использование ошибок синхронизации для атак

крис касперски ака мыщъх, no-email

ошибки синхронизации потоков встречаются так же часто как переполняющиеся буфера и, точно так же как и переполняющиеся буфера, их можно использовать для удаленных атак с засылкой shell-кода, тем более, что пока от них никто не пытается защищаться, открывая тем самым огромные возможности для хакерства и первые ласточки (в смысле черви) уже выползли из гнезда…

введение

Интенсивность атак на переполняющиеся буфера, достигнув своего пика в первых годах 21 века, начинает неуклонно иди на спад, сдаваясь под напором контратаки большого количества противохавкрских мер, предпринятых со стороны производителей процессоров, компиляторов и операционных систем. Неисполняемый стек, контроль целостности адреса возврата, рандоимизация адресного пространства — все это и многое другое затрудняет атаки, вынуждая хакеров искать обходные пути. И такие пути действительно есть!

3 сентября 2006 года в 14:37:48 по восточному стандартному времени хакер **johnny cache** описал принципиально новую атаку на драйвера устройств беспроводной связи Intel Centrino PRO, открывающую новую страницу в книге переполняющихся буферов: lists.immunitysec.com/pipermail/dailydave/2006-September/003459.html.

Вслед за этим, буквально месяц спустя, 11 октября 2006 года David Maynor из SecureWorks, Inc. и независимый исследователь Jon Ellch опубликовали сообщение о дыре в драйвере TOSRFBD.SYS, разработанного фирмой Toshiba для своих Bluetooth-устройств, используемых многими производителями оборудования, в числе которых оказались ASUS, Dell, Sony и другие бренды.

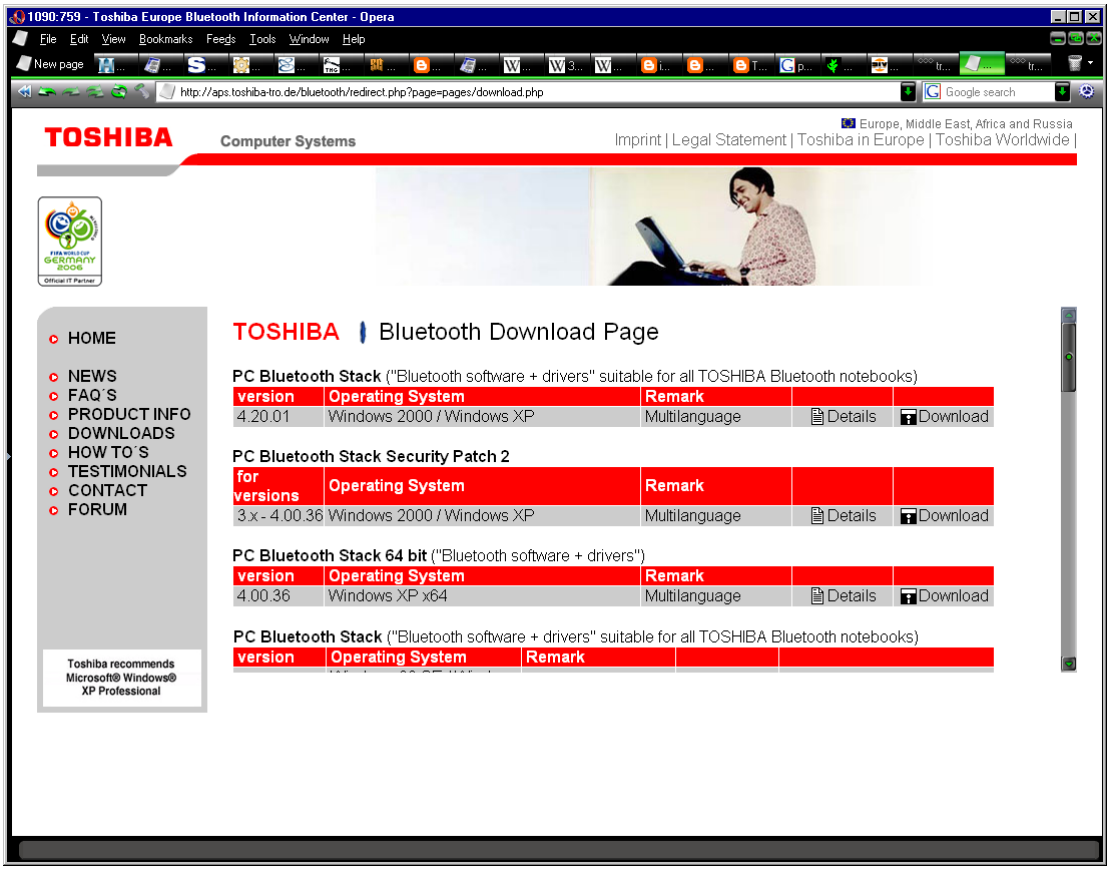

Рисунок 1 Toshiba выпускает заплатки для драйверов своих Bluetooth устройств

Обе атаки основаны на ошибках синхронизации потоков (race condition). Такие ошибки часто встречаются в драйверах, обрабатывающих асинхронные запросы от сетевых устройств и охватывающие широкий спектр оборудования простилающийся от инфракрасных адаптеров до DSL-модемов. Как известно, программы, работающие с большим количеством соединений (например, клиенты файло-обменных сетей), способны обрушивать систему в BSOD с посмертным сообщением IRQL_NOT_LESS_OR_EQUAL. Это — следствие "удара по памяти", возникающего из-за разрушения базовых структур данных в небрежно написанном драйвере сетевого устройства, страдающего ошибками синхронизации. Прикладные программы лишь создают условия наиболее "благоприятные" для проявления ошибки, но сами по себе они не виноваты и претензии пользователей направлены не по адресу, а претензий таких — много! Настолько много, что у большинства программ подобного рода соответствующий пункт явно включен в faq.

Аналогичная картина наблюдается и с Bluetooth-устройствами, сырость драйверов которых, породила целый класс атак под общим называнием **BlueSmack**, основанный на шторме эхо запросов, направленных на жертву и вызывающих все тот же "голубой экран смерти". Долгое время никто из хакеров не интересовался, что именно происходит при этом, какой характер носят разрушения и можно ли осуществить нечто более умное, чем банальный отказ в обслуживании.

johnny cache стал первым хакером, вызывавшим ***направленный*** удар по памяти, в результате которого ему удалось воздействовать на регистр EIP с передачей управление на shell-код, исполняющийся в режиме ядра, то есть на наивысшем уровне привилегий. За ним начали подтягиваться и остальные. Есть все основания утверждать, что через несколько лет ошибки синхронизации станут одним из основных типов удаленных атак. Причем, если для атаки на Bluetooth-устройство необходимо находится в радиусе его действия (которое в лучшем случае составляет несколько километров, да и то лишь в случае применения хакером специальной антенны), DSL-модемы доступы со всех концов Интернет!

реализация направленного удара по памяти

Механизмы обработки асинхронных событий драйверами подробно описаны в статье "многоядерные процессоры и проблемы ими порождаемые" из которой следует, что при работе с разделяемыми данными программист должен соблюдать особую осторожность и тщательно следить за синхронизацией. Сборка пакетов из фреймов как раз и представляет собой пример работы с разделяемыми данными, однако, про синхронизацию программисты традиционно забывают (или реализуют ее неправильно) в результате чего возникает угроза разрушения данных — если обработка одного фрейма прерывается в "неудобном" месте, — разделяемые структуры данных оказываются недостроенными, а поведение драйвера становится непредсказуемым.

В большинстве случаев в недостроенных структурах находится "мусор", среди которого встречаются и поля, ответственные за размер пакета, и указатели/индексы на другие данные. Очевидно, что попытка обращения по "мусорному" указателю (в том числе и указателю, вычисленному на основе "мусорного" индекса) с высокой степенью вероятности приводит к ошибке доступа. В драйверах уровня ядра она обычно заканчивается голубым экраном смерти, сопровождаемым сообщением IRQL_NOT_LESS_OR_EQUAL. Так происходит потому, что на том уровне IRQL, на котором работают отложенные процедуры и обработчики прерываний, подкачка страниц не работает и при обращении к странице, отсутствующей в памяти, операционная система генерирует ошибку IRQL_NOT_LESS_OR_EQUAL, что часто вводит в заблуждение многих пользователей и программистов относительно истинной причины аварии.

Для реализации DoS атаки обычно достаточно утилиты наподобие ping, генерирующей большое количество пакетов и позволяющей варьировать промежутки времени между посылками. Атака носит вероятностный характер и чтобы добиться проявления ошибки синхронизации необходимо послать достаточно большое количество пакетов через определенные промежутки времени, обусловленные конструктивными особенностями драйвера и "железки", которой он управляет. На некоторых промежутках ошибка синхронизации может вообще не провялятся, а на некоторых — выпрыгивать после нескольких пакетов (ну, "нескольких" — это в идеале). Можно посылать как UDP, так и ICMP пакеты, направленные как на открытый, так и на закрытый порт. Некоторые ошибки синхронизации проявляют себя на всем "спектре", некоторые — только на эхо запросах. Интервал времени от которого следует "плясать" составляет порядка 500 микросекунд.

Перед началом экспериментов необходимо настроить систему на сохранение полного дампа памяти в случае аварии (панель управления → система → дополнительно → загрузка и восстановление → запись отладочной информации → полный дамп памяти) или установить отладчик soft-ice, перехватывающий "голубые экраны смерти". В случае возникновения ошибки это поможет понять что же, собственно, произошло и какие перспективы в плане выполнения shell-кода оно дает. Главное — не забывать, что ошибки синхронизации носят вероятностный характер и потому обрушения могут происходить в различных местах, одни из которых допускают передачу на shell-код, а другие — нет.

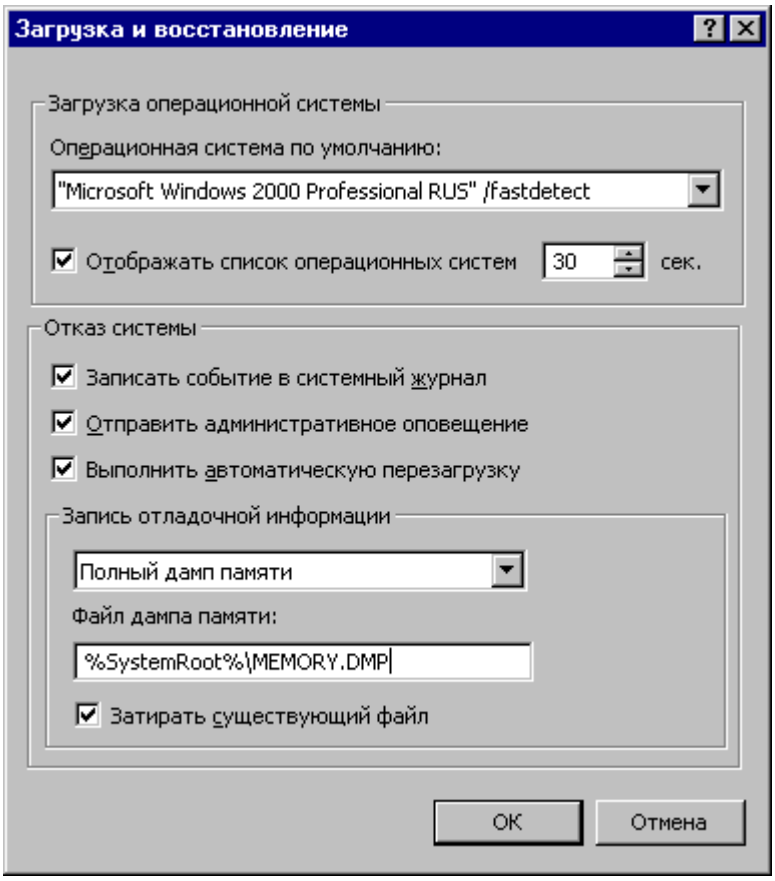

Рисунок 2 для анализа причин аварии необходим полный дамп памяти

В качестве иллюстрации можно привести пару дампов памяти, полученных johnny cache: http://www.802.11mercenary.net/~johnycsh/prone_to_deletion/dd/crash2.zip и http://www.802.11mercenary.net/~johnycsh/prone_to_deletion/dd/crash3.zip. Объектом атаки выступил драйвер беспроводного устройства Intel Centrino PRO, которому посылались UDP пакеты размером в 1400 байт, заполненных CCh байтами, и посылаемые на 2048-порт (прослушиваемый утилитой netcat) с интервалом в 400 микросекунд.

Через непродолжительное время после начала атаки, операционная система (которой в данном случае являлась Windows XP) выбрасывает "голубой экран" и сохраняет дамп памяти в файл memory.dmp. Для его анализа необходимо иметь отладчик Microsoft Kernel Debugger, входящий как в распространяемый по подписке DDK, так и в доступный для бесплатного скачивания Microsoft Debugging Tools: www.microsoft.com/whdc/devtools/debugging/default.mspx.

Загрузка дампа в отладчик в общем случае осуществляется так:

```
i386kd -z C:\WINNT\memory.dmp -y SRV*D:\sym*http://msdl.microsoft.com/download/symbols
```

Листинг 1 командная строка для загрузки дампа в отладчик

Здесь: "i386kd" — имя исполняемого файла консольной версии отладчика, если вы предпочитаете GUI – используйте вместо него windbg. Он нагляднее, но беднее по функциональности, однако, для наших задач вполне хватит и его; "C:\WINNT\memory.dmp" — путь к файлу памяти, по умолчанию создаваемом в системном каталоге Windows; "SRV*D:\sym*http://msdl.microsoft.com/download/symbols" — путь к каталогу, в котором

хранятся файлы с символьной информацией (в данном случае это "D:\sym") и адрес сервера Microsoft, откуда отладчик будет автоматически скачивать эти символы при необходимости. Символьные файлы занимают значительный объем и на медленных каналах их загрузка может занять значительное время. По умолчанию загружаются только символы ядра, после чего отладчик высвечивает приглашение и мерцающий курсор.

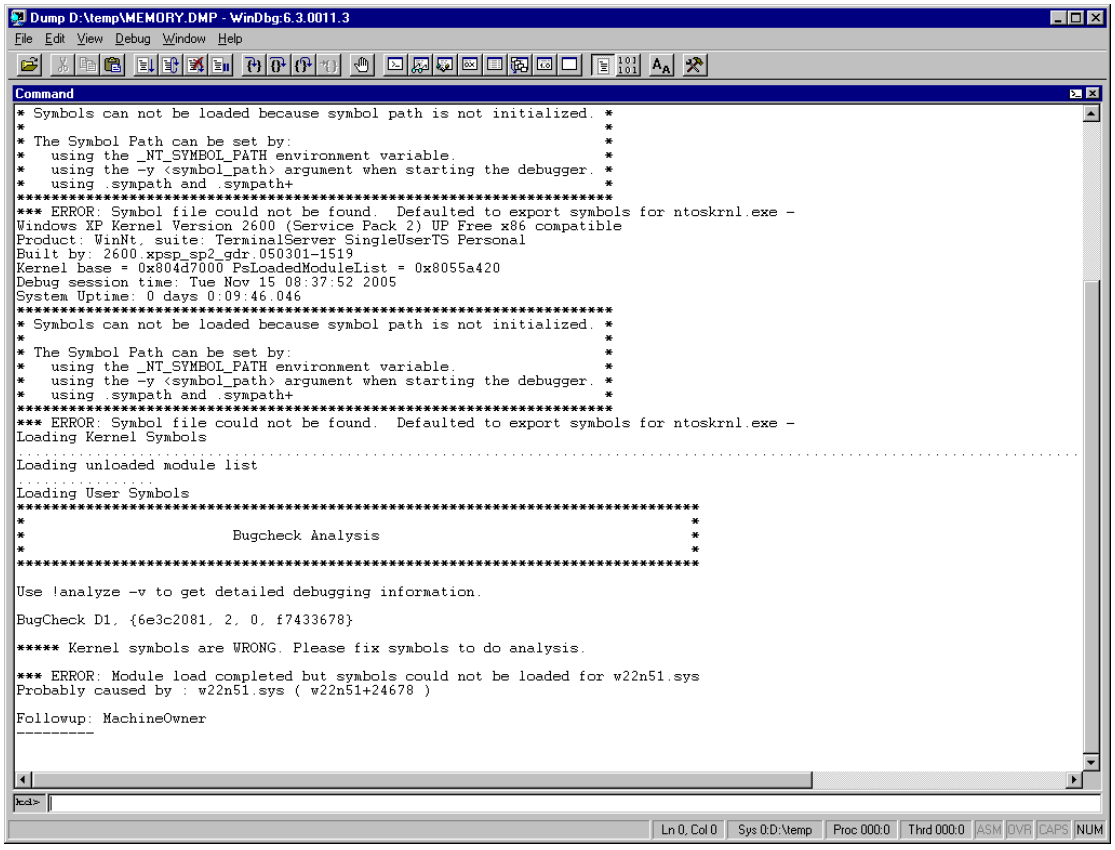

Рисунок 3 анализ дампа памяти в отладчике Microsoft WinDbg

Если теперь дать команду "u eip" (дизассемблирование по адресу EIP), то отладчик покажет отнюдь не истинное место сбоя, а процедуру обработки ошибки (в данном случае, KiTrap0E+0x233), которая не несет никакой полезной информации:

```
kd> u eip
nt!KiTrap0E+0x233:
804e187f f7457000000200        test    dword ptr [ebp+0x70],0x20000
804e1886 740d                  jz      nt!KiTrap0E+0x249 (804e1895)
```

Листинг 2 в загруженном дампе памяти регистр EIP указывает не на место сбоя, а на адрес, лежащий внутри обработчика исключения

Определить значение регистра EIP в момент сбоя позволяет команда "!analyze -v", результат работы которой приведен ниже:

```
kd> !analyze -v
DRIVER_IRQL_NOT_LESS_OR_EQUAL (d1)
Arguments:
Arg1: 6e3c2081, memory referenced
Arg2: 00000002, IRQL
Arg3: 00000000, value 0 = read operation, 1 = write operation
Arg4: f7433678, address which referenced memory

Debugging Details:
------------------
READ_ADDRESS: 6e3c2081
CURRENT_IRQL: 2

FAULTING_IP:
w22n51+24678
```

```
f7433678 8a11  mov     dl,[ecx]       ds:0023:6e3c2081=??

STACK_TEXT:
8054e9e0 65537365 2f3c746e 656d616e 200a0d3e w22n51+0x24678
7479426c 00000000 00000000 00000000 00000000 0x65537365

IMAGE_NAME:  w22n51.sys
```

Листинг 3 анализ дампа памяти, вызвавшего BSOD, но не оказавшего воздействия на регистр EIP

О чем это говорит? Во-первых, мы видим, что ошибка произошла на уровне IRQL равном 2, что соответствует DISPATCH_LEVEL, на котором выполняются отложенные процедуры. Если бы сбой происходил в обработчике прерывания, то уровень IRQL был бы выше, но в большинстве случаев обработчик прерывания не выполняет реальной обработки данных, а поручает это отложенным процедурам, которые вынуждены работать с разделяемыми данными и которые могут одновременно выполняться на несколько процессорах (как физических, так и виртуальных). Именно в отложенных процедурах сосредоточено наибольшее количество ошибок синхронизации. В результате одной из таких ошибок драйвер и пытается прочесть ячейку 6E3C2081h, находящуюся в странице памяти, вытесненной на диск (как уже говорилось, на уровне DISPATCH_LEVEL подкачка не работает).

```
nezumi.bat
C:\Program Files\Debugging Tools for Windows>i386kd -z D:\temp\crash2\memory.dmp -y D:\SYM -logo log2

Microsoft (R) Windows Debugger  Version 6.3.0011.2
Copyright (c) Microsoft Corporation. All rights reserved.

Loading Dump File [D:\temp\crash2\memory.dmp]
Kernel Complete Dump File: Full address space is available

Symbol search path is: D:\SYM
Executable search path is:
Windows XP Kernel Version 2600 (Service Pack 2) UP Free x86 compatible
Product: WinNt, suite: TerminalServer SingleUserTS Personal
Built by: 2600.xpsp_sp2_gdr.050301-1519
Kernel base = 0x804d7000 PsLoadedModuleList = 0x8055a420
Debug session time: Tue Nov 15 08:37:52 2005
System Uptime: 0 days 0:09:46.046
Loading Kernel Symbols
...............................................................................................
...............................................................
Loading unloaded module list
...............
Loading User Symbols
*******************************************************************************
*                                                                             *
*                        Bugcheck Analysis                                    *
*                                                                             *
*******************************************************************************

Use !analyze -v to get detailed debugging information.

BugCheck D1, {6e3c2081, 2, 0, f7433678}

*** ERROR: Module load completed but symbols could not be loaded for w22n51.sys
Probably caused by : w22n51.sys ( w22n51+24678 )

Followup: MachineOwner
---------

kd>
```

Рисунок 4 анализ дампа памяти в отладчике Microsoft i386kd

В момент сбоя регистр EIP равен F7433678h и указывает на машинную команду "mov dl, [ecx]", где ecx == 6E3C2081h, принадлежащую драйверу (IMAGE_NAME) w22n51.sys, щелкнув на свойства которого можно узнать, что это и есть тот самый дырявый драйвер, разработанный Intel.

Остается только выяснить как попадает в регистр ECX неправильный указатель и можно ли на него повлиять. Даем команду "u f7433678-10", чтобы дизассемблировать за 10h байт до места сбоя и смотрим:

```
kd>u f7433678-10
f7433668 0fb745f0      movzx   eax,word ptr [ebp-0x10]
f743366c 83f801        cmp     eax,0x1
f743366f 0f8e25240000  jle     w22n51+0x26a9a (f7435a9a)
```

```
f7433675 8b4df4        mov     ecx,[ebp-0xc]
f7433678 8a11          mov     dl,[ecx]        ; <- здесь произошел сбой
f743367a 8855ea        mov     [ebp-0x16],dl
f743367d 8b45f4        mov     eax,[ebp-0xc]
f7433680 8a4801        mov     cl,[eax+0x1]
f7433683 884dff        mov     [ebp-0x1],cl
f7433686 8b55f4        mov     edx,[ebp-0xc]
f7433689 83c202        add     edx,0x2
```

Листинг 4 окрестности команды, вызывавшей "голубой экран смерти" из-за ошибки синхронизации

Мы видим, что содержимое ECX загружается из локальной переменной [EBP-0Ch]. Вне зависимости от того, как оно туда попадает, драйвер копирует один байт из [EBP-0Ch] в [EBP-16h]. Обе переменных находятся в стеке, что уже хорошо (атаки на стек хорошо известны и давно отработаны). Допустим, нам удалось засунуть в [EBP-0Ch] указатель на определенную ячейку памяти, тогда, если мы сможем подменить [EBP-16h], данная конструкция будет действовать как оператор POKE языка Бейсик, записывающая произвольный байт по произвольному адресу, конечно, при условии, что он лежит в невытесненной на диск странице памяти, доступной на запись, например, стеке ядра, в котором хранится адрес возврата из функции. К сожалению, модификации одного-единственного байта для передачи управления на shell-код (размешенный здесь же, в стеке, в котором происходит сборка пакетов) явно недостаточно, но не будем отчаиваться! Ведь атака, как мы помним, носит вероятностный характер и сбои могут происходить в разных местах. Попробуем направить на жертву шторм пакетов еще раз, может быть, тогда нам повезет больше.

Ниже приведен анализ дампа памяти, в котором помимо разрушения памяти произошло воздействие на регистр EIP:

```
kd> !analyze -v
DRIVER_IRQL_NOT_LESS_OR_EQUAL (d1)
Arguments:
Arg1: 5c01abf7, memory referenced
Arg2: 00000002, IRQL
Arg3: 00000001, value 0 = read operation, 1 = write operation
Arg4: cccccccf, address which referenced memory

Debugging Details:
------------------
WRITE_ADDRESS: 5c01abf7
CURRENT_IRQL:  2
FAULTING_IP:
+ffffffffcccccccf
cccccccf 01963b10ffd6  add [esi+0xd6ff103b],edx

LAST_CONTROL_TRANSFER:  from ff103b96 to cccccccf

STACK_TEXT:
cccccccc ff103b96 01c486d6 00000000 00000000 0xcccccccf
01c486d6 00000000 00000000 00000000 00000000 0xff103b96
```

Листинг 5 анализ дампа памяти, оказавшего воздействие на регистр EIP

Уровень IRQL осталось прежним (DISPATCH_LEVEL), а вот значение регистра EIP изменилось драматически и вылетело далеко за пределы драйвера, попав в страницу с адресом CCCCCCCFh, каким-то чудом оказавшуюся в памяти и вызвавшую исключение по записи в инструкции "ADD [ESI+0XD6FF103B],EDX". Адрес CCCCCCCFh поразительно похож на "начинку" атакующих пакетов, но если это так, откуда тогда взялось число CFh? Ведь исходя из самых общих рассуждений, исключение должно было возникнуть по адресу CCCCCCCCh!

Но, как говорится, если факты не совпадают с теорией, тем хуже для фактов. С какой это стати исключение должно возникать по адресу CCCCCCCCh? Это было бы так, если бы страница оказалась вытесненной на диск, но по воле случая она очутилась в оперативной памяти. Выполнение команд началось с адреса CCCCCCCCh и продолжилось вплоть до инструкции, вызывавшей исключение, которой и оказалась команда "`add [esi+0xd6ff103b], edx`", обратившаяся к ячейке 5C01ABF7h, отсутствующей в оперативной памяти. Если бы не она, выполнение продолжалось бы и дальше!

Как это можно доказать? Хм, имея один лишь дамп памяти на руках, ничего доказать уже нельзя, но! Если наши рассуждения верны (а они верны), то команды, находящиеся между

адресами CCCCCCCCh и CCCCCCCFh должны выполняться без исключений, что легко подтвердить путем дизассемблирования.

"Скармливаем" отладчику команду "u CCCCCCCC" и получаем следующий результат:

```
kb>u cccccccc
cccccccc d6                  ??? ; setalc Set AL to Carry Flag
cccccccd 86c4                xchg    ah,al
cccccccf 01963b10ffd6        add     [esi+0xd6ff103b],edx
cccccccd5 86c4               xchg    ah,al
cccccccd7 0100               add     [eax],eax
cccccccd9 0000               add     [eax],al
cccccccdb 0000               add     [eax],al
cccccccdd 0000               add     [eax],al
```

Листинг 6 дизассемблерный листинг окрестностей места сбоя

Непосредственно по адресу CCCCCCCCh располагается неизвестная отладчику i386kd недокументированная команда SETALC (Set AL to Carry Flag), с опкодом D6h (но, например, тот же HIEW ее прекрасно понимает), а следом за ней — XCHG AH,AL. Обе команды не обращаются к оперативной памяти и при любом раскладе выполняются без исключений. А вот команда "ADD [ESI+0xD6FF103B], EDX", расположенная по адресу CCCCCCCFh, обращается к ячейке памяти, определяемой регистром ESI, который в данном случае оказался равен 7AFD6444h, в результате чего произошла попытка записи двойного слова по адресу 5C01ABF7h, расположенному (опять-таки волею случая) в вытесненной странице памяти, следствием чего стало исключение, которое и вызвало голубой экран смерти.

Таким образом, использование ошибок синхронизации для воздействия на регистр EIP все-таки возможно! Shell-код можно разместить прямо в пакете и при удачном стечении обстоятельств на него будет передано управление. При неудачном — жертва окажется свидетелем "голубого экрана смерти".

Остается только выяснить, какое именно двойное слово попадет в регистр EIP при удачной атаке. Это сложно определить путем анализа дампа памяти, так что лучше действовать экспериментально, заполняя разные части пакета различными байтам. Легко показать, что действуя методом "вилки", мы найдем нужное значение менее, чем за 30 попыток. Однако, следует помнить, что в силу вероятностного характера атаки, в регистр EIP могут попадать различные двойные слова, в результате чего количество необходимых попыток возрастет в несколько раз, но все равно останется сравнительно небольшим.

механизмы защиты от синхро-атак

В Windows Vista компания Microsoft реализовала множество защитных механизмов, главным образом нацеленные на переполняющиеся буфера, но так же затрудняющие и другие типы атак. Представляет интерес выяснить как обстоит дело с атаками, основанными на ошибках синхронизации. Все сказанное ниже, относится к Windows Vista RC1 – самой свежей версии на момент написания этих строк. Конечно, с выходом финальной версии (и многочисленных заплаток к ней) многие аспекты наверняка изменятся (в сторону усиления защиты), однако, никому не дано предугадать какой путь выберет Microsoft, поэтому будем изучать то, что дают (в смысле, что она раздает).

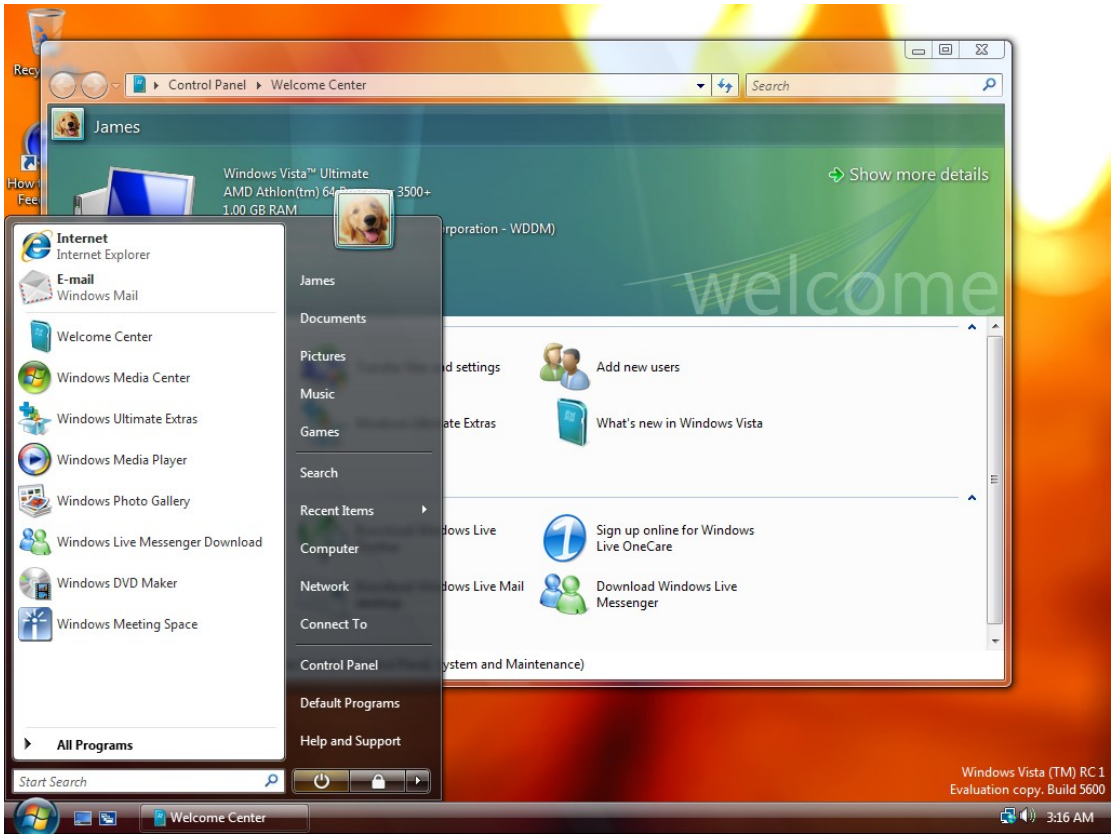

Рисунок 5 внешний вид операционной системы Windows Vista RC1

Самой значительной инновацией стала возможность переноса части драйверов с нулевого кольца на прикладной уровень. Новая драйверная модель, названная "User Mode Driver Framework" и являющая частью общей архитектуры драйверов "Windows Driver Foundation" впервые за всю историю существования Windows, наконец-то предоставила набор API-функций, позволяющих управлять USB и Firewire-устройствами непосредственно с прикладного уровня. Тоже самое относится и к драйверам-фильтрам.

Для обеспечения обратной совместимости, User Mode Driver Framework так же доступен и для Windows XP, что позволяет разработчикам писать драйвера, работающие в обоих операционных системах. Весь вопрос в том, какой процент из них этим воспользуется и как скоро начнется массовая миграция драйверов на прикладной уровень. Системные программисты крайне настороженно относятся к любым инновациям и не станут переписывать уже написанный код до тех пор, пока не почувствуют явной выгоды. Поскольку, безопасность драйвера не является основной потребительской характеристикой, есть все основания предполагать, что в ближайшие несколько лет никаких радикальных изменений не произойдет.

А вот драйвера для новых устройств, по-видимому будут писать с использованием User Mode Driver Framework уже хотя бы потому, что это значительно упрощает их отладку. И что тогда? Ошибки синхронизации как были, так и останутся, только вместо нулевого кольца, злоумышленник получает привилегии прикладного режима. Казалось бы — невелика разница! К тому же, программировать shell-код на прикладном режиме намного проще, поскольку можно использовать высокоуровневые API-функции.

На самом деле, реализовать атаку на User-Mode драйвера на два порядка сложнее! (Во всяком случае теоретически). Все дело в том, что в Windows Vista появилась рандомизация адресного пространства (Address Space Layout Randomization или, сокращенно, ASLR) и системные библиотеки теперь грузятся по одному из 256 возможных базовых адресов, а для обхода аппаратного DEP (см. статью "переполнение буфера на системах с неисполняемым стеком") необходимо знать адреса API-функций KERNEL32.DLL еще до начала атаки! Очевидно, что теперь вероятность успешного захвата системы в 256 раз ниже, чем раньше! Но не нужно быть специалистом в криптографии, чтобы понять, что 1/256 это _очень_ большое число, тем более, что при возникновении исключения в ядре, вспыхивает "голубой экран смерти", уводящий систему в перезагрузку, а драйвера прикладного режима просто перезапускаются, позволяя хакеру перебирать различные базовые адреса KERNEL32.DLL без

останова системы. Следовательно, _реальные_ шансы на успешную атаку не только не снизились, но даже возросли, причем весьма значительно. Короче, ситуация напоминает картину: вытащили голову — хвост увяз.

Но все-таки вернемся к ядру, в котором нам еще предстоит провести несколько лет. Хорошая новость (для хакеров) — в Windows Vista появилось множество новых native-API функций, существенно, упрощающих процесс кодирования rootkit'ов. В частности, появилась функция NtCreateUserProcess, запускающая новый процесс на выполнение. До этого процессы (из ядра!) приходилось запускать приблизительно так: отрываем PE-файл с флагами SYNCHRONIZE и FILE_EXECUTE, создаем новую секцию вызовом ZwCreateSection(,,,SEC_IMAGE), создаем процесс (вернее, не сам процесс, а некоторую "заготовку") через ZwCreateProcess, инициализируем блок параметров процесса функцией RtlCreateProcessParameters, выделяем в процессе память, подготавливаем блок параметров и устанавливаем на него указатель Peb->ProcessParameters, выделяем память для стека и инициализируем контекст стартового потока процесса, после чего создаем сам поток вызовом, ZwCreateThread. Наконец, регистрируем процесс в клиент-серверной подсистеме win32 (CSRSS), после чего пытаемся понять где мы допустили ошибку и почему процесс не запускается.

Рисунок 6 процессор AMD Athlon 64 с поддержкой технологии аппаратной виртуализации Pacifica

Другая хорошая новость — с появлением в процессорах поддержки аппаратной виртуализации (у AMD она скрывается за кодовым названием Pacifica, у Intel это — Vanderpool/Silvervale для IA32/IA64 архитектур соответственно), хакеры открыли для себя возможность перехода в режим гипервизора, переводящий операционную систему в гостевой

режим, и не дающий себя обнаружить никакими средствами. О большем подарке разработчики rootkit'ов не могли и мечтать! Первый rootkit нового поколения (названный "Голубой Пилюлей" — Blue Pill) был продемонстрирован Жанной Рутковской (Joanna Rutkowska) на конференциях SyScan (Сингапур) и Black Hat'e (США, Лас-Вегас), состоявшихся 21 июля и 3 августа 2006 года соответственно. Она же разработала методику скрытой передачи трафика, который не возможно обнаружить никакими существующими средствами контроля. Текст презентации (вместе с исходными кодами некоторых хакерских утилит и демонстрационными avi-роликами) можно найти на сайте Жанны: www.invisiblethings.org.

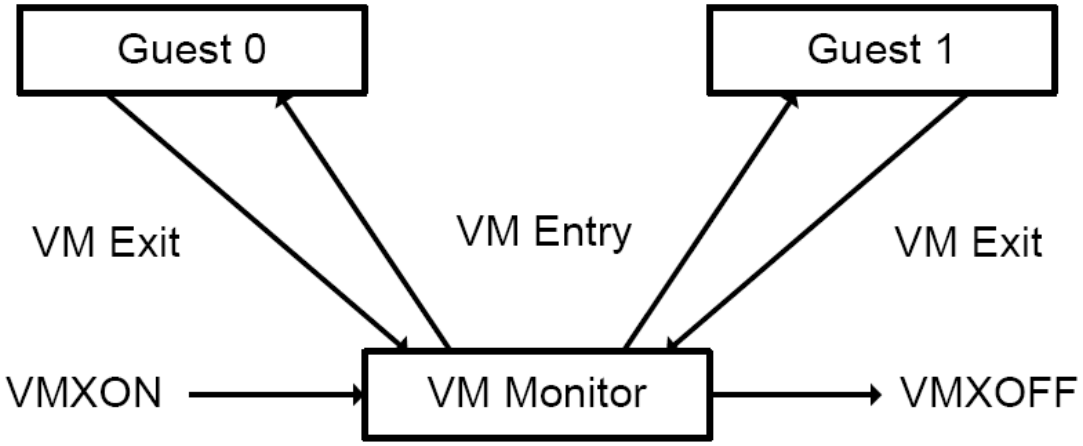

Рисунок 7 выполнение команды VMXON на Intel Pentium приводит к установке монитора виртуальных машин (в терминологии AMD – гипервизора), переводящего операционную систему в "гостевой" режим, полностью контролируемый гипервизором

Попытки разработать "Красную Пилюлю", определяющую присутствие гипервизора, пока ни к чему не привели. Единственное, что можно сделать — отключить аппаратную виртуализацию в BIOS (если BIOS это позволяет), но это сделает невозможным эффективную эмуляцию процессора, ради которой аппаратная виртуализация и затевалась.

Третья хорошая новость и не просто хорошая, а очень даже сногсшибательная: в 64-битной редакции Windows Vista появился механизм, контролирующий целостность ядра и названный Patch-Guard, так же реализованный в пакетах обновления для XP 64-bit и Server 2003 64-bit (http://www.microsoft.com/whdc/driver/kernel/64bitpatching.mspx).

Отныне запрещается перехватывать ядерные функции, модифицировать таблицу дескрипторов прерываний и совершать другие "противоправные" действия без которых существование надежных антивирусов и персональных брандмауэров невозможно себе представить! Предлогом для этого шага послужила небрежность большинства коммерческих и некоммерческих продуктов, неумело модифицирующих ядро и вызывающих нестабильную работу системы, вплоть до невозможности загрузки или частых появлений BSOD. Ну а пользователи, как обычно, во всем винят Microsoft и ее "кривую" Windows.

На самом деле, Microsoft просто вытесняет игроков с рынка, проталкивая свои собственные и далеко не самые лучшие решения. Но даже если бы они были лучшими — что с того? Задача обхода произвольного антивируса/брандмауэра в общем случае не разрешима, а тащить за собой кучу специализированных решений — слишком сложно, громоздко и неэффективно. Теперь же этого не понадобиться, поскольку антивирус и брандмауэр будут интегрированы в ядро, а продукты сторонних разработчиков работать не смогут. И хотя Microsoft предоставила ряд API-функций для контроля за открываемыми файлами и сетевым трафиком — такую "защиту" очень легко обойти. Одно приложение (антивирус) устанавливает фильтр, другое (вирус) снимает его легальными средствами и антивирус никак не может этому воспрепятствовать (в лучшем случае он просто узнает, что фильтр снят). Самое забавное, что Patch-Guard очень легко "отломать", о чем подробно рассказано в статье "Bypassing PatchGuard on Windows x64" (http://uninformed.org/index.cgi?v=3&a=3&t=sumry), но легальные разработчики не могут пользоваться подобными хакерскими средствами! Им придется либо забросить свой бизнес, либо сменить ориентацию (например, ограничится автономным файл сканером).

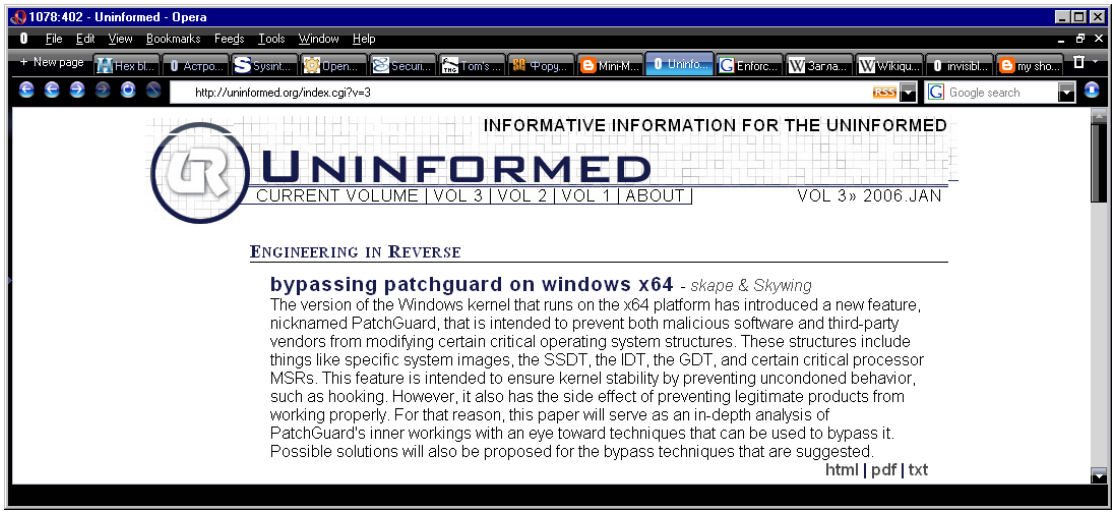

Рисунок 8 статья, описывающая различные пути обхода механизма Patch-Guard, следящего за целостностью ядра и позволяющего его модифицировать

Но как бы там ни было, хакеры находятся в более выигрышном положении, чем все остальные. Остается только надеяться, что 64-битные процессоры никогда не займут господствующего положения и основной архитектурой останется IA32.

заключение

Защитные механизмы на несколько шагов отстают от передовой хакерской мысли и хотя время от времени этот разрыв сокращается, "догнать и перегнать" получается только на бумаге, а в реальной жизни хакеры продолжают процветать. Причем, это проблема касается не только Microsoft, но и остальных производителей программного обеспечения. Ведь драйвера для сетевых устройств совсем не Microsoft пишет! А в Intel и Toshiba вовсе не дураки сидят. Почему же мы имеем такую плачевую ситуацию? Одна из причин — бешенная "гонка вооружений". Всякая фирма стремится выбросить свой продукт первой. Сжатые временные рамки не позволяют как следует протестировать код и если разработчикам удается довести его до минимально рабочего состояния (запускается и не падет), это уже хорошо.

До тех пор, пока безопасность не войдет в ряды потребительских критериев, определяющих объемы продаж, никто не станет ей всерьез заниматься. Microsoft привлекла большое количество хакеров для тестирования Windows Vista, но чего она добилась при этом?! Пожалуй, ничего (за исключением рекламы). Трудно поверить, что хакеры пропустили такие вопиющие дефекты системы безопасности, о которых написано выше (что же за хакеры это были?!). Скорее всего, им не позволили воздействовать на концептуальные архитектурные решения, принятые "свыше". Microsoft не занимается безопасностью Windows (только разводит "пиар"), и не будет ею заниматься, точно так как не будет исправлять старые ошибки без добавления новых, потому что "работа над ошибками" требует вложения огромных денег, но не приносит никакого дохода.

Поэтому, в ближайшие несколько лет следует ожидать новой волны хакерских атак, от которых можно защититься только переходом на надежные операционные системы, например, OpenBSD, но, увы, это решение подходит только для серверов. Создание офисных рабочих станций на чем-то отличном от Windows, создает гораздо больше проблем, чем их решает, а про домашние компьютеры (являющиеся по совместительству игровыми приставками) мы вообще молчим.

Рисунок 9 путь, которым нас ведет Microsoft