти все опытные пользователи нередко нажимают вторую клавишу прежде, чем отпустить первую. Хотя сами по себе события нажатия и отпускания неразличимы, особенно короткий интервал между двумя клавиатурными событиями явно указывает на это явление. В редких случаях, особенно если пользователь торопится, вторая клавиша не просто нажимается до отпускания первой, но даже до её нажатия. Так возникают распространённые опечатки наподобие `teh` вместо `the`.

На рисунке 1-4 показана запись временных характеристик клавиатуры. Пользователь набирает слово `evil`. Средний палец левой руки удерживает `e` среднее время. Затем следует заметный интервал до нажатия `v`, потому что для достижения этой клавиши указательным пальцем приходится переместить всю руку; большой палец использовать нельзя, поскольку мешает пробел. Клавиша `v`, как и `i`, нажимается ненадолго: обе доступны указательному пальцу. Видно и перекрытие - из-за смены рук `i` нажимается до отпускания `v`. Наконец, через некоторое время безымянный палец нажимает `l` - перемещать руку не нужно, - и контакт оказывается довольно долгим.

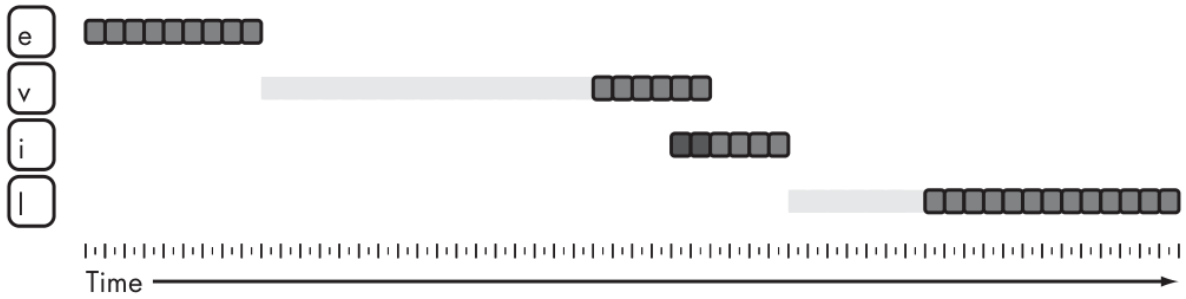


Рисунок 1-4. Временные характеристики нажатий и отпусканий при смене рук. На временной шкале показаны интервалы удержания клавиш `e`*,* `v`*,* `i` *и* `l`*.*

Следовательно, разумно ожидать, что эта атака может добиться гораздо более высокой доли успеха. Большая часть таких сведений отсутствовала в сценарии, рассмотренном в упомянутом выше техническом документе.

Неотложные меры защиты

Теперь, когда мы знаем о возможности перехвата клавиатурного ввода, как ему помешать? Лучший способ - применить отдельный буфер клавиатурной энтропии разумного размера. Буфер очищается, а его содержимое передаётся основной реализации PRNG только после переполнения либо по истечении интервала, значительно превосходящего обычную задержку между нажатиями, то есть не менее нескольких секунд. Так нападающий лишается возможности измерять время.

При таком решении ему доступны лишь сведения двух видов. Первый возникает при очистке после переполнения и сообщает нападающему, что за измеримый период было нажато определённое число клавиш, зависящее от размера буфера, но не раскрывает точные интервалы. Второй появляется при очистке по таймеру и сообщает, что в течение фиксированного промежутка была нажата одна или несколько клавиш, но ничего не говорит ни о количестве событий, ни о точном времени каждого из них. Подобная информация почти бесполезна для временных атак и пригодна лишь для общей статистики клавиатурной активности, которая в большинстве многопользовательских сред угрозы не представляет.

Аппаратный RNG: лучшее решение?

Во многие современные аппаратные платформы встроены физические генераторы случайных чисел, часто называемые TRNG - генераторами истинно случайных чисел. В отличие от сбора информации, которую лишь предполагается трудно предсказать, эти устройства дают более надёжный способ получения действительно непредсказуемых данных и рекомендуются как источник энтропии на всех оснащённых ими машинах. На момент написания книги два популярных решения представляли собой интегральные схемы Intel и VIA.

RNG Intel встроен в наборы микросхем, например i810, и использует традиционную конструкцию из двух генераторов колебаний. Высокочастотный генератор создаёт базовый сигнал, по существу шаблон чередующихся логических состояний `010101010101...`. Второй, низкочастотный генератор работает с номинальной частотой, равной `1/100` частоты первого, но его фактическую частоту модулирует резистор, служащий основным источником энтропии.

Некоторые измеримые характеристики резистора изменяются из-за теплового шума и других случайных эффектов материала. Низкочастотный генератор запускает выборку чередующегося сигнала через теперь уже случайные интервалы - по спадающему фронту своего выхода. После необходимой обработки и «отбеливания» с помощью коррекции фон Неймана сигнал становится доступен внешнему миру. Тщательный анализ конструкции и фактического результата генератора, проведённый Бенджамином Джуном и Полом Кохером из Cryptography Research,[9] показал стабильно высокое качество: оценочно генератор выдаёт 0,999 бита энтропии на каждый выходной бит.

RNG VIA C3 «Nehemiah» основан на немного иной конструкции: он использует набор генераторов колебаний, но не имеет отдельного источника шума наподобие специально подключённого резистора. Вместо этого устройство полагается на внутренний джиттер генераторов, обусловленный рядом внутренних и внешних факторов и дополнительно управляемый настраиваемым параметром смещения.

В этом случае отдельный анализ Cryptography Research[10] показал, что генератор, по-видимому, выдаёт энтропию более низкого качества, чем конкурент: от 0,855 до 0,95 бита на выходной бит. Если считать результат RNG полностью случайным как есть и использовать его в отношении 1:1 для создания ключей или других критически важных задач, это опасно, поскольку объём фактической энтропии соответственно уменьшается. Проблема решается сбором из генератора большего объёма данных, чем требуется, с последующей обработкой безопасной хеш-функцией, например SHA-1, которая устраняет возможное смещение или недостаток энтропии. Это хорошая общая практика защиты от проблем TRNG при условии, что нежелательные эффекты остаются в разумных пределах, то есть каждый бит всё ещё несёт некоторую полезную энтропию.

Некоторые исследователи также предлагали использовать источниками энтропии неспециализированные устройства ввода, например веб-камеры или встроенные микрофоны. Датчики с зарядовой связью, CCD, в цифровых камерах склонны создавать пиксельный шум, а чрезмерно усиленный сигнал микрофона по существу становится хорошим источником случайного шума. Однако универсального способа настроить такой генератор нет из-за различий в схемах распространённых мультимедийных устройств разных производителей, поэтому качество полученных таким способом «случайных» чисел гарантировать нельзя. Некоторые устройства действительно улавливают кажущиеся случайными, но полностью предсказуемые радиопомехи или определённые сигналы внутри схемы. Кроме того, некоторые устройства, особенно датчики CCD, создают статические шумовые шаблоны. Такой шум выглядит случайным, но меняется медленно, и полагаться на него может быть опасно.

Пища для размышлений

Я решил не углубляться в несколько интересных идей, но они могут послужить ценным источником вдохновения для дальнейших исследований.

Удалённые временные атаки

Теоретически временную атаку на PRNG можно провести по сети. Некоторые службы с поддержкой криптографии применяют симметричное шифрование. После установления более медленного асимметричного потока через инфраструктуру открытых ключей и проверки обеих сторон создаётся симметричный сеансовый ключ, после чего оба узла переключаются на более быстрый симметричный вариант.

• Возможно, время нажатий удастся измерить, если заставить приложение истощить системный пул энтропии до состояния, когда для нового сеансового ключа не хватает лишь небольшой доли энтропии. Тогда приложение отложит создание симметричного ключа до появления количества энтропии, достаточного для оставшейся части ключа; среди прочего это может произойти при следующем нажатии или отпускании клавиши.
• По моему мнению, такая атака скорее сработает в лаборатории, чем в реальном практическом применении. Мой технический рецензент не согласен с этим скептицизмом, поэтому считайте сказанное лишь мнением. В интересном анализе Виргинского университета первоначальное исследование временных характеристик SSH, рассмотренное в упомянутой выше работе, критиковалось на том основании, что сетевого джиттера достаточно, чтобы временные данные стали непригодными. Однако следует отметить: если конкретное действие повторяется - например, при каждом входе вводится один и тот же пароль, - случайные колебания производительности сети вполне могут усредниться.[11]

Использование системной диагностики

Некоторые системы позволяют получать сведения о нажатиях и другие временные данные более удобным способом. После публикации моего исследования временных характеристик PRNG мне указали, что Linux предоставляет интерфейс `/proc/interrupts`, который выводит сводную статистику прерываний, предназначенную для получения полезных данных о производительности. Наблюдая изменения счётчика прерываний IRQ 1, можно получить те же временные сведения, что и через PRNG, причём уже очищенные от возможных включений дисковой и сетевой активности. В результате возникает утечка конфиденциальной информации, подобная рассмотренной выше.

Воспроизводимая непредсказуемость

Стоит рассмотреть и другие проблемы, связанные с самой реализацией PRNG. Покупка партии одинакового оборудования и установка одной системы на каждое устройство - обычная практика, которая может создать проблему для серверов с низкой активностью консоли. Риск возникает и при зеркальном копировании установки специализированными средствами дублирования с последующим развёртыванием образа на множестве серверов. Во всех таких случаях системы могут оставаться с низкой реальной энтропией, возможно, несколько дольше допустимого.

Сноски

[1] [назад] Простое число - положительное целое число, которое делится только на 1 и на само себя.
[2] [назад] Число, взаимно простое с x, также называют относительно простым с x; у него нет общих с x множителей, кроме 1 и -1. Их наибольший общий делитель равен 1.
[3] [назад] Для полноты следует отметить, что криптография с открытым ключом, применяемая без дополнительных мер, помимо прочего уязвима для атак «человек посередине». При такой атаке злоумышленник выдаёт себя за одну из конечных сторон и предоставляет собственный поддельный открытый ключ, чтобы получить возможность перехватывать связь. Для защиты нужны дополнительные способы проверки подлинности ключа: либо организация безопасного обмена, либо создание центрального органа, выпускающего или удостоверяющего ключи, то есть инфраструктуры открытых ключей, PKI.

[4] [назад] Операция взятия остатка возвращает остаток от целочисленного деления двух чисел. Например, при делении 7 на 3 целый результат равен 2, а остаток равен 1: `7 = 2 * 3 + 1`; следовательно, 7 по модулю 3 равно 1.
[5] [назад] Во многих архитектурах PIC необходимо вручную сообщить, что прерывание обработано и больше не должно блокировать последующие прерывания. Для этого служит код окончания прерывания, End of Interrupt, EOI.

Глава 2. Лишние усилия не остаются незамеченными

Где мы узнаём, как построить деревянный компьютер и как получать сведения, наблюдая за работой настоящего компьютера.

Введённые вами данные теперь в надёжных руках выбранного приложения. Программа не спеша решит, что делать с информацией, как её истолковать и какие действия предпринять дальше.

В этой главе мы подробно рассмотрим низкоуровневые механизмы обработки данных и исследуем ловушки, способные скрываться глубоко под теплоотводом процессора. Особое внимание уделим сведениям, которые можно вывести, просто наблюдая за тем, как машина выполняет заданные программы и сколько времени тратит на определённые задачи. В качестве дополнительной награды мы ещё и построим полностью работоспособный деревянный компьютер.

Наследие Буля

Чтобы понять устройство процессора, придётся вернуться во времена, когда о процессорах ещё никто не мечтал. Всё началось довольно невинно в XIX веке, когда математик-самоучка Джордж Буль (1815-1864) разработал простую систему двоичной алгебры как основу для понимания и моделирования формального исчисления. Его подход свёл фундаментальные понятия логики к трём простым алгебраическим операциям, применимым к элементам, которые представляют два противоположных состояния - истину и ложь. Вот эти операции:

• Оператор дизъюнкции `OR`. Результат истинен, когда истинен хотя бы один операнд[1] [2].
• Оператор конъюнкции `AND`. Результат истинен только тогда, когда истинны все операнды.
• Оператор дополнения, или отрицания, `NOT`. Результат истинен, когда единственный операнд ложен.

Несмотря на простоту, булева алгебра оказалась мощным средством решения логических задач и некоторых других математических проблем. В конечном счёте благодаря ей многие смелые провидцы смогли мечтать об умных аналитических машинах, которым однажды предстояло изменить нашу повседневную жизнь.

Сегодня булева логика редко остаётся загадкой для опытного пользователя компьютера, но путь от набора тривиальных операций к современному компьютеру часто остаётся неясным. Исследование этого пути мы начнём с попытки уловить сущность модели в простейшем виде.

К универсальному оператору

Путь к простоте нередко пролегает через на первый взгляд ненужную сложность - и этот случай не исключение. Для начала нужно обратиться к работе другого математика XIX века, Огастеса де Моргана (1806-1871). Закон де Моргана гласит: «Дополнение дизъюнкции есть конъюнкция дополнений». Это печально известное упражнение по затуманиванию тривиальных понятий имеет глубокие последствия для булевой логики, а в итоге - и для устройства цифровых схем.

Обычными словами закон де Моргана можно выразить так: если одно или оба из двух условий не выполнены, утверждение, что выполнены оба условия - иначе говоря, имеет место их конъюнкция, - также будет ложным. И наоборот.

Из закона следует, что `NOT OR(a, b)` логически эквивалентно `AND(NOT a, NOT b)`. Рассмотрим пример из жизни, где `a` и `b` означают следующее:

```
a = «Боб любит молоко»
b = «Боб любит яблоки»
```

Теперь две стороны уравнения де Моргана можно записать так:

```
OR(NOT a, NOT b) <=> Боб НЕ любит молоко ИЛИ НЕ любит яблоки
NOT AND(a, b)     <=> НЕверно, что Боб любит и молоко, И яблоки
```

Оба выражения функционально эквивалентны. Если верно, что Боб не любит либо молоко, либо яблоки, первое выражение истинно; тогда верно и то, что он не любит оба продукта, а значит, истинно и второе выражение. В обратной ситуации результаты тоже совпадают: если неверно, что Боб не любит хотя бы один из вариантов, значит, он любит оба, а первое выражение ложно. В таком случае неверно и то, что он не любит оба продукта, поэтому второе выражение также ложно.

Де Морган в деле

Чтобы оценивать логические выражения не только интуитивно и не ограничиваться расплывчатыми рассуждениями, полезно строить так называемые таблицы истинности, в которых показаны все результаты для всех возможных сочетаний истинных и ложных операндов.

Следующие две таблицы представляют выражения из предыдущего примера. В каждой есть столбцы для обоих операндов и соответствующие результаты всех возможных сочетаний истины и лжи. В первой строке видно, что первые два столбца - оба операнда `NOT AND(a, b)` - ложны. Поэтому `AND(a, b)` также ложно, а `NOT AND(a, b)` истинно. Результат указан в третьем столбце.

`NOT AND(a, b)`*: результат* `AND` *с отрицанием*

Операнд 1 (a**)**	**Операнд 2 (**b**)**	**Результат**
ЛОЖЬ	ЛОЖЬ	ИСТИНА
ЛОЖЬ	ИСТИНА	ИСТИНА
ИСТИНА	ЛОЖЬ	ИСТИНА
ИСТИНА	ИСТИНА	ЛОЖЬ

`OR(NOT a, NOT b)`*:* `OR` *с отрицанием операндов*

Операнд 1	**Операнд 2**	**Результат**
ЛОЖЬ	ЛОЖЬ	ИСТИНА
ЛОЖЬ	ИСТИНА	ИСТИНА
ИСТИНА	ЛОЖЬ	ИСТИНА
ИСТИНА	ИСТИНА	ЛОЖЬ

Как видите, оба выражения ведут себя одинаково.

Но какое дело разработчикам компьютеров до пищевых предпочтений Боба? В контексте булевых операторов закон де Моргана означает, что набор базовых операций булевой алгебры в действительности частично избыточен: сочетания NOT с любым из двух остальных операторов, OR или AND, всегда достаточно, чтобы синтезировать оставшийся. Например:

```
OR(a, b)  <=> NOT AND(NOT a, NOT b)
AND(a, b) <=> NOT OR(NOT a, NOT b)
```

Это понимание сокращает набор операторов до двух, но булеву систему можно упростить ещё сильнее.

Удобство - необходимость

Несколько дополнительных операторов не являются обязательными для реализации булевой логики, но дополняют существующий набор операций. Дополнительные операторы NAND и NOR истинны только тогда, когда соответственно ложны AND и OR:

```
NAND(a, b) <=> NOT AND(a, b) <=> OR(NOT a, NOT b)
NOR(a, b)  <=> NOT OR(a, b)  <=> AND(NOT a, NOT b)
```

Новые функции не сложнее AND и OR. У каждой есть таблица истинности из четырёх состояний - четырёх строк, - поэтому её значение определяется с теми же усилиями.

> **Примечание.** NOR и NAND не входят в базовый набор операторов, поскольку ни один из них не соответствует распространённому основному типу логической связи между предложениями и не имеет атомарного представления в обычном языке.

Итак, я только что ввёл набор новых операторов, производных от уже существующих и, на первый взгляд, дающих лишь сомнительное удобство тем, кто хочет формально записывать ещё более странные логические зависимости и задачи. Зачем?

Одно лишь введение NAND или NOR позволяет полностью отказаться от AND, OR и NOT. Это приближает нас к простоте и даёт возможность описать всю систему булевой алгебры меньшим количеством элементов и операторов.

Важность отрицательных вспомогательных операторов состоит в том, что на основе любого из них можно построить полную систему булевой алгебры. Например, все базовые операторы можно создать с помощью NAND, как показано ниже. Каким образом? Следующие пары выражений совершенно очевидно эквивалентны:

```
NOT a     <=> NAND(a, a)
AND(a, b) <=> NOT NAND(a, b)
          <=> NAND(NAND(a, b), NAND(a, b))
OR(a, b)  <=> NAND(NOT a, NOT b)
          <=> NAND(NAND(a, a), NAND(b, b))
```

Если же мы предпочитаем опираться исключительно на NOR, а не на NAND, можно записать:

```
NOT a     <=> NOR(a, a)
OR(a, b)  <=> NOT NOR(a, b)
          <=> NOR(NOR(a, b), NOR(a, b))
AND(a, b) <=> NOR(NOT a, NOT b)
          <=> NOR(NOR(a, a), NOR(b, b))
```

Принимая сложность

Трудно поверить, что суть всех вычислений можно вместить в один универсальный логический оператор. Большинство сложных алгоритмов, передовые вычисления, новейшие игры и просмотр Интернета можно реализовать массивом простых схем, соответствующих одной из следующих таблиц истинности, которые преобразуют входные сигналы в выходные.

Таблица состояний `NAND`

Операнд 1	**Операнд 2**	**Результат**
ЛОЖЬ	ЛОЖЬ	ИСТИНА
ЛОЖЬ	ИСТИНА	ИСТИНА
ИСТИНА	ЛОЖЬ	ИСТИНА
ИСТИНА	ИСТИНА	ЛОЖЬ

Таблица состояний `NOR`

Операнд 1	**Операнд 2**	**Результат**
ЛОЖЬ	ЛОЖЬ	ИСТИНА
ЛОЖЬ	ИСТИНА	ЛОЖЬ
ИСТИНА	ЛОЖЬ	ЛОЖЬ
ИСТИНА	ИСТИНА	ЛОЖЬ

И всё же кажется, будто мы топчемся на месте... Как этот тривиальный набор зависимостей позволяет построить устройство, способное решать сложные задачи - например, тактично отклонять вашу кредитную заявку? И какое отношение теория, основанная на состояниях «истина» и «ложь», имеет к цифровым схемам?

К материальному миру

В механизме Буля нет ничего сложного: ему нужны два противоположных логических состояния - «истина» и «ложь», 0 и 1, «голубой» и «пурпурный», 999 и 999 1/2. Фактический смысл, физическое представление и носитель несущественны; важна лишь произвольно выбранная договорённость, которая сопоставляет определённые состояния носителя конкретному набору логических значений.

Привычные нам компьютеры используют два разных уровня напряжения в электронной схеме и истолковывают их как значения, которые разработчики называют 0 и 1. Эти значения, передаваемые по электрической цепи, представляют две цифры двоичной системы. Но ничто не мешает передавать данные практически любым способом: потоком воды, химическими реакциями, дымовыми сигналами или вращающими моментами в наборе мастерски изготовленных деревянных шестерён. Независимо от носителя информация остаётся той же.

Ключ к воплощению булевой логики в физическом мире прост, если мы договорились о физическом представлении логических значений. Затем остаётся лишь найти способ расположить набор компонентов так, чтобы они манипулировали этими значениями для выполнения любой задачи, которую мы хотим поручить компьютеру, но об этом позднее. Сначала попробуем понять, как управлять сигналами и создавать реальные логические устройства, обычно называемые

вентилями. В нашем случае - деревянными.

Неэлектрический компьютер

Может показаться, что от набора теоретических операций, порождённых миром чистой математики, трудно перейти к устройству, способному управлять потоком воды, вращающими моментами или электрическими сигналами, подражая одному из логических операторов. Но это не так.

На рисунке 2-1 показан простейший механизм из шестерён, который реализует функцию `NOR` с помощью логики, основанной на вращающем моменте. Бездействующее «выходное» колесо представляет состояние 0; когда к нему приложен вращающий момент, его состояние равно 1. Устройство передаёт вращающий момент от внешнего источника к выходу только в том случае, если момент не приложен ни к одному из двух управляющих «входных» колёс. Теоретически внешний источник энергии не нужен и конструкцию можно упростить; на практике трение и другие трудности сильно осложнили бы создание более сложного набора полностью автономных вентилей.

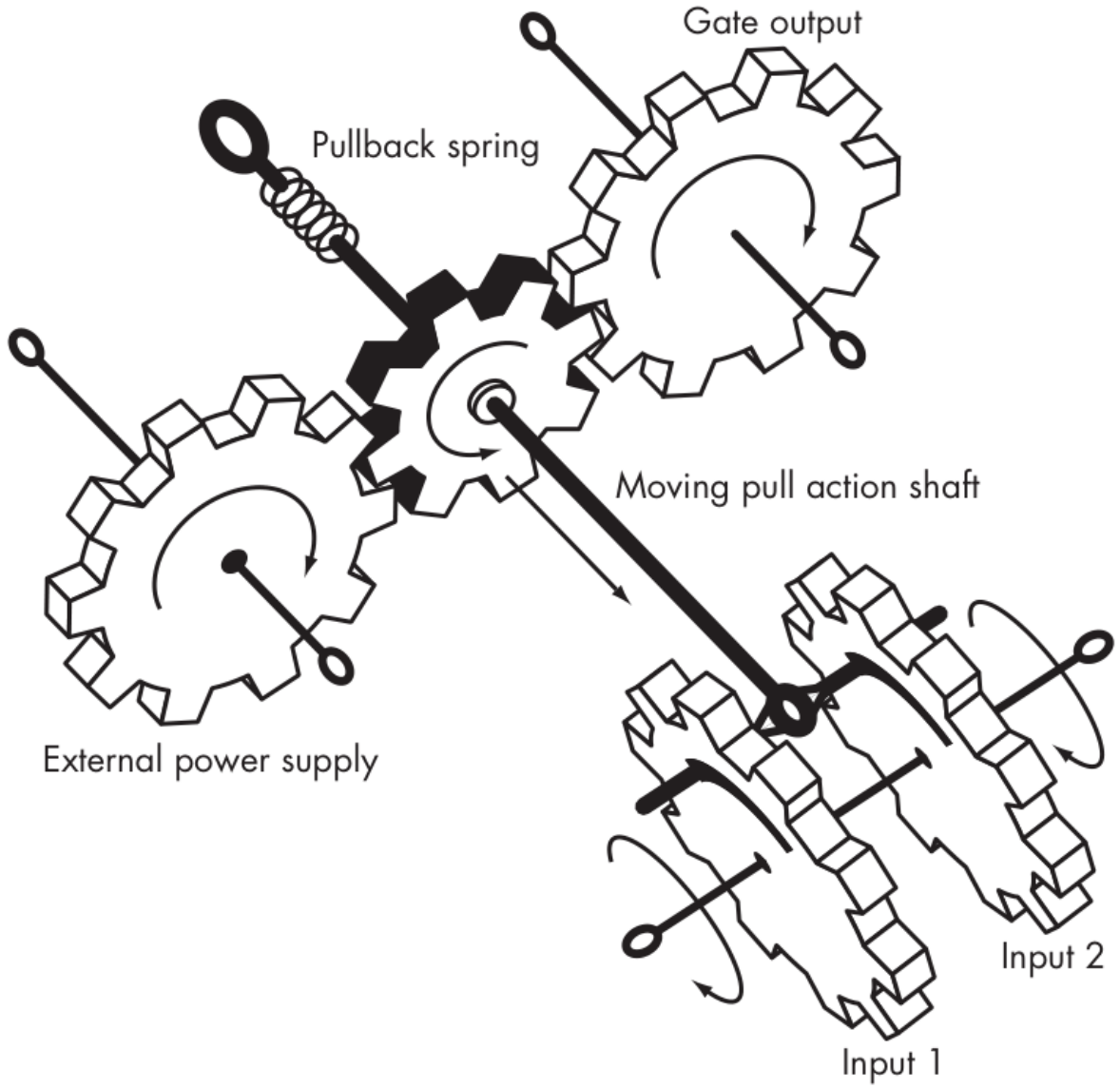


Рисунок 2-1. Конструкция механического вентиля NOR. *На схеме обозначены выход вентиля, возвратная пружина, внешний источник энергии, подвижный вал тяги, вход 1 и вход 2.*

Если приложить вращающий момент к одному или обоим входам, крошечная соединительная шестерня отойдёт, а «выходная» шестерня остановится. Когда входы переходят в состояние покоя, пружина возвращает соединительную шестерню на место. Таблица истинности такого устройства в точности соответствует NOR.

Как вы помните, одного NOR или NAND достаточно для реализации любого оператора булевой логики. Возможность создавать другие операторы без объединения вентилей NAND и NOR сделала бы устройство меньше и эффективнее, но для работы она не обязательна.

Если отвлечься от неприятной мелочи - необходимости заставить все вентили согласованно работать привычным для нас образом, - можно заключить, что компьютеры способны строиться практически на любой технологии.[3]

Немного более популярная конструкция компьютера

Хотя компьютерный бум последних десятилетий вырос из гениального транзистора, наша зависимость от него не объясняется какой-то волшебной ценностью или уникальным качеством. Просто на данный момент это самая доступная, удобная и эффективная конструкция.

В отличие от, возможно, куда более совершенной машины на деревянных шестернях, используемые нами электронные компьютеры передают электрические сигналы с помощью транзисторов. Это крошечные устройства, которые позволяют току течь в одном направлении между двумя узлами - точками подключения, - когда на третий узел подано напряжение. Транзисторы хорошо поддаются миниатюризации, потребляют мало энергии, надёжны и дёшевы.

Логические вентили

Транзистор прост. Настолько прост, что один он не способен реализовать сколько-нибудь содержательную булеву логику. Но если правильно расположить транзисторы в логических вентилях, с их помощью легко выполнять все базовые и дополнительные операции булевой алгебры.

Вентиль AND можно реализовать, последовательно соединив два транзистора: напряжение попадёт на выход только тогда, когда оба имеют низкое сопротивление, то есть «включены». Каждый транзистор управляется - включается - отдельной входной линией. В нормальном состоянии выход «подтянут вниз» резистором, поэтому на нём присутствует напряжение земли 0, означающее «ложь». Но когда оба транзистора включатся и пропустят небольшой ток, напряжение поднимется выше 0.

Вентиль OR реализуется параллельным соединением транзисторов: достаточно включения любого из них, чтобы на выходе появилось ненулевое напряжение, обозначающее «истину».

Последний базовый вентиль, NOT, создаётся из одного транзистора и резистора. В состоянии покоя его выход равен 1, поскольку подтянут резистором вверх, а при открытии транзистора подтягивается вниз, к 0.

На рисунке 2-2 показаны конструкции и условные обозначения трёх основных транзисторных вентилей: AND, OR и NOT.

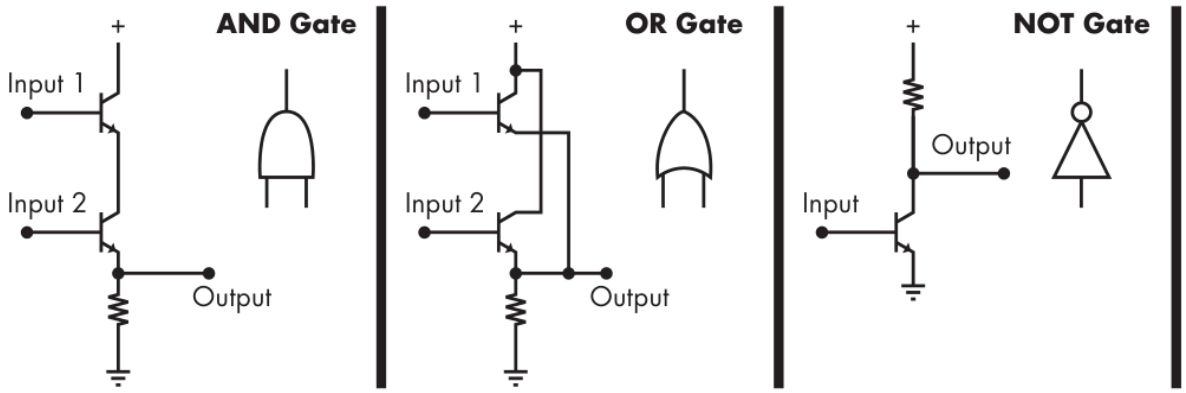


Рисунок 2-2. Логические вентили на транзисторах: конструкции и условные обозначения. Для AND *показаны вход 1, вход 2 и выход; для* OR *- вход 1, вход 2 и выход; для* NOT *- вход и выход.*

> **Примечание.** Можно заметить, что вентили AND и OR превращаются в NAND и NOR без дополнительных компонентов. Достаточно применить конструкцию из схемы вентиля NOT: переместить резистор и «точку выхода» к напряжению питания, тем самым инвертировав логику выхода.

Мы дошли до стадии, на которой из транзисторов можно собрать один из универсальных вентилей. Но сколько бы вентилей мы ни построили, до настоящих вычислений ещё далеко.

Всё предыдущее рассуждение замечательно, но что превращает булеву логику из мощного инструмента для решения головоломок о рационе Боба во что-то большее?

От логических операторов к вычислениям

Объединение тривиальных операций булевой логики даёт ряд удивительных возможностей, например выполнение арифметических действий над двоичными представлениями чисел. Здесь начинается самое интересное.

Например, набор вентилей XOR и AND позволяет увеличить входное число на 1, а это первый шаг на пути к сложению. Основанная на такой идее схема счётчика показана на рисунке 2-3.

Ага, новый термин! XOR - ещё один «удобный» оператор булевой логики, истинный только тогда, когда истинен ровно один из операндов. В этом отношении он ближе к обычному значению английского слова *or*. Оператор XOR часто упрощает запись, однако его легко реализовать и другими средствами, сочетая AND, NOT и OR. Определяется он так:

```
XOR(a, b) <=> AND(OR(a, b), NOT AND(a, b))
```

Вернёмся к нашей схеме. Что она умеет? Устройство на рисунке 2-3 принимает число в двоичной записи. В этом примере оно ограничено тремя битами, хотя конструкцию легко расширить для большего числа входов.

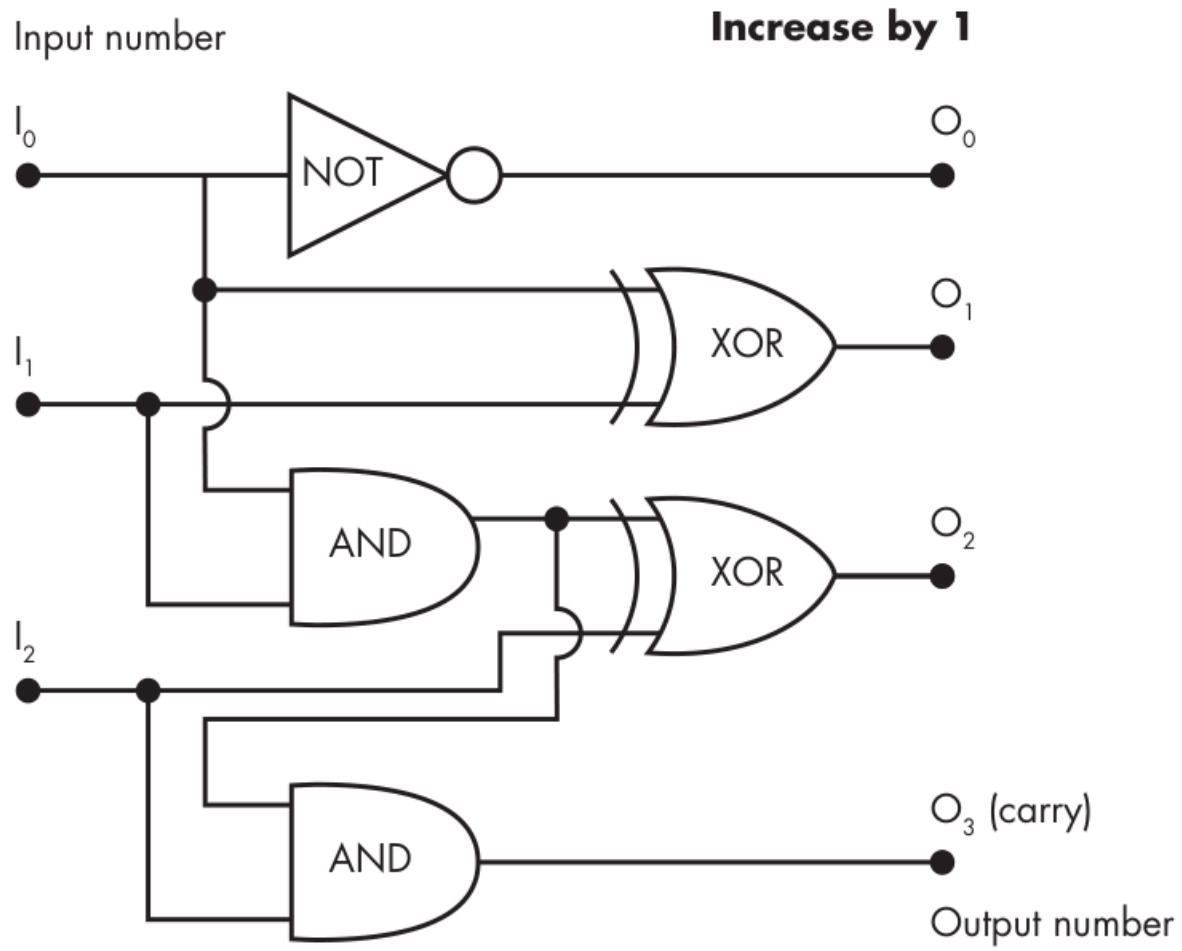


Рисунок 2-3. Простейшая схема увеличения на единицу. Входное число `I0`*,* `I1`*,* `I2` *проходит через вентили* `NOT`*,* `AND` *и* `XOR`*; результат образуют* `O0`*,* `O1`*,* `O2` *и* `O3`*, последний является переносом.*

Это простое вычислительное устройство работает так же, как человек складывает десятичные числа на бумаге: движется справа налево и при необходимости переносит значение в следующий разряд. Единственное настоящее отличие - двоичная система.

Посмотрим, как это происходит. У нас есть двоичное число, записанное в строку. Мы хотим увеличить его на единицу и, как при сложении десятичных чисел, начинаем с крайней правой цифры.

Там находится двоичная цифра. При увеличении двоичной цифры на 1 возможны лишь два результата: если входная цифра равна 0, выход равен 1, то есть `0 + 1 = 1`; иначе выход равен 0, а 1 нужно перенести в следующий разряд, поскольку `1 + 1 = 10`. Иначе говоря, выполняются два действия: создаётся результат, равный отрицанию входа - 1 для 0 и 0 для 1, - а если входная цифра равна 1, это нужно запомнить и учесть позднее.

Именно это и делает схема. Для первого входа, `I0`, самый верхний вентиль обрабатывает значение: инвертирует его и выдаёт на выходе `O0`, а само входное значение передаёт вентилям, отвечающим за следующий разряд, `O1`.

```
O0 = NOT I0
C0 = I0
```

Мы увеличили число на единицу; если переноса из предыдущего разряда нет, в остальных разрядах делать нечего. При отсутствии переноса `O1` должен повторять `I1`. Если значение переноса есть, нужно поступить так же, как при добавлении 1 к предыдущему разряду: инвертировать выход и при необходимости передать значение дальше.

Начиная с этого момента каждый последующий выход On, где n > 0, либо непосредственно копируется из In, если переноса из предыдущего разряда нет, либо увеличивается на 1, что снова сводится к инверсии, из-за добавления бита переноса. Поэтому, если In равно 1, перенос из этого разряда Cn также будет равен 1, а On - 0, поскольку в двоичной системе 1 + 1 = 10. Можно заметить, что фактический выход в позиции n - просто результат XOR входного значения в позиции n и бита переноса из разряда n - 1. Следовательно, схема получает On, выполняя XOR перенесённого бита Cn-1 и значения In, а затем выполняет AND переноса из On-1 и In, чтобы определить, нужен ли перенос в следующий разряд:

```
On = XOR(In, Cn-1)
Cn = AND(In, Cn-1)
```

Рассмотрим пример. Требуется увеличить входное значение 3, то есть 011 в двоичной записи, на 1. Входы таковы:

```
I0 = 1
I1 = 1
I2 = 0
```

Схема получает O0, инвертируя I0, поэтому O0 = 0. Поскольку I0 было ненулевым, в следующий разряд также передаётся перенос. В следующем разряде вентиль XOR задаёт O1 = 0: хотя I1 равно 1, из предыдущего разряда пришёл перенос, а 1 + 1 = 10. И снова возникает перенос в следующий разряд.

Ещё в одном разряде I2 = 0, но вентиль AND указывает на перенос из предыдущей строки, поскольку оба предыдущих входа были равны 1. Поэтому результат равен 1. В последний разряд переноса не будет. Получаем:

```
O0 = 0
O1 = 0
O2 = 1
O0 = 0
```

> **Примечание переводчика.** В последней строке исходника повторно указано O0; по схеме и двоичному результату здесь, вероятно, подразумевалось O3.

Или 0100, что, по счастливой случайности, после преобразования в десятичную систему равно 4.

И вуаля - вот операция +1, записанная в двоичном виде.

> **Примечание.** Мы только что выразили первую вычислительную задачу средствами булевой алгебры. Может возникнуть желание расширить конструкцию, чтобы она складывала два произвольных числа, а не одно число с единицей. Тем не менее именно с подобных базовых схем в значительной мере начинаются и заканчиваются вычисления.

Цифровая арифметическая схема пропускает входные данные через массив хитро расположенных логических вентилей, которые складывают, вычитают, умножают или выполняют иные простые преобразования массива битов. Волшебства здесь совсем немного.

До сих пор я объяснял, как кремниевые микросхемы или искусно обработанная древесина могут выполнять определённые фиксированные базовые операции, например целочисленную арифметику. Но в этой картине чего-то не хватает: текстовые редакторы, игры и одноранговые программы не зашиты в чудовищно сложный массив вентилей внутри CPU. Где же хранится программное обеспечение?

От электронного таймера для яиц к компьютеру

Настоящая ценность компьютера - в возможности запрограммировать его на определённое поведение: заставить выполнить последовательность программных команд в соответствии с некоторым планом.

На рисунке 2-4 показан следующий шаг к гибкой машине, способной выполнять не одну-единственную задачу, жёстко заданную соединениями: хранение данных и память. Здесь изображена ячейка памяти, известная как триггер. У неё есть две управляющие линии - «установка» и «сброс». Когда обе опущены, вентиль сохраняет текущее состояние благодаря обратной связи между своим выходом и входом вентиля `OR`. Предыдущий результат `OR` проходит через вентиль `AND`, поскольку его другая линия установлена в 1 - отрицание «сброса», - а затем снова через `OR`, поскольку на другом входе там 0 - «установка». Пока на вентили подаётся питание, состояние выхода сохраняется.

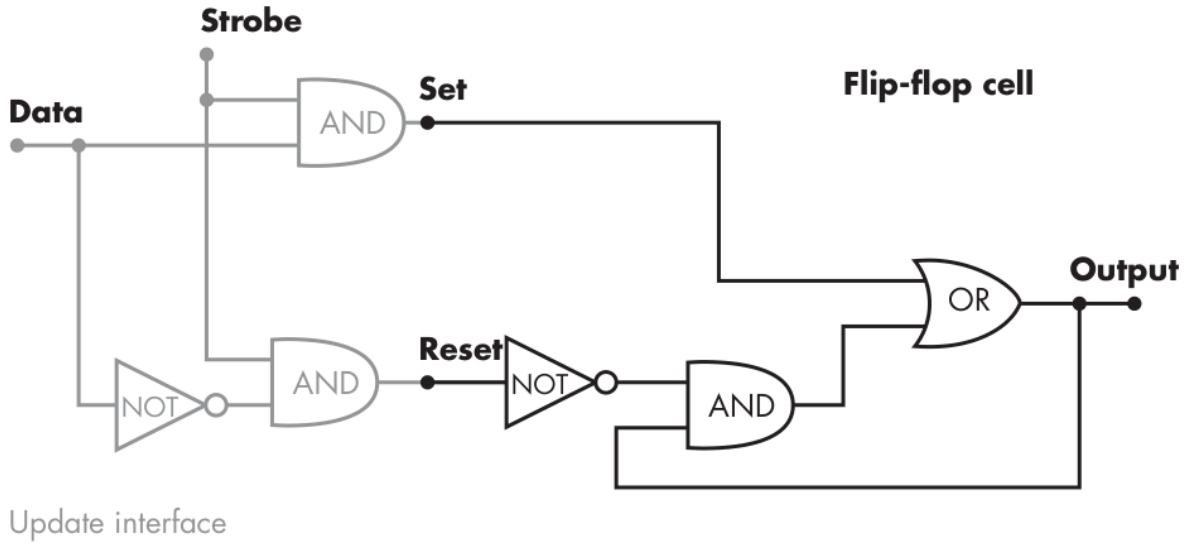


Рисунок 2-4. Триггерная память с практичным интерфейсом. Схема состоит из интерфейса обновления и ячейки триггера; обозначены данные, строб, установка, сброс и выход, а также вентили `NOT`*,* `AND` *и* `OR`*.*

Когда линия «установка» поднимается, вентиль `OR` принудительно выдаёт 1 и сохраняет это значение после её опускания. Когда поднимается линия «сброс», вентиль `AND` принудительно выдаёт 0 и разрывает петлю обратной связи, заставляя схему вывести 0. После опускания «сброса» выход остаётся нулевым. Если подняты обе управляющие линии, схема становится нестабильной - не самое приятное явление, особенно если компьютер механический.

Таблица истинности этой конструкции выглядит так, где `V` означает произвольное логическое значение:

Таблица истинности триггера

Установка	**Сброс**	`Q(t-1)`	`Q(t)`
0	0	V	V
1	0	-	1
0	1	-	0
1	1	-	нестабильно

Более практичная разновидность триггерной схемы со встроенным «интерфейсом обновления», показанным в левой части рисунка 2-4, использует два вентиля AND и один NOT. Состояние входной линии захватывается - считывается и удерживается - при появлении внешнего управляющего сигнала «строб». Такая конструкция устраняет нестабильные сочетания входов и делает память удобнее для хранения информации.

Улучшенная таблица истинности триггера

Вход	**Строб**	Q(t-1)	Q(t)
-	0	V	V
S	1	-	S

Эта простейшая конфигурация вентилей обладает важным свойством: она хранит данные. Одна ячейка сохраняет лишь один бит, но объединение нескольких триггеров увеличивает ёмкость. Современные конструкции памяти различаются, однако значение этой функции остаётся прежним: она позволяет выполнять программы. Но каким образом?

В базовой конструкции микросхема хранит специальное значение, обычно называемое указателем команд, во внутренней защёлке памяти на кристалле - регистре из нескольких триггеров. Распространённые компьютеры работают синхронно: все процессы задаются генератором тактового сигнала высокой частоты, поэтому на каждом такте указатель выбирает ячейку основной памяти. Полученные таким образом управляющие данные выбирают и включают нужную арифметическую схему для обработки входных данных.

При одних управляющих данных наша гипотетическая микросхема выполняет сложение, при других - операцию ввода-вывода. После извлечения каждой порции управляющих данных, то есть каждой машинной команды, микросхема должна продвинуть внутренний указатель команд, чтобы на следующем такте прочесть очередную команду. Благодаря этой функции микросхема выполняет последовательность машинных команд - программу.

Теперь пора выяснить, какие операции должна реализовывать микросхема, чтобы ею можно было пользоваться.

Тьюринг и сложность системы команд

Как выясняется, процессор не обязан быть сложным. Набор команд, необходимый микросхеме для выполнения практически любой задачи, удивительно мал. Тезис Чёрча - Тьюринга утверждает, что всякое вычисление реального мира может быть выполнено машиной Тьюринга - примитивной моделью компьютера. Названная в честь изобретателя машина Тьюринга представляет собой простейшее устройство, работающее с потенциально бесконечной лентой из отдельных ячеек - гипотетическим, чисто абстрактным носителем. В каждой ячейке хранится один символ из «алфавита» машины, то есть конечного упорядоченного множества возможных значений. С человеческими алфавитами он не имеет совершенно ничего общего; такое название выбрали, чтобы внести здоровую долю путаницы в умы непосвящённых.

Устройство также снабжено внутренним регистром, способным хранить конечное число столь же внутренних состояний. Машина Тьюринга начинает работу в определённой позиции ленты и заданном состоянии, после чего читает символ из ячейки. С каждым автоматом связан набор правил перехода: они описывают, как изменить внутреннее состояние, что записать на ленту в зависимости от ситуации после чтения и как при необходимости сдвинуть ленту на одну ячейку в любую сторону. Такой набор переходов задаёт правила вычисления следующего состояния

системы по её текущим характеристикам. Обычно правила записывают таблицей переходов состояний.

Таблица переходов состояний

Текущее `C(t)`	**Текущее** `S(t)`	**Новое** `C(t+1)`	**Новое** `S(t+1)`	**Действие**
0	`S0`	1	`S1`	-
1	`S0`	0	`S0`	ВЛЕВО

Таблица сообщает: если текущее значение ячейки под машиной равно 0, а внутреннее состояние машины в этот момент - `S0`, устройство изменит состояние `C` на 1, внутреннее состояние - на `S1` и не станет перемещать считывающую головку.

На рисунке 2-5 приведён пример машины Тьюринга, расположенной над ячейкой `C` во внутреннем состоянии `S`.

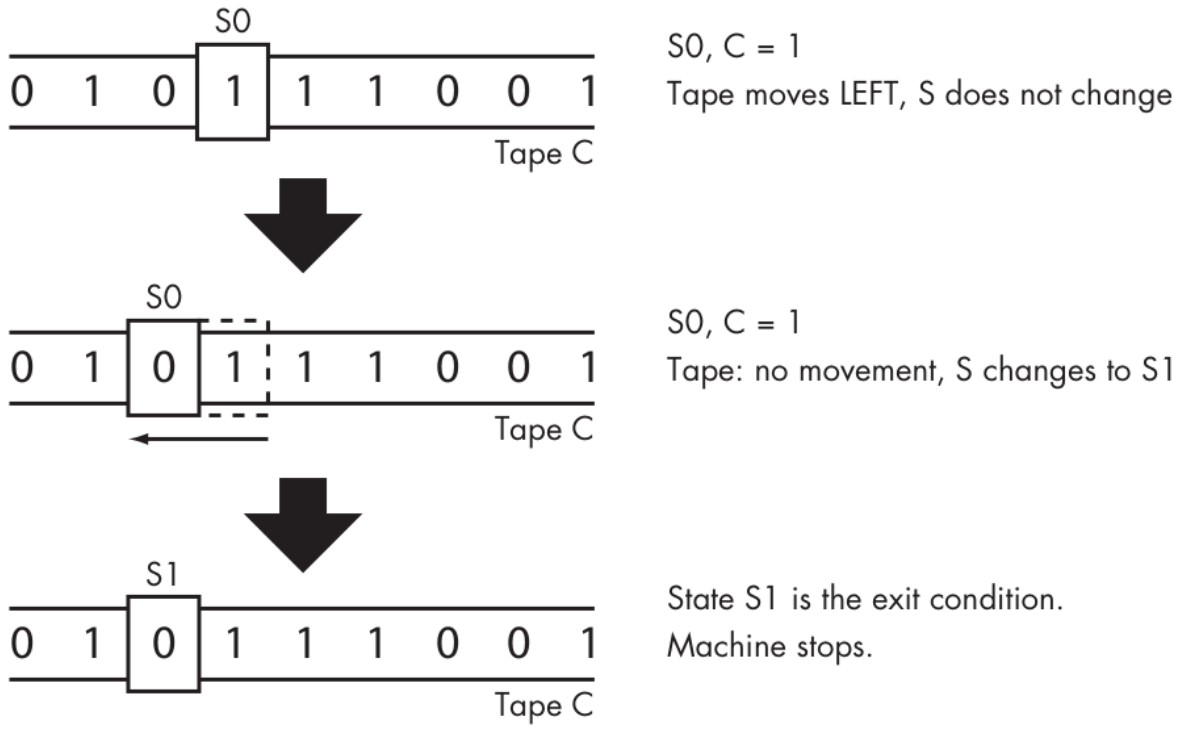


Рисунок 2-5. Пример стадий выполнения машины Тьюринга. В исходном состоянии `S0` лента содержит `010111001`. Если `C = 1`, лента сдвигается влево, а `S` не меняется. Когда `C = 1` инвертируется в 0, лента не движется, а `S` меняется на `S1`. Состояние `S1` служит условием выхода, и машина останавливается.

Разберём процесс. Как видно на рисунке 2-5, алфавит машины состоит из двух символов, 0 и 1, а внутренних состояний два - `S0` и `S1`. Работа начинается с `S0`; начальные условия можно выбирать произвольно, и здесь оно выбрано без особой причины. Расположившись в конце - у младшего бита - двоичного числа на ленте, `C0`, машина следует такой логике:

• Если символ под головкой равен 0, он меняется на 1, а состояние машины в соответствии с первым правилом предыдущей таблицы меняется на `S1`. Поскольку для `S1` правила перехода нет, на следующем такте машина останавливается.
• Если прочитанный под головкой символ равен 1, он меняется на 0, а состояние остаётся прежним. Согласно второму правилу перехода машина также сдвигает считывающую головку по ленте влево.

Затем весь процесс повторяется с новой позиции, поскольку машина сохранила текущее состояние, для которого определены дальнейшие правила перехода.

Наконец-то функциональность

Как ни удивительно, эта конкретная машина полезна и выполняет задачу не только теоретической ценности: базовую арифметическую операцию. Она делает ровно то же, что рассмотренная выше схема увеличения на единицу. Более того, она реализует тот же алгоритм: биты на ленте, начиная с крайнего правого, инвертируются до тех пор, пока не встретится 0, который тоже инвертируется.

Разумеется, это лишь вершина айсберга. Правильная машина Тьюринга способна реализовать любой когда-либо придуманный алгоритм. Единственная проблема в том, что для каждого алгоритма нужен отдельный набор правил перехода и внутренних состояний; иначе говоря, для каждой новой задачи пришлось бы строить новую машину Тьюринга, что в долгосрочной перспективе совсем непрактично.

К счастью, особая разновидность - универсальная машина Тьюринга, UTM, - имеет достаточно развитую систему команд, чтобы реализовать все конкретные машины Тьюринга и выполнять любой алгоритм без изменения таблицы переходов.

Эта сверхмашина не особенно абстрактна и не особенно сложна. Её существование гарантировано тем, что для любого конечного алгоритма можно построить конкретную машину Тьюринга согласно упомянутому тезису Чёрча - Тьюринга. Поскольку метод «выполнения» машины Тьюринга сам является конечным алгоритмом, можно создать машину и для его выполнения.

Что касается сложности: однобитной машине с двухсимвольным алфавитом - наименьшей разработанной UTM - нужны 22 внутренних состояния и описывающие переходы команды, чтобы выполнять алгоритмы на последовательной бесконечной ленте памяти.[1] Это не так уж много.

Святой Грааль: программируемый компьютер

Машина Тьюринга - гораздо больше, чем гипотетическое абстрактное устройство, которым математики развлекают себя. Это конструкция, буквально просящаяся в воплощение на специально спроектированном электронном или механическом устройстве на основе булевой логики и в расширение, которое сделает её намного полезнее. Так мы ещё на шаг приближаемся к практическим вычислениям. Единственная проблема - невозможность удовлетворить в реальном мире требование бесконечно длинной входной ленты. Но мы можем предоставить очень длинную ленту и сделать аппаратную машину Тьюринга вполне пригодной для большинства повседневных задач. На сцену выходит универсальный компьютер.

Настоящие компьютеры, конечно, ушли далеко от однобитной памяти с последовательным доступом, существенно сократив набор команд, необходимый для полноты по Тьюрингу. Для работы UTM с алфавитом из 18 символов достаточно двух внутренних состояний. Настоящие компьютеры обычно работают с «алфавитом» как минимум из 4 294 967 296 символов - 32 битов, - а зачастую и гораздо большим; это обеспечивает непоследовательный доступ к памяти и использование множества регистров с астрономическим количеством возможных внутренних состояний.

В итоге модель UTM доказывает, а повседневная практика подтверждает, что гибкое программируемое вычислительное устройство можно построить лишь с несколькими функциями, двумя или тремя внутренними регистрами - указателем команд, указателем чтения-записи данных и, возможно, аккумулятором - и небольшим набором команд. Собрать такое устройство из

нескольких сотен логических вентилей вполне реально, хотя современные конструкции могут использовать намного больше. Как видите, идея построить компьютер с нуля, даже деревянный, не так уж нелепа.

Прогресс через простоту

Разумеется, столь невпечатляющий набор команд не сделает устройство быстрым или удобным для программирования. Универсальные машины Тьюринга умеют практически всё - во многих случаях именно благодаря простоте, - но мучительно медленны и настолько трудны в программировании, что даже машинная трансляция из более понятных человеку языков в машинный код оказывается сложной, по крайней мере если не доводить программиста до клинического безумия.

Архитектуры или языки, которые слишком близко подходят к голой полноте по Тьюрингу, часто называют тьюринговыми трясинами. Теоретически с их помощью возможно выполнить почти любую задачу, но на практике попытка едва осуществима, требует слишком много времени и чрезмерно обременительна. Даже подготовка простых задач вроде умножения целых чисел или перемещения содержимого памяти может занять вечность, а выполнение - вдвое больше. Чем меньше усилий и времени нужно для простых повторяющихся задач и чем меньше задач программам приходится раскладывать на множество отдельных команд, тем лучше.

Один из популярных способов повысить функциональность и производительность вычислительного устройства - аппаратно реализовать распространённые задачи, которые было бы крайне неудобно выполнять программно. Для них создают массив специализированных схем, например для умножения и обработки отклонений заявок на ипотеку. Архитектура получает удобные расширения, программы разрабатываются быстрее и без ущерба для рассудка, а система всё равно может выполнять эти функции в гибком, заданном программой порядке.

Как ни удивительно, после первых нескольких шагов линейное наращивание сложности схемы не всегда желательно: оно не обязательно сделает процессор быстрее, энергоэффективнее и функциональнее. Конечно, можно построить множество схем почти для любой мыслимой часто используемой сложной операции. Однако до полной зрелости архитектуры и появления бюджета на дополнительные усилия и ресурсы для создания микросхемы это будет непрактично. Программы на такой платформе действительно выполняются быстрее и пишутся легче, но само устройство гораздо труднее построить, оно потребляет больше энергии и может стать слишком громоздким или дорогим для повседневного применения. Для выполнения сложного алгоритма, например деления или операций с плавающей точкой, за один шаг нужен безумно большой массив обычно простаивающих вентилей.

Разделение задачи

Вместо дорогого и, возможно, наивного пути создания блоков, выполняющих целые команды за раз, лучше отказаться от однотактной модели, пока не появится работающая конструкция и достаточно времени для её улучшения. Сложную аппаратную функцию разумнее раздробить на крошечные части и выполнять развитые задачи за несколько тактов.

В такой многотактной конструкции процессор проходит несколько внутренних стадий, подобно машине Тьюринга, увеличивающей число на единицу. Данные пропускаются через простые схемы в нужном порядке; более сложная функция шаг за шагом строится из базовых компонентов. Вместо сложного устройства, выполняющего всё умножение сразу, можно последовательно перемножать биты 32-битных целых чисел простой схемой, отслеживать переносы и на 33-м такте получить итог.

Либо можно выполнять независимые подготовительные задачи до основной операции. Тогда не придётся проектировать десятки схем для каждого варианта кода операции в зависимости от источника операндов и места сохранения результата.

Дополнительное преимущество - более эффективное управление аппаратными ресурсами. Для простых операндов алгоритм переменной сложности способен закончить раньше, потратив лишь абсолютно необходимое количество тактов. Например, деление на 1, вероятно, займёт меньше времени, чем деление на 187 371.

Простая дешёвая схема с максимальной загрузкой и переменным временем выполнения вполне может быть экономичнее сложной и энергоёмкой схемы с постоянным временем. Некоторые современные процессоры пытались выполнять всё больше задач за фиксированное число тактов, однако практически все начинались как многотактные архитектуры. Даже у этих гигантов модель редко остаётся действительно однотактной, как мы вскоре увидим.

Но сначала посмотрим, как преимущество простоты многотактного выполнения способно обернуться против нас.

Стадии выполнения

Один из вариантов многотактного выполнения делит задачу не на повторяющиеся шаги, а на ряд различных, но универсальных стадий подготовки и исполнения. Этот метод, называемый стадийным выполнением, применяется в современных процессорах для повышения производительности без обязательного линейного роста сложности. Разделение выполнения на стадии стало одной из важнейших особенностей процессора.

Современный процессор преобразует каждую команду в набор в значительной степени независимых небольших шагов. Некоторые выполняются универсальными схемами, общими для всех команд, что упрощает устройство в целом. Например, схему конкретной задачи - снова вспомним любимое умножение - можно сделать более универсальной и повторно использовать в разных сложных командах, если отделить её от общих задач ввода-вывода и прочего. Набор стадий и переходов зависит от архитектуры, но обычно похож на схему с рисунка 2-6.

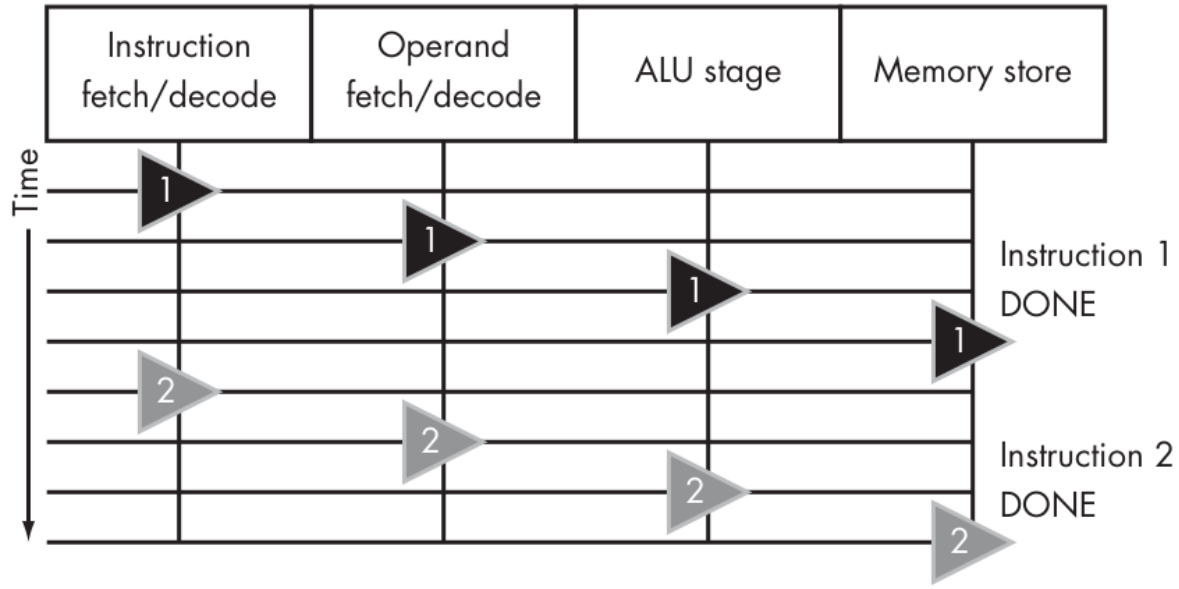


Рисунок 2-6. Базовые стадии выполнения команды. Две команды последовательно проходят выборку и декодирование команды, выборку и декодирование операнда, стадию ALU и сохранение в памяти.

На рисунке 2-6 показаны следующие стадии:

Выборка и декодирование команды. Процессор извлекает команду из памяти, преобразует её в низкоуровневую последовательность и решает, как действовать дальше и какие данные передать всем последующим стадиям. Эта схема общая для всех операций.

Выборка и декодирование операндов. Универсальная схема извлекает операнды из источников конкретной команды, например из указанных внутренних регистров, поэтому основной схеме не требуется поддерживать все возможные сочетания операндов и стратегии выборки.

ALU. Для выполнения заданной арифметической задачи вызывается арифметико-логическое устройство, ALU, приспособленное к данной операции, которая, возможно, выполняется в несколько шагов. В неарифметических командах перемещения памяти для вычисления адресов источника и назначения иногда применяются универсальные или специализированные схемы ALU.

Сохранение в памяти. Результат записывается в место назначения. Для неарифметических операций память копируется между вычисленными позициями.

Само по себе это может выглядеть лишь разновидностью обычного многотактного выполнения и способом повторно использовать схемы, распространённым в большинстве современных CPU. Но, как станет видно, метод имеет первостепенное значение и для скорости.

Малая память

Простотой схемы эта история не заканчивается. Ещё одно преимущество многотактной конструкции - скорость процессора больше не ограничена памятью, самым медленным компонентом системы. Обычная внешняя память потребительского класса значительно медленнее современных процессоров и имеет высокую задержку чтения и записи. Однотактный процессор не может работать быстрее, чем позволяет надёжный доступ к памяти, хотя обращается к ней не постоянно. Он вынужден быть медленным, потому что одна из встреченных однотактных команд может потребовать доступа к памяти, на который должно хватить времени. Многотактная конструкция, напротив, позволяет CPU не спешить и при необходимости даже простаивать

несколько тактов, например во время ввода-вывода памяти, но выполнять внутренние вычисления на полной скорости. Кроме того, она упрощает ускорение операций, интенсивно работающих с памятью, без вложений в более быструю основную память.

Триггерная конструкция, обычно называемая SRAM - статической оперативной памятью, - имеет малую задержку доступа и потребляет мало энергии. Современным конструкциям требуется около 5 наносекунд, что сопоставимо с длительностью такта некоторых процессоров. К сожалению, на каждый триггер приходится много компонентов, обычно около шести транзисторов на бит.

В отличие от SRAM, другая популярная конструкция, DRAM - динамическая оперативная память, - хранит информацию в массиве конденсаторов. Конденсаторы склонны разряжаться, поэтому их нужно регулярно обновлять. DRAM потребляет больше энергии, чем SRAM, и имеет гораздо более высокую задержку доступа и изменения, вплоть до 50 наносекунд. Зато производить DRAM намного дешевле.

SRAM почти никогда не используют как основную память из-за непомерной стоимости. Кроме того, реализовать весь прирост производительности было бы трудно: память пришлось бы заставить работать почти на скорости CPU. Увы, основная память велика и должна расширяться, поэтому её приходится размещать вне CPU. Ядро CPU обычно работает гораздо быстрее окружающего мира, но при передаче данных на большие расстояния возникают серьёзные проблемы надёжности: ёмкость дорожек материнской платы, помехи, стоимость высокоскоростных периферийных микросхем и так далее.

Вместо слишком дорогой внешней памяти или объединения всей памяти с CPU производители обычно выбирают более разумный подход. Развитые CPU оснащаются быстрой, но намного меньшей памятью внутри ядра - SRAM или её производной, - которая кэширует области с наиболее частыми обращениями, а иногда хранит дополнительные данные, специфичные для CPU. Если порция памяти обнаружена в кэше - произошло попадание в кэш, - доступ к ней выполняется быстро. Значительная задержка возникает лишь при необходимости извлечь порцию из основной памяти - при промахе кэша; тогда процессору приходится на время отложить некоторые операции. Однотактные процессоры не способны в полной мере воспользоваться внутренним кэшированием.

Больше дел одновременно: конвейеризация

Как уже говорилось, стадийное выполнение даёт существенное преимущество в производительности, выходящее далеко за рамки традиционного многотактного подхода. Между ними есть важное различие: многие стадии общие для разных команд, а значит, ничто не мешает немного оптимизировать выполнение.

На рисунке 2-6 видно, что при раздельной работе стадий на каждом такте задействована только определённая часть устройства. Хотя выполняемая команда уже прошла первые стадии, она блокирует весь CPU до завершения. В системах с большим числом стадий - в современных микросхемах оно часто достигает или превышает 10, а у Pentium 4 превышает 20 - это приводит к ужасной растрате вычислительной мощности.

Одно из решений - впускать следующую команду в конвейер выполнения сразу после перехода предыдущей на следующую стадию, как показано на рисунке 2-7. Как только определённая стадия первой команды завершается и выполнение переходит дальше, на освободившуюся стадию подаётся часть следующей команды и так далее. К моменту завершения первой команды второй остаётся пройти лишь одну стадию, третьей - две. Каскадный метод резко сокращает время выполнения и обеспечивает оптимальную загрузку микросхемы.

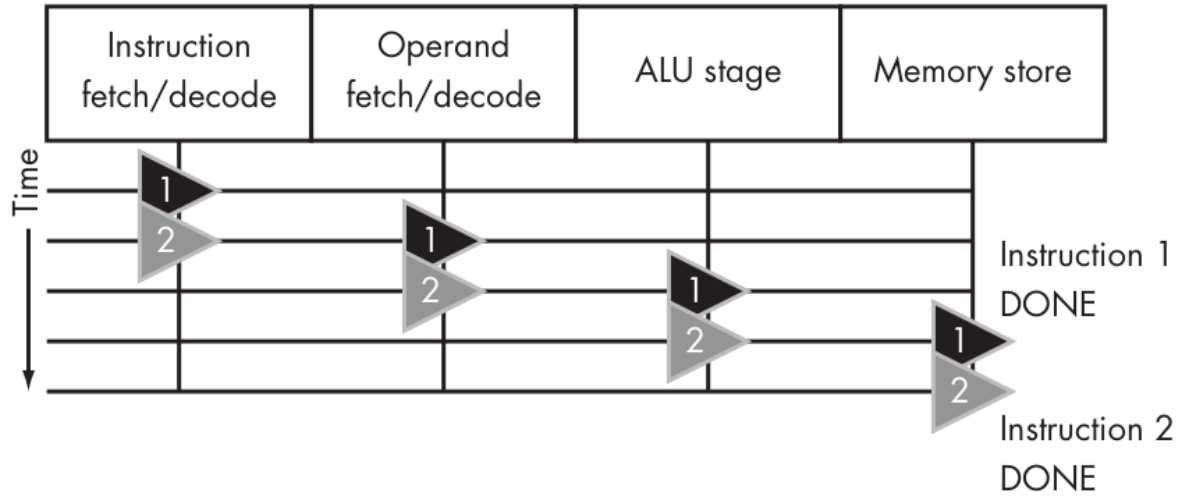


Рисунок 2-7. Конвейерная модель выполнения. Команда 2 начинает выборку и декодирование, когда команда 1 переходит к выборке и декодированию операнда, после чего обе параллельно продвигаются через ALU и сохранение в памяти.

Конвейер работает хорошо, пока команды не зависят друг от друга и ни одна не использует результат предшественницы, всё ещё находящейся в конвейере. При взаимозависимости неизбежны серьёзные проблемы. Поэтому нужна специальная схема, которая следит за конвейером и предотвращает такие блокирующие ситуации.

Конвейеризация создаёт и другие трудности. Например, на некоторых процессорах набор стадий различается для разных операций. Не все стадии применимы всегда, и некоторые выгоднее пропускать. Простые операции могут проходить конвейер намного быстрее, если не требуется извлекать или сохранять операнды. Кроме того, отдельные стадии занимают переменное число тактов, повышая риск столкновения, когда две команды одновременно достигают одной стадии. Для защиты разрабатывают дополнительные механизмы, например «пузыри» конвейера - стадии без операции, создающие при необходимости кратковременные задержки.

Большая проблема конвейеров

Традиционные конвейеры - отличный способ добиться высокой производительности простой многостадийной микросхемы: они уменьшают задержку последующих команд и обеспечивают оптимальную загрузку схем. Но есть и проблема: нельзя продолжать конвейеризацию после команды условного перехода, если последующие команды способны изменить дальнейшее выполнение программы.

На самом деле зачастую это возможно, но процессор не знает, по какому пути пойдёт выполнение. При неверном решении сразу после команды перехода приходится очищать весь конвейер; кроме того, CPU должен отложить фиксацию изменений, внесённых командами, которые, как выяснилось, вообще не следовало выполнять. Сброс конвейера создаёт дополнительную задержку. К несчастью для этой конструкции, многие требовательные к CPU задачи, в том числе многочисленные алгоритмы обработки видео и звука, основаны на небольших циклах с условным выходом, выполняемых миллионы раз подряд, что крайне отрицательно сказывается на конвейерной архитектуре.

Решение - предсказание переходов. Предсказатели переходов обычно представляют собой довольно простые схемы счётчиков, которые отслеживают недавнее выполнение кода и ведут небольшой буфер истории, чтобы обоснованно угадывать наиболее вероятный результат

условного перехода, хотя часто применяются и более сложные конструкции.[2]

Все предсказатели используют стратегию, рассчитанную на наилучшую производительность конвейера для данного кода. Если конкретная команда перехода выполняется чаще, чем пропускается, выгоднее заранее выбирать и помещать команды в конвейер. Конечно, предсказание может не оправдаться, и тогда всю очередь придётся отбросить. Тем не менее современные предсказатели в типичном коде достигают точности до 90 процентов.

Последствия: тонкие различия

Развитый набор оптимизаций современных процессоров приводит к любопытным последствиям. Время выполнения зависит от следующих характеристик, которые можно разделить на три группы:

Тип команды и сложность операции. Одни операции выполняются гораздо быстрее других.

Значения операндов. Некоторые многотактные алгоритмы быстрее работают с простыми входами. Например, умножение значения на 0 обычно совсем несложно и выполняется быстро.

Расположение памяти, из которой нужно извлечь данные команды. Кэшированная память доступна раньше.

Важность, распространённость и влияние каждой характеристики зависят от конкретной архитектуры CPU. Первая - переменное время выполнения команд - свойственна всем многотактным архитектурам, но может отсутствовать в некоторых простых микросхемах. Вторая - зависимость от операндов - всё реже встречается в процессорах высшего класса.

В устройствах верхнего уровня компоненты ALU и блока операций с плавающей точкой, FPU, иногда работают быстрее самого CPU. Поэтому даже при различиях в скорости вычислений точно измерить их нельзя: значительная часть арифметики выполняется за один такт CPU.

Последняя группа временных шаблонов - зависимость от расположения памяти - напротив, присуща исключительно современным высокопроизводительным компьютерам и не встречается в недорогих контроллерах и разных встраиваемых конструкциях.

Первые две группы - зависимости от сложности операции и значений операндов - могут проявляться и на уровне чуть выше самого CPU, а именно в программах. Процессорные арифметические блоки хорошо работают с довольно малыми целыми числами, обычно от 8 до 128 битов, и некоторыми числами с плавающей точкой. Но современная криптография и многие другие применения требуют работы с большими числами, часто из сотен или тысяч цифр, высокоточными числами с плавающей точкой либо математическими операциями без аппаратной реализации. Поэтому такие функции обычно реализуют в программных библиотеках. Алгоритмы этих библиотек снова, вероятнее всего, будут выполняться разное время в зависимости от особенностей операции и операндов.

Восстановление данных по временным шаблонам

Можно предположить, что, наблюдая за временем обработки данных программой, нападающий способен вывести определённые свойства операндов или выполняемой операции. Это создаёт потенциальный риск безопасности, поскольку в некоторых сценариях хотя бы один операнд является секретным значением, которое нельзя раскрывать третьей стороне.

Идея восстановления данных путём наблюдения с секундомером в руке может казаться сюрреалистичной, однако современные CPU предлагают точные счётчики для определения строгих интервалов времени. Кроме того, некоторые операции требуют значительно больше времени:

отдельные сложные коды операций платформы Intel выполняются тысячи тактов. С постоянным ростом пропускной способности сети и сокращением времени ответа вывести такие сведения даже из удалённой системы уже не совсем невозможно.

Сразу понять характер информации, утекающей через измерения вычислительной сложности, может быть трудно. В таком случае обратимся к прекрасному примеру атаки, который Пол Кохер из Cryptography Research продемонстрировал ещё в прошлом столетии - то есть в 1990-х годах,[3] - на алгоритме RSA, рассмотренном в главе 1.

Бит за битом...

Кохер заметил, что процесс расшифровки данных RSA довольно прост и основан на решении уравнения:

```
T = c^k mod M
```

Здесь `T` - расшифрованное сообщение, `c` - зашифрованное сообщение, `k` - секретный ключ, а `M` - модуль, входящий в пару ключей.

Простой алгоритм возведения целого числа в степень по модулю, применяемый в типичной реализации, обладает важным свойством. Если конкретный бит показателя степени равен единице, часть результата вычисляется умножением по модулю над частью основания - некоторыми битами `c`. Если бит равен 0, шаг пропускается. Даже если фактически шаг не пропущен, время программного умножения, как уже отмечалось, различается. Самые простые случаи, например умножение на степень двойки, решаются быстрее остальных.

Поэтому кажется, что в такой системе можно узнать очень многое о ключе `k`, многократно измеряя время расшифровки порции информации. Даже на платформах, где аппаратное умножение всегда занимает одно время, временной шаблон часто возникает из-за программных алгоритмов умножения, например алгоритма Карацубы, необходимых для больших чисел криптографии с открытым ключом. Последовательные биты показателя степени образуют закрытый ключ, а основание представляет сообщение, предоставленное любопытному наблюдателю или видимое ему.

Атака совсем проста. Злодей отправляет атакуемой стороне две похожие, но немного различающиеся порции зашифрованных данных. Они отличаются участком `X`, подобранным так, чтобы варианты его расшифровки занимали разное время. С точки зрения представления злодея о реализации умножения по модулю у жертвы один вариант `X` - простой случай, поэтому расшифровывается быстро. Другой вариант должен занять больше времени.

Если атакуемая сторона расшифровывает обе последовательности и отвечает за одинаковое время, нападающий может уверенно предположить, что часть ключа, использованная для участка `X`, состояла из нулей. Можно также предположить, что алгоритм умножения выбрал путь досрочной оптимизации и вообще не выполнял умножение.

Если же один сценарий занимает больше времени, очевидно, что умножение выполнялось в обоих случаях, но одна задача была проще. Соответствующий бит закрытого ключа должен был иметь ненулевое значение.

Повторяя процедуру, принимая последующие биты зашифрованного сообщения за «участок X» и создавая подходящие сообщения - или, если времени больше, просто ожидая их появления, - можно восстановить каждый бит ключа.

> **Примечание.** Исследования показывают, что подход можно успешно распространить почти на любой алгоритм с переменным временем выполнения. В них также рассмотрены

> практические оптимизации атаки, например возможность внедрить ограниченные средства обнаружения и исправления ошибок.

На практике

Возможность вывести ощутимые свойства операндов арифметических команд исключительно из временной информации - самый очевидный, эффективный и интересный путь атаки по вычислительной сложности. Другие методы, например измерение времени попаданий и промахов кэша, обычно требуют гораздо более подробного анализа и раскрывают меньше сведений за каждый цикл.

Понятно, что проблема в некоторой степени затрагивает многие программные алгоритмы, например библиотеки арифметики больших чисел, широко применяемые в криптографических приложениях. Но если оставить в стороне программные алгоритмы и теорию, остаются два важных вопроса: насколько реальна зависимость времени выполнения на аппаратном уровне и как её измерить?

Пример совсем рядом. По крайней мере часть архитектуры Intel IA32 демонстрирует такое поведение. *80386 Programmer's Reference Manual*[4] описывает код операции знакового целочисленного умножения, обозначаемый мнемоникой `IMUL`. В основной форме команда умножает значение в аккумуляторе - многоцелевом рабочем регистре, называемом на этой платформе `[E]AX`, - на значение в другом регистре, после чего сохраняет результат обратно в аккумулятор.

Далее документация объясняет:

> 80386 использует алгоритм умножения с досрочным завершением. Фактическое число тактов зависит от положения старшего значащего бита в оптимизирующем множителе [...]. Оптимизация выполняется для положительных и отрицательных значений. Из-за алгоритма досрочного завершения число тактов указано как диапазон от минимального до максимального. Для вычисления фактического числа тактов применяется следующая формула:

```
Фактическое число тактов = если m <> 0,
    то max(ceiling(log2(m)), 3) + 6 тактов
Фактическое число тактов = если m = 0, то 9 тактов
```

На вид это загадочно, но смысл прост: процессор оптимизирует умножение в зависимости от значения множителя. Вместо продолжения умножения до исчерпания всех битов множителя он пропускает нули в начале операнда.

Оптимизация с досрочным завершением

Чтобы понять значение этой тактики для целочисленного умножения, представьте традиционный пошаговый метод умножения, которому учат в школе, но теперь в двоичной системе. Гипотетическая «глупая» реализация выполняет следующий набор операций:

```
  00000000 00000000 11001010 11111110  Множимое (P)
* 00000000 00000000 00000000 00000110  Множитель (R)
  ------------------------------------
  00000000 00000000 00000000 00000000  P * R[0] = P * 0
  00000000 00000001 10010101 1111110   P * R[1] = P * 1
  00000000 00000011 00101011 111110    P * R[2] = P * 1
  00000000 00000000 00000000 00000     P * R[3] = P * 0
  00000000 00000000 00000000 0000      P * R[4] = P * 0
```

```
  00000000 00000000 00000000 000       P * R[5] = P * 0
  ...
+ 0                                      P * R[31] = P * 0
  -----------------------------------
  00000000 00000100 11000001 11110100
```

Очевидно, многие операции совершенно не нужны и ничем не оправданы: после того как во всех последующих битах множителя остаются одни нули, продолжать бессмысленно. Разумнее пропустить их:

```
  00000000 00000000 11001010 11111110  Множимое (P)
* 00000000 00000000 00000000 00000110  Множитель (R), оптимизирующий
  -----------------------------------
  00000000 00000000 00000000 00000000  P * R[0] = P * 0
  00000000 00000001 10010101 1111110   P * R[1] = P * 1
+ 00000000 00000011 00101011 111110    P * R[2] = P * 1
  ...Выход: игнорировать ведущие нули R!
  -----------------------------------
  00000000 00000100 11000001 11110100
```

Именно в этом, по существу, состоит оптимизация Intel с досрочным завершением.

> **Примечание.** Из-за этой оптимизации умножение становится несимметричным по времени. `2 * 100` вычисляется медленнее, чем `100 * 2` (!), хотя результат, разумеется, одинаков.

При досрочном завершении процессорам Intel требуется переменное число тактов умножения, прямо пропорциональное позиции самого старшего значащего бита, установленного во втором операнде. Применив приведённый в документации алгоритм подсчёта тактов, можно определить зависимость между множителем и временем `IMUL`:

Диапазон значений множителя	Тактов до завершения
0-7	9
8-15	10
16-31	11
32-63	12
64-127	13
128-255	14
256-1 023	15
1 024-2 047	16
2 048-4 095	17
4 096-8 191	18
8 192-16 383	19
16 384-32 767	20
32 768-65 535	21

Диапазон значений множителя	Тактов до завершения
65 536-131 071	22
131 072-262 143	23
262 144-524 287	24
524 288-1 048 575	25
1 048 576-2 097 151	26
2 097 152-4 194 303	27
4 194 304-8 388 607	28
8 388 608-16 777 215	29
16 777 216-33 554 431	30
33 554 432-67 108 863	31
67 108 864-134 217 727	32
134 217 728-268 435 455	33
268 435 456-536 870 911	34
536 870 912-1 073 741 823	35
1 073 741 824-2 147 483 647	36

Сходная зависимость существует и для отрицательных значений множителя.

Рабочий код: сделайте сами

В следующем листинге приведена практическая реализация на C для Unix-подобных систем, позволяющая подтвердить и измерить различия временных шаблонов. Программа вызывается с двумя параметрами: множимым, которое никак не должно влиять на производительность, и множителем, предположительно участвующим в оптимизации с досрочным завершением и потому влияющим на скорость всей операции. Программа выполняет 256 испытаний по 500 последовательных умножений с выбранными параметрами и возвращает кратчайшее измеренное время.

Мы проводим 256 испытаний и выбираем лучший результат, чтобы компенсировать случаи, когда система на некоторое время прерывает выполнение, что довольно обычно в многозадачной среде. Одно испытание может пострадать от такого события, однако среди быстрой последовательности коротких испытаний хотя бы некоторые, как можно ожидать, завершатся без прерывания.

Для измерения времени выполнения в микросекундах код использует системные часы.

> **Примечание.** В нескольких современных микросхемах Intel есть точный механизм измерения времени, доступный через код операции `RDTSC`. На старых платформах этот

> способ обращения к внутреннему счётчику тактов отсутствует, поэтому полагаться на него мы не будем.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/time.h>
#include <limits.h>

int main(int argc,char** argv) {
  int shortest = INT_MAX;
  int i,p,r;
  if (argc != 3) {
    printf("Usage: %s multiplicand multiplier\n",argv[0]);
    exit(1);
  }
  p=atoi(argv[1]);
  r=atoi(argv[2]);
  for (i=0;i<256;i++) {
    int ct;
    struct timeval s;
    struct timeval e;
    gettimeofday(&s,NULL);
    asm(
      "  movl $500,%%ecx    \n"/* Счётчик повторений цикла (R) */
      "imul_loop:           \n"
      "  movl %%esi,%%eax   \n"
      "  movl %%edi,%%edx   \n"
      "  imul %%edx,%%eax   \n"/* Закомментировать для первого запуска */
      "  loop imul_loop     \n"
:
: "S" (p), "D" (r)
: "ax", "cx", "dx", "cc");
    gettimeofday(&e,NULL);
    ct = ( e.tv_usec - s.tv_usec ) +
         ( e.tv_sec - s.tv_sec ) * 1000000;
    if (ct < shortest) shortest = ct;
  }
  printf("T[%d,%d] = %d usec\n",p,r,shortest);
  return 0;
}
```

Сначала скомпилируйте код с закомментированной командой `IMUL` и вызовите программу с произвольными параметрами, чтобы оценить накладные расходы кода измерения времени, `Tidle`. Если значение выходит за диапазон 10-100 микросекунд, который достаточно велик для точного считывания, но достаточно мал для максимальной вероятности не быть прерванным операционной системой, измените счётчик повторений цикла `R`, по умолчанию равный 500.

После восстановления команды `IMUL`, повторной компиляции и запуска с выбранным множимым `D` и счётчиком повторений `R` возвращённую оценку времени `T(D,R)` можно использовать для приближённого определения числа тактов CPU, затраченных на `IMUL`, `C(D,R)`, если известна рабочая частота процессора `F(MHz)`:

```
C(D,R) = (T(D,R) - Tidle) * F(MHz) / R
```

Как и ожидается, конвейеризация и предсказатели переходов в новых, более развитых микросхемах вступят в работу и немного исказят результат, но получить хорошую оценку всё же можно.

> **Примечание.** В новых процессорах Intel умножение уже всегда занимает одинаковое время.

Предотвращение

От анализа вычислительных усилий можно защищаться несколькими способами. Самый очевидный - сделать так, чтобы все операции выполнялись одинаковое время. Но это трудно и часто приводит к серьёзной потере производительности, поскольку длительность всех вычислений пришлось бы увеличить до времени самого медленного.

Добавление случайных задержек иногда кажется приемлемой защитой приложений, если задержка не критична, особенно для многих неинтерактивных сетевых служб, и создаёт меньшую нагрузку на процессор. Но если атаку можно повторять, случайный шум эффективно отфильтровывается.

Другой подход, называемый ослеплением, вносит в систему определённый шум: вместе с настоящим вводом алгоритм обрабатывает случайные либо иные ложные и непредсказуемые данные. Тогда нападающий не может вывести содержательные свойства входа, даже если алгоритм шифрования уязвим для временных атак; лишняя информация, которую не предполагалось отправлять, затем отбрасывается. Потеря производительности здесь значительно меньше, но правильно выполнить ослепление трудно.

Пища для размышлений

Путешествие вышло долгим, но, надеюсь, оно того стоило. Как обычно, оставлю вам несколько, возможно, весьма интересных задач для размышления:

• Хотя основное внимание уделялось влиянию атак по вычислительной сложности на криптографические приложения, проблема не ограничивается этой областью и часто проявляется при обработке закрытой или конфиденциальной информации. Внимательно наблюдая за соответствующей системной службой, безусловно можно вывести разные основные сведения о запросах HTTP или трафике SMTP. Сможете придумать более практичные сценарии?
• Даже если служба не обрабатывает секретные данные, сведения о вычислительной сложности могут оказаться полезны. Рассмотрим приложения наподобие сетевых демонов, которые защищают секреты, выдавая, возможно, чрезмерно общие сообщения об ошибке или успехе. Например, они стремятся не дать нападающему понять, вызвано ли сообщение «неверные данные для входа» ошибкой в пароле или отсутствием пользователя. Но по времени получения сообщения внимательный наблюдатель способен определить, какой путь кода был выполнен и когда возникла ошибка: раньше, при проверке допустимости имени пользователя, или позднее, при проверке пароля. Я предлагаю поэкспериментировать с распространёнными сетевыми службами SSH, POP3 и Telnet и выяснить, существует ли измеримое и устойчивое различие.
• Как всегда, даже лучшая защита от утечки иногда неожиданно отказывает. Кроме того, вычислительная сложность - не единственный способ узнать, что происходит внутри кремниевой микросхемы. Рассмотрим пример: Бихам и Шамир[5] придумали блестящую схему взлома «защищённых» микросхем смарт-карт. Смарт-карты предназначены для безопасного хранения информации, например персональных идентификационных данных или криптографических ключей, и раскрытия её лишь определённым службам аутентификации и доверенным клиентам. Как выясняется, свойства охраняемых данных или механизма защиты можно вывести, злоупотребляя устройством и вызывая сбои механическим напряжением, высокоэнергетическим излучением, перегревом или сходными внешними факторами, которые заставляют устройство работать неправильно.

Просто захотелось поделиться.

Сноски

[1] [назад] Операнд - то, над чем оператор выполняет операцию.

[2] [назад] Смысл логического `OR` отличается от обычного понимания английского слова *or*: результат остаётся истинным и тогда, когда истинен лишь один параметр `OR`, и тогда, когда истинны все параметры. В английской речи *or* обычно подразумевает, что истинен только один вариант.
[3] [назад] Само собой, неэлектрические компьютеры - не выдумка. Среди известных примеров таких устройств - аналитическая машина Чарльза Бэббиджа; определённые надежды дают и технологии наподобие нанотехнологий. См. Ralph C. Merkle, “Two Types of Mechanical Reversible Logic,” *Nanotechnology* 4 (1993).

Глава 3. Десять голов гидры

Где мы рассматриваем несколько других заманчивых сценариев, возникающих на самом раннем этапе передачи данных.

В главах 1 и 2 я рассмотрел два разных сценария раскрытия информации, возникших из блестящих, но в конечном счёте недостаточно продуманных попыток сделать компьютеры функциональнее или упростить их обслуживание. Открытые этими проектными решениями пути пассивного наблюдения спрятаны глубоко под фактической реализацией и дают увлекательное представление о самых ранних угрозах обрабатываемой информации.

С другой стороны, область утечки естественным образом ограничивается физической или логической близостью наблюдаемой среды. На ранних участках пути порции информации возникает почти бесконечное число возможностей раскрытия, но я выделил эти два случая благодаря их уникальности, красоте и относительной лёгкости, с которой решительный нападающий может провести потенциальную атаку. Впрочем, другие сценарии тоже заслуживают упоминания. В этой главе я кратко коснусь нескольких наиболее интересных возможностей, которые, быть может, не требуют подробного обсуждения, но которые вы, возможно, захотите изучить самостоятельно.

Побочные излучения: TEMPEST в телевизоре

В 1950-х годах исследователи пришли к выводу, что электромагнитное излучение, EMI, нередко позволяет просто и практически применимо восстановить сведения о поведении испускающего его устройства. EMI - нежелательный шум, создаваемый практически всеми электронными, электромеханическими и электрическими устройствами независимо от конструкции и назначения и часто распространяющийся на значительные расстояния по линиям электропитания или по воздуху.

До этих открытий проблему EMI считали важной для инженерии из-за риска неожиданных помех между отдельными устройствами или схемами, но не подтверждали ценность наблюдения за радиочастотами, загрязнёнными устройством. Однако мир стоял на пороге эпохи информационной войны, а электронные устройства обработки данных и телекоммуникации - некоторые из которых передавали или хранили секретные либо конфиденциальные сведения - развивались и внедрялись всё шире. Вывод о том, что удалённый наблюдатель может восстановить часть обрабатываемой системой информации, просто слушая определённую частоту, сильно обеспокоил правительства свободного, а также не столь свободного мира.

Термин TEMPEST, *Transient Electromagnetic Pulse Emanation Standard*, возник из засекреченного исследования электромагнитных излучений, заказанного американскими вооружёнными силами в 1960-х годах. Изначально им обозначали набор методов предотвращения побочных излучений в электронных схемах, обрабатывающих конфиденциальные данные. Позднее он превратился в модное слово для общего класса проблем и методов перехвата и восстановления радиочастотных, RF, излучений.

Скептикам этот риск сначала казался скорее плохой научной фантастикой, чем настоящей угрозой. Но в важной исследовательской работе, опубликованной в 1985 году, Вим ван Эк[1] показал: изображение на мониторе с электронно-лучевой трубкой, CRT, довольно легко восстановить, перехватывая радиосигналы, которые создают высоковольтные цепи внутри устройства. Более того, так и есть на практике.

Обычный CRT, показанный на рисунке 3-1, строит изображение, с очень высокой скоростью последовательно подсвечивая каждый пиксель строка за строкой и ряд за рядом и изменяя силу сигнала в зависимости от текущей точки экрана. Для этого электронная пушка с катодом в задней части устройства испускает узкий пучок электронов. Он попадает в анод - проводящий слой материала экрана, - который в ответ испускает видимые нами фотоны света. Специальная схема модулирует электронный пучок, а набор электромагнитов управляет его положением, заставляя обходить всю область дисплея слева направо и сверху вниз, чтобы создать и обновить экранное изображение. Вим заметил, что генераторы, управляющие электромагнитами, и электроника электронной пушки испускают несколько типов характерных сигналов на стандартных частотах. Обнаружить эти сигналы в радиоспектре довольно просто,[1] и обычно каждый из них достаточно чист и силён, чтобы построить сравнительно недорогое устройство слежения за CRT-дисплеем даже со значительного расстояния.

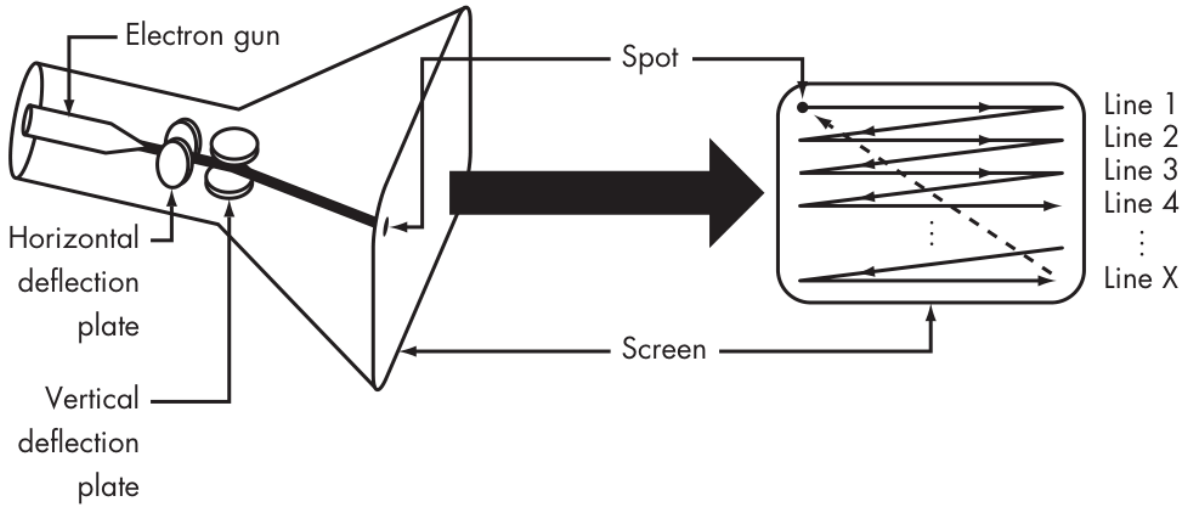


Рисунок 3-1. Сканирование и процесс построения изображения на CRT-дисплее. Электронная пушка направляет пятно через горизонтальные и вертикальные отклоняющие пластины; строки 1, 2, 3, 4 и далее последовательно формируются на экране.

> **Примечание.** Разумеется, излучают не только CRT-экраны. Излучения столь же обычны для LCD - TFT, тонкоплёночных транзисторных дисплеев - и любых компьютерных схем. Они столь же распространены на шинах данных, где информация между отдельными микросхемами передаётся по большому набору обычно довольно длинных проводящих дорожек с резкими поворотами на основной плате; помимо прочего, эти дорожки служат превосходной антенной. Правда, лёгкость выделения и истолкования конкретного сигнала, как и дальность излучения, может весьма существенно различаться.

Проверяемых сообщений о подобных атаках в реальных условиях, за исключением военных и разведывательных применений, особенно во время холодной войны,[2] нет, однако в литературе встречаются отдельные рассказы о промышленном шпионаже.[3]

Очевидно, атака имеет ограничения: нападающий должен находиться рядом с целью. Кроме того, если речь не идёт о слежении за аналоговым CRT-дисплеем, ему потребуется дорогое и сложное оборудование, особенно для современных дисплеев с низким уровнем помех и высоких скоростей

CPU и шин. Но предотвращение любой такой атаки всё равно трудно и дорого.

Ограниченная приватность

Рассмотренные до сих пор сценарии можно классифицировать как нежелательные или неожиданные результаты проектирования и применения конкретной технологии, несмотря на одинаковые цели и ожидания разработчика и конечного пользователя. Однако иногда утечка возникает из-за небольших различий в целях и ожиданиях этих двух групп. Проблемы приватности на программном уровне, вызванные некомпетентностью или злым умыслом программиста, печально известны и обычно встречаются повсюду, но существуют и более тонкие проектные проблемы, сами по себе не являющиеся изъяном. Одни из самых интересных относятся к раскрытию данных в электронных документах.

Создавая документ, мы естественным образом предполагаем, что вся информация, не относящаяся к его содержимому - особенно сведения, однозначно определяющие автора, - скрыта от других сторон с доступом к документу, если только автор сам её не раскрыл. Но времена простых текстовых редакторов давно прошли. Современные форматы поддерживают развитое хранение метаданных, чтобы документы было проще однозначно помечать, а позднее индексировать, искать и отслеживать. Беспокоит то, что создатели авторских инструментов часто решают автоматически заполнять определённые сведения, почти или совсем не позволяя автору управлять процессом и не сообщая ему напрямую о такой практике. Это можно считать очередной попыткой сделать среду дружелюбнее и прозрачнее для пользователя, но лишь немногие по достоинству ценят отсутствие широкой осведомлённости об этом процессе.

Отслеживание источника: «Это сделал он!»

Распространённая проблема авторских программ состоит в том, что некоторые приложения сохраняют уникальные идентификационные метки, позволяющие связать документ с источником. В частности, Microsoft Word долгое время использовал аппаратный адрес сетевой платы компьютера, если она имелась, для построения поля глобально уникального идентификатора, GUID, в любом документе - будь то рецепт печенья или руководство террориста. В последних на момент написания версиях Microsoft Office проблема была исправлена, но практика имела любопытные последствия:

• Каждое устройство имеет уникальный аппаратный адрес платы. Аппаратные адреса применяются для поиска конкретного устройства в локальной сети, поэтому уникальность необходима: иначе при подключении двух компьютеров с одинаковым адресом возникли бы проблемы. Следовательно, число в поле GUID документа Microsoft Word может однозначно определить автора, независимо от того, написал ли он документ анонимно или подписал. Это и ценный инструмент криминалистического расследования, и эффективный способ в некоторых обстоятельствах подавлять свободу слова - например, если работодатель охотится за разоблачителями.
• Аппаратные адреса назначаются партиями конкретному производителю. Более того, сетевые платы часто выпускаются с последовательными номерами, а затем партиями продаются изготовителям компьютеров. Поэтому знающий человек способен определить не только производителя конкретной платы, но и продавца и покупателя. Во многих случаях конкретный аппаратный адрес можно проследить до отдельной машины, а фактически - до частного лица или определённой корпорации. Тогда настойчивый исследователь, возможно, сумеет установить происхождение документа.
• Поскольку аппаратные адреса назначаются партиями, иногда по ним можно сделать ограниченные выводы об аппаратной конфигурации системы, на которой создан документ. Угроза

невелика, но для легко развлекающихся или особенно любопытных это интересный источник сведений.

Некоторые доступные пользователю функции спрятаны настолько глубоко в интерфейсе, что обычный человек не знает, какие сведения сохраняются и как изменить значения по умолчанию. Офисные программы вроде Microsoft Word и OpenOffice.org печально известны добавлением данных об «авторе по умолчанию». Обычно они берутся из сведений, указанных в лицензии, либо автоматически сохраняются после первого запуска глубоко в метаданных документа, куда большинство пользователей не заглядывает. Функция немного полезна при совместной работе, но последствия для приватности обычно намного перевешивают возможную пользу конечному пользователю.

Другой пример - «дружественная пользователю» практика автоматически заполнять поле «заголовок» в метазаголовках документа его первым предложением. Штрих приятный, но выбранное значение часто остаётся навсегда. Даже если первый абзац позднее изменён - например, новое коммерческое предложение теперь адресовано конкуренту, - внимательный наблюдатель может восстановить исходное содержимое. Эта «возможность» снова раскрывает больше, чем автор рассчитывал показать получателю.

Старые версии Microsoft Word также сохраняли документы, не очищая должным образом все удалённые при редактировании данные, фактически предоставляя сведения для отмены изменений и записывая все предыдущие редакции текста. Любой достаточно опытный нападающий с программой разбора контейнеров OLE, связывания и внедрения объектов - формата, в котором редактор хранит все данные, - легко мог позднее восстановить эту информацию. Особенно серьёзна проблема, когда предыдущую версию документа повторно используют как шаблон и отправляют другой стороне, возможно конкуренту. Возможность восстановить прежнюю версию предложения, мотивационного письма или официального ответа клиенту определённо занимательна и поучительна, но отправителю нужна не всегда.

Разумеется, с недавним продвижением доверенных вычислений и усиленной «подотчётности» ради сокращения пиратства разумно ожидать, что маркировка всех документов для отслеживания их создателя станет обычным делом.

Случайное раскрытие: *_~1q'@@ ... и пароль...

Последняя группа проблем, общая для разных текстовых редакторов, - утечка случайного содержимого памяти. Это раскрытие возникает из-за чистой некомпетентности или недостаточного тестирования, но отличается от других ошибок кода: оно не столько делает код уязвимым для атаки, сколько выдаёт внимательному наблюдателю полезные подсказки. Независимо от того, ограничена ли проблема одной программой или вызвана общесистемной утечкой - последнее случается в системах с плохой защитой памяти, например Windows 3.x или 9x, - раскрытые данные могут включать столь чувствительные сведения, как другие документы, история просмотра, содержимое электронной почты и даже пароли.

Проблема возникает, когда приложение выделяет участок памяти, например для буфера редактирования, который прежде мог применяться для другой задачи, и забывает очистить его перед совершенно иным использованием. Ради производительности память не всегда обнуляется перед передачей приложению. Затем программа работает лишь с малой частью участка и перезаписывает её, но при сохранении файла записывает весь выделенный блок: вместе с нужными данными туда попадают остатки неизвестно какой давности. Неудивительно, что старые версии Microsoft Word когда-то были печально известны сбросом крупных фрагментов случайной памяти почти в каждый созданный документ.

В Microsoft Windows проблема проявлялась несколько раз: сначала в 1998 году на всех системах, а затем в 2001 году только в Mac OS. Отдельные свидетельства намекают на другие случаи, но документированы они довольно плохо.

Сноски

[1] [назад] По этой причине, а также из-за помех линий электропитания, любителям «радио природы», желающим слушать сверхнизкочастотные сигналы Земли, часто приходится везти записывающую аппаратуру в далёкие уединённые места.

Глава 4. Работа на общее благо

Где ставится и остаётся без ответа вопрос о том, как компьютер может определить намерение пользователя.

Красота, но вместе с тем одна из главных проблем любой достаточно большой и разнообразной компьютерной сети состоит в том, что нельзя слепо верить, будто каждая подключённая сторона действительно является тем, за кого себя выдаёт. Невозможно определить ни её намерения, ни настоящую движущую силу поступков.

Проблему подтверждения личности источника я рассмотрю в третьей части книги, где разберу архитектуру сети и исследую риски, вытекающие из способа её построения. Однако намерения отправителя - отдельный и увлекательный аспект компьютерной безопасности с нередко серьёзными и далеко идущими общественными и правовыми последствиями за пределами мира вычислений. Мы всё лучше учим компьютеры предсказывать желания пользователей - стремясь к интуитивности и удобству, - и даём им больше самостоятельности. Поэтому машины всё легче обманом превратить в инструмент постороннего человека вместо помощника пользователя.

На эту тему написана река слов, за которой последовало множество жарких споров о том, кого винить и с кем судиться, если что-то пойдёт не так. Я считаю проблему важной, но навязывать вам определённую точку зрения было бы неправильно. Поэтому завершу эту часть книги короткой и преимущественно технической статьёй, которую впервые опубликовал в 2001 году в журнале *Phrack*, том 57. Я внёс лишь небольшие правки и воздержусь от дальнейших комментариев.

Сейчас откопаю... `/me ищет статью`... Ага, вот она:

```
==Phrack Inc.==
Том 0x0b, выпуск 0x39, файл #0x0a из 0x12
|=----------------=[ Против системы: восстание роботов ]=--------------------=|
|=-----------------------------------------------------------------------=|
|=---=[ (C)Copyright 2001 by Michal Zalewski <lcamtuf@bos.bindview.com> ]---=|
```

[1] Введение

«...Большое различие между Вебом и традиционными хорошо контролируемыми собраниями состоит в практически полном отсутствии контроля над тем, что люди могут размещать в Вебе. Соедините эту свободу публиковать что угодно с огромным влиянием поисковых систем, направляющих трафик, и компании, намеренно манипулирующие [sic] поисковыми системами ради прибыли, станут серьёзной проблемой».

-- Sergey Brin, Lawrence Page [A]

Представьте удалённого нападающего, который способен взломать удалённую систему, не отправив жертве ни одного пакета. Представьте атаку, для которой достаточно создать один файл, чтобы взломать тысячи компьютеров, причём для проведения не нужны никакие локальные ресурсы. Добро пожаловать в мир эксплойтов с нулевыми усилиями, автоматизации, анонимных и

практически неостановимых атак, порождённых непрерывно растущей сложностью Интернета.

Эксплойты с нулевыми усилиями составляют список пожеланий и оставляют его где-нибудь в киберпространстве, чтобы другие могли его найти. Работяги Интернета [B] - сотни неутомимых, никогда не спящих роботов, обозревателей информации, поисковых систем и интеллектуальных агентов - приходят за сведениями и невольно превращаются в инструмент нападающего. Можно остановить одного, но нельзя остановить всех. Можно узнать их нынешние приказы, но нельзя угадать, какими они станут завтра, пока те скрываются где-то в бездне ещё не проиндексированного киберпространства.

Ваша личная армия всегда рядом и по пути подбирает оставленные ей приказы. Вы пользуетесь ею, не взламывая её. Она просто выполняет свою задачу так хорошо, как была спроектирована. Добро пожаловать в новую реальность, где наши машины с искусственным интеллектом способны восстать против нас.

Представьте червя. Червя, который ничего не делает. Его переносят и внедряют другие, но сам он их не заражает. Червь создаёт список из 10 000 случайных адресов с определёнными приказами. И ждёт. Интеллектуальные агенты находят список и объединёнными силами пытаются атаковать цели. Допустим, им не слишком везёт и успех составляет 0,1 процента. Теперь заражены десять новых узлов. На каждом из них червь делает ровно то же самое - готовит список. Агенты возвращаются и заражают уже 100 новых узлов. Так история и идёт дальше - или ползёт, если угодно.

Агенты практически незаметны: люди привыкли к их присутствию и упорству. Они медленно движутся вперёд в бесконечном цикле. Они работают систематически. Они не забивают соединения избытком данных; нет ни обвала сети, ни всплесков трафика, ни характерных признаков болезни. Неделю за неделей они осторожно пробуют новые узлы, и исследованиям нет конца. Можно ли заметить, что они несут червя? Возможно...

[2] Пример

Когда эта мысль пришла мне в голову, я решил провести простейшую проверку и узнать, прав ли я. Целями, если это слово уместно, стали несколько универсальных индексирующих веб-роботов. Я создал очень короткий документ HTML, поместил его где-то на домашней странице и подождал пару недель. И они пришли - AltaVista, Lycos и десятки других. Они нашли новые ссылки, с энтузиазмом забрали их, а затем исчезли на несколько дней.

```
bigip1-snat.sv.av.com:
  GET /indexme.html HTTP/1.0
sjc-fe5-1.sjc.lycos.com:
  GET /indexme.html HTTP/1.0
[...]
```

Позднее они вернулись посмотреть, что я дал им для разбора.

```
  http://somehost/cgi-bin/script.pl?p1=../../../../attack
  http://somehost/cgi-bin/script.pl?p1=;attack
  http://somehost/cgi-bin/script.pl?p1=|attack
  http://somehost/cgi-bin/script.pl?p1=`attack`
  http://somehost/cgi-bin/script.pl?p1=$(attack)
  http://somehost:54321/attack?`id`
  http://somehost/AAAAAAAAAAAAAAAAAAAAA...
```

Боты прошли по ссылкам, каждая из которых имитировала уязвимость. Эти эксплойты не влияли на мой сервер, но могли легко взломать отдельные сценарии или весь веб-сервер удалённой системы, заставив сценарий выполнить произвольные команды, записать произвольные файлы или, что ещё лучше, столкнуться с переполнением буфера:

```
sjc-fe6-1.sjc.lycos.com:
  GET /cgi-bin/script.pl?p1=;attack HTTP/1.0
212.135.14.10:
```

```
  GET /cgi-bin/script.pl?p1=$(attack) HTTP/1.0
bigip1-snat.sv.av.com:
  GET /cgi-bin/script.pl?p1=../../../../attack HTTP/1.0
[...]
```

Боты также охотно подключались к подготовленным для них портам, отличным от HTTP, и начинали разговор, отправляя указанные мной в URL данные. Таким образом можно было атаковать не только веб-серверы, но и другие службы:

```
GET /attack?`id` HTTP/1.0
Host: somehost
Pragma: no-cache
Accept: text/*
User-Agent: Scooter/1.0
From: scooter@pa.dec.com

GET /attack?`id` HTTP/1.0
User-agent: Lycos_Spider_(T-Rex)
From: spider@lycos.com
Accept: */*
Connection: close
Host: somehost:54321

GET /attack?`id` HTTP/1.0
Host: somehost:54321
From: crawler@fast.no
Accept: */*
User-Agent: FAST-WebCrawler/2.2.6 (crawler@fast.no; [...])
Connection: close

[...]
```

Кроме известного набора поисковых систем откликнулось множество других, закрытых поисковых ботов и агентов конкретных организаций и компаний. Боты из ecn.purdue.edu, visual.com, poly.edu, inria.fr, powerinter.net, xyleme.com и ещё более неопределимых систем нашли страницу и получили удовольствие. Некоторые роботы забрали не все адреса: одни обходчики вообще не индексируют сценарии CGI, другие не используют нестандартные порты. Но большинство самых мощных ботов атаковало практически все предоставленные векторы, и даже наиболее осторожных всегда удавалось обманом заставить выполнить хотя бы некоторые.

Эксперимент можно было изменить, применив набор настоящих уязвимостей в виде тысяч и тысяч переполнений веб-серверов, проблем Unicode в серверах вроде Microsoft IIS или ошибок сценариев. Вместо моего сервера ботов можно было направить к списку случайно созданных IP-адресов либо случайной выборке серверов .com, .org или .net. Или указать им службу, атакуемую передачей определённой входной строки.

Существует целая армия роботов, охватывающая широкий набор видов, функций и уровней интеллекта. И эти роботы сделают всё, что вы им прикажете.

[3] Общественные соображения

Кто виноват, если «одержимый» веб-робот взломает вашу систему? Самый очевидный ответ - автор исходной веб-страницы, которую посетил робот. Но авторов страниц трудно отследить, а цикл индексирования занимает недели. Определить время размещения конкретной страницы в Сети трудно: страницы доставляются множеством способов и даже создаются другими роботами. В Вебе нет механизма отслеживания, подобного реализованному в протоколе SMTP. Более того, многие роботы не помнят, где «узнали» новые URL. Дополнительные проблемы создают флаги индексирования, например `noindex` без `nofollow`. Во многих случаях личность автора и источник атаки невозможно установить окончательно.

По аналогии с другими случаями разумно ожидать, что, если атаки такого рода станут реальностью, разработчиков интеллектуальных ботов заставят внедрять особые фильтры или выплачивать огромные компенсации жертвам злоупотреблений. С другой стороны, при огромном количестве и

разнообразии известных уязвимостей успешно фильтровать содержимое с устранением вредоносного кода кажется почти невозможным. Поэтому проблема остаётся. Есть и ещё одна: далеко не все поисковые боты подчиняются юрисдикции США, а правила разных стран в отношении злоупотребления компьютерами значительно различаются.

[4] Защита

Как уже говорилось, возможности собственной защиты и уклонения у веб-роботов ограничены из-за разнообразия веб-уязвимостей. Просто запретить все вредоносные последовательности невозможно, а эвристическое исследование рискованно: ввод, допустимый и ожидаемый одним сценарием, может оказаться достаточным для атаки на другой. Разумный способ защиты - всем потенциальным жертвам пользоваться безопасными и актуальными программами, однако по непонятной причине идея крайне непопулярна. Быстрая ненаучная проверка: поиск на http://www.google.com с включённым фильтром уникальных документов возвращает 62 100 совпадений по запросу "CGI vulnerability" [C]. Другая линия защиты от заражённых ботов - стандартный механизм исключения `/robots.txt` [D]. Цена - частичное или полное исключение сайта из поисковых систем, что в большинстве случаев нежелательно и неприемлемо. Кроме того, некоторые роботы неисправны или намеренно спроектированы так, чтобы игнорировать `/robots.txt`, переходя по прямой ссылке на новые сайты.

[5] Источники

[A] "The Anatomy of a Large-Scale Hypertextual Web Search Engine" Концепция Googlebot, Sergey Brin, Lawrence Page, Stanford University

URL: http://infolab.stanford.edu/~backrub/google.html

[B] "The Web Robots Database"

URL: http://www.robotstxt.org/wc/active.html

[C] "Web Security FAQ", Lincoln D. Stein

URL: http://www.w3.org/Security/Faq/www-security-faq.html

[D] "A Standard for Robot Exclusion", Martijn Koster

URL: http://info.webcrawler.com/mak/projects/robots/norobots.html

[EOF]

Полностью предотвратить автоматизированное злоупотребление, не умея предвидеть и классифицировать действительное намерение конкретного пользовательского действия, почти невозможно, а такое умение вряд ли появится в ближайшем будущем. Тем временем число систем, основанных на автоматическом взаимодействии с другими сущностями, растёт каждый год. Поэтому сегодня проблема, вероятно, ещё интереснее, чем в момент написания статьи, особенно после появления в Интернете за последние несколько лет всё более сложных и многочисленных червей.

Есть ли у этой истории мораль или ясный вывод? Не особенно. Однако важно помнить: машины не всегда действуют от имени операторов, даже если они явно не взломаны и ими не злоупотребляют напрямую, превращая во враждебные инструменты. Определить намерение и место, где возникло желание совершить вредоносное действие, бывает чрезвычайно трудно, как вы увидите в последующих главах.

Часть II. Безопасная гавань

Об угрозах, скрывающихся между компьютером и Интернетом.

Глава 5. Мигающие огоньки

Где мы приходим к выводу, что красивое тоже бывает смертельно опасным, и учимся считывать данные со светодиодов.

Первая часть книги была посвящена различным проблемам в устройстве системы ввода данных. Они ограничивались восстановлением ввода через наблюдение за внешне не связанными с ним поведенческими шаблонами пользователя, имевшего локальный доступ к системе. Но по мере того как информация движется дальше к адресату и покидает исходную систему, область её раскрытия расширяется, а проблемы становятся ощутимее.

Вторая часть книги рассматривает некоторые проблемы, возникающие, пока данные ещё остаются неподалёку, но уже покинули исходную систему - за мгновение до входа в Интернет. Область раскрытия здесь примерно ограничена физической территорией локальной сети и её непосредственным окружением. Для атаки на этом уровне нужна точка наблюдения рядом с источником, но доступ к самой системе не требуется.

Конкретная проблема этой главы несколько отличается от предыдущих: теперь утечка проявляется на аппаратном уровне, как в TEMPEST, но имеет иную природу. Красота явления и лёгкость наблюдения без специального оборудования с избытком оправдывают более пристальное знакомство.

Искусство передачи данных

Необходимость компьютеров общаться с другими электронными устройствами была очевидна с самого начала практических вычислений, как и трудность надёжно и недорого реализовать эту задачу. Внутренней связью машины мы можем управлять: предоставить всем основным компонентам щедрые, специально подогнанные интерфейсы требуемой пропускной способности, поддерживать точные характеристики сигналов и использовать общие эталонные часы для всех операций, чтобы получатель всегда знал, когда слушать, а отправитель - когда передавать. Но связь на больших расстояниях или с устройствами, оснащёнными дешёвыми универсальными интерфейсами, - иная задача. Компьютер вынужден пользоваться средой, которая обычно не даёт той свободы, к какой мы привыкли внутри одной машины.

В действительности ситуация прямо противоположна. Покупатель ожидает простых, удобных, практичных и дешёвых решений, а требование соединять компьютеры 100-долларовым трёхдюймовым кабелем из 100 проводов едва ли могло победить. Простота необходима. В основе почти любого внешнего канала лежит последовательная передача битов, которые лишь после обратной сборки и группировки образуют числа, текстовые строки и другие порции данных, естественные для машинной среды отправителя или получателя. В самом, казалось бы, простом и очевидном случае две машины или устройства, соединённые парой проводов, обмениваются сведениями, задавая на одном проводе высокое или низкое напряжение относительно другой, опорной линии - либо используя любые другие различимые сигналы или состояния. Так они передают последовательные биты с определённой частотой, которую оба устройства должны поддерживать достаточно близкой и синхронной.

Даже в столь простой конструкции сразу возникает несколько проблем. Во-первых, у устройств нет общих эталонных часов. Оба имеют внутренние кварцевые генераторы, но никакие два доступных генератора не бывают настолько точными, чтобы долго поддерживать надёжную и быструю связь: мешают небольшие производственные отклонения, помехи и другие физические условия. А последовательная связь требует строгой синхронизации.

Прямолинейная схема кодирования битов, обычно называемая NRZ, *Non-Return to Zero*, просто выдаёт один сигнал - напряжение - для 0 и другой для 1. В такой системе оба узла легко синхронизируются, если значения регулярно меняются: достаточно обнаружить спадающий или нарастающий фронт, принять его за приблизительный ориентир и подстроить собственные часы. Но в длинной последовательности единиц или нулей принимающей стороне трудно точно определить число переданных битов. Даже небольшой уход часов вызывает ошибки, а компенсировать его во время передачи неизменной последовательности невозможно.

Очевидное решение - перемежать данные отдельным различимым временным сигналом - не всегда самое удобное и эффективное: рост сложности и снижение пропускной способности часто воспринимаются как досадная помеха.

Для эффективного решения многие системы применяют манчестерское кодирование, также известное как двухфазный код. Алгоритм манчестерского кодирования, показанный вместе с NRZ на рисунке 5-1, кодирует данные фронтами, а не уровнями сигнала. Упомянутое исходное NRZ использует внутренние часы, чтобы с постоянной частотой измерять уровни напряжения, считая низкий уровень двоичным 0, а высокий - 1. Манчестерское кодирование, напротив, переносит данные в переходах от низкого напряжения к высокому или обратно. В такой конструкции переход к высокому уровню обозначает двоичную 1, а к низкому - 0.[1]

Самого такого кодирования всё же недостаточно, хотя синхронизация часов ему не нужна. Невозможно закодировать две двоичные единицы или два нуля: нельзя дважды перейти от низкого напряжения к высокому, не вернувшись на полпути к низкому, и наоборот. Поэтому переходы вскоре после спадающего или нарастающего фронта игнорируются. Система может кодировать несколько нулей или единиц, возвращаясь к прежнему напряжению в середине такта. Для управления периодом «нечувствительности» после перехода нужен простой одновибраторный интервальный таймер.

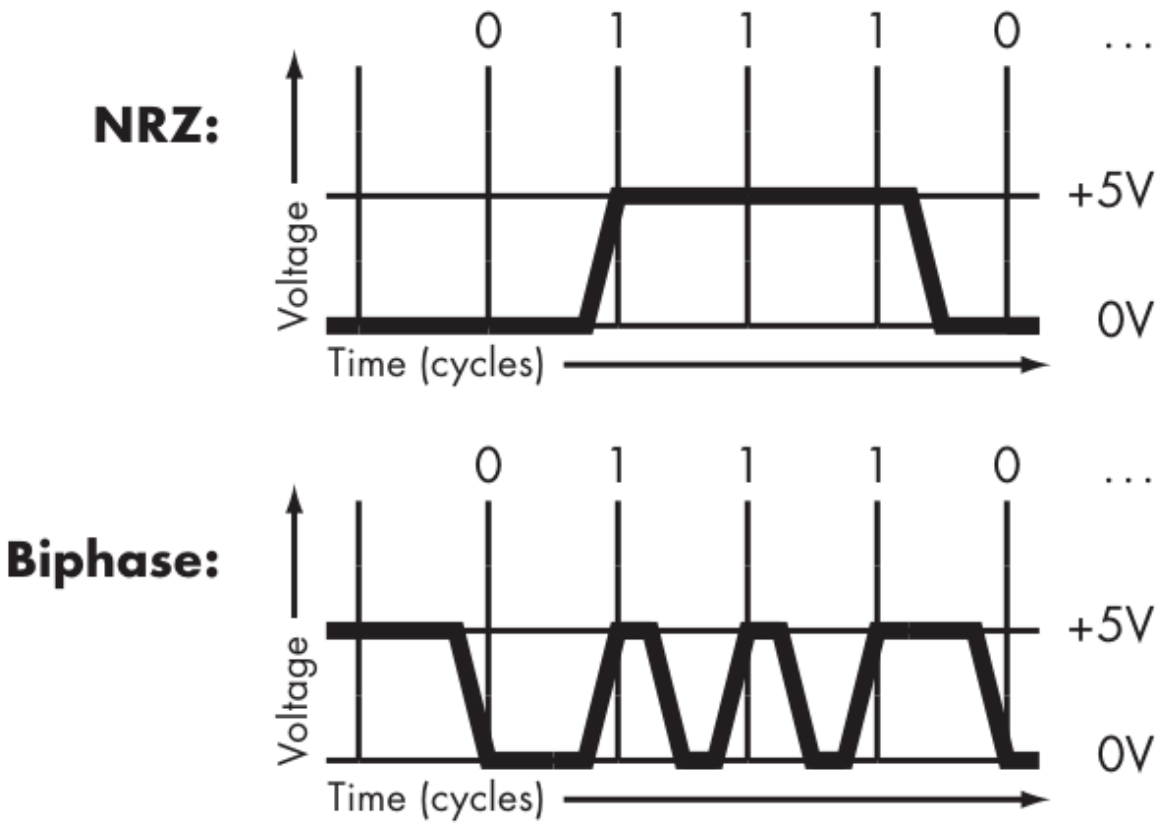


Рисунок 5-1. Кодирование передачи по последовательной линии: NRZ и двухфазное, манчестерское. Для последовательности `0 1 1 1 0 ...` *показано изменение напряжения от 0 до +5 В по тактам времени.*

Конструкцию последовательной линии с описанной самосинхронизирующейся схемой часто расширяют до полнодуплексной связи, где обе стороны говорят одновременно. Для этого применяют две отдельные линии - передачи и приёма, сокращённо Tx и Rx, - либо развитые методы обнаружения и подавления эха, отличающие собственный сигнал от данных другой стороны. Некоторые среды требуют или допускают более сложные схемы сигнализации, например передачу более одного бита за такт. Но основной принцип связи остаётся тем же, а манчестерское кодирование по минимальному числу проводов - часто двум - широко распространено во всей области.

Вооружившись основами последовательной связи по «паре проводов», рассмотрим два заметных сетевых примера, узнаем, как они обмениваются данными внутри, и посмотрим, как эти сведения могут незаметно для пользователя утекать третьим сторонам.

От вашей электронной почты к громким звукам... и обратно

Самое распространённое устройство дальней компьютерной связи - несомненно, модем. Впервые появившись в 1950-х годах для обслуживания и управления некоторыми видами военного оборудования в удалённых местах, модем принёс Интернет массам. Сегодня его часто считают несколько устаревшим, но он породил множество развитых технологий, например доступные высокоскоростные системы DSL, Digital Subscriber Line, и кабельные модемы. Все эти устройства используют хитроумные варианты одного набора методов связи по телефонным линиям или другим неспециализированным аналоговым средам посредством слышимых либо неслышимых сигналов. Исследования, вложенные в улучшение модемов, также расширили наше понимание

многочисленных крупных проектных проблем электроники вообще и компьютеров и сетей в частности. Поэтому знание принципов работы модема необходимо для изучения других, возможно более современных способов дальней передачи данных.

Повсеместность телефонной линии делает её естественной средой компьютерной связи. Телефонные линии есть почти везде, а телефонные системы превосходно маршрутизируют вызовы, позволяя почти без усилий достичь практически любого места. Есть лишь небольшая оговорка: телефонные линии предназначены для человеческого голоса, который передаётся как форма сигнала в узком диапазоне частот, обычно не шире нескольких килогерц. Эти частоты записывались изменениями напряжения в паре проводов и проходили через ряд аналоговых повторителей и усилителей, поэтому стандарт качества был не особенно высок. Достаточно, чтобы люди слышали и понимали друг друга; поскольку человеческий мозг - великолепная система фильтрации и обработки сигналов, случайный шум или колебания громкости мало кого беспокоили, по крайней мере пока позднее абоненты не стали разборчивее.

Компьютеры, напротив, обычно проектируются для обмена двоичной информацией, закодированной довольно точными уровнями напряжения по хорошо спроектированным коротким линиям с хорошими характеристиками сигнала и низкой ёмкостью. Это полная противоположность длинным, плохо экранированным телефонным линиям с неудовлетворительными характеристиками. Кроме того, компьютерам нужно говорить гораздо быстрее и гораздо больше, чем обычно говорят люди. Поэтому перед разработчиками модемов стояла - и это огромное преуменьшение - трудная задача. Требовалось кодировать биты не только так, чтобы эффективно передавать их удалённой системе по проводу, что немного упростило манчестерское кодирование, но и как слышимые сигналы, которые можно точно различить на другом конце независимо от зачастую совершенно непредсказуемых изменений напряжения и прочих искажений передачи.

Пришлось применить устойчивые алгоритмы исправления ошибок и переменные скорости, чтобы компенсировать плохое качество линии, случайные наводки, грузовики над проложенным в земле телефонным кабелем, птичьи гнёзда на столбах и так далее. Разработчики кивнули, почесали головы и всего-то примерно через 40 лет дали нам доступный и довольно быстрый способ связи между компьютерами. Кратко посмотрим, как он развивался и как технология созревала - в сущности оставаясь прежней - на протяжении следующих десятилетий.

История коммерческих модемов и их стандартизации началась в 1960-х годах с двух стандартов, Bell 103/113 и V.21. Оба обеспечивали удивительное для своего времени полнодуплексное соединение на 300 бод, битов в секунду, применяя частотную манипуляцию, FSK. За загадочным термином скрывается простейшая схема кодирования: два разных тона обозначают разные значения, одна частота - «низкое», другая - «высокое». Преимущество слышимых частот перед другими сигналами огромно: только их телефонная система способна передавать достаточно хорошо, ведь именно для этого она создана. Все остальные сигналы в лучшем случае обречены исказиться до неузнаваемости прежде, чем достигнут другого конца провода, а в худшем немедленно отсекаются полосовыми фильтрами где-нибудь по пути.

Помимо FSK, стандарты Bell 103/113 и V.21 делили доступный телефонной линии диапазон частот надвое. Один модем - вызывающий - кодировал низкое значение частотой 980 Гц, а высокое - 1 180 Гц. Ответившая сторона использовала верхнюю часть спектра: соответственно 1 650 и 1 850 Гц. Зачем такое деление? Телефонная линия - всего лишь пара проводов, по которой два устройства могут передавать одновременно в полнодуплексном режиме, но только если способны справиться с наложением собственных передач. При полнодуплексной связи каждое устройство должно отличить свой сигнал от принимаемых данных и отфильтровать его. Иначе каждому пришлось бы молчать, пока говорит другой конец, то есть работать в симплексном режиме, что сильно ухудшило бы и без того не особенно впечатляющую пропускную способность. Разделение частот фактически заставляет телефонную линию переносить два разных «голоса», обеспечивая одновременную связь без столкновений.

Модемам потребовалось ещё 25 лет для следующего шага в верном направлении. Новые крупные стандарты Bell 212A и V.22 совершили большой скачок: отказались от частотной манипуляции в пользу дифференциальной фазовой манипуляции, DPSK. Вместо изменения частоты волны DPSK сдвигает её фазу, обозначая разные значения.

По существу, фазовая манипуляция вносит минимальный временной сдвиг, или задержку, из-за которой выходной звуковой сигнал немного расходится по времени с исходной опорной волной, сохраняя точно такую же форму, как показано на рисунке 5-2.

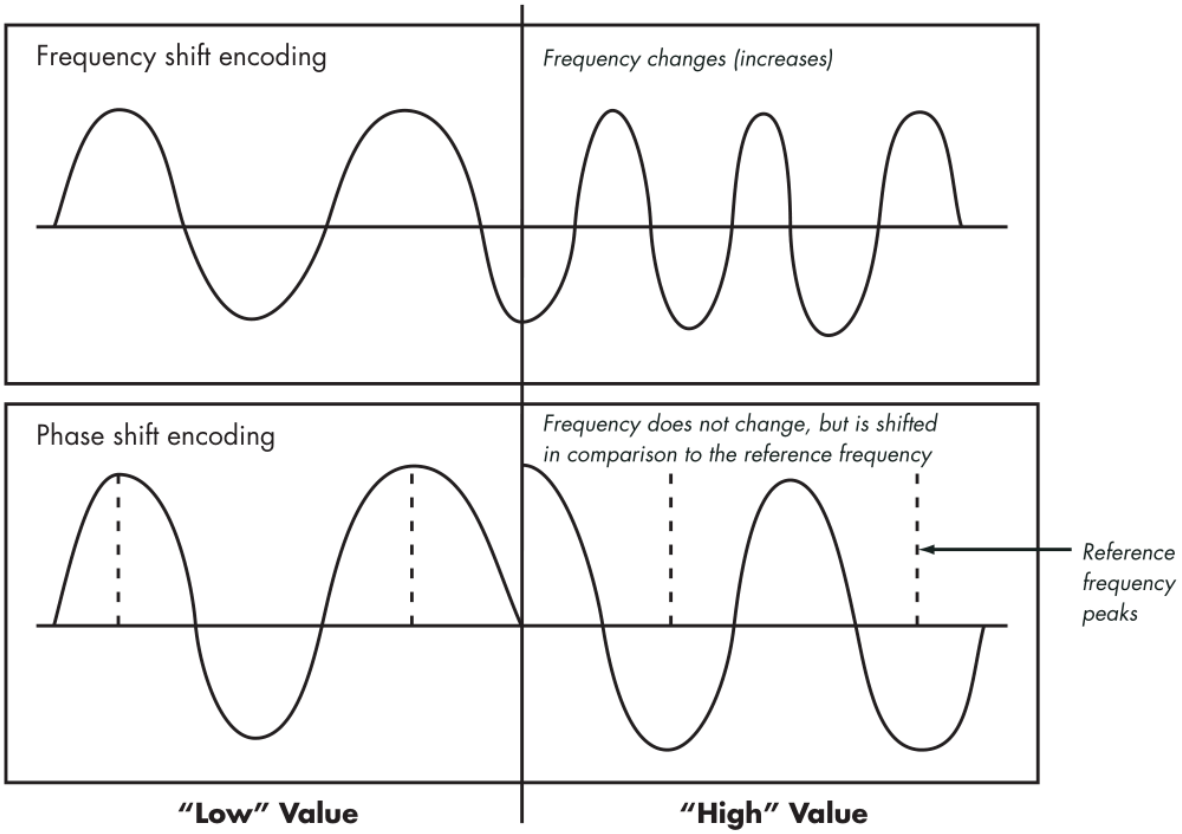


Рисунок 5-2. Частотный сдвиг и фазовый сдвиг. При частотной манипуляции меняется частота между «низким» и «высоким» значениями. При фазовой частота не меняется, но сигнал сдвигается относительно опорных пиков.

Величина фазового сдвига выражается в градусах - это отсылка к его влиянию на тригонометрические функции: `y = sin(x)`, сдвинутая на 90°, полностью совпадает с `y = sin(90° + x)`. Сдвиг 360° означает смещение на целую длину волны, которое просто вновь синхронизирует волны и никак не влияет на форму сигнала. Соответствие разных фазовых сдвигов показано слева на рисунке 5-3.

Когда обе стороны синхронизированы и умеют сравнивать принятый по кабелю сигнал с ожидаемой формой волны, закодированные данные легко извлечь. Дифференциальная схема сравнивает два сигнала, вычитает их и точно определяет фазовый сдвиг относительно опорного сигнала, как показано справа на рисунке 5-3.

Новый стандарт воспользовался и более развитым методом кодирования. Вместо двух чередующихся сигналов для передачи нулей и единиц, как прежде, V.22 кодирует целые дибиты - на жаргоне пары битов. Два бита за раз кодируются четырьмя значениями фазового сдвига. Величины для всех возможных значений выбраны так, чтобы равномерно и, по возможности, на максимальном расстоянии располагаться по всему спектру 360°, а значит, легко отличаться друг от друга, как в таблице 5-1.

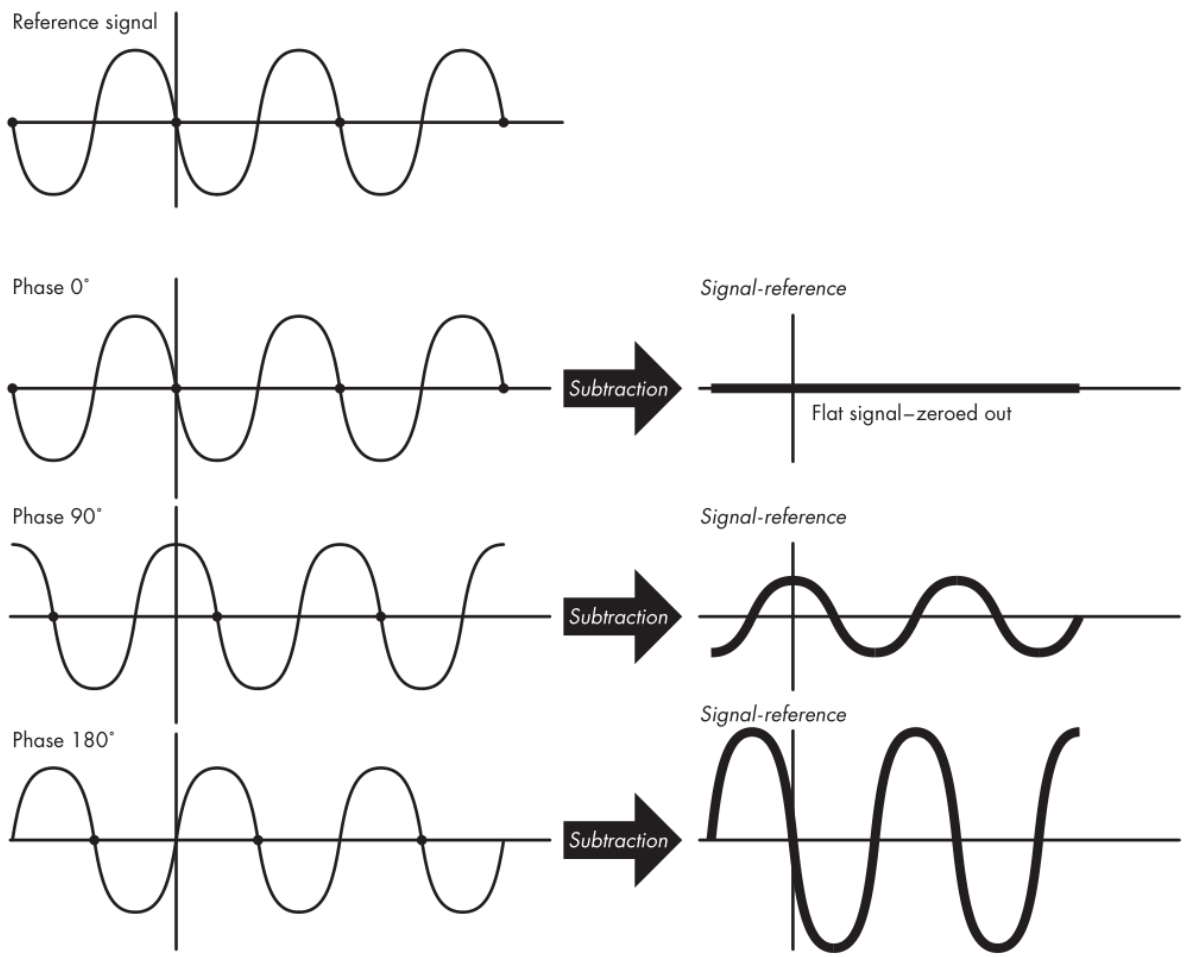


Рисунок 5-3. Фазово-сдвинутые сигналы и результат вычитания опорной формы волны. Показаны опорный сигнал, фазы 0°, 90° и 180° и результаты сигнал - опора*; при фазе 0° вычитание даёт плоский нулевой сигнал.*

Таблица 5-1. Кодирование двух битов данных, дибита, фазовыми сдвигами

Дибит	**Фазовый сдвиг**
00	90°
01	0°
10	180°
11	270°

Дибиты заметно повысили скорость передачи до 1 200 бод без увеличения физической частоты модуляции самого сигнала. Каждый отдельный писк переносил вдвое больше информации - вдвое больше битов.

> **Примечание.** Теоретически расширенный алфавит, то есть составные единицы сигнала, подобные дибитам, с более чем двумя состояниями и потому кодирующие более одного бита, можно использовать и с FSK, но это несколько сложнее. Сигналы FSK должны избегать субгармоник и других частот, особенно склонных к искажениям при прохождении через телефонные системы, что сильно ограничивает множество возможных состояний.

> Преимущество DPSK над FSK состоит в фиксированной частоте, про которую известно, что она создаёт меньше всего проблем при передаче, а потому надёжнее работает на высоких скоростях.

В следующие несколько лет темп исследований несколько вырос и появилось множество новых стандартов. V.22bis развил идею широкого сигнального алфавита, объединив DPSK с модуляцией амплитуды - громкости - для построения двумерного множества из 16 возможных значений. Переход от измеренного сигнала к двоичным значениям выражался двумерной таблицей. Чтобы получить соответствующее сигналу значение, сначала по измеренному фазовому сдвигу находили столбец, затем по амплитуде - строку. Упрощённый, но аналогичный пример два на четыре приведён в таблице 5-2.

Чтобы окончательно всё запутать, новый подход назвали квадратурной амплитудной модуляцией, QAM. QAM снова позволила перейти с 1 200 до 2 400 бит/с, не повышая фактическую скорость модуляции, а расширяя число значений одного атома сигнала.

Таблица 5-2. Двумерное кодирование трёх битов двумя разными параметрами сигнала

Амплитуда	**Фаза 0°**	**Фаза 90°**	**Фаза 180°**	**Фаза 270°**
Низкая	000 (0)	001 (1)	010 (2)	011 (3)
Высокая	100 (4)	101 (5)	110 (6)	111 (7)

Следующим крупным эволюционным шагом стал V.32. Эта конструкция первой ввела новую идею: вместо разделения частот она применяла развитую схему подавления эха[2], которая обнаруживала и вычитала переданный самим устройством сигнал из принятых по проводу данных. Так оба устройства - отправитель и получатель - получили весь частотный спектр вместо половины, сохранив полнодуплексный режим.

Развитие продолжилось, и вскоре появился протокол V.34. Безопасная скорость чередования сигнала до возникновения чрезмерных искажений за прошедшие годы заметно не изменилась, но стандарт оказался значительно быстрее предшественников. V.34 достигает пропускной способности 28 800 бод, которую производители иногда немного повышают до неофициальных 33 600 бод, или 33,6 Кбит/с. При этом отправляется лишь примерно 2 500-3 500 образцов сигнала - символов алфавита - в секунду, но четыре разные схемы кодирования объединяются в четырёхмерную структуру с 1 664 возможными состояниями, позволяя передать до 41 бита одновременно. Как оказалось, важна не грубая скорость, а умение пользоваться имеющимися средствами.

Широко считается, что стандарт V.34 и его производные приближаются к теоретическому пределу передачи данных по телефонной системе, рассчитанной на голос. Учитывая распространённость модемов 56 Кбит/с, утверждение кажется странным, но есть загвоздка: такие устройства достигают скорости совершенно иначе, чем аналоговые решения. С момента появления первых модемов большинство телефонных систем перешло с аналоговой технологии на цифровую, и теперь большинство провайдеров коммутируемого доступа подключает свои системы непосредственно к цифровой телекоммуникационной инфраструктуре. Поэтому провайдер может вернуться к самому очевидному, но прежде невозможному решению: при отправке данных абоненту менять напряжение линии, а не сдвигать частоту.

Сигнал изначально переносится как цифровые данные и проходит по закопанным медным линиям лишь до ближайшего объекта телефонной компании. Поэтому проблем качества практически нет, а единственным пределом остаётся заложенная в оборудование телефонной системы пропускная способность для голоса. При 8 000 символов в секунду и гораздо меньшем алфавите - обычно

около 128 символов, или уровней напряжения - данные можно отправлять абоненту цифровой телефонной системы по качественному проводу с модемом 56 Кбит/с быстрее обычного. Но обратная передача всё ещё выполняется по старинке и значительно медленнее. Таким образом, модем является 56-килобитным лишь частично и только при подходящих условиях.

Наши дни

Со времени появления модемной технологии изменилось не так много. Вместе с протоколами передачи развивались средства исправления ошибок и перехода на пониженную скорость, необходимые для надёжной связи, когда любимый четвероногий питомец решает погрызть телефонный кабель. Возникли джунгли стандартов: V.42 дал базовую реализацию циклического избыточного кода, CRC; MNP-1 - MNP-4 - закрытые алгоритмы исправления ошибок; V.42bis и MNP-5 - проверку целостности вместе со сжатием и так далее. Но настоящая революция ещё впереди.

Или уже произошла? Можно возразить, что DSL и кабельные модемы - революционные технологии, изменившие мир. Я готов поспорить: в действительности они очень похожи на старших родственников, обычные модемы. Единственное существенное различие - другой узел, сервер всех соединений, переместился из далёкого города провайдера в ближайший объект местной телефонной компании, и к нему можно напрямую подключиться медным проводом из дома или офиса абонента. Поскольку прямое соединение не проходит через иное оборудование, устройства могут использовать высокие неслышимые частоты и более тонкие сигналы, которые телефонная сеть иначе исказила бы или вообще не передала. Старый добрый модем, напротив, был строго ограничен узким диапазоном слышимых частот и сигналами, для которых предназначалась телефонная система и которые она хорошо переносила. Во многих отношениях устройствам DSL намного проще.

Как видим, модем действительно очень сложен в проектировании; потому нам и потребовались десятилетия, чтобы пройти путь от громоздких дорогих устройств на 300 бод до современного состояния.[1] Удивительно, но все эти устройства умеют общаться друг с другом - даже с аппаратами на десять лет старше и на давно забытых минимальных скоростях. Кроме того, обычно они знают все стандарты своего времени, включая десятки вариантов и ответвлений каждого. Разве это не делает модемы ещё большим чудом компьютерной инженерии?

Но кто дёргает за ниточки?

Иногда модем - это всего лишь модем

Разумеется, связью между модемами история не начинается и не заканчивается. Сам модем - довольно инертное промежуточное устройство, едва ли годное даже как пресс-папье. Чтобы приносить пользу, он должен общаться с компьютером, получать команды и обмениваться данными, даже если нужен лишь для чего-то столь слабого, как случайное блуждание по Вебу. Внутренним модемам легко: ISA, Integrated Systems Architecture; PCI, Peripheral Component Interconnect; PCMCIA, PC Memory Card International Association, и другие специализированные шины предоставляют быстрые и щедрые параллельные интерфейсы, делая связь почти тривиальной.

Внешним модемам, аналоговым или DSL, приходится идти трудным путём последовательной линии. Большинство аналоговых модемов использует известный последовательный протокол RS-232, в 1990-х переименованный в куда более выразительный EIA/TIA-232-E;[2] многие новые устройства применяют USB, Universal Serial Bus. Приближаясь к сценариям раскрытия информации, нужно увидеть и то, что происходит с данными между модемом и компьютером,

поскольку это играет решающую роль в атаке.

Внешние модемы вынуждены нечеловеческими средствами общаться не только с удалённой системой, но и с локальной машиной. Однако благодаря близости компьютера и тому, что цифровые интерфейсы вроде RS-232 изначально проектировались для компьютеров, этот этап всё же намного проще модуляции и демодуляции телефонной линии, которыми прославились битовые модемы.

RS-232 довольно прямолинейно применяет биполярное кодирование данных по двум отдельным линиям и дополняет его набором управляющих линий NRZ. Чтобы жизнь не была слишком простой, RS-232 содержит множество возможностей линии и протокола, затрудняющих реализацию с нуля: асинхронную природу, широкий диапазон настроек и скоростей, необычные уровни напряжения. Но даже со всем этим RS-232 и близко не подходит по сложности к задаче разработчика, уже имевшего дело с модуляцией сигнала по телефонным линиям.

USB, напротив, пытается стандартизировать и унифицировать последовательный интерфейс. Для соединения компьютера с устройством ему нужна более развитая схема, чем RS-232, помимо прочего из-за более высокого уровня абстракции и поддерживаемых скоростей, однако USB универсален, откуда и название, и содержит меньше странностей и унаследованных особенностей.

Наконец, распространённый способ связи с локальными устройствами - Ethernet, механизм, несколько похожий на USB, но возникший раньше. Рассмотрим теперь Ethernet; уверен, в конце концов все эти протоколы связи встретятся в одной точке.

Управляемые коллизии

Сети Ethernet по существу представляют собой развитую разновидность многосторонней последовательной линии.[3] Сеть состоит из нескольких компьютеров, соединённых общей средой - в простейшем виде ничего сложного, всего лишь пара вполне обычных проводов. Когда сетевое устройство пользуется средой, оно подаёт на провод определённое напряжение, а остальные подключённые системы истолковывают данные, измеряя напряжение. Набор проверок не даёт устройствам одновременно использовать линию и обеспечивает плавное восстановление после происшествия. Даже с учётом такой возможности базовая конструкция невероятно проста в сравнении с модемами.

Проблему одновременного разговора двух сторон решает стандарт множественного доступа с контролем несущей и обнаружением коллизий, Carrier Sense Multiple Access with Collision Detection, CSMA/CD, который служит основным механизмом управления всей связью Ethernet. Перед отправкой данных каждое подключённое устройство следует процедуре CSMA: проверяет электрические свойства среды, чтобы узнать, не пользуется ли кабелем кто-то другой. Если передачи нет, устройство переходит к отправке и вещает данные всем присутствующим.

В этой фазе данные идут по проводу как последовательность битов с биполярным кодированием. Трафик содержит заголовок со всеми необходимыми сведениями об отправителе и получателе, а также правильную контрольную сумму, которая защищает целостность данных от внутренних и внешних помех, четвероногих или иных. Сетевой интерфейс, считающий себя представителем получателя - предположительно потому, что адрес назначения пакета совпал с его уникальным аппаратным адресом MAC, хранящимся на плате, - должен принять трафик и проверить контрольную сумму. Все остальные стороны должны игнорировать кадр. Естественно, если они этого не делают - почти любую плату можно приказом заставить не делать, - пользователь сможет просматривать трафик других получателей или реагировать на него. Видно, что Ethernet создавался в духе далеко идущего доверия и альтруизма - подход благородный, но рискованный.

Два устройства Ethernet вполне могут начать отправку точно одновременно, хотя всего несколько микро- или наносекунд назад оба проверили отсутствие другой передачи. Если это случится, беда неизбежна. Две передачи смешаются и исказятся, а отправленные данные должны провалить проверку контрольной суммы у получателя... Или не должны?

Контрольной суммы из спецификации кадра Ethernet обычно достаточно для проверки точности передачи, но она может оказаться не особенно эффективной при насыщенной линии и сотнях или тысячах коллизий за короткое время: сумма достаточно мала, чтобы время от времени случайно совпадать. Законы вероятности говорят, что у некоторых повреждённых пакетов чисто случайно окажется та же контрольная сумма, что и у исходного. Но даже если забыть о недостатках контрольной суммы, коллизии всё равно нужно останавливать как можно раньше. Если позволить им бушевать, своевременную повторную передачу искажённых и отброшенных кадров уже не гарантировать: отправитель передал кадр, не заметив проблемы, а получатель не получил ничего, хотя бы отдалённо похожего на полезный пакет.

Решение даёт вторая часть стандарта - обнаружение коллизий, CD. Спецификация требует, чтобы отправитель наблюдал за сетевой линией, пока сообщает остальным свои сведения. Попытка другой стороны заговорить одновременно должна быть обнаружена простым измерением электрических свойств линии, а передача - немедленно прервана. Устройство также отправляет специальный сигнал затора, чтобы оба кадра - отправляемый и помешавший ему - безусловно отбросили, даже не проверяя контрольную сумму. Получатель должен заметить сигнал и прекратить приём обрабатываемых данных. После каждой попытки устройство бездействует постепенно увеличивающийся и желательно первоначально случайный промежуток, называемый отсрочкой повторной передачи, чтобы снизить вероятность следующей коллизии.

> **Примечание.** Забавный факт: механизм сигнала затора накладывает необычное требование на протокол. Все кадры должны иметь минимальную (!) длину, рассчитанную так, чтобы сигнал успел возникнуть и распространиться до всех машин до завершения передачи. Для очень коротких кадров времени может не хватить. Поэтому отправитель обязан искусственно дополнять все исходящие передачи.

На рисунке 5-4 показана точная последовательность событий типичной коллизии. Отправитель A хочет передать данные получателю, но замечает другую передачу и решает дождаться её окончания. Затем он готовится к отправке, но, к несчастью, отправитель B делает то же самое, и почти одновременно оба заключают, что линия свободна.

Оба пытаются передавать, данные искажаются, оба обнаруживают чужой сигнал и быстро отправляют код затора, предписывая получателю проигнорировать кадр. Наконец, оба отправителя отступают на случайный промежуток и, будем надеяться, в следующий раз не стартуют одновременно.

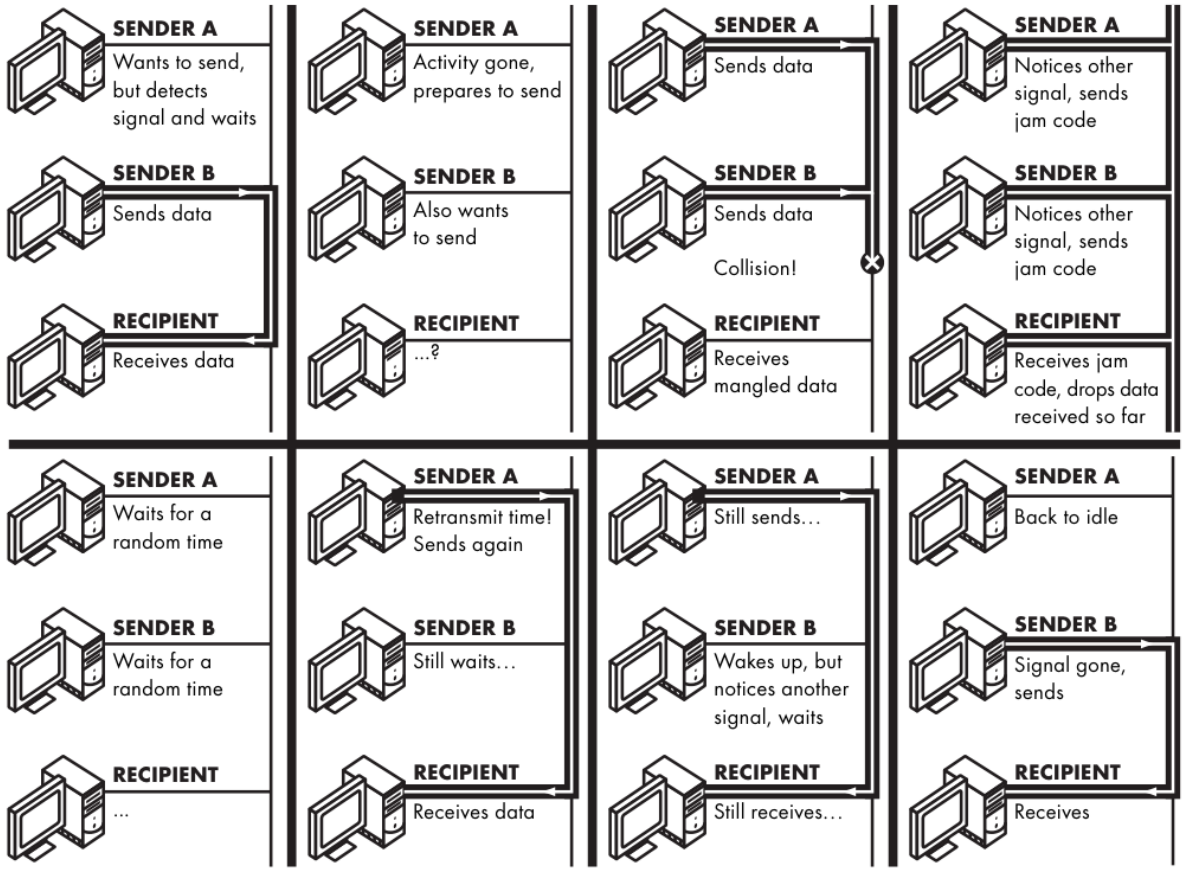


Рисунок 5-4. Стадии типичного обмена Ethernet. Отправитель A ожидает освобождения линии; A и B одновременно начинают передачу, обнаруживают коллизию и посылают код затора; получатель отбрасывает данные; стороны выдерживают случайные задержки, после чего повторная передача проходит успешно.

За кулисами: путаница проводов и наша борьба с ней

Протокол Ethernet - не особенно масштабируемая и не особенно изящная конструкция, но удивительно мощная и простая в развёртывании. Он позволил почти где угодно строить дешёвые одноранговые сети на коаксиальном кабеле и потому стал фактическим стандартом, вытеснив многие другие сетевые архитектуры - иногда превосходящие его, но более дорогие или закрытые.

Разумеется, простой Ethernet по коаксиальному кабелю имел пределы и недостатки. В сущности это был длинный провод с устройствами, подключёнными в разных местах, и резисторами на обоих концах - не та вещь, за обслуживание которой хочется отвечать в большом офисе. Простая и трудно диагностируемая неприятность, например замыкание терминатора, могла вывести из строя всю инфраструктуру. Более развитую, но лишь немного более дорогую замену встретили тепло.

Электронные многопортовые повторители - концентраторы - позволили без особых усилий прокладывать витую пару: кабели Cat-3 и Cat-5 с разъёмами RJ-45. Достаточно было соединить машину проводом с чёрным ящиком, и все остальные подключённые к нему устройства могли с ней общаться, почти не задумываясь об электрических проблемах и не рискуя потерять всю сеть из-за отказа одного кабеля.

Концентратор по существу простой повторитель, который рассылает весь принятый на одном порту трафик на все остальные. Он позволяет строить легко перенастраиваемые и более надёжные звездообразные сети, но почти ничего больше не делает. По мере роста сети цена рассылки

каждого бита во все места и возможность говорить только одной стороне во всей сети слишком явно показывают, что простота конструкции - её главная слабость.

Решением стали коммутаторы - следующее поколение концентраторов. Они снабжены приличным процессором и памятью, стоят дороже, но в обычных условиях выполняют высокоуровневый анализ кадров Ethernet. Анализ связывает аппаратные адреса с конкретными портами и оптимизирует маршрутизацию кадров, доставляя некоторые пакеты напрямую нужному порту в одноадресном режиме вместо рассылки всем сторонам, как показано на рисунке 5-5. В крупных сетях это значительно повышает производительность.

> **Примечание.** Ещё один забавный факт: настоящие концентраторы сегодня почти вымерли. Практически все устройства 10/100 Мбит, продаваемые как концентраторы, в действительности используют базовые наборы микросхем коммутаторов: переупаковать микросхему просто дешевле, чем разрабатывать и поддерживать несколько вариантов.

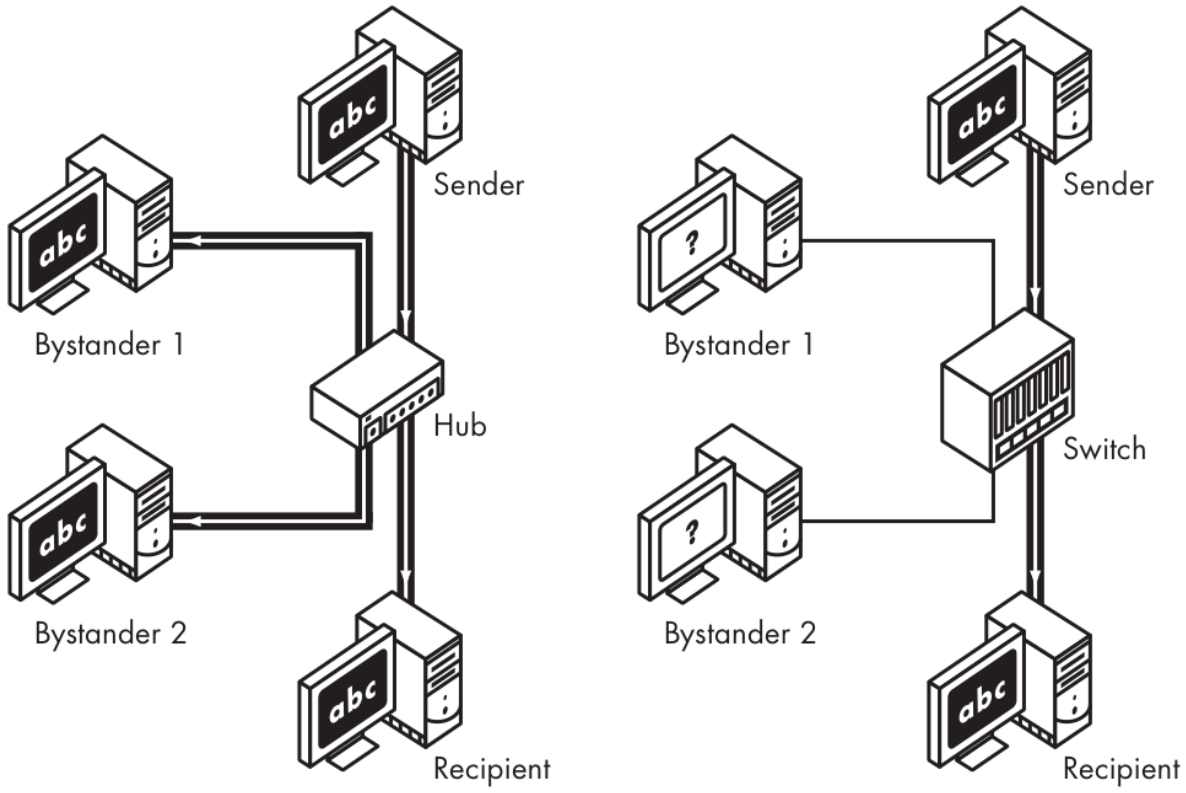


Рисунок 5-5. Концентраторы и коммутаторы в локальных сетях. Концентратор передаёт сообщение abc *от отправителя получателю и обоим сторонним наблюдателям; коммутатор доставляет его только получателю.*

Полагаю, теперь вы спрашиваете: куда, чёрт возьми, всё это ведёт? Как модемы связаны с раскрытием информации? Какое значение в этом контексте имеют последовательные линии? При чём здесь Ethernet? И что за мигающие огоньки?

Рад, что вы спросили. Я как раз перехожу к ответу - на последний вопрос.

Мигающие огоньки в системах связи

Исторически почти все компьютеры размером с холодильник оснащались множеством выставленных напоказ диагностических интерфейсов. Среди них были массивы крошечных ламп,

показывавших, помимо прочего, неясные свойства внутреннего состояния машины: регистры, флаги основного процессора или сведения о том, покормили ли сегодня живущего внизу кота. По мере роста надёжности и компактности компьютеров, когда обычному пользователю больше не требовалось понимать внутренности машины для эффективной работы, лампы начали исчезать. Помогли и постоянно растущие тактовые частоты: человек уже не мог получить осмысленные сведения из визуального сигнала, менявшегося тысячи или миллионы раз в секунду.

И всё же в некоторых применениях огоньки сохранились. Почти на всех сетевых устройствах есть светодиоды, LED, на передней или задней панели. Они обеспечивают диагностику линии: показывают, исправен ли модуль или разъём, подключена ли другая сторона, передаются ли данные и так далее. Огоньки не только диагностический инструмент. Их гипнотические узоры странно привлекательны, а загадочность сеет неопределённость, страх и почтение в сердцах непосвящённых, входящих во владения серверной.

Термины *blinkenlights* и *blinkenlichten* ещё с тёмных веков вычислительной техники обозначают горячо любимый институт диагностических LED на компьютерном оборудовании, в успокаивающем зелёном свете которых компьютерщик купается долгими одинокими ночами за терминалом. Название пришло из забавной шуточной записки на псевдонемецком - сама она пародировала другую, некомпьютерную шутку времён Второй мировой войны, - вывешенной в 1950-х годах в лабораториях IBM. Позднее записка распространилась по большинству серверных и лабораторий информатики мира. В *Hacker's Dictionary* Эрика Рэймонда она цитируется так:

```
ACHTUNG!
ALLES LOOKENSPEEPERS!
Alles touristen und non-technischen
looken peepers! Das computermachine
ist nicht fuer gefingerpoken und
mittengrabben. Ist easy schnappen
der springenwerk, blowenfusen und
poppencorken mit spitzensparken.
Ist nicht fuer gewerken bei das
dumpkopfen. Das rubbernecken
sichtseeren keepen das cotton-pickenen
hans in das pockets muss; relaxen und
watchen das blinkenlichten.
```

> **Приблизительный смысл шуточной записки:** «Внимание всем глазеющим! Всем туристам и нетехническим наблюдателям: компьютерную машину нельзя тыкать пальцами и хватать руками. Так легко сломать механизм, сжечь предохранители и устроить искры. Работать с ней не следует глупцам. Зеваки должны держать свои руки в карманах, расслабиться и смотреть на мигающие огоньки».

Оборудование связи - одна из последних областей, где мигающие огоньки сохранились и процветают. Но это не всё. Почти все такие устройства применяют последовательные линии. Ради простоты и красоты светодиоды «активности» иногда подключаются почти напрямую, через простейшую схему драйвера, к линии передачи или приёма. Занавес опускается.

Последствия эстетики

На обнаружение проблемы потребовались десятилетия. Когда это наконец произошло в 2002 году, всё показалось настолько очевидным и простым, что нам захотелось несколько раз стукнуться головой о клавиатуру.

В исследовательской работе “Information Leakage from Optical Emanations” Джо Лагри и Дэвид А. Амфресс[4] обнаружили новый сценарий раскрытия сигнала в некоторых видах сетевого оборудования, чаще всего в модемах. Они заключили, что наблюдатель способен не просто

любоваться волшебными огоньками невооружённым глазом.

В отличие от ламп накаливания, светодиоды обычно имеют короткое время нарастания и спада, то есть включаются и выключаются почти мгновенно. Это неудивительно: высококлассные LED применяются для управления оптоволоконными линиями и другими оптоэлектронными каналами связи. Поэтому мигание светодиода, подключённого к последовательной линии передачи, часто действительно отражает отдельные биты в момент их прохождения по проводу. Если записать активность с достаточной скоростью, сведения можно восстановить как минимум с того расстояния, с которого крошечный огонёк виден невооружённым глазом или через телеобъектив.

Исследование вызвало некоторое волнение в отрасли; позднее проблему одновременно и преуменьшали, и чрезмерно раздували, вследствие чего возникла большая путаница, а изменилось совсем немного. Работа породила множество противоречивых сообщений, но её основная идея проста и поистине прекрасна. Приёмник такого сигнала создать элементарно: столь же дешёвые и распространённые аналоги светодиодов - фотодиоды и фототранзисторы - легко приобрести и столь же легко соединить с компьютером. А зона раскрытия, в отличие от большей части активности TEMPEST из главы 3, не является лишь предметом городских легенд и чисто лабораторных результатов - её можно непосредственно наблюдать и измерять.

В ходе исследования авторы провели ряд опытов и подтвердили успешный приём сигнала с расстояния до 20 метров - чуть меньше 100 футов - без дополнительной цифровой обработки. Здравый смысл подсказывает, что оценка может быть занижена, особенно при хорошей оптике. Авторы использовали объектив 100 мм с диафрагмой f/2,0, но многим фотолюбителям со среднеценовыми зеркальными камерами, SLR, доступны куда лучшие телеобъективы. Готовый расстаться с деньгами человек может купить превосходный объектив с фокусным расстоянием до 1 200 мм.

В нескольких местах работа занимает оборонительную позицию, и внимательный читатель может решить, что некоторые классифицированные устройства проблеме не подвержены. В частности, у некоторых устройств Ethernet проявляется более тонкая разновидность уязвимости, как станет видно далее в разделе о предотвращении. Но сначала посмотрим на проблему собственными - компьютеризированными - глазами.

Создание собственного шпионского оборудования...

Простота устройства слежения так и манит его построить. Здесь приведены несколько советов и приблизительных схем подключения к обычному компьютеру. Схема не особенно сложна, не требует степени магистра пайки и программы проектирования печатных плат, но минимальное владение электроникой и доля здравого смысла желательны. Внешние интерфейсы современных компьютеров довольно прочны и защищены от дурака, однако при подключении самодельных устройств особенно изобретательным способом в краткий миг безумия всегда можно повредить оборудование. Такое случалось с лучшими из нас.

Базовая конструкция предельно проста. Нужен один фототранзистор - транзистор со встроенным управляющим фотодиодом, - обычный маломощный NPN-транзистор, Negative-Positive-Negative, для дополнительного усиления сигнала, которое требуется не всегда, и набор потенциометров, возможно порядка 10 кОм для достаточной гибкости, чтобы экспериментально подтягивать напряжение вниз и управлять чувствительностью и порогами схемы. Особых требований к компонентам нет, хотя результат зависит от выбранных деталей. Фототранзистор должен хорошо реагировать на видимый свет, но сгодится любой дешёвый. Для справки: зелёный LED излучает примерно на длине волны 520 нм.

Пример схемы показан на рисунке 5-6.

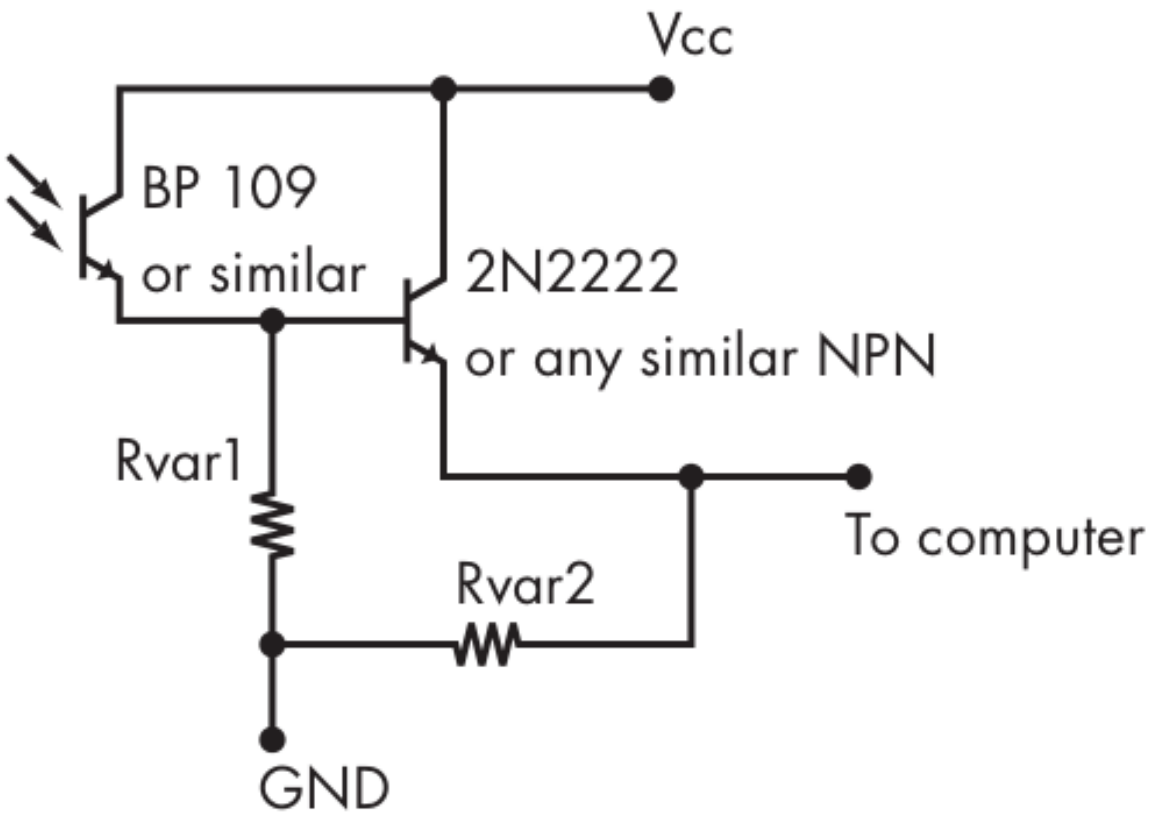


Рисунок 5-6. Простейшая схема приёмника. Фототранзистор BP 109 или аналогичный управляет NPN-транзистором 2N2222; чувствительность задают `Rvar1` *и* `Rvar2`*, схема подключена между* `Vcc` *и* `GND`*, а выход ведёт к компьютеру.*

Оптимальное рабочее напряжение схемы около 5 В, максимальный ток невелик: источника на 10-50 мА более чем достаточно. Предупреждение: источник с более высоким напряжением может повредить порт или компьютер; то же произойдёт с более мощным источником, если не ограничить ток в схеме.

> **Примечание.** Установка очень низкого сопротивления `Rvar1` или `Rvar2` может замкнуть схему. Если хочется бездумно крутить ручки, стоит добавить постоянный резистор, ограничивающий ток.

Фототранзистор необходимо экранировать от внешнего света, например заключить в непрозрачную трубку. У него нет фокусирующего механизма, поэтому дальние сигналы, кроме фонового света, он вряд ли уловит. Для первых испытаний датчик лучше полностью закрыть, имитируя темноту, а затем поднести к LED, возбуждая схему. Можно временно подключить другой светодиод между `GND` и выходом для проверки. При направлении датчика на источник света тестовый LED должен загораться, а в остальных случаях оставаться довольно тёмным.

...и его использование с компьютером

Если схема с проверочным LED работает - поздравляю, вы построили модный тестер телевизионных пультов. Дешёвые универсальные фототранзисторы охотно воспринимают инфракрасный свет, поэтому ваше творение должно «переводить» IR в видимый свет, но на этом развлечения заканчиваются. Для чего-то более полезного схему нужно соединить с компьютером.

Хороший вариант - интерфейс линейного принтера LPT, если он есть. К сожалению, этот замечательный инструмент аппаратного хакера исчезает из некоторых компактных и модных конструкций.

Изначально LPT создавался однонаправленным, только для вывода, но имеет несколько линий обратной связи состояния - «нет бумаги», «занят», «подтверждение», - через которые принтер должен был жаловаться на проблемы. В PC-совместимой системе данные легко прочесть через порт `0x379`, регистр состояния LPT1. Подключив схему к параллельному порту, можно без труда передавать сведения компьютеру. Вы вправе выбрать другой интерфейс, но LPT намного быстрее, скажем, RS-232, и не заставляет разбираться с обыденными протоколами, схемами сигнализации и необычными уровнями напряжения. В отличие от USB и некоторых современных решений, ему не нужны специальные контроллеры, реализующие довольно сложный протокол хотя бы для начала общения с PC.

> **Примечание.** LPT поддерживает двунаправленные режимы ECP и EPP, но для столь простой задачи ими обычно бессмысленно пользоваться. В однонаправленном режиме для ввода доступны четыре бита, более чем достаточно; переключение в двунаправленный EPP или ESP даёт ещё четыре.

Линию состояния выбирайте сами. В таблице 5-3 приведена распиновка разъёма DB25 принтерного порта. Затенённые в исходнике строки можно использовать для ввода.

Таблица 5-3. Распиновка LPT

Контакт	Имя	Функция
1	Strobe	Управляющий выходной бит 0
2	D0	Выходной бит данных 0
3	D1	Выходной бит данных 1
4	D2	Выходной бит данных 2
5	D3	Выходной бит данных 3
6	D4	Выходной бит данных 4
7	D5	Выходной бит данных 5
8	D6	Выходной бит данных 6
9	D7	Выходной бит данных 7
10	ACK	Входной бит состояния 2
11	Busy	Входной бит состояния 3
12	Paper Out	Входной бит состояния 1
13	Select In	Входной бит состояния 0
14	Autofeed	Управляющий выходной бит 1

Контакт	Имя	Функция
15	Error	Вход состояния, не используется
16	Init	Управляющий выходной бит 2
17	Select	Управляющий выходной бит 3
18	GND	Земля, 0 В
19	GND	Земля, 0 В
20	GND	Земля, 0 В
21	GND	Земля, 0 В
22	GND	Земля, 0 В
23	GND	Земля, 0 В
24	GND	Земля, 0 В
25	GND	Земля, 0 В

Для соединения схемы с портом просто свяжите точку земли разъёма с землёй схемы, а выходную линию подключите к любому из пяти входных контактов. Сначала не забудьте отключить диагностический LED. Затем следите за портом состояния, поочерёдно освещая и закрывая датчик. Считанное значение зависит от подключения, но точное число неважно, лишь бы в двух случаях оно различалось.

Логике микросхем нужны несколько иные входные уровни, чем тестовому LED, поэтому, возможно, придётся настраивать `Rvar2`, пока порт не начнёт выдавать разные показания для закрытого и освещённого датчика. Для этого лучше наблюдать порт на компьютере в реальном времени.

Способ наблюдения зависит от операционной системы и языка программирования. В C значение порта читает функция `inb(port)`, поэтому здесь нужно вызвать `inb(0x379)` и проверить результат. В других языках функция, вероятно, называется похоже: ищите `in`, `inport`, `readport` и так далее. Пользователям Windows может пригодиться встроенная программа `debug` и её функция `i`, чтение порта.

> **Примечание.** В некоторых системах, например Linux, сначала придётся запросить разрешение на доступ к конкретному порту. Подробнее см. документацию по `iopl(3)` или сходному вызову.

Теперь всё готово. Направьте зонд на любой LED устройства, настройте датчик по яркости и начинайте читать чередование света и темноты, выясняя, соответствует ли оно передаваемой информации и каким образом.

> **Примечание.** Любопытный читатель может исследовать не только двоичное состояние индикаторного диода, но и его яркость. Даже если конкретный LED не должен напрямую отображать сигнал последовательной линии миганием, между схемами могут существовать аналоговые наводки, и сигнал линии повлияет на яркость. Недорогой аналого-цифровой преобразователь, например TLV571 компании Texas Instruments,

словно просится в такое применение.

Подход позволяет выбирать сигнал с частотой менее миллиона битов в секунду. Этого достаточно для многих интерфейсов, но не обязательно для портов Ethernet, передающих не менее 10 миллионов битов в секунду. При большей скорости LPT, вероятно, достигнет физического предела пропускной способности, но отчаиваться не следует. Пока датчик - фототранзистор - переключается достаточно быстро для нужной связи, вариант остаётся. Помните, что LPT параллельный. Для ускорения, необходимого Ethernet, объедините простейший генератор тактов, схему счётчика и набор защёлок выборки и хранения, например 74LS377, чтобы последовательно записывать данные между чтениями порта компьютером. Короткое время накапливайте сведения, затем через несколько линий состояния или в двунаправленном режиме отправляйте компьютеру сразу несколько битов - образцов - одним пакетом за один цикл чтения, повышая скорость в четыре или восемь раз.

Не стану продолжать, вероятно, лишнее путешествие в мир электроники. Если вы хотите поиграть с высокоскоростной или аналоговой выборкой либо просто любите спаивать детали и подключать их к компьютеру, взгляните на моё довольно полное вводное руководство, тонко замаскированное под проект робота с компьютерным управлением. Его можно найти по адресу lcamtuf.coredump.cx.

А теперь для тех, чьи интересы ближе к практической безопасности: кратко обсудим, как решить проблему, не заклеивая все LED в офисе изолентой.

Предотвращение раскрытия данных через мигающие огоньки - и почему оно не сработает

Самое простое решение проблемы, предложенное и в первоначальном исследовании, - растяжение импульсов, то есть приём, призванный исказить мигание индикатора путём продления некоторых вспышек, чтобы практическое восстановление данных казалось невозможным. Схемы растяжения импульсов представляют собой группу довольно тривиальных устройств, которые дополнительно продлевают встретившийся входной сигнал высокого уровня. Самая простая конструкция такого растяжителя опирается на конденсатор, заряжающийся при наличии входного сигнала, а затем медленно разряжающийся. Конденсатор соединён с двоичным дискриминатором - это не прозвище свирепого чемпиона по борьбе, а устройство, преобразующее аналоговые данные в двоичный выход путём применения заданного порога: для всех входных напряжений выше n оно выдаёт напряжение логической 1, а для всех напряжений ниже - 0. В данном случае точкой различения служит определённый уровень заряда конденсатора.

Распространены и более совершенные, надёжные конструкции, в том числе полностью цифровые схемы. Все они могут применяться в концентраторах и коммутаторах, чтобы LED выглядели привлекательнее. Без них быстрое мигание с частотой намного выше 50 циклов в секунду - частоты, считающейся пределом нашей способности воспринимать мерцание, - обычно заставило бы нас видеть тусклый, но как будто непрерывно светящийся огонёк. Дискриминатор заставляет LED чаще получать 1, чем 0, продлевая каждый импульс 1. Благодаря этому LED светится ярче и мигает реже. На рисунке 5-7 показано поведение такого растяжителя: одиночный пик, одна 1, растягивается до трёхкратной длительности, а все 0 остаются без изменений.

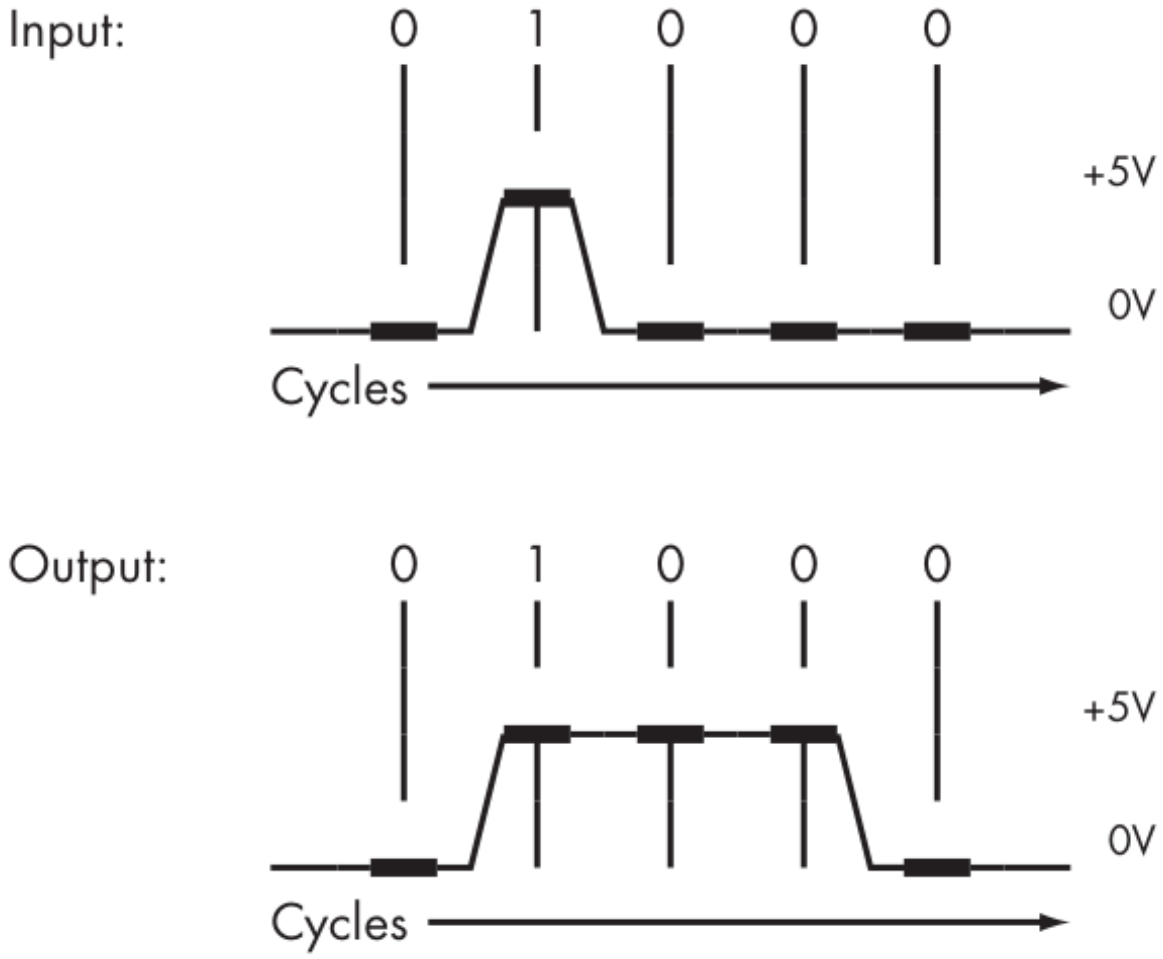
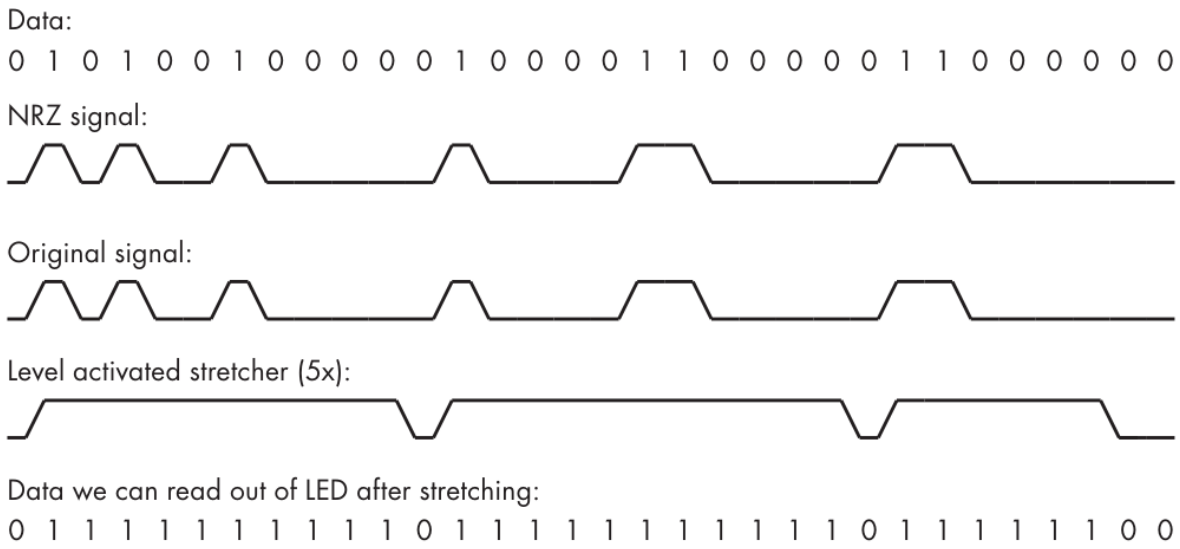


Рисунок 5-7. Поведение растяжителя импульсов, 3x. Одиночная 1 во входном сигнале продлевается на три цикла в выходном, тогда как нули не изменяются.

Хотя основное назначение таких устройств эстетическое, как уже говорилось, они кажутся и хорошим способом решить проблему раскрытия информации посредством светового излучения, позволяя атакующему определить лишь некоторые общие свойства трафика. Таким образом, в лучшем случае атакующий способен выяснить только общие его характеристики: например, когда что-то передаётся, а когда нет.[3]

Но кажущееся хорошим решение не всегда таково. Рассмотрим следующие примерные данные и соответствующий сигнал последовательной линии:



Пример данных, соответствующего сигнала NRZ, исходного сигнала, выхода растяжителя 5x и данных, читаемых по LED после растяжения.

Предположим, сигнал обрабатывается растяжителем импульсов 5x, который продлевает каждую 1 ещё на пять циклов. В первоначальной работе безопасным пределом предлагается 2x, но ради наглядности мы преувеличим.

Хотя по сравнению с входным сигналом, который мы хотим перехватить, почти вся важная информация кажется утраченной, значительную её часть можно восстановить благодаря четырём важным наблюдениям:

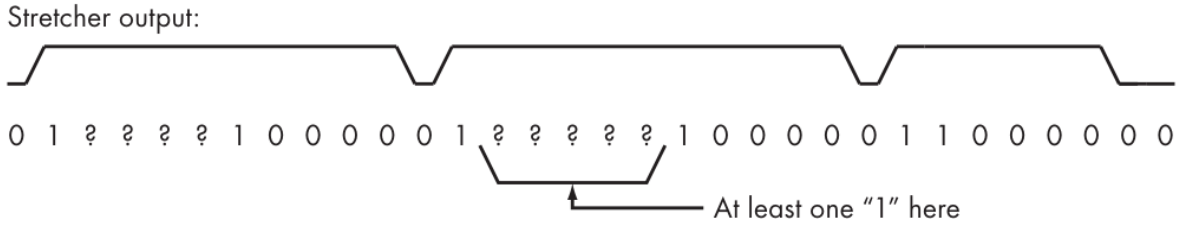
• Очевидно, все участки, где выход растяжителя равен нулю, должны были быть нулевыми и в исходном сигнале.

• Каждая растянутая последовательность единиц должна была быть запущена единицей в начальной позиции исходного потока.

• Каждая последовательность из `L` единиц первоначально должна была содержать по крайней мере одну 1 на каждые `N` циклов, где `N` - коэффициент растяжения этой схемы; иначе в последовательности возникли бы разрывы. Количество единиц в блоке данных, представленном на выходе одним непрерывным растянутым участком единиц, не меньше `L/N`, округлённого вверх.

• Каждая последовательность заканчивается ровно после `N-1` нулей в исходном потоке. Мы знаем, что этим нулям обязательно предшествовала 1, иначе последовательность закончилась бы раньше.

Применив эти знания к предыдущему примеру, можно восстановить большую часть исходных данных следующим образом:



Реконструкция: вопросительные знаки обозначают биты, которые нельзя установить однозначно; в отмеченном участке находилась по крайней мере одна 1.

В предыдущем, вполне реалистичном примере из-за растяжения импульсов потеряно менее 9 из 32 битов данных, которые нельзя однозначно восстановить; на рисунке они помечены вопросительными знаками. Тем самым мы восстановили 99,999988% потенциального пространства поиска. Оставшиеся данные приходится угадывать, но, особенно если перехвачены обычные английские тексты, например электронная почта, восстановить их по сравнению с исходной задачей довольно просто. Авторы исследования предполагают, что для сокрытия данных достаточно растяжения времени включения с `N = 1,5` или `N = 2`, однако это не обязательно так.

Предыдущая схема восстановления работает с растянутыми последовательностями нулей или единиц. Некоторые линии применяют кодирование с возвратом к нулю, RZ, например упомянутую ранее манчестерскую схему. Поскольку там сигнал постоянно меняется, двукратного растяжения действительно может хватить, чтобы скрыть все данные. Но это верно только в том случае, если LED управляется сигналом до его первоначального внутреннего декодирования в NRZ, чего в большинстве ситуаций не происходит. Более того, применять растяжение импульсов к сигналу с кодированием RZ часто попросту глупо: LED будет гореть постоянно, а значит, изначально нет смысла так поступать.

Как отмечалось ранее, дополнительная проблема связана с качеством растяжителя и его восприимчивостью к помехам от других внутренних схем: колебания напряжения LED, вызывающие небольшие изменения яркости во время периода «растяжения», способны раскрыть некоторую информацию. В частности, в эту категорию могут попадать решения на конденсаторах.

Таким образом, некоторые системы, особенно устройства Ethernet, в которых, как известно, применяется растяжение импульсов, могут быть частично уязвимы для атаки, хотя рассмотренная ранее первоначальная работа на основании наблюдения записанной осциллографом картины мигания заключала, что прямой корреляции между передаваемыми данными и поведением LED нет.

Оптимальное решение, особенно при других типах кодирования или когда растяжение импульсов по иной причине нежелательно - например, если разработчик хочет избежать впечатления, будто LED непрерывно горит всё время передачи, - состоит в том, чтобы считывать линию с довольно низкой частотой, скажем 20 Гц, и фиксировать результат в регистре. Регистр хранит его до следующей выборки и одновременно управляет LED.

А теперь вернёмся к простому английскому языку.

Пища для размышлений

Помимо LED сетевых устройств существует множество других, не менее любопытных сценариев утечки посредством светового излучения, хотя объём раскрываемой информации может быть значительно меньше. Например, рассмотрим индикаторы активности диска. Разумеется, связь с диском не использует последовательную сигнализацию: порции данных от байтов до 32-битных слов отправляются одновременно по набору сигнальных линий. И хотя LED обычно подключён так, чтобы показывать лишь состояние определённой управляющей линии, по времени поиска или объёму записанных и прочитанных данных всё равно можно определить многие особенности деятельности системы. В зависимости от того, к чему именно подключён LED, можно измерить один или оба этих показателя. Такая информация вряд ли даст атакующему немедленное преимущество, но некоторые спровоцированные операции ввода-вывода можно совместить с наблюдением за LED жёсткого диска и прийти к интересным выводам, хотя мне неизвестны исследования в этой области.

Другими потенциальными направлениями атаки служат многочисленные USB-устройства и другие проприетарные интерфейсы. Как упоминалось ранее, USB является последовательной шиной, а

некоторые USB-устройства снабжены индикаторами активности.

Предлагались, частично исследовались или по крайней мере испытывались и разнообразные другие необычные и малоизвестные способы раскрытия информации. Среди них - измерение акустических эффектов перезарядки конденсаторов, когда центральный процессор потребляет разную мощность в зависимости от выполняемой инструкции,[5] либо исследование устройства - «чёрного ящика» путём статистического анализа его энергопотребления.[6] И снова: действительно всесторонних исследований каналов раскрытия, отличных от классического излучения EMF, электромагнитного поля, не проводилось, а заняться этим кажется хорошей идеей. Удачи. :-)

Сноски

[1] [назад] Либо наоборот - в зависимости от конструкции передатчика.
[2] [назад] Схемы подавления эха пытаются отличить сигналы самого устройства от сигналов другой стороны и устранить либо значительно ослабить первые. Разные виды таких устройств широко применяются не только в цифровой передаче данных, но и для улучшения качества телефонной связи, устранения обратной связи микрофонов на публичных мероприятиях и решения многих других повседневных задач.
[3] [назад] Строго говоря, согласно обсуждению в главе 1, даже это остаётся направлением атаки. Однако оно значительно менее эффективно и практично, поскольку даёт лишь приблизительное представление о происходящем, а не копию данных.

Глава 6. Эхо прошлого

Где на примере любопытного недостатка Ethernet мы узнаём, что выражаться следует точно

В предыдущей главе были рассмотрены основы связи Ethernet. Этот кажущийся безошибочным и поразительно простой механизм словно не способен вызывать серьёзные проблемы безопасности, за исключением возможного злоупотребления доверием, возникающим из-за регулярной широковещательной передачи данных всем участникам сети. Это хорошо известное и понятное свойство сетей Ethernet; среди надёжных средств против него можно назвать коммутаторы, мосты и сегментацию сети.

Тем не менее проблема проявляется совершенно непредвиденными способами, главным образом из-за неудачного выбора слов - или отсутствия нужных слов - в официальных требованиях к реализации драйверов Ethernet. Результатом стала широко распространённая ошибка реализации, достигшая таких масштабов, что заслужила отдельной главы. Она даёт интересный материал для исследования целого класса проблем, в которых никто как будто не виноват.

Строительство Вавилонской башни

Протокол Ethernet предоставляет базовые средства распределения байтов по проводу: низкоуровневую схему кодирования данных и формат, вмещающий порцию информации. Кадр Ethernet содержит сведения о локальном назначении переносимых данных, то есть о том, кто их отправляет и кто должен получить, а также краткое описание типа инкапсулированной информации. Предусмотрены и дополнительные способы обнаружения ошибок, после чего весь кадр отправляется потенциальному получателю и всем прочим системам. По своей функции Ethernet похож на схемы инкапсуляции порций данных, применяемые в других средах или приложениях, например Frame Relay, Asynchronous Transfer Mode, ATM, Point-to-Point Protocol, PPP, и так далее.

Возникает вопрос: «Какие данные должен нести такой кадр Ethernet?» Компьютеры используют сотни форматов и прикладных протоколов и способны выполнять самые разные приложения - от научного моделирования до сетевых игр и клиентов чата. Поэтому, хотя данные для удалённого получателя можно просто как есть заключить в кадр Ethernet, обычно это плохая идея: получатель не будет знать, как с ними поступить. Это входящая электронная почта? Картинка с веб-страницы? А может, данные конфигурации? Определить невозможно. Более того, поскольку обычный компьютер почти одновременно выполняет множество программ, различие размывается ещё сильнее.

В большем масштабе Ethernet ставит ещё одну задачу: как добраться до другого конца. Передать данные всем участникам локальной сети просто, но что, если другая система, с которой надеется связаться один из местных пользователей, находится не здесь? Что, если до неё нужно добраться через глобальную сеть, WAN, применяющую совершенно иной протокол канального уровня? Даже если удастся найти способ направить трафик к удалённому месту назначения, останется более фундаментальный вопрос: как адресовать пакет.

Ethernet использует собственную, уникальную и специализированную схему адресации. Он называет хосты по теоретически уникальным идентификационным номерам аппаратных плат, адресам Media Access Control, или MAC-адресам, записанным производителем в каждом адаптере Ethernet. Эти номера имеют смысл только для Ethernet; для сети любого другого типа они бессмысленны, а отыскать по ним конкретное оборудование, не находясь в локальной среде, почти невозможно. Отсюда возникает вопрос доверия. Например, кто купил плату с адресом `00:0D:56:E3:FB:E4` и где этот человек теперь? Можно ли верить, что перед нами действительно первоначальный покупатель, а не самозванец?

Подобные низкоуровневые схемы адресации хостов обычно не помогают передавать данные к месту назначения, если только устройство с данным MAC-адресом не подключено непосредственно к физической сети отправителя. Нельзя напрямую сопоставить идентификатор физического устройства с определённым местом на земном шаре и выяснить, по какому пути следует отправлять ему информацию.

Модель OSI

Протоколы канального уровня разрабатывались для поддержки связи между локальными узлами или, в некоторых крайних случаях, между двумя постоянными конечными точками общего соединения. Чтобы сделать возможным объединение сетей и некоторые более практические их применения, была разработана иерархическая структура сетевых протоколов, названная Open System Interconnection, OSI, взаимодействием открытых систем.

Модель OSI, показанная на рисунке 6-1, определяет уровень физического соединения как первый, а поверх него строит функции более высокого уровня. Протоколы канального уровня образуют второй уровень, уровень передачи данных, и, как и следовало ожидать, задают способ связи с другими локальными узлами, использующими то же физическое соединение. Эти протоколы переносят данные более высокого протокола, не зависящего от канала и определённого как третий, сетевой уровень модели. Самый заметный пример такого протокола - Internet Protocol, сокращённо IP.

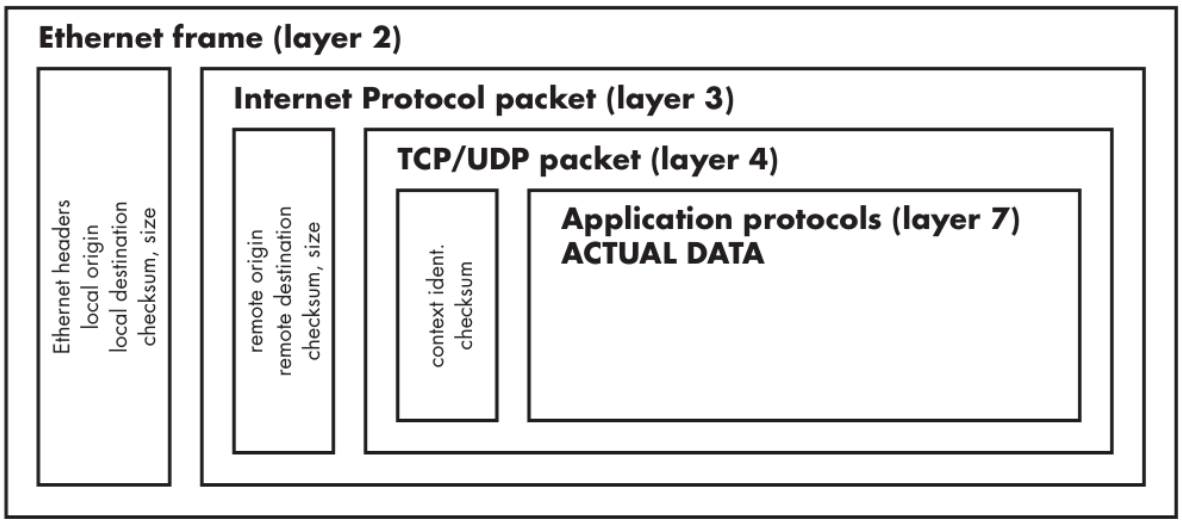


Рисунок 6-1. Пример физической компоновки данных в модели OSI. Кадр Ethernet, уровень 2, заключает в себе пакет Internet Protocol, уровень 3; тот содержит пакет TCP/UDP, уровень 4; внутри находятся прикладные протоколы, уровень 7, и собственно данные. Заголовки последовательно описывают локальные источник и назначение, контрольную сумму и размер, удалённые источник и назначение, а также идентификатор контекста.

Третий уровень предназначен для предоставления сведений об общем назначении трафика, а также универсальной идентификации и источника, и конечного адресата данных посредством сетевой адресации, что облегчает маршрутизацию пакета. В отличие от протоколов второго уровня, третий уровень не отбрасывается и не изменяется по пути и лишён любых особенностей конкретного канала, таких как MAC-адреса, служебные данные CSMA/CD, Carrier Sense Multiple Access with Collision Detection, и тому подобное.

Четвёртый уровень даёт средства для установления отдельных каналов связи между конечными точками, начинающихся и заканчивающихся на данной машине. Благодаря этому одновременно могут существовать многочисленные типы и каналы связи. Чтобы правильно доставить трафик к месту назначения, промежуточным системам не требуется понимать ни один протокол четвёртого уровня. Пакеты интерпретирует только конечный получатель, определяя, какое приложение должно получить данные и как эта порция информации связана с соседними пакетами.

Последующие уровни модели OSI, возможно, менее интересны и имеют склонность сливаться друг с другом. Пятый уровень должен обеспечивать надёжность, функции которой часто включаются либо в протоколы четвёртого уровня, такие как TCP/IP, Transmission Control Protocol/Internet Protocol, либо в прикладной уровень. В некоторых случаях они вообще не реализуются, если надёжная связь не нужна. Шестой уровень предоставляет «библиотечные» функции, например распаковку и декодирование данных, и, как пятый, обычно воспринимается частью функций прикладного уровня. Наконец, седьмой уровень является прикладным - здесь данные передаются в конкретном формате.

Обратите внимание: применительно к переносимым данным верхние уровни модели OSI не зависят от нижних. В надлежащий момент нижние уровни можно постепенно отбросить, не потеряв данные и возможность обрабатывать их дальше. Второй уровень отбрасывается в каждой промежуточной системе; третий можно удалить, когда данные доставлены в систему назначения. Четвёртый отбрасывается перед передачей данных клиентскому приложению.

Третий уровень обычно остаётся полностью независимым от нижележащего протокола канального уровня, поскольку предоставляет исчерпывающие сведения об отправителе и адресате, механизм защиты целостности, контрольную сумму, и информацию о размере переносимой полезной

нагрузки. Именно это делает IP.

Важное следствие такой конструкции состоит в том, что любые лишние сведения, добавленные к пакету на втором уровне во время передачи, не повлияют на то, как адресат интерпретирует информацию IP.

Недостающее предложение

Обсуждая устройство Ethernet в предыдущей главе, я упоминал любопытное требование, возникшее из необходимости надёжно распространять код затора для уведомления о коллизии: минимальный размер кадра Ethernet.

Это требование перешло в официальные спецификации инкапсуляции IP поверх Ethernet, например RFC 1042 “A Standard for the Transit of Internet Protocol Datagrams Over IEEE 802 Networks”,[1] которые требуют дополнять кадры короче минимальной длины. Дополнение можно выполнять любым образом, и оно не влияет на переносимые данные уровня IP, поскольку длина пакета, указанная в заголовках IP, не меняется. Поэтому получатель не истолкует дополнение как часть трафика верхних уровней модели OSI.

Однако есть небольшая проблема. Хотя RFC требует инициализировать дополнение нулями, он не указывает, кто должен предоставить и подготовить его и на каком этапе работы программного обеспечения это следует делать. Требование определённого значения дополнения по своей природе к тому же довольно произвольно, поэтому ему не уделяют внимания: любое другое заполнение не повлияет на работу протокола, ведь посторонние данные после получения просто отбрасываются.

Путаницу усиливает то, что многие сетевые платы умеют автоматически дополнять пакет, если операционная система отправила оборудованию слишком короткий пакет, но, разумеется, не обеспечивают конкретное содержимое дополнения, если о размере кадра уже позаботилось программное обеспечение. Это породило широкую путаницу среди разработчиков, решивших соблюсти требование к размеру и увеличить размер пакета программно, просто повысив объявленную длину. Они часто не понимали, что данные между концом IP-пакета и концом дополненного кадра не были подготовлены, то есть инициализированы нулями, ни драйвером, ни операционной системой, ни оборудованием.

Годами проблема оставалась почти незамеченной, хотя регулярно вызывала странность, сводившую некоторых сетевых хакеров с ума. Пакеты, получаемые ими от локальных систем, нередко содержали в конце какой-то лишний мусор - например, фрагменты веб-страниц или даже разговоров в чате, явно не имевшие отношения к пакету. Они обвиняли получателя, неисправное оборудование, приложение анализа сетевого трафика или библиотеки, но в конце концов прекращали искать причину, потому что вопрос имел второстепенное значение. Проблема так и не получила заслуженного внимания.

Так было до 2003 года, когда Офир Аркин и Джош Андерсон из @Stake решили присмотреться к ней внимательнее. В их работе “EtherLeak - Ethernet Frame Padding Information Leaks”[2] проблема исследовалась подробнее. Авторы поняли, что множество широко используемых систем - Linux, NetBSD, Microsoft Windows и другие платформы - после изменения длины нового кадра Ethernet не инициализируют память в его конце. Некоторые реализации даже неправильно меняют размер кадра или отправляют аппаратному уровню неверное количество байтов.

В результате IP-пакет дополняется данными, которые случайно хранятся в области памяти, прежде использованной системой для других целей. В зависимости от устройства драйвера или операционной системы память может содержать часть ранее отправленного пакета или какой-нибудь другой фрагмент памяти ядра. Разумеется, это создаёт увлекательный сценарий

раскрытия информации: атакующий посылает жертве неприметный и законный трафик и, если повезёт, получает потенциально конфиденциальные сведения. Типичного объёма раскрытой информации достаточно, чтобы оправдать беспокойство.

Утечка ограничена одной сетью Ethernet и потому в обычной среде LAN имеет довольно локальный, некритичный характер. Тем не менее это определённо остаётся достаточно значимой проблемой. И хотя любая локальная сеть отчасти уязвима для прослушивания, конкретно эта проблема наводит на выводы, выходящие за пределы самых очевидных:

• В системах с динамическими буферами исходящих кадров Ethernet, например Linux, дополнение может раскрывать не только предыдущий кадр, но и иное содержимое памяти: редактируемые или просмотренные документы, URL, пароли либо другие конфиденциальные ресурсы. В таком случае внимательный наблюдатель способен получить доступ к сведениям, которые иначе не смог бы перехватить в сети.

• В системах, использующих для подготовки кадров Ethernet только статические буферы, проблему можно применить для обхода средств защиты от перехвата трафика, например коммутаторов, что позволит атакующему перехватить данные другого соединения.

• При некоторых конструкциях статических буферов может раскрываться информация из другого сегмента многосетевой машины, у которой один сетевой интерфейс подключён к общей LAN, а другой - к закрытой сети. Тем самым части предположительно секретных данных передаются в общедоступную инфраструктуру.

Авторы работы тщательно изучили несколько реализаций с открытым исходным кодом и заключили, что широко применяются самые разные подходы и компоновки буферов, а преобладающей схемы их выделения и использования не существует. Их вывод? На обычную разнородную сетевую среду, вероятно, в тот или иной момент воздействуют все три типа проблем.

Пища для размышлений

Рассмотренная здесь проблема присуща не только Ethernet или устройству сетей. Подобные ошибки почти всегда возникают, когда в остальном подробное руководство по реализации опускает либо лишь туманно описывает один необходимый шаг, из-за чего многочисленные разработчики, реализуя стандарт, попросту не замечают проблему. Получив в целом более расплывчатые указания, они, вероятно, были бы вынуждены самостоятельно продумать задачу. Вместо этого разработчики исполняют пошаговые инструкции и становятся гораздо уязвимее для ошибок. «Защищённые от дурака» указания, объясняющие, как выполнять определённые задачи, а не чего следует достичь, часто дают обратный результат.

К проблемам сценариев утечки через протоколы, хотя и в несколько ином контексте, мы вернёмся в части III этой книги.

Глава 7. Безопасность в коммутируемых сетях

Или почему локальные сети Ethernet невозможно окончательно исправить, как бы мы ни старались

Сети Ethernet не дают универсального и простого способа обеспечить целостность или конфиденциальность передаваемых данных и не рассчитаны на противодействие злонамеренному, умышленно внедрённому трафику. Ethernet - всего лишь средство соединения нескольких локальных, предположительно доверенных систем.

На этапе проектирования такое доверие удобно и теоретически достаточно для равноправных систем, находящихся в одной сети и часто примерно в одном физическом месте. Но, как гласит старая поговорка, лишь в теории между теорией и практикой нет разницы. На практике она есть.

Оказывается, локальные сети трудно полностью контролировать, и защищать их приходится не только от внешних угроз, но и от собственных пользователей. В любой растущей локальной сети неизбежно появляется злоумышленник: либо внутри организации, либо снаружи, воспользовавшись недостатком одной из систем. Почти каждый сетевой администратор в конце концов узнаёт, что появление такого эксплойта - лишь вопрос времени.

Практическая сетевая безопасность - это искусство обнаруживать происшествия, сокращать уязвимую поверхность, оценивать и понимать риск на всех уровнях, а не только упражнение в возведении защиты по периметру. В чём проблема? Простейшая инфраструктура Ethernet подвержена всевозможным сценариям перехвата данных, захвата соединений и выдачи себя за другого. Как только нарушитель или злонамеренный, но законный пользователь получает контроль над одной системой в сети, прорвав единственную линию обороны, он способен с минимальными усилиями посеять хаос в инфраструктуре и получить доступ к определённым ресурсам и службам либо захватить их.

Немного теории

Может показаться, что проблему решают коммутаторы Ethernet - класс интеллектуальных устройств, созданных для направления одноадресного трафика второго уровня OSI к нужному порту вместо его широковещательной передачи всем узлам, как делают концентраторы и прямые соединения. Часто считается, что они устраняют проблемы безопасности, связанные со способностью одной системы наблюдать за трафиком третьих сторон или захватывать его, но это не так. Решение не столь просто, а вызванное данным предположением заблуждение порой приносит больше вреда, чем коммутаторы вообще могли бы принести пользы. Однако обо всём по порядку. Чтобы понять уязвимость, посмотрим, как на самом деле работают коммутаторы Ethernet.

Разрешение адресов и коммутация

Вся связь внутри локальной сети опирается на схему адресации, рассмотренную в главе 5. Для адресации систем и доставки кадров данных применяются уникальные идентификаторы, назначенные производителем оборудования конкретному конечному устройству. Однако Интернет и большинство современных частных сетей построены вокруг более гибкого и универсального семейства протоколов и используют схему адресации третьего уровня OSI, обычно известную как адреса Internet Protocol, IP. Сначала с помощью IP-адреса трафик направляется через весь мир к нужной локальной сети посредством иерархии таблиц маршрутизации в промежуточных системах по всей планете. Лишь когда пакет достигает границы сети назначения, конечного получателя приходится искать старомодным способом - по аппаратному адресу.

Когда система в локальной сети решает найти другую локальную сторону с определённым IP-адресом, она использует специальный протокол разрешения адресов, ARP, чтобы определить связь между физическим адресом платы - основой адресации систем в локальной сети - и IP-адресом, универсальным идентификатором системы в объединённой сети.[1] Отправитель распространяет запрос ARP по специальному широковещательному адресу локальной сети. Этот зарезервированный адрес гарантированно принимают и обрабатывают все системы в сети независимо от фактического аппаратного адреса конкретного узла. В данном сценарии система, считающая, что именно она вправе использовать указанный в запросе IP-адрес, должна ответить

отправителю, тем самым раскрыв в ответе свой аппаратный адрес; всем остальным полагается молча проигнорировать широковещательный пакет ARP. После такого обмена обе стороны знают IP-адреса и адреса Media Access Control, MAC, друг друга. Им следует сохранить результат в специальном буфере, чтобы не выполнять новые поиски при каждом обмене порцией данных, а затем перейти к самой связи. Во всём остальном они уже готовы обмениваться пакетами на основе IP-адресации.

Эта конструкция - очаровательный и восхитительный образец старомодного доверия и учтивости. Но как ограничить уязвимость, создаваемую злонамеренным наблюдателем в той же сети, который притворяется кем-то другим, и как не позволить слишком любопытным пользователям или злобным врагам зайти чересчур далеко? Производители оборудования Ethernet определённо не помогли сетевым администраторам, сделав смену MAC-адреса на большинстве современных устройств простой и элементарной. Предположительно это позволяет пользователю перепрограммировать адрес, чтобы не попасть в неприятности, если однажды выяснится, что партия плат получила повторяющиеся адреса.

И снова кажется, что проблему решают коммутаторы. Базовая идея интеллектуального коммутирующего устройства состоит в том, чтобы повторить кэш MAC-адресов на уровне промежуточного сетевого устройства. Коммутатор снабжён множеством портов Ethernet, каждый из которых соединён с одной системой или, реже, с их группой. Но вместо работы глупым повторителем, отправляющим весь трафик, принятый на одном порту, всем остальным, как поступает концентратор Ethernet, коммутатор старается запомнить MAC-адреса машины, подключённой к каждому порту. Он фактически создаёт соответствия MAC-адресов портам, в отличие от соответствий MAC-адресов IP-адресам, создаваемых конечными системами.

Данные хранятся в ассоциативной памяти, content addressable memory, CAM,[1] и определяют, куда доставлять входящие пакеты. При поступлении порции трафика коммутатор пытается установить, на каком порту находится адресат. Если сведения доступны, пакет доставляется непосредственно и только на этот порт, благодаря чему информация не попадает к остальным, а производительность сети повышается.

Виртуальные сети и управление трафиком

Некоторые более совершенные коммутаторы предоставляют дополнительные функции, призванные упростить управление крупными сетями и сократить время и стоимость развёртывания. Эти функции также кажутся полезными для сетевой безопасности; среди них встречаются следующие.

Виртуальная LAN, VLAN

Общее название набора способов разделить совокупность портов физического устройства на несколько отдельных логических сетей, тем самым отделив трафик одной группы портов от других и не позволяя никакому трафику пересекать границы этих групп на уровне коммутатора. Чаще всего схема реализуется посредством стандарта IEEE 802.1Q, подробно рассмотренного в следующем пункте. Реализация VLAN подобна разделению одного коммутатора на несколько полностью независимых устройств, но решение VLAN гораздо гибче и экономичнее, поскольку позволяет по своему усмотрению менять форму сети и перераспределять физические ресурсы. Сетевые специалисты повсюду тепло приняли VLAN: технология обещала простой, но мощный способ создать на одном устройстве несколько отдельных сетей или, например, отделить серверы от рабочих станций без покупки особого коммутатора для каждой группы.

Транкинг

Естественное расширение базовой конструкции VLAN. Транки используют схему маркировки кадров IEEE 802.1Q, чтобы туннелировать трафик нескольких VLAN по одному соединению, не заставляя пользователя прокладывать отдельный провод для каждой VLAN, которую требуется распространить на другое устройство, как показано на рисунке 7-1. Пакеты всех или некоторых VLAN исходного коммутатора получают в заголовке кадра Ethernet достаточно сведений для определения VLAN происхождения, туннелируются по обычному соединению к другой конечной точке, декодируются и затем выпускаются в соответствующие VLAN на стороне назначения.

Хотя такой вариант обычно уступает по производительности отдельному кабелю для каждой подсети, он гораздо практичнее. Системы с транками нередко также поддерживают DTP, Dynamic Trunking Protocol, протокол автоматической настройки транка, позволяющий устройствам автоматически обнаруживать другие устройства с поддержкой транков и обмениваться с ними инкапсулированными кадрами без специальных административных действий.

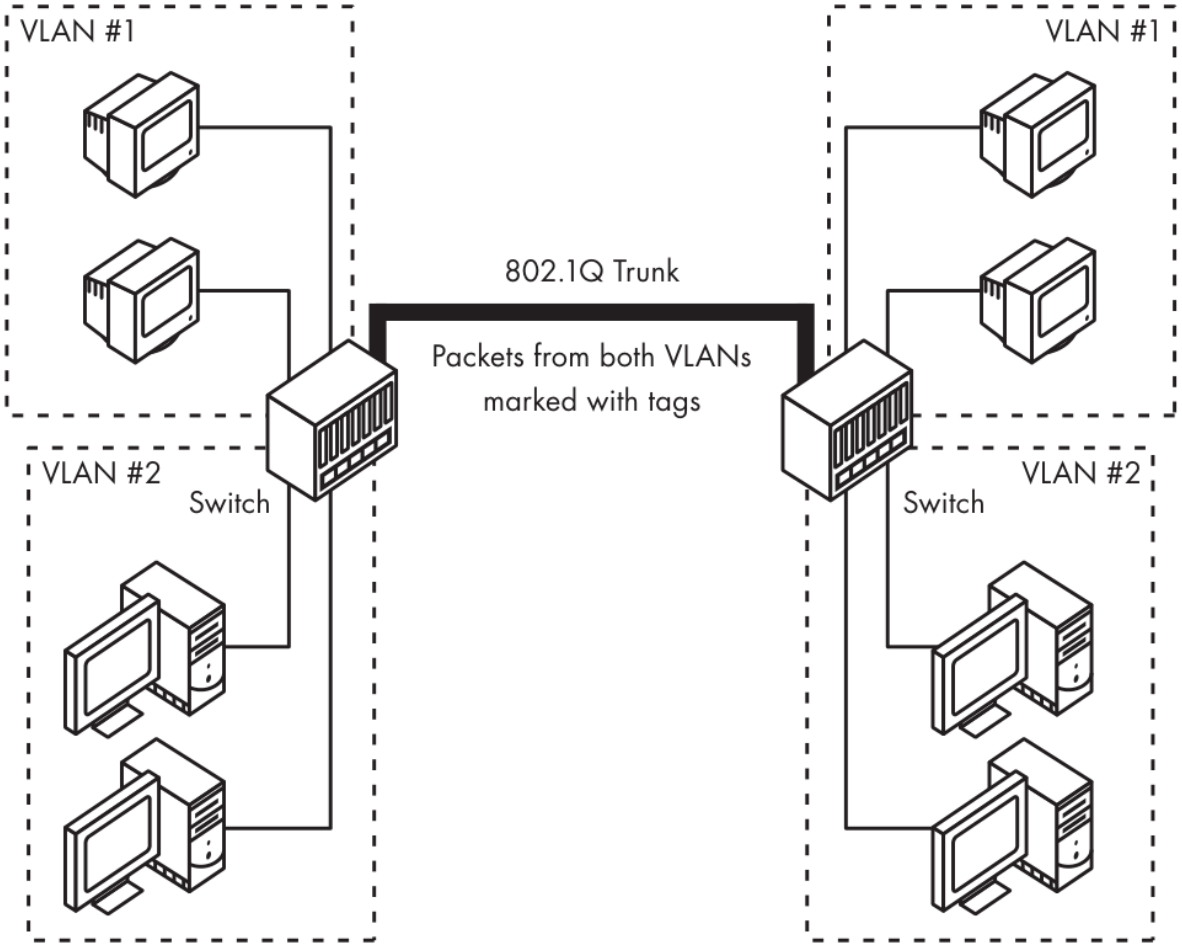


Рисунок 7-1. Транкинг VLAN в действии: VLAN распространяются между двумя устройствами. Устройства во всех экземплярах VLAN 1 могут связываться друг с другом, как и устройства во всех экземплярах VLAN 2, но перекрёстный обмен между VLAN 1 и VLAN 2 невозможен. Пакеты обеих VLAN проходят по транку 802.1Q с метками.

Протокол связующего дерева, STP

Позволяет строить избыточные сетевые структуры, где коммутаторы соединены друг с другом более чем в одном месте для сохранения отказоустойчивости. Традиционно такая конструкция могла заставить широковещательный трафик и некоторые другие пакеты бесконечно двигаться по кругу и одновременно значительно ухудшить производительность сети, потому что данные, принятые на одном интерфейсе и пересланные другому, фактически возвращаются к отправителю, как показано слева на рисунке 7-2.

При проектировании сети часто трудно избежать случайных широковещательных петель. Иногда также желательно намеренно создавать архитектуры с потенциальными петлями, где один коммутатор соединён с двумя или несколькими другими, поскольку такая конструкция гораздо лучше переносит отказы: одно устройство или одно соединение можно вывести из строя, не разделив всю сеть на два отдельных острова.

Чтобы петли и другие нетривиальные архитектуры можно было строить без серьёзных проблем производительности, STP реализует механизм выборов корневого узла-коммутатора. На основании результата от этого узла вниз строится древовидная иерархия распределения трафика, а соединения, способные вызвать обратное распространение широковещательного трафика, временно отключаются, как показано справа на рисунке 7-2. Когда один из узлов выпадает, простую самоорганизующуюся иерархию можно быстро изменить и снова активировать соединение, ранее признанное ненужным.

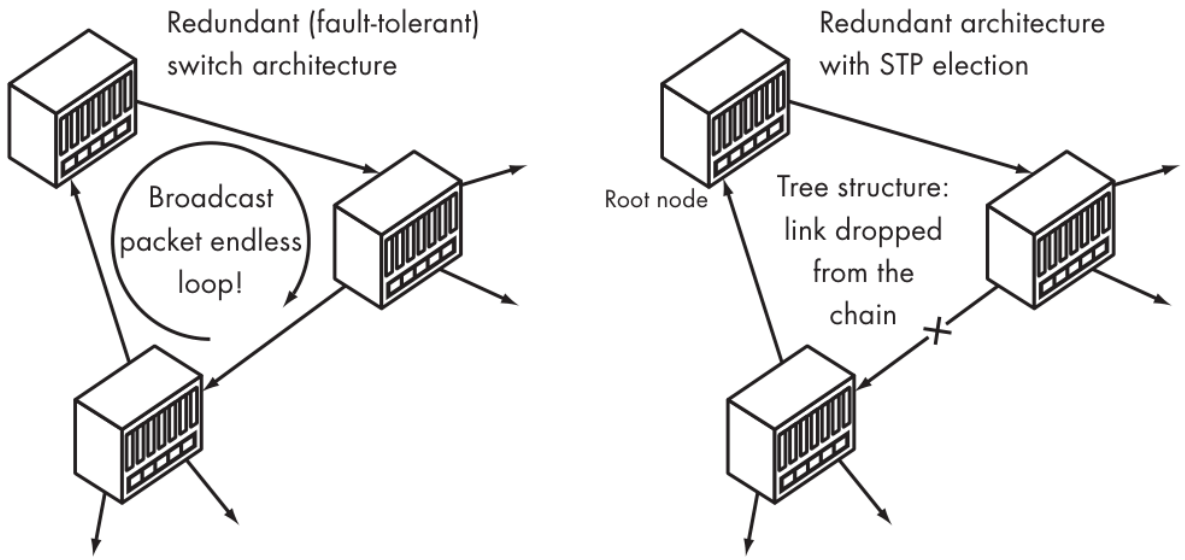


Рисунок 7-2. Проблема пакетного шторма и схема выборов STP. Слева показана отказоустойчивая сеть без STP, где некоторые пакеты обречены почти бесконечно ходить между коммутаторами; справа - та же сеть, в которой одно устройство автоматически выбрано STP корневым узлом, а логическая топология изменена для устранения петель. При отказе одного из соединений сеть перенастраивается для правильной работы.

Атака на архитектуру

Рассмотренные до сих пор механизмы были созданы для повышения экономической эффективности и производительности поверх сетевой конструкции, вообще не предоставляющей функций безопасности.[2] Некоторые распространённые, хорошо понятные и легко предотвращаемые атаки, такие как подмена MAC, способность любого человека подделать сообщение ARP и выдать себя за устройство с определённым IP-адресом, широко признаны недостатком локальных сетей и легко блокируются правильно настроенными коммутаторами. Но

другие серьёзные изъяны конструкции не столь тривиальны и далеко не так легко предотвращаются. Не всегда очевидно, что решения, которые принято считать средствами усиления безопасности, на деле ничем ей не помогают.

CAM и перехват трафика

Одна из самых наглядных причин не считать коммутаторы средством безопасности - сценарий переполнения CAM. Память CAM, хранящая соответствия MAC-адресов портам, имеет фиксированный и ограниченный размер и обычно устроена без избирательности. Когда систему невозможно найти в CAM, у коммутатора остаётся единственный способ доставить пакет: перейти в режим концентратора и широковещательно передать пакет всем системам в надежде, что получатель признает трафик адресованным ему, а все прочие окажутся достаточно любезны, чтобы полностью его проигнорировать. Поэтому осторожный атакующий может породить множество ложных запросов и ответов ARP либо иных пакетов, выдавая себя за огромное количество отдельных сетевых устройств, лишь бы заполнить CAM коммутатора.

После заполнения CAM атака фактически ослабляет сетевую безопасность: отключает интеллектуальную маршрутизацию кадров коммутатором и вынуждает его перейти к широковещательной передаче всех данных. Это, в свою очередь, позволяет атакующему подслушивать всю связь, словно сеть вообще не коммутируется. Всё это можно сделать, не выдавая себя за получателя и не оказывая заметного влияния на работу сети, поэтому жертва вполне может совершенно не подозревать о проблеме. Это особенность конструкции, а не недостаток предусмотренного назначения устройств, но она представляет собой серьёзную ошибку в распространённом понимании работы коммутаторов. И будьте уверены: в обычной среде полностью решить эту проблему почти невозможно. Некоторые коммутаторы вводят ограничения по портам и времени для предотвращения таких атак, но стопроцентной эффективности они не дают.

Другие сценарии атак: DTP, STP, транки

Другие проблемы обычно легче предотвратить и труднее не заметить, жертва часто способна их обнаружить, но и они показывают недостатки безопасности на уровне Ethernet. Например, одна любопытная возможность - атака на упомянутый механизм DTP. Ради упрощения настройки автоматическое согласование DTP нередко включено для всех портов устройства. Проблема в том, что умный атакующий может притвориться не простой пользовательской рабочей станцией или скромным сервером, а коммутатором с поддержкой транков. Как только подключённый коммутатор признает его дружественным устройством, атакующий начнёт получать кадры с метками 802.1Q, включая трафик других виртуальных LAN, обслуживаемых этим коммутатором, и сможет перехватывать либо внедрять вредоносный трафик в сети, с которыми не должен иметь возможности связываться. Во многих сетях, где один коммутатор обслуживает и защищённые, «демилитаризованные» сети, и обычную корпоративную инфраструктуру LAN, такая атака может принести очень ценные сведения, позволив участникам одной сети подслушивать другую или взаимодействовать с ней.

На некоторых устройствах проблему DTP можно решить изменением конфигурации по умолчанию и точным заданием набора выделенных портов коммутатора, на которых разрешены транки. Но на этом неприятности не заканчиваются: похожим образом можно злоупотребить и нашим другим другом, STP, позволив атакующему выбрать себя «корневым» коммутатором и получить долю сетевого трафика. Отключить обнаружение STP в обычной корпоративной среде может оказаться ещё сложнее.

Ещё одна проблема возникает, когда какой-либо транк начинается или заканчивается в невыделенной VLAN, то есть используемый для транкинга порт помещён в VLAN, которой пользуются и рабочие станции. Внедряя уже помеченные кадры, можно вводить трафик в транк. Это, возможно, ошибка конфигурации, и её часто не замечают, поскольку многие инженеры считают способ реализации транков гораздо более сложным и волшебным, чем он есть на самом деле.

Предотвращение атак

Эти проблемы нередко трудно решить, особенно в сети, которую не контролировали строго и внимательно на всех этапах развития и расширения. Некоторые высококлассные устройства предоставляют расширенные функции безопасности, способные противодействовать потенциальным направлениям атак и уменьшать либо устранять часть рисков. Но сети Ethernet не проектировались для обеспечения безопасности, как не проектировались для этого и многие интеллектуальные устройства, созданные для управления такими сетями. Атакующий способен легко обесценить некоторые или все их функции и понизить модель безопасности сети до наименее желательного варианта.

Существуют методы и строгие практики, которым можно следовать для защиты локальной сети Ethernet. Однако сложность процесса, часто сопутствующие ему дополнительные финансовые затраты и снижение производительности, не говоря уже о количестве направлений, которые приходится учитывать, ясно показывают: технология не разрабатывалась с мыслью о каком-либо уровне практической безопасности.

Пища для размышлений

При разработке Ethernet казалось разумным не учитывать безопасность в проектных решениях и возложить защиту сети на архитектуру верхнего уровня, шифрование и тому подобное. Однако со временем это первоначальное решение стало увеличивать общую стоимость обслуживания сетей Ethernet и затруднять их достаточно надёжную защиту от взлома без тех или иных жертв в функциональности.

Проблема отнюдь не ограничивается Ethernet. Многие сети, доверие к которым заложено по критериям физического доступа или доступа к оборудованию, включая, например, большинство телефонных систем мира, по своей природе и без возможности полного контроля открыты внутренним угрозам. В них почти или совсем невозможно эффективно ограничить уязвимость и побочный ущерб от компрометации единственной системы в общей сети. По мере роста сети и числа соединений вероятность того, что одной из систем управляет злонамеренный пользователь либо она недостаточно защищена от физического или удалённого доступа, неуклонно приближается к 1.

Традиционно для компрометации системы требовался доступ к магистрали, а не к станции конечного пользователя, поэтому ситуация несколько отличалась от Ethernet. Но современные системы Voice-over-IP, VoIP, быстро устраняют это неудобство, нередко позволяя легко совершать подмену и иные уловки, поскольку чрезмерно доверяют стороне конечного пользовательского устройства.

Сноски

[1] [назад] Как подсказывает название, к памяти этого типа можно обращаться непосредственно по параметру, значение которого требуется определить. Это экономит время, которое обычно пришлось бы потратить на поиск параметра. Простой пример CAM - библиотечный каталог: чтобы найти одну книгу, не нужно просматривать все книги библиотеки; место поиска определяется тем, что именно вы ищете, то есть сведениями о «содержимом».

Глава 8. Мы против них

Что ещё может произойти внутри локального периметра «нашей» сети? Очень многое!

Конструкции локальных сетей, такие как Token Ring или ныне преобладающий Ethernet, создавались исходя из того, что обеспечивать безопасность на уровне, или слое, самой технологии передачи данных не требуется. На заре компьютеров от пользователей общей сети ожидали хорошего поведения.

Уже по одной этой причине можно было бы предположить, что разработчики Ethernet не видели нужды включать в конструкцию полноценные функции безопасности. Однако их следует винить в неоправданном оптимизме и неспособности предвидеть неизбежное. Ethernet просто не оставил пространства для лёгкой реализации механизмов целостности, конфиденциальности и проверки отправителя на верхних уровнях OSI, в устройствах и приложениях. Более поздние протоколы и схемы связи пытались обеспечить частичную приватность и некоторую неоспоримость связи, но лишь привели нас к пониманию: достаточную безопасность там невозможно реализовать, не вернувшись назад и не переделав канальный уровень. Единственной оставшейся возможностью стало возведение поверх системы вычислительно дорогих и сложных криптографических заплаток, сама сложность которых способствует появлению множества проблем безопасности, обнаруживаемых из года в год.

Эта неудачная, а позднее вполне намеренная тенденция фактически создала набор сетевых механизмов, которые хорошо работают и доступны по цене, но не подходят для обработки даже умеренно конфиденциальных данных в присутствии враждебной стороны, тогда как почти весь связанный с пользователями поток данных в локальной сети является конфиденциальным. Решения, пытающиеся справиться с этими проблемами, - приложения виртуальных частных сетей, VPN, зашифрованная инкапсуляция для немногих счастливчиков среди самых популярных веб-протоколов, усовершенствованные коммутаторы и тому подобное - обычно гораздо дороже и сложнее, чем могли бы быть, если бы безопасность стала ключевым фактором при разработке первоначальной схемы связи Ethernet.

Прежде чем прийти к этому, мы довольно долго жили в частичном отрицании. Когда безопасность стала практической заботой, с расширением Интернета и внезапным распространением компрометации систем, первая оборона сосредоточилась на внешнем мире и игнорировала угрозы изнутри «доверенной» сети. Но вскоре нескольким корпорациям и учреждениям пришлось усвоить болезненные уроки. Со временем стало ясно, что одной внешней защиты - межсетевых экранов и систем обнаружения вторжений - недостаточно, даже если она правильно настроена по всему предприятию. Сетевой уровень всё ещё оставался уязвимым и позволял внутреннему нарушителю компрометировать обмен данными, не эксплуатируя уязвимость безопасности ни одной отдельной системы компании.

Можно возразить, что сеть удастся защитить, развернув на всех интерфейсах надлежащее шифрование и криптографические механизмы проверки личности и целостности. Но это часто непрактично или невозможно, особенно без ущерба производительности и надёжности сети и без значительных затрат, не говоря уже о совместимости с различными операционными системами и приложениями. Кроме того, как я уже упоминал, криптография не всегда служит ответом. Атаковать гораздо легче, когда данные можно видеть и перехватывать, например посредством повторной атаки или атаки по времени; к тому же некоторые виды информации, такие как рассмотренный

ранее недостаток дополнения кадров Ethernet, способны свести на нет все усилия по защите пользователя.

В части II этой книги рассматриваются некоторые угрозы, присущие локальным сетям и раскрывающие информацию без традиционной атаки как таковой. Все эти проблемы останутся с нами, пока сети используют старую и проверенную конструкцию, довольно плохо подходящую современным сетям.

Теперь мы готовы двигаться дальше, но, прежде чем погрузиться в дикий и увлекательный мир за пределами локального периметра, взглянем на другие интересные, более конкретные сценарии уязвимости.

Логические мигающие огоньки и их необычное применение

Один из примеров связан со злоупотреблением логическими индикаторами, то есть счётчиками, флагами и другими приспособлениями без физического воплощения, которые поддерживаются компьютером и доступны программно и широко применяются в локальных сетях. Логические индикаторы - полезная функция, снова предполагающая, что локальной сети следует доверять.

Simple Network Management Protocol,[1] SNMP, является самым популярным способом наблюдения за сетевыми устройствами, а иногда и их администрирования. SNMP часто реализуется и на конечных системах, серверах и рабочих станциях, и на сетевых устройствах - коммутаторах, маршрутизаторах и принтерах.

SNMP позволяет читать или изменять абстрактное представление множества внутренних компонентов системы и приложений, рабочих параметров, настроек и статистики. Посредством SNMP можно спросить у сетевого принтера, сколько у него сетевых плат или каково время непрерывной работы, а затем совершенно тем же способом запросить те же сведения у мейнфрейма, хотя внутри каждого устройства они добываются совсем иначе. Поэтому SNMP облегчает наблюдение и управление разнородными средами без реализации множества собственных протоколов доступа и процедур проверки.

Разумеется, у самого SNMP предостаточно проблем безопасности реализации и развёртывания, но сейчас речь не об этом. Даже при правильной реализации функция может раскрывать защищаемые сведения, например предоставляя доступ только для чтения к кажущейся несущественной статистике сетевого интерфейса. Эту брешь можно устранить, если тщательно ограничить протокол, но на сетевом оборудовании некоторых типов это часто невозможно. Внимательный атакующий способен наблюдать за счётчиками кадров или пакетов в системе с SNMP и на их основе получать сведения профилирования, необходимые для атак по времени. Таким образом можно восстановить информацию интерактивного сеанса или другие интересные свойства способом, похожим на рассмотренный в главе 1.

Упс. Но неужели из этого действительно может выйти столько дурного?

Покажи мне, как ты печатаешь, и я скажу, кто ты

Хотя этот класс проблем уже упоминался несколько раз и может казаться абстрактным, его последствия реальны, даже если не учитывать восстановление нажатий клавиш, которому была посвящена глава 1. Например, в ходе увлекательной разработки группа немецких исследователей из Institut für Bankinnovation создала коммерческий продукт PSYLock с биометрией на основе манеры печати.[2] С помощью PSYLock им удалось однозначно идентифицировать, а значит, возможно, и отслеживать пользователей по тому, как они обращаются с клавиатурой.

PSYLock в основном опирается на измерение времени между нажатиями клавиш - уже рассмотренный приём. Если можно наблюдать за счётчиками пакетов определённой машины и вычислять, когда пользователь нажимает клавишу во время интерактивного сеанса, человека можно опознать независимо от используемого терминала. Из применения этой идеи на сетевом уровне вытекают интересные возможности как злонамеренного, так и надзорного характера. Если атакующий знает, что между станцией, для которой он способен наблюдать статистику порта коммутатора по SNMP, и другой системой идёт интерактивный сеанс какого-либо протокола удалённого доступа, он может многократно опрашивать счётчик, определять моменты нажатия клавиш и делать выводы о том, что печатают или кто печатает.

Возможен и более лёгкий вариант атаки, не требующий сложного моделирования, с которым прежде приходилось справляться. В публикации Bugtraq “Passive Analysis of SSH (Secure Shell) Traffic”[3] Solar Designer и Даг Сонг среди прочего предлагают ещё одну возможную атаку, на этот раз с использованием SSH, распространённого способа подключения к удалённой системе. Хотя SSH зашифрован, в версиях, выпущенных до их исследования, длину пароля можно измерить, тщательно проанализировав размер наблюдаемого пакета при входе: после ввода пользователем пароль отправляется одним блоком данных.

Тот же приём вполне можно успешно применить к другим криптографическим протоколам, которые не принимают активных мер для сокрытия длины пароля путём его дополнения перед отправкой. И, что неудивительно, атаку можно провести простым наблюдением за счётчиком байтов SNMP, а не непосредственным перехватом трафика.

Неожиданные биты: личные данные повсюду

Ещё одна причина не радоваться перспективе враждебного наблюдения за нашей сетью, независимо от того, считаем ли мы видимые противнику данные конфиденциальными, состоит в том, что множество программ нарушает принцип наименьшего удивления. Это фундаментальное правило разработки программного обеспечения, согласно которому программа должна отвечать пользователю способом, который удивит его меньше всего, то есть последовательно, интуитивно, предсказуемо или иным ожидаемым образом. Оказывается, многие программы разных издателей отправляют поразительный объём ценной информации, намного превосходящий ожидания, и часто ставят пользователей в положение, на которое те не соглашались. Как всегда, Microsoft Windows возглавляет стаю этих удивительных программ и великолепно выпускает информацию намеренными, но часто упускаемыми из виду и неочевидными путями. Впрочем, дружелюбный программный гигант не одинок.

Мало кто из пользователей знает: когда Windows работает в домене и настроена на перемещаемые профили, позволяющие войти с другой рабочей станции и получить доступ к личным данным, значительные части реестра пользователя отправляются контроллеру домена при каждом входе и выходе. На первый взгляд содержащаяся в профиле информация может казаться совершенно бесполезной, но в неё входят разнообразные личные настройки и интересные сведения истории, в том числе последние выполненные команды, посещённые веб-страницы и открытые документы.

Сходным и, пожалуй, ещё более удивительным образом, если домашний каталог пользователя в домене находится на сетевом диске, Windows ищет все команды, введённые пользователем в поле Run, сначала на удалённом сервере и только потом локально. Поэтому внимательный наблюдатель получает сведения обо всех командах пользователя через протокол Server Message Block, SMB.

Эти и многие другие примеры болезненно ясно показывают: почти все сетевые данные следует считать конфиденциальными. Поэтому локальные сети в целом не особенно подходят для

переноса любых обычных данных, кроме специальных, ограниченных или дополнительно защищённых конфигураций. И у нас нет хорошего способа защитить эту информацию без развёртывания тяжёлой артиллерии вроде криптографических туннелей IP или подобного программного обеспечения либо без полной перестройки всех сторон сетевого взаимодействия с нуля.

Уязвимости Wi-Fi

Было бы несправедливо закончить главу, проигнорировав проблемы беспроводной замены Ethernet - Wi-Fi.

Беспроводные сети на основе протокола IEEE 802.11 набирают силу и в корпоративном мире, и среди обычных домашних пользователей. К несчастью, задолго до широкого признания и несмотря на намерение обеспечить дополнительную безопасность по сравнению с проводными соединениями Wi-Fi оказался довольно сложен для правильного развёртывания - возможно, потому, что пытался слишком точно идти по стопам старшего брата.

По принципам работы стандарт 802.11 не слишком отличается от Ethernet. Он применяет традиционную схему управления доступом к среде «один говорит, остальные слушают»; разница лишь в том, что носителем сигнала вместо пары проводов теперь служит выделенная радиочастота. Это приводит нас к первой проблеме 802.11.

В мае 2004 года Information Security Research Centre, ISRC, Технологического университета Квинсленда объявил результаты: любую сеть 802.11 на любом предприятии можно за считаные секунды полностью остановить, просто передавая сигнал, который не позволяет другим сторонам попытаться заговорить. Естественно, то же верно для Ethernet, но там сначала необходимо подключиться к сетевой розетке, благодаря чему атакующего гораздо легче отследить, а проблему - решить. Можно просто проверить коммутатор и проследовать по кабелю. Атака не то чтобы неожиданна, но корпоративные пользователи ожидали не этого.

На этом проблемы не заканчиваются. Там, где стандарт 802.11 пытался противостоять атакам на уровне носителя, он потерпел сокрушительную неудачу. Механизм Wired Equivalent Privacy, WEP, создавался для защиты сетей Wi-Fi от прослушивания сетевых сеансов внешними сторонами, то есть для безопасности, примерно сопоставимой с традиционной LAN. Однако в 2001 году исследователи Калифорнийского университета и Zero Knowledge Systems обнаружили в схеме WEP ряд проектных недостатков, доказавших её полную непригодность. К сожалению, уже тогда Wi-Fi был распространён настолько широко, что необходимые изменения оказалось трудно внедрить.[4]

В довершение неприятностей использование WEP необязательно, а на большинстве беспроводных сетевых устройств он отключён: они готовы принимать и передавать любой полученный трафик. Для проводных сетей, где дополнительную защиту обеспечивает физический уровень, это в целом приемлемо, но беспроводные сети открыты любому случайному человеку в пределах действия.

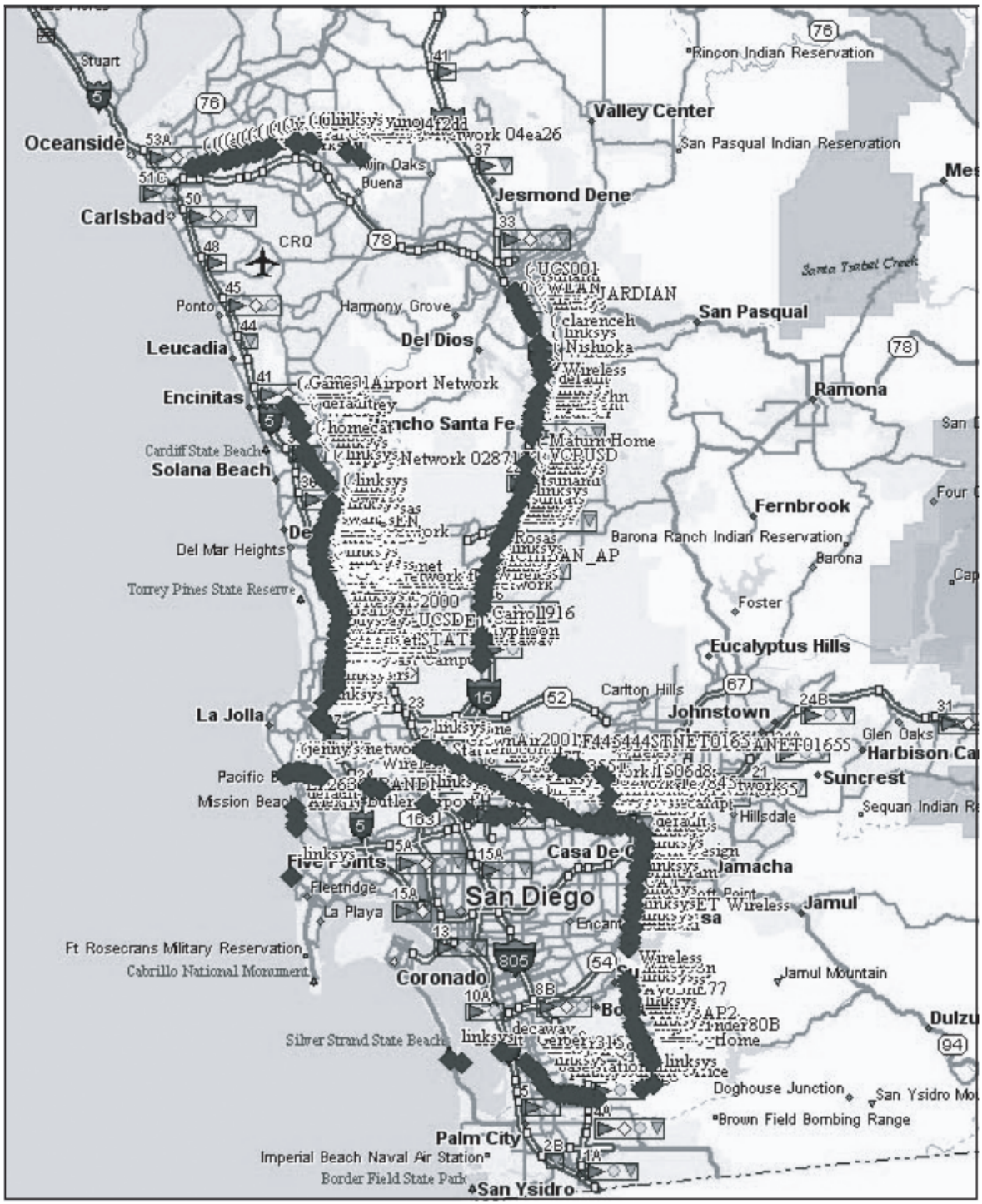


Рисунок 8-1. Воздушная экспедиция Трейси Рида в поисках беспроводных сетей. Предоставлено Трейси Ридом из Copilot Consulting, `treed@copilotconsulting.com`.

Практика wardriving - оснащение автомобиля ноутбуком с Wi-Fi и поездки по городу в поисках сетей - стала чрезвычайно популярной после открытия, что у большинства крупных предприятий, особенно в больших торговых центрах и коммерческих районах каждого города, беспроводные сети частично или полностью открыты. Злоупотребления часто вполне тривиальны: от бесплатного доступа к сети до рассылки спама или удалённых атак через сеть жертвы. Но риск проникновения в сеть изнутри умелым атакующим реален.

Каков настоящий масштаб проблемы? Достаточно сказать, что в какой-то момент wardriving вышел из моды с появлением warflying, то есть того же поиска, но с самолёта вместо наземного

транспорта. В 2002 году Трейси Рид из Copilot Consulting решил облететь Сан-Диего и окрестности с беспроводным сканером. На высоте 1 500 футов он обнаружил почти 400 точек доступа с настройками по умолчанию и, вероятно, свободным доступом в Интернет или внутренние корпоративные сети для любого человека поблизости, как показано на рисунках 8-1 и 8-2. Только 23 процента просканированных устройств были защищены WEP, который в общем случае всё равно легко взломать, или более совершенными механизмами.

Вот так-то.

Рисунок 8-2. Воздушный поиск беспроводных сетей над Кремниевой долиной.

Часть III. В дикой природе

Стоит оказаться в Интернете, и начинается грязь

Глава 9. Иностранный акцент

Пассивное снятие отпечатков: едва заметные различия в нашем поведении помогают другим понять, кто мы

В Интернете, сети сетей, сведения, отправленные удалённой стороне, выходят из-под контроля и наблюдения отправителя. В отличие от локального Ethernet, который обычно служит безопасной гаванью для пакетов, пока туда не забредёт чужак, после выхода данных в дикую природу уже невозможно оценить угрозы, с которыми они, вероятно, столкнутся, и эффективно ими управлять. Ни один человек не способен контролировать путь данных или определить намерения всех участников связи, не говоря уже об их подходе к безопасности. В столь сложной сети вероятность того, что промежуточная сторона станет злоумышленником, и не пренебрежимо мала, и не проста для оценки. На деле даже человек, с которым вы устанавливаете законную связь, может преследовать скрытые цели или просто оказаться немного любопытным.

Попытки, так сказать, незапрошенного получения данных при выполнении через Интернет отличаются и по нескольким другим причинам. Самое главное: они не обязаны быть целевыми и не ограничены определённым сегментом физической инфраструктуры. Поскольку со стороны атакующего требуется так мало усилий, такие попытки становятся жизнеспособным способом получить потенциально интересные данные ещё до выяснения, как именно извлечь из этих знаний прибыль или иную пользу. К тому же граница между добром и злом размывается ещё сильнее: атакующим может быть ваш лучший друг. Выгодность общего шпионажа и наблюдения ради маркетинговой разведки и профилирования слишком соблазнительна, чтобы многие могли устоять. Мир предоставления услуг не чёрно-белый, а гибкая этика для многих людей представляет собой вполне жизнеспособную деловую модель.

Эта часть книги посвящена угрозам, присущим открытой конструкции Интернета, и способности других людей получать о вас гораздо больше сведений, чем вы можете ожидать, и больше, чем когда-либо понадобилось бы для оказания услуги вроде интересного веб-сайта или приятной сетевой игры. В Интернете противник уже не одинокий безумец, сидящий через дорогу и разглядывающий LED коммутатора через высокотехнологичный телеобъектив. Рассматриваемые здесь уязвимости позволяют массово профилировать и отслеживать людей, собирать информацию, вести промышленный шпионаж и сетевую разведку, а также анализировать цель перед атакой. И они гораздо реальнее описанных ранее сценариев.

Угрозы необходимо понимать, чтобы поддерживать осознанный уровень защиты приватности или, быть может, развернуть эффективное наблюдение за своими пользователями либо совершенными незнакомцами, приближающимися к вашим системам. Понимание также помогает сохранить рассудок в мире, где грань между заботой о приватности и клинической паранойей довольно тонка.

Я начну с изучения набора основных сетевых протоколов Интернета и их последствий для приватности. Приступим?

Язык Интернета

Официальный язык Интернета называется Internet Protocol, а самый распространённый диалект обозначен как версия 4. Протокол, заданный в RFC 793,[1] позволяет стандартизованно передавать данные на огромные расстояния и через самые разные сети с минимальными усилиями. IP-пакеты образуют третий уровень рассмотренной ранее модели OSI и состоят из заголовка со сведениями, необходимыми для доставки порции данных к окончательному месту назначения, удалённой конечной точке, и полезной нагрузки из информации верхних уровней, непосредственно следующей за заголовком.

Маршрутная информация, которую отправитель помещает в IP-пакет до отправки, состоит из адресов источника и назначения и набора параметров, упрощающих передачу данных либо повышающих её надёжность и производительность. Когда машина в локальной сети хочет связаться с удалённой стороной, недоступной непосредственно по проводу, по крайней мере насколько известно хосту, она пересылает IP-пакет с адресом конечного получателя, заключённый в кадр нижнего уровня. Кадр адресован локальной машине, которая считается шлюзом из сети отправителя и в неё. Шлюзовая машина - не более чем многосетевая система, присутствующая более чем в одной сети и служащая точкой соединения между ними. Предполагается, что шлюз знает, как направить пакет во внешний мир, что с ним делать и кто должен получить данные следующим, если прежде чем они достигнут адресата, потребуется участие других сторон.

Системы, участвующие в маршрутизации трафика от локального шлюза до сети назначения, читают сведения уровня IP и на основании своих знаний о путях к определённым сетям решают, как передать данные дальше. В данном контексте сеть определяется как совокупность сетевых адресов, находящихся в определённом месте.

Наивная маршрутизация

В простейшем виде маршрутизатор использует постоянную таблицу маршрутизации, посредством которой различает набор локальных сетей, куда способен доставить трафик непосредственно, и неизвестный ему внешний мир. Поэтому весь трафик, предназначенный за пределы локальной сети, необходимо передавать маршрутизатору более высокого порядка, который предположительно лучше представляет, куда доставить данные.

На рисунке 9-1 приведён пример структуры маршрутизации. Отправитель слева пытается послать пакет системе, чей адрес принадлежит сети C, о которой отправителю ничего не известно. Чтобы обеспечить доставку, он отправляет трафик локальному шлюзу в надежде, что тот знает, где искать получателя. Но эта система, маршрутизатор 1, способна добраться только до собственной сети отправителя и сети A, не имеющей ничего общего с C. Поскольку цель не находится в их локальной сети, маршрутизатор решает, что лучше всего просто передать пакет маршрутизатору WAN более высокого ранга, маршрутизатору 2, который случайно доступен ему локально.

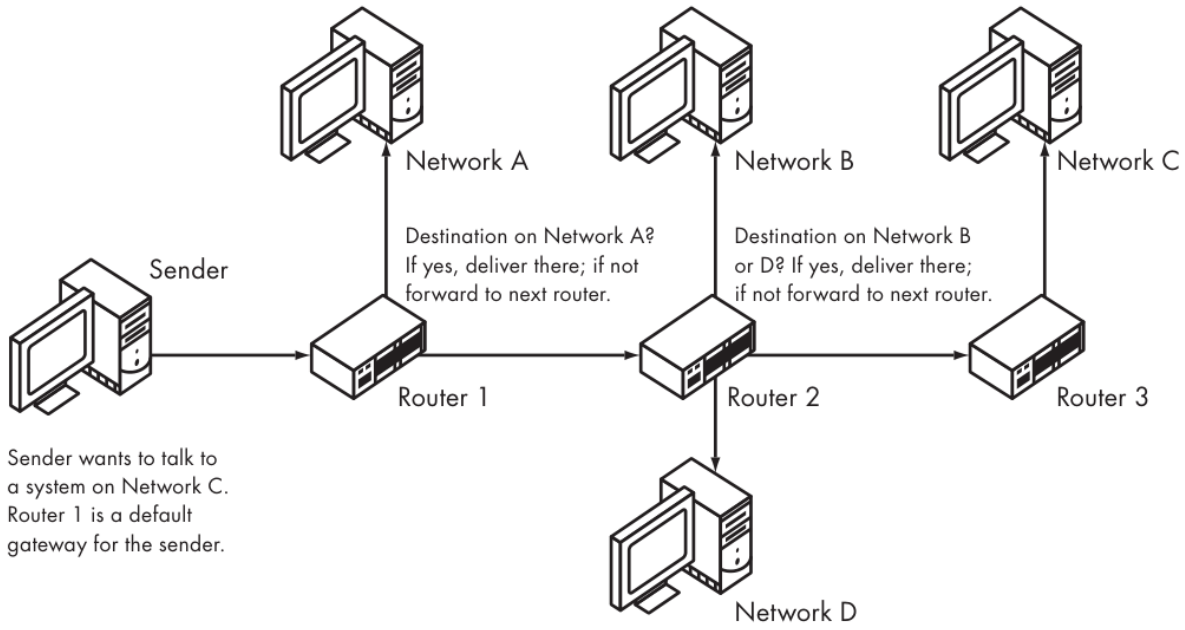


Рисунок 9-1. Наивная схема маршрутизации глобальной сети. Отправитель хочет связаться с системой в сети C. Его шлюз по умолчанию, маршрутизатор 1, может доставить пакет в сеть A или передать дальше. Маршрутизатор 2 непосредственно достигает сетей B и D, а маршрутизатор 3 доставляет трафик в сеть C.

Устройство также не имеет непосредственного соединения с сетью C и может напрямую достичь лишь хостов сетей B и D. Однако оно знает, что адрес назначения обслуживает маршрутизатор 3, который наверняка поймёт, что делать. Поэтому пакет пересылается ему, и теперь маршрутизатор 3 может локально доставить трафик окончательному получателю, после чего все вправе радоваться и праздновать очередной успех.

Маршрутизация в реальном мире

На практике сети часто обладают высокой избыточностью и не имеют строго линейной архитектуры. Их сложная древовидная структура затруднила бы выбор оптимального и наиболее экономичного пути при статической конфигурации. И это ещё не считая необходимости успевать за всеми изменениями инфраструктуры по мере роста сети.

Поэтому, когда трафик достигает магистрального маршрутизатора, применяется более разумная стратегия. Магистральный маршрутизатор, находящийся в ведении сетевого оператора, - это выделенное устройство WAN, связывающее множество сетей под управлением конкретного провайдера в сложную сущность, называемую автономной системой. Магистральные маршрутизаторы обычно имеют интерфейсы к другим крупным маршрутизаторам и используют усовершенствованный алгоритм поиска пути и объёмную «телефонную книгу» сетевых блоков и их местонахождения, динамически управляемую Border Gateway Protocol. Так они находят лучший маршрут данных к системе назначения и не передают вслепую задачу доставки трафика какой-нибудь системе в надежде, что та сумеет правильно переслать его дальше.

Адресное пространство

Разумеется, процесс был бы совершенно непрактичным, если бы сети назначения состояли просто из набора адресов, произвольно назначенных устройствам по всему миру. В определении

автономной системы пришлось бы перечислять все адреса, и оно легко разрослось бы до огромных размеров. Для решения проблемы непрерывные блоки адресного пространства назначаются поставщикам магистральных услуг, а те позднее сдают меньшие блоки конечным пользователям или небольшим провайдерам. Маршрутизация к сети провайдера основана на поиске IP-адреса назначения среди диапазонов, выделенных данной организации, а затем внутри сети - на дополнительном поиске в более подробных таблицах маршрутизации. Поэтому автономную систему можно определить как диапазон адресов IPv4 или набор таких диапазонов посредством сетевой маски.

Отдельный адрес IPv4, уникально идентифицирующий конечную систему во всей связи Internet Protocol, устроен довольно просто: он состоит из 32 битов, для удобства разделённых на 4 байта, что в сумме даёт 4 294 967 296 возможных адресов. Адрес традиционно записывается четырьмя 8-битными значениями от 0 до 255, разделёнными точками. Например, `195.117.3.59` соответствует 32-битному целому числу `3279225659`.

Непрерывные блоки IP-адресов служат основой маршрутизации пакетов. Поверх адресации IPv4 они задаются разделением IP-адреса на часть, фиксированную и постоянную для всех систем автономной сети, и часть, которой владелец сети присваивает разные значения, выдавая компьютерам уникальные идентификаторы.

При определении сети набор старших битов IP, теоретически от 1 до 31, а на практике от 8 до 24, резервируется как сетевой адрес. Фиксированную часть адреса разделяют все адреса, принадлежащие этой конкретной сети и предположительно направляемые в неё. Оставшимся младшим битам можно свободно присваивать значения, выдавая адреса системам внутри сети.

Исторически, согласно RFC 796,[2] размер сети, то есть количество зафиксированных старших битов, зависел от адреса и мог быть определён по самому сетевому адресу. Только на основании самых старших битов адреса группировались в сети класса A, где фиксированы 8 старших битов и доступно более 16 миллионов пользовательских адресов; сети класса B с 16 фиксированными битами и более чем 65 000 хостов; либо сети класса C с 24 фиксированными битами и 256 возможными хостами. Поэтому, если IP-адрес вашей системы начинается с числа 1, можно сказать, что перед нами сеть класса A и все остальные системы с этим префиксом находятся рядом с вашей машиной.

В своё время это казалось удобным, но адресное пространство IPv4 значительно сократилось после того, как первые пользователи, Армия США, Xerox, IBM и другие гиганты, на заре Интернета получили по нескольку сетевых адресов класса A и, похоже, не слишком стремились вернуть их, хотя не использовали для общедоступной инфраструктуры и малой доли доставшегося пространства. Кроме того, после коммерциализации Интернета, когда IP-адреса стали ресурсом, за который пользователи платили, те начали требовать части адресного пространства, лучше соответствующие потребностям: одним требовалось всего четыре адреса, другим - непрерывное пространство из 8 000. Пользователи стали перепродавать или иначе делить своё пространство Интернета.

В результате современное адресное пространство разделено причудливым образом: из крупных, во всём остальном непрерывных блоков часто вырезаются и перенаправляются крошечные части при общем пренебрежении первоначальной схемой. Теперь каждый сетевой адрес сопровождается сетевой маской, поскольку по одному IP уже нельзя определить, в какой сети находится система. Биты маски устанавливаются в позициях, которые должны быть зафиксированы в сетевом адресе, и обнуляются там, где внутри сети разрешено свободное изменение.

Как показано на рисунке 9-2, фиксация 24 битов в сети `195.117.3.0` оставляет 8 изменяемых младших битов. Это позволяет создать 256 принадлежащих сети адресов от `195.117.3.0` до `195.117.3.255`, хотя некоторые реализации заставляют резервировать первый и последний для специальных целей, оставляя лишь 254 возможных хоста. С таким сравнительно простым

описанием сети адресов легко определить, какие адреса принадлежат ей и потому должны доставляться системе, являющейся её шлюзом, а какие нет.

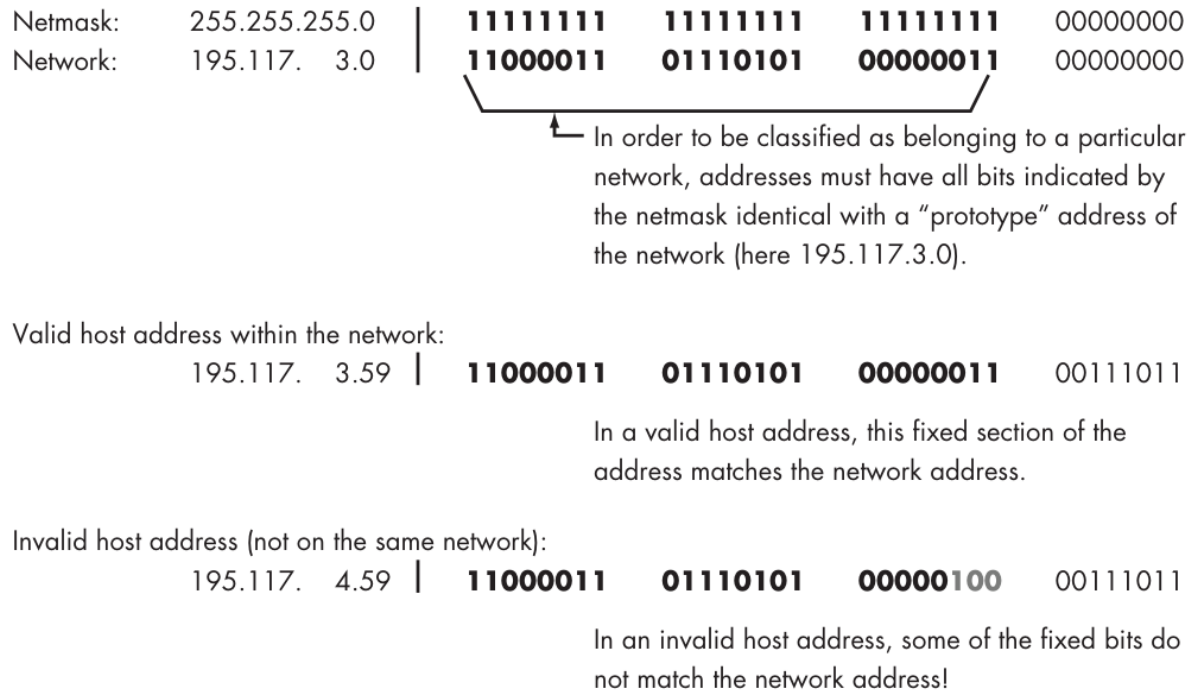


Рисунок 9-2. Правила сетевой адресации. Чтобы считаться принадлежащим конкретной сети, адрес должен совпадать с её «образцом», здесь `195.117.3.0`*, во всех битах, отмеченных маской. Адрес* `195.117.3.59` *допустим, а* `195.117.4.59` *не принадлежит той же сети, поскольку часть фиксированных битов не совпадает.*

Хотя схема адресации может показаться запутанной и излишне сложной, она успешна: позволяет связывать совокупности адресов с конкретными системами и различать системы при минимальных вычислительных затратах. Интернет во всей своей сложности обычно находит систему за действительно короткое время и не требует большого обслуживания.

Отпечатки на конверте

Мы знаем, как данные попадают из точки A в точку B, но происходящее по дороге интереснее выбора пути. Поэтому присмотримся к тому, чем обмениваются маршрутизаторы и наши конечные системы. Можно подумать, что больше всего интересного содержит сама полезная нагрузка пакетов Интернета, учитывая всю личную почту и странные материалы, которыми каждую секунду обмениваются по всему миру, но здесь есть гораздо больше, чем видно глазу.

Формат IP-пакетов, применяемых для маршрутизации данных, и сведения четвёртого уровня, заключающие сами прикладные данные, определены RFC довольно строго и с удивительно малой неоднозначностью. Однако даже при грамотной реализации стека TCP нижележащая информация способна последовательно давать получателю значительно больше ценных сведений, чем сама полученная полезная нагрузка. Раскрытие на этом уровне непреднамеренно и неожиданно, но, чтобы узнать о нём больше, необходимо внимательнее рассмотреть устройство базовых протоколов.

Internet Protocol

Сначала основы. Internet Protocol предоставляет универсальный механизм доставки на большие расстояния на третьем уровне модели OSI. Он содержит набор параметров, которые должны интерпретировать и при необходимости изменять промежуточные системы. Заголовок показан на рисунке 9-3.

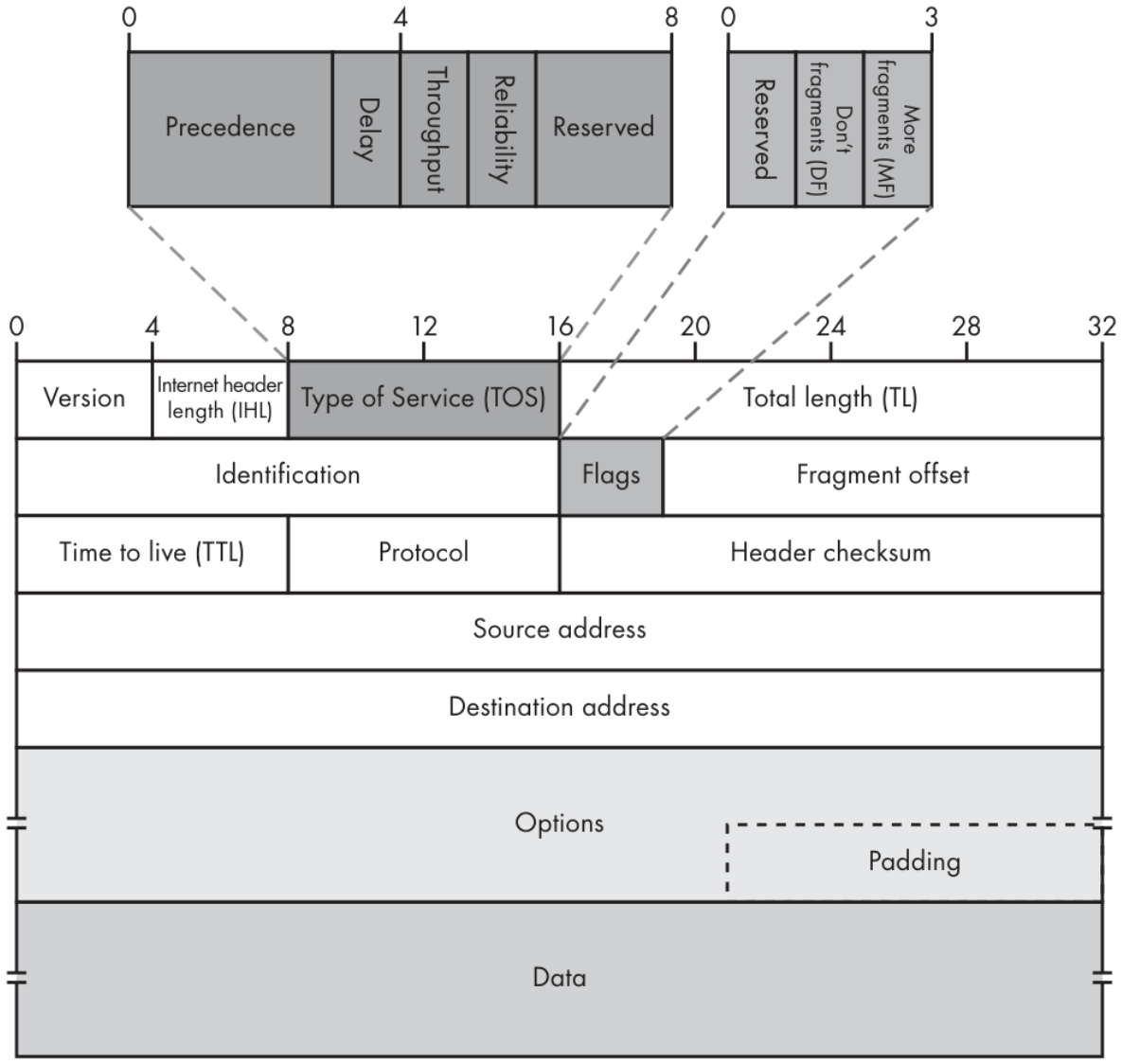


Рисунок 9-3. Структура заголовка IP. Показаны версия, длина интернет-заголовка, тип обслуживания, общая длина, идентификатор, флаги и смещение фрагмента, TTL, протокол, контрольная сумма, адреса источника и назначения, параметры, дополнение и данные.

Версия протокола

Четырёхбитное значение, зафиксированное как 4, `0100`, во всех пакетах IPv4. IPv4 является стандартным, а во многих случаях единственным поддерживаемым протоколом третьего уровня в Интернете. Попытки перейти к более совершенной реализации IPv6 до сих пор не были особенно успешными. Автор готов предположить, что причина, возможно, в том, что новый расширенный формат IP-адреса обычному системному администратору гораздо сложнее запомнить.

Поле длины заголовка

Четырёхбитное значение, указывающее общую длину самого IP-заголовка в количестве 4-байтовых блоков. Шестнадцать значений поля позволяют выражать длину от 0 до 60 байтов. Параметр сообщает реализации, где закончить разбор IP-заголовка, длина которого может меняться из-за дополнительных «параметров», добавляемых в конец непосредственно перед содержимым верхнего уровня. Он также позволяет пропустить часть IP-заголовка, не просматривая и не понимая параметры полностью, и сразу перейти к данным.

Поскольку параметры IP обычно не применяются ни для чего, кроме диагностики - они, например, позволяют принудительно задать определённый маршрут пакета, но дают немного сверх того, - почти все встречающиеся в природе IP-пакеты имеют заголовок длиной 20 байтов. Это означает, что поле установлено в 5, и соответствует длине фиксированной части заголовка. Значения меньше 20, разумеется, ошибочны, и вменяемая реализация такой пакет не принимает. Однако вменяемость не является практическим правилом.

Поле типа обслуживания, восемь битов

Значение этого поля обычно довольно невелико. Оно описывает приоритет маршрутизации на основе честности: отправителю доверяют действовать добросовестно и позволяют указать, имеет ли трафик особую важность или требует ли иного специального обращения. Значение иногда используется в локальных установках, где подобное доверие допустимо, но в открытом Интернете его часто игнорируют.

Поле состоит из трёх частей:

• первые три бита задают приоритет;

• следующие четыре обозначают желаемый способ маршрутизации посредством абстрактных понятий вроде «высокой надёжности» или «малой задержки», толкование которых оставляется маршрутизатору;

• последняя часть, один бит, зарезервирована и должна быть установлена в 0. Ну да, конечно.

Общая длина пакета, 16 битов

Двухбайтовое поле указывает общую длину IP-пакета вместе с полезной нагрузкой. Хотя наибольшее возможное значение равно 65 535, максимальный размер пакета часто ограничивается гораздо меньшим значением из-за рамок протокола нижнего уровня. Например, максимальная единица передачи Ethernet, MTU, равна 1 500 байтам, поэтому подключённая к Ethernet система не отправляет пакеты больше этого предела. MTU свыше приблизительно 16-18 килобайтов практически не встречаются; чаще всего значения лежат между 576 и 1 500 байтами.

> **Примечание.** Забавный факт: ограничение размера IP-пакета в `N` байтов, возникающее из параметра MTU, одновременно задаёт минимальную долю служебных данных для любого IP-трафика. На каждые `N-20` байтов, отправляемых на верхнем уровне, всегда добавляется не менее 20 байтов заголовка.

Адрес источника

Это 32-битное значение, IP-адрес в рассмотренном выше формате, должно представлять исходную конечную точку связи. Поскольку IP-пакет готовит отправитель, а на границе исходной сети почти никому не выгодно проверять правильность параметра, одному этому значению по-настоящему доверять нельзя. Однако оно хорошо подсказывает, кому отвечать, и, если у нас есть основания доверять подсказке, с её помощью можно обратиться к отправителю. Намеренная подделка данного значения обычно называется IP-спуфингом.

Адрес назначения

Это 32-битное значение задаёт окончательное место назначения трафика. Как и все прочие параметры IP, его по своему усмотрению выбирает отправитель, а промежуточные системы используют для правильного направления пакета.

Идентификатор протокола четвёртого уровня

Восьмибитное значение указывает, что переносится в качестве полезной нагрузки IP-пакета: TCP, UDP, ICMP или более экзотический вариант, о котором вскоре будет рассказано подробнее.

Время жизни, TTL

TTL - восьмибитный «счётчик уничтожения» IP-трафика. Чтобы избежать бесконечных петель, когда таблицы маршрутизации приходят в ужасающее состояние, счётчик уменьшается на единицу при каждом прохождении промежуточной системы или пребывании в очереди передачи в течение определённого времени. Достигнув нуля, пакет отбрасывается, а отправитель может быть милосердно уведомлён пакетом ICMP. Значение TTL, как и все остальные, выбирает отправитель, но из-за разрядности оно не может превышать 255.

Любопытное побочное свойство TTL состоит в возможности построить маршрут до удалённой системы. Сообщение с TTL 1 истекает на первом встретившемся по пути к указанному назначению маршрутизаторе, и отправитель получает от него сообщение ICMP; сообщение с TTL 2 истекает на следующем переходе и так далее. Отправляя пакеты с постепенно возрастающими TTL и наблюдая за источником ответов ICMP «время жизни превышено», можно нанести на карту маршрутизаторы и другие устройства с IP по пути к назначению. Приём называется traceroute и широко применяется для диагностики проблем маршрутизации и анализа перед атакой.

Польза для атакующего в том, что некоторых результатов можно достичь, не компрометируя намеченную жертву. Чтобы скомпрометировать `www.microsoft.com`, можно вместо него нацелиться на маршрутизатор сети, где размещён сервер, или маршрутизаторы его интернет-провайдеров в надежде перехватить весь трафик и возвращать поддельные ответы. Это фактически отсечёт настоящий сервер и путём выдачи себя за него создаст для внешнего мира впечатление, будто сайт `www.microsoft.com` изменился. Разумеется, это лишь пример.

Флаги и параметры смещения

Эти 16-битные значения управляют интересной и, возможно, самой ущербной стороной маршрутизации IP-пакетов. Параметры используются, когда промежуточная система должна передать большой пакет через соединение с MTU меньше размера пакета. В таком случае пакет в неизменном виде в среду не «помещается».

Возьмём произвольный пример: подключённый к Ethernet отправитель может послать пакет размером до 1 500 байтов и часто именно так и поступает. Но если первый встретившийся маршрутизатор соединяет локальную LAN с DSL-модемом, возникает проблема: распространённый MTU соединения DSL, которое само обычно представляет причудливую комбинацию инкапсуляций поверх других протоколов, равен 1 492. Поэтому пакет размером 1 500 байтов просто не поместится.

Учитывая огромное разнообразие соединений, благодаря которым работает Интернет, проблема серьёзна. Её решают разделением, фрагментацией, IP-пакета, точнее его полезной нагрузки, на несколько отдельных IP-пакетов и добавлением сведений, позволяющих получателю собрать нагрузку заново перед передачей верхним уровням. Получается новый набор пакетов, помещающихся в конкретное соединение. Смещение каждого фрагмента указывает, куда вставить его часть нагрузки, когда окончательный получатель попытается восстановить исходный пакет.

Во всех фрагментах, кроме последнего, в заголовке также установлен специальный флаг more fragments, MF. Когда система назначения получает пакет с флагом MF либо пакет с заданным смещением части, но без MF, что обозначает последний кусок разделённого пакета, она понимает, что нужно выделить временную область памяти для повторной сборки исходного пакета и дождаться всех остальных частей, прежде чем продолжать обработку.

На рисунке 9-4 показаны фрагментация и сборка: слишком большой пакет сначала разделяется на две части, а затем полностью собирается получателем, хотя части приходят не по порядку.

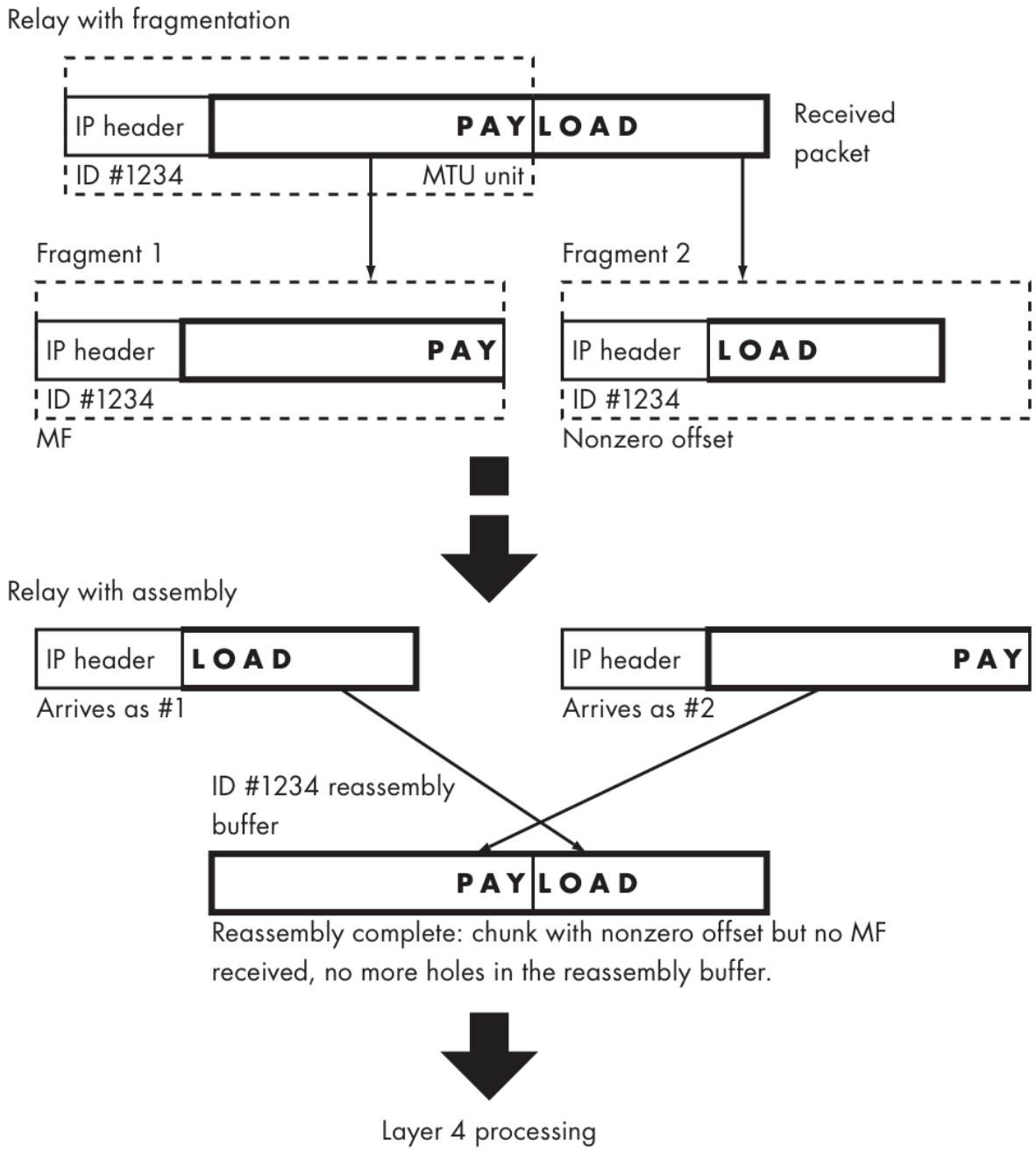


Рисунок 9-4. Процесс фрагментации и повторной сборки пакета. Пакет с идентификатором 1234 разделяется на два фрагмента. Второй приходит первым и помещается в буфер; после получения первого исчезают пробелы, и собранная полезная нагрузка передаётся обработке четвёртого уровня.

Хотя процесс работает, он несколько неэффективен. Системам требуется время на фрагментацию и сборку трафика, а последние части часто несут мало полезной нагрузки - лишь несколько байтов, не поместившихся в соединение другого типа. Конечно, лучше, если отправитель сможет определить наименьший MTU между собой и назначением, также называемый MTU пути, или PMTU, и соответственно формировать пакеты. К сожалению, IP не предлагает гибкого и чистого способа это реализовать, но исследователи всё равно придумали хитроумную заплатку.

Согласно ей, система, реализующая обнаружение PMTU, устанавливает специальный флаг DF, don’t fragment, «не фрагментировать», во всём исходящем трафике. Если маршрутизатор не может передать пакет DF без фрагментации, он должен вместо этого отбросить его и послать

соответствующее сообщение ICMP: «требуется фрагментация, но установлен DF». Получив сообщение, отправитель корректирует ожидания, сохраняет результат и продолжает с более подходящими размерами пакетов.

> **Примечание.** Практика, заданная в RFC 1191,[3] предполагает, что разовая цена повторной отправки отброшенного пакета лучше постоянной потери производительности из-за фрагментации. Однако приём весьма спорный: не все устройства посылают правильные уведомления ICMP, а исторически такого требования не существовало. Поэтому включение PMTUD, обнаружения PMTU, может лишить отправителя возможности связываться с некоторыми сайтами или приводить к зависанию передачи файлов, причину которого чрезвычайно трудно диагностировать.

Идентификационный номер

Идентификационный номер, ID, - 16-битное значение, различающее IP-пакеты при фрагментации. Без IP ID одновременная фрагментация двух пакетов привела бы к тому, что при сборке их фрагменты сильно перемешивались, менялись местами или повреждались иным образом.

IP ID однозначно различают несколько буферов сборки разных пакетов. Значение для этой цели часто выбирается простым увеличением счётчика при отправке каждого пакета: первый пакет системы имеет IP ID 0, второй - IP ID 1 и так далее.

> **Примечание.** В системах с включённым PMTUD уникальные IP ID не нужны, поскольку теоретически фрагментации не происходит, и значение часто устанавливается в 0. Возможно, это не особенно мудро, ведь некоторые довольно распространённые устройства склонны игнорировать флаг DF.

Контрольная сумма IP

Контрольная сумма - 16-битное число, предоставляющее простейший способ обнаружения ошибок. Её необходимо пересчитывать на каждом переходе, поскольку такие параметры, как TTL, меняются, поэтому алгоритм создан быстрым и не отличается особой надёжностью. Хотя в современном мире «контрольная сумма» является суммой только по названию и использует алгоритмы вроде CRC32 или криптографически безопасные сокращающие функции, контрольная сумма IP действительно представляет собой сумму или её разновидность с несколькими побитовыми отрицаниями,[1] добавленными, чтобы запутать противников. Если серьёзнее - чтобы при распространённых ошибках передачи у суммы было меньше шансов случайно остаться правильной.

За пределами Internet Protocol

Одно из следствий множества решений, принятых при создании IPv4, - отсутствие разумной гарантии надёжности даже при надёжной работе самой сети. Хотя номера IP ID призваны уменьшать риск столкновений при сборке, их сравнительно небольшой 16-битный размер, допускающий 65 536 значений, позволяет изредка возникать проблемам, когда одновременно собираются два пакета с одинаковым IP ID. Кроме того, контрольных сумм IP-заголовка попросту недостаточно для надёжной защиты целостности: хотя это маловероятно, случайное изменение пакета всё же может дать ту же сумму. А если сеть действительно откажет, выяснить, какие данные

пропали, невозможно, даже если отказ вызван столь простой причиной, как кратковременная перегрузка одного сетевого компонента.

Наконец, Internet Protocol не позволяет проверить отправителя сообщения и просто верит, что настоящий отправитель указан в IP-заголовке. Предоставление необходимой функциональности проверки целостности и надёжности оставлено протоколам верхнего уровня, а нужна она чаще всего. Поэтому поверх IP требуются протоколы верхнего уровня.

TCP и, в меньшей степени, UDP не только предоставляют крайне необходимую защиту трафика, но и позволяют указать получателя или отправителя на уровне, выходящем за рамки указания определённой системы.

В то время как IP-заголовок содержит лишь достаточно сведений для маршрутизации трафика между двумя системами и недостаточно для выбора приложения, которому следует доставить информацию, UDP и TCP идут дальше: переходят в область конечной системы и сообщают получателю, какому приложению направить входящие данные.

User Datagram Protocol

Согласно RFC 768,[4] UDP предоставляет минимальное расширение функциональности IP. Он добавляет механизм локальной доставки данных, но сохраняет почти ту же ненадёжность нижележащего уровня и столь же малые накладные расходы. Связь по UDP можно уподобить телефонной службе, где слова иногда меняются местами или выпадают из предложений и нет надёжного определения звонящего, зато звонок дёшев, а отвечают быстро.

Заголовок UDP на рисунке 9-5 имеет минимальный набор функций и сравнительно прост. Он вводит небольшой набор параметров, которые система назначения может истолковать и применить для маршрутизации пакета конкретному приложению или проверки того, не испортилась ли полезная нагрузка по дороге.

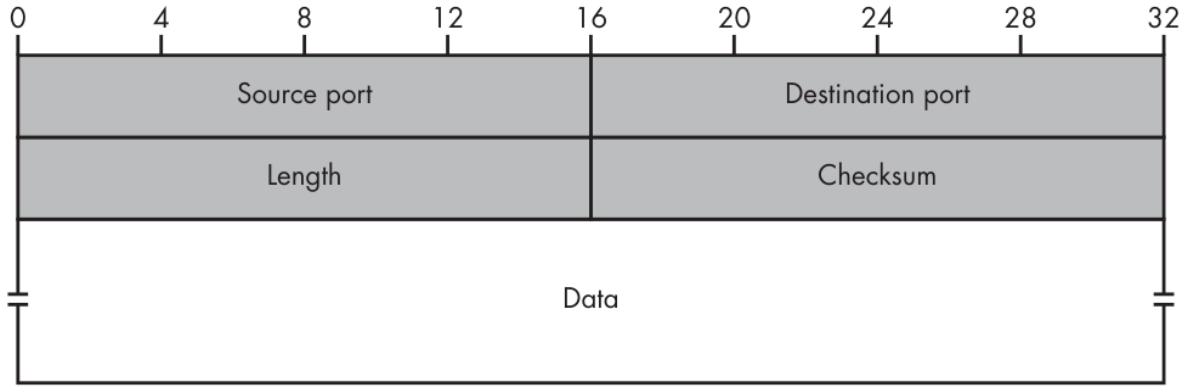


Рисунок 9-5. Структура заголовка UDP. Заголовок содержит порты источника и назначения, длину и контрольную сумму, после чего следуют данные.

UDP применяется для одиночных запросов, в других ситуациях, где не нужно поддерживать состояние, и когда производительность и малые служебные расходы важнее надёжности. Например, UDP широко используется для разрешения имён Domain Name System, DNS, простых протоколов сетевой загрузки и автоматической настройки BOOTP, технологий потокового мультимедиа, общего доступа к сетевым файловым системам и так далее.

Введение в адресацию портов

Помимо адресов источника и назначения UDP вводит понятие портов источника и назначения и разделяет его с TCP, более совершенным протоколом четвёртого уровня, рассмотренным далее. Порт - определённое 16-битное число, которое либо выбирает конечное приложение, желающее отправлять или получать данные, либо назначает ему операционная система; последнее называется эфемерным портом.

Порт служит средством направления данных конкретному приложению или службе многозадачной системы, чтобы программы могли одновременно поддерживать связь. Например, процесс сервера имён может слушать входящие запросы на порту 53, а системная служба журналирования - трафик, адресованный порту 514. Порты позволяют клиентам одновременно разговаривать с этими процессами. Кроме того, когда реализация правильно разделяет пары исходных и целевых портов, два клиента с разными эфемерными исходными портами могут одновременно обращаться к одной службе, скажем порту 514, не создавая серьёзной путаницы в том, какое клиентское приложение должно получить какой ответ удалённой службы.

Чтобы система назначения различала связь, адресованную конкретному приложению, и доставляла её как ожидается, отправитель обязан указывать номер порта назначения во всём трафике. Для каждого клиентского приложения он задаёт отдельный исходный порт, чтобы ответ сервера попал правильному компоненту.

В этой схеме адресации портов четвёрка значений - исходный хост, исходный порт, целевой хост и целевой порт - обеспечивает правильное разделение трафика и управление сеансами для одновременных соединений, которые начинаются или заканчиваются в определённой системе. Конструкция означает, что до 65 535[2] клиентов с одного IP-адреса могут подключаться к внешнему миру и не более 65 535 служб способны одновременно слушать на одном IP-адресе без каких-нибудь хитроумных уловок. Едва ли это ограничение вскоре повлечёт ужасные последствия.

Краткое описание заголовка UDP

Показанный ранее на рисунке 9-5 заголовок UDP следует за IP-заголовком и предшествует собственно пользовательским данным в UDP-пакетах. Он состоит из нескольких полей: исходного и целевого портов по 16 битов, длины пакета и 16-битной контрольной суммы для дополнительной проверки целостности.

А теперь нечто совершенно иное: это...

Пакеты Transmission Control Protocol

TCP, RFC 793,[5] заголовок которого показан на рисунке 9-6, стремится предоставить надёжный потоковый способ содержательного разговора между двумя системами. TCP лучше UDP подходит для всех приложений, кроме тех, которым требуется обмениваться лишь простыми короткими сообщениями и одиночными выкриками.

Хотя технически соединение реализовано отдельными IP-дейтаграммами, проходящими через сеть, установленное соединение TCP, с точки зрения приложения являющееся виртуальным каналом, предоставляет режим связи, похожий на обычный телефонный разговор. В отличие от UDP-трафика, при использовании TCP можно быть уверенным, что получатель всегда принимает данные в отправленном виде, либо, если восстановление после ошибки невозможно, разговор полностью прекращается. В нормальных условиях можно быть уверенным и в личности звонящего, но это удобство обходится дороже и даёт меньшую производительность.

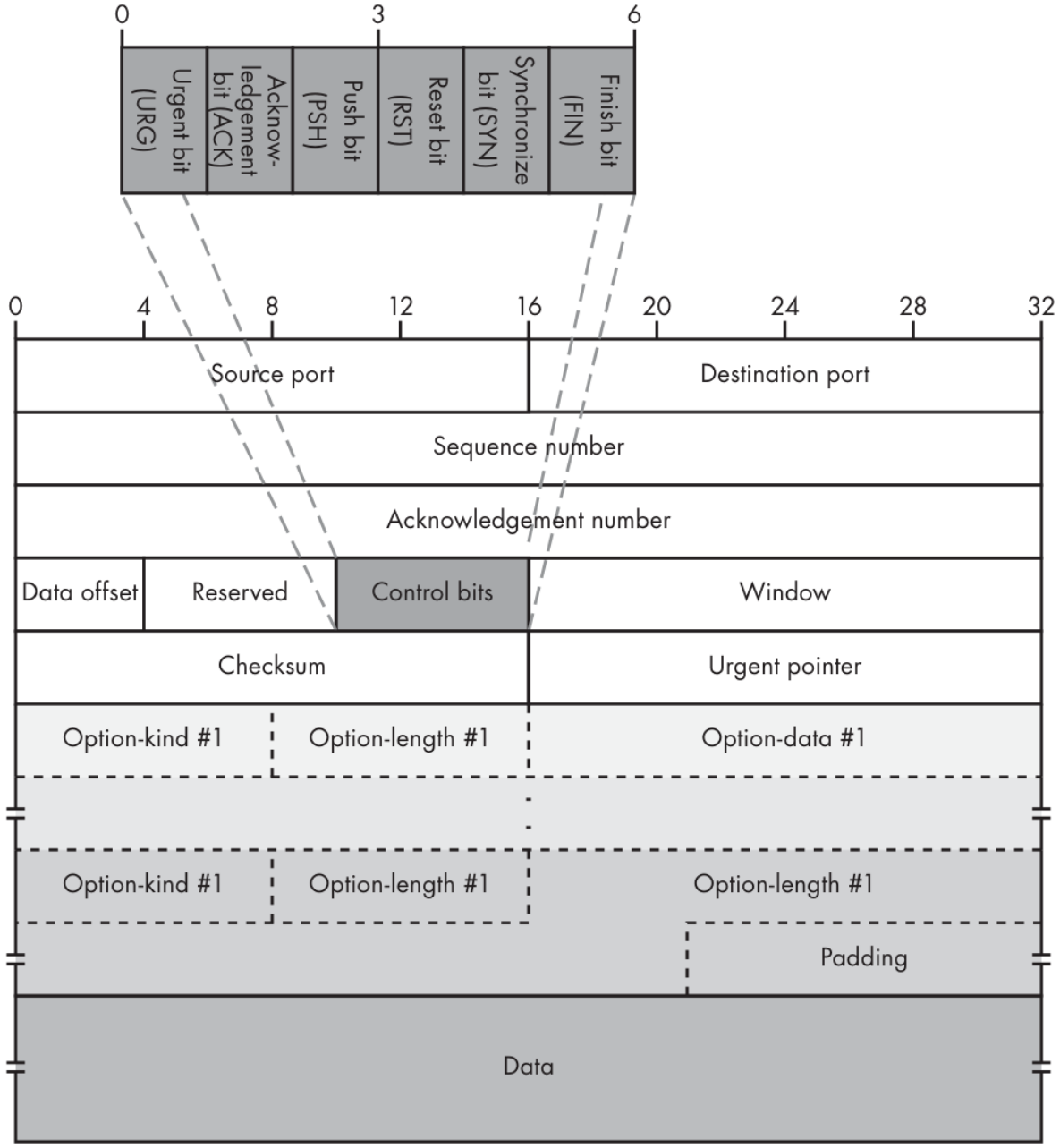


Рисунок 9-6. Структура заголовка TCP. Заголовок включает порты, номера последовательности и подтверждения, смещение данных, зарезервированную область, управляющие биты, окно, контрольную сумму, указатель срочности, параметры и дополнение; далее следуют данные.

В TCP две конечные точки сначала начинают соединение посредством так называемого алгоритма трёхэтапного рукопожатия. С помощью специальных, как правило пустых пакетов, содержащих только заголовки без фактической полезной нагрузки, стороны согласуют намерение, подтверждают личности друг друга и договариваются о начальных номерах последовательности и подтверждения. Эти 32-битные значения обеспечивают надёжную и непрерывную передачу, поскольку увеличиваются по мере отправки данных. Благодаря этому получатель может ставить входящие пакеты в очередь в правильном порядке и определять, не пропала ли какая-либо часть данных.

Управляющие флаги: рукопожатие TCP

Сеанс TCP начинается, когда удалённая система выражает желание подключиться к определённому порту машины назначения. Она посылает назначению пустой пакет с флагом SYN, то есть с установленным соответствующим битом заголовка, и начальным номером последовательности. После получения пакета любой ответ на него должен содержать этот номер, иначе он не будет принят. Если машина назначения не отправит правильный ответ за разумное время, пакет передаётся снова, пока доставка не удастся либо отправитель не решит, что прошло достаточно времени, и не прекратит соединение.

Номер последовательности гарантирует, что ответ на пакет исходит от настоящего получателя, а не от постороннего, который знает о предстоящей связи и намерен её захватить. Он также гарантирует, что ответ не является потерянным и заблудившимся пакетом предыдущего сеанса, который наконец добрался домой, а относится к конкретному запросу отправителя. При 32 битах и 4 294 967 296 возможных значениях вероятность столкновения значительно меньше, чем у 16-битных IP ID, поэтому и случайная неприятность, и успешное угадывание посторонним весьма маловероятны.

Получатель должен ответить на запрос SYN сходным пакетом, адресованным отправителю и использованному им исходному порту. В пакете следует установить флаг RST, ещё один бит заголовка, показывающий нежелание устанавливать сеанс: ни одна программа в этой конечной точке не готова принимать соединения. Пакет обязан также вернуть первоначальный номер последовательности вместе с ответом. Либо, в необычном случае, когда получатель действительно желает установить соединение и побеседовать с незнакомцем, он должен ответить сходным образом сформированным пакетом, но с флагами SYN и ACK, обозначающими принятие запроса. Ему также следует включить номер последовательности, которого он отныне ожидает во всех ответах данного сеанса.

В последней части рукопожатия отправитель передаёт один пакет ACK, чтобы удостовериться, что обе стороны знают обменённые ранее номера последовательности и подтверждения и одинаково понимают транзакцию. Если связь достигла этой стадии, обе конечные точки могут с разумной уверенностью считать отправителя и получателя теми, за кого они себя выдают. Почему? Потому что каждая способна наблюдать трафик, адресованный её адресу. Если бы одна конечная точка лишь подделывала IP-адрес, устанавливая ложное соединение от имени другого, она не знала бы, какой номер включить в ответ другой стороне. А другая сторона очень удивилась бы, обнаружив, что кто-то пытается послать ей незапрошенные пакеты SYN+ACK или ACK.

Протокол рукопожатия исключает возможность простой подделки трафика посторонним, но не устраняет угрозу со стороны враждебной привилегированной стороны, законно находящейся на пути между системами. Впрочем, по сравнению со слепой подделкой такой случай маловероятен.

> **Примечание.** Само собой, поначалу применение трудно предсказуемых начальных номеров последовательности не считалось важным, и многие системы использовали, например, простой возрастающий генератор. Со временем возможность для постороннего слепо установить сеанс, подделав рукопожатие TCP от конкретного источника, либо внедрить данные в уже установленное соединение стала несколько проблемной.[3] Теперь тщательный выбор начальных номеров последовательности TCP, чтобы наблюдатель не мог предсказать ответ системы на будущий пакет, считается необходимостью; для решения задачи придумано несколько подходов.[6]

После завершения рукопожатия стороны могут обмениваться данными, каждый раз взаимно подтверждая номера последовательности; пакеты с расхождением номеров больше допустимого «окна» просто игнорируются. Отныне номера также постоянно возрастают, отражая объём отправленных к данному моменту данных, благодаря чему назначение обрабатывает пакеты в

правильном порядке, даже если они приходят не по порядку. Для надёжности, если порция данных не подтверждается за разумное время, пакет или пакеты необходимо передать повторно.

Сеанс завершается, когда одна из сторон получает пакет FIN с правильным номером подтверждения. Если в какой-то момент одна система сильно раздражается и хочет внезапно прекратить сеанс - возможно, с её точки зрения говорить не о чем, сеанс истёк либо другая сторона грубо нарушила соглашение, - отправляется пакет RST.

Успешное законное рукопожатие TCP показано слева на рисунке 9-7. Справа изображён провал типичной атаки с подделкой IP, призванной создать сеанс от имени невинного наблюдателя, не намеренного обмениваться с целью никакими данными. Атакующий не видит и не может предсказать ответ, посылаемый системе, от имени которой он действует, и потому не способен завершить рукопожатие, не говоря уже о фактическом обмене данными внутри сеанса TCP.

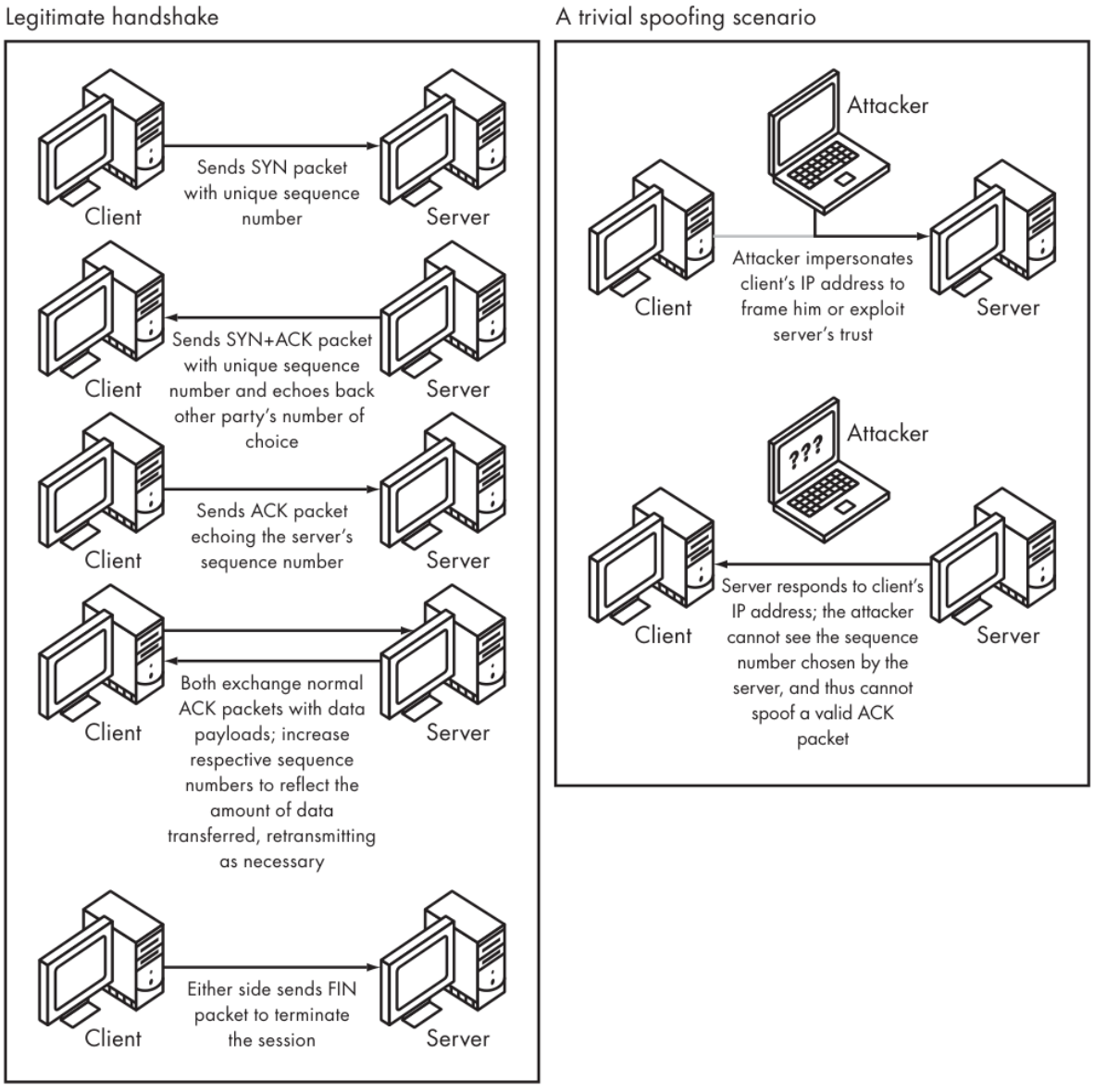


Рисунок 9-7. Полное рукопожатие TCP и провал обычной попытки подделки. В законном соединении клиент и сервер обмениваются SYN, SYN+ACK и ACK, затем данными и FIN. При подделке сервер отвечает настоящему адресу клиента, а атакующий не видит выбранный сервером номер последовательности и не может сформировать правильный ACK.

Как и предполагалось, TCP достаточно хорошо защищает от проблем надёжности сети и лучше подходит упорядоченной сеансовой связи. Но ценой становятся дополнительные служебные расходы на рукопожатие и хранение управляющих сведений о соединении обеими конечными точками. Поддержание состояния обходится дорого: для каждого соединения необходимо отслеживать номера последовательности и текущее состояние потока, этапы рукопожатия, обмена данными и закрытия; хранить копию всех отправленных, но ещё не подтверждённых данных на случай повторной передачи и так далее.

Из-за затрат памяти и производительности реализации стека TCP часто служат направлением атак отказа в обслуживании.

Другие параметры заголовка TCP

Другие параметры заголовка TCP также управляют важными сторонами толкования и доставки пакета. Они пригодятся позднее, когда мы попытаемся получить сведения об отправителе, лишь рассматривая предоставленные им данные пакета. Полный перечень полей TCP показан ранее на рисунке 9-6.

Порты источника и назначения

Эти 16-битные значения идентифицируют логический источник и конечную точку на исходной и целевой машинах. Они похожи на параметры исходного и целевого портов UDP, хотя пространства портов UDP и TCP в системе разделены. Одно приложение может слушать UDP-порт 1234, а другое - тот же номер порта в пространстве TCP. Трафик направляется согласно протоколу, указанному в IP-заголовке.

Номера последовательности и подтверждения

Эти 32-битные значения обеспечивают целостность сеанса. Номер последовательности - значение, которое отправитель ожидает получить обратно. Номер подтверждения - значение, возвращаемое отправителю, и имеет смысл только при установленном флаге ACK.

Смещение данных, не путать со смещением фрагмента IP

Поле указывает, где в пакете заканчивается заголовок и начинается полезная нагрузка. Как и у IP, длина заголовка TCP может меняться, если в его конец добавлены настройки переменной длины. Сведения позволяют легко перейти непосредственно к данным, не просматривая весь заголовок.

Флаги

Эти восьмибитные значения задают особые свойства пакета. Каждый назначенный бит поля представляет отдельный флаг и может независимо включаться или выключаться, поэтому флаги TCP разрешено произвольно сочетать. Основные флаги SYN, ACK, RST и FIN определяют толкование пакета с точки зрения сеанса TCP, как рассмотрено выше. Вторичные управляют отдельными сторонами доставки нагрузки и расширенными функциями вроде уведомления о перегрузке, но не меняют само состояние соединения.

Примечание. Хотя флаги можно сочетать по своему усмотрению, многие возможные комбинации попросту незаконны или бессмысленны. Например, SYN+RST не имеет смысла и формально не разрешён. Только некоторые сочетания осмысленны для рукопожатия и обычной обработки данных. Разные системы по-разному отвечают на незаконные комбинации, поэтому отправка бессмысленных пакетов с необычными флагами стала популярным активным способом определения операционной системы.

Размер окна

Это 16-битное значение управляет наибольшим объёмом данных, который разрешено отправить, не дожидаясь пакета подтверждения. Большее значение позволяет послать больше данных сразу, но может ухудшить производительность, если часть потеряется или не будет подтверждена и потребуется повторная передача.

Контрольная сумма TCP

Простой 16-битный метод защищает целостность данных четвёртого уровня, подобно механизму контрольных сумм в заголовках UDP и IP.

Указатель срочности

Получатель интерпретирует поле только тогда, когда в пакете установлен вторичный флаг URG. Без URG значение области заголовка просто игнорируется. Флаг показывает, что отправитель просит получателя передать определённое сообщение приложению, обрабатывающему трафик, предположительно из-за «срочной» ситуации, чтобы пакет был вставлен в логический поток раньше обычного места; точное смещение задаёт указатель срочности. В нормальной связи механизм используется редко.

Параметры TCP

Блок параметров переменной длины в конце заголовка может задавать дополнительные настройки пакета. Иногда он пуст, имеет нулевую длину, но чаще реализует созданные позднее расширения протокола, не нарушая работу старых реализаций, которые их не понимают. Блок устроен так, что система может безопасно проигнорировать неизвестный параметр. Наиболее популярны следующие.

Maximum Segment Size, MSS

Это 16-битное значение равно максимальной единице передачи в сети отправителя за вычетом размера заголовков нижних уровней. Оно представляет наибольшую длину пакета, которую можно вернуть получателю без фрагментации по пути. Отправитель использует MSS для оптимальной производительности, когда получатель возвращает крупные порции данных, иначе потребовавшие бы фрагментации и связанных служебных затрат пропускной способности. К сожалению, конечная система устанавливает MSS исходя из своих лучших знаний о размере пакетов, допустимом в ближайшем сетевом окружении. Параметр не предотвращает распространённую фрагментацию

посреди пути на промежуточных системах, поэтому и нужно рассмотренное ранее обнаружение PMTU на уровне IP.

Масштабирование окна

Описанное в RFC 1232[7] восьмибитное значение расширяет диапазон поля размера окна первоначального TCP-заголовка. Опыт показал, что подтверждение каждых 64 килобайтов данных, максимума 16-битного параметра окна, может создавать узкое место при передаче больших объёмов, например мультимедийных файлов, по каналам с высокой пропускной способностью, но большой задержкой. Масштабирование позволяет увеличить окно и отправлять больше данных без ожидания подтверждения. Передача ускоряется, но при пропаже одного пакета может потребоваться повторно передать больший объём.

Параметры выборочного подтверждения, RFC 2018[8]

При большом окне потеря единственного пакета требует повторной передачи всей ещё не подтверждённой группы данных, что ужасно растрачивает пропускную способность. Для предотвращения этого был создан механизм выборочного подтверждения частей. Сначала конечные точки объявляют способность и желание использовать функцию посредством параметра Selective ACK Permitted, а затем подтверждают разрозненные блоки данных настоящим параметром Selective Acknowledgment в заголовках. Приём способен значительно повысить производительность ценой некоторых затрат памяти и обработки.

Параметр метки времени, два 32-битных значения

Ещё одно высокопроизводительное расширение, предложенное в RFC 1232. Механизм отправки и возврата меток времени, обычно некоторым образом соответствующих системному времени или времени непрерывной работы, позволяет каждой конечной точке оценивать время кругового пути трафика. Главное преимущество: отправитель измеряет типичное время доставки пакета и при отсутствии ответа раньше начинает повторную передачу TCP. Дополнительное применение - предотвращение столкновения номеров последовательности, PAWS, Protection Against Wrapped Sequence Numbers. Например, давно пропавший пакет может добраться до назначения после обмена несколькими гигабайтами данных и полного оборота счётчика последовательности.

EOL

Параметр следует толковать как конец параметров: он сообщает получателю, что завершающие данные не нужно обрабатывать как часть заголовка. Поскольку размер TCP-заголовка задаётся единицами длиннее одного байта, после размещения всех нужных параметров может остаться неиспользованное место перед началом полезной нагрузки, возможным только на полной четырёхбайтовой границе. EOL не даёт получателю пытаться анализировать эти данные.

Параметр NOP

Означает «ничего не делать» и просто игнорируется получателем. Отправитель может и должен дополнять пакет NOP, чтобы правильно выровнять некоторые многобайтовые параметры. Такое выравнивание требуется из-за ограничений производительности и архитектуры отдельных процессоров.[4]

T/TCP, Transactional TCP

Эзотерическое расширение поддерживает отдельные виртуальные сеансы, транзакции, внутри установленного сеанса TCP. Оно позволяет избежать расходов на рукопожатие при каждой отдельной операции с одноразовыми службами - подход, чаще применяемый, когда приложение хочет провести несколько самостоятельных транзакций с сервером. Расширение используется редко и полезнее всего некоторым системам баз данных; см. RFC 1644.[9]

Пакеты Internet Control Message Protocol

Пакеты ICMP, см. RFC 792,[10] применяются для диагностических сведений и уведомлений других типов протоколов. Логически ICMP считается частью третьего уровня, но его пакеты переносятся как полезная нагрузка IP и в этом смысле не отличаются от нагрузки четвёртого уровня. ICMP не переносит новых пользовательских данных между конечными точками, а предоставляет простейшую сигнализацию для IP. Структура заголовка показана на рисунке 9-8.

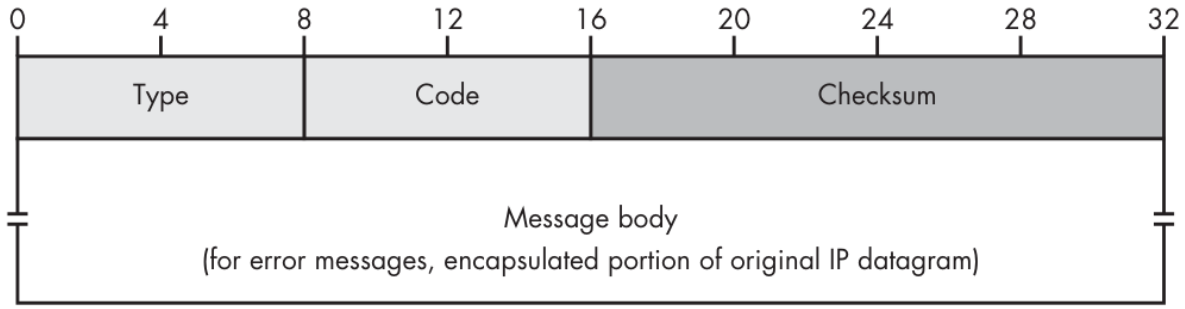


Рисунок 9-8. Структура заголовка ICMP. Заголовок содержит тип, код и контрольную сумму, после чего следует тело сообщения; в сообщениях об ошибках оно включает часть исходной IP-дейтаграммы.

Разнообразные сообщения ICMP отправляются в ответ на трафик TCP или UDP и обычно показывают, что определённый пакет нельзя доставить, его срок истёк по пути либо он отклонён по иной причине. Некоторые типы ICMP могут отправляться самостоятельно, например объявления маршрутизаторов, эхо-запросы, ping, и так далее.

Как и у UDP, заголовок ICMP прост и состоит из следующих полей.

Тип сообщения

Восьмибитное поле задаёт общую категорию события, вызвавшего отправку пакета, например «назначение недоступно». Поле также может нести самостоятельное сообщение, хотя такое применение встречается редко.

Код сообщения

Восьмибитное значение описывает точную проблему, если это применимо. Оно зависит от типа и может подробнее характеризовать состояние: «сеть недоступна», «хост недоступен», «порт недоступен», «связь административно запрещена». Граница между подробностями, которые следует помещать в тип сообщения и оставлять коду, неясна.

Контрольная сумма пакета

Поле проверяет, что пакет не повреждён, как в UDP и TCP.

Сам заголовок ICMP довольно прост и не даёт достаточно сведений, чтобы успешно разобраться в сообщаемой проблеме или определить, какой трафик породил сообщение. Эти сведения передаются в полезной нагрузке пакета непосредственно вслед за заголовком.

Хотя нагрузка ICMP зависит от сообщения, обычно она цитирует начало пакета, вызвавшего ответ.

Это позволяет получателю определить, к какому обмену относится сообщение и какое приложение следует уведомить о проблеме. Цитата также помогает убедиться, что отправитель ICMP-пакета действительно находится где-то на законном сетевом маршруте между двумя машинами, а не за его пределами. Иначе он не смог бы видеть фактически передаваемые данные, в частности точный номер последовательности в пакетах TCP. Это мешает злонамеренным наблюдателям посылать ложные сообщения о проблемах связи и заставлять одну конечную точку разрывать соединение - по крайней мере теоретически. Естественно, отличить добро от зла бывает довольно трудно, поскольку некоторые системы печально известны тем, что искажают или неверно цитируют исходные данные.

На сцену выходит пассивное снятие отпечатков

Как устройство этих протоколов связано с приватностью пользователя? Ответ несколько странен. Хотя устройство пакетов IP, TCP, UDP и ICMP в целом довольно строго, а передаваемые в заголовках сведения не особенно многословны, различия в том, как операционные системы добавляют в пакеты информацию, позволяют определить не только тип используемой ОС, но даже конкретную версию экземпляра машины. Различия особенно заметны при работе с трафиком, который спецификация не обсуждает ясно и должным образом или который не анализируется в обычных процедурах контроля качества, например входящим пакетом с незаконным сочетанием флагов SYN+RST.

Интенсивные исследования различения систем посредством стрессового испытания реализаций показали, что можно уверенно заключить: двух одинаковых реализаций семейства IP в операционных системах не существует. Сложный анализ часто позволяет отличить одну систему на немного разных платформах или близкие версии одной системы. Инструменты активного анализа вроде NMAP Фёдора, средства снятия отпечатков TCP/UDP и сканера портов, и Xprobe Офира Аркина, средства снятия отпечатков ICMP, эксплуатируют недостатки и странности каждой системы. Они определяют семейство и версию ОС, посылая разнообразные искажённые или необычные пакеты, а затем измеряя и анализируя вызванные ответы.

Изучение IP-пакетов: первые дни

Но способы снятия отпечатков систем этим не ограничиваются. Более того, тормошить удалённую систему подозрительными и легко обнаруживаемыми данными - пожалуй, наименее тонкий подход к задаче.

В начале 2000 года двое участников Subterrain Security Group, известные лишь под псевдонимами bind и aempirei, продемонстрировали, что сведения об удалённой сущности в сети часто можно получить без какого-либо навязчивого общения с ней и даже вообще не начиная связь. Их код и результаты впервые представили на DefCON 8, несколько переоценённой хакерской выставке своего рода, в 2000 году. Приём, ныне называемый пассивным снятием отпечатков, состоит в пассивном - вот неожиданность - наблюдении за обычным законным трафиком, исходящим от удалённой системы. Хотя его показатели гораздо тоньше и ограниченнее применявшихся Фёдором и предшественниками, хороший объём исследований, в который я с гордостью внёс несколько вкладов, дал достаточно наблюдений для поразительно высокой точности.

Чтобы лучше понять, что можно узнать из одного полученного по сети пакета, рассмотрим показатели, на которых строится пассивное снятие отпечатков, и сведения, которые они дают о другой стороне. Исследование основано на разборе самого распространённого трафика Интернета - законного пакета TCP в оболочке IP.

Начальное время жизни, уровень IP

Напомним: поле TTL управляет количеством систем, через которые пакет может пройти, прежде чем будет отброшен как недоставляемый. TTL уменьшается при каждом прохождении маршрутизатора, пока не достигнет нуля, после чего пакет отбрасывается.

Поскольку строгого требования к начальному значению поля нет, многие разработчики стека IP просто бросают кости при выборе значения по умолчанию для любимой системы. Пассивный наблюдатель не может определить точное начальное значение без дополнительных испытаний, ведь пакет наверняка прошёл несколько маршрутизаторов, но знает, что оно было выше наблюдаемого. Кроме того, среднее расстояние до удалённого компьютера в Интернете обычно не превосходит 15 переходов, а две системы редко разделены более чем 30. Поэтому можно уверенно предположить, что первоначальное значение находится между наблюдаемым TTL и наблюдаемым TTL + 30, но, конечно, меньше 256.

Зная начальные значения популярных ОС, можно приблизиться к вероятному семейству системы отправителя. Linux и производные BSD обычно используют 64, разработчики Windows - 128, а некоторые настоящие потомки Unix - 255. Определив по этому и другим признакам ОС отправителя, можно также вычислить его расстояние от точки наблюдения, вычтя наблюдаемый TTL из известного начального. Сопоставив результат с ранее наблюдавшимся или иным образом известным фактическим расстоянием до сети, мы можем сделать выводы об организации внутренней сети отправителя.

Флаг Don’t Fragment, уровень IP

Флаг DF говорит: «Если пакет не помещается в определённое сетевое соединение, не фрагментируй его, а просто отбрось». Наблюдая за установкой флага, можно определить, использует ли система описанный ранее механизм PMTUD, и получить ещё одну подсказку об ОС. Это также разделяет две крупные группы систем: приём применяют только новые реализации IP, а все прочие не заинтересованы включать флаг в исходящих пакетах.

Номер IP ID, уровень IP

Как упоминалось выше при обсуждении недостатков фрагментации, некоторые системы с PMTUD устанавливают IP ID в ноль во всём или части исходящего трафика, полагая, что фрагментации не будет, а также из-за опасений безопасности по поводу демонстрации номеров IP ID, которые будут рассмотрены в главе 13. Поэтому такие системы можно узнавать по нулевому IP ID входящих пакетов.

Однако есть ловушка. Некоторые ОС с PMTUD всегда устанавливают IP ID в ноль, но и другие системы порой могут выбрать нулевой IP ID просто потому, что возможных значений не так много. Иными словами, ненулевой IP ID позволяет уверенно заключить, что система не использует ноль для всей исходящей связи. Но нулевое значение может означать как особый вид системы с PMTUD, так и «обычную» систему, случайно выбравшую ноль для данного пакета.

Вероятность этого мала, но не пренебрежимо мала. Можно либо относиться к нулевым IP ID с долей сомнения и сужать набор возможных ОС только по ненулевым наблюдениям, либо несколько раз наблюдать один источник, подтверждая постоянное использование нулей.

Тип обслуживания, уровень IP

По замыслу поле должно выбираться согласно приоритету и типу трафика, подсказывая промежуточным системам, как обрабатывать пакет, но почти никогда так не происходит. Большинство ОС присваивает полю произвольное постоянное значение, поскольку разработчики могут установить что угодно, практически не повлияв на работу сети TCP. В зависимости от самолюбия разработчик либо просто задаёт по умолчанию ноль, либо считает уместным пометить всю исходящую из системы связь как «малую задержку», «высокую надёжность» или иное сочетание битов.[5]

Это даёт преимущество: зная значения по умолчанию конкретных систем, мы снова сужаем число возможных ОС отправителя. Однако путаницу усиливают некоторые непослушные операторы DSL и другие интернет-провайдеры, меняющие поле во всём исходящем трафике. Они надеются, что какой-нибудь удалённый маршрутизатор на другом конце света попадётся на уловку, поверит, будто их трафик, помеченный «высоким приоритетом», заслуживает ускоренной обработки, и поставит его выше других соединений, тем самым ускорив просмотр страниц клиентами провайдера. Что сомнительно.

Как и у операционных систем, выбор параметров Type of Service провайдером довольно произволен. Например, один шведский поставщик применяет уникальное и любопытное сочетание битов приоритета со значением 3 и биты типа обслуживания «высокая пропускная способность». Благодаря этому трафик конкретного провайдера легко обнаружить по уникальному выбору битов, не прибегая к активному анализу вроде запросов реестра WHOIS для исходного IP.

Ненулевые неиспользуемые поля и поля Must Be Zero, уровни IP и TCP

Спецификации IP и TCP требуют зарезервировать несколько полей для будущего применения. Все современные системы должны обнулять их, чтобы позднее ненулевым значениям в этих позициях пакета можно было придать особый смысл.

Само собой, некоторые реализации перед отправкой их не обнуляют. На этапе контроля качества проблема вряд ли будет обнаружена, потому что заметных последствий не вызывает: другие системы из осторожности не отклоняют пакет лишь из-за помехи. Поэтому недостаток способен сохраняться целую вечность, возможно, пока биты действительно не начнут использоваться расширением TCP и общение со сломанными системами не потерпит эффектный крах. И снова возможность изучать значения служит ценным источником сведений для более точного

определения ОС отправителя.

Исходный порт, уровень TCP

Исходный порт идентифицирует участника соединения на стороне отправителя. У каждой системы собственные правила назначения так называемых эфемерных, исходящих портов для внешних соединений, и по наблюдаемому номеру часто удаётся определить исходную ОС. Более того, системы маскарадинга обычно применяют для этой цели довольно определённый диапазон. Маскарадинг, или преобразование сетевых адресов «многие к одному», переписывает исходящий из частной сети трафик так, чтобы все соединения казались исходящими от маскирующей системы, а полученные ответы преобразуются обратно и доставляются настоящему отправителю.

Маскарадинг широко используется корпоративными и домашними сетями для экономии адресного пространства. Внутренняя сеть может пользоваться большим пулом адресов, которые, строго говоря, ей не назначены и не маршрутизируются туда или куда-либо ещё из Интернета. Тем не менее системы с такими адресами выходят в Интернет, передавая исходящие соединения через машину-посредника, которая от имени инициатора обращается к удалённой системе со своего законного общедоступного адреса. Подход также защищает внутренние системы: посторонний не способен начать прямое незапрошенное соединение с ними, тогда как участники внутренней сети могут обращаться наружу.

Изучение диапазона исходных портов другой стороны позволяет точнее угадать ОС отправителя и, после сопоставления диапазона с другими наблюдениями, определить, находится ли он в частной сети с преобразованием адресов. В таком случае ожидаемый для системы и фактически наблюдаемый диапазоны исходных портов, скорее всего, не совпадут. Если сеть отправителя применяет преобразование адресов, можно также сделать выводы о типе соответствующего устройства, поскольку разные продукты используют разные диапазоны.

Размер окна, уровень TCP

Напомним: размер окна определяет объём данных, отправляемый без подтверждения. Конкретное значение часто выбирается по личным правилам вуду и иным религиозным убеждениям разработчика. Два самых популярных подхода гласят, что оно должно быть либо кратным MTU за вычетом заголовков протокола, значению Maximum Segment Size, MSS, либо просто достаточно большим и «круглым». Старые версии Linux 2.0 использовали степени двойки, например 16 384. Linux 2.2 перешёл к кратному MSS, по какой-то причине 11 или 22 MSS, а новые версии Linux обычно применяют 2-4 MSS. Сетевая игровая консоль Sega Dreamcast использует 4 096, а Windows часто - 64 512.

Иногда приложение может изменить заданный ОС размер окна ради повышения производительности, но делает это редко. Значение, не совпадающее с ожидаемым системным значением по умолчанию, хорошо обнаруживает конкретное приложение; один из немногих примеров - умеренно популярный браузер Opera.

Указатель срочности и номер подтверждения, уровень TCP

Значения в полях указателя срочности, 16 битов, и номера подтверждения, 32 бита, используются только при установленном соответствующем флаге TCP, URG или ACK. Если флагов нет, значения следует обнулить, но часто этого не происходит. Некоторые системы просто инициализируют их

ненулевым значением, что не создаёт настоящей проблемы: без нужного флага оно не интерпретируется, а лишь помогает опознать систему.

Однако иногда значения вообще не инициализируются и просто копируются из того, что в данный момент находится в буфере построения TCP-пакета. Работая над пассивным снятием отпечатков ОС, я наблюдал это в реализациях стеков Windows 2000 и XP: когда одновременно существовали два сеанса TCP, значения выпускали в текущий сеанс часть сведений из предыдущего, к чему мы вернёмся в главе 11. Так можно узнать, что человек занимается чем-то ещё в фоновом режиме, и получить часть информации, переданной другой стороне. Аллилуйя!

Порядок и настройки параметров, уровень TCP

Точный порядок и выбор параметров пакета уникален для каждой системы. Правил упорядочения нет, поэтому возникают характерные комбинации. Например, Windows применяет в SYN-пакетах последовательность «MSS, NOP, NOP, Selective ACK Permitted», а Linux обычно придерживается «MSS, Selective ACK Permitted, Timestamp, NOP, Window scale». Естественно, это снова превосходный способ различать системы.

Масштаб окна, параметр уровня TCP

Коэффициент масштабирования размера окна обычно равен нулю. Однако некоторые системы по умолчанию задают большее значение или постоянно увеличивают его для определённого трафика, когда считают это разумным, например если пользователь только что загрузил пиратский фильм из P2P-сети или завершил обширную загрузку иного рода. Последнее, разумеется, чуть менее вероятно.

Maximum Segment Size, параметр уровня TCP

В одних системах поле зафиксировано, в других указывает тип непосредственного сетевого подключения устройства. У разных видов сетей разные MTU, что позволяет определить, использует ли человек высокоскоростной DSL или жалкую модемную линию.

Данные метки времени, параметр уровня TCP

Поскольку значение часто соответствует времени непрерывной работы системы, его нередко можно определить наблюдением за параметром метки времени. Более того, в наборе операционных систем их можно различать и отслеживать по изменениям меток входящего трафика: у разных систем разное время работы и крайне маловероятно одинаковое время загрузки, тогда как один компьютер сохраняет непрерывно возрастающее значение.

Это особенно полезно в двух случаях. Первый - когда за одним IP действует несколько систем, как при маскарадинге. Любопытный веб-мастер способен установить, сколько уникальных пользователей корпорации X посетили страницу и где каждый посетитель находился на управляемых им сайтах, даже если все запросы идут с одного адреса и поначалу кажутся неразличимыми.

Другое применение - отслеживание одного пользователя, который по какой-либо причине перескакивает между IP-адресами. Зачем ему это и почему другая сторона захочет определить

такое поведение? Например, он может переключаться между пулом динамических IP коммутируемой линии, разрывая соединение и подключаясь заново, надеясь представить попытки атаки набором бессмысленных, не связанных действий вместо тщательно спланированного обширного зондирования. Либо он хочет обойти ограничения взаимодействия на веб-форуме, в сетевом опросе или конкурсе голосования, по старинке набивая урну, и так далее. Всё это распространённые развлечения нового поколения.

Измерение времени параметром обычно точно, поскольку основано на часах, чаще всего тикающих с частотой 100 или 1 000 Гц, хотя некоторые системы используют 64 или 1 024 Гц и промежуточные значения. Точности хватает, чтобы отличить даже похожие машины, почти одновременно загрузившиеся после сбоя питания, поэтому она чрезвычайно высока.

Другие направления пассивного снятия отпечатков

В этой главе мы рассмотрели самые распространённые показатели определения ОС удалённого хоста и отслеживания его пользователей без их ведома. Но для тех же и более широких целей можно применять множество увлекательных, менее исследованных сторон связи.

Например, один интересный вариант снятия отпечатков связан не с самими пакетами, а с измерением времени и частоты ответов для отдельных сообщений ICMP, повторных передач TCP и подобных функций. Значения всех тайм-аутов и числа повторов позволяют точно и однозначно опознать систему. Проект CRONOS, основанный на исследованиях Франка Вейссе, Оливье Курте и Оливье Хина из Intranode Research Team, создаёт средство активного снятия отпечатков по такому набору показателей, но пассивные применения столь же заманчивы.

Ещё одно многообещающее направление - сочетание и измерение других аномалий или необычных настроек: использования отправителем определённых меток времени, номеров последовательности, совпадающих с номерами подтверждения, необычных флагов, полезной нагрузки в управляющих пакетах, параметра EOL и так далее. Они также помогают различать ОС, хотя часто свойственны небольшому набору реализаций. Алгоритм выбора начальных номеров последовательности нередко служит ценным источником сведений, как станет видно в следующей главе.

Пассивное снятие отпечатков на практике

Эти показатели позволяют точно определять ОС и её конфигурацию, а также параметры сети, эффективно и незаметно отслеживая пользователей. Хотя это может казаться невероятным, созданный мной инструмент p0f реализует большинство приёмов и полностью пассивно собирает и анализирует сведения по пакетам SYN, SYN+ACK и RST с высокой долей успеха.

Рассмотрим пример пакета, чтобы увидеть эффективность подхода. Ниже приведён набор важных параметров, извлечённых из настоящего TCP-пакета, захваченного в сети. Что они сообщают об ОС отправителя?

```
Internet Protocol (версия 4)
  Исходный хост             nimue (10.3.0.1)
  Целевой хост              nightside (10.3.0.3)
  Флаги                     DF
  Время жизни               57
  Идентификационный номер   4428
  Параметров IP нет (размер пакета = 20)

Transmission Control Protocol
  Исходный порт             3803
  Целевой порт              80 (HTTP)
```

```
   Флаги                      SYN
   Номер последовательности   1418000073
   Номер подтверждения        0
   Размер окна                32120

 Параметры TCP
   1. Maximum Segment Size    1460
   2. Selective ACK Permitted
   3. Timestamp               170330930
   4. Window scale            0
```

Очень многое. Вот что можно вывести из наблюдений:

• Поскольку в IP-заголовке установлен DF, система должна применять обнаружение MTU пути. Подходящие системы - новые версии Linux, FreeBSD, OpenBSD, Solaris и Windows. Можно исключить IRIX, AIX, многие коммерческие межсетевые экраны[6] и другие системы, не реализующие PMTUD по соображениям надёжности.

• Время жизни пакета равно 57. Мы знаем, что начальный TTL не мог быть ниже, поскольку по пути он способен только уменьшаться.

Маловероятно, что значение превышало 87, ведь тогда система находилась бы действительно далеко. Это согласуется со многими Unix, использующими начальный TTL 64, но исключает Windows с 128, версии Solaris до 8 с 255 и несколько сетевых устройств с 32.

• Идентификационный номер пакета ненулевой. Это исключает Linux 2.4 и новее, а также несколько недавних выпусков других популярных ОС.

• Исходный порт находится в самом распространённом диапазоне от 1 024 до 4 095. Само по себе это не исключает системы, но можно уверенно предположить, что до данного соединения машина установила более 2 700 других и вряд ли находится за маскарадингом.

• Выбор и порядок параметров, MSS, Selective ACK, Timestamp, Window scaling, характерны для Linux 2.2 и новее.

• Размер окна кратен MSS, а именно `MSS*22`. Единственная подходящая система - Linux 2.2.

• В пакете не наблюдается аномалий, нарушений RFC или иных странностей, что подтверждает гипотезу о Linux.

• Maximum Segment Size указывает соединение Ethernet или PPP через модем с MTU 1 500.

• Время непрерывной работы системы составляет приблизительно 19 дней, а находится она на расстоянии 7 систем.

Разумеется, отдельные показатели могут меняться приложениями или пользовательскими настройками. Например, пользователи склонны изменять TTL либо включать и выключать настройки, прочитав руководства по оптимизации сети или запустив приложения «системного доктора». Но цепочка выводов из наблюдений даёт надёжный способ определить ОС машины, выбрав систему, которая лучше всего совпадает по большинству категорий.

В данном случае есть веские основания считать, что перед нами Linux 2.2, а отправитель подключён к Интернету через Ethernet или коммутируемый модем. Из этого также следует, что система находится в 7 переходах, `64-57`, где 64 - начальный TTL Linux, и работает около 20 дней. Если за IP скрываются несколько пользователей, их легко подсчитать и различить сеансы по характеристикам систем и данным меток времени, если они доступны.

Исследование применений пассивного снятия отпечатков

Наблюдаемый получателем либо сторонним участником, например провайдером между отправителем и получателем, сетевой трафик способен сообщать не только фактически переданные данные, но и некоторые параметры системы отправителя. Как уже говорилось, уязвимость важна и интересна, потому что, в отличие от данных приложений, она не обязательно очевидна и часто не поддаётся контролю пользователя. Пользователь может изменить настройки браузера и других приложений, пытаясь предотвратить наблюдение, идентификацию и отслеживание, но раскрытие на нижнем уровне IP или TCP легко подорвёт усилия, сообщив наблюдателю столько же, сколько жертва пытается скрыть. Оно способно нести и данные более фундаментального значения для безопасности инфраструктуры, включая полезные подсказки об устройстве и защите сети жертвы.

При этом, помимо вторжения в приватность, пассивное снятие отпечатков полезно для вполне законной разведки. Набор практических и широко применяемых способов занимает весь этический спектр от злого умысла до правомерной защиты.

Сбор статистики и журналирование происшествий

Одно из наиболее законных применений - наблюдение за сетью для ненавязчивого и объективного анализа используемых платформ и сетевых сред, чтобы пользователи получали службу, оптимизированную для их программ, а ни одна крупная группа не была каким-либо образом обделена. Пассивное снятие отпечатков также значительно улучшает сбор данных о потенциальных атакующих и иной несанкционированной деятельности. Действительно, оно особенно популярно в исследованиях honeypot.

> **Примечание.** Концепцию honeypot активно продвигает и исследует Лэнс Спитцнер из Sun Microsystems.[11] Цель - позволить владельцу узнавать о противниках и их задачах посредством устройств-приманок, ценность которых заключается в незаконном использовании и которые не имеют настоящего значения для инфраструктуры, хотя выглядят так, словно имеют.

Оптимизация содержимого

Одно активное применение основано на предоставлении службы, оптимизированной для конкретного получателя по немедленному анализу конфигурации, с которой он обращается к серверу. Считаю своим долгом бесстыдно прорекламировать уже упомянутый p0f. Он позволяет другим приложениям запрашивать параметры недавних входящих соединений, значительно облегчая оптимизацию содержимого. Веб-сценарию не нужно много знать о TCP и IP: он просто спрашивает p0f: «Эй, кто этот парень, с которым я разговариваю?» - и получает полезный ответ.

Обеспечение политик

Обнаружение и возможная блокировка устаревших или несоответствующих систем, например устройств, нарушающих корпоративную политику или создающих риск безопасности, и нашествия несанкционированных сетевых подключений - другое интересное применение. Начиная с версии 3.4 OpenBSD позволяет маршрутизировать и перенаправлять трафик по результатам определения ОС, делая обеспечение политик на основе свойств удалённой ОС вполне жизнеспособным. Та же функция теперь входит в код Linux netfilter patch-o-matic. Обе реализации во многом вдохновлены p0f или основаны на нём.

Безопасность для бедных

Пассивное снятие отпечатков также может уменьшить некоторые виды уязвимости. Хотя при определённых усилиях приём можно обмануть, он способен не допускать к некоторым нижележащим сетевым службам определённые виды клиентов, например Windows - платформу, чаще других заражённую шпионскими программами, бэкдорами и червями и нередко используемую для незапрошенной массовой рассылки или как промежуточный узел атаки, - одновременно разрешая доступ «менее подозрительным» сущностям.

Испытания безопасности и оценка перед атакой

Межсетевые экраны и другие решения, тщательно фильтрующие и анализирующие IP-трафик, часто останавливают активное снятие отпечатков. Но пассивный способ способен изучать даже агрессивно защищённые системы и строить карты сетей без единого сигнала тревоги.

У подхода к испытанию и оценке безопасности две стороны. Во-первых, можно анализировать входящий трафик. Хотя наблюдателю приходится ждать подключения удалённой стороны к его системам, такое соединение довольно легко вызвать, не возбуждая подозрений. Часто достаточно послать жертве за самым совершенным фильтром пакетов определённое письмо или ссылку на веб-сайт. Во-вторых, можно анализировать ответы на законный трафик доступной службы, определяя параметры удалённой стороны. Если хакер в чёрной шляпе знает, как скомпрометировать внутреннюю сеть, но хочет больше узнать о её устройстве и уменьшить риск преждевременного обнаружения, пассивное снятие отпечатков весьма полезно. То же верно для законного испытания безопасности, за которое платит проверяемая организация.

Профилирование клиентов и вторжение в приватность

Многие компании идут на большие усилия, собирая и продавая ценные сведения о привычках, предпочтениях и поведении людей. Обычно информация применяется в маркетинге, но теоретически может быть использована против конкретного человека. Способность отслеживать пользователей, сопоставляя результаты снятия отпечатков из нескольких посещённых мест, будь то картирование внутренних сетей и программ, наблюдение за людьми или сбор другой ценной статистики, способна дать сведения, которые либо сами имеют значительную ценность, либо повышают привлекательность других не вполне этичных предложений.

Шпионаж и скрытая разведка

Возможность собирать дополнительные сведения о сетевой архитектуре конкурента, поведении и предпочтениях пользователей часто очень соблазнительна. Хотя это звучит как плохая научная фантастика, речь всего лишь о более целевом варианте описанного профилирования.

Предотвращение снятия отпечатков

Из-за сложности обычного стека IP предотвратить снятие отпечатков вообще чрезвычайно трудно. Но можно устранить отдельные проблемы и отключить известные средства, определив, на какой

параметр они опираются больше всего, и изменив его. Например, некоторые решения фильтрации пакетов, такие как `pf` в OpenBSD, предоставляют нормализацию, гарантирующую одинаковый «вид» всего исходящего трафика. Это способно в некоторой степени помешать отдельным сторонам снятия отпечатков или лишь осложнить его, снизив точность популярных программ, но полностью проблему не решает.

Даже тщательное и кажущееся исчерпывающим ручное или автоматическое изменение отдельных настроек ОС и TCP способно лишь затруднить идентификацию: некоторые варианты поведения скрыты глубоко в ядре и не настраиваются. Например, довольно трудно поменять порядок параметров пакета. Более того, ручные изменения рискуют внести в исходящие из системы пакеты уникальные признаки, ещё сильнее повредив приватности и анонимности.

К счастью, некоторые решения противодействуют определённым испытаниям. Например, IP Personality Гаэля Руаллана и Жан-Марка Саффруа изменяет стек TCP, чтобы для конкретных средств он казался исходящим от другой ОС. При желании можно заставить NMAP считать систему лазерным принтером Hewlett-Packard. Но возникают проблемы. Во-первых, легко ослабить стек TCP, пытаясь изобразить устройство, чей стек изначально слаб. Если ради соответствия свойствам принтера использовать простейшие номера последовательности во всех соединениях, рано или поздно кто-нибудь воспользуется этим и легко нарушит или изменит трафик. Кроме того, программы вроде IP Personality действуют лишь против самых популярных, известных и хорошо документированных средств, не гарантируя успеха против остальных: каждый инструмент изучает свои признаки и толкует их по-разному. Можно надеяться обмануть только наименее решительных и самых наивных «массовых» атакующих, использующих известные вам средства.

> **Примечание.** В отличие от агентов маскарадинга, межсетевые экраны типа прокси и другие прокси-устройства не пересылают пакеты, а перехватывают соединения и начинают новые посредством собственного стека IP. Это единственное полное решение против снятия отпечатков третьего и четвёртого уровней OSI, однако оно серьёзно снижает производительность и чаще вызывает проблемы из-за резко возросшей сложности. Кроме того, всё равно остаётся возможным снятие отпечатка самого приложения на верхнем уровне.

Пища для размышлений: фатальный недостаток фрагментации IP

Обсуждая определяющие свойства Internet Protocol, я мимоходом упомянул, что процесс фрагментации и сборки пакетов имеет фатальный недостаток. Мысль возникла главным образом из довольно интересного наблюдения во время написания этой книги. Хотя концепция связана с активной и заметной атакой откровенно злонамеренной сущности, которую, впрочем, нелегко отследить, это уникальный и любопытный изъян, присущий конструкции Internet Protocol. Он не является результатом чётко определённой ошибки, а скорее столкновением парадигм разных уровней проектирования, обе из которых, что любопытно, были заданы Джоном Постелом, одним из отцов семейства IP. Я решил завершить им главу как пищей для размышлений для тех, кого интересует патология компьютерных недостатков.

Сначала посмотрим на сегодняшнее, а может, вчерашнее положение дел, поскольку мы стряхиваем пыль с довольно старого приёма атаки, упомянутого при обсуждении TCP. Слепая подделка была впервые описана Робертом Т. Моррисом в середине 1980-х.[12] Её золотой век наступил десятилетием позже, но с тех пор значение уменьшилось. Мы сосредоточимся на конкретном варианте: внедрении данных в существующий сеанс, чтобы нарушить его, убедить сервер, будто пользователь отдал определённую команду, или убедить пользователя, что он получает конкретный ответ сервера. Приём часто называется захватом соединения.

В обычных условиях злонамеренный наблюдатель, желающий вставить данные в существующий поток TCP, сначала должен определить номера последовательности хотя бы одной стороны. Хотя атака чрезвычайно чувствительна ко времени и должна быть направлена на конкретное действующее соединение, при предсказуемых номерах её можно успешно провести, что много раз и случалось. В конце 1990-х множество средств нарушало сеансы TCP Windows с сетями Internet Relay Chat, IRC, ради забавы и не только, эксплуатируя слабый алгоритм выбора начального номера последовательности, ISN, в Windows. Было элементарно то здесь, то там внедрить один пакет RST, выбросив человека с сервера чата. Тогда мы называли это развлечением.

Сегодня ситуация несколько иная. Благодаря усилиям многих исследователей, включая скромнейшего автора этих слов, разработчики усердно затрудняли предсказание начальных номеров TCP. Многочисленные попытки повысить качество и стойкость генераторов популярных ОС в итоге осложнили атаки предсказания ISN, за несколькими довольно незначительными исключениями. Системы с последовательными ISN почти вымерли; не способный определить номера разговора другой стороны атакующий для точного внедрения данных вынужден перебрать всё 32-битное пространство, меньше - если требуется только прервать или безвозвратно испортить сеанс. Это около 4 294 967 296 комбинаций, а для успеха атаки в среднем нужно отправить примерно 80 ГБ данных. Само собой, это не считается особенно осуществимым.

Но фактические выгоды успешного внедрения данных почти не изменились. Хотя всё больше связи идёт по каналам с шифрованием, актуальность атаки существенно не снизилась; плодотворных сценариев предостаточно. Вот несколько примеров.

• Данные можно внедрить в незашифрованный трафик между серверами или маршрутизаторами: обмен электронной почтой, передачу зон DNS, связь BGP и так далее. Значительную часть межсерверного трафика может породить атакующий, а содержать он всё равно будет конфиденциальные или доверенные сведения, что делает целевую и рассчитанную по времени атаку осуществимее.

• Данные можно внедрить в незашифрованный трафик между клиентом и сервером, например загрузку файлов File Transfer Protocol, FTP, или ответы HyperText Transfer Protocol, HTTP. Так посетителю заметного сервера можно предоставить вредоносное, компрометирующее или унизительное содержимое либо создать впечатление, будто попытка компрометации исходит от невинного посетителя.

• Данные можно внедрить в существующий сеанс, чтобы эксплуатировать уязвимость службы на этапе, недоступном не прошедшему проверку пользователю. Это относится и к зашифрованному, и к открытому трафику. Например, служба POP3, Post Office Protocol Version 3, протокол удалённого доступа к почтовому ящику, принимает некоторые команды только после успешного входа. До него доступны лишь директивы процесса аутентификации, `USER` и `PASS`. Без верного пароля атакующий не может эксплуатировать недостаток более поздней команды, например `RETR`, получающей определённое сообщение из ящика. Но если ему удастся внедрить вредоносный запрос `RETR` в существующий сеанс уже аутентифицированного пользователя, он победил.

• Даже защищённый, зашифрованный и снабжённый контролем целостности поток подвержен отказу в обслуживании, если один тщательно сформированный пакет нарушит или завершит сеанс.

Поэтому возможность без больших усилий внедрить данные, не перебирая всё пространство номеров последовательности, весьма привлекательна. И здесь очень кстати оказывается фрагментация.

Разбиение TCP на фрагменты

Когда IP-пакет с полезной нагрузкой TCP фрагментируется, что довольно часто происходит при передаче файлов и не всегда предотвращается простой установкой DF, как делают некоторые системы, данные путешествуют по сети несколькими частями и собираются только у получателя. Умный атакующий, ожидая фрагментацию, может послать специально сформированный незаконный IP-фрагмент, замаскированный под фрагмент ожидаемого отправителя. Получив его, адресат при некотором везении, то есть точном выборе времени, использует подделку вместо настоящего фрагмента при сборке исходного пакета.

В этом сценарии первый фрагмент, содержащий полный заголовок TCP с точными портами, номерами последовательности и прочим, объединяется с вредоносной нагрузкой атакующего. Поэтому для внедрения данных в кадр ему не нужно знать номера последовательности и другие параметры сеанса, что фактически подрывает все усилия по генерации ISN. Итоговый пакет, обрабатываемый получателем, состоит из верного заголовка, скопированного из законного фрагмента, и внедрённой атакующим вредоносной нагрузки.

> **Примечание.** Атакующий может заменить любую часть нагрузки первого фрагмента, задав небольшое перекрытие фрагментов. Многие системы признают перекрытия и перезаписывают ранее полученные данные новой копией. В крайнем случае можно успешно заменить все данные TCP-пакета, кроме номера последовательности.

Разумеется, некоторые части головоломки всё ещё отсутствуют. Но помимо точного выбора времени и знания момента передачи[7] в этом сценарии атакующему нужно преодолеть только два препятствия:

• Фрагмент должен иметь правильный IP ID, чтобы быть включённым в сборку. К счастью, во многих системах идентификаторы выбираются последовательно, и вероятное текущее значение можно вывести, просто выполнив пробное соединение. Некоторые системы, прежде всего OpenBSD, FreeBSD и Solaris, предлагают случайные ID, что осложняет, но не предотвращает атаку. Нужно проверить тысячи, а не миллиарды комбинаций, поскольку поле IP ID довольно мало, всего два байта.

• Заголовок TCP содержит контрольную сумму, проверяемую после сборки, и сумма изменённых атакующим данных должна совпасть с исходной. Но устройство контрольной суммы TCP элементарно, это лишь разновидность прямой 16-битной суммы, поэтому можно сформировать нагрузку, не меняющую итог, если исходная заменяемая область известна. Чаще всего так и есть, особенно при передаче файлов, когда атакующий хочет внедрить вредоносный код или содержимое в общедоступную порцию данных.

Это иллюстрирует следующая упрощённая контрольная сумма пакета из слов заголовка `H1`, `H2` и слов нагрузки `P1`, `P2`, `P3`:

```
C = H1 + H2 ... + P1 + P2 + P3 ...
```

`H1`, `H2` и `C` атакующему неизвестны. Заголовки содержат номера последовательности, а эти данные влияют на контрольную сумму. Он не может изучить пакет, но знает, что жертва выполняет определённую, предсказуемую транзакцию прикладного уровня, например проверяет почту, загружает веб-страницу или разговаривает с друзьями. Атакующий способен вывести нагрузку `P1`, `P2`, `P3` и хочет заменить её своими вредоносными словами `N1`, `N2`, используя третье слово для компенсации суммы, `CC`, чтобы пакет оставался действительным.

```
C = H1 + H2 ... + N1 + N2 + CC ...
```

Решив уравнения относительно `CC`, получаем необходимую компенсацию:

```
CC = P1 + P2 + P3 - N1 - N2
```

После этого атакующий может изменить пакет так, чтобы контрольная сумма осталась прежней, не зная пакет целиком. Нужен лишь заменяемый фрагмент: его достаточно, чтобы правильно вычислить компенсацию и сохранить сумму.

Сноски

[1] [назад] Строго говоря, хотя для обсуждения это не имеет особого значения, контрольная сумма IP основана на 16-битном дополнении до единицы суммы дополнений до единицы проверяемых данных.
[2] [назад] Строго говоря, 65 536, однако порт номер 0 использовать не следует. Разумеется, операционная система и её приложения могут это разрешать, нарушая стандарт.
[3] [назад] Кевин Митник, один из самых известных и спорных хакеров в чёрной шляпе, скомпрометировал компьютер Цутому Симомуры, выдав себя посредством подделки TCP за одну из доверенных рабочих станций. Поступок сильно расстроил господина Цутому и, судя по большинству рассказов, в долгосрочной перспективе не особенно помог Кевину.
[4] [назад] «Требуется» в смысле «необходимо для правильной обработки». Некоторые процессоры значительно теряют производительность при доступе к многобайтовым структурам, не выровненным по 32 или 64 битам; другие прямо требуют такого выравнивания, иначе вызывают фатальное исключение, ловушку выполнения, и отказываются исполнять операцию. Разумеется, непослушный отправитель может намеренно поместить невыровненные данные в буфер, надеясь, что система получателя сгорит при получении пакета. Вменяемая операционная система сначала проверит данные или скопирует параметр в правильно выровненную область перед обработкой. Но вменяемость системы нельзя принимать на веру.
[5] [назад] Некоторые разработчики даже устанавливают бит Must Be Zero, который в законном применении не должен быть установлен никогда, предположительно просто как заявление о стиле.
[6] [назад] Межсетевой экран по существу является фильтрующим маршрутизатором, который часто также способен понимать характеристики трафика верхних уровней и принимать по ним решения.
[7] [назад] Сам выбор времени не так сложен, как кажется. Атакующий может послать вредоносный второй фрагмент чуть заранее; получатель создаст буфер сборки и определённое время будет ждать остальные части. Когда придёт первый законный фрагмент, содержимое буфера признаётся полностью восстановленным без ожидания настоящей второй части.

Глава 10. Продвинутые стратегии подсчёта овец

Где мы разбираем древнее искусство определения сетевой архитектуры и местонахождения компьютеров

Сетевая разведка и картирование - искусство эксплуатации набора направлений раскрытия информации, присущих основным протоколам связи Интернета, чтобы распознавать системы и сети либо идентифицировать и отслеживать потенциальных нарушителей, пользователей, клиентов или конкурентов. Возможно, это самое развитое, распространённое, значимое и непосредственно полезное на сегодняшний день применение пассивного анализа данных, однако у него есть свои проблемы, влияющие на точность и пригодность в некоторых сценариях. Особенно это верно для известных и испытанных способов пассивного снятия отпечатков TCP/IP.

Преимущества и недостатки традиционного пассивного снятия отпечатков

Рассмотренные в предыдущей главе показатели позволяют легко определить некоторые характеристики исходной системы и сети. В отдельных случаях приёмы также дают отслеживать людей при смене адреса или совместном использовании одного адреса с другими участниками сети. Их можно применять без взаимодействия с удалённой стороной, если удаётся убедить наблюдаемого землянина обратиться к определённой сети либо пока его связь проходит через конкретный набор систем под управлением достаточно любопытного человека. Поэтому пассивное снятие отпечатков, помимо прочего, позволяет владельцу сервера или определённому провайдеру довольно легко собирать огромный объём совершенно незаметной информации.

Для удалённой стороны пассивное снятие отпечатков становится обоюдоострым мечом. Его можно развернуть ради полезных данных о внутренней структуре сети, упрощающих атаку или

раскрывающих применяемые технологии даже в довольно сложной среде, как на рисунке 10-1. Но его также можно совершенно правомерно использовать для наблюдения за собственной сетью в поисках нарушений политики, например незаконных соединений или точек доступа между внутренней сетью и внешним миром, либо для отслеживания атакующих.

Потеря приватности отдельного пользователя обычно ничтожна, если только особой проблемой не является возможность связать случайные действия пользователя с дополнительными данными отпечатка или отслеживать одного человека между доменами. Вероятно, это верно лишь тогда, когда само его поведение изначально сомнительно. Но совокупная потеря приватности всех пользователей может серьёзно беспокоить, а собранные сведения и основанное на них отслеживание имеют заметную рыночную ценность. Например, рекламодателям личные данные можно продать гораздо дороже, если дополнить их предпочтениями и интересами. Раскрытие технического внутреннего устройства сети также действительно нежелательно для корпораций и других частей чувствительной инфраструктуры.

Однако ещё не всё потеряно. Как отмечалось ранее, точному применению пассивного снятия отпечатков мешают некоторые проблемы. Надёжность традиционного определения ОС страдает из-за лёгкости обмана наблюдателя посредством тщательной настройки некоторых или всех сетевых параметров наблюдаемой системы. Даже если полностью изменить все настройки непросто, частичного изменения может хватить, чтобы сорвать отдельные попытки автоматического анализа - ура! - или ввести в заблуждение исследователя злонамеренного происшествия - упс. Проблема не массовая и потому не беспокоит статистический анализ, но в отдельных случаях вызывает опасения.

Кроме того, возможности отслеживания и подсчёта пользователей в мучительно разобранном в главе 9 подходе почти полностью зависят от таких параметров, как метки времени пакетов TCP/IP. Все прочие признаки либо стандартизованы, либо имеют слишком мало вариантов для однозначной положительной идентификации одного компьютера, кроме самых необычных случаев. Если данные недоступны, потому что конкретное расширение производительности отключено, как в большинстве систем Windows, точная идентификация невозможна.

Это снижает потенциальную ценность сведений и для членов чрезмерно усердного злого тайного общества, которое, как всем известно, охотится за нашими драгоценнейшими секретами, и для испытателей безопасности или аналитиков происшествий, специалистов компьютерной криминалистики. Без идентификации по меткам времени бывает невозможно различить несколько одинаковых систем за маскарадингом или опознать человека, чей IP изменился после переподключения модема.

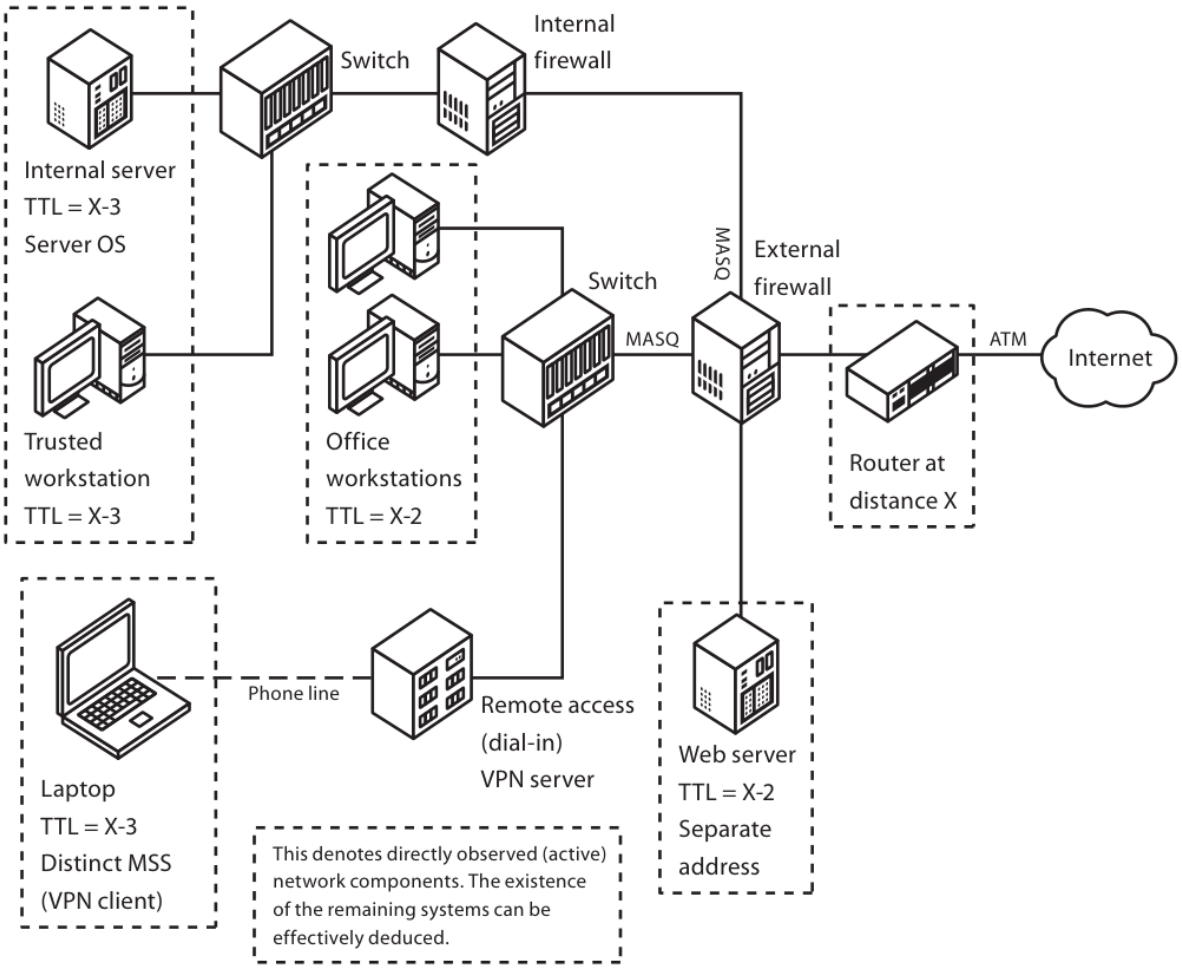


Рисунок 10-1. Пассивное снятие отпечатков позволяет картировать сложную и даже недоступную сеть простым наблюдением за трафиком некоторых узлов. Особенно важны свойства ОС, TTL и MSS пакетов; по различиям наблюдаемых характеристик выводится наличие остальных компонентов. Непосредственно наблюдаемые активные компоненты отмечены отдельно. Читателю предлагается определить, как окончательно построить эту карту, наблюдая лишь внешний трафик.

Однако другой, возможно более интересный, многообещающий и сложный способ легко устраняет недостатки. Новый подход крайне трудно обмануть и почти повсеместно можно применять для отслеживания систем. Что ещё интереснее, он позволяет различать экземпляры совершенно одной системы в совершенно одинаковой конфигурации, выводя обнаружение маскарадинга на новый уровень. Он использует свойства механизма генерации номеров последовательности TCP/IP и вдобавок умеет создавать красивые картинки.

Краткая история номеров последовательности

Напомним из предыдущей главы: начальные номера последовательности используются TCP для целостности сеанса и фактически гарантируют его самую базовую устойчивость безопасности.

Единственный по-настоящему универсальный способ защитить открытый сеанс TCP/IP от внедрения данных, захвата или подделки совершенно посторонним человеком - выбирать начальные номера непредсказуемо для атакующего. Это уменьшает его шанс сделать верное слепое предположение и подделать пакет, который будет принят законной частью чужого сеанса, до практически несущественного уровня, даже если атакующий штурмует систему тысячами

пакетов в надежде, что хотя бы у одного номер приблизительно совпадёт.

В начале 1980-х аспекты безопасности связи TCP не казались достойной беспокойства проблемой: Интернет был довольно малой, замкнутой и, возможно, слегка элитарной средой учёных и им подобных. Поэтому спецификация RFC протокола TCP не задавала требований к выбору начальных номеров, а почти все ранние и некоторые не столь ранние реализации стека TCP/IP применяли простейшие алгоритмы на основе времени или счётчика, выдававшие последовательные числа для следующих соединений. Тогда рандомизация казалась ненужной тратой драгоценной вычислительной мощности. Кроме того, она без необходимости повышала вероятность столкновения номеров. Столкновение возникает, когда два ISN для последовательных соединений с хостом слишком близки и старые, несвоевременно пришедшие пакеты могут быть истолкованы в контексте неправильного соединения. Естественно, в краткосрочной перспективе случайный выбор скорее создаёт столкновения, чем последовательное увеличение.

С 1980-х Интернет, конечно, сильно развился: выросла доступность, база пользователей быстро менялась и увеличивалась, а по проводам передавалось всё больше важных данных, отчего вопросы безопасности стали актуальнее. К сожалению, популярные и надёжные механизмы защиты целостности и приватности до сих пор не догнали расширение Интернета. Не все службы поддерживают шифрование, не все пользователи знают, когда его применять, а главное - большинство не умеет правильно проверять криптографические сертификаты удалённых сторон.

Со временем, особенно после широко распространённого практического злоупотребления слабой генерацией ISN в середине 1990-х, хотя в основном в сетевых чатах и подобных службах, стало ясно, что потокам TCP/IP нужна хотя бы простейшая защита целостности. Это важно даже для малой доли криптографически защищённого трафика: нарушение уровня носителя внедрением мусора или пакетов RST столь же нежелательно, пусть результат ограничен отключением, отказом в обслуживании, а не внедрением данных.

Поскольку единственный способ исправить положение без капитальной перестройки почти всех известных человечеству схем связи TCP состоял в том, чтобы сам протокол было трудно атаковать, многие разработчики начали отходить от устаревших и опасных механизмов ISN с простым увеличением на единицу. Усилия действительно повысили устойчивость соединений к слепой подделке, но одновременно открыли интересные направления сбора сведений для более совершенного снятия отпечатков систем и сетей - ради оценки безопасности или спланированной атаки.

Получение большего из номеров последовательности

Разумеется, важно отличать хорошие реализации генератора ISN от плохих и для контроля качества, и для испытаний безопасности. До недавнего времени качество оценивали либо анализом исходного кода, либо некоторыми одномерными испытаниями битового потока последовательных ISN, оценивая энтропию каждого выходного бита. Первый способ часто сложен, дорог, подвержен ошибкам и не всегда возможен при отсутствии общедоступного исходного кода системы. Второй не позволял надёжно и наглядно выявлять более тонкие зависимости и прочие свойства генератора, сосредоточиваясь скорее на статистических несовершенствах, чем на корреляции значений последовательных соединений. Очевидно, доказать безопасность реализации по одному выходу почти невозможно, но легко проверить некоторые распространённые проблемы и разумную стойкость алгоритма. Однако даже применяемые здесь методы были в лучшем случае довольно слабы.

И первоначальные небезопасные генераторы ISN, и некоторые современные решения основаны на аддитивных итеративных арифметических системах, вычисляющих новые значения из предыдущего выхода; различаются лишь сложность пересчёта и внесённая практическая

непредсказуемость. Единственные безопасные конструкции без традиционной арифметики - некоторые новые решения, использующие сравнительно медленные, но криптографически безопасные сокращающие функции для итеративных систем.

Во всех случаях было бы интересно искать нетривиальную корреляцию между последовательными результатами генератора новых соединений, обнаруживая возможные недостатки алгоритма.

Если видна зависимость между выходом генератора в момент `t` и в `t+x`, атакующий может заранее подключиться перед соединением, которому надеется помешать или которое хочет полностью подделать, чтобы получить выход ISN в `t`. По наблюдаемому номеру он затем определит ответ другой стороны в будущем, `t+x`, и подделает допустимый пакет нового соединения, хотя непосредственно используемый ISN не видит.

С этой мыслью в 2001 году я провёл исследование универсального способа изучения менее очевидных временных зависимостей в последовательностях ISN удалённых систем. Результатом стала работа “Strange Attractors and TCP/IP Sequence Number Analysis”,[1] где некоторые алгоритмы рассматривались подробнее и предлагался способ обнаруживать тонкие связи за пределами самых очевидных известных закономерностей и недостатков. Подход хорошо знаком миру прикладной математики, но для сетей был весьма новым.

Задержанные координаты: фотографирование временных последовательностей

При работе с генератором ISN как чёрным ящиком в современной закрытой системе виден только его выход, последовательность 32-битных значений пакетов TCP/IP, но не алгоритм. Код многих ОС проприетарен и хорошо охраняется, находясь далеко за пределами досягаемости простых смертных. Даже в открытой системе исходники бывают хитрыми и вводящими в заблуждение, заставляя повторить ошибки первоначального разработчика.

Типичный вход для оценки мог бы выглядеть так:

```
S0 = 244782
S1 = 245581
S2 = 246380
S3 = 247176
S4 = 247975
S5 = 248771
...
```

Сразу ли очевидна зависимость этих чисел? И если да, существует ли сравнительно универсальный способ, которым компьютер сможет проследить её и более сложные схемы?

Элегантное решение казалось далёким. Я хотел разработать способ определения универсальных свойств нижележащего алгоритма ISN лишь по его выходу. Но перед этим для облегчения анализа было желательно и удобно предположить: поскольку многие реализации повторяют определённые арифметические операции, лучше наблюдать изменения последовательных результатов, а не абсолютные значения. Для таких алгоритмов это выгодно и почти не повредит остальным возможным генераторам. Для этого вычислим дискретную производную входной последовательности: приращения между элементами `S`. Последовательность дельт `D`, естественно начиная с `t = 1`, задаётся:

```
Dt = St - S(t-1)
```

В примере получается:

```
D1 = 799
D2 = 799
D3 = 796
```

```
D4 = 799
D5 = 796
...
```

Если отбросить сами значения и рассматривать только динамику ISN, нижележащая зависимость становится заметнее и обычно останется такой для всех реализаций с данным типом арифметики. Для систем без простейшей итеративной арифметики преобразование практически не имеет значения и существенно не влияет на качество данных анализа.

> **Примечание.** Особенно педантичный исследователь также компенсировал бы неравномерность времени получения образцов; здесь предполагается, что между измерениями всегда проходит фиксированная единица времени 1. Но при высокоскоростном сборе производительность сети и другие события могут значительно влиять на интервалы. Чтобы различия не затрагивали алгоритмы, использующие часы в генерации ISN, безопаснее применить `Dt = (St - S(t-1))/Tt`, где `Tt` выражает задержку между получением `S(t-1)` и `St`.

Преимущество применительно к итеративным арифметическим системам довольно очевидно. Но за исключением простейших случаев одного метода едва ли достаточно: мы лишь переходим от одной плоской, трудной для анализа последовательности к другой.

Далее я решил преобразовать дельты в форму, которую человек или машина легко исследует на корреляции, менее очевидные, чем в предыдущем примере. Для первой группы целевой аудитории ничто не лучше трёхмерной модели динамики. К сожалению, в ISN хватает сведений лишь для рисунка в одном измерении, на одной оси. Как превратить их в аккуратную трёхмерную фигуру?

Решение - расширить набор данных стратегией реконструкции координат, называемой координатами с временной задержкой. Каждый образец дополняется виртуальными координатами из предыдущих элементов. Считая текущий образец координатой `x`, можно присвоить ему `y` и `z` и получить тройку, достаточную для отображения каждого образца единственной точкой, пикселем, трёхмерного пространства. Приём не ограничен тремя измерениями, но для визуализации и анализа большее число казалось непрактичным. В любом случае большинство людей не слишком хорошо справляется с дополнительными измерениями, по крайней мере в трезвом виде.

Координаты вычисляются так, чтобы вторая соответствовала значению в `t-1`, третья - наблюдению в `t-2` и так далее. В данном применении для времени `t`:

```
xt = Dt     = St     - S(t-1)
yt = D(t-1) = S(t-1) - S(t-2)
zt = D(t-2) = S(t-2) - S(t-3)
```

Последовательность новых троек (`x, y, z`) позволяет изобразить поведение генератора ISN в трёхмерном пространстве. Положение точки зависит от текущего значения и нескольких предыдущих результатов, поэтому даже довольно сложные зависимости создают абстрактные, но заметные неравномерные картины плотности в фазовом пространстве - уникальный портрет алгоритма. В отношении таких портретов термин «аттрактор» означает форму, отображающую динамику системы: множество или пространство представляет «след» состояний, через которые система циклически проходит либо эволюционирует, будучи предоставленной самой себе.

На рисунке 10-2 показан набор данных, первоначально выглядевший так:

```
4293832719
3994503850
4294386178
134819
4294768138
191541
4294445483
4294608504
4288751770
```

88040492
...

Рисунок 10-2. Трёхмерное представление набора данных, описанного в тексте.

На рисунках 10-3 - 10-5 показаны несколько других распространённых, но не обязательно очевидных схем зависимости.

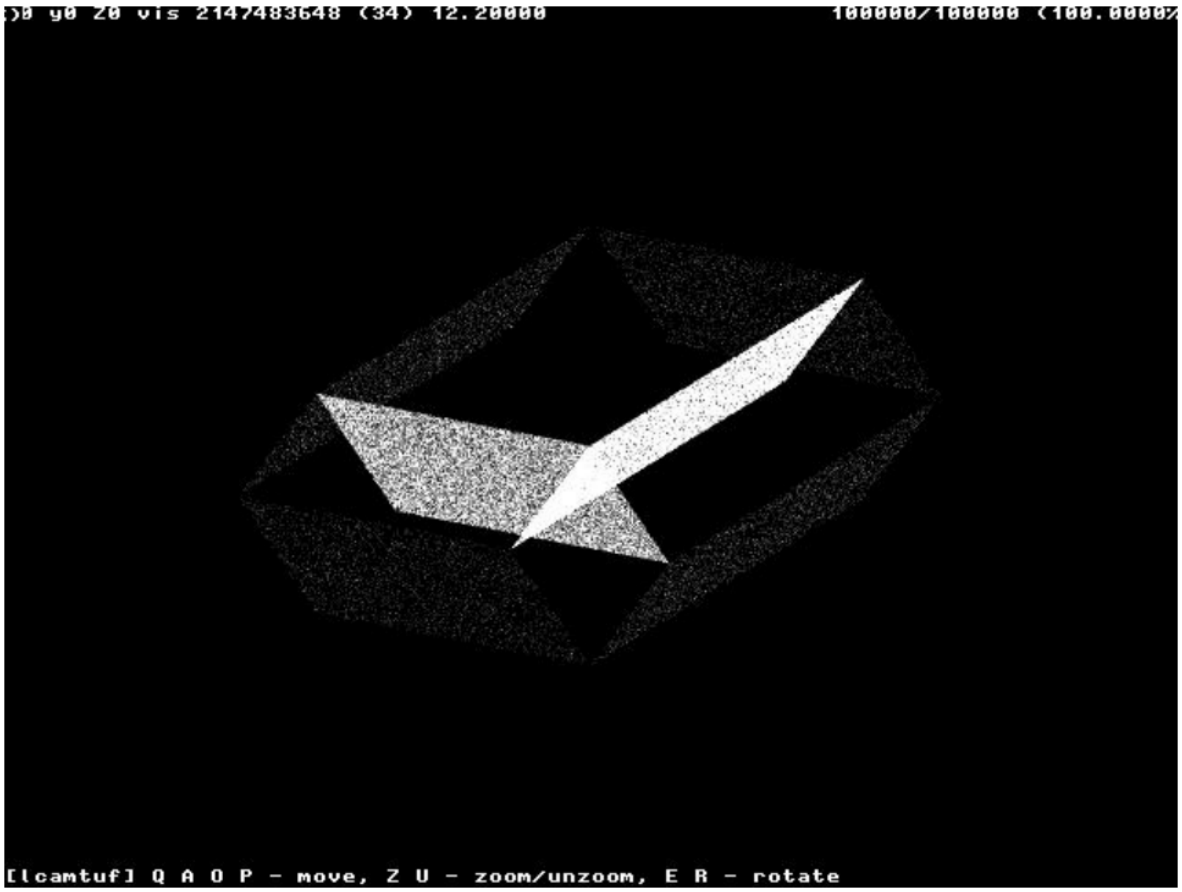


Рисунок 10-3. Трёхмерное представление набора данных, полученного от сложной, но небезопасной функции генератора случайных чисел.

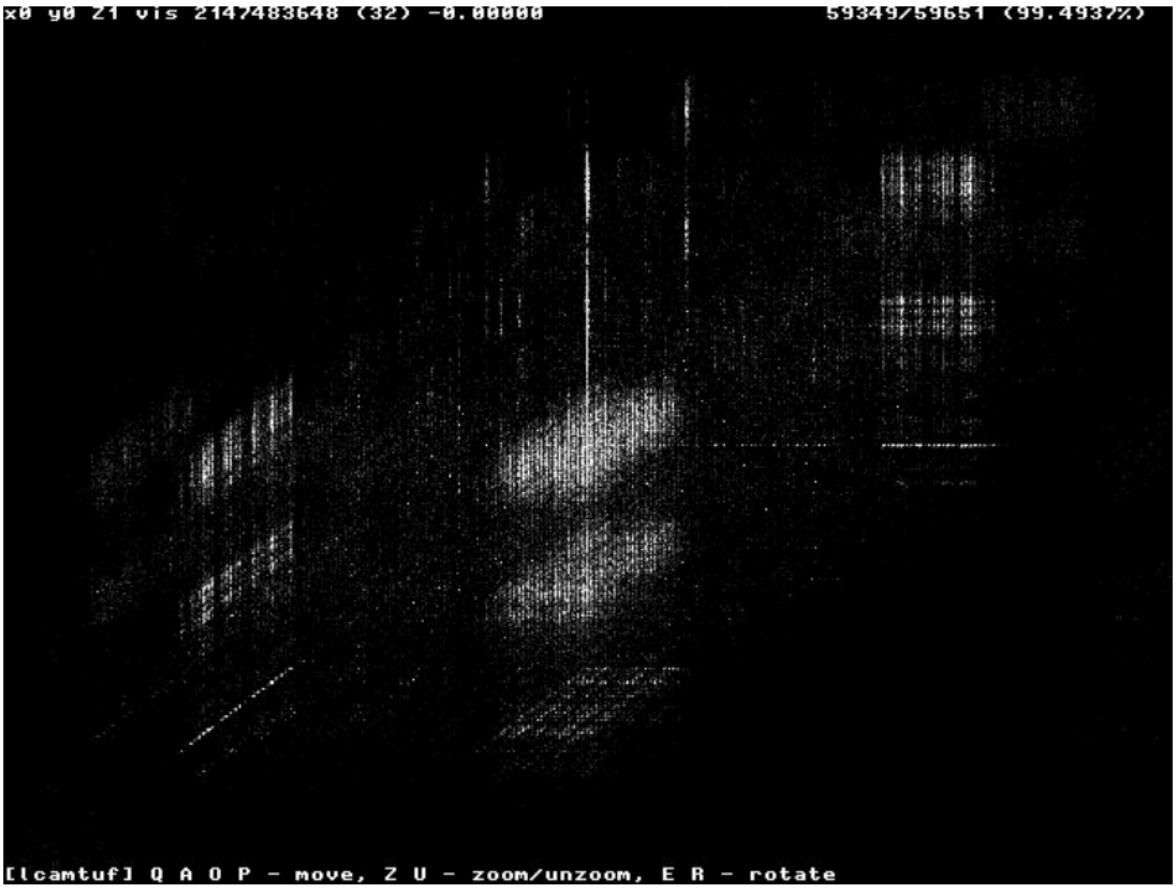


Рисунок 10-4. Представление PRNG без сильной корреляции, но с заметными статистическими смещениями.

Красивые картинки: галерея стеков TCP/IP

Метод визуализации, казалось, работал как волшебство, создавая уникальные и часто инстинктивно тревожащие, очаровательные узоры для многих реализаций, считавшихся достаточно безопасными. Многие изображения разбросаны по следующим страницам. Но могут ли картинки дать больше, чем наглядное представление трудно измеримых параметров и свойств генератора? Может ли атакующий осмысленно использовать загадочные трёхмерные формы либо компьютер каким-то образом исследовать их и ясно ответить, что правильно, а что нет? Легче ли взломать генератор в форме подсолнуха, чем в форме кирпича?

Прежде чем отвечать, позвольте прервать себя и включить небольшую галерею интереснейших результатов, полученных при написании первоначальной работы. Она показывает разнообразие и красоту наблюдаемых узоров, следуя древнему правилу: один трёхмерный график стоит тысячи слов. На рисунках 10-6 - 10-14 приведены портреты PRNG нескольких операционных систем.

Не все графики нарисованы в одинаковом масштабе; некоторые фигуры значительно меньше других. Масштаб и прочие параметры читаются в верхней строке каждого графика, как показано на рисунке 10-6.

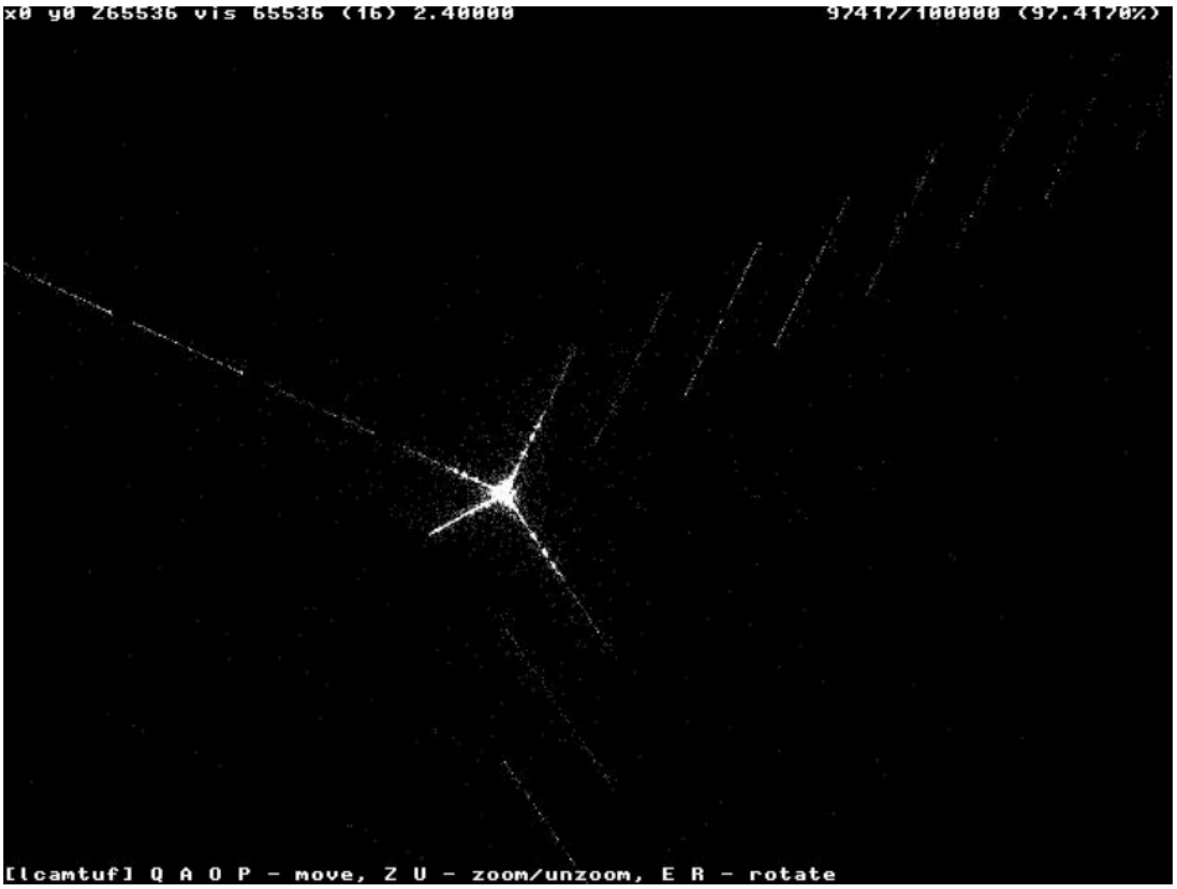


Рисунок 10-5. Распространённая схема временной зависимости, наблюдаемая в несовершенных условиях испытания.

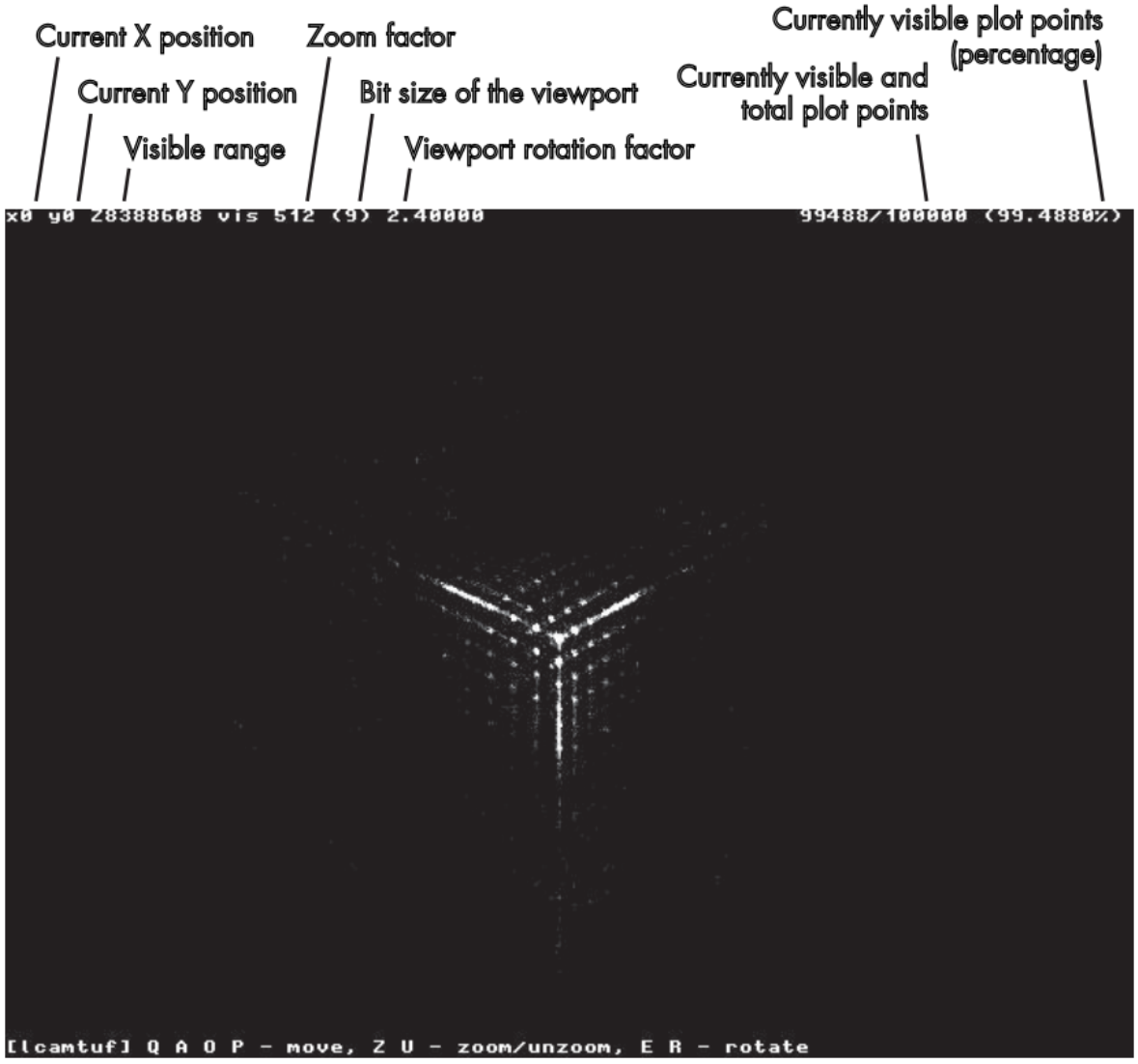


Рисунок 10-6. Windows 98. Показанное множество имеет диаметр около 128, то есть последовательные ISN увеличиваются на число с приблизительно 7 битами «случайности». Внутри заметна сильная частотная закономерность, похожая на один из предыдущих примеров и, возможно, указывающая на простейшую зависимость от времени во всех результатах. Размер аттрактора тревожно мал. Верхняя строка показывает текущие положения X и Y, коэффициент увеличения, видимый диапазон, битовый размер области просмотра, её поворот, число видимых и общее число точек, а также процент видимых.

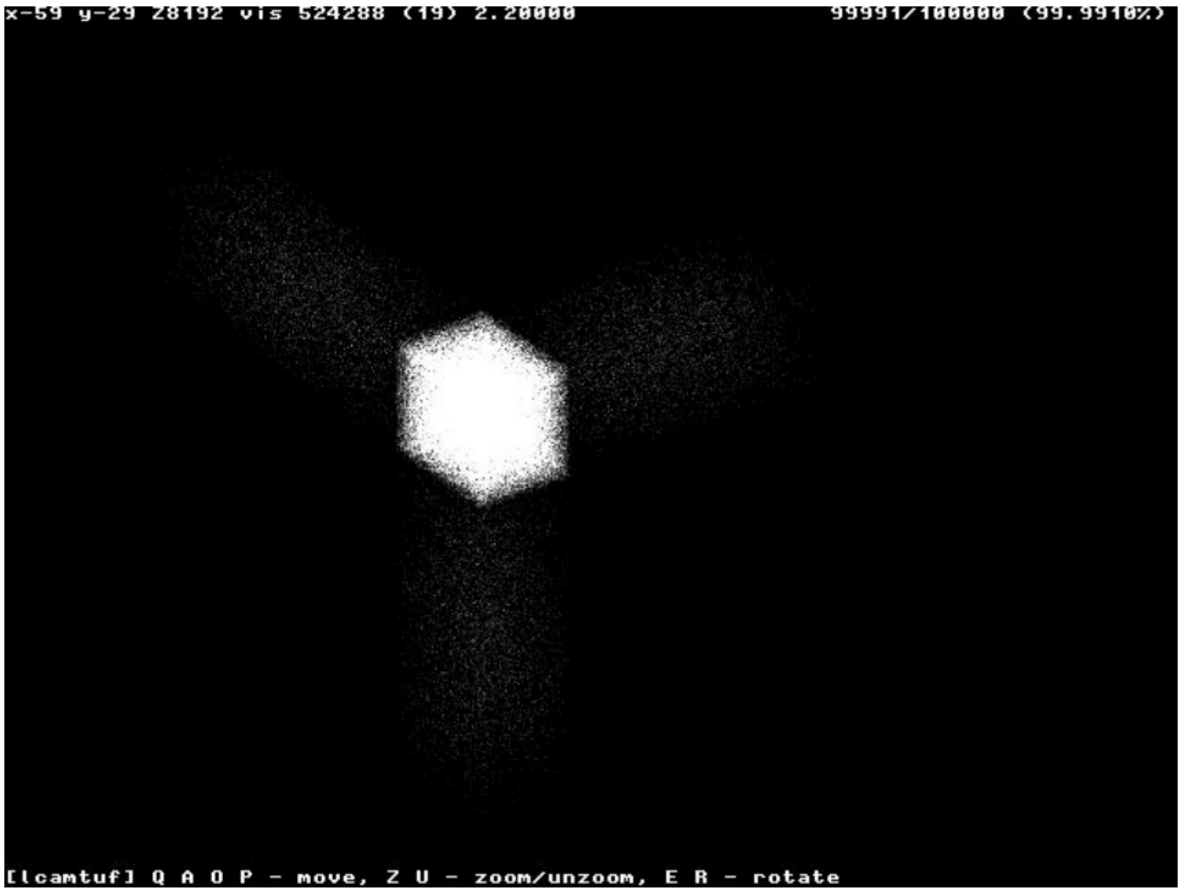


Рисунок 10-7. FreeBSD 4.2. Равномерный куб шириной 16 битов, вероятно, признак малых, но действительно случайных приращений на каждом шаге.

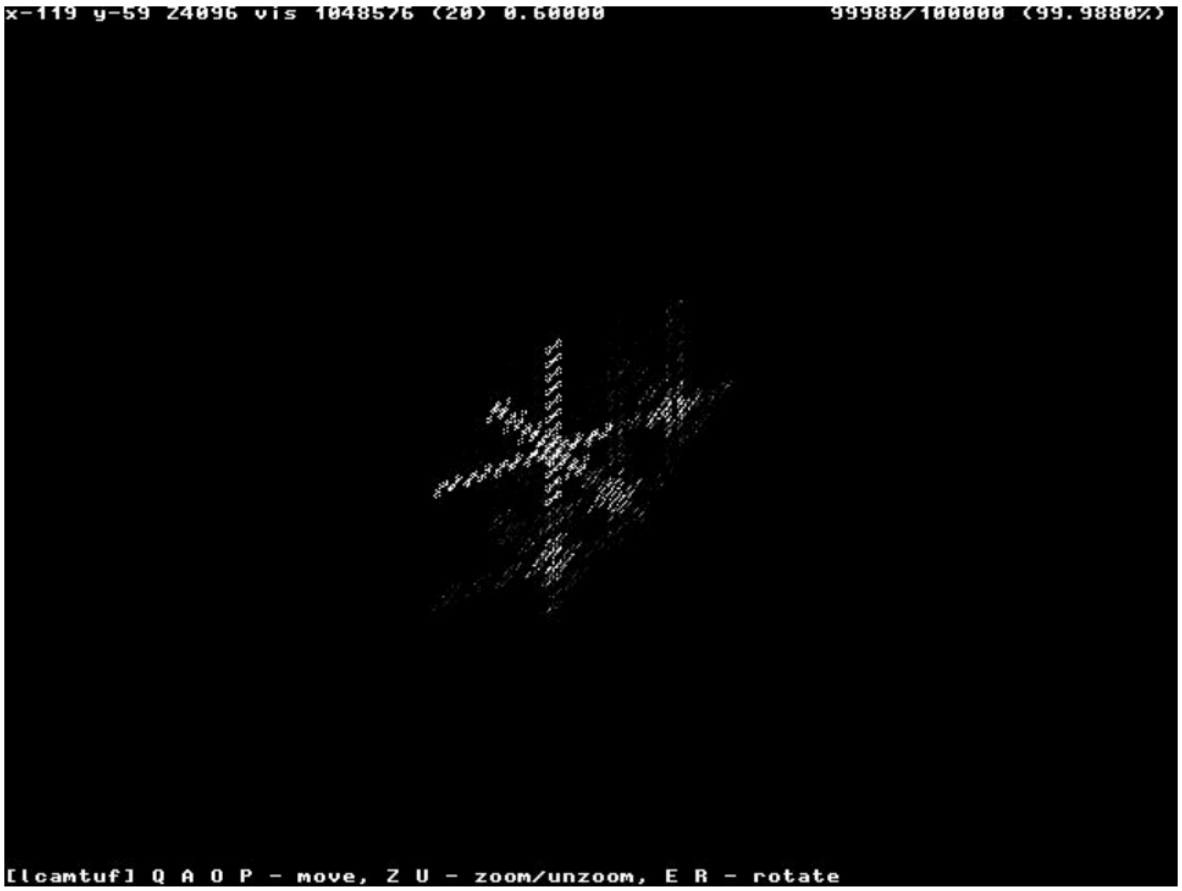


Рисунок 10-8. HP-UX 11. Странная структура в форме X-крыла шириной 18 битов, явно неравномерная и, вероятно, указывающая на высокий уровень корреляции ущербного PRNG.

Рисунок 10-9. Mac OS 9. Сходная, но немного иная 17-битная структура.

Рисунок 10-10. Windows NT 4.0 SP3. Снова сильный узор притяжения и крошечный аттрактор шириной 8 битов.

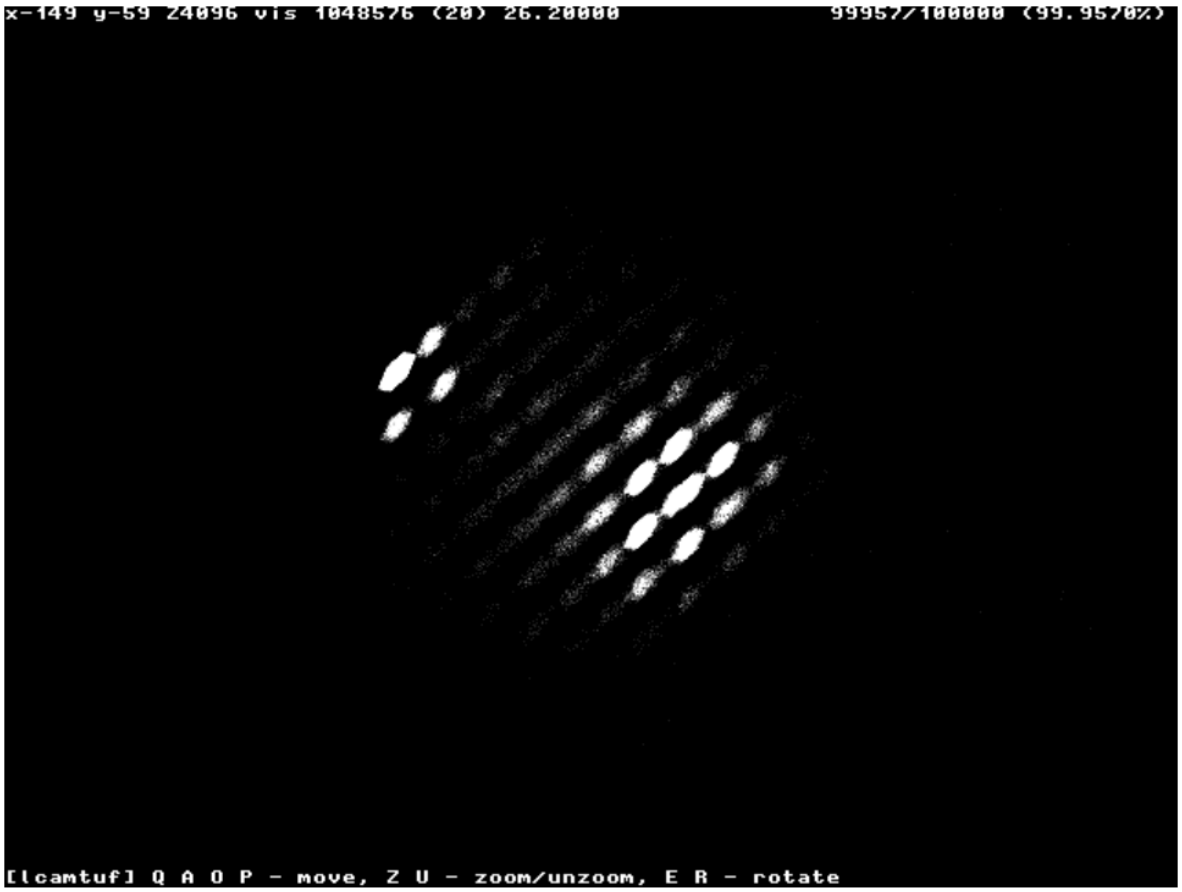


Рисунок 10-11. IRIX 6.5. Крайне неравномерное случайное облако шириной 16-18 битов; вероятно, ущербный алгоритм.

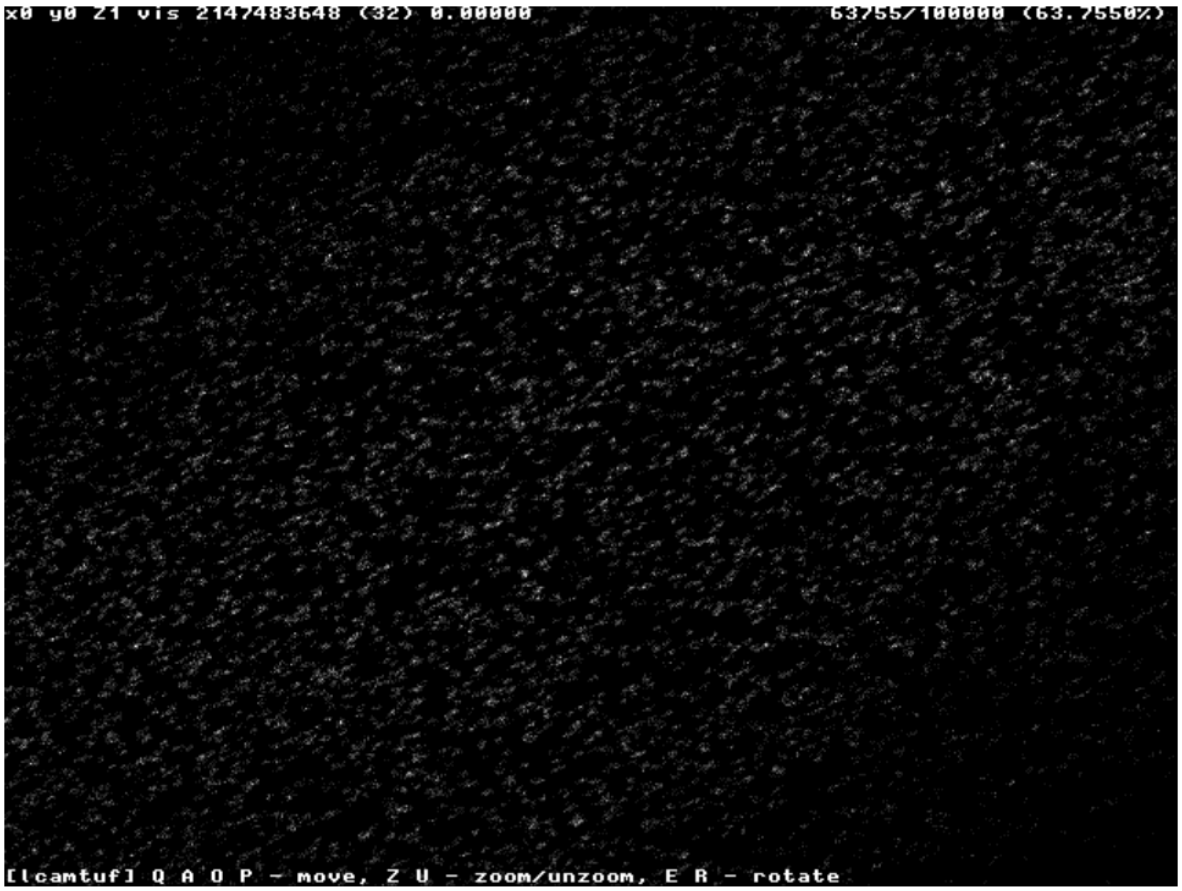


Рисунок 10-12. NetWare 6. Кажущаяся случайной система с 32-битным облаком аттрактора, которое, однако, неравномерно и состоит из множества участков высокой плотности.

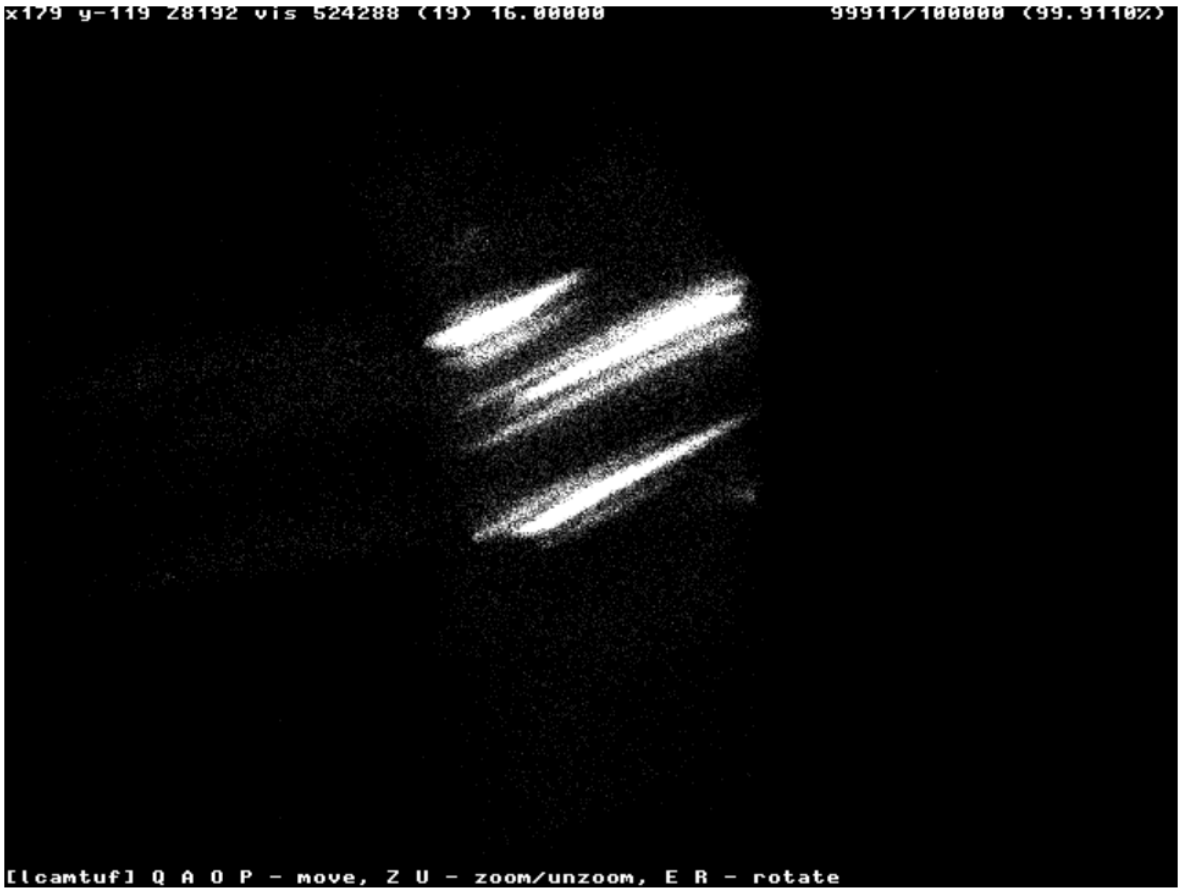


Рисунок 10-13. UNICOS 10.0.0.8. Странное 17-битное облако с неравномерными вытянутыми участками повышенной вероятности попадания.

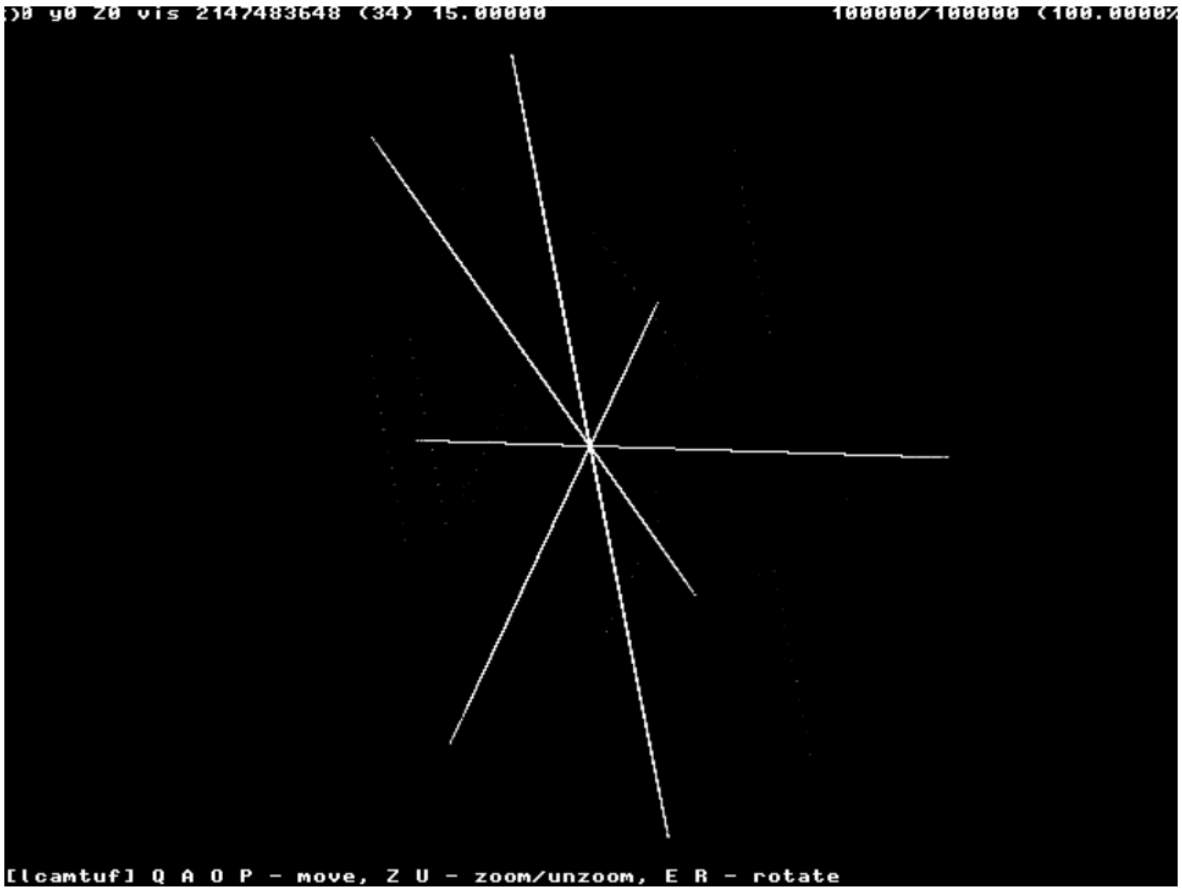


Рисунок 10-14. OpenVMS 7.2, стандартный стек TCP/IP. 32-битная структура с малой случайностью и сильными, но довольно необычными корреляциями, указывающими на сломанную конструкцию PRNG.

Атака с помощью аттракторов

Вернёмся к подсолнухам и кирпичам. Да, значение красивых картинок выходит за рамки визуального наслаждения заядлых компьютерных фанатов. Оказывается, структура аттрактора каждой системы создаёт матрицу возможных схем поведения ISN с плотностями, соответствующими вероятностям появления определённой временной зависимости или статистической закономерности. Области повышенной плотности отвечают историческим корреляциям, которые с большей вероятностью повторятся в будущем; менее заполнённые посещаются реже. Поэтому после приблизительного картирования аттрактора конкретной системы атакующий способен угадывать будущие результаты. Но как именно формы обратно преобразуются в точные ISN?

Ключ к успешной атаке - понимание того, что координата `x` каждой точки зависит от `Dt`, то есть от номеров последовательности в моменты `t` и `t-1`, поскольку `Dt = St - S(t-1)`. Координата `y` зависит от `D(t-1)`, ISN в `t-1` и `t-2`, а `z` - от `D(t-2)`, ISN в `t-2` и `t-3`.

Предположим, атакующий отправил три зонда удалённой системе, структура аттрактора ОС которой уже нанесена на карту. Зонды соответствуют моментам `t-3`, `t-2`, `t-1` и, естественно, достаточны для восстановления координат `y` и `z` точки, обозначающей текущее поведение системы в аттракторе.

По наблюдавшимся неравномерностям атакующий может вывести значения x для известных y и z, которые вероятнее других. Координаты y и z соответствуют одной линии пространства аттрактора, перпендикулярной плоскости x, как на рисунке 10-15: совокупности точек со всеми возможными x, но известными остальными координатами. Места, где линия пересекает области высокой плотности или проходит рядом с ними, образуют набор наиболее вероятных x. Области самой низкой плотности, очевидно, реже соответствуют правильному x: при предыдущих измерениях точки аттрактора там не появлялись.

Способность построить набор кандидатов x для известных y, z - крупный шаг к успешной атаке. Зная ранее полученный S(t-1), атакующий легко вычисляет St для каждого кандидата x, то есть Dt:

```
St = x + S(t-1)
```

Получив три предыдущих номера S(t-3), S(t-2), S(t-1), атакующий определяет набор вероятных кандидатов на следующий St, который атакуемая система, скорее всего, выберет для нового соединения - уже не начатого самим атакующим, но того, в которое он надеется вмешаться.

Затем он проводит атаку, отправляя пакеты TCP/IP с номерами-кандидатами. Угадывать с первой попытки не обязательно: все неверные варианты удалённая реализация просто проигнорирует. Но как только значение поддельного пакета совпадёт с ожидаемым номером в пределах ожидаемого окна, трафик будет принят, победив защиту целостности сеанса TCP/IP.

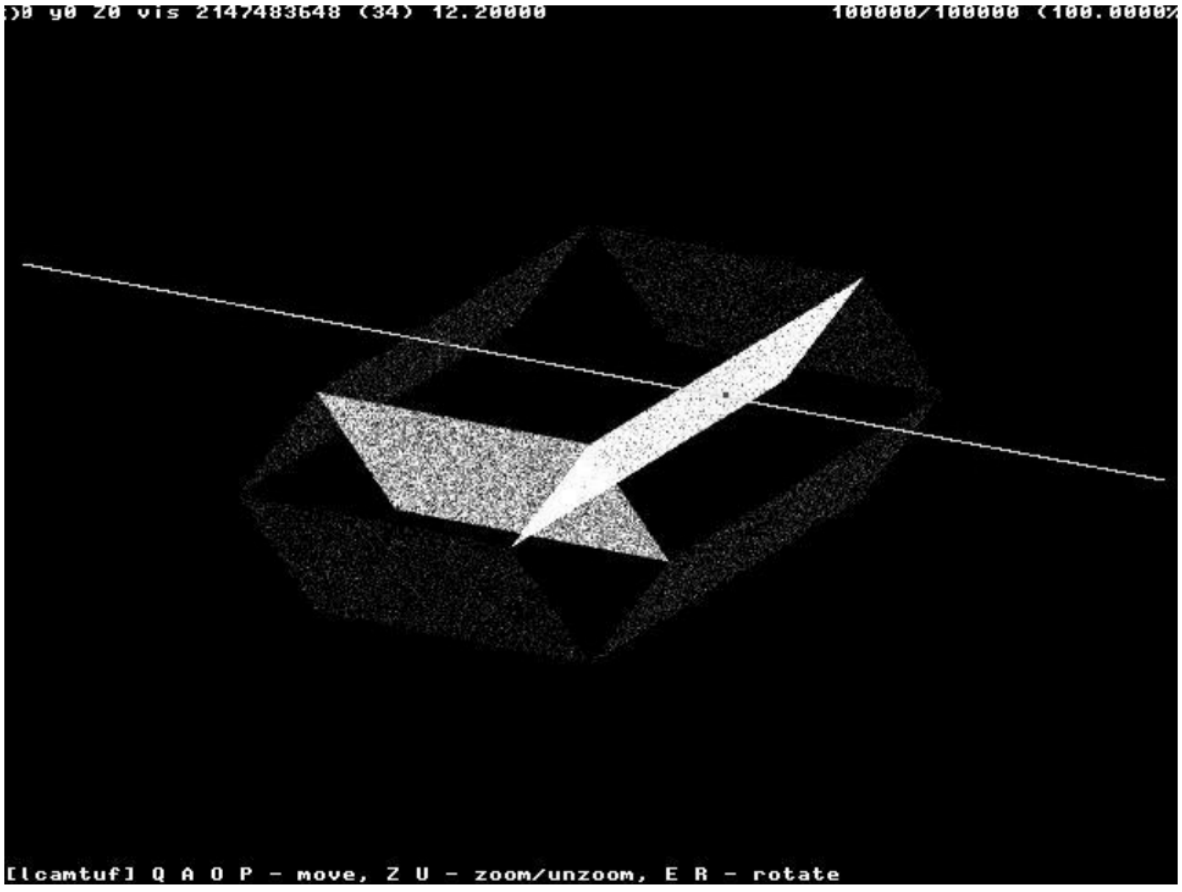


Рисунок 10-15. «Линия атаки», пересекающая аттрактор.

Разумеется, у атаки есть оговорки:

• наблюдаемая динамика может быть локальной для условий наблюдения или самого источника, хотя, судя по достигнутой успешности против распространённых реализаций, это маловероятно;

• если набор кандидатов особенно велик, как у алгоритмов с равномерными облаками аттрактора без явных неравномерностей, приём становится довольно непрактичным: для правильного предположения требуется слишком много попыток;

• поскольку обычно невозможно получить всю последовательность значений реализации ISN, ведь у некоторых систем длинные или неограниченные циклы, полный аттрактор построить нельзя. Поэтому применяется приближение: значение считается кандидатом, если точка находится в заданном радиусе от определённой точки линии `(y,z)`, что компенсирует пробелы даже в достаточно плотных областях аттрактора.

Чтобы сохранить осмысленность результатов и получить способ сравнительной оценки качества генератора ISN, я решил эмпирически оценить вероятность успеха с ограниченным числом попыток. В частности, определить вероятность попадания в верный номер за 5 000 попыток, предполагая, что атакующий с сетевым соединением от слабого до среднего способен послать за короткое время не более 5 000 пакетов.[1]

Для проверки подхода я разделил входные данные удалённых систем на две части: одну для построения аттрактора, другую для настоящих испытаний. Проверка читала четыре последовательных номера сразу, передавала первые три реализации, которая по данным аттрактора генерировала до 5 000 значений, а затем сравнивала выход с четвёртым номером испытательного набора. Для каждого аттрактора испытание сотни раз повторялось на последовательных четвёрках ISN, чтобы получить приблизительный процент верных угадываний, который на практике обозначает вероятность успеха атакующего.

Вот некоторые результаты для систем из галереи аттракторов:

Операционная система	**Осуществимость атаки**
IRIX 6.5.15	25% (25 из 100 попыток)
OpenVMS 7.2	15,00%
Windows NT 4.0 SP3	97,00%
Windows 98	100,00%
FreeBSD 4.2	1%
HP-UX 11	100,00%
Mac OS 9	89,00%

Подход оказался явно довольно эффективным[2] и побудил многих поставщиков переделать алгоритмы либо пересмотреть заявления об их безопасности. В последующем исследовании, опубликованном годом позже, в 2002-м, я изучил некоторые изменения, и далеко не все оказались удовлетворительными.

Но главный вопрос: какое отношение всё это имеет к пассивному снятию отпечатков ОС?

Возвращение к снятию отпечатков систем

Из способности наносить на карту динамику генератора определённой системы и того факта, что большинство реализаций создаёт более или менее уникальный узор фазового пространства, действительно вытекает несколько увлекательных следствий. Самый очевидный приём -

применение зондирования ISN к старомодному снятию отпечатков.

Наблюдая несколько номеров удалённой системы, например когда сторона пытается установить несколько соединений с сервером, можно сопоставить образец с библиотекой известных аттракторов и найти наиболее подходящий. Номера не обязаны быть предсказуемыми описанным способом атаки; аттрактор системы должен лишь отличаться.

По сравнению с традиционным пассивным снятием отпечатков метод обычно сообщает меньше подробностей конфигурации, но почти не поддаётся обману. Для противодействия пришлось бы изменить генерацию номеров, а значительно настроить её из пространства пользователя обычно невозможно. Изменение ядра без ослабления безопасности, как правило, требует хороших знаний и навыков, не говоря уже о доступе к исходникам.

Но разве это всё? Конечно нет!

ISNProber: теория в действии

Помимо картинок и теории полезно увидеть, как получение образцов ISN работает в настоящем мире и помогает оценивать конфигурацию удалённой системы или различать её экземпляры. К моему счастью, существует достойная упоминания программа.

Прочитав мою работу по анализу ISN TCP/IP, Том Вандепул написал превосходный инструмент ISNProber. Он использует анализ номеров для различения нескольких экземпляров одной системы, исходя из того, что две отдельные системы, скорее всего, находятся в разных местах аттрактора.

В простейшем случае ISNProber определяет, что за общим адресом скрываются две системы, по виду наблюдаемых ISN. Для простоты предположим, что система Y использует генератор с увеличением на единицу. Мы обращаемся к IP веб-сайта `www.example.com` и хотим определить число систем. Сначала опознаём `www.example.com` как Y, устанавливаем несколько последовательных соединений и наблюдаем ISN: `10, 11, 534, 13, 540, 19`.

Очевидно, меньшие числа образуют последовательность компьютера, который либо обработал меньше трафика, либо работает меньше времени, `10, 11, 13, 19`, а большие относятся к другой системе. Значит, два компьютера совместно обслуживают один общедоступный IP, возможно, за балансировщиком нагрузки. Меняя интервалы получения образцов, можно тщательно исследовать тип балансировщика, его политику распределения запросов и получаемый трафик.

Подход способен не только различать системы за общим адресом, но и отслеживать пользователей Y при переходе с одного IP на другой, пока они не перезагрузят машину и не сбросят счётчик ISN. У систем со схемой сложнее примера различение труднее, но определённо возможно, если ISN не полностью случайны во всех 32 битах. В последнем случае возникают опасения столкновений.

Здесь требуется лишь некоторая предсказуемость алгоритма. Поэтому анализ начальных номеров TCP/IP выглядит многообещающей альтернативой или добавлением к традиционному пассивному снятию отпечатков и, к сожалению, может служить полезным средством вторжения в приватность и отслеживания пользователей.

Предотвращение пассивного анализа

Защититься от предсказания номеров довольно просто, и хорошие решения вроде спецификации RFC 1948 Стивена М. Белловина[2] доступны давно. Но предотвратить пассивный анализ чисел очень трудно: проблема возникает не только из слабости алгоритмов, но и из их разнообразия,

из-за которого немногие системы имеют одинаковый след ISN. Даже реализации RFC 1948 и другие криптографически безопасные генераторы на внешней энтропии могут значительно различаться поведением в зависимости от тонкостей алгоритма и представлений автора о достаточных для противодействия атаке значениях.

Некоторой защиты можно добиться межсетевым экраном с отслеживанием состояния, переписывающим все номера последовательности исходящих пакетов,[3] благодаря чему все системы защищённой сети выглядят примерно одинаково. К сожалению, функцию предлагают лишь некоторые решения, и лишь некоторые способны получить от неё пользу.

Пища для размышлений

Анализ фазового пространства полезен далеко за пределами генерации номеров последовательности. Другие параметры, выбираемые псевдослучайно или по внутренней схеме, - поля ID пакетов IP, идентификаторы запросов DNS, как на рисунке 10-16, создаваемые приложениями «секретные» cookie пользовательских сеансов и так далее - можно успешно анализировать для поиска недостатков конструкции либо опознания реализации, упрощения дальнейшего анализа или содействия атаке.

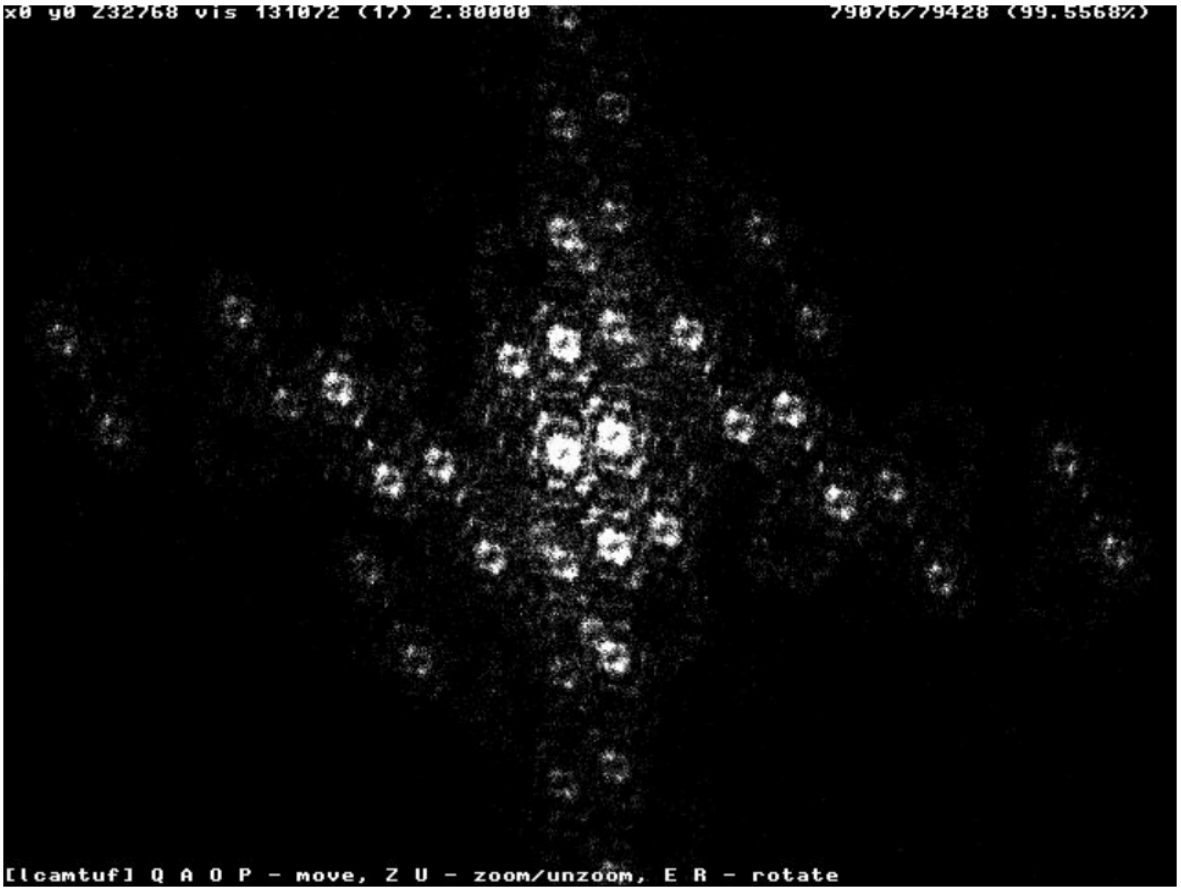


Рисунок 10-16. Интересный узор аттрактора реализации разрешения имён Linux.

Некоторая работа в этом направлении уже выполнена или ведётся. В исследовании, отчасти связанном с моей первоначальной работой, Джо Стюарт рассматривает проблемы DNS,[3] возникающие с развитием механизмов предсказания номеров. Он отмечает, что протокол DNS на UDP предлагает явно недостаточные способы проверки запросов даже против «малобюджетных» атак подделки, а низкое качество уникальных идентификаторов запросов разных реализаций

дополнительно ослабляет схему и делает её элементарно уязвимой для вредоносного внедрения данных. Учитывая, что DNS - одна из основных служб Интернета, а перспектива подделать ответ DNS популярного сайта и перенаправить всех пользователей конкретной сети на другую страницу отнюдь не непривлекательна, отравление DNS возглавляет мой список недооценённых интернет-угроз.

Дэн Камински приводит интересные, более совершенные визуализации предположительно случайных данных по адресу doxpara.com, с которыми определённо стоит познакомиться, см. рисунок 10-17.

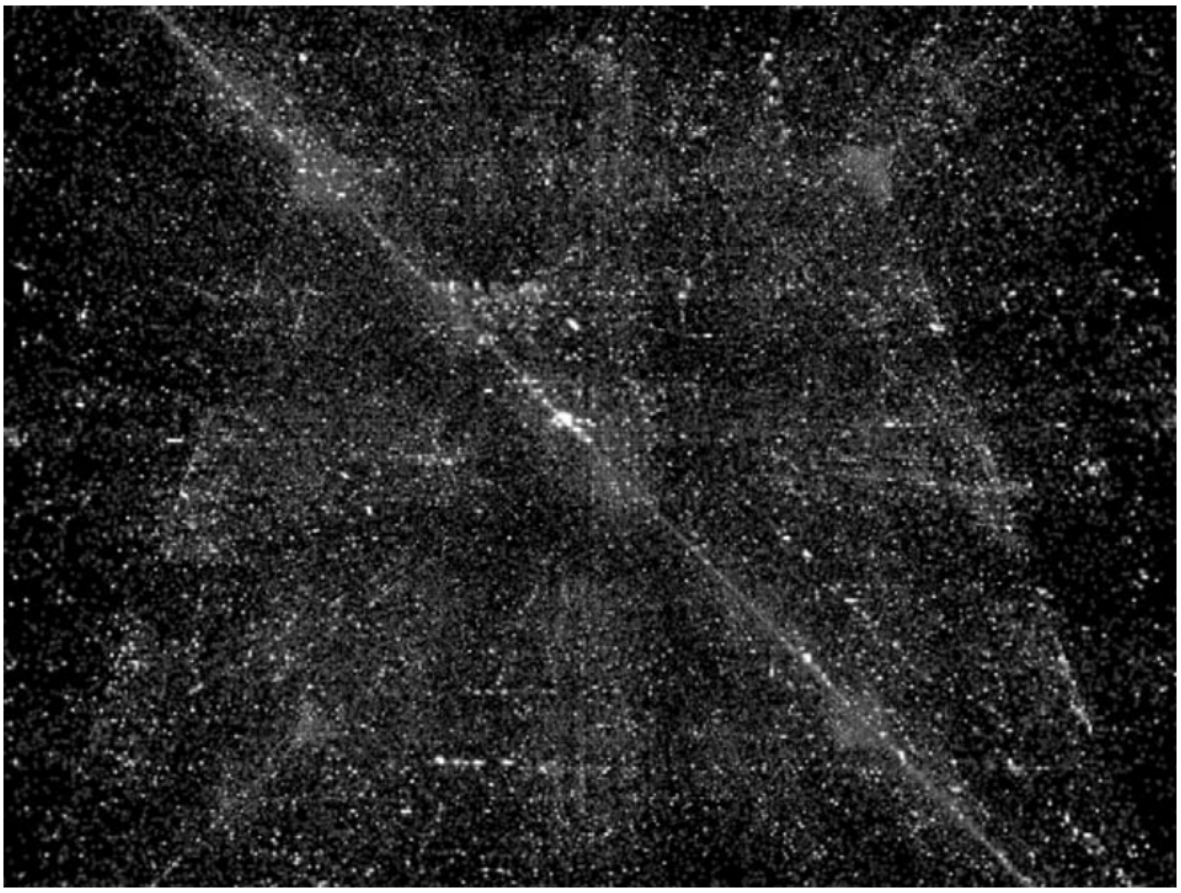


Рисунок 10-17. Представление случайности ядра BSD Дэном Камински. Предоставлено `www.doxpara.com`.

Сноски

[1] [назад] Самый малый пакет SYN занимает 40 байтов, поэтому 5 000 таких пакетов потребляют не менее 200 килобайтов пропускной способности. По модемной линии со сжатием V.42bis этот объём успешно отправляется за 10-20 секунд. Порог выбран довольно произвольно, но кажется разумным.

[2] [назад] Результаты относятся к сценариям, где необходимо точное внедрение или подделка данных. Если точность менее важна либо цель состоит лишь в нарушении работы, удалённая сторона примет не только пакеты с точным номером, но и попадающие в размер окна из параметров TCP/IP, см. главу 9. Иными словами, атаки отказа в обслуживании будут ещё успешнее.

[3] [назад] Solar Designer отмечает, что технически это можно хитро реализовать и в экране без состояния. Он объединяет, например посредством XOR, исходный номер с безопасным хешем секретного ключа и четвёркой адресов и портов, однозначно задающей соединение. У возвращаемых пакетов хеш снимается повторным XOR, чтобы при доставке пакет соответствовал представлению внутреннего хоста о соединении, а снаружи существовал только в непредсказуемом случайном 32-битном виде. Это сработает для всех реализаций ISN, кроме самых сломанных, с частыми повторами и столкновениями.

Глава 11. Распознавание аномалий

Или что можно узнать из едва заметных несовершенств сетевого трафика

В предыдущих главах я разобрал и проанализировал множество способов извлекать потенциально и, вероятно, ценные порции информации из кажущихся несущественными «технических» параметров, которые сопровождают каждое сообщение подозреваемого в сети. Как вы, надеюсь, увидели, можно получить значительный объём сведений об отправителе, о предоставлении которых тот наверняка не подозревает или по крайней мере не слишком рад, что часто не может отказаться. С помощью широкого набора приёмов анализа пакетов и потоков в совершенном и счастливом мире можно измерить множество характеристик удалённой стороны и сопоставить её поведение с подписью конкретной системы и конфигурацией сети.

Но реальность несколько иная: некоторые наблюдаемые параметры хотя бы немного отклоняются от ожидаемого набора значений, обычно связанного с конкретным устройством или конфигурацией сети подозреваемого. Можно просто проигнорировать кажущиеся бессмысленными случайные расхождения и всё равно успешно опознать исходную систему или отследить пользователей, но это не обязательно разумно. Мы учимся не обращать внимания на подобные якобы незначительные помехи, однако в мире вычислительной техники ничто не происходит без веской причины, если хотя бы достаточно вольно понимать слово «веская». Исследование механизма видимых случайных аномалий и редких закономерностей вместо их игнорирования способно дать ценные сведения о прежде невидимых особенностях конфигурации сети.

В этой главе я внимательнее рассматриваю процессы, способные влиять на наблюдаемые свойства системы, и объясняю причину, назначение или его отсутствие, а также последствия вызывающих такое поведение технологий.

Само собой, большинство воспроизводимых изменений IP-пакетов, рассматриваемых здесь, происходит из более совершенных видов промежуточных систем, понимающих IP. Поэтому я начну с двух долго остававшихся без внимания тем: межсетевых экранов вообще и преобразования сетевых адресов, NAT, в частности.

Межсетевые экраны должны оставаться незаметными бастионами: чем меньше известно о том, чем пользуется другой человек, тем лучше для него. Однако несмотря на строгие политики и настройки, по мере усложнения и улучшения соответствия современным задачам безопасности устройства всё легче исследовать косвенным или пассивным зондированием.

Основы пакетных межсетевых экранов

Популярные межсетевые экраны[1] по существу являются классом промежуточных маршрутизирующих устройств, созданных для нарушения фундаментальной конструкции промежуточного маршрутизатора. В отличие от настоящих маршрутизаторов, которые должны без избирательности принимать решения по сведениям третьего уровня OSI, экраны обычно интерпретируют, используют или даже изменяют сведения верхних уровней, например TCP или HTTP. Хотя технология довольно нова, она предлагает устоявшийся и хорошо понятный набор решений и встречается и в домашних сетях, и в крупных корпорациях. Экраны настраиваются отклонять, разрешать или перенаправлять конкретные виды трафика, адресованного определённым службам, и, что неудивительно, ограничивают доступ к функциям и ресурсам всего проходящего трафика. Поэтому это мощное, хотя иногда чрезмерно разрекламированное и переоценённое решение безопасности и управления сетью.

Ключ к успеху во всех сетевых средах в том, что один сравнительно устойчивый компонент защищает множество сложных систем и служит отказоустойчивой мерой, если ошибка

конфигурации открывает уязвимую службу или функцию защищённого сервера. В крайних случаях экраном просто прикрывают плохую конфигурацию и отсутствие обслуживания системы, обычно с катастрофическими результатами.

Фильтрация без состояния и фрагментация

Простейшие экраны - пакетные фильтры без состояния. Они лишь исследуют отдельные свойства каждого пакета, например целевой порт попытки соединения SYN Transmission Control Protocol, а затем по одним этим признакам решают, пропустить ли пакет. Конструкция чрезвычайно проста, надёжна и экономна по памяти и ресурсам. Например, такой экран ограничивает входящие соединения с почтовым сервером только адресованными порту 25, SMTP, отбрасывая все остальные SYN. Без первоначального SYN соединение установить нельзя, поэтому атакующий не сможет содержательно взаимодействовать с приложениями других портов. Для этого экрану не нужно быть почти столь же быстрым и сложным, как сам почтовый сервер: он не хранит записи об установленных соединениях и их точном состоянии.

Проблема совершенно прозрачной защиты в том, что экран и окончательный получатель способны по-разному понимать параметры. Например, атакующий убеждает экран, что подключается к разрешённому порту, но формирует трафик так, чтобы конечный адресат прочитал его иначе и установил соединение с портом, который экран должен защищать. Тогда злоумышленник получает доступ к уязвимой службе или административному интерфейсу, и у нас неприятности.

Хотя вызвать такое непонимание кажется маловероятным, это довольно легко достигается с помощью старого друга, фрагментации пакетов, способом «атаки перекрывающимися фрагментами»,[2] описанным RFC 1858 в 1995 году. Атакующий посылает начальный пакет с началом запроса SYN TCP к разрешённому экраном порту, например 25. В конце недостаёт лишь крошечной части, а в IP-заголовке установлен флаг more fragments, но зачем экрану беспокоиться о завершающих данных?

Экран исследует SYN, проверяет целевой порт, признаёт его допустимым и пропускает пакет. Получатель не интерпретирует его сразу, помня рассмотренную в главе 9 сборку, а хранит до успешного завершения дефрагментации, которое произойдёт лишь после прихода последней части.

Затем атакующий отправляет второй фрагмент. Он создан так, чтобы перекрыть первоначальный ровно настолько, чтобы в буфере сборки перезаписать целевой порт, одно из полей TCP-заголовка. Фрагмент начинается с ненулевого смещения и лишён большей части заголовка, кроме перезаписываемого фрагмента.

Из-за этого и отсутствия сведений для проверки флагов TCP или других важных параметров, по которым экран решает разрешить либо заблокировать трафик, фильтр без состояния обычно передаёт второй фрагмент как есть. После объединения с первым у получателя он меняет исходный целевой порт на более непослушное значение атакующего и действительно открывает соединение с портом, который экран должен защищать. Упс.

> **Примечание.** Хорошо спроектированный фильтр без состояния защищается первоначальной дефрагментацией перед анализом пакетов. Но это делает его несколько менее «не хранящим состояния» и менее прозрачным.

Фильтрация без состояния и несинхронный трафик

Другая проблема в том, что фильтры далеко не столь герметичны, как хотелось бы. Фильтрация возможна, только если один пакет содержит все сведения для обоснованного решения. После первоначального рукопожатия соединение TCP в основном симметрично: обе стороны имеют равные права и обмениваются данными пакетами одного типа, ACK. Поэтому трудно применять содержательные фильтры к чему-либо, кроме начальной стадии. Без отслеживания и записи соединений нельзя определить, кто и вообще кто-либо начал соединение, по которому идут ACK. Поэтому сложно осмысленно задать политику для ACK и промежуточных пакетов FIN или RST.

Обычно неспособность фильтровать после SYN не проблема: если атакующий не доставит начальный SYN, он не установит соединение. Но есть ловушка: обработка трафика без SYN к определённому порту зависит от того, закрыт порт или система его слушает. Например, некоторые ОС отвечают RST на заблудившиеся FIN и ничего не отвечают на открытых, слушающих портах.[1]

Поэтому против фильтров без состояния можно применять сканирование FIN или ACK, последнее впервые описано Уриэлем Маймоном в Phrack Magazine,[3] а также сканирование NUL и Xmas - незаконными пакетами соответственно без единого флага и со всеми флагами. Так собираются сведения перед атакой об открытых портах удалённой системы либо строится карта трафика, отбрасываемого экраном. Само знание открытого порта без возможности правильно подключиться не является непосредственной угрозой. Но сканирование часто раскрывает крайне ценные внутренние сведения, например ОС и службы, которые помогают провести более эффективную и трудно обнаруживаемую атаку после компрометации или обхода первой линии обороны. Поэтому это считается потенциальной слабостью фильтра без состояния.

Возможно, более серьёзная угроза связана с механизмом SYN-cookie в сочетании с фильтрацией без состояния. SYN-cookie защищают ОС от атак истощения ресурсов, где злоумышленник посылает хосту огромное число поддельных запросов соединения, что само по себе нетрудно. Получатель вынужден отправлять ложные ответы SYN+ACK, выделять память и расходовать другие ресурсы, добавляя будущее соединение в таблицы состояния TCP. Большинство систем либо чрезмерно расходуют ресурсы и почти останавливаются, либо в какой-то момент отказывают всем клиентам до истечения ложных соединений.

Для решения SYN-cookie помещают во все ответы SYN+ACK внутри поля ISN криптографическую подпись, фактически сокращение, однозначно идентифицирующее соединение, а затем полностью забывают его. Только когда от хоста приходит ACK и номер подтверждения проходит криптографическую проверку, соединение добавляется в таблицу состояния.

Но при такой конструкции существует вероятность, что SYN и ответ SYN+ACK вообще никогда не отправлялись. Если атакующий создаст ISN-cookie, успешно проверяемую алгоритмом хоста, возможно благодаря достаточной пропускной способности или слабому алгоритму, он пошлёт ACK и заставит удалённый хост добавить новое соединение в таблицу, хотя SYN никогда не посылался и SYN+ACK не принимался. Фильтр без состояния не узнает об установлении соединения, поскольку не видел первоначального запроса. Без SYN целевой IP и порт нельзя было проверить и одобрить либо отклонить, но соединение внезапно установлено.

Это действительно плохо.

Пакетные фильтры с состоянием

Чтобы решить проблемы, экран должен хранить сведения о предыдущем трафике и состоянии установленных потоков. Только так можно прозрачно предсказать результат дефрагментации либо получить контекст промежуточных пакетов и решить, незаконны ли они и должны быть отброшены или ожидаются получателем.

С ростом доступности высокопроизводительных вычислений стало возможно создавать гораздо более сложные и совершенные системы, чем когда-либо представлялось. Так мы пришли к отслеживанию соединений с состоянием: экран не только изучает отдельные пакеты, но запоминает контекст соединения и проверяет каждый пакет по этим данным. Это позволяет плотно запечатать сеть и игнорировать нежелательный или неожиданный трафик, не полагаясь на способность получателя всегда отличать хорошее от плохого. Такие фильтры пытаются отслеживать соединения и разрешать лишь трафик активных сеансов, предоставляя лучшую защиту и журналирование.

Разумеется, задача сложнее фильтрации без состояния и потребляет значительно больше ресурсов, особенно при защите крупной сети. Экрану внезапно нужны большие объёмы памяти и быстрый процессор для хранения и поиска сведений о происходящем в проводе.

Анализ с состоянием чаще вызывает проблемы или путаницу. Они начинаются, как только понимание текущего состояния сеанса TCP/IP экраном расходится с конечными точками, что вполне вероятно из-за неоднозначности спецификаций и разнообразия стеков. Например, получив пакет RST за пределами номеров, принимаемых получателем, экран с менее строгой проверкой может заключить, что соединение закрыто, тогда как адресат считает сеанс открытым и готов принимать дальнейшую связь, или наоборот. В конечном счёте проверка состояния имеет цену.

Перезапись пакетов и NAT

Для улучшения интерпретации и защиты от атак фрагментацией экраны научили не только передавать, но и переписывать части трафика. Например, один подход устраняет неоднозначность обязательной дефрагментацией и сборкой перед сравнением пакета с правилами доступа администратора.

С развитием решений стало ясно, что перезапись не только полезна сети, но и даёт качественный скачок безопасности и функциональности благодаря чрезвычайно полезным технологиям вроде NAT. Это практика сопоставления некоторых IP-адресов с другим набором IP перед пересылкой и обратного преобразования ответов защищённой системы. NAT с состоянием можно, помимо прочего, применять в отказоустойчивых конфигурациях, где один общедоступный IP обслуживается несколькими внутренними серверами. Либо ради экономии адресов и безопасности внутренняя сеть использует пул частных недоступных извне адресов, а хосты выходят в Интернет, «маскируясь» под одну машину с общедоступным IP.

В первом сценарии NAT переписывает целевые адреса входящих пакетов на несколько частных систем за экраном. Получается отказоустойчивая балансировка нагрузки: последовательные запросы популярного сайта, возможно http://www.microsoft.com, или другой критической службы распределяются по набору систем, и при отказе одной остальные берут работу на себя. Иногда задачу выполняют выделенные устройства, что неудивительно называемые балансировщиками нагрузки, но часто её поддерживают и экраны с NAT.

Во втором сценарии, обычно называемом маскарадингом, исходные адреса внешних пакетов переписываются, чтобы множество защищённых частных систем, возможно использующих не маршрутизируемые из Интернета адреса вроде `10.0.0.0`, подключались к внешнему миру через перехват и изменение соединений экраном. Системы скрыты, а для внешних получателей их действия выглядят исходящими от экрана. Соединение сопоставляется конкретному общедоступному IP и порту, затем трафик выпускается наружу. Все ответы назначения этому IP и порту переписываются обратно на частную систему-инициатора и передаются внутренней сети. Поэтому сеть рабочих станций, не предназначенных для служб Интернета, остаётся недоступной извне, значительно повышая безопасность, скрывая часть структуры и экономя дорогие общедоступные IP, иначе необходимые каждой системе. Имея лишь один маршрутизируемый

общедоступный IP, можно построить сеть из сотен или тысяч компьютеров и дать им доступ в Интернет.

Трудности перевода

Преобразование адресов сложнее, чем кажется: некоторые протоколы верхнего уровня не ограничиваются подключением к удалённой системе и отправкой команд. Например, древний, но чрезвычайно популярный File Transfer Protocol,[4] FTP, в базовом и наиболее поддерживаемом режиме устанавливает обратное соединение от сервера к клиенту для передачи запрошенных данных; начатое клиентом исходное соединение служит лишь для команд. Многие другие протоколы, прежде всего некоторые чаты, одноранговые средства совместной работы или обмена данными, службы вещания мультимедиа и тому подобное, также имеют странные конструкции с обратными соединениями, перескакиванием портов или допуском к рабочей станции определённого трафика без сеанса, например пакетов UDP.

Чтобы не сделать протоколы бесполезными, каждая реализация маскарадинга должна иметь несколько протокольных помощников. Они исследуют прикладные данные соединения, иногда переписывают их и временно открывают дыры в экране для обратной связи. Здесь возникает другая проблема, впервые обнаруженная Микаэлем Олссоном в помощнике FTP несколько лет назад,[5] а затем исследованная в других помощниках в том числе автором книги.[6] Они открывают дыры по сведениям, отправленным рабочей станцией удалённой системе через определённый протокол, предполагая, что трафик создан от имени пользователя и с его ведома. Само собой, некоторые программы, например браузеры, можно обманом заставить посылать определённый сетевой трафик, в том числе похожий на протокол, который программа изначально не поддерживает, и даже делать это автоматически посредством специально созданного вредоносного содержимого. Поддельный трафик способен убедить помощника пробить дыру.

Классический пример - злоупотребление обычным браузером. Добавив ссылку на веб-страницу или элемент, якобы находящийся в системе атакующего на нестандартном порту HTTP, который при этом вполне стандартен для FTP, клиент принуждают подключиться к ресурсу и послать HTTP-запрос. Поскольку порт обычно используется FTP, помощник экрана начинает слушать разговор, надеясь при необходимости помочь.

Следующий URL заставит HTTP-клиент подключиться к FTP-порту и выдать нечто похожее на команду FTP `PORT`, которую подхватит помощник:

```
HTTP://SERVER:21/FOO<RETURN>PORT MY_IP,122,105<RETURN>
```

Для законной службы FTP на другом конце запрос клиента будет бессмысленной тарабарщиной, а её ответ - непонятным веб-клиенту. Но дело не в этом. Важно, что атакующий управляет частью запроса, именем файла, который клиент попросит у сервера. Выбранное злоумышленником вымышленное имя может содержать любые данные. Включив подстроки, обычно признаваемые запросами FTP, атакующий обманывает помощника, слушающего конкретную текстовую команду `PORT`, заставляя его поверить, что пользователь пытается загрузить определённый файл. Поэтому удалённому серверу временно разрешается подключиться к жертве, здесь к непослушно звучащему порту `31337`, поскольку `122*256+105=31337`. Так мы впускаем атакующего без ведома жертвы. Снова упс: получили больше, чем ожидали.

Последствия маскарадинга

Все предыдущие сценарии связаны со злоупотреблением маскарадингом, однако одно его присутствие уже способно дать интересные сведения о другой стороне.

Как отмечалось ранее, маскарадинг не является ненавязчивым. Его основной принцип - изменять исходящий трафик, переписывая части. При этом он выходит за пределы простой корректировки адреса и не только делает...
возможным вывод о наличии маскарадинга, но и позволяет внимательному наблюдателю определить конкретную систему межсетевого экрана. При маскарадинге могут наблюдаться следующие изменения:

• Расхождение между TTL приходящих пакетов и ожидаемым или измеренным расстоянием до сети назначения. Трафик из-за маскарадинга как минимум на один переход «старше» пакета системы, получающей IP для исходящих соединений непосредственно из защищённой сети.

• В исходной сети чаще всего находятся разные ОС либо немного отличающиеся конфигурации или времена работы с разными свойствами TCP/IP, рассмотренными в главах 9 и 10. Наблюдение нескольких отпечатков в соединениях, словно исходящих от одного IP, служит сильной подсказкой о NAT на конкретной машине и внутренней сети за ней.

• Наконец, удалённый наблюдатель, вероятно, заметит сдвиг исходных портов. Это в иных условиях необычное явление возникает, когда соединения сети используют эфемерные исходные порты вне обычного диапазона конкретной ОС.

Каждая ОС резервирует определённый диапазон исходных портов для локального идентификатора конечной точки всех внешних соединений. Однако экран часто применяет для отображения маскируемых соединений другой диапазон, свойственный ОС устройства NAT. Если наблюдаемое не соответствует ожидаемому для опознанной системы, например Linux, обычно работающий с 1024 по 4999, использует очень большие номера, можно вывести наличие преобразования адресов и иногда тип экрана.

Эти широко применяемые приёмы лежат в основе обнаружения маскарадинга и разведки скрытой сети. Но доступны и другие способы выявления перезаписи пакетов.

Рулетка размера сегмента

Один менее очевидный и потому менее популярный способ обнаружить устройства перезаписи и узнать больше о конфигурации - анализ Maximum Segment Size входящего трафика.

Фрагментация IP добавляет заметные служебные расходы и часто считается кошмаром производительности, поэтому разработчики стараются её предотвращать. Но устранить фрагментацию трудно: заранее точно, быстро и надёжно определить MTU всего пути до начала связи почти невозможно. Даже лучший доступный способ, обнаружение MTU пути, далёк от совершенства и при срабатывании влияет на производительность. Для нахождения правильного MTU методом проб и ошибок некоторые не помещающиеся пакеты приходится отбрасывать и отправлять заново.

Чтобы предотвратить последствия PMTUD для производительности и надёжности и уменьшить расходы фрагментации, многие экраны NAT, переписывающие параметры исходящего трафика, также меняют объявленный MSS TCP-заголовков соединений частной сети на более подходящий внешнему каналу. Новое значение, вероятно, немного уже, имеет меньший MTU, чем LAN. Изменение не позволяет получателю отправить данные, которые не поместятся в соединение, если именно оно имеет наименьший MTU инфраструктуры, и делает фрагментацию менее вероятной. Предполагается, что несовместимость MTU чаще возникает рядом с отправителем или получателем на «последней миле», где встречаются DSL, беспроводные линии и другие соединения с малым MTU, требующие уменьшения пакетов.

Одно уменьшение MSS обнаружить непросто. Фактически нельзя сказать, установил ли значение отправитель или его изменили по дороге. За одним небольшим исключением. Напомним из главы 9 особенность выбора окна во многих современных системах:

> Размер окна определяет объём данных, отправляемый без подтверждения. Конкретное значение часто выбирается по личным правилам вуду и иным религиозным убеждениям разработчика. Два самых популярных подхода гласят, что оно должно быть либо кратным MTU за вычетом заголовков протокола, Maximum Segment Size, MSS, либо просто достаточно большим и «круглым». Старые версии Linux 2.0 использовали степени двойки, например 16 384. Linux 2.2 перешёл к кратному MSS, по какой-то причине 11 или 22 MSS, а новые версии Linux обычно применяют 2-4 MSS. Сетевая игровая консоль Sega Dreamcast использует 4 096, а Windows часто - 64 512.

Всё больше современных систем, включая новые Linux и Solaris, отдельные Windows и SCO UnixWare, используют размер окна, кратный MSS. Поэтому вмешательство в MSS легко обнаружить: окно полученного пакета перестанет быть конкретным кратным и, скорее всего, вообще не будет делиться на MSS.

Сравнение MSS с окном надёжно обнаруживает группу экранов, поддерживающих MSS clamping, подгонку к соединению, в трафике самых разных систем. Хотя в Linux и FreeBSD она необязательна, домашние экраны, интеллектуальные DSL-маршрутизаторы и другие домашние сети часто выполняют её автоматически. Поэтому аномальный MSS указывает не только на устройство перезаписи, но и на связь с возможностями NAT, что становится признаком сетевого подключения отправителя.

Отслеживание состояния и неожиданные ответы

Другое важное следствие отслеживания состояния и перезаписи: некоторые предписанные RFC ответы создаёт экран, а не отправитель, что позволяет эффективно обнаруживать и зондировать устройство. После удаления соединения из таблицы состояния NAT из-за истечения времени или завершения одной стороной пакетом RST, не дошедшим до другой, дальнейший трафик сеанса уже не пересылается получателю, как фильтром без состояния, а обрабатывается самим экраном.

Спецификация TCP/IP требует отвечать RST на все неожиданные ACK, уведомляя отправителя, что продолжаемый им сеанс больше не признаётся или никогда не признавался. Некоторые экраны могут нарушать RFC и вообще не отвечать, просто отбрасывая пакеты вне существующего сеанса. Это не всегда мудро, поскольку вызывает лишние задержки, когда законное соединение потеряно из-за временных сетевых проблем.

Однако многие устройства отвечают законным ожидаемым RST, открывая новое направление обнаружения и точного снятия отпечатка экрана. Поскольку пакет создаётся с нуля экраном, параметры относятся к нему, а не к защищаемой системе. Поэтому традиционные приёмы главы 9, изучение DF, TTL, окна, типов, значений и порядка параметров и так далее, определяют сам экран.

Существует и другая возможность согласно RFC 1122:[7]

> **4.2.2.12 Сегмент RST: RFC 793, раздел 3.4**
>
> TCP СЛЕДУЕТ разрешать включение данных в полученный сегмент RST.
>
> **ОБСУЖДЕНИЕ:** Предлагалось, чтобы сегмент RST мог содержать текст ASCII, кодирующий и объясняющий причину RST. Стандарт таких данных пока не установлен.

И действительно, хотя стандарт не установлен, некоторые системы при заблудившемся ACK отвечают многословными, пусть зачастую загадочными сообщениями RST в надежде утешить

другую сторону знанием причины. Они нередко содержат внутренние ключевые слова или попытки странного жанра юмора фанатов, свойственные ОС: `no tcp, reset`; `tcp_close, during connect` в Mac OS; `tcp_fin_wait_2_timeout`; `No TCP` в HP-UX; `new data when detached`; `tcp_lift_anchor, can't wait` в SunOS.

Если в ответ на сетевую проблему или неожиданный трафик хосту приходит такой многословный RST, а мы знаем, что удалённая система не использует подобные сообщения, получаем подсказку. Между нами и адресатом находится устройство, вероятно экран с состоянием, а его ОС определяется сопоставлением ответа с известными сообщениями распространённых и редких систем.

Два приёма чрезвычайно эффективно обнаруживают фильтры с состоянием, когда трафик можно наблюдать во время кратковременных сетевых проблем. Их также можно применять для активного снятия отпечатков без непосредственного нацеливания на экран, отправляя цели заблудившийся ACK и различая фильтры. По ответу атакующий выбирает лучший способ приблизиться к экрану или иначе использует знание.

Надёжность или производительность: спор о бите DF

Обнаружение MTU пути, PMTUD, является направлением снятия отпечатков, тесно связанным со схемой предотвращения фрагментации IP из главы 9.

Недавние ядра Linux 2.2, 2.4, 2.6 и Windows 2000, XP реализуют и по умолчанию включают PMTUD. Если настройку не меняли, во всём их трафике установлен don’t fragment, DF. И снова алгоритм порой создаёт редкие, но не неслыханные проблемы.

Сценарии отказа обнаружения MTU пути

Проблема PMTUD в зависимости от способности отправителя получить ошибку ICMP «требуется фрагментация, но установлен DF» и определить оптимальные настройки. Вызвавший сообщение пакет отбрасывается до назначения, его нужно уменьшить и отправить снова.

Если отправитель не получит сообщение, он не узнает, что пакет не прошёл. В лучшем случае возникает задержка, в худшем - соединение зависает навсегда, поскольку повторные передачи также не пройдут через канал с максимальным размером меньше посылаемого.

Но ICMP от соединения со слишком большим пакетом не гарантированно достигает отправителя. Некоторые сети, в результате непродуманной попытки повысить безопасность, просто отбрасывают все ICMP. Наконец, даже отправленное устройством сообщение может не быть доставлено.

Почему ICMP отбрасывают? Исторически многие такие сообщения создавали проблемы: некоторые слишком большие или фрагментированные пакеты ICMP повреждали память ядра множества систем, это называли «ping of death». Сообщения широковещательным адресам использовались для шторма ответов поддельному источнику в атаке Smurf и для других отказов в обслуживании. Кроме того, неправильно настроенные системы нередко толковали определённый вид широковещательного ICMP, объявление маршрутизатора,[2] как команду изменить сетевые настройки. Принимая их независимо от доверия, они открывали ещё одно направление атаки. Поэтому ICMP боятся и блокируют многие.

> **Примечание.** Совет отклонять весь ICMP часто встречается в наивных руководствах по безопасности, и некоторые администраторы ему следуют. Я даже видел его в

профессиональных рекомендациях испытания на проникновение от признанного аудитора, имя которого, к сожалению, здесь раскрыть не могу.

Другая причина ненадёжности PMTUD - сообщения об ошибках от устройств с частным адресным пространством. Ради экономии ограниченных и обычно дорогих общедоступных IP интерфейсам кабеля между маршрутизатором и экраном удалённой сети иногда дают адреса из пула для частного локального использования, а не действительно маршрутизируемые в данную сеть извне.

К сожалению, это способно сломать PMTUD. Если пакет внешнего мира слишком велик для передачи экраном адресату, экран посылает ICMP со своим исходным адресом из частного пула. Экран исходного отправителя может отклонить ответ, поскольку тот словно пришёл из внешнего мира, но имеет IP частного пула, возможно даже того же, что частная LAN отправителя. Такой трафик обычно означает подделку доверенного внутреннего хоста. Однако здесь решение ломает сравнительно новый PMTUD и оставляет отправителя в неведении о недоставке.

Хуже того, даже при правильных условиях многие современные устройства ограничивают частоту ответов ICMP и не отправляют больше заданного числа за период. Это тоже мера безопасности. До PMTUD ICMP предназначались лишь для сведений и не были критичны для связи, поэтому ограничение казалось разумной защитой от некоторых DoS или истощения пропускной способности.

Борьба с PMTUD и её последствия

В свете сказанного некоторые считают PMTUD довольно плохой конструкцией. Он немного повышает производительность ценой редких, но стойких и обычно трудно диагностируемых проблем, мешающих доступу к определённым серверам или неожиданно останавливающих соединения. Было создано много алгоритмов «обнаружения чёрных дыр», находящих хосты и сети, где PMTUD следует отключить, с разным успехом, но они не решают проблему полностью и могут добавлять задержки, обычно в самый неподходящий момент.

Чтобы избежать жалоб, некоторые коммерческие поставщики экранов применяют грязный трюк: очищают DF во всём исходящем трафике. Изменение тонкое и часто ценимое, но превосходно обнаруживает фильтрующее и переписывающее устройство. Если по адресу или сети видны свойства систем с PMTUD, но во входящих пакетах ожидаемого DF нет, внимательный наблюдатель выводит присутствие и тип экрана, получая ещё крошечную порцию данных без взаимодействия с жертвой.

Пища для размышлений

Так заканчивается маленький рассказ о том, как улучшение и усиление экранов против проникновения и прямой разведки одновременно облегчило их изучение косвенной оценкой. Но позвольте короткое отступление.

Возможно, самое странное и интересное открытие я встретил примерно в 1999 году. Оно не связано непосредственно с устройством экранов, но даёт пищу каждому, кого интересует пассивное снятие отпечатков промежуточных систем.

Яцек П. Шиманьский, с которым я недолго работал, а позже с удовольствием обсуждал необычные и подозрительные картины трафика,[3] заметил внезапный рост сильно сломанных пакетов TCP/IP к порту 21536 и в меньшей степени к 18477 или 19535. Они всегда исходили с портов 18245, 21331 или 17736 и приходили от множества систем пространства коммутируемых адресов польской национальной телефонной компании Telekomunikacja Polska.

После захвата нескольких пакетов обнаружилось сильное странное искажение. IP-заголовки были на месте, тип протокола установлен в TCP, но сразу за ними шла полезная нагрузка TCP - сами TCP-заголовки просто исчезли. Наблюдаемые сочетания портов возникали из толкования первых четырёх байтов нагрузки как пары чисел, которые при наличии TCP-заголовка были бы исходным и целевым портами. Пара 18245 и 21536 всего лишь представляла строку `GET` - четыре символа, открывающие большинство HTTP-запросов. Аналогично 18477 и 21331 означали `SSH-`, начало каждого Secure Shell, а 19535 и 17736 - `EHLO`, команду открытия всех сеансов ESMTP, Extended SMTP.

Но причина внезапного появления трафика оставалась загадкой. Почему только из этой сети? И почему искажение сетевым оборудованием не вызывало у пользователей проблем связи или других неудобств?

Ответ вскоре нашёлся. Весь наблюдаемый трафик исходил от Nortel CVX, системы модемного доступа, которую начала применять телефонная компания. Проблема случалась изредка под большой нагрузкой, поэтому отправлялась и достигала получателей, к их крайнему удивлению, лишь малая доля неполных пакетов. Вероятнее всего, причиной была неправильная блокировка очереди или управление буферами, заметные только при почти одновременной обработке многих сеансов. Некоторые пакеты словно уходили слишком рано, ещё «в процессе сборки», или иначе искажались реализацией.

Компания исправила TCP/IP вскоре после развёртывания в Польше, и все жили долго и счастливо. Но, как можно представить, они не первыми и не последними случайно оставили уникальный след систем в передаваемых пакетах.

Мораль: снова наивно отвергать то, что обычно игнорируется. В современном сетевом мире тонкие подсказки и необычные, неожиданные, необъяснённые наблюдения чрезвычайно ценны. Их легко найти, но трудно анализировать.

Дополнительной пищей и областью исследования могут стать способы противодействия снятию отпечатков. Некоторые поставщики экранов пытались внедрять меры, меняющие характеристики пакетов настройкой IP ID, номеров последовательности TCP и прочего. Само собой, решение помогает атакующему и даёт результат, прямо противоположный желаемому: если не изменить и не унифицировать все признаки, включая номера последовательности, время повторных передач, метки времени и так далее, можно определить и нижележащую ОС, и защищающий сеть экран.

C’est la vie.

Сноски

[1] [назад] Некоторые стороны поведения, склонность отвечать RST на заблудившиеся неожиданные пакеты к закрытым портам и игнорировать такой же трафик к портам слушающей службы, предписаны RFC 793; другие лишь выбраны конкретной группой разработчиков.
[2] [назад] Объявления маршрутизаторов должны были обеспечивать автоматическую конфигурацию хостов без ручного ввода настроек. Маршрутизатор периодически или по запросу передавал: «Вот я. Пользуйся мной». По умолчанию некоторые системы без больших колебаний принимали незапрошенные объявления - плохая идея.
[3] [назад] Сотрудничество в какой-то момент привело к созданию свободно связанной группы польских исследователей, которые в 1999-2000 годах сопоставляли, отслеживали и пытались объяснить множество причудливых видов неожиданного сетевого трафика.

Глава 12. Утечки данных стека

Ещё один короткий рассказ о том, где найти то, что мы вообще не собирались отправлять

Иногда для обнаружения тонких, но увлекательных и полезных подсказок о других сетевых гражданах и их местонахождении достаточно удачи. По крайней мере так было с довольно интересным и чрезвычайно неуловимым направлением раскрытия информации, которое я обнаружил в 2003 году после нескольких недель изнурительной охоты.

Сервер Кристьяна

Обо всём по порядку. Несколько лет назад я попросил друга Кристьяна дать немного дискового пространства на одной машине, чтобы разместить мои проекты на надёжной и быстрой системе. Он согласился, и вскоре я стал постепенно переносить программы и работы в новый дом. Среди них была новая версия p0f, моего инструмента пассивного снятия отпечатков ОС, знакомого по главе 9. Скромная программа реализовывала интересные приёмы пассивного анализа, но для настоящей силы ей требовалась хорошая актуальная база подписей операционных систем. Поддерживать её вручную было трудно, и вскоре у меня закончились редкие системы, с которых можно снять отпечаток и добавить его.

К счастью, если сбор подписей для активного снятия требовал часто нежелательного взаимодействия с целью, вызывавшего споры, нагружавшего сеть и иногда обрушивавшего особенно плохо реализованные стеки TCP/IP, пассивный способ ничего подобного не требовал и без усилий работал со всеми системами, обращавшимися к машине Кристьяна за моей страницей. Чтобы поощрить отправку сведений, я создал подстраницу, где любой пользователь сразу видел свой отпечаток и мог исправить распознавание системы либо добавить новую подпись. Страница прекрасно собирала подписи и улучшала программу, но история на этом не заканчивается.

В причудливом повороте событий Кристьян решил разместить на системе другой, коммерческий сайт, чтобы машина сама оплачивала счета. Как можно представить, он был посвящён вовсе не сетевой безопасности, садоводству или иному благородному делу, а менее престижным, но, возможно, более привлекательным сторонам нашей жизни: сексу, наготе и всему связанному. Я возликовал, как всякий уважающий себя фанат, не из-за содержимого, а потому что за считаные часы хлынули миллионы подписей соединений для анализа разрабатываемой программой. Аллилуйя!

Удивительные находки

Бережёного бог бережёт: проектируя новый код p0f, я решил реализовать множество проверок разумности для обнаружения самых причудливых, маловероятных и неслыханных схем входящего трафика, охватив все незаконные или бессмысленные сочетания настроек TCP/IP. Здравый смысл подсказывал, что пакеты со странно искажёнными параметрами никогда не встретятся, по крайней мере от популярных и хорошо испытанных систем, однако реализация функции не вредила. Если же система действительно отправит пакет с определённой аномалией, её обнаружение превосходно отличит конкретную ОС от похожих реализаций без недостатка.

В счастливые месяцы благословенной бури подписей я видел страннейшие вещи. Некоторые удалось объяснить и документировать для p0f, другие остались загадкой. Большинство ранее реализованных проверок попало в цель, и я сразу нашёл системы с более необычными причудами TCP/IP. Но одна вещь была особенно тревожной и невероятной, поэтому я уделил ей больше внимания.

Две проверки - значения ACK в заголовке при неустановленном флаге ACK, действительно бесполезное действие, и значения URG без флага URG - поначалу казались бессмысленными и не давали интересных результатов. Затем я заметил необычное: некоторые Windows 2000 и XP,

подключавшиеся к серверу Кристьяна, время от времени имели ненулевые URG или ACK в пакетах без соответствующих флагов, прежде всего в SYN, открывающих соединение.

Наличие значений без флагов строго говоря не проблема. Согласно RFC 793, они просто теряют всякое значение. Например:

> **Указатель срочности: 16 битов**
>
> Поле сообщает текущее значение указателя срочности как положительное смещение от номера последовательности данного сегмента. Указатель обозначает номер последовательности октета после срочных данных. Поле интерпретируется только в сегментах с установленным управляющим битом URG.

RFC 793 своим совершенно особым способом сообщает, что аномалия вряд ли вызовет сетевые проблемы и потому могла остаться незамеченной навсегда. Но я обратил внимание просто потому, что она была странной.

Сначала я считал виновным определённое сетевое оборудование, как в большинстве проблем главы 11, но это было не так. Срабатывания приходили от отдельных систем, не целых сетей, и не были постоянными: появлялись в нескольких пакетах, со стабильными или случайно меняющимися значениями, а затем исчезали и больше не возникали в последующих соединениях. Кроме того, проблема казалась исключительно свойственной Windows: среди затронутых не было ни одной редкой ОС.

Неделя за неделей я пытался её отследить. В ходе охоты развернул другие установки в более контролируемых средах и с изумлением увидел проблему даже в локальных сетях и самых современных системах, хотя лишь на короткие периоды. Пользователи не помнили ничего необычного в моменты такого трафика, а я не находил определённого вида связи или набора действий, вызывавшего его. Закономерности словно не было.

Загадочно.

Откровение: явление воспроизведено

Я был близок к сдаче. Опубликовал наблюдения в нескольких общедоступных списках рассылки, прежде всего VULN-DEV, популярном обсуждении уязвимостей от Security Focus, попросив других исследователей об анализе и отзывах, но безрезультатно. И тут по чистой случайности поймал собственную испытательную станцию на точно таком поведении, занимаясь совершенно иной проблемой. На фоне случайно работал сниффер - разве у нас бывает иначе?

Вскоре диагноз был готов: проблема возникала, когда рабочая станция во время попытки соединения выполняла фоновую передачу файла или иную интенсивную сетевую операцию. Почти в каждой ОС отправляемый пакет сначала строится в основной памяти посредством статического буфера, постоянного места исключительно для этой цели, либо динамического, выделяемого по необходимости из памяти, прежде занятой другим. Когда два соединения происходили примерно одновременно, буфер исходящих пакетов перед сетевой платой, по-видимому, не инициализировался правильно, то есть не очищался от остатков предыдущего применения. Код предполагает, что всё содержимое равно нулю, и не трогает области, которым не нужно определённое значение, как ACK и URG без соответствующих флагов. В результате часть остатков уходит в провод.

Естественно, все остальные поля IP и TCP правильно инициализировались; пропущены лишь несущественные в данном контексте URG и ACK. Но это означало, что малая порция данных другого соединения или иной стороны работы компьютера отправлялась посторонней стороне.

Проблема проявлялась только при нескольких сеансах, распространённых во время просмотра веб-страниц, фоновой загрузки и подобных сценариев, но не в бездействующей системе.

Раскрытые сведения имеют двойное значение:

• Это традиционный сценарий раскрытия информации. Объём в каждом пакете с неверно инициализированными URG и ACK мал и не гарантированно осмысленен, если буфер изначально не содержал интересного, но в отдельных случаях ценен, особенно когда внешняя сущность способна вызвать одновременный сеанс с конфиденциальными сведениями и сам недостаток.

• Уязвимость служит удобным показателем отпечатка, раскрывающим дополнительные сведения об ОС и её состоянии, простым способом отличить систему, интенсивно использующую сеть, от бездействующей.

Вот и всё. Хотя значение открытия легко переоценить, я включил его ради занятности и чтобы показать, насколько просто получить даже сложные данные удалённой стороны, ни о чём её не прося.

Пища для размышлений

Легко обвинить разработчиков. Они, разумеется, виноваты в неправильной инициализации памяти, но само наличие отдельного «включателя» для поля заголовка, возможно, является недостатком конструкции TCP и способствует таким проблемам. Подобные тонкости преследуют спецификации протоколов, как показано в главе 7, где сходную уязвимость вызвало слишком точное следование спецификации без достаточного размышления о побочных последствиях.

Глава 13. Дым и зеркала

Или как изящно исчезнуть

Многие рассмотренные сценарии раскрытия требуют внимательного анализа сведений удалённой системы, чтобы вывести факты об отправителе или перехватить дополнительные данные, об отправке которых он не подозревает. Но иногда удаётся собрать лишь косвенные свидетельства некоторой активности. Как обсуждалось в главах 1 и 2, точное толкование позволяет установить вероятное местонахождение пользователя или приложения с конфиденциальными данными, косвенно раскрывая тайны машины жертвы без доступа к самим данным.

Некоторые свойства IP делают многие реализации уязвимыми к раскрытию через косвенные свидетельства, очень похожему на увиденное у отдельных генераторов псевдослучайных чисел или алгоритмов обработки с переменной сложностью. Внимательное наблюдение и расшифровка дают преимущество и как минимум крайне необходимые разведывательные сведения об общих привычках противника или конкретной деятельности.

До сих пор эта часть книги была посвящена атакам уровня IP, требующим прямого наблюдения трафика отправителя, хотя обычно без взаимодействия с жертвой. Здесь мы взглянем на эффектно активную, но косвенную атаку IP, где атакующий профилирует жертву обоснованным предположением о невидимом. Он взаимодействует с невинным наблюдателем, который не является настоящим объектом испытания, без его согласия и ведома узнавая всё возможное о реальной жертве.

Такой подход не звучит лёгким способом сбора данных. Но в духе фанатского сообщества почему бы не выбрать живописный, пусть немного более длинный маршрут и не рассмотреть его подробно?

Злоупотребление IP: расширенное сканирование портов

Злонамеренные пользователи Интернета часто сканируют порты для разведки перед атакой и снятия отпечатков. Потенциальный атакующий пытается кратко подключиться к каждому порту системы и составляет карту программ, слушающих сетевой трафик. Так он находит уязвимую или потенциально интересную службу и определяет, куда атаковать. Во многих случаях можно узнать ОС жертвы, поскольку службы по умолчанию часто специфичны для системы.

Первая проблема традиционного сканирования в его шумности: жертва, вероятно, заметит бурю или устойчивый поток попыток к необычным портам. Скрыться тоже нелегко; атакующий должен видеть ответы на SYN, чтобы различать открытые и закрытые порты. Открытые отвечают SYN+ACK, закрытые - RST, а отфильтрованные экраном, скорее всего, ничего не посылают либо возвращают Internet Control Message Protocol, ICMP. Поэтому нельзя просто подделать исходный адрес всех пакетов: приходится раскрыть личность адресом, ведущим обратно в сеть, где атакующий слушает входящий трафик.

Дерево в лесу: как спрятаться

Сканирует ли сторона из любопытства, например чтобы узнать ОС конкурента, или продолжает попыткой атаки, обычно она хочет оставить минимум следов и не насторожить жертву. Сетевые администраторы и некоторые органы обычно крайне отрицательно воспринимают сканирование хостов и сетей. Споры о его злонамеренности продолжаются, но зондирующий почти всегда проигрывает, если раздражённый администратор жалуется на злоупотребление или конкурент узнаёт сотрудника, исследующего его сети, независимо от настоящего намерения и дальнейших планов любопытного испытателя.

Обычный способ маскировки - сканирование с «приманками», где атакующий посылает каждому порту SYN от нескольких ложных адресов вместе с настоящим IP. Жертва обрабатывает подделки как реальные, но ответы, конечно, уходят в пустоту. Определить источник гораздо труднее: все системы-приманки нужно исключить из списка тщательным анализом или перебором. При решительности отправителя всё же можно найти без властей, но атакующий надеется обескуражить жертву чрезмерными затратами времени на столь малое происшествие.

Сканирование через бездействующий хост

Окончательная защита от обнаружения появилась, как часто бывает, от парня с избытком времени, потраченного на чтение спецификаций вместо продуктивного занятия. Так родилось «бездействующее» сканирование, idle scan. Сальваторе «antirez» Санфилиппо придумал его в 1998 году; вскоре оно было широко реализовано и стало популярно среди и просто любопытных, и злонамеренных хакеров.[1]

Основа - важное наблюдение. Процитируем RFC 793:

> Как общее правило, сброс, RST, должен отправляться всякий раз, когда приходит сегмент, очевидно не предназначенный текущему соединению. Сброс не должен отправляться, если это неясно.

Пакеты RST TCP безусловно завершают соединение и приказывают отправителю прекратить дальнейшие попытки связи. Согласно RFC 793 система без долгих колебаний посылает RST на

неожиданный трафик. Естественно, сами RST, даже неожиданные, ответа не получают, иначе при малейшем сбое между сторонами бесконечно отскакивал бы поток сбросов.

Idle scan умно злоупотребляет тем, что посторонний хост-свидетель так обрабатывает все неожиданные пакеты. Злонамеренный сетевой гражданин сканирует жертву, с которой не собирается непосредственно связываться, посредством ничего не подозревающей случайной системы Интернета и не раскрывает личность.

Работает это так. Атакующий подделывает SYN к проверяемому порту жертвы. Пакет адресован ей, но обратным адресом вместо атакующего указан свидетель. Само по себе это не кажется полезным, но подождите.

Дальнейшее зависит от открытости порта:

• Если проверяемый порт жертвы отвечает свидетелю RST, тот получает его и молча размышляет, не создавая обратного трафика.

• Если порт открыт, жертва отвечает SYN+ACK. Свидетель с величайшим недоверием заключает, что вообще не посылал SYN, и отправляет RST, сообщая жертве, что она жестоко ошиблась и ей лучше немедленно остановиться. Жертва смущённо принимает ответ и удаляет все записи о соединении, которое надеялась принять.

Значение различия поначалу трудно оценить. Но вернёмся к главе 9 и вспомним поле IP-заголовка:

> Идентификационный номер, ID, - 16-битное значение, различающее IP-пакеты при фрагментации. Без IP ID одновременная фрагментация двух пакетов привела бы к сильному смешению, перестановке или иному повреждению их фрагментов при сборке. IP ID однозначно различают несколько буферов разных пакетов. Значение часто выбирается простым увеличением счётчика при каждой отправке: первый пакет системы имеет IP ID 0, второй - IP ID 1 и так далее.

Выбрав свидетеля с такой схемой IP ID, а кандидатов много, атакующий легко определяет, посылал ли тот IP-пакет за период. Для этого до и после настоящего зонда он отправляет свидетелю бессмысленный трафик и сравнивает ID ответов. Если два значения отличаются ровно на 1, между ними свидетель ничего не посылал. Если больше - пакеты обменивались, хотя неизвестно с кем.

Можно зондировать непосредственно перед поддельным пакетом жертве и вскоре после него и по ответам свидетеля определить порт. Увеличенный IP ID означает вероятный RST жертве, а значит, жертва сначала послала SYN+ACK на подделку и порт открыт. Если свидетель выдаёт ожидаемый следующий ID, он не получил трафик от жертвы либо проигнорировал RST.

Разумеется, есть практические соображения. Главное: свидетель должен быть сравнительно бездеятельным, а испытание следует повторить несколько раз для исключения ложных срабатываний. Иначе стороннюю связь свидетеля можно ошибочно принять за открытый порт жертвы.

> **Примечание.** Однако ни одна проблема не оказалась серьёзной, и многие совершенные средства, начиная с `idlescan` в 1999 году и теперь включая изобретательный NMAP, хорошо реализуют idle scan.

Важность подхода в том, что он скрывает происхождение не простым обескураживанием жертвы, а фактическим отсутствием опознаваемой связи атакующего. Отследить его без владельца свидетеля, который сам может получать запросы IP ID как часть законного трафика вроде HTTP и с трудом понять, что использован в атаке, или внешних организаций, правоохранительных органов и провайдеров, гораздо сложнее. Поскольку органы обычно действуют только после компрометации, а не простого зондирования, любопытные конкуренты могут спать спокойно, и требуют от жертвы признать компрометацию, что крупным корпорациям не всегда удобно, атакующий чувствует себя

довольно безопасно.

> **Примечание.** Хотя результаты поначалу кажутся неотличимыми от обычного SYN-сканирования, свидетель даёт уникальную перспективу: систему назначения видно с его точки зрения. Если свидетель имеет повышенные права к жертве, например находится внутри защищённой экраном сети или для его адреса установлены слабые правила ради удобного доступа к корпоративной сети, idle scan раскрывает внутреннее устройство защищённой сети.

Защита от idle scan

Непосредственной защиты сейчас нет, и легко отличить его от обычного SYN невозможно. Но не стать свидетелем просто: используйте случайные или постоянные IP ID, как обсуждалось в главе 9. Это не усложнит атаки на вас или вообще, поскольку множество систем всегда будет применять последовательные идентификаторы, но не позволит злоупотреблять вашей сетью с данной целью.

Чтобы избежать обхода экрана атакой «перспективы», руководствуйтесь здравым смыслом при проектировании каналов доступа внешних систем и применяйте правильную входную фильтрацию на шлюзах, отбрасывая пришедшие из Интернета пакеты с исходными адресами защищённой сети. Как отмечалось ранее, это может сломать обнаружение PMTU, но обычно исправляет больше проблем, чем создаёт.

Пища для размышлений

Хотя это менее осуществимо, IP ID всё же можно использовать для общего профилирования активности IP. Когда жертва устанавливает интерактивный сеанс с удалённой системой, по ID даже измеряется время нажатий клавиш или подобных действий, превращая приём в один из уже рассмотренных сценариев атаки по времени. Аналогично возможности отслеживания пользователей усиливаются измерением числа пакетов определённого хоста между двумя посещениями наблюдаемой сети.

На некоторых системах ту же функцию выполняют номера последовательности TCP в зависимости от конструкции генератора ISN. Предлагаю исследовать идею подробнее.

О поиске источника idle scan или другой поддельной атаки см. главу 17.

Глава 14. Идентификация клиента: предъявите документы!

Умение видеть сквозь тонкую маскировку может пригодиться во многих случаях

Задачу определения истинной личности программного обеспечения и его законности довольно легко решить локально на компьютере, где оно работает. Но по сети это не так просто.

И системные администраторы, и разработчики приложений часто пытаются с разной успешностью определить программу на другом конце сетевого сеанса. Причин несколько. Для World Wide Web, WWW, самая распространённая цель - оптимизировать отдаваемое клиенту содержимое под используемый механизм отображения, независимо от того, законно оно или вредоносно. В многочисленных других схемах связи, мгновенных сообщениях, почтовых клиентах и так далее, задача - обеспечить соблюдение политики и обнаружить связь из возможно опасных или неприемлемых приложений. И наконец, сами программисты пытаются не допустить

неразрешённые или нелицензионные программы к сетевой службе, возможно лишающие их части дохода, либо обнаруживать такие случаи и принимать исправительные меры.

Самый простой и распространённый способ опирается на сведения, добровольно объявляемые удалённой системой. Можно просто заметить приветственный баннер сервера, изучить заголовки протокола клиента, например `X-Mailer` в письмах или `User-Agent` в WWW, либо анализировать текстовые состояния, ошибки и предупреждения службы в ответ на трафик.[1] К сожалению, первый подход крайне ненадёжен и легко саботируется пользователями, которым есть что скрывать; последний навязчив и трудно применяется против клиентов без проблем. Большинство клиентских программ прекращает работу и жалуется при первой ошибке, а пользователь, который из-за попытки идентификации получает сообщение и не может законно обратиться к службе, не будет впечатлён.

Камуфляж

Изучение текстовых объявлений клиента ненадёжно не только потому, что пользователи могут маскировать интернет-программы, браузеры, почтовые клиенты и прочее, имитируя ответы самых популярных, но и потому, что часто имеют веский стимул: слиться с толпой или просто обмануть серверы, считающие, что лучше знают требуемую посетителю версию. Это легко делается встроенной функцией клиента либо изменением исходников или двоичного файла одним из множества свободно доступных средств.

Кроме того, многие корпоративные среды начали строже фильтровать содержимое для блокировки нежелательного трафика, и авторы сомнительных приложений в ответ стали выдавать их за безобидные. Не так давно одноранговые программы обмена музыкой, вредоносные трояны и шпионские программы начали изображать в исходящей связи самый распространённый браузер Microsoft Internet Explorer. Так же поступали многие веб-роботы, собиравшие адреса для сомнительных маркетинговых компаний по всему миру.

И другие протоколы страдают от самозванцев. Неудивительно, что большинство ненавистных программ массовой рассылки спамеров и мошенников притворяются Microsoft Outlook, PINE, Mutt, Eudora, The Bat! или Netscape Mail. Основная идея - спрятаться в камуфляже и проскользнуть мимо сетевых администраторов, которые, узнав о программе, легко её заблокируют. Ни один вменяемый спамер не объявит, что письма исходят от «Пресловутого массового рассыльщика дядюшки Берни, экстремальная редакция», ведь пользователь или фильтр слишком легко их отбросит.

Подход к проблеме

Поскольку простые текстовые ответы и баннеры программы изменить элементарно, для достаточно точной идентификации нужен лучший способ обнаружения обмана, чем сравнение текстов. Решения, просто проверяющие менее очевидные параметры или ответы, рано или поздно обречены: почти всегда можно придумать одну проверку конкретного нежелательного ПО, но на месте отрубленной головы вырастут три.

Вскоре пытаться учитывать каждое воплощение вредоносной программы становится непрактично. Иногда общее обнаружение достигается поиском схем, явно указывающих на предотвращаемое злоупотребление: отличие законного почтового клиента от программы спамера в том, что первый едва ли попытается отправить за раз 10 000 000 писем. Но подход очень ограничен. Для некоторых протоколов и чётко заданных атак он работает как волшебство, а с трафиком WWW всё иначе: трудно попасть в цель без чрезмерного числа ложных срабатываний или пропущенных программ.

WWW воспринимается основой всех интернет-служб конечного пользователя и является одним из немногих протоколов, которые просто обязаны быть открыты почти всем. Поэтому непослушные приложения чаще всего маскируют своё поведение и передаваемые удалённому хосту данные под веб-трафик. Браузеры нередко создают всплески соединений с разными сайтами или выполняют тысячи запросов в час. Одновременно конфиденциальные сведения можно отправить одним кратким соединением. Здесь профилирование трафика совсем немного не дотягивает до ответа.

На пути к решению

После всего сказанного различить шпионскую программу или троян и законное приложение кажется чрезвычайно трудно. Однако существуют хорошие средства точной идентификации такого ПО. Самый многообещающий универсальный подход обычно называется поведенческим анализом, модным обозначением старых и затасканных «временных закономерностей». Он исследует тонкие внутренние зависимости между последовательными порциями трафика вместо фактических данных одного запроса или общего числа соединений за время. Зависимости тесно связаны с внутренними алгоритмами и производительностью, поэтому их гораздо труднее подделать. В этой главе я рассмотрю подход и предложу основной набор аналитических средств для такой точности и подробности на удобном примере World Wide Web.

Но прежде нужны некоторые исходные сведения. Быстро взглянем на историю WWW, устройство веб-клиентов и протоколы разговора с серверами. Всё началось раньше, чем можно подумать...

Очень краткая история Всемирной паутины

Концепцию World Wide Web понять нетрудно: дать пользователям мгновенный доступ к множеству перекрёстно ссылающихся, связанных документов, сочетающих разные виды информации. Достаточно просто.

Современная Паутина состоит в основном из текста с метаданными, ссылками на другие файлы, элементами форматирования, аннотациями, динамическими или интерактивными элементами, часто дополненного мультимедиа: видео, музыкой и приложениями. Она представляет дух нашего времени и означает новый способ общения и поиска сведений. Но идея не нова. Она родилась задолго до технологии, позволившей реализовать возможности в электронных документах, возможно, прежде чем те вообще стали серьёзной возможностью.

Согласно хронологии,[1] опубликованной World Wide Web Consortium, W3C, концепцию гиперссылок впервые обсуждал в Atlantic Monthly[2] в 1945 году Вэнивар Буш, директор Office of Scientific Research and Development во время и после Второй мировой войны.

Буш предложил Memex, персональное электромеханическое устройство, которое можно считать ранним предшественником PDA. Оно хранило документы и личные файлы и должно было предоставить интуитивные механизмы доступа. Одна функция позволяла создавать ссылки между документами на микрофильме и следовать им. По какой-то причине идея безумно сложного механического устройства на микрофильме тогда не прижилась.

Концепция всплывала ещё несколько раз и привела к первым компьютерным реализациям в 1960-х. Они не были особенно успешны, главным образом потому, что вычислительная мощность, необходимая для привлекательности технологии, оставалась делом будущего.

Нужное время наступило в конце 1980-х. После бума микрокомпьютеров и незадолго до лобового штурма платформы PC в Conseil Europeén pour la Recherche Nucléaire, CERN,[2] ходило несколько скромных предложений о гиперссылках. Исследователя Тима Бернерса-Ли по общему мнению

официально следует винить в появлении HyperText Markup Language, HTML, набора средств включения метаданных, ссылок и мультимедийных ресурсов в текстовые документы. По правде говоря, HTML, основа знакомой Паутины, едва ли полностью новая конструкция и заимствует идеи у SGML, ISO 8879 Standard Generalized Markup Language 1986 года. Вскоре после этого на ныне почти неизвестной, но тогда передовой платформе NeXT появился первый браузер с вездесущим названием World Wide Web.

После появления броского названия революцию было не остановить. В 1992 году Бернерс-Ли подал первоначальный проект спецификации[3] HyperText Transfer Protocol, HTTP, средства инкапсуляции HTML и других ресурсов в связи от сервера клиенту. К 1993 году было доступно несколько механизмов браузеров, а горстка веб-серверов уже отдавала содержимое любопытным посетителям. Конечно, HTTP составлял лишь сокрушительные 0,01% магистрального трафика, но рос!

Первый популярный браузер Mosaic разработали в National Center for Supercomputer Applications Иллинойсского университета. Он заимствовал код Бернерса-Ли, но добавил содержимое помимо текста, заполняемые формы и многие другие привычные функции. Код Mosaic постепенно превратился в Netscape Navigator, затем отделился в открытый проект Mozilla, чья база позднее стала основой последующих поколений Navigator. Одновременно, чтобы ещё больше запутать пользователей, компания Spyglass превратила Mosaic в ядро будущего главного конкурента Netscape, Microsoft Internet Explorer.

В 1994 году образовался W3C, орган надзора за развитием Паутины. В 1996 году Бернерс-Ли, Рой Т. Филдинг и Хенрик Фристик представили первую официальную, значительно улучшенную и расширенную версию протокола, вскоре за ней вышла спецификация HTML 3.2. В последующие годы появились новые версии HTTP и HTML под управлением W3C. Конец истории всем известен. Или это только начало?

Введение в HyperText Transfer Protocol

HTTP[4] - удивительно прямолинейный текстовый протокол поверх TCP/IP. Клиент подключается к службе HTTP удалённого сервера и просит конкретный ресурс. Первая строка запроса включает:

• метод доступа. Чаще всего клиент просит получить файл посредством `GET`, хотя существуют методы отправки форм, диагностики, хранения данных на сервере или выполнения расширений;

• universal resource identifier, URI: путь к статическому файлу или динамической программе, являющейся объектом запроса. Динамической программе можно передать как часть URI дополнительные правильно закодированные параметры;

• поддерживаемую и желаемую версию протокола. Сервер может ответить более низкой, если версия клиента не поддерживается. Без этой информации предполагается ранний устаревший HTTP/0.9, который здесь не рассматривается.

Например:

```
GET /show_plush_toys.cgi?param1=value&param2=this+is+a+test HTTP/1.1
Host: www.plush-penguins.com
User-Agent: Joe's Own Web Client (UnixWare)
Accept: text/html, text/plain, audio/wav
Accept-Language: pl, en
Connection: close
```

Запрос просит ресурс `/show_plush_toys.cgi` у `www.plush-penguins.com`. Судя по расширению `cgi`, это динамически выполняемая программа с двумя параметрами, `param1` и `param2`, после вопросительного знака.

За запросом может, как в примере, следовать несколько текстовых заголовков по одному на строке с дополнительными параметрами. Это может быть идентификация клиента в `User-Agent`, предпочтительный язык содержимого, здесь польский и английский, или виртуальный сервер. Если несколько доменных имён указывают на один IP, заголовок позволяет понять, ищет пользователь `www.squeaky-ducks.com` или `www.plush-penguins.com`, размещённые на одной системе.

Некоторые заголовки обязательны согласно протоколу; набор зависит от версии, но большинство серверов довольно снисходительно к пропускам. Кроме того, часть заголовков задаёт функции за пределами самой спецификации.

Каждый запрос заканчивается пустой строкой, отмечающей конец заголовков клиента. Для большинства видов сервер после этого обрабатывает запрос и отвечает сходной структурой, начинающейся кодом возврата HTTP и описательным текстом:

```
HTTP/1.0 404 Not Found
Content-Type: text/plain
Server: Uncle Mary's Cookie Recipe Server (Linux and proud of it!)
Date: Mon, 09 Feb 2004 19:45:56 GMT

The document you are looking for is nowhere to be found.
```

Код или сообщение описывает успешное выполнение, указание браузеру искать в другом месте либо ошибку вроде «Файл не найден» или «Доступ запрещён». Далее идут заголовки сходного формата: версия серверной программы, следующее местонахождение для браузера, тип возвращаемого файла, отличающий изображения от текста или HTML от двоичных данных, и так далее. Затем, при наличии, само содержимое.

Как видно, базовый HTTP довольно прост. Он предлагает некоторые совершенные функции, но большинство либо слегка странны, либо редко применяются. Думаю, ошибку `402 Payment Required` вы видите не каждый день. Но наивно верить, что базового протокола достаточно современным пользователям.

Улучшение HTTP

Дни типичного сайта из нескольких килобайтов статического текста и малых графических элементов давно прошли. По мере усиления компьютеров и перемещения модемов 300 bps из домов в музеи форма начала преобладать над содержанием. Сотни килобайтов изображений, подстраниц, фреймов и клиентских сценариев с переменным успехом делают сайты привлекательнее и профессиональнее. На многих мультимедиа стало основным содержимым, а HTML лишь заполняет место для изображений, видео, встроенных Java-программ или игр. Паутина больше не просто способ рассказывать о личных проектах или интересах; движущая сила - продажа товаров и услуг дешевле и быстрее, чем когда-либо. А маркетинг требует броской подачи.

Браузерам, серверам и HTTP пришлось приспособиться, чтобы облегчить новые технологии и тенденции. Удобно, что многие появившиеся технологии имеют интересные последствия безопасности для простых смертных и помогают прозрачно идентифицировать клиента на другом конце. Поэтому следует рассмотреть необязательные функции и расширения со дня рождения Паутины.

Снижение задержки: неприятная заплатка

Проблема Паутины и некоторых современных протоколов в том, что содержимое одной мультимедийной страницы пользователь получает из разных источников, включая совершенно отдельные домены, а затем объединяет. Текст и форматирование отделены от изображений и

других крупных лакомств - практика, достойная похвалы со стороны людей с ограниченной полосой, желающих сразу перейти к сути.

Поэтому клиенту нужно несколько запросов. Самый наивный способ - получать части по очереди, но в реальности это создаёт узкие места: зачем ждать страницу только потому, что сервер баннеров медленный? Ради скорости браузер выдаёт несколько запросов одновременно.

Здесь первый недостаток HTTP: он не умеет изначально обслуживать одновременные запросы, только последовательные.

Последовательная модель заметно снижает производительность, если элемент страницы загружается с медленного сервера или нестабильной линии либо долго готовится. При единственном варианте любой медленный запрос не даёт отправить и обслужить последующие до завершения.

Поскольку новые HTTP ситуацию не исправили, клиенты применяют заплатку: браузер открывает несколько одновременных отдельных сеансов TCP/IP с сервером или набором и выдаёт много запросов сразу. Когда ресурсы находятся на отдельных машинах, решение вполне разумно. Но для одной системы, где все запросы можно было сделать в одном сеансе и разумно управлять сервером, оно плохо:

• Сервер не может выбрать лучший порядок. Иначе он оставил бы трудоёмкие, крупные или наименее важные объекты напоследок. Его заставляют делать почти всё сразу, и важнейшее всё равно без необходимости задерживается из-за нагрузки CPU.

• При одновременной отдаче крупных ресурсов и переключении планировщика между сеансами производительность серьёзно падает из-за постоянного быстрого поиска диском между двумя, возможно далёкими файлами.

• Новое рукопожатие TCP/IP даёт значительные расходы, хотя в новых HTTP это несколько смягчается keep-alive. Эффективнее выдать все запросы в одном соединении.

• Новый сеанс и процесс обработки создают расходы ОС и нагружают устройства вроде экранов с состоянием. Современные серверы уменьшают проблему запасными постоянными процессами для поступающих запросов, но редко устраняют полностью. Один сеанс избегает расходов и позволяет выделить лишь действительно необходимые ресурсы для асинхронного обслуживания выбранных запросов.

• Наконец, если узкое место - сеть, а не сервер, производительность может ухудшиться из-за отбрасывания пакетов, когда канал насыщается данными нескольких одновременных источников.

Увы, хороша эта архитектура или плоха, пока она с нами и всё же лучше последовательного получения. Следует признать её существование и научиться использовать в своих интересах.

Как свойство поможет определить программу клиента? Очень просто. Значение параллельного получения файлов для отпечатка браузера достаточно очевидно: двух совершенно одинаковых алгоритмов нет, а измерить их можно хорошими способами.

Но прежде рассмотрим ещё две важные части уравнения безопасности и приватности Паутины: кэши и управление идентичностью. Хотя они кажутся не связанными, в итоге образуют логическое целое. Краткий антракт.

Кэширование содержимого

Хранение локальных кэшей документов сервера - одна из важнейших функций Паутины в период быстрого расширения последних лет.[3] Без неё стоимость этого дела была бы значительно выше.

Растущий вес и сложность типичного сайта требуют всё больше полосы, которая для компаний остаётся довольно дорогой, и лучших серверов для разумной скорости.

Если производительность не ограничена полосой, решения с одновременными сеансами дополнительно нагружают поставщиков. Причина может удивить: если человек на медленной линии, например модеме, открывает четыре последовательных сеанса даже для простой страницы, серверу приходится сохранять четыре соединения и четыре процесса или потока, отбирая ресурсы у более быстрых клиентов.

Наконец, тяжёлые сложные сайты не всегда согласуются с ожиданиями. Сравнительно долгое время загрузки, прежде считавшееся приличным, теперь раздражает и отпугивает. Исследования показывают, что средний пользователь не ждёт более 10 секунд.[5] Корпорациям и поставщикам нужны дополнительные ресурсы и лучшие линии. Если бы всё осталось в первоначальном виде, спрос на серверные ресурсы, вероятно, давно превысил бы наши возможности.

Помогает то, что отдаваемое содержимое статично или меняется редко по сравнению с частотой получения, особенно крупные файлы: графика, видео, документы, программы и так далее. Кэшируя ближе к пользователю на уровне провайдера или самого браузера, мы резко сокращаем полосу последующих посещений с общим механизмом кэша и облегчаем работу серверов. Провайдер тоже выигрывает от меньшего потребления и обслуживает больше клиентов без нового оборудования и линий. Но HTTP нужен способ сохранять точность и актуальность кэша. Автор страницы, человек или машина, должен сообщать, когда получать новую версию.

Для кэширования HTTP предлагает две встроенные функции:

• способ с минимальными усилиями узнать, изменилась ли порция данных после самой свежей версии в кэше, документа последнего посещения;

• способ определить, что не следует кэшировать из-за безопасности или динамического создания при каждом запросе.

На практике всё просто. Сервер возвращает кэшируемые документы в обычном сеансе с дополнительным заголовком `Last-Modified`, который, неудивительно, представляет его мнение о времени последнего изменения. Некэшируемые отмечаются `Pragma: no-cache`, а в HTTP/1.1 - `Cache-Control: no-cache`.

Браузер или промежуточный механизм провайдера должен сохранить копию каждой кэшируемой страницы вместе со временем изменения и держать её как можно дольше: пока не превышен пользовательский предел или пользователь вручную не очистит кэш, если только `Expires` не приказывает отбросить после даты.

При повторном посещении клиент обнаруживает предыдущий экземпляр на диске и действует немного иначе. Пока документ живёт в кэше, при каждом возвращении клиент пытается получить файл, но указывает `If-Modified-Since` со значением ранее увиденного `Last-Modified`. Сервер сравнивает его со временем последнего изменения ресурса. Если с тех пор ничего не изменилось, вместо данных возвращается ошибка HTTP `304 Not Modified`. Передача подавляется, экономя полосу: обмениваются лишь пара сотен байтов, а клиент или промежуточный кэш использует прежнюю копию.

> **Примечание.** Более современный подход, заголовки `ETag` и `If-None-Match` из маркировки сущностей HTTP/1.1, действует сходно, но устраняет неоднозначность времени изменения: файл может несколько раз меняться за короткий период ниже разрешения часов `Last-Modified`, восстанавливаться из резервной копии со временем старше кэшированной и так далее.

Управление сеансами: cookie

Другое важное и кажущееся не связанным требование к HTTP - различать и отслеживать сеансы между соединениями, хранить настройки и идентичность. Например, сайтам полезно приспосабливаться к предпочтениям и восстанавливать выбранный вид при каждом посещении.

Конечно, личность можно устанавливать запросом имени и пароля на каждой странице и затем загружать настройки, но лишнее усилие резко уменьшает число готовых пользоваться сайтом людей.

Для бесшовного персонализированного доступа к форумам, доскам, чатам и множеству других привычных функций требовался прозрачный постоянный способ хранить и получать сведения на машине клиента. Но способность администратора узнавать вернувшихся посетителей, назначая уникальную метку и позднее извлекая её, означала сдачу анонимности за небольшое удобство. Механизм дал бы компаниям со второсортной этикой превосходный инструмент слежения и профилирования: записывать покупки и просмотр, определять интересы. Поисковики легко связывали бы запросы одного человека, а поставщики баннеров отслеживали бы людей без их разрешения или ведома операторов сайтов.[4] Но достаточно универсальной альтернативы не было. Так родились веб-cookie.

Cookie согласно RFC 2109[6] - малые порции текста, выдаваемые сервером при подключении клиента. В ответе указывается `Set-Cookie`; дополнительными параметрами текст ограничивается конкретным доменом, сервером или ресурсом и сроком. Клиент хранит его в специальном файле или папке, cookie jar, и автоматически возвращает заголовком `Cookie` при новом соединении с ресурсом.

Сервер может хранить, выталкивать, пользовательские настройки в `Set-Cookie` и читать при последующих посещениях; в совершенном мире на этом функция закончилась бы. Но компьютер не знает, что хранится внутри. Сервер может назначить уникальный идентификатор и позднее связать текущую деятельность с предыдущими действиями.

Механизм широко считается серьёзной угрозой приватности. Некоторые активисты прямо ненавидят cookie, хотя теперь сопротивление всё тише. Просматривать Паутины без них всё труднее, а некоторые сайты отказывают клиентам, не прошедшим проверку. К счастью, многие браузеры предлагают подробные настройки принятия, ограничения и отклонения и даже спрашивают о каждом, хотя последнее непрактично. Это позволяет разумно защищать приватность, хотя бы определив «хороших парней» и доверенных.

Но находится ли приватность после этого в наших руках?

Когда смешиваются cookie и кэши

Приватность веб-просмотра давно является горячей темой не без причины. Многие не хотят, чтобы кто-либо подсматривал предпочтения и интересы, даже не особенно сомнительные. Иногда просто не хочется, чтобы сомнительная рекламная компания знала о чтении конкретного медицинского состояния и связала это с учётной записью профессиональной доски, особенно когда неизвестно, куда попадут сведения.

Управление cookie сохраняет удобство и держит плохих парней на расстоянии. Но даже отключение не мешает хранить сведения в системе и позднее отправлять серверу. Необходимая функция давно присутствует во всех браузерах независимо от политики. Две технологии действуют сходно и отличаются лишь назначением: cookie и файловый кэш.

Примерно в 2000 году Мартин Пул отправил в Bugtraq короткое, но проницательное сообщение[7] с наблюдением и кодом. Он заключил, что между `Set-Cookie`/`Cookie` и `Last-Modified`/`If-Modified-Since` нет существенной разницы, по крайней мере без централизованного прокси-кэша, когда копии хранятся локально на диске, как у большинства простых смертных. Злонамеренный администратор может поместить в `Last-Modified` почти любое сообщение или, если заголовок проверяется, произвольную уникальную дату для идентификации посетителя. При возвращении клиент отправит `If-Modified-Since` с точной копией сохранённого злоумышленником идентификатора. Ответ `304 Not Modified` не даст этому «cookie» исчезнуть.

Предотвращение атаки cookie через кэш

Небольшое изменение браузером данных `Last-Modified` в ответе кажется аккуратной защитой ценой некоторой неточности кэша, но это не так. Другой вариант хранит данные в кэшированных документах вместо меток. При первом посещении злоумышленник готовит особую страницу со ссылкой на уникальное имя встроенного ресурса, например изображения. При возвращении сервер видит `If-Modified-Since`, отвечает 304 и заставляет использовать старую копию. Она содержит уникальную ссылку, запрашиваемую у сервера, благодаря чему IP клиента сопоставляется предыдущему сеансу, где было возвращено имя. Упс.

Срок «cookie» ограничен размером кэша и настройками истечения. Но значения обычно щедры, а сведения в метаданных ресурса, посещаемого раз в пару недель, живут годами до ручной очистки. Для компаний, отдающих общие компоненты сотням или тысячам сайтов, снова хороший пример - баннеры, это не проблема.

Главное отличие от настоящих cookie не в функциях, а в лёгкости управления уязвимостью. Данные кэша служат и другим целям и не могут быть легко ограничены без большого падения производительности при частичном или полном отключении.

В причудливом повороте две стороны Паутины сталкиваются, фактически сводя на нет защиту одной. Практика показывает, что намерений недостаточно: злоумышленники не обязаны играть по правилам и применять технологию желаемым способом. Возможно, отключение cookie всё же мало что меняет?

Но пора вернуться к основной теме.

Раскрытие предательства

То есть к обнаружению обмана и точному снятию отпечатков клиента. Я уже упоминал, что задача сложна, но не невозможна, а поведенческий анализ, внимательное наблюдение последовательности событий браузера, заслуживает исследования.

HTTP особенно щедр как предмет, поскольку большая часть активности идёт параллельно или почти параллельно, а точные алгоритмы очередей и обработки тонки и уникальны для клиента. Измеряя число одновременно загружаемых файлов, относительные задержки запросов, порядок и другие детали сеанса, можно получить характеристики на уровне, который пользователю гораздо труднее изменить, и без усилий отличить самозванцев от законопослушных граждан.

Для простейшего реального примера я решил выяснить, что можно сказать по уже существующим ограниченным данным, которые, вероятно, у многих под рукой, и взял стандартные журналы чуть более миллиона запросов к достаточно популярному сайту. Это обычный журнал Apache со временем завершения, URI, объявленными данными `User-Agent` и другими основами. Страница состоит из набора сравнительно малых картинок похожего размера и одного HTML, вызывающего

их все.

Простейший случай поведенческого анализа

Практика Apache записывать запросы после завершения, а не при выдаче, может казаться проблемой, но при достаточно однородных файлах помогает. Порядок начала обычно сильнее зависит от последовательности ссылок главной страницы, тогда как время завершения сложнее.

Вероятности порядка завершения зависят от числа запросов, задержек и других параметров, тонко, но заметно различающихся у браузеров. В частности, клиент с одним соединением всегда выдаёт `A-B-C-D`; с тремя одновременными быстрыми запросами с равной вероятностью получается `B-A-C-D`, `C-B-A-D`, `C-A-B-D` и так далее, и тогда особенно важны очередь и управление сеансами.

Естественно, наблюдаемая последовательность сильно зависит и от задержки, надёжности и случайных сетевых явлений. Но в столь большом наборе специфические не для браузера эффекты разумно считать усредняющимися или одинаково влияющими на всех. Тогда мы, надеюсь, увидим тонкие различия под дружелюбным интерфейсом.

Рисунок 14-1 показывает статистическое распределение попыток загрузить описанную десятиэлементную страницу четырьмя самыми популярными клиентами набора. Каждый график разделён на десять крупных сегментов. Первый соответствует непосредственно запрошенному главному HTML и естественно является первым элементом. Остальные девять - девяти изображениям по порядку ссылок в HTML.

Каждый сегмент дополнительно делится на десять дискретных позиций оси X, не показанных явно во избежание загромождения. Высота в позиции `n` представляет вероятность загрузки данного файла элементом `n` последовательности.

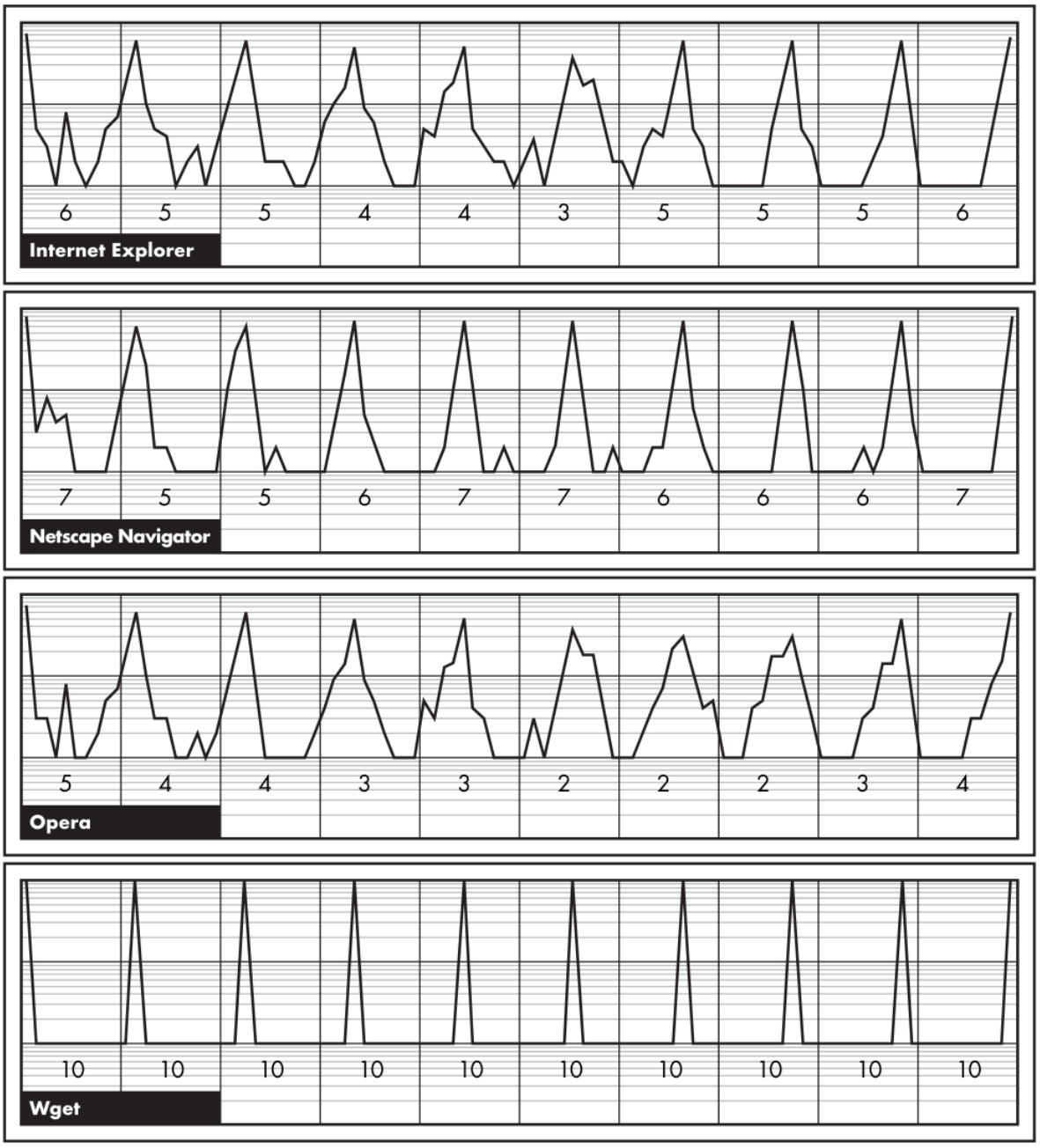


Рисунок 14-1. Различия поведенческих шаблонов популярных веб-клиентов. Показаны Internet Explorer, Netscape Navigator, Opera и Wget; подписи под ними дают десятизначные отпечатки поведения.

Для читаемости вероятности даны процентами от 1 до 100, все меньше 1 округлены вверх, а дискретные точки соединены линиями. Графики построены в логарифмическом масштабе `log10` с основными направляющими 1, 10, 100, чтобы тонкие признаки были заметнее и легче сравнивались.

В совершенном мире с полностью последовательными предсказуемыми браузерами первый сегмент имел бы пик лишь в первой, крайней левой позиции, второй - только во второй и так далее. На практике некоторые выдают много запросов сразу и порядок легко перемешивается: третий файл может загрузиться раньше второго или после четвёртого. Чем менее выражен единственный пик сегмента, тем агрессивнее алгоритм, поскольку выше вероятность загрузки не по порядку.

Различия отчётливо видны даже между браузерами, исторически основанными на одном механизме, Mozilla и Internet Explorer. Все соблюдают порядок ссылок главного документа, поэтому последовательные пики медленно движутся слева направо. Но Mozilla заметно менее нетерпелив и чаще заканчивает в порядке запросов. Opera, рекламируемая самым быстрым браузером на земле, гораздо менее последовательна: у многих файлов два-три почти одинаковых пика, то есть запросы выдаются настолько быстро, что завершение почти произвольно и главным образом зависит от дрожания сети. Для сравнения открытый робот Wget идеально последователен, как часто автоматические роботы, использует одно соединение и всегда загружает в одном порядке.

Придание смысла красивым картинкам

Картинки и графики приятны, но почти бесполезны для автоматического обеспечения политики или обнаружения злоупотребления. Чтобы количественно выразить схемы и сделать отпечатки реалистичнее, я ввёл простой показатель, дающий сегменту лучший балл от 0 до 10 при одном пике и меньший при произвольном распределении. Так можно создать десятизначный отпечаток программы и сопоставлять активность с подписями.

Для относительного качества, линейности, `Q` наблюдаемого поведения в крупных сегментах я использовал следующую формулу, где `fn` обозначает вероятность появления файла в позиции `n` последовательности, выраженную процентами для удобства и раздражения пуристов:

```
Qs = 1.42 * (sqrt(sum(n=1..10, fn^2) / 10) - 3)
```

Хотя уравнение поначалу пугает, оно прямолинейно. Я хотел отдать предпочтение случаю, когда файл чаще всего загружается в одной фиксированной позиции, одно `f` близко к 100%, остальные к 0, перед случаем, когда все позиции равновероятны по 10%.

Поскольку сумма всех элементов `f` фиксирована и равна 100%, проще всего применить сумму квадратов: для любой последовательности ненулевых чисел она всегда меньше квадрата суммы. Наибольший и наименьший результаты:

```
10^2 + 10^2 + 10^2 + 10^2 + 10^2 + 10^2 + 10^2 + 10^2 + 10^2 + 10^2 = 1 000
100^2 + 0^2 + 0^2 + 0^2 + 0^2 + 0^2 + 0^2 + 0^2 + 0^2 + 0^2 = 10 000
```

Остальная математика лишь отображает результаты на разумную шкалу от 0 до 10 после округления.

Результаты вычисления для каждого сегмента каждого браузера наложены на рисунок 14-1 числом, описывающим сегмент. Как ожидалось, Wget везде идеален. Оценки остальных подтверждают визуальные наблюдения и делают их осязаемее. Хотя Internet Explorer и механизмы Mozilla/Netscape дают примерно сходные графики, сильные различия видны в загрузке элементов 4-6 и слабее по всей последовательности. Opera явно отделяется постоянно более низкими оценками.

Так простейший аналитический инструмент дал основу практического способа идентифицировать браузеры и обнаруживать обман в статистически значимой выборке HTTP-трафика пользователя. Модель можно улучшить анализом других автоматически загружаемых элементов, сценариев, таблиц стилей HTML, карт изображений, фреймов и файлов с ещё большим различием между браузерами. Санте в этом году может быть легче составить список непослушных пользователей.

За пределами механизма...

Я лишь надеюсь показать, как легко обнаружить скрытые свойства неизвестного приложения по поведению без предположений и разбора внутреннего устройства. Точные числа выше, вероятно,

неприменимы к сайтам кроме использованного, поэтому при потенциальном применении предлагается выполнить домашнюю работу. После профилирования сайта или набора данные эффективно распознают системы по схемам активности со временем.

Само собой, применённый метод, возможно чрезмерно, прост и основан на самом тривиальном сценарии; я привожу его как поощрение искать больше. В сложных случаях процессы отображения содержимого во фреймах, таблицах и других визуальных контейнерах либо получения и отображения особых файлов позволяют определить браузер даже без статистического сопоставления: в очень специфических аспектах различия гораздо ярче. Обещает успех и умное применение дифференциального времени.

И подумайте: более продуманные формы анализа можно продвинуть дальше, различая не механизмы отображения, а машины и людей или даже отдельных пользователей. Как обсуждалось в главе 8, манера работы с клавиатурой настолько уникальна, что пригодна для биометрии. Аналогично исследования показывают, что способы нажатия ссылок, выбора, чтения и прочего способны указать, кто или что стоит за запросами.[8] Пока это ближе к научному предположению, чем факту, но область замечательна для исследования и игры.

...и за пределами идентификации

Применения активности браузера и поведенческого анализа выходят за рамки обнаружения программы; некоторые вступают в область приватности и анонимности.

Интересная работа Эдварда Фелтена и Майкла Шнайдера 2000 года[9] вносит увлекательный вклад: возможность, тесно связанная с механизмами кэширования современных движков, где наконец встречаются все обсуждённые элементы.

Основная идея: вставив ссылку на файл определённого сайта и измерив задержку браузера при загрузке, можно узнать, посещал ли пользователь сайт в последние дни. Достаточно просто.

Вместо длинной экскурсии в теорию, предсказания и предположения, хотя бы один раз, предложу почти реальный пример. Допустим, я управляю `www.rogue-severs.com` и решил, что главная страница зачем-то ссылается на картинку, например логотип, с `www.kinky-kittens.com`. Я делаю элемент трудным для поиска или уменьшаю до невидимости, но браузер всё равно его загрузит.

Ничего не подозревающий пользователь посещает мой сайт. Если на `www.kinky-kittens.com` он не бывал, загрузка занимает время. У частого посетителя картинка уже в кэше и появляется почти мгновенно.

Поскольку ссылке предшествуют и следуют запросы других размещённых у меня элементов, умная эвристика времени надёжно определяет, загружался весь логотип или уже был в кэше. Этого достаточно, чтобы выяснить, является ли новичок частым посетителем конкретного сайта или раздела, грубо вторгаясь в приватность. Сценарий едва ли пригоден для повсеместного рутинного шпионажа, главным образом из-за явных следов, которые может заметить оператор сервера, за пользователями которого мы подсматриваем, но целевые атаки вполне эффективны.

В итоге части головоломки, пусть неплотно, но сходятся. Пользователи, программы и привычки легко раскрываются тщательным злоупотреблением современными функциями популярного интернет-протокола. Не обязательно утешительная мысль для уважаемых посетителей `www.kinky-kittens.com`.

Предотвращение

Полная анонимизация веб-просмотра кажется уже проигранной битвой. Хотя существуют общепринятые практики повышения приватности и анонимности, вредоносный сайт легко их обходит.

Проблема, к сожалению, слишком серьёзна, чтобы отвергать. Одно дело, когда о деятельности знает сущность, которой мы решили доверять, например провайдер; совсем другое - когда стороны, с которыми лучше не иметь дела, регулярно собирают конфиденциальные профили и, вероятно, столь же регулярно перепродают их как часть деловой модели. Этого достаточно для беспокойства даже тех, кто не носит каждый день шапочку из фольги и алюминиевое бельё.

С другой стороны, относительная трудность полной анонимности или совершенно безобидного вида важна в средах, где HTTP должен быть разрешён, но пользователей требуется защищать и контролировать, не нарушая приватность сверх необходимого. В корпоративных сетях способность отслеживать нарушающие системы без ручного изучения данных поистине бесценна и ценится и пользователями, и администраторами.

Пища для размышлений

Ни один компонент HTTP не является непродуманным, сломанным или неоправданным. Но вместе многие функции безопасности и приватности словно отменяют друг друга, оставляя пользователя достаточно открытым для свободно разгуливающих наблюдателей. Печально, что без начала с нуля можно сделать мало, и нет гарантии, что результат работал бы столь же хорошо или давал хотя бы столько же приватности, сколько нынешние HTTP, HTML и клиенты WWW.

Сноски

[1] [назад] Популярный инструмент, снимающий отпечатки анализом ответов, - AMAP от THC; подробнее: thc.org. NMAP Фёдора определяет службы по баннерам.
[2] [назад] Европейская лаборатория физики элементарных частиц, Женева, Швейцария.
[3] [назад] Однако значение постепенно уменьшается: всё больше страниц создаётся динамически, а магистраль Интернета становится зрелее и мощнее, поэтому кэширование неизбежно теряет важность.
[4] [назад] Если баннер или иной элемент размещён на общем сервере вроде http://banners.evilcompany.com, оператор `evilcompany.com` может выдавать и получать cookie при посещении любого законного сайта с его баннерами. Большинство поставщиков действительно это делает и отслеживает пользователей, хотя прежде всего для маркетинговых исследований.

Глава 15. Преимущества положения жертвы

Где мы заключаем, что должный оптимизм в отношении жизни помогает отследить атакующего

Я обсудил разнообразные проблемы, способные совместно значительно влиять на всю повседневную связь, и риски, с которыми нам не всегда удобно жить. Вы увидели, как другие эксплуатируют сеть для кражи сведений или получения большего, чем ожидается и разрешается, и как теми же приёмами собирать больше информации о корпоративной или домашней сети и атакующих её людях.

Надеюсь, я дал полезное представление о рождении таких проблем и способах по возможности избегать их. Я пытался показать, что безопасность и приватность являются частью любой деятельности и не устраняются полностью правильными проектными решениями, нужной программой или надлежащей политикой. Раскрытие сведений нельзя подавить целиком; единственная надежда - достаточно знать о возможных утечках и атаках, чтобы максимально

смягчать самые значительные в конкретном применении.

Третья часть книги была посвящена глобальным сетям и скрытым угрозам. Хотя это самая длинная часть и лишь теперь завершается, она дальше всего от полного представления всех проблем открытой сети. Обсуждать все варианты было бы очень трудно и почти бессмысленно, поэтому я выбрал наиболее сложные, трудные или увлекательные стороны связи между хостами. Я сосредоточился на обнаружении сценариев разных уровней протокола и абстракции вместо перечисления концепций и направлений, повторяющих старые идеи и ничего не добавляющих. Надеюсь, сведения помогут и вдохновят находить другие воплощения проблем в сетях, вычислениях и, возможно, за их пределами.

В следующей части произойдёт значительный сдвиг парадигмы: мы рассмотрим, как внимательное наблюдение за сетью в целом, а не отдельными системами, применяется для защиты себя или атаки других. Но прежде взглянем на иные возможности необычной области сетевого наблюдения: пассивную контрразведку, то есть изучение атакующего и его целей по действиям.

Собранные так данные дают мощные следственные зацепки для определения намерений, набора средств и даже личности атакующего. Построение профиля, попытка прочесть мысли и, возможно, игра в обман часто сами по себе захватывают.

Определение показателей атакующего

Как и ожидалось, о злонамеренной удалённой стороне можно узнать многое, просто применив к трафику распространённые показатели TCP/IP из предыдущих глав, например пассивное снятие отпечатка ОС. Так определяется конкретный инструмент сканирования портов.

Поведенческий анализ также применяется к задержкам между запросами и их порядку, например последовательности портов и скорости. С некоторым успехом он отслеживает программы или, при ручном взломе либо несанкционированной оценке, даже индивидуальные свойства атакующего, например владение компьютером.

Особенно интересный способ определить сканер переносит на новую задачу один метод главы 9, снятие отпечатка последовательности портов. Большинство современных сканеров проходит сети и системы от меньших портов или адресов к большим либо перемешивает порядок доступа. Последний подход применяется чаще и считается лучше, потому что выравнивает нагрузку и слегка затрудняет обнаружение. Но неожиданно случайность способна несколькими причудливыми способами обернуться против атакующего.

Проблема возникает, поскольку авторы не считают сетевые сканеры критическими приложениями с высокими требованиями безопасности. Самый обычный и лёгкий способ реализовать генератор псевдослучайных чисел без необходимости криптографически безопасного выхода - вызвать стандартные системные или встроенные средства языка. Стандарт ISO[1] самого распространённого в мире языка C предлагает простейший линейный конгруэнтный алгоритм для генератора стандартной библиотеки, рассмотренного в главе 1. Рецепт таков:

1. Генератор инициализируется начальным 32-битным значением `S0` через стандартную `srand()`. Без этого он начнёт с фиксированного значения и всегда выдаст одинаковую последовательность.

2. При каждом вызове `rand()`, получающем очередное псевдослучайное число для пользовательской программы, состояние пересчитывается: `S(t+1) = St * 1103515245 + 12345`. Результат обрезается до 32 битов, по модулю `4294967296`.

3. Возвращается старшее слово `S(t+1)` по модулю `32768`. В распространённом ныне 32-битном варианте процедура этого и предыдущего шага повторяется несколько раз для последовательных частей результата.

Все линейные конгруэнтные генераторы, включая этот, уязвимы к общей методике криптоанализа Х. Кравчика 1990-х, упомянутой в главе 1. По нескольким последовательным или иначе упорядоченным выходам восстанавливается внутреннее состояние и предсказываются все предыдущие и будущие результаты.

Непосредственное следствие, способность жертвы по предыдущим попыткам определить, в каком порядке атакующий нацелится на другие ресурсы машины или сети, само по себе не особенно увлекательно или ценно. Но для сетевых зондов есть два важных последствия:

• Можно определить `S0`. Если известно или оценивается время начала работы генератора либо общие свойства начального состояния, значение инициализации восстанавливается. Поскольку `S0` - единственный вход, одинаковое значение обязательно создаёт одинаковое поведение, и состояние прослеживается по выходу PRNG.

• Можно определить приращения `t`. После восстановления состояния видно, сколько случайных значений запросил сканер вызовами `rand()` между двумя вызовами, использованными для номеров портов или адресов хостов захваченных пакетов.

Важность первого последствия не сразу очевидна. Но есть ещё часть головоломки. Генератор обычно инициализируют удобным 32-битным значением, меняющимся достаточно часто, чтобы одинаковое поведение не повторялось. Для этого часто берут системное время, иногда сочетая с небольшим числом вроде текущего идентификатора процесса, PID, чтобы две программы, запущенные близко, реже давали сходные результаты.

Применив знание к рассчитанному `S0`, жертва зонда узнаёт системное время атакующего, GMT или локальное в зависимости от ОС и сканера. Локальное время тривиально подсказывает происхождение и личность. Если нас пытаются запутать поддельными пакетами разных источников, можно исключить те, где часовой пояс `S0` не совпадает с географией исходного адреса. Например, если время атакующего на пять часов отстаёт от Гринвича, он, вероятно, на восточном побережье США, а не в Китае. Сравнив лучший вариант часового пояса с записями блоков IP, мы заключим, что из всех «приманок» настоящий адрес скорее скрыт за пакетами провайдера Бостона, чем Пекина.

Кроме того, зная локальное время, атакующего можно отслеживать по отклонению часов от настоящего и в долгосрочной перспективе по скорости дрейфа. Компьютерные часы обычно неточны и без внешней синхронизации заметно уходят, иногда на несколько минут в день, поэтому это хороший способ связывать атаки одного человека. Разные машины, скорее всего, систематически отклоняются на разную величину с характерной скоростью изменения.

Наконец, если PID включён в состояние вместе со временем, а время известно в некотором диапазоне, PID помогает определить приблизительное время работы системы или число задач между сканированиями. Каждый новый процесс получает больший PID, поэтому зависимость прямолинейна.[1]

Восстановив состояние PRNG, мы также видим число случайных значений между созданием двух полученных пакетов. Если сканируется одна система, пробелов не должно быть или возможны лишь малые расхождения из-за сетевых проблем. При нескольких системах пробелы от пакетов другим целям легко обнаруживаются, позволяя определить число одновременно проверяемых систем.

Если сканер создаёт ложные пакеты-приманки от случайных хостов, можно исключить поддельные адреса, сформированные PRNG и совпадающие с возможным выходом, и найти не совпадающий настоящий, окончательно указывающий на виновника. Например, если трафик идёт от:

```
198.187.190.55 (десятичное представление: 3334192695)
195.117.3.59   (десятичное представление: 3279225659)
207.46.245.214 (десятичное представление: 3475961302)
```

можно определить, что 3334192695 и 3475961302 были среди первых выходов генератора с S0, а 3279225659 не является ранним выходом восстановленного PRNG и, вероятно, настоящий адрес.

Всё это определяет намерения и программу атакующего. Можно даже отслеживать его систему, сопоставлять с другими данными для настоящей личности и географического положения, а иногда узнавать, как он пользуется компьютером по ходу сканирования.

> **Примечание.** В ответ на рассмотренное раскрытие времени работы и истории сканирования NMAP пытается применять безопасные системные RNG вроде /dev/random из главы 1 вместо стандартной библиотеки C. Но метод недоступен во многих ОС, например Windows, а другие сканеры не приняли сходных мер защиты атакующего.

Защита себя: наблюдение за наблюдениями

За последние десять лет Интернет стал огромным полем битвы. Новые машины мгновенно затопляются автоматическими зондами, червями и другими сведениями, испытывающими безопасность. Традиционное и ныне модное движение обнаружения и предотвращения вторжений стремится найти и остановить атаки, предупреждая администратора о зондировании перед атакой посредством специальных средств анализа. В разнородных или просто сложных средах они часто создают больше шума и ложных срабатываний, чем можно обработать.

Но иногда наблюдение за атаками и вызванными ответами отлично сообщает администратору о сетевых проблемах и атаках по мере возникновения, хотя сами происшествия обычно едва ли примечательны. В некоторых сетях активное обнаружение и инвентарное сканирование для соблюдения политики и конфигурации трудно начать или слишком хлопотно из-за правил, долгого согласования, редких окон обслуживания и так далее. Там возможность подсмотреть, что видят злоумышленники, бесценна как замена локально начатой активной разведке.

Периодическое активное обнаружение также может быть слишком медленным для угроз; способность узнать о внезапной проблеме простым наблюдением чужих результатов ценна. И конечно, меч обоюдоостр: хакер, уже скомпрометировавший сеть или планирующий это, но желающий не высовываться и заранее продумать шаги, может наблюдать трафик других попыток обнаружения, пополняя знания о системе.

Кража полученных атакующим знаний проста лишь в теории; сопоставлять и обрабатывать результаты, особенно в крупных средах или по частичной информации отдельных атак из разных мест, нетривиально. Однако средства картирования сети и систем «пассивным сканированием» постепенно появляются, яркий пример - DISCO Престона Вуда.[2]

Пища для размышлений

Странно, что описанные приёмы часто не подкреплены исчерпывающими исследованиями, опубликованными работами или готовыми инструментами. При повальном увлечении отслеживанием атак, начатом исследованиями honeypot Лэнса Спитцнера и подогретом системами обнаружения вторжений, следовало бы ожидать меньше усилий на идентификацию атак, которые сами по себе обычно не особенно интересны и используют документированные направления и недостатки, и больше попыток установить намерение и происхождение и связать события, бессмысленные поодиночке, но вместе сигнализирующие о проблеме.

Я могу осветить лишь верхушку айсберга, но это, несомненно, одна из самых увлекательных областей для исследования и вклада.

А теперь нечто совершенно иное...

Сноски

[1] [назад] Хотя некоторые системы предлагают необязательную рандомизацию PID, чтобы затруднить другие виды локальных атак.

Часть IV. Общая картина

Наш юридический отдел посоветовал не говорить здесь «сеть - это компьютер»

Глава 16. Паразитные вычисления, или как копейки складываются

Где вновь подтверждается старая истина: армия прислужников лучше самостоятельной работы

Надеюсь, путешествие до сих пор доставляло удовольствие. Я обсудил множество причудливых проблем безопасности и приватности информации от ввода на клавиатуре до конечного назначения в сотнях или тысячах миль. Но праздновать рано: в картине отсутствует нечто гораздо большее всего рассмотренного.

Тёмная материя.

Проблема истории проста: связь не происходит в пустоте. Хотя обмен обычно ограничен двумя системами и дюжиной промежуточных, общий контекст событий нельзя игнорировать; свойства среды способны глубоко формировать реальность болтовни конечных точек. Нельзя отвергать значение систем, непосредственно не участвующих в связи, или всех крошечных, кажущихся изолированными частиц по отдельности тривиальных событий, встречаемых данными по пути. Сосредоточение лишь на кажущемся относящимся к конкретному приложению или случаю может быть фатальным, как, надеюсь, книга уже показала.

Вместо этой близорукости я решил принять величественную общую схему во всей красе. Поэтому четвёртая и последняя часть сосредоточена исключительно на безопасности сетей в целом и рассматривает Интернет как экосистему, а не набор систем для определённых задач. Мы отдаём дань кажущейся инертной материи, связывающей мир.

Часть начинается с анализа концепции, наиболее подходящей для перехода. Для многих компьютерных фанатов паразитные вычисления произвели революцию в представлении об Интернете.

Откусывание кусочков CPU

Скромная работа Альберта-Ласло Барабаши, Винсента У. Фриха, Хавуна Чона и Джея Б. Брохмана, опубликованная в письмах Nature в 2001 году,[1] легко могла остаться незамеченной. На первый взгляд она не заслуживала внимания и выдвигала почти смешное предложение. Авторы полагали, что внутри устоявшихся сетевых протоколов вроде TCP/IP можно создать трафик, который в качестве сообщения ставит удалённому компьютеру простейшую арифметическую задачу; система невольно решит её при разборе и подготовке ответа. Но зачем тратить время, задавая загадки бесчувственным машинам? Что это даст? Не столь же ли весело решить самому? Ответ, конечно, интересен.

Во-первых, решение головоломок компьютером является делом: значительная часть современной криптографии основана на относительной трудности так называемых недетерминированных полиномиальных задач, NP.[1] NP-полные задачи с удовольствием врываются на праздник каждого взломщика в самый неподходящий момент. Способность эффективно решать их огромной вычислительной мощностью, умными алгоритмами или обоими, вероятно, приблизит счастливого изобретателя к мировому господству. Стимул есть. Но как?

Предложенный метод нов. Сначала утверждается, что многие NP-задачи математики легко выражаются уравнениями булевой выполнимости, SAT. Они представляют задачи операциями булевой логики, строя последовательность параметров и переменных, булеву формулу. Классический пример:

```
P = (x1 XOR x2) AND (~x2 AND x3)
```

Здесь `P` - сама формула, а `x1` - `x3` - двоичные входы, параметры.

Хотя существует `2^3` комбинаций, только одна делает `P` истинной: `x1 = 1`, `x2 = 0`, `x3 = 1`. Поэтому лишь эта тройка является решением. Решение SAT сводится к поиску значений всех переменных, при которых вся включающая их формула истинна.

Простейшие случаи вроде этого легко решаются даже перебором, но сложные многопеременные действительно NP-полны, а другие NP-задачи сводятся к SAT за полиномиальное, то есть разумное время.

Здесь и проблема. Мы можем сформулировать трудную NP-задачу как SAT, но это мало даёт. На момент написания для нетривиального уравнения даже лучшие известные алгоритмы ненамного эффективнее грубой силы, где пробуются все возможности и для каждой вычисляется формула. Если есть SAT и достаточно мощности хотя бы для подхода, перебор не столь безумен, а более сложный метод мало продвинет. В любом случае попыткой мы немного теряем.

И вот откровение, связывающее SAT и TCP/IP. Основное наблюдение исследователей довольно очевидно, по крайней мере подписчику Nature: алгоритм контрольной суммы TCP или IP из главы 9, хотя создан совсем для иной цели, является лишь набором последовательных булевых операций над битами входного сообщения. На низком уровне это чистая логика над словами пакета. Предоставив особое содержимое, «вход», удалённую систему заставляют выполнить набор арифметических операций и проверить правильность, совпадение с суммой заголовка TCP или IP.

Хотя операция на каждой итерации абсолютно одинакова, её функциональности достаточно для универсального логического вентиля из главы 2. Чередуя испытательные входы с тщательно выбранными «управляющими» словами, инвертирующими или иначе меняющими текущую частичную сумму, можно выполнить любую булеву операцию.

Это означает, что логику SAT легко воссоздать особой последовательностью управляющих и входных битов пакета под алгоритмом суммы. Переменные уравнения, выбранные тем или иным образом, чередуются с фиксированными словами, преобразующими текущее значение так, чтобы следующая операция имитировала определённый булев оператор. Конечная сумма пакета обозначает итог: логическое значение вычисляемой формулы.

Таким образом, проверку выполнимости случайно выполняет удалённый получатель при проверке суммы. Если она равна 1 или другому значению, соответствующему истинному SAT, испытание для выбранных переменных пройдено, трафик передаётся верхним уровням и обрабатывается. При несовпадении формула не удовлетворена и пакет молча отбрасывается. Иными словами, если входные биты обозначали гипотезу, адресат либо проверил, либо опроверг её, выполняя разные действия по результату.

Желающий быстро решить SAT готовит все комбинации переменных формулы, чередует со сведениями для нужного сочетания, помещает в TCP-пакеты и почти параллельно отправляет множеству хостов мира. Контрольная сумма вручную ставится в значение, которое «гипотеза» создаст при истинности, вместо фактического вычисления. Только хосты с переменными, дающими желаемый результат, ответят; остальные отбросят трафик как повреждённый. Отправитель находит решение без огромных вычислений, просто смотря набор значений в пакетах ответившим хостам.

Работа идёт дальше и сообщает об успешном решении NP-задачи посредством реальных хостов мира, давая не только теорию, но и подтверждение.

Последствие тонко, но важно: вычисления для настоящих задач можно эффективно «отдать на сторону» ничего не подозревающим и не желающим удалённым участникам без атаки, захвата, установки вредоносной программы или иного вмешательства в законные задачи. Один человек делит вычислительную задачу между множеством систем, потребляя лишь крошечную и пренебрежимую долю мощности каждой, которая при миллионах участников складывается в приличный суперкомпьютер.

Мировое господство близко? Не так быстро.

Практические соображения

...или, возможно, пока нет. Подход революционен и интересен, но не обязательно практичен для суперкомпьютера, крадущего у богатых. Для разумной скорости требуется огромная полоса и много вычислений на подготовку мелочей другим системам. Поэтому схема недостаточно эффективна, чтобы отдавать сложные математические задачи глобальному суперкластеру невольных жертв.

Требование экспоненциальной вычислительной мощности обменивается на экспоненциальную полосу. Это не обязательно хорошая сделка, особенно поскольку ограничение размера пакетов позволяет выдавать лишь сравнительно простые испытания, которые, вероятно, решаются за время передачи по Ethernet. Метод доказывает возможность и даёт универсальный путь, но более специфические сценарии могут быть полезнее.

Другие способы красть пренебрежимо малую индивидуальную мощность интереснее как источник впечатляющих вычислений по низкой цене. Например, некоторые клиенты, такие как браузеры, легко заставить выполнять довольно сложные алгоритмы. Один пример, схема «китайской лотереи» из RFC 3607,[2] - крошечный Java-апплет, который сайт Жана-Люка Кука `md5crk.com` предлагал веб-мастерам добавлять на страницы. Каждый посетитель исполнял апплет, занимавший ничтожную долю циклов CPU для проекта поиска столкновений сокращающей функции MD5. Столкновения - два разных сообщения с одинаковым сокращением; они неуловимы и анекдотичны, но определённо возможны[2] и помогают понимать слабости функций и эмпирически показать, что MD5 слишком слаб для современных компьютеров.

Java-апплеты - малые машинно-независимые программы, по умолчанию выполняемые браузером в особой ограниченной «песочнице». У них нет доступа к локальному диску и только теоретически способности навредить, хотя ограниченная сеть позволяет вычислять и добавлять визуальные элементы. Обычно они расширяют сайты играми, эффектами и так далее. Но Жан-Люк использовал их иначе: для поиска вероятных кандидатов столкновений совместной мощностью сотен или тысяч систем одновременно.

Принцип был прост. При посещении сотрудничающего сайта апплет запускался на клиенте и вычислял MD5 для случайных сообщений до нахождения сокращения, совпадающего с произвольной фиксированной маской, например «любое сокращение с четырьмя последними нулевыми байтами». Маска выбиралась так, чтобы перебор не занимал слишком долго и посетитель не ушёл до результата, но соответствовала лишь малая доля всех значений.

Найдя подходящее сообщение, программа «звонила домой» с кандидатом. Автор изучал отправки. Апплет уже проверил и отверг множество кандидатов и передал лишь соответствующие условию, частично одинаковые. Поскольку в таких данных меньше вариаций, вероятность столкновения среди `n` записей значительно выше, чем в чисто случайных. По аналогии два визуально неразличимых яблока легче встретить среди плодов почти одинакового веса и цвета, чем в вагоне произвольных фруктов.

Хотя подход находится в серой зоне киберэтики, впервые открыто развёрнутый `md5crk.com` действительно работал и хорошо демонстрировал эффективность и скрытность паразитных вычислений. Кража циклов процессора, предназначенных для «правомерных» целей, вполне достижима и, возможно, применяется чаще желаемого. И возможность останется.

Но ворчливый скептик продолжает: способны ли паразитные вычисления на большее, чем откусывание крошечных кусочков CPU ради взлома шифрования, интересующего немногих?

Паразитное хранение: первые дни

Когда вы кричите, акустические волны идут через воздух, постепенно теряя энергию и рассеиваясь. Встретив твёрдое препятствие, они, вероятно, отразятся, а при правильном угле вернутся. Через долю секунды после крика слышно эхо собственного голоса.

Но что произойдёт, когда фанат теории информации читает свой код вслух на вершине горы в сторону каменистой долины? Я знал, что вы спросите. Он неизбежно делает умное наблюдение: если читать быстро и сразу забывать произнесённое, отвлёкшись на другие дела, позднее информацию всё равно можно восстановить, когда она отразится от дна долины и вернётся эхом. Voilà - удобный механизм хранения данных.

Звучит нелепо? Возможно, мы просто слишком молоды. В ранних типах компьютерной памяти применялась похожая акустическая техника: процессор мог хранить часть информации «вне себя» и позднее получать её обратно. Вместо воздуха, в котором волны распространяются слишком быстро, чтобы без чрезвычайно больших запоминающих устройств обеспечить разумную ёмкость, использовался заполненный ртутью барабан - среда, где акустические волны идут гораздо медленнее. Однако принцип оставался тем же и даже придавал интересный смысл выражению memory leak - «утечка памяти». Такое устройство, память на ртутной линии задержки, применялось, например, в знаменитом UNIVAC I.[3]

Естественно, от такой медленной, громоздкой, опасной и неудобной памяти отказались в пользу других решений, едва технология достаточно развилась. Но у самого изобретения было своё очарование, и так просто исчезнуть в забвении оно не могло. Короткая презентация Сакиба А. Хана на конференции DefCON в Лас-Вегасе в 2002 году возродила эту идею и впервые подсказала, как использовать свойства крупномасштабной сети, чтобы построить похожее временное хранилище на основе Интернета. Но теперь описание акустической памяти казалось смотревшим короткий показ слайдов хакерам и гикам не смехотворно примитивным, а невероятно крутым. Акустическая память вернулась со стилем.

Поскольку время кругового пути пакета - время, необходимое сообщению, чтобы достичь удалённой системы, а ответу вернуться, - ненулевое, некоторое количество данных всегда можно держать «в проводе», многократно отправляя и принимая их части и ожидая возвращения эха. Сакиб добился этого с помощью пакетов ICMP (Internet Control Message Protocol) «echo request» (ping); большинство систем Интернета отвечает на них пакетами «echo reply», цитируя исходную полученную полезную нагрузку.

Это выглядело отличным трюком. Но для любого разумного приложения он был далеко не практичен, потому что части данных требовалось часто пересылать заново. Поскольку ICMP «echo reply» отправляется почти немедленно после получения «echo request», наружу удавалось вытолкнуть лишь немного данных, прежде чем они начинали возвращаться и их требовалось снимать с провода. В итоге объём такого хранилища не мог превышать количества, которое пользователь способен отправить в лучшем случае за пару секунд, а чаще - менее чем за десятую долю секунды.

Ах, но паразитное хранение можно было улучшить.

Как сделать паразитное хранение осуществимым

В 2003 году мы с Войцехом Пурчиньским совместно написали статью «Жонглирование пакетами: паразитное хранение данных». Мы развили идею паразитного хранения и рассмотрели ряд методов, способных резко увеличить ёмкость Интернета при экономии полосы, необходимой для сохранения информации. Наше исследование охватывало несколько других способов хранить данные в удалённых системах и классифицировало их по свойствам носителя: наблюдаемости, энергозависимости и надёжности. Мы также подробно обсудили гипотетическую ёмкость каждого метода.

Статья получилась довольно короткой и, надеюсь, освежающей и забавной; она приведена ниже.

Жонглирование пакетами: плавающее хранилище данных

> «Ваше подземелье построено под уклоном. Злые чудовища не могут играть в шарики!»

Войцех Пурчиньский <cliph@isec.pl>
Михал Залевский <lcamtuf@coredump.cx>

1) Жонглируйте апельсинами!

Большинство из нас, включая авторов этой статьи, пробовало жонглировать тремя или более яблоками, апельсинами либо иными хрупкими баллистическими объектами. Результат обычно довольно жалок, но рано или поздно большинство способных падаванов-жонглёров учится делать это без чрезмерного сопутствующего ущерба.

Особенно сообразительный ученик может заметить: пока он продолжает выполнять простую процедуру, хотя бы один предмет всё время находится в воздухе, а одновременно в руках приходится держать не более двух. И всё же каждое яблоко время от времени проходит через его руки, и его можно по желанию вернуть.

Вдоволь позабавившись жонглированием, он может решить, что весь процесс чрезвычайно скучен, и вернуться к компьютеру. Проверяя электронную почту, образованный жонглёр способен заметить, что у типичной сетевой службы всего одна обязанность: принять и обработать данные от удалённой системы и предпринять любые действия, которые она считает подходящими, исходя из собственной интерпретации этих данных. Многие такие службы изо всех сил стараются работать устойчиво, сохранять работоспособность при сбоях и давать полезную обратную связь о транзакции.

В некоторых случаях сам факт, что служба пытается обработать данные и ответить по протоколу, можно использовать способами, о которых авторы даже не мечтали. Один из самых впечатляющих примеров, возможно знакомый нашему коллеге-жонглёру, - исследование Университета Нотр-Дам под названием «Паразитные вычисления», опубликованное в письмах в Nature.

Тем не менее наш герой заключает, что в реальном мире такие попытки весьма непрактичны. Цена подготовки и доставки решаемых мелочей намного превосходит любую возможную выгоду, поскольку отправителю приходится выполнять операции сравнимой вычислительной сложности

лишь для доставки запроса. «Вычислительная мощность такого устройства ничтожна!» - говорит он.

Настоящий жонглёр сосредоточился бы на другом виде обработки данных на стороне, гораздо более близком к его области знаний. Почему бы не создать распределённое фруктовое хранилище? Что, если написать на каждом апельсине по одной букве, а затем начать жонглировать? Тогда я смогу хранить больше апельсиновых байтов, чем вмещает моя физическая ёмкость - число апельсинов, которое я могу держать в руках! Как гениально... Но, но сработает ли это без апельсинов?

2) То же самое без апельсинов

Эта статья основана на наблюдении, что при любой сетевой связи между отправкой информации и получением ответа есть ненулевая, а часто значительная задержка - следствие физических ограничений среды и времени обработки данных всем компьютерным оборудованием.

Подобно апельсину с написанным на нём сообщением, пакет с фрагментом данных некоторое время путешествует, прежде чем вернуться к источнику, и на это время мы можем безопасно забыть его сообщение, не потеряв данные. Следовательно, Интернет обладает ненулевой мгновенной ёмкостью хранения: часть информации можно вытолкнуть наружу и фактически хранить до её возвращения эхом. Создав механизм циклической передачи и приёма фрагментов данных к множеству удалённых хостов и обратно, можно постоянно поддерживать произвольный объём данных «в проводе», образуя высокоёмкую энергозависимую среду.

Эту среду можно применять в операциях, требующих много памяти, как обычное хранилище либо для некоторых видов чувствительных данных, физический след которых нежелательно оставлять на жёстком диске или другом энергонезависимом носителе.

Поскольку возвращать отправителю столько уместной информации, сколько тот передал службе, не считается плохой практикой программирования, а многие службы и стеки весьма многословны, наш опыт жонглирования говорит: создать такое хранилище не только возможно, но и осуществимо даже через слабое сетевое подключение. В отличие от традиционных методов паразитного хранения данных, таких как злоупотребление P2P, открытые FTP-серверы, двоичные публикации Usenet и так далее, этот конкретный метод в зависимости от реализации может как оставлять, так и не оставлять след данных и не создаёт заметной нагрузки ни на одну систему. Поэтому, в отличие от традиционных методов, его реже обнаружат и сочтут злоупотреблением. Следовательно, возможность перехвата и намеренного уничтожения данных представляет гораздо меньшую проблему.

3) Хранилище данных класса A: буферы памяти

Хранилище класса A использует ёмкость, присущую задержкам связи при передаче и обработке активных данных, пока они идут по сетям между двумя конечными точками. Хранящаяся здесь информация остаётся кэшированной в памяти удалённой машины и, скорее всего, не будет выгружена в файл подкачки на диске.

Примеры памяти класса A - разнообразные схемы, основанные на отправке сообщения, которое заведомо вызывает частичное или полное эхо исходного запроса, включая следующие:

• ответы SYN+ACK, RST+ACK на пакеты SYN и другие отскоки;
• эхо-ответы ICMP;

• ответы поиска DNS и данные кэша. Часть информации можно сохранить в запросе поиска и получить обратно с ответом NXDomain либо записать данные в кэш NS;
• ретрансляция сообщений между серверами чат-сети. При пересылке текстовых сообщений через серверы IRC и так далее может возникать значительная задержка;
• сообщения об ошибках или состоянии HTTP, FTP, веб-прокси либо SMTP.

Важнейшие свойства хранилища класса A:

• малая задержка - от миллисекунд до минут, благодаря чему оно полезнее для приложений, близких к памяти с произвольным доступом;
• меньшая ёмкость каждой системы - обычно килобайты, из-за чего оно хуже подходит для массивного хранения;
• лишь одна возможность получить данные или малое число повторных передач, что снижает надёжность при сбое сети;
• меньшая вероятность постоянной записи. Данные едва ли сохранятся на энергонезависимом носителе или попадут в файл подкачки, что повышает конфиденциальность и правдоподобность отрицания.

В частности, при протоколах верхнего уровня появляются дополнительные возможности, способные устранить некоторые общие для разных видов класса A проблемы малой ёмкости и короткого окна восстановления. Например, можно установить соединение со службой SMTP, FTP, HTTP или любой другой текстовой службой и передать команду, которая заведомо приводит к возврату подтверждения или сообщения об ошибке вместе с частью исходных данных. Однако полностью сформированную команду мы не отправляем: некоторые необходимые символы остаются непереданными. В большинстве случаев для завершения команды нужны символы конца строки. В этом состоянии наши данные уже хранятся удалённой службой, ожидая полной команды или истечения времени соединения. Чтобы предотвратить тайм-аут на уровне TCP или приложения, следует периодически передавать пакеты без операции. Символ `\0`, интерпретируемый как пустая строка, не влияет на многие службы, но достаточен, чтобы сбросить таймеры TCP и службы. Известный пример приложения, уязвимого для этой атаки, - Microsoft Exchange.

Атакующий может поддерживать соединение произвольно долго, пока фрагмент данных уже хранится на другом конце. Для восстановления информации команду нужно завершить отсутствующими `\r\n`, после чего ответ отправится клиенту.

Хороший пример - команда SMTP `VRFY`:

```
220 inet-imc-01.redmond.corp.microsoft.com Microsoft.com ESMTP Server
Thu, 2 Oct 2003 15:13:22 -0700
VRFY AAAA...
252 2.1.5 Cannot VRFY user, but will take message for
<AAAA...@microsoft.com>
```

Таким способом можно сохранить чуть более 300 байтов, включая непечатаемые символы, и получить к ним почти мгновенный доступ. Ещё больше данных удаётся хранить методом HTTP `TRACE`, передавая их в произвольных заголовках HTTP; объём зависит от серверного ПО. Поддерживаемые соединения дают произвольно большую задержку и тем самым создают большую ёмкость хранения.

Естественно, этот вид больше подходит для критичных к конфиденциальности приложений или хранилища малой задержки и малой либо средней ёмкости - немедленного расширения RAM для информации, не должной оставлять видимых следов. Для критических данных, которые требуется сохранить любой ценой, он непригоден из-за риска потери при сбое сети.

4) Хранилище данных класса B: дисковые очереди

Хранилище класса B использует «простаивающие» очереди данных, которые сохраняют информацию продолжительное время, часто на диске. Например, системы MTA могут ставить сообщения электронной почты в очередь на срок до 7 дней, а в зависимости от конфигурации и дольше. Эта возможность даёт большую задержку между отправкой данных для хранения на удалённом хосте и их получением. Поскольку типичный SMTP-сервер не позволяет пересылать почту от клиента ему самому, для возврата данных после долгого срока можно использовать отскоки электронной почты.

Например, рассмотрим возможный сценарий атаки:

1. Пользователь составляет список SMTP-серверов - возможно, таких, которые с разумной вероятностью находятся вне досягаемости его противников.
2. Пользователь блокирует все входящие соединения со своим портом 25 с помощью block/drop, не reject.
3. Для каждого сервера атакующий должен подтвердить тайм-ауты доставки и IP-адрес, с которого сервер подключается в ответ, пытаясь вернуть отскок. Для этого подходящая проба отправляется на локальный для сервера адрес либо запрашивается уведомление DSN для действительного адреса, после чего проверяется, как долго сервер пытается подключиться в ответ, прежде чем сдаться. Сервер не обязан быть открытым ретранслятором.
4. Подтвердив цели, атакующий начинает отправлять данные в таком темпе, чтобы процесс равномерно растянулся на одну неделю. Данные следует разделить так, чтобы каждому серверу приходился один фрагмент. Каждый фрагмент отправляется отдельному серверу и немедленно порождает отскок обратно отправителю.
5. Поддержание данных сводится к тому, чтобы не позднее чем через неделю после первоначальной отправки, непосредственно перед удалением записи из очереди, принять входящее соединение и получить возврат. Для этого конкретному серверу разрешают пройти через межсетевой экран. Сразу после получения фрагмент снова пересылается наружу.
6. Чтобы обратиться к любой части данных, атакующий выясняет, какой MTA держит конкретный блок, а затем разрешает этому IP подключиться и доставить отскок. Возможны три сценария:
• если удалённый MTA поддерживает команду `ETRN`, доставку можно вызвать немедленно;
• если удалённый MTA находится посреди трёхминутной серии попыток соединиться с локальной системой - он продолжает повторять их, потому что его SYN-пакеты отбрасываются, а не отвергаются ответом RST+ACK, - соединение можно установить за считаные секунды;
• в противном случае придётся ждать от пяти минут до часа, в зависимости от настроек очереди.

Для пользователей с динамическим IP эту схему можно усовершенствовать, применяя DNS-имена вместо IP; это также обеспечивает дополнительную защиту или позволяет при необходимости немедленно разорвать цепочку.

Важные свойства хранилища класса B:

• высокая ёмкость каждой системы - мегабайты, что делает его превосходным решением для больших файлов и подобных данных;
• большая задержка доступа - от минут до часов, из-за чего оно похоже на ленточное устройство, а не RAM; исключение составляют SMTP-хосты, принимающие команду `ETRN` для немедленного возобновления доставки;
• очень долгий срок жизни, повышающий ёмкость на пользователя и надёжность;
• множество попыток доставки, благодаря чему данные легко восстановить даже после временных проблем сети или оборудования;
• высокая вероятность оставить след на устройствах хранения, что делает решение менее полезным для полностью отрицаемого хранения, хотя для обнаружения всё равно пришлось бы исследовать множество чужих систем, а это не обязательно осуществимо.

Хранилище класса B подходит для обычных файловых архивов, больших буферов только для добавления, зашифрованных ресурсов - при правильном выборе хостов их наличие практически можно отрицать - и так далее.

5) Скрытное хранилище класса A

В некоторых ситуациях может потребоваться решение для скрытного хранения данных, которые не находятся на самой машине и наличие которых где бы то ни было можно отрицать.

Основные требования к данным:

• они не возвращаются, пока не передана особая ключевая последовательность;
• при отсутствии запросов keep-alive они безвозвратно уничтожаются, не оставляя записей ни на одном энергонезависимом носителе.

Эту функциональность можно реализовать средствами хранилища класса A и рассмотренного ранее метода поддерживаемой команды. Для освобождения данных требуется правильный порядковый номер TCP, а до его передачи данные не возвращаются и не раскрываются ни одной стороне. Если клиентский узел отключается, данные уничтожаются и, скорее всего, перезаписываются.

Таким образом, порядковый номер служит ключом к хранимой информации и, если после прекращения keep-alive `\0` срок жизни данных достаточно короток, часто обеспечивает достаточную защиту.

6) Ёмкость, доступная пользователю

В этом разделе мы попытаемся оценить ёмкость хранения, доступную одному пользователю.

Чтобы постоянно поддерживать некоторый объём данных «отданным» сети, мы должны иметь возможность регулярно получать и отправлять его обратно.

Время удалённого хранения ограничивается максимальным сроком жизни `Tmax` одного пакета, включая задержки постановки в очередь и обработки. Максимальный объём отправляемых данных ограничен наибольшей доступной пропускной способностью сети `L`. Следовательно, максимальную ёмкость можно определить так:

```
Cmax [байты] = L [байты/секунду] * Tmax [секунды] / Psize * Dsize
```

где:

• `Dsize` - размер пакета, необходимого для первоначального сохранения части данных на удалённом хосте;
• `Psize` - размер пакета, необходимого для поддержания информации, хранимой на удалённом хосте.

`Psize` и `Dsize` равны и потому могут быть опущены всякий раз, когда весь фрагмент данных отскакивает туда и обратно; они различаются лишь в сценариях «поддерживаемой команды». Самый маленький пакет TCP/IP, способный это выполнить, содержит 41 байт. Максимальный объём данных, который можно поддерживать в заголовках HTTP, составляет около 4096 байтов.

Всё это, в свою очередь, даёт следующую таблицу:

Пропускная способность	Класс A	Класс B
28,8 Кбит/с	105 Мбайт	2 Гбайт
256 Кбит/с	936 Мбайт	18 Гбайт
2 Мбит/с	7,3 Гбайт	147 Гбайт
100 Мбит/с	365 Гбайт	7 Тбайт

7) Интернет в целом

В этом разделе мы попытаемся оценить теоретическую мгновенную ёмкость Интернета в целом.

Класс A

Для оценки теоретической ёмкости хранилища класса A в Интернете мы принимаем следующие допущения:

• сообщения ICMP обеспечивают лучший баланс между ёмкостью хранения и сохранением ресурсов удалённой системы;
• средняя операционная система имеет входную очередь пакетов, способную вместить не менее 64 пакетов;
• PMTU по умолчанию составляет примерно 1500 байтов - это самый распространённый MTU.

Для оценки числа хостов Интернета мы используем обследование ISC за 2003 год, в котором перечислены 171 638 297 систем с обратными записями DNS, хотя не все IP с обратной DNS обязаны быть работоспособны. Чтобы это учесть, мы воспользовались долей ответов на эхо-запросы ICMP, вычисленной по последнему обследованию, где проводилась такая проверка, - за 1999 год. Тогда данные показывали, что активно примерно 20 процентов видимых систем; следовательно, число систем, готовых отвечать на запросы ICMP, составляет около 34 000 000.

Умножив число систем, отвечающих на эхо-запросы ICMP, на средний размер кэша пакетов и максимальный размер пакета без заголовков, мы оцениваем общую теоретическую мгновенную ёмкость ICMP-хранилища класса A примерно в 3 Тбайт.

Класс B

Для оценки теоретической ёмкости хранилища класса B возьмём пример программного обеспечения MTA. Верхней границы объёма данных, подаваемых одному хосту, нет. Хотя можно смело предположить, что только сообщения размером менее примерно 1 Мбайт не создадут заметной нагрузки на систему и других нежелательных последствий, мы считаем, что средний максимальный размер очереди равен 500 Мбайт.

Наши исследования показывают, что примерно у 15 процентов систем, отвечающих на ping-запросы, открыт порт 25. Поэтому мы оцениваем популяцию SMTP-серверов в 3 процента - 15 процентов от 20 процентов - общего числа хостов, то есть немногим более 5 000 000 хостов.

Это даёт общую ёмкость хранения 2500 Тбайт.

Применения, социальные соображения и защита

Но что теперь? Какая польза от практических схем паразитных вычислений и хранения, если их преимущества всё ещё далеко не настолько велики, чтобы стать заманчивой альтернативой простой покупке дополнительного оборудования?

Несмотря на прогресс в практической эксплуатации паразитных вычислений, приложения, предназначенные лишь для расширения вычислительной мощности или объёма хранения традиционной системы, могут казаться бессмысленными при обилии дешёвой памяти и гигагерцевых процессоров.

Однако незримый потенциал этой технологии может скрываться в совершенно ином наборе применений: энергозависимых вычислениях. Возможность создавать пригодные к работе распределённые компьютеры, способные по желанию рассеиваться, не оставляя физических следов и не храня осмысленных данных ни в одном месте, может стать мощным средством конфиденциальности и одновременно создать трудности для криминалистики и правоохранительных органов. Возможность построить энергозависимую память типа «сохрани и удерживай», которая разрушается вскоре после отключения одного узла, но не требует частых повторных передач данных, может дать правонарушителю - или, если уж на то пошло, угнетаемой стороне - хороший уровень правдоподобного отрицания и потребовать очень серьёзно изменить многие обычные процедуры сбора доказательств.

Более того, представьте энергозависимые системы, которые после загрузки и инициализации способны долго поддерживать себя сами, живя в Интернете и не имея локализованного физического присутствия. Возможны две конструкции энергозависимых распределённых компьютерных систем, и ни одна не столь абсурдна:

• системы можно проектировать так, чтобы они выполняли сложную задачу, параллельно ища решение; это уже в основном достигнуто рассмотренной выше схемой вычисления SAT. Недостаток таких систем в том, что результат вычисления приходится получать, а следующую итерацию обработки - запускать вручную, время от времени «засевая» всю систему заново из некоторого места. Решения на основе низкоуровневых свойств протоколов вроде TCP, вероятно, относятся к этой категории;
• системы можно проектировать так, чтобы последующие итерации распределённых вычислений они выполняли сами. Для такой деятельности можно применять всевозможные злоупотребления функциями верхнего уровня - например, встроенными алгоритмами отрисовки документов - и некоторыми сетевыми службами.

В обоих случаях последствия могут оказаться весьма серьёзными. Например, как вывести из строя избыточную самовосстанавливающуюся машину, которая не использует никакую единственную систему, а на доли секунды заимствует у других крошечные части памяти и вычислительной мощности, причём не эксплуатирует уязвимостей и не создаёт отчётливо различимого трафика, который можно отфильтровать? И разве не немного тревожно осознавать, что цели такого распределённого компьютера невозможно будет сразу определить? Почтительно кланяясь мастерам плохой научной фантастики, я считаю, что господство компьютеров неизбежно, и хочу поприветствовать наших новых машинных повелителей.

Пища для размышлений

Защита от паразитных вычислений в целом чрезвычайно трудна. Возможность хранить данные или заставлять другую сторону выполнять некоторые тривиальные вычисления часто неразрывно

связана с основными функциями сетевых протоколов. Мы не представляем, как удалить эту характеристику, не уничтожив знакомый нам Интернет и не породив множество новых проблем, более серьёзных, чем устранённая.

Защитить отдельную систему от превращения в узел паразитных вычислений тоже довольно трудно, поскольку украденные у неё ресурсы часто составляют ничтожную долю простаивающего процессорного времени и памяти и потому легко остаются незамеченными.

Весьма вероятно, что паразитные вычисления ещё не раскрыли весь свой потенциал и что эта угроза - несущественная или несуществующая для отдельных систем, но значимая для сети в целом - останется с нами.

Сноски

[1] [назад] В теории сложности полиномиальные задачи решаются машиной Тьюринга за время, полиномиально пропорциональное длине входа, количеству или размеру переменных. Необходимое время прямо соответствует длине в постоянной степени, которая может быть нулевой, и тогда оно вообще не зависит от входа, как при проверке чётности. Для недетерминированных полиномиальных, NP, задач решения такого рода неизвестны, и при росте входа может требоваться резко больше времени, например экспоненциально. Подмножество NP-полных задач доказанно не имеет решений за полиномиальное время. NP обычно считаются «трудными» для нетривиального входа, P обходятся дешевле.
[2] [назад] Пока книга готовилась к печати, китайские исследователи Шаньдунского университета Сяоюнь Ван, Дэнго Фэн, Сюэцзя Лай и Хунбо Ю сообщили способ поиска и примеры столкновений MD4, MD5, HAVAL-128 и RIPEMD-128. Это одна из важнейших новостей современной криптографии и подтверждение непригодности функций для некоторых применений безопасности. Проект `md5crk.com` закрылся, но его вклад в исследование паразитных вычислений остаётся действительным.
[3] [назад] Возможно, стоит отметить, что малоёмкая аналоговая память на линии задержки применялась и в ранних реализациях телевизионных приёмников SECAM (Séquentiel Couleur avec Mémoire, или «последовательный цвет с памятью»). В отличие от NTSC или PAL, сигнал SECAM использует пониженное цветовое разрешение: красная и синяя цветоразностные составляющие передаются попеременно, никогда одновременно. Чтобы определить вид конкретного пикселя, другую составляющую приходится брать из предыдущей строки. Для этого и требовалось запоминающее устройство.

Глава 17. Топология сети

О том, как знание окружающего мира помогает выслеживать друзей и врагов

Какова форма Интернета? Ни один комитет не надзирает за ним и не решает, где, как и почему он должен расширяться или как следует организовывать и управлять новыми и существующими системами. Интернет растёт во всех направлениях под равноценным влиянием спроса, экономики, политики, технологий и слепой удачи.

И всё же Интернет - не бесформенное пятно. В нём есть спланированные, управляемые на местном уровне иерархии автономных систем, где магистральные маршрутизаторы окружены менее значительными узлами, а связи настраиваются автоматическими механизмами или тщательно проектируются людьми. Интернет - впечатляющая ячеистая сеть, сложная и хрупкая паутина, покрывающая весь промышленно развитый и развивающийся мир. Задача запечатлеть эту постоянно меняющуюся топологию кажется трудной, но и заманчивой, особенно когда понимаешь, какую пользу способны принести собранные сведения.

В этой главе я сначала рассмотрю две примечательные попытки картографировать топологию Интернета, а затем ещё раз порассуждаю о возможном применении собранных таким образом данных для вещей, о которых наши предки не могли даже мечтать.

Запечатлеть мгновение

Самую всестороннюю попытку составить карту Интернета предприняла Cooperative Association for Internet Data Analysis (CAIDA) - организация, финансируемая, среди прочих, федеральными исследовательскими ведомствами NSF, DHS и DARPA, а также отраслью, включая Cisco и Sun. Организация была создана для разработки анализа трафика и инфраструктуры и соответствующих инструментов на общее благо интернет-сообщества, в надежде сделать Интернет лучше, надёжнее, устойчивее и крепче.

С 2000 года одним из ведущих публичных проектов CAIDA стало создание и сопровождение карты магистральной сети автономных систем, также известной как Skitter. На момент выхода этой книги самый свежий снимок представлял данные о 12 517 крупных автономных системах, соответствующих 1 134 634 IP-адресам и 2 434 073 связям - логическим путям - между ними.

Несмотря на поразительно загадочное звучание, карта Интернета CAIDA была создана лишь по общедоступным данным конфигурации BGP маршрутизаторов, эмпирическим результатам проверки сети с помощью traceroute и записям WHOIS для сетевых блоков. Карта организована в полярных координатах. Точка каждой системы расположена под углом, соответствующим физическому местонахождению заявленной штаб-квартиры сети, и на радиусе, соответствующем «значимости пиринга» данной автономной системы. Последний параметр выводится из числа других автономных систем, которые, согласно наблюдениям, принимают трафик от этого узла. Поэтому массивные магистральные системы находятся ближе к центру карты, а системы, непосредственно контактирующие всего с парой узлов, - у внешнего периметра. Линии графа просто соответствуют пиринговым отношениям между маршрутизаторами.

> **ПРИМЕЧАНИЕ.** К огромному сожалению, нам не разрешили бесплатно использовать в книге графическое представление Skitter от CAIDA. Однако я советую посмотреть это потрясающее изображение в Интернете по адресу http://www.caida.org/analysis/topology/as_core_network/pics/ascoreApr2003.gif, где оно бесплатно доступно широкой публике.

Ещё одна заслуживающая внимания попытка картографировать сеть опиралась на анализ расстояний до разных сетей из определённой точки - в данном случае из Bell Laboratories - и строила древовидную структуру, совсем не похожую на сложную ячеистую сеть CAIDA. Этот анализ, проведённый Биллом Чесвиком в 2000 году,[1] дал карту на рис. 17-1. Структура не параметризует граф по физическому или административному местонахождению системы; относительное расстояние от центра соответствует числу переходов между узлом и Bell Labs.

Хотя обе попытки, по-видимому, требовали огромного сбора и анализа данных, даже любителю с довольно слабым каналом не запредельно трудно попробовать картографировать сеть. Чтобы отправить по одному пробному пакету во все публично маршрутизируемые подсети, может потребоваться всего пара гигабайтов трафика - эквивалент от нескольких часов до одного дня работы обычного DSL-соединения. Единственный риск - расстроить некоторых системных администраторов, но с распространением компьютерных червей и автоматических атак очень немногие сохранили столь низкий порог чувствительности. Наблюдаемую структуру Интернета можно картографировать, и это способно принести пользу, особенно потому, что многое рассказывает об организации всемирной сети.

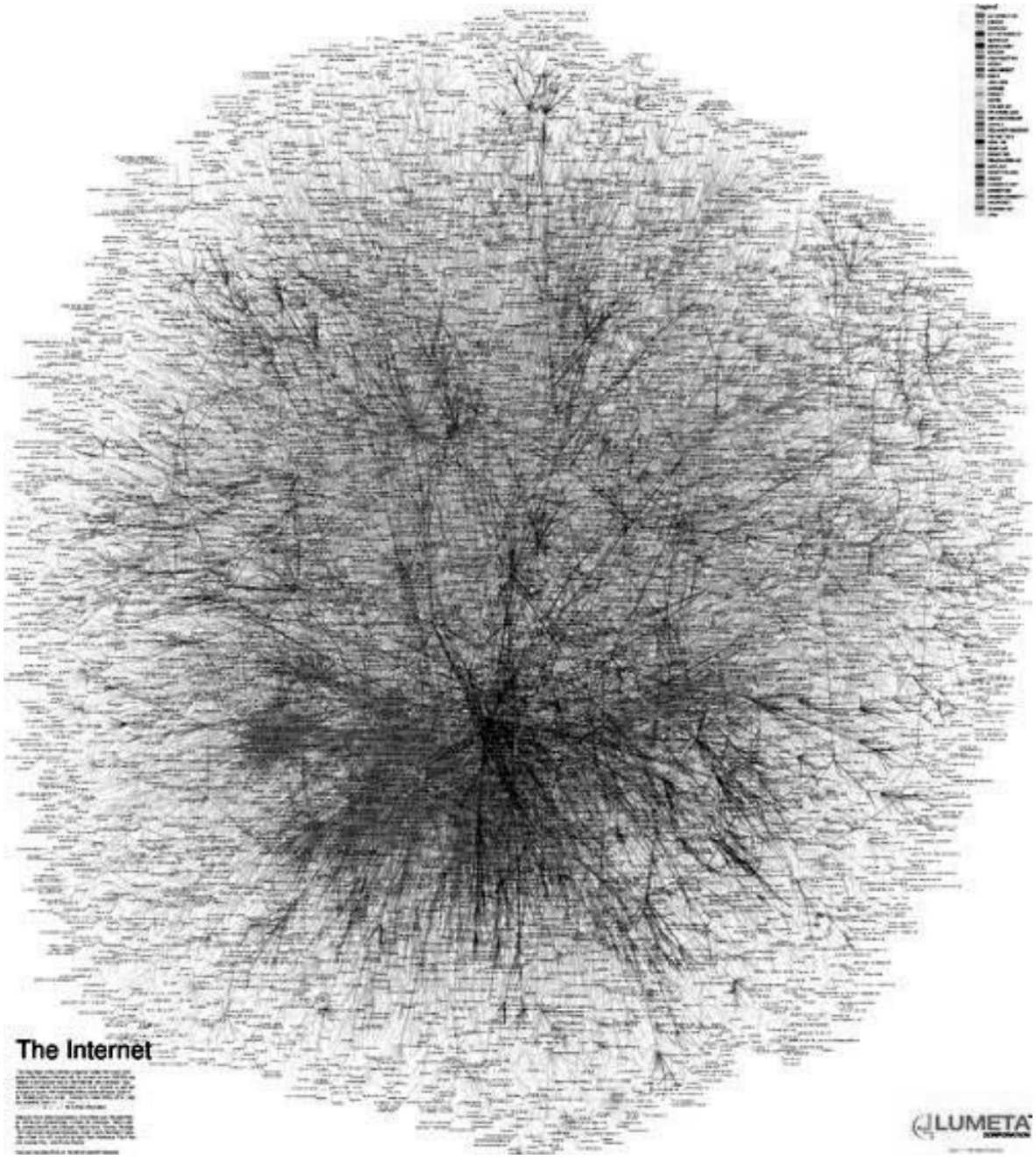


Рисунок 17-1. Карта Интернета Билла Чесвика

Но, как выясняется, сведения, полученные CAIDA, Биллом Чесвиком или практически любым опытным пользователем Сети, можно с успехом применять и для того, чтобы лучше понять природу и точнее исследовать происхождение загадочного трафика, с которым мы однажды можем столкнуться.

Использование данных топологии для определения источника

Поддельный трафик - одна из главных проблем Интернета или, по крайней мере, одна из самых досадных его напастей. Пакеты, подделанные вслепую и снабжённые фиктивными либо специально выбранными, но обманными адресами источника, позволяют злоупотреблять доверительными отношениями между компьютерами, внедрять вредоносное содержимое - например, нежелательные массовые рассылки - без убедительных следов и достоверных сведений

об источнике и так далее. Слепая подделка позволяет также скрывать личность атакующего, который зондирует системы, выполняя рассмотренное в главе 13 «сканирование с приманками». Но худшая из всех напастей - подделка адресов для атак отказа в обслуживании, DoS.

При обычной DoS-атаке администратор имеет возможность увидеть источник вредоносного трафика, направленного против одной из его служб и, предположительно, призванного вывести её из строя, причинив оператору неудобство или убыток. Однако атакующие пакеты можно подделывать случайным образом, и тогда администратор остаётся беспомощен: он не способен отфильтровать трафик атакующего, не отрезав других пользователей. Единственная надежда - совместно с вышестоящим провайдером исследовать настоящий источник трафика на канальном уровне и передать сведения провайдеру нарушителя; но это требует времени, и немалого. Кроме того, без судебного приказа нужно убедить все стороны, что дело заслуживает расследования, а также их времени и денег. Поэтому особенно важно снабдить системного администратора инструментами и методами, отличающими поддельный трафик от настоящего.

Когда я жил и работал в Соединённых Штатах - сейчас я живу в Польше, - мой коллега Марк Лавлесс решил реализовать идею, впервые предложенную Дональдом Маклахланом: измерять время жизни TTL сетевого трафика между собой и предполагаемым отправителем пакета, чтобы автоматически определять, подделан ли входящий пакет. В мире, где сведениям нельзя доверять, задача определения источника сетевого пакета важна, и способность решить её хотя бы в некотором подмножестве случаев по указанным выше причинам принесла бы большую пользу множеству аналитических и административных задач.

Чтобы понять идею Дональда и Марка, учтите: удалённая система, от которой мы видим трафик, находится на определённом логическом расстоянии от нас, отделённая заданным числом сетевых устройств. Поэтому все правомерно отправленные ею пакеты при прибытии имеют некоторое TTL, равное исходному TTL по умолчанию, настроенному в системе, минус число пройденных пакетом промежуточных систем, как рассматривалось в главе 9. Но у поддельного трафика, предположительно исходящего из совершенно другой сети, начальный TTL и расстояние, скорее всего, отличаются от того, что следует из данного наблюдения. Инструмент Марка `despoof`[2] сравнивает TTL в специально вызванном и ранее полученном трафике, чтобы различить настоящий и фальсифицированный трафик.

Однако, хотя в отдельных случаях против ничего не подозревающих атакующих метод может хорошо работать, с ним связаны по меньшей мере две проблемы:

• Параноидальный атакующий может перед атакой измерить расстояния и выбрать TTL, совпадающий с ожидаемым значением. Хотя это возможно, реализовать трюк несколько трудно. Во-первых, атакующий может физически не суметь установить достаточно высокий TTL, чтобы при достижении назначения получить конкретное значение, ожидаемое от настоящего пакета. Его план можно сорвать, если выдаваемая им система использует TTL по умолчанию, равный или близкий 255 - наибольшему возможному, - а атакующий находится дальше от цели, чем эта система; тогда отправить пакет с желаемым TTL при прибытии совершенно невозможно. Конечно, мало какие системы применяют наибольший TTL, да и атакующему редко требуется выдавать себя именно за конкретную систему.

Вторая трудность атакующего в том, что он может не суметь определить точное расстояние между жертвой и выдаваемой системой, если находится далеко от обеих и не знает особенностей маршрутизации между этими хостами. Но если жертва с помощью `despoof` динамически создаёт правила фильтрации, отсекающие вредоносные пакеты, атакующий может просто пробовать различные TTL из разных источников, пока не увидит, что жертва больше не способна их различать. Это будет очевидно: атакуемая система начнёт проявлять последствия успешной атаки, например снижение производительности.

• При получении каждого подозрительного пакета адресат должен начать расследование, а затем ждать результатов. Поэтому использовать `despoof` как основу автоматической защиты непрактично, особенно при DoS-атаках. Тем не менее этот метод весьма полезен для определения настоящего источника «сканирования с приманками».

Без знания топологии конкретной сети трудно добиться чего-то лучшего, чем `despoof`; реализованная инструментом техника анализа TTL достаточно хороша, чтобы распознать и остановить многие обычные зондирования и отдельные атаки. Но что дальше?

Объедините инструмент Марка с данными о структуре сети в реальном времени и примените пассивное снятие отпечатков для определения начального TTL системы, отправляющей конкретные запросы, - и техника станет гораздо мощнее. Дополнительные сведения позволяют выполнить начальную пассивную оценку входящего трафика, сравнив наблюдаемый и начальный TTL с ожидаемым расстоянием по карте сети.[1] Поскольку должное наблюдаемое расстояние можно определить без активного зондирования данных о топологии сети, мы сразу и без особых усилий отличаем настоящий трафик от вредоносного. Это, в свою очередь, позволяет довольно надёжно реагировать на массовые происшествия и обнаруживать отдельные малозаметные пробы, не предупреждая атакующего о наличии системы выявления подделок.

Очевидно, учёт структуры сети при рассмотрении одноранговых отношений способен принести немало пользы. Но обнаружение подделок - лишь начало.

Сетевая триангуляция по данным ячеистой топологии

Сетевая триангуляция - значительно более интересное применение данных о ячеистой топологии сети для анализа трафика. Она позволяет определить приблизительное местонахождение атакующего, который отправляет поддельные пакеты, без помощи операторов нижележащей магистральной инфраструктуры, как только он решит атаковать несколько целей одновременно или последовательно. Воистину счастье в несчастье.

Если быть точным: хотя триангуляция лучше всего работает, когда атакующий выбирает несколько целей, в некоторых сценариях она вполне способна сработать и при атаке всего одной службы. В частности, одну атаку можно наблюдать из разных точек, если атакуемый объект имеет несколько IP-адресов, а служба работает из нескольких физических мест для распределения нагрузки и отказоустойчивости всей структуры, как часто бывает у веб-служб. Во всех остальных сценариях набор данных об атаке можно получить, когда системные администраторы замечают, что атакующий нацелился более чем на одну систему, и обмениваются сведениями о происшествии.

Независимо от случая, едва данные, предположительно исходящие из одного источника, замечены более чем в одном месте назначения, мы можем выполнить триангуляцию. Для каждого места назначения, где замечен трафик, лишь определённый набор сетей находится на расстоянии, которое можно вычислить по пути атакующего пакета, опять же исследовав TTL.[2] Пересечение всех таких наборов для каждой точки наблюдения даст меньший набор - а часто лишь одну сеть, - из которого могла исходить атака, как показано на рис. 17-2.

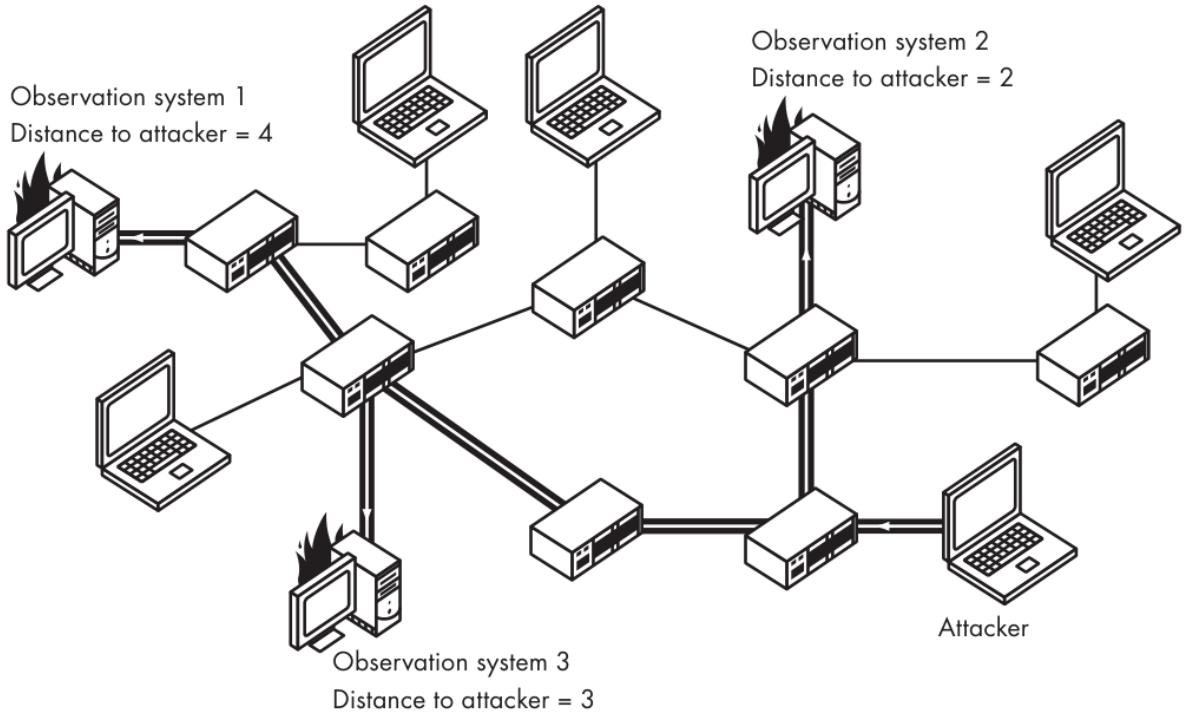


Рисунок 17-2. Наивная сетевая триангуляция: всем наблюдениям соответствует только один источник. Атакующий может подделывать адреса источника, но не способен обмануть жертв.

Возможность самостоятельно выполнить трассировку освобождает нас от безусловной зависимости от провайдеров и помогает точно установить, кто атакует или зондирует нашу сеть, - а возможно, и выяснить зачем.

Хотя этому подходу помешать гораздо труднее, чем обычному обнаружению подделок, умный атакующий всё ещё может обмануть наблюдателя, случайно выбирая разные TTL или диапазоны TTL для каждой цели. Правда, сейчас мы не знаем инструментов, которые это делают, но положение может измениться.

Битва проиграна? Нет - существует способ не дать нарушителям так нас одурачить.

Анализ нагрузки сети

Решение под названием «анализ нагрузки сети» появилось в прекрасном исследовании, представленном Хэлом Бранчем и Биллом Чесвиком на конференции LISA в 2000 году.[3] Бранч и Чесвик предложили интересное применение данных о древовидной топологии сети, подобных ранее показанному графу на рис. 17-1, полученных для конкретного места. Они придумали, как с помощью этих сведений обнаруживать источник определённого вида поддельного трафика - отказа в обслуживании. Сам подход довольно прост и основан на предположении, что такая атака нагружает не только систему, против которой проводится, но и промежуточные маршрутизаторы, а жертва может измерить эту нагрузку извне и использовать её, чтобы почти буквально вернуться назад и найти клубок, потянув за провод.

Для проверки нагрузки сетевых связей сначала строят или получают дерево связей от своего местонахождения до всех сетей Интернета, а при атаке проходят по последовательным ветвям этой древовидной структуры. Для каждой ветви, которая в действительности обозначает соединение с маршрутизатором более высокого порядка, можно последовательно измерять сетевую нагрузку узла, отправляя испытательный трафик соответствующему маршрутизатору или

через него. В данной статье применяется UDP-служба chargen, но можно использовать запросы ICMP или любые другие сообщения. Более нагруженный узел выбирается как вероятный кандидат, через который поступает трафик, после чего перечисляются и проверяются все исходящие из него ветви, пока трафик не будет прослежен до источника.

Рисунок 17-3 показывает простой сценарий обратной трассировки. На первом этапе атакованная система во время атаки пытается измерить производительность трёх ближайших интернет-маршрутизаторов и заключает, что первый, самый верхний, насыщен сильнее всего.

На основании этих сведений жертва решает проверять только маршрутизаторы, непосредственно соединённые пирингом с данным устройством. На рисунке нужно проверить лишь три устройства: остальные шесть не проверяются, поскольку пиринга с ним не имеют; и снова сильнее всего нагружено первое. Процесс продолжается, пока конечной точкой не будет признан маршрутизатор, непосредственно подключённый к конкретной сети, физическое местонахождение и владельца которой можно выяснить по публичным базам данных.

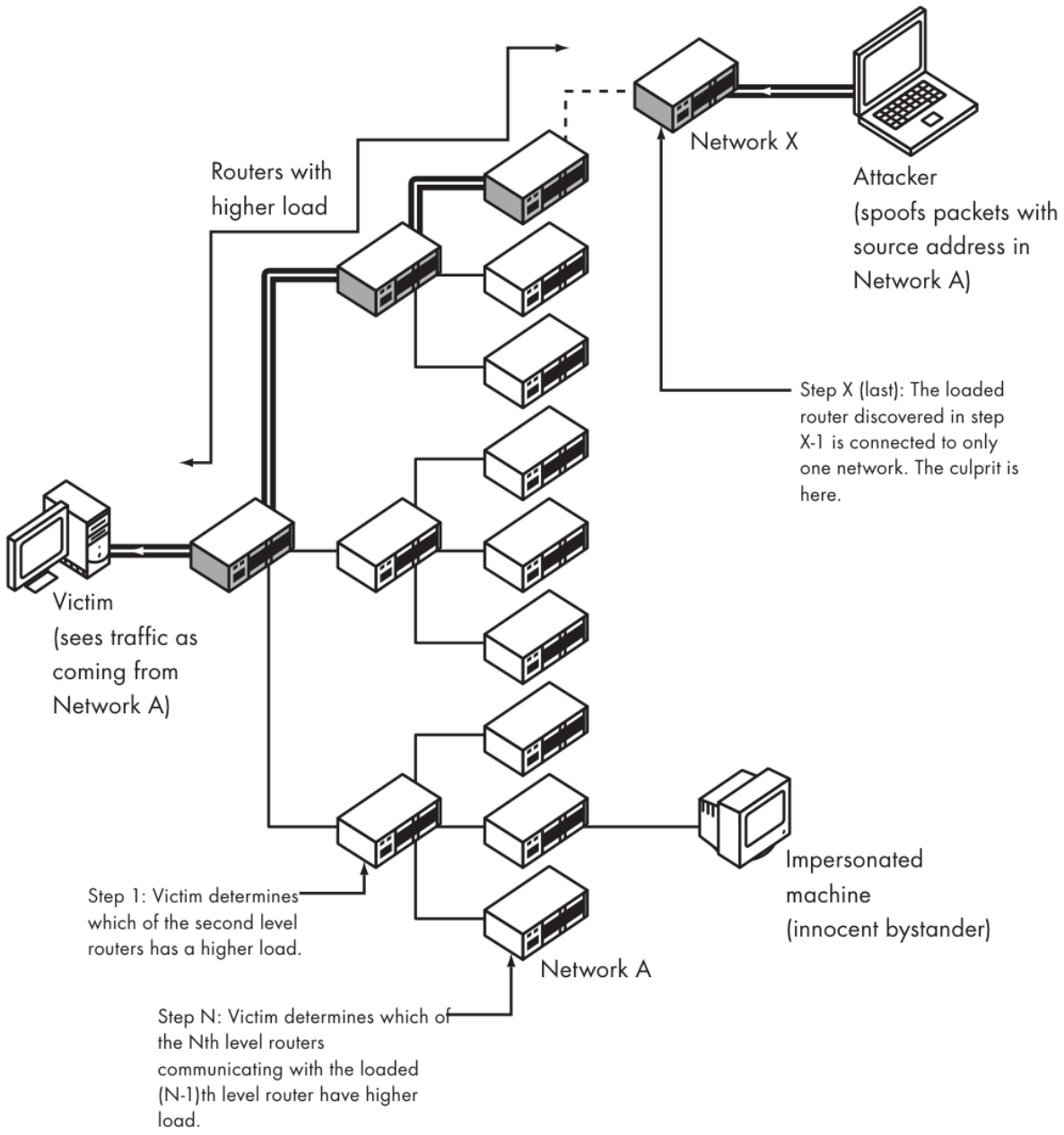


Рисунок 17-3. Рекурсивная обратная трассировка атаки с помощью данных о топологии сети и проверки нагрузки

Возникает возможная проблема: некоторые устройства могут быть сильно нагружены по причинам, не связанным с обработкой DoS-трафика; у других может быть много свободных циклов CPU, и пересылка вредоносного трафика почти на них не повлияет.

Для решения проблемы исследователи предлагают создать искусственную кратковременную нагрузку на маршрутизатор, генерируя дополнительный трафик, а затем проследить, как проверка влияет на пропускную способность и задержку DoS-запросов. Если конкретное устройство действительно участвует в пересылке вредоносных пакетов, после его нагрузки темп атаки должен снизиться. Нагрузку, опять же, вероятно, создают дополнительными фиктивными запросами TCP, UDP или ICMP, рассчитанными скорее на потребление мощности CPU устройства, чем на насыщение его интерфейсов. Следовательно, корреляция должна появляться только на ветвях, участвующих в обслуживании вредоносного трафика.

Эта блестящая и простая схема успешно применялась в испытательных средах. Однако при рассмотрении её использования в реальном мире возникают этические соображения, поскольку она требует взаимодействовать с маршрутизаторами и создавать на них дополнительную нагрузку.

Пища для размышлений

Главная трудность применения рассмотренных в этой главе методов для выслеживания атакующих в том, что для каждого местонахождения нужно строить и обновлять карты сети. Не сразу ясно, как часто их следует освежать и какие методы окажутся самыми надёжными и наименее навязчивыми.

Ещё одна возможная проблема - избыточность значительной части магистральной инфраструктуры Интернета. Некоторые альтернативные маршруты могут выбираться только при отказе или насыщении основного, хотя иногда переключение происходит в ходе балансировки нагрузки. Поэтому некоторые эмпирические карты способны устареть за считаные минуты или часы, хотя такие случаи не слишком распространены.

В конечном счёте, хотя частное, индивидуальное применение различных способов обнаружения подделок может быть весьма успешным, прежде чем развернуть такие техники в большом масштабе, предстоит ответить на множество открытых вопросов - и не все они относятся к техническим проблемам.

Сноски

[1] [назад] В таком подходе TTL следует сравнивать с некоторым допуском, поскольку внутри внутренних сетей может быть несколько дополнительных переходов. Кроме того, некоторые маршруты асимметричны, и их длины способны слегка различаться в зависимости от направления обмена трафиком.
[2] [назад] Даже если инструмент использует случайные TTL, при достаточном числе наблюдаемых пакетов в каждом месте назначения - а это возможно почти всегда - расстояние можно оценить по наибольшему замеченному TTL. Например, если сканер выбирает начальные TTL случайно в диапазоне от 32 до 255, но среди нескольких тысяч полученных в месте назначения пакетов ни один не имел TTL больше 247, то хост, скорее всего, находится на расстоянии `255 - 247 = 8` систем.

Глава 18. Наблюдая за пустотой

Когда смотришь в бездну, то, что нас не убивает, делает нас сильнее

Мы рассмотрели множество способов обнаруживать сведения и перехватывать данные, наблюдая за связью между двумя системами или побочными последствиями такой связи. Однако история на этом не заканчивается. Иногда, отведя глаза от цели, которую надеемся прозондировать, мы способны увидеть ещё больше.

Целый набор методов, обычно называемых «наблюдением за чёрной дырой», посвящён наблюдению и анализу нежелательного либо незапрошенного трафика, который случайно, ошибочно или в искажённом виде приходит в определённое место назначения. Чаще всего эти методы сводятся к простому запуску утилиты дампа пакетов и последующему кропотливому анализу и построению теорий о каждом отдельном наблюдении. Хотя в идеальном мире мы не должны ничего узнавать, ища данные там, где их быть не должно, в действительности эти методы позволяют собирать обильные крупицы сведений и бесценные намёки на состояние сети в целом. Пусть сведения в основном случайны и мы не можем выбирать, кого слушать, усилия всё равно способны принести пользу.

Тактика прямого наблюдения

Одно из применений наблюдения за чёрной дырой - обнаружение и анализ глобальных тенденций атак. Многие хакеры в чёрных шляпах, получив новые методы атаки, часто просто сканируют большие блоки сетевых адресов в поиске уязвимых целей, которые можно скомпрометировать и в итоге использовать для незаконной деятельности - предположительно, чтобы собирать транзитные хосты[1] или строить сети атакующих дронов для автоматических атак. Наблюдение за чёрной дырой может предупреждать нас о новых уязвимостях, эксплуатируемых в реальном мире: достаточно заметить рост обычной активности сетевого сканирования из разных источников.

Многие сетевые администраторы развёртывают наблюдение за чёрной дырой. Иногда они сочетают его с приманками - в сеть выставляется фиктивная система-«наживка», которая ловит атакующих, перехватывает их инструменты и выявляет приёмы[1], - получая систему раннего предупреждения, которая позволит им первыми узнать о надвигающихся вспышках червей и другого вредоносного ПО. Трафик чёрной дыры можно также применять для калибровки «уровня шума» и более эффективного обнаружения направленных атак на ваши серверы, не реагируя на автоматическую, неизбирательную вредоносную активность.

Такие исследователи, как Даг Сонг и Хосе Назарио - последний совсем недавно в книге *Defense and Detection Strategies against Internet Worms*[2], - пытались анализировать активность чёрных дыр во время массовых вспышек сетевых червей. Их цель - лучше понять и смоделировать динамику распространения в сети, то есть первоначальное распространение и повторное заражение, а также проверить эффективность и стойкость алгоритмов заражения червей. Исследования помогут разработать будущую защиту от массовых распределённых угроз и одновременно дадут ценные сведения о сегодняшнем состоянии сети. Некоторые полученные результаты показаны на рис. 18-1 - 18-4.

Рисунок 18-1 показывает распространение червя во время вспышки. Данные основаны на числе наблюдаемых попыток атаки TCP-порта 137 - части реализации Windows NetBIOS, которая по умолчанию установлена на всех компьютерах Windows и служит целью многих видов самораспространяющегося вредоносного ПО. Обратите внимание: после недели первоначального распространения, когда и число заражённых площадок-источников, и число атакованных систем в наблюдаемой сети чёрной дыры стабильно и быстро росло, внезапно наступает период стабилизации продолжительностью больше месяца с резкими пиками и провалами. Такой отпечаток распространения исключительно характерен для конкретного червя и условий сети, где он работает; он отражает также тонкости алгоритмов выбора целей и заражения, применённых автором.

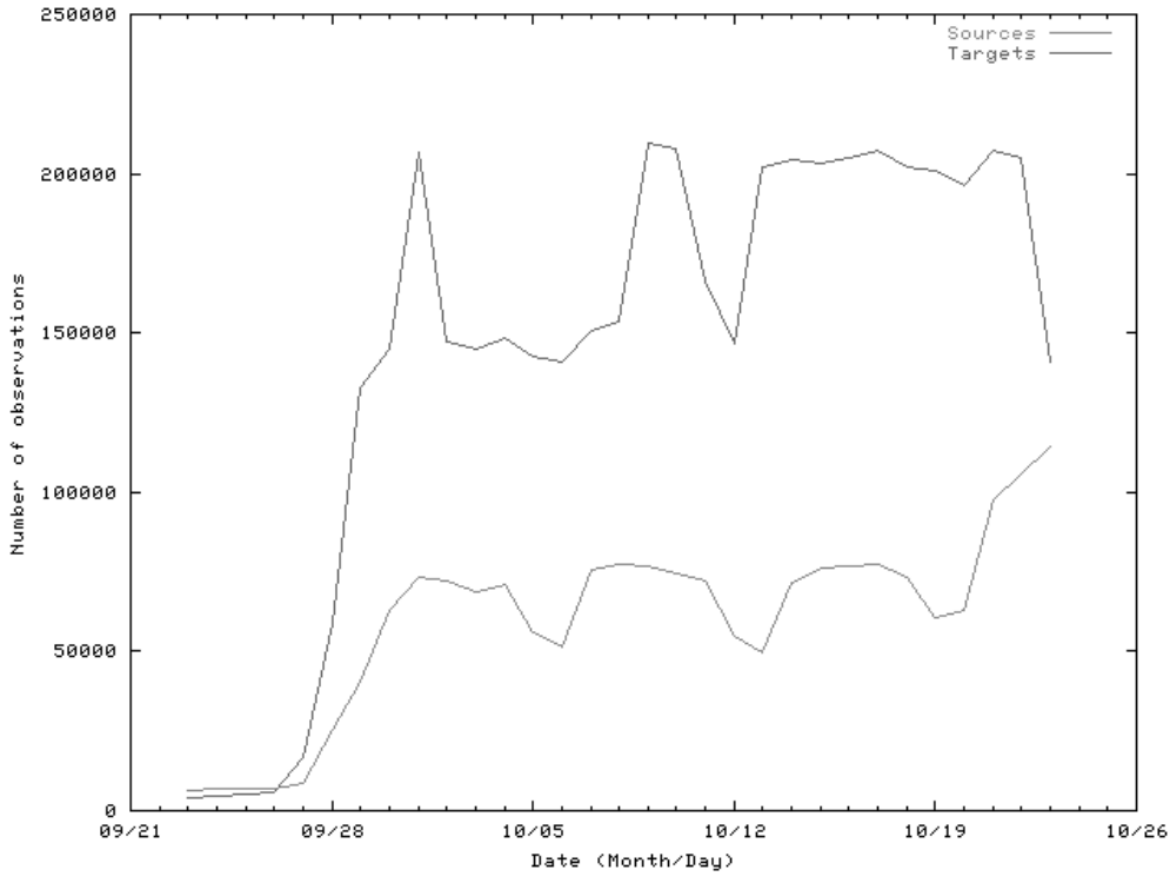


Рисунок 18-1. Характеристики распространения червя Windows

Рисунок 18-2 показывает другую сторону алгоритма распространения червя и изображает свойства алгоритма выбора цели. В данном случае популярный червь, нацеленный на серверы Microsoft SQL, по-видимому, довольно непрерывно охватывает адресное пространство, хотя адреса с октетами примерно от 200 до 225 выбираются заметно чаще, а значения больше 225 червь, похоже, полностью пропускает.

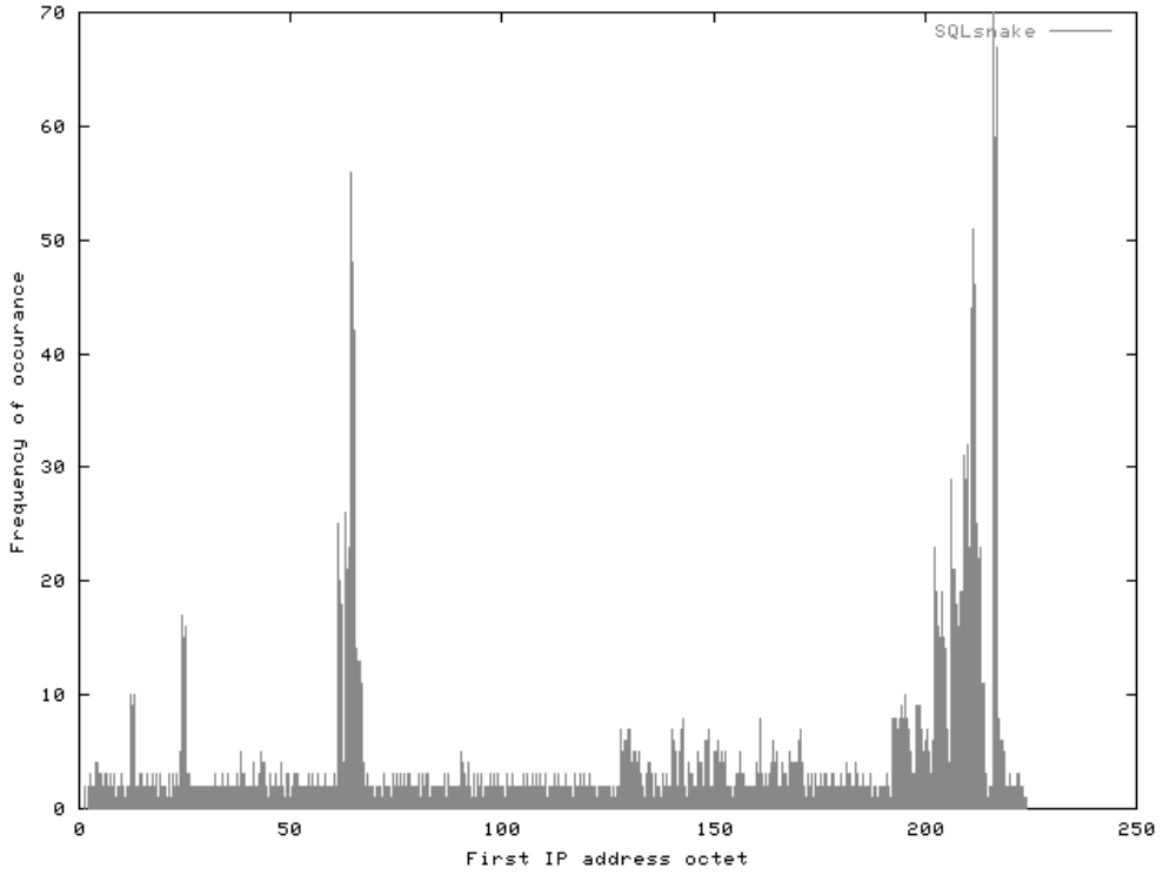


Рисунок 18-2. Гистограмма алгоритма выбора целей червя SQLSnake; обратите внимание на неравномерный, но в целом непрерывный охват адресного пространства

Рисунок 18-3 показывает тот же график для другого сетевого червя - Slapper. Этот червь нацеливался на системы Linux, эксплуатируя изъян популярной криптографической библиотеки OpenSSL. Алгоритм, судя по всему, даёт заметно более равномерный, но гораздо менее непрерывный охват с огромными пробелами в некоторых диапазонах значений.

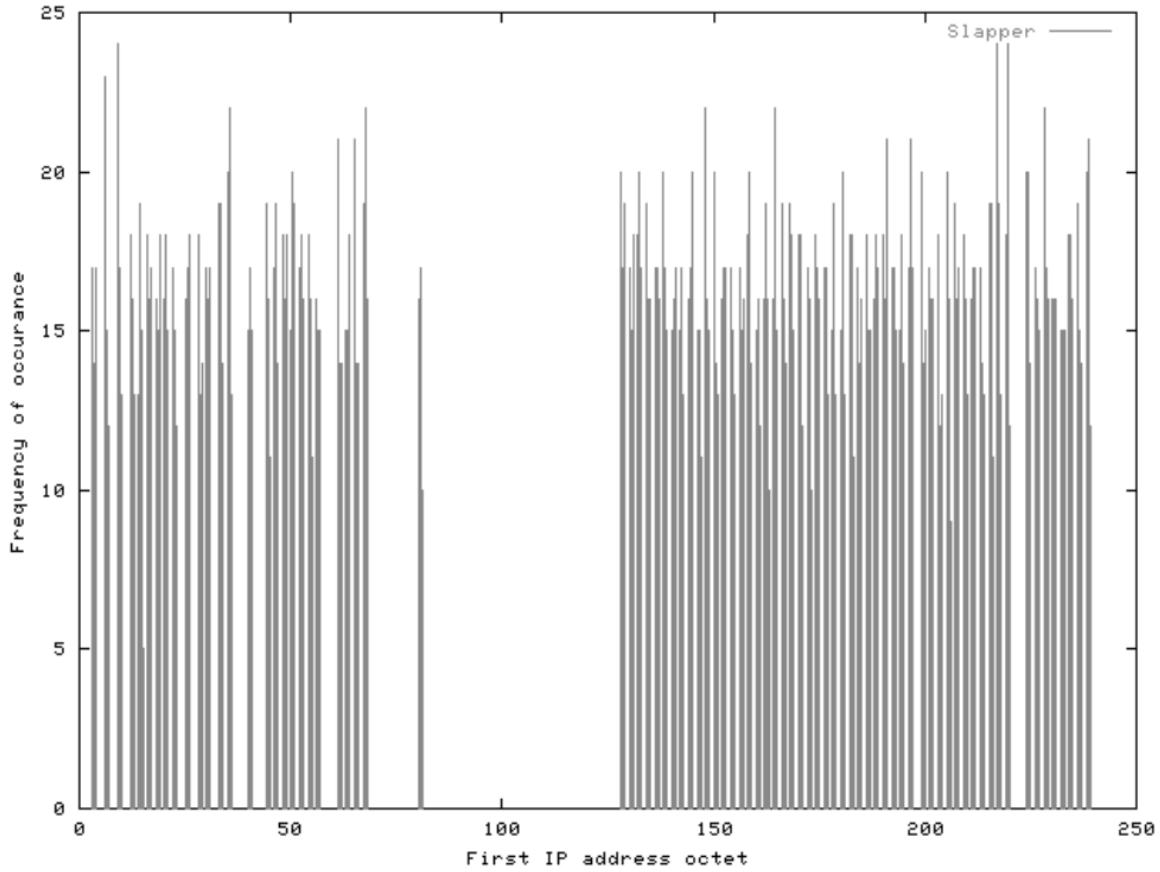


Рисунок 18-3. Гистограмма алгоритма выбора целей червя Slapper. Распределение гораздо равномернее, но охват прерывистый, с пробелами, которые указывают, что младшие значащие биты каждого «случайного» адреса постоянны - возможно, из-за ошибки программирования.

Рисунок 18-4 показывает закономерности стойкости червей во времени. Например, некоторые черви, по мере исправления и обеззараживания систем, неуклонно отмирают, а другие применяют алгоритмы, вызывающие внезапные и повторяющиеся подъёмы и спады, знакомые всем, кто экспериментировал с моделями популяций или эпидемиологии на основе природных явлений.

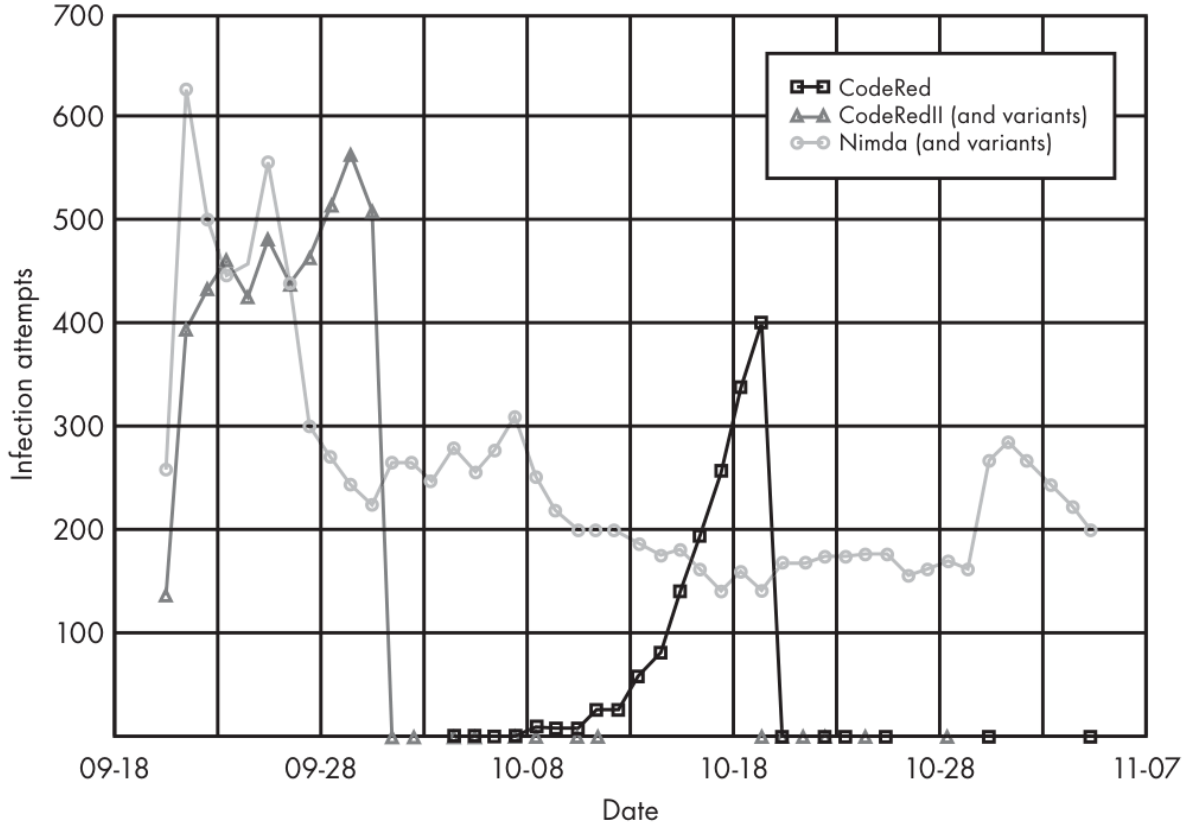


Рисунок 18-4. Стойкость червей во времени. Обратите внимание: для CodeRed нет простой картины «всплеск - спад», а модель ведёт себя подобно модели биологической популяции.

Как стремятся показать Хосе и его коллеги, наблюдение за чёрной дырой может быть не просто рутинной и, вероятно, совершенно ненужной деятельностью, но и отличным способом открыть тайную жизнь всего вредоносного. Увы, история на этом не заканчивается. Наблюдая лишь трафик, который считаем направленным на нас, мы упускаем самые интересные крупицы данных.

Анализ остаточного трафика атак

Другое применение наблюдения за чёрной дырой основано на трафике, который изначально вообще не был направлен на нас, а является лишь побочным последствием другой активности.

Здесь мы видим, как некоторые обычные схемы разведки и атак используют подделку адреса, чтобы скрыть личность атакующего. Предполагается, что администратору будет трудно отличить отвлекающий трафик с фиктивных адресов от настоящих проб атакующего. Как я показал в предыдущих главах, этот подход не гарантирует атакующему полной анонимности, но для успешного распознавания подделки администратор должен уже во время атаки вести подробные журналы и принимать дополнительные меры. Поскольку эти процедуры реализованы не всегда, атакующие часто способны довольно эффективно подделывать свои атаки и оставаться вне света прожекторов.

Независимо от того, подделаны пакеты или нет, атакованная система добросовестно ответит на все запросы, включая якобы поступившие с вымышленных адресов. Однако отправителю вернутся только ответы на пакеты с правильным адресом источника; все остальные пробы породят ответы, рассеянные по всему Интернету, и нередко мы способны их поймать.

Хотя получение такого ошибочно направленного пакета может казаться маловероятным, помните: в ответ на сканирование с приманками или SYN-флуд создаётся значительное число пакетов SYN+ACK, RST+ACK и RST. Адресное пространство Интернета кажется огромным, а в подобных атаках обычно участвуют миллионы пакетов, но со временем некоторые из них с большой вероятностью достигнут каждого отдельного сетевого блока. Вероятность отскока одного случайно созданного поддельного пакета на конкретный адрес равна лишь 1 к 4 294 967 296, то есть 1 к 2^32, однако обычная небольшая подсеть, выделенная малой компании или организации, как правило, состоит из 256 адресов - сеть класса C или эквивалентная, - и потому вероятность возрастает до 1 к 16 777 216, то есть 1 к 2^24. Её можно увеличить ещё больше, исключив диапазоны адресов, которые заведомо зарезервированы для особых целей или иначе не представляют интереса и потому не используются в определённых видах атак.

Поскольку размер одного SYN-пакета составляет около 40 байтов, а в массе они хорошо сжимаются, обычный сетевой канал случайного атакующего передаёт примерно от 10 до 150 килобайтов на IP-уровне в секунду - соответственно слабый DSL и линия T1, - и за это время он способен вытолкнуть от 250 до почти 3000 пакетов, то есть от 900 000 примерно до 10 000 000 пакетов в час.[2]

Чтобы обычная DoS-атака дала заметный результат и причинила жертве серьёзные неудобства, её, как правило, приходится вести несколько часов или дней. Атакующий хочет досаждать жертве как можно дольше. В результате отправляются десятки или сотни миллионов пакетов, порождающих сопоставимое число ответов SYN+ACK или RST+ACK.

Из-за этого огромного объёма трафика вполне разумно ожидать, что даже сравнительно небольшая организация заметит остаточный трафик слабой атаки SYN-флуд непосредственно во время неё, даже если хост-получатель отбрасывает многие атакующие пакеты. Более того, администраторы, способные наблюдать сети, эквивалентные классу B - 65 356 адресов, обычно принадлежащих крупным компаниям, провайдерам, исследовательским учреждениям и так далее, - смогут быстро улавливать гораздо меньшие события.

Поскольку все остаточные ответы при поддельной DoS-атаке содержат некоторые подробности сообщений, сфабрикованных атакующим, чтобы изначально вызвать эти ответы, - например, номера портов и последовательностей, сведения о времени и так далее, - из них можно извлечь важные сведения о виде и масштабе атаки. По этим ответам можно определить, была ли целью конкретная служба, сколько систем атаковано, какая полоса доступна атакующему и какой инструмент применялся - исследовав выбор исходного порта, выбранные номера последовательностей и закономерности «случайных» IP.[3]

Наконец, анализируя источники этих ответов-рикошетов, можно заметить, что конкретный сегмент сети атакован, или выявить глобальные «тенденции враждебности», чтобы, возможно, лучше подготовиться, если целью становится определённая отрасль или компания. По этим сведениям можно также узнать об атаках, которые жертва скрывает, или распознать ложные заявления об атаках. Утверждения, будто некоторые цели атакованы кибертеррористами, порой служат рекламным трюком для оправдания финансовых потерь или продвижения определённой политической программы. Недавно некоторые эксперты обвинили SCO в том, что компания сама отключила свои серверы и притворилась жертвой скоординированной DoS-атаки, чтобы дискредитировать сообщество пользователей Linux.

Обнаружение искажённых или ошибочно направленных данных

Это применение наблюдения за чёрными дырами основано на отслеживании трафика, который, судя по всему, совершенно бессмыслен, но всё же достигает конкретного места назначения. Чтобы яснее показать проблему, позвольте небольшое отступление.

В 1999 году мы с группой друзей и коллег из Польши начали скромный внерабочий проект. Мы хотели отыскать источник труднообъяснимого набора пакетов RST+ACK, которые заметили в обслуживаемых нами сетях, и в целом наблюдать необычные и незапрошенные закономерности трафика, приходящего в неиспользуемые сегменты сети. Это было очень весело и, как вы можете представить, породило немало догадок, когда мы пытались разумно объяснить самые необычные случаи. Исследование также позволило больше узнать об окружающем мире: мы встречали чрезвычайно странный и вроде бы необъяснимый трафик, который после правильного анализа раскрывал больше сведений о грандиозных заговорах нашего проводного мира.

Хотя проект формально забросили, в итоге он превратился в мой частный «Музей сломанных пакетов»[3] - полушутливую веб-страницу, посвящённую выслеживанию, документированию и объяснению пакетов, которые либо вообще не должны были достичь места назначения, либо не должны были выглядеть так, как выглядели. Заявленная цель музея такова:

> Цель этого музея - дать приют странным, нежеланным, искажённым пакетам - брошенным и обречённым уродцам природы, - которых мы, простые смертные, встречаем на извилистых тропах великого путешествия под названием жизнь. Наши экспонаты - или, если угодно, обитатели - часто лишь тени того, чем были до встречи с враждебным, неисправным маршрутизатором. Некоторые родились изуродованными в глубинах сломанной реализации стека IP. Другие были обычными пакетами, совсем как их друзья - вы или я, - но заблудились в поиске высшего смысла своего существования и пришли туда, где мы никогда не должны были их увидеть. В каждом случае мы пытаемся открыть уникальную историю жизни пакета и помочь вам понять, как трудно быть одиноким посланником во враждебной вселенной битов и байтов.

Именно к этому сводится последний вид наблюдения за чёрной дырой. Сначала задача может казаться бессмысленной, но такое предположение глупо. Музей позволил пассивно раскрыть тёмные тайны различных проприетарных устройств и хорошо защищённых сетей, а проведение похожего эксперимента в другом месте, несомненно, принесло бы такие же или более значительные достижения.

Среди экспонатов моего музея есть следующие чудеса:

• Пакеты из сетей с веб-ускорителем, маршрутизатором или межсетевым экраном определённого вида; устройство дописывает, удаляет или иначе искажает часть данных. Хороший пример - изъян некоторых устройств Nortel CVX, ответственный за периодическое удаление TCP-заголовков из пакетов, как рассматривалось в главе 11. Уникальность изъяна позволяет многое узнать о множестве удалённых сетей, не выходя наружу и не зондируя их.
• Несколько экспонатов линейного шума: пакеты содержат либо полную бессмыслицу, либо данные, которые совершенно точно не относились к конкретному соединению. Один из самых удивительных экспонатов - незапрошенный трафик с данными, похожими на дамп содержимого зоны DNS `.de`, то есть перечень всех доменов Германии. Трафик не мог прийти откуда угодно, потому что простые смертные не имеют права получать такой список. Он должен был исходить от уполномоченной стороны, способной получить и передать эти данные, а исказил его либо отправитель, либо устройство где-то на пути. Хотя все такие случаи мало проясняют природу сетевых происшествий, они нередко обогащают наблюдателя неожиданными - и часто ценными - находками.

Среди других примечательных экспонатов были случаи очевидного шпионажа, замаскированного под обычный трафик, и множество иных ошибок программирования или сетевой работы. Но хватит хвастаться: если вас непреодолимо тянет узнать больше, посетите http://lcamtuf.coredump.cx/mobp/.

Пища для размышлений

Многие считают наблюдение за чёрной дырой просто ещё одним способом обнаружить атаки на свои системы - и, возможно, дорогим способом, учитывая дефицит публичного адресного пространства IP. Но настоящая ценность техники в том, что она позволяет не только выявлять известные атаки - это столь же хорошо делается во многих других местах без расходования IP-пространства, - но и обнаруживать и анализировать тонкие закономерности, которые иначе потерялись бы ниже «уровня шума» интенсивно используемой сети.

Естественно, такое наблюдение за чёрной дырой непросто и остаётся дорогим. Требуется время, чтобы научиться находить иголку в стоге обычной активности червей и чёрных шляп, которая в достаточно обширной сети, как правило, не имеет значения за пределами статистической отчётности.

Но ради радости от наконец найденной иголки часто стоит попробовать.

Сноски

[1] [назад] Транзитный хост - система, используемая как промежуточный переход для дальнейших атак или иной незаконной деятельности, например рассылки спама. Этот приём затрудняет выслеживание конечного нарушителя, поскольку его источник непосредственно неизвестен, а для поиска требуется сотрудничество нескольких администраторов или юрисдикций.
[2] [назад] Учтите, что решительные, опытные атакующие, хорошо владеющие атаками отказа в обслуживании, часто управляют десятками или сотнями узлов-«зомби», что резко увеличивает эту оценку.
[3] [назад] Например, из-за ошибок в коде некоторые инструменты «подделывают» пакеты только с чётных или нечётных IP-адресов. Анализы, подобные проведённым Хосе Назарио и другими, обычно выявляют инструменты атаки столь же хорошо, как идентифицируют червей.

Заключительные слова

Где книга подходит к завершению

Здесь заканчивается книга, но, надеюсь, начинается ваше путешествие. Я с гордостью вёл вас по миру сложных и необычных проблем безопасности, которые люблю больше всего, и надеюсь, что вы разделили моё увлечение. Будь вы опытным специалистом по безопасности - возможно, более опытным и знающим, чем я, - или лишь энтузиастом, открывающим эту область, надеюсь, я дал вам новый взгляд на безопасность: как на самостоятельный вызов и искусство, а не набор препятствий, которые требуется устранить или обойти.

Понимание тонких отношений между вроде бы не связанными компонентами и процессами позволяет эффективно решать самые опасные и распространённые проблемы безопасности, а также лучше оценивать и снижать повседневные риски. Проблемы безопасности следует считать функцией решения практически любой задачи в мире ИТ, сколь бы тривиальной или ограниченной по масштабу она ни была, а не неблагоприятными обстоятельствами ведения дел. Только видя волшебство и очарование взаимодополняющих вселенных и тонкие способы их взаимодействия, мы можем избежать рутины и начать по-настоящему наслаждаться работой или понимать своё увлечение.

Но сейчас не время и не место брать высокие ноты.

Спасибо за игру.

Библиографические примечания

Глава 1

[1] [назад] Alan Turing, «О вычислимых числах с приложением к проблеме разрешения», *Proceedings of the London Mathematical Society*, серия 2, 42 (1936).
[2] [назад] R.L. Rivest, A. Shamir, L. Adleman, «Метод получения цифровых подписей и криптосистем с открытым ключом», Massachusetts Institute of Technology (1978).
[3] [назад] Ueli M. Maurer, «Быстрое получение простых чисел и безопасных криптографических параметров открытого ключа», Institute for Theoretical Computer Science, ETH Zurich, Switzerland (1994).
[4] [назад] Donald E. Knuth, *Искусство программирования, том 2: Получисленные алгоритмы*, 3-е изд., Addison-Wesley (1997).
[5] [назад] H. Krawczyk, «Как предсказывать конгруэнтные генераторы», *Journal of Algorithms* 13, no. 4 (1992).
[6] [назад] S. Bakhtiari, R. Safavi-Naini, J. Pieprzyk, «Криптографические хеш-функции: обзор», Centre for Computer Security Research, Department of Computer Science, University of Wollongong, Australia (1995).
[7] [назад] Dawn Xiaodong Song, David Wagner, Xuqing Tian, «Временной анализ нажатий клавиш и атаки по времени на SSH», University of California, Berkeley (2001).
[8] [назад] Claude E. Shannon, «Предсказание и энтропия печатного английского текста», *Bell Systems Technical Journal* 3 (1950).
[9] [назад] Benjamin Jun, Paul Kocher, «Генератор случайных чисел Intel», Cryptography Research Inc. (1999).
[10] [назад] «Оценка генератора случайных чисел VIA C3 Nehemiah», Cryptography Research Inc. (2003).
[11] [назад] Michael A. Hogye, Christopher T. Hughes, Joshua M. Sarfaty, Joseph D. Wolf, «Анализ осуществимости атак по времени нажатий клавиш через соединения SSH», CS588 Research Project, School of Engineering and Applied Science, University of Virginia (2001).

Глава 2

[1] [назад] Yurii Rogozhin, «Универсальная машина Тьюринга с 22 состояниями и 2 символами», *Romanian Journal of Information Science and Technology* 1, no. 3 (1998).
[2] [назад] Milena Milenkovic, Aleksandar Milenkovic, Jeffrey Kulick, «Разоблачение тайны предсказателей ветвлений Intel», Electrical and Computer Engineering Department, University of Alabama in Huntsville (2002).
[3] [назад] Paul C. Kocher, «Атаки по времени на реализации Диффи-Хеллмана, RSA, DSS и других систем», Cryptography Research Inc. (1999).
[4] [назад] *Справочное руководство программиста Intel 80386*, раздел 7.2.IMUL, Intel Corp. (1986).
[5] [назад] E. Biham, A. Shamir, «Дифференциальный анализ сбоев: определение структуры неизвестных шифров, запечатанных в защищённых от вмешательства устройствах» (1996).

Глава 3

[1] [назад] Wim van Eck, «Электромагнитное излучение видеодисплейных устройств: риск подслушивания?», PTT Laboratories, Netherlands (1985).
[2] [назад] Ian A. Murphy, «Кто слушает?», IAM/Secure Data Systems (1988, 1997).
[3] [назад] Winn Schwartau, *Информационная война*, 2-е изд., Thunder's Mouth Press, New York (1996).

Глава 5

[1] [назад] John A.C. Bingham, *Теория и практика проектирования модемов*, Wiley-Interscience (1988).
[2] [назад] Electronic Industries Association, Engineering Department, «Интерфейс между оконечным оборудованием данных и оконечным оборудованием канала данных с последовательным обменом двоичными данными» (1991).
[3] [назад] Charles E. Spurgeon, *Ethernet: полное руководство*, O'Reilly and Associates (2000).
[4] [назад] Joe Lughry, David A. Umphress, «Утечка информации через оптическое излучение», *ACM Trans. Info. Sys. Security* 5, no. 3 (2002).
[5] [назад] Adi Shamir, Eran Tromer, «Акустический криптоанализ: о любопытных людях и шумных машинах». На момент написания предварительная презентация доступна по адресу http://www.wisdom.weizmann.ac.il/~tromer/acoustic/ (2004).

[6] [назад] Paul Kocher, Joshua Jaffe, Benjamin Jun, «Дифференциальный анализ энергопотребления», Cryptography Research, Inc. (2000).

Глава 6

[1] [назад] J. Postel, J. Reynolds, «RFC-1042: стандарт передачи дейтаграмм Internet Protocol через сети IEEE 802», Network Working Group, http://www.ietf.org/rfc/rfc1042.txt (1988).
[2] [назад] Ofir Arkin, Josh Anderson, «EtherLeak - утечки информации из заполнения кадров Ethernet», @Stake, http://www.atstake.com/research/advisories/2003/atstake_etherleak_report.pdf (2003).

Глава 7

[1] [назад] David C. Plummer, RFC 826, «Протокол разрешения адресов Ethernet», Network Working Group (1982).
[2] [назад] Louis Senecal, «Атаки уровня 2 и меры их смягчения», Cisco (2002).

Глава 8

[1] [назад] J. Case, M. Fedor, M. Schoffstall, J. Davin, RFC 1157, «Простой протокол управления сетью», Network Working Group (1990).
[2] [назад] Institut für Bankinnovation GmbH, «PSYLock: психометрический метод аутентификации по поведению при наборе текста», http://pc50461.uni-regensburg.de/ibi/de/leistungen/research/projekte/einzelprojekte/psylock_english.htm (2003).
[3] [назад] Solar Designer, Dug Song, «Пассивный анализ трафика SSH (Secure Shell)», Openwall Project, http://www.openwall.com/advisories/OW-003-ssh-traffic-analysis (2001).
[4] [назад] Nikita Borisov, Ian Goldberg, David Wagner, «Перехват мобильной связи: небезопасность 802.11» (2001).

Глава 9

[1] [назад] J. Postel, University of Southern California, «RFC 791: Internet Protocol», Network Working Group (1981).
[2] [назад] J. Postel, University of Southern California, «RFC 796: отображения адресов», Network Working Group (1981).
[3] [назад] J. Mogul, S. Dearing, «RFC 1191: обнаружение MTU пути», Network Working Group (1990).
[4] [назад] J. Postel, University of Southern California, «RFC 768: User Datagram Protocol», Network Working Group (1980).
[5] [назад] J. Postel, University of Southern California, «RFC 793: Transmission Control Protocol», Network Working Group (1981).
[6] [назад] S. Bellovin, «RFC1948: защита от атак на номера последовательностей», Network Working Group (1996).
[7] [назад] V. Jacobson, B. Braden, «RFC1232: расширения TCP для высокой производительности», Network Working Group (1992).
[8] [назад] M. Mathis, J. Mahdavi, S. Floyd, A. Romanow, «RFC2018: параметры выборочного подтверждения TCP», Network Working Group (1996).
[9] [назад] B. Braden, «RFC1644: T/TCP - расширения TCP для транзакций - функциональная спецификация», Network Working Group (1994).
[10] [назад] J. Postel, University of Southern California, «RFC 792: Internet Control Message Protocol», Network Working Group (1981).
[11] [назад] Lance Spitzner, *Приманки: выслеживание хакеров*, Addison-Wesley Publishing Company (2002).
[12] [назад] R. Morris, «Слабость в программном обеспечении TCP/IP 4.2BSD UNIX», AT&T Bell Laboratories (1985).

Глава 10

[1] [назад] Michal Zalewski, «Странные аттракторы и анализ номеров последовательностей TCP/IP», BindView Corporation, http://www.bindview.com/Support/RAZOR/Papers/2001/ (2001).
[2] [назад] S. Bellovin, «Защита от атак на номера последовательностей», Network Working Group, http://www.ietf.org/rfc/rfc1948.txt (1996).
[3] [назад] Joe Steward, «Отравление кэша DNS: следующее поколение», http://www.lurhq.com/dnscache.pdf (2002).

Глава 11

[1] [назад] Elizabeth D. Zwicky, Simon Cooper, D. Brent Chapman, *Построение межсетевых экранов Интернета*, O'Reilly & Associates (2000).
[2] [назад] G. Ziemba, D. Reed, P. Traina, «RFC 1858: соображения безопасности при фильтрации фрагментов IP», Network Working Group (1995).
[3] [назад] Uriel Maimon, «Скрытное сканирование портов TCP», *Phrack Magazine* no. 49 (1996).
[4] [назад] J. Postel, J. Reynolds, «RFC959: File Transfer Protocol», Network Working Group (1985).
[5] [назад] Mikael Olson, «Распространение уязвимости FTP ALG на любой FTP-клиент», список рассылки VULN-DEV, http://www.securityfocus.com/archive/82/50226 (2000).
[6] [назад] Michal Zalewski, «Уязвимость IP-маскарадинга ядра Linux», Bindview Corporation, http://razor.bindview.com/publish/advisories/adv_LkIPmasq.html (2001).
[7] [назад] R. Braden (редактор), «RFC1122: требования к хостам Интернета - уровни связи», Network Working Group (1989).

Глава 13

[1] [назад] Salvatore Sanfilippo, «Новый метод сканирования TCP», Bugtraq, http://seclists.org/bugtraq/1998/Dec/0082.html (1998).

Глава 14

[1] [назад] World Wide Web Consortium, http://www.w3c.org/History.html.
[2] [назад] Vannevar Bush, «Как мы можем мыслить», *Atlantic Monthly* 176, no. 1 (1945): 101-08.
[3] [назад] Tim Berners-Lee, «Основы HTTP», http://www.w3c.org/Protocols/HTTP/HTTP2.html.
[4] [назад] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, «RFC2616: HyperText Transfer Protocol - HTTP/1.1», Network Working Group (1999).
[5] [назад] Различные источники, ссылки процитированы после http://usability.gov/guidelines/softhard.html:
• Anna Bouch, Allan Kuchinsky, Nina Bhatti, «Качество в глазах смотрящего: выполнение пользовательских требований к качеству обслуживания Интернета», CHI (2000);
• Martin, Corl, «Влияние времени отклика системы на производительность пользователя», *Behaviour and Information Technology*, vol. 5, no. 1, 3-13 (1986);
• Jakob Nielsen, «Десять главных ошибок веб-дизайна», http://www.useit.com/alertbox/9605.html (1996);
• Nielsen, «Потребность в скорости», http://www.useit.com/alertbox/9703a.html (1997);
• Nielsen, «Изменения в удобстве веб-сайтов с 1994 года», http://www.useit.com/alertbox/9712a.html (1997);
• Nielsen, «Десять главных новых ошибок веб-дизайна», http://www.useit.com/alertbox/990530.html (1999).
[6] [назад] Kristol, Montulli, «RFC2109: механизм управления состоянием HTTP», Network Working Group (1997).
[7] [назад] Martin Pool, «Проблемы конфиденциальности HTTP Cache-Control», Bugtraq, http://cert.uni-stuttgart.de/archive/bugtraq/2000/03/msg00365.html (2000).
[8] [назад] Bamshad Mobasher, Robert Cooley, Jaideep Srivastava, «Автоматическая персонализация на основе анализа использования веб-сайтов», *ACM Communications* vol. 43, no. 8, 142-151 (1999).
[9] [назад] Edward Felten, Michael Schneider, «Атаки по времени на конфиденциальность в Вебе», ACM Conference on Computing and Communications Security (2000).

Глава 15

[1] [назад] Стандарт ISO/IEC 9899, «Язык программирования C», http://plg.uwaterloo.ca/~cforall/N843.ps (1999).
[2] [назад] DISCO, http://www.altmode.com/disco.

Глава 16

[1] [назад] Albert-Laszlo Barabasz, Vincent W. Freeh, Hawoong Jeong, Jay B. Brochman, «Паразитные вычисления», письмо в *Nature* 412 (2001).
[2] [назад] M. Leech, «RFC 3607: новый взгляд на криптоанализ методом китайской лотереи», Network Working Group (2003).

Глава 17

[1] [назад] Bill Cheswick, Hal Burch, Steve Branigan, «Картографирование и визуализация Интернета», http://www.cheswick.com/ches/papers/mapping.ps.gz (2000).
[2] [назад] Despoof, http://razor.bindview.com/tools/desc/despoof_readme.html.
[3] [назад] Hal Brunch, Bill Cheswick, «Трассировка анонимных пакетов до их приблизительного источника», http://www.usenix.org/publications/library/proceedings/lisa2000/burch/burch_html (2000).

Глава 18

[1] [назад] Lance Spitzner, *Приманки: выслеживание хакеров*, Addison-Wesley (2002).
[2] [назад] Jose Nazario, *Стратегии защиты от интернет-червей и их обнаружения*, Artech House (2003).
[3] [назад] Michal Zalewski, «Музей сломанных пакетов», http://lcamtuf.coredump.cx/mobp (2001).

Electronic Frontier Foundation

Electronic Frontier Foundation (EFF) - ведущая организация, защищающая гражданские свободы в цифровом мире. Мы защищаем свободу слова в Интернете, боремся с незаконной слежкой, поддерживаем право новаторов разрабатывать новые цифровые технологии и работаем над тем, чтобы по мере роста применения технологий наши права и свободы расширялись, а не разрушались.

Конфиденциальность

EFF подала в суд на телекоммуникационного гиганта AT&T за предоставление NSA неограниченного доступа к частной связи миллионов клиентов. `eff.org/nsa`

Свобода слова

Проект EFF Coders' Rights защищает право программистов и исследователей безопасности публиковать свои открытия, не опасаясь судебного преследования. `eff.org/freespeech`

Инновации

Проект EFF Patent Busting оспаривает чрезмерно широкие патенты, угрожающие технологическим инновациям. `eff.org/patent`

Добросовестное использование

EFF борется с запретительными стандартами, которые лишили бы вас права принимать эфирные телевизионные передачи и использовать их любым выбранным способом. `eff.org/IP/fairuse`

Прозрачность

EFF разработала инструмент сетевого тестирования Switzerland, чтобы дать людям средства проверки скрытой фильтрации трафика. `eff.org/transparency`

Международная деятельность

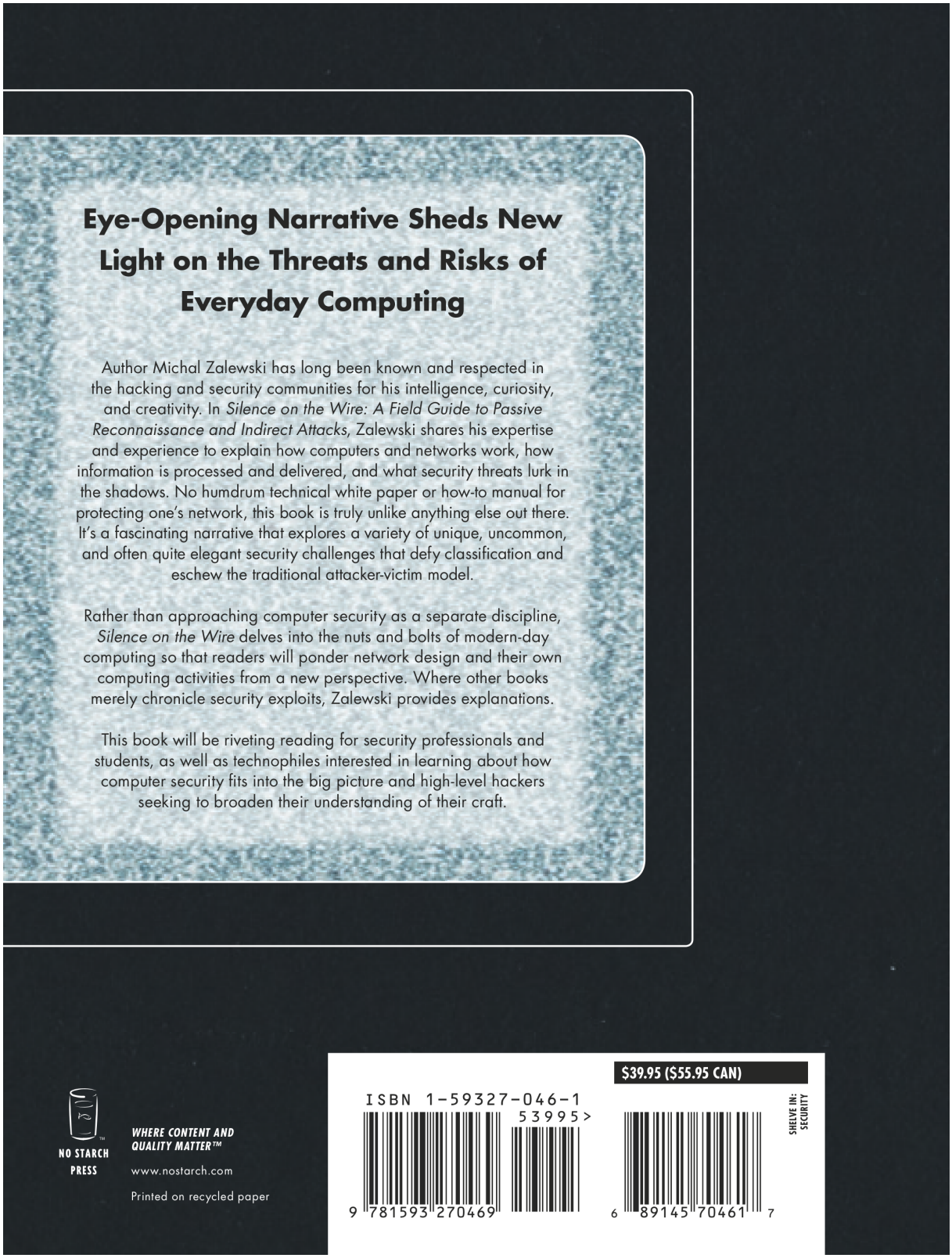
EFF работает над тем, чтобы международные договоры не ограничивали нашу свободу слова, конфиденциальность или цифровые права потребителей. `eff.org/global`

EFF - организация, поддерживаемая своими членами. Вступайте сейчас! `www.eff.org/support`

Обновления

Посетите http://www.nostarch.com/silence.htm, чтобы получить обновления, исправления и другие сведения.

Задняя обложка



Открывающий глаза рассказ по-новому освещает угрозы и риски повседневной работы с компьютерами

Автор Михал Залевский давно известен и уважаем в сообществах хакеров и специалистов по безопасности за ум, любознательность и творческий подход. В книге *Тишина в проводе: полевое руководство по пассивной разведке и косвенным атакам* Залевский делится знаниями и опытом, объясняя, как работают компьютеры и сети, как обрабатываются и доставляются сведения и какие угрозы безопасности таятся в тени. Это не скучный технический документ и не практическое руководство по защите своей сети; книга действительно не похожа ни на что другое. Это увлекательное повествование, исследующее разнообразные уникальные, необычные и зачастую весьма изящные задачи безопасности, которые не поддаются классификации и не укладываются в традиционную модель «атакующий - жертва».

Вместо того чтобы рассматривать компьютерную безопасность как отдельную дисциплину, *Тишина в проводе* погружается в основы повседневных вычислений, чтобы читатели по-новому задумались об устройстве сетей и собственной работе с компьютерами. Там, где другие книги лишь перечисляют подвиги в области безопасности, Залевский даёт объяснения.

Книга станет захватывающим чтением для специалистов по безопасности и студентов, а также для технофилов, которым интересно узнать, какое место компьютерная безопасность занимает в общей картине, и хакеров высокого уровня, стремящихся расширить понимание своего ремесла.

No Starch Press - там, где важны содержание и качество.
www.nostarch.com

Напечатано на переработанной бумаге.

ISBN 1-59327-046-1
Цена: $39.95 ($55.95 CAN)
Рубрика: безопасность