обнаружение компрометации ядер Linux и xBSD или руткиты тоже оставляют следы…

крис касперски, a.k.a. мыщъх, a.k.a. nezumi, no-email

рост популяции руткитов, оккупировавших никсы, продолжает свое наступление ударными темпами, поражая системы, не обремененные антивирусами, ядерными отладчиками и прочими защитными механизмами, уже давно ставшим привычными средствами обороны в мире windows. руководства по обнаружению руткитов устаревают быстрее, чем совершенствуются методики внедрения. рецептурные справочники абсолютно бесполезны и нужно курить что-то концептуальное.

введение

Согласно общепринятой классификации, руткитами называются программы (обычно безвредные), предназначение для сокрытия сетевых соединений, процессов и дисковых файлов, других программ чаще всего довольно агрессивных по своей натуре (а чего им тогда шифроваться?!). Классификация это, конечно, прекрасно, но вот попытка натянуть ее на реальную ситуацию срывает крышу и высаживает на измену. Огромное количество червей (и прочей малвари) имеет встроенные руткиты с полиморфным движком.

Короче, на практике нам приходится бороться не с руткитами в чистом виде (тоже мне, понимаешь, сферические кони в вакууме), а с различными механизмами маскировки. Вот времена MS-DOS все было проще — маскирующиеся программы назывались "стелсами" независимо от степени их вредоносности. Поэтому, чтобы вернуть крышу на место, условимся понимать под руткитами любую нечисть, занимающуюся сокрытием системных объектов (файлов, процессов, сетевых соединений). Своих или чужих — не важно.

Попробуем разобраться — как же работает эта шапка-невидимка и какие способы обнаружения руткитов существуют?

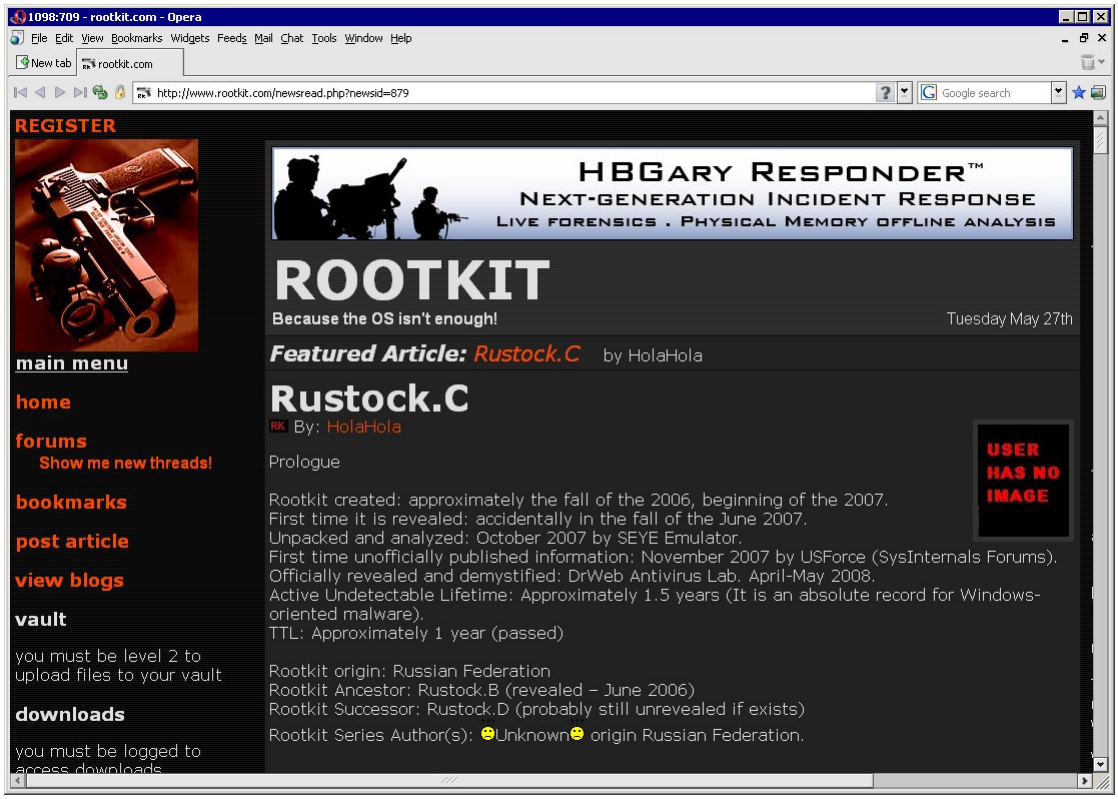

Рисунок 1 www.rootkit.com – основное место около-руткитной туссовки

кочевые племена против оседлых форм жизни

Существуют два типа руткитов: первые, внедрясь в систему, создают новые файлы или модифицируют уже существующие, получая управление при каждой загрузке операционной системы. Другие же — вообще не прикасаются к диску, не создают новых процессов, ограничиваясь модификацией оперативной памяти. Естественно, руткиты такого типа умирают при перезагрузке, и выглядят не слишком-то жизнеспособными, однако, до тех пор пока дыра, через которую проникает руткит, остается не залатанной, он будет приходить вновь и вновь. Закрытие дыры ничего не изменит, ибо там где есть одна дыра, найдутся и другие — создателю руткита достаточно, выражаясь образным языком, передернуть затвор, переписав несколько десятков строк кода, ответственных за внедрение первичного загрузчика в целевую систему.

В распределенных сетях (ботнетах) перезагрузка одного или нескольких узлов — это вообще не проблема, к тому же, после перезагрузки узел будет инфицирован вновь, причем этот факт очень трудно обнаружить, ведь никаких изменений на диске нет! А сетевые соединения современные руткиты скрывают весьма эффективно. Прошли те времена, когда открытые порты обнаруживались тривиальным сканированием с соседней машины. Продвинутые руткиты не открывают никаких портов, они садятся на сетевой интерфейс, контролируя трафик и модифицируя определенные поля в заголовках TCP/IP пакетов, значения которых согласно RFC выбираются случайным образом. Скремблер скроет факт модификации (т. к. независимо от передаваемых руткитом данных, мы получим такое же хаотичное распределение, как и на незараженной машине), а несимметричный шифратор предотвратит декодирование перехваченной информации, даже если мы заведомо знаем, что руткит есть.

А откуда мы узнаем, что он у нас есть? Объем трафика не изменяется. Никаких изменений на диске не наблюдается, что кардинальным образом отличается от руткитов первого типа, которые обнаруживаются настолько тривиально, насколько это вообще возможно себе представить. Загружаемся с LiveCD и проверяем контрольные суммы всех файлов (или просто осуществляем побайтовое сравнение с дистрибутивом). Конечно, для серверов такой способ не очень-то пригоден — их вообще лучше не перезагружать, но, сервера, критичные к перезагрузкам, обычно оснащены RAID-массивами с hot-plug'ом, так что просто вытаскиваем один набор дисков из матрицы, ставим его на другую машину, проверяем контрольную сумму и делаем орг. выводы.

Короче говоря, руткиты, вносящие изменения в файловую систему, нам совершенно неинтересны и дальше мы будем говорить исключительно о заразе, обитающей непосредственно в оперативной памяти и получающей управление путем модификации ядра, поскольку на прикладном уровне нормальному руткиту делать нечего.

>>> врезка

тема сокрытия трафика подробно разжевана Жанной Рутковской, так что не будем повторять уже сказанное, а просто откроем ее блог и почитаем http://theinvisiblethings.blogspot.com/

Рисунок 2 сайт Жанны Рутковской, посвященный технике создания и поимке руткитов нового поколения

методы борьбы или была бы катана — сделал харакири

Прежде чем продвигаться вглубь, сразу выбросим на помойку несколько популярных, но безнадежно устаревших способов борьбы с руткитами. Чтение памяти ядра через /dev/[k]mem (при активом рутките!) — это курам на смех. Поиск следов компрометации при помощи GDB – из той же оперы. Руткиту ничего не стоит отследить обращение к любому файлу/устройству, "вычистив" следы своего пребывания или совершить "харакири" при запуске GDB. Чуть сложнее — ввести в заблуждение GDB, оставаясь при этом активным, живым и здоровым.

А достойных отладчиков ядерного уровня под никсы нет. Ну, не то, чтобы совсем нет, но в штатный комплект поставки уж точно не входит ни один. Хорошо еще, если установка отладчика не требует перекомпиляции ядра, не говоря уже о перезагрузке. Самих же отладчиков довольно много: NLKD, KDB, LinIce, DDB и ни один из них не обладает неоспоримыми преимуществами перед остальными. И у каждого администратора есть свой любимец, что делает невозможным создание пошаговых руководств в стиле "а сейчас мы нажмем такую-то клавишу", что в конечном счете даже к лучшему. К тому же, для ловли руткитов иметь готовый к употреблению отладчик совершенно необязательно. Достаточно написать несложный драйвер, точнее, загружаемый модуль ядра, считывающий и передающий на прикладной уровень, все критичные к перехвату структуры данных вместе с машинным кодом (естественно, ядро должно быть скомпилировано с поддержкой модульности). Что это за данные — мы сейчас выясним.

три магических аббревиатуры — GDT, LDT, IDT

Сокрытие чего бы там ни было базируется на перехвате/модификации ядерных структур данных/системного кода. Способов перехвата придумано огромное множество и

каждый день появляются все новые. Однако, количество самих системных структур ограничено, что существенно упрощает борьбу с заразой.

Начнем с простого — с таблиц глобальных/локальных дескрипторов (Global/Local Description Table или, сокращенно, GDT/LDT соответственно), хранящих базовые адреса, лимиты и атрибуты селекторов. Чем они могут помочь руткиту? Ну, кое-чем могут помочь. Linux/BSD используют плоскую модель памяти, при которой селекторы CS (код), DS (данные) и SS (стек) "распахнуты" на все адресное пространство: от нуля до самых верхних его окраин. Создание нового селектора с базой, отличной от нуля, с последующей его загрузкой в одни из сегментных регистров — существенно затрудняет дизассемблирование руткита, особенно тех экземпляров, что выдраны из памяти чужой машины и таблицы дескрипторов в распоряжении реверсера нет и уже не будет (руткит умер). Грубо говоря, в этом случае мы вообще не можем сказать к каким данным осуществляется обращение, ведь база селектора неизвестна! Реверсеров и сотрудников антивирусных компаний такие руткиты просто бесят, затягивая анализ, а вместе с ним и приготовление "вакцины".

Побочным эффектом данного антиотладочного приема становится появление новых селекторов в таблице дескрипторов, которых там никогда не наблюдалось ранее. Отладчики ядерного уровня позволяют просматривать таблицы дескрипторов в удобочитаемом виде, но при активном рутките пользоваться отладчиком не рекомендуется, лучше написать свой загружаемый модуль ядра, считывающий содержимое таблицы дескрипторов командами SGDT/SLDT, описанных (вместе с форматами самих таблиц) в документации на процессоры Intel и AMD.

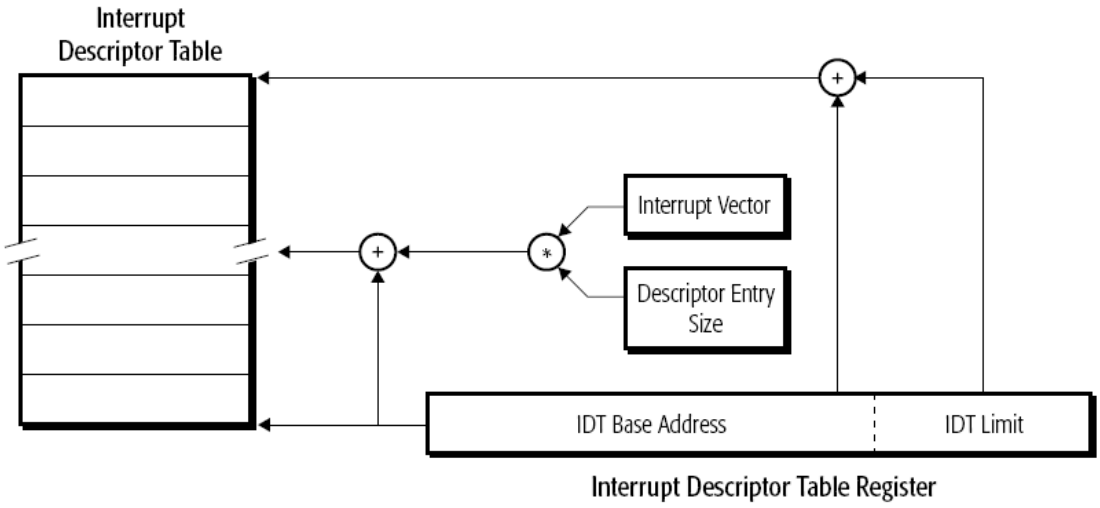

Рисунок 3 обработка прерываний в защищенном режиме

Огромное количество руткитов модифицирует таблицу дескрипторов прерываний (Interrupt Description Table или, сокращенно, IDT), позволяющую им перехватывать любые прерывания и исключения, в том числе и системные вызовы, реализованные на некоторых системах именно как прерывания, но о системных вызовах мы еще поговорим, а пока лишь отметим, что модификация IDT позволяет руткиту перехватывать обращения к страницам, вытесненным на диск (т. к. при обращении к ним возникает исключение Page Fault), а так же прочие исключения, например, общение исключение защиты (General Protection Fault), отладочное и пошаговое исключение (отличный способ борьбы с отладчиками), не говоря уже за прерывания, поступающие от аппаратных устройств — клавиатуры, сетевой карты и прочего оборудования, прямое обращение к которому очень полезно для сокрытия "преступной" деятельности.

Таблица прерываний, отображаемая отладчиками в удобочитаемом виде, может быть прочитана процессорной командой SIDT, что намного надежнее, поскольку ее выполнение руткит перехватить уже не в состоянии.

практическая магия системных вызовов

Системные вызовы — это основной механизм взаимодействия ядра с прикладными процессами, обеспечивающий базовый функционал и абстрагирующий приложения от

особенностей конкретного оборудования. В частности, системный вызов sys_read обеспечивает унифицированный способ чтения данных с файлов, устройств и псевдоустройств. Соответственно, перехват sys_read позволяет руткиту контролировать обращения ко всем файлам и (псевдо)устройствам. Даже если руткит не создает и не скрывает никаких файлов, ему необходимо заблокировать возможность чтения памяти ядра, иначе любой, даже самый примитивный антивирус его тут же обнаружит.

В зависимости от типа и версии ОС системные вызовы реализуются по разному. Самый древний механизм — это далекий вызов по селектору семь, смещение ноль — CALL FAR 0007h:00000000h (или, тоже самое, но в AT&T синтаксисе — lcall $7,$0). Работает практически на всех x86-клонах UNIX'a, однако, практического значения не имеет, поскольку им пользуются только некоторые ассемблерные программы в стиле "hello, world!", ну и… вирусы, так же написанные на ассемблере.

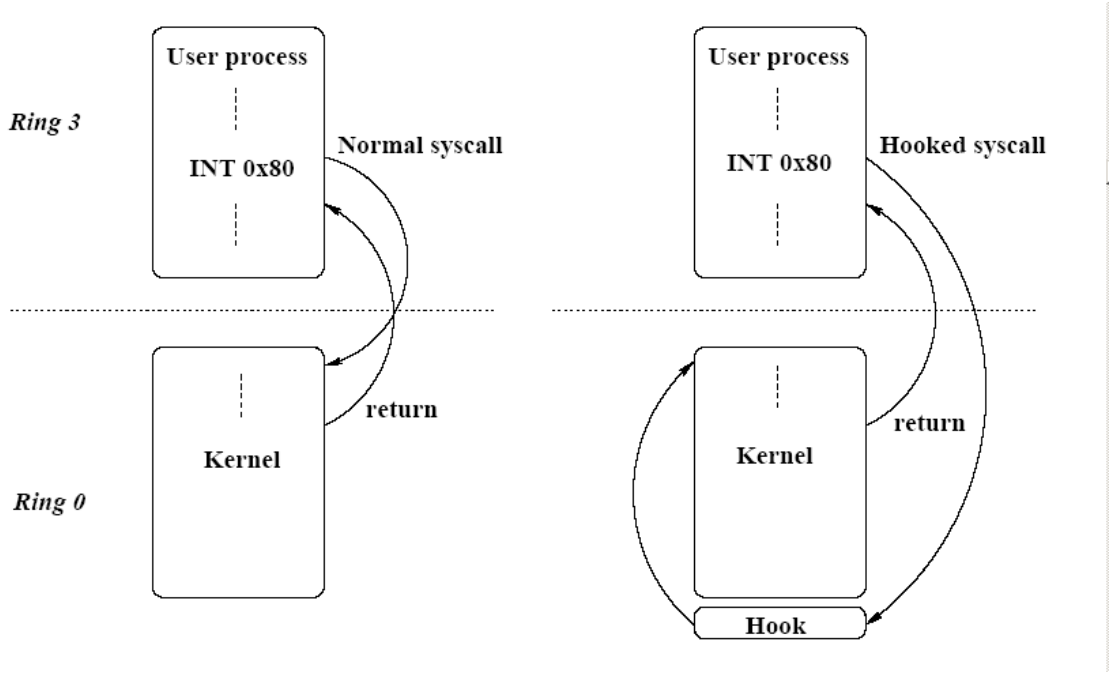

Рисунок 4 перехват системных вызовов через INT 80h (слева показана незараженная система, справа — инфицированная)

Стандартом де-факто стал программный вызов прерывания 80h (INT 80h), работающий как в Linux, так и во Free-BSD. Как руткит может перехватить его? Да очень просто — посредством модификации таблицы дескрипторов прерываний, переназначая вектор 80h на свой собственный код. Однако, это не единственный вариант. Стандартно INT 80h передает управление на функцию system_call, адрес которой можно определить по файлу System.map, если он, конечно, не удален администратором по соображениям безопасности, — тогда руткит либо читает вектор 80h через SIDT, либо находит system_call эвристическим путем, поскольку она, как и любой другой обработчик прерывания, содержит довольно характерный код. Вставив в начало (середину) этой функции команду перехода на свое тело, руткит будет получать управление при всяком вызове системного вызова.

Следовательно, мы должны считать код функции system_call из памяти, сравнив его с оригиналом, который можно позаимствовать из неупакованного ядра, выдернутого из дистрибутивного диска (распаковка ядра — тема отдельного разговора уже порядком навязшего в зубах и запутавшегося в хвосте).

После выполнения системного вызова управление получает другая интересная функция — ret_from_sys_call, идущая следом за system_call и так же, как и system_call, присутствующая в System.map. Ее перехватывают многие руткиты, что вполне логично, поскольку "вычистить" следы своего пребывания лучше всего _после_ отработки системного вызова, а не до него. Но вот популярные руководства по поиску руткитов об этом почему-то забывают, а зря!!! ret_from_sys_call следует проверять в первую очередь, так же сравнивая ее код с кодом оригинальной ret_from_sys_call, ну или просто дизассемблируя его на предмет наличия посторонних переходов.

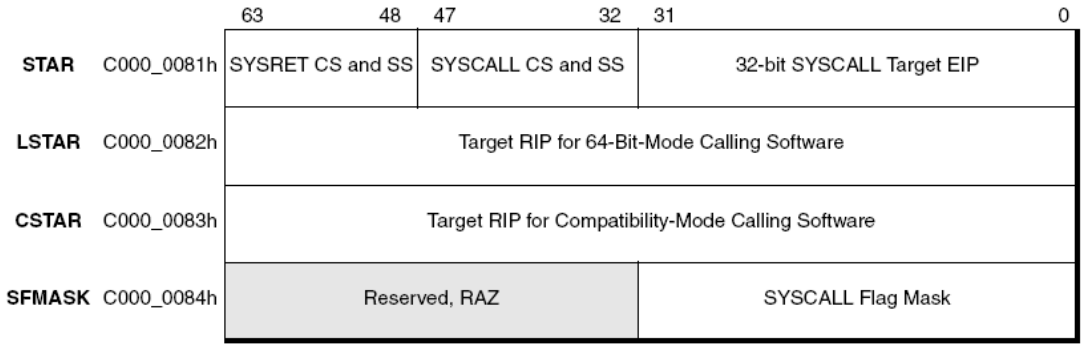

Рисунок 5 SYSCALL и используемые им MSR-регистры

Начиная с 2.5 версии ядра, Linux поддерживает механизм быстрых системных вызовов, реализуемый командами SYSENTER/SYSEXIT (Intel) и SYSCALL/SYSRET (AMD), существенно облегчающий перехват и делающий его чрезвычайно трудно заметным. Команда SYSENTER передает управление с 3-го кольца прикладного уровня на ядерный уровень, используя специальные MSR-регистры, а конкретно: IA32_SYSENTER_CS — содержит селектор целевого сегмента, IA32_SYSENTER_EIP — целевой адрес перехода, IA32_SYSENTER_ESP — новое значение регистра ESP при переходе на ядерный уровень, при этом селектор стека равняется (IA32_SYSENTER_CS + 08h).

SYSCALL работает практически аналогичным образом, только MSR регистры другие: STAR, LSTAR и CSTAR (подробнее об этом можно прочитать в описании самой команды SYSCALL в спецификации от AMD, ну или от Intel, с учетом того, что она поддерживает эту команду в той же манере, в какой AMD поддерживает SYSENTER).

Суть в том, что целостность MSR регистров долгое время никто не проверял, чем руткиты с успехом и воспользовались, изменяя MSR-регистры таким образом, чтобы управление получал не системный обработчик, а код руткита со всеми вытекающими отсюда последствиями. Далеко не все отладчики отображают содержание MSR регистров, но это легко осуществить с ядерного уровня командой RDMSR, которую руткит так же не может перехватить, а потому все его махинации с MSR регистрами будут немедленного разоблачены. Естественно, помимо проверки MSR-регистров (они должны указывать на тот же самый системный обработчик, что и в заведомо неинфицированной системе с той же самой версией ядра — только не спрашивайте у меня где ее взять!!!), мы так же должны проверить код самого обработчика, ибо он может изменен руткитом для перехвата управления без модификации MSR (впрочем, одно другому не мешает и многие руткиты используют гибридный вариант).

Поддержка SYSENTER/SYSCALL не отменяет INT 80h, по прежнему присутствующую в ядре и вызываемую из старых прикладных библиотек, некоторых ассемблерных программ, ну и, конечно, вирусов, работающих на прикладном уровне, так что руткитам теперь приходится перехватывать и то, и другое, хотя перехват SYSENTER/SYSCALL, конечно, намного более перспективен, т. к. INT 80h используется все реже и реже.

А вот разработчики Free-BSD от INT 80h отказываться пока не собираются и хотя существует патч от David'a Xu, написанный в конце 2002 года, и переводящий систему на SYSENTER/SYSCALL (см. http://people.freebsd.org/~davidxu/sysenter/), по умолчанию он не включен в стабильный релиз. Впрочем, сторонние составители дистрибутивов его активно используют (взять, к примеру, Dragon-Fly).

модификация таблицы системных вызовов

Указатели на системные вызовы перечислены в таблице sys_call_table, адрес которой можно найти все в том же System.map или вычислить эвристическим путем (так что удаление System.map'a не слишком-то усиливает безопасность).

Подмена указателя на оригинальный системный вызов указателем на код руткита — это классика перехвата, элементарно обнаруживаемая путем сравнения оригинальной таблицы системных вызовов, выдернутой из неупакованного ядра дизассемблером, с ее "живой" сестрицей, прочитать которую можно либо отладчиком, либо "руками", то есть командой mov, вызываемой из загружаемого модуля ядра. Оба способа _абсолютно_ ненадежны и выявляют только пионерские руткиты. "Зверюшки" посерьезнее сбрасывают страницы, принадлежащие

таблице системных вызовов, в NO_ACCESS, в результате чего при обращении к ним процессор выбрасывает исключение, подхватываемое руткитом, который смотрит откуда пришел вызов на чтение — если это функция system_call, то все ОК, если же нет, то руткит возвращает подложные данные, в результате чего таблица системных вызовов выглядит как девственная плева. Конечно, можно перед чтением проверить атрибуты страницы, но весь фокус в том, что функция определения атрибутов страниц реализована как системный вызов, находящийся в той же самой таблице, контролируемой руткитом. Опс! Приехали! Ладно, перед чтением мы назначим свой собственный обработчик исключений, который выручит нас только в том случае, если руткит не модифицировал IDT. Решение заключается в "ручном" разборе страничного каталога, формат которого описан в руководствах по системному программированию на процессоры Intel/AMD и представляет собой намного более простую задачу, чем это кажется поначалу, так что дерзайте.

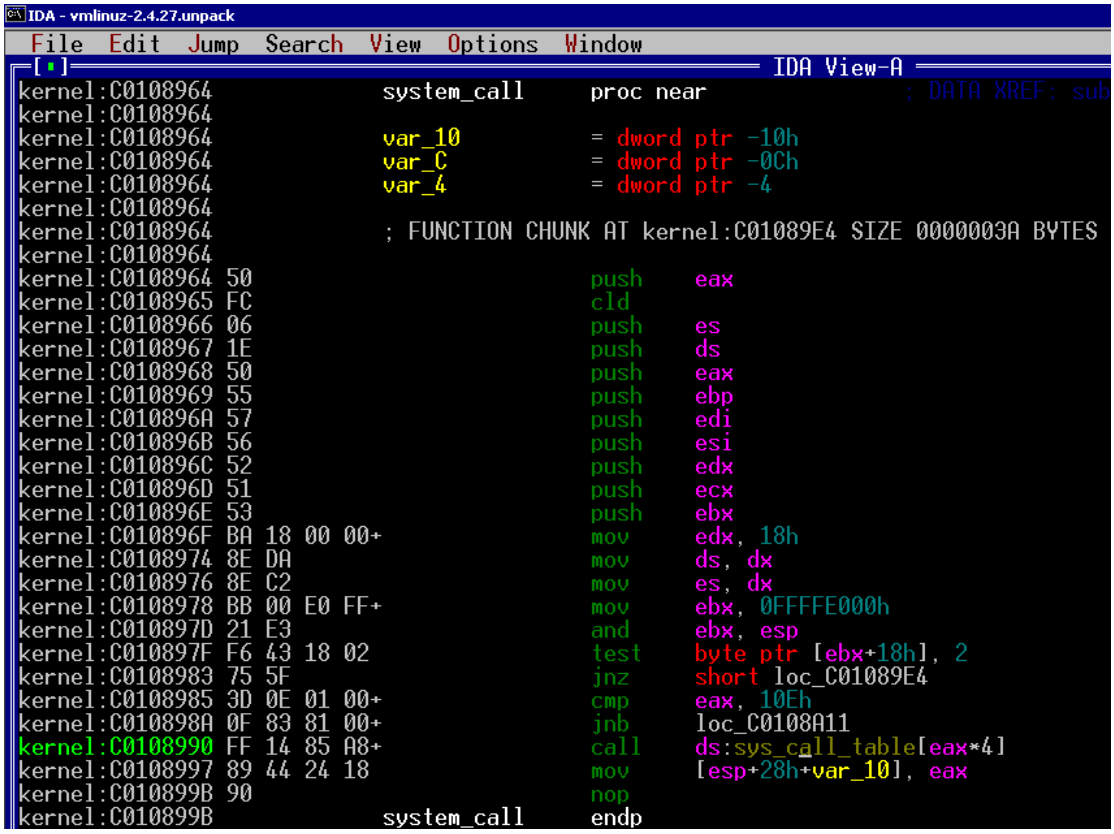

Рисунок 6 дизассемблерный листинг функции system_call, обращающейся к таблице системных вызовов sys_call_table

Естественно, кроме таблицы системных вызовов необходимо проверить и целостность самих системных вызовов, помня о том, что руткиты могут внедрять команду перехода на свое тело не только в начало функции системного вызова, но так же в ее конец или середину, хотя для этого им придется тащить за собой дизассемблер длин инструкций, тем не менее, "серединный перехват" — стандарт де-факто для всех "серьезных" руткитов.

дайте мне мыло, веревку или… дропер!

Сотрудники антивирусных компаний получают вирусы/руткиты из трех основных источников. Первый – свои собственные HoneyPot'ы, второй - малварь, присланная коллегами (другими антивирусными компаниями), третий (самый плодотворный) — файлы, полученные от пользователей.

В какой-то момент вирусописателям надоело, что их творения разносят в пух и прах в считанные дни, когда на создание руткита и его отладку уходят многие недели — обидно да? Вот они и стали искать пути как затруднить анализ малвари и ведь нашли! Очень просто — специальная программа, называемая дропером (от английского to drop – бросать), "сбрасывает" основное тело малвари на целевой компьютер, при этом основное тело малвари шифруется

ключом, сгенерированным на основе данных о конфигурации системы и наружу торчит только расшифровщик (зачастую полиморфный). Как нетрудно догадаться, малварь такого вида работает только на том компьютере, на который она попала "естественным" путем, а всякая попытка запуска ее на другой машине не дает ничего! То есть абсолютно! И чем длиннее ключ шифрования тем этот абсолют все абсолютнее!

Чтобы проанализировать малварь, необходимо вместе с ней получить данные о конфигурации, что довольно затруднительно, если не сказать, что в общем случае вообще невозможно. Так что остается либо вешаться, либо искать дропер, то есть ждать, пока малварь не влипнет в HoneyPot, принадлежащий реверсеру или одному из его партнеров.

заключение

Ну вот, начали за здравие, а кончили за упокой! А что в этом, собственно говоря, удивительного?! 90% руткитов — это полный отстой, написанный пионерами и обнаруживающий сам себя по нестабильному поведению системы. Остальные 10% — это что-то более или менее стоящее со сложностью обнаружения, варьирующейся от "элементарно" до "практически невозможно". И чтобы не отстать от прогресса, необходимо постоянно отлавливать свежих руткитов, анализируя их. Откуда взять столько времени — это уже другой вопрос.