взлом patch-guard

крис касперски, ака мыщъх, no-email

в 64-битных версиях Windows XP/Vista, Server 2003/Longhorn появился широко разрекламированный механизм Patch-Guard, контролирующий целостность системы и призванный уберечь пользователей от rootkit'ов и некорректно работающих программ, вламывающихся в ядро словно слон в посудную лавку. насколько надежна такая защита и можно ли ее обойти?

введение

Ядро операционной системы — это фундамент, на который опираются все остальные компоненты программные (драйвера, службы, приложения). В многозадачных (и тем более — многопользовательских) средах крайне важно изолировать один компонент от другого так, чтобы он случайно или предумышленно не мог ему навредить.

Фундамент Windows NT построен по гибридной схеме — ***монолитное ядро*** плюс ***загружаемые модули*** (в свойственной им терминологии называемые *драйверами*). NT поддерживает два типа драйверов: подключаемые на стадии загрузки операционной системы и загружаемые на лету, что создает проблемы устойчивости и безопасности.

Все драйвера работают в едином с ядром адресном пространстве на том же самом уровне привилегий, что и оно, "благодаря" чему драйвера могут свободно вмешиваться в работу ядра, модифицируя его по своему усмотрению. А ведь x86-процессоры предоставляют целых четыре кольца защиты (из которых NT использует только два) и технически ничего не стоит изолировать ядро от драйверов, запуская их в менее привилегированном кольце. Кстати говоря, в висте наконец-то появились "драйвера", работающие на прикладном уровне и обслуживающие запросы от устройств, поставляемые драйверами нижнего уровня (например, драйвер USB-радио). Крах высокоуровневого драйвера уже не приводит к краху всей системы, однако, сам по себе драйвер становится чрезвычайно уязвим со стороны прикладных приложений, плюс взаимодействие с драйверами нижнего уровня требует частых переходов в режим ядра (с последующими возвращениями оттуда обратно), что отнюдь не способствует производительности. С передачей больших объемом данных также возникают проблемы. Если внутри ядра они могут легко передаются по ссылке, то межуровневая передача должна осуществляться только по значению, то есть путем копирования через промежуточный буфер, многократно увеличивающий требования к системным ресурсам, особенно при работе с быстрыми устройствами.

Ну и ради чего это все?

проблемы гибридных ядер

Разработчики монолитных ядер стремятся включить в них все драйвера, которые только могут понадобится конечному пользователю, что увеличивает их размер, однако, обеспечивает наивысший уровень стабильности и безопасности. По статистике, основным источником "голубых экранов смерти" является отнюдь не сама Windows, а драйвера, разработанные "пионерами", после плановой раскурки местной подзаборной. Хуже всего, что некоторые из них загружаются без ведома пользователя и не имеют никаких средств для деинсталляции. С другой стороны: монолитное ядро без возможности загрузки драйверов — никому, за исключением серверов, не нужно. Сервера работают на вполне предсказуемом железе, им не нужны ни навороченные звуковые карты ни сверхбыстрое видео.

Рабочие станции – другое дело. Новое железо появляется чуть ли не ежедневно и по сложившейся традиции вместе с ним идет драйвер, разработанный производителем, нанявших двух голодных китайских студентов ваяющих драйвер параллельно с изучением DDK. Вот так и появляются драйвера, запускающиеся лишь на машинах их создателей и вместо использования документированных интерфейсов, за каким-то хреном лезущих в глубь ядра. Последствия такого подхода известны — нестабильность, плохая совместимость с различными версиями NT, бесконечные голубые экраны смерти…

Но не стоит валить в одну кучу "пионерство" и системное программирование. Ядро экспортирует множество функций (как документированных, так и нет), но самые интересные оставляет внутри себя, не представляя к ним никаких рычагов управления. Вот и приходится вламываться в ядро, нарушая все запреты.

Отдельного разговора заслуживают rootkit'ы, модифицирующие ядро в целях своей маскировки. Для этого они перехватывают функции, работающие с файлами, процессами, сетевыми соединениями и "вычищают" всякое упоминание о себе. Какому пользователю понравится, что на его машине работает нечто, о чем он даже не подозревает и делает вещи, в которых он не нуждается? (Например, устанавливает backdoor или ворует конфиденциальную информацию).

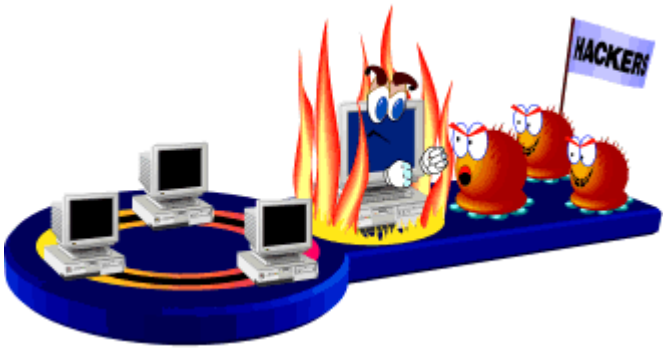

Рисунок 1 гибридные ядра — великолепный объект для атаки

атака на ядро в x86 системах

Попытки защитить ядро предпринимались задолго до выхода висты. Начиная с W2K Microsoft предприняла первый радикальный шаг для защиты ядра от зловредных драйверов, так и норовящих модифицировать кое-что. Однако, для сохранения обратной совместимости с уже написанными программами была предусмотрена лазейка, отключающая защиту, а точнее целых пять:

I. установка параметра EnforceWriteProtection типа DWORD ветки системного реестра HKLM\SYSTEM\CurrentControlSet\Control\SessionManager\MemoryManagement в "0" (изменения вступят в силу только после перезагрузки);
II. сброс флага WP (WriteProtect) в управляющем регистре CR0 (см. листинг 1, 2), — не требует перезагрузки, но доступно только из режима ядра;
III. репаминг страниц — отображаем физический адрес страницы, которую мы хотим модифицировать, на виртуальное адресное пространство "своего" процесса посредством вызова функции NtMapViewOfSection. назначаем все необходимые права и атрибуты, после чего делаем с ней все хотим! таким образом, можно открыть доступ к ядру даже с прикладного уровня, причем _без_ перезагрузки!
IV. запись в псевдоустройство PhysicalMemory, представляющее собой образ физической памяти до трансляции виртуальных адресов и позволяющее модифицировать память ядра даже с прикладного уровня обладая всего лишь правами администратора и без перезагрузки (см. листинг 3);
V. модификация файла NTOSKNRL.EXE на диске — самый уродливый способ из всех, требующий обхода SFC, коррекции контрольной суммы EXE-файла, перезагрузки и к тому же вызывающий серьезные конфликты при установке Service Pack'ов.

```
mov     eax, cr0        ; грузим управляющий регистр cr0 в регистр eax
and     eax, 0FFFEFFFFh; сбрасываем бит WP, запрещающий запись
mov     cr0, eax        ; обновляем управляющий регистр cr0
```

Листинг 1 код, отключающий защиту ядра от записи

Соответственно, чтобы включить защиту, бит WP нужно установить, что и делают следующие машинные команды:

```
mov     eax, cr0        ; грузим управляющий регистр cr0 в регистр eax
or      eax, 10000h     ; сбрасываем бит WP, запрещающий запись
mov     cr0, eax        ; обновляем управляющий регистр cr0
```

Листинг 2 код, включающий защиту ядра

"Политически корректная" программа должна не просто отключать/включать защиту от записи, а запоминать текущее состояние бита WP перед его изменением, а затем восстанавливать его обратно "как было", иначе можно непроизвольно включить защиту в самый неподходящий момент, серьезно навредив, вирусу или rootlit'у.

Запись в физическую память осуществляется сложнее (к тому же, необходимо предварительно найти код самого ядра, что можно сделать, например, поиском сигнатуры модифицируемой функции в PhysicalMemory, при этом сама функция должна присутствовать в памяти, а не валяться в страничном файле, т. е. прежде чем модифицировать функцию ее необходимо вызывать хотя бы с фиктивными аргументами):

```
// разные переменные
NTSTATUS ntS; HANDLE Section; OBJECT_ATTRIBUTES ObAttributes;
INIT_UNICODE(ObString, L"\\Device\\PhysicalMemory");

// инициализация атрибутов
InitializeObjectAttributes(&ObAttributes, &ObString,
                           OBJ_CASE_INSENSITIVE | OBJ_KERNEL_HANDLE, NULL, NULL);

// открываем секцию PhysicalMemory
ntS = NtOpenSection(&Section, SECTION_MAP_READ|SECTION_MAP_WRITE, &ObAttributes);
```

Листинг 3 открытие псевдоустройства PhysicalMemory

В частности, soft-ice работает использует первый способ, большинство антивирусов, брандмауэров и rootkit'ов — второй и третий. Четвертый способ в основном встречается в rootkit'ах (и программах, предназначенных для борьбы с ними, типа SDTRestore). Пятый способ обычно используется для превращения 180-дневной версии Windows в "лицензионную".

Первое наступление на rootkit'ы Microsoft предприняла в Windows 2003 Server SP1, закрыв доступ к PhysicalMemory не только администратору, но даже приложениям с привилегиями SYSTEM! Следующий шаг, предпринятый уже в висте — защита ядра цифровой подписью, предотвращающей его модификацию на диске. Во всяком случае теоретически. На практике же, хакеры модифицируют ядро _вместе_ с механизмом проверки цифровой подписи, отламывая последний за ненадобностью.

Предпринятые меры не слишком-то усилили защищенность системы, да и не могли ее усилить, поскольку, закрытие всех лазеек привело бы к неработоспособности огромного количества легальных программ, на что Microsoft пойти не могла, поэтому даже 32-битная редакция висты по-прежнему остается незащищенной.

атака на ядро в 64-битных системах

Воспользовавшись появлением новых процессорных архитектур x86-64 (AMD) и IA64 (Intel), Microsoft перенесла на них свои системы: XP, висту, Server 2003 и Server Longhorn, провозгласив новую политику модификации ядра — то есть, никакой модификации. Действуйте только через легальные средства или до свидания! В добавок к этому, Microsoft заблокировала загрузку драйверов без цифровой подписи, пообещав, что никакой неавторизованный код не сможет проникнуть на уровень ядра. Типа, никаких червей и rootkit'ов отныне не будет, спите спокойно! (Более подробно об этом рассказывают следующие официальные документы: http://download.microsoft.com/download/c/2/9/c2935f83-1a10-4e4a-a137-c1db829637f5/windowsvistasecuritywp.doc, http://download.microsoft.com/download/9/c/5/9c5b2167-8017-4bae-9fde-d599bac8184a/KMCS_Walkthrough.doc, http://download.microsoft.com/download/9/c/5/9c5b2167-8017-4bae-9fde-d599bac8184a/kernel-en.doc и др.).

Какая трогательная забота о пользователях! И какое пренебрежительное отношение к разработчикам! Microsoft делает вид, что проблемы обратной совместимости для 64-битных систем не существует в природе, как не существует готовых программ для них. Это неправда. Большинство программ (в том числе и драйверов!) переносятся путем простой перекомпиляции с легкой ретушью, учитывающей специфику 64-битных архитектур.

Естественно, чем глубже вгрызается драйвер в 32-битное ядро, тем сложнее его отдирать, так что переход на 64-битные системы затянется надолго, а некоторые системные утилиты придется переписывать с чистого листа. Самое обидное, что ни вирусов, ни rootkit'ов это никак не коснется. Механизм обхода цифровой подписи путем модификации файла подкачки на секторном уровне был предложен Жанной Рутковской еще до появления висты на прилавках и заткнуть его Microsoft не в силах. Но даже если ей это удастся, в системе полно драйверов от сторонних разработчиков, позволяющих атакующему делать с ядром все что угодно, плюс дыры самой системы. Наконец, ничего не мешает написать драйвер,

отключающий проверку цифровой подписи, и подписать его, воспользовавшись поддельным удостоверениями личности. Конечно, это большой геморрой и вообще незаконно, но мы же не о законности говорим, а рассматриваем степень (не)защищенности.

Осознавая все это, Microsoft внедрила в свои 64-битные системы (XP, виста, Server 2003 SP1, Server Longhorn) специальный механизм, контролирующий целость ядра и титулованный гордым званием Patch-Guard. Стражник, значит. Между прочим, не претерпевшим никаких значительных изменений со времен Server 2003 S1, где он, впервые, собственно говоря, и появился. Как говорил пра-пра-пра-пра-дедушка Билла Гейтса "полковник Кольт уравнял людей в правах" и Patch-Guard, обладающий теми же самыми привилегиями, что и зловредные драйвера, имеет все шансы быть пропатченным прежде, чем успеет сказать "мяу".

Подробное описание внутреннего устройства Patch-Guard'a можно найти в статье "Bypassing PatchGuard on Windows x64" от skape и Skywing (на самом деле, в операции по трепанации этой твари было задействовано гораздо больше людей, упомянутых в статье): http://uninformed.org/index.cgi?v=3&a=3&t=sumry. Так же не помешает ознакомится с официальными документами от самой Microsoft: www.microsoft.com/whdc/driver/kernel/64bitpatching.mspx и www.microsoft.com/whdc/driver/kernel/64bitpatch_FAQ.mspx.

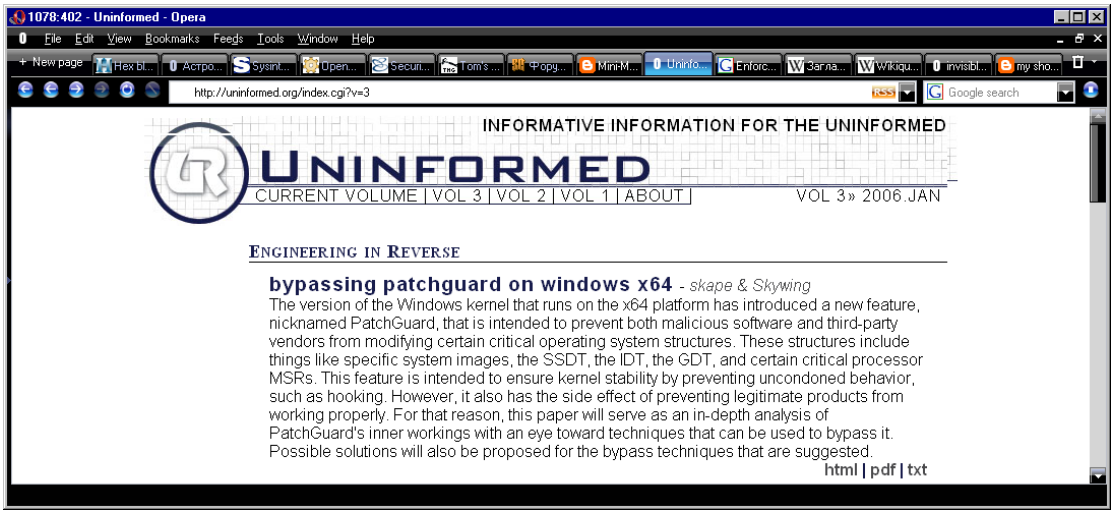

Рисунок 2 они первыми взлома Patch-Guard

Patch-Guard инициализируются _только_ в отсутствии системного отладчика, подключающегося при загрузке (или его имитации в виде "затычки"), в противном случае отладчик не смог бы устанавливать программные точки останова (представляющие собой однобайтовую машинную команду CCh) и делать массу других жизненно важных вещей.

Будучи инициализированным, Patch-Guard в случайные промежутки времени (приблизительно один раз в 5-10 минут) запускает процедуру проверки целостности ядра, наводящую в системе настоящий шмон. Текущие версии создают копии всех критических структур данных и подсчитывают контрольные суммы NTOSKRNL.EXE (ядро), HAL.DLL (библиотека абстрагирования от оборудования) и NDIS.SYS (один из самых низкоуровневых сетевых драйверов). Почему Patch-Guard работает по принципу обходчика a-la "в Багдаде все спокойно", вместо того, чтобы пресекать всякую попытку модификации в момент ее появления? Некоторые утверждают, что во всем виноват процессор, не предоставляющих таких возможностей. Ну, на самом деле, процессор предоставляет, только парни из Рэйдмонда воспользоваться этим не умеют, во всяком не в той мере, в какой это необходимо.

Полный список контролируемых элементов приведен ниже (естественно, в последующих версиях висты он может измениться):

- таблица глобальных дескрипторов —GDT;
- таблица дескрипторов прерываний — IDT;
- стек ядра, выделенный не ядром, а кем-то еще;
- таблица дескрипторов системных сервисов — SSDT;
- образы следующих системных файлов: NTOSKRNL.EXE, NDIS.SYS, HAL.DLL;
- служебные MSR регистры STAR/LSTAR/CSTAR/SFMASK, отвечающие за syscall'ы;
- [AMD x86-64 предоставляет возможность контроля за образом ядра без расчета CRC]

Контроль целостности системных файлов уже не позволяет перехватывать функции путем внедрения в них начало jump'a на хакерский обработчик, а слежение за SSDT препятствует подмене адресов сервисных функций на хакерские переходники к ним. Так же хакер не может вмешиваться в работу менеджера исключений или воздействовать на внутренние аргументы команды SYSCALL, хранящиеся в MSR-регистрах.

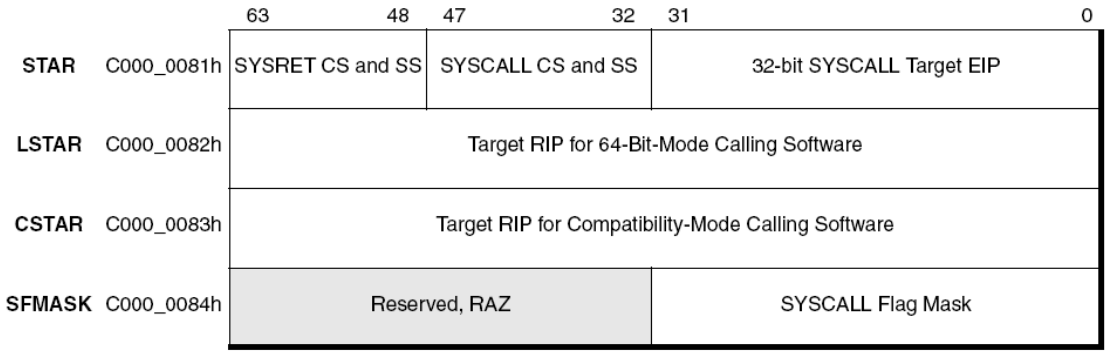

Рисунок 3 MSR регистры, обеспечивающие работу машинной команды syscall на x86-64

Вот с таким противникам нам придется сразится. Но так ли он страшен, как кажется?! Чтобы не пересказывать своими словами статью "Bypassing PatchGuard on Windows x64" (чего лишний раз повторяться?), авторы который раздербанили Patch-Guard в пух и прах, мыщхъ сосредоточится на том, чего в этой статье не было.

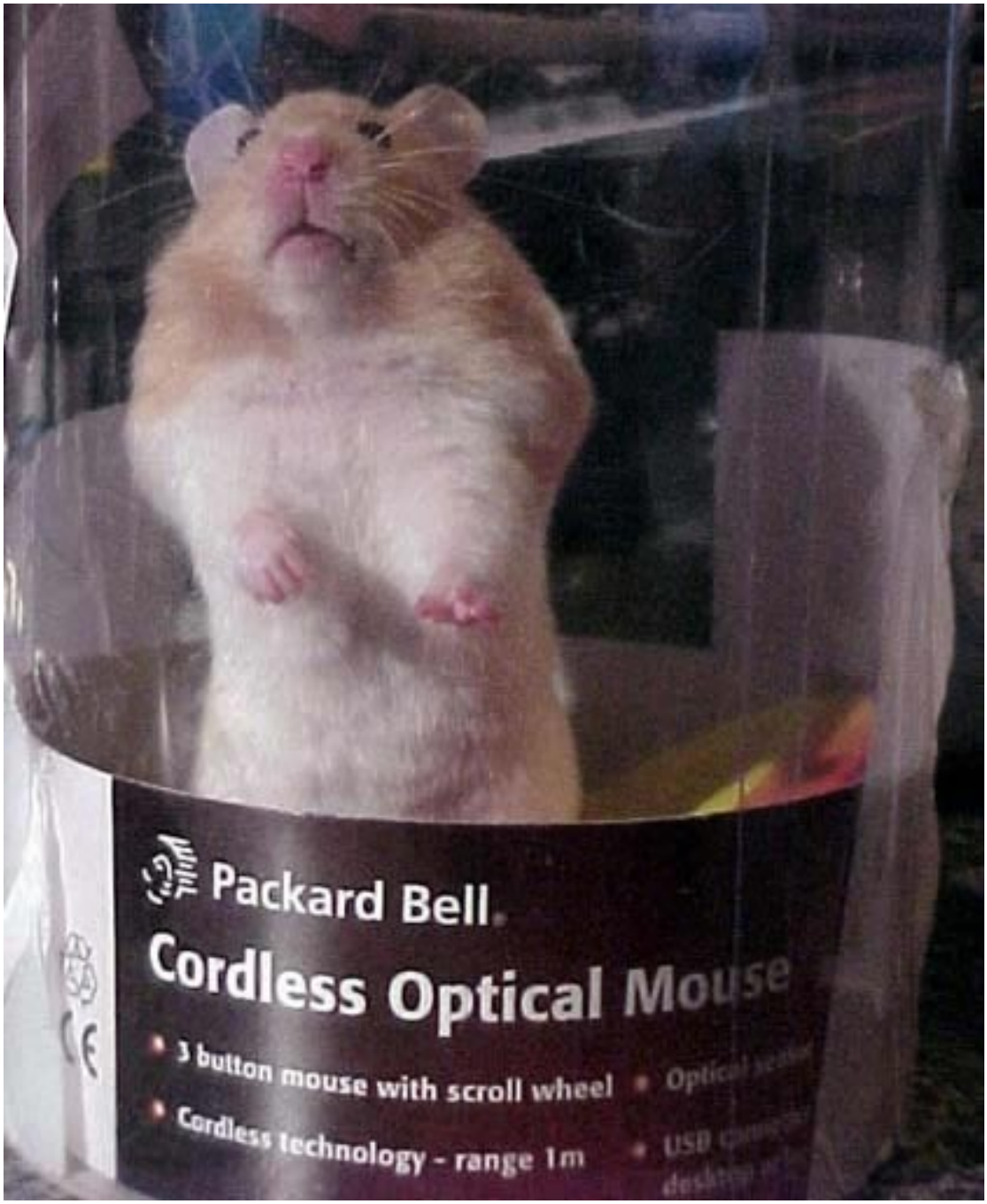

Рисунок 4 а первый взгляд Patch-Guard кажется неприступным…

Рисунок 5 …но на самом деле его очень легко одолеть (пускай даже придется побуреть и разметать кал от натуги)

симбиотические пути выживания

Собственно говоря, а чего это мы так встрепенулись? Чем нам, хакерам, мешает этот Patch-Guard и зачем его убивать? Пускай живет и защищает _нашу_ машину от всяких непрошеных тварей (типа кривых пионерских драйверов или червей), в то время как мы запустим свой хвост в чужие.

Рисунок 6 симбиоз — и никого убивать не надо

Все современные 64-разрядные процессоры семейства AMD x86-64 и Intel IA64, выпущенные после августа 2006 года, поддерживают механизмы аппаратной виртуализации. Достаточно, находясь в режиме ядра, выполнить машинную команду VMCALL (AMD) или VMXON (Intel), чтобы запустить гипервизор (в терминологии Intel – монитор виртуальных машин), переводящий операционную систему в гостевой режим и полностью контролирующий ее, перехватывая все обращения к портам ввода/вывода, прерывания, чтение/запись MSR-регистров (через которые, в частности, реализована инструкция SYSCALL). Мы как бы "подминаем" под себя операционную систему, работая в минусовом кольце (естественно, "минусовом" чисто условно) и осуществляем перехват без модификации базового кода/данных гостевой системы, так что Patch-Guard ничего не сможет определить. Причем, заткнуть эту дыру без _значительного_ ущерба для функциональности Microsoft не сможет хотя бы по чисто маркетинговым соображениям, иначе — прощай аппаратная виртуализация!

С другой стороны, даже если оставить виртуализацию в покое и вернуться в реальный мир — Patch-Guard может контролировать только неизменяемые области кода и данных, а в том же самом драйвере NDIS.SYS полно указателей, находящихся в стеке, пуле памяти и других изменяемых местах. Остается найти такой указатель, который был бы инициализирован после загрузки драйвера в память и указывал на вызываемый код. Хакеры уже давно и небезуспешно научились устанавливать хуки на NDIS.SYS в обход Patch-Guard'a, маскируя "нежелательную" сетевую активность или направляя/принимая пакеты, скрывающиеся от брандмауэра. Главный и единственный минус этого решения — не универсальность. Приходится либо "тащить" смещения указателей каждой версии NDIS'a за собой (что монстроузно, да и всех версий все равно не учтешь), использовать эвристические методы (которые весьма ненадежны), или… скачивать символьную информацию прямо с сервера Microsoft, воспользовавшись утилитой symchk, входящий в комплект поставки "Debugging Tools". Не слишком-то изящное решение, но зато надежное.

```
symchk NDIS.SYS /s srv*.\*http://msdl.microsoft.com/download/symbols -v
```

Листинг 4 добываем символьную информацию для текущей версии NDIS.SYS

обходи мента сзади, а Patch-Guard спереди

Для проверки целостности образом системных библиотек разработчики Patch-Guard'a не стали использовать тяжеловесные алгоритмы типа MD5, а взяли быстрый и легкий CRC. Очевидно, они не учили мат. часть, поскольку, CRC хоть и относится к _надежным_ алгоритмам, он ориентирован на _непреднамеренные_ искажения. Помехи на линии например. Дописав несколько "корректирующих" байт (для CRC8 — один, для CRC16 – два, для CRC 32 — четыре и для CRC64 – восемь), мы добьемся того, что контрольная сумма модифицированного образа останется _неизменной_ и Patch-Guard ничего не заметит!

Поскольку, Patch-Guard контролирует блоки фиксированного размера, то наша задача слегка усложняется и вместо дописывания корректирующих байт, мы должны записать их поверх неиспользуемых нулевых байт, или ненулевых — какая разница?! Главное, чтобы их значение можно было безболезненно менять на любое другое. В каждом файле легко найти множество команд NOP (опкод 90h), расположенных между функциями и служащих для выравнивания.

Остается только расковырять саму функцию, используемую Patch-Guard'ом для подсчета контрольной суммы. Листинг, приведенный ниже, позаимствованный из статьи "Bypassing PatchGuard on Windows x64":

```
PPATCHGUARD_SUB_CONTEXT PgCreateBlockChecksumSubContext(
        IN PPATCHGUARD_CONTEXT Context,
        IN ULONG Unknown,
        IN PVOID BlockAddress,
        IN ULONG BlockSize,
        IN ULONG SubContextSize,
        OUT PBLOCK_CHECKSUM_STATE ChecksumState OPTIONAL)
{
        ULONG64 Checksum = Context->RandomHashXorSeed;
        ULONG   Checksum32;

        // Checksum 64-bit blocks
        while (BlockSize >= sizeof(ULONG64))
        {
        Checksum    ^= *(PULONG64)BaseAddress;
        Checksum     = RotateLeft(Checksum, Context->RandomHashRotateBits);
        BlockSize   -= sizeof(ULONG64);
        BaseAddress += sizeof(ULONG64);
        }

        // Checksum aligned blocks
        while (BlockSize-- > 0)
        {
        Checksum    ^= *(PUCHAR)BaseAddress;
        Checksum     = RotateLeft(Checksum, Context->RandomHashRotateBits);
        BaseAddress++;
        }

        Checksum32 = (ULONG)Checksum;

        Checksum >>= 31;

        do
        {
                Checksum32  ^= (ULONG)Checksum;
                Checksum   >>= 31;
        } while (Checksum);
}
```

Листинг 5 код функции PgCreateBlockChecksumSubContext, рассчитывающий контрольную сумму заданного блока

Как можно видеть, на выходе функции PgCreateBlockChecksumSubContext мы получаем 32-битный Checksum, записываемый в структуру BLOCK_CHECKSUM_STATE вместе с базовым адресом и размером контролируемого блока (кстати говоря, префикс "Pg" определенно означает Patch-Guard, позволяя нам легко и быстро отделять принадлежащие к Patch-Guard'у функции ото всех остальных функций ядра):

```
typedef struct BLOCK_CHECKSUM_STATE
{
        ULONG   Unknown;
        ULONG64 BaseAddress;
```

```
        ULONG   BlockSize;
        ULONG   Checksum;
} BLOCK_CHECKSUM_STATE, *PBLOCK_CHECKSUM_STATE;
```

Листинг 6 структура, хранящая 32-битный CRC вместе с другими данными

Информационная емкость 32-битной контрольной суммы составляет всего 4 байта, которые элементарно рассчитываются даже без всякого перебора. Подробнее об этом можно прочитать в моей статье: "как подделывают CRC16/32" (валяющейся на ftp://nezumi.org,ru), а так же в замечательном хакерском руководстве, ориентированном на астматиков и доходчиво рассказывающим как вычисляется и подделывается CRC32 путем дописывания 4х корректирующих байт в конец контролируемого блока: http://foff.astalavista.ms/tutorialz/Crc.htm, добротный перевод которой на русский лежит на: www.pilorama.r2.ru/library/pdf/crcrevrs.pdf. Учитывая, что Microsoft в любой момент может пересмотреть свой позиции, расширив контрольную сумму до 8-байт (что на 64-разрядных процессорах очень легко сделать), нелишне будет заблаговременно ознакомится с базовыми принципами алгоритма CRC64, описание которого припрятано на: http://www.pdl.cmu.edu/mailinglists/ips/mail/msg02982.html.

Аналогичным образом осуществляется и модификация GDT/IDT – в них полно незадействованных полей и выкроить четыре байта не будет проблемой. А вот с SSDT дела обстоят похуже, поскольку Patch-Guard сверяет "рабочую" копию с ее "оригиналом", хранящимся внутри образа NTOSKRNL.EXE по адресу nt!KeServiceDescriptorTable. Кажется, что ситуация финиш, но нет! Ведь мы уже умеем безболезненно модифицировать образ ядра, следовательно, нам ничего не будет стоит синхронно произвести изменения в обоих таблицах и Patch-Guard снова ничего не заметит.

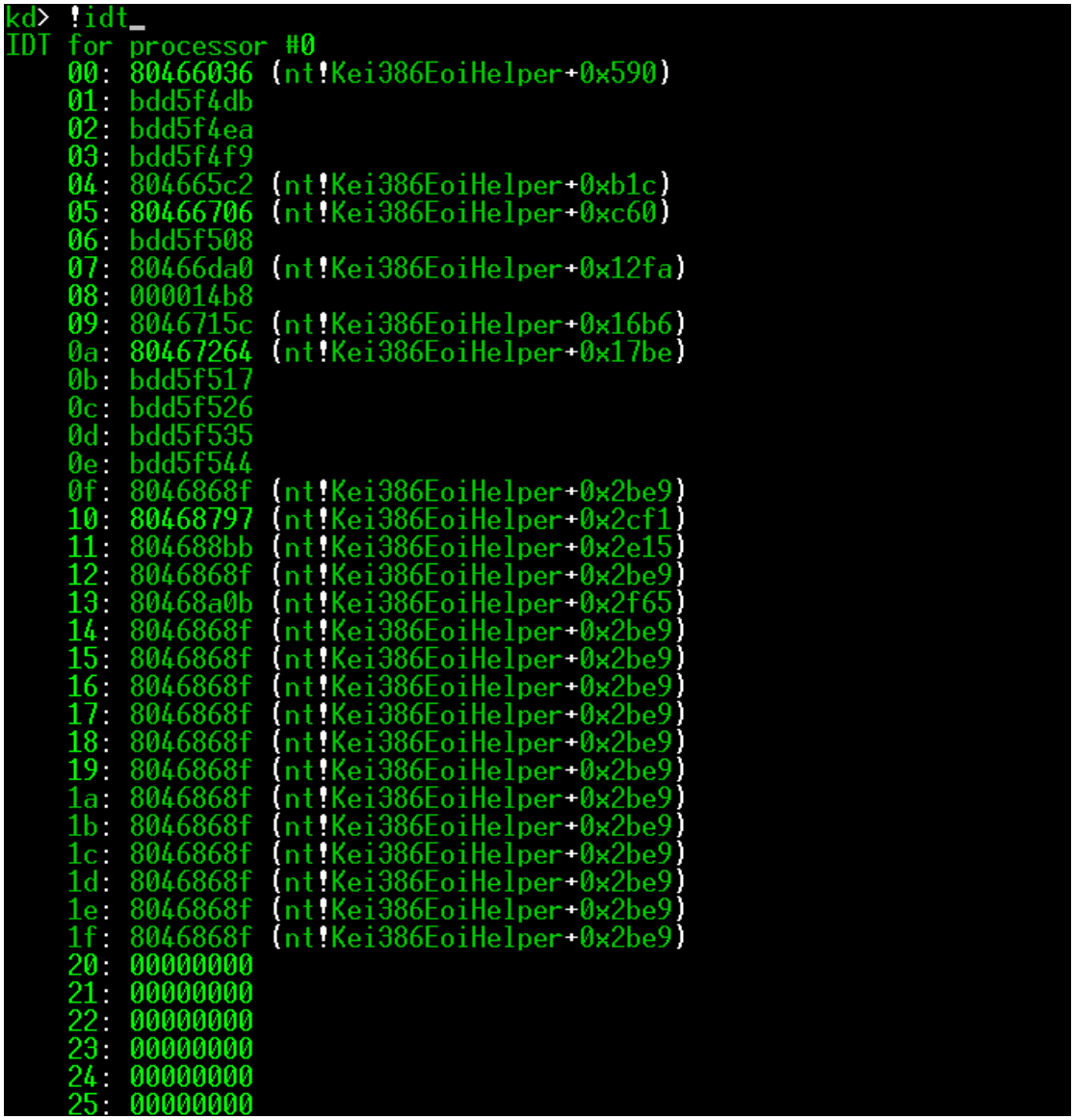

Рисунок 7 в IDT полно неиспользуемых нулей, которые можно использовать для коррекции

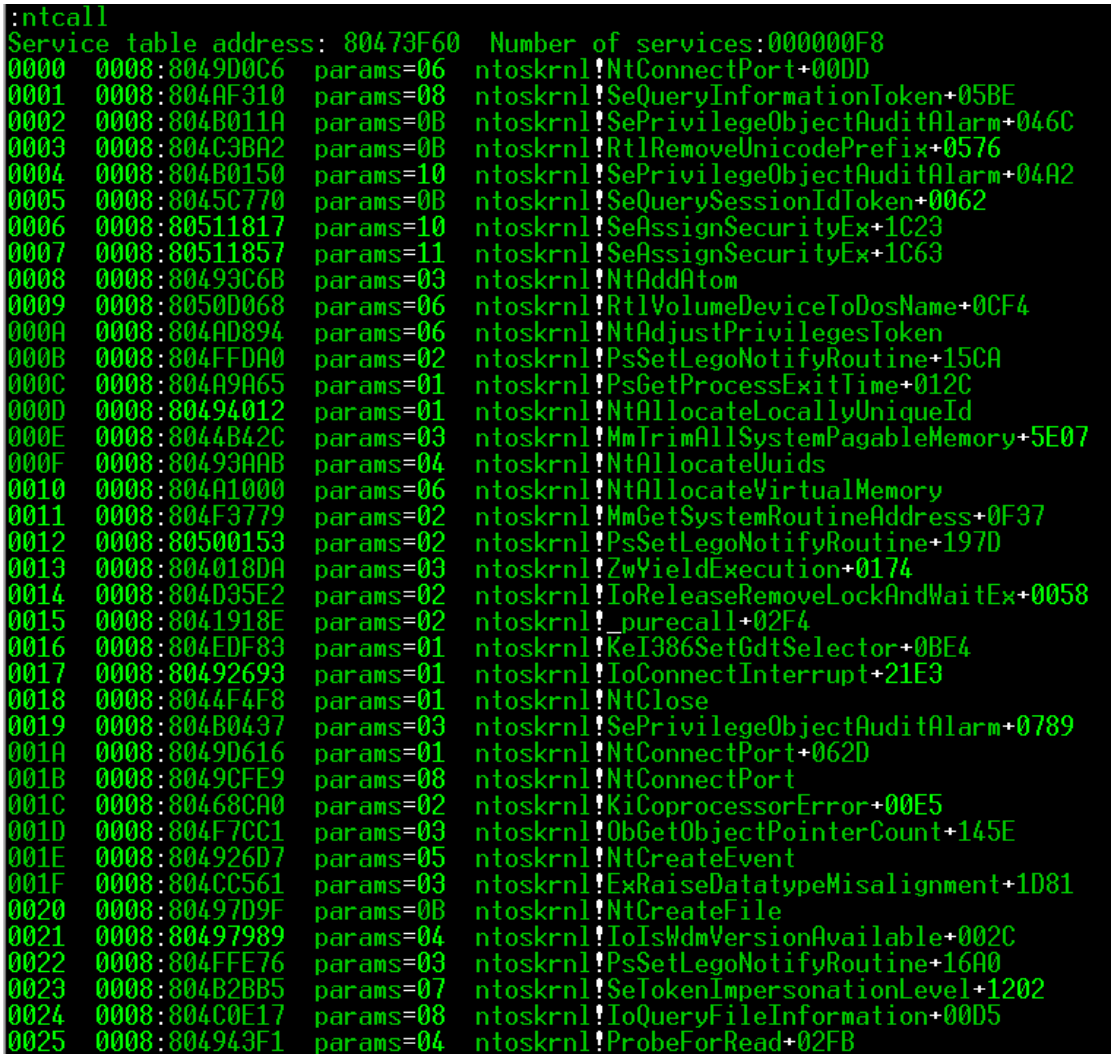

Рисунок 8 вместо подсчета контрольной суммы SSDT, Patch-Guard сверяет ее с оригиналом, поэтому обе копии приходится править синхронно

Весь вопрос в том — насколько надежен и "законен" такой хак? Ведь модификация образа ядра — далеко не атомарная операция! Сначала мы должны рассчитать CRC32 "исправленного" файла, затем модифицировать некое количество байт образа, после чего внедрить четыре "корректирующих" байта. А что если в промежутке между модификацией и "коррекцией" неожиданно проснется Patch-Guard и закричит: "измена!!! нас поимели!!!". Чтобы заставить его заткнуться, достаточно вспоминать, что "стражники" представляют собой обыкновенные отложенные процедуры — (Deferred Procedure Call) или, сокращенно, DPC, исполняющиеся на IRQL уровне равному двум. Если мы повысим IRQL (не забывая, что в многопроцессорных системах каждый процессор имеет свой IRQL), то никакие DPC исполняться не смогут пока уровень вновь не будет понижен. Правда, вместе с DPC не будет работать и подкачка с диска, так что можно очень легко нарваться на голубой экран смерти, но! Если сначала обратиться к той странице образа, которую мы собирались модифицировать, а затем — к странице, куда собрались внедрять корректирующие байты, обе страницы гарантированно окажутся в памяти и мы можем смело повышать IRQL на время модификации.

заключение

Так все-таки, насколько надежен Patch-Guard и от чего он нас реально защищает? По правде говоря (положа руку на хвост), как и все сделанное Microsoft, Patch-Guard катастрофически ненадежен. Во-первых, он может быть элементарным образом отключен (что и было продемонстрировано авторами статьи "Bypassing PatchGuard on Windows x64"), во-вторых, после установки гипервизора основная операционная система неожиданно для себя переходит в гостевой режим, полностью подконтрольный монитору виртуальных машин. В третьих, подсчет

контрольной суммы защищаемых объектов выполнен в "пионерском" стиле и не обнаруживает преднамеренные искажения. В принципе! И это все методы, не требующие перезагрузки!!! А если добавить сюда возможность создания собственного загрузчика, имитирующего наличие системного отладчика, то получится вообще косяк!

Черви, хакеры и rootkit'ы ничуть не пострадают, а вот легальным разработчикам придется попыхтеть. "По уму" в висте должна быть опция — задействовать Patch-Guard или нет, позволяющая пользователю выбирать несекьюрную машину, не совместимую с кучей программ, или несекьютрную машину, на которой все программы работают исправно. Но… Microsoft как всегда идет своим путем, повторяя дело, начатое Иваном Сусаниным.