Security of BIOS/UEFI System Firmware from Attacker and Defender Perspectives

Section 2. Bootkits and Secure Boot

Yuriy Bulygin *
Alex Bazhaniuk *
Andrew Furtak *
John Loucaides **

* Advanced Threat Research, McAfee
** Intel

License

Provide the following attribution:

Section 2. Bootkits and Secure Boot

2.1 In the beginning

Early days. Boot sector infection

Elk Cloner – one of the first known viruses. Created by CAS freshman Richard Skrenta (Sprengelmeyer) in 1981. Infected Apple II DOS 3.3 OS, occupied unused space inside “boot sector”. Resident in memory. Infects uninfected floppy disks.

http://virus.wikidot.com/elk-cloner

Early days. Boot sector infection

Brain – considered to be the first PC virus.

This virus originated in January, 1986, in Lahore Pakistan. The first noticeable infection problems did not surface until 1988.

The Brain is a boot sector infector, approximately 3 K in length, that infects 5 1/4" floppies.

The virus stores the original boot sector, and six extension sectors, containing the main body of the virus, in available sectors which are then flagged as bad sectors.

Brain is the only virus yet discovered that contains the valid names, phone numbers and addresses of the creators.

http://virus.wikidot.com/brain

http://www.textfiles.com/virus/braininf.vir (David Stang, NCSA)

Early days. BIOS infection

BIOS Meningitis – the worlds first flash BIOS infecting virus. Infects floppy boot-sector and hard drive MBR as well. It was coded by Qark of VLAD and appeared in Issue 2 of VLAD magazine in November 1994.

BIOS Meningitis uses various INT 16h AH=E0h (BIOS Flash routines) calls to manipulate the Flash memory. Hooks INT 19h (BIOS Boot Strap Loader) handler. The virus INT 19h handler copies the virus to 0000:7C00h (standard boot sector load address), emulates an infected MBR execution and then does the job of a standard INT 19h handler – boots from floppy or HDD.

http://virus.wikidot.com/bios-meningitis

http://www.wiw.org/~meta/vlad/vlad2/art44.htm (source code)

BIOS damage - CIH

In 1998-99 **CIH** ("Chernobyl") virus infected **60 million** and damaged **0.5 million** computers causing **~$1B** in damages

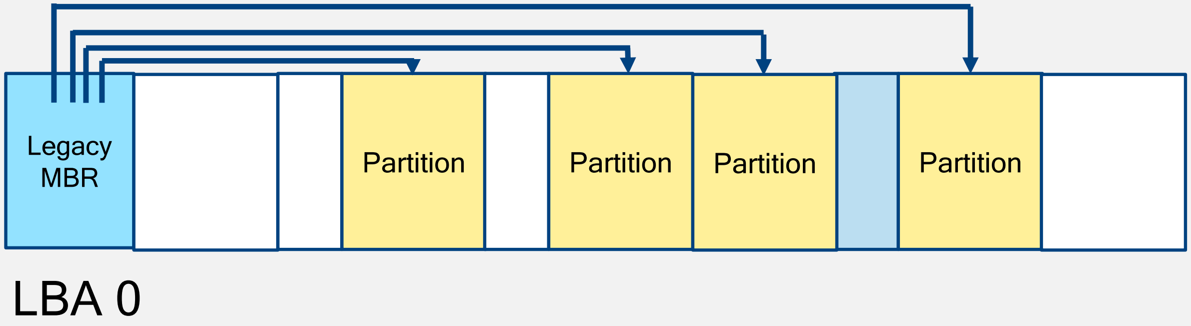
CIH destructive payload, when activated, attempts to **corrupt system BIOS in ROM** and 2048 **sectors** on all hard drives including boot sectors

Sources: Wikipedia, F-Secure, Kaspersky Lab, GRC

2.2 Boot Sectors

Master Boot Record (MBR)

- MBR helps BIOS to locate OS partition and boot the OS
- MBR is located at LBA 0 (the first boot sectors) of the disk
- MBR contains:
  - boot code (initial program loader) which is loaded and invoked by the BIOS
  - up to four partition records each defining starting and ending logical block addresses (LBA) for corresponding partitions on a disk



Source: https://technet.microsoft.com/en-us/library/cc976786.aspx

Legacy MBR structure

Mnemonic	Byte Offset	Byte Length	Description
BootCode	0	424	X86 code used on a no-UEFI system to select an MBR partition record and load the first logical block of the partition. This code shall not be executed on UEFI systems
UniqueMBRDiskSignature	440	4	Unique Disk Signature. This may be used by the OS to identify the disk from other disks in the system. This value is always written by the OS and is never written by EFI firmware
Unknown	444	2	Unknown. This field shall not be used in UEFI firmware
PartitionRecord	446	16*4	Array of four legacy MSR partition records
Signature	510	2	Set to 0xAA55 (i.e., byte 510 contains 0x55 and byte 511 contains 0xAA)
Reserved	512	Logical BlockSize - 512	The rest of the logical block, if any, is reserved.

As defined in UEFI Spec 2.5, Chapter 5 GPT Disk Layout

Source: https://technet.microsoft.com/en-us/library/cc976786.aspx

Legacy MBR partition record

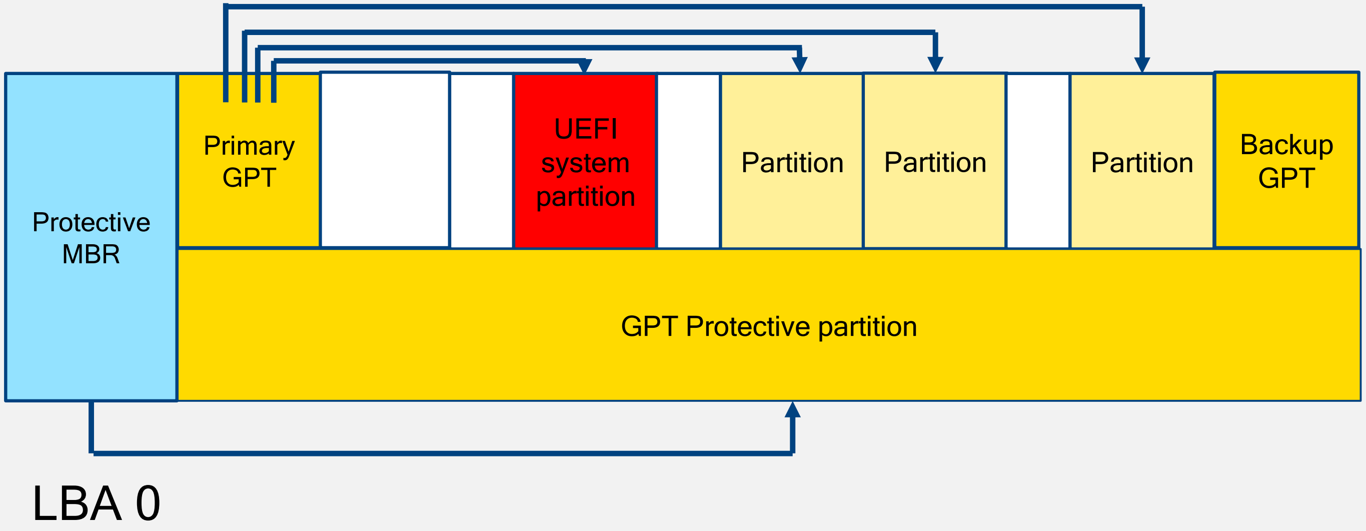
Mnemonic	Byte Offset	Byte Length	Description
`BootIndicator`	0	1	0x80 indicates that this is the bootable legacy partition. Other values indicate that this is not a bootable legacy partition. This field shall not be used by UEFI firmware
`StartingCHS`	1	3	Start of partition in CHS address format. This field shall not be used by UEFI firmware
`OSType`	4	1	Type of partition.
`EndingCHS`	5	3	Array of four legacy MSR partition records
`StartingLBA`	8	4	End of partition in CHS address format. This field shall not be used by UEFI firmware.
`SizeInLBA`	12	4	Size of the partition of LBA units of logical blocks. This field is used by UEFI firmware to determine the size of the partition

- The partition defined by each MBR Partition Record must physically reside on the disk (i.e., not exceeding the capacity of the disk).
- Each partition must not overlap with other partitions.

Source: https://technet.microsoft.com/en-us/library/cc976786.aspx

Protective MBR

- Instead of legacy MBR a protective MBR may be located at LBA0 of the disk if it is using the GPT disk layout
- One of the Partition Records shall be defined in a way reserving the entire space on the disk after the Protective MBR itself for the GPT disk layout



Reference: http://blog.fpmurphy.com/2011/10/fedora-16-mbr-grub-legacy-to-gpt-grub2.html

Protective MBR structure

The Protective MBR precedes the GUID Partition Table Header to maintain compatibility with existing tools that do not understand GPT partition structures

Mnemonic	Byte Offset	Byte Length	Description
`Boot Code`	`0`	`424`	Unused by UEFI systems
`Unique MBR Disk Signature`	`440`	`4`	Unused. Set to zero
`Unknown`	`444`	`2`	Unused. Set to zero
`Partition Record`	`446`	`16*4`	Array of four MSR partition records. Contains: • One partition record • Three partition record each set to zero
`Signature`	`510`	`2`	Set to 0xAA55 (i.e., byte 510 contains 0x55 and byte 511 contains 0xAA)
`Reserved`	`512`	`Logical BlockSize - 512`	The rest of the logical block, if any, is reserved. Set to zero

Reference: https://en.wikipedia.org/wiki/GUID_Partition_Table#Protective_MBR_.28LBA_0.29

Volume Boot Record (VBR)

A **Volume Boot Record** (**VBR**) is the first sector of a partition (opposite to **MBR** which is the first sector of a hard disk).

VBR (just like **MBR**) also contain some code and data, but it's far less standard.

The code is always OS specific, but in common all versions does the same: locate the kernel on the partition, load and execute it.

A really good example for **VBR** is the original *DOS bootsector*, which used FAT and loaded IO.SYS and MSDOS.SYS from the root directory.

Source: OS Dev Wiki http://wiki.osdev.org/Volume_Boot_Record

2.3 Bootkits (Boot Rootkits)

eEye BootRoot

eEye BootRoot – Our Solution

26

- eEye BootRoot was the first pubic proof-of-concept bootkit developed by Derek Soeder and Ryan Permeh
- Hooked INT 13h (Disk access ISR) to patch OSLOADER
- OSLOADER patch was able to modify OS further, e.g. patch boot drivers

Chronology

OS Kernel Rootkits (~ 1999+)

- Research: NTRootkit, SucKIT, adore, knark
- In-the-Wild: HackerDefender, Haxdoor

MBR,VBR Bootkits (~ 2005+)

- Defense: Windows DSE, Patch Guard
- Research: eEye BootRoot, BOOT-KIT, Vbootkit, Stoned Bootkit, Deep Boot, EvilCore
- In-the-Wild: Mebroot, TDL4, FIN1, Rovnix, Olmasco, XPAJ, Gapz, Petya and Goldeneye

BIOS Rootkits (~ 2006+)

- Research: Heasman's ACPI and PCI OpROM Rootkits, Clear Hat SMM Rootkit, Phrack 65 SMM, Persistent BIOS Infection, Phrack 66 "A Real SMM Rootkit", Rakshasa
- In-the-Wild: IceLord BIOS Rootkit, Mebromi, ANT catalog

UEFI Bootkits (~ 2012+)

- EFI/UEFI (support in Windows Vista, Windows 7, Server 2008)
- Research: Andrea Allievi's UEFI Bootkit, Dreamboot

(U)EFI Firmware Rootkits (~ 2012+)

- Defense: Windows 8 and UEFI Secure Boot
- Research: Angry Evil-Maid SRTM rootkit, A Tale of Secure Boot Bypass UEFI Bootkit, Project Maux, snare's Mac EFI Rootkit, Thunderstrike 1 and 2, Light Eater, firmware rootkit vs VMM, Dmytro Oleksiuk's SmmBackdoor
- In-the-Wild: HackingTeam UEFI Rootkit, Mac EFI Der Starke/DarkMatter implant, Sonic Scredriver

Types of Bootkits

MBR

- MBR Bootstrap code area modification (TDL4 here & here, Goblin, Petya & Goldeneye ransomware)
- MBR Partition Table modification (Olmasco)

VBR

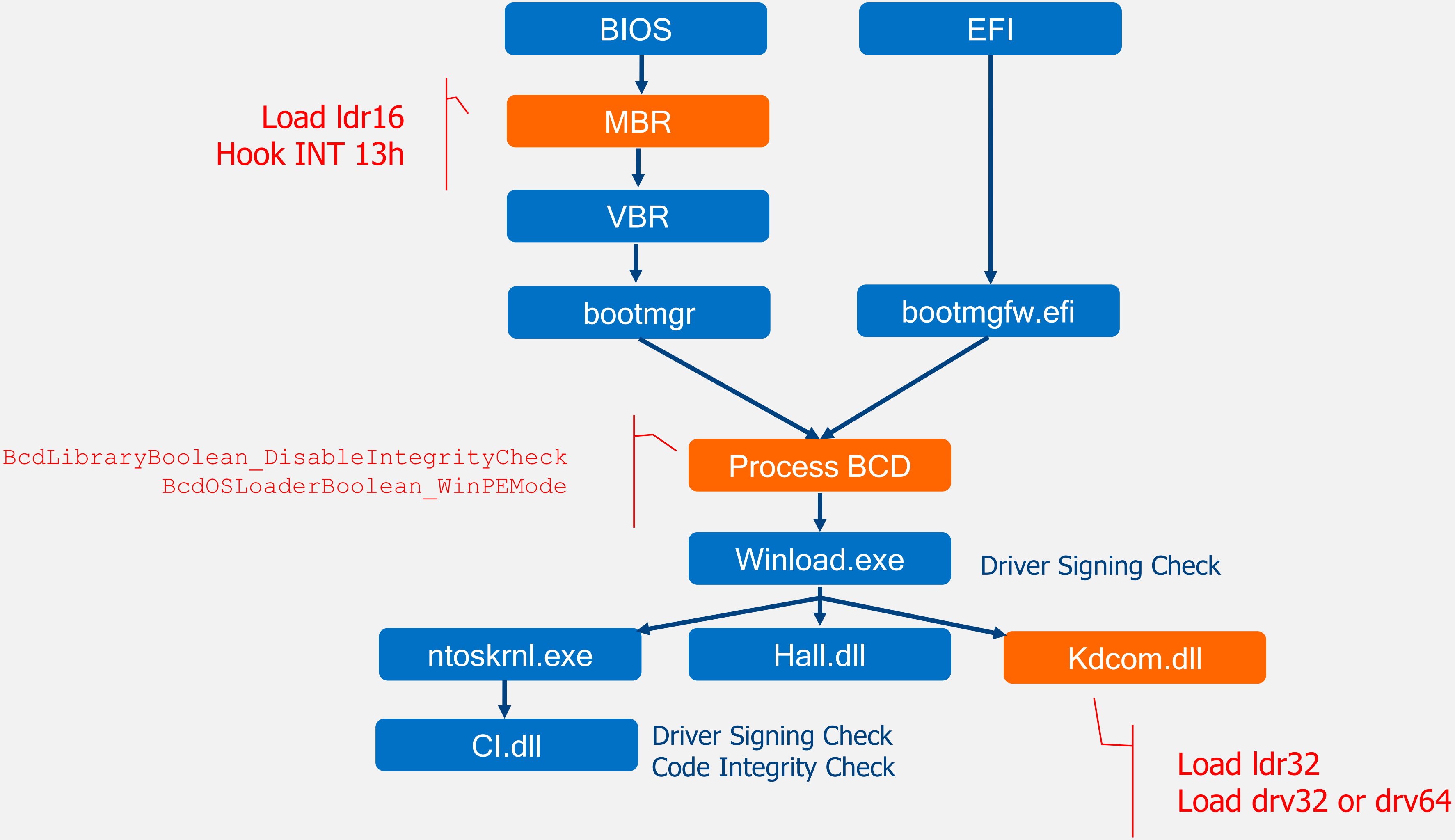
- VBR boot code (IPL) modification (FIN1, Rovnix)
- VBR BIOS Parameter Block (BPB) modification (Gapz here & here)

BIOS, UEFI

- Injecting malicious Option ROMs (Mebromi)
- Replacing EFI boot loaders
- Installing custom firmware (EFI DXE) executables (HackingTeam UEFI Rootkit)

Additional references: Bootkits step by-step

MBR Bootkit: TDL4 (Olmarik, Alureon)



Load ldr16
Hook INT 13h

BcdLibraryBoolean_DisableIntegrityCheck
BcdOSLoaderBoolean_WinPEMode

Driver Signing Check

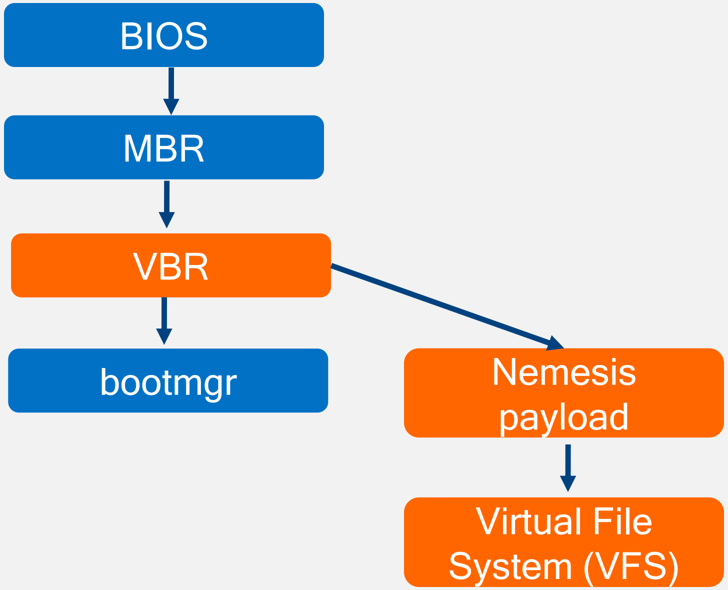
Driver Signing Check
Code Integrity Check

Load ldr32
Load drv32 or drv64

Source: What's different with TDL4 and TDL3

VBR Bootkit: Fin1

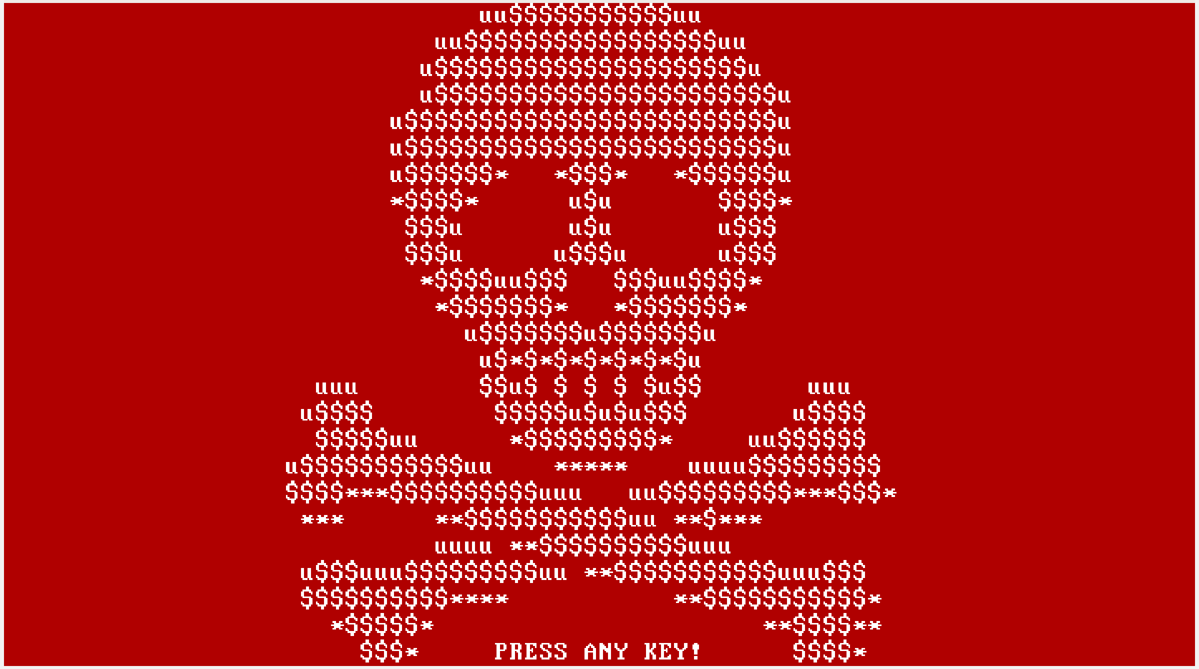
1. The installer (BOOTRASH) reads the original boot sector into memory
2. Saves an backup copy of the VBR code at 0xE sectors from the start of the partition. Apple integrity check.
3. Installer decodes the new bootstrap code from one of its embedded resources and overwrites the existing bootstrap code
4. BOOTRASH hijacks boot process in order to load the Nemesis payload before the OS.
5. BOOTRASH creates its own custom virtual file system (VFS) to store the components of the Nemesis



Source: fin1 targets boot record

Ransomware Bootkit: Petya

- Launched by Windows executable dropper
- Overwrites the beginning of the disk (including MBR) and makes an XOR encrypted backup of the original data
- The second stage is executed by the fake CHKDSK scan. After this, the file system is destroyed and cannot be read
- This first ransomware targeting disk with MBR rather than individual files



```
             uu$$$$$$$$$$$uu
          uu$$$$$$$$$$$$$$$$$uu
         u$$$$$$$$$$$$$$$$$$$$$u
         u$$$$$$$$$$$$$$$$$$$$$$$u
       u$$$$$$$$$$$$$$$$$$$$$$$$$u
       u$$$$$$$$$$$$$$$$$$$$$$$$$u
       u$$$$$$*   *$$$*   *$$$$$$u
       *$$$$*      u$u       $$$$*
        $$$u       u$u       u$$$
        $$$u      u$$$u      u$$$
         *$$$$uu$$$   $$$uu$$$$*
          *$$$$$$$*   *$$$$$$$*
            u$$$$$$$u$$$$$$$u
             u$*$*$*$*$*$*$u
  uuu        $$u$ $ $ $ $u$$       uuu
 u$$$$        $$$$$u$u$u$$$       u$$$$
  $$$$$uu      *$$$$$$$$$$*    uu$$$$$$
u$$$$$$$$$$$uu    *****    uuuu$$$$$$$$$
$$$$***$$$$$$$$$$uuu   uu$$$$$$$$$***$$$*
 ***      **$$$$$$$$$$$uu **$***
          uuuu **$$$$$$$$$$uuu
 u$$$uuu$$$$$$$$$uu **$$$$$$$$$$$uuu$$$
 $$$$$$$$$$****           **$$$$$$$$$$$*
   *$$$$$*                      **$$$$**
     $$$*     PRESS ANY KEY!      $$$$*
```

Source: Petya - Taking Ransomware To The Low Level

Mebromi BIOS Infection

Mebromi malware includes BIOS infector & MBR bootkit components

- Patches BIOS ROM binary injecting malicious ISA Option ROM with legitimate BIOS image mod utility
- Triggers SW SMI `0x29/0x2F` to erase SPI flash then write patched BIOS binary

```
# chipsec_util.py smi 0x2F
```

- Infected BIOS injects boot strap code to MBR

Exercise 2.1

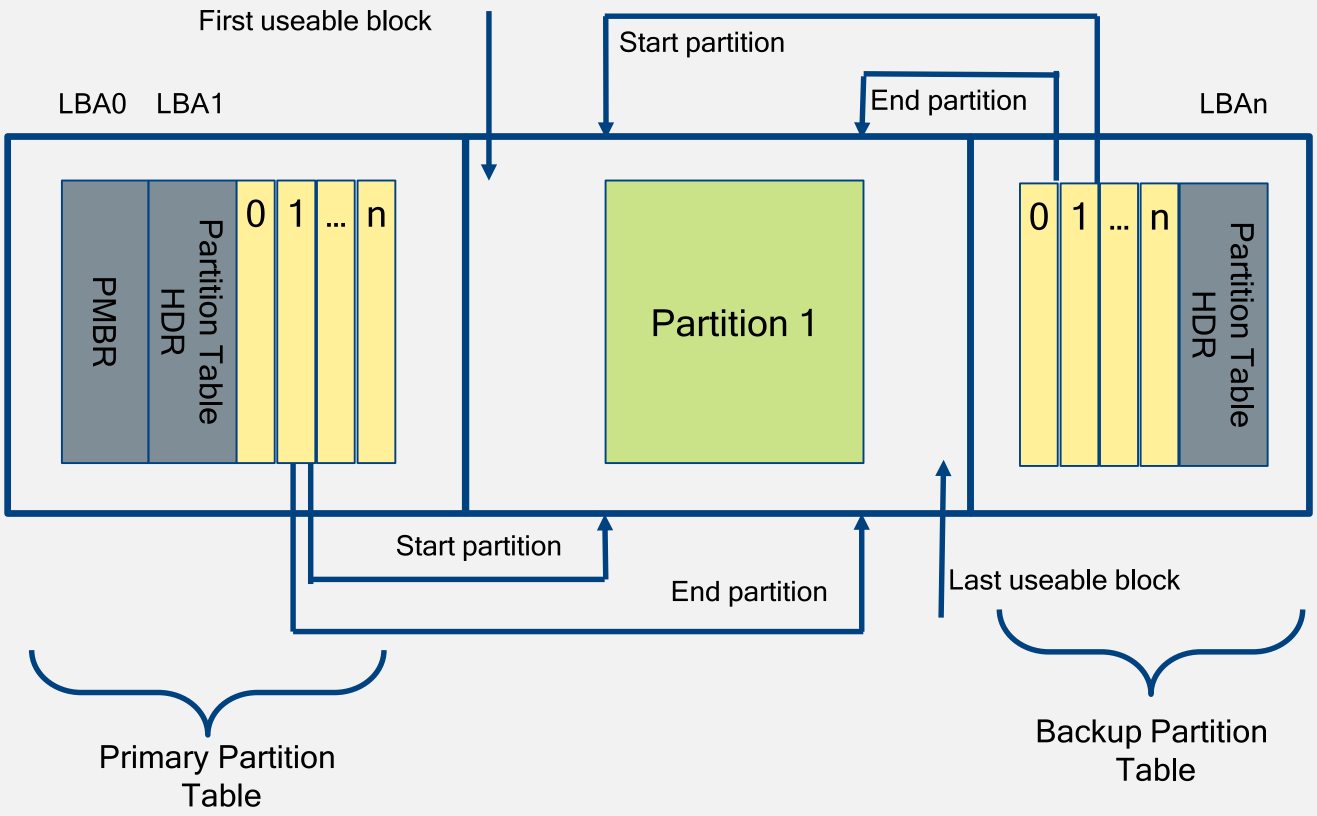
Extract and Parse Master Boot Record

2.3 GUID Partition Table

GUID Partition Table

- GUID Partition Table (GPT) is a hard disk partition table layout using GUIDs defined as part of UEFI standard
- The GPT Header defines the range of LBAs that are usable by GPT Partition Entries
- Disk GUID is a GUID that uniquely identifies the entire GPT Header and all its associated storage

GUID Partition Table Disk Layout



Source: http://kurtqiao.github.io/uefi/2014/12/31/GUID-Partition-Table-Disk.html

GUID Partition Table Header

Offset	Length	Contents
0x00	8 bytes	Signature ("EFI PART", 45h 46h 49h 20h 50h 41h 52h 54h or 0x5452415020494645ULL on littleendian machines)
0x08	4 bytes	Revision (for GPT version 1.0 (through at least UEFI version 2.3.1), the value is 00h 00h 01h 00h)
0x0C	4 bytes	Header size in little endian (in bytes, usually 5Ch 00h 00h 00h or 92 bytes)
0x10	4 bytes	CRC32 of header (offset +0 up to header size), with this field zeroed during calculation
0x14	4 bytes	Reserved; must be zero
0x18	8 bytes	Current LBA (location of this header copy)
0x20	8 bytes	Backup LBA (location of the other header copy)
0x28	8 bytes	First usable LBA for partitions (primary partition table last LBA + 1)
0x30	8 bytes	Last usable LBA (secondary partition table first LBA 1)
0x38	bytes	Disk GUID (also referred as UUID on UNIXes)
0x48	8 bytes	Starting LBA of array of partition entries (always 2 in primary copy)
0x50	4 bytes	Number of partition entries in array
0x54	4 bytes	Size of a single partition entry (usually 80h or 128)
0x58	4 bytes	CRC32 of partition array
0x5C	*	Reserved; must be zeroes for the rest of the block (420 bytes for a sector size of 512 bytes; but can be more with larger sector sizes)

GPT Entry

Source: http://ntfs.com/guid-part-table.htm

GUID Partition Table Entry

Offset	Length	Contents
0 (0x00)	16 bytes	Partition type GUID
16 (0x10)	16 bytes	Unique partition GUID
32 (0x20)	8 bytes	First LBA (little endian)
40 (0x28)	8 bytes	Last LBA (inclusive, usually odd)
48 (0x30)	8 bytes	Attribute flags (e.g. bit 60 denotes readonly)
56 (0x38)	72 bytes	Partition name (36 UTF16LE code units)

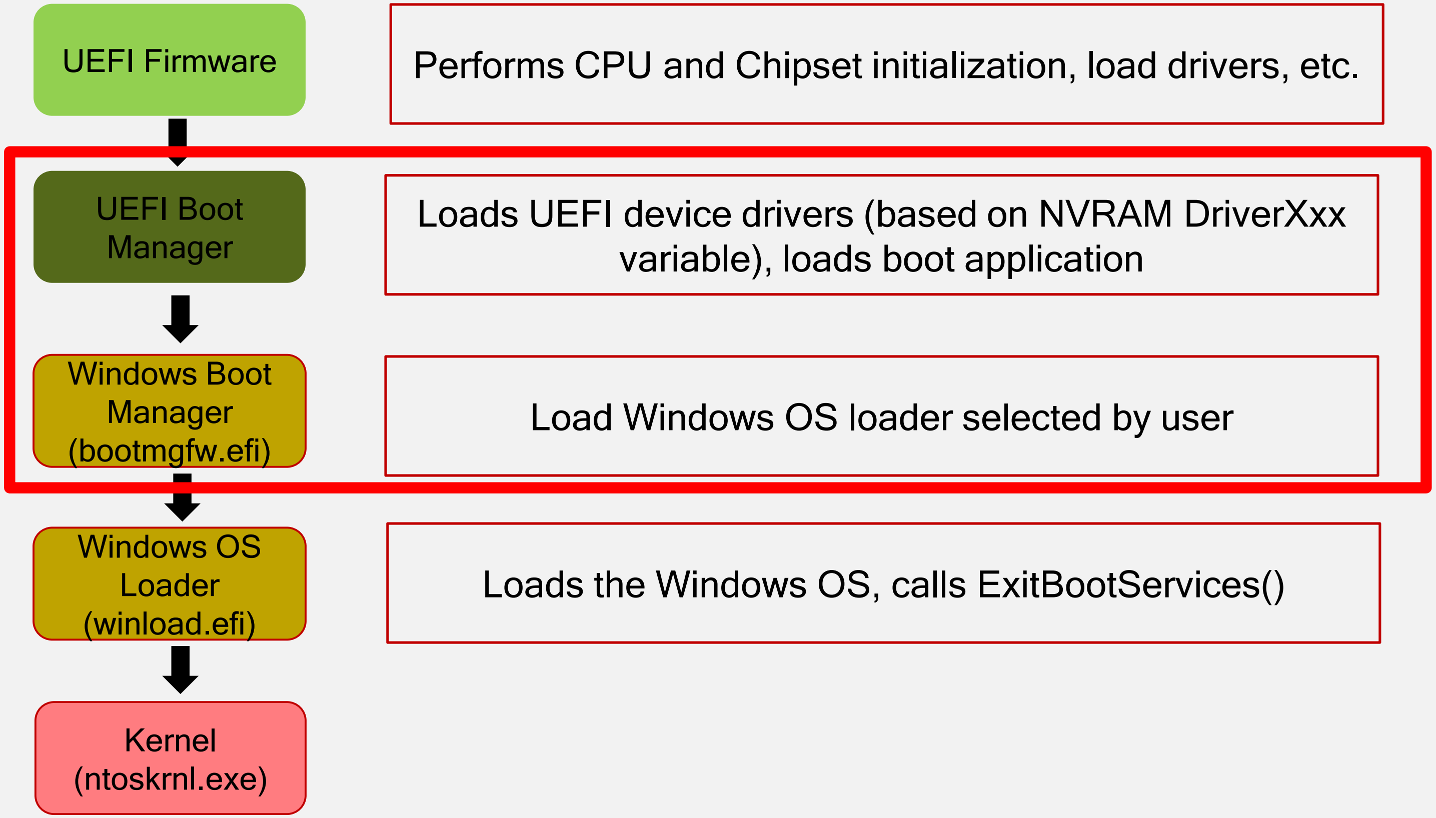
Source: http://www.datarecovery.institute/guid-partition-table-windows-10/

Exercise 2.2

Access ESP from Linux, UEFI shell and Windows; find OS boot loaders and GPT

2.4 UEFI Bootkits

UEFI Based Windows Boot



UEFI Firmware

Performs CPU and Chipset initialization, load drivers, etc.

UEFI Boot Manager

Loads UEFI device drivers (based on NVRAM DriverXxx variable), loads boot application

Windows Boot Manager (bootmgfw.efi)

Load Windows OS loader selected by user

Windows OS Loader (winload.efi)

Loads the Windows OS, calls ExitBootServices()

Kernel (ntoskrnl.exe)

Source: Windows Boot Environment by Murali Ravirala

Types of UEFI Bootkits

- **Replacing Windows Boot Manager**

  EFI System Partition (ESP) on Fixed Drive

  `ESP\EFI\Microsoft\Boot\bootmgfw.efi`

  UEFI technology: say hello to the Windows 8 bootkit! by ITSEC

- **Replacing Fallback Boot Loader**

  `ESP\EFI\Boot\bootx64.efi`

  Dreamboot by Sébastien Kaczmarek, QUARKSLAB

- **Adding New Boot Loader (bootkit.efi)**

  Modified `BootOrder / Boot####` EFI variables

- **Adding/Replacing DXE Driver**

  Stored on Fixed Drive

  Not embedded in Firmware Volume (FV) in ROM

  Modified `DriverOrder / Driver####` EFI variables

Types of UEFI Bootkits

- **Patching UEFI "Option ROM"**

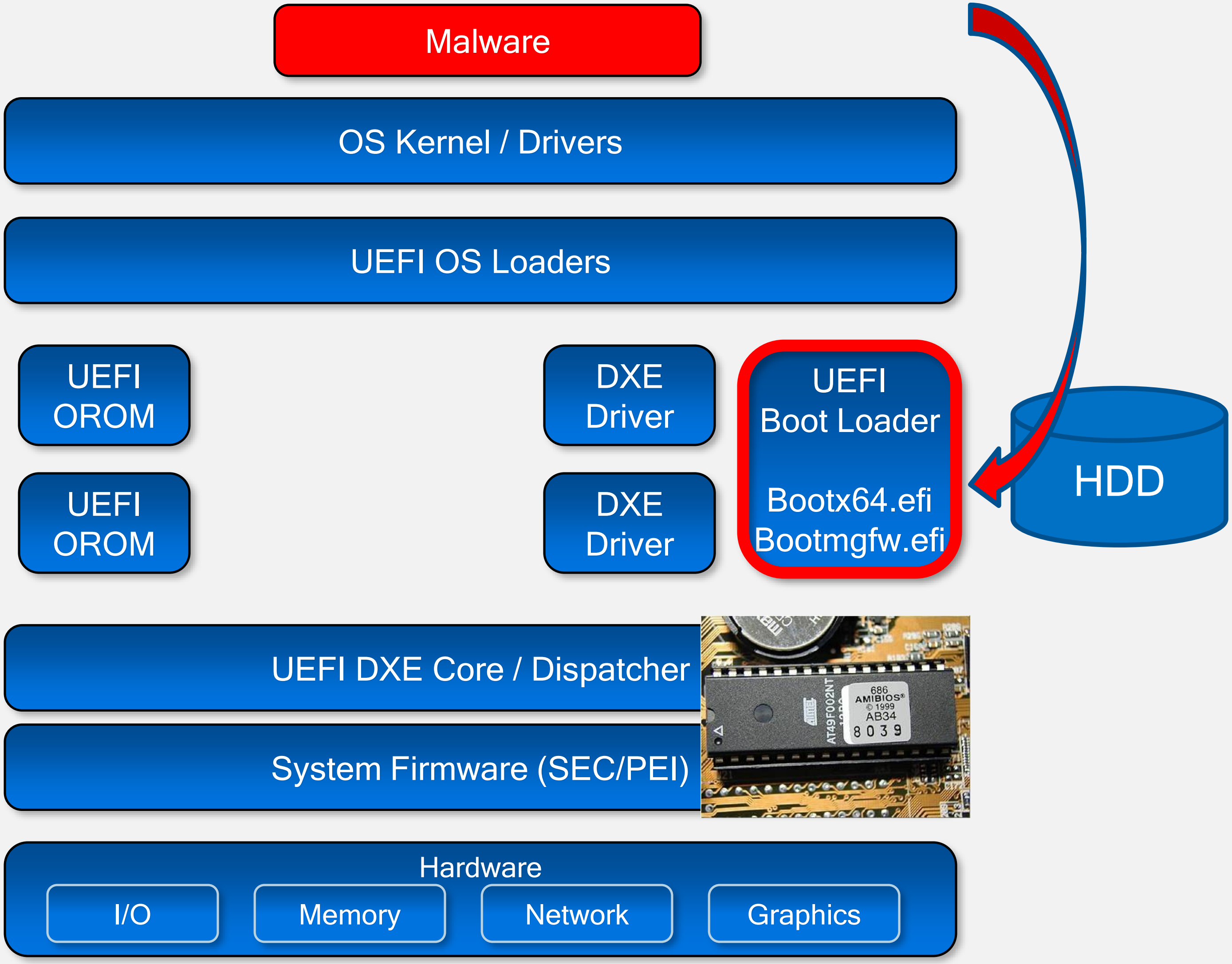
  UEFI DXE Driver in Add-On Card (Network, Storage..)

  Non-Embedded in FV in ROM

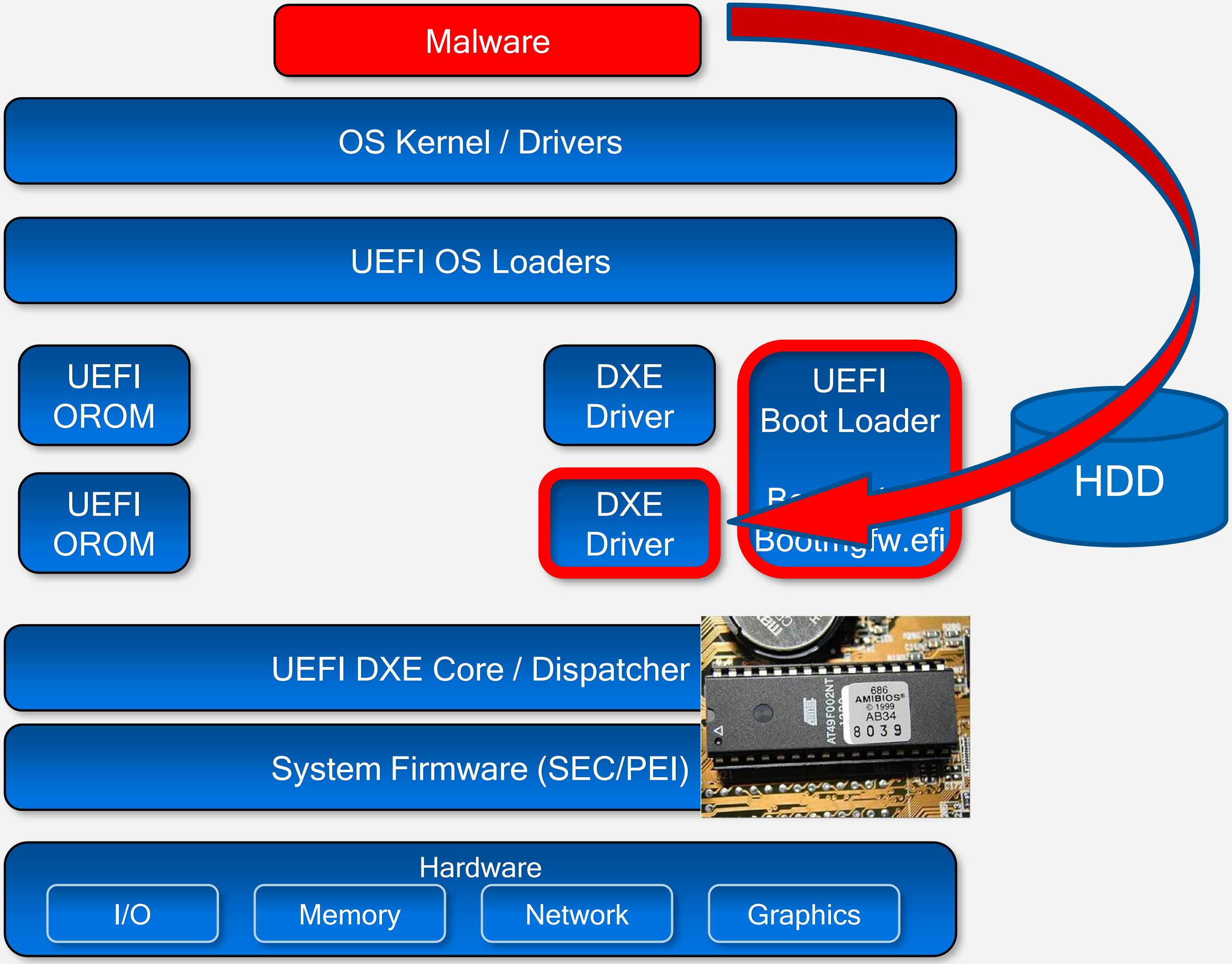
  Mac EFI Rootkits by @snare, Black Hat USA 2012

  Thunderstrike Mac EFI Rootkit by Trammell Hudson
- **Replacing OS Loaders (winload.efi, winresume.efi)**
- **Patching GUID Partition Table (GPT)**

Replacing Boot Loaders



Adding/Replacing DXE Drivers



Malware

OS Kernel / Drivers

UEFI OS Loaders

UEFI OROM

UEFI OROM

DXE Driver

DXE Driver

UEFI Boot Loader

[illegible]
Bootmgfw.efi

HDD

UEFI DXE Core / Dispatcher

System Firmware (SEC/PEI)

Hardware

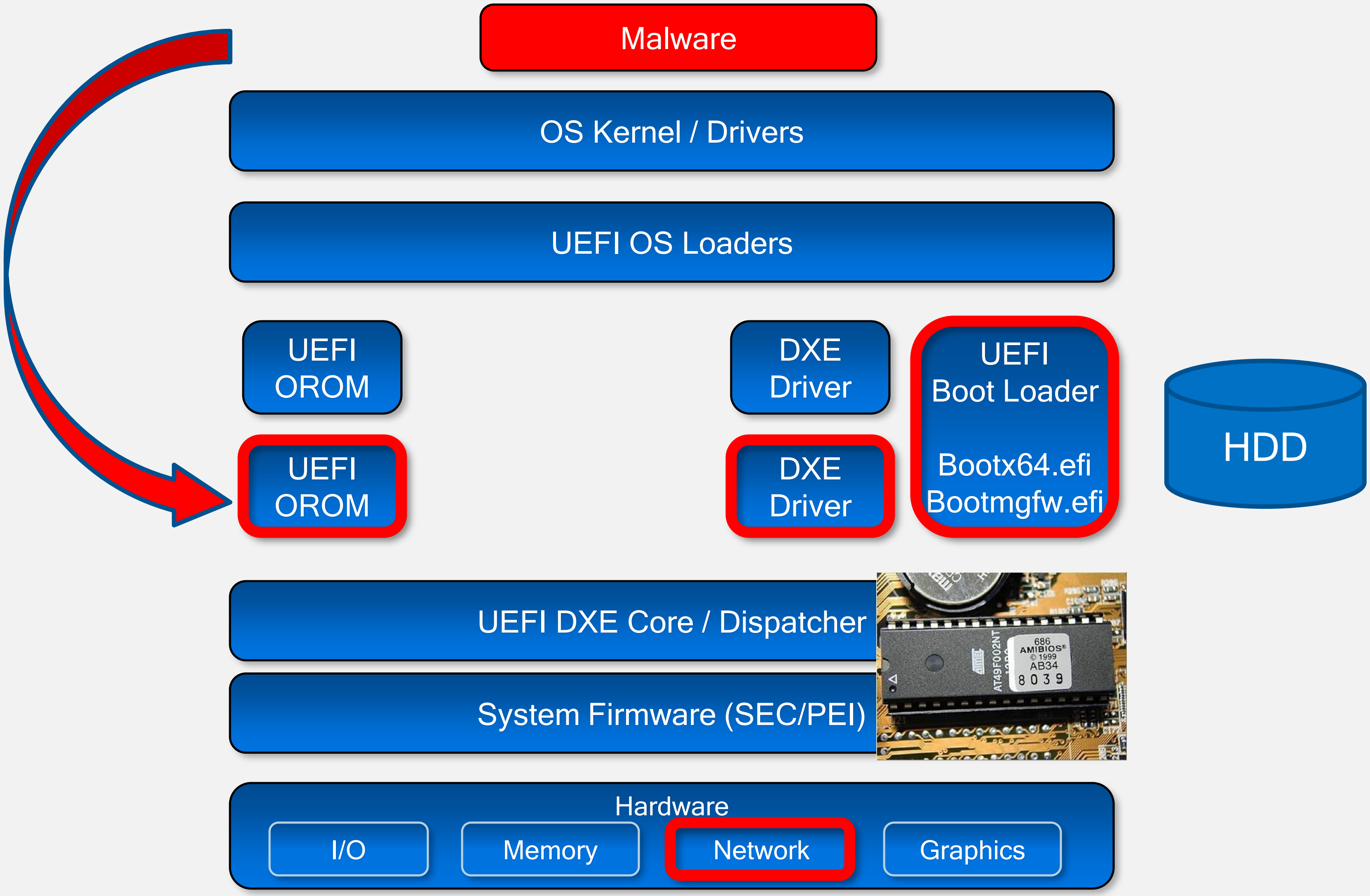
I/O

Memory

Network

Graphics

Replacing DXE Option ROM Drivers



Malware

OS Kernel / Drivers

UEFI OS Loaders

UEFI OROM

DXE Driver

UEFI Boot Loader

Bootx64.efi

Bootmgfw.efi

HDD

UEFI OROM

DXE Driver

UEFI DXE Core / Dispatcher

System Firmware (SEC/PEI)

Hardware

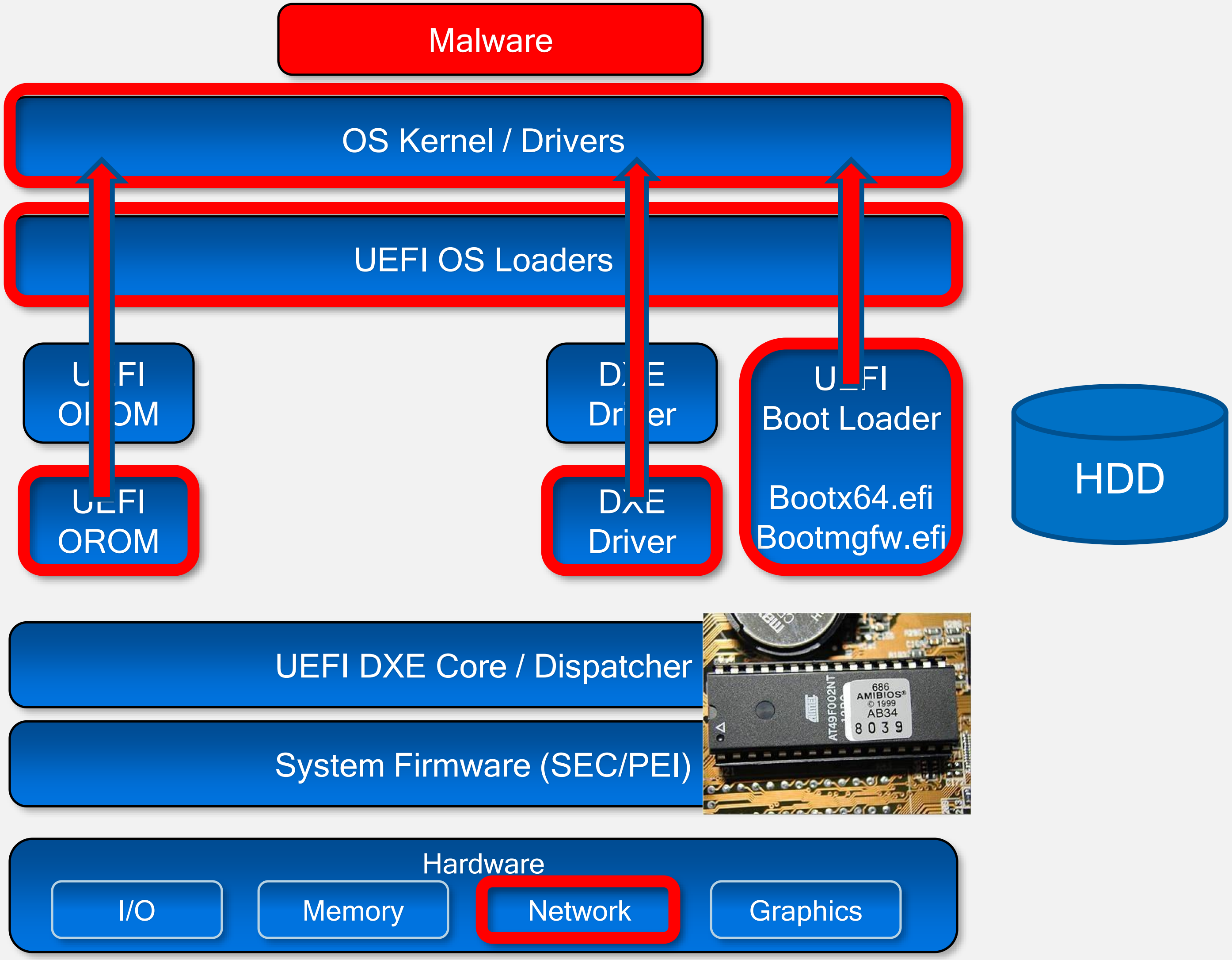
I/O

Memory

Network

Graphics

UEFI Bootkits

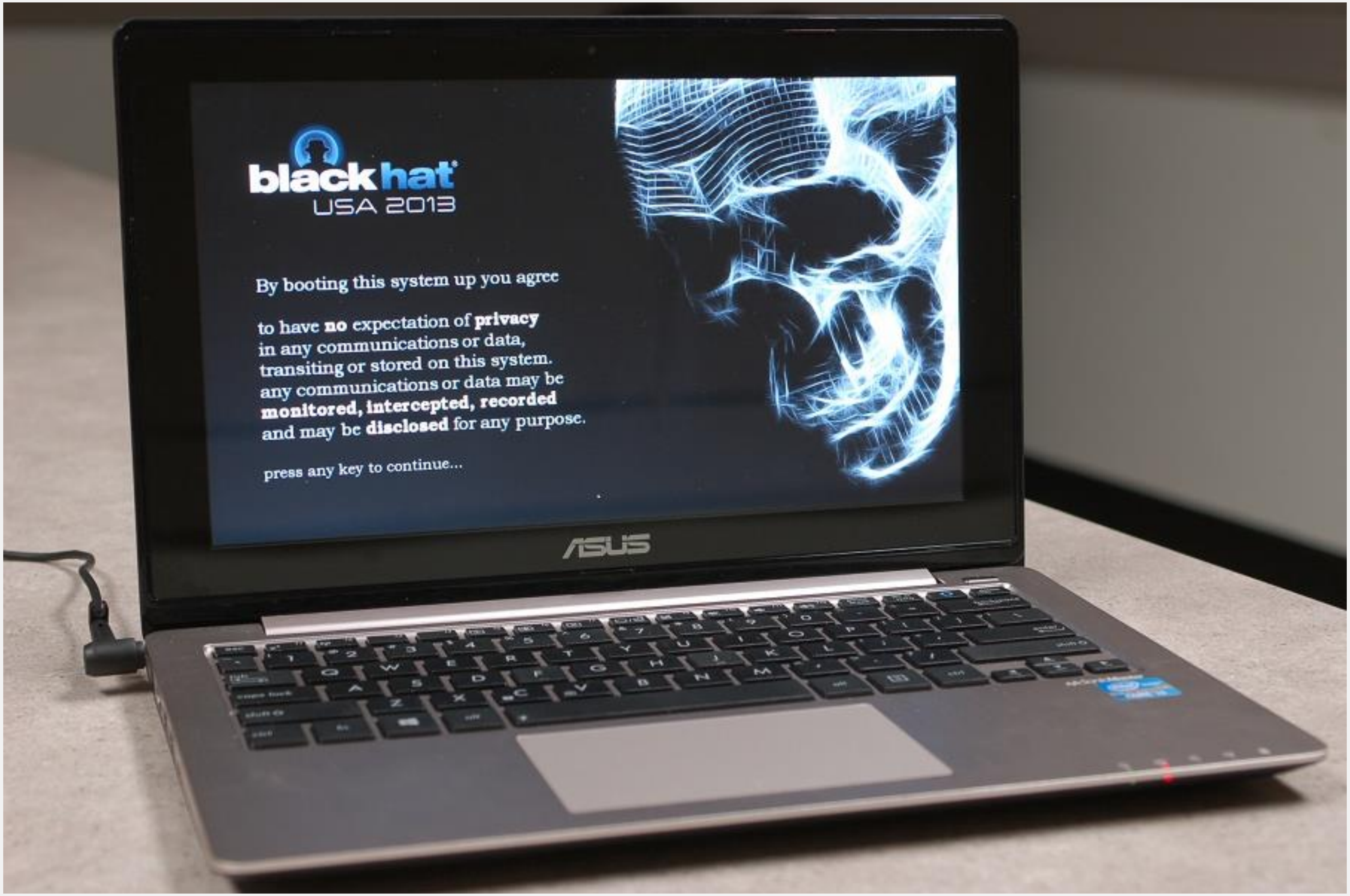


UEFI Bootkit: Dreamboot

- Replaces Windows 8 loader
- Patches OS to disable kernel protections
    - Disables DEP (NX flag)
    - Disables Windows Patch Guard
- Open source: https://github.com/quarkslab/dreamboot

Source: UEFI and Dreamboot by Sebastien Kaczmarek

Windows 8 UEFI Bootkit



By booting this system up you agree

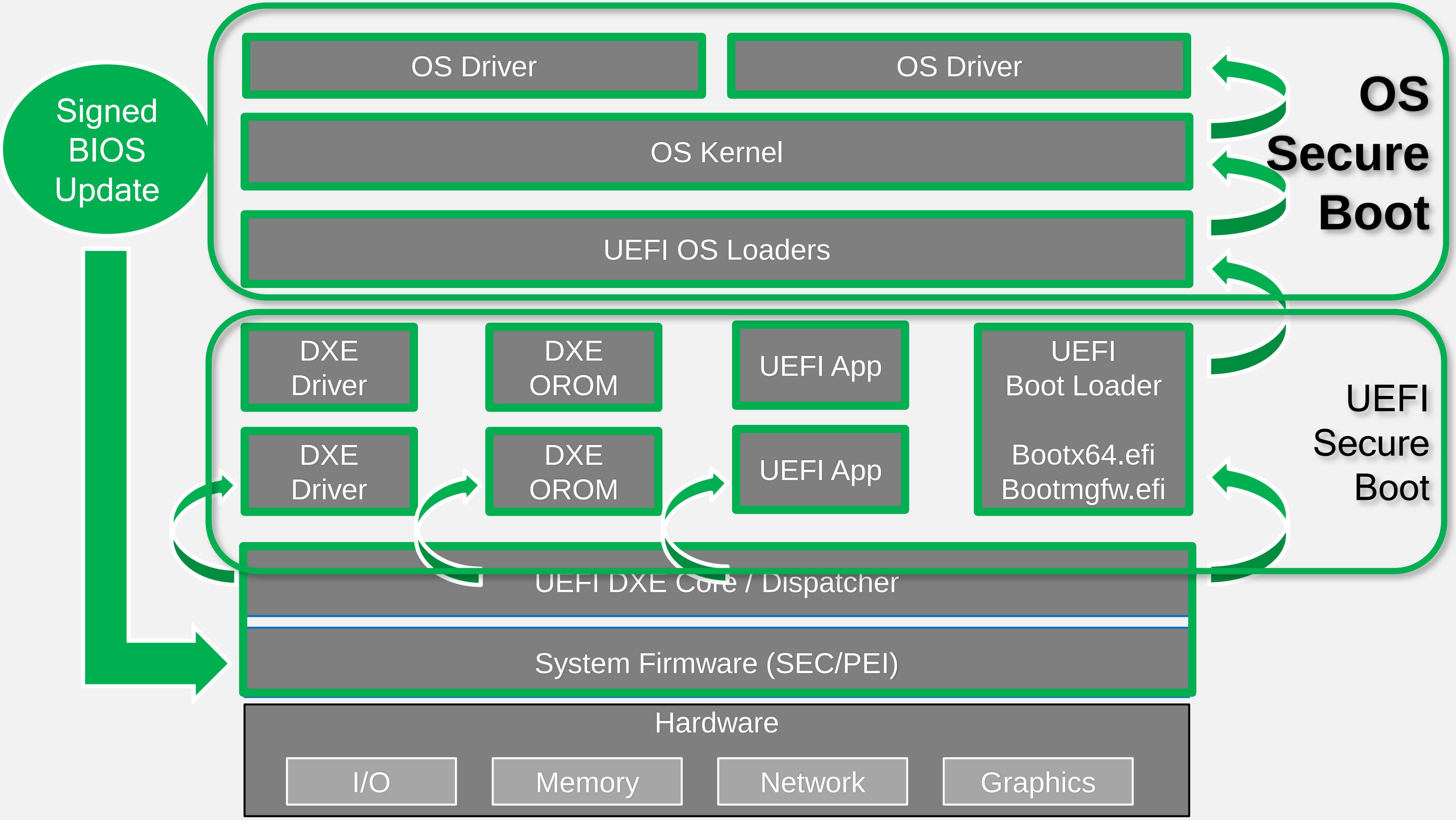
to have **no** expectation of **privacy** in any communications or data, transiting or stored on this system. any communications or data may be **monitored, intercepted, recorded** and may be **disclosed** for any purpose.

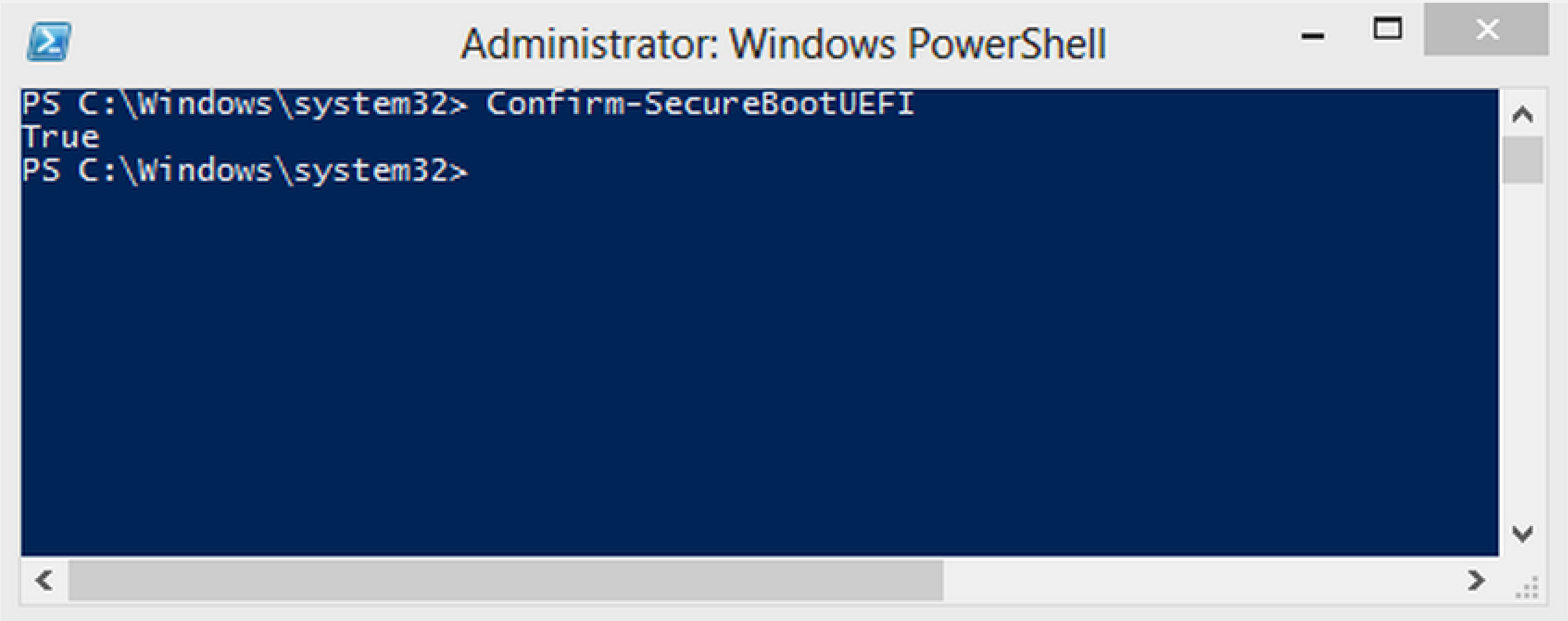
press any key to continue...

Source: A Tale of One Software Bypass of Windows 8 Secure Boot

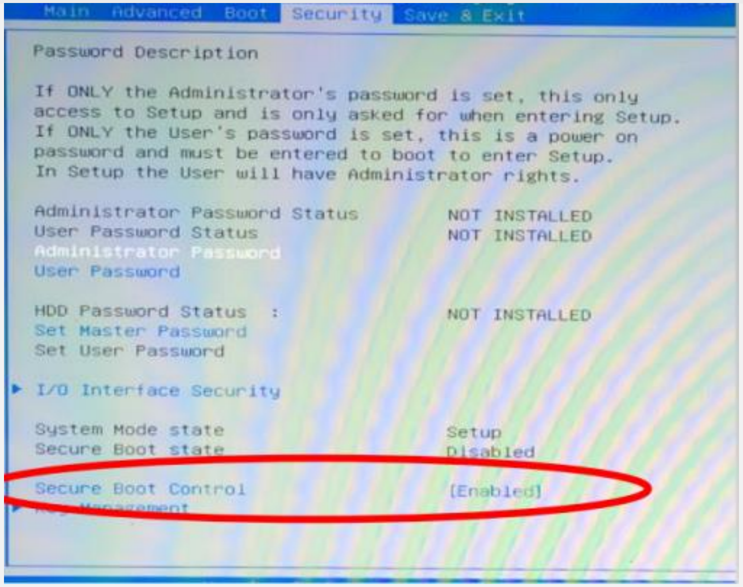
2.5 UEFI Secure Boot

Secure Boot (UEFI + OS)

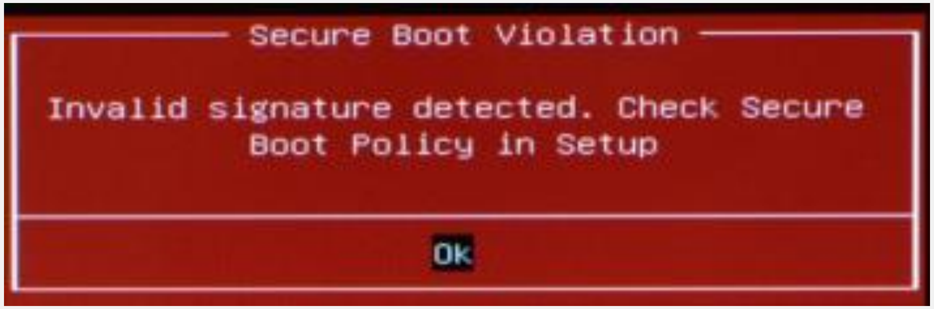




```
PS C:\Windows\system32> Confirm-SecureBootUEFI
True
PS C:\Windows\system32>
```



```
Password Description

If ONLY the Administrator's password is set, this only
access to Setup and is only asked for when entering Setup.
If ONLY the User's password is set, this is a power on
password and must be entered to boot to enter Setup.
In Setup the User will have Administrator rights.

Administrator Password Status          NOT INSTALLED
User Password Status                   NOT INSTALLED
Administrator Password
User Password

HDD Password Status  :                 NOT INSTALLED
Set Master Password
Set User Password

▶ I/O Interface Security

System Mode state                      Setup
Secure Boot state                      Disabled

Secure Boot Control                    [Enabled]
▶ Key Management
```



```
Secure Boot Violation

Invalid signature detected. Check Secure
Boot Policy in Setup
```

UEFI Secure Boot Configuration

SecureBoot

Enables/disables image signature checks

SetupMode

PK is installed (**`USER_MODE`**) or not (**`SETUP_MODE`**)

`SETUP_MODE` allows updating KEK/db(x), self-signed PK

CustomMode

Modifiable by physically present user

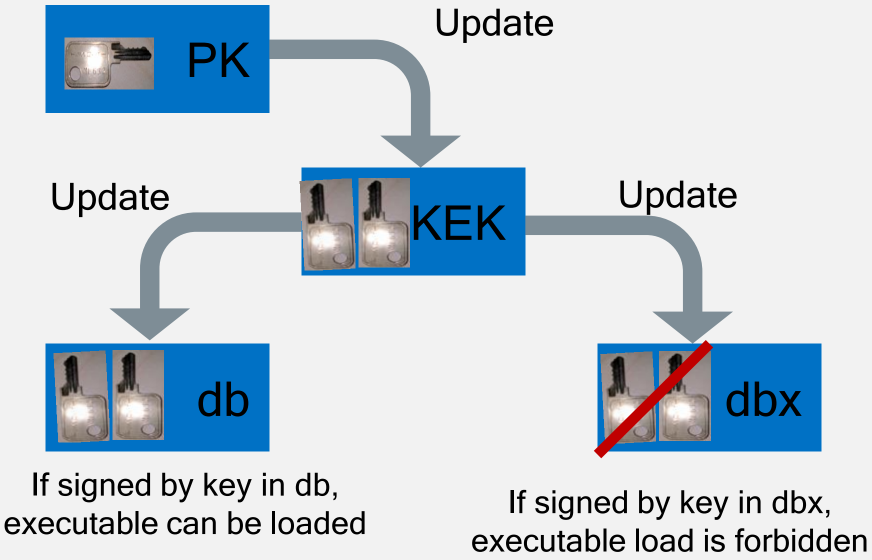
Allows updating KEK/db/dbx/PK even when PK is installed

SecureBootEnable

Global non-volatile Secure Boot Enable

Modifiable by physically present user

Secure Boot Key Hierarchy



If signed by key in db, executable can be loaded

If signed by key in dbx, executable load is forbidden

Platform Key (PK)

- Verifies KEKs
- Platform Vendor’s Cert

Key Exchange Keys (KEKs)

- Verify db and dbx
- Earlier rev’s: verifies image signatures

Authorized Database (db)

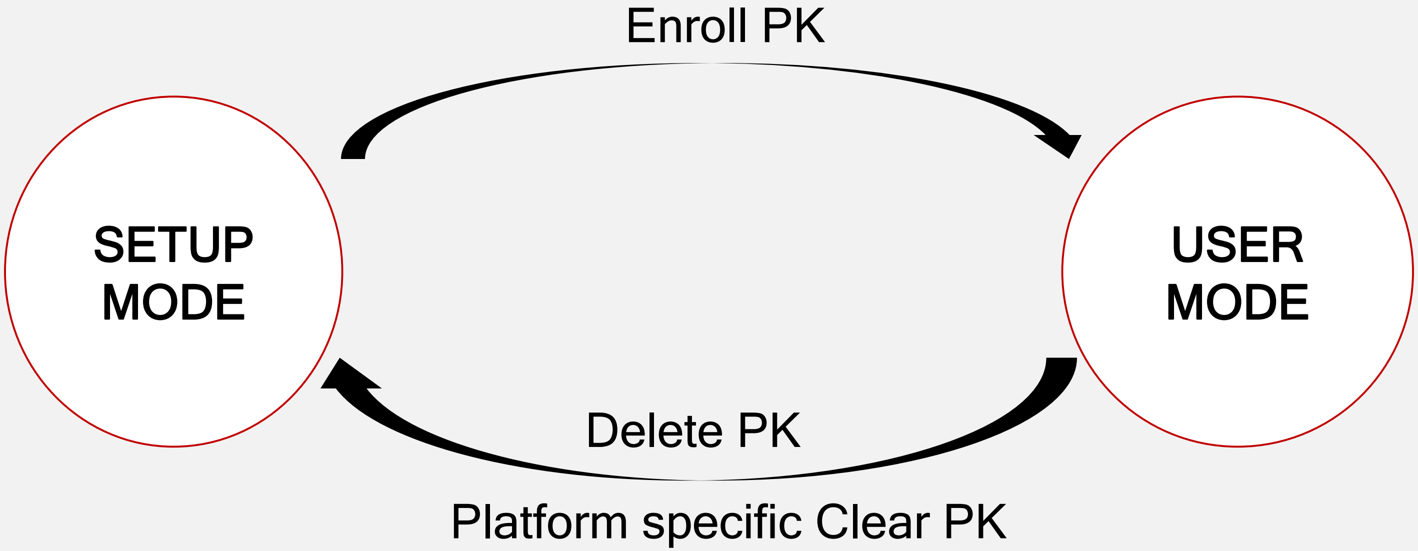
Forbidden Database (dbx)

- X509 certificates, SHA1/SHA256 hashes of allowed & revoked images
- Earlier revisions: RSA-2048 public keys, PKCS#7 signatures

PK (openssl x509 -in PK.pem –text)

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            53:41:e0:15:c4:3a:f8:a8:48:36:b9:a5:ff:69:14:88
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: CN=ASUSTeK MotherBoard PK Certificate
        Validity
            Not Before: Dec 26 23:34:50 2011 GMT
            Not After : Dec 26 23:34:49 2031 GMT
        Subject: CN=ASUSTeK MotherBoard PK Certificate
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:d9:84:15:36:c5:d4:ce:8a:a1:56:16:a0:e8:74:
..
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            2.5.29.1:
?=.../0-1+0)..U..."ASUSTeK MotherBoard PK Certificate..SA...:..H6...i..
    Signature Algorithm: sha256WithRSAEncryption
         73:27:1a:32:88:0e:db:13:8d:f5:7e:fc:94:f2:1a:27:6b:c2:
..
-----BEGIN CERTIFICATE-----
MIIDRjCCAi6gAwIBAgIQU0HgFcQ6+KhINrml/2kUiDANBgkqhkiG9w0BAQsFADAt
..
-----END CERTIFICATE-----
```

Secure Boot Modes



Source: https://www.wzdftpd.net/blog/tag/debian.html

Secure Boot Modes

`PK` variable exists in NVRAM?

Yes: Set `SetupMode` variable to `USER_MODE`

No: Set `SetupMode` variable to `SETUP_MODE`

`SecureBootEnable` variable exists in NVRAM?

Yes

- `SecureBootEnable` variable is `SECURE_BOOT_ENABLE` and `SetupMode` is `USER_MODE`? Set `SecureBoot` to `ENABLE`
- Else? Set `SecureBoot` to `DISABLE`

No

- `SetupMode` variable is `USER_MODE`? Set `SecureBoot` to `ENABLE`
- `SetupMode` variable is `SETUP_MODE`? Set `SecureBoot` to `DISABLE`

Image Verification Policies

`DxeImageVerificationLib` defines policies applied to different types of images and on security violation

```
IMAGE_FROM_FV (ALWAYS_EXECUTE), IMAGE_FROM_FIXED_MEDIA,
IMAGE_FROM_REMOVABLE_MEDIA, IMAGE_FROM_OPTION_ROM
```

```
ALWAYS_EXECUTE, NEVER_EXECUTE,
ALLOW_EXECUTE_ON_SECURITY_VIOLATION
DEFER_EXECUTE_ON_SECURITY_VIOLATION
DENY_EXECUTE_ON_SECURITY_VIOLATION
QUERY_USER_ON_SECURITY_VIOLATION
```

Let’s have a look at the Secure Boot image verification process

`SecurityPkg\Library\DxeImageVerificationLib`

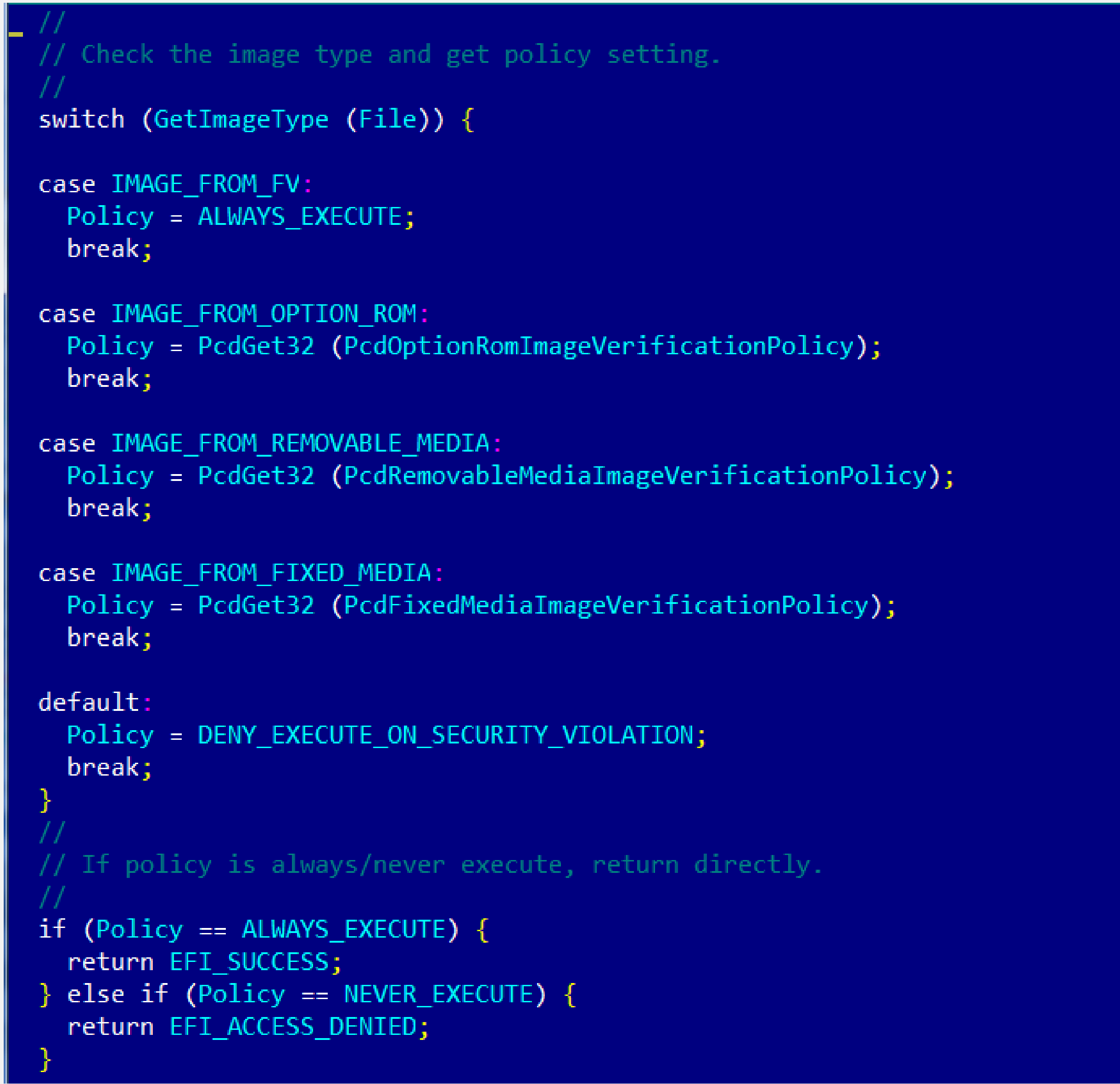
http://sourceforge.net/apps/mediawiki/tianocore/index.php?title=SecurityPkg

Verifying Policies

Image Verification Policy

(IMAGE_FROM_FV)
ALWAYS_EXECUTE?
EFI_SUCCESS

NEVER_EXECUTE?
EFI_ACCESS_DENIED



```
//
// Check the image type and get policy setting.
//
switch (GetImageType (File)) {

case IMAGE_FROM_FV:
  Policy = ALWAYS_EXECUTE;
  break;

case IMAGE_FROM_OPTION_ROM:
  Policy = PcdGet32 (PcdOptionRomImageVerificationPolicy);
  break;

case IMAGE_FROM_REMOVABLE_MEDIA:
  Policy = PcdGet32 (PcdRemovableMediaImageVerificationPolicy);
  break;

case IMAGE_FROM_FIXED_MEDIA:
  Policy = PcdGet32 (PcdFixedMediaImageVerificationPolicy);
  break;

default:
  Policy = DENY_EXECUTE_ON_SECURITY_VIOLATION;
  break;
}
//
// If policy is always/never execute, return directly.
//
if (Policy == ALWAYS_EXECUTE) {
  return EFI_SUCCESS;
} else if (Policy == NEVER_EXECUTE) {
  return EFI_ACCESS_DENIED;
}
```

Image Verification Handler

SecureBoot EFI variable doesn’t exist or equals to **SECURE_BOOT_MODE_DISABLE?** **EFI_SUCCESS**

File is not valid PE/COFF image? **EFI_ACCESS_DENIED**

SecureBootEnable NV EFI variable doesn’t exist or equals to **SECURE_BOOT_DISABLE?** **EFI_SUCCESS**

SetupMode NV EFI variable doesn’t exist or equals to **SETUP_MODE?** **EFI_SUCCESS**

Authenticating EFI Images (EDK2)

1. **Image is not signed**
   - Image signature or SHA256 hash in **`DBX`**`?` **`EFI_ACCESS_DENIED`**
   - Image signature or SHA256 hash in **`DB`**`?` **`EFI_SUCCESS`**

2. **Image is signed**

For each signature in PE file:

- Signature verified by root/intermediate cert in **`DBX`**`?` **`EFI_ACCESS_DENIED`**
- Image signature or SHA256 hash in **`DBX`**`?` **`EFI_ACCESS_DENIED`**

For each signature in PE file:

- Signature verified by root/intermediate cert in **`KEK`** or **`DB?`** **`EFI_SUCCESS`**
- Image signature or SHA256 hash in **`DB`**`?` **`EFI_SUCCESS`**

Else **`EFI_ACCESS_DENIED`**

Open source packages for Secure Boot Flow

MdeModulePkg
LoadImage Boot Service
gBS->LoadImage
CoreLoadImage()

EFI_SECURITY_ARCH_PROTOCOL
SecurityStubDxe

SecurityStubAuthenticateState()

DxeSecurityManagementLib

RegisterSecurityHandler()
ExecuteSecurityHandlers()

SecurityPkg
DxeImageVerificationLib

DxeImageVerificationHandler()
HashPeImage()
HashPeImageByType()
VerifyWinCertificateForPkcsSignedData()
DxeImageVerificationLibImageRead()
IsSignatureFoundInDatabase()
IsPkcsSignedDataVerifiedBySignatureList()
VerifyCertPkcsSignedData()

Authenticated Variables
gRT->GetVariable

MdePkg
BasePeCoffLib

PeCoffLoaderGetImageInfo()

CryptoPkg
BaseCryptLib

Sha256Init()
Sha256Update()
Sha256Final()
Sha256GetContextSize()

AuthenticodeVerify()
Pkcs7Verify()
WrapPkcs7Data()

OpenSslLib

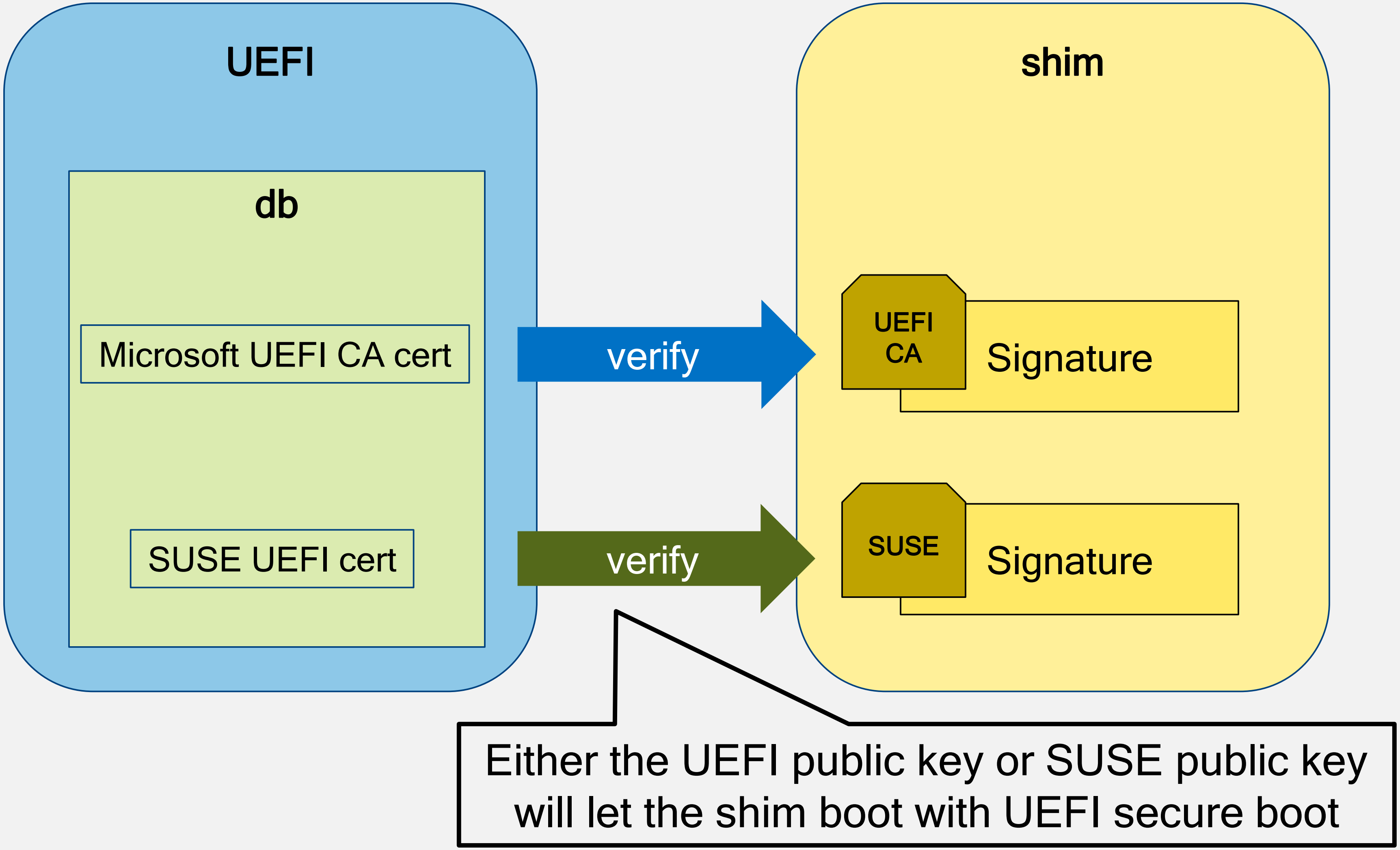
Openssl-0.9.8w

IntrinsicLib

Source: A Tour Beyond BIOS into UEFI Secure Boot by Rosenbaum, Zimmer

2.6 Linux Secure Boot

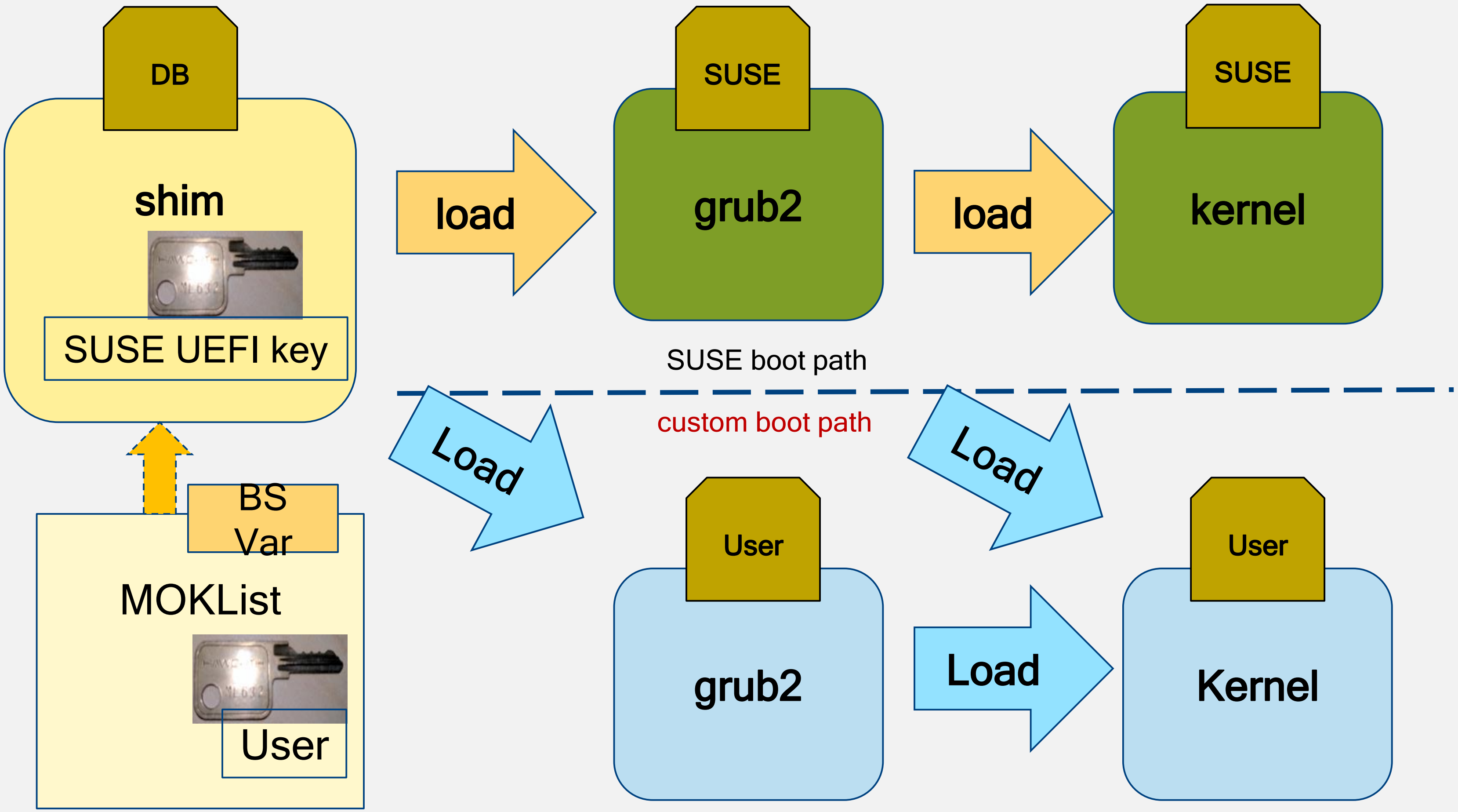
Linux Secure Boot with Shim



Either the UEFI public key or SUSE public key will let the shim boot with UEFI secure boot

Source: UEFI Open Platforms by Vincent Zimmer

Multiple OS Boot with MOK



Source: UEFI Open Platforms by Vincent Zimmer

Exercise 2.3

Secure Boot on Linux

Training materials are available on Github

https://github.com/advanced-threat-research/firmware-security-training

Yuriy Bulygin	@c7zero
Alex Bazhaniuk	@ABazhaniuk
Andrew Furtak	@a_furtak
John Loucaides	@JohnLoucaides