Статья Как буткиты внедряются в современные прошивки и чем UEFI отличается от Legacy BIOS

На связи Антон Белоусов и Алексей Вишняков, и мы продолжаем вместе с вами изучать буткиты — наиболее опасный класс вредоносного ПО. Гонка вооружений между разработчиками решений в области ИБ и вирусописателями не останавливается ни на секунду: первые активно внедряют новые механизмы противодействия киберугрозам, а вторые — инструменты их обхода. Как раз с точки зрения безопасности нас заинтересовал современный стандарт предзагрузки операционных систем UEFI. Поэтому в этом посте мы решили:

- разобраться, чем загрузка в режиме UEFI отличается от загрузки в режиме Legacy BIOS;
- рассказать о новых экземплярах буткитов, нацеленных на компрометацию UEFI;
- выяснить, похожи ли они на своих предшественников (обширную группу так называемых legacy-буткитов мы подробно изучили в прошлый раз);
- рассмотреть используемые злоумышленниками техники и слабые места систем на платформе UEFI.

> Кто не хочет читать много букв, может ознакомиться с этой же информацией в формате вебинара. Такой же уровень полезности гарантируем!

Буткиты бывают двух типов. Первые заточены на более ранние платформы, так называемые Legacy BIOS. Вторые адаптированы под современные решения, которые имеют универсальный интерфейс UEFI для связи ОС с программами, которые управляют низкоуровневыми функциями устройств (далее — UEFI, прошивка).

Несмотря на более современный подход к обеспечению безопасности UEFI, в коде его прошивок регулярно обнаруживают уязвимости. Например, в феврале этого года наши коллеги из компании Binarly выявили более двух десятков брешей в UEFI, которые затрагивают миллионы устройств от крупнейших производителей. Злоумышленники, конечно же, не дремлют и стараются их активно эксплуатировать. Поэтому актуальность исследования защищенности платформы только растет. Большинство буткитов универсальны. Например, ESPecter и FinSpy умеют заражать как старые, так и новые платформы. Одним из недавно обнаруженных UEFI-буткитов стал MoonBounce.

Архитектура UEFI

UEFI (Unified Extensible Firmware Interface, единый интерфейс расширяемой прошивки) — это небольшая операционная система, которая начинает работать при включении компьютера. Она пришла на замену устаревшей модели BIOS. UEFI позволяет унифицировать процесс разработки и создавать отдельные модули, а не только прошивку целиком. Фреймворк имеет хорошо читаемый и документированный код на C. Многие пользователи отмечают удобный интерфейс, который разрешает пользоваться мышью. Среди других преимуществ платформы — поддержка сети «из коробки» и возможность работать с дисками объемом более 2 ТБ. Все это позволяет ускорить процесс разработки и сделать его безопаснее.

Архитектура UEFI

Unified Extensible Firmware Interface

- унификация
- модульность
- удобство разработки
- GUI, поддержка языков, сеть
- поддержка дисков > 2 ТБ

Интерфейс UEFI

Архитектура UEFI

GUID Partition Table (GPT) вместо MBR

Длина адреса блока – 64 бита

512 → 9.44 ZB ($2^{64} \times 512$)

4096 → 75.6 ZB ($2^{64} \times 4096$)

GUID Partition Table Scheme

Вычисление размера диска

По требованиям платформы диск должен быть размечен в формате GPT. Соответственно, длина адреса блока будет равна 64 битам. Опираясь на эти данные, давайте вычислим гипотетический размер диска, который можно адресовать с

помощью этого формата. Если использовать сектор размером 512 байтов, получим более 9 зеттабайт.

Код для инициализации оборудования, в том числе код для загрузки ОС, разумеется, важный элемент платформы, однако наибольший интерес для нас представляет интерфейс, который позволяет взаимодействовать с UEFI.

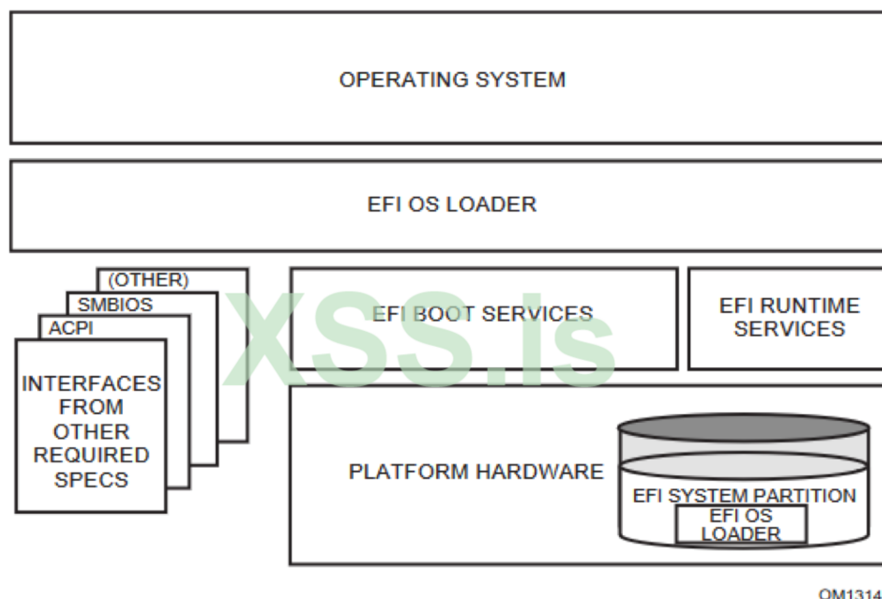


Figure 1-1 UEFI Conceptual Overview

UEFI Specification, Version 2.9

Высокоуровневый взгляд на UEFI

Приложения и драйвера UEFI имеют две таблицы для работы с API-интерфейсом — EFI Boot Services Table (службы доступны в процессе загрузки) и EFI Runtime Services Table (службы доступны и в процессе загрузки, и в процессе работы ядра ОС). API-интерфейс необходим, чтобы взаимодействовать с оборудованием, выделять память и выполнять другие простые функции по аналогии с WinAPI.

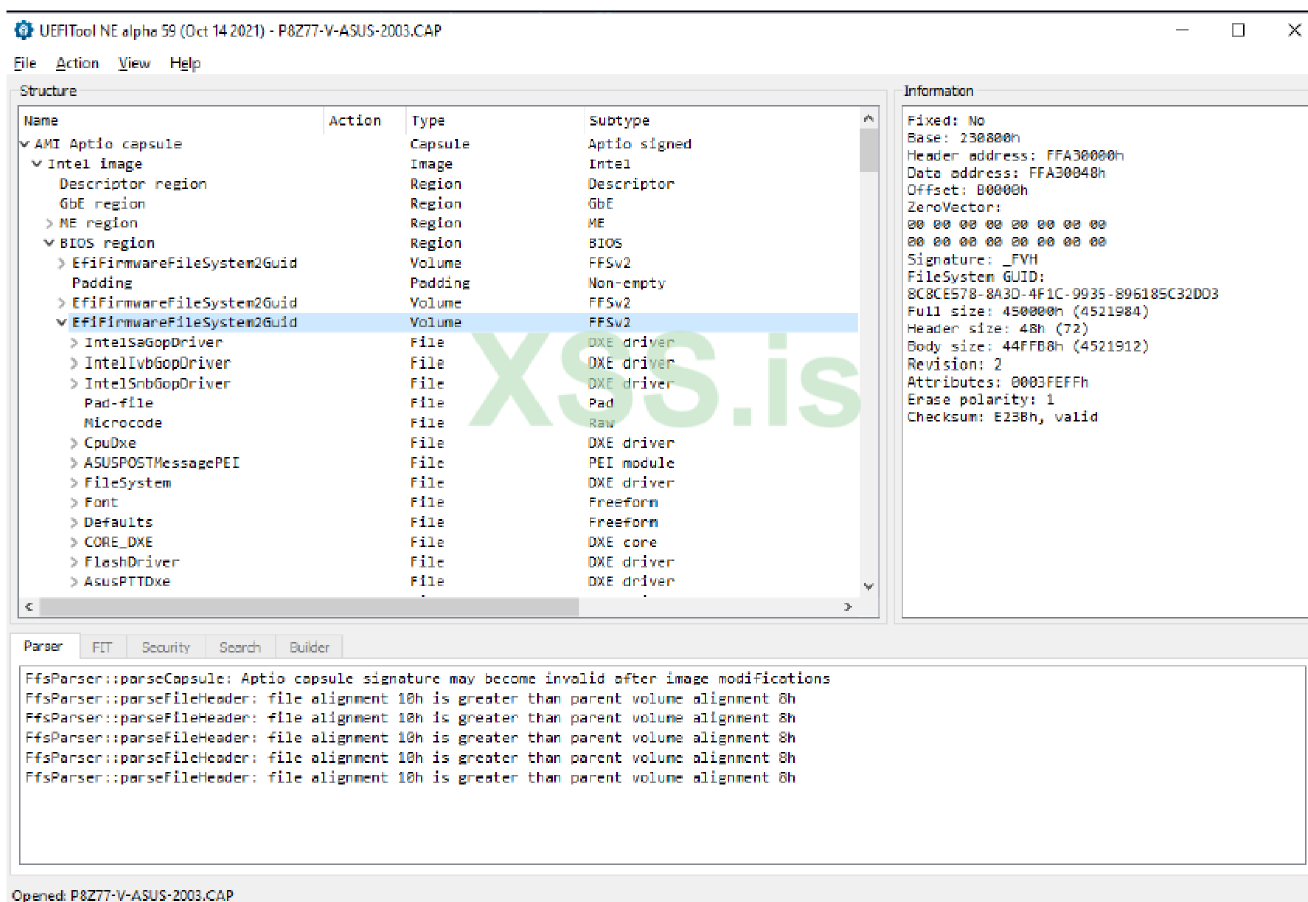
```
///
/// EFI Boot Services Table.
///
typedef struct {
  ///
  /// The table header for the EFI Boot Services Table.
  ///
  EFI_TABLE_HEADER                Hdr;

  //
  // Task Priority Services
  //
  EFI_RAISE_TPL                   RaiseTPL;
  EFI_RESTORE_TPL                 RestoreTPL;

  //
  // Memory Services
  //
  EFI_ALLOCATE_PAGES              AllocatePages;
  EFI_FREE_PAGES                  FreePages;
  EFI_GET_MEMORY_MAP              GetMemoryMap;
  EFI_ALLOCATE_POOL               AllocatePool;
  EFI_FREE_POOL                   FreePool;
```

```
///
/// EFI Runtime Services Table.
///
typedef struct {
  ///
  /// The table header for the EFI Runtime Services Table.
  ///
  EFI_TABLE_HEADER                Hdr;

  //
  // Time Services
  //
  EFI_GET_TIME                    GetTime;
  EFI_SET_TIME                    SetTime;
  EFI_GET_WAKEUP_TIME             GetWakeupTime;
  EFI_SET_WAKEUP_TIME             SetWakeupTime;

  //
  // Virtual Memory Services
  //
  EFI_SET_VIRTUAL_ADDRESS_MAP     SetVirtualAddressMap;
  EFI_CONVERT_POINTER             ConvertPointer;
```

XSS.is

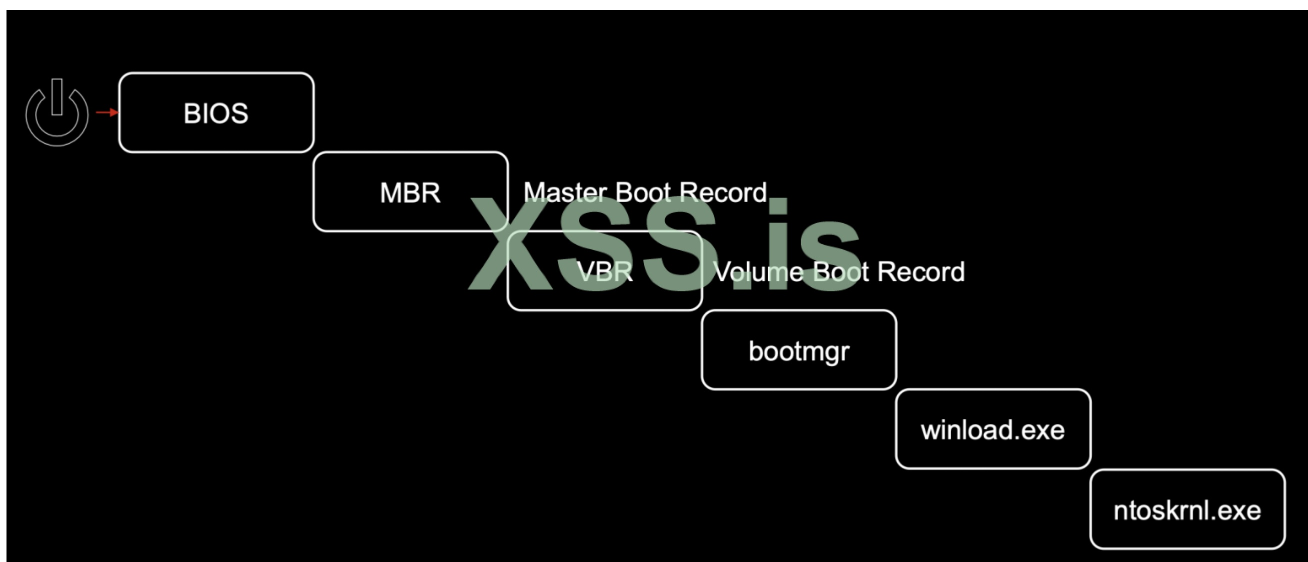
EFI Boot Services Table и EFI Runtime Services Table

Как мы уже говорили ранее, прошивку UEFI можно дополнять модулями. Утилита UEFI Tool позволяет парсить файлы прошивок различных вендоров. Например, файл прошивки Asus содержит множество драйверов и модулей. С помощью этой утилиты в процессе разработки можно менять, удалять или добавлять модули в зависимости от случая.



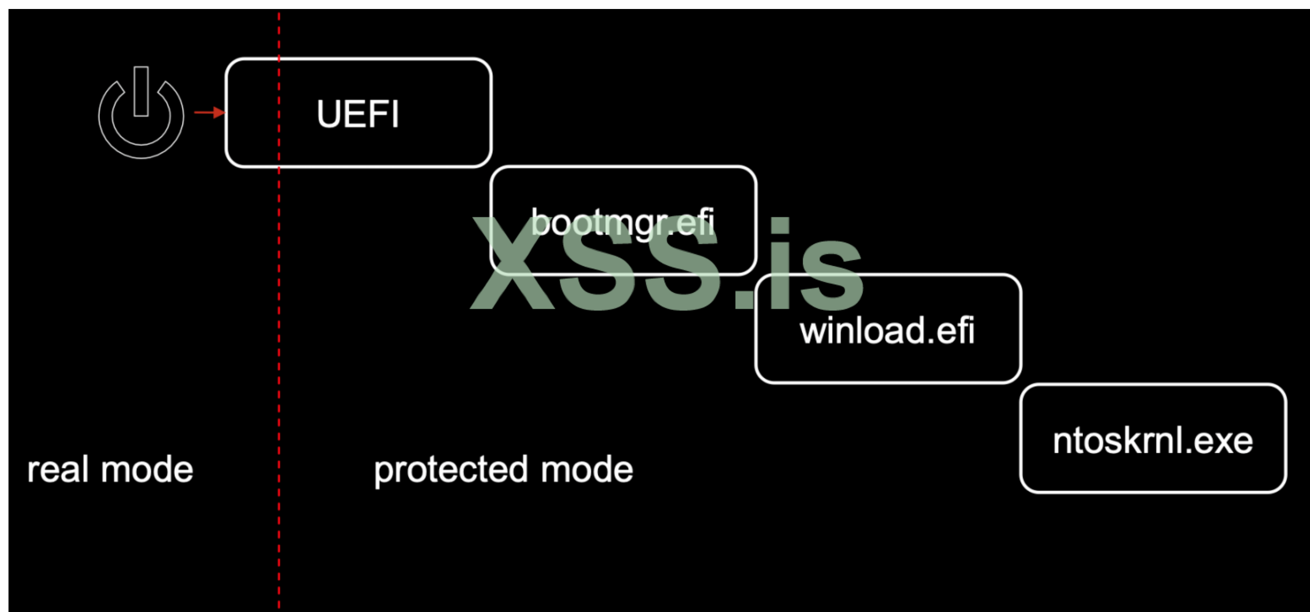
Файл прошивки Asus, открытый в UEFI Tool

Напомним схему цепочки программ в Legacy BIOS (ее мы рассматривали в прошлой статье).



Загрузка в режиме BIOS

При переходе к UEFI такие компоненты, как MBR и VBR, исчезают, и их функции частично берут на себя другие модули.



Загрузка в режиме UEFI

В UEFI процессор тоже стартует в реальном режиме, но переключение происходит на раннем этапе исполнения кода прошивки. За счет этого компоненты, которые находятся в файловой системе и затрагивают ОС, сразу работают в защищенном режиме.

Код UEFI достаточно сложный. Он включает семь стадий работы прошивки: Security, Pre-EFI Initialization, Driver Execution Environment, Boot Device Selection, Transient System Load, Runtime и Afterlife. Что интересно, работа на стадии SEC происходит во временной памяти, формирующейся из кэша процессора. На этом уровне прошивка становится корнем доверия всей системы.

На стадии DXE выполняются DXE-драйверы для устройств. Именно на данном этапе злоумышленникам удобнее всего встраивать вредоносные сущности в полезную нагрузку. BDS — стадия выбора загрузочного устройства, которое отвечает за запуск ОС. Runtime — стадия работы ОС. Afterlife — стадия завершения работы компьютера.

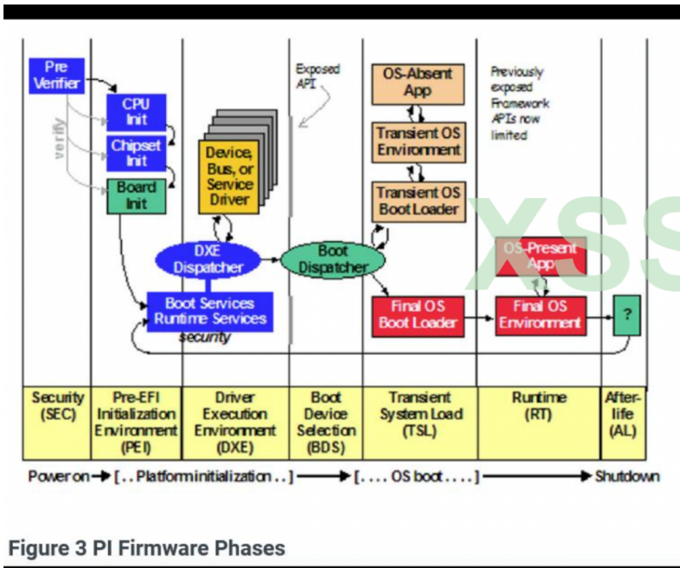


Figure 3 PI Firmware Phases

1. Security (SEC)
2. Pre-EFI Initialization (PEI)
3. Driver Execution Environment (DXE)
4. Boot Device Selection (BDS)
5. Transient System Load (TSL)
6. Runtime (RT)
7. Afterlife (AL)

XSS.is

Стадии кода UEFI Источник: https://edk2-docs.gitbook.io/edk-ii-build-specification/2_design_discussion/23_boot_sequence

NVRAM-переменные помогают UEFI понимать, с какого устройства требуется загрузиться. Они хранятся в энергонезависимой памяти. Есть стандартные переменные и те, которые определяют разработчики платформ. Например, переменные db и dbx используются Secure Boot, они содержат базу данных с ключами и хешами.

Нам интересна переменная BootOrder, определяющая порядок загружаемых устройств, и переменная Boot#### с номером устройства.

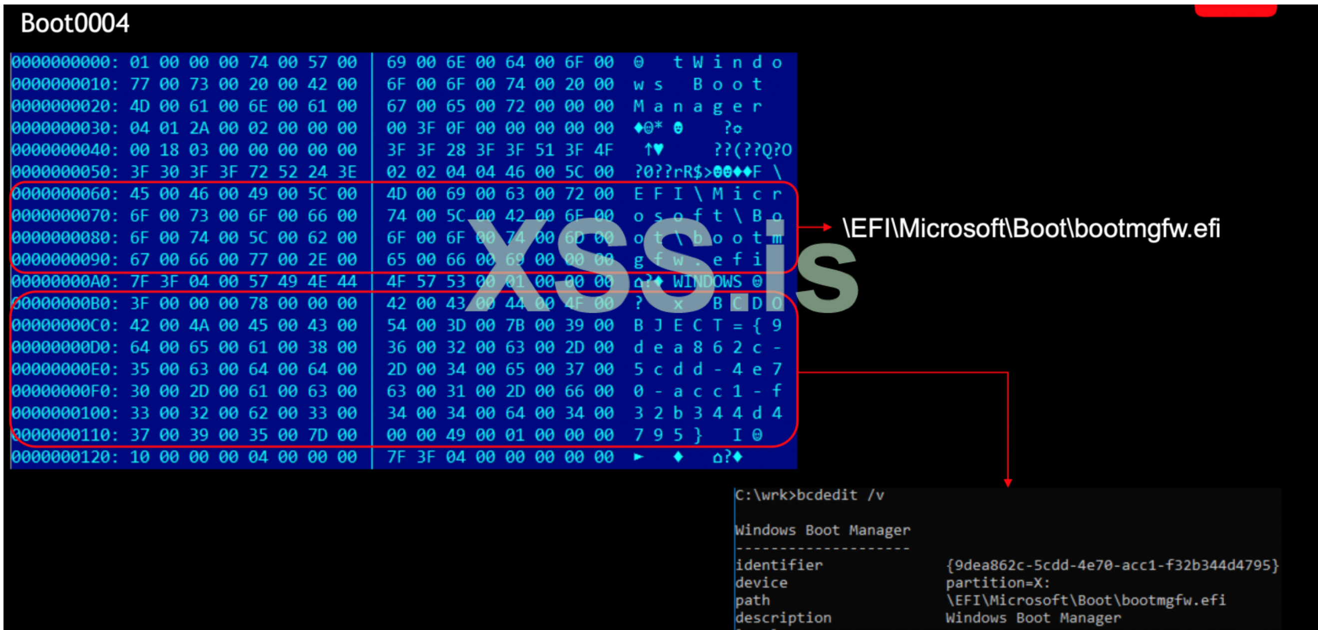
Table 3-1 Global Variables

Variable Name	Attribute	Description
AuditMode	BS, RT	Whether the system is operating in Audit Mode (1) or not (0). All other values are reserved. Should be treated as read-only except when DeployedMode is 0. Always becomes read-only after ExitBootServices() is called.
Boot####	NV, BS, RT	A boot load option. #### is a printed hex value. No 0x or h is included in the hex value.
BootCurrent	BS, RT	The boot option that was selected for the current boot.
BootNext	NV, BS, RT	The boot option for the next boot only.
BootOrder	NV, BS, RT	The ordered boot option load list.
BootOptionSupport	BS,RT,	The types of boot options supported by the boot manager. Should be treated as read-only.
ConIn	NV, BS, RT	The device path of the default input console.
ConInDev	BS, RT	The device path of all possible console input devices.
ConOut	NV, BS, RT	The device path of the default output console.
ConOutDev	BS, RT	The device path of all possible console output devices.
dbDefault	BS, RT	The OEM's default secure boot signature store. Should be treated as read-only.
dbrDefault	BS, RT	The OEM's default OS Recovery signature store. Should be treated as read-only.
dbtDefault	BS, RT	The OEM's default secure boot timestamp signature store. Should be treated as read-only.
dbxDefault	BS, RT	The OEM's default secure boot blacklist signature store. Should be treated as read-only.

XSS.is

Список NVRAM-переменных

Если рассмотреть дамп переменной Boot004 (она относится к первому устройству в списке BootOrder), можно обнаружить путь к менеджеру загрузки bootmgfw.efi. Так переменная указывает прошивке, где искать менеджер загрузки. В переменной также есть отсылка в виде GUID на хранилище параметров загрузки BCD (Boot Configuration Data). В зависимости от системы важные данные можно сохранять непосредственно в NVRAM-область.



Дамп переменной Boot0004

Ниже представлена схема диска, размеченного с помощью GPT. В нулевом секторе расположен Protective MBR. Его формат предполагает всего один раздел, описывающий весь диск. Намеренно указывается специальный диск этого раздела: это необходимо, чтобы компьютер без поддержки UEFI мог распознать диск формата GPT и случайно не затер его.

В первом секторе размещен заголовок. Он содержит параметры диска, разметку и адреса смещения данных.

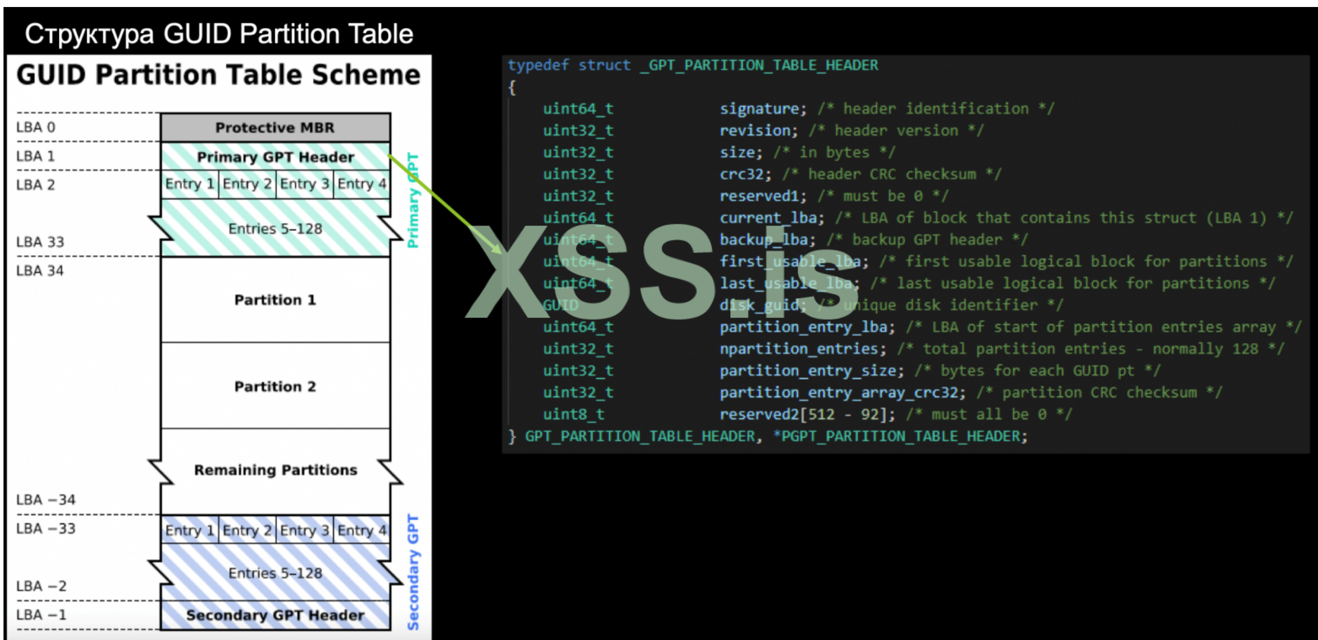
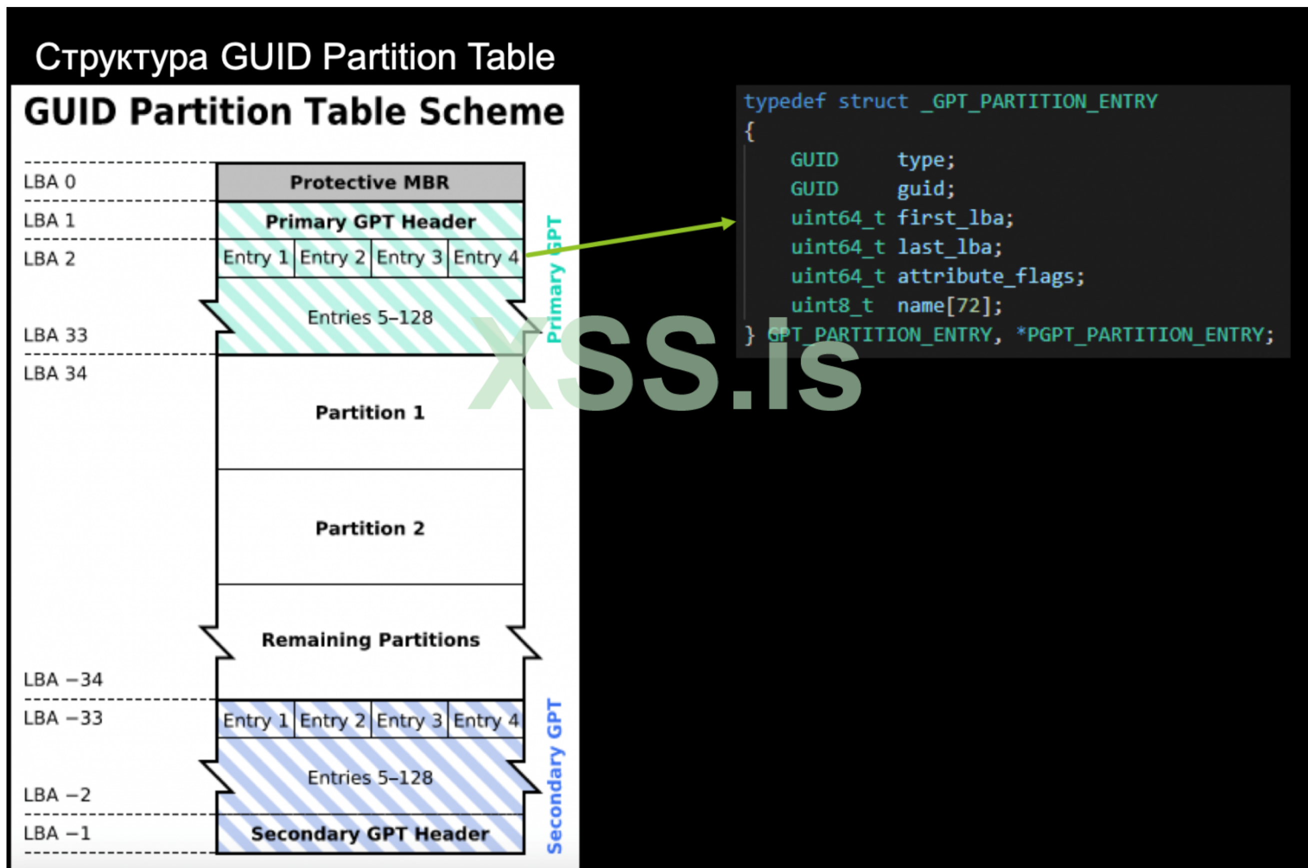


Схема диска, размеченного GPT

За первым сектором следует таблица разделов. В ней 128 элементов, каждый из которых описывает определенный логический раздел на диске.



Структура элемента таблицы

Больше всего интересен первый элемент — type (член структуры, описывающей раздел), который позволяет найти раздел EFI System Partition. Там расположен менеджер загрузки.

```
       0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
0200h: 45 46 49 20 50 41 52 54 00 00 01 00 5C 00 00 00  EFI PART....\...
0210h: EA 64 BF CE 00 00 00 00 01 00 00 00 00 00 00 00  êd¿Î............
0220h: FF FF 7F 0C 00 00 00 00 22 00 00 00 00 00 00 00  ÿÿ......".......
0230h: DE FF 7F 0C 00 00 00 00 87 17 52 E5 4B 82 58 4C  Þÿ......‡.RåK‚XL
0240h: 9D 7E 57 F1 3B D1 12 C4 02 00 00 00 00 00 00 00  .~Wñ;Ñ.Ä........
0250h: 80 00 00 00 80 00 00 00 BE C2 72 37 00 00 00 00  €...€...¾Âr7....
0260h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0270h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0280h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0290h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
02A0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
02B0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

XSS.is

Template Results - Drive.bt

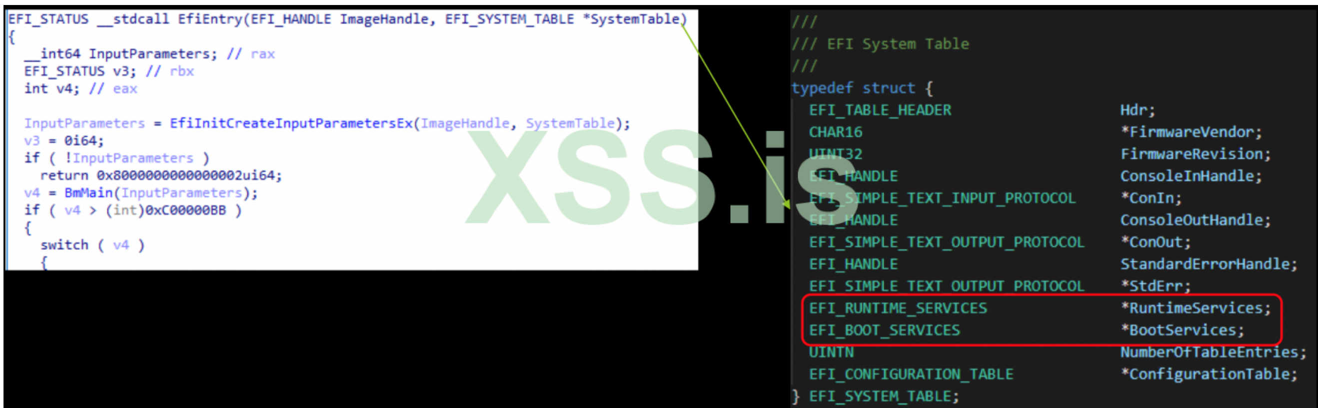
Name	Value	Start	Size
> struct MASTER_BOOT_RECORD boot_mbr		0h	200h
∨ struct EFI_PARTITION efi[0]		200h	400h
> struct EFI_PARTITION_HEADER header		200h	200h
> struct EFI_PARTITION_ENTRY partitions[0]	Basic data partition (EFI_RECOVERY)	400h	80h
∨ struct EFI_PARTITION_ENTRY partitions[1]	EFI system partition (EFI_SYSTEM)	480h	80h
enum EFI_TYPE Type	EFI_SYSTEM (C12A7328h)	480h	4h
> struct GUID TypeGUID	{C12A7328-F81F-11D2-BA4B-00A0C93EC93B}	480h	10h
> struct GUID PartitionGUID	{EA28A9B9-5198-4FC8-B330-978B7252243E}	490h	10h
UQUAD FirstLBA	FA000h	4A0h	8h
UQUAD LastLBA	12B7FFh	4A8h	8h

Раздел EFI System Partition

Раздел EFI System Partition имеет структуру формата FAT32, содержит таблицы кластеров, адрес корневой директории и древовидную структуру расположения всех файлов. Его можно распарсить, прочитать и найти все необходимые файлы. Например, для Windows есть специальная утилита mountvol.exe, позволяющая смонтировать по умолчанию скрытый раздел EFI System Partition и посмотреть его содержимое. Для этого необходимо указать ключ /S и букву диска, куда его следует смонтировать. В директории, которая была в NVRAM-переменной, находится файл менеджера загрузки. Помимо него, там есть и другие файлы: модуль, отвечающий за работу отладчика, и файл BCD (куст реестра с BCD-параметрами). Параметры будут необходимы в процессе работы менеджера загрузки и инициализации системы.

Использование утилиты mountvol.exe

Менеджер загрузки является UEFI-приложением, поэтому должен соответствовать соглашению о точке входа. Точка входа менеджера загрузки имеет определенную сигнатуру. Ее второй параметр — SystemTable — указывает на структуру, которая, в свою очередь, содержит указатели на таблицу RuntimeServices и таблицу BootServices. Их задача — предоставить API-интерфейс для работы с оборудованием.



Точка входа менеджера загрузки

Давайте обобщим ключевые отличия загрузки в режиме UEFI от загрузки в режиме Legacy BIOS:

- **переключение режимов выполняется UEFI на ранней стадии;**
- **для работы с оборудованием используется API-интерфейс, в частности RuntimeServices и BootServices;**

- **менеджер загрузки (bootmgf.efi) инициализирует механизм проверки целостности кода (Code Integrity);**
- **загрузчик ОС winload.efi заворачивает UEFI-сервисы для использования ядром через HAL.**

Изучаем Secure Boot

На наш взгляд, самый интересный компонент прошивки UEFI — Secure Boot. Его архитектура хорошо описана и проиллюстрирована в книге «Руткиты и буткиты. Обратная разработка вредоносных программ и угрозы следующего поколения» (12+) за авторством Алекса Матросова, Евгения Родионова и Сергея Братуся.

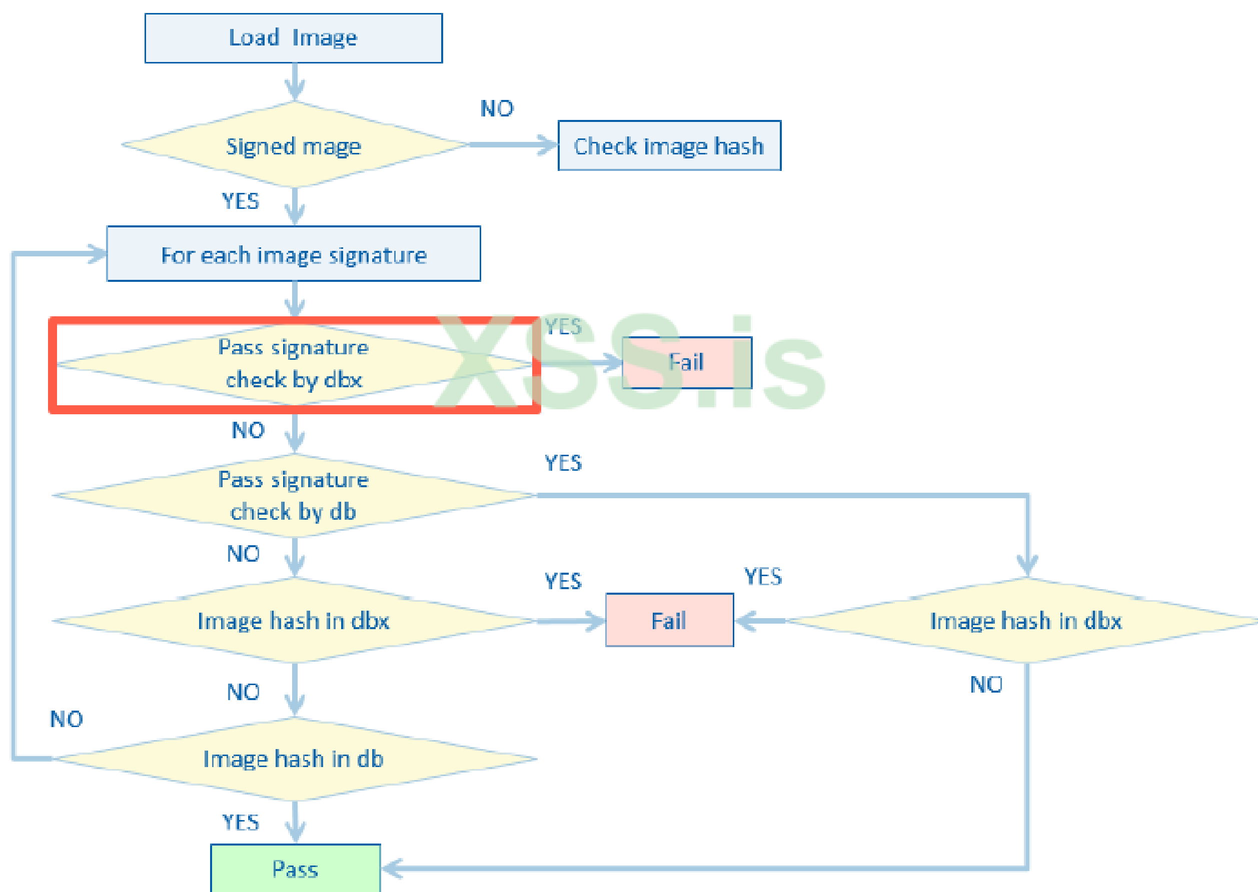
В основе Secure Boot лежит набор ключей разного уровня. Самый важный среди них — platform key (PK), который содержится в прошивке. Начальный PK верифицирует key exchange key (KEK).

Стоит отметить, что существует две базы ключей:

- db — база ключей для аутентификации, отвечает за проверку подписей модулей;
- dbx — база запрещенных ключей и хешей.

Согласно алгоритму работы Secure Boot, при загрузке нового модуля проверяется не только его подпись, но и хеш. Дело в том, что бывают модули UEFI-драйверов с абсолютно легитимными подписями. При обнаружении критически опасной уязвимости, даже если модуль пройдет проверку подписи, но его хеш будет отмечен в базе исключений, он не будет допущен к загрузке и исполнению.

> UEFI-модули могут быть двух форматов:
> · Portable Executable (привычный для Windows);
> · Terse Executable. Этот формат похож на Portable Executable, но лишен части полей и имеет другие сигнатуры.



Алгоритм работы Secure Boot Источник: https://edk2-docs.gitbook.io/unders...in/secure_boot_chain_in_uefi/uefi_secure_boot

Как работает UEFI-буткит

Есть два варианта заражения:

1. Заражение прошивки

2. Заражение в EFI System Partition, в частности:

· подмена bootmgfw.efi

· добавление нового модуля

Известны успешные атаки, в ходе которых злоумышленникам удавалось перезаписать код в SPI-чипе, то есть, по сути, заразить его. Позднее появилась пара буткитов, нацеленных на компрометацию менеджера загрузки.

На данный момент мы можем выделить два вектора атаки на UEFI-платформу: перепрошивка SPI и модификация менеджера загрузки. Касательно второго вектора следует выделить частный случай добавления нового модуля в раздел EFI System Partition и последующего изменения пути в NVRAM-переменной, которая указывает, с какого устройства необходимо загрузиться.

Первая ласточка — буткит LoJax

Первый UEFI-буткит был найден и описан специалистами ESET. В LoJax использовался подход перезаписи содержимого прошивки. Другими словами, он считывал прошивку, находил маркеры, что-то добавлял и записывал обратно.

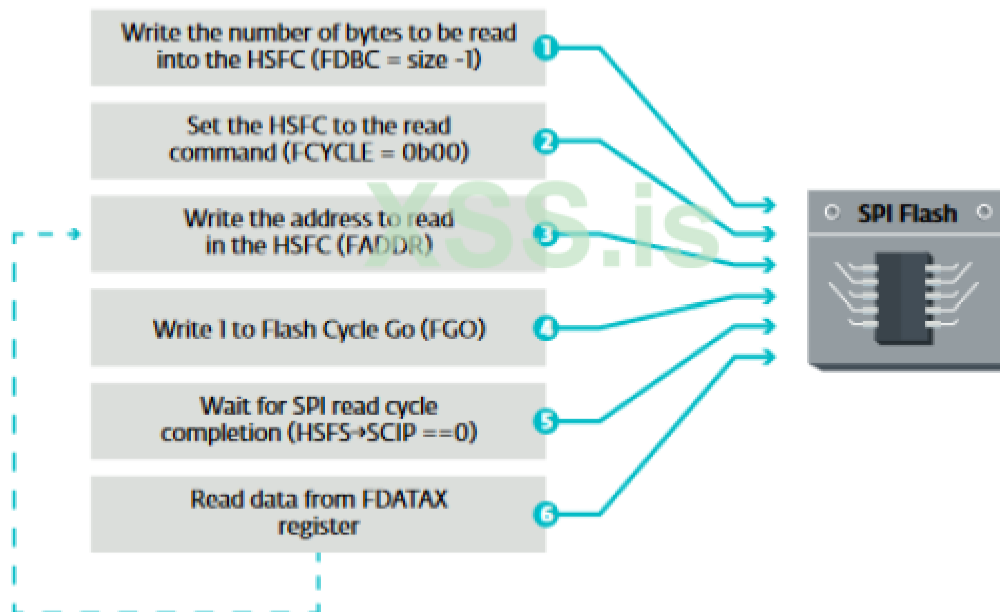


Figure 8 // Operation sequence to read from the SPI flash memory

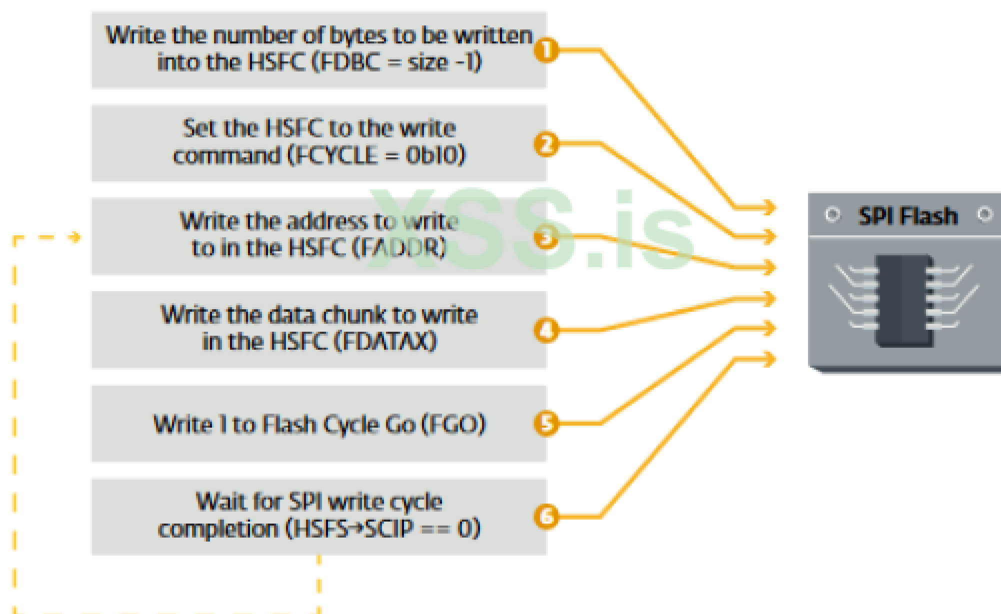


Figure 13 // Operation sequence to write to the SPI flash memory

Схема работы буткита LoJax Источник: https://www.welivesecurity.com/wp-content/uploads/2018/09/ESET-LoJax.pdf

Важно понимать, что чип SPI представляет собой некое PCI-устройство. Следовательно, чтобы с ним взаимодействовать нужен особый интерфейс. Поэтому чтение и запись содержимого чипа — это многоступенчатая и сложная операция, включающая работу с различными флагами и параметрами. Если изучить код одного из модулей LoJax, умеющих читать и писать в SPI, можно убедиться, что это действительно PCI-устройство. В модуле прописан PCI-адрес устройства, а с устройством этот модуль взаимодействует посредством драйвера с помощью функции DeviceIoControl.

```
if ( ioctl == 0x222840 )
{
  *(_DWORD *)&InBuffer.value = 0;
  if ( DeviceIoControl(hRwDrv, 0x222840u, &InBuffer, 0xCu, &InBuffer, 0xCu, &BytesReturned, 0) )// PCI config read
  {
    memcpy_0(value, &InBuffer.value, v6);
    log_write("GetPCIConfRegs -> Bus:0x%x Dev:0x%x Func:0x%x Offset:0x%x Value = 0x%x\n", 0, 31, 0, offset, *value);
    return 0;
  }
  GetLastError();
  log_write("GetPCIConfRegs -> failed. Error = 0x%x\n", 1);
  return GetLastError();
}
else if ( ioctl == 0x222834 )              // PCI Config write
{
  InBuffer.value = *value;
  if ( DeviceIoControl(hRwDrv, 0x222834u, &InBuffer, 0xCu, &InBuffer, 0xCu, &BytesReturned, 0) )
    return 0;
  LastError = GetLastError();
  log_write("GetPCIConfRegs -> failed. Error = 0x%x\n", LastError);
  return GetLastError();
```

XSS.is

Функция DeviceIoControl

Интерфейс работы с SPI непрозрачен. Кроме того, механизмы безопасности защищают его от простого считывания. В случае LoJax в основу подхода по компрометации легла эксплуатация уязвимости race condition. Стоит отметить, что данная уязвимость присутствует не во всех системах. У SPI-чипа есть набор флагов, управляющих не только разрешением на запись, но и его блокировкой. Один из флагов можно сбросить с помощью кода, работающего в режиме System Management Mode (SMM). Это привилегированный режим, в котором приостанавливается исполнение другого кода, в том числе кода ОС и гипервизора. Как только происходит попытка записи, SMM блокирует возможность записывать. Поэтому создатели буткитов используют два потока. Таким способом они одновременно пытаются сбросить защитные флаги и записать что-то в чип.

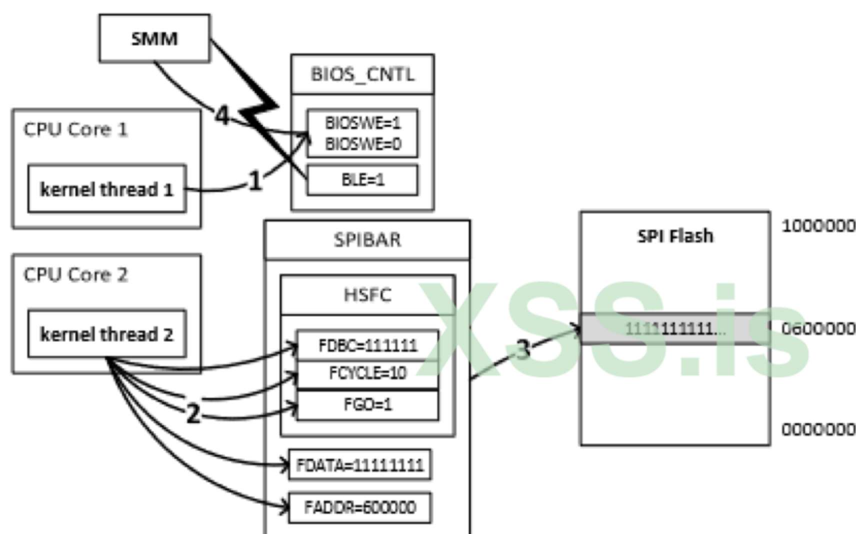
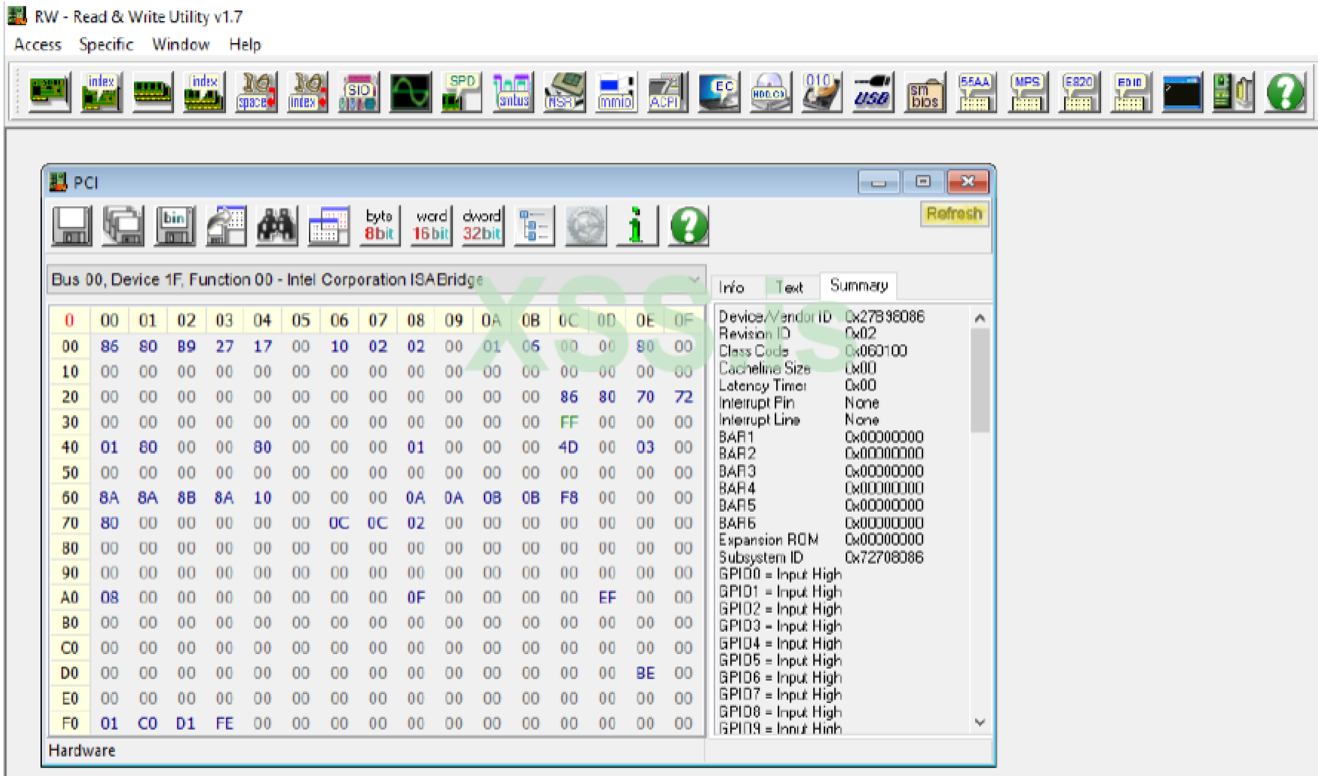


Figure 3: BIOS_CNTL Race Condition Exploited

Speed Racer: Exploiting an Intel Flash Protection Race Condition/ Corey Kallenberg, Rafal Wojtczuk

Схема эксплуатации race condition для перезаписи SPI Источник: https://composter.com.ua/documents/Exploiting_Flash_Protection_Race_Condition.pdf

Чтобы взаимодействовать с PCI-устройством, LoJax использует драйвер в составе утилиты Read & Write Everything. Утилита позволяет работать с низкоуровневыми компонентами системы: физической памятью, устройствами ввода-вывода и другими. Чтобы выполнять функции чтения-записи физической памяти, взаимодействия с устройствами и различными низкоуровневыми подсистемами и настройками, она имеет подписанный легитимный драйвер, который как раз и использовали разработчики LoJax.



Утилита RW

Отметим, что PCI-устройства должны следовать соглашениям и в том числе иметь в своем составе блок данных PCI Configuration Space. PCI Configuration Space находится непосредственно в PCI-устройстве. Чтобы читать этот блок или записывать в него, есть два порта ввода-вывода: 0xCF8 и 0xCFC.

Device ID		Vendor ID	
Status		Command	
Class Code			Revision ID
BIST	Header Type	Lat. Timer	Cache Line S.
Base Address Registers			
Cardbus CIS Pointer			
Subsystem ID		Subsystem Vendor ID	
Expansion ROM Base Address			
Reserved			Cap. Pointer
Reserved			
Max Lat.	Min Gnt.	Interrupt Pin	Interrupt Line

0x0CF8 – PCI_CONFIG_ADDRESS
0x0CFC – PCI_CONFIG_DATA

XSS.is

Структура PCI Configuration Space Источник:
https://en.wikipedia.org/wiki/PCI_configuration_space

Через порт PCI_COFIG_ADDRESS указывается адрес внутри блока данных, откуда можно считать (read) или записать (write) данные. Так, можно считать данные из PCI Configuration Space, например идентификатор устройства или его производителя, а также специфические для этого устройства значения. Порт PCI_CONFIG_DATA используется для считывания и записи данных. Направление зависит от применяемой инструкции: IN — считывание, OUT — запись.

Кроме того, у PCI-устройства есть Base Address Registers — область, содержащая специфические значения для данного устройства. Например, адрес в физической памяти, куда отображены управляющие регистры SPI. По сути, за счет взаимодействия с PCI-регистрами, отображенными в физической памяти, и выполняется чтение прошивки и запись в нее.

```
offset = *((unsigned __int16 *)SystemBuffer + 2);// 0x222840 -> PCI Read Dword
function = *((unsigned __int8 *)SystemBuffer + 2);
device = *((unsigned __int8 *)SystemBuffer + 1);
bus = *(unsigned __int8 *)SystemBuffer;
_disable();
__outdword(
  0xCF8u,
  (offset & 0xFC) + ((function + 8 * (device + 32 * (bus + ((((offset >> 8) & 0xF) + 128) << 8)))) << 8));
v59 = __indword((offset & 3) + 0xCFC);
*((_DWORD *)SystemBuffer + 2) = v59;
```

Чтение PCI-регистров

XSS.is

Запись в физическую память

```
v21 = MmMapIoSpace(
        (PHYSICAL_ADDRESS)SystemBuffer->PhysicalAddress,
        (unsigned int)SystemBuffer->NumberOfBytes,
        MmNonCached);
v22 = v21;
if ( v21 )
{
  v24 = (int *)SystemBuffer->OutBuffer;
  v25 = SystemBuffer->NumberOfBytes;
  v26 = v21;
  while ( v25 )
  {
    v27 = SystemBuffer->Granularity;
    if ( v27 )
    {
      v28 = v27 - 1;
      if ( v28 )
      {
        if ( v28 == 1 )
        {
          v29 = *v24++;
          *v26++ = v29;
          v25 -= 4;
        }
      }
```

Функция записи и чтения прошивки

Обращаем внимание, что в памяти отображается не содержимое чипа, а, например, четыре или восемь байтов, каждый из которых может быть либо флагом, либо управляющим значением. Эти значения изменяются — и выполняется взаимодействие. Если расписать последовательность шагов, то сначала мы:

- считываем PCI-регистры из Base Address Registers с помощью портов CF8 и CFC;
- получаем адрес регистров, отображенных в физической памяти;

- пишем в отображенные регистры команды управления, в частности указываем адрес, откуда необходимо считать новую прошивку.

Все перечисленные операции выполняются в цикле. Он включает установку контрольного флага, начало чтения (или записи) и изменение дополнительных параметров.

```
    SimpleFileSystemProtocol->OpenVolume(SimpleFileSystemProtocol, Root);
    v2 = (*Root)->Open(*Root, v6, v15, 1i64, 16i64);
    if ( v2 )
      continue;
    if ( !(__int64)(*v6)->Open(*v6, v5, v14, 1i64, 16i64) )
    {
      if ( !(__int64)(*v5)->Open(*v5, v3, (CHAR16 *)L"rpcnetp.exe", 1i64, 32i64) )
        goto LABEL_12;
LABEL_11:
      (*v5)->Open(*v5, v3, (CHAR16 *)L"rpcnetp.exe", 0x8000000000000003ui64, 32i64);
      (*v3)->Write(*v3, (UINTN *)&v12, &off_4BFE0);
      goto LABEL_12;
    }
    if ( (__int64)(*v6)->Open(*v6, v5, v11, 1i64, 16i64) )
      goto LABEL_14;
    if ( (__int64)(*v5)->Open(*v5, v3, (CHAR16 *)L"rpcnetp.exe", 1i64, 32i64) )
      goto LABEL_11;
LABEL_12:
    (*v3)->Close(*v3);
LABEL_14:
    v2 = (*v6)->Open(*v6, v5, v11, 1i64, 16i64);
    if ( !v2 )
    {
      if ( (__int64)(*v5)->Open(*v5, v3, (CHAR16 *)L"autoche.exe", 1i64, 6i64) )
      {
        (*v5)->Open(*v5, v3, (CHAR16 *)L"autoche.exe", 0x8000000000000003ui64, 6i64);
        (*v3)->Write(*v3, (UINTN *)v16, &unk_4B3E0);
        v2 = (*v3)->Close(*v3);
      }
      else
      {
        v2 = (*v3)->Close(*v3);
```

Манипуляция с байтами в сегменте физической памяти

Эти, на первый взгляд, сложные манипуляции помогают буткиту LoJax достичь цели: записать вредоносный код в прошивку, заменить модуль с легитимным кодом на свой и исполнить его на ранней стадии загрузки системы. Так он будет недосягаем для антивирусов и других средств защиты, а атакующие смогут получить необходимый им persist.

Знакомьтесь: MosaicRegressor

Еще один интересный буткит — MosaicRegressor, исследованный специалистами «Лаборатории Касперского». В его случае начальный вектор заражения неизвестен, то есть никто не знает, как буткит был установлен.

Name	Action	Type	Subtype	Text
UEFI image		Image	UEFI	
Padding		Padding	Non-empty	
8C8CE578-8A3D-4F1C-9935-896185C32DD3		Volume	FFSv2	
F50258A9-2F4D-4DA9-861E-BDA84D07A44C		File	DXE driver	SmmInterfaceBase
F50248A9-2F4D-4DE9-86AE-BDA84D07A41C		File	DXE driver	Ntfs
EAEA9AEC-C9C1-46E2-9D52-432AD25A9B0C		File	Application	SmmReset
EAEA9AEC-C9C1-46E2-9D52-432AD25A9B0B		File	Application	SmmAccessSub
5C266089-E103-4D43-9AB5-12D7095BE2AF		File	DXE driver	IntelSaGopDriver
5BBA83E6-F027-4CA7-BFD0-16358CC9E123		File	DXE driver	IntelGopDriver
5007A40E-A5E0-44F7-86AE-662F9A91DA26		File	DXE driver	FvOnFv2Thunk

XSS.is

```
EFI_STATUS __fastcall ModuleEntryPoint(EFI_HANDLE ImageHandle, EFI_SYSTEM_TABLE *SystemTable)
{
  EFI_EVENT pEvent; // [rsp+30h] [rbp-18h] BYREF

  init_BS_RT(ImageHandle, SystemTable);
  g_Finished = 0;
  g_BootServices->CreateEventEx(
    EVT_NOTIFY_SIGNAL,
    TPL_NOTIFY,
    (EFI_EVENT_NOTIFY)NotifyFunction,
    0i64,
    &EFI_EVENT_GROUP_READY_TO_BOOT,
    &pEvent);
  return 0i64;
}
```

Состав MosaicRegressor

В зараженной прошивке исследователи обнаружили четыре модуля, два драйвера и два приложения, исполняющих вредоносную нагрузку. Рассмотрим их подробнее. Первым исполняется модуль SmmInterfaceBase. В его задачу входит создание с помощью API-функции CreateEventEx события EVENT_GROUPR_READY_TO_BOOT. Это событие наступает перед тем, как управление передается менеджеру загрузки. Тогда же вызывается обратный вызов NotifyFunction.

Name	Action	Type	Subtype	Text
UEFI image		Image	UEFI	
Padding		Padding	Non-empty	
8C8CE578-8A3D-4F1C-9935-896185C32DD3		Volume	FFSv2	
F50258A9-2F4D-4DA9-861E-BDA84D07A44C		File	DXE driver	SmmInterfaceBase
F50248A9-2F4D-4DE9-86AE-BDA84D07A41C		File	DXE driver	Ntfs
EAEA9AEC-C9C1-46E2-9D52-432AD25A9B0C		File	Application	SmmReset
EAEA9AEC-C9C1-46E2-9D52-432AD25A9B0B		File	Application	SmmAccessSub
5C266089-E103-4D43-9AB5-12D7095BE2AF		File	DXE driver	IntelSaGopDriver
5BBA83E6-F027-4CA7-BFD0-16358CC9E123		File	DXE driver	IntelGopDriver
5007A40E-A5E0-44F7-86AE-662F9A91DA26		File	DXE driver	FvOnFv2Thunk

XSS.is

```
g_BootServices->CopyMem(v6 + 4, &SmmAccessSub_GUID, sizeof(EFI_GUID));
v7 = (char *)Buffer + (unsigned int)Size + 20;
*v7 = 0x7F;
v7[1] = 0xFF;
v7[2] = 4;
v7[3] = 0;
Status = g_BootServices->LoadImage(0, g_ImageHandle, (EFI_DEVICE_PATH_PROTOCOL *)Buffer, 0i64, 0i64, &ImageHandle);
if ( Status >= 0 )
  g_BootServices->StartImage(ImageHandle, 0i64, 0i64);
else
  v13 = Status;
```

Состав модулей MosaicRegressor

Интересно, что происходит внутри callback? Давайте посмотрим. С помощью уникального идентификатора приложения (GUID) обнаруживается модуль SmmAccessSub. Затем он запускается.

```
char drop_usermode_agent()
{
  CHAR16 *drop_file_path; // rbx MAPDST
  CHAR16 *v2; // rcx
  EFI_FILE_PROTOCOL *FileHandle; // [rsp+40h] [rbp+8h] BYREF
  EFI_FILE_PROTOCOL *Users; // [rsp+48h] [rbp+10h] BYREF
  UINTN BufferSize; // [rsp+50h] [rbp+18h] BYREF

  Users = 0i64;
  FileHandle = 0i64;
  qword_3CD0 = 0i64;
  BufferSize = 0x3400i64;
  if ( (gVolumeRoot->Open(gVolumeRoot, &Users, (CHAR16 *)L".\\Users", 1ui64, 0i64) & 0x8000000000000000ui64) == 0i64
    && (Users->Close(Users) & 0x8000000000000000ui64) == 0i64 )
  {
    drop_file_path = (CHAR16 *)allocate_data_pool_zero(0x208ui64);
    wcscat(drop_file_path, (CHAR16 *)L"\\ProgramData\\Microsoft\\Windows\\Start Menu\\Programs\\Startup\\");
    wcscat(drop_file_path, (CHAR16 *)L"IntelUpdate.exe");
    if ( (gVolumeRoot->Open(
            gVolumeRoot,
            &FileHandle,
            v2,
            EFI_FILE_MODE_CREATE|EFI_FILE_MODE_WRITE|EFI_FILE_MODE_READ,
            0i64) & 0x8000000000000000ui64) == 0i64
      && (FileHandle->Write(FileHandle, &BufferSize, &payload) & 0x8000000000000000ui64) == 0i64
      && (FileHandle->Close(FileHandle) & 0x8000000000000000ui64) == 0i64 )
    {
      g_BootServices->FreePool(drop_file_path);
    }
  }
  return 1;
}
```

Модуль SmmAccessSub

В этом модуле SmmAccessSub, по нашему мнению, довольно тривиальный подход к persist. Он заключается в сбросе пользовательского модуля в папку Startup. При этом никаких действий с драйверами или подмен не происходит.

Еще одним модулем в зараженной прошивке был SmmReset. Он сбрасывает значение EFE до нуля.

```
EFI_STATUS __fastcall ModuleEntryPoint(EFI_HANDLE ImageHandle, EFI_SYSTEM_TABLE *SystemTable)
{
  EFI_BOOT_SERVICES *BootServices; // rax
  EFI_RUNTIME_SERVICES *RuntimeServices; // rax
  char Data; // [rsp+40h] [rbp+8h] BYREF
  char v6; // [rsp+48h] [rbp+10h] BYREF
  UINTN v7; // [rsp+50h] [rbp+18h] BYREF

  BootServices = SystemTable->BootServices;
  gImageHandle = ImageHandle;
  gBootServices = BootServices;
  RuntimeServices = SystemTable->RuntimeServices;
  gSystemTable = SystemTable;
  gRuntimeServices = RuntimeServices;
  Data = 0;
  v7 = 1i64;
  RuntimeServices->GetVariable((CHAR16 *)L"fTA", (EFI_GUID *)&VendorGuid, 0i64, &v7, &Data);
  v6 = 0;
  gRuntimeServices->SetVariable((CHAR16 *)L"fTA", (EFI_GUID *)&VendorGuid, 7u, 1ui64, &v6);
  return 0i64;
}
```

Модуль SmmReset

Буткит MosaicRegressor не использует SmmReset, однако модуль для работы с точно такой же переменной есть в утекшем репозитории Hacking Team и применяется, чтобы определить, заражена прошивка или нет.

```
BOOLEAN
EFIAPI
SetfTA()
{
        EFI_STATUS                    Status = EFI_SUCCESS;
        UINT8 VarData;

        VarData=1;
        Status=gRT->SetVariable(L"fTA", &gEfiGlobalFileVariableGuid, EFI_VARIABLE_NON_VOLATILE | EFI_VARIABLE_BOOTSERVICE_ACCESS |

        if(Status!=EFI_SUCCESS)
        {
#ifdef FORCE_DEBUG
                Print(L"Non riesco a settare fTA\n");
#endif
                return FALSE;
        }
#ifdef FORCE_DEBUG
        Print(L"fTA settato a TRUE\n");
#endif
    return TRUE;
}
```

XSS.is

Фрагмент кода из репозитория vector-edk. Установка флага заражения прошивки Источник: https://github.com/hackedteam/vector-edk/

MoonBounce — самый примечательный экземпляр

MoonBounce можно назвать новинкой в списке буткитов, угрожающих UEFI. Его обнаружили в конце января 2022 года. Вредонос отличает сложная и интересная схема заражения. Так же, как и в случае с MosaicRegressor, начальный вектор проникновения неизвестен.

В прошивке буткит начинает работать до стадии исполнения драйверов. При этом вредоносные действия выполняются не в драйвере, а непосредственно в коде прошивки — в начале стадии DXE. Принцип работы MoonBounce следующий: буткит перехватывает несколько функций Boot-сервисов. Первая функция размещает в памяти шеллкод уровня Ring 0, другая — перехватывает функцию, которая передает управление ядру ОС, а затем перехватывает функцию выделения памяти внутри ядра ExAllocatePool. Все это MoonBounce делает, чтобы исполнить шеллкод, который был «замаплен» еще на стадии работы прошивки. Дальше в дело вступает сам шеллкод: он загружает и вызывает вредоносный драйвер, который обладает возможностями руткита. Как вы, наверное, уже поняли, необычно здесь то, что случилось не «аккуратное внедрение» модуля, а заражение всей прошивки.

Принцип работы MoonBounce Источник: https://securelist.com/moonbounce-the-dark-side-of-uefi-firmware/105468/

Атаки на Boot-менеджеры

Атаки на Boot-менеджеры рассмотрим на примере буткитов FinSpy и ESPecter. FinSpy, также известный как FinFisher, подменяет оригинальный менеджер загрузки на вредоносный. В частности, при передаче управления от UEFI к менеджеру загрузки вызывается вредоносный менеджер, который выполняет некоторые функции (например, перехватывает функции передачи управления ядру или патчит код, ответственный за проверку цифровых подписей), а затем загружается оригинальный Boot-менеджер.

Содержимое зараженного EFI System Partition и код для монтирования EFI System Partition Источник: https://securelist.com/finspy-unseen-findings/104322/

Для заражения раздела EFI System Partition злоумышленникам достаточно воспользоваться банальным набором API. Это позволяет им смонтировать раздел и в дальнейшем работать с ним, как с любым другим диском.

Принцип работы FinSpy

Алгоритм работы FinSpy выглядит так:

- подмененный Boot-менеджер загружает оригинальный Boot-менеджер;
- подменный менеджер патчит в оригинальном функцию, передающую управление сначала загрузчику, а затем ядру.

Второе действие позволяет перехватить ядерную функцию создания системного потока. Она, в свою очередь, дает возможность сбросить вредоносную нагрузку. Прошивка UEFI в этой атаке затронута не была.

ESPecter замыкает пятерку буткитов, на текущий момент известных исследователям. Его корни уходят как минимум в 2012 год. В процессе своей работы ESPecter патчит менеджер загрузки, чтобы заменить легитимный драйвер (либо beep.sys, либо winsys.dll) на вредоносный. Кроме того, загрузчик буткита патчит проверку целостности загрузчика (да-да, загрузчик проверяет сам себя на пропатченность).

Логика работы ESPecter Источник: https://www.welivesecurity.com/2021/10/05/uefi-threats-moving-esp-introducing-especter-bootkit/

Буткит ESPecter отключает проверку цифровой подписи драйверов, чтобы загрузить в систему подменный beep.sys или winsys.dll и запустить его.

Вредоносный код, позволяющий обойти проверку цифровой подписи Источник: https://www.welivesecurity.com/2021/10/05/uefi-threats-moving-esp-introducing-especter-bootkit/

Подмена легитимного драйвера на вредоносный при выключенной проверке подписи драйверов Источник: https://www.welivesecurity.com/2021/10/05/uefi-threats-moving-esp-introducing-especter-bootkit/

В ядре Windows предусмотрена функция, отвечающая за инициализацию проверки целостности. Если ее запатчить и приравнять к нулю переменную CiOptions, проверки будут проходить так, словно драйверы подписаны. Этот подход ESPecter применяет для legacy-систем.

Что это за зверь такой — System Management Mode

Рассказывая о буткитах, нельзя не упомянуть System Management Mode — привилегированный режим работы процессора. Он необходим для управления электропитанием и проприетарными функциями (их определяют сами вендоры).

Расширенная схема уровней привилегий в ПК Источник: https://medium.com/swlh/negative-ri...ts-youve-probably-never-heard-of-d725a4b6f831

Режим SMM, который часто еще называют Ring -2, непрозрачен для ОС. Переходя в него, процессор может выполнять код. Код обычно находится в области памяти SMRAM. Она начинается с адреса SMBASE. SMBASE по дефолту имеет фиксированный адрес, но также может меняться с помощью специальных MSR-регистров. В режиме SMM процессор сохраняет весь свой контекст (значения почти всех регистров в верхней области памяти).

Структура памяти SMRAM

Перейти в режим SMM процессор может несколькими способами. Например, он может выполнить прерывание (System Management Interrupt, SMI). Выделяют три вида прерывания:

· аппаратное,

· системное,

· программное.

С позиции атакующего системные прерывания сложны в реализации. Тогда как программные вполне можно вызвать кодом на уровне драйвера. Для этого на порты b3 и b2 необходимо отправить определенное значение (оно отличается для разных устройств), и процессор переключится в режим SMM.

Фрагмент кода из фреймворка CHIPSEC для вызова программного SMI Источник: https://github.com/chipsec/chipsec/blob/main/drivers/win7/amd64/cpu.asm

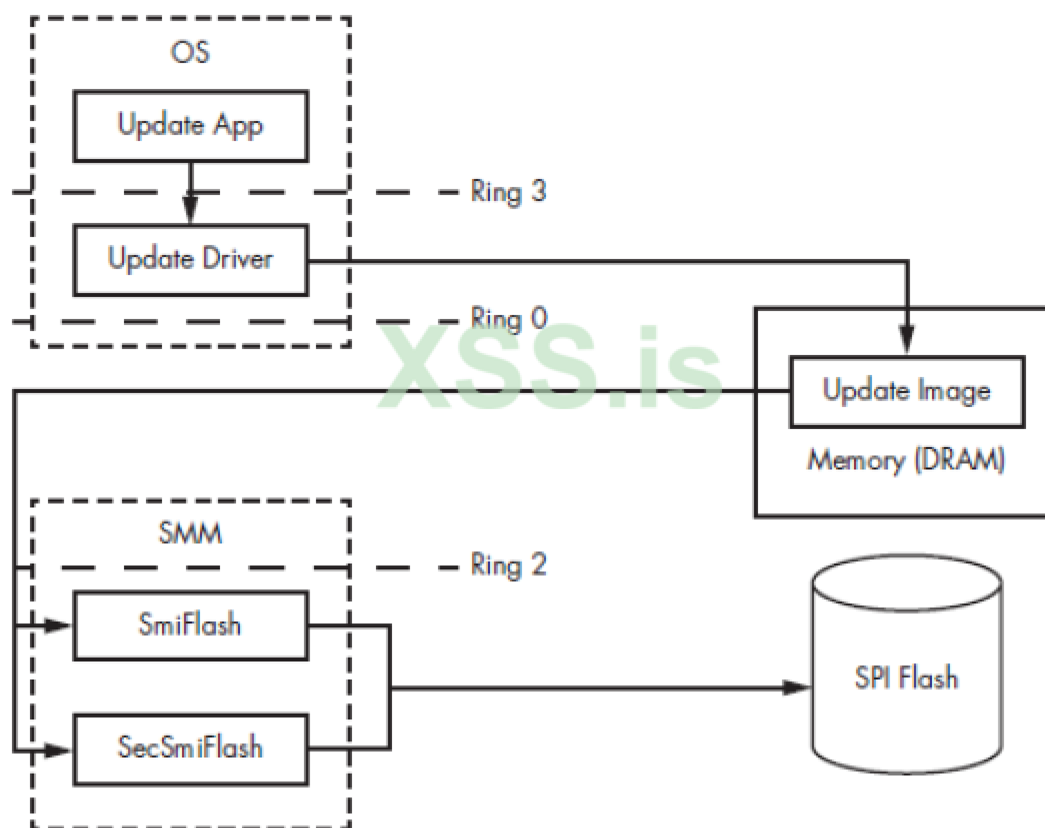


Figure 16-5: High-level representation of the BIOS update process from the OS

Высокоуровневое представление процесса обновления BIOS из ОС Источник: Rootkits and Bootkits by Alex Matrosov, Eugene Rodionov, and Sergey Bratus (No Starch Press, 2019)

Режим SMM опасен еще и тем, что имеет доступ ко всей памяти системы. Из этого вырисовывается несколько возможных вариантов эксплуатации: использование руткита с доступом к системе либо перезапись содержимого чипа SPI. SMM имеет много уязвимостей, преимущественно вендороспецифичных, например, SMM Callout, которая позволяет выполнить код в режиме SMM за пределами SMRAM. Убедиться в неиллюзорности угроз можно, ознакомившись со списком уязвимостей. По большей части они характерны либо для конкретного девайса, либо фреймворка.

Резюме

В заключение кратко перечислим главные моменты статьи:

- UEFI-буткиты могут обойти все механизмы защиты и получить контроль над всей системой;
- SPI-буткит невозможно удалить ни переустановкой ОС, ни форматированием жесткого диска;
- для борьбы с буткитами есть механизмы защиты, такие как Secure Boot и Intel Boot Guard, но они не панацея;
- наибольшую опасность сегодня представляют уязвимости в режиме SMM и подсистеме Intel ME.

Как эффективно обнаруживать UEFI-буткиты, расскажем на следующем вебинаре. Stay tuned!

Посмотреть видеоверсию статьи и скачать презентацию вы можете на сайте Positive Technologies.

Авторы:

- *Антон Белоусов, старший специалист отдела обнаружения вредоносного ПО экспертного центра безопасности Positive Technologies*
- *Алексей Вишняков, руководитель отдела обнаружения вредоносного ПО экспертного центра безопасности Positive Technologies*

Источник: https://habr.com/ru/company/pt/blog/668154/
Jun 18, 2022