blue pill/red pill — the matrix has windows longhorn

крис касперски ака мыщъх, no-email

совершенствование stealth-технологий в конечном счете (в середине 2006 года) привело к созданию rootlik'ов принципиально нового типа, которых практически невозможно обнаружить, а тем более остановить. стоит только компьютеру проглотить "голубую пилюлю" как операционная система погружается в виртуальный мир, полностью контролируемый rootkit'ом. прежний, реальный, мир перестает существовать и, чтобы увидеть его, необходимо проглотить "красную пилюлю", над созданием которой бьются лучшие хакерские умы, но… пока не слишком успешно

введение

Не успела Windows Vista/Server Longhorn поступить в продажу, как была зверки взломана польской пани Жанной Рутковской (Joanna Rutkowska), выступившей 21 июля 2006 года на конференции SyScan (Сингапур) и двумя неделями позже — на Black Hat'e (3 августа, США, Лас-Вегас) где она и продемонстрировала rootkit нового поколения, получивший название "Blue Pill" ("Голубая Пилюля") — явный реверанс в сторону "Матрицы" (текст самой презентации вместе с кучей других полезных материалов и утилит можно найти на сайте Жанны: www.invisiblethings.org).

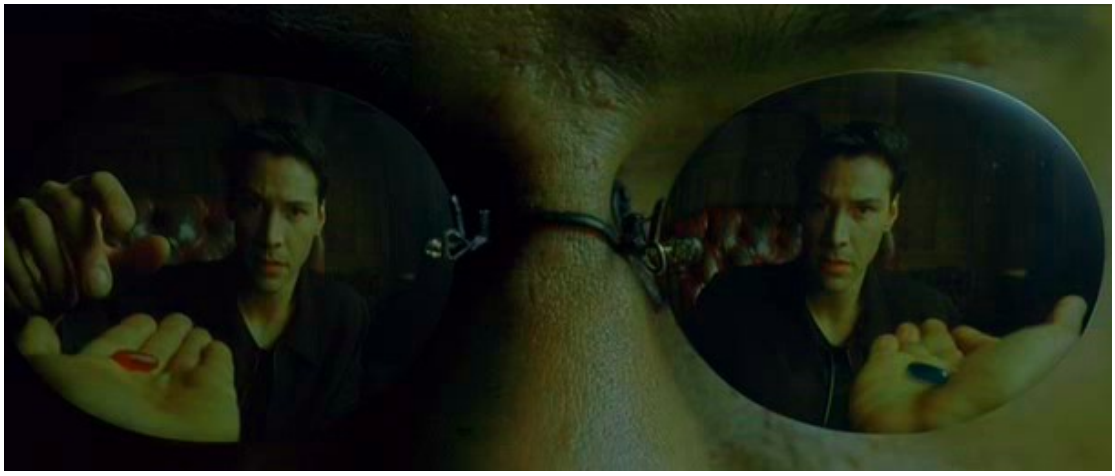

Рисунок 1 выпив голубую пилюлю мы попадем в виртуальный мир, выпив красную – обретаем способность видеть мир таким, какой он есть

Реакция представителей Microsoft оказалась на удивление спокойной: подумаешь, подломали бету! Никто же и не утверждал, что взломать Висту/Longhorn невозможно! Идет нормальный процесс "обкатки" системы и чем больше ошибок будет выявлено на стадии бета тестирования, тем меньше их окажется в финальной версии продукта. Однако, Vista RC1, выпущенная двумя месяцами позже, не претерпела никаких изменений и по-прежнему остается уязвимой. Microsoft, проанализировала ситуацию и, вместо того, чтобы заткнуть дыру, сделала вид, что никакой дыры здесь нет! (см. выступление одного из сотрудников Microsoft: http://blogs.msdn.com/windowsvistasecurity/archive/2006/08/07/691441.aspx).

Мол, с правами администраторами (а "Голубая Пилюля" требует их) еще и не такое возможно! А, что, собственно говоря, с ними, возможно?! Загрузить неподписанный драйвер либо каким бы то ни было другим _легальным_ способом проникнуть на уровень ядра — нельзя, что доставляет множество проблем как самим администраторам, так и разработчикам. Во имя ее величества Безопасности с этим можно было бы и смириться, если бы Microsoft заткнула _все_ лазейки, а так получается, что нас вынуждают поступиться частью свобод и удобств, предлагая в замен… ничего! Где логика?! Как всегда, логика на стороне Microsoft, преуспевшей только в одном — в продвижении своих глюкодромов на рынок.

Рисунок 2 Жанна Рутковская на конференции Black Hat

>>> врезка Жанна или Джоанна

Польское имя Жанны, записанное латиницей, многие переводчики переводят буквально — "Джоанна" (Joanna), забыв о том, что существует специальные правила транслитерации (и даже ГОСТ!), которые никто не отменял. Смотрели фильм "Joanna d'Arc"? Нет?! Значит, вы определенно не поляк, поскольку в американский прокат фильм вышел под названием "The Messenger: The Story of **Joan** of Arc", в то время как на французском имя главной героини записывается как "**Jeanne** d'Arc", то есть Joanna — эта Жанна, записанная на польский манер. И никакая она не "Джоана". Эх, из чего же только не делают переводчиков ;(

Рисунок 3 Жанна д'Арк (слева) и Жанна Рутковская (справа)

Что же касается выбора между "Рустковска" (Rutkowska) или "Рутковская", то тут правила транслитерации допускают определенную вольность: сохранить оригинальное написание — Рутковска (поимев при этом проблемы склонения) или переложить его на русский манер, добавив окончание "-ая" – Рутковская. Это уже как кому больше нравится. Подробнее о переводе польских имен можно прочитать на: http://www.spelling.spb.ru/rosenthal/alpha/r149.htm и http://www.proz.com/kudoz/593898.

внутри "blue pill"

"Голубая Пилюля" базируется на двух основных концепциях — **обходе цифровой подписи драйверов** (обязательной в x86-64 редакциях Windows, начиная с Vista Beta 2 build 5384) и **установке гипервизора** (hyper-visor), использующего технологии аппаратной виртуализации AMD Pacifica/Intel Vanderpool, позволяющие запускать операционную систему на эмуляторе, контролирующим все "интересные" события. В грубом приближении это можно проиллюстрировать на примере 80836 ЦП, поддерживающего режим "виртуального 8086" (он же V86), обеспечивающего одновременную работу нескольких MS-DOS сессий. Ну вот, а теперь, появился режим "виртуального 386+", причем правильно спроектированный гипервизор (так же называемый *Монитором Виртуальных Машин — Virtual Machine Monitor*, VMM) не позволяет гостевой системе определить: исполняется ли она на "живом" процессоре или нет.

Технология обхода цифровой подписи драйверов актуальна _только_ для 64-битных версий Windows (в 32-битных загрузить неподписанный драйвер можно и так), а механизмы аппаратной виртуализации _никак_ не связаны с конкретной осью и замечательно работают на Linux, BSD, Mac OS и т. д. (разумеется, при поддержке со стороны процессора). Любая ось, позволяющая хакеру пробиться на уровень ядра, может быть атакована.

Таким образом, "Голубая Пилюля" состоит из _двух_ компонентов, лишь один из которых по-настоящему "голубой". Он-то и отвечает за погружение операционной системы в виртуальный мир. Другой компонент — независимая "затравка", специально спроектированная для обхода защиты 64-битных версий Windows и забрасывающая (точнее, сбрасывающая) на ядерный уровень любую полезную нагрузку, в роли которой вполне может выступать и обычный rootkit.

Рисунок 4 два компонента "Голубой Пилюли" — один пробивается на уровень ядра, другой погружает ось в виртуальный мир

обход цифровой подписи

Механизм обхода цифровой подписи, предложенный Жанной, основан на ***модификации файла подкачки на секторном уровне*** (назовем его **page-file attack**). Сама атака состоит из шести этапов:

I. находим в каталоге /WINNT/System32/Drives редко используемый драйвер (например, NULL.SYS), считываем его содержимое и выделяем уникальную последовательность байт (сигнатуру), позволяющую однозначно отождествить его. сигнатура должна находится в ветке IRP_MJ_DEVICE_CONTROL процедуры DeviceDispatcher (адрес последний легко определить путем дизассемблирования драйвера), причем сигнатура не имеет права пересекать границы страницы (в файле подкачки соседние страницы не всегда оказываются рядом друг с другом), то есть должно выполняться условие: **(virtual_address_of_signature % 1000h) + sizeof(virtual_address_of_signature) < 1000h**;

II. запускаем программу "memory-eater", "съедающую" всю доступную памятью (например, путем вызова API-функции VirtualAlloc) и вынуждающую операционную систему свопиться на диск, вытесняя в том числе и ядерные компоненты (***внимание!*** *если параметр "DisablePagingExecutive", находящийся в следующей ветке реестра HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\MemoryManagement, равен 1 (по умолчанию 0), ядерные компоненты вытесняться не будут! изменения вступают в силу только после перезагрузки);*

III. открываем устройство "\\.\C:" (логический диск) или "\\.\PHYSICALDRIVE0" (физический диск) API-функцией CreateFile и читаем/пишем на секторном уровне API-функциями ReadFile/WriteFile соответственно. Так же можно воспользоваться хорошо

документированным интерфейсом SPTI, позволяющим передавать диску SCSI-команды через API-функцию DeviceIoControl — за что отвечает IOCTL-код IOCTL_SCSI_PASS_THROUGH_DIRECT (4D014h). ось автоматически транслирует pseudo-SCSI команды в native-команды конкретного накопителя (например, IDE HDD). два недокументированных IOCTL-кода IOCTL_IDE_PASS_THROUGH и SCSIOP_ATA_PASSTHROUGH позволяют передать IDE-накопителям native-ATA команды, что дает над ними неограниченную власть, правда ухудшает совместимость (что если у жертвы установлен SCSI-диск?!). все вышеупомянутые интерфейсы требует администраторских прав, которые не всегда у нас есть, однако, ASPI-интерфейс, разработанный компанией Adaptec, не отягощен такими ограничениями!!! и хотя корректно установленный ASPI-драйвер (кстати говоря, исключенный из штатной поставки Windows много лет тому назад) дает доступ _только_ к ATAPI-устройствам (таким как CD, DVD), достаточно часто в этот список попадают и жесткие диски, то есть атаку (теоретически) можно реализовать даже без администраторских прав!!! Если ASPI-драйвер на целевой машине не установлен, rootkit должен либо установить его самостоятельно (кстати, сам драйвер подписан и бесплатен), либо поискать другие драйвера, установленные приложениями, работающими с HDD-, CD- или DVD-дисками на низком уровне (дисковые редакторы, копировщики защищенных дисков, программы для прожига CD/DVD). многие из них позволяют манипулировать жесткими дисками, не требуя прав администратора (подробнее обо всем этом можно прочитать в моей книге "техника защиты лазерных дисков от копирования", черновую версию которой можно найти на ftp://nezumi.org.ru, только помните, что сервер работает не всегда);

IV. дождавшись выгрузки драйвера на диск (*момент которого легко определить эвристическим путем — при исчерпании оперативной памяти, система вытеснит часть страниц только что выделенных VirtualAlloc, скачкообразно увеличивая количество доступной физической памяти, объем которой легко определить API-функцией VirtualQuery*), начинаем прочесывать диск на секторном уровне в поисках ранее обозначенной сигнатуры драйвера-жертвы;

V. вычисляем адрес IRP_MJ_DEVICE_CONTROL и записываем поверх него shell-код, отключающий проверку цифровой подписи (что в дальнейшем позволит нам беспрепятственно загружать неподписанные драйвера), либо загружаем весь необходимый код на ядерный уровень самостоятельно;

VI. вызываем API-функцию CreateFile, передавая ей имя хакнутого драйвера (в данном случае NULL.SYS) и… операционная система тут же считывает модифицированные страницы с диска, вызывает IRP_MJ_DEVICE_CONTROL, передавая shell-коду управление, ну а дальше, как говорится, дело техники!

Успешно осуществив атаку, Жанна (как и положено "белому" хакеру) тут же предложила несколько контрмер: самое простое, но не самое умное, что может сделать Microsoft – это заблокировать выгрузку ядерных компонентов на диск (тем более, что все необходимые ингредиенты у нее _уже_ есть, достаточно только убрать из реестра ключ DisablePagingExecutive, пожизненно установив его значение в "1"). В результате мы потеряем некоторое количество физической памяти (по подсчетам Жанны около 80 МБайт, а по подсчетам мыщъх'а даже меньше: совокупный объем драйверов на мыщъх'иной машине составляет 30 Мбайт, и мыщъх не думает, чтобы на висте их размер был бы _сильно_ больше), гарантируя при этом, что никакой драйвер не будет скомпрометирован.

Учитывая, что сама виста требует не менее 1 Гбайта RAM (а для реальной работы понадобиться как минимум два), потеря 30-80 Мбайт навряд ли покажется значительной, однако, можно пойти другим путем, подсчитывая контрольную сумму каждой страницы перед вытеснением ее на диск и проверяя ее при загрузке. Но, поскольку, контрольные суммы надо где-то хранить (причем не на диске, а в невытесняемой оперативной памяти), мы не получаем никакого выигрыша, впустую расходуя процессорное время. Можно, конечно, шифровать страницы каким-нибудь высокоскоростным криптоалгоритом, храня в памяти всего лишь ключ, случайным образом генерируемый при загрузке, но это уже через чур. К тому же существует возможность модификации самого файла ядра операционной системы, отключающая все защитные механизмы и тут же инициирующая перезагрузку, против которой предложенные защитные меры бессильны!

Атака на файл подкачки — это действительно прорыв, который Microsoft закроет не скоро (во всяком случае в Vista RC1 еще не закрыла). Важно отметить, что все вышесказанное

относится _исключительно_ к 64-битной версии Windows, поскольку только в ней администраторы лишены прав загружать неподписанные драйвера.

погружение в виртуальный мир

Механизмы аппаратной виртуализации (известные под кодовым именем "**Pacifica**") реализованы во всех процессорах семейства Athlon 64/Turion 64, выпущенных фирмой AMD после мая 2006 года, так же планируется поддержка виртуализации в Opteron'e. Но это все x86-64 платформы, которые нам не сильно интересны, поскольку их рыночная доля крайне мала. AMD не смогла справится с виртуализацией x86, сославшись на сложность реализации и непригодность этой архитектуры для подобных целей (читай: кишка тонка), а вот Intel _смогла_ за что ей честь и хвала!

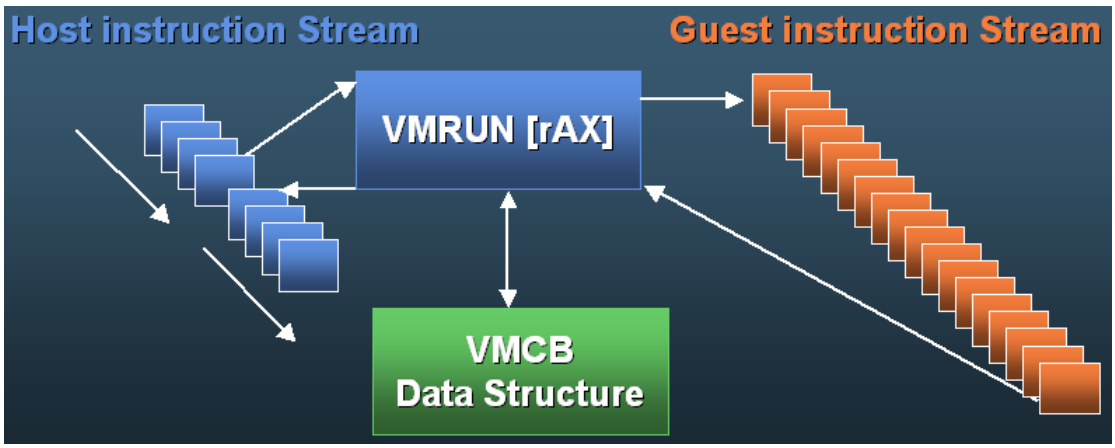

Рисунок 5 технология аппаратной виртуализации Pacifica, реализованная в процессорах фирмы AMD: при выполнении машинной команды VMRUM, процессор создает новую SVM (Secure Virtual Machine – Защищенную Виртуальную Машину), управляемую гипервизором (hyper-visor) и структурой данных под названием Virtual Memory Control Block (Блок Управления Виртуальной Памятью) или, сокращенно VMCB

Технология с кодовым именем "**Vanderpool**", воплощенная в процессорах Intel Pentium 4 6x2, Pentium D 9xx, Xeon 7xxx, Core Duo, Core 2 Duo и перекочевавшая туда из Itanium'a (IA64), где она была известна под именем "**Silvervale**", теперь во избежании путаницы объедена с последней в обезличивающую официальную аббревиатуру **VT-X** (Virtualization Technology X — X-Технология Виртуализации).

VT-X существенно отличается от Pacific'и, но по сути предоставляет те же самые возможности, а именно — запуск гипервизора, захватывающего контроль над операционной системой и переводящего ее в "гостевой" виртуальный режим, который с _ее_ точки зрения ничем не отличается от "реального".

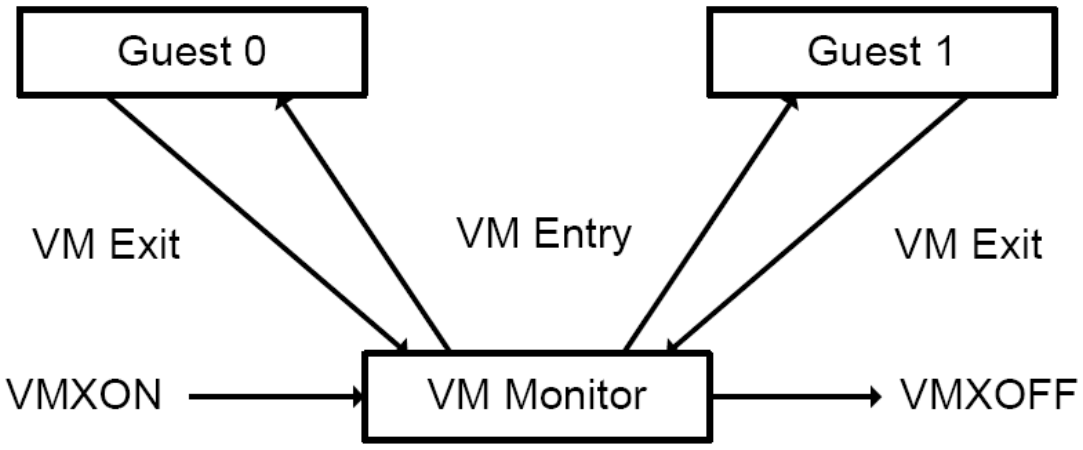

Рисунок 6 технология аппаратной виртуальной виртуализации Vanderpool/Silvervale, реализованная в процессорах фирмы Intel – монитор виртуальных машин (VM Monitor),

запускаемый машинной командой VMXON, создает столько "виртуальных" процессоров, сколько угодно

Гипервизор (в случае VT-X называемый Монитором Виртуальных Машин — "Virtual Machine Monitor" или, сокращенно, VMM) передает гостевой операционной системе управление и получает его назад при наступлении определенных "интересных" событий: возбуждении аппаратного/программного прерывания, обращения к служебным регистрам процессора и т. д.

Гипервизор не может непосредственно перехватывать API-функции гостевой операционной системы, но способен следить за обращениями к портам ввода/вывода, или самостоятельно взаимодействовать с оборудованием в обход оси. Косвенный перехват API-функций осуществляется путем установки аппаратных точек останова на исполнение (которых, увы, всего четыре) с последующем пресечением попыток гостевой системы "подсмотреть" истинное содержимое регистров DRx.

Подробное описании технологии Pacifica содержится в техническом руководстве "AMD64 Architecture Programmer's Manual Vol. 2: System Programming", выложенным в общий доступ на http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/24593.pdf.

В отличие от AMD, Intel выпустила отдельный документ, а не стала валить все в одну большую кучу: ftp://download.intel.com/technology/computing/vptech/C97063-002.pdf (и хотя без руководства по системному программированию при написании собственного монитора виртуальных машин все равно не обойтись, его основные положения большинству хакеров уже известны. конечно, для тех, кто еще не знаком с защищенным режимом, создание "Голубой Пилюли" окажется настоящим испытанием, но… это уже их проблемы).

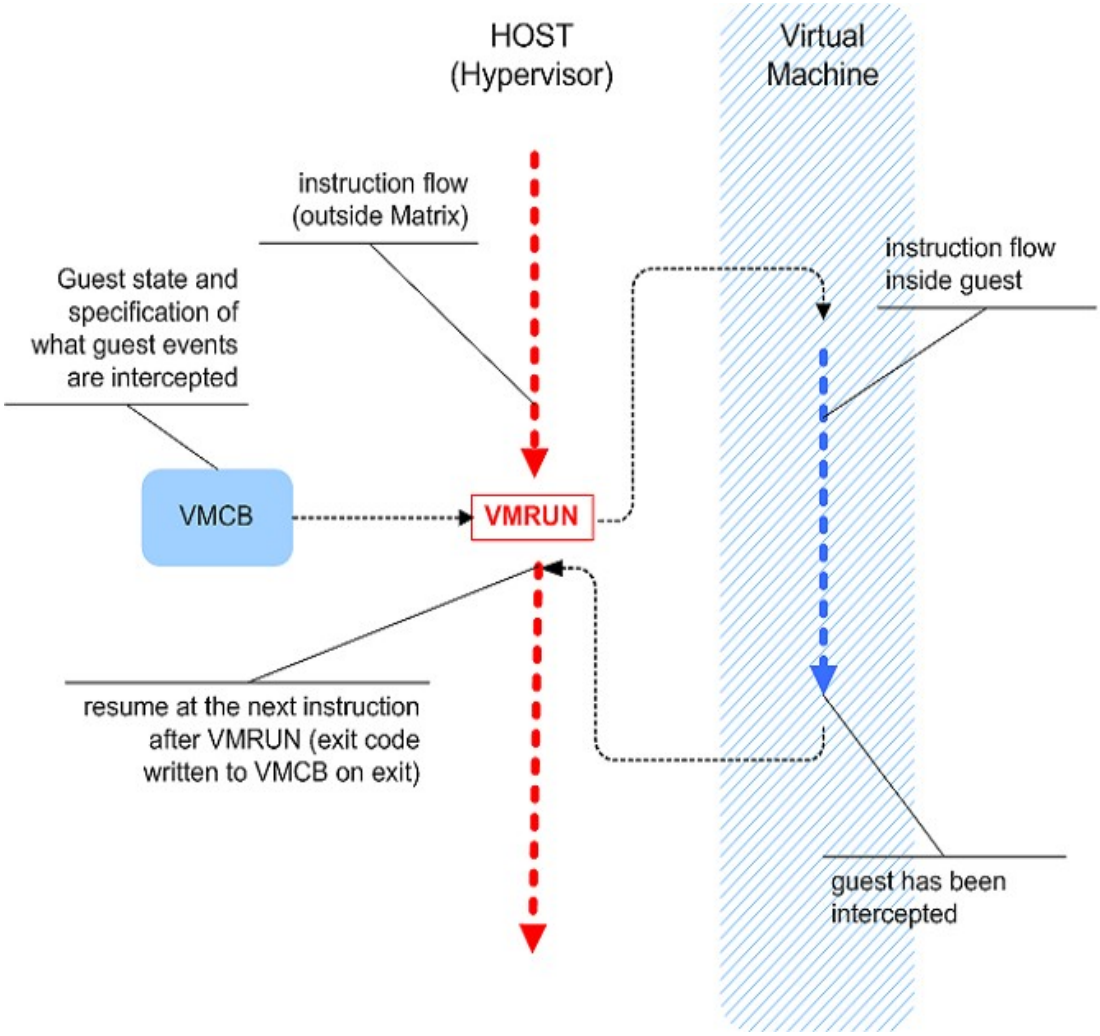

Рисунок 7 выполнение машинной команды VMRUN на процессорах AMD x86-64 приводит к захвату контроля над операционной системой

Рассмотрим устройство "Голубой Пилюли", заточенной Жанной специально под процессоры AMD x86-64 (на процессорах Intel все будет точно так же, только немного по-другому). Проглотив "Голубую Пилюлю" (CALL bluepill) ядро передает управление главной функции rootkit'a (условно обозначенной "PROC blue-pill"), которая создает виртуальную машину, подготавливает все необходимые структуры данных и вызывает машинную команду "VMRUM" (на Intel процессорах это будет "VMXON"), устанавливающую гипервизор/VMM, погружающий операционную систему в виртуальный мир и натягивающий на ее глаза "очки", перехватывающие _все_ каналы взаимодействия последней с "внешним миром" — портами ввода/вывода, служебными регистрами процессора, физической оперативной памятью и т. д.

Гипервизор будет "подсовывать" операционной системе только ту информацию, которую ей позволено видеть, надежно скрывая свое присутствие от ее глаз.

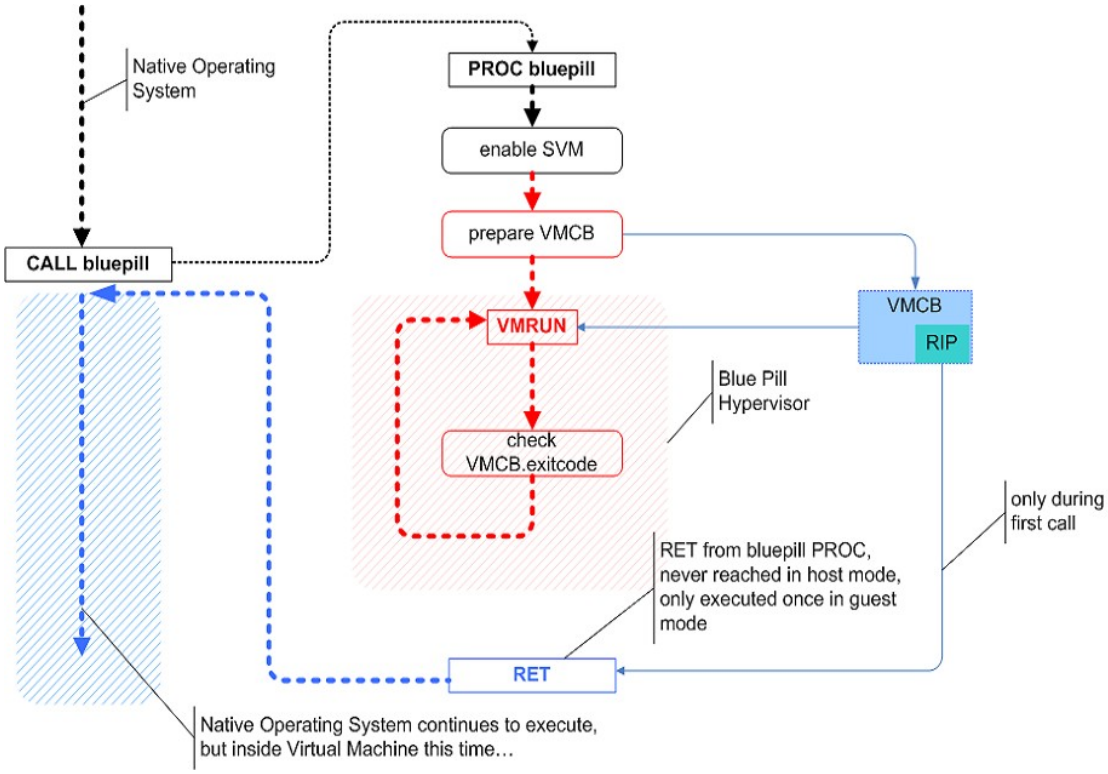

Рисунок 8 упрощенная блок-схема "Голубой Пилюли", разработанная Жанной

Гипервизор/VMM представляет достаточно сложную программу, которую не так-то просто написать и еще сложнее отладить, но ведь нам так хочется реализовать свою собственную "Голубую Пилюлю", не правда ли? Сама Жанна, кстати, так и не справилась с этой задачей, о чем открыто признается в своем блоге в заметке "The Blue Pill Hype" ("Слухи вокруг Голубой Пилюли"): theinvisiblethings.blogspot.com/2006/07/blue-pill-hype.html, фрагмент из которой мыщъх цитирует ниже:

"All the hype started from this article in eWeek by Ryan Naraine (http://www.eweek.com/article2/0,1895,1983037,00.asp). The article is mostly accurate, despite one detail - the tile, which is a little misleading... It suggests that I already implemented "a prototype of Blue Pill which creates 100% undetectable malware", which is not true. Should this be true, I would not call my implementation "a prototype", which suggests some early stage of product… The Blue Pill prototype I currently have is not yet complete, but this is not that important, because having successfully moved the OS into a virtual machine, implementing all the other features is just a matter of following the Pacifica specification"

(*Все началось со статьи Нарьяна Рьяна из eWeek. Статья вполне адекватная, за исключением одной маленькой детали, вводящей читателей в заблуждения. В статье утверждается, что я уже реализовала "прототип Голубой Пилюли, создающую на 100% не обнаруживаемую мальварь", что неправда. Если бы это было правдой, я бы не стала называть свою реализацию "прототипом", подразумевающего наличие опытного продукта… Прототип "Голубой Пилюли", имеющийся у меня в настоящий момент, еще не полностью реализован, но это неважно, поскольку создание виртуальной машины для запуска операционной системы и*

реализация всех остальных фич, это всего лишь вопрос следования спецификациям на Pacific'у — перевод мыщъх'а).

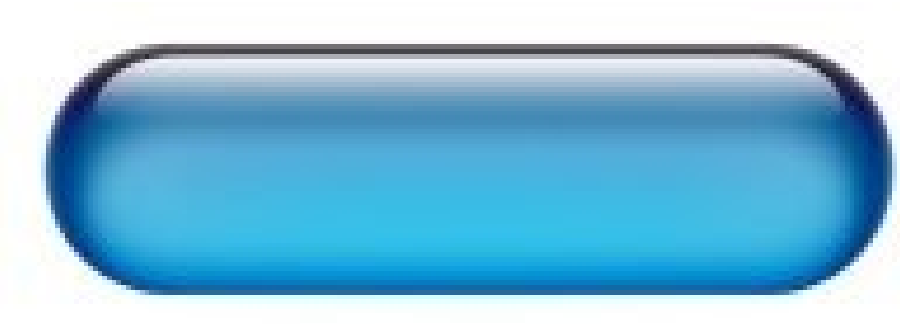

Рисунок 9 проглотив "Голубую Пилюлю" операционная система проваливается в виртуальный мир

Чтобы приблизить себя к цели на несколько световых лет и не повторять уже сделанное, мы можем "выдрать" ядро из готового эмулятора и слегка доработать его "напильником" для наших хакерских целей, дописав сравнительно небольшую порцию кода. Естественно, это должен быть эмулятор, распространяющийся в открытых исходных текстах на бесплатной основе, например, XEN, поддерживающий обе архитектуры (Pacifica + Vanderpool одновременно) и абстрагирующий нас от конкретных аппаратных особенностей, хотя и не избавляющий реализовывать отдельные версии rootkit'a для x86 и x86-86 платформ. Архив с исходными текстами третьей версии XEN'a (текущей стабильной версии на данный момент) можно слить с http://www.cl.cam.ac.uk/Research/SRG/netos/xen/downloads/xen-3.0-testing-src.tgz, а сам сайт находится по адресу: http://www.cl.cam.ac.uk/Research/SRG/netos/xen.

В частности, ядро, отвечающее за поддержку x86 процессоров Intel, сосредоточено в файле /xen-3.0-testing/xen/include/asm-x86/hvm/vmx/vmx.c

красная пилюля

В "Матрице" чтобы увидеть реальный мир было достаточно принять красную пилюлю. А как на счет операционной системы? Может ли она каким-нибудь образом определить, что работает под виртуальным эмулятором? Программу, позволяющую обнаружить присутствие эмулятора, обычно называют "Красной Пилюлей" и такие пилюли находят самое широкое применение как в хакерской, так и администраторской среде (первые используют их для детектирования VM Ware или других _программных_ эмуляторов, вторые — для обнаружения rootkit'ов). С аппаратной виртуализацией все намного сложнее…

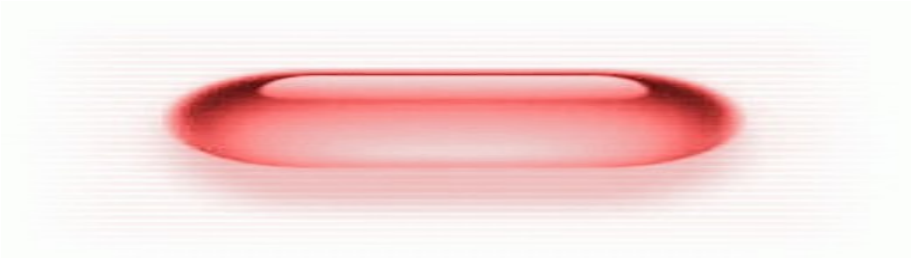

Рисунок 10 приняв "Красную Пилюлю" операционная система обнаруживает эмулятор (если он есть)

Первое, что приходит на ум — попытаться вызывать VMCALL/VMXON и, если мы находимся под эмулятором (процессор не поддерживает аппаратной виртуализации или она отключена в BIOS), вызов проваливается. Написать такую программу — плевое дело, имея готовый скелет драйвера, это можно сделать меньше, чем за минуту, вот только… как отличить ситуацию "процессор не поддерживает" от "мы под эмулятором"?! Существует только один путь — отодрать от кремния радиатор и физически посмотреть на его маркировку, сличив ее с "показаниями" машинной команды CPUID (эмулятор, если только он не лось, возвратит подложные данные, убеждая нас в том, что поддержка аппаратной виртуализации на этом

процессоре отсутствует). Но какой процент пользователей отважится на столь радикальный шаг?! Отодрать радиатор они, положим, отдерут, но вот приставят ли обратно?

Тем более, что путем некоторых ухищрений гипервизор может _эмулировать_ выполнение команд VMCALL/VMXON, реализуя вложенную виртуализацию. Это как бы один виртуальный мир в другом и уровень вложенности, в принципе, неограничен. Естественно, с каждой проглоченной "Голубой Пилюлей" мы будем терять производительность. Уже в первом виртуальном мире быстродействие операционной системы сокращается на десятки процентов (никто никогда и не утверждал, что аппаратная виртуализация дает 100% КПД).

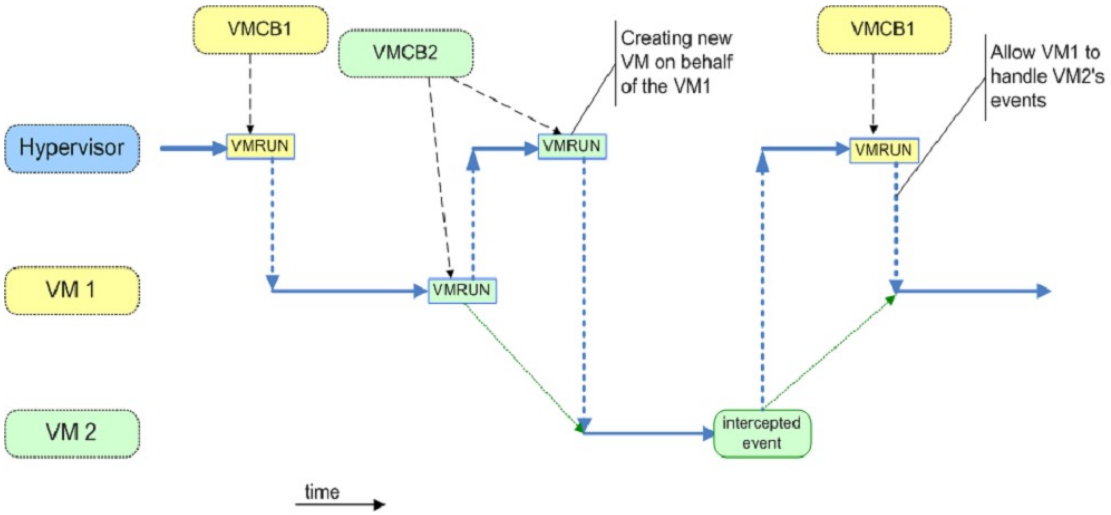

Рисунок 11 виртуальные миры, вложенные друг в друга как матрешки

Стоп! Стоп! Стоп! Производительность! Вот она — нить Ариадны! Замеряя время выполнения машинных инструкций, перехваченных гипервизором (и, в первую очередь, инструкции RDMSR EFER, читающий регистр EFER, 12-бит которого указывает на присутствие гипервизора), мы легко заметим, что в виртуальном мире они выполняются намного дольше, чем в реальном.

Вся проблема в том, что нам _нечем_ мерять время их исполнения. Команда RDTSC (читающая значение регистра, увеличивающегося с каждым тактом процессора) отпадает сразу и однозначно, поскольку контролируется гипервизором, корректирующим ее показания так, как будто бы мы находится в реальном мире. Для упрощения решения этой задачи, процессор поддерживает специальную "калибровочную" переменную VMCB.TSC_OFFSET, указывающую сколько экстра тактов ему следует вычитать при выполнении команды RDTSC. Так что корректировка показаний RDTSC происходит автоматически даже без вмешательства эмулятора.

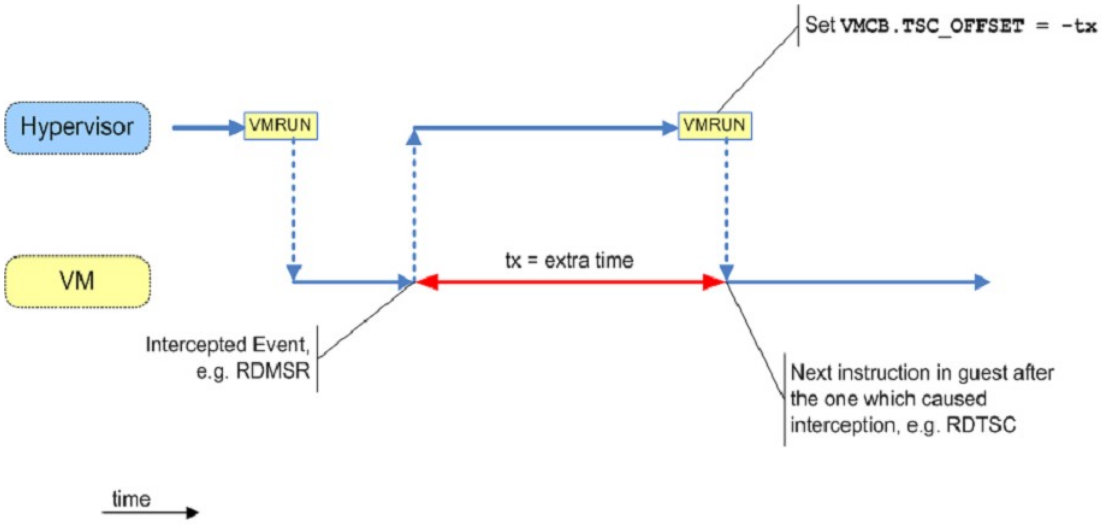

Рисунок 12 корректировка времени выполнения инструкций, перехваченных гипервизором

Теоретически можно воспользоваться часами реального времени, атомными часами, доступными через сеть или даже внешним (по отношению к компьютеру) хронометром. Гипервизор способен "подкручивать" часы реального времени (правда, при этом они начнут отставать, что пользователь сможет заметить), перехватывать и корректировать сетевой трафик, если в нем присутствует атомное время (хм, а не надорвется он это делать?!), но на хронометр, сжимаемый ладонью пользователя он воздействовать не в состоянии (если, конечно, пользователь сам не находится в виртуальном мире ;-)

Рисунок 13 попытка обнаружить присутствие гипервизора с хронометром в руках так же обречена на провал

Отличная "Красная Пилюля" получилась, нечего сказать... "Сейчас будет запущена тестовая программа, пожалуйста воспользуйтесь своими электронными часами и определите время ее выполнения, для уменьшения погрешности продолжительность теста составит около 60 секунд". Ну и кто это будет делать?! А с _чем_ сравнивать полученные показания мы подумали?! Чтобы обнаружить присутствие гипервизора необходимо иметь эталонный компьютер с таким же точно процессором, погруженным в термостат, поскольку многие процессоры меняют свою тактовую частоту в зависимости от температуры. Но даже в этом случае, гипервизор может распознать команду RDMSR EFER, выполняющуюся в цикле, и пропускать некоторые итерации для достижения идентичного времени выполнения.

Некоторые горячие головы предлагают читать содержимое оперативной памяти через DMA, записывая ее на диск, а потом искать в этой куче дерьма следы присутствия гипервизора. Ну, во-первых, они _очень_ сильно переоценивают возможности контроллеров DMA по чтению памяти, во-вторых, гипервизор, контролируя обращения к портам ввода/вывода, легко это отследит, а, в-третьих, даже если и не отследит — что искать-то?! При условии, что сигнатура "Голубой Пилюли" неизвестна (или она построена на полиморфной основе), мы ни за что не различим ее!

вместо заключения

Так что же, выходит, "Красной Пилюли" не существует?! А это еще как сказать... Ведь и "Голубой Пилюли" не существует тоже. Во всяком случае пока… Да, технологии аппаратной виртуализации позволяют создать "Голубую Пилюлю", которую _никак_ нельзя обнаружить. Теоретически. Практически же все упирается в сложность реализации, так что не стоит

обсуждать конструкцию сферических коней в вакууме, а лучше решать проблемы по мере их возникновения.