exploits review
1Eh выпуск

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

в конце октября в Малайзии состоялась конференция HITB (Hack In The Box), где мыщъх зачитал доклад, посвященный ошибкам ЦП, допускающих локальные и удаленные атаки на систему, что вызвало огромный общественный интерес и потому сегодняшний обзор сплоитов мы решили посвятить процессорным багам, продемонстрировав технику вторжения в чужие компьютеры

Intel Core – еще одна ошибка когерентности L1 кэша

brief: в середине октября (непосредственно перед конференцией HIBT) Intel обнаружила еще один дефект кэш-контролера первого уровня, который и описала в errata под номером AZ73. Приведенной информации оказалось достаточно для воспроизведения ошибки на ноутбуке, но отладить exploit в условиях гостиничного номера за оставшееся перед выступлением время не получилось, тем не менее модификация ядерной памяти операционной системы с прикладного уровня проходила стабильно и вполне успешно, а потому есть все основания ожидать, что данная техника будет взята на вооружение malware-писателями. Для этого достаточно заставить несколько процессорных ядер обрабатывать разделяемые данные, находящиеся в кэш-памяти первого уровня между двумя кэш-линейками. Согласно errata при этом происходит нарушение очередности последовательности записей/чтения, что не совсем соответствует действительности. Intel "забыла" упомянуть "удар по памяти" — кэш-контроллер "забывает" подлинный адрес скэшированных ячеек памяти и выгружает модифицированные данные совершенно в другое место, причем, выгрузка проводится уже после проверки атрибутов защиты страниц за счет чего становится возможным атаковать код операционной системы с прикладного уровня, причем атаковать можно очень большое количество приложений в том числе и операционную систему, даже если сами по себе они и не содержат ошибок. Атакующему нужно чтобы целевой код обрабатывал разделяемые данные в двух или более потоках, причем, один поток записывал, а другой — читал. Драйвер TCP/IP стека представляет собой отличную мишень для атаки, поскольку, в момент прихода очередного IP пакета сетевая карта генерирует прерывание, подхватываемое свободным процессором и, если в этот момент упадет еще один пакет, операционная система автоматически отправит его на второй процессор, вынужденный взаимодействовать с первым (а как еще иначе собирать TCP пакеты из IP?!). Направленный шторм TCP/IP пакетов — "пробивает" любую операционную систему (ес-но, из тех, что поддерживают многопроцессорность, Windows 9x не поддерживает и потому ни хвоста ее таким образом не сломаешь). Shell-код, разумеется, привязан к конкретной операционной системы и должен проектироваться с учетом специфики ее архитектуры, однако, это уже вторая фаза атаки, которая не так интересна, как принципиальная возможность удаленного "впрыскивания" кода в память ядра. О том, как это сделать можно прочитать в Specification Update: http://download.intel.com/design/mobile/specupdt/320121.pdf (AZ73. Memory Ordering Violation With Stores/Loads Crossing a Cacheline Boundary);

targets: Intel Core 2 Extreme Quad-Core Mobile, Intel Core 2 Quad Mobile, Intel Core 2 Extreme Mobile, Intel Core 2 Duo Mobile Processor, Intel Core 2 Solo Mobile и Intel Celeron 45-nm;

exploit: ниже приводится ключевой фрагмент proof-of-concept exploit'a разработанного хакером Zen Lee с целью завешивания целевой системы. Циклы CORE1/CORE2 должны выполняться в раздельных потоках, "закрепленных" за своими ядрами, регистр EDX – указывает на заранее выделенный блок памяти размером 64Кбайт, общий для всех потоков:

```
CORE1:
        mov     eax, dword_405030
        imul    eax, 343FDh
        add     eax, 269EC3h
```

```
        mov     dword_405030, eax
        sar     eax, 10h
        and     eax, 7FFFh
        cdq
        mov     ecx, 1FFh
        idiv    ecx
        mov     edx, dword_4054C2[edx]
        mov     dword_405438, edx
JMP CORR1

CORE2:
        mov     eax, dword_405030
        imul    eax, 343FDh
        add     eax, 269EC3h
        mov     dword_405030, eax
        sar     eax, 10h
        and     eax, 7FFFh
        cdq
        mov     ecx, 1FFh
        idiv    ecx
        mov     eax, dword_405438
        mov     dword_4054C2[edx], eax
        jmp     short _main
JMP CORR2
```

Листинг 1 ключевой фрагмент боевого exploit'a от Zen'a Lee, демонстрирующий дефект кэш-контролера (внимание: exploit работает только с 32-битными операционными системами!)

solution: Intel работает над исправлением микрокода, который после выхода в свет будет доступен разработчикам BIOS, однако, далеко не все из них включают последние версии микрокода в очередную версию своей прошивки, да и обновления BIOS'a на ноутбуке операция не из штатных, а потому ситуация ласты и кто не спрятался, тот сам себя и наказал.

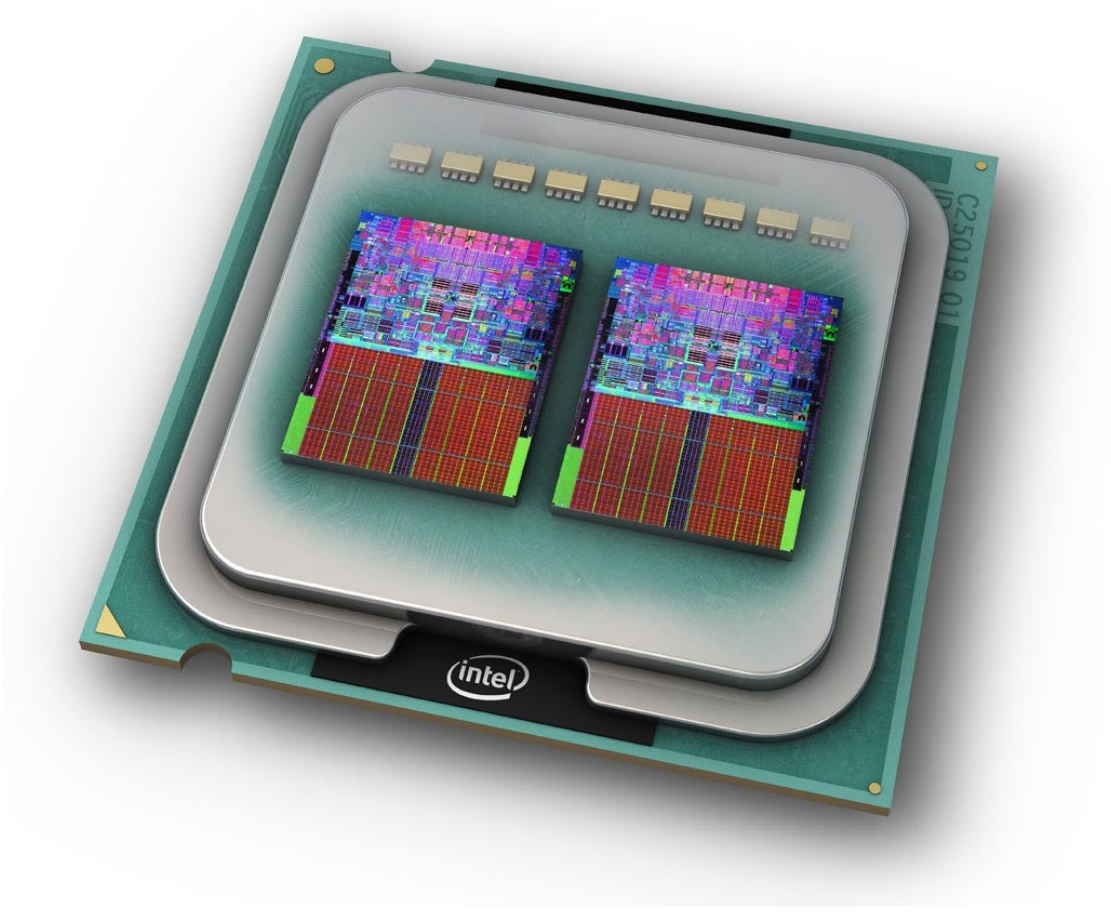

Рисунок 1 процессор Intel Core 2 Extreme Quad-Core кишками наружу

Intel Core – множественные ошибки XRSTOR/XSAVE

brief: команда XRSTOR, восстанавливающая состояние процессора, сохраненное командой XSAVE, оказалась жутко багистной — только за последний месяц в ней обнаружилось три новых ошибки, ведущих к краху системного программного обеспечения, использующего ее в своих целях. Ошибка, проходящая в errata (см. http://download.intel.com/design/mobile/specupdt/320121.pdf) под кодовым номером AZ74 (The XRSTOR Instruction May Fail to Cause a General-Protection) позволяет процессору взводить зарезервированные биты 63:9 регистра XFEATURE_ENABLED_MASK (XCR0) _без_ генерации исключения общей защиты, как это следует из документации, что само по себе неопасно, однако, позволяет создать код, работающий только на багистных процессорах. В малвари, отловленной в дикой природе, данный баг использовался для детекции уязвимых процессоров, а, быть может, и еще для чего-то. Две других ошибки относятся к инструкции XSAVE. Согласно errata, AZ71 (The XSAVE Instruction May Erroneously Set Reserved Bits in the XSTATE_BV Field), как и следует из ее названия, при определенных обстоятельствах взводит зарезервированные биты регистра XCR0, что ведет к непредсказуемому поведению программного обеспечения, уверенного, что эти биты равны нулю (как утверждает документация). Ошибка AZ72 (Store Ordering Violation When Using XSAVE) намного более коварна: если программист использует XSAVE для сохранения одного лишь SSE-контекста, то происходит переупорядочивание операций записи — инструкции отгружают данные в память совсем не в том порядке, в котором этого ожидает программист, что ведет к непредсказуемому поведению приложения, зачастую сопровождаемого крахом;

targets: Intel Core2 Extreme Quad-Core Mobile, Intel Core 2 Quad Mobile, Intel Core 2 Extreme Mobile, Intel Core 2 Duo Mobile Processor, Intel Core 2 Solo Mobile и Intel Celeron 45-nm;

exploit: отсутствует;

solution: Intel исправила ошибку в процессорах со степпингом E-0, а так же выпустила обновленный микрокод, однако, ввиду того, что ошибки реализации XRSTOR/XSAVE невозможно использовать ни для удаленных атак, ни для локального повышения привилегий, смысла накладывать заплатки на процессор нет;

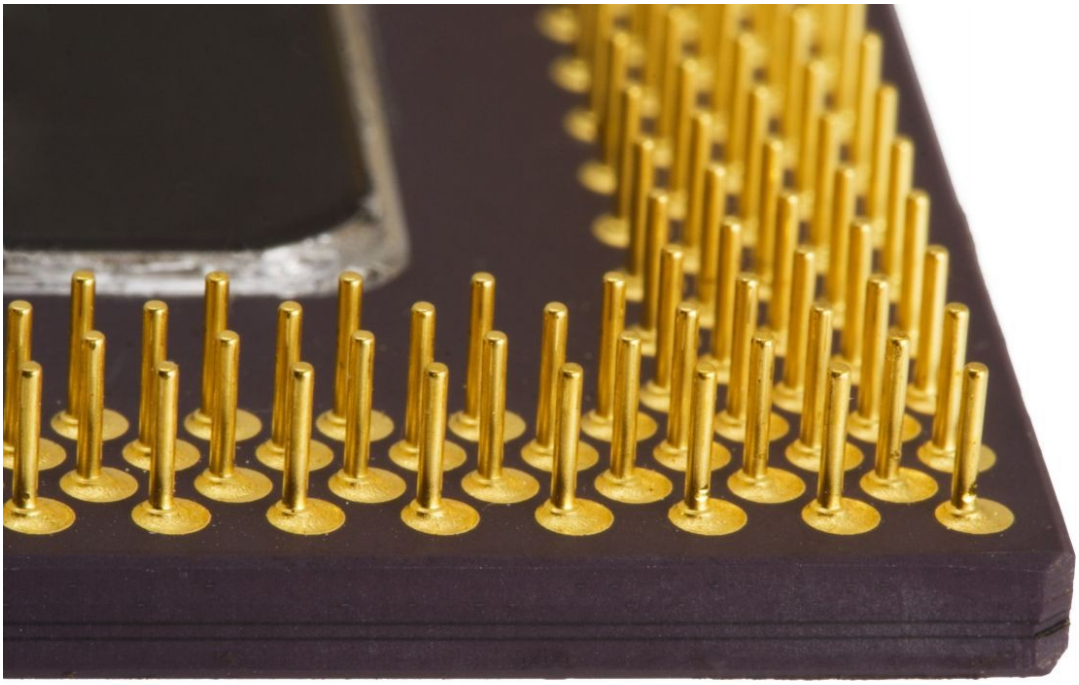

Рисунок 2 процессор Intel Core2 Extreme Quad-Core под микроскопом

Intel Core – разрушение регистра CS при переходе в PM

brief: наконец-то Intel обнародовала информацию, объясняющую логику работы антиотладочных приемов, встречающихся в некоторых rootkit'ах, впрочем, "обнародовала" это сильно сказано! Всего лишь приоткрыла дверь в потайную комнату, и в образовавшуюся щель просочился крошечный лучик света, позволяющий (не без мата, конечно) разобраться в ситуации. На первый взгляд сообщение об ошибке AZ70 (Corruption of CS Segment Register During RSM While Transitioning From Real Mode to Protected Mode) носит невменяемый характер: если при переходе из реального в защищенным режим неожиданно придет прерывание от Системного Менеджера Прерываний (System Management Interrupt или, сокращенно, SMI), случившееся после того, как бит PE (Protection Enable) регистра CR0 уже взведен, но JMP FAR еще не начинал выполняется, процессор угробит два младших бита регистра CS, однако, все будет работать… во всяком случае, если программист не попытается прочитать содержимое CS инструкцией MOV или каким другим способом. Ситуация, прямо скажем, невероятная. Вероятность возникновения прерывания на столь узком временном промежутке близка к нулю, к тому же с угробленным CS можно какое-то время работать пока не начать читать его содержимое. И в чем здесь соль? А в том, что регистр CS, как и другие сегментные регистры, является лишь "надстройкой" над механизмом трансляции адресов и внутри процессора явным образом не используется. Именно потому с испорченным CS система продолжает работать, а испортить регистр можно множеством способов (связанных с ошибками реализации команд гипервизора, например). Прикладная программа после порчи регистра CS продолжает работать нормально, а вот отладчики выбрасывают исключение, ругаясь на неправильный селектор. Хакер смотрит — и офигивает. Селектор, действительно, неправильный! Однако, попытки разобраться: откуда берется неправильный селектор в таблице дескрипторов и куда исчезает — обречены на провал, поскольку на самом деле, селектор ни откуда не берется и никуда не девается, это всего лишь следствие разрушения регистра CS. Вот такой антиотладочный прием. Перезагрузка CS валидным значением позволяет продолжить отладку.

targets: Intel Core2 Extreme Quad-Core Mobile, Intel Core 2 Quad Mobile, Intel Core 2 Extreme Mobile, Intel Core 2 Duo Mobile Processor, Intel Core 2 Solo Mobile и Intel Celeron 45-nm;

exploit: отсутствует;

solution: Intel исправила ошибку в процессорах со степпингом E-0, а так же выпустила обновленный микрокод;

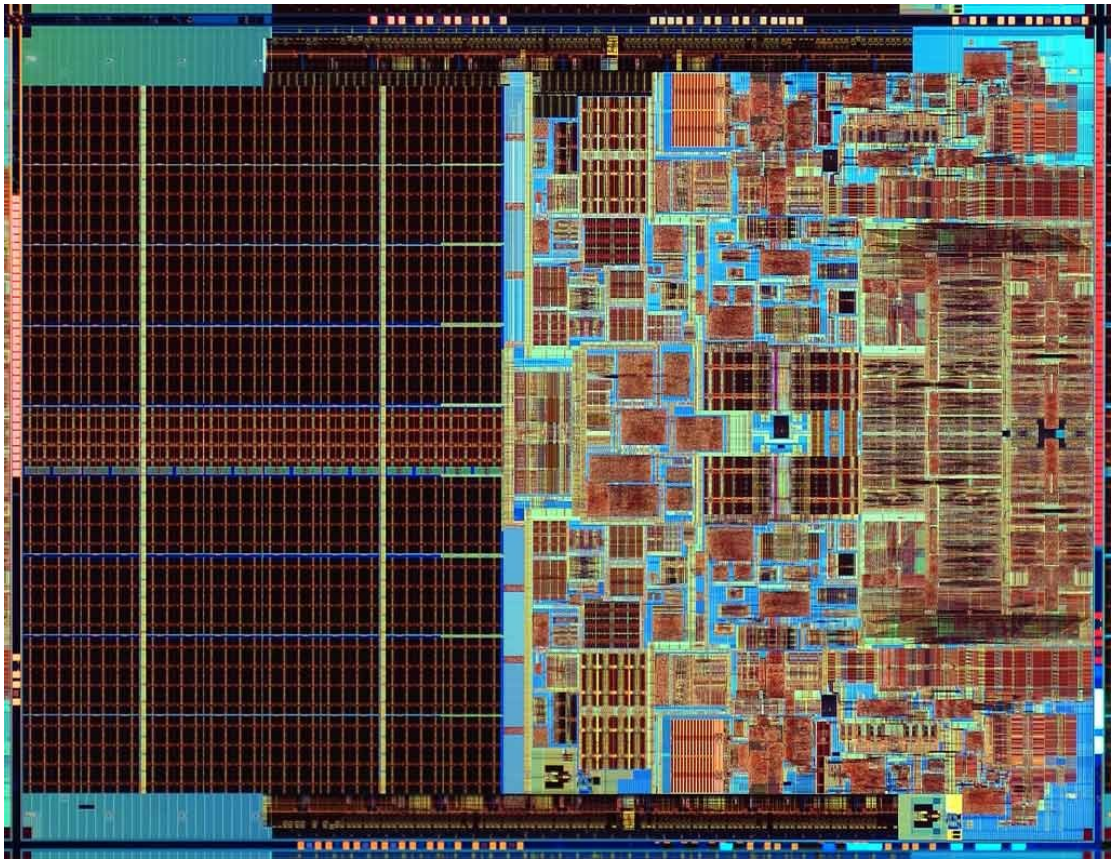

Рисунок 3 микромир процессора Intel Core2 Extreme Quad-Core

full disclose

Cross-Modifying Code Attacks

Первое (в жизни!) мыщъхиное выступление закончилось провалом. Только взошел на подиум — тут же забыл половину из того, что хотел сказать, в результате вместо запланированных 60 минут говорил только полчаса с ужасным русским акцентом. Народ обречено втыкал в слайды, набранные мелких шрифтом, а качество мультимедийного проектора было, скажем так, далеко не на высоте. Чувство уверенности, что это действительно провал, укрепилось на заключительной вечеринке, где к мыщъху подходили симпатичные японки и по-дружески толкая плечом говорили, что все было круто, типа, не переживай. Ага, понятно. Если бы все было действительно круто, они бы не подходили. Впрочем, мыщъх и не думал переживать. В конце концов надо же с чего-то начинать…

Рисунок 4 в Куала Лумпуре (столице Малайзии) совершенно нереально заблудиться. Башни-Близнецы, возвышаясь над остальными небоскребами, видны из любой точки города, образуя устойчивый ориентир, различимый даже из прилегающих к столице деревень в одной из которых живет очаровательная малайка, с которой мыщъх познакомился на Sky-Bridge, напросившись в гости. Лучше бы не просился. В деревнях такси хрен поймаешь и до ближайшего автобуса пришлось топать четыре часа!

Однако, презентация подняла намного больше вопросов, чем их решила и обсуждая с хакерами перспективы сложившийся ситуации, мыщъх обогатил свой хвост свежими идеями, открывающими двери в мир новых атак. Скромные рамки журнальной статьи не позволяют рассказать обо всех атаках целиком, поэтому, приходится выбирать что-то одно. Наибольший интерес вызвал класс атак официально обозначенный Intel'ом как Cross-Modifying Code Bug или, сокращенно, XMC.

Рисунок 5 Башни, экзотическое растение и Sky Bridge между ними

Как известно, Pentium-процессоры используют раздельный кэш первого уровня. Один для кода, другой — для данных, причем, внутри кристалла реализован специальный механизм, отслеживающий модификацию ячеек памяти, уже загруженных в кодовый кэш и вызывающий его перезагрузку. Вплоть до Pentium-III включительно детект самомодифицирующего кода не

представлял большой проблемы, поскольку кэш был устроен предельно просто, но начиная с Pentium-4, кодовый кэш хранит не оригинальное содержимое оперативной памяти, а декодированные микроинструкции, что существенно затрудняет проверку их принадлежности к модифицированным ячейкам. Механизм распознавания самомодифицирующего кода резко усложнился и в нем появились ошибки…

При определенных ситуациях (о которых мы еще поговорим) процессор продолжает исполнять старый код, игнорируя факт его модификации на другом ядре (реже — на том же самом ядре), в результате чего мы получаем в свое распоряжение превосходный антиотладочный прием, используемый еще во времена древних XT/AT – при "живом" прогоне программы модификация идет лесом, то есть не воспринимается процессором, поскольку модифицируемые команды уже находятся на конвейере, а сбросить конвейер некому — детектор самомодифицирующего кода ловит муху.

Рисунок 6 проблема Азии в том, что в ней нет азиатов. Монорали. Час пик. Одни европейские лица. И стоило ради этого тащится за 10 тыс км?!

Отладчик — совсем другое дело. При пошаговой трассировке между соседними командами процессор выполняет сотни и даже тысячи других команд, а потому самомодифицирующийся код исполняется как положено, то есть без ошибок. Аналогичным образом обстоят дела с дизассемблерами и эмуляторами, которых очень легко зациклить. В самом деле, пусть самомодифицирующийся код вырубает команду безусловного (условного) перехода L1: jmp L1, тогда программа будет выполняется только на живом процессоре, но не под отладчиком. Это, разумеется, дебильный примем, который нетрудно обнаружить, однако, если модифицировать расшифровщик основного тела программы, ситуация окажется весьма неоднозначной и чтобы разобраться в каких случаях процессор игнорирует модификацию кода, а в каких нет, понадобиться ящик пива и мешок травы.

Рисунок 7 азиатов действительно мало, но на них стоит посмотреть! грациозной походкой впорхнуть в переполненный вагон — не каждый русский на это способен!

Но это еще что! Существует возможность модифицировать код, загруженный в кэш первого уровня так, чтобы содержимое кэша данных первого уровня второго ядра осталось неизменным. Замечательное средство для обхода защит, контролирующих целостность кода. Поскольку, напрямую прочитать содержимое кодового кэша невозможно, приходится довольствоваться содержимым кэша данных, надеясь на то, что процессор поддерживает их в согласованном состоянии. Ага, разбежались! Дефекты кэш-контроллера ведут к нарушению когерентности и потому в кодовом кэше оказывается записано одно, а в кэше данных — совсем другое, причем, модифицированные ячейки кэша данных вытесняются в кэш второго уровня (кодовый кэш не вытесняется никогда!), а оттуда уже попадают в оперативную память.

Рисунок 8 на центральных улицах поразительно малолюдно

Комбинирование двух типов ошибок поддержи когерентности позволяет создать весьма устойчивый "голландский гибрид", переживающий вытеснение модифицированных ячеек из кэша данных в оперативную память. Алгоритм атаки выглядит следующим образом:

- ядро L1 модифицирует содержимое кэш-памяти первого уровня D2 таким образом, чтобы ядро L2 об этом ничего не знало;
- ядро L1 передает на модифицированный код выполнение, загружая его в кодовый кэш C1, при этом кэш D1 находится в согласованном состоянии с кэшем C1;
- ядро L2 по прежнему ничего не знает о факте модификации и потому попытка проверки целостности кода показывает, что все нормально — лучшего способа для маскировки зловредного кода, пожалуй, и не придумать, вот только… у злоумышленника нет никаких гарантий, что проверка будет выполняться именно на ядре L2, а не L1, поэтому, необходимо предпринять дополнительные действия;
- ОК, у нас имеется: модифицированный код и данные в C1 и D1, а так же не модифицированные данные в D2 — выполняя модификацию модифицрованных ячеек памяти в D1, хакер возвращает их в исходный вид. C1 об этом ничего не знает и потому не перезагружает кэш. А вот в кэш второго уровня будут вытеснены именно дважды модифицированные данные, то есть фактически не модифицированные, с тщательно вычищенными следами порчи. Как следствие — факт вторжения становится очень трудно обнаружить!

Естественно, после перезагрузки C1 кэша все придется начинать сначала. То есть, перехватить системную функцию таким образом получится только на очень короткое время, впрочем, вполне достаточное для большинства задач, стоящих перед вирусами и червями. Ключевой фрагмент proof-of-concept exploit'a, демонстрирующего технику XMC-атак приведен ниже (см. листинг 2), ну а над законченным оружием возмездия еще предстоит поработать. Кстати говоря, идея использовать XMC-атаки для "ослепления" механизмов проверки целостности кода/данных возникла в ходе разговора с Александром Терешкинм — ведущим исследователем (principal researcher) из фирмы Invisible Things Lab. Да-да! Той самой, в которой работает Жанна Рутковская, поклявшаяся убить любого, кто снова спутает атаку на файл подкачки Windows с Голубой Пилюлей, последняя версия которой поддерживает вложенную виртуализацию, то есть, говоря человеческим языком, исправно работает под запущенным аппаратным эмулятором типа XEN'a, что делает ее обнаружение очень проблематичным, если

вообще возможным. Попутно — Жанна, которую все почему-то называют Джоанной, оказывается никакая не Жанна, но и не Джонна, а Юанна.

Рисунок 9 ночная Малайзия (вид из отеля Crown PlazA – именно так, с ударением на последний слог!)

К сожалению, сама Юанна на конференции не обозначилась и мы с Александром туссовались в компании гордого одиночества на тридцатом этаже отела Crown Plaza Malaysia. Других русскоязычных хакеров на HIBT замечено и не было. И вот пока мы с ним так туссовались, к нам подходили разные люди, чтобы обсудить проблемы виртуализации и атак на процессоры. Одним из них оказался Анатолий Зборальски, работающий в индонезийской конторе Bellua Asia Pacific. Идея обхода Patch Guard'a посредством багистных процессоров — его. Ну а сейчас самое время продемонстрировать код с помощью которого это можно сделать. Кстати, код тоже не мой и его происхождение весьма любопытно. Изначально он задумывался как головоломка в стиле: "угадай, что эта программа делает?", заброшенная на форум WASM'a хакером PROFi (см. пост http://www.wasm.ru/forum/viewtopic.php?id=28983#8), однако, в результате неправильно выбранного смещения регистра ESP, вместо того, чтобы затирать саму себя (как это задумывалась), инструкция POP перезаписывала команду, следующую за командой, вызывающей исключение, в результате чего на процессорах Intel Core 2 мы поимели XMC. Способ модификации команды может быть любым. Не обязательно использовать именно POP — MOV или STOS сработают ничуть не хуже. Исключение так же не обязательно нарушение доступа. Сгодится и "инвалидная команда" (скажем UD2). Главное — чтобы модифицируемая команда располагалась после команды, вызывающей исключение, а модифицирующая команда находилась в пределах досягаемости конвейера и между ними отсутствовали ошибочно предсказанные ветвления, инструкции сериализации типа CPUID и прочий stuff.

```
.00403854:  BC59384000        mov     esp, 000403859
.00403859:  8F442404          pop     d, [esp] [04]
.0040385D:  33C0              xor     eax, eax
.0040385F:  8B00              mov     eax, [eax]
```

Листинг 2 ключевой фрагмент кода, демонстрирующий XMC-атаку (только для 32-битных систем!)

При одновременном выполнении обозначенного кода на двух ядрах, одно ядро "почувствует" модификацию, а другое – нет. Разумеется, при условии, что в процессоре присутствует неисправленный баг. Небольшое расследование, проведенное мыщъхом, показало,

что среди мобильных ЦП багистные встречаются _намного_ чаще, в то время как в последних партиях десктопных процессорах, ошибки уже исправлены.