Прошивки стандарта UEFI: преимущества и опасности

Семинар ОСО ИММ УрО РАН

25.01.2012

Аннотация

UEFI – это новый стандартный интерфейс взаимодействия OS и прошивок системных плат для вычислительных систем, относяшихся к классу персональных компьютеров. Он обеспечивает более богатую и гибкую среду разработчикам оборудования, снимая традиционные для PC-AT BIOS ограничения: на объём одновременно доступной всем сервисам времени загрузки памяти, на архитектуру процессора, на возможность подключения устройств через неоднородные шины, на объёмы поддерживаемой энергонезависимой памяти (SSD или HDD) и прочие. Однако, эти новые возможности являются новыми возможностями и для создателей вредоносного ПО, противодействовать которым новые версии UEFI предлагают при помощи криптографических механизмов Secure Boot. Надёжность и практичность которых вызывает определённые сомнения. А заложенная в них возможность привязки оборудования к конкретным версиям конкретных OS представляет опасность для сообщества разработчиков свободного ПО, лежащего в основе Национальной Программной Платформы РФ.

Зачем нужен UEFI?

- Поддержка загрузки с дисков большого объёма, больше 2TB. В принципе, RAID уже сейчас позволяет создавать большие дисковые пространства, но интерфейс PC-AT BIOS не позволяет из загрузчика обращаться к далёким секторам в таких конфигурациях.
- Современные BIOS-ы таковы, что производителям оборудования (OEM) очень сложно дописывать к ним свои модули. А если у них и получается это сделать, операционным системам сложно этими модулями воспользоваться, потому что это не предусмотрено в интерфейсе.
  - Иногда хочется новые шины добавить.
  - Иногда – драйвер инициализации к экзотической аппаратной экзотике (эта экзотика может добавляться пользователем самостоятельно – новый SoundBlaster).
- PC-AT BIOS расчитан на использование только с x86-процессорами, работающими в 16-битовом режиме. Во-первых, и Intel, и AMD хочется уже избавиться от этого режима работы процессора, потому что сложность декодеров инструкций, несмотря на заверения Intel, мешает повышать энергоэффективность. Во-вторых, производители процессоров других архитектур желают попасть на рынок PC (ARM, Intel с Itanium-ами(?)).
- Необходимость уметь работать в 16-битовом режиме ограничивает творческие амбиции производителей оборудования: им кажется, что полноценный GUI с мышкой и высоком разрешении со стрелочными индикаторами параметров работы системы наделяют продукт дополнительный привлекательностью. А объём адресуемой памяти ограничен: изображение размером 1600x900 и 32-битовыми пикселями требует для обработки более 5MiB адресного пространства.

- Хочется уметь загружаться через новые виды контроллеров, например, NVHCI
- И т.д.

UEFI изначально разрабатывался Intel для систем на базе Itanium, потом к проекту присоединилась Apple и начала использовать EFI в своих основанных на x86 системах. Затем, на стандарт начали влиять многие другие производители, в том числе и Microsoft. При разработке пытались обеспечить по максимуму совместимость с существующими уже технологиями. UEFI утилизирует...

- Стандартные таблицы BIOS.
- ACPI.
- Прежние способы разметки дисков.
- Протоколы загрузки системы через сеть (PXE).

Intel утверждает, что среди OEM есть некие наработанные практики насчёт того, как надо писать BIOS, и в UEFI эти практики старались так же сохранить.

В результате получилось нечто вроде операционной системы с драйверами, оболочкой и приложениями. Способное работать на следующих архитектурах.

- ARM
- IA32
- IA64
- x64
- EFI Byte Code (зачем нужно всё остальное - не особо ясно)

Писать приложения и драйверы можно на Си/Си++ при помощи SDK с открытыми кодами, ведущей организацией для которого является Intel[1].

Есть подробное описание соглашений на условия выполнения кода на Си: установлены ли трансляции виртуальных адресов, если установлены, то какие; как можно портить регистры, а как нельзя, и так далее.

Ядро UEFI состоит из сервисов двух видов: Boot и Runtime. Последние, в отличии от первых, доступны во время работы OS: таймеры, переменные, через которые программируется поведение UEFI-совместимой прошивки, и ещё кое-что. Переменные особо важны, потому что в переменных вида BootABCD записываются указания для Boot Manager-а о том, что и откуда можно загружать.

Влияние Microsoft на дизайн всей системы было сильным, поэтому всё общение с этим многообразием происходит через GUID-ы. А точка входа для работы с некой службой - это Протокол. Протокол - это, на самом деле, обычный интерфейс: таблица указателей на функции. Можно сказать, что упрощённый COM-объект. Они разные бывают.

- `EFI_LOADED_IMAGE_PROTOCOL` – через это загружаемые драйверы для оборудования, которые UEFI должно уметь загружать не только из Option ROM'ов на железяках, могут узнавать про себя интересное.
- `EFI_SERIAL_IO_PROTOCOL`
- `EFI_EBC_PROTOCOL`

[1] http://www.intel.com/content/www/us/en/architecture-and-technology/unified-extensible-firmware-interface/efi-homepage-general-technology.html

- `EFI_FILE_PROTOCOL` – как всегда, у Microsoft всё «согласовано» и образы загружаются не через этот протокол, а при помощи специальной функции `LoadImage()`.
- `EFI_USB_IO_PROTOCOL`
- `EFI_TCP4_PROTOCOL` – опционально может быть поддержка и IPv6.
- ...

Это всё нужно для драйверов и приложений. Они могут открывать эти протоколы и общаться с устройствами и китайской разведкой (именно на производстве в Китае осуществляется запись прошивок).

Основным среди Boot-сервисов является Boot Manager, который загружает. Он загружает всё: и драйверы, и загрузчики OS, и приложения. Программируется он в достаточно широких диапазонах. Например, можно запрограммировать загрузку приложение через нажатие горячих клавиш. Видимо, предполагаются сценарии использования подобные таким: «Нажмите F2, чтобы посмотреть температуру процессора; нажмите F4, чтобы проверить электронную почту», или чтобы выбирать для загрузки нужную OS (тут неясность).

Загружать можно определённый файл. Теоретически, загружать можно с любого устройства и любой FS, лишь бы устройство поддерживало `EFI_BLOCK_IO_PROTOCOL`, а для файловой системы существовал соответствующий контроллер. Но, скорее всего, загрузка будет идти со специального UEFI-раздела, который поддерживает протокол `EFI_SIMPLE_FILE_SYSTEM_PROTOCOL` (простые файловые системы: FAT, ISO9660) по фиксированному пути.

Для извлекаемых media главное, чтобы они поддерживали `BLOCK_IO`, через который к ним можно докрутить `LOAD_FILE` или `SIMPLE_FILE_SYSTEM`, а загрузка будет вестись для файла по пути: `EFI\BOOT\BOOT<архитектура>.EFI`.

Из файловых систем можно загружать и драйверы, и приложения. Их можно загружать и из памяти устройств, однако, метод такой загрузки является platform specific, и отдаётся разработчикам системы: они должны реализовать для работы с памятью устройства `BLOCK_IO_PROTOCOL`.

Boot Manager умеет работать с MBR таблицами файловых систем. И он знает, что раздел с меткой 0xEF является системным UEFI-разделом. В основном, туда записываются загрузчики, драйверы от OEM и прочее.

Если MBR имеет специальный формат, то UEFI считает, что имеет дело с диском, размеченным в соответствии с GPT (GUID Partition Table).

В MBR должна быть одна запись о разделе максимальной для MBR величины, с 1-го по 0xffffffff сектор, служащая как бы защитой от софта, который GPT не понимает. Поэтому, это всё называется Protective.

После нулевого сектора MBR идёт сектор GPT Partition Header, в котором описывается массив описателей разделов диска, которых может быть сколь угодно много; чем, например, пользуется Google в Chromebook-ах.

Тут ничего сложного. Каждый раздел определяется GUID-ом для файловой системы (для ext4 он уже есть), далее GUID самого раздела (GUID – это 16-ти байтовое значение), далее 64-битовыми LBA адресами указан отрезок дискового пространства, занимаемый разделом; указываются также флаги, метка длиной 72 байта; выделены резервные поля.

Единственная сложность здесь в том, что надо раздобыть GUID. Если мы пишем OS, то надо его непротиворечиво придумать, а как? Метод не стандартизирован.

Таким образом, загружаться можно с разнообразных носителей. У пользователя всегда есть UEFI System Partition, куда можно всегда что-нибудь записать.

То есть, если OS уязвимая и не способна защитить структуру файловых систем, то разместить в системе нежелательный код становится просто и сделать это можно

стандартными средствами, не обязательно взламывать BIOS и внедрять в прошивку некий код, редактируя её образ при помощи редактора байтов.

Тут встаёт вопрос: а способен ли Windows защитить свои загрузочные записи? На который в Microsoft твёрдо отвечают: НЕТ! При чём настолько твёрдо, что заставляют Марка Руссиновича написать книжку про страшные Российские Буткиты, используемые страшными Исламскими Террористами, на которых работают страшные Чеченские Убийцы из Лондона. Русский код способен внедряться куда угодно, даже в ROM медицинского оборудования, вне зависимости от архитектуры процессора и типа операционной системы. После прочтения данного опуса под названием День Ноль, возникает вопрос: Who is mister Russinovich? Как он, высококвалифицированный программист, может писать такое?

Видимо, Microsoft, будет нагнетать сомнения и дальше различными способами.

И призывать на этой волне Intel и прочих участников комитета по UEFI включить в стандарт 2.3.1 механизмы Secure Boot. Впрочем, многие участники сами с удовольствием к этому склоняются, потому что это позволяет им эффективно ограничивать варианты использования PC (теперь уже не только для планшетки и смартфона) совместимой системы, что позволяет разбивать рынок на сегменты и получать дополнительные доходы. Привет, Apple.

Secure Boot умеет проводить авторизацию через RADIUS, а ещё умеет проверять цифровые подписи загружаемых образов: драйверов, загрузчиков OS, приложений.

Проверяет он это при помощи двух типов ключей: PK и KEK. PK - это Platform Key, записываемый в систему её владельцем (понятно, что записывается открытая часть ключа, как в любой схеме с цифровой подписью).

KEK - это Key Exchange Keys, которые записываются операционной системой, эти ключи необходимы для того, чтобы OS могла подтверждать свою подлинность.

Система может работать в двух режимах Setup (PK не записан) или User (PK записан). Если система в Setup режиме, то KEK-и можно записывать без ограничений, если в User, то они быть подписаны при помощи секретной части PK.

Важны именно KEK-и, потому что UEFI Secure Boot должен загружать только те образы драйверов и приложений (в частности загрузчиков OS), сигнатуры которых помечены, как valid в специальной базе сигнатур, которую ведёт сама прошивка. Чтобы записать сигнатуру в эту базу, её нужно подписать одним из KEK-ов (это тоже не однозначный момент в документации, как гарантировать, что вредоносное ПО не заставит пользователя фальшивым запросом подписать вредоносный ключ?).

В стандарте ничего не сказано о форматах ключей – это может создать некий небольшой барьер для таких сценариев, однако, это же может создать неприятный барьер для авторов свободного ПО.

Microsoft же, для сертификации под Windows 8, требует от производителей обязательного наличия в системе механизмов Secure Boot. Для ARM-систем требуется запрет возможности перевода системы в Setup Mode, то есть, PK-ключ записан на заводе, это ключ Microsoft (значит, по определению, системой владеет MS).

Для не-ARM систем Setup режим разрешён. Но тут могут быть варианты. Например, как в Chromebook'ах в Setup-режиме может включаться долгая пауза перед загрузкой системы. Он может не быть включён по умолчанию (и тогда придётся бегать по всему кластеру и включать его вручную), не известно, какие настройки будут в Setup-режиме для загрузки со съёмных носителей. И т.д.

А если возникнет желание перевести систему в User режим со своим PK, то это может и не получиться: ведь, формат ключей не стандартизирован. Есть стандарт лишь для протоколов для абстрактного манипулирования ими. Возникает вопрос: а насколько качественно будут генерироваться ключи UEFI-ем? Дополнительные сложности создаёт и то, что работают сервисы ключей при этом только в Boottime, а в Runtime (собственно, режим, когда работает установщик OS) ключи нужно записывать в UEFI-переменные, и для этого формат данных не указан.

В общем, следует опасаться наклейки на оборудовании: сертифицировано для Windows 8. Потому что, свободно обращаться с подобным оборудованием будет затруднительно.

Хотя, ответственные за Российскую НПП обещают, что мы сможем выставить требования к поставщикам оборудования включать некие ключи НПП (что это за ключи – не ясно, какая схема использования – не понятно, кто будет подписывать ядра Linux, скомпилированные с нужной оптимизацией – тайна), либо (или «или»?) требовать от поставщиков поставки систем сразу в Setup Mode.

При этих всех сложностях, создаваемая цепочка доверия очень хрупкая. Если кому-то удастся исполнить код в режиме ядра, то этот некто, естественно, сможет создать искусственное UEFI-окружение и перезапустить в нём ядро OS, заставив его доверять вредоносному коду. Возможно даже, если у OS будут закрытые части KEK-ов (о модели распространения ключей ничего не известно), то такой злоумышленник сможет записать свой код в цепочку загрузки.

С этим Microsoft предполагает бороться запретом на загрузку не подписанных самой Microsoft драйверов. До свидания, любимый бабушкин принтер, здравствуй стимуляция потребления нового оборудования с выходом каждой новой версии Windows.

Но в такой ситуации, ключи для платформы или KEK-ки станут очень желанной целью для злоумышленников: с какой скоростью они попадут на чёрный рынок? Ведь, ключи воруют даже у профессиональных провайдеров цифровых подписей и банков. Можно ли полагаться в данном случае на секретность секретных ключей? Ведь, если эти ключи будут скомпрометированы, и если Microsoft будет безоговорочно им доверять (механизм отзыва ключей в UEFI 2.3.1 не предусмотрен: нельзя заменить PK-ключ в системе, если она не поддерживает Setup режим), даже не спрашивая о степени доверия к ним у пользователя, то?...