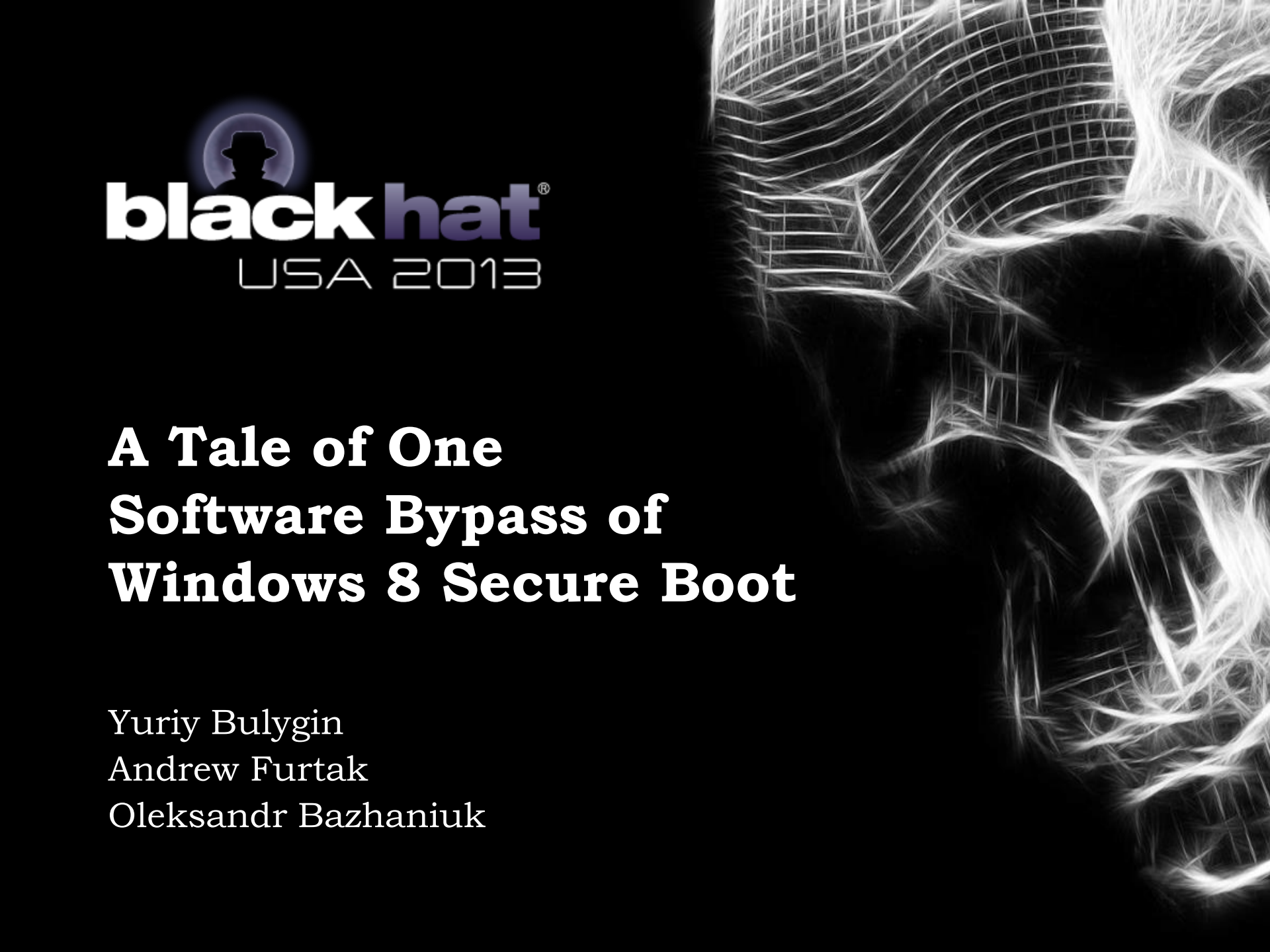


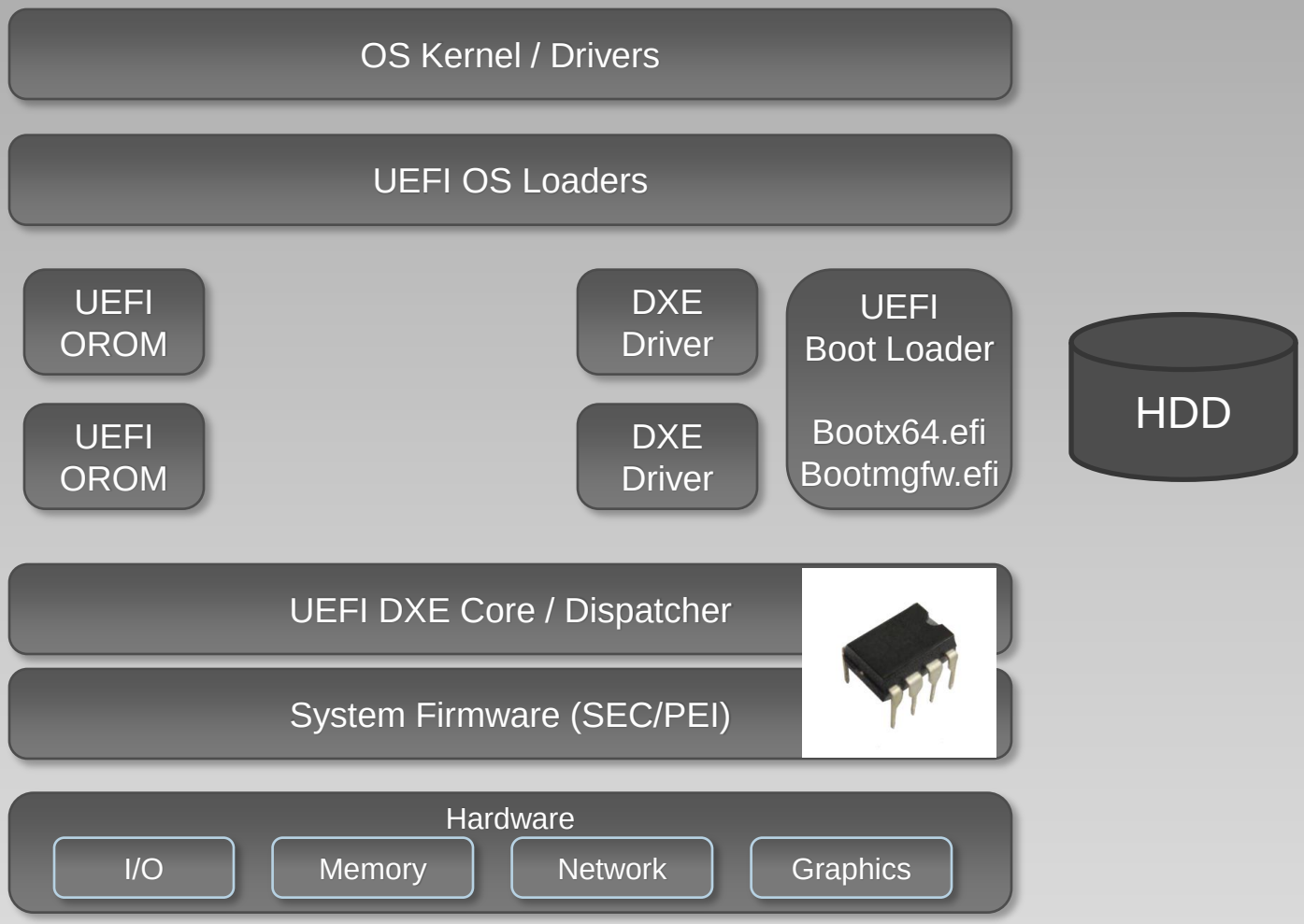
A Tale of One Software Bypass of Windows 8 Secure Boot

Yuriy Bulygin
Andrew Furtak
Oleksandr Bazhaniuk

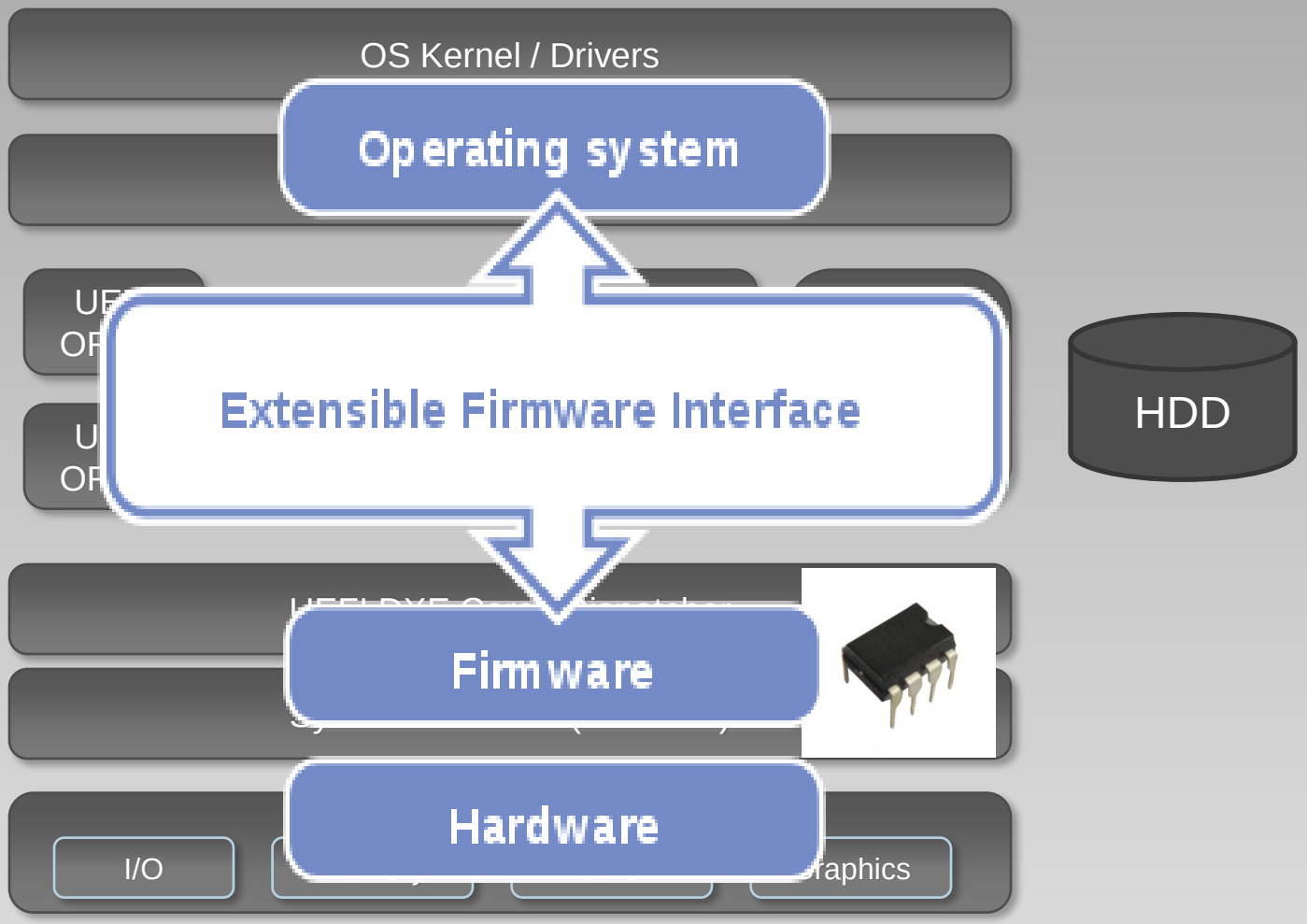
- UEFI and Bootkits
- Windows 8 Secure Boot
- Attacking Secure Boot
- Recommendations

Agenda

UEFI and Bootkits



Unified Extensible Firmware Interface (UEFI)



Unified Extensible Firmware Interface (UEFI)

Industry Standard Interface Between Firmware & OS

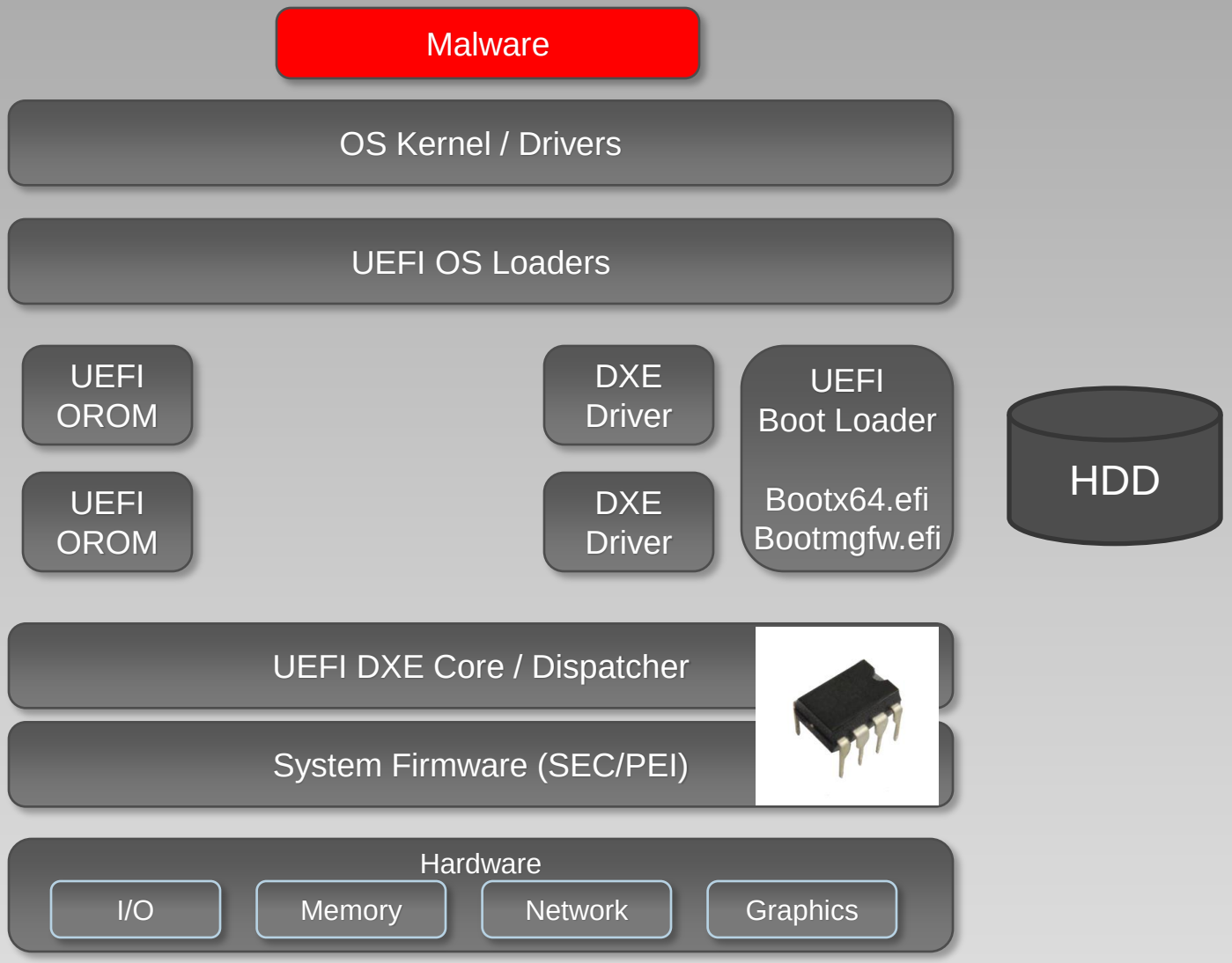
Processor Architecture and OS Independent

C Development Environment (EDK2/UDK)

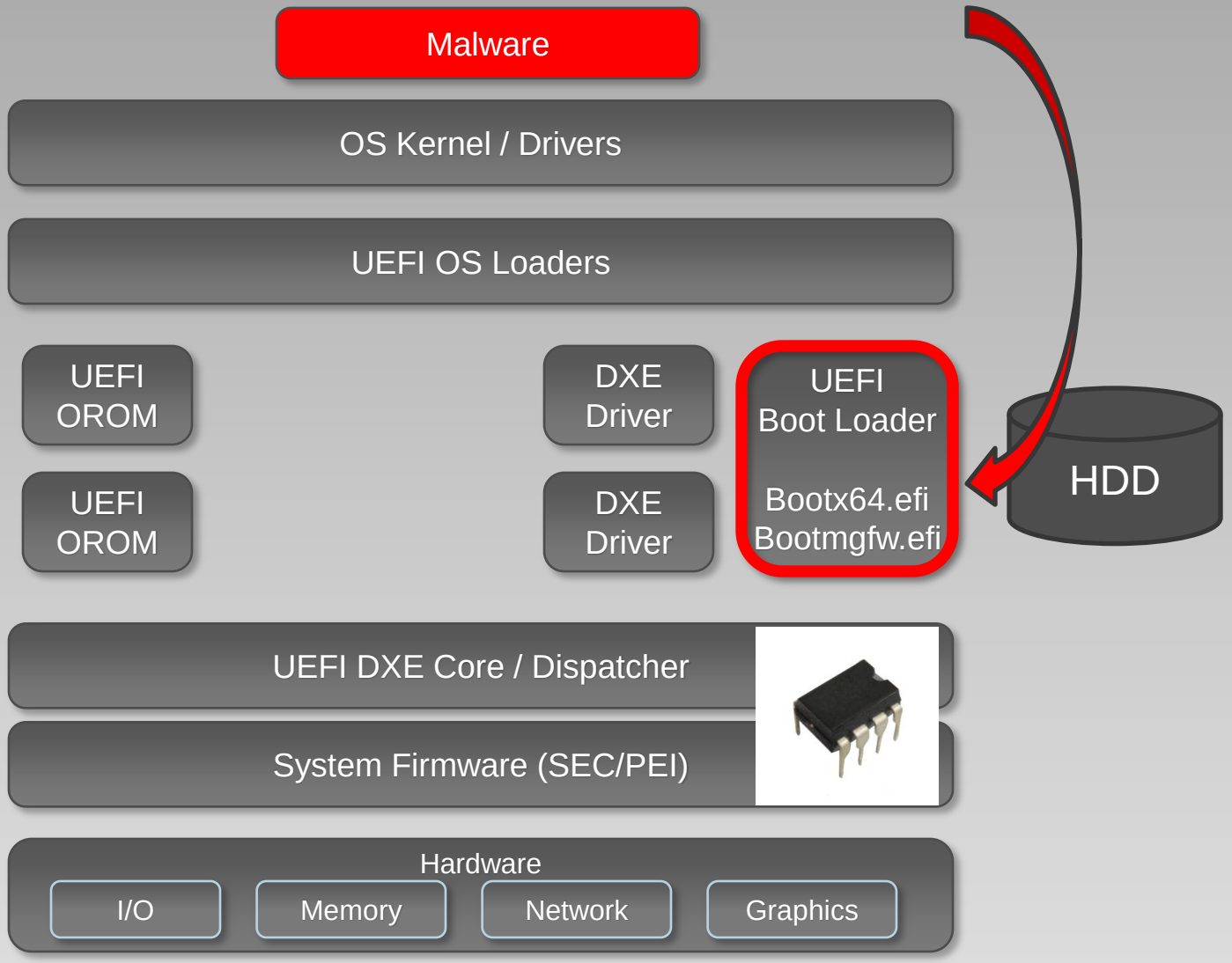
Rich GUI Pre-Boot Application Environment

Includes Modular Driver Model

Unified Extensible Firmware Interface (UEFI)



UEFI Bootkits



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

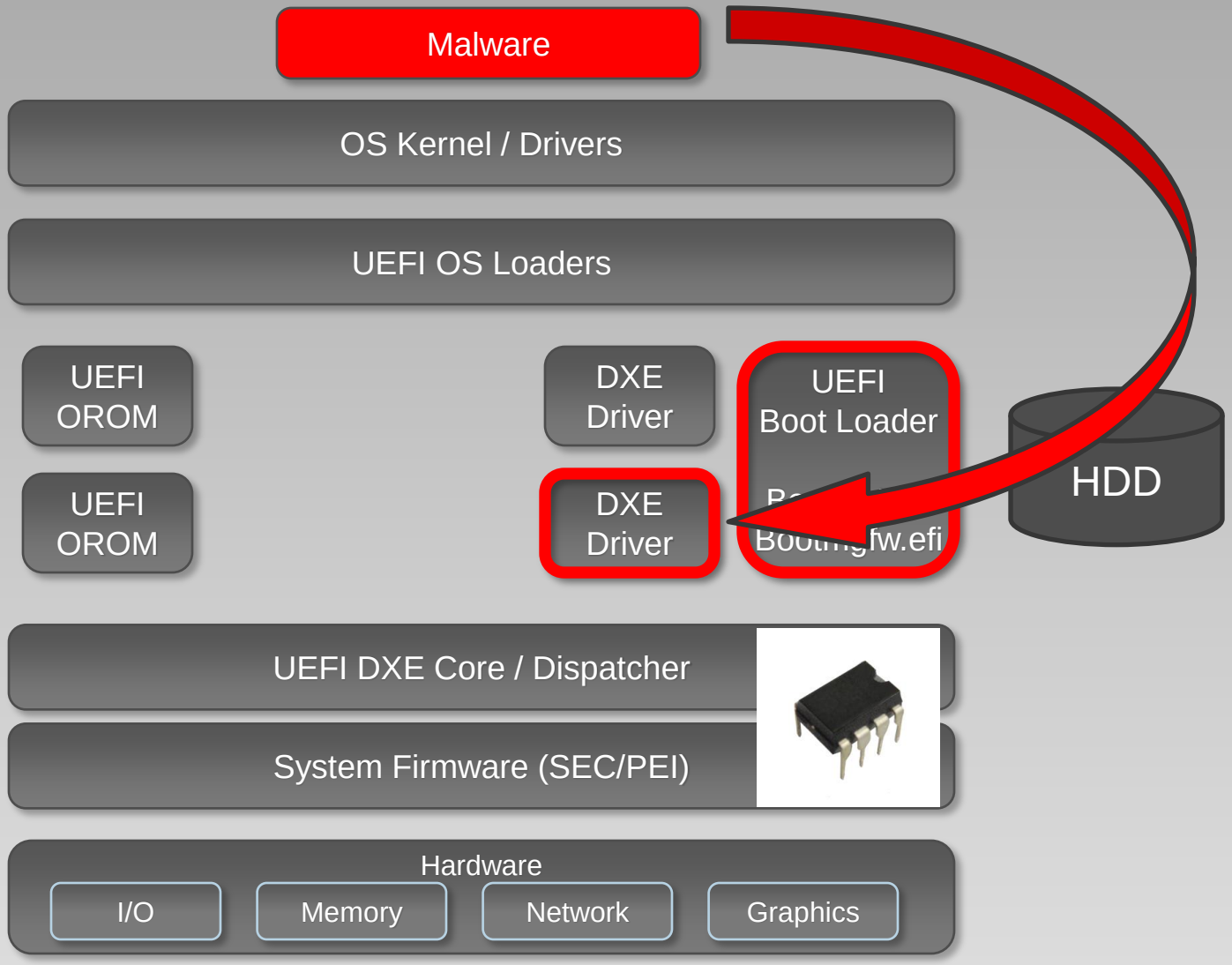
Replacing Fallback Boot Loader

ESP\EFI\Boot\bootx64.efi

UEFI and Dreamboot by Sébastien Kaczmarek, QUARKSLAB

Adding New Boot Loader (bootkit.efi)

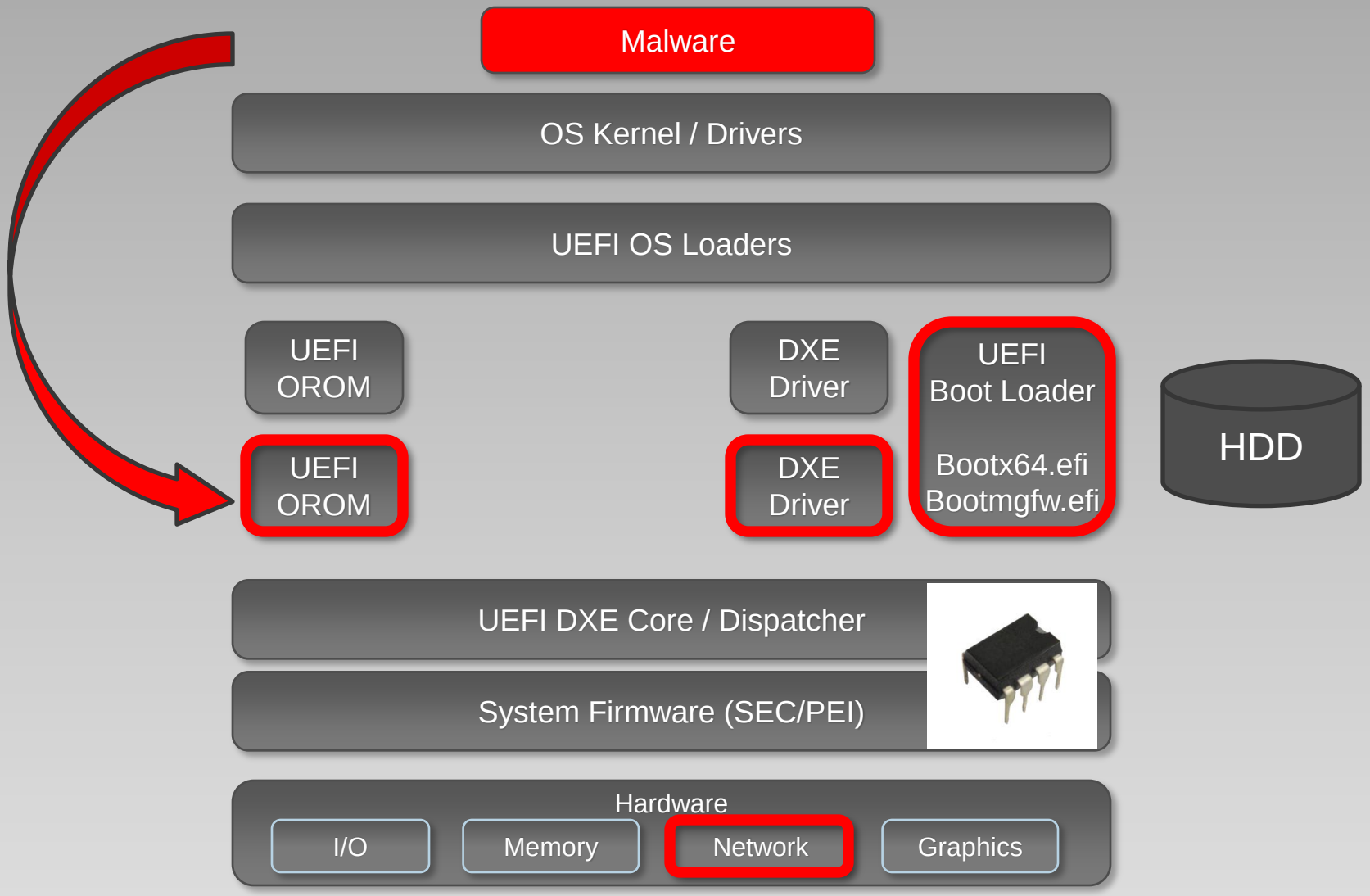
Modified BootOrder / Boot#### EFI variables



UEFI Bootkits

Adding/Replacing DXE Driver

- Stored on Fixed Drive
- Not embedded in Firmware Volume (FV) in ROM
- Modified DriverOrder + Driver#### EFI variables



UEFI Bootkits

Patching UEFI “Option ROM”

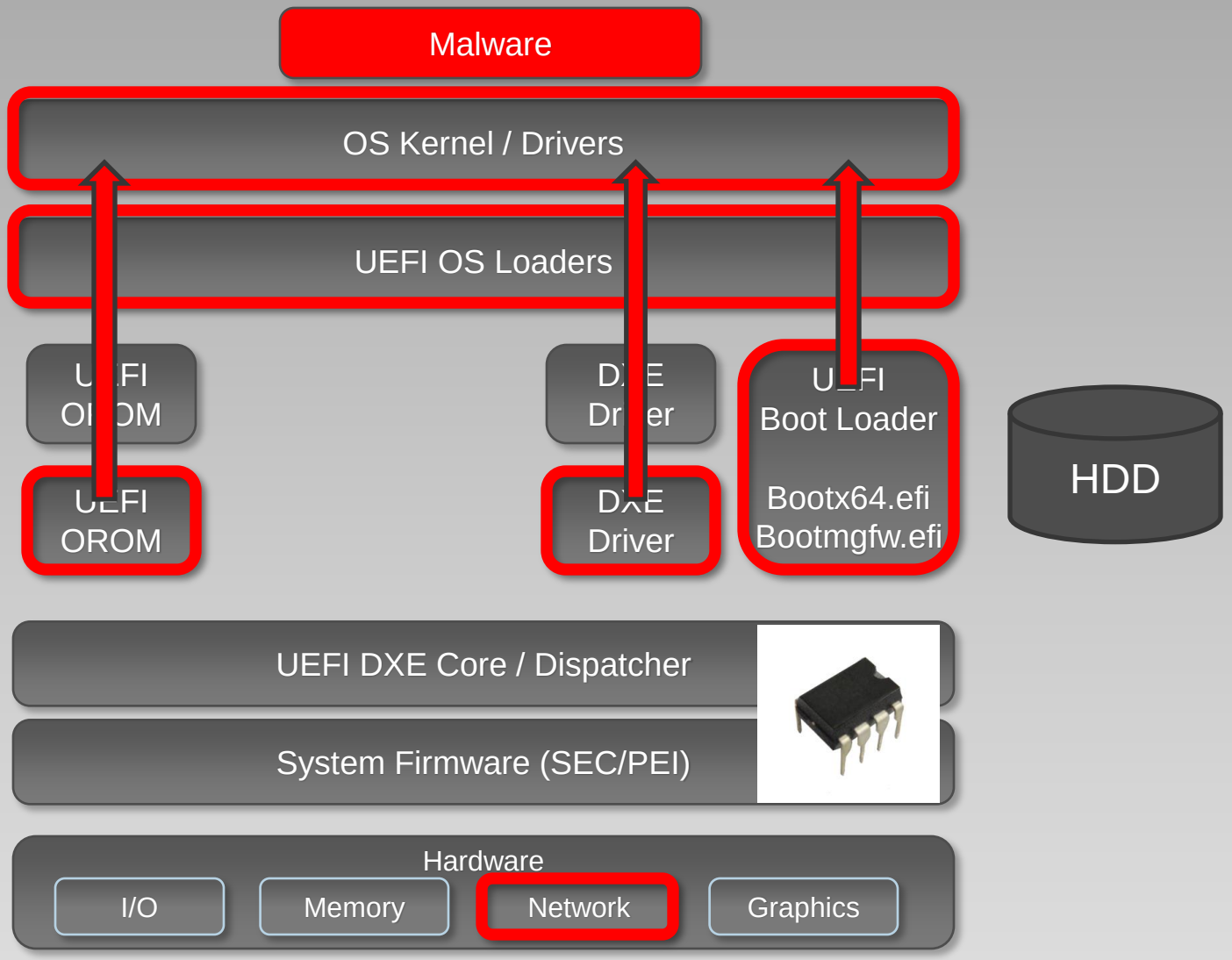
UEFI DXE Driver in Add-On Card (Network, Storage..)

Non-Embedded in FV in ROM

Mac EFI Rootkits by @snare, Black Hat USA 2012

Replacing OS Loaders (winload.efi, winresume.efi)

Patching GUID Partition Table (GPT)



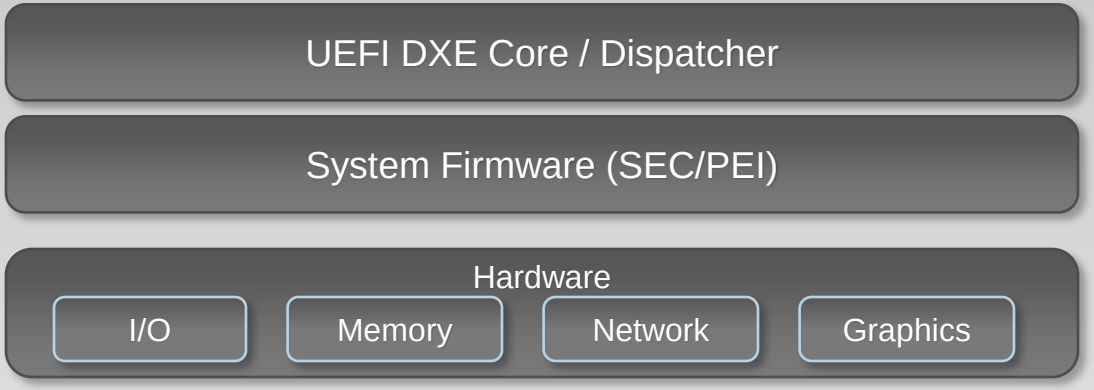
UEFI Bootkits

UEFI OS Boot Is Trivially Subverted

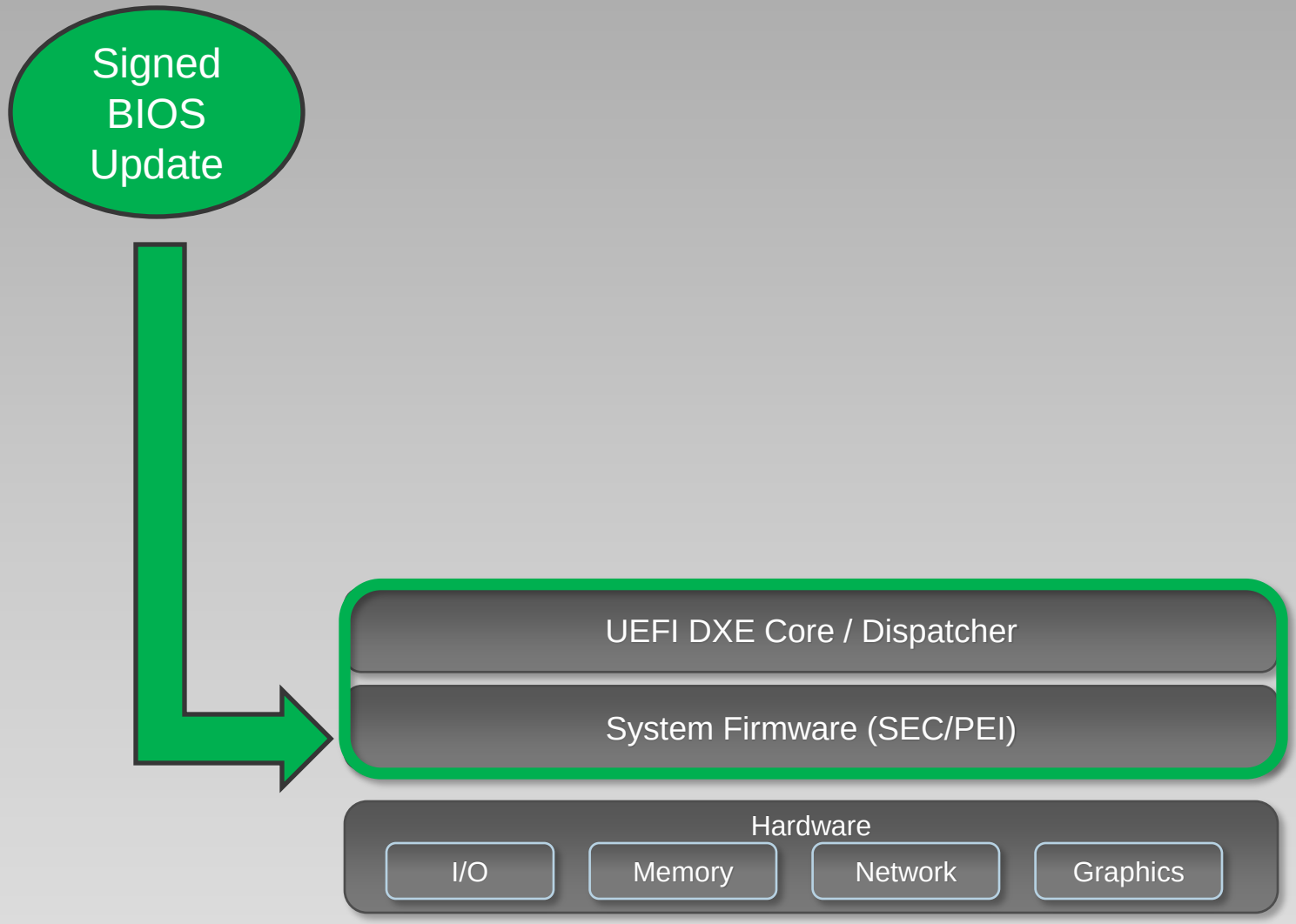
Replacing legacy OS boot with UEFI boot will also replace legacy M/VBR bootkits with UEFI bootkits.

So can bootkit problem be fixed?

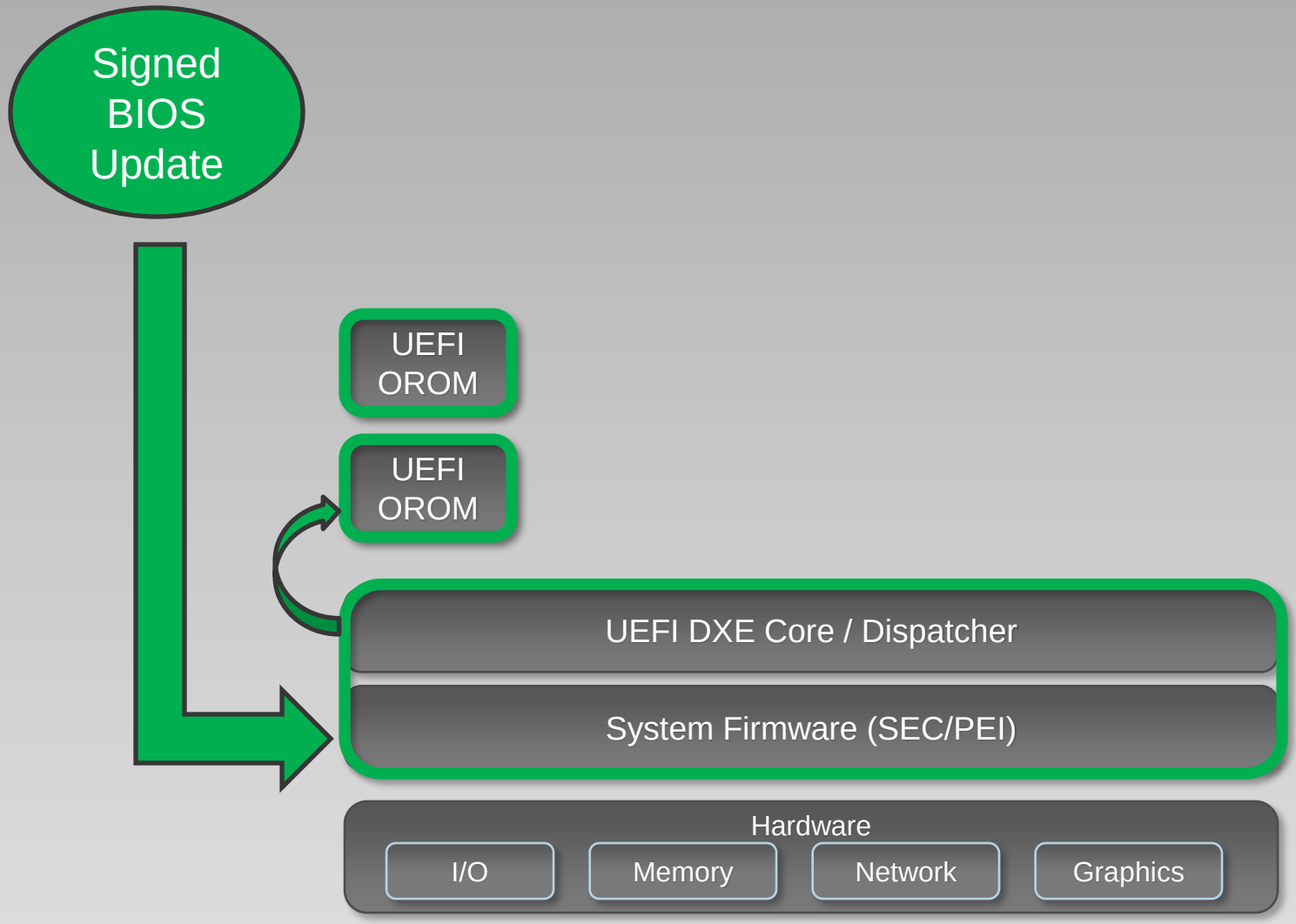
Windows 8 Secure Boot



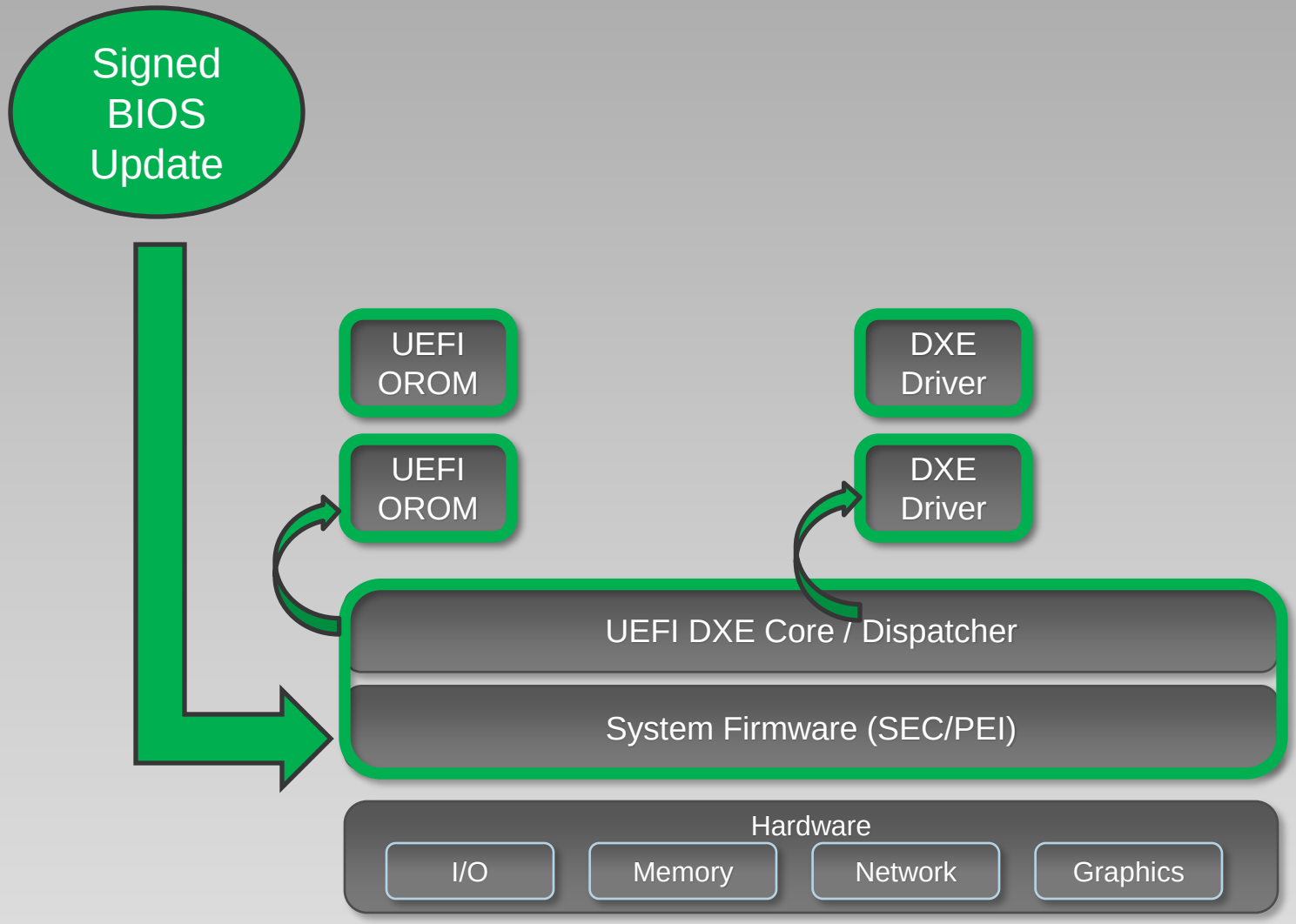
System Firmware and NVRAM Are in ROM



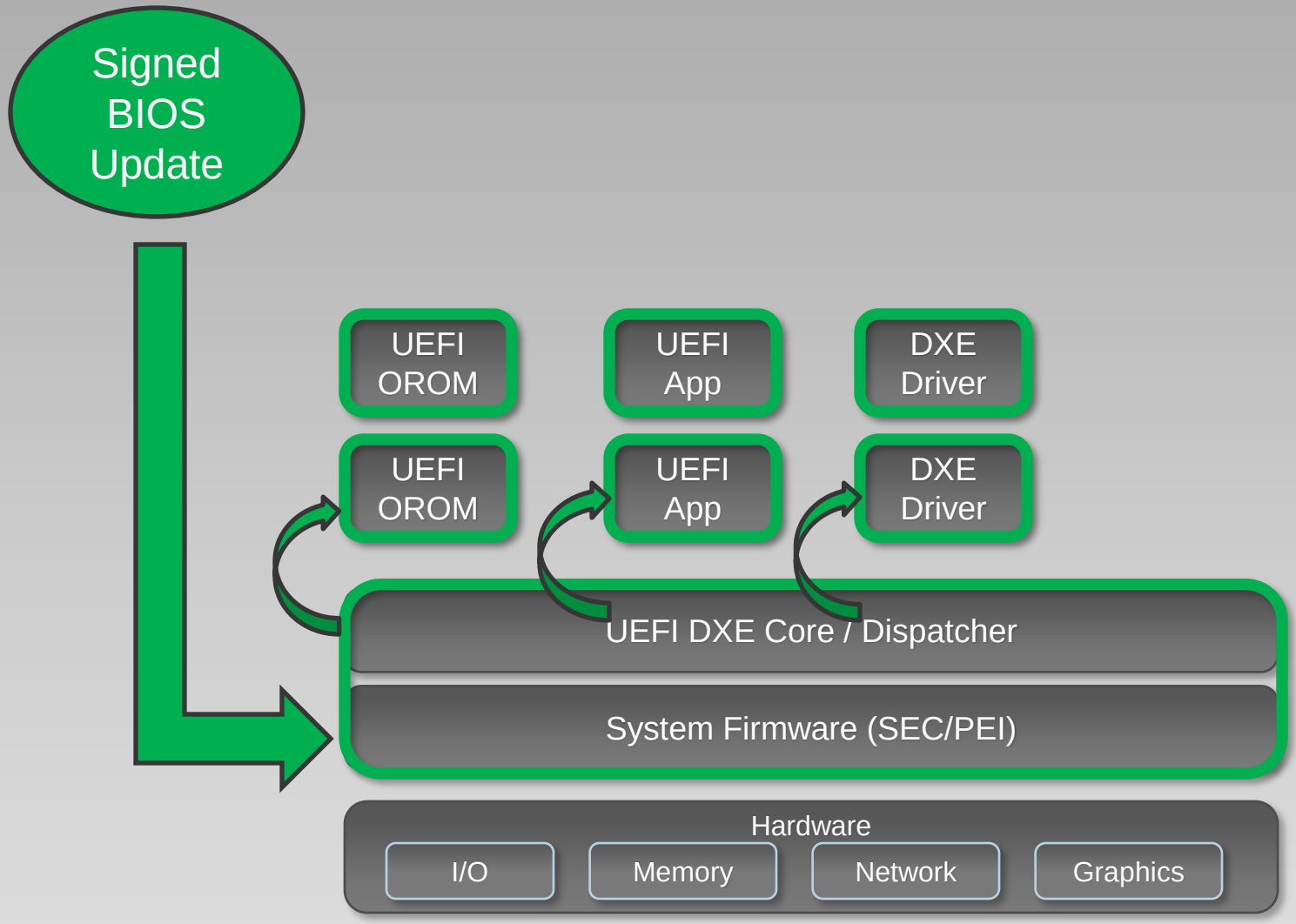
UEFI Firmware Relies on Secure Update



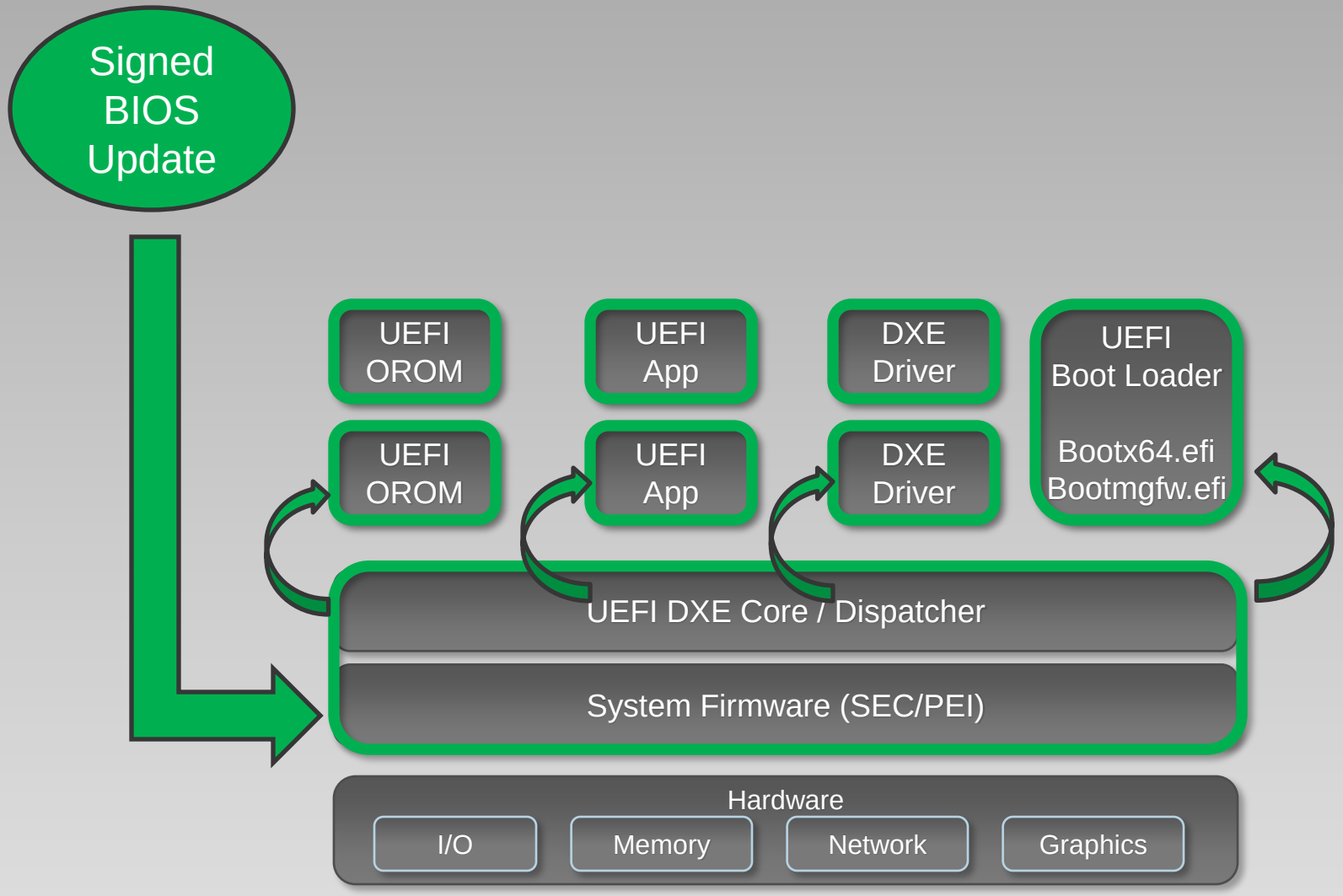
DXE Verifies Non-Embedded UEFI OROMs



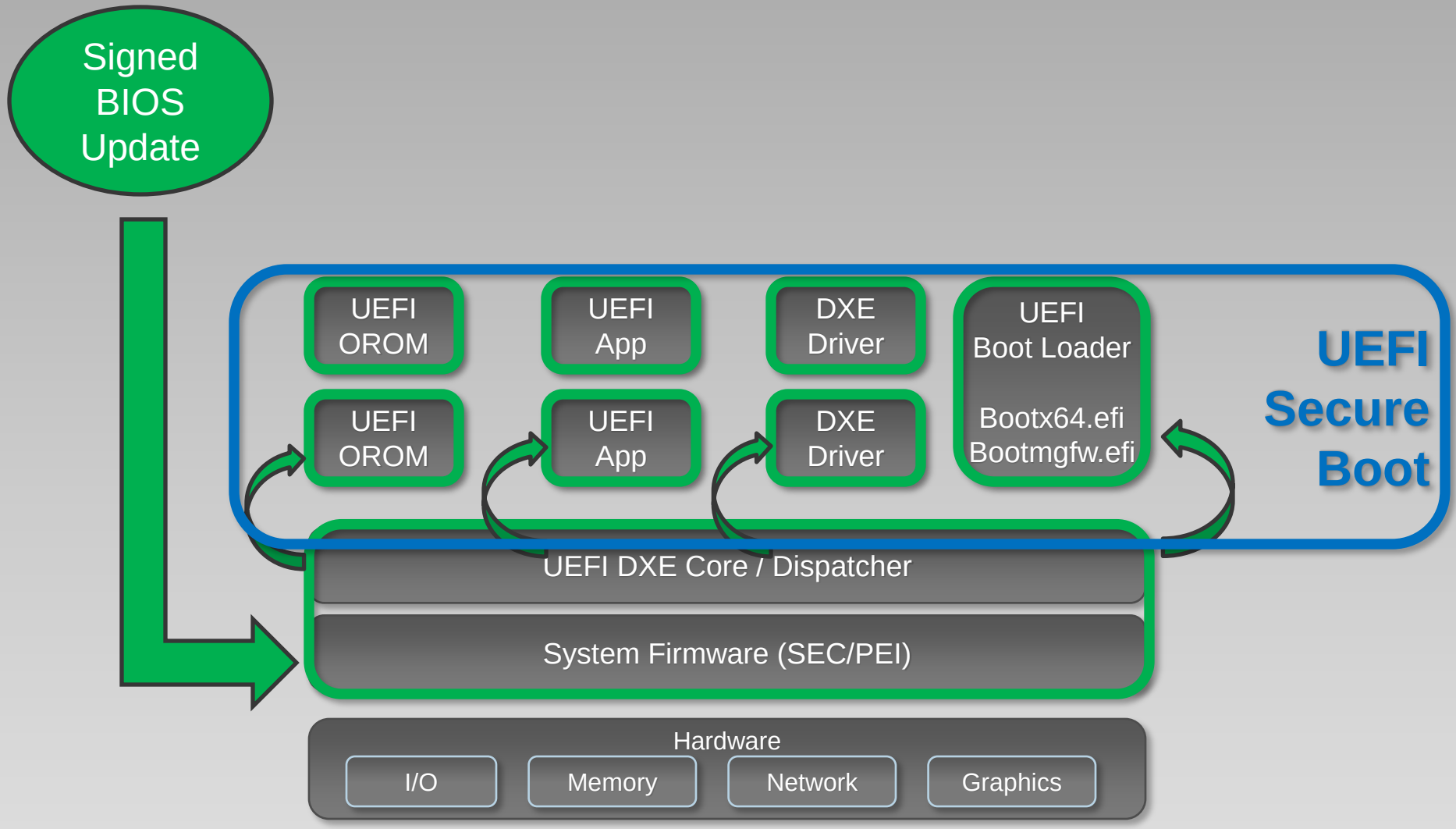
DXE Verifies Non-Embedded DXE Drivers



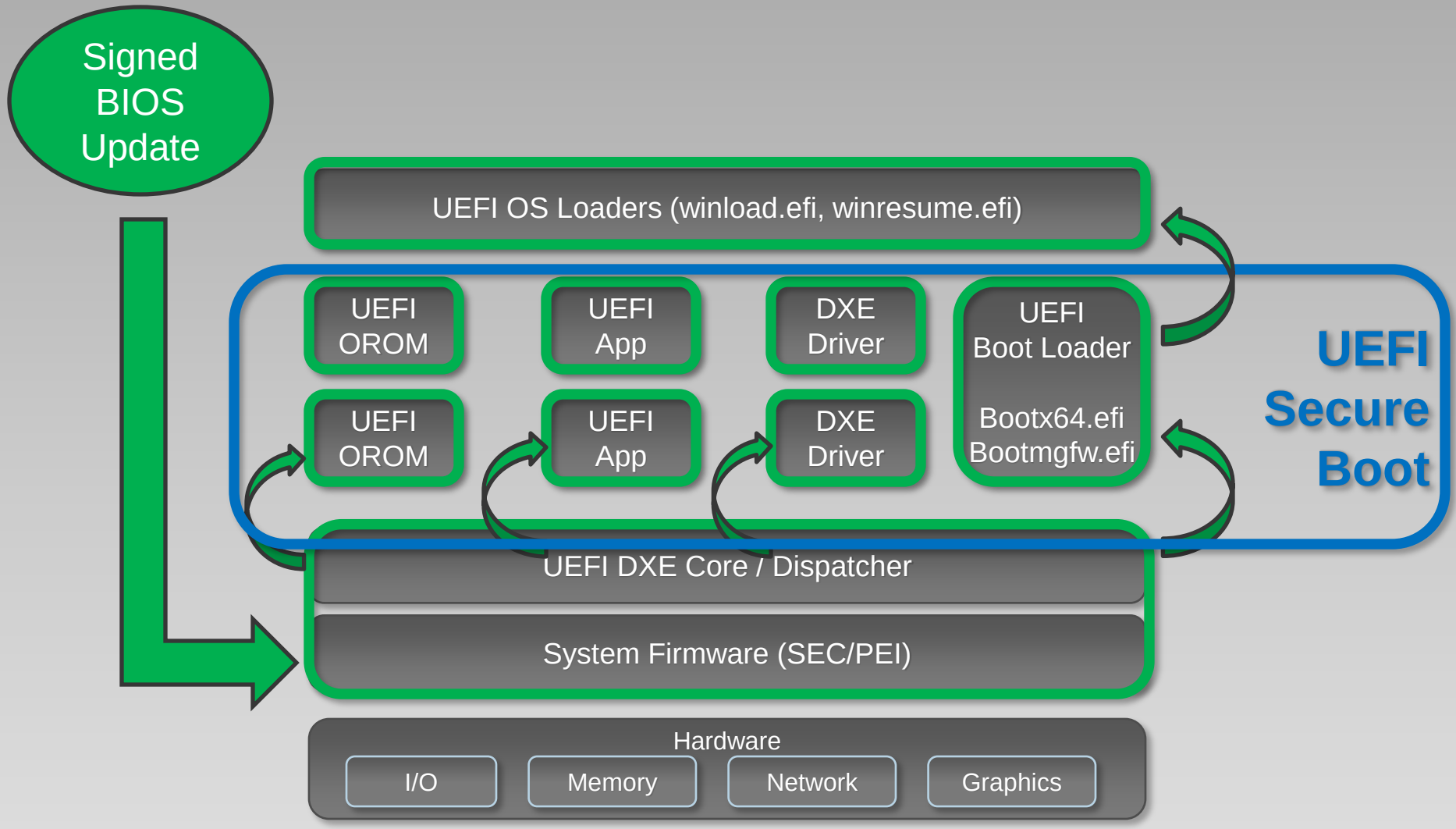
DXE Core Verifies UEFI Applications



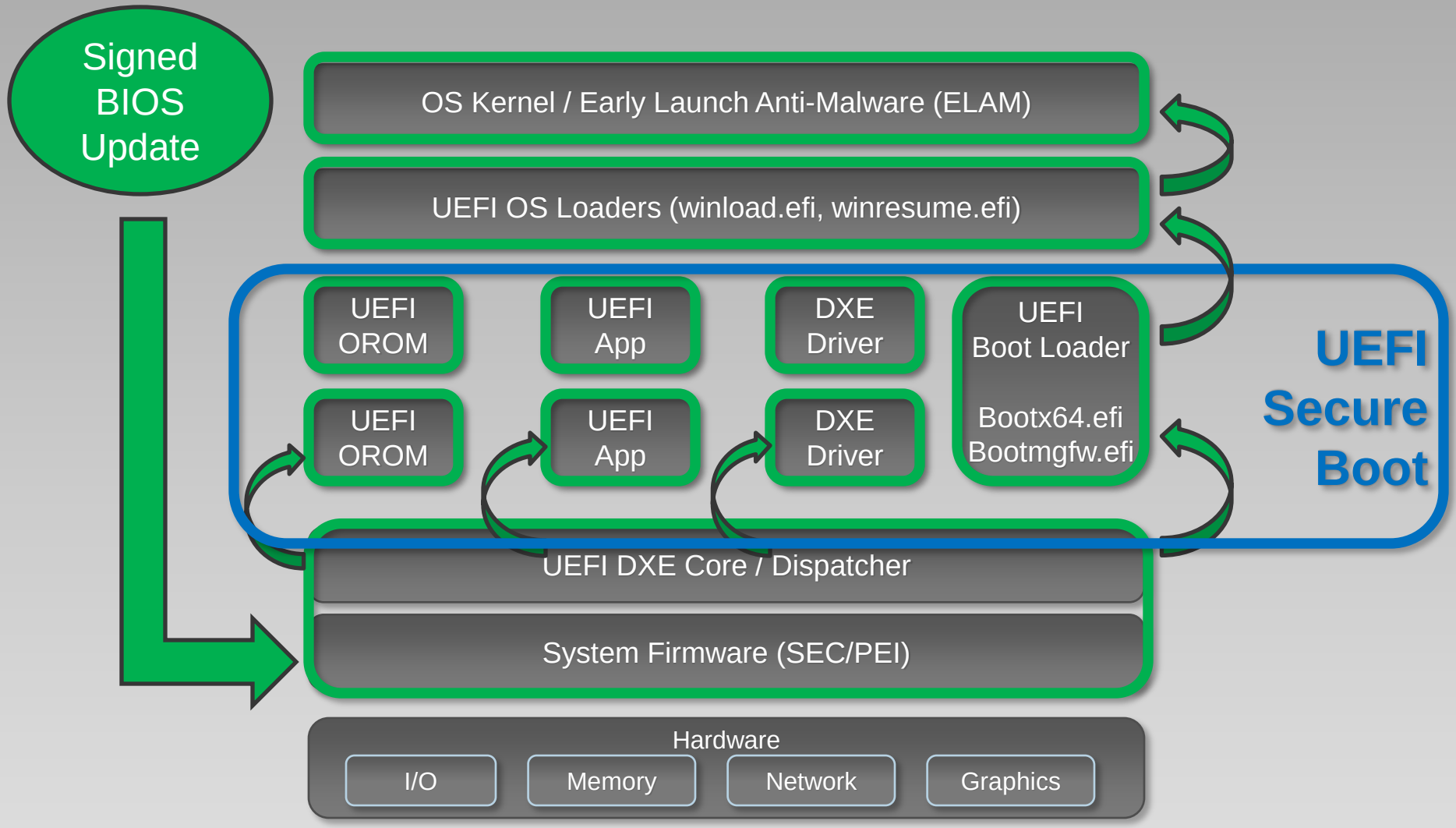
DXE Core Verifies UEFI Boot Loader(s)



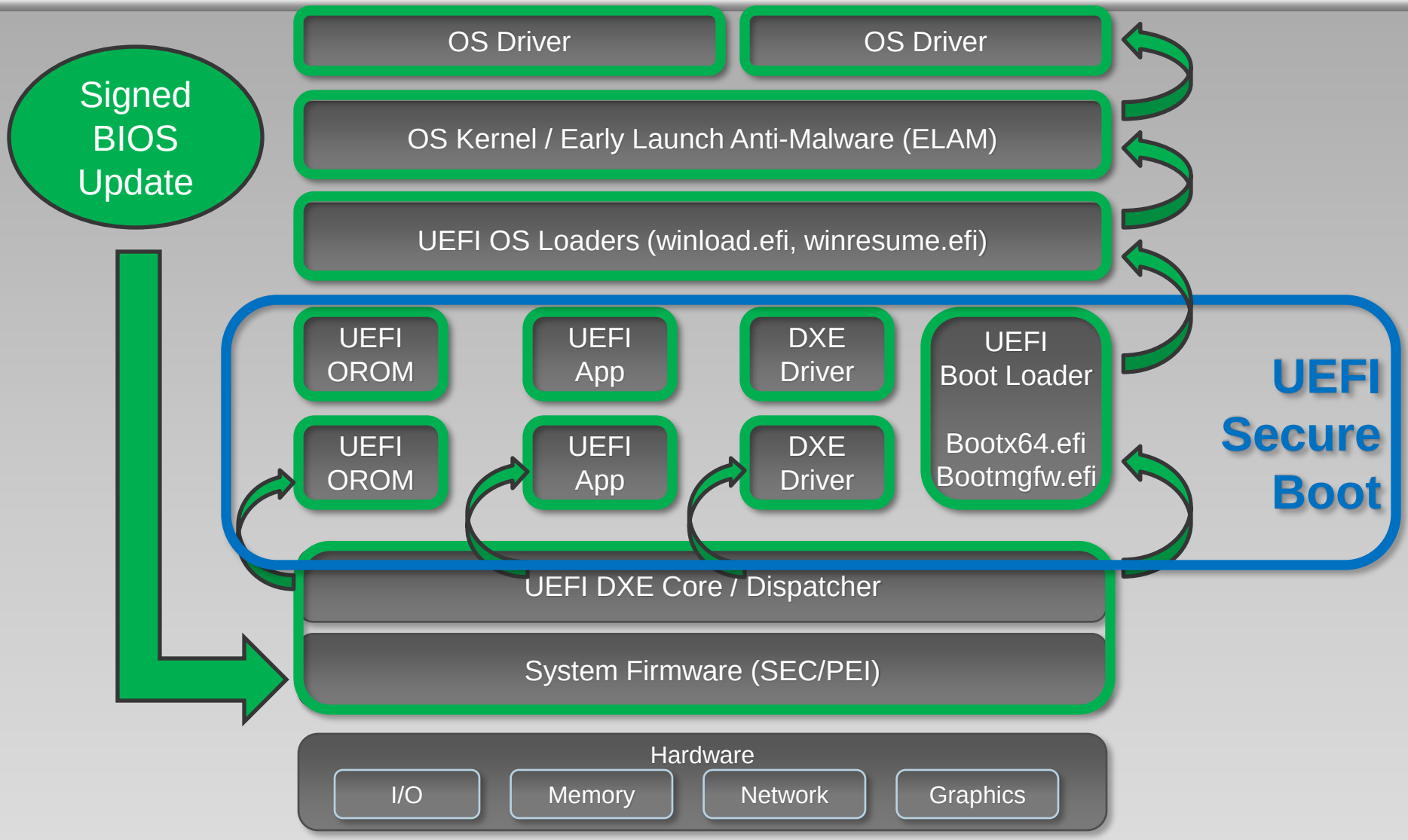
UEFI Secure Boot



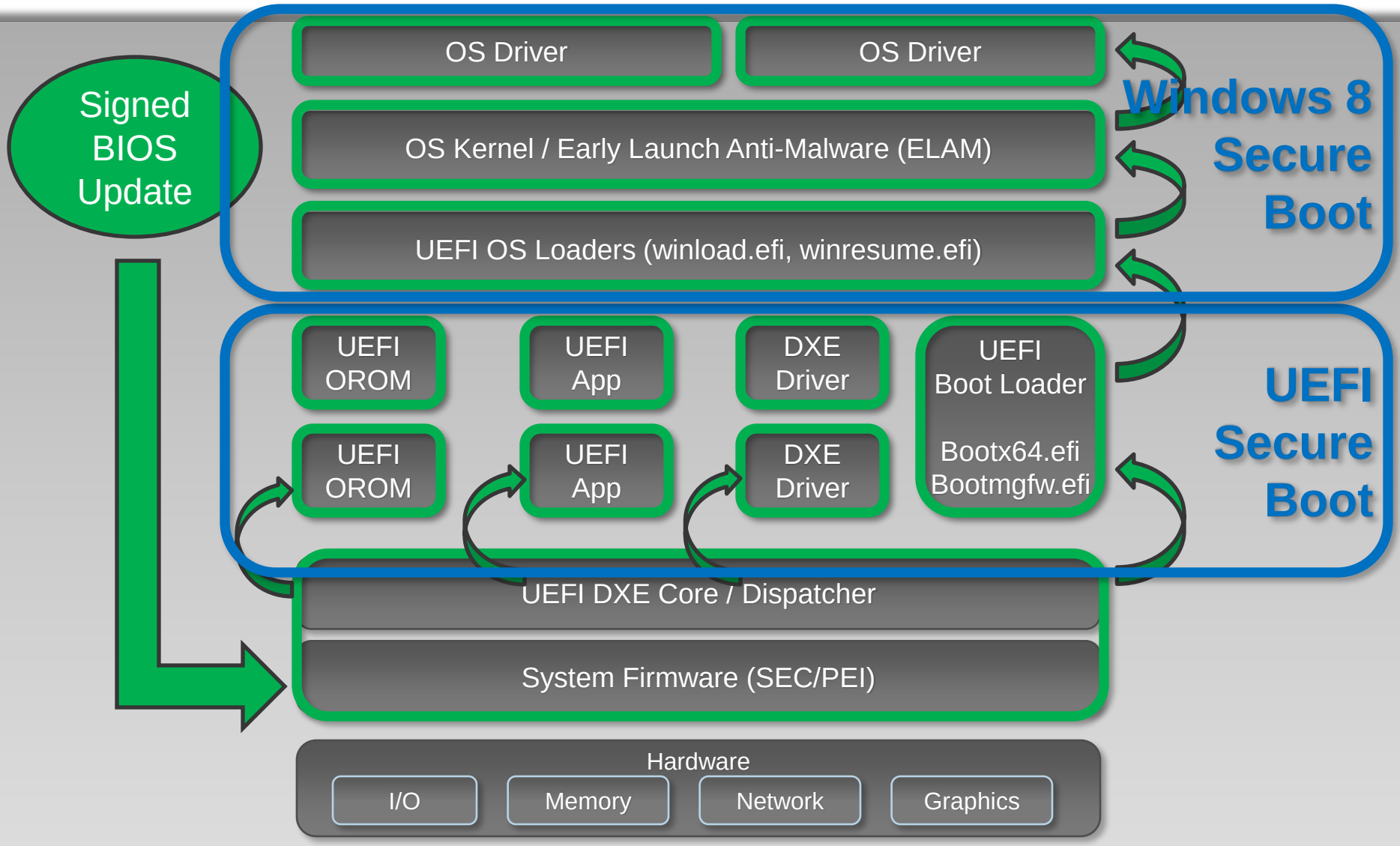
UEFI Boot Loader Verifies OS Loader



OS Loader Verifies OS Kernel



OS Kernel Verifies OS Device Drivers



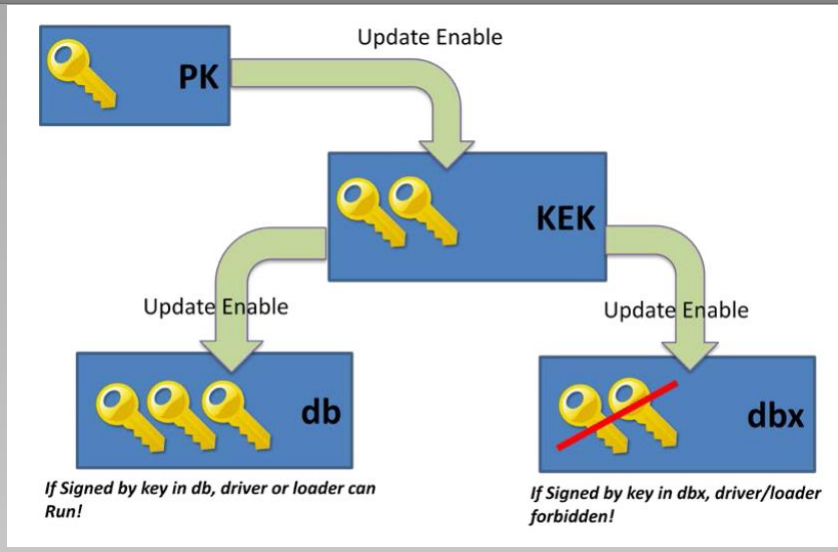
Windows 8 Secure Boot

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, image SHA1/SHA256 hashes of allowed and revoked images
- Earlier rev's: RSA-2048 public keys, PKCS#7 Signatures

Secure Boot Keys

Non-Volatile (NV)

- Stored in SPI Flash based NVRAM

Boot Service (BS)

- Accessible to DXE drivers / Boot Loaders at boot time

Run-Time (RT)

- Accessible to the OS through run-time UEFI SetVariable/GetVariable API

Time-Based Authenticated Write Access

- Signed with time-stamp (anti-replay)
- PK cert verifies PK/KEK update
- KEK verifies db/dbx update
- certdb verifies general authenticated EFI variable updates

Secure Boot Key Protections

```
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-----
```

PK (openssl x509 -in PK.pem -text)

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 Configuration

PK variable exists in NVRAM?

- **Yes. Set SetupMode to USER_MODE**
- **No. Set SetupMode to SETUP_MODE**

SecureBootEnable variable exists in NVRAM?

- **Yes**
  - **SecureBootEnable is SECURE_BOOT_ENABLE and SetupMode is USER_MODE? Set SecureBoot to ENABLE**
  - **Else? Set SecureBoot to DISABLE**
- **No**
  - **SetupMode is USER_MODE? Set SecureBoot to ENABLE**
  - **SetupMode is SETUP_MODE? Set SecureBoot to DISABLE**

Dependencies (AutenticatedVariableService)

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

Image Verification Policies

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

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;
}
```

- **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**

Image Verification (Configuration)

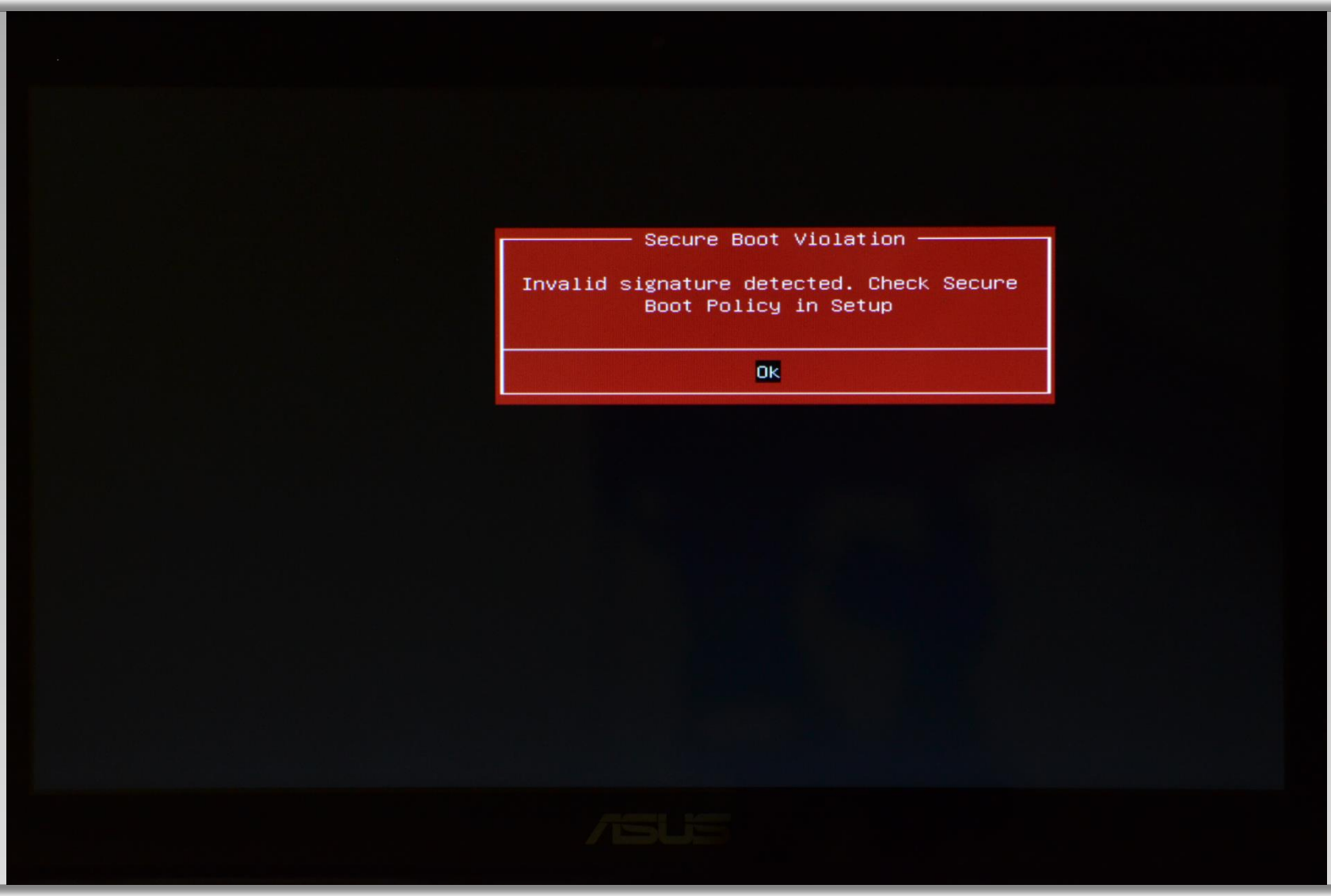
Image signed?

- No
  - Image SHA256 hash in dbx? EFI_ACCESS_DENIED
  - Image SHA256 hash in db? EFI_SUCCESS
- Yes

  For each signature in PE file:
  - Signature verified by root/intermediate cert in dbx? EFI_ACCESS_DENIED
  - Image hash in dbx? EFI_ACCESS_DENIED

  For each signature in PE file:
  - Signature verified by root/intermediate cert in db? EFI_SUCCESS
  - Image hash in db? EFI_SUCCESS
- EFI_ACCESS_DENIED

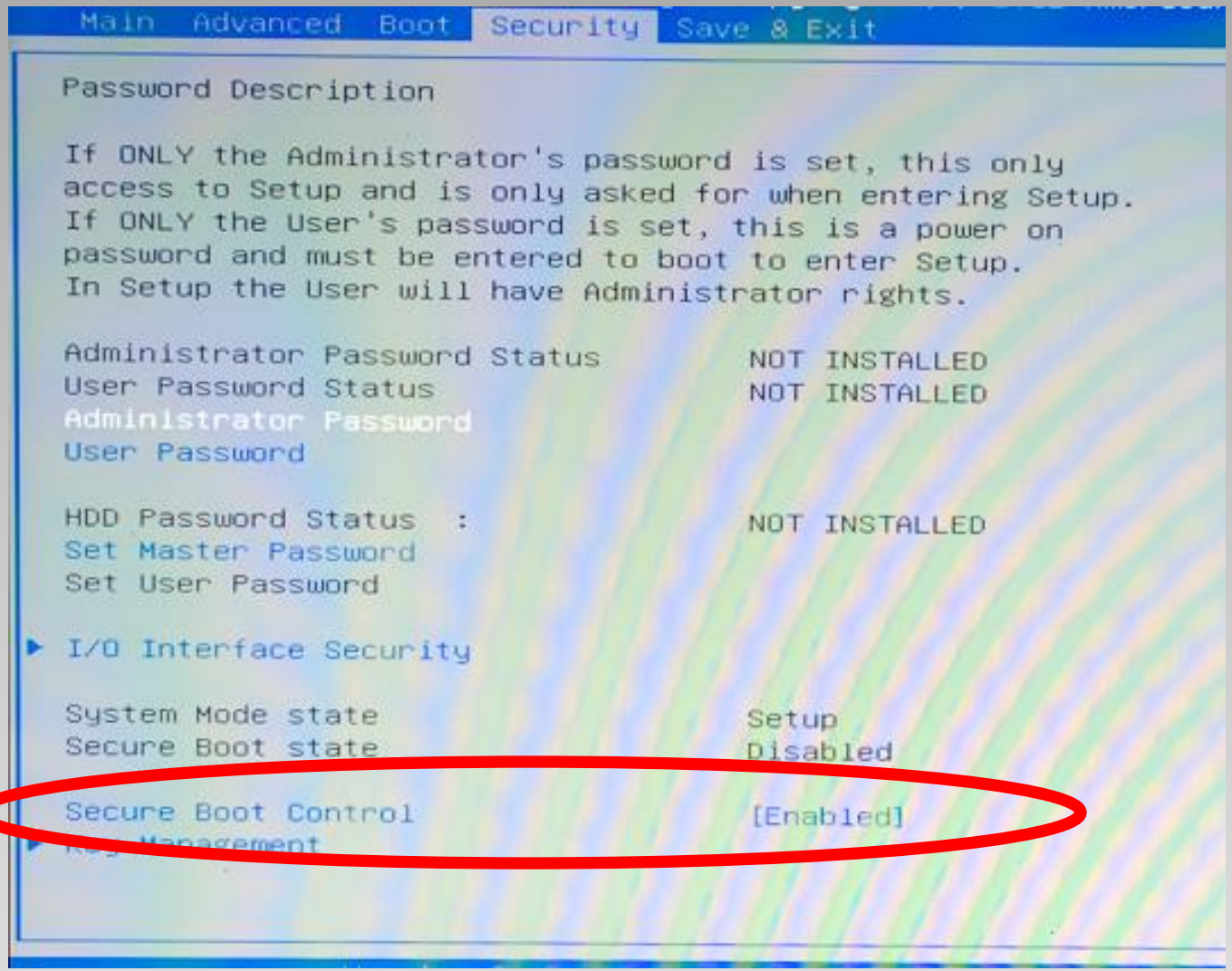
Image Verification (Crypto)



Secure Boot in Action

Windows 8/UEFI Secure Boot is pretty important protection from boot malware!

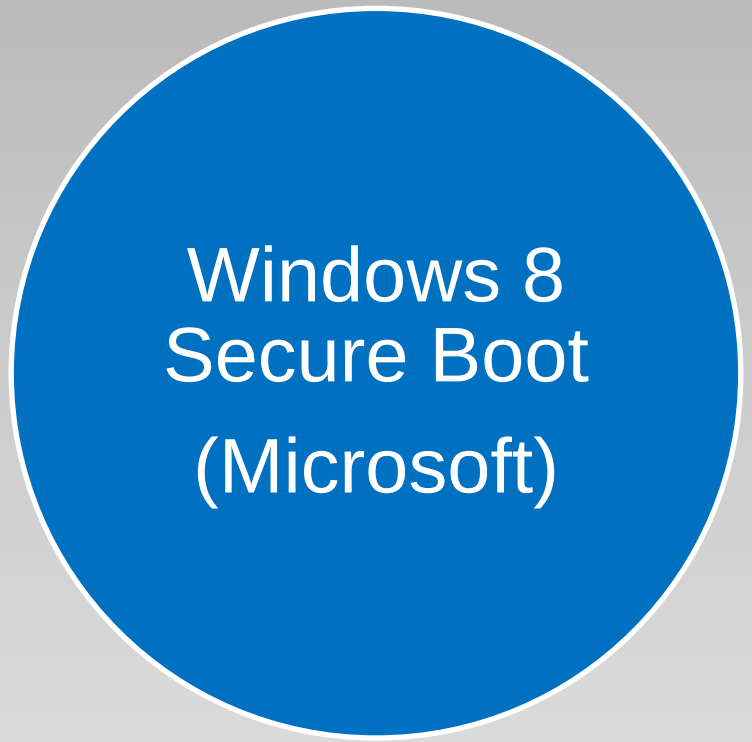
Attacking Windows 8 Secure Boot



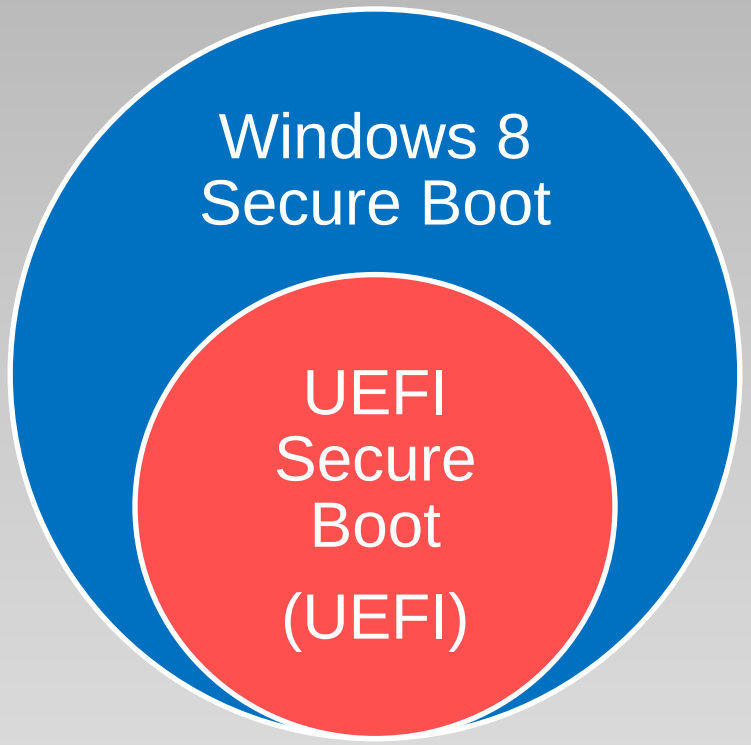
Just turn it off in the BIOS setup screen ;)

Just to be clear, the issues are in the implementation of Secure Boot and required UEFI firmware protections on certain platforms

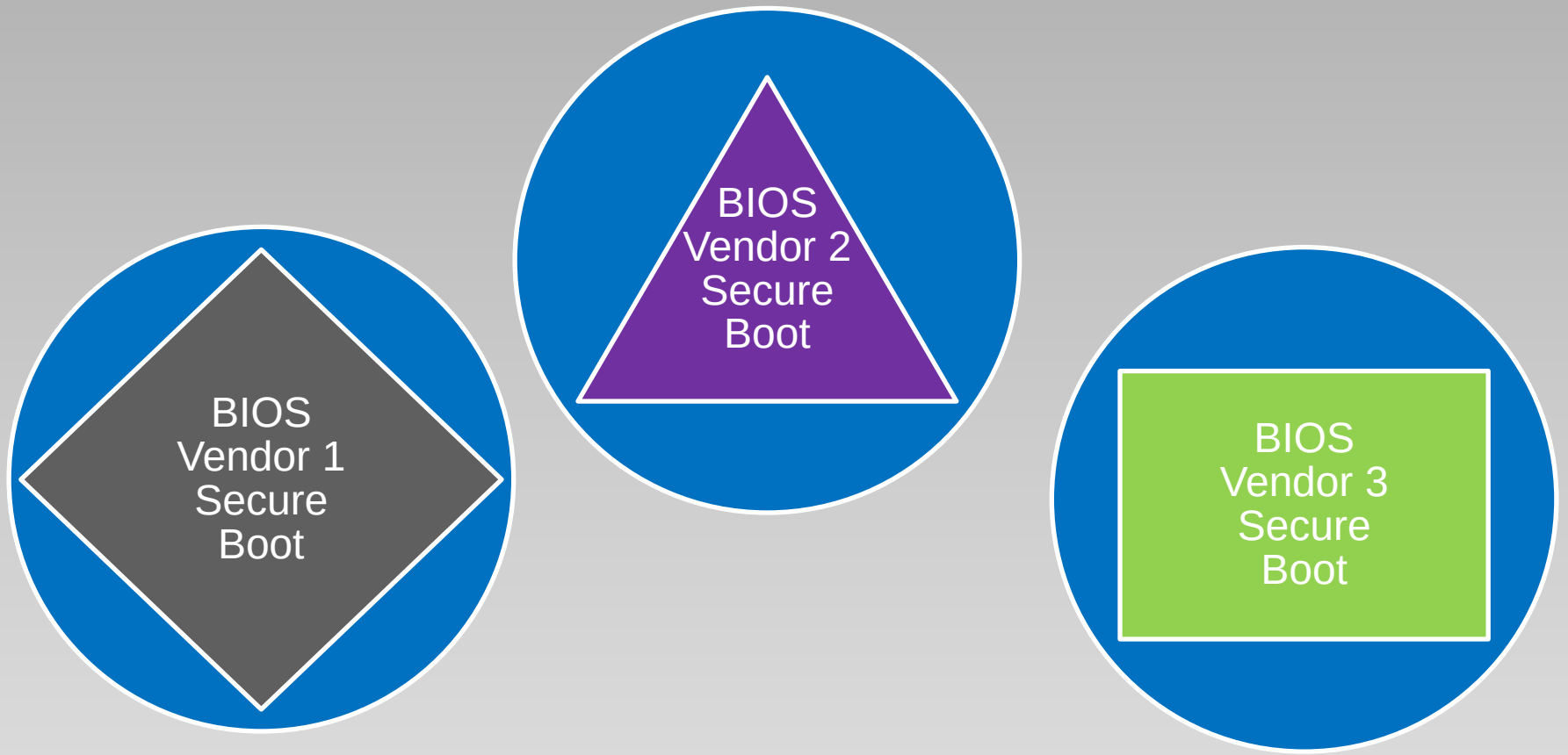
We think Windows 8 Secure Boot looks like this

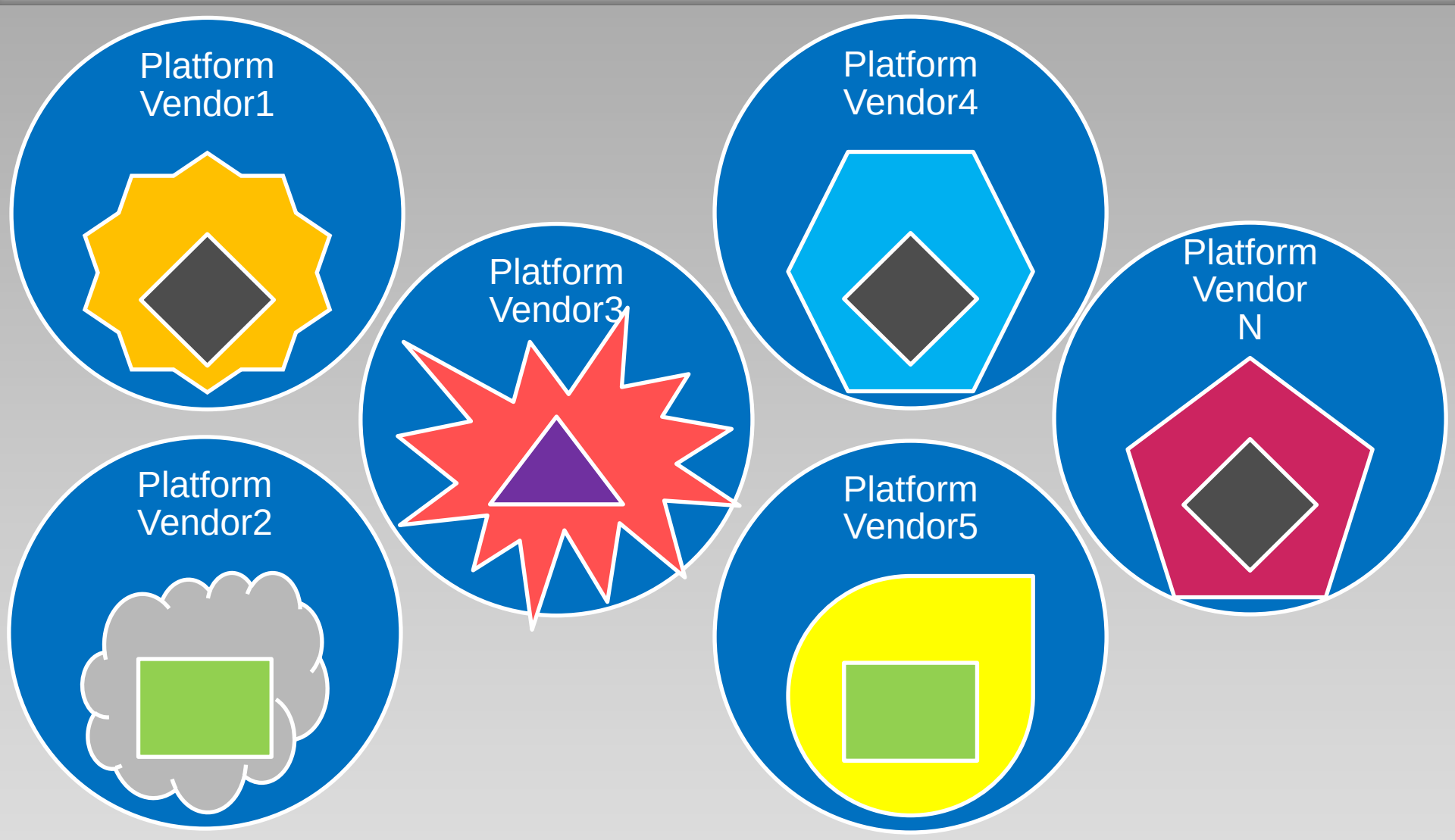


Or more like this



How exciting! … But still not close

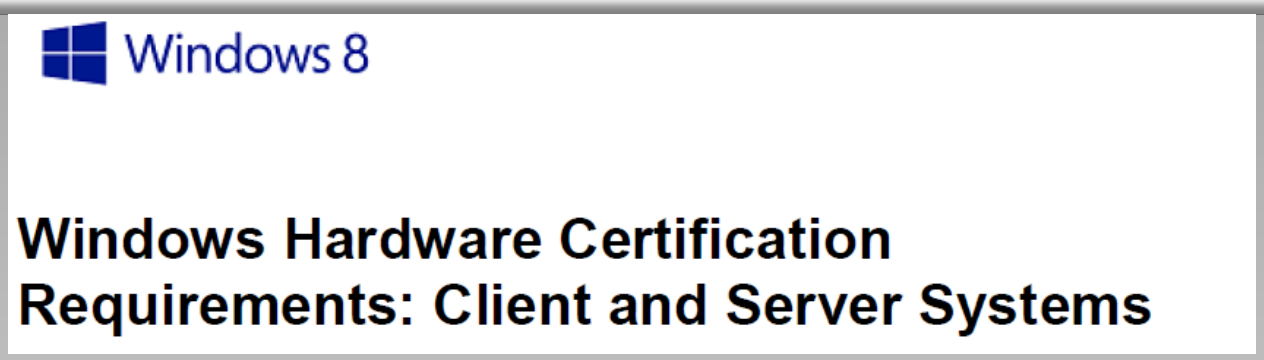




The Reality Is Much More Exciting

Windows 8 Secure Boot is only secure when ALL platform/BIOS vendors do a couple of things correctly

- Allow signed UEFI firmware updates only
- Protect UEFI firmware in SPI flash from direct modification
- Protect firmware update components (inside SMM or DXE on reboot)
- Program SPI controller and flash descriptor securely
- Protect SecureBootEnable/CustomMode/PK/KEK/db(x) in NVRAM
- Implement VariableAuthenticated in SMM and physical presence checks
- Protect SetVariable runtime API
- Securely disable Compatibility Support Module (CSM), unsigned legacy Option ROMs and MBR boot loaders
- Configure secure image verification policies (no ALLOW_EXECUTE)
- Build platform firmware using latest UEFI/EDK sources
- Correctly implement signature verification and crypto functionality
- **And don’t introduce a single bug in all of this…**



System.Fundamentals.Firmware.UEFISecureBoot

3 When Secure Boot is Enabled, CSM must NOT be loaded

7 Secure Boot must be rooted in a protected or ROM-based Public Key

8 Secure firmware update process

9 Signed Firmware Code Integrity Check

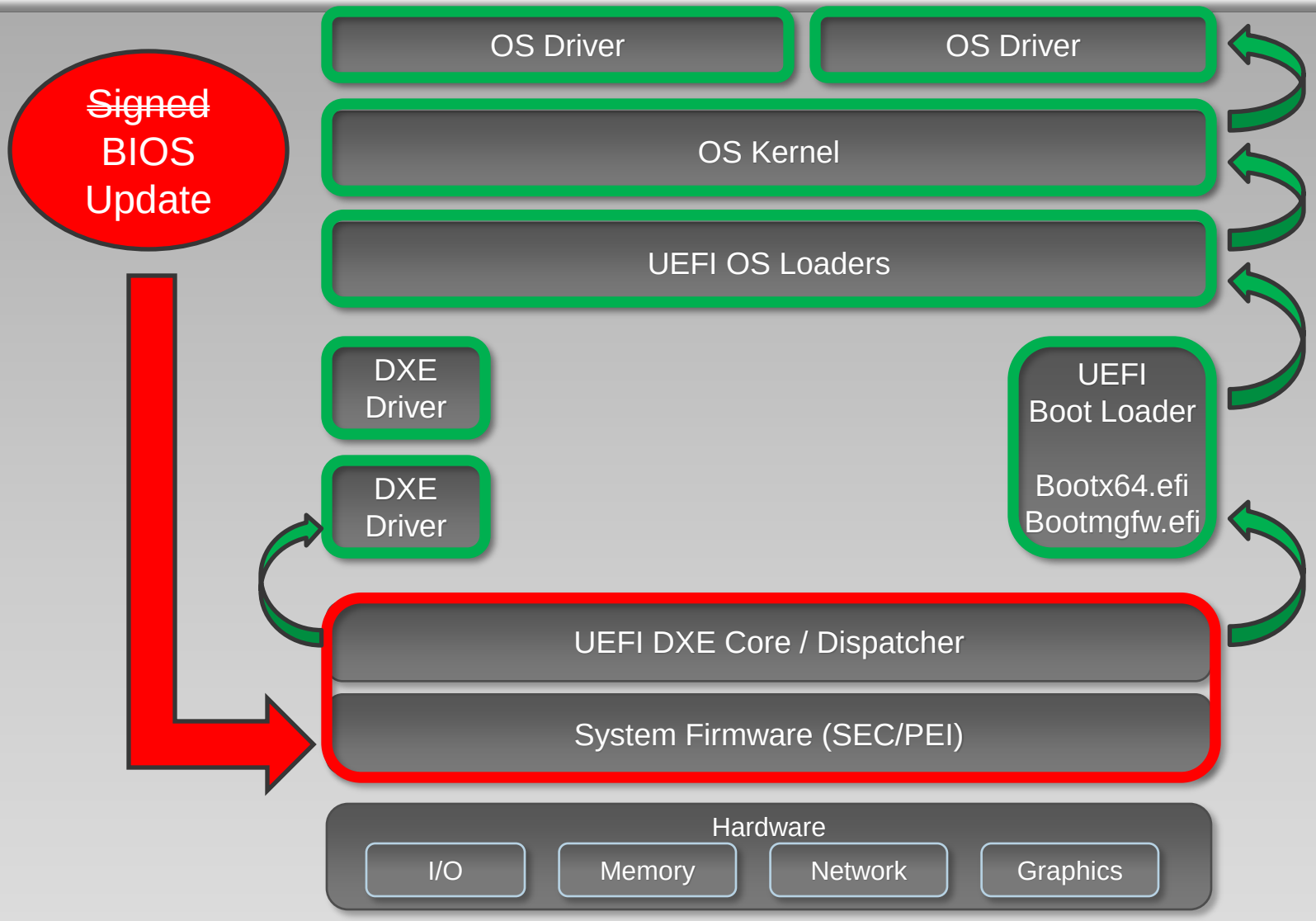
14 No in-line mechanism is provided whereby a user can bypass Secure Boot failures and boot anyway

…

Windows 8 Secure Boot Requirements


Master

What If UEFI BIOS Updates Are Not Signed?



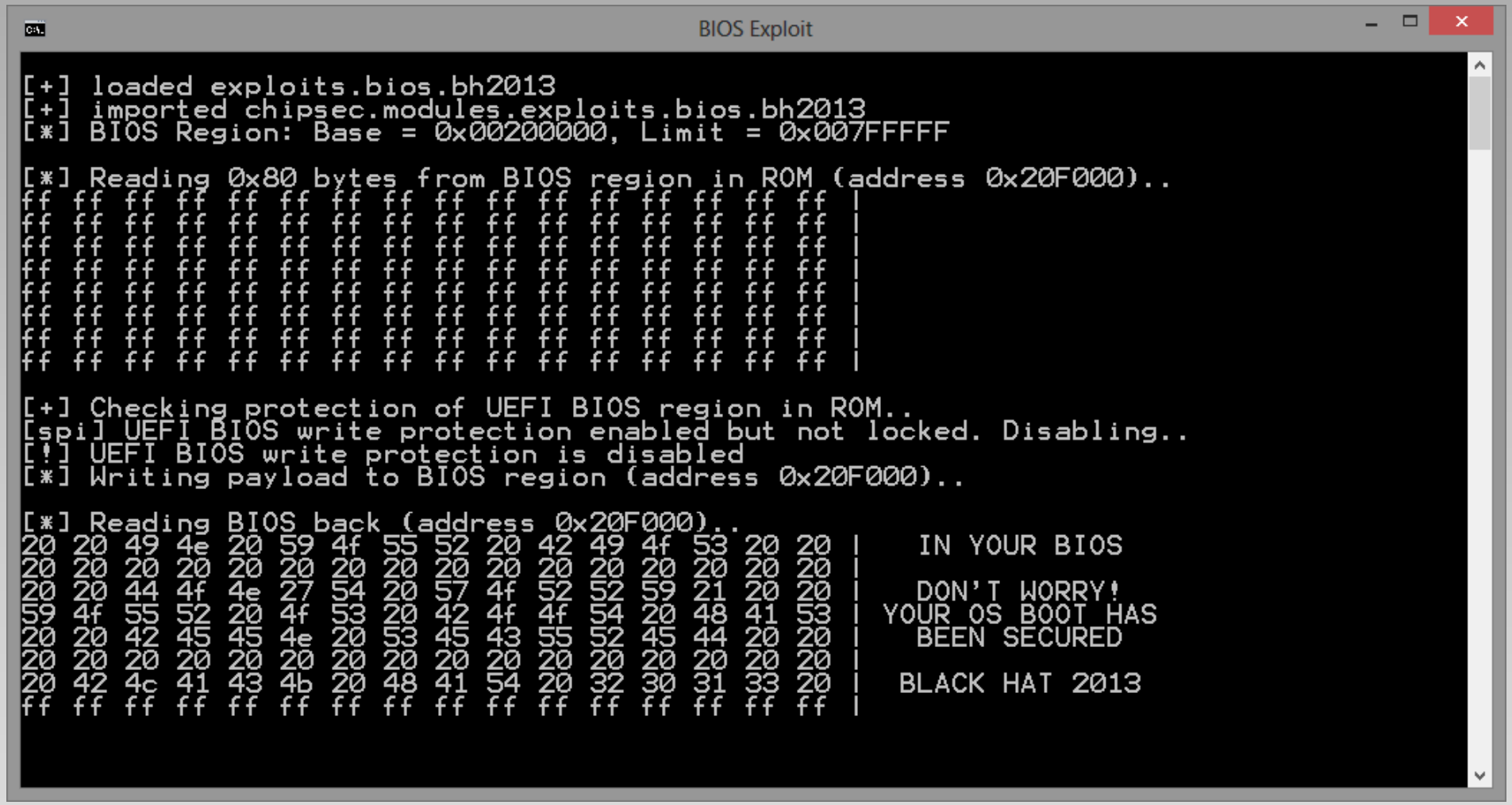
When UEFI Firmware Updates Are Not Signed

No luck 

UEFI firmware update capsules are signed

RSA-PSS 2048 / SHA-256 / $e=F_4$

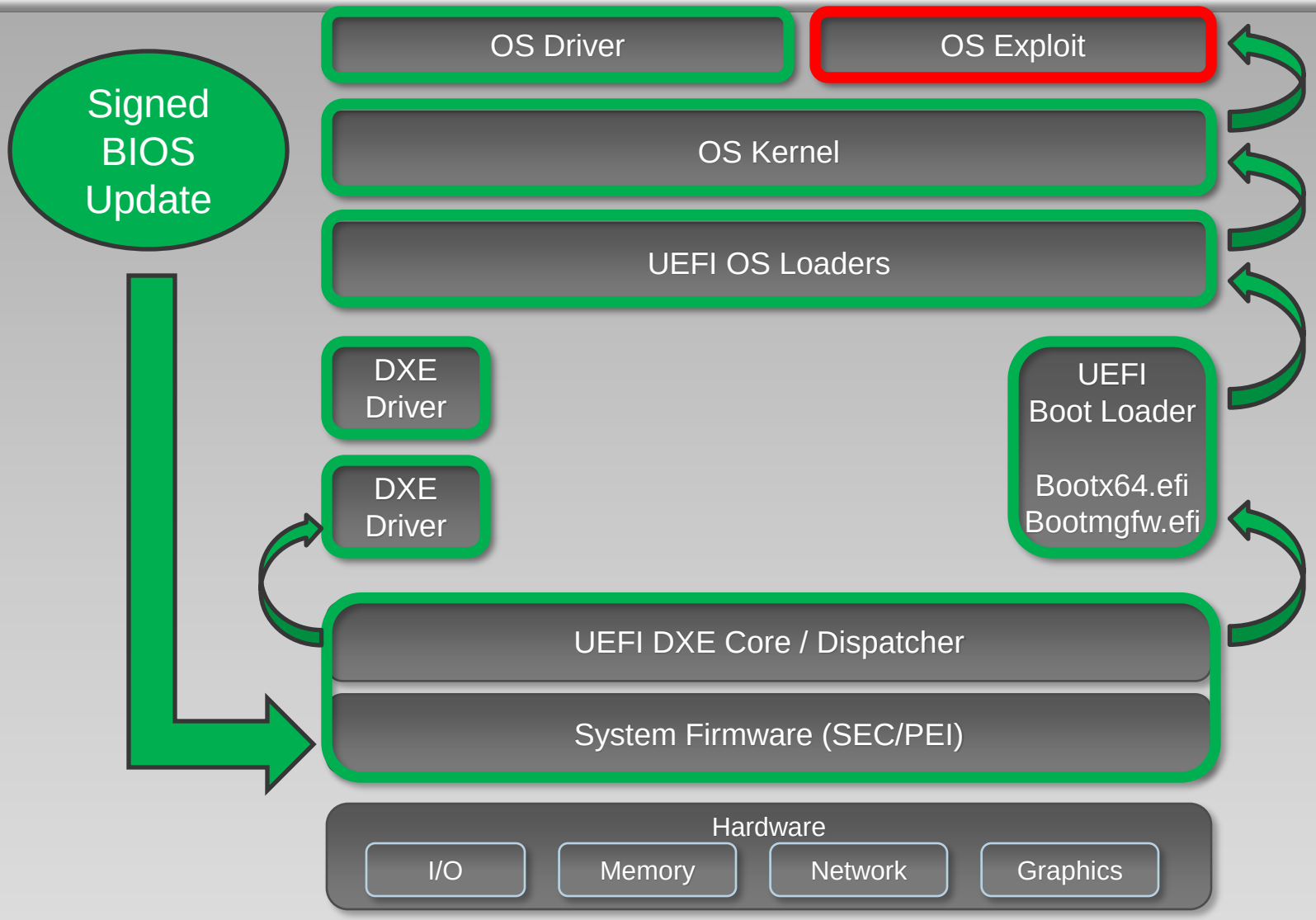
Wait, let's check one little thing...



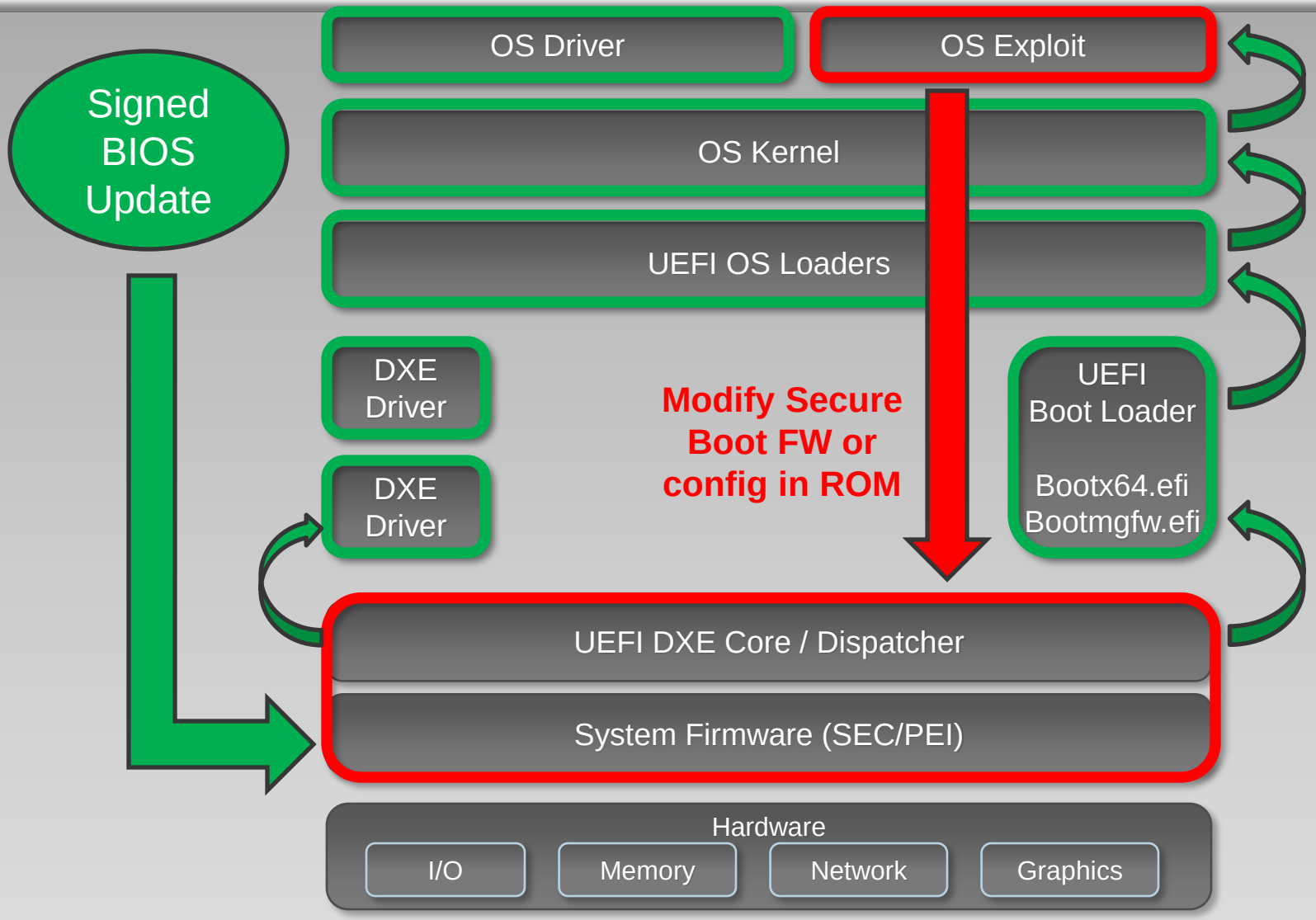
Can We Write to UEFI Firmware in ROM?

So UEFI firmware updates are signed but firmware is directly writeable in SPI flash? So is NVRAM with EFI variables. Hmm… What could go wrong?

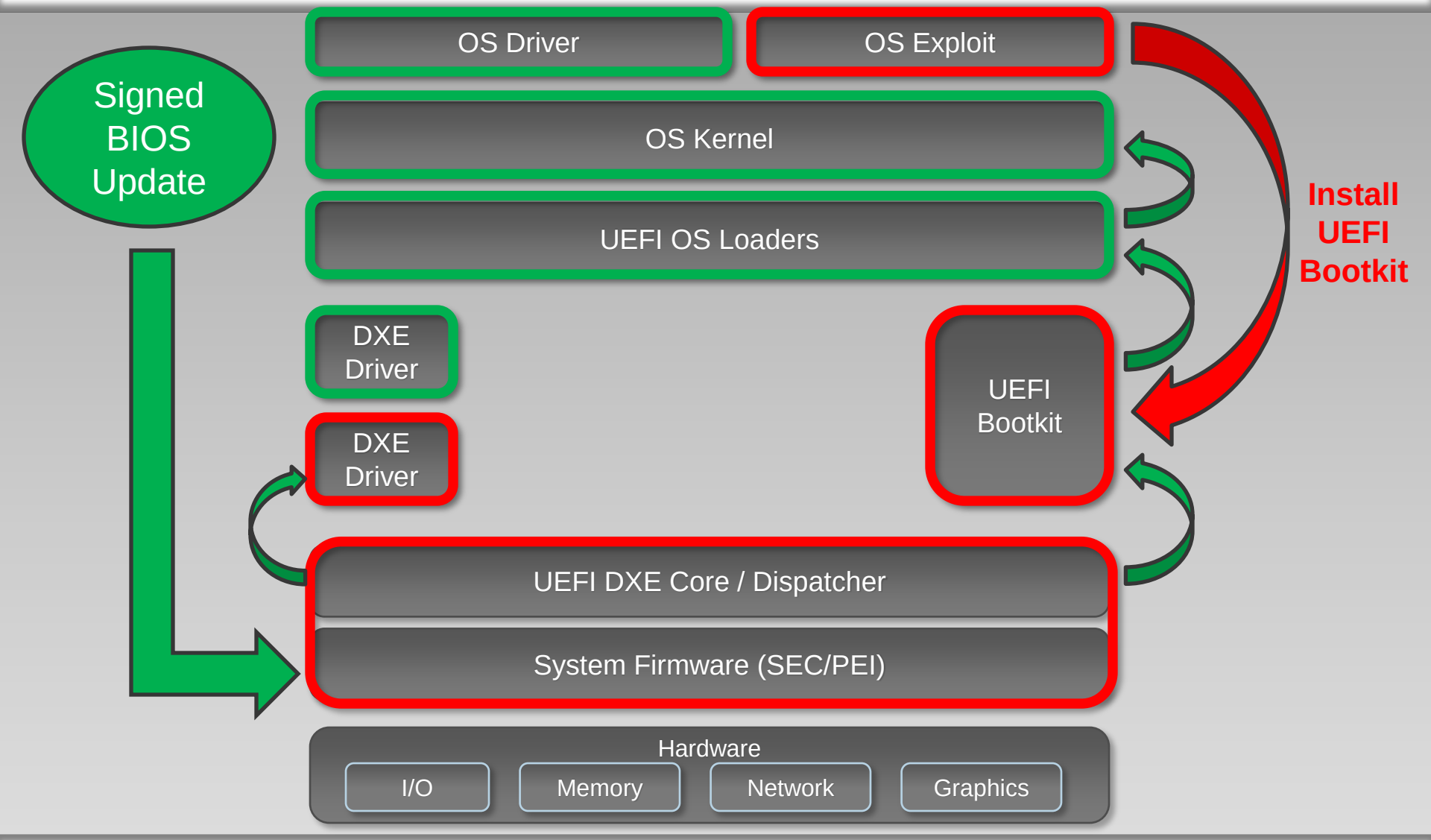
Hint: Malware could patch DXE Image Verification driver in ROM or it could change persistent Secure Boot keys/configuration in NVRAM



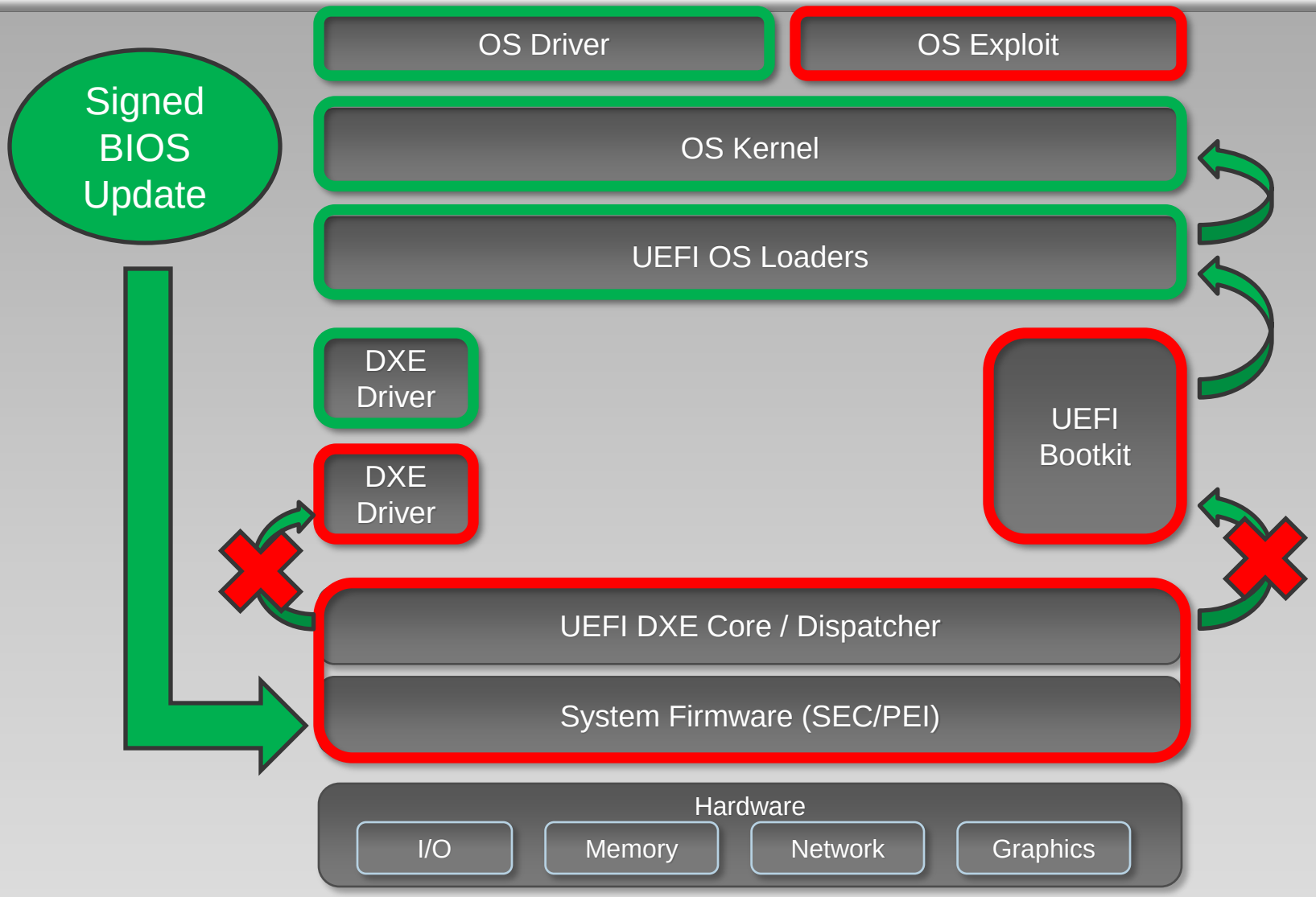
When Firmware Is Not Protected in ROM



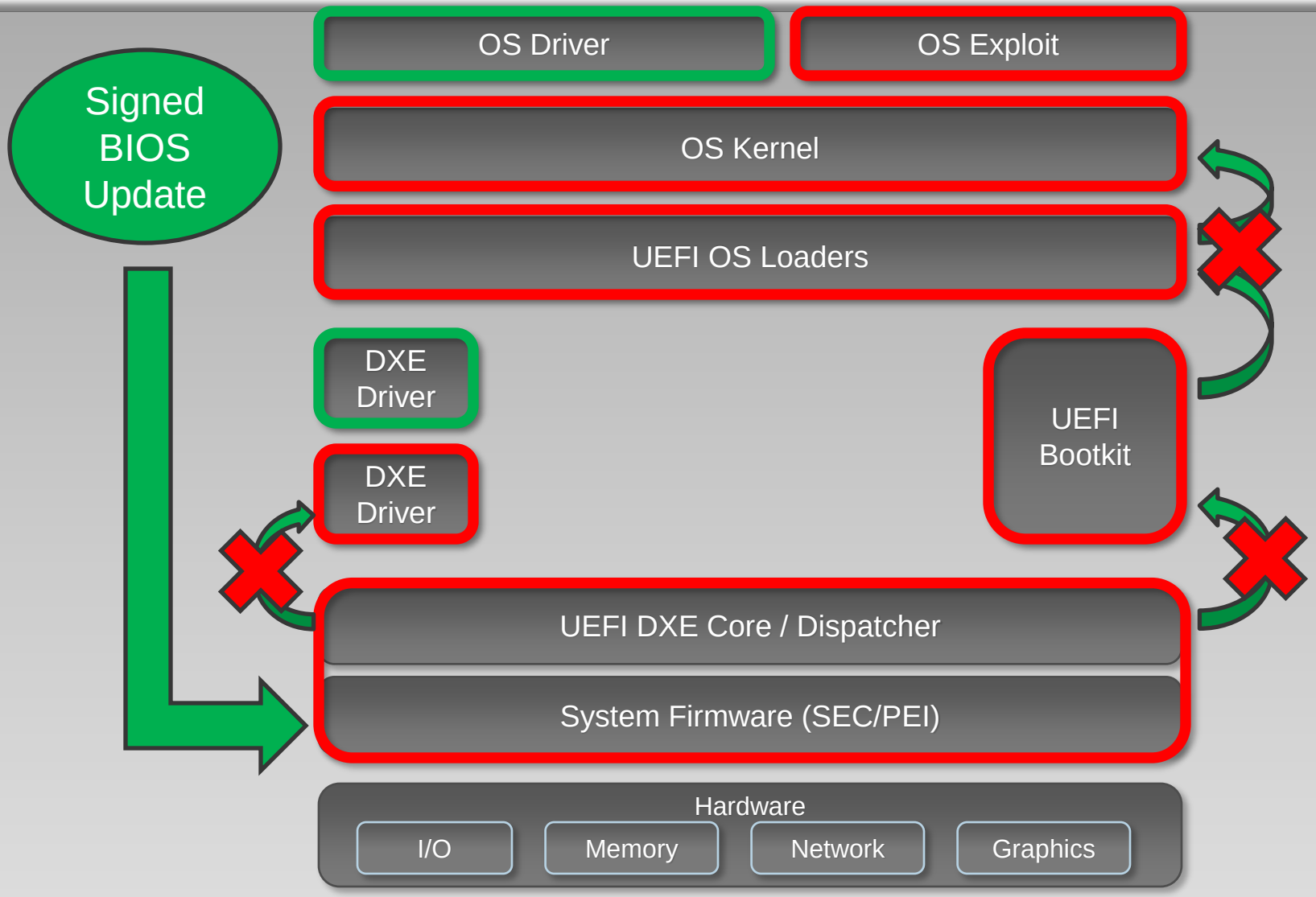
Malware Modifies UEFI Firmware in ROM (directly programming SPI controller)



Then Installs UEFI Bootkit



Firmware Doesn't Enforce Secure Boot



UEFI Bootkit Now Patches OS Loaders/Kernel

Patch DXE ImageVerificationLib driver code

- Differ from one platform/vendor to another
- Different versions of EDK and BIOS Cores

Replace/add hash or Cert in db

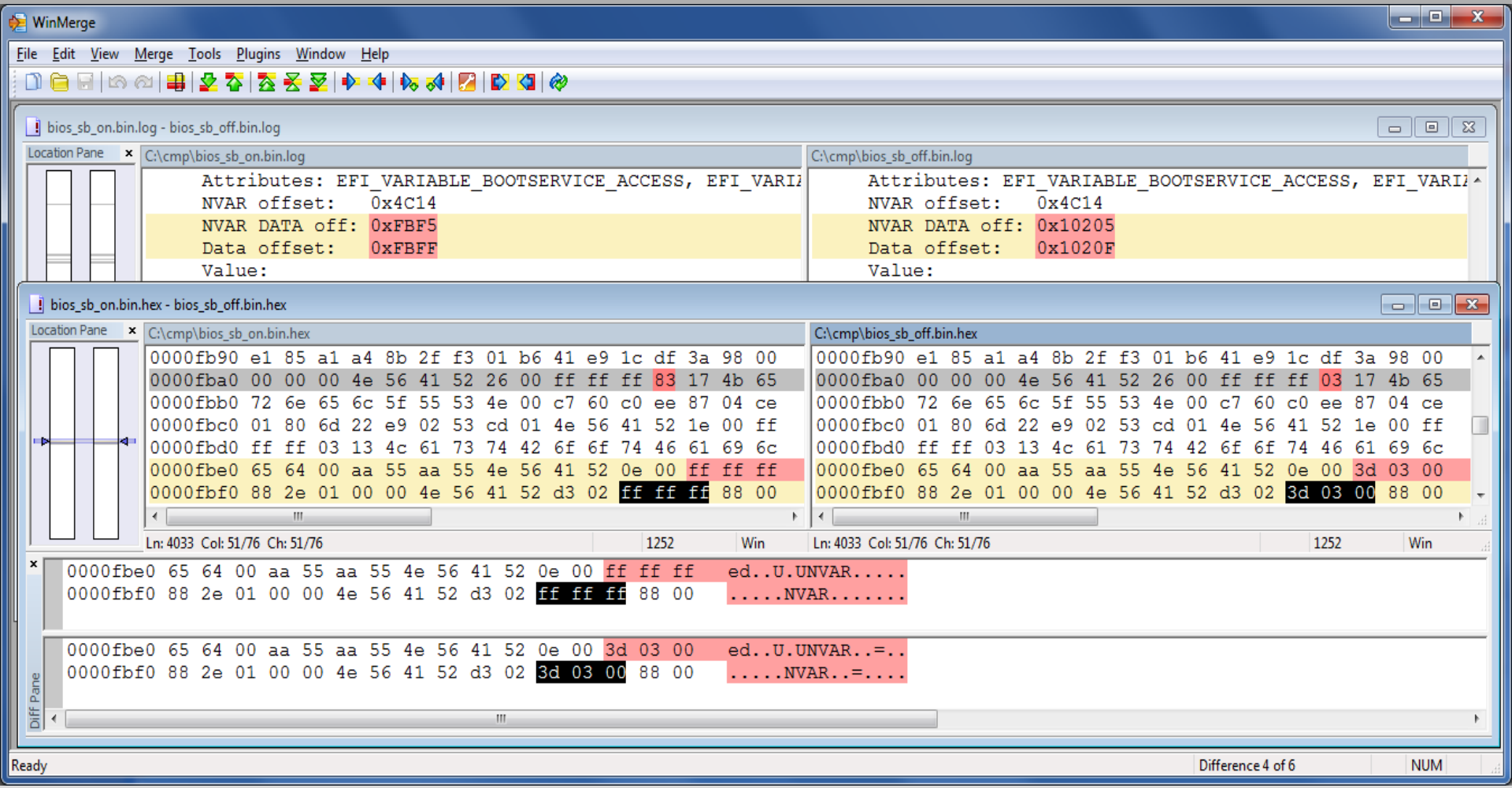
- Bootkit hash is now allowed
- Generic exploit, independent of the platform/vendor
- Can be found by inspecting “db” in ROM

Replace/add RootCert in KEK or PK with your own

- Bootkit signature is now valid

Clear SecureBootEnable variable

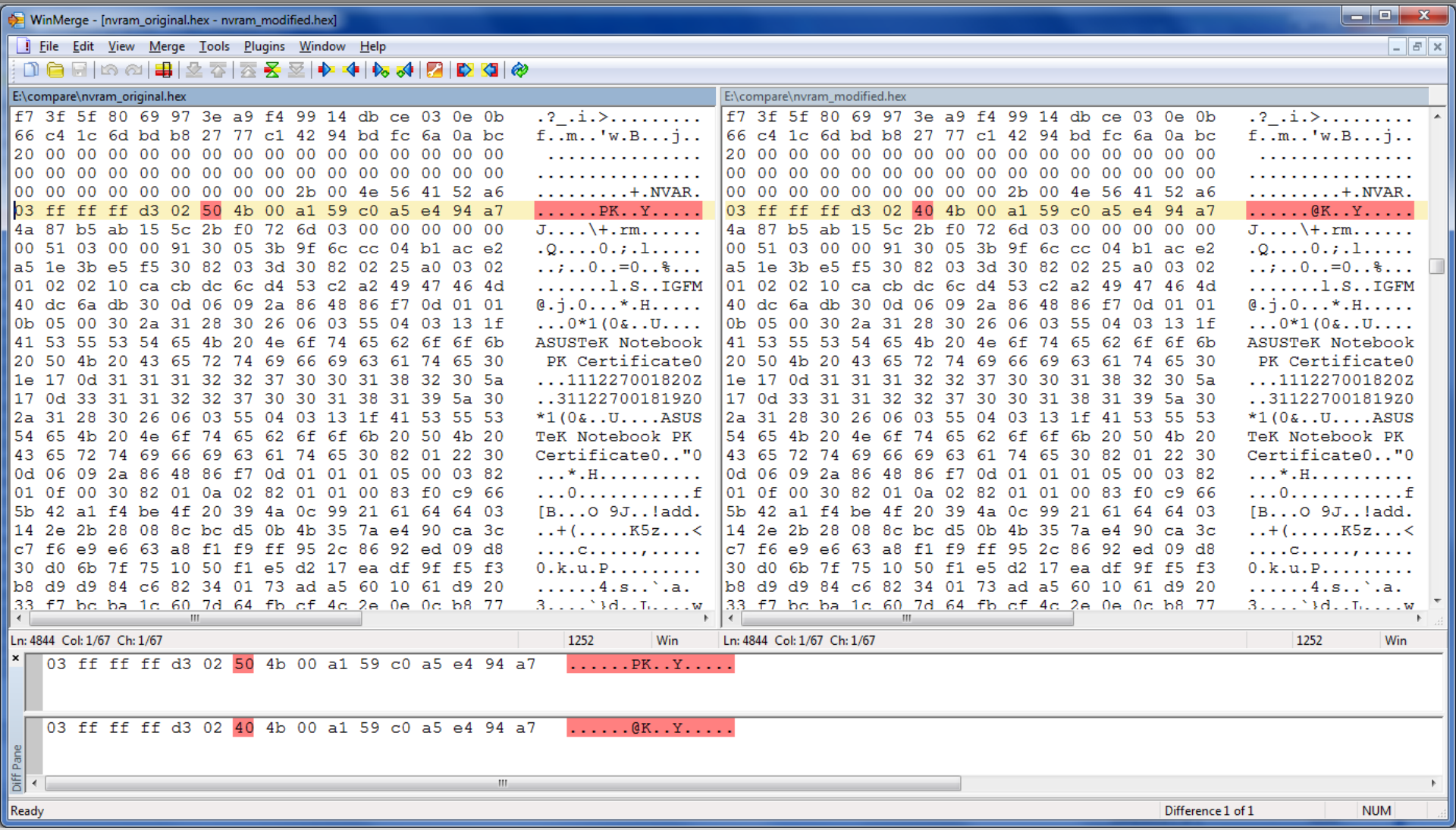
- Despite UEFI defines “SecureBootEnable” EFI variable platform vendors store Secure Boot Enable in platform specific places
- Format of EFI NVRAM and EFI variable in ROM is platform/vendor specific
- May require modification in multiple places in NVRAM ➔ parsing of platform specific NVRAM format
- Replacing entire NVRAM or even entire BIOS region to SB=off state is simpler but takes a while



Parsing Proprietary EFI NVRAM

Corrupt Platform Key EFI variable in NVRAM

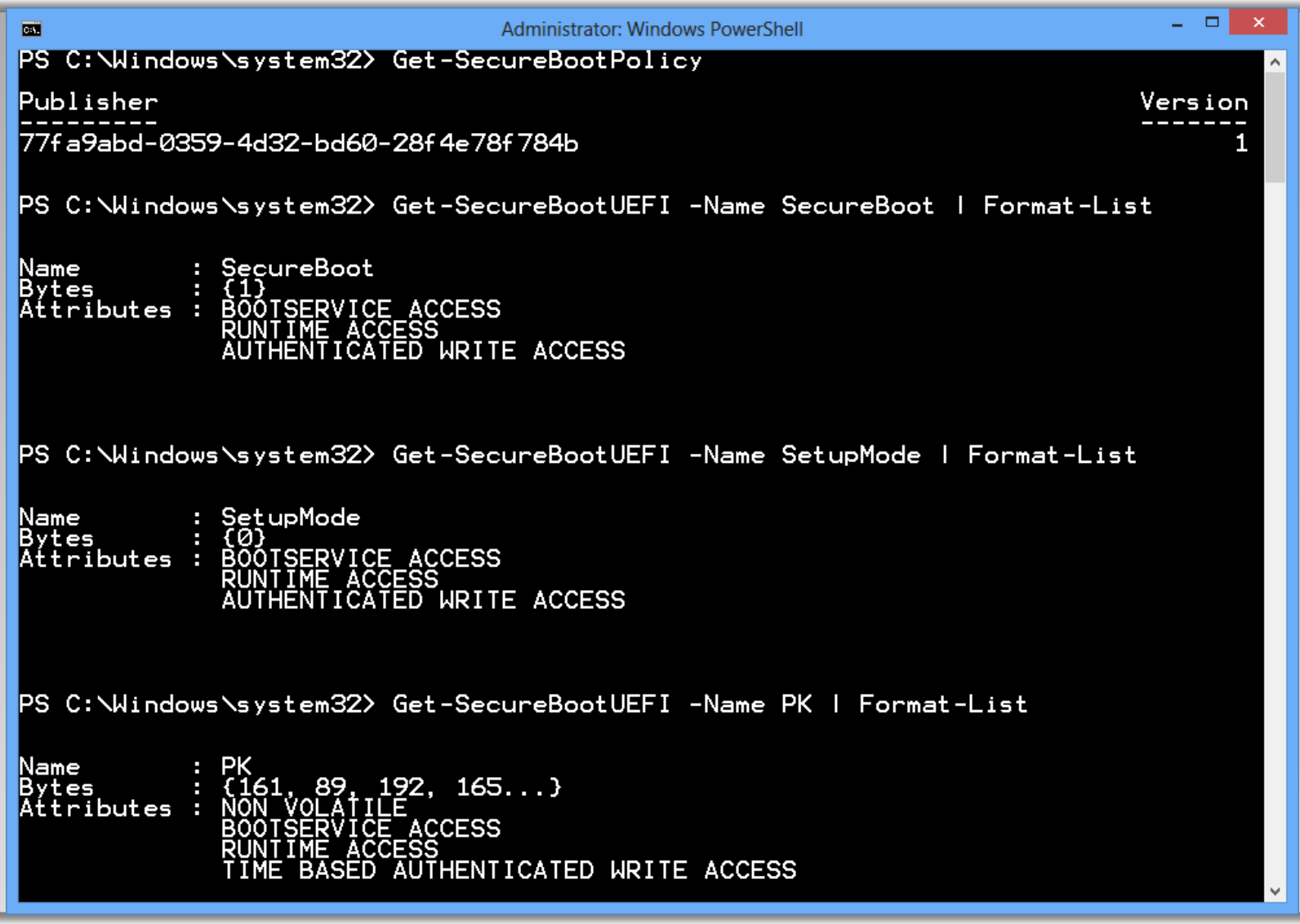
- Name (“PK”) or Vendor GUID (**8BE4DF61-93CA-11D2-AA0D-00E098032B8C**)
- Recall that AutenticatedVariableService DXE driver enters Secure Boot **SETUP_MODE** when correct “PK” EFI variable cannot be located in EFI NVRAM
- Main volatile SecureBoot variable is then set to DISABLE
- ImageVerificationLib then assumes Secure Boot is off and skips Secure Boot checks
- Generic exploit, independent of the platform/vendor
- **1 bit modification!**



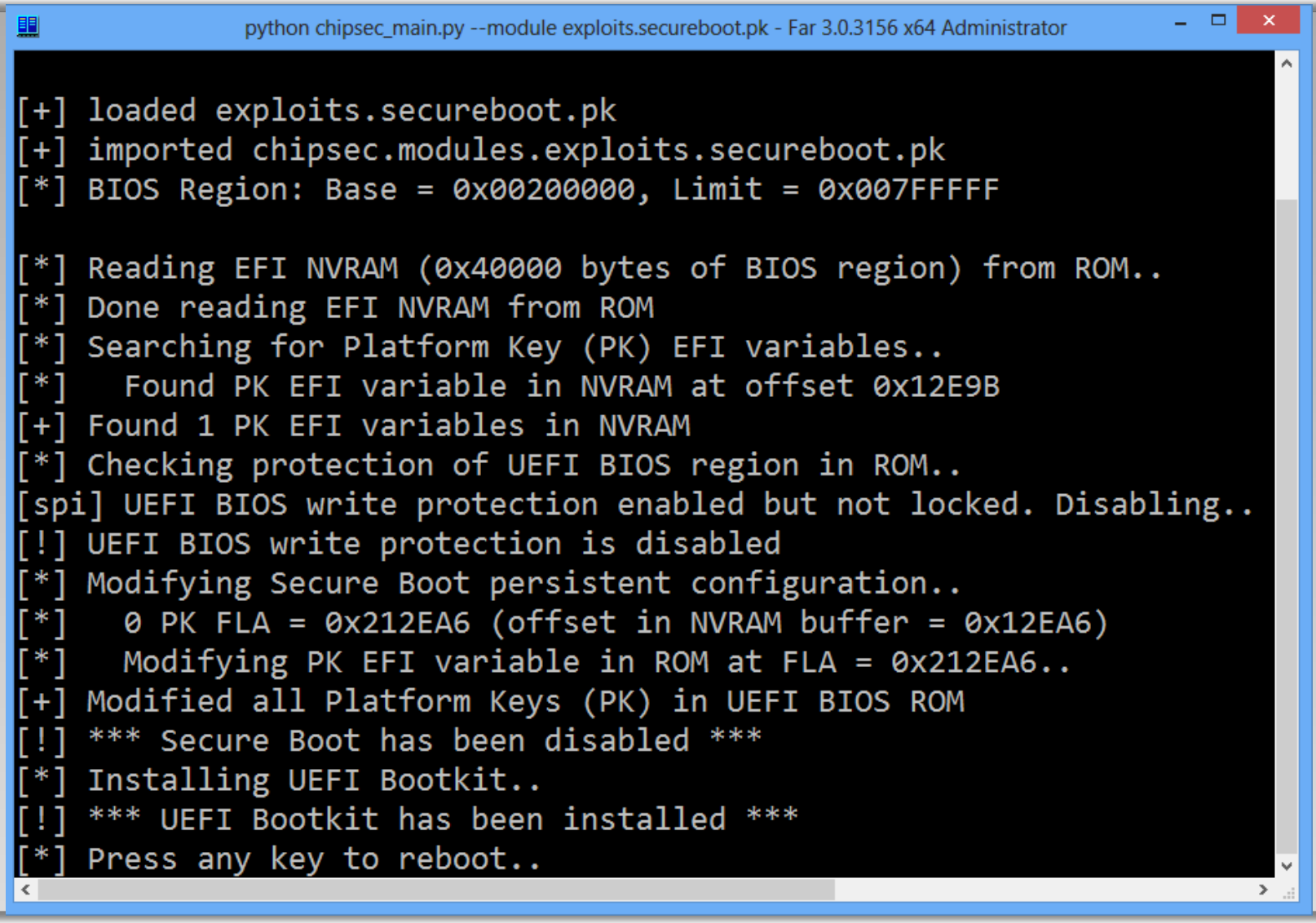
Corrupting Platform Key in ROM

Windows 8 HW Certification Requires Platforms to Protect UEFI Firmware and NVRAM with Secure Boot keys!

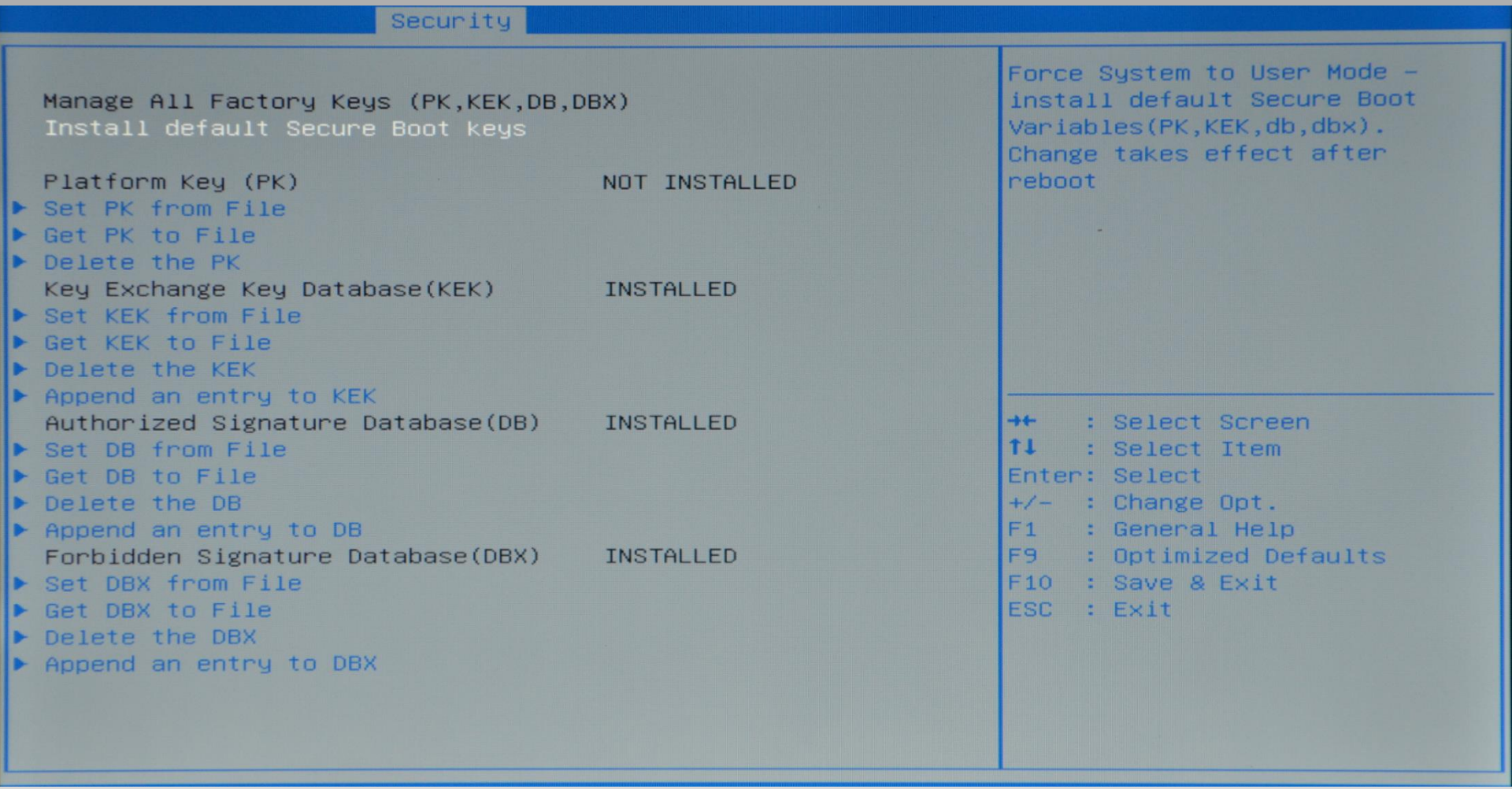
7. Mandatory. **Secure Boot must be rooted in a protected or ROM-based Public Key**. Secure Boot must be rooted in an RSA public key with a modulus size of at least 2048 bits, and either be based in unalterable ROM or otherwise protected from alteration by a secure firmware update process, as defined below.



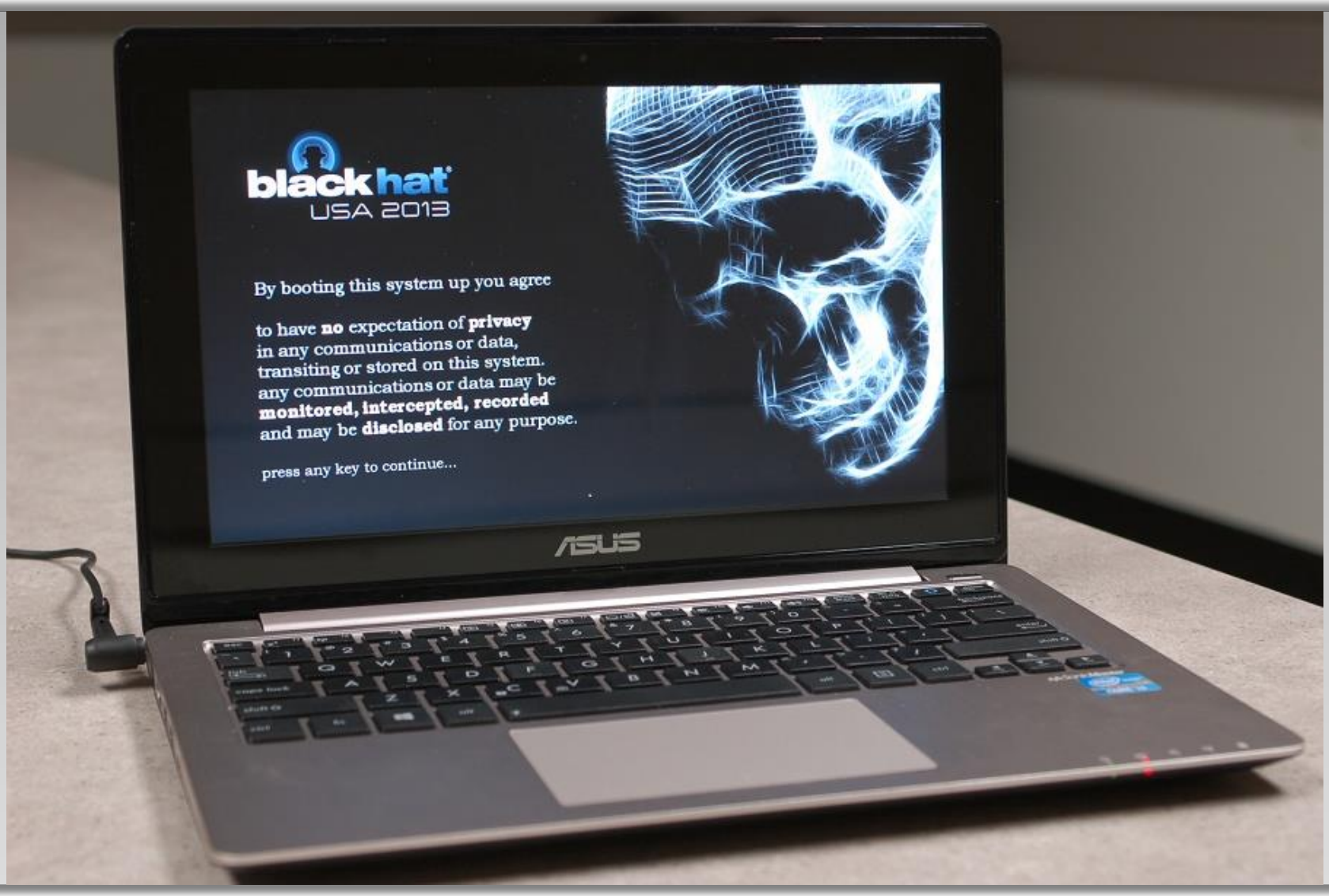
Secure Boot Is Enabled



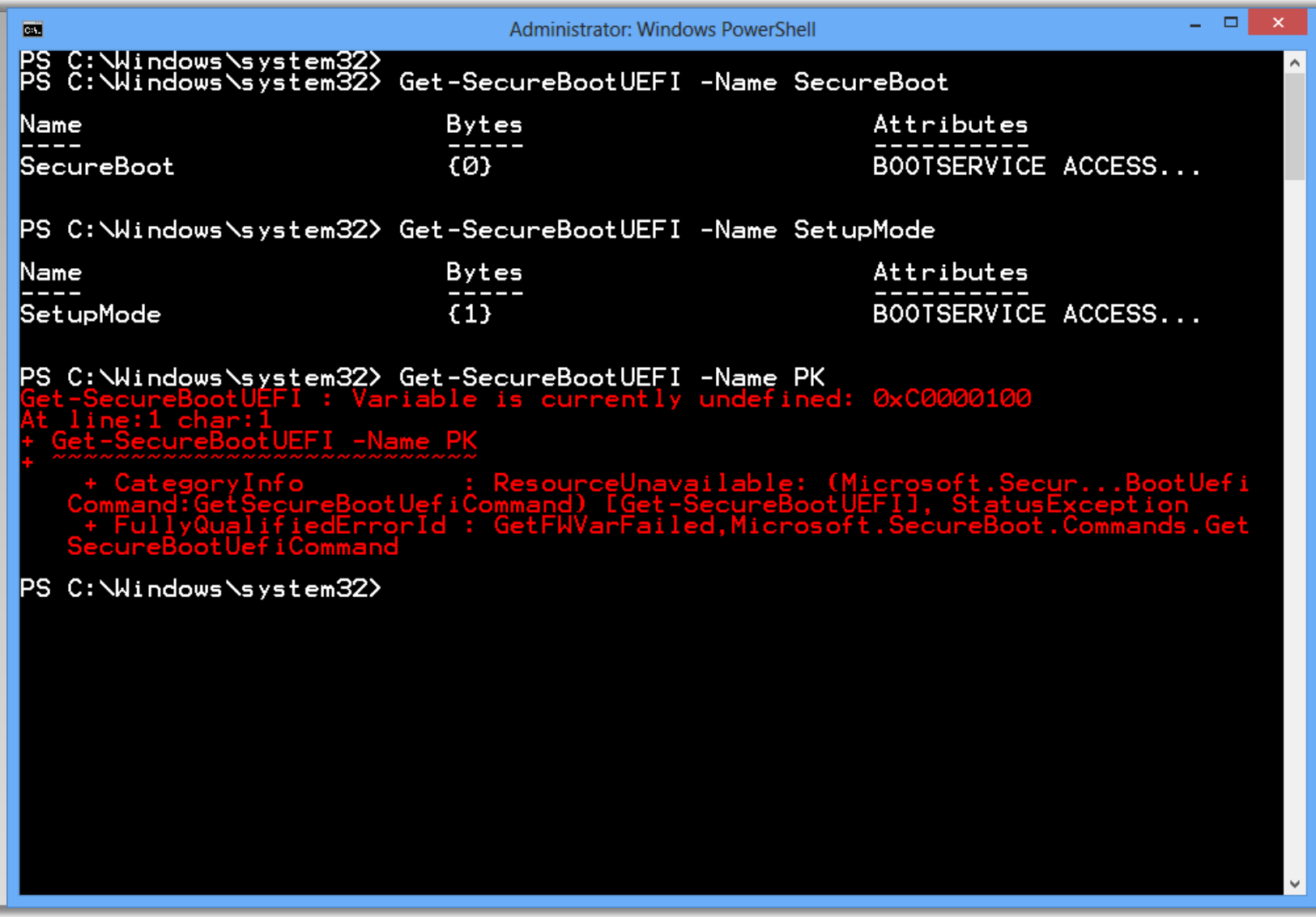
Corrupting Platform Key in NVRAM



Platform Key Is De-Installed



Для загрузки необходим номер вашей кредитной карты на securecreditcardz.ru



Back to Setup Mode ➔ Secure Boot Is Off

Demo 1

Attacking Windows 8 Secure Boot on ASUS VivoBook Q200E

This issue does not affect platform vendors correctly protecting their UEFI BIOS in ROM and during BIOS Update but

When UEFI firmware is not adequately protected (in ROM or during update), subverting UEFI Secure Boot is not the only thing to worry about!

S-CRTM and TPM based Measured Boot including Full-Disk Encryption solutions relying on the TPM can also be subverted

Evil Maid Just Got Angrier

BIOS Chronomancy by John Butterworth, Corey Kallenberg, Xeno Kovah

Or you can get infected with UEFI BIOS or SMM malware

a.k.a. “extremely persistent malware” © .gov

Persistent BIOS Infection by Anibal Sacco, Alfredo Ortega
Hardware Backdooring is Practical by Jonathan Brossard
The Real SMM Rootkit by core collapse
SMM Rootkits by Shawn Embleton, Sherri Sparks, Cliff Zou

Huh! It requires kernel exploit?

Why Not Just Directly Modify Secure Boot Keys from the OS? There's an API for that

```
# chipsec_util.py uefi writevar PK 8BE4DF61-
  93CA-11D2-AA0D-00E098032B8C PK_forged.bin
```

SetFirmwareEnvironmentVariable failed

[Error 5] Access is Denied.

```
C:\Users\bh2013\Desktop\bh2013\chipsec-1.0>python chipsec_util.py uefi writevar PK 8BE4DF61-93CA-11D2-AA0D-00E098032B8C PK_forged.bin
WConio package is not installed. No colored output
[CHIPSEC] Writing EFI variable Name='PK' GUID={8BE4DF61-93CA-11D2-AA0D-00E098032B8C} from 'PK_forged.bin' via Variable API..
Writing EFI variable:
Name      : PK
GUID      : 8BE4DF61-93CA-11D2-AA0D-00E098032B8C
[-] ERROR: SetFirmwareEnvironmentVariable failed (GetLastError = 0x5)
[Error 5] Access is denied.
[CHIPSEC] (uefi) time elapsed 0.000

C:\Users\bh2013\Desktop\bh2013\chipsec-1.0>python chipsec_util.py uefi writevar SetupMode 8BE4DF61-93CA-11D2-AA0D-00E098032B8C SetupMode_mod.bin
WConio package is not installed. No colored output
[CHIPSEC] Writing EFI variable Name='SetupMode' GUID={8BE4DF61-93CA-11D2-AA0D-00E098032B8C} from 'SetupMode_mod.bin' via Variable API..
Writing EFI variable:
Name      : SetupMode
GUID      : 8BE4DF61-93CA-11D2-AA0D-00E098032B8C
[-] ERROR: SetFirmwareEnvironmentVariable failed (GetLastError = 0x5)
[Error 5] Access is denied.
[CHIPSEC] (uefi) time elapsed 0.000
```

Secure Boot Variables

Remember Secure Boot Key variables are "Authenticated Write Access"

You have to sign EFI variable and have corresponding X509 Cert in NVRAM (PK/KEK/certdb)

Is it possible to bypass Windows 8 Secure Boot and install UEFI bootkit by remote user mode exploit?

Coordinated disclosure of multiple vulnerabilities to affected BIOS and platform vendors is ongoing but we can offer a demo

Demo 2

Attacking Windows 8 Secure Boot from user-mode

Now what?

Only signed updates should be allowed

- Signed UEFI Capsule based update via S3/reset
- Run-time update from within SMM only

Protect UEFI firmware in ROM

- Use BIOS Control to enable write protection of the entire BIOS region in SPI flash
- Use Protected Range registers to write-protect ranges of SPI flash

Protect EFI variable store (NVRAM) in ROM

- NVRAM contains Secure Boot keys and NV configuration

Measuring Secure Boot configuration into PCR[7] may prevent or complicate certain exploits

- Facilitates detection via remote attestation
- Windows BitLocker may seal encryption keys to PCR[7]
- Microsoft Hardware Certification Requirements

Recommendations (Platform Vendors)

- **Make sure platform has Windows 8 Logo**
  - Such platform has to adhere to the security requirements in **System.Fundamentals.Firmware.UEFISecureBoot**
- **Check if UEFI firmware updates are signed**
  - Corrupt firmware update binary and feed it to the BIOS update utility
- **Ask if and how platform adheres to the BIOS Protection Guidelines (NIST SP 800-147)**

Recommendations

Acknowledgements

Black Hat organizers and review committee

Bruce Monroe, John Loucaides, Ben Parmeter, Paul Alappat; Microsoft's MSRC and Secure Boot team for coordinating vendor disclosures

Nicholas Adams, Kirk Brannock, doughty, Dhinesh Manoharan, Brian Payne, Mickey Shkatov, Vincent Zimmer, Monty Wiseman for help & support of this work

Greetz to Misha, @j4istal, @apebit, @drraid, sharkey, secoeites, @tobyhush, @matrosov

- *BIOS SECURITY* by John Butterworth, Corey Kallenberg and Xeno Kovah
- *UART THOU MAD?* by Toby Kohlenberg and Mickey Shkatov
- *ANDROID: ONE ROOT TO OWN THEM ALL* by Jeff Forristal

Don’t Miss These Talks!

Thank You!

@c7zero
@Abazhanyuk
andrew.furtak@gmail.com