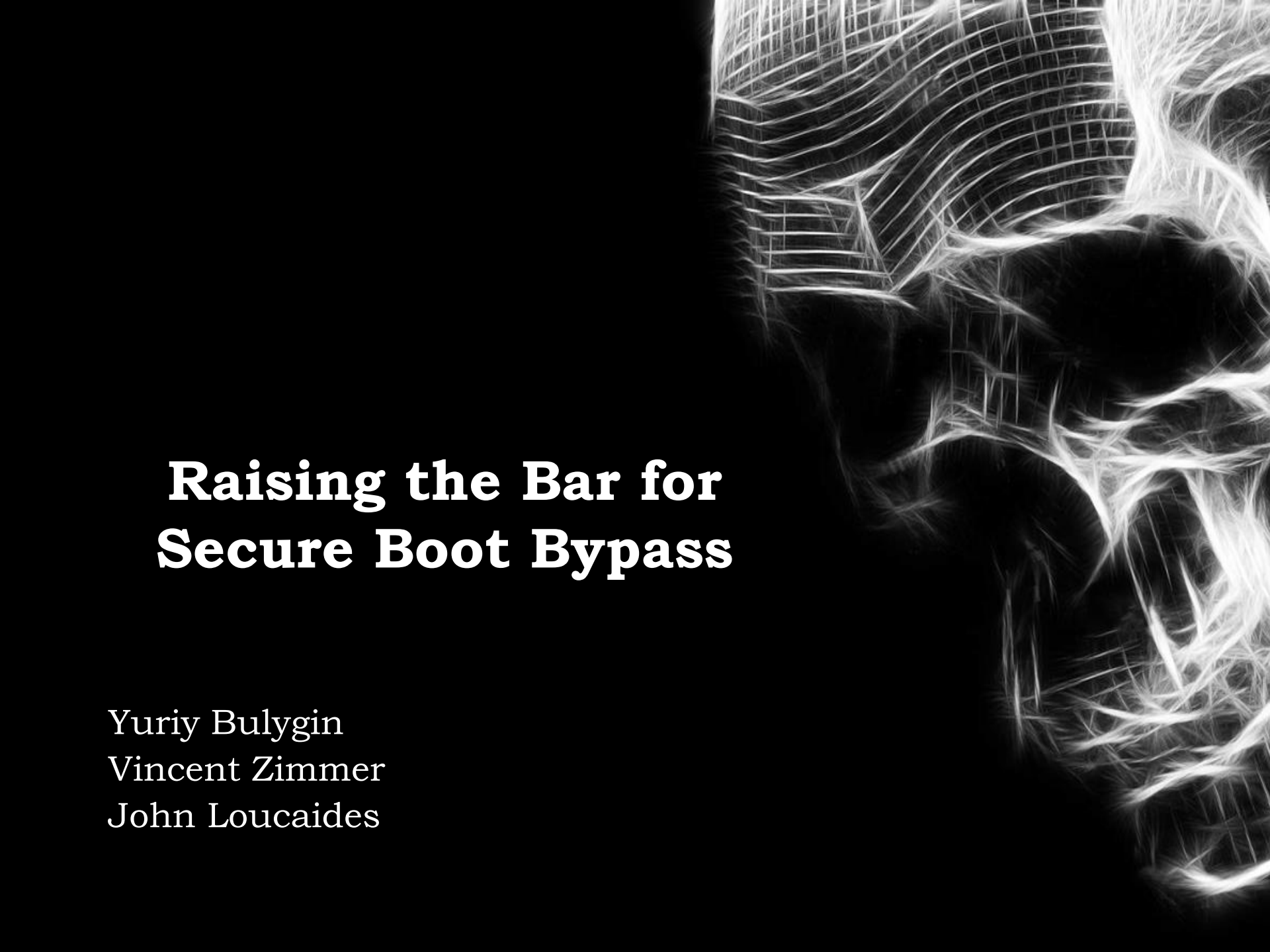


Raising the Bar for Secure Boot Bypass

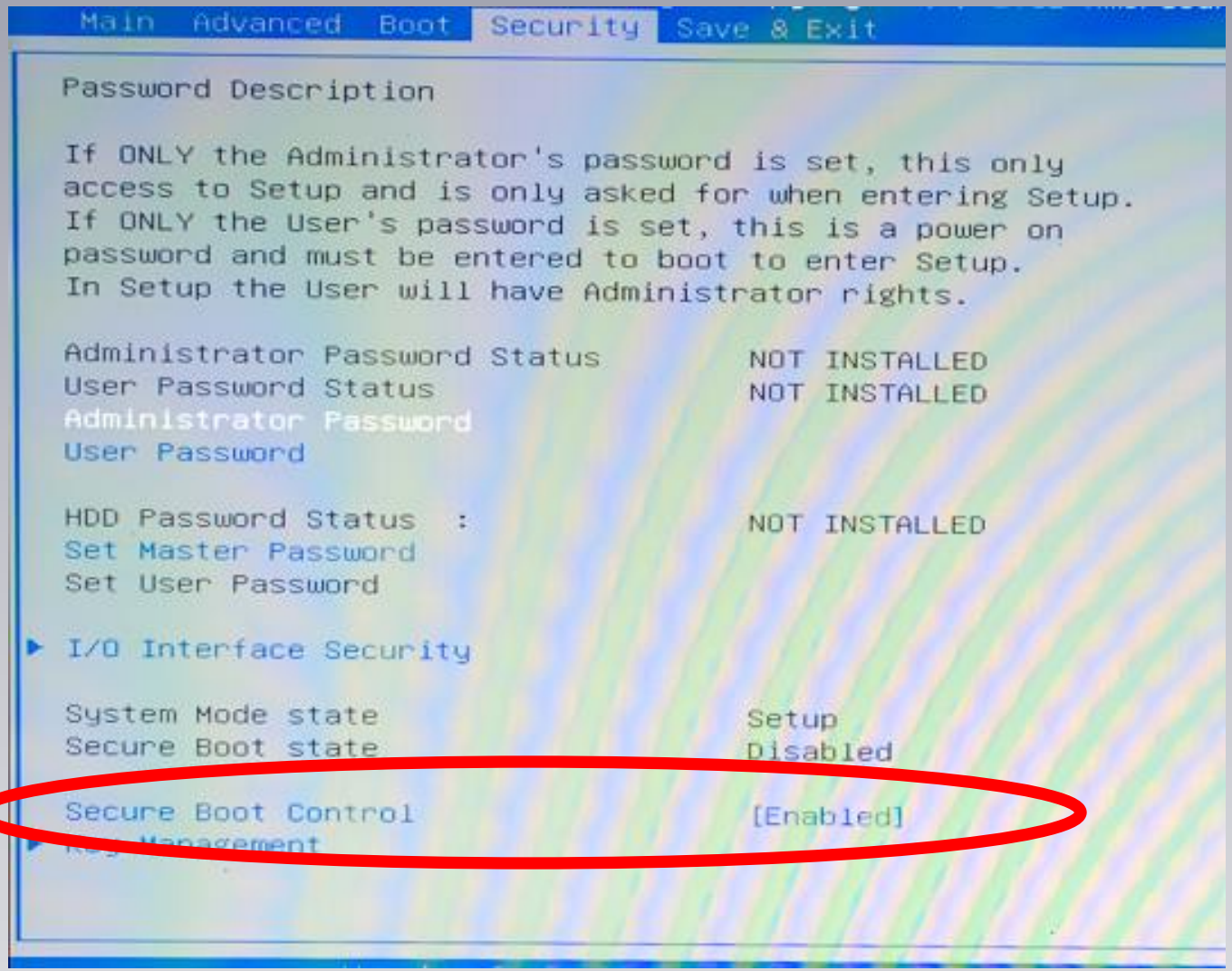
Yuriy Bulygin
Vincent Zimmer
John Loucaides

- Attacks
- Protection
- Coordination

Agenda

Attacking Windows 8 Secure Boot

Based on A Tale of One Software Bypass of Windows 8 Secure Boot by Andrew Furtak, Oleksandr Bazhaniuk and Yuriy Bulygin

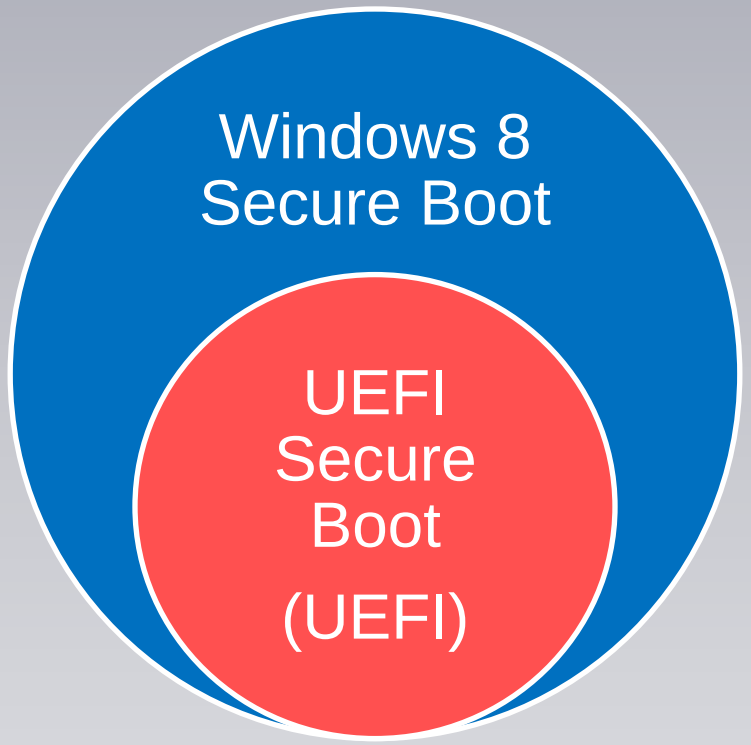

Main Advanced Boot Security Save & Exit
Password Description
If ONLY the Administrator's password is set, this only
access to Setup and is only asked for when entering Setup.
If ONLY the User's password is set, this is a power on
password and must be entered to boot to enter Setup.
In Setup the User 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]

Demo 1

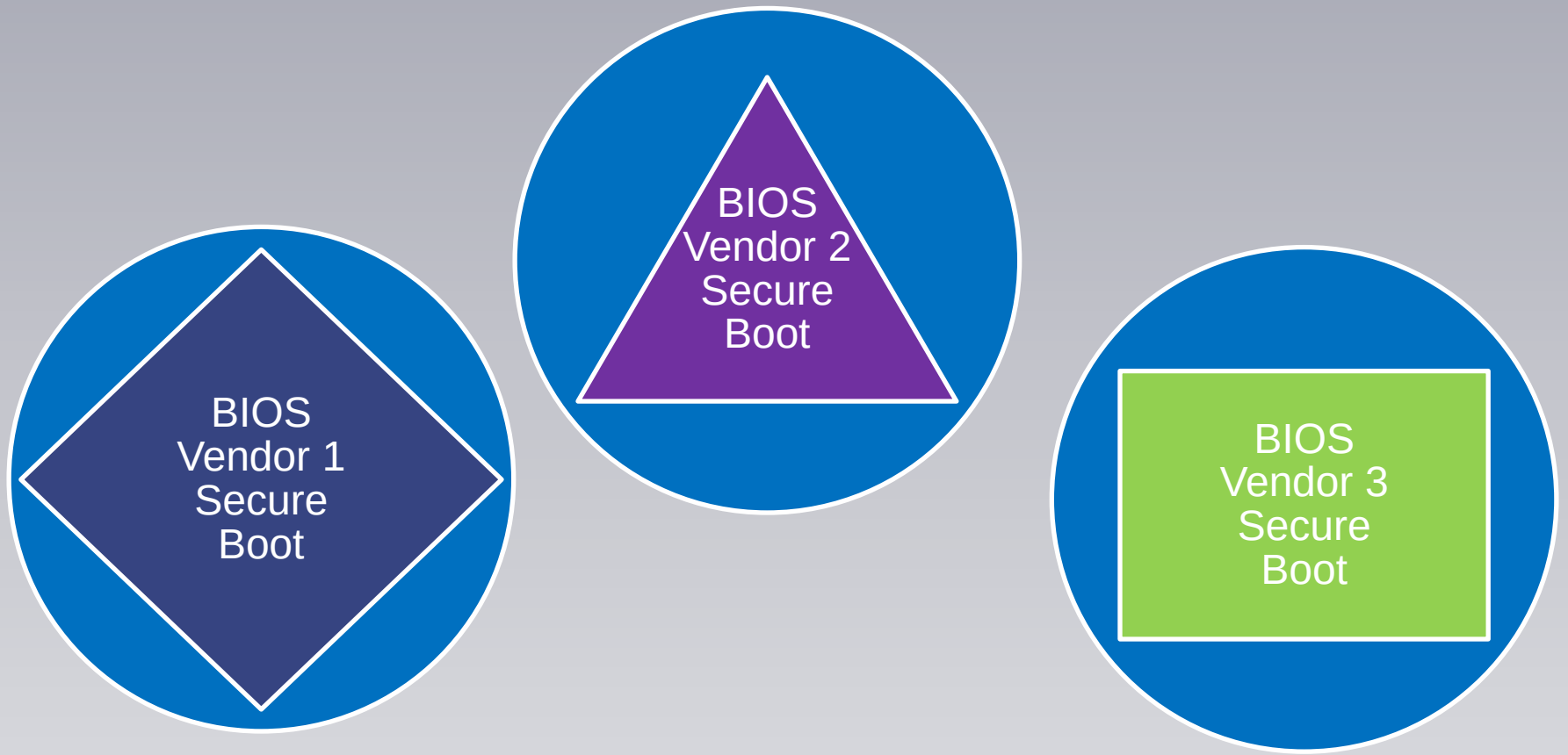
Attacking Windows 8 Secure Boot on ASUS VivoBook Q200E

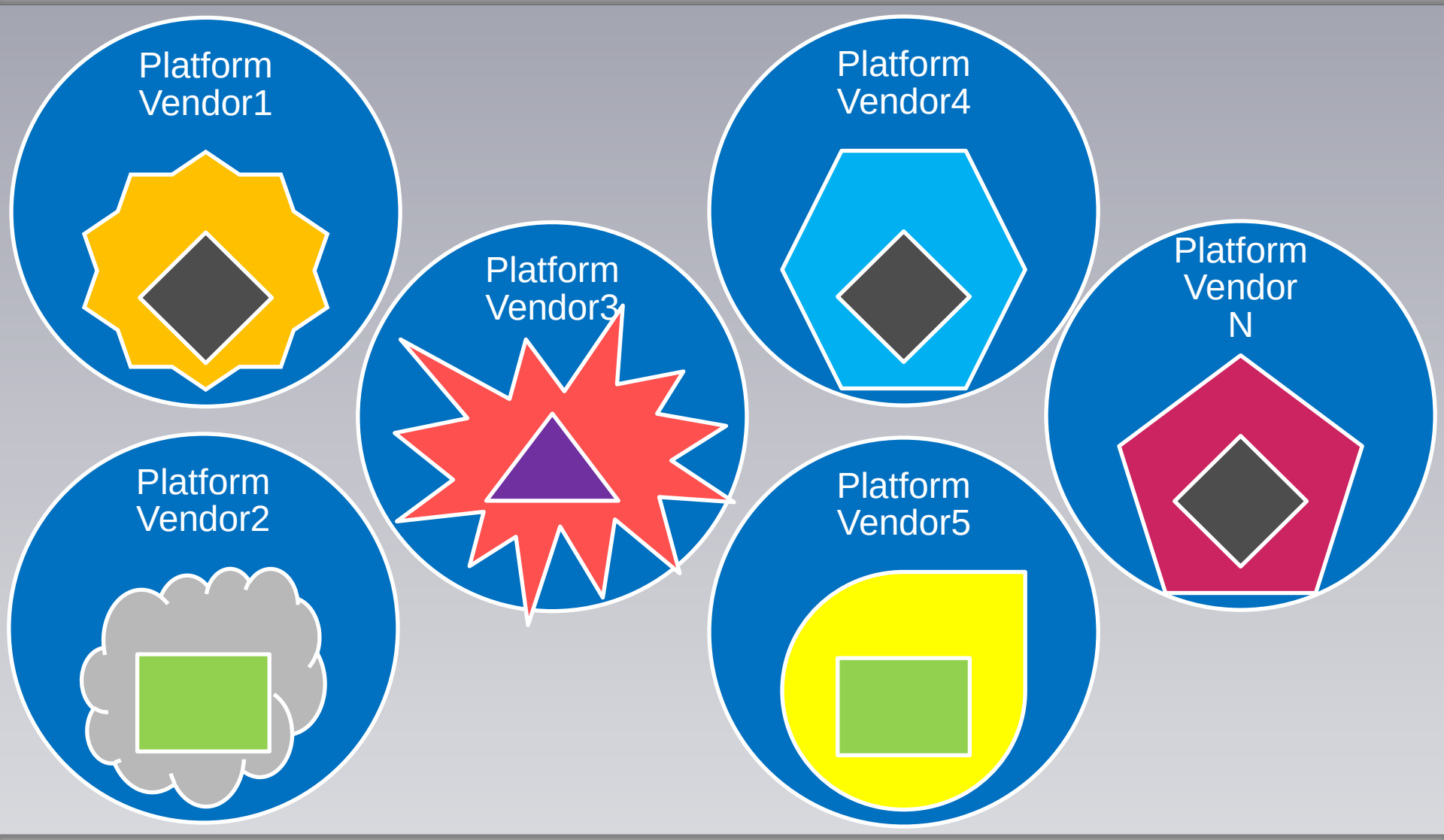
We think Windows 8 Secure Boot looks like this

Or more like this



How exciting! … But still not close

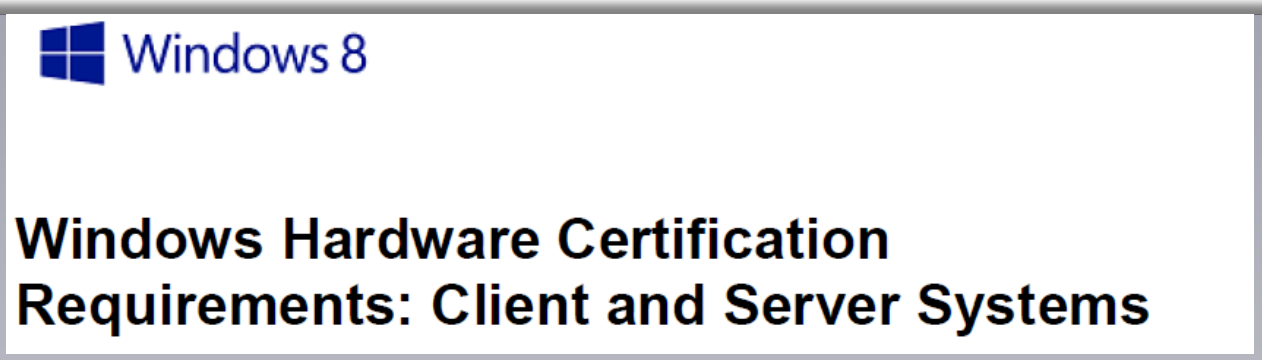




The Reality Is Much More Exciting

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

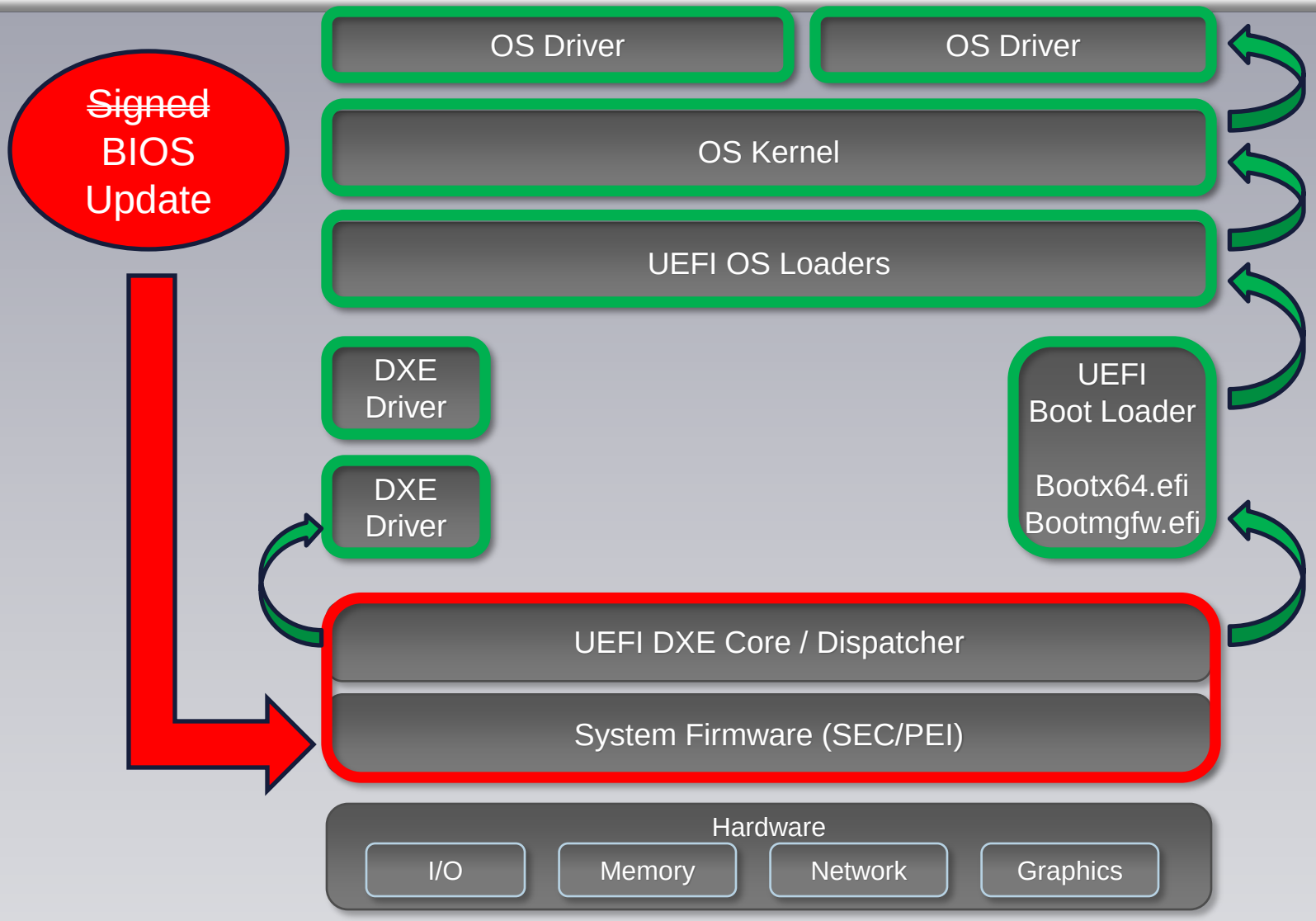
14 No in-line mechanism is provided whereby a user can bypass Secure Boot failures and boot anyway

…

Windows 8 Secure Boot Requirements


Master

What If UEFI BIOS Updates Are Not Signed?



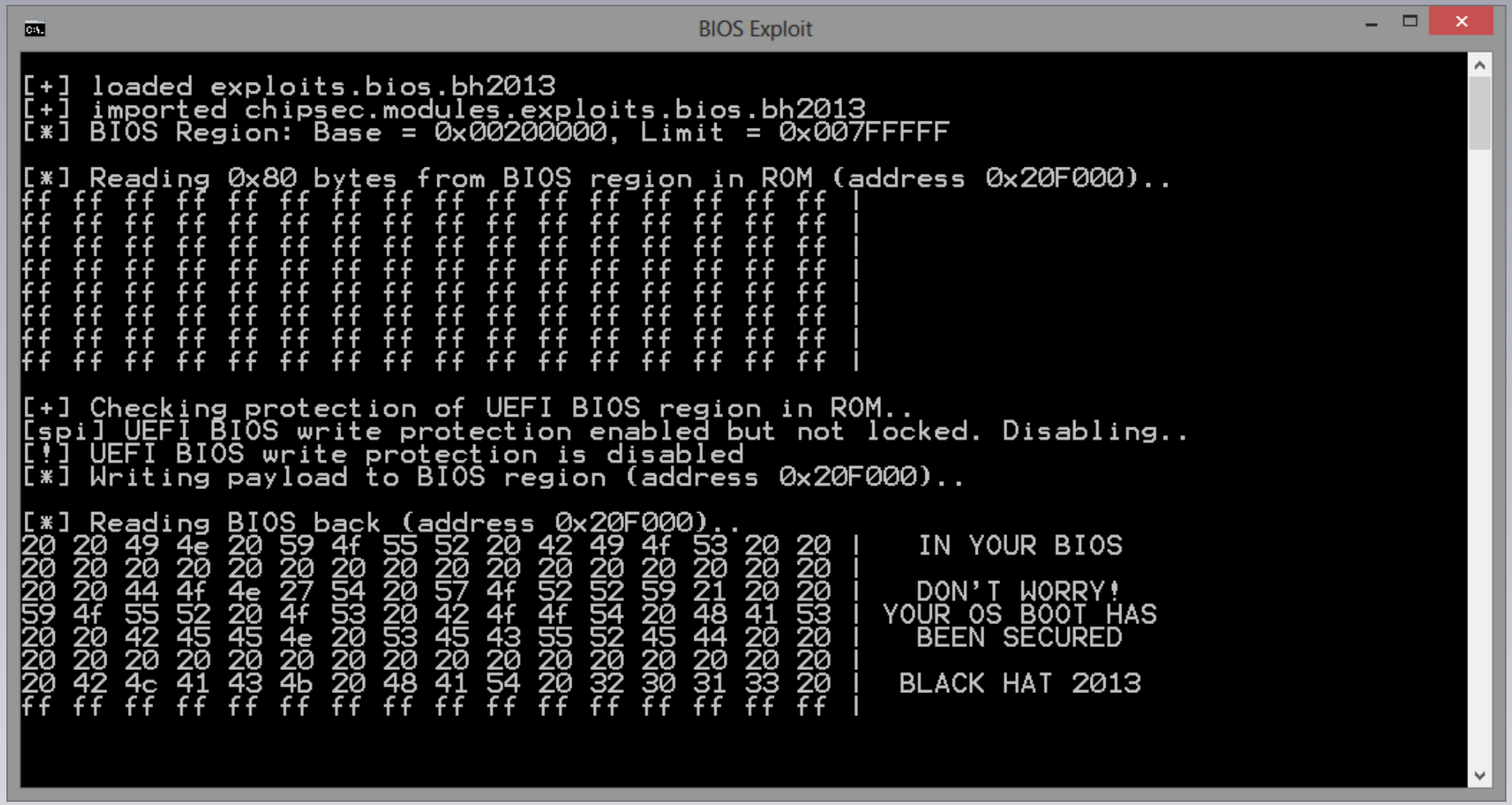
When UEFI Firmware Updates Are Not Signed

No luck 

UEFI firmware update capsules are signed

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

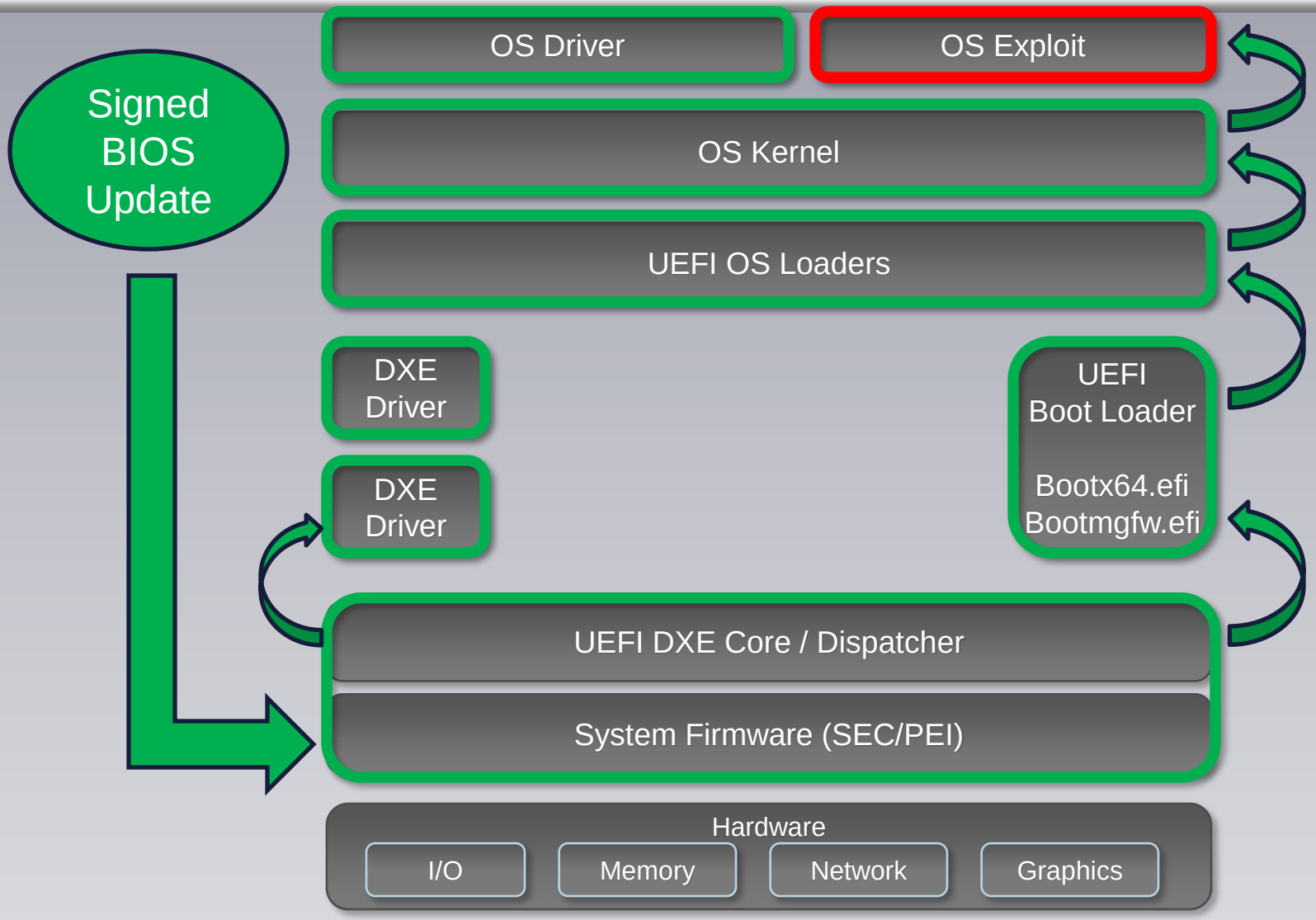
Wait, let's check one little thing…



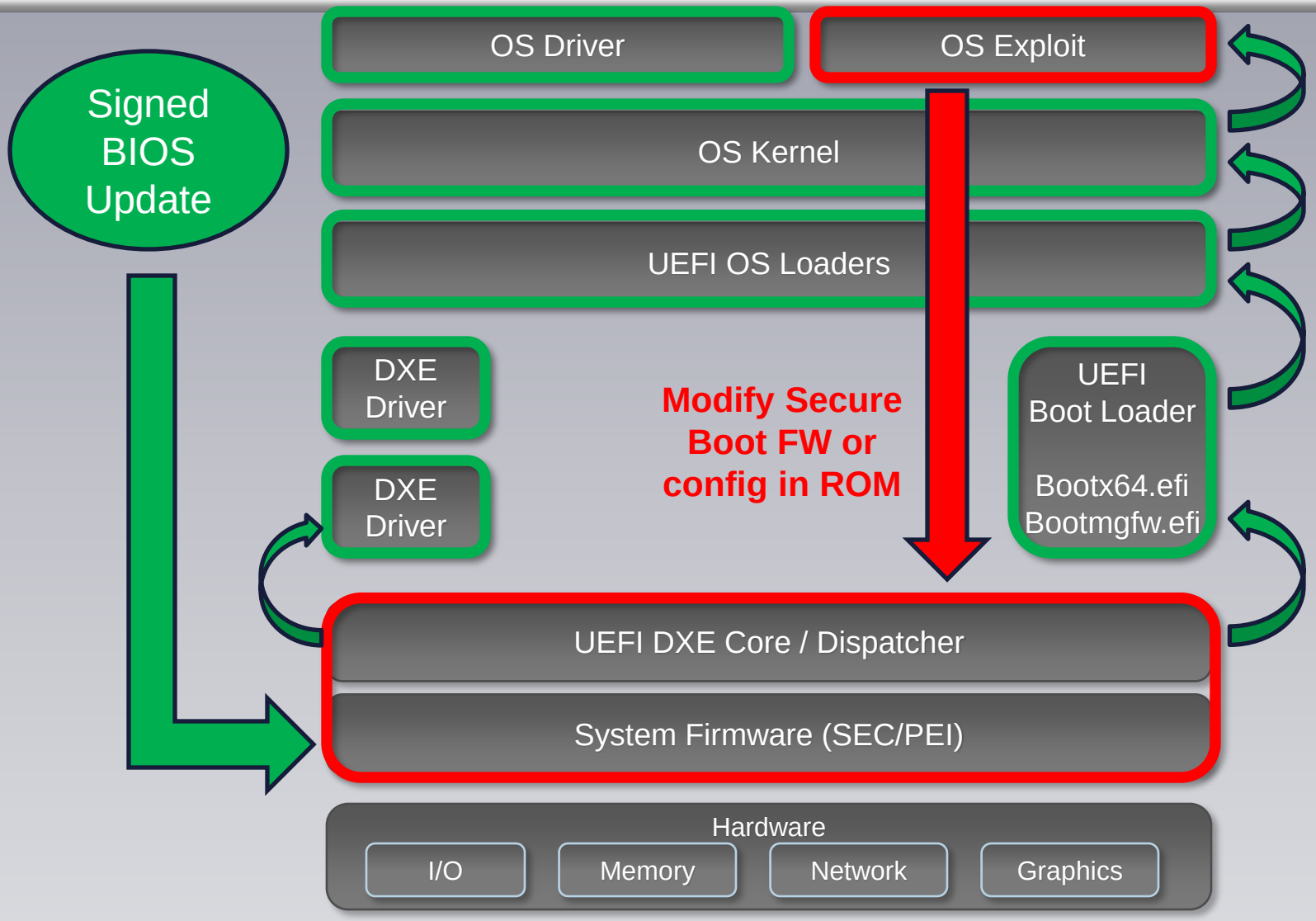
Can We Write to UEFI Firmware in ROM?

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?

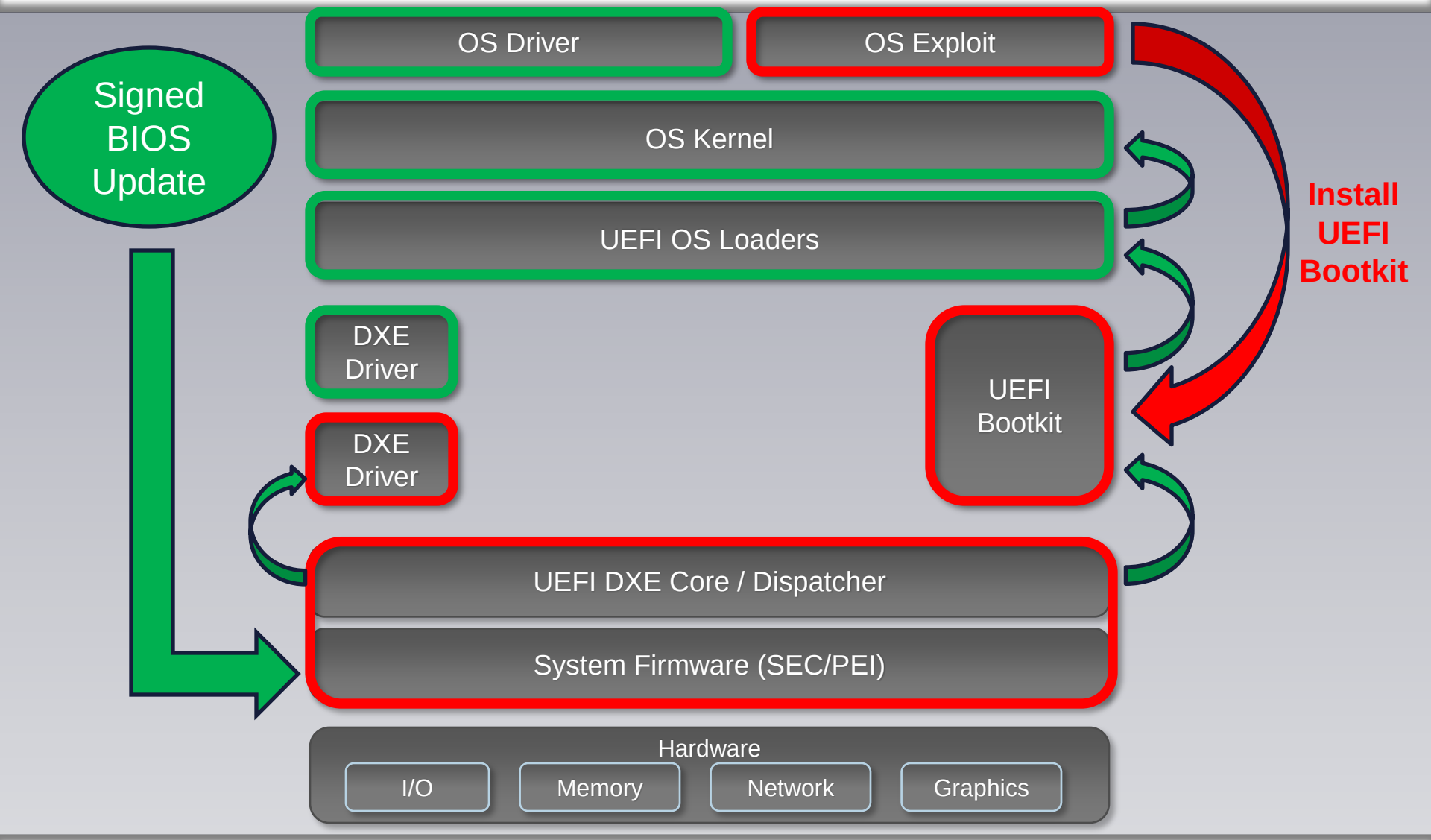
Hint: Malware could patch DXE Image Verification driver in ROM or it could change persistent Secure Boot keys/configuration in NVRAM



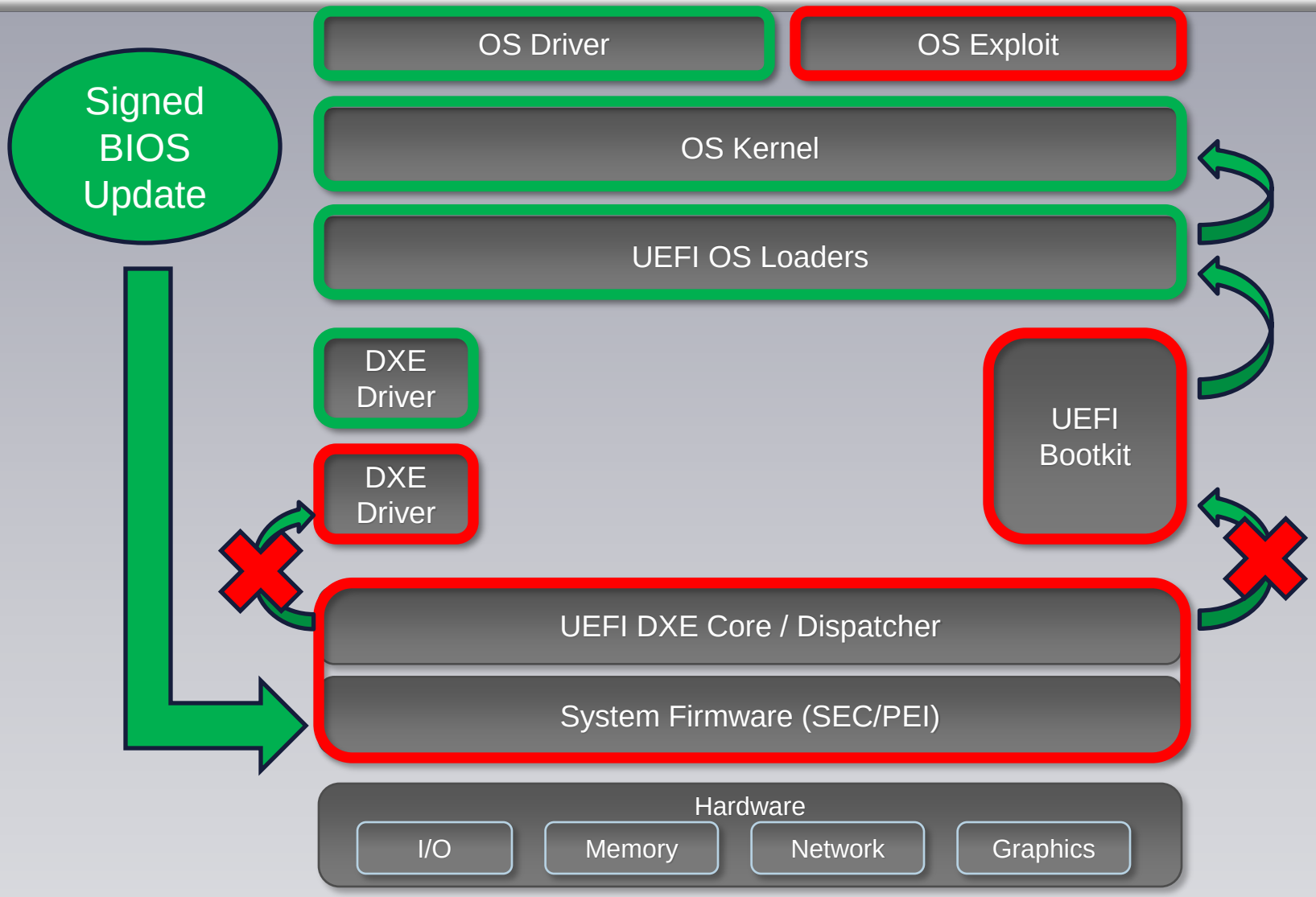
When Firmware Is Not Protected in ROM



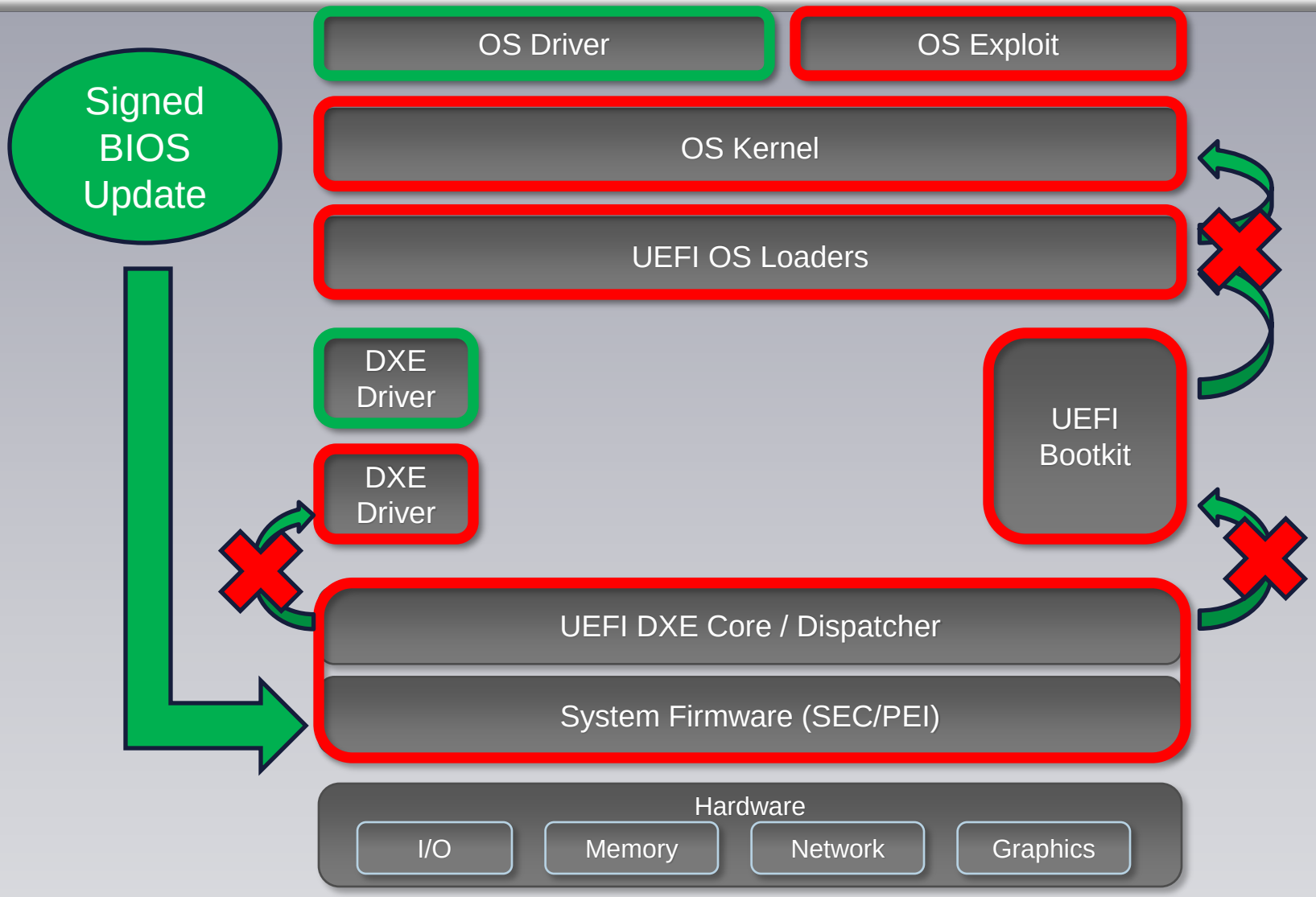
Malware Modifies UEFI Firmware in ROM (directly programming SPI controller)



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

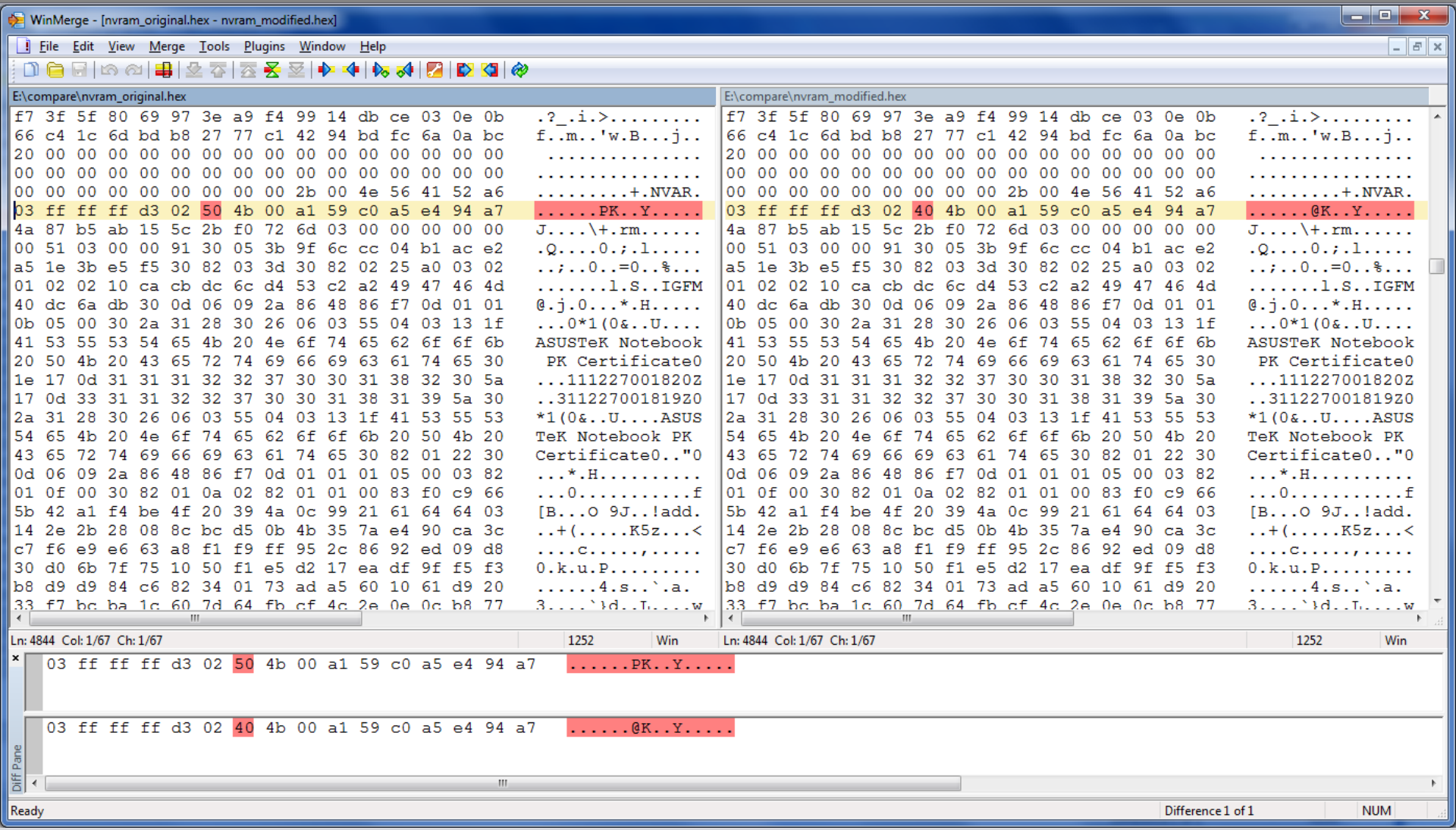
Exploit Strategies

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

Corrupt Platform Key EFI variable in NVRAM

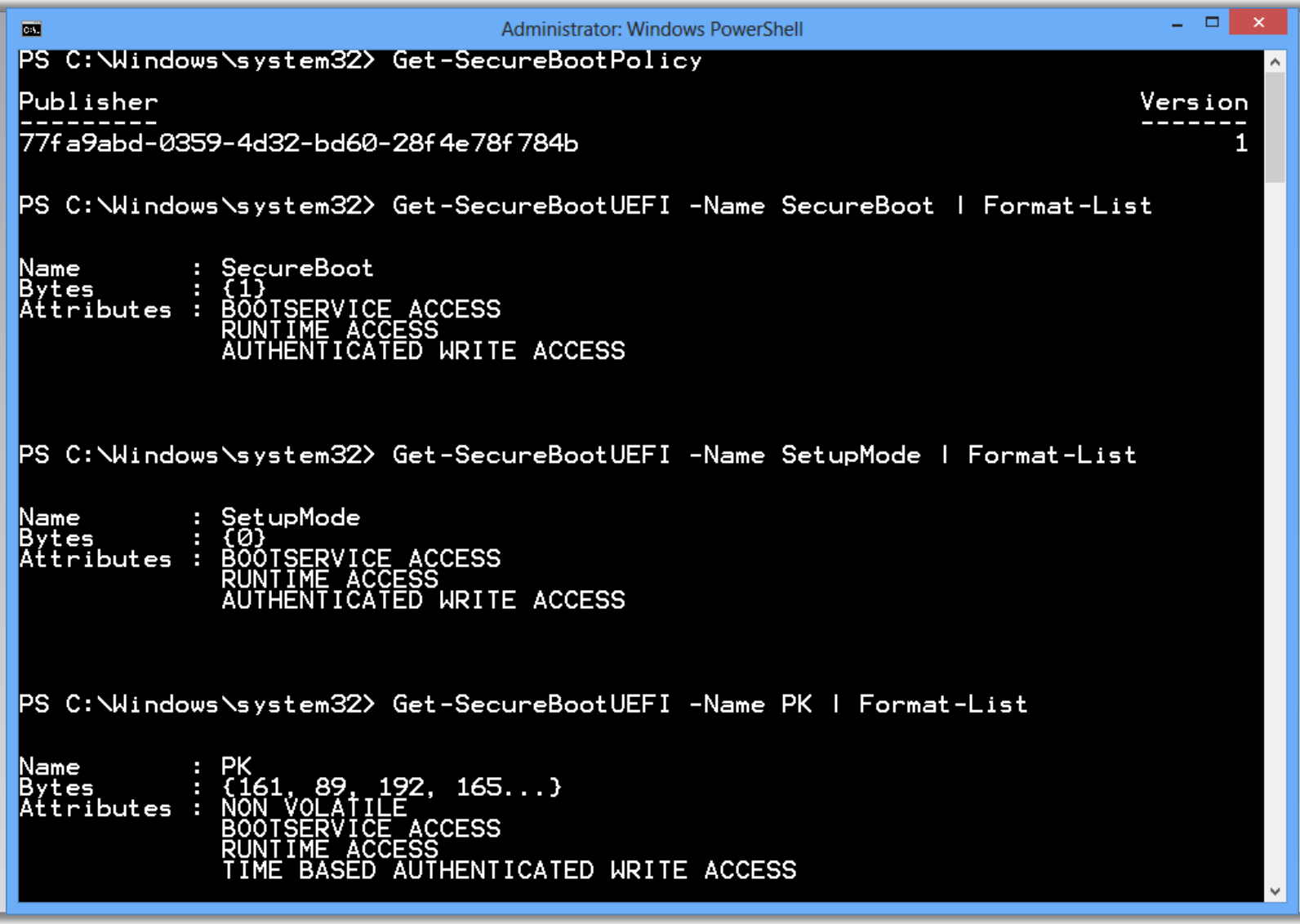
- Name (“PK”) or Vendor GUID (**8BE4DF61-93CA-11D2-AA0D-00E098032B8C**)
- Recall that AutenticatedVariableService DXE driver enters Secure Boot **SETUP_MODE** when correct “PK” EFI variable cannot be located in EFI NVRAM
- Main volatile SecureBoot variable is then set to DISABLE
- ImageVerificationLib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



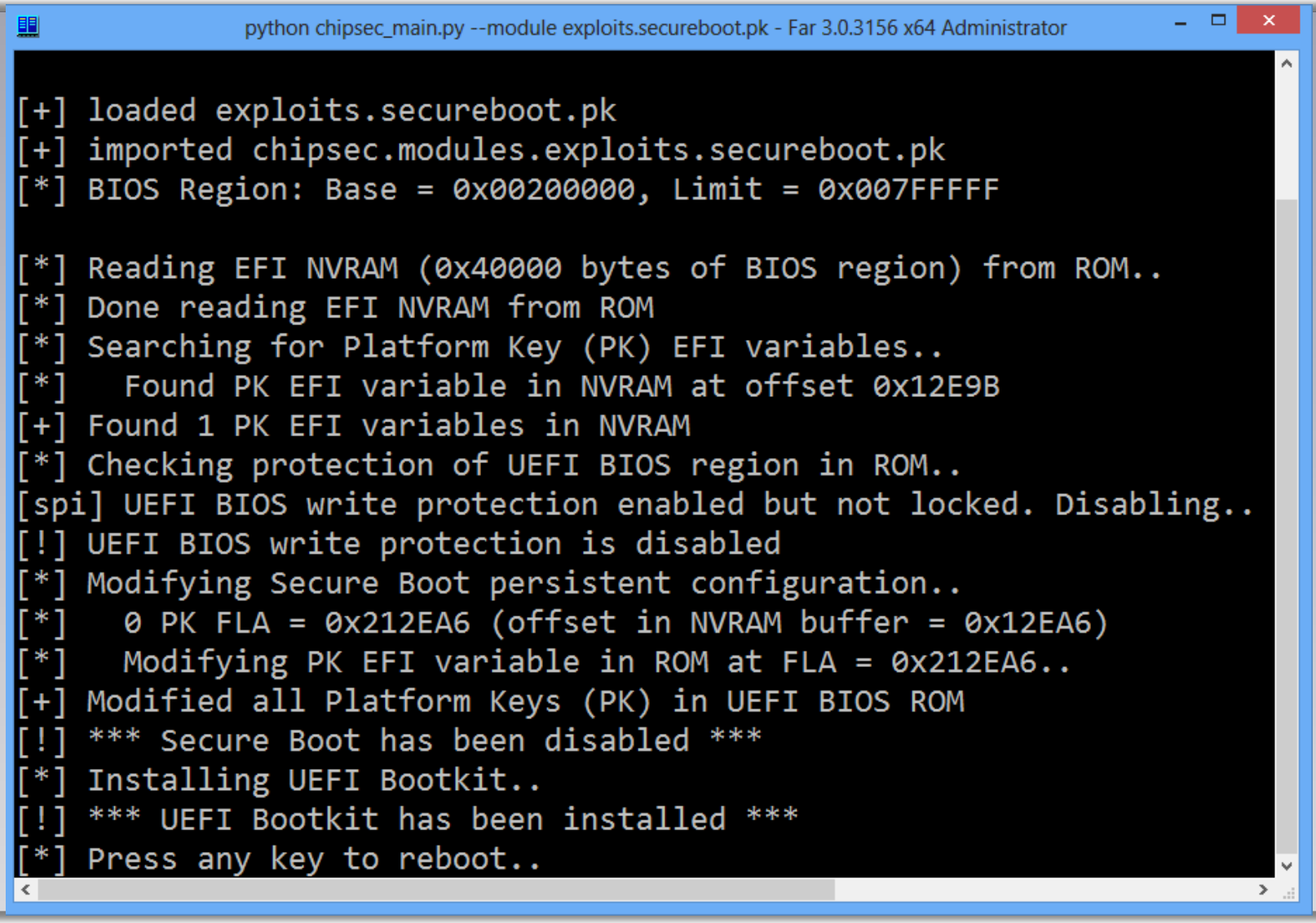
Corrupting Platform Key in ROM

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

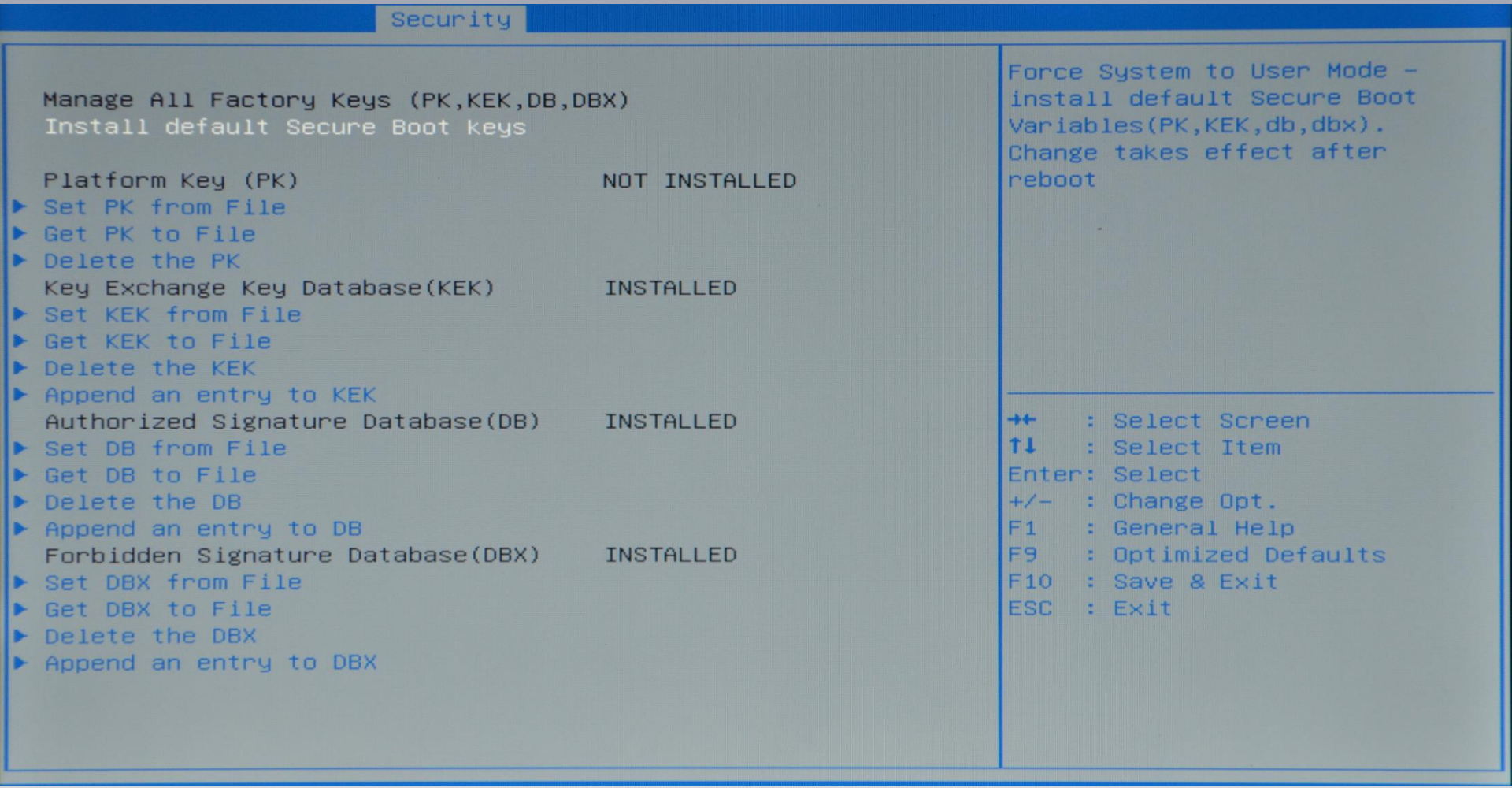
7. Mandatory. **Secure Boot must be rooted in a protected or ROM-based Public Key**. Secure Boot must be rooted in an RSA public key with a modulus size of at least 2048 bits, and either be based in unalterable ROM or otherwise protected from alteration by a secure firmware update process, as defined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



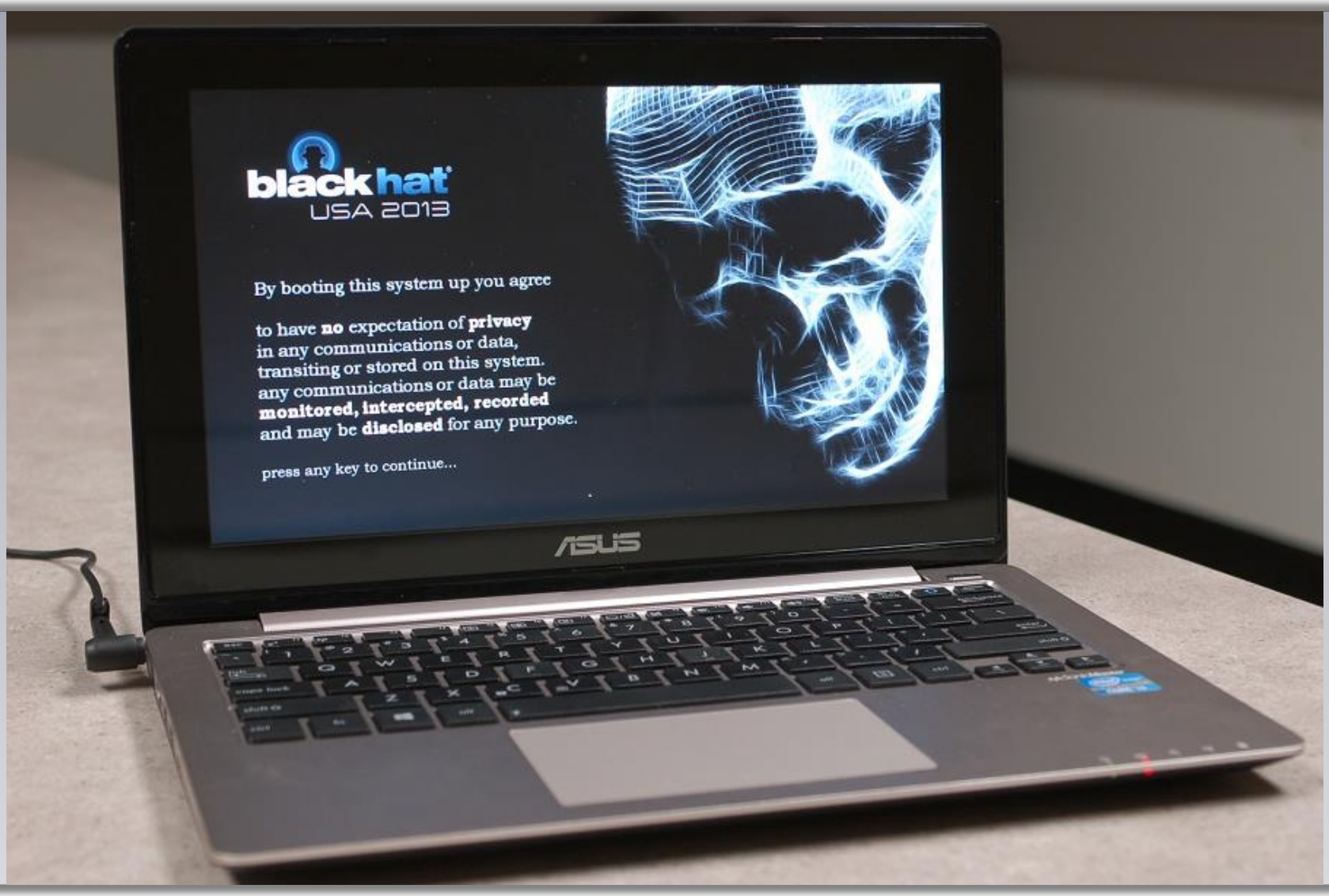
Secure Boot Is Enabled



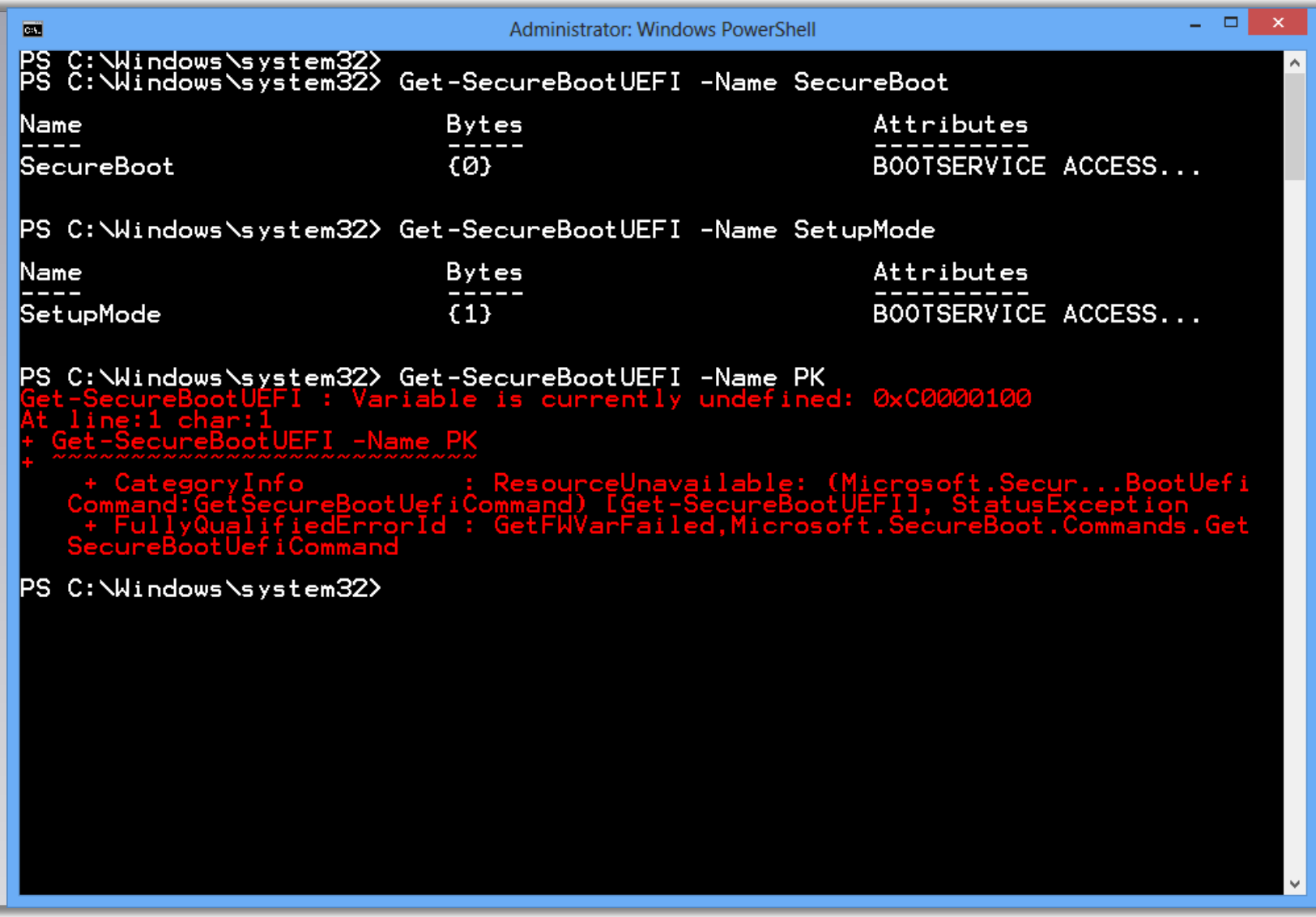
Corrupting Platform Key in NVRAM



Platform Key Is De-Installed



Для загрузки необходим номер вашей кредитной карты на securecreditcardz.ru



Back to Setup Mode ➔ Secure Boot Is Off

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?

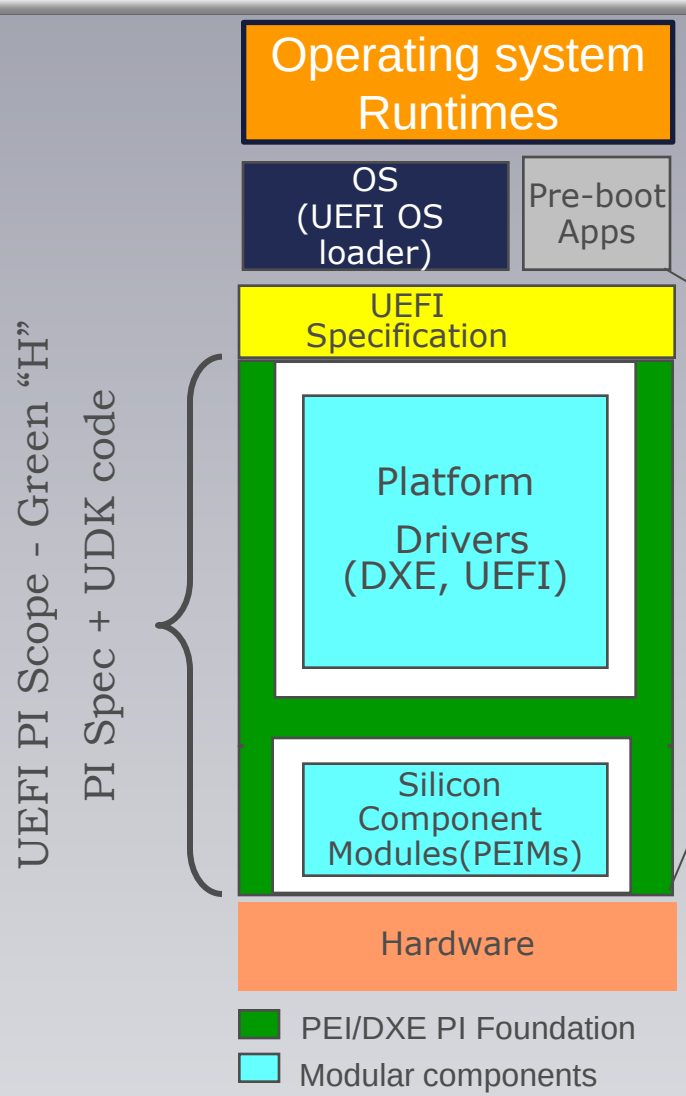
Is it possible to bypass Windows 8 Secure Boot and install UEFI bootkit by remote user mode exploit?

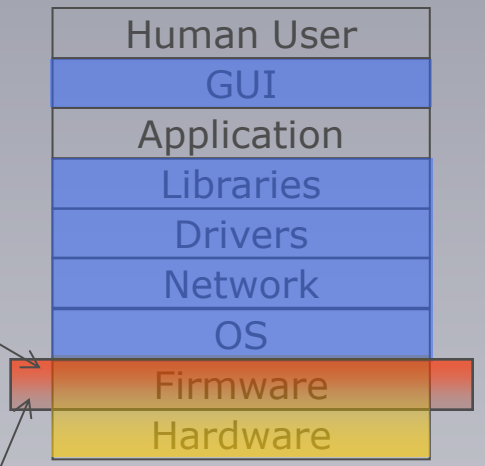
Coordinated disclosure of multiple vulnerabilities to affected BIOS and platform vendors is ongoing but we can offer a demo

Demo 2

Attacking Windows 8 Secure Boot from user-mode

UEFI Protections

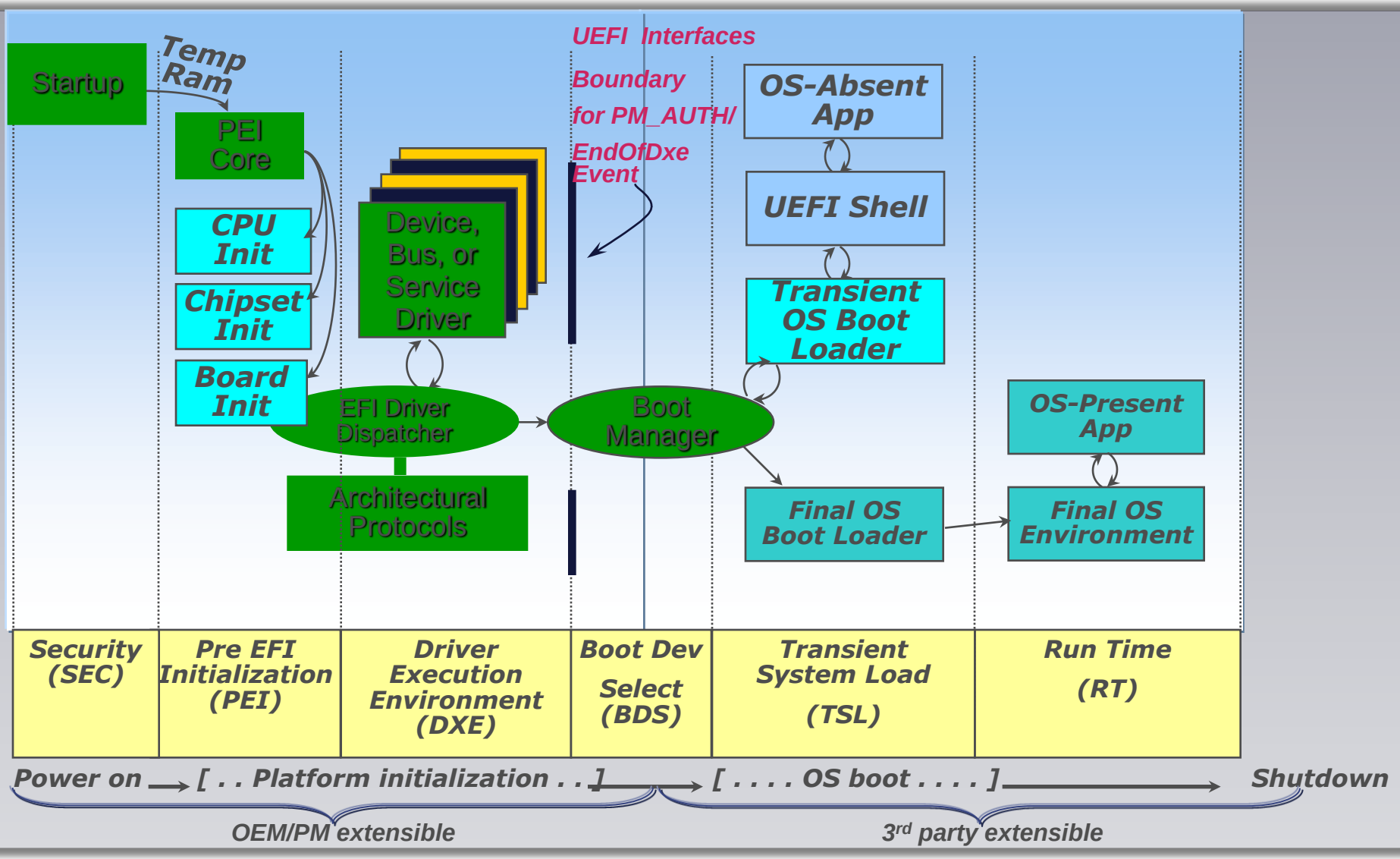




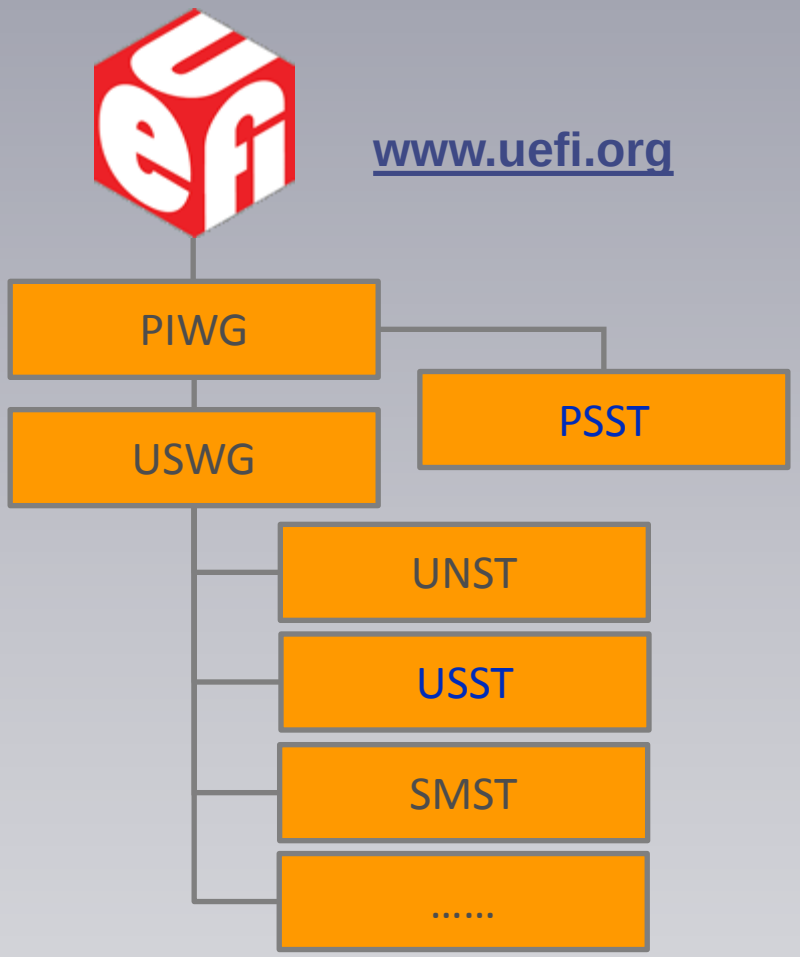
Full system stack (user -> hardware)

- UEFI 2.3.1c specifies how firmware boots OS loader and drivers– **3rd party extensible**
- UEFI's Platform Initialization (PI) 1.2.1
- Architecture specifies how Driver Execution Environment (DXE) Drivers and Pre-EFI Initialization (PEI) Modules (PEIMs) initialize Silicon and the platform
- DXE is preferred UEFI Implementation
- PEIMs, UEFI and DXE drivers implements networking, Update, other security features - **Only Update by the OEM**

Building UEFI – Platform Initialization (PI)



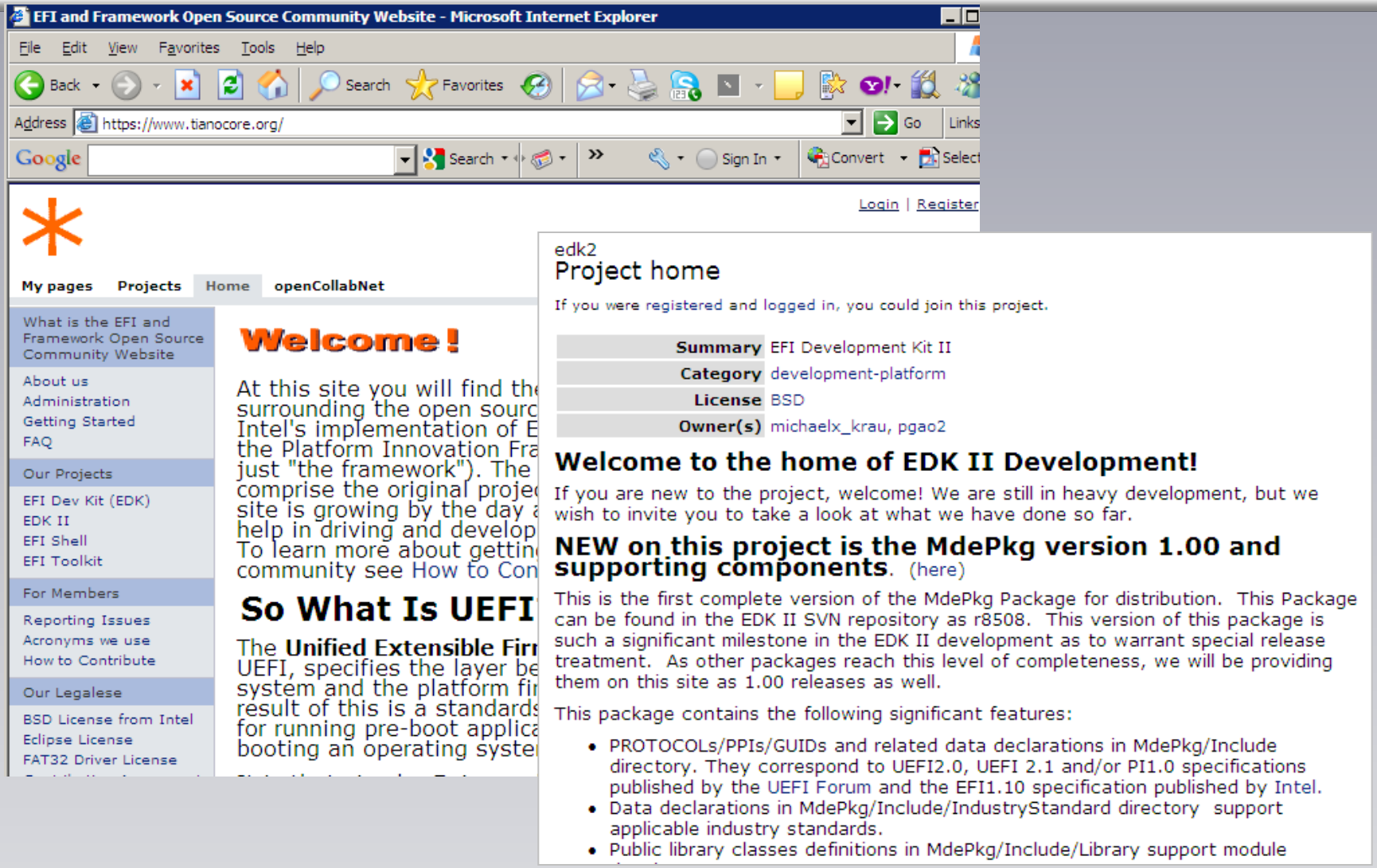
Overall Boot Timeline



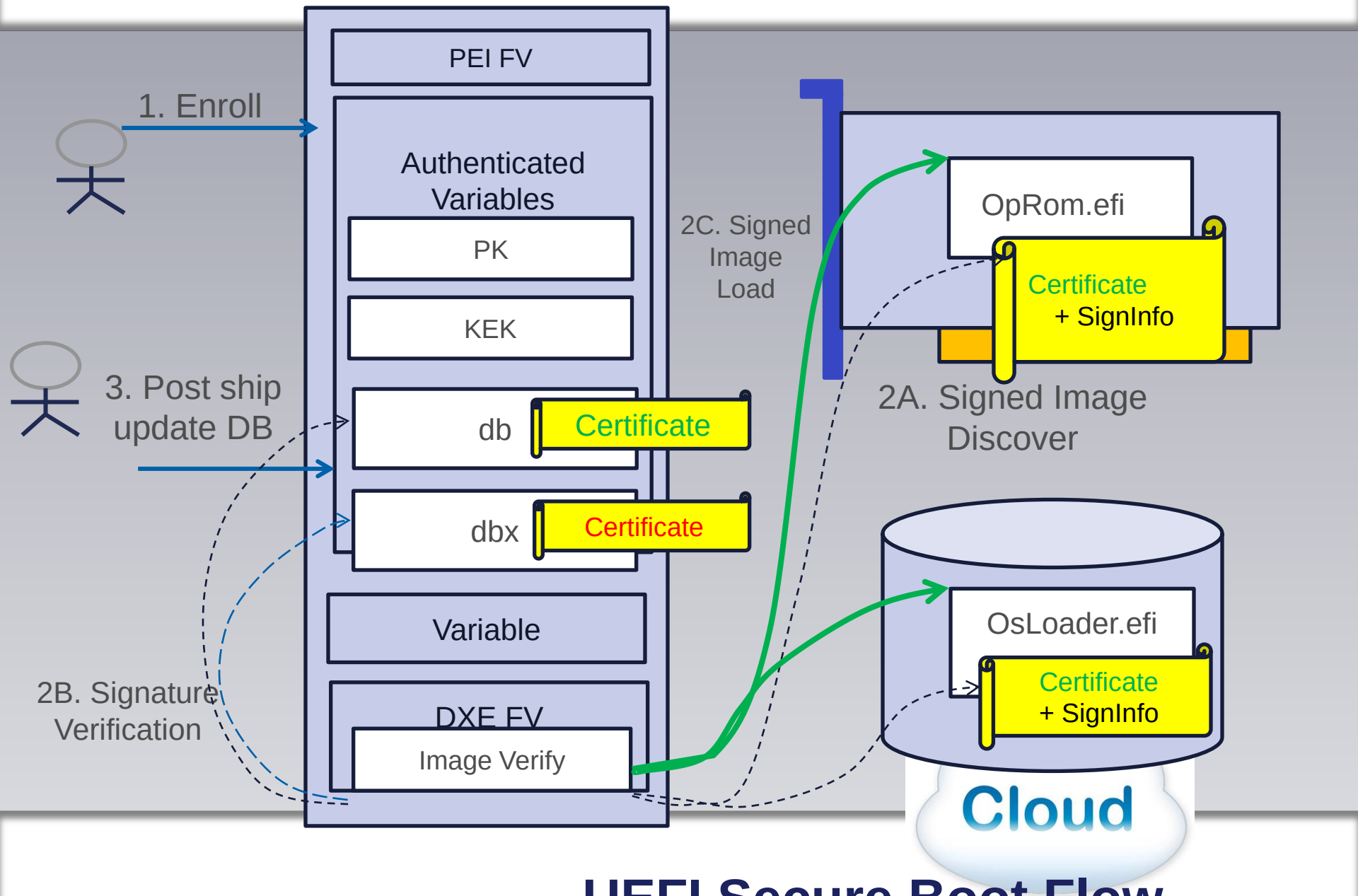
Note: Engaged in firmware/boot
Related WG's of Trusted Computing Group (TCG), IETF, DMTF

- **USST**
  - **U**SWG **S**ecurity **S**ub-**t**eam
  - Chaired by Vincent Zimmer (Intel)
  - Responsible for all security related material and the team has been responsible for the added security infrastructure in the UEFI
- **PSST**
  - PIWG Security Sub-team
  - Chaired by Vincent Zimmer (Intel)
  - Produce design guide(s) that define integrity protection business goals, provide a security model within which these goals are expressed as security requirements, and identify architectural and implementation issues that cause the requirements not to be met.
- **UNST**
  - UEFI Network Sub-team
  - Chaired by Vincent Zimmer (Intel)
  - Evolve network boot & network security infrastructure for UEFI Specification

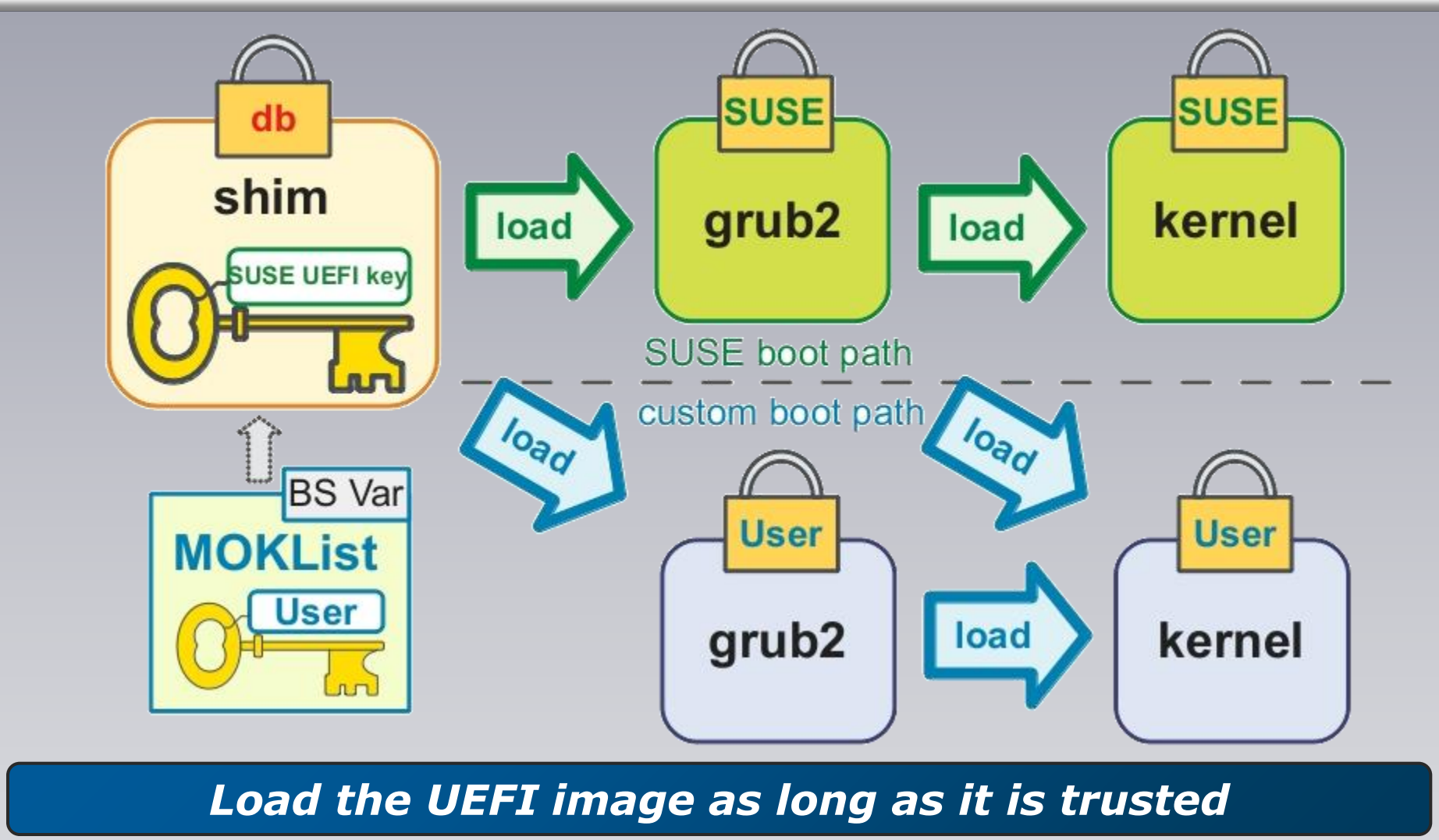
Security Working Groups in UEFI



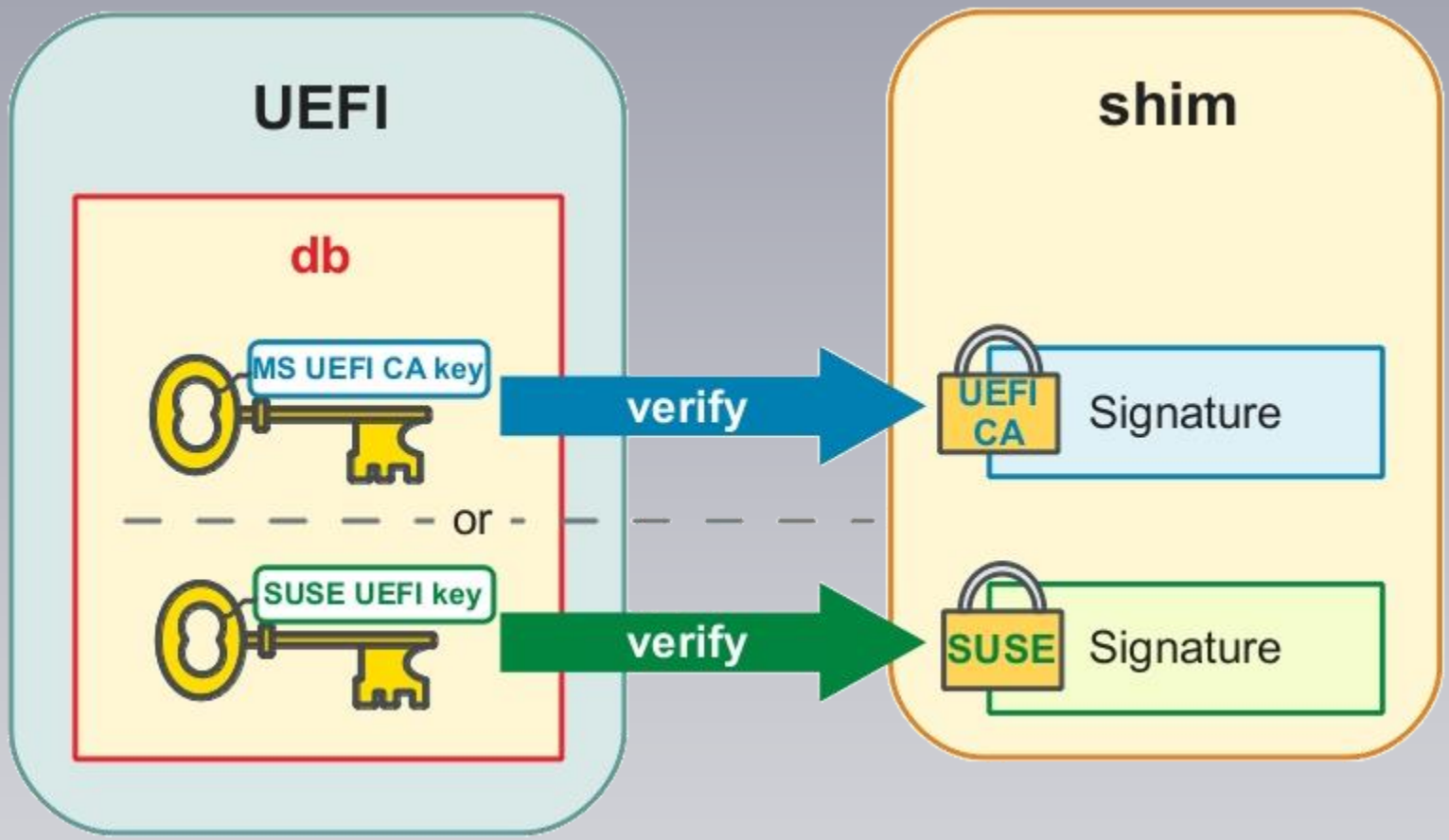
UDK2010 Available on Tianocore.org



UEFI Secure Boot Flow



Linux Update – Multiple OS Boot with MOK



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

Multi-Signature Support for Shim

RandomNumberGenerator

UEFI driver implementing the EFI_RNG_PROTOCOL from the UEFI2.4 specification

TCG

PEI Modules & DXE drivers implementing Trusted Computing Group measured boot EFI_TCG_PROTOCOL and EFI_TREE_PROTOCOL from the TCG and Microsoft MSDN websites, respectively

UserIdentification

DXE drivers that support multi-factor user authentication

Chapter 31 of the UEFI 2.4 specification

Library

DxeVerificationLib for “UEFI Secure Boot”, chapter 27.2 of the UEFI 2.4 specification + other support libs

VariableAuthenticated

SMM and runtime DXE authenticated variable driver, chapter 7 of the UEFI2.4 specification

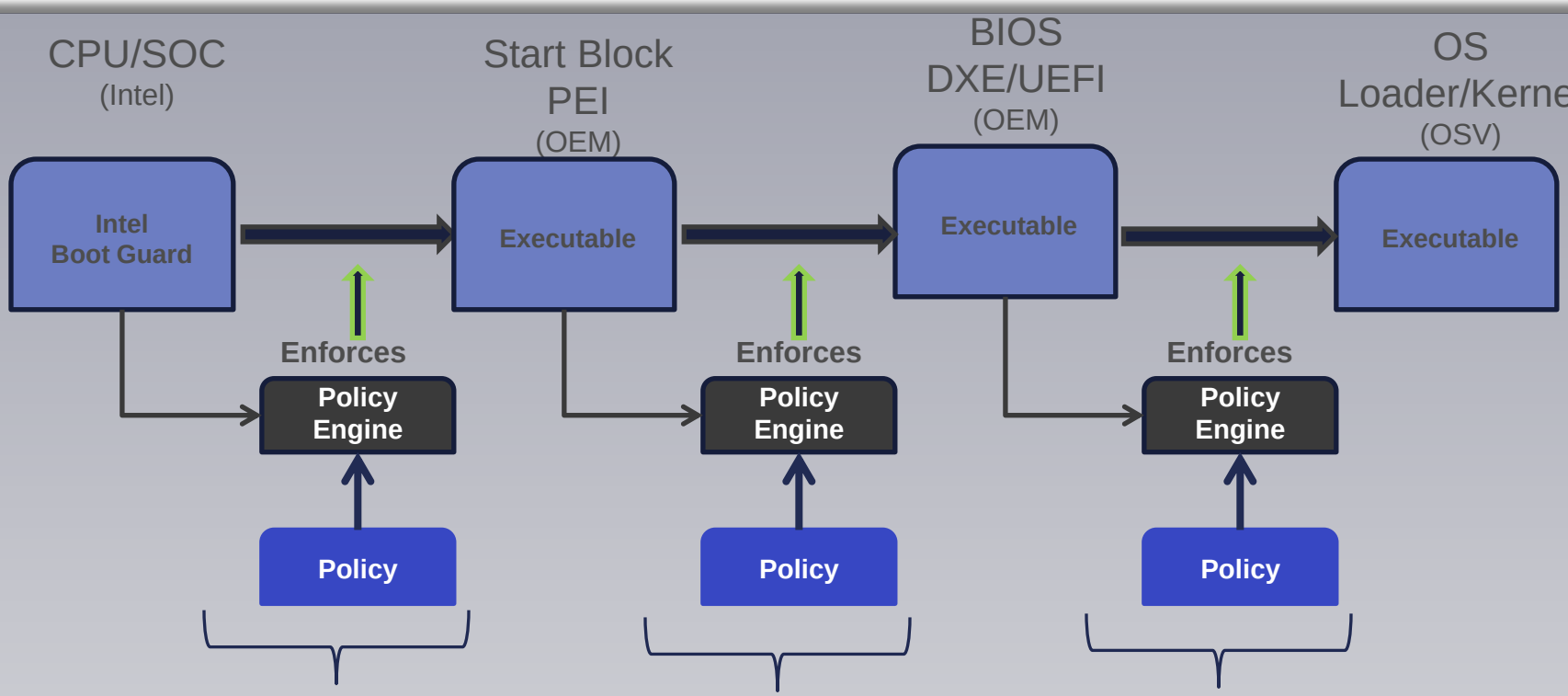
https://svn.code.sf.net/p/edk2/code/trunk/edk2/SecurityPkg

UDK2010 SecurityPkg

Analyze and Mark external Interfaces where input can be attacker controlled data, comment headers

```
/**
  Install child handles if the Handle supports GPT partition structure.

  Caution: This function may receive untrusted input.
  The GPT partition table is external input, so this routine
  will do basic validation for GPT partition table before install
  child handle for each GPT partition.

  @param[in]  This       Calling context.
  @param[in]  Handle     Parent Handle.
  @param[in]  DiskIo     Parent DiskIo interface.
  @param[in]  DiskIo2    Parent DiskIo2 interface.
  @param[in]  BlockIo    Parent BlockIo interface.
  @param[in]  BlockIo2   Parent BlockIo2 interface.
  @param[in]  DevicePath Parent Device Path.

  @retval EFI_SUCCESS           Valid GPT disk.
  @retval EFI_MEDIA_CHANGED     Media changed Detected.
  @retval other                 Not a valid GPT disk.

**/
EFI_STATUS
PartitionInstallGptChildHandl
```

UDK2010 example: http://edk2.svn.sourceforge.net/svnroot/edk2/trunk/edk2/MdeModulePkg/Universal/Disk/PartitionDxe/Gpt.c



Intel® Device Protection Technology with Boot Guard

http://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/4th-gen-core-family-mobile-brief.pdf

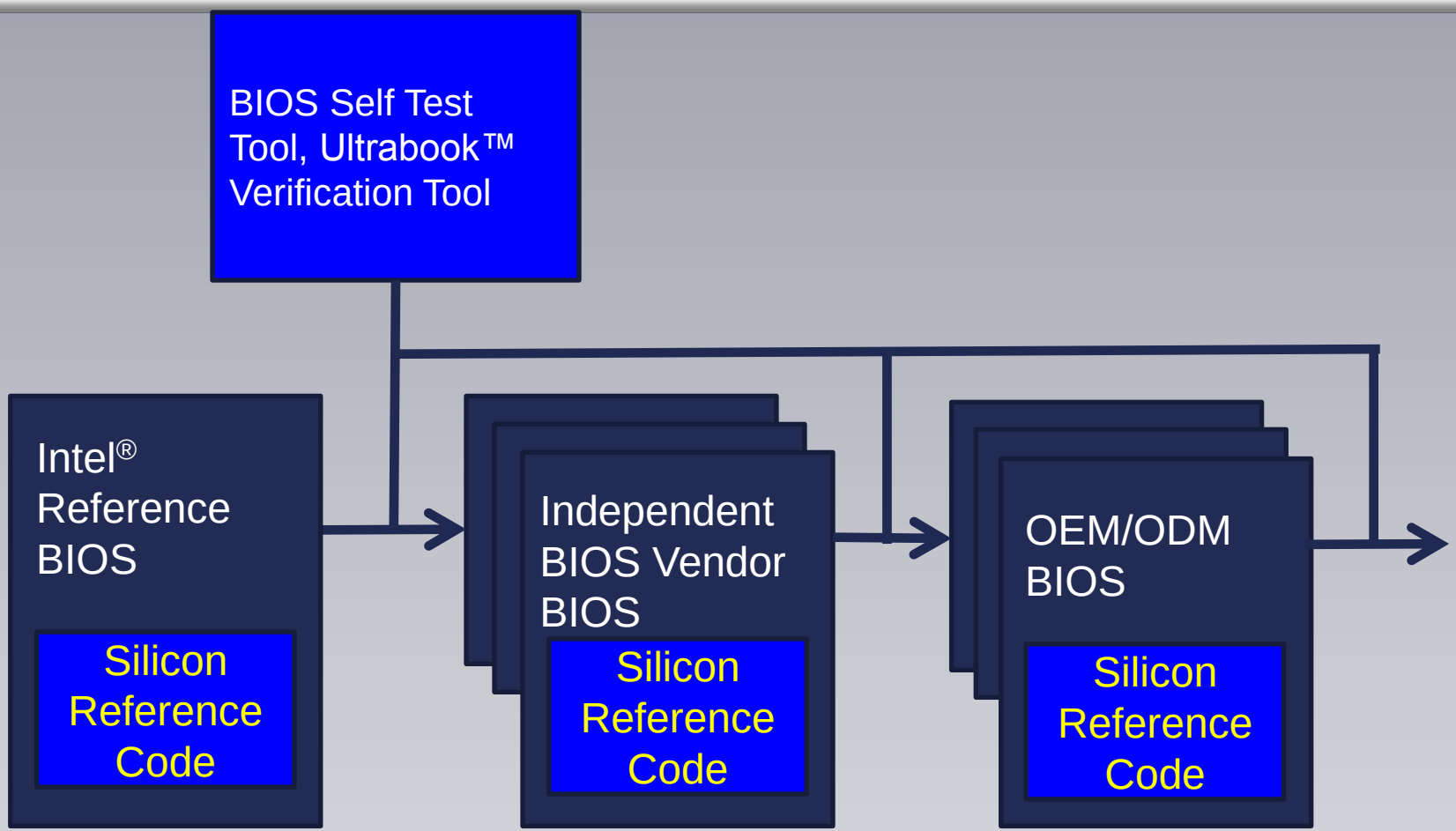
OEM PI Verification Using PI Signed Firmware Volumes

Vol 3, section 3.2.1.1 of PI 1.3 Specification

OEM UEFI 2.4 Secure Boot

Chapter 27.2 of The UEFI 2.4 Specification

From IDF 2013 - ULBS006, Ultrabook™ Design and Lifecycle Considerations for BIOS and Flash Security

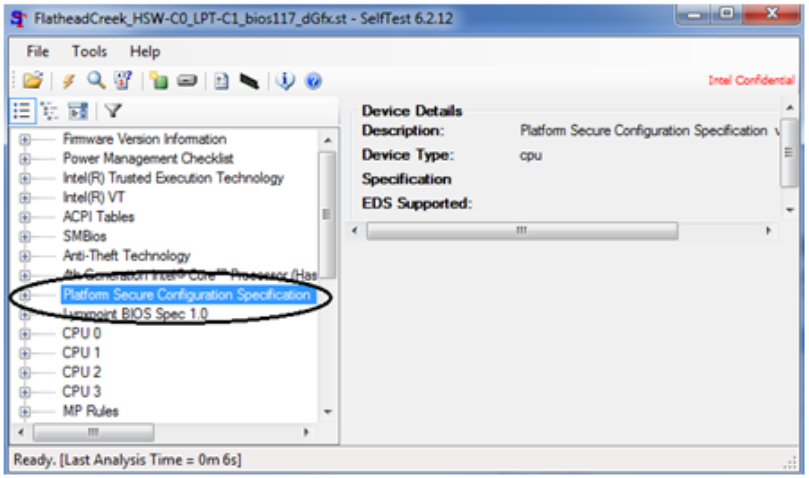


Suggested path to improve BIOS security assurance. Specific tests to verify SiRC are added in test tools.

Checking BIOS Security Compliance

SelfTest BIOS Validation

- Platform Secure Configuration Specification: Used to verify BIOS security



- Download SelfTest from CDI Doc# 434688
  http://www.intel.com/cd/edesign/library/asmo-na/eng/434688.htm

DOCUMENT TITLE	DOC ID	MODIFIED	EXPIRE DATE	TYPE	SIZE
Intel SelfTest 6.2.12	434688	31-Jan-2013	31-Jan-2014	exe	76.23 MB

SelfTest Checks BIOS Programming for Compliance

IDF13

28

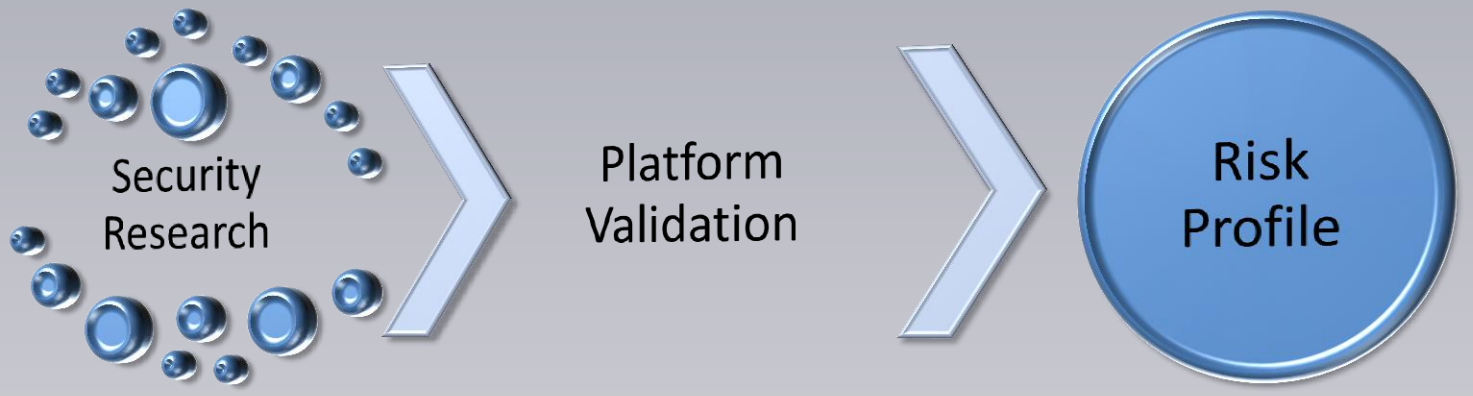
Platform Secure Configuration Spec/Tool

Coordinating the Mitigations

BIOS Coordination (John Loucaides)

- You never know what is vulnerable
  - Many OEMs, IBVs, versions, customizations…
- Increasing attention
  - Pierre Chifflier (PacSec 2013)
  - MITRE (BlackHat 2013, PacSec 2013)
  - Rootkit Detection Framework for UEFI (BackHat 2013)
  - #badBios by @dragosr (and here)

A New Strategy

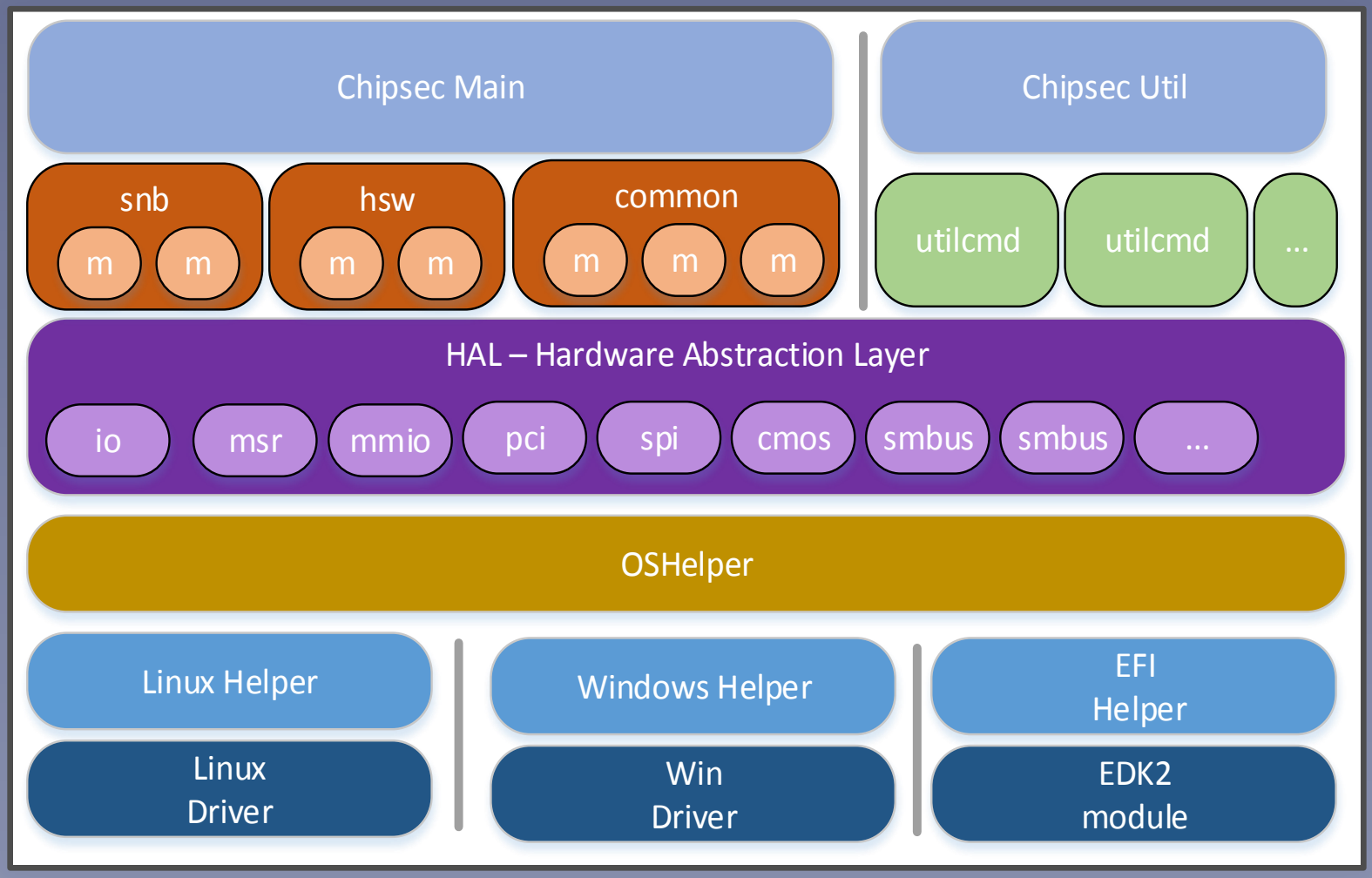


New Attacks

Test Modules
for
OEMs/IBVs

End-user Risk

Turn Research into Tests/Risks



One Test Works Everywhere!

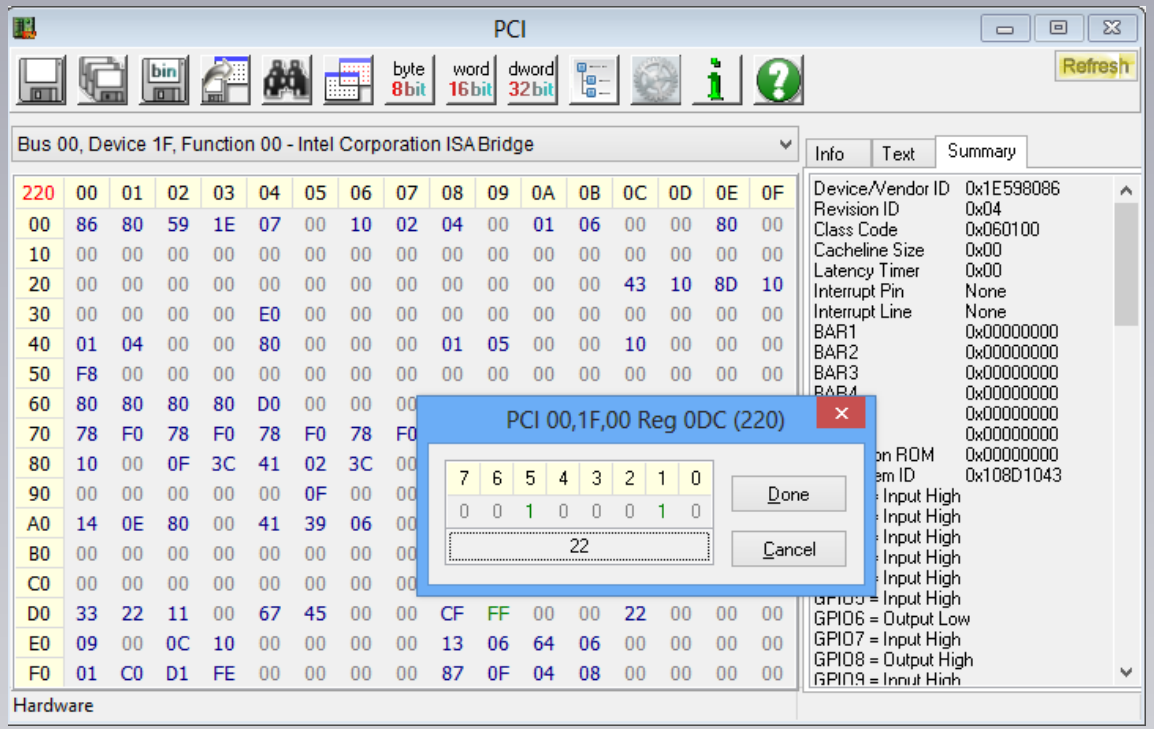
Check that UEFI FW/NVRAM are protected in ROM

Windows:

RWEverything ➔

Linux:

```
setpci -s 00:1F.0 DC.B
```



Alternative: BIOS Write Protection

Have HW Security Features Been Used Correctly?

```
chipsec_main.py --module COMMON.BIOS_WP
```

Is Platform HW Configured Securely?

```
chipsec_main.py --module COMMON.SPI_FD
```

Does FW Implement Required Security Features?

```
chipsec_main.py -m COMMON.SECUREBOOT.KEYS
chipsec_main.py -t COMMON.SECUREBOOT.VARIABLES
```

Are There Known HW/FW Vulnerabilities?

```
chipsec_main.py -m COMMON.BIOS_KBRD_BUFFER
```

Do all/Secure Boot tests pass on this system?

```
chipsec_main.py
chipsec_main.py -t SECUREBOOT
```

Hardware/Firmware Security Test Suite (NIST SP 800-147?)

Demo

Verifying SPI Flash is write-protected and Secure Boot EFI variables are authenticated

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)

HW/FW PoC/Exploit Development

```
chipsec_main.py -m exploits.secureboot.pk
```

Security Testing Tools/Fuzzers

```
chipsec_main.py -m tools.hot_fuzz -a 17
```

Manual Experimentation with Hardware Access

```
chipsec_util.py msr 0x79 0x0 0xDEADBEEF
chipsec_util.py mem 0x0 0xC0000 0x1000
chipsec_util.py pci 0 0 0 0x0 dword
```

Security Validation Framework

SPI Flash Programmer

```
chipsec_util.py spi info|dump|read|write|erase
```

BIOS/FW Forensics

```
chipsec_util.py decode rom.bin
```

Parse SPI Regions, Flash Descriptor, EFI FW Volumes (FV)

Extract EFI binaries from FV, EFI variables from NVRAM

Secure Boot Forensics

```
chipsec_util.py uefi keys db.bin
```

Extract x509 certs, Pub Keys, SHAx from db/dbx/PK/KEK

Demo

Forensics: Parsing SPI Flash image, UEFI Firmware and Secure Boot variables

This was a brief introduction to

CHIPSEC

Platform Security Assessment Framework

Thank You!

Yuriy Bulygin
Chief Threat Research Architect – yuriy.bulygin@intel.com

Vincent Zimmer
Principal Engineer – vincent.zimmer@intel.com

John Loucaides
Product Security Response – john.loucaides@intel.com