ручной поиск руткитов в Linux/xBSD и NT

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

агрессивное развитие руткитов до сих пор остается безнаказанным и продолжает свое наступление не встречая никакого существенного сопротивления со стороны защитных технологий, большинство из которых хорошо работает только на словах и ловит общедоступных руткитов взятых с rootkits.com или аналогичных ресурсов. руткиты, написанные "под заказ", обнаруживаются значительно хуже, если вообще обнаруживаются, причем, даже такие продвинутые технологии детекции как удаленное сканирование портов оказываются бессильными перед новейшими версиями руткитов, которые реально обнаруживаться только руками, хвостом и головой (трава опционально прилагается)

введение

Мыщъх постоянно держит включенным honey-pot на базе VM Ware, засасывающий кучу малвари, анализ которой обнаруживает неуклонный рост количества руткитов, обитающих исключительно в памяти и не записывающей себя на диск, в результате чего у них отпадает необходимость в сокрытии файлов и ветвей реестра, прямо или косвенно ответственных за автозагрузку. Они не создают новых процессов, предпочитая внедряться в адресное пространство уже существующих. Они не открывают новых портов, перехватывая входящий трафик с помощью "сырых" сокетов или внедряются в сетевые драйвера (например, в TCPIP.SYS или NDIS.SYS).

В результате, ни в реестре, ни в файловой системе не происходит никаких изменений и потому ничего прятать не приходится! Естественно, перезагрузка убивает руткиты такого типа наповал и потому многие администраторы полагают, что никакой опасности нет. Не так уж сложно перезагрузить сервер при возникновении подозрений на его компрометацию. Однако, именно установка факта компрометации является первоочередной и самой сложной задачей, стоящей перед администратором! Если сервер действительно был скомпрометирован, то необходимо выяснить _как_ именно он был скомпрометирован! В противном случае повторные атаки не заставят себя ждать, не говоря уже о том, что после удаления малвари требуется как минимум изменить пароли на все ресурсы, иначе хакер сможет прожить и без руткита, читая корпоративную переписку, используя ранее перехваченный пароль на почтовый ящик (например).

Сам по себе руткит _обычно_ не предоставляет никакой угрозы и удаленный shell типа backdoor сейчас открывать уже не в моде. А вот завести новую учетную запись и добавить IP-адрес "своего" proxy-сервера в список доверенных адресов — не только проще, но и надежнее, поскольку в отличии от backdoor'a это не обнаруживается ни антивирусами, ни другими средствами защиты.

Таким образом, постановка задачи сводится к ответу на вопрос: утекли ли наши пароли на сторону или нет. К сожалению, в общем случае задача не имеет решения. Руткит, обитающий в оперативной памяти и существующий короткое время, обнаружить практически невозможно, тем более ручными методами.

Так что мы будем рассматривать лишь долгоживущие руткиты, которых на данный момент — большинство.

антивирусы и другие автоматизированные средства

Руткит, известный антивирусу, элементарно обнаруживается путем сканирования почтовых вложений или сетевых пакетов, однако, даже в этом случае у хакера имеется масса способов обломать рога антивирусу. Допустим, руткит забрасывается через дыру в браузере, некорректно обрабатывающего TIFF-файлы. Тогда атакующему остается всего лишь заманить жертву на ссылку вида https://www.xxxx.com, чтобы антивирус пропустил зашифрованные сетевые пакеты мимо своих ушей.

Что же касается поиска активных руткитов, то даже если они известны антивирусу, у них остаются все шансы уйти от возмездия, особенно если антивирус известен руткиту. Вот, например, существует такая интересная утилита как Rootkit Revealer (см. рис. 1) от Марка Руссиновича (http://www.microsoft.com/technet/sysinternals/Utilities/RootkitRevealer.mspx), обнаруживающая по утверждению его создателя _все_ руткиты, представленные на

www.rootkits.com, что не соответствует действительности, ибо тривиальная проверка выявляет большое количество малвари, отслеживающий запуск Rootkit Revealer'a и модифицирующей его код в памяти таким образом, чтобы он ничего не показывал. Естественно, подобная техника работает только со строго определенными версиями Rootkit Revealer'a (руткит должен знать _точное_ расположение машинных команд в памяти), а поскольку разработчикам малвари отслеживать выход новых версий ломы, они ограничиваются атакой типа WM_X, сводящейся к манипуляции элементами пользовательского интерфейса путем посылки соответствующих сообщений (Window Messages), удаляющих обнаруженные руткиты из списка, отображаемого Rootkit Revealer'ом, что работает со всеми версиями, но в лог-файл обнаруженные руткиты все-таки попадают (если, конечно, в него кто-то смотрит…).

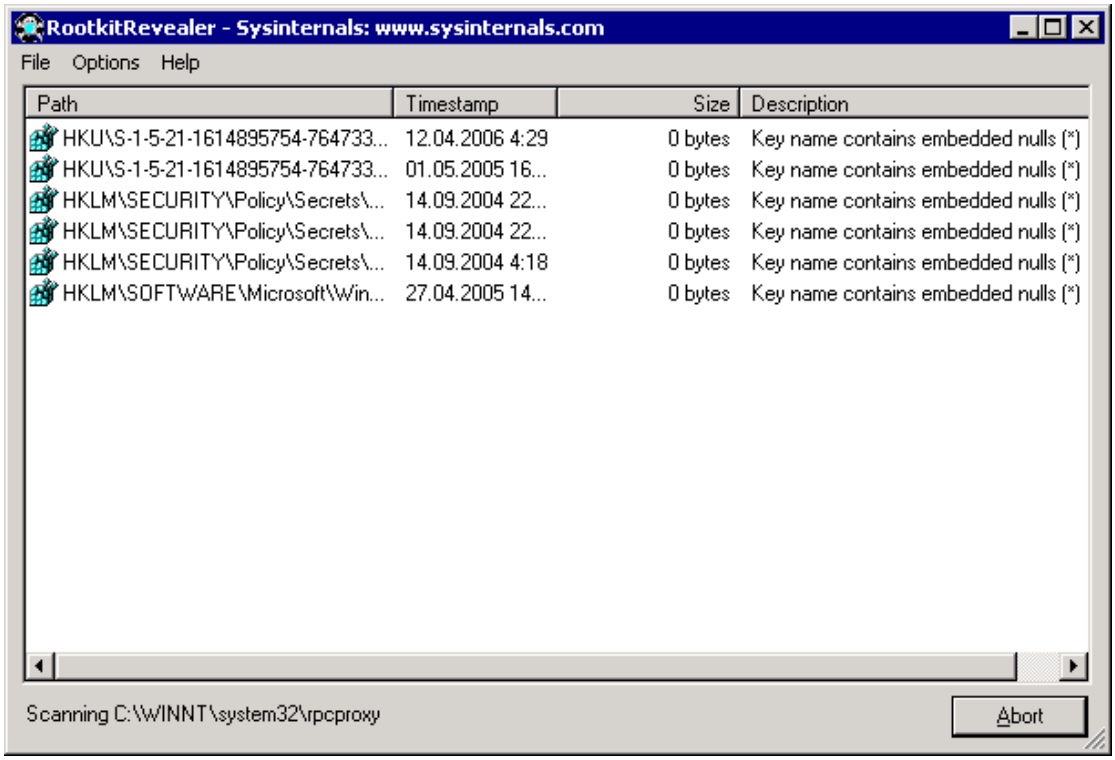

Рисунок 1 Rootkit Revealer за работой

К тому же Rootkit Revealer обнаруживает только те руткиты, которые: а) модифицируют реестр и/или файловую систему; б) скрывают следы своего присутствия. Если хотя бы одно из этих условий не выполняется — руткит не будет обнаружен. Анализ кода некоторых руткитов показывает, что они отслеживают появление окна Rootkit Revealer'a и прекращают свою маскировку на время его работы. Разработчикам защитных утилит уже давно пора взять полиморфизм на вооружение — до тех пор, пока они будут обнаруживаться руткитами, ни о какой защите не стоит и говорить! Антивирус не должен иметь постоянной сигнатуры (равно как и окон с заранее известными заголовками)!

Мир руткитов не ограничивается теми демонстрационными экземлярами, что выложены на www.rootkits.com. Судите сами: разработка качественного руткита — сложная инженерная задача и за один вечер такие руткиты не пишутся. Торговать руткитами (в силу их полулегального положения) отваживаются только самые нуждающиеся (или отчаявшиеся). Так какой же резон выкладывать руткит в общественный доступ? Разве, чтобы заявить о себе и посостязаться в крутости с другими хакерами. Но! Профессиональные программисты уже давно миновали стадию самоутверждения и вместо того, чтобы работать за идею, предпочитают кодить за деньги по индивидуальным заказам (по крайней мере, будет на что нанимать адвоката).

Реюз (то есть повторное использование кода) в таких руткитах практически не встречается и в антивирусные базы попадают лишь немногие из них. Как уже говорилось выше, правильно спланированная атака предполагает самоуничтожение руткита по истечении некоторого времени. Да и как его ловить, если он существует только в оперативной памяти?! Можно, конечно, передавать антивирусным компаниям дамп ядра операционной системы, но:

а) какая антивирусная компания будет в нем ковыряться?! б) это же сколько трафика займет! в) в дампе помимо руткита содержится до фига секретной информации, которую разглашать крайне нежелательно, например, пароли.

Важно понять, что в отличии от вирусов и червей, распространяющихся от компьютера к компьютеру и рано или поздно попадающих в антивирусные капканы, "настоящие" руткиты существуют в единичных экземплярах и потому обнаружить их могут лишь проактивные технологии, например, эвристический анализ, однако, если заказчик руткита хоть немного дружит с мозгами, он обязательно проверит: палится ли руткит последними версиями антивирусов на самом строгом режиме эвристики (на котором антивирус ругается даже на честные программы) и если да — возвратит его назад на доработку.

Поэтому, в качестве рабочего тезиса необходимо принять, что руткиты антивирусами _не_ обнаруживаются, как бы нам ни промывали мозги их создатели.

удаленное сканирование портов

В эпоху расцвета backdoor'ов удаленное сканирование портов считалось абсолютно надежным методом обнаружения рутикитов. Действительно, как бы не маскировал сетевые соединения руткит и какие бы системные вызовы он не перехватывал, все это воздействует лишь на _локальную_ машину. Да, конечно, можно обдурить и tcpdump, и netstat, но… только _локально_. Всякая же попытка сканирования зараженного компьютера с соседней машины немедленно выявит открытые порты, если они, разумеется, там есть. И руткит _никак_ не может этому противостоять.

Однако, зачем маскировать факт открытия портов, если никакие порты вообще можно не открывать, а использовать уже открытые? Мыщъх исследовал несколько руткитов, которые путем перехвата системных функций мониторили HTTP-трафик и передавали на хакерский узел через 80й порт информацию о текущем номере последовательности TCP/IP-соединения, чтобы хакер мог послать "левый" пакет (с командами для руткита), который бы воспринимался системной как правильный, а что бы соединение с текущим WEB-узлом не разрывалось, хакер посылал ему еще один пакет, предотвращающий срыв "синхронизации" номера последовательности.

Другими словами, руткит передавал/принимал данные в контексте существующего TCP/IP соединения, инициированного компьютером-жертвой и потому сканирование портов ничего подозрительно не выявляло, а вот внимательный анализ TCP/IP пакетов показывал, что пакеты, переданные руткитом хакеру, имели IP-адрес отличный от IP-адреса целевого узла, с которым и было установлено соединение.

Однако, не стоит обольщаться — не все руткиты такие простые и при желании трафик можно спрятать так, что его никто и никогда не найдет. На сайте Жанны Рутковской (http://www.invisiblethings.org/tools.html) выложены готовые утилиты, прячущие сам факт присутствия постороннего трафика, да еще и шифрующего его алгоритмом RSA (см. рис. 2), благодаря которому разбор логов tcpdump'а становится пустой тратой времени.

Выдвигаем следующий тезис: руткиты не открывают новых портов, а генерируемый ими трафик ни локальными, ни удаленными снифферами не обнаруживается.

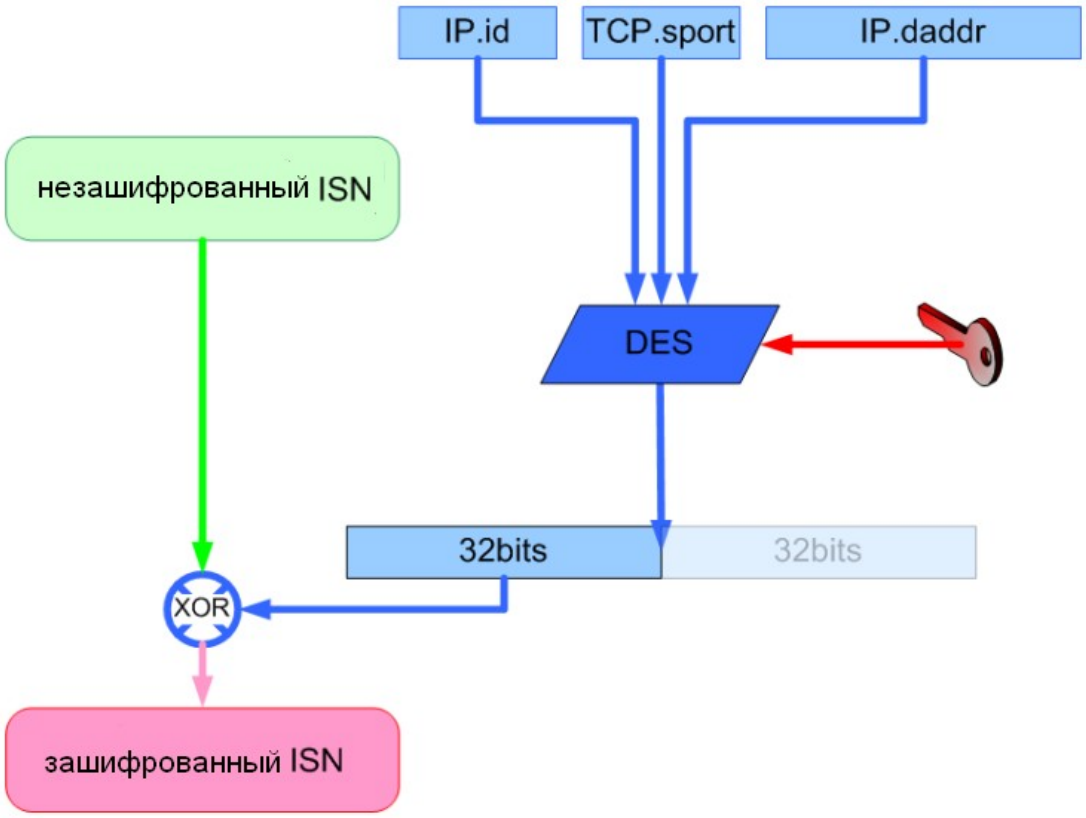

Рисунок 2 стелсирование и шифрование трафика

свет в конце тоннеля или встречный?

Извечный вопрос: как быть, что делать?! Побороть новые руткиты старыми средствами уже не удается, а новых средств — за отсутствием таковых. Поэтому, приходится возвращаться к скомпрометированной машине и искать руткит непосредственно на ней. Весь вопрос в том — как найти произвольный руткит, если о нем заранее ничего не известно? Ни сигнатур, ни других опознавательных признаков у нас нет.

К счастью, существует не так уж много методов перехвата системных функций и все они оставляют за собой вполне осязаемые следы, обнаружить которые можно даже без глубоких знаний особенностей реализации операционной системы и не будучи знатоком ассемблера. Естественно, никаких гарантий у нас нет, но все-таки ручной поиск намного надежнее автоматизированного, пускай он и требует определенной квалификации. Запустить антивирус может и домохозяйка, что нивелирует разницу между ней и опытным хакером, так что лучше развивать свои собственные способности, тренировать "нюх", чем доверять безопасность компьютера чужим дядям.

Руткиты принято классифицировать по двум основным критериям — месту обитания (ядро или прикладной уровень) и способу внедрения (перехват функций путем битхака или правда базовых структур данных операционной системы). Такая классификация очень условна и в реальной жизни сплошь и рядом встречаются гибридные варианты, одновременно работающие как на уровне ядра, так и на прикладном уровне. Забавно, но руткиты, полностью работающие на прикладном уровне, обнаружить сложнее всего, поскольку им доступно огромное количество методик внедрения в чужие процессы, а для манипуляций с трафиком никаких функций вообще перехватывать не нужно — достаточно воспользоваться "сырыми" сокетами. Но руткиты прикладного уровня это "не круто" и вообще не по "понятиям". Взор хакеров устремлен в ядро, в котором можно делать все, что угодно. Вот только методик перехвата системных функций там раз два и обчелся, а потому обнаружение ядерных руткитов представляет собой довольно простую задачу.

Мы будем говорить именно о руткитах уровня ядра, семейство которых делится на два подтипа: одни внедряются путем правки машинного кода, внедряя в начало (редко — в середину) функции команду jmp или call для перехода на свое тело, а другие — модифицируют

структуры данных, например, таблицу системных вызовов, хранящую указатели на функции. В NT оба подтипа руткитов встречаются приблизительно с одинаковой частотой, а в Linux/xBSD в основном преобладает второй подтип, что связано с тем фактом, что ядро NT экспортирует native-API функции как обычная динамическая библиотека (DLL), а Linux/BSD – чтобы найти native-API функции следует очень постараться, да только зачем стараться, если таблица системных вызовов у нас под рукой?!

Существует множество утилит проверяющих целостность таблицы системных вызовов и восстанавливающую ее в случае необходимости, но мне не известна ни одна утилита, проверяющая целостность самих системных функций, внедрение в которых существенно усиливает жизнестойкость руткита (механизм PatchGuard, реализованный в x86-64 версиях NT, мы не рассматриваем, поскольку его очень легко обойти).

Собственно говоря, при всем различии NT и Linux/BSD техника поиска руткитов одна и та же. Первым делом нам необходимо заполучить дамп ядра или запустить ядерный отладчик. Теоретически, руткиты могут перехватывать любые операции, в том числе и попытку сохранения дампа. В NT для этого им достаточно перехватить native-API функцию KeBugCheckEx и прежде, чем возвратить ей управление, вычистить все следы своего пребывания в оперативной памяти. Технически реализовать это несложно. Понадобиться не больше пары сотен строк ассемблерного кода, но… мне не известен ни один руткит, реально делающий это. Так же, можно обхитрить и ядерный отладчик — устанавливаем всем хакнутым страницам атрибут только на исполнение (если ЦП поддерживает бит NX/XD) или ставим страницу в NO_ACCESS, а при возникновении исключения смотрим — пытаются ли нас прочесть или исполнить и, если нас читают, то это явно отладчик, для обмана которого временно снимаем перехват. Но это всего лишь теория. Практически она еще никем не реализована и когда будет реализована — не известно.

Увы, абсолютно надежный способов детекции руткитов не существует и на любую меру есть своя контрмера. Но не будем теоретизировать, а вернемся к _реально_ существующим руткитам, а точнее, к получению дампа памяти. В NT в "свойствах системы" (<Win+Pause>) необходимо выбрать "полный дамп" (см. рис. 3), затем запустить "Редактор Реестра", открыть ветвь HKLM\System\CurrentControlSet\Services\i8042prt\Parameters и установить параметр CrashOnCtrlScroll (типа REG_DWORD) в любое ненулевое значение, после чего нажатие на CTRL с последующим двойным нажатием PAUSE будет вызвать голубой экран смерти с кодом E2h (MANUALLY_INITIATED_CRASH). К сожалению, чтобы изменения реестра вступили в силу, необходимо перезагрузить машину, прибив при этом руткит, который мы пытаемся найти, так что эту операцию следует осуществлять заблаговременно.

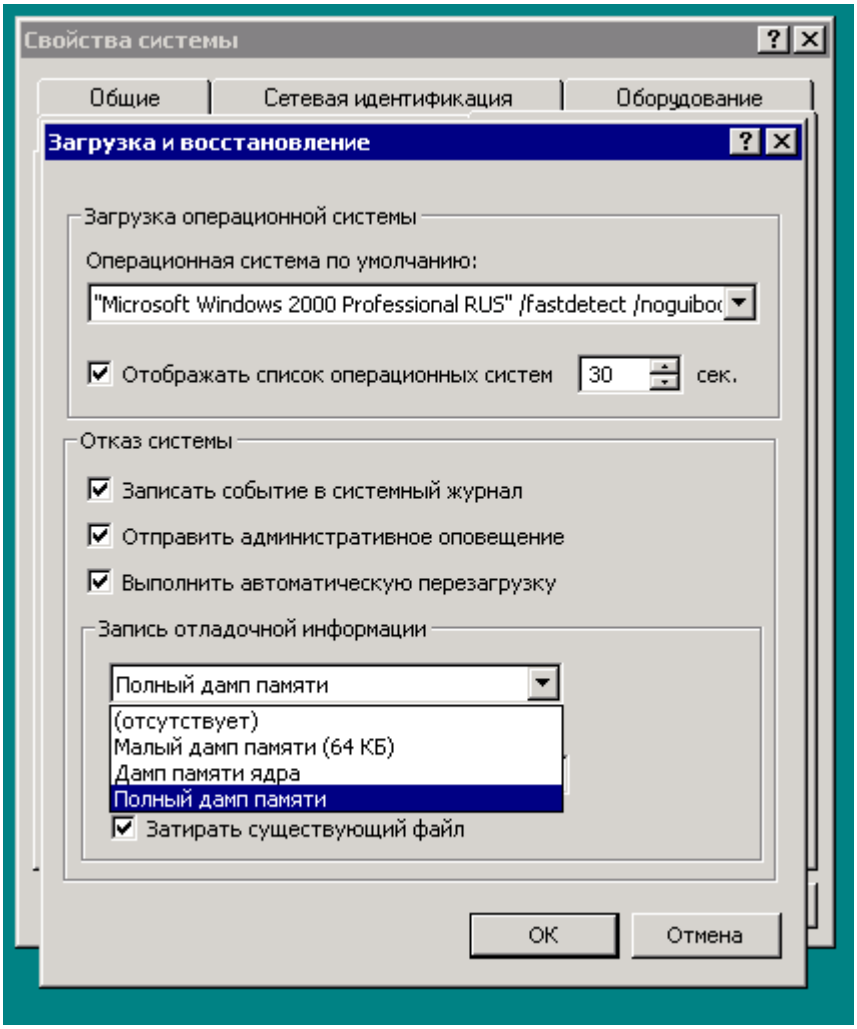

Рисунок 3 настройка системы на сохранение полного дампа памяти ядра

Кстати говоря, последовательность <CTRL+SCROLL LOCK+SCROLL LOCK> срабатывает даже тогда, когда машина ушла в нирвану и уже не реагирует на <ALT-CTRL-DEL>, причем, в отличии от RESET, комбинация <CTRL+SCROLL LOCK+SCROLL LOCK> выполняет сброс дисковых буферов, что уменьшает риск потери данных, так что настроить CrashOnCtrlScroll стоит даже в том случае, если мы не собираемся охотиться на руткиты.

В тех случаях, когда CrashOnCtrlScroll не настроен, а перезагрузка не приемлема, можно взять любой драйвер из NTDDK и вставить в начало DriverEnrty какую-нибудь недопустимую операцию — деление на нуль, обращение к памяти по нулевому указателю и т. д. Тогда при загрузке драйвера немедленно вспыхнет голубой экран, а на диск будет сброшен полный дамп памяти ядра со всей малварью в нем содержащейся.

В Linux'e (для ядер версий 2.2 и старше) ручной сброс дампа осуществляется при нажатии на <ALT-SysRq-C> (при этом, ядро должно быть откомпилировано с параметром CONFIG_MAGIC_SYSRQ равным "yes" или выполнена следующая команда "echo 1 > /proc/sys/kernel/sysrq").

В xBSD-системах комбинация <CTRL-ALT-ESC> (кстати говоря, измененная в некоторых раскладках клавиатуры) вызывает всплытие ядерного отладчика (аналог <CTRL-D> для soft-ice в NT), который, к сожалению, по умолчанию не входит в ядро и потому его необходимо предварительно перекомпилировать добавив строки "options DDB" и "options BREAK_TO_DEBUGGER" в файл конфигурации ядра. Если же последняя опция не обозначена (о ней часто забывают), то в отладчик войти можно и с консоли командой "sysctl debug.enter_debugger=ddb".

Полученный дамп ядра можно анализировать любой сподручной утилитой, благо недостатка в них ощущать не приходится. Например, в NT для этой цели обычно используется WinDbg, но мыщъх предпочитает исследовать систему в живую с помощью Soft-Ice, ближайшим аналогом которого в мире Linux является Lin-Ice.

Значит, нажимаем мы <CTRL-D> (Soft-Ice), <CTRL-Q> (Lin-Ice) или <CTRL-ALT-ESC> (xBSD) и оказываемся в ядре. Далее пишем "u имя_функции" и последовательно перебираем имена всех функций (ну, или не всех, а самых "соблазнительных" для перехвата), список которых под NT можно получить командой "DUMPBIN .EXE ntoskrnl.exe /EXPORT > output.txt" (где DEUMPBIN.EXE – утилита, входящая в состав Microsoft Visual Studio и Platform SDK), а под Linux/xBSD эту же задачу можно решить, изучив символьную информацию несжатого и нестрипутного ядра.

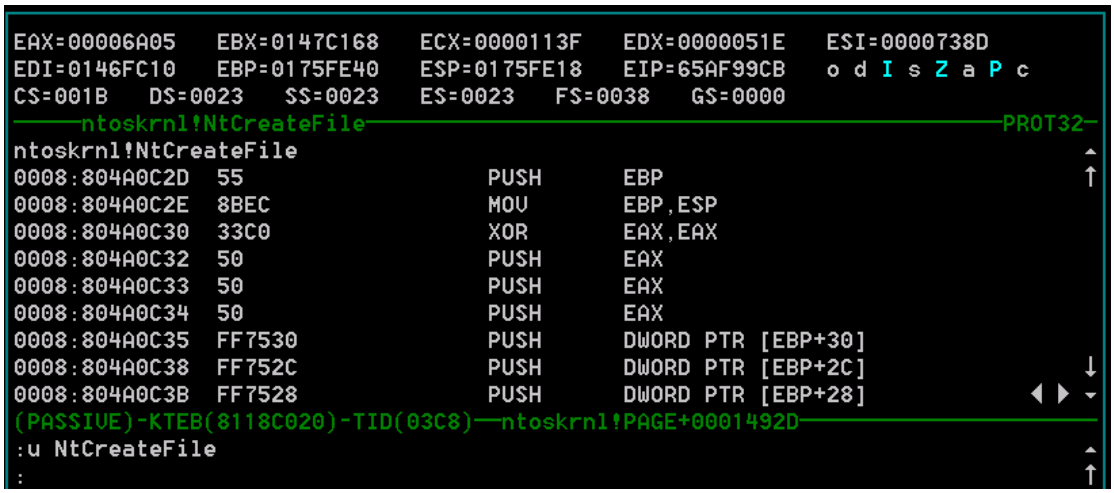

Рисунок 4 внешний вид не перехваченной функции

В начале нормальных не перехваченных функций должен находится стандартный пролог вида PUSH EBP/MOV EBP, ESP или что-то типа того (см. рис. 4), если же туда воткнут JMP или CALL (см. рис. 5), то с вероятностью близкой к единице данная функция кем-то перехвачена. А вот кем — это воооопрос. Кроме руткитов перехватом занимаются антивирусы, брандмауэры и другие программы, поэтому, прежде чем отправляться на поиск малвари, необходимо хорошо изучить особенности своей системы со всеми установленными приложениями.

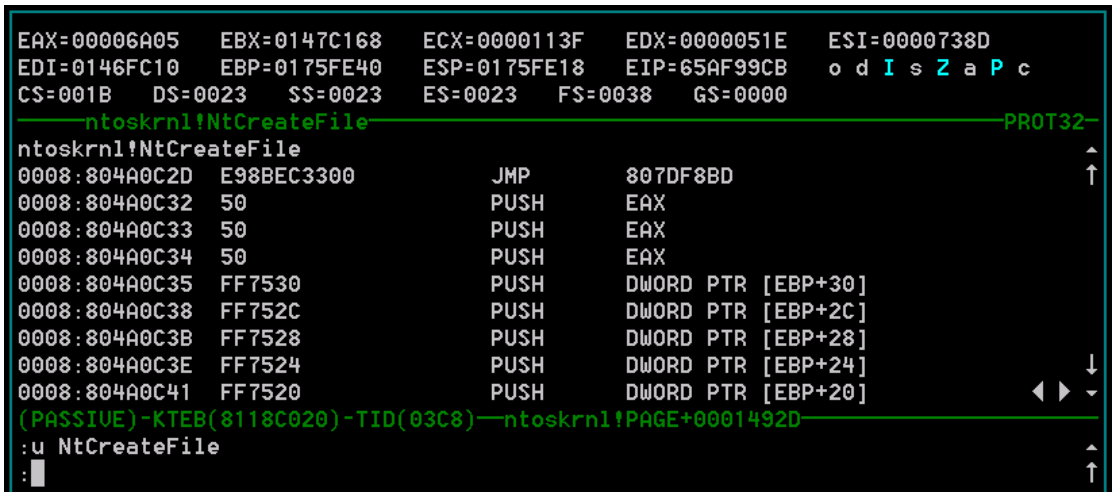

Рисунок 5 внешний вид перехваченной функции

Продвинутые руткиты внедряют JMP/CALL не в начало функции, а в ее середину, маскируясь под нормальное поведение. На самом деле, проанализировав код хакнутой функции, легко убедиться в его "нелогичности" и "ненормальности". Левый JMP/CALL просто не вписывается в алгоритм! Однако, чтобы прийти к подобному заключению необходимо не только знать ассемблер, но и иметь опыт дизассемблирования. К счастью, продвинутые руткиты встречаются достаточно редко и подавляющее большинство из них внедряется в самое начало.

Просмотрев все функции и убедившись в отсутствии следов явного перехвата, приступаем к просмотру таблицы системных функций, которая под Soft-Ice вызывается

командой NTCALL, и "D sys_call_table" в Lin-Ice. Поскольку, функции, перечисленные в таблице, не экспортируются ядром NT, то, в отсутствии символьной информации (которую можно получить с сервера Microsoft с помощью утилиты SymbolRetriver от NuMega), Soft-Ice отображает имя _ближайшей_ экспортируемой функции + смещение (см. рис. 6), а потому мы не можем быстро сказать: перехвачена данная функция или нет и нам приходится набирать команду "u адрес_функции", чтобы посмотреть что там находится: нормальный не перехваченный пролог или JMP/CALL. В никсах информация о символах присутствует по умолчанию и подобных проблем не возникает.

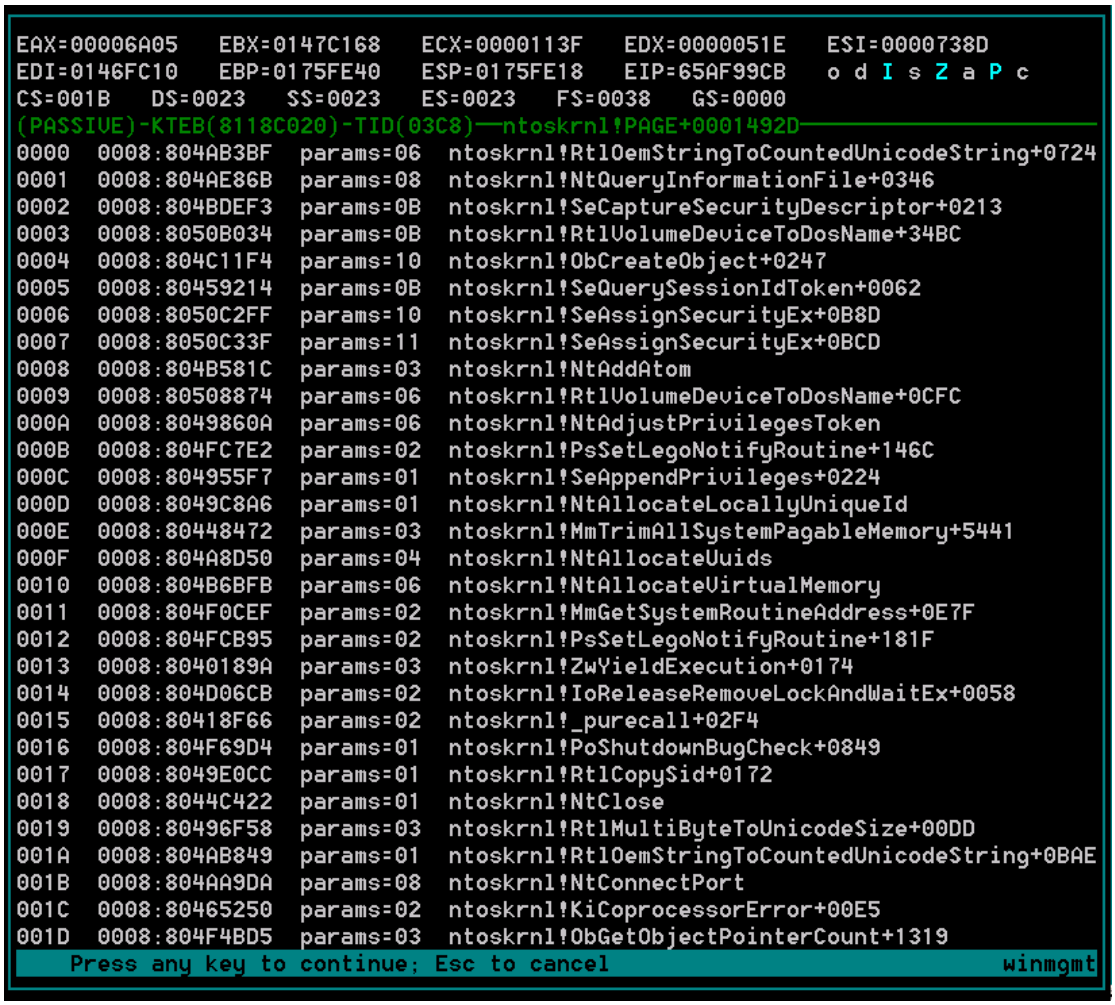

Рисунок 6 таблица системных вызовов

Естественно, помимо описанных, существуют и другие методики перехвата, используемые руткитами, однако, они слишком сложны для понимания и требуют предварительной подготовки, а потому здесь не рассматриваются.

заключение

Немецкий философ Карл Рэймонд Поппер (Karl Raimund Popper) сформулировал критерий фальсифицируемости, позволяющий отсеивать ненаучные теории и вошедший в методологию современной науки. Теория считается научной, если существует возможность постановки эксперимента, при котором она "откажет". Напротив, теория, работающая во всех мыслимых и не мыслимых случаях — ненаучная, какой бы соблазнительной она не выглядела.

Применительно к малвари — методы поиска руткитов, претендующие на универсальность, согласно критерию Поппера, ненаучны и по ходу дела отправляются в \dev\nul. У каждого метода есть свои ограничения и к нему в обязательном порядке должна прилагаться "сопроводиловка", рассматривающая ситуации (неважно — реальные или гипотетические) в который данный метод отказывает и становится несостоятельным.

Вместе с технологией поиска руткитов мыщъх описывает и методы ее преодоления, о которых нельзя забывать, если мы действительно хотим найти малварь, а не просто сымитировать бурную деятельность. Увы, действительно универсальных методик не существует, ибо, если методика работает во _всех_ случаях и не существует никаких путей ее обхода, то это не методика, а туфта, свидетельствуя об ошибках в наших рассуждениях или о незнании реальных возможностей руткитов.