СОДЕРЖАНИЕ

ОБ АВТОРАХ

Алекс Матросов – ведущий специалист по наступательной безопасности в компании NVIDIA. Больше двадцати лет занимается обратной разработкой, продвинутым анализом вредоносных программ, безопасностью на уровне прошивок и методами эксплуатации уязвимостей. До перехода в NVIDIA работал главным исследователем по безопасности в Центре передовых технологий безопасности Intel (SeCoE), больше шести лет работал в группе исследования новых угроз в Intel и занимал должность старшего исследователя по безопасности в компании ESET. Алекс является автором и соавтором многочисленных статей и часто выступает на конференциях по безопасности, в т. ч. REcon, ZeroNights, Black Hat, DEFCON и других. Удостоился награды от компании Hex-Rays за плагин с открытым исходным текстом HexRaysCodeXplorer, который начиная с 2013 года поддерживается командой REhint.

Евгений Родионов, PhD, специалист по безопасности в Intel, занимается безопасностью BIOS для клиентских платформ. До этого участвовал во внутренних исследовательских проектах и отвечал за углубленный анализ комплексных угроз в ESET. В сферу его интересов входят безопасность на уровне прошивки, программирование в режиме ядра, технологии противодействия руткитам и обратная разработка. Много раз выступал на конференциях по безопасности, включая Black Hat, REcon, ZeroNights и CARO, является соавтором многочисленных научных статей.

Сергей Братусь – научный сотрудник и доцент факультета информатики Дартмутского колледжа. Ранее работал в компании BBN Technologies, где занимался обработкой естественных языков. Братуся интересуют все аспекты безопасности Unix, в особенности безопасность ядра Linux, а также обнаружение и обратная разработка вредоносного ПО в Linux.

О ТЕХНИЧЕСКОМ РЕЦЕНЗЕНТЕ

Родриго Рубира Бранко (BSDaemon) работает главным исследователем по безопасности в корпорации Intel Corporation, где возглавляет группу STORM (Strategic Offensive Research and Mitigations). Родриго обнаружил десятки уязвимостей во многих важных технологиях и опубликовал новаторские работы по эксплуатации, обратной разработке и анализу вредоносных программ. Входит в группу RISE Security Group и является одним из организаторов Hackers to Hackers Conference (H2HC), старейшей конференции по безопасности в Латинской Америке.

ВСТУПИТЕЛЬНОЕ СЛОВО

Невозможно отрицать тот факт, что вредоносные программы представляют растущую угрозу компьютерной безопасности. Всюду мы видим тревожную статистику, свидетельствующую о росте финансовых потерь, сложности и разнообразии вредоносного ПО. Все больше исследователей, в промышленности и в академических кругах, изучают вредоносное ПО и публикуют свои результаты, пользуясь различными каналами – от блогов и конференций до университетских курсов и книг, посвященных этому предмету. В этих публикациях тема рассматривается под всевозможными углами зрения: обратная разработка, передовые практики, методология и лучшие комплекты инструментов.

Таким образом, дискуссии по поводу инструментов для анализа вредосного ПО и его автоматизации уже идут и с каждым днем все ширятся. А раз так, возникает вопрос: зачему нужна еще одна книга на эту тему? Что в ней может быть такого, чего нет в других?

Прежде всего, хотя эта книга посвящена обратной разработке передового – я имею в виду *инновационного* – вредоносного ПО, она включает и фундаментальные знания о том, для чего тот или иной фрагмент вредоносной программы вообще написан. В книге объясняются механизмы работы различных обсуждаемых компонентов – от начальной загрузки платформы и загрузки операционной системы до различных частей ядра и программ прикладного уровня, которые все равно рано или поздно обращаются к ядру.

Я сам не раз объяснял, что *фундаментальное* освещение материала – не то же самое, что *базовое*, хотя в обоих случаях речь идет об основополагающих строительных блоках, на которых покоятся компьютеры и вычисления. И под таким углом зрения эта книга – больше, чем обсуждение вредоносного ПО. В ней описывается, как работают компьютеры, как в современных программных стеках используются базовые возможности машины и пользовательские интерфейсы. Зная все это, вы вдруг начинаете *автомагически* понимать, как и почему вещи ломаются и как их можно употребить во вред.

Кто лучше проведет по этому пути, чем авторы, послужной список которых включает распутывание – и неоднократное – по-настоящему изобретательного вредоносного кода, раздвигавшего границы возможного? Добавьте к этому хорошо продуманные и старательные усилия связать свой опыт с основаниями информатики и представить его в более широком контексте, например рассказать о том, как анализировать и классифицировать различные проблемы с концептуально сходными характеристиками, – и вы поймете, почему эта книга должна занять одно из верхних мест в списке ожидающих прочтения.

Но раз содержание и выбранная методология полностью оправдывают потребность в такой книге, то почему никто не написал ее раньше? Я наблюдал за эволюцией этой книги (более того, имел честь быть ее активным участником и, надеюсь, привнес что-то полезное), она заняла несколько лет напряженного труда, несмотря даже на изобилие исходного материала, имевшегося в распоряжении авторов. И я понял, почему никто не попытался написать ее прежде: это не только трудно, но и требует правильного сочетания знаний и навыков (как раз такого, которое есть у авторов), поддержки со стороны редакторов (которую обеспечило издательство No Starch, терпеливо взявшее на себя процесс редактирования и мирившееся с задержками, неизбежными ввиду постоянно меняющейся обстановки в сфере наступательной безопасности) и, наконец, энтузиазма читателей предварительных вариантов книги (которые неустанно гнали работу к финишной черте).

Большое внимание в книге уделено тому, как достигается (или не достигается) доверие в современном компьютере и как различными уровнями и переходами между ними можно злоупотребить, нарушив предположения, принятые следующим уровнем. Задача в том, чтобы высветить две основные проблемы при реализации безопасности: композицию (надлежащее функционирование возможно только при правильном поведении нескольких взаимозависимых уровней) и допущения (каждый уровень должен предполагать, что предыдущий работает правильно). Авторы также делятся своим опытом применения инструментов и подходов к чрезвычайно сложному анализу поведения компонентов, работающих на ранних стадиях загрузки и на самых нижних уровнях операционной системы. Описание такого сквозного межуровневого подхода само по себе достойно отдельной книги и составляет книгу внутри книги. Как читатель я обожаю такие акции «два по цене одного», но лишь немногие авторы их предлагают.

Размышляя о природе знания, я пришел к выводу, что если ты что-то знаешь от и до, то можешь это хакнуть. Применение обратной разработки для понимания кода, который хакает обычное поведение системы, – это потрясающее техническое свершение, которое зачастую приносит много новых знаний. Возможность учиться у профессионалов, в послужном списке которых немало таких свершений и которые готовы поделиться своим пониманием, методами, рекомендациями и опытом, да еще следовать за ними в удобном для себя темпе, – это уникальный шанс. Не упустите его! Идите вглубь: пользуйтесь вспомогательными материалами, практикуйтесь, привлекайте сообщество, друзей и даже профессоров (которые, надеюсь, оценят, сколько полезного эта книга может дать аудитории). Эта книга не просто для чтения – она для изучения.

Родриго Рубира Бранко
(BSDaemon)

ВВЕДЕНИЕ

Идея этой книги пришла нам в голову, когда после публикации серии статей и постов в блогах о руткитах и буткитах мы поняли, что эта тема раскрыта куда хуже, чем заслуживает. Мы нутром чуяли, что картина шире, и захотели иметь книгу, которая сводила бы все воедино – обобщала разношерстное собрание трюков, наблюдений за архитектурой операционных систем, паттернов проектирования, применяемых атакующими, и новаторских придумок защитников. Мы искали подобную книгу и не нашли. И тогда решили написать такую, которую сами захотели бы прочитать.

На это у нас ушло четыре с половиной года – дольше, чем мы планировали, и гораздо дольше того срока, который, по нашим прикидкам, могли бы выдержать потенциальные читатели, поддержавшие ознакомительные редакции книги. Если вы один из них и все же читаете эту книгу, мы склоняем голову перед вашей преданностью!

За это время мы наблюдали совместную эволюцию средств нападения и защиты. В частности, мы видели, как Microsoft Windows положила конец нескольким важным направлениям проектирования руткитов и буткитов. Эту историю вы найдете на страницах книги.

Мы также были свидетелями появления новых классов вредоносного ПО, нацеливающихся на BIOS и прошивки чипсетов, до которых текущие средства защиты Windows не могли дотянуться. Мы расскажем об этой совместной эволюции и о том, каковы, скорее всего, будут ее следущие этапы.

Еще одна тема данной книги – развитие методов обратной разработки, нацеленных на ранние стадии процесса загрузки ОС. Так сложилось, что чем раньше выполняется код в длинном процессе загрузки ПК, тем сложнее наблюдать за ним. И эти затруднения долго

путали с безопасностью. Однако же, анализируя буткиты и импланты для BIOS, подрывающие такие низкоуровневые технологии ОС, как Secure Boot, мы видим, что и здесь «безопасность через неведение» ничем не лучше, чем в других разделах информатики. Проходит немного времени (совсем чуть-чуть на временной шкале интернета) – и подход, опирающийся на безопасность через неведение, начинает приносить атакующим больше выгод, чем защитникам. Эта идея еще не получила достаточного освещения в книгах на данную тему, так что мы попытаемся восполнить пробел.

Для кого предназначена эта книга

Мы адресуем свой труд очень широкому кругу специалистов по информационной безопасности, интересующихся тем, как передовые отряды вредоносного ПО обходят механизмы безопасности на уровне ОС. В фокусе нашего внимания вопросы обнаружения, обратной разработки и эффективного анализа этих продвинутых угроз. Каждая часть книги отражает новый этап эволюционного развития угроз – от появления в виде доказательства правильности концепции до последующего распространения в среде носителей угроз и, наконец, включения в тайный арсенал точно нацеленных атак.

Однако мы ориентируемся на более широкую аудиторию, состоящую не только из аналитиков вредоносного ПО. В частности, мы надеемся, что разработчики встраиваемых систем и специалисты по безопасности облачных систем сочтут книгу полезной, учитывая, сколь опасные масштабы может принять угроза руткитов и других имплантов в их экосистемах.

Структура книги

Мы начнем с изучения руткитов в части I, где познакомимся с внутренними механизмами ядра Windows, которое исторически послужило площадкой для отработки руткитов. Затем в части II мы сместим акцент на процесс загрузки ОС и буткиты, которые были разработаны после того, как Windows начала укреплять свой режим ядра. Мы разбиваем процесс загрузки на этапы с точки зрения атакующего, обращая особое внимание на новые схемы прошивки UEFI и их уязвимости. Наконец, в части III мы поговорим о компьютерно-технической экспертизе классических атак на ОС с помощью руткитов и более современных атак на BIOS и прошивки с помощью буткитов.

Часть I. Руткиты

В этой части описываются классические руткиты уровня ОС во времена их расцвета. Эти исторические примеры руткитов проливают свет на то, как атакующий видит операционную систему изнутри и находит способы надежно внедрить свои импланты, используя собственные структуры ОС.

Глава 1. Что такое руткит: пример TDL3. Мы начнем изучение работы руткитов с рассказа об одном из самых интересных руткитов своего времени, основанного на нашем собственном опыте столкновений с различными его вариантами и анализа угроз.

Г**лава 2. Руткит Festi: самый продвинутый бот для спама и DDoS-атак.** В этой главе анализируется знаменитый руткит Festi для рассылки спама и организации DDoS-атак, в котором использовались самые передовые на тот момент методы обеспечения скрытности. В частности, руткит включал собственный стек TCP/IP на уровне ядра.

Глава 3. Обнаружение заражения руткитом. В этой главе мы продолжим путешествие вглубь ядра операционной системы и расскажем о трюках, с помощью которых атакующие стремятся получить контроль над более глубокими уровнями ядра, например перехватывать системные события и вызовы.

Часть II. Буткиты

Вторая часть книги посвящена эволюции буткитов, условиям, подхлестнувшим эту эволюцию, и методам обратной разработки таких угроз. Мы увидим, как буткиты научились имплантировать себя в BIOS и эксплуатировать уязвимости прошивок UEFI.

Глава 4. Эволюция буткита. В этой главе мы подробно рассмотрим движущие силы совместной эволюции, которые породили буткиты и направляли их развитие. Мы опишем некоторые из первых обнаруженных буткитов, в частности знаменитый Elk Cloner.

Глава 5. Основы процесса загрузки операционной системы. Рассматриваются внутренние детали процесса загрузки Windows и его изменение со временем. Мы поговорим о главной загрузочной записи, таблицах разделов, конфигурационных данных и модуле *bootmgr*.

Глава 6. Безопасность процесса загрузки. Эта глава представляет собой обзор технологий защиты процесса загрузки Windows, в частности модули раннего запуска антивредоносного ПО (Early Launch Anti-Malware – ELAM), политику подписания кода режима ядра и ее уязвимости, а также новые средства безопасности на основе виртуализации.

Глава 7. Методы заражения буткитом. В этой главе мы тщательно проанализируем методы заражения загрузочных секторов и посмотрим, как они видоизменялись со временем. В качестве примеров будем использовать хорошо известные буткиты: TDL4, Gapz и Rovnix.

Глава 8. Статический анализ буткита с помощью IDA Pro. Здесь рассматриваются методы и инструменты статического анализа заражения буткитом. Для примера мы подвергнем анализу буткит TDL4 и предоставим материалы, которые вы сможете ис-

пользовать, когда будете заниматься анализом самостоятельно, в т. ч. образ диска.

Глава 9. Динамический анализ буткита: эмуляция и виртуализация. В этой главе мы сместим акцент на методы динамического анализа с применением эмулятора Bochs и встроенного в VMware отладчика GDB. И снова мы вместе с вами пройдем все шаги динамического анализа зараженных буткитами MBR и VBR.

Глава 10. Эволюция методов заражения MBR и VBR: Olmasco. В этой главе мы проследим эволюцию скрытых методов внедрения буткитов на нижние уровни процесса загрузки. В качестве примера рассмотрим буткит Olmasco и его методы заражения и закрепления, вредоносную функциональность и внедрение полезной нагрузки.

Глава 11. Буткиты начального загрузчика программы: Rovnix and Carberp. Здесь мы заглянем под капот двух самых сложных буткитов, Rovnix и Carberp, нацеленных на интернет-банкинг. Это были первые буткиты, избравшие мишенью начальный загрузчик (IPL) и ускользнувшие от тогдашних средств защиты. Для их анализа мы воспользуемся VMware и IDA Pro.

Глава 12. Gapz: продвинутое заражение VBR. Мы сорвем покров тайны с венца эволюции скрытных буткитов: таинственного руткита Gapz, в котором использовались самые передовые на тот момент методы заражения загрузочной записи тома (VBR).

Глава 13. Взлет программ-вымогателей, заражающих MBR. В этой главе мы рассмотрим, как буткиты превратились в вымогателей.

Глава 14. Сравнение процессов загрузки с помощью UEFI и MBR/VBR. Мы изучим, как устроен процесс загрузки в UEFI BIOS – данная информация важна для отслеживания эволюции новейших вредоносных программ.

Глава 15. Современные UEFI-буткиты. Эта глава включает наши собственные исследования различных имплантов BIOS – как доказательства правильности концепции, так и реально встречающиеся буткиты. Мы обсудим методы проникновения и закрепления в UEFI BIOS и рассмотрим реальный буткит такого типа, Computrace.

Глава 16. Уязвимости прошивок UEFI. Углубленный взгляд на различные классы уязвимостей современных BIOS, делающих возможным внедрение имплантов BIOS. На примерах рассматриваются уязвимости и эксплойты UEFI.

Часть III. Методы защиты и компьютерно-технической экспертизы

В последней части книги рассматривается компьютерно-техническая экспертиза буткитов, руткитов и других угроз BIOS.

Глава 17. Как работает безопасная загрузка UEFI. В этой главе мы детально рассмотрим, как устроена технология безопасной загрузки Secure Boot, ее эволюцию, уязвимости и эффективность.

Глава 18. Подходы к анализу скрытых файловых систем. Приводится обзор скрытых файловых систем, используемых вредоносными программами, и методов их обнаружения. Мы подвергнем разбору образ скрытой файловой системы и познакомимся с разработанным нами же инструментом, HiddenFsReader.

Глава 19. Компьютерно-техническая экспертиза BIOS/UEFI: подходы к получению и анализу прошивок. В этой, последней главе мы обсудим подходы к обнаружению самых передовых и современных угроз: аппаратные, микропрограммные и программные, с применением различных инструментов, в частности UEFITool и Chipsec.

Как читать эту книгу

Все примеры угроз, обсуждаемые в этой книге, а также дополнительные материалы имеются на сайте книги по адресу *https://nostarch.com/rootkits/*. Там же вы найдете ссылки на инструменты, применяемые для анализа буткитов, в т. ч. исходный код плагинов к IDA Pro, которыми мы пользовались в своих исследованиях.

ЧАСТЬ I

РУТКИТЫ

1

ЧТО ТАКОЕ РУТКИТ: TDL3

В этой главе мы познакомимся с руткитами на примере *TDL3*. Этот руткит для Windows – хороший пример продвинутой техники перехвата потоков управления и данных, в которой задействуются нижние уровни архитектуры ОС. Мы увидим, как TDL3 заражает систему и как он осуществляет диверсию против интерфейсов и механизмов ОС, чтобы закрепиться и остаться незамеченным.

Механизм заражения TDL3 напрямую загружает код в ядро Windows, поэтому средства контроля целостности ядра, введенные Microsoft в 64-разрядных версиях Windows, обезвредили этот руткит. Однако способы, применяемые TDL3 для вклинивания в ядро, все еще сохраняют ценность, будучи примером того, как можно надежно и эффективно встроиться в исполнение ядра, если удастся обойти механизмы контроля целостности. Как и во многих других руткитах, встраивание кода TDL3 в ядро опирается на ключевые особенности архитектуры ядра. В некотором смысле механизмы встраивания руткита могут оказаться лучшим руководством по истинной структуре ядра, чем официальная документация, и уж точно с ними ничто не сравнится, когда нужно понять недокументированные системные структуры данных и алгоритмы.

На самом деле преемником TDL3 стал TDL4, который имеет с TDL3 много общего в части ускользания от обнаружения и противодействия компьютерно-технической экспертизе (КТЭ), но принял на вооружение методы *буткитов* для обхода механизма подписания кода режима ядра в Windows, действующего в 64-разрядных системах (мы опишем эти методы в главе 7).

В этой главе мы поговорим о конкретных интерфейсах и механизмах ОС, которые подрывает TDL3. Мы объясним, как TDL3 и похожие на него руткиты устроены и как они работают, а затем в части II обсудим методы и инструменты, позволяющие их обнаружить, наблюдать и анализировать.

История распространения TDL3 по миру

Впервые обнаруженный в 2010 году[1], руткит TDL3 стал одним из самых изощренных образчиков вредоносного ПО на тот момент. Его механизмы маскировки стали вызовом для всей антивирусной индустрии (так же произошло и с пришедшим ему на смену буткитом TDL4, ставшим первым широко распространившимся буткитом для платформы x64).

Примечание *Это семейство вредоносного ПО известно также под названиями TDSS, Olmarik или Alureon. Такое изобилие имен для одного и того же семейства – обычное дело, потому что производители антивирусов часто дают вредоносам разные имена в своих отчетах. Также исследовательские группы нередко присваивают разные имена различным компонентам одной атаки, особенно на разных этапах анализа.*

TDL3 распространялся в соответствии с бизнес-моделью *оплаты за количество установок* (Pay-Per-Install – PPI) через партнерские программы DogmaMillions и GangstaBucks (обеим уже пришел конец). Схема PPI, популярная у киберпреступников, похожа на схемы, часто применяемые для распространения инструментальных панелей браузеров. Распространители таких панелей отслеживают их использование, создавая специальные сборки со встроенным уникальным идентификатором (UID) для пакетов, распространяемых через конкретный канал. Это позволяет разработчику узнать количество установок (пользователей) с данным UID, а значит, вычислить доход, генерируемый каждым каналом. Точно так же информация о дистрибуторе встраивалась в исполняемый файл руткита TDL3, а специальные серверы подсчитывали количество установок через данного дистрибутора и начисляли ему плату.

Партнерам киберпреступных групп присваивался уникальный логин и пароль, с которым можно было связать количество установок. У каждого партнера был также персональный менеджер, с которым можно было проконсультироваться в случае технических проблем.

---

[1] *http://static1.esetstatic.com/us/resources/white-papers/TDL3-Analysis.pdf.*

Чтобы свести к минимуму риск обнаружения антивирусом, партнер часто перепаковывал дистрибутивный пакет и использовал изощренные защитные меры, чтобы обнаружить присутствие отладчиков или виртуальных машин, что затрудняло анализ вредоноса исследователями[1]. Партнерам также запрещалось использовать ресурсы типа VirusTotal, проверяющие возможность обнаружения текущей версии защитными программами, за это полагались штрафы. Опасались, что образцы, отправленные на VirusTotal, могут привлечь внимание исследователей безопасности и тем самым сократить полезную жизнь вредоносного ПО. Если у дистрибуторов вредоноса возникали вопросы по поводу его скрытности, то им было рекомендовано использовать службы, управляемые разработчиком, – аналогичные VirusTotal, но гарантирующие, что отправленные образцы не попадут в руки производителей защитных программ.

Процедура заражения

После того как инфектор TDL3 попал в систему пользователя по одному из каналов распространения, он инициирует процесс заражения. Чтобы пережить перезагрузку системы, TDL3 инфицирует один из первоочередных драйверов (boot-start drivers), необходимых для загрузки ОС, внедряя вредоносный код в исполняемый файл драйвера. Первоочередные драйверы загружаются вместе с образом ядра на ранней стадии процесса инициализации ОС. В результате при загрузке зараженной машины загружается модифицированный драйвер и вредоносный код перехватывает управление инициализацией.

Итак, при работе в режиме ядра процедура заражения просматривает список первоочередных драйверов, поддерживающих важнейшие компоненты операционной системы, и случайным образом выбирает жертву. Каждый элемент списка описывается недокументированной структурой `KLDR_DATA_TABLE_ENTRY`, показанной в листинге 1.1, на которую ссылается поле `DriverSection` структуры `DRIVER_OBJECT`. Любому драйверу, работающему в режиме ядра, соответствует структура `DRIVER_OBJECT`.

Листинг 1.1. *Структура* `KLDR_DATA_TABLE_ENTRY`*, на которую ссылается поле* `DriverSection`

```
typedef struct _KLDR_DATA_TABLE_ENTRY {
    LIST_ENTRY InLoadOrderLinks;
    LIST_ENTRY InMemoryOrderLinks;
    LIST_ENTRY InInitializationOrderLinks;
    PVOID ExceptionTable;
```

[1] Rodrigo Rubira Branco, Gabriel Negreira Barbosa, and Pedro Drimel Neto «Scientific but Not Academic Overview of Malware Anti-Debugging, Anti-Disassembly and Anti-VM Technologies» (работа представлена на конференции Black Hat USA 2012, состоявшейся 21–26 июля в Лас-Вегасе, штат Невада), *https://media.blackhat.com/bh-us-12/Briefings/Branco/BH_US_12_Branco_Scientific_Academic_WP.pdf*.

```
    ULONG ExceptionTableSize;
    PVOID GpValue;
    PNON_PAGED_DEBUG_INFO NonPagedDebugInfo;
    PVOID ImageBase;
    PVOID EntryPoint;
    ULONG SizeOfImage;
    UNICODE_STRING FullImageName;
    UNICODE_STRING BaseImageName;
    ULONG Flags;
    USHORT LoadCount;
    USHORT Reserved1;
    PVOID SectionPointer;
    ULONG CheckSum;
    PVOID LoadedImports;
    PVOID PatchInformation;
} KLDR_DATA_TABLE_ENTRY, *PKLDR_DATA_TABLE_ENTRY;
```

Выбрав драйвер-жертву, инфектор TDL3 модифицирует образ драйвера в памяти, перезаписывая первые несколько сотен байтов его секции ресурсов, *.rsrc*, вредоносным заголовком. Заголовок очень прост: он всего лишь подгружает с жесткого диска оставшуюся часть вредоносного кода во время загрузки.

Перезаписанные оригинальные байты секции *.rsrc*, необходимые для правильной работы драйвера, сохраняются в файле *rsrc.dat* в скрытой файловой системе, поддерживаемой вредоносом. (Отметим, что после заражения размер файла драйвера не изменяется.) Выполнив эту модификацию, TDL3 изменяет поле адреса точки входа в заголовке PE-файла, так чтобы оно указывало на загрузчик вредоноса. Теперь точка входа в зараженный драйвер находится в секции ресурсов, чего при нормальных условиях быть не должно. На рис. 1.1 показан первоочередной драйвер до и после заражения, блок **Заголовок** ссылается на один из заголовков секций.

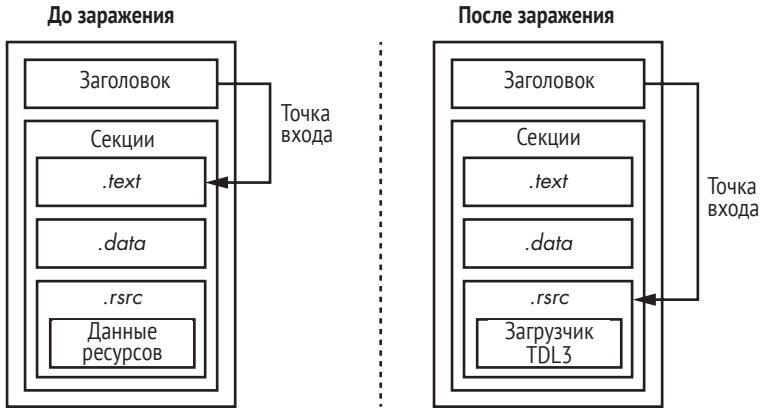

Рис. 1.1. Модификации первоочередного драйвера, работающего в режиме ядра, после заражения системы

Эта схема заражения файлов в формате PE – основном двоичном формате исполняемых файлов и динамически компонуемых библиотек (DLL) в Windows – типична скорее для вирусных инфекторов, а не для руткитов. Заголовок PE и таблица секций – неотъемлемая принадлежность любого PE-файла. Заголовок PE содержит важнейшую информацию о местоположении кода и данных, системных метаданных, размере стека и т. д., а в таблице секций хранятся сведения о секциях исполняемого файла и их местоположении.

Чтобы завершить процесс заражения, вредонос перезаписывает поле каталога метаданных .NET в заголовке PE теми же значениями, которые находятся в поле каталога данных безопасности. Вероятно, смысл этого шага в том, чтобы помешать статическому анализу зараженных образов, поскольку это может привести к ошибке в процессе разбора заголовка PE стандартными инструментами анализа вредоносных программ. И действительно, попытка загрузить такой образ вызывала крах IDA Pro версии 5.6 – с тех пор эта ошибка исправлена. Согласно спецификации формата PE/COFF, опубликованной Microsoft, каталог метаданных .NET содержит данные, необходимые общеязыковой среде выполнения (Common Language Runtime – CLR) для загрузки и выполнения .NET-приложений. Однако это поле несущественно для первоочередных драйверов, работающих в режиме ядра, потому что все они – непосредственно исполняемые файлы, не содержащие управляемого кода. Поэтому это поле не проверяется загрузчиком ОС, что позволяет зараженному драйверу успешно загрузиться, хотя формально его содержимое некорректно.

Заметим, что описанная техника заражения TDL3 ограничена – она работает только на 32-разрядных платформах, потому что в 64-разрядных системах действует политика подписания кода режима ядра, организующая обязательную проверку целостности кода. Так как содержимое драйвера в процессе заражения системы изменено, его цифровая подпись уже недействительна, и 64-разрядная ОС откажется его загружать. Разработчики вредоносного ПО ответили на это буткитом TDL4. Саму политику и способ ее обхода мы обсудим в главе 6.

Управление потоком данных

Чтобы обеспечить себе скрытность, руткиты ядра должны изменить поток управления или поток данных (или оба) в системных вызовах в тех случаях, когда оригинальный поток управления или данных ОС мог бы выявить присутствие компонентов вредоноса на диске (например, файлов) или в процессе выполнения (например, в структурах данных ядра). Для этого руткиты обычно внедряют свой код в какое-то место на пути выполнения системного вызова; размещение таких точек подключения – один из самых поучительных аспектов руткитов.

Приходи со своим компоновщиком

Подключение – это по существу компоновка. Современные руткиты содержат собственные компоновщики для связывания их кода

с системой; мы назвали этот паттерн проектирования «Приходи со своим компоновщиком» (Bring Your Own Linker). Чтобы скрытно внедрить такие «компоновщики», TDL3 следует нескольким распространенным принципам проектирования.

Во-первых, жертва должна и дальше работать надежно, несмотря на дополнительно внедренный код, поскольку атакующий ничего не приобретет, но много потеряет от краха программы-жертвы. С точки зрения программной инженерии, подключение – это форма композиции программ, которая требует тщательного планирования. Атакующий должен позаботиться о том, чтобы в момент выполнения нового кода система находилась в предсказуемом состоянии, иначе возможен крах или аномальное поведение, которое может привлечь внимание пользователя. Может показаться, что размещение точек подключения ограничено только воображением автора руткита, но на самом деле автор должен соблюдать стабильные границы ПО и не нарушать согласованные интерфейсы. Неудивительно поэтому, что подключение нацеливается на те же самые структуры – документированные или нет, – которые используются при стандартной динамической компоновке. Таблицы обратных вызовов, методов и других указателей на функции, которые связывают уровни абстрагирования или программные модули, – самые безопасные места для точек подключения; также хорошо работает подключение к прологам функций.

Во-вторых, место точки подключения не должно быть совсем уж очевидным. Первые руткиты подключались к таблице системных вызовов ядра верхнего уровня, но эта техника быстро вышла из моды, т. к. оказалась слишком заметной. Ее обнаружение в рутките Sony 2005 года[1] было сочтено анахронизмом и вызвало немалое удивление. По мере отработки руткитов точки подключения сместились вниз по стеку – от главных таблиц диспетчеризации системных вызовов в подсистемы ОС, которые предлагают единые API для различных реализаций, например виртуальной файловой системы (Virtual File System – VFS), и еще дальше – в таблицы методов и обратных вызовов конкретных драйверов. TDL3 дает особенно наглядный пример такой миграции.

Как работают точки подключения в TDL3

Чтобы остаться незамеченным, TDL3 использовал довольно остроумную технику подключения, которая раньше никогда не встречалась в полевых условиях: он перехватывал запросы на чтение и запись к жесткому диску на уровне драйвера порта/мини-порта устройства хранения (драйвер аппаратного устройства хранения, который находится в самом низу стека драйверов). *Драйверы портов* – это системные модули, предоставляющие интерфейс программирования драйверов мини-портов, которые поставляются производителями

[1] *https://blogs.technet.microsoft.com/markrussinovich/2005/10/31/sony-rootkits-and-digital-rights-management-gone-too-far/.*

соответствующих устройств хранения. На рис. 1.2 показана архитектура стека драйверов устройства хранения в Microsoft Windows.

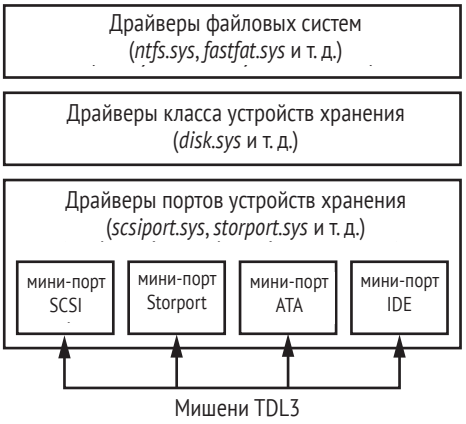

Рис. 1.2. Архитектура стека драйверов устройства хранения в Microsoft Windows

Обработка пакета запроса ввода-вывода (I/O request packet – IRP), адресованного какому-то объекту на устройстве хранения, начинается на уровне драйвера файловой системы. Этот драйвер определяет, на каком устройстве хранится объект (например, раздел и экстент диска – непрерывная область диска, зарезервированная для файловой системы), и отправляет другой IRP объекту драйвера класса устройств хранения. Последний, в свою очередь, транслирует запрос ввода-вывода в объект соответствующего устройства мини-порта.

Согласно документации по комплекту инструментов для разработки драйверов (Windows Driver Kit – WDK), драйверы портов устройств хранения предоставляют интерфейс между аппаратно-независимым драйвером класса устройств и зависящим от архитектуры хоста (host-based architecture – HBA) драйвером мини-порта. Располагая этим интерфейсом, TDL3 организует точку подключения к ядру на самом нижнем аппаратно-независимом уровне в стеке драйверов устройства хранения, обходя тем самым средства мониторинга и защиты, работающие на уровне файловой системы или драйвера класса устройств. Такие точки подключения могут быть обнаружены только инструментами, знающими о нормальном составе этих таблиц для конкретного набора устройств или о заведомо исправной конфигурации конкретной машины.

Чтобы успешно реализовать подобное подключение, TDL3 сначала получает указатель на объект драйвера мини-порта соответствующего объекта устройства. Точнее, код подключения пытается открыть описатель *\??\PhysicalDriveXX* (где *XX* соответствует номеру жесткого диска), но на самом деле эта строка является символической ссылкой на объект устройства *\Device\HardDisk0\DR0*, которое создано драйвером класса устройств хранения. Спускаясь вниз по стеку устройств

от *\Device\HardDisk0\DR0*, мы в самом низу найдем объект мини-порта. Имея объект мини-порта устройства, уже легко получить указатель на объект его драйвера из поля `DriverObject` документированной структуры `DEVICE_OBJECT`. В этот момент вредонос располагает всей информацией, необходимой для подключения к стеку драйверов устройства хранения.

Затем TDL3 создает новый вредоносный объект драйвера и перезаписывает поле `DriverObject` в объекте драйвера мини-порта указателем на вновь созданный драйвер, как показано на рис. 1.3. Это позволяет вредоносу перехватывать запросы ввода-вывода к соответствующему жесткому диску, т. к. адреса всех обработчиков указаны в структуре объекта драйвера, а именно в массиве `MajorFunction` структуры `DRIVER_OBJECT`.

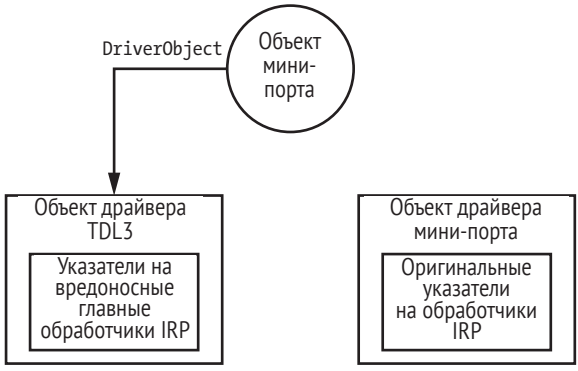

Рис. 1.3. Подключение к объекту драйвера мини-порта устройства хранения

Вредоносные обработчики, показанные на рис. 1.3, перехватывают запросы `IRP_MJ_INTERNAL_CONTROL` и `IRP_MJ_DEVICE_CONTROL` для следующих кодов управления вводом-выводом (Input/Output Control – IOCTL), чтобы отслеживать и модифицировать запросы ввода-вывода к тем областям жесткого диска, где хранятся зараженный драйвер и образ скрытой файловой системы, реализованной вредоносом:

- `IOCTL_ATA_PASS_THROUGH_DIRECT`;
- `IOCTL_ATA_PASS_THROUGH`.

TDL3 не дает инструментам Windows прочитать или случайно перезаписать сектора жесткого диска, занятые защищенными данными, и тем самым обеспечивает скрытность и целостность руткита. В случае операции чтения TDL3 обнуляет возвращаемый буфер после ее завершения, а операцию записи вообще пропускает. Техника подключения TDL3 позволяет ему обходить некоторые методы обнаружения вмешательств в ядро, поскольку внесенные модификации не затрагивают часто защищаемые и подвергаемые мониторингу области, в т. ч. системные модули, таблицы дескрипторов системных служб (System

Service Descriptor Table – SSDT), глобальную таблицу дескрипторов (Global Descriptor Table – GDT) и таблицу дескрипторов прерываний (Interrupt Descriptor Table – IDT). Пришедший ему на смену буткит TDL4 применяет тот же подход для обхода защиты от модификаций ядра PatchGuard, включенной в 64-разрядные операционные системы Windows, поскольку унаследовал изрядную долю функциональности в режиме ядра от TDL3, в т. ч. и технику подключения к драйверу мини-порта устройства хранения.

Скрытая файловая система

TDL3 был первой вредоносной системой, хранящей свои конфигурационные файлы и полезную нагрузку в скрытой зашифрованной области устройства хранения, вместо того чтобы пользоваться услугами файловой системы. Сегодня подход TDL3 принят и другими сложными угрозами, например буткитами Rovnix, ZeroAccess, Avatar и Gapz.

Эта техника скрытного хранения значительно усложняет КТЭ, потому что вредоносные данные хранятся в зашифрованном контейнере, находящемся где-то на жестком диске вне области, зарезервированной ОС для собственной файловой системы. В то же время вредонос может обращаться к содержимому скрытой файловой системы с помощью стандартных Win32 API типа `CreateFile`, `ReadFile`, `WriteFile` и `CloseHandle`. Это упрощает разработку полезной нагрузки, т. к. авторы могут читать и записывать ее в область хранения, не занимаясь изобретением и поддержкой специальных интерфейсов. Это важное проектное решение, поскольку наряду с использованием стандартных интерфейсов для подключения оно повышает надежность руткита; с точки зрения программной инженерии, это хороший и достойный подражания пример повторного использования кода! Формула успеха от самого исполнительного директора Microsoft звучит так: «Разработчики, разработчики, разработчики и еще раз разработчики!» Иными словами, навыки имеющихся разработчиков считаются ценным капиталом. В TDL3 точно так же задействуются навыки программирования Windows, имеющиеся у разработчиков, перешедших на темную сторону, – для облегчения перехода и повышения надежности вредоносного кода.

TDL3 размещает образ своей скрытой файловой системы в секторах жесткого диска, не занятых собственной файловой системой ОС. Образ растет от конца к началу диска, т. е. может в конечном итоге перекрыть данные пользователя, хранящиеся в файловой системе. Образ разделен на блоки по 1024 байта. Первый блок (в конце жесткого диска) содержит таблицу файлов, элементы которой описывают содержащиеся в файловой системе файлы и включают следующую информацию:

- имя файла длиной не более 16 символов, включая завершающий нуль;
- размер файла;

- истинное смещение файла, вычисляемое путем вычитания начального смещения файла, умноженного на 1024, из смещения начала файловой системы;
- время создания файла.

Содержимое файловой системы шифруется нестандартным алгоритмом поблочно. В разных версиях руткита используются разные алгоритмы. Например, в некоторых вариантах использовался шифр RC4 с ключом, равным логическому адресу блока (logical block address – LBA) первого сектора блока. А в другом варианте применялось шифрование посредством выполнения XOR с фиксированным ключом 0x54, увеличивающимся после каждой операции XOR, – в итоге получался довольно слабый шифр, в котором легко было опознать специфический паттерн, соответствующий зашифрованному блоку, содержащему нули.

Полезная нагрузка, работающая в режиме пользователя, обращается к скрытой файловой системе, открывая описатель объекта устройства *\Device\XXXXXXXX\YYYYYYYY*, где *XXXXXXXX* и *YYYYYYYY* – случайно сгенерированные шестнадцатеричные числа. Отметим, что код, выполняемый для доступа к этому устройству хранения, пролегает через многие компоненты Windows, которые, хочется надеяться, уже отлажены Microsoft и потому безопасны. Имя объекта устройства генерируется заново при каждой загрузке системы, а затем передается в качестве параметра модулям полезной нагрузки. Руткит отвечает за обработку запросов ввода-вывода к этой файловой системе. Например, когда модуль полезной нагрузки выполняет операцию с файлом в скрытой области хранения, ОС передает запрос руткиту и вызывает для обработки его функции.

На примере этого паттерна проектирования TDL3 иллюстрирует общую тенденцию, характерную для руткитов. Вместо того чтобы писать новый код для всех операций, возлагая на сторонних разработчиков вредоносного ПО бремя изучения всех тонкостей этого кода, руткит опирается на уже имеющуюся и знакомую функциональность Windows – с условием, что применяемые трюки и используемые интерфейсы Windows не слишком хорошо известны. Конкретные методы заражения эволюционируют вместе с изменением мер защиты, но сам подход не меняется, потому что следует принципам обеспечения надежности кода, общим для разработки благодетельного и вредоносного программного обеспечения.

Итог: TDL3 встретил свою Немезиду

Как мы только что видели, TDL3 – это изощренный руткит, в котором впервые было опробовано несколько методов скрытного закрепления в зараженной системе. Его способ подключения к ядру и скрытая файловая система не остались не замеченными другими разработчиками вредоносного ПО и со временем вошли в иные комплексные

угрозы. Единственное ограничение процедуры заражения – то, что она ориентирована только на 32-разрядные системы.

Поначалу TDL3 выполнял все, что задумали его разработчики, но по мере увеличения количества 64-разрядных систем, возник спрос на заражение систем на платформе x64. Для этого разработчикам вредоносов пришлось придумать, как обойти политику подписания кода, работающего в 64-разрядном ядре, и загрузить свой код в адресное пространство ядра. В главе 7 мы увидим, что авторы TDL3 выбрали технологию *буткитов*, чтобы ускользнуть от проверки подписи.