HACKING

The Art of Exploitation

Jon Erickson

NO STARCH
PRESS

HIGH TECH

ХАКИНГ

Искусство эксплойта

Джон Эриксон

СИМВОЛ®

Санкт-Петербург — Москва
2005

Серия «High tech»

Джон Эриксон

Хакинг: искусство эксплойта

Перевод С. Маккавеева

Главный редактор	*А. Галунов*
Зав. редакцией	*Н. Макарова*
Научный редактор	*А. Петухов*
Редактор	*В. Овчинников*
Художник	*В. Гренда*
Корректор	*О. Макарова*
Верстка	*Н. Гриценко*

Эриксон Д.
Хакинг: искусство эксплойта. – Пер. с англ. – СПб: Символ-Плюс, 2005. – 240 с., ил.
ISBN 5-93286-076-6

Это не каталог эксплойтов, а учебное пособие по основам хакинга, построенное на примерах. В нем подробно рассказано, что должен знать каждый хакер и, что важнее, о чем должен быть осведомлен каждый специалист по безопасности, чтобы принять меры, которые не позволят хакеру совершить успешную атаку. От читателя потребуется хорошая техническая подготовка и полная сосредоточенность, особенно при изучении кода примеров. Но это очень интересно и позволит многое узнать. О том, как создавать эксплойты с помощью переполнения буфера или форматных строк, как написать собственный полиморфный шеллкод в отображаемых символах, как преодолевать запрет на выполнение в стеке путем возврата в libc, как перенаправлять сетевой трафик, прятать открытые порты и перехватывать соединения TCP, как расшифровывать данные беспроводного протокола 802.11b с помощью атаки FMS.

Автор смотрит на хакинг как на искусство творческого решения задач. Он опровергает распространенный негативный стереотип, ассоциируемый со словом «хакер», и ставит во главу угла дух хакинга и серьезные знания.

ISBN 5-93286-076-6
ISBN 1-59327-007-0 (англ)

Издательство «Символ-Плюс». 199034, Санкт-Петербург, 16 линия, 7,
тел. (812) 324-5353, edit@symbol.ru. Лицензия ЛП N 000054 от 25.12.98.
Налоговая льгота – общероссийский классификатор продукции
ОК 005-93, том 2; 953000 – книги и брошюры.
Подписано в печать 22.07.2005. Формат 70x100 1/16 . Печать офсетная.
Объем 15 печ. л. Тираж 2000 экз. Заказ N
Отпечатано с готовых диапозитивов в ГУП «Типография «Наука»
199034, Санкт-Петербург, 9 линия, 12.

Оглавление

Отзывы на книгу «Хакинг»

Очень рекомендую эту книгу. Ее написал человек, который знает, о чем говорит, а программный код, инструменты и примеры вполне работоспособны.

– IEEE CIPHER

Из всех прочитанных мною книг эту считаю для хакера главной.

– SECURITY FORUMS.COM

Непрограммисты узнают из этой книги, насколько легко взломать систему, в которой исполняется уязвимый код. А программисты поймут, насколько проще может быть устранение уязвимостей в ПО с открытым исходным кодом.

– COMPUTER POWER USER MAGAZINE

...очень глубокая, информативная книга. Читая ее, я получил огромное удовольствие и посоветовал бы ее всякому, кто по-настоящему интересуется компьютерной безопасностью.

– GEEKSHELTER.COM

Тем, кто имеет дело с компьютером лишь время от времени, эта книга может показаться немного перенасыщенной технической информацией, но это захватывающее чтение как для тех, кто хочет узнать больше, так и для тех, кто желает отточить свое профессиональное мастерство.

– THE TRIBUNE REVIEW

Эта книга должна входить в программу обязательного чтения для любого программиста, стремящегося к совершенству, и по ней надо преподавать фундаментальные основы программирования во всех учебных заведениях, где готовят программистов.

– FLASH-MX.COM

...нужна поистине фантастическая книга, чтобы отучить кого-нибудь от видеоигр. Штабель из коробок с ними все рос, а я не мог оторваться от чтения. Одно только это говорит о качестве книги.

– PGNX.NET

Из книг о программировании я бы отметил только эту.

– UNIXREVIEW

Эриксон так представляет материал, что не запутаешься и читать приятно.

– IEEE SECURITY & PRIVACY

Отличная книга.

– ABOUT.COM

Благодарности

Я благодарю Билла Поллока (Bill Pollock), Карол Хурадо (Karol Jurado), Энди Кэррола (Andy Carroll), Лэя Сэкса (Leigh Sacks) и всех остальных сотрудников издательства No Starch Press, сделавших возможным выход этой книги и позволивших мне творчески участвовать в процессе. Кроме того, я благодарю своих друзей Сета Бенсона (Seth Benson) и Аарона Эдамса (Aaron Adams) за чтение корректуры и редактирование, Джека Мэтсона (Jack Matheson) за помощь с ассемблером, д-ра Зейделя (Dr. Seidel) за поддержание интереса к вычислительной технике, моих родителей за покупку того самого первого Commodore Vic-20 и хакерское сообщество за стремление к новому и творческий дух, породившие технологии, о которых рассказывается в книге.

Предисловие

В книге рассказывается о деталях различных хакерских технологий, многие из которых являются весьма специальными. Здесь также приводятся начальные сведения о базовых понятиях программирования, на которых основываются эти технологии, но для лучшего понимания последних желательно общее знакомство читателя с программированием. Приводимые в книге примеры программ написаны для компьютера x86, работающего под Linux. Во время чтения полезно иметь под рукой подобный компьютер, тогда можно будет самостоятельно получить такие же результаты и провести собственные исследования. В этом и заключается хакерство.

Во время написания использовался дистрибутив Gentoo Linux, доступный по адресу *http://www.gentoo.org*.

0x100

Введение

Слова «хакинг» или «хакерство» могут вызывать в воображении картины электронного вандализма, шпионажа, крашеных волос и тел, разукрашенных пирсингом. У большинства людей хакинг ассоциируется с нарушением закона, а потому все, кто занимается хакерской деятельностью, становятся в их глазах преступниками. Несомненно, есть люди, применяющие хакерские технологии в нарушение законов, но суть хакерства состоит не в этом. На самом деле хакинг больше тяготеет к соблюдению законов, чем к их нарушению.

Сущность хакинга состоит в поиске непреднамеренных или упущенных из виду применений законов и свойств данной ситуации и последующем применении их новыми и изобретательными способами для решения некоторой задачи. Такая задача может состоять в получении доступа к компьютерной системе или отыскании способа использования устаревшего телефонного оборудования для управления моделью железной дороги. Обычно хакеры предлагают для решения таких задач уникальные способы, невообразимые для тех, кто придерживается стандартной методологии.

В конце 1950-х годов клуб железнодорожного моделирования Массачусетского технологического института получил в подарок ряд деталей в основном от старой телефонной аппаратуры. С помощью этого оборудования члены клуба смастерили сложную систему, позволявшую нескольким операторам управлять различными частями дороги путем набора номера соответствующего участка. Такое необычное и оригинальное применение оборудования они назвали «хакингом», и многие считают эту группу первыми хакерами. Затем они занялись написанием программ на перфокартах и перфолентах для первых компьютеров, таких как IBM 704 и TX-0. В то время как все другие удовлетворялись

тем, что писали программы, которые решали задачи, первые хакеры были озабочены тем, чтобы писать программы, которые решают задачи хорошо. Программа, способная получить тот же результат с помощью меньшего количества перфокарт, считалась лучшей, хотя она и делала то же самое. Главным отличием было то, как программа получает результаты, – ее *элегантность*.

Умение сократить количество перфокарт, необходимых для программы, свидетельствовало об артистизме в управлении компьютером, вызывавшем восхищение у тех, кто способен был его оценить. Аналогично деревянная колода справляется с ролью подставки для вазы, но стол, красиво вырезанный из нее с применением изящной техники, выглядит гораздо лучше. Первые хакеры превращали программирование из технической задачи в разновидность искусства, которая, как и многие другие виды искусства, могла быть оценена только посвященными и оставалась непонятой остальными.

Такой подход к программированию создал информационную субкультуру, разделившую тех, кто ценил красоту хакинга, и тех, кто ее не замечал. Эта субкультура была в основном нацелена на более глубокое изучение данного искусства и овладение вершинами мастерства в нем. Ее участники верили, что информация должна свободно распространяться, а все препятствия на пути этой свободы должны так или иначе преодолеваться. В число таких препятствий входили авторитетные личности, администрация учебных заведений и дискриминация. В отличие от большинства студентов, движимых желанием получить документы об образовании, эта неофициальная группа хакеров отвергала традиционное стремление к хорошим оценкам, преследуя задачу получения собственно знаний. Эта тяга к непрерывной учебе и исследованиям выходила даже за пределы традиционных границ, очерченных дискриминацией, о чем свидетельствует признание этой группой 12-летнего Питера Дойча, который продемонстрировал свое знание TX-0 и желание учиться. Возраст, раса, пол, внешность, ученые степени и положение в обществе не были основными критериями ценности участников – не из стремления к равенству, а из стремления развивать нарождающееся искусство хакинга.

Хакеры обнаружили красоту и элегантность в таких обычно сухих науках, как математика и электроника. Они рассматривали программирование как вид художественного самовыражения, и компьютер был инструментом их искусства. Их желание все препарировать и понять принцип работы не было направлено на разоблачение художественного творчества, а служило средством добиться более высокой оценки со стороны. Эти ценности, обусловленные стремлением к знаниям, в итоге назовут «хакерской этикой»: понимание логики как формы искусства, способствование свободному обмену информацией, преодоление традиционных пределов и ограничений с единственной целью – лучше понять окружающий мир. Такая позиция не нова: схо-

жую этику и субкультуру создали пифагорейцы в Древней Греции, хотя компьютеров у них и не было. Они увидели красоту математики и открыли многие базовые понятия геометрии. Эта жажда знаний и ее благотворные побочные продукты встречаются на всем протяжении истории от пифагорейцев до Ады Лавлейс, Алана Тьюринга и хакеров из клуба железнодорожного моделирования MIT. Развитие вычислительных наук продолжилось и дальше, приведя к таким фигурам, как Ричард Столмен и Стив Возняк. Эти хакеры дали нам современные операционные системы, языки программирования, персональные компьютеры и многие другие технологические достижения, используемые теперь в повседневной жизни.

Так как же различить хороших хакеров, творящих чудеса технологического прогресса, и плохих хакеров, крадущих номера наших кредитных карточек? В какой-то момент для обозначения плохих хакеров стали использовать термин «крэкер», который должен был отличать их от хороших хакеров. Журналистам объяснили, что крэкеры – нехорошие ребята, а хакеры – хорошие. Хакеры придерживались хакерской этики, а крэкеры думали только о том, как преступить закон. Считалось, что крэкеры значительно менее талантливы, чем элита хакеров, и просто пользуются написанными хакерами инструментами и скриптами, не понимая, как они работают. Слово «крэкер» должно было стать общим ярлыком для всех, кто делает с помощью компьютера что-либо бессовестное: ворует программное обеспечение, уродует веб-сайты и, что хуже всего, не понимает, как он это делает. Но сегодня этот термин употребляют очень немногие.

Термин не прижился, может быть, из-за конфликта определений: первоначально словом «крэкер» называли тех, кто нарушал авторские права и вскрывал системы защиты от копирования. А может быть, причина кроется в новом определении, относящемся как к тем, кто занимается незаконной деятельностью с помощью компьютеров, так и к тем, кто является относительно малоквалифицированными хакерами. Очень немногие журналисты считают необходимым описывать тех, кто не обладает высокой квалификацией, с помощью термина «крэкеры», незнакомого широкой публике. Напротив, большинство людей слышало о таинственности и мастерстве, ассоциируемых со словом «хакер». Журналисты, не колеблясь, делают выбор между терминами «крэкеры» и «хакеры». Аналогично для обозначения крэкеров иногда употребляют термин «скрипт-кидди», но в нем нет такого оттенка журналистской сенсационности, как в темном хакере. Некоторые все же станут утверждать, что между хакерами и крэкерами есть четкая разграничительная линия, но я считаю, что хакер – это всякий, в ком живет хакерский дух, независимо от того, какие законы он, возможно, нарушает.

Это нечеткое различие между хакерами и крэкерами становится еще более расплывчатым благодаря современным законам, накладывающим ограничения на применение криптографических средств и иссле-

дования в этой области. В 2001 г. профессор Эдвард Фелтен (Edward Felten) и группа ученых из Принстонского университета собирались опубликовать результаты своих исследований в статье, обсуждавшей слабости различных систем цифровых водяных знаков. Эта статья была ответом на вызов, брошенный группой основателей стандарта SDMI (Secure Digital Music Initiative), в котором всем желающим предлагалось попытаться взломать эти системы водяных знаков. Однако против публикации этой статьи и с угрозами в адрес исследователей выступили Фонд SDMI и Американская ассоциация звукозаписывающей индустрии (RIAA). Несомненно, что Акт об авторских правах цифрового тысячелетия (Digital Millennium Copyright Act, DMCA) от 1998 года делает незаконным обсуждение или предоставление технологии, с помощью которой можно обходить устанавливаемый индустрией контроль над покупателями. Это тот закон, который был применен против Дмитрия Склярова, русского программиста и хакера. Он написал программу, позволяющую обходить довольно простое шифрование в программах фирмы Adobe, и представил свои результаты на конференции хакеров в Соединенных Штатах. ФБР налетело, как сокол, арестовав его, за чем последовали долгие юридические баталии. По закону сложность системы контроля над покупателями не имеет значения: формально незаконным является вскрытие или даже обсуждение «поросячьей латыни», если она используется как промышленное средство контроля над покупателями. Так кого теперь считать хакерами, а кого – крэкерами? Если закон становится препятствием для свободы слова, то «хорошие ребята», которые говорят то, что думают, внезапно становятся плохими? Я полагаю, что дух хакерства выше правительственных законов и не должен определяться ими. Но в любой группе специалистов всегда найдутся те, кто использует свои знания для совершения нехороших поступков.

Такие науки, как ядерная физика и биохимия, могут использоваться для того, чтобы убивать, но при этом имеют большое значение для научного прогресса и современной медицины. Само знание не является ни плохим, ни хорошим; стандарты нравственного поведения относятся к применению этого знания. При всем желании мы не смогли бы запретить знание о том, как превратить материю в энергию или остановить непрерывный технологический прогресс в обществе. Точно так же невозможно отменить сущность хакерства, как и подвергнуть его упрощенной классификации или анализу. Хакеры всегда будут раздвигать существующие границы, вынуждая нас к дальнейшим исследованиям.

К сожалению, существует немало так называемых «книжек для хакеров», представляющих собой всего лишь краткие руководства по хакам, осуществленным другими людьми. Они предлагают читателю воспользоваться инструментами на прилагаемом CD, не объясняя теории, лежащей в основе этих инструментов, порождая людей, умеющих пользоваться чужими инструментами, но не способных понять,

как они устроены, или создать собственные. Возможно, что термины «крэкер» и «скрипт-кидди» еще не отжили свое.

Настоящие хакеры – это первопроходцы, разрабатывающие методы и создающие инструменты, которыми заполняются упомянутые диски. Если оставить в стороне юридические вопросы и порассуждать логически, то для каждого эксплойта, о котором можно прочесть в книге, существует «патч» (заплата), позволяющий защититься от него. Должным образом пропатченная система должна быть защищена от атак данного класса. Добычей агрессора, взявшего на вооружение эти методы, не внеся в них ничего нового, неизбежно окажется лишь слабость и глупость. Настоящие хакеры – творцы, умеющие обнаруживать дыры и слабости в программном обеспечении и создавать собственные эксплойты. Если хакеры решают не сообщать создателю программного обеспечения о найденных ими уязвимостях, то могут с помощью этих эксплойтов беспрепятственно проходить через полностью пропатченные и «защищенные» системы.

А если нет никаких патчей, то как можно помешать хакерам обнаружить в программах новые дыры и воспользоваться ими в своих интересах? Для этого и существуют группы исследования компьютерной безопасности, которые пытаются обнаружить эти дыры и уведомить о них поставщиков программного обеспечения, прежде чем будут созданы эксплойты. Происходит полезная совместная эволюция хакеров, обеспечивающих защиту систем, и хакеров, пытающихся осуществить взлом. Благодаря соревнованию между ними мы получаем более прочную защиту и более утонченные методы атаки. Появление и развитие систем обнаружения несанкционированного доступа (Intrusion Detection Systems, IDS) служит главным примером этого совместного эволюционного процесса. Хакеры, строящие защиту, пополняют свой арсенал системами IDS, а атакующие хакеры разрабатывают методы противодействия IDS, которые учитываются при создании новых и более мощных IDS. Итоговый результат такого взаимодействия оказывается положительным, потому что приводит к появлению более квалифицированных специалистов, более совершенной защиты, более стабильных программ, изобретательных технологий решения проблем и даже новой экономики.

Эта книга написана с целью показать подлинную суть хакинга. Мы рассмотрим различные хакерские технологии, как старые, так и современные, и проанализируем их, чтобы понять, как и почему они работают. Благодаря выбранному способу изложения информации вы получите такое понимание и представление о хакинге, которое позволит вам усовершенствовать существующие или даже изобрести оригинальные новые методы. Я надеюсь, что книга послужит стимулом для развития в читателе любознательного хакерского начала и побуждением внести свой вклад в искусство хакинга, по какую бы сторону баррикад он ни находился.

0x200

Программирование

Термин «хакинг» употребляют как те, кто пишет код, так и те, кто злоупотребляет им (создает или использует «эксплойты»). Несмотря на разные конечные цели, обе эти группы хакеров применяют для решения задач схожие методы. И поскольку знание программирования помогает тем, кто злоупотребляет кодом, а знание способов злоупотребления помогает тем, кто программирует, многие хакеры занимаются тем и другим одновременно. Интересные хаки можно найти как в приемах написания элегантного кода, так и в методах злонамеренного использования программ. Фактически хакинг – это открытие искусного и неочевидного решения какой-либо проблемы.

Хаки, применяемые в программных эксплойтах, как правило, основаны на использовании законов функционирования компьютера непредусмотренными способами, которые приводят к достижению чудодейственных, на первый взгляд, результатов, часто имеющих целью обход системы защиты. Аналогично и в обычных программах хаки используют законы функционирования компьютера новыми и творческими способами, но их конечная цель часто состоит в том, чтобы решить поставленную задачу наиболее эффектным и лучшим из возможных способов. Можно написать бесчисленное множество программ, которые решают конкретную задачу, но большинство из них окажутся излишне пространными, сложными или неряшливыми, и лишь незначительная часть решений будет невелика по размеру, эффективна и аккуратна. Программы, обладающие такими качествами, считаются элегантными, а искусные и изобретательные решения, которые приводят к такой эффективности, называются хаками. Хакеры с обеих противостоящих сторон склонны высоко оценивать как красоту элегантного кода, так и остроумие удачных хаков.

Из-за резкого роста производительности компьютеров и временного экономического бума «дот-комов», связанного с развитием Интернета, созданию искусных хаков и элегантного кода придавалось не так много значения, как задаче возможно более быстрого и дешевого создания работоспособного кода. С точки зрения бизнеса нет смысла тратить лишние пять часов, чтобы создать чуть более быстрый и более эффективно расходующий память фрагмент кода, если увеличение скорости и экономия памяти оборачиваются всего лишь миллисекундами для имеющихся у клиентов современных процессоров и долями процента памяти из сотен миллионов байт, доступных современным компьютерам. Если все определяется в конечном счете деньгами, то тратить время на искусные хаки для оптимизации просто не имеет смысла.

По-настоящему оценить элегантность программы могут только хакеры – компьютерные энтузиасты, конечной целью которых является не прибыль, а желание выжать из своего старенького Commodore 64 все, на что он способен; создатели эксплойтов, которые пишут изумительные крохотные фрагменты кода, способные проскользнуть через узкие щелки в системе защиты; каждый, кого увлекает задача найти лучшее из возможных решений. Вот те люди, которые действительно увлечены программированием и действительно ценят красоту элегантного кода или остроумие изобретательного хака. Для того чтобы понимать, как можно злонамеренно использовать программы, необходимо разбираться в программировании, поэтому последнее послужит для нас естественной отправной точкой.

0x210 Что такое программирование?

Программирование – вполне естественное и интуитивное понятие. *Программа* представляет собой лишь ряд предложений, написанных на особом языке. С программами мы встречаемся повсеместно, и даже технофобы ежедневно имеют с ними дело. Указания о том, как проехать в нужное место на машине, кулинарные рецепты, футбольные матчи и ДНК – все это программы, существующие в жизни или даже определяющие клеточный состав любого человека. Типичная «программа» проезда в нужное место может выглядеть так:

```
Начать движение по Главной улице в восточном направлении.
Ехать по Главной улице, пока справа не покажется церковь. Если движение
закрыто из-за строительства, повернуть там на 15-ю улицу, повернуть налево
на Сосновую улицу, затем повернуть направо на 16-ю улицу. В противном случае
продолжить движение до 16-й улицы и повернуть направо. Продолжать движение
по 16-й улице, затем свернуть на Дорогу К Цели. Проехать по Дороге К Цели
5 миль до дома с правой стороны. Адрес – Дорога К Цели, дом 743.
```

Каждый, кто понимает человеческий язык, может понять и выполнить эти указания. Конечно, они не очень красноречивы, но каждая инструкция проста, по крайней мере, для того, кто умеет читать.

Но компьютер не понимает обычную человеческую речь, он понимает только *машинный язык*. Чтобы заставить компьютер что-то сделать, надо написать ему инструкции на его языке. Однако машинный язык выглядит непонятным, и с ним трудно работать. Машинный язык состоит только из битов и байтов и специфичен для каждой машинной архитектуры. Поэтому чтобы написать программу на машинном языке для процессора Intel x86, необходимо выяснить численное значение, соответствующее каждой команде, особенности ее выполнения и немыслимое количество прочих деталей, относящихся к низкому уровню программирования. В таком виде программирование трудоемко и обременительно, и уж явно не интуитивно.

Преодолеть сложность написания программ на машинном языке позволяет транслятор. Одним из видов трансляторов в машинный язык является *ассемблер* – программа, которая транслирует текст на языке ассемблера в машинно-читаемый код. Язык ассемблера не так загадочен, как машинный язык, поскольку различные команды и переменные в нем записываются при помощи имен, а не чисел. Однако и язык ассемблера является далеко не наглядным. Названия команд понятны лишь посвященным, а язык остается специфичным для данной архитектуры. Это означает, что так же, как машинный язык для процессоров Intel x86 отличен от машинного языка для процессоров Sparc, так и язык ассемблера для x86 отличен от языка ассемблера для Sparc. Любая программа, написанная на языке ассемблера для процессора одной архитектуры, не сможет работать на процессоре другой архитектуры. Если программа написана на языке ассемблера x86, ее придется переписывать для выполнения в архитектуре Sparc. Кроме того, чтобы писать эффективные программы на языке ассемблера, необходимо знать многие подробности архитектуры этого процессора.

Эти проблемы становятся менее существенными, если применяется другой вид транслятора, который называется *компилятором*. Компилятор преобразует язык высокого уровня в машинный язык. Языки высокого уровня гораздо более наглядны, чем язык ассемблера, и могут преобразовываться в машинные языки различных типов для каждой из различных архитектур процессоров. Это означает, что на языке высокого уровня программу можно написать один раз и один и тот же программный код будет преобразован компилятором в машинный язык различных конкретных архитектур. C, C++ и FORTRAN являются примерами языков высокого уровня. Написанная на языке высокого уровня программа значительно легче читается и больше похожа на естественный язык, чем язык ассемблера или машинный язык, но тем не менее должна придерживаться очень жестких правил написания команд, иначе компилятор не сможет их понять.

У программистов есть еще один вид языка программирования, называемый *псевдокодом*. Псевдокод – это естественный язык, структура которого похожа на структуру языка высокого уровня. Его не понима-

ют компиляторы, ассемблеры и какие-либо компьютеры, но он полезен программистам для организации команд. Псевдокод не имеет четкого определения. Разные люди пишут на псевдокоде немного по-разному. В некотором роде это туманное отсутствующее звено между естественными языками, например английским, и языками программирования высокого уровня, например C. Если приведенные выше указания по движению преобразовать в псевдокод, то может получиться что-нибудь вроде следующего:

```
Начать движение на восток по Главной улице;
Пока (справа не появится церковь)
{
  Ехать по Главной;
}
Если (улица закрыта)
{
  Повернуть(направо, 15-я улица);
  Повернуть(налево, Сосновая улица);
  Повернуть(направо, 16-я улица);
}
иначе
{
  Повернуть (направо, 16-я улица);
}
Повернуть (налево, Дорога К Цели);
Для (5 итераций)
{
  Ехать прямо 1 милю;
}
Остановиться у дома 743 по Дороге К Цели;
```

Каждая команда помещается на своей отдельной строке, а управляющая логика движения разбита на управляющие структуры. Без управляющих структур программа просто была бы рядом последовательно выполняемых команд. Но наши инструкции движения не так просты. В них есть такие команды, как «ехать по Главной улице, пока справа не покажется церковь» и «Если улица закрыта из-за строительства...». Их называют *управляющими структурами*, и они изменяют порядок выполнения программы с простого последовательного на более сложный и полезный.

Кроме того, инструкции для поворота машины в действительности гораздо сложнее, чем просто «повернуть направо на 16-й улице». Поворот машины может подразумевать нахождение улицы, на которую нужно свернуть, замедление движения, включение сигнала поворота, поворот рулевого колеса и, наконец, набор скорости, соответствующей движению транспорта по новой улице. Поскольку многие из этих действий одинаковы при повороте на любую улицу, их можно описать в виде *функции*. Функция принимает в качестве входных данных ряд аргументов, выполняет на основе входных данных свой набор команд

и возвращается в то место, откуда была вызвана. На псевдокоде функция поворота может быть записана так:

```
Функция Повернуть(направление, улица)
{
  найти улицу;
  замедлить движение;
  если(направление == направо)
  {
    включить сигнал поворота направо;
    повернуть руль вправо;
  }
  иначе
  {
    включить сигнал поворота налево;
    повернуть руль влево;
  }
  набрать скорость
}
```

Многократно применяя эту функцию, можно повернуть машину на любую улицу, в любом направлении и не писать каждый раз отдельные команды. Вызывая функции, важно помнить, что при этом выполнение программы действительно переходит скачком в новое место, а затем снова возвращается туда, где оно было прервано.

Последнее важное замечание относительно функций: у каждой из них есть собственный контекст. Это значит, что имеющиеся в функции локальные переменные уникальны внутри каждой функции. У каждой функции есть собственный контекст, или окружение, в котором она выполняется. Основу всей программы также образует функция со своим собственным контекстом, и при вызове из этой главной функции любой другой функции внутри главной функции для нее создается новый контекст. Если вызванная функция вызывает еще одну функцию, то для последней создается новый контекст внутри контекста предыдущей функции и т. д. Такое наслаивание функциональных контекстов позволяет каждой функции быть в некотором отношении атомарной.

Управляющие структуры и понятия функций, присутствующие в приведенном псевдокоде, можно найти во многих языках программирования. У псевдокода нет определенной формы, но приведенный пример напоминает язык программирования C. Это удобное сходство, поскольку C – очень распространенный язык программирования. В действительности большая часть Linux и других современных реализаций операционных систем Unix написана на C. Поскольку Linux является операционной системой с открытым кодом и свободным доступом к компиляторам, ассемблерам и отладчикам, она представляет собой отличную платформу для обучения. В данной книге предполагается, что все действия выполняются на машине с процессором x86 под управлением Linux.

0x220 Программные эксплойты

Программные эксплойты составляют основу хакинга. Программа – это просто сложный набор правил, определяющих некоторый порядок исполнения и в конечном счете указывающих компьютеру, что он должен сделать. Программный эксплойт – это искусный способ заставить компьютер выполнить то, что вам нужно, даже если выполняемая в данный момент программа была разработана с намерением не допустить таких действий. Поскольку в действительности программа может работать только так, как она написана, пробелы в защите фактически представляют собой ошибки или недосмотры, допущенные при разработке программы или среды, в которой она выполняется. Для обнаружения таких брешей и написания программ, в которых они закрыты, требуется творческий склад ума. Иногда эти бреши – результат относительно очевидных ошибок программиста, но встречаются и менее явные ошибки, на которых основаны более сложные технологии эксплойтов, применимые в большинстве ситуаций.

Программа может лишь в точности делать то, на что она запрограммирована. К сожалению, текст программы не всегда соответствует тому, что программа должна была бы делать по замыслу программиста. Проиллюстрируем это положение следующим шутливым рассказом.

> Некто, идя по лесу, обнаруживает на земле волшебную лампу. Он инстинктивно поднимает лампу с земли, вытирает ее рукавом, и из нее появляется джинн. Джинн благодарит человека за свое освобождение и предлагает исполнить три его желания. Человек в восторге, точно зная, чего он хочет.
>
> «Во-первых, – говорит он, – хочу миллион долларов».
>
> Джинн щелкает пальцами, и из воздуха возникает чемодан, полный денег.
>
> Раскрыв в изумлении глаза, человек продолжает: «Теперь хочу „феррари“».
>
> Джинн щелкает пальцами, и из ничего появляется «феррари».
>
> Человек продолжает: «Наконец, я хочу стать неотразимым для женщин».
>
> Джинн щелкает пальцами, и человек превращается в коробку шоколадных конфет.

Точно так же, как последнее желание человека было выполнено в соответствии с тем, что он сказал, а не с тем, что он подумал, так и программа выполняет свои инструкции буквально, а результаты могут оказаться не теми, которые предполагал программист. Иногда просто катастрофическими.

Программисты – тоже люди, и порой пишут не совсем то, что имеют в виду. Например, распространенной ошибкой программирования яв-

ляется «ошибка плюс-минус один». Как следует из названия, она возникает, когда программист при подсчете ошибается на единицу. Это происходит гораздо чаще, чем можно было бы предположить, и проще всего проиллюстрировать суть таким вопросом: если вы строите изгородь длиной 30 метров и ставите столбы через каждые три метра, то сколько столбов вам понадобится? Очевидный ответ – 10 столбов, но он не верен, потому что в действительности потребуется 11 столбов. Подобные ошибки происходят, когда программист по ошибке считает предметы вместо промежутков между предметами или наоборот. Другой пример: выбор программистом интервала чисел или элементов, которые надо обработать, например элементы с номера N по номер M. Если $N = 5$ и $M = 17$, то сколько элементов надо обработать? Очевидный ответ $M - N$, или $17 - 5 = 12$ элементов. Но это неправильно, потому что на самом деле элементов $M - N + 1$, то есть всего 13. На первый взгляд это кажется нелогичным, но именно это и приводит к такого рода ошибкам.

Часто такие ошибки остаются незамеченными, потому что программы не подвергаются тестированию для всех возможных случаев, а последствия ошибки могут не сказаться при обычном выполнении программы. Однако если программе передать такие входные данные, которые заставят ошибку проявиться, они могут оказать разрушительное действие на всю остальную логику программы. Правильно построенный на ошибке «плюс-минус один» эксплойт превращает защищенную, казалось бы, программу в уязвимую.

В качестве довольно свежего примера можно привести OpenSSH – комплекс программ защищенной связи с терминалом, который должен был заменить небезопасные и не использующие криптографии службы, такие как telnet, rsh и rcp. Однако в коде, выделяющем каналы, была допущена ошибка обсчета на единицу, которая интенсивно эксплуатировалась. А именно в операторе `if` был такой код:

```
if (id < 0 || id > channels_alloc) {
```

Тогда как правильный код должен выглядеть следующим образом:

```
if (id < 0 || id >= channels_alloc) {
```

На обычном языке этот код означает: «Если ID меньше 0 или ID больше количества выделенных каналов, выполнить следующее...», тогда как правильным было бы «Если ID меньше 0 или ID больше или *равно* количеству выделенных каналов, выполнить следующее...».

Эта простая ошибка на единицу позволила создать эксплойт, с помощью которого обычный зарегистрировавшийся в системе пользователь получал в ней неограниченные права администратора. Разумеется, подобная функциональность не входила в намерения разработчиков такой защищенной программы, как OpenSSH, но компьютер может выполнять только те инструкции, которые ему даются, даже если это не такие инструкции, которые предполагались изначально.

Другая ситуация, в которой часто возникают ошибки, становящиеся основой последующих эксплойтов, связана с поспешной модификацией программы в целях расширения ее функциональности. Такое расширение функциональности увеличивает рыночные возможности программы и ее ценность, но при этом растет и сложность программы, а значит, и вероятность оплошностей в ней. Программа веб-сервера Microsoft IIS должна предоставлять пользователям статическое и интерактивное содержимое. Для этого она должна разрешать пользователям читать, записывать и выполнять программы и файлы внутри некоторых каталогов. Такие возможности, однако, должны предоставляться лишь в некоторых выделенных каталогах. В противном случае пользователи получат полный контроль над системой, что, очевидно, недопустимо с точки зрения безопасности. С этой целью в программу был включен код проверки маршрутов, который запрещал пользователям с помощью символа обратной косой черты перемещаться вверх по дереву каталогов и входить в другие каталоги.

Однако с добавлением в программу поддержки кодировки символов Unicode ее сложность увеличилась. Unicode представляет собой набор символов, записываемых двумя байтами, и содержит символы любых языков, включая китайский и арабский. Используя для каждого символа два байта вместо одного, Unicode позволяет записывать десятки тысяч различных символов, а не всего несколько сотен, как при однобайтовых символах. Дополнительная сложность привела к тому, что символ обратной косой черты стал представляться несколькими способами. Например, %5c в кодировке Unicode транслируется в символ обратной косой черты, но эта трансляция происходит уже *после* выполнения кода, проверяющего допустимость маршрута. Поэтому ввод символов %5c вместо \ действительно сделал возможным перемещение по дереву каталогов, что открывало лазейку для злоупотреблений, о которой говорилось выше. Два червя – Sadmind и Code-Red – использовали просмотр в преобразовании кодировки Unicode такого типа для искажения (дефейса) веб-страниц.

Еще один близкий пример данного принципа буквального исполнения, уже не относящийся к программированию, известен как «лазейка Ламаччиа». Подобно правилам компьютерных программ, в законодательстве США иногда обнаруживаются правила, говорящие не то, что под ними подразумевалось. И, подобно компьютерным эксплойтам, эти юридические лазейки можно использовать, чтобы обойти смысл закона. В конце 1993 года 21-летний компьютерный хакер и студент МТИ по имени Дэвид Ламаччиа (David LaMacchia) организовал электронную доску объявлений под названием Cynosure (Полярная звезда) с целью обмена пиратскими программами. Те, у кого были какие-нибудь программы, могли загружать их на сервер, а те, у кого не было, могли брать их с сервера. К концу шестой недели существования эта служба породила такой усиленный сетевой трафик по всему свету, что привлекла внимание университетских и федеральных властей. Компа-

нии-производители программного обеспечения заявили, что Cynosure нанесла им убытки в размере миллиона долларов, и федеральное большое жюри предъявило Ламаччии обвинение в заговоре с неизвестными лицами и нарушении закона о мошенничестве с применением электронных средств. Однако обвинение было снято, поскольку действия Ламаччии не являлись преступлением согласно закону об авторских правах, так как они не имели целью получение коммерческих преимуществ или личной финансовой выгоды. Очевидно, законодателям в свое время не пришло в голову, что кто-нибудь станет заниматься такой деятельностью, имея другие цели, не связанные с личным обогащением. Позднее, в 1997 году, Конгресс США закроет эту лазейку законом об электронном воровстве. Хотя в этом примере не участвует эксплойт компьютерной программы, но судьи и суды могут рассматриваться здесь как компьютеры, выполняющие программу юридической системы буквально, как она написана. Абстрактные понятия хакинга выходят за рамки компьютеров и могут применяться к различным жизненным ситуациям, в которых участвуют сложные системы.

0x230 Общая технология эксплойта

Ошибки «плюс-минус один» или при трансляции Unicode трудно заметить в момент совершения, но задним числом они хорошо видны любому программисту. Однако есть несколько распространенных ошибок, на которых строятся далеко не столь очевидные эксплойты. Влияние этих ошибок на безопасность не всегда бросается в глаза, и соответствующие проблемы с защитой обнаруживаются в коде повсеместно. Поскольку одни и те же ошибки совершаются в разных местах, на их основе были разработаны обобщенные методы эксплойтов, которыми можно воспользоваться в различных ситуациях.

Два самых распространенных вида обобщенных методов эксплойтов – это эксплойт переполнения буфера и эксплойт форматной строки. В обоих случаях конечной целью является получение контроля над выполнением атакуемой программы с тем, чтобы заставить ее выполнить вредоносный фрагмент кода, который теми или иными средствами удалось поместить в память. Это называется «выполнением произвольного кода», поскольку хакер может заставить программу делать практически что угодно.

Что делает эти виды эксплойтов интересными, так это разработанные для них различные искусные хаки, с помощью которых достигаются впечатляющие конечные результаты. В понимании этих методов заключается гораздо большая сила, чем в конечном результате каждого отдельного эксплойта, потому что их можно применять и развивать для создания массы других эффектов. Однако для понимания этих методов эксплойтов необходимо предварительно глубоко изучить права доступа к файлам, переменные, выделение памяти, функции и язык ассемблера.