tianocore

UEFI & EDK II TRAINING

Introduction to Platform Firmware Security w/ UEFI

tianocore.org

tianocore

LESSON OBJECTIVE

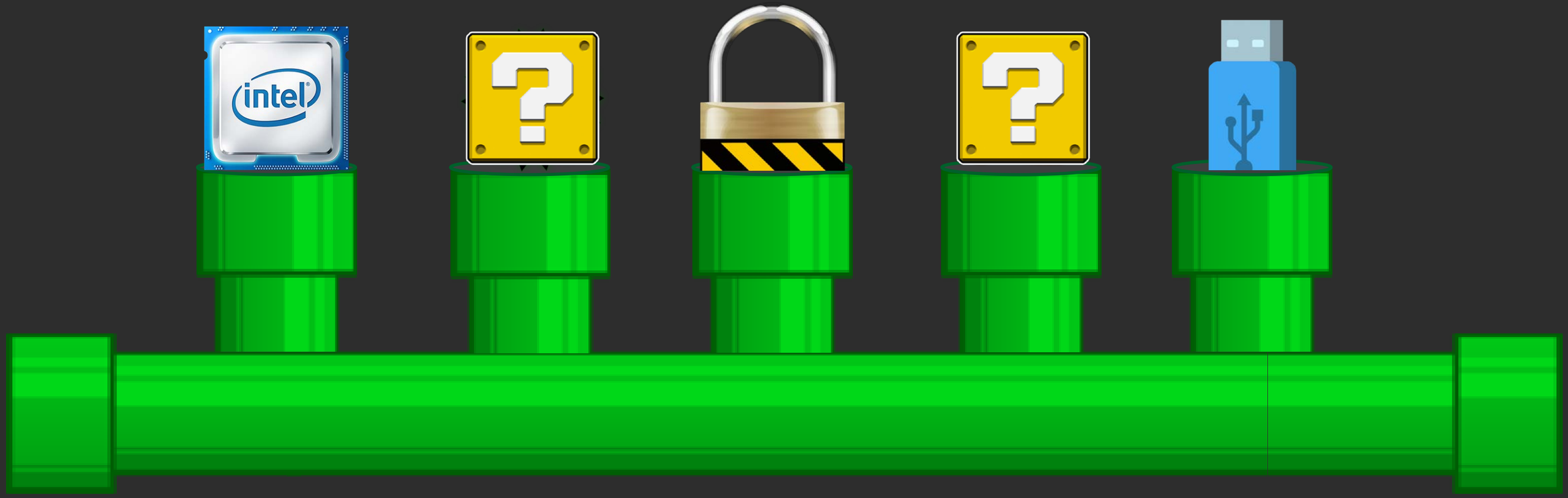
- Why is platform firmware Security important
- UEFI boot flow with the threat model
- Security technologies overview
- Tools and resources on how to test firmware for security

tianocore

Why is Platform Firmware Security Important ?

tianocore

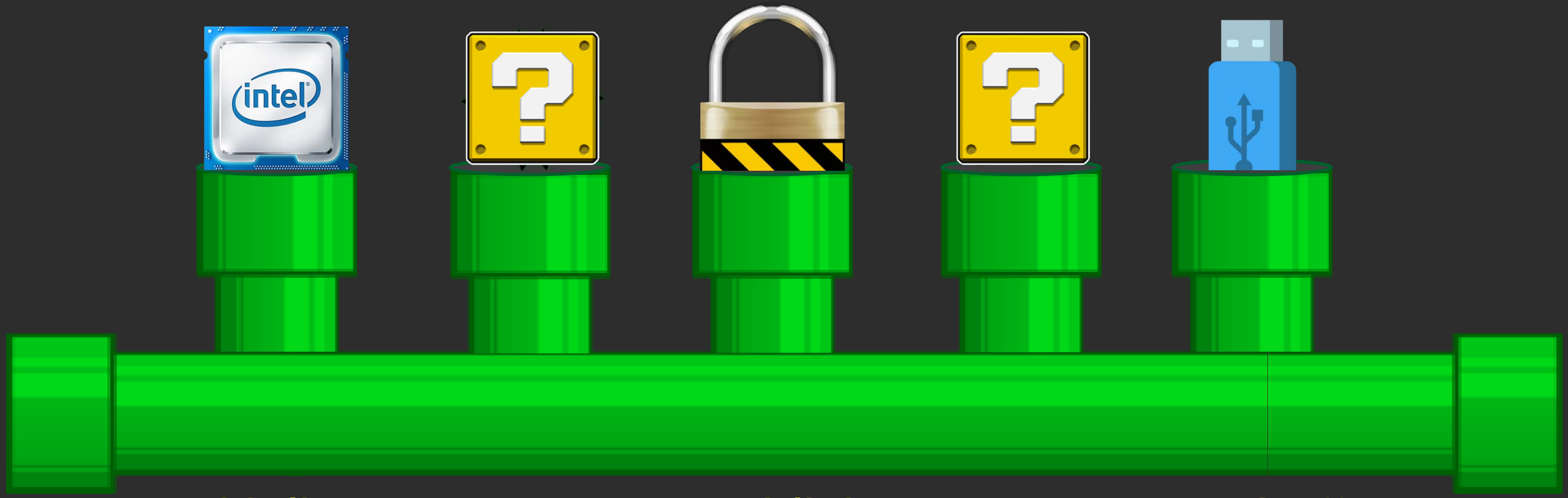
Why is Platform Firmware Security Important ?



tianocore

Why is Platform Firmware Security Important ?

Firmware is the Infrastructure



Initialize hardware

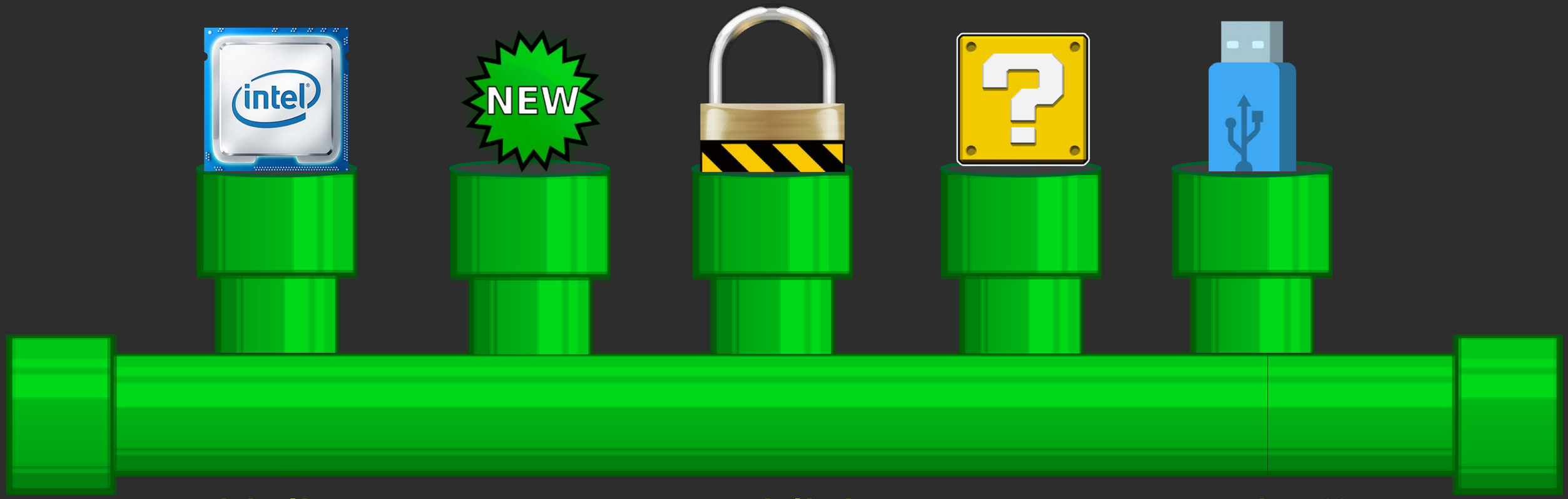
Establish root-of-trust

Hand-off to the OS

tianocore

Why is Platform Firmware Security Important ?

Firmware is the Infrastructure



Initialize hardware

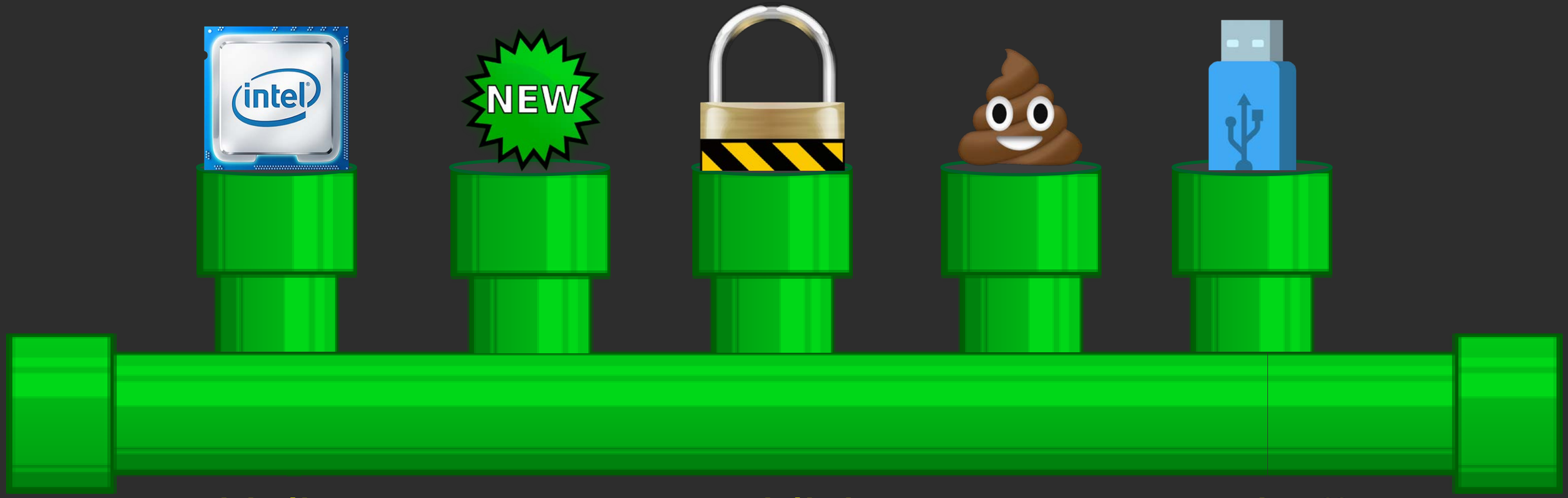
Establish root-of-trust

Hand-off to the OS

tianocore

Why is Platform Firmware Security Important ?

Firmware is the Infrastructure



Initialize hardware

Establish root-of-trust

Hand-off to the OS

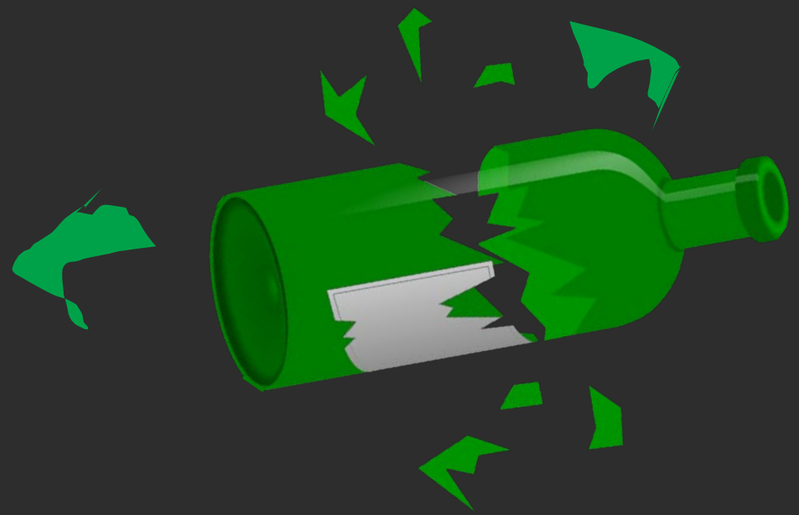
tianocore

Why??? Security

Without

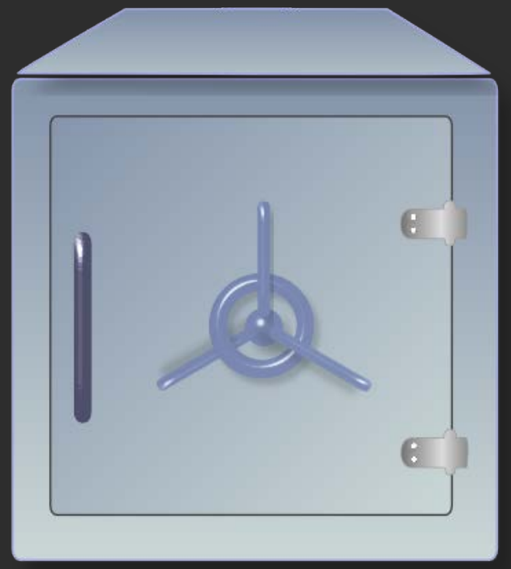
Possible corrupted or destroyed data

- BootKit virus – MBR Rootkits
- Network boot attacks e.g. PXESPOILT
- Code Injection Attacks



With

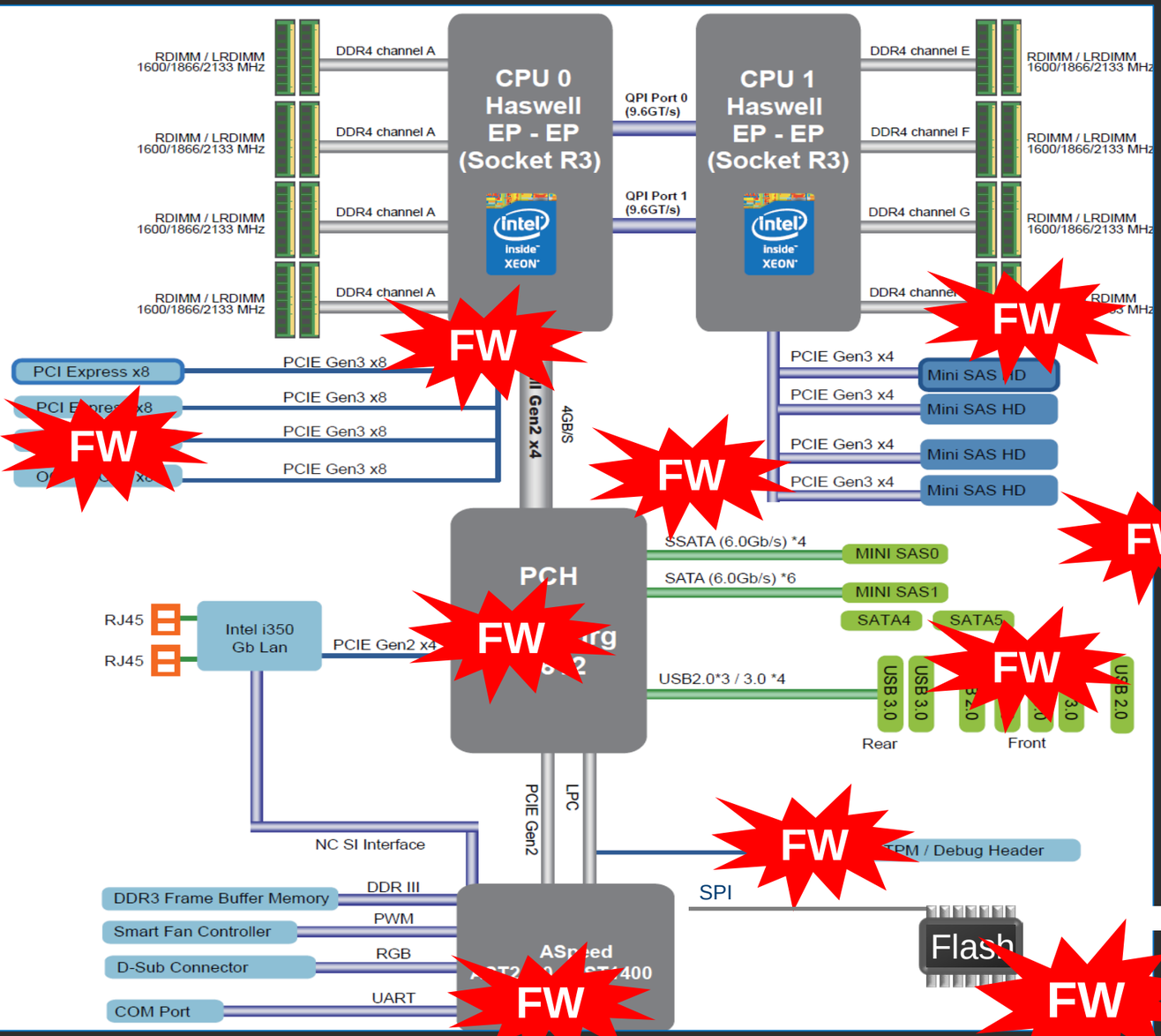
- **Data integrity**
- **Trusted boot to OS**
- **Trusted drivers**
- **Trusted Applications**



Confidentiality
Integrity
Availability

tianocore

Firmware is Everywhere



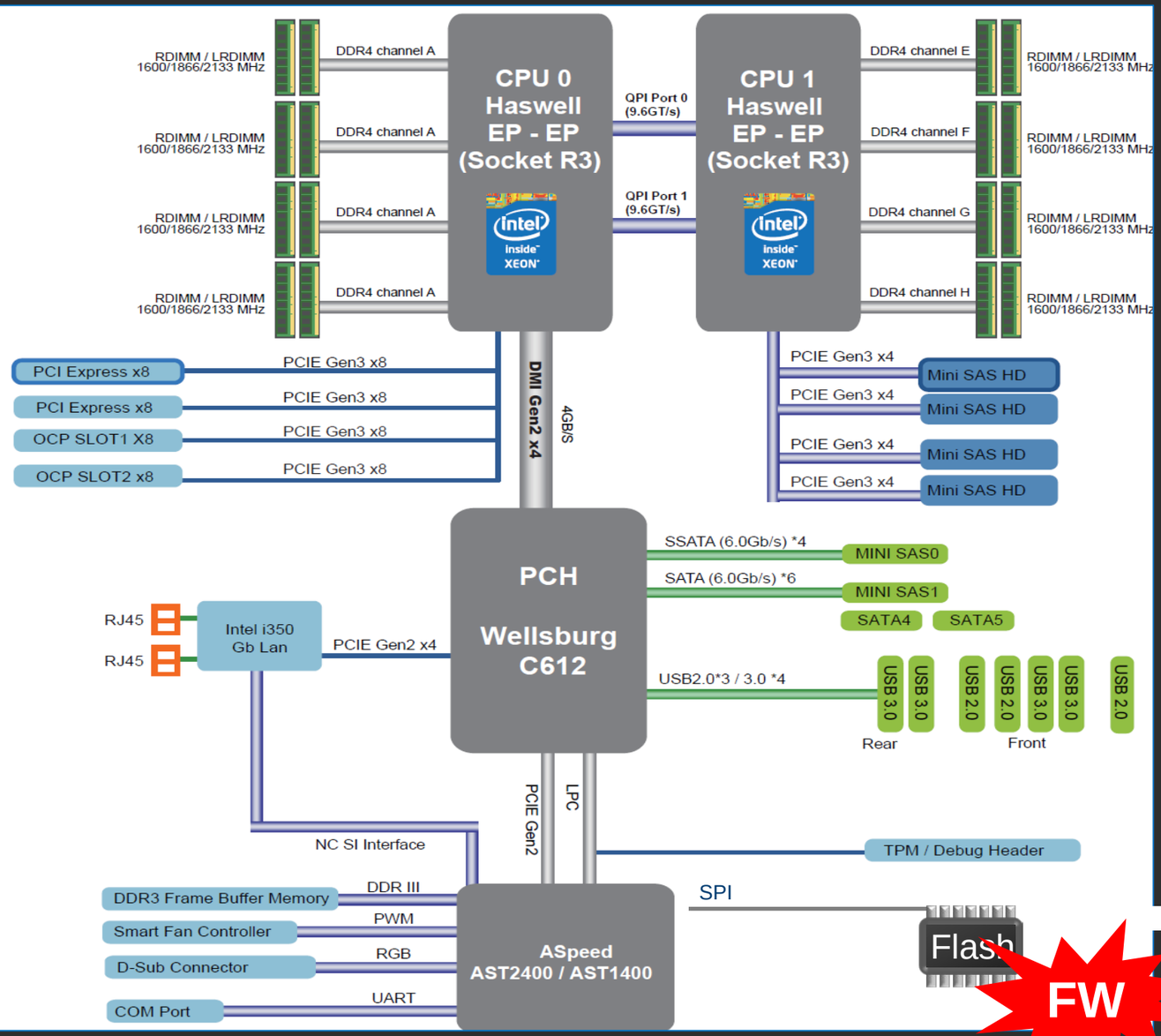
- GBe NIC, WiFi, Bluetooth, WiGig
- Baseband (3G, LTE) Modems
- Sensor Hubs
- NFC, GPS Controllers
- HDD/SSD
- Keyboard and Embedded Controllers
- Battery Gauge
- Baseboard Management Controllers (BMC)
- Graphics/Video
- USB Thumb Drives, keyboards/mice
- Chargers, adapters
- TPM, security coprocessors
- Routers, network appliances

Main system firmware (BIOS, UEFI firmware, coreboot)

Image source

tianocore

Firmware is Everywhere

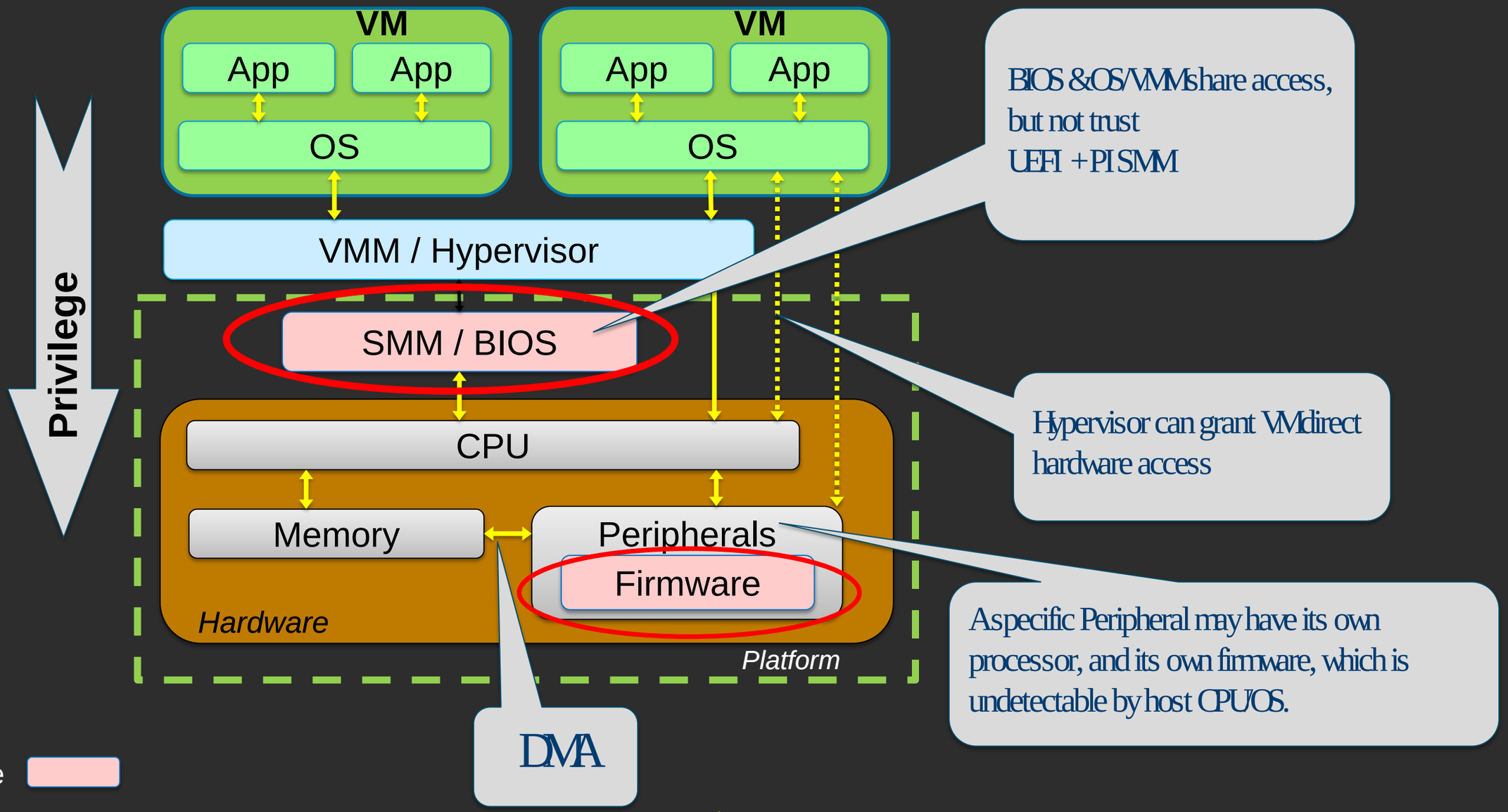


Main system firmware (BIOS, UEFI firmware, coreboot)

Image source

tianocore

Where is UEFI firmware?



tianocore

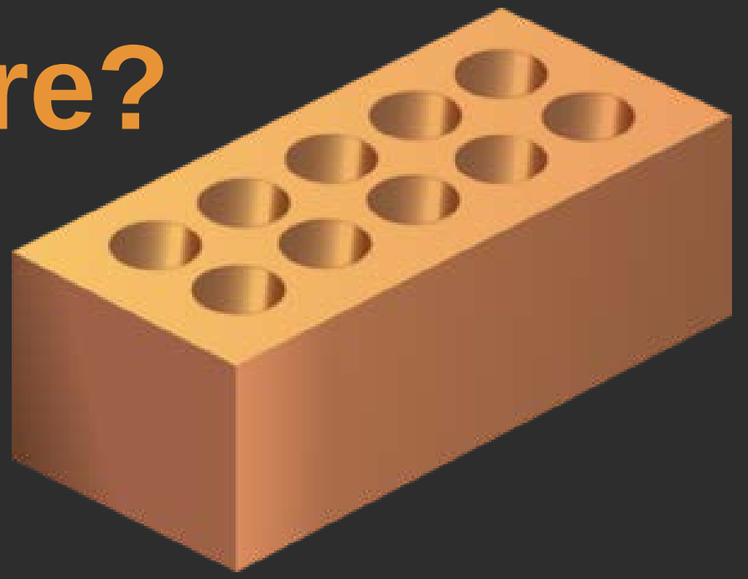
In-the-wild Firmware Attacks

- Legacy Bootkits (TDL4, Gapz…)
- Mebromi BIOS rootkit
- Stuxnet
- EQUATION Group HDD firmware malware
- Hacking Team [UEFI rootkit]
- Petya MBR Ransomware
- Legitimate BIOS “backdoors”: SuperFish, Computrace

Source: Wild Tiger CC IFAW

tianocore

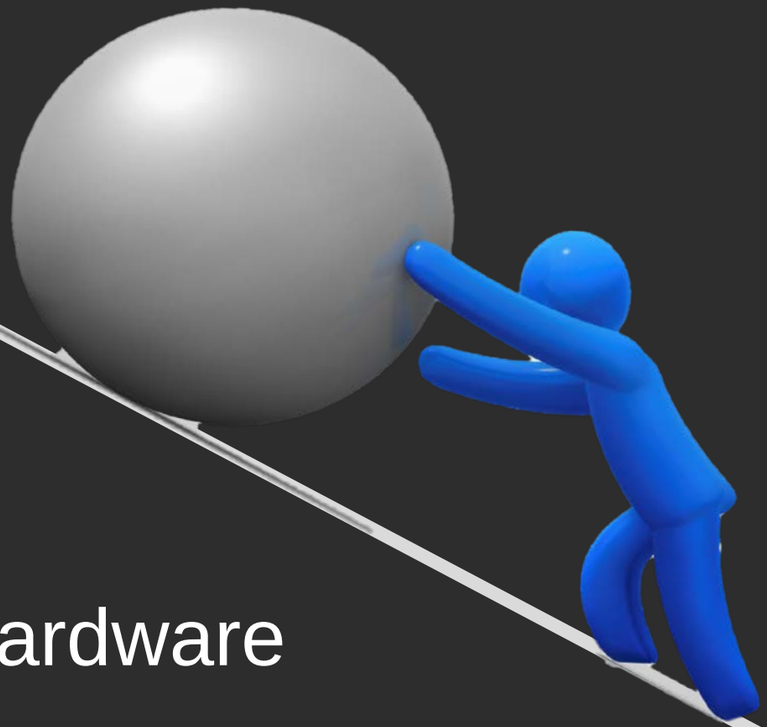
Why Attack Firmware?



- Extreme persistence
- Unfettered access to hardware
- Stealth
- OS independence
- Bypass software (OS or VMM) based security
- Making the system unbootable (bricking)

Firmware Attack Methods

Extreme Persistence



- System firmware rootkit (in SMM or BIOS/UEFI)
- Replaces OS boot loader every boot
- Which patches OS kernel
- Firmware rootkit is protected by the hardware write protections
- Only way to fully remove the infection is to physically re-flash the flash “ROM” chip

STEALTH

Is Security Software Real Protection?

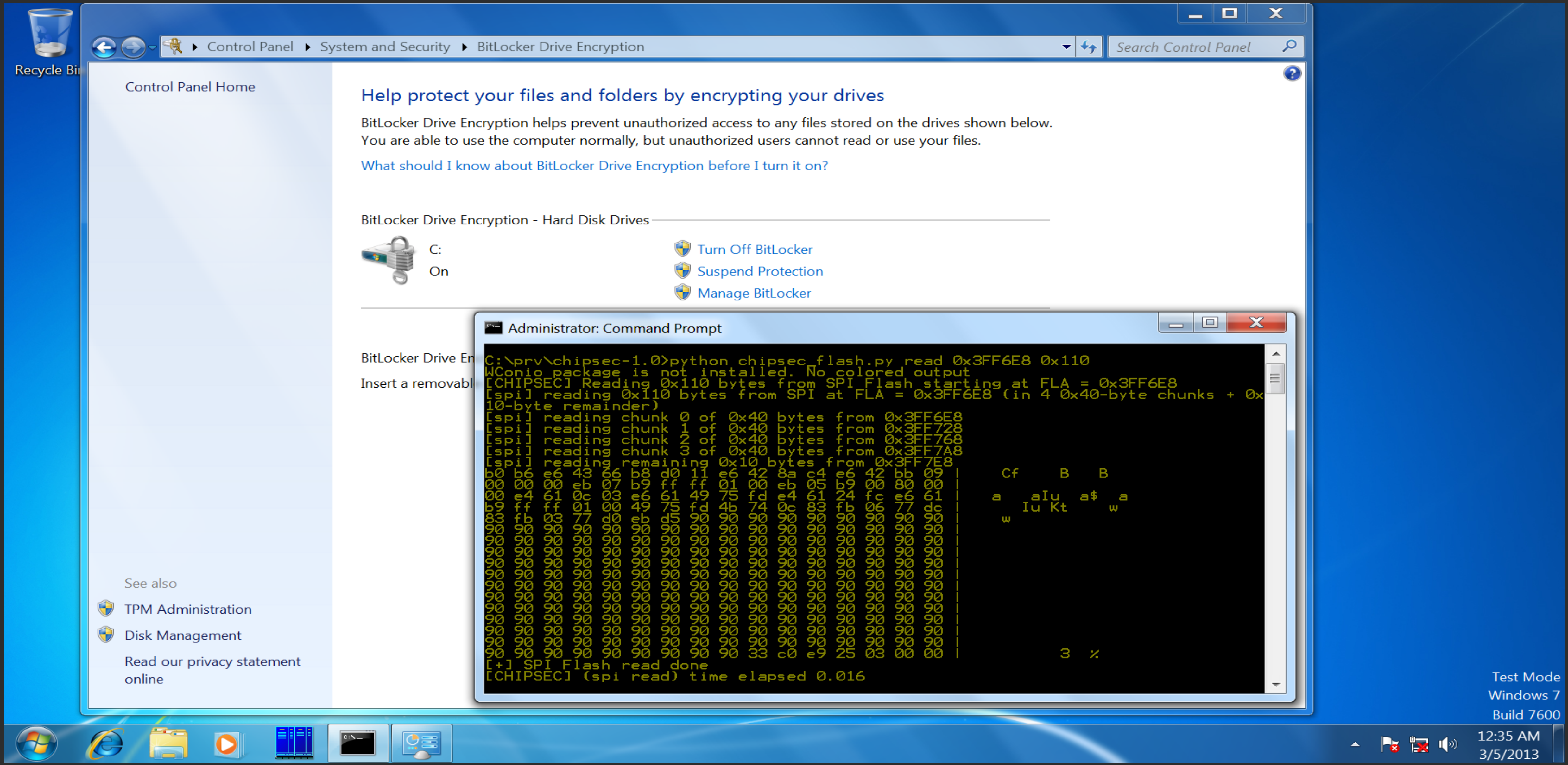
- Monitor of ~~All~~ firmware
- Reliably tell if infected False | Positive
- Devices use *obscure* hardware interfaces to their firmware
- Which Tool for which firmware infection?
  - Rootkit in firmware of SSD, NIC, EC, BMC, modem, USB thumb-drive, battery gauge, charger

Bypass Software Security – Full Disk Encryption

```
OS      : Windows 7 6.1.7600 AMD64
Chipset:
  VID:        8086
  DID:        0100
  Name:       Sandy Bridge (SNB)
  Long Name:  Sandy Bridge CPU / Cougar Point PCH

[+] loaded exploits.bios.bitlocker
[+] imported chipsec.modules.exploits.bios.bitlocker
[*] Reading 0x1000-byte block from SPI at FLA = 0x3FF000..
[+] Done reading block from SPI flash
[*] Filling with 0x109 bytes of NOP slide..
[*] Injecting 0x47 bytes of payload at offset 0x6E8 in the block..
[+] Checking protection of UEFI BIOS region in SPI flash..
[spi] UEFI BIOS write protection enabled but not locked. Disabling..
[!] UEFI BIOS write protection is disabled
[*] Erasing original 0x1000-byte SPI block at FLA = 0x3FF000..
[+] Done erasing block of SPI flash
[*] Writing SPI block with payload to FLA = 0x3FF000 (payload at 0x3FF6E8)..
[+] Done writing modified UEFI BIOS boot block in SPI flash
[!] Reboot to verify that BitLocker decrypts Windows volume
```

Source: Evil Maid Just Got Angrier: Why Full-Disk Encryption with TPM is not Secure on Many Systems

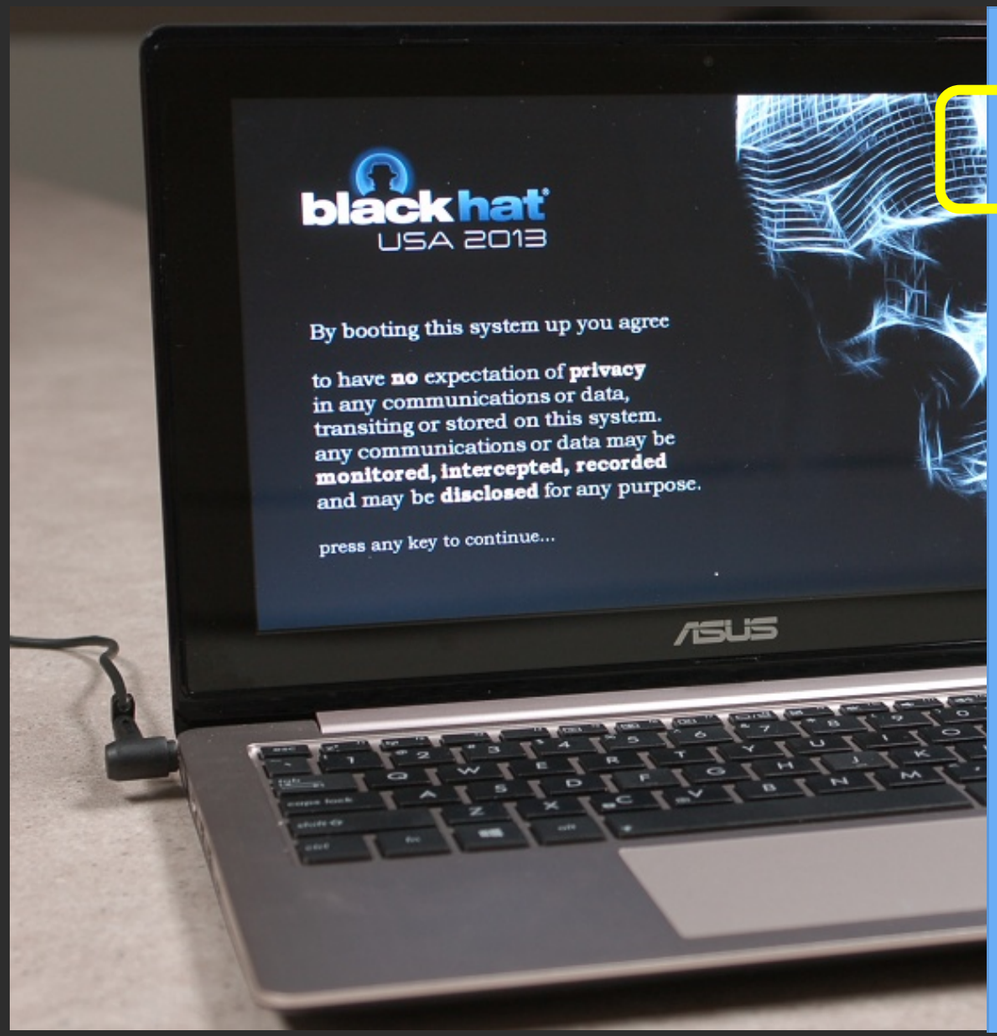
Bypass Software Security – Full Disk Encryption



Source: Evil Maid Just Got Angrier: Why Full-Disk Encryption with TPM is not Secure on Many Systems

tianocore

Bypass Software Security – UEFI Secure Boot



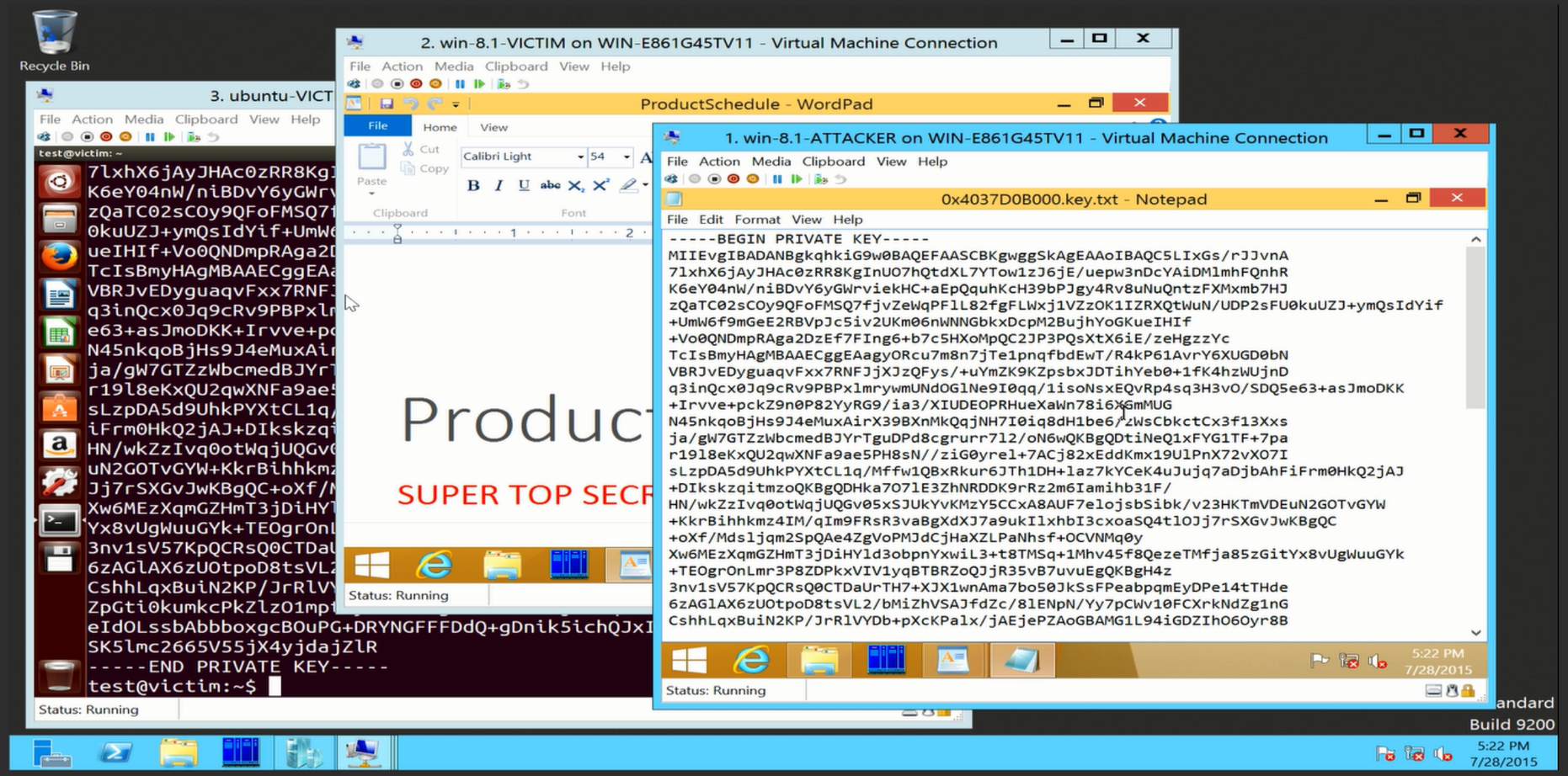
Administrator: Windows PowerShell

```
PS C:\Windows\system32>
PS C:\Windows\system32> Get-SecureBootUEFI -Name SecureBoot

Name                           Bytes                          Attributes
----                           -----                          ----------
SecureBoot                     {0}                            BOOTSERVICE ACCESS...

PS C:\Windows\system32> Get-SecureBootUEFI -Name SetupMode

Name                           Bytes                          Attributes
----                           -----                          ----------
SetupMode                      {1}                            BOOTSERVICE ACCESS...

PS C:\Windows\system32> Get-SecureBootUEFI -Name PK
Get-SecureBootUEFI : Variable is currently undefined: 0xC0000100
At line:1 char:1
+ Get-SecureBootUEFI -Name PK
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : ResourceUnavailable: (Microsoft.Secur...BootUefi
   Command:GetSecureBootUefiCommand) [Get-SecureBootUEFI], StatusException
    + FullyQualifiedErrorId : GetFWVarFailed,Microsoft.SecureBoot.Commands.Get
   SecureBootUefiCommand

PS C:\Windows\system32>
```

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

tianocore

Bypass Software Security – Virtual Machine Manager



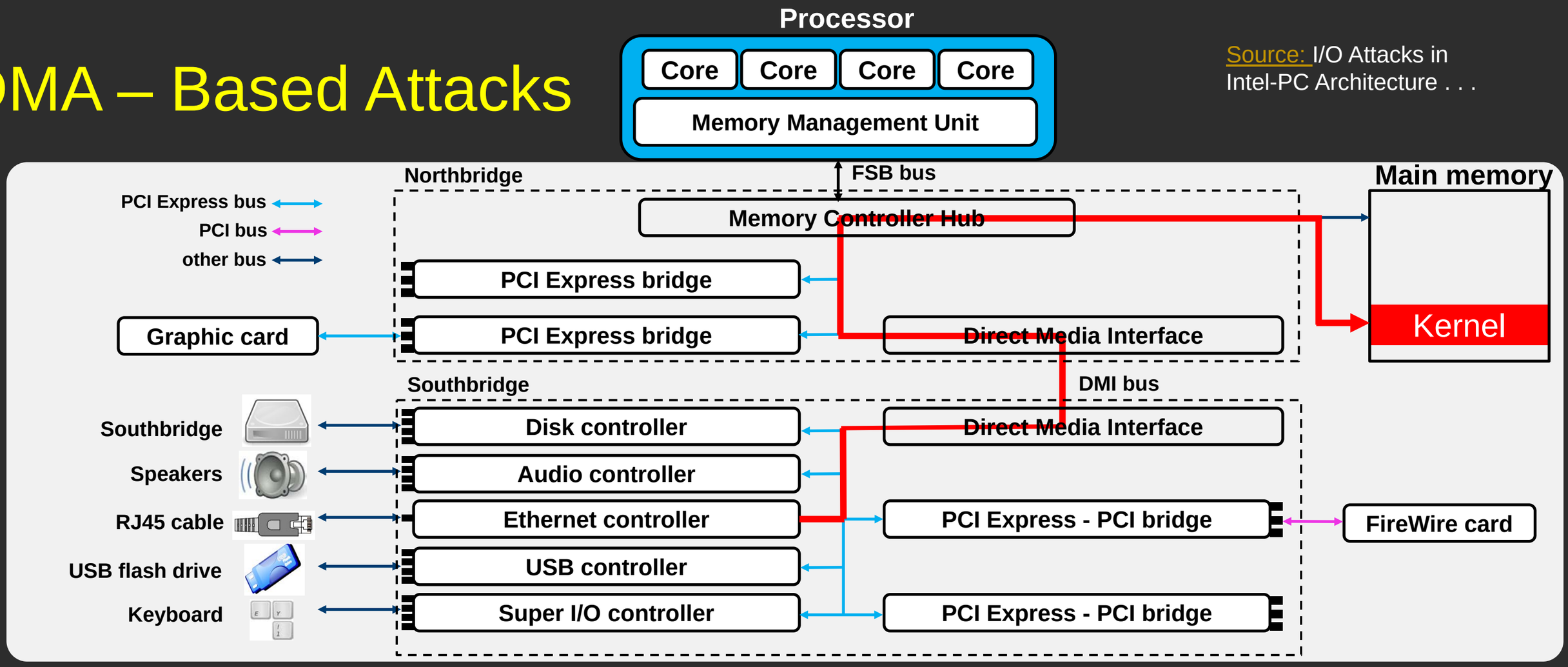
Attacker VM exposes secrets of other VMs through a backdoor opened by the rootkit

Source: Attacking Hypervisors via Firmware and Hardware

Unfettered Access to Hardware - DRAM

DMA – Based Attacks

Source: I/O Attacks in Intel-PC Architecture . . .

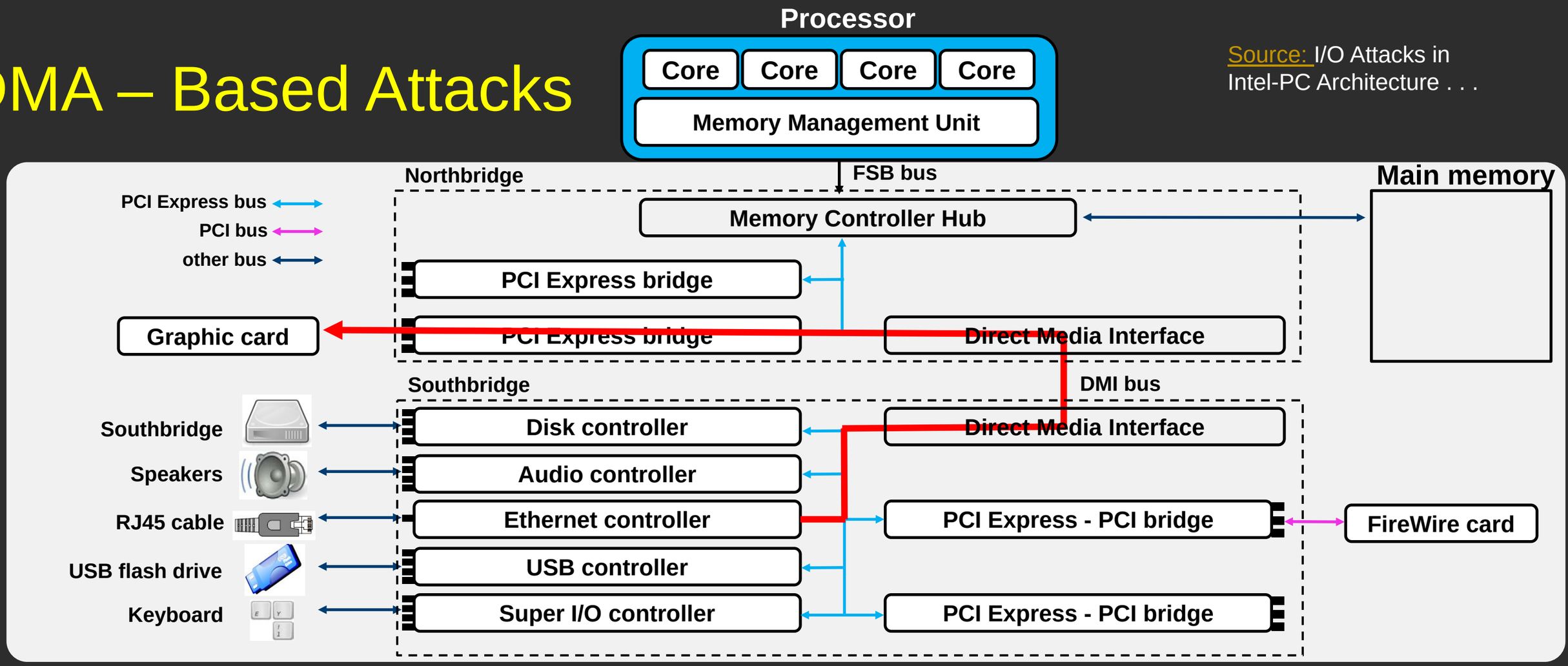


DMA attacks aiming at the main memory

Unfettered Access to Hardware - DRAM

DMA – Based Attacks

Source: I/O Attacks in Intel-PC Architecture . . .



DMA attacks aiming at I/O controllers' internal memory

Unfettered Access to Hardware – HDD/SSD

Access to all data stored on HDD/SSD

Even when data is stored on self-encrypting drives (SED)

Cyber espionage groups utilize advanced form of malware that is **very hard to remove** once it's infected your PC.

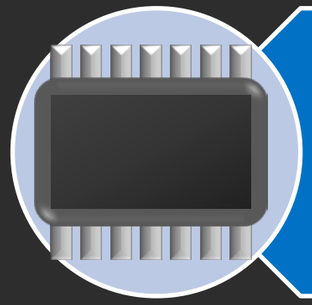
Malware reprograms the hard drive's firmware, creating hidden sectors

Destroying your hard drive is the only way to stop this super advanced malware

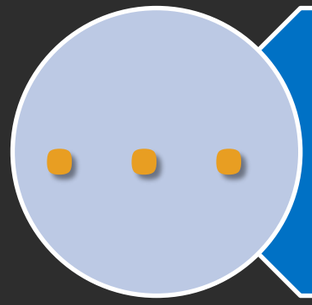
Source: PC World – Destroying your hard drive is the only way to stop this super advanced malware

Unfettered Access to Hardware – Etc.

NIC, WiFi, baseband modem firmware rootkits have direct access to network communications



EC or BMC firmware rootkit has access to platform management functions (power, thermal, NIC, keystrokes)



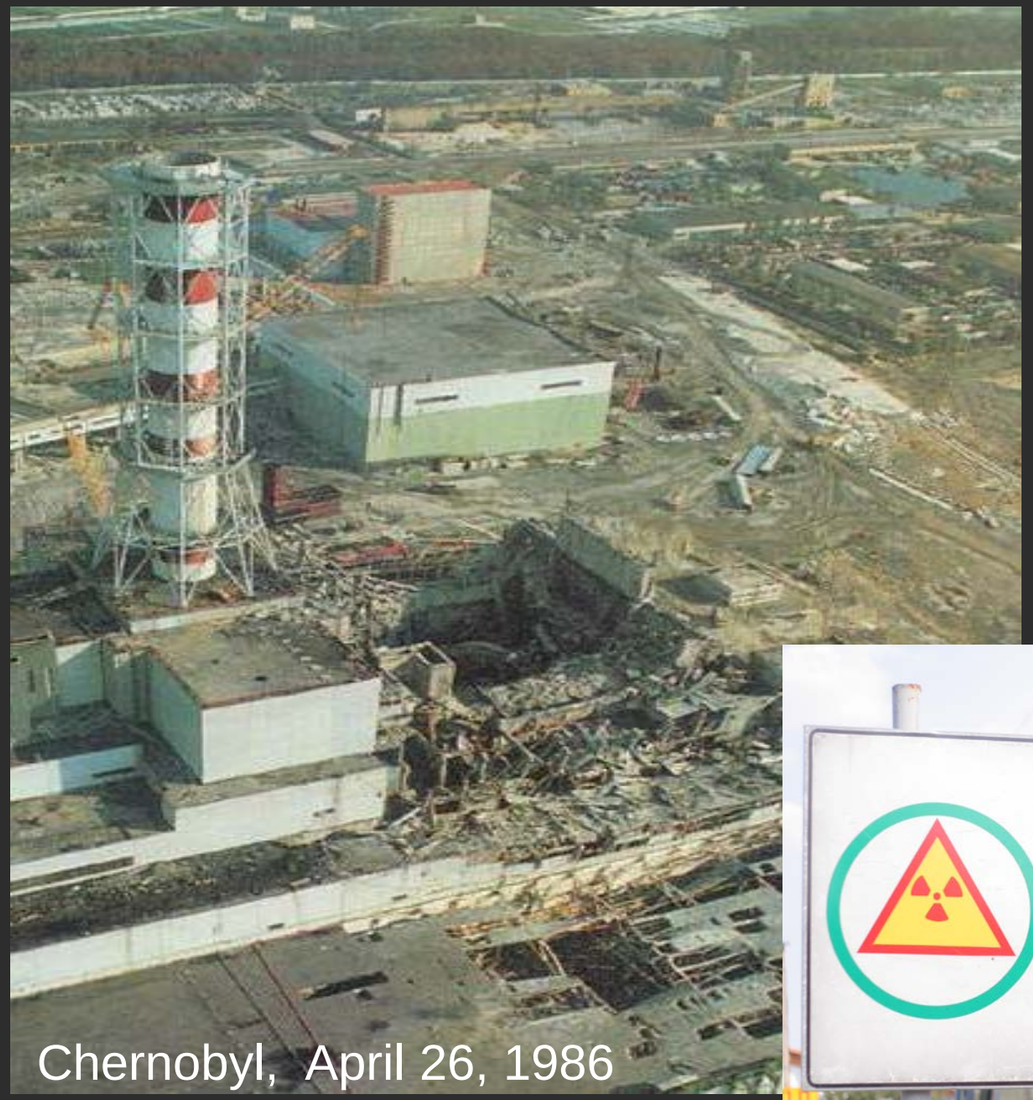
Etcetera

EC – embedded controllers
BMC – Baseboard Management Component

tianocore

Making System Unbootable (Bricking)

- Corrupt firmware or
- Corrupt critical configuration
- Stored in flash “ROM” memory
- Of a device which is critical for system to boot or operate



Chernobyl, April 26, 1986

CIH virus, was first discovered in 1998

Image Source: Crooks and Liars and Chernobyl image sign

tianocore

Pre-Boot Threats

- Firmware Malware
- UEFI Rootkits
- Bootkits
- SMM Rootkits
- Device FW Malware
- ACPI Rootkits
- Option ROM Malware
- Evil Maid
- HVM[1] Rootkits (Blue Pill)
- HW Trojans

- Summary Why Attack Firmware

[1] Hypervisor Virtual Machine

Summary

Platform Firmware Security – Why is it important?

Why is platform firmware Security important

Prevent low level attacks that could “brick” the system

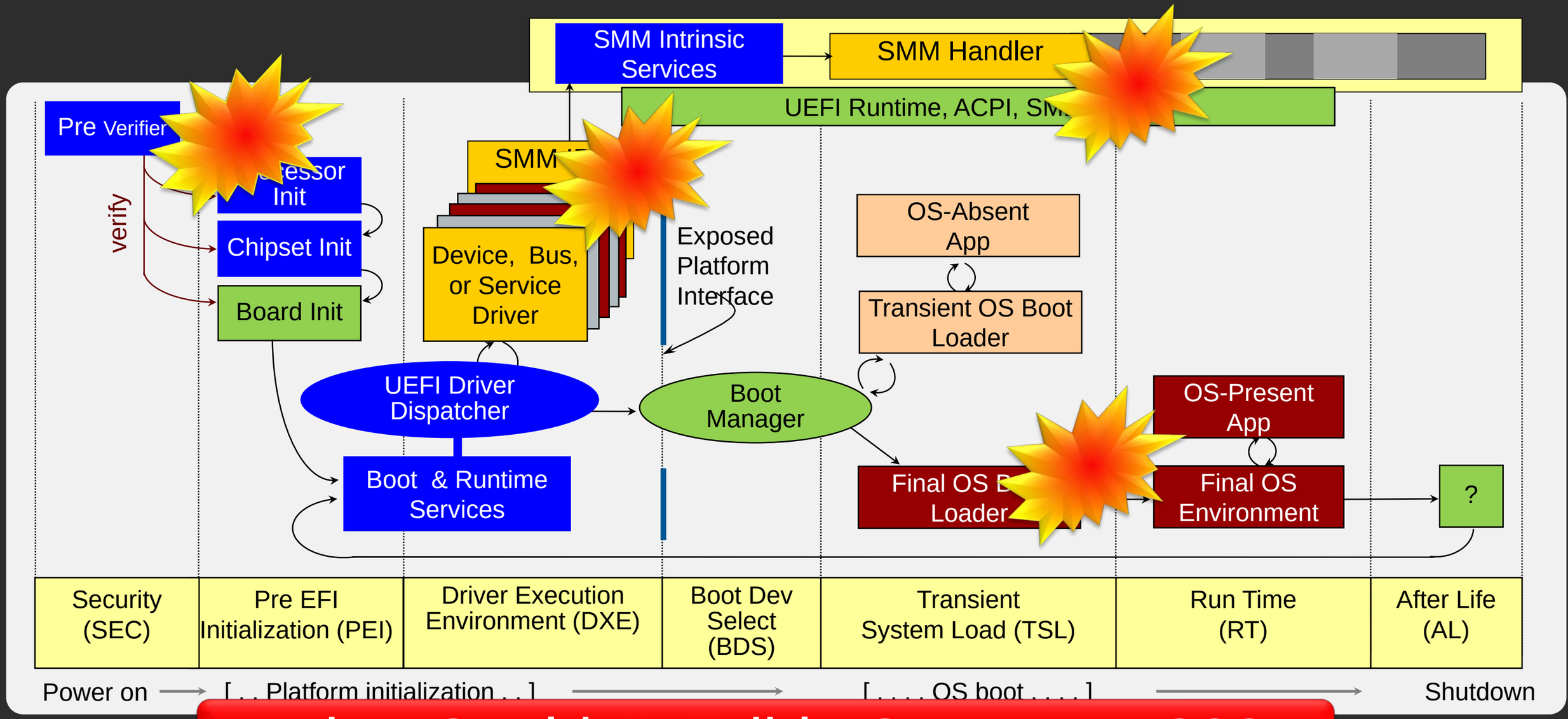
tianocore

UEFI BOOT FLOW

UEFI boot flow with the threat model

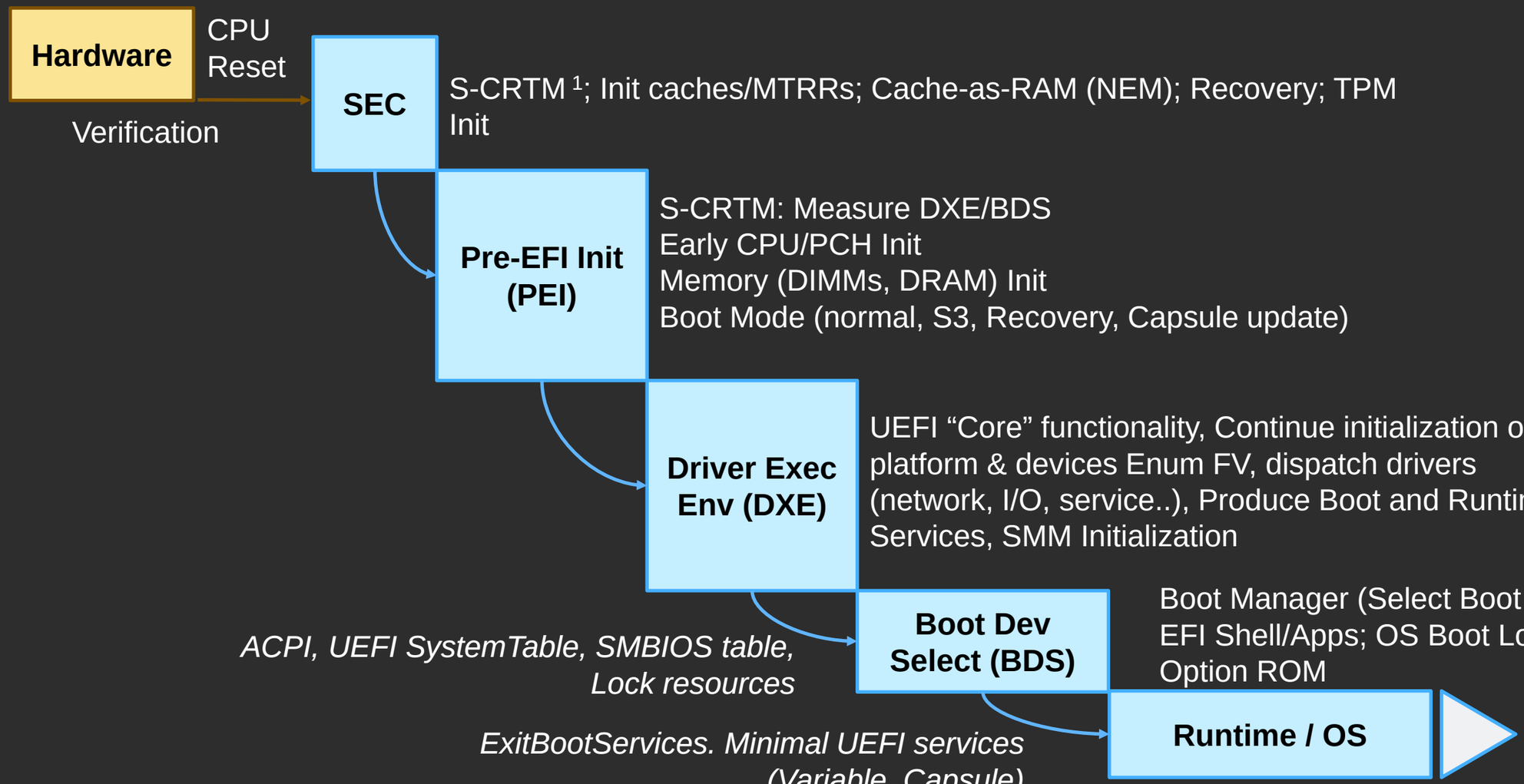
tianocore

UEFI Boot Execution Flow



What Could Possibly Go Wrong ???

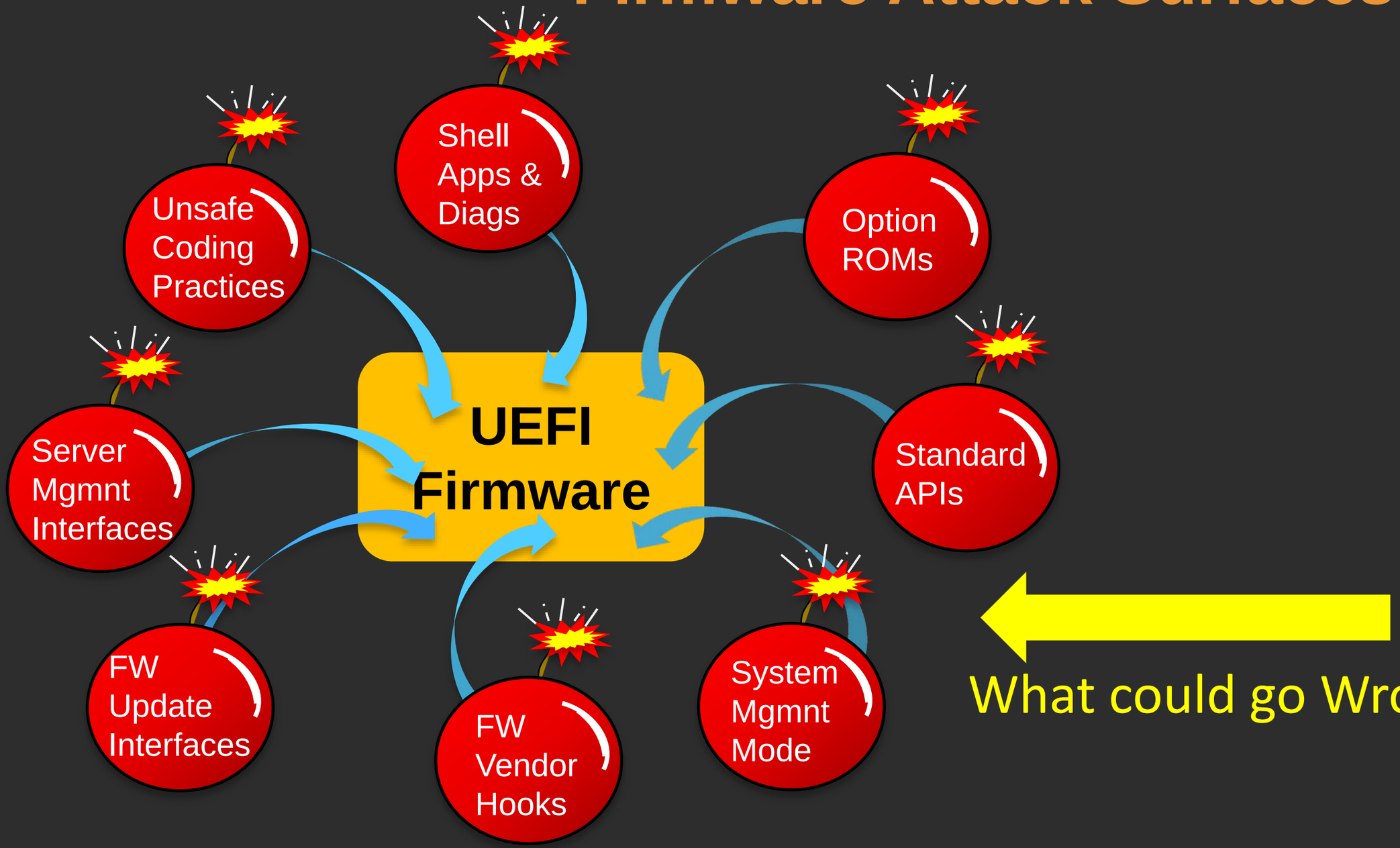
UEFI & Platform Initialization Task Flow



[1] S-CRTM - static core root of trust for measurement

tianocore

Firmware Attack Surfaces



tianocore

Goals of security architecture and assets that are protected

NIST SP 800-33 – IT Security Objectives

Availability, Integrity, Confidentiality, Accountability, Assurance

The goal of information technology security:

Enable an organization to meet all of its mission/business objectives by implementing systems with due care consideration of IT-related risks to the organization, its partners and customers.

tianocore

Goals of security architecture and assets that are protected

NIST SP 800-33 – IT Security Objectives

Availability, Integrity, Confidentiality, Accountability, Assurance

Confidentiality	• Protect against unauthorized access
Integrity	• Protection of Content & Quality
Availability	• Ensure access
Accountability	• Also Authenticity - traced uniquely to the source entity
Assurance	• Guarantee on the correctness, Non-repudiation

All of these objectives are interdependent with Assurance

What to build & defend – Rationale for a threat model

"*My house is secure*" is almost meaningless

- Against a burglar? Against a meteor strike? A thermonuclear device?

"*My system is secure*" is almost meaningless

- Against what? To what extent?

Threat modeling is a process to define the goals and constraints of a (software) security solution

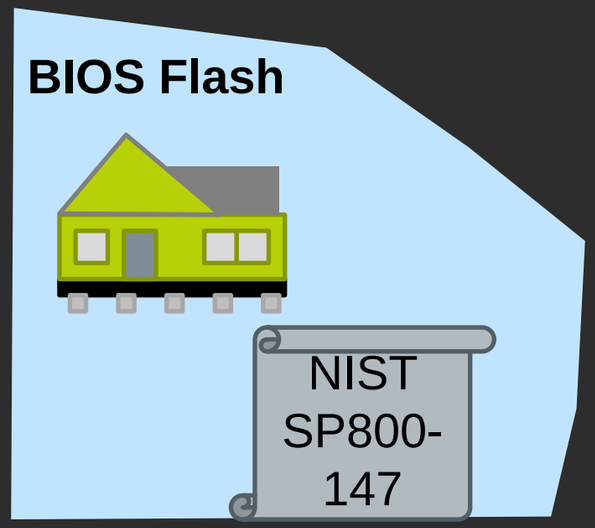
- Translate user requirements to security requirements

We use threat modeling for our UEFI / PI codebase

- We believe the process and findings are applicable to driver implementations as well as UEFI implementations in general

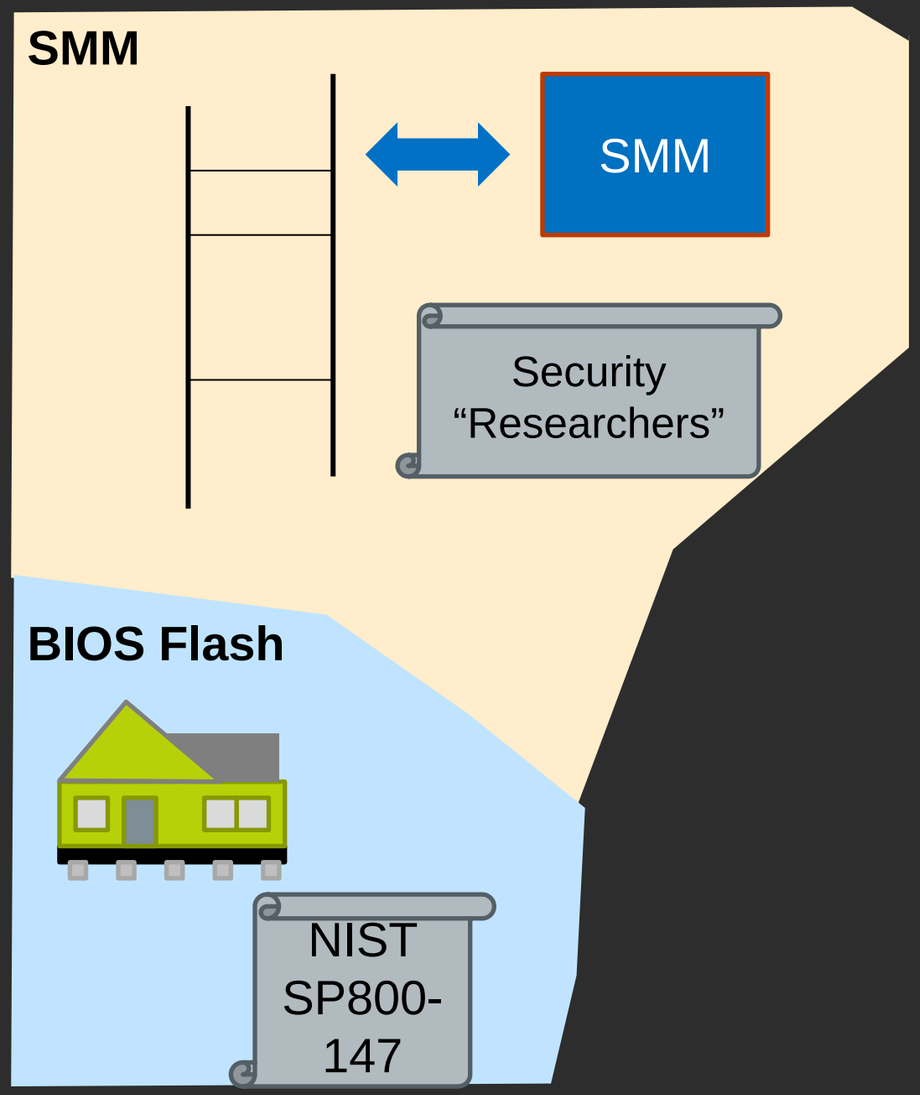
We Need to protect our Assets from Threats

We don’t always get to choose our Assets



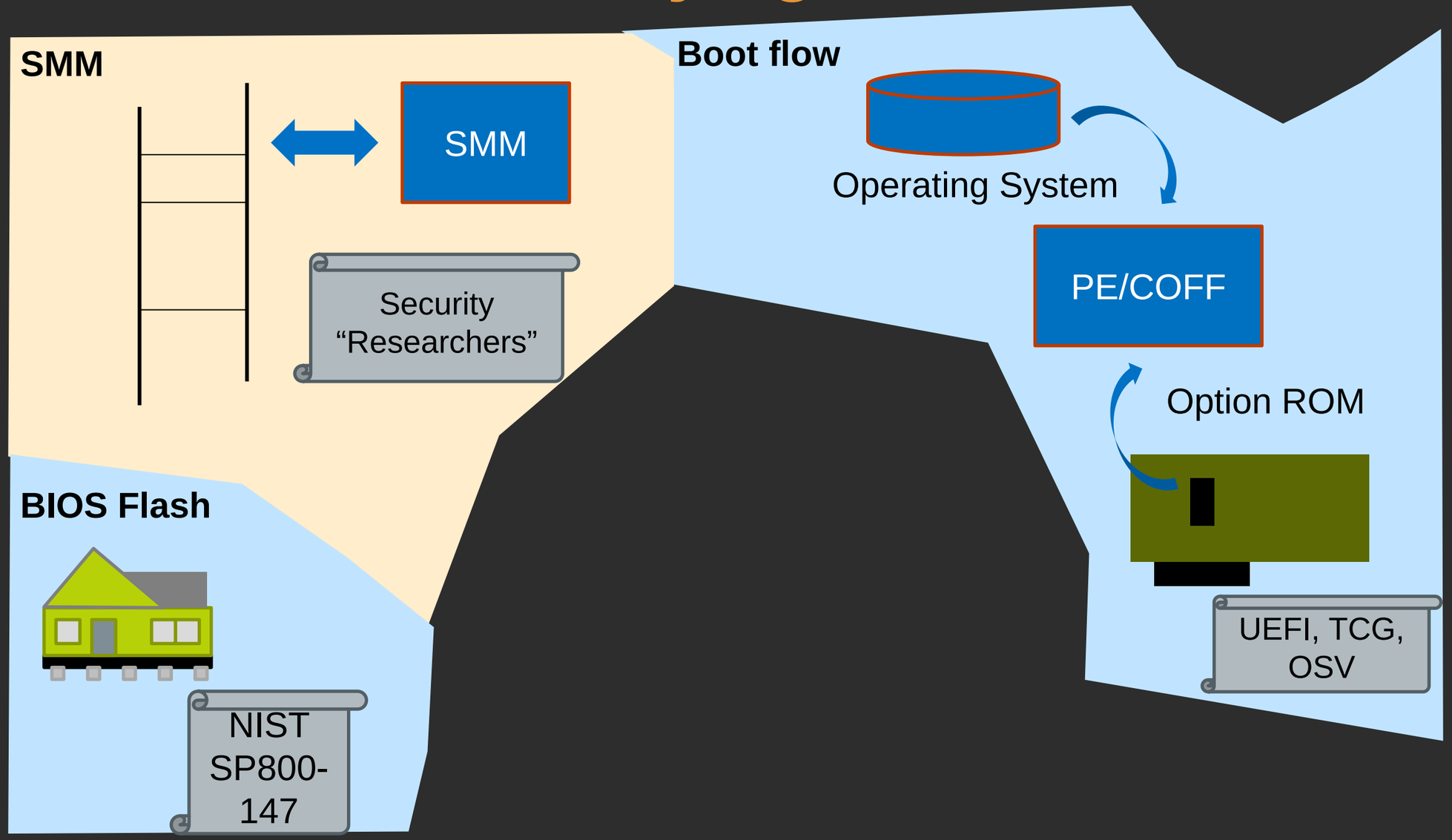
Source: UEFI open Platforms_Vincent.ppt

We don’t always get to choose our Assets



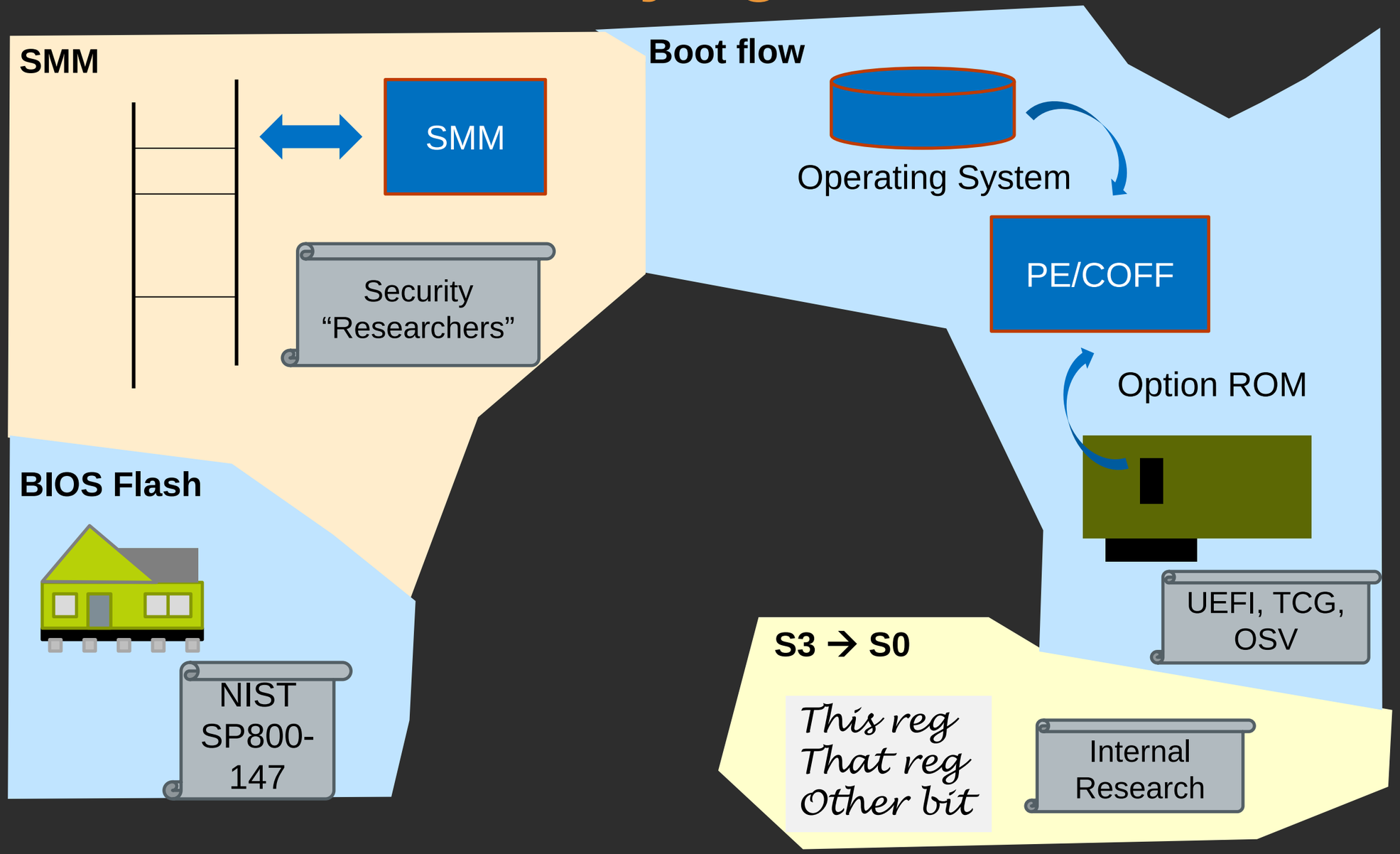
Source: UEFI open Platforms_Vincent.ppt

We don’t always get to choose our Assets



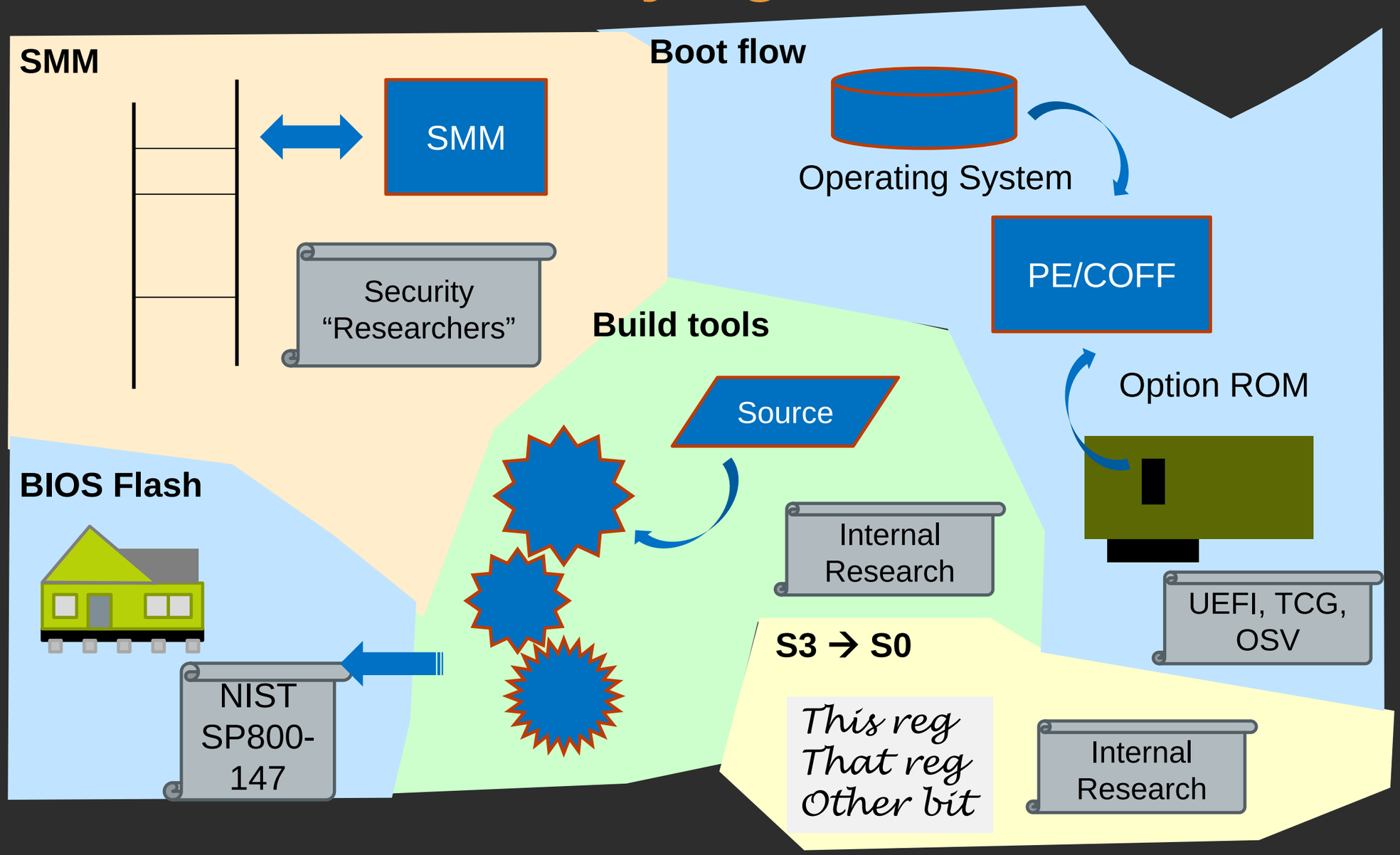
Source: UEFI open Platforms_Vincent.ppt

We don’t always get to choose our Assets



Source: UEFI open Platforms_Vincent.ppt

We don’t always get to choose our Assets

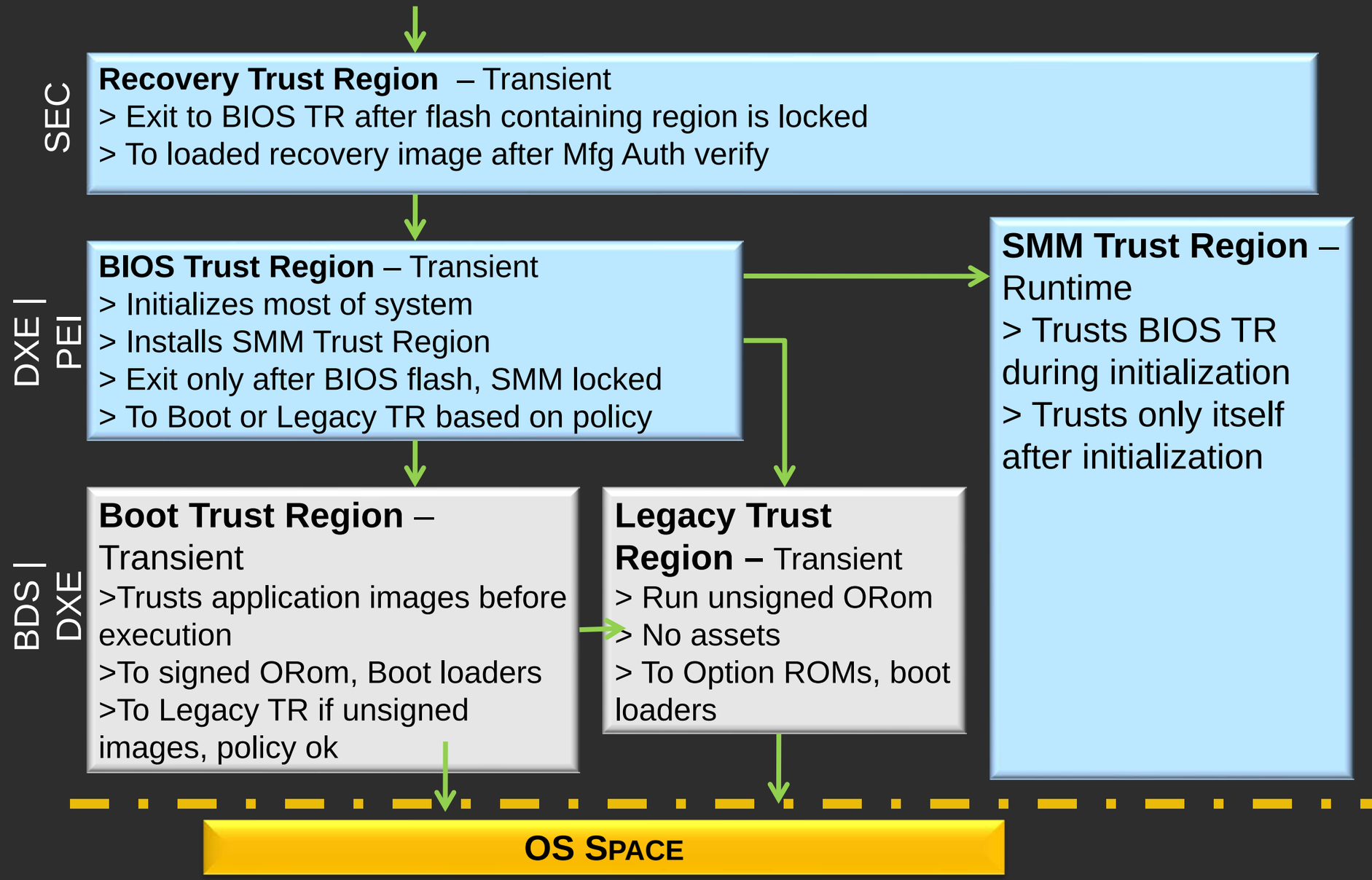


Source: UEFI open Platforms_Vincent.ppt

Baseline: Boot trust regions

A Threat Model (TM) defines the security assertions and constraints for a product

- Assets:
- Threats:
- Mitigations:



Threat Model with Examples

Boot Trust Regions and Assets

Asset	Example Threats	Mitigations	Checks
Firmware/BIOS Flash Contents	CIH attack: erase boot block	SPI locks, descriptor	CHIPSEC
SMM	Callouts; Access to SMM	TSEG, SMRR, SMM_CODE_CHK	CHIPSEC
Execution During Boot Flow	Run malware in Op ROM	Secure Boot, DMA protection	Manual testing (eg CHIPSEC)
S3 Boot Script & S3 Resume Boot Flow	Resume reconfiguration losing locks	SMM Lock Box	Manual testing (eg CHIPSEC)
UEFI Variables (includes Authenticated & non-Authenticated)	Variable store full; Content change	Attributes, Lock Protocol	Manual testing (eg CHIPSEC)
Etc . . .			

Summary

Platform Firmware Security – Why is it important?

Why is platform firmware Security important	Prevent low level attacks that could “brick” the system
UEFI boot flow with the threat model	Identify where UEFI firmware is vulnerable and define a Threat Model

tianocore

SECURITY TECHNOLOGIES

Security Technologies Overview

tianocore

BOOT SECURITY

BOOT SECURITY TECHNOLOGIES

Hardware Root of Trust	Boot Guard, Intel® TXT
Measured Boot	Using TPM[1] to store hash values
Verified Boot	Boot Guard + UEFI Secure Boot

[1]TPM – Trusted Platform Module

Resources: https://firmwaresecurity.com/2015/07/29/survey-of-boot-security-technologies/

HARDWARE ROOT OF TRUST

Boot Guard

CPU verifies signature

Verification occurs before BIOS starts

Hash of signature is fused in CPU

Verification

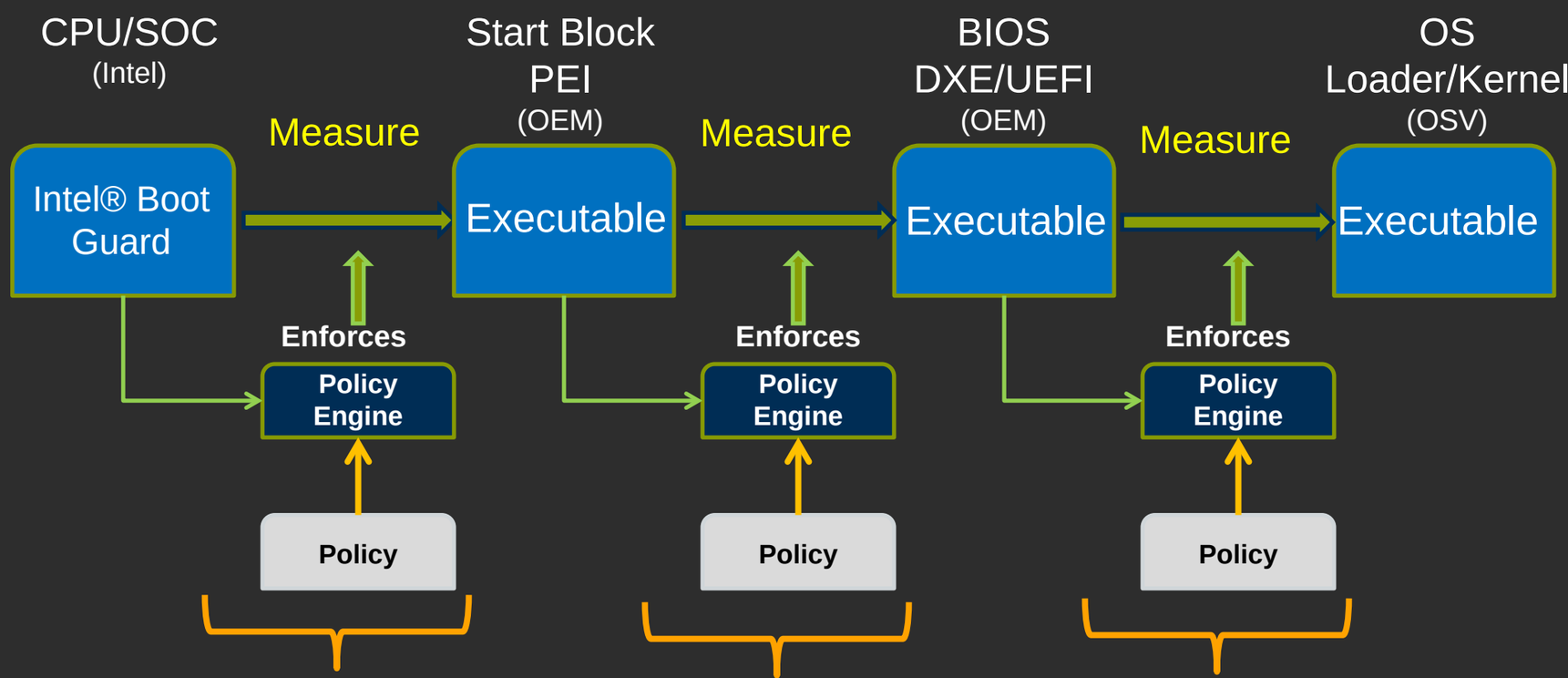
Intel® TXT

Uses a Trusted Platform Module (TPM) & cryptographic

Provides Measurements

Measurements

Full Verified Boot Sequence



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

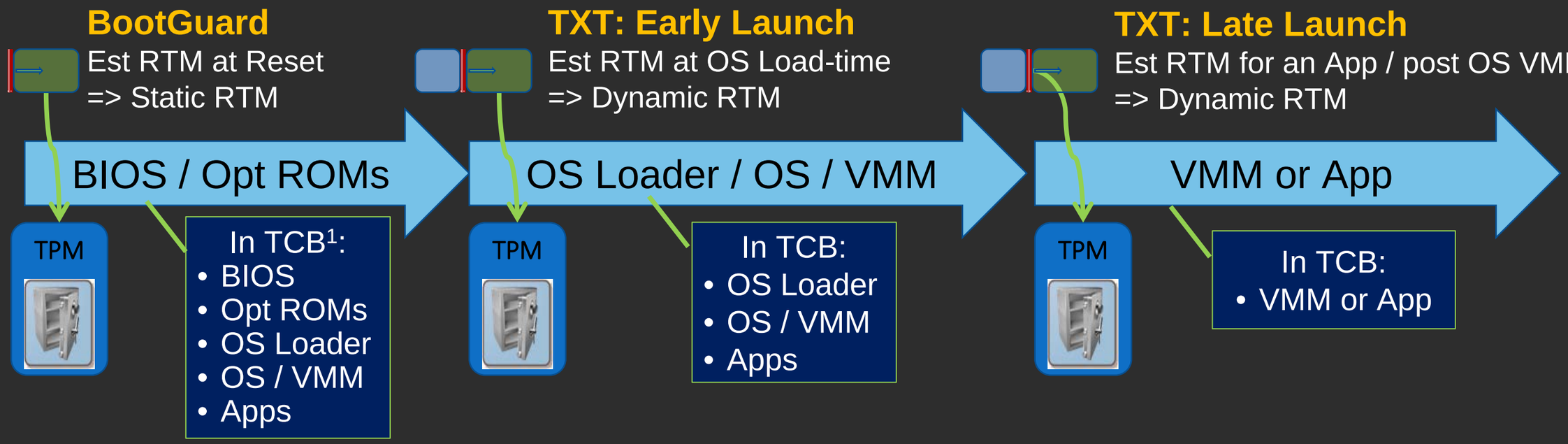
Chapter 27.2 of
The UEFI 2.4
Specification

https://firmware.intel.com/sites/default/files/resources/Platform_Security_Review_Intel_Cisco_White_Paper.pdf

Boot Guard and TXT

Both features rooted in Trusted Computing.

Establishing a Trusted Environment (Intel provides 3 modes)
Starts with Root of Trust for Measurement (RTM)

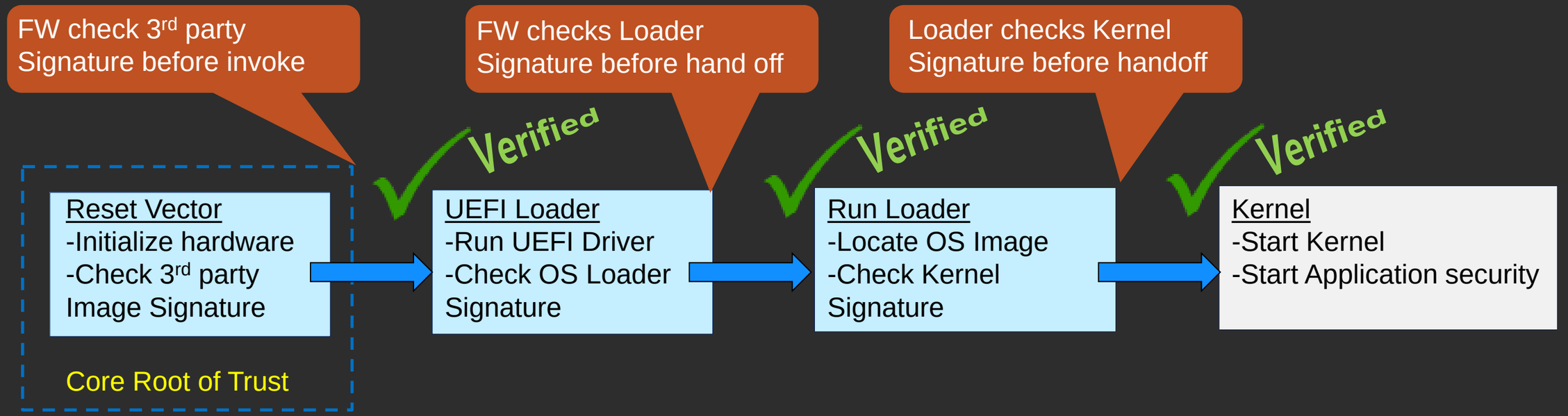


1. Trusted computing base (TCB)

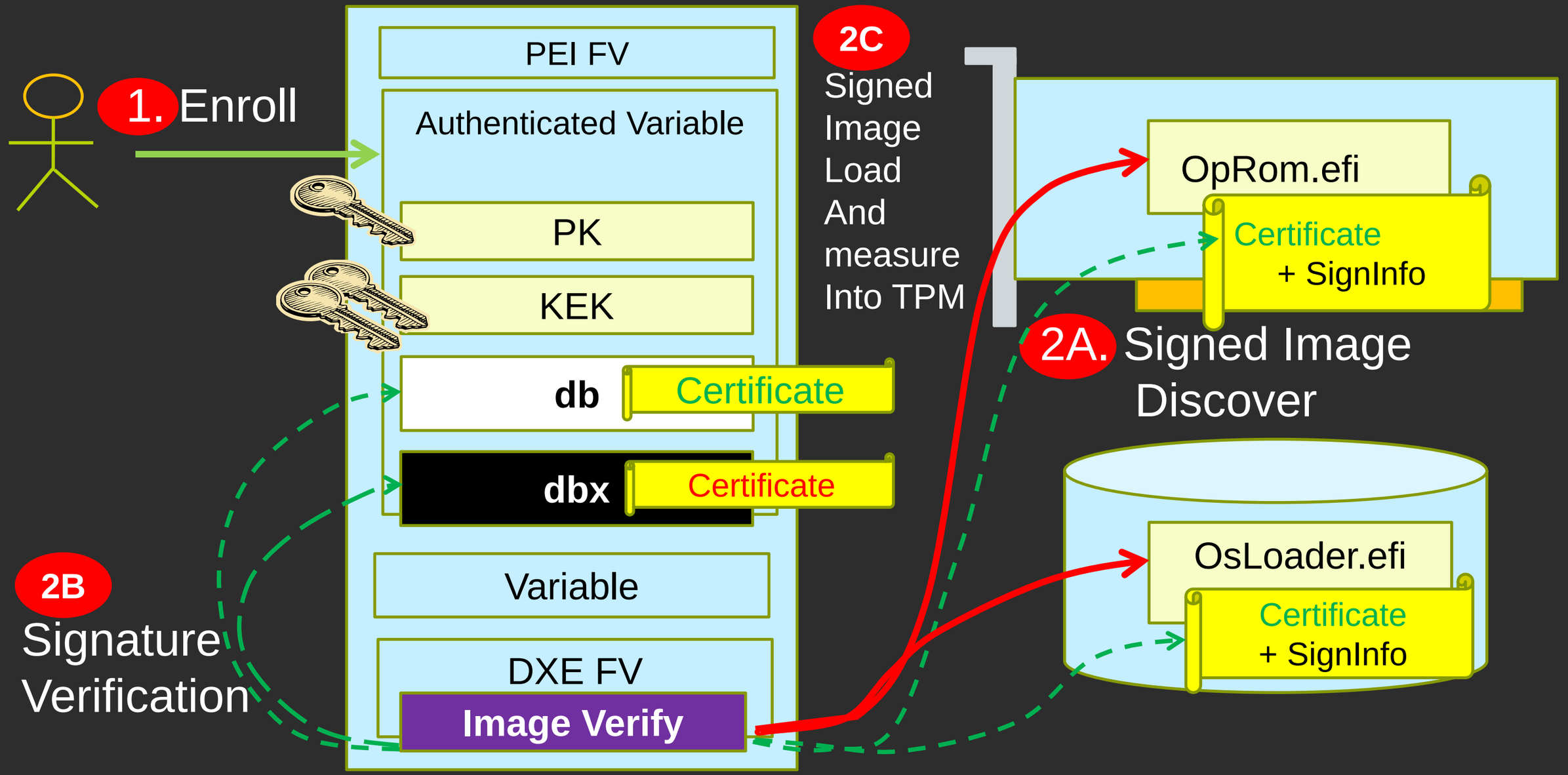
UEFI SECURE BOOT

Software ID checking during every step of the boot flow:

1. UEFI System FW (updated via secure process)
2. Add-In Cards (signed UEFI Option ROMs)
3. OS Boot Loader (checks for “secure mode” at boot)



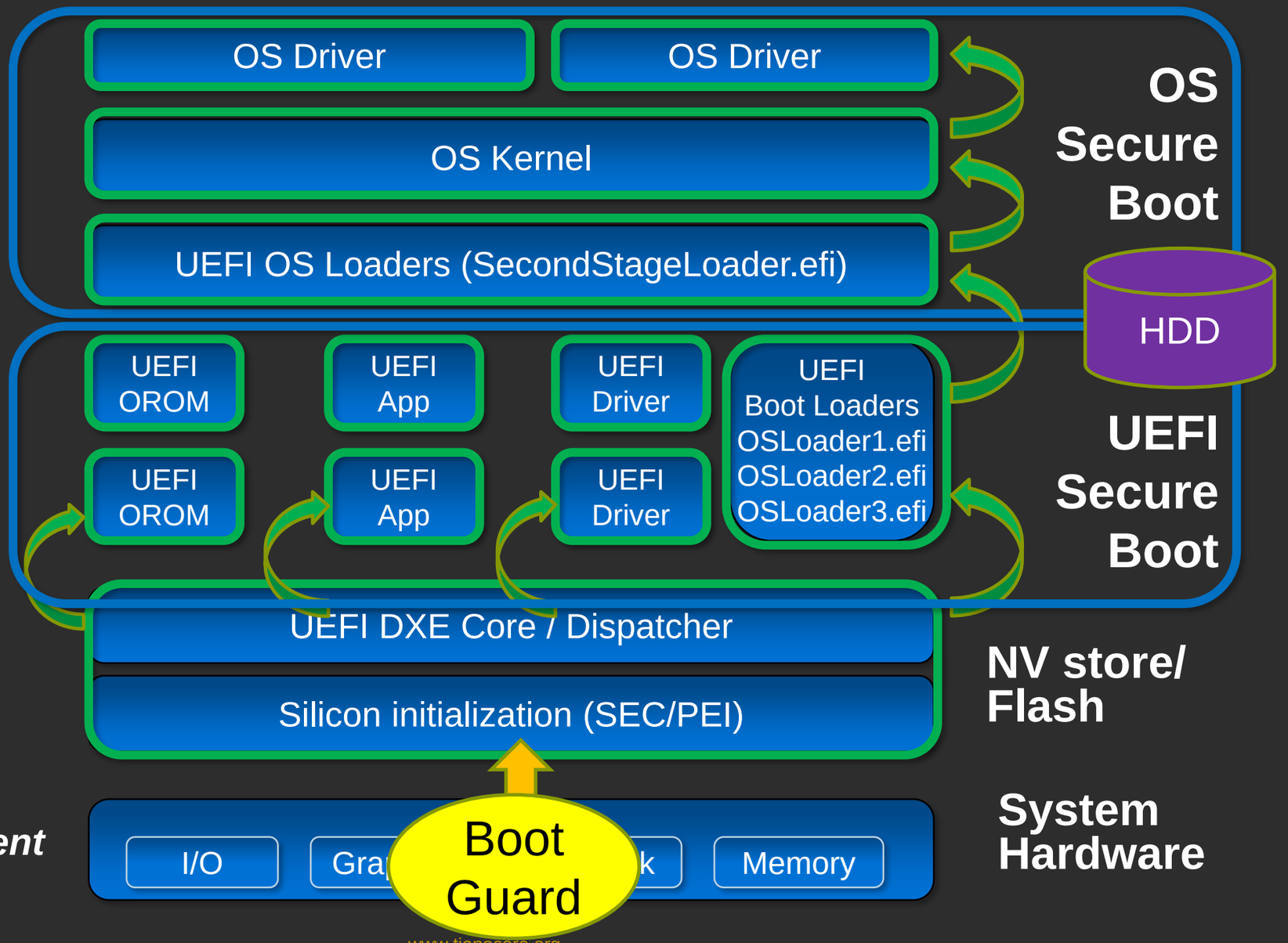
UEFI Secure Boot Flow



tianocore

End to End Platform Integrity

Reference:
Where do I 'sign' up?



Intel® Device Protection Technology with Boot Guard – Secure *Boot Policy Enforcement*

tianocore

FLASH DEVICE

tianocore

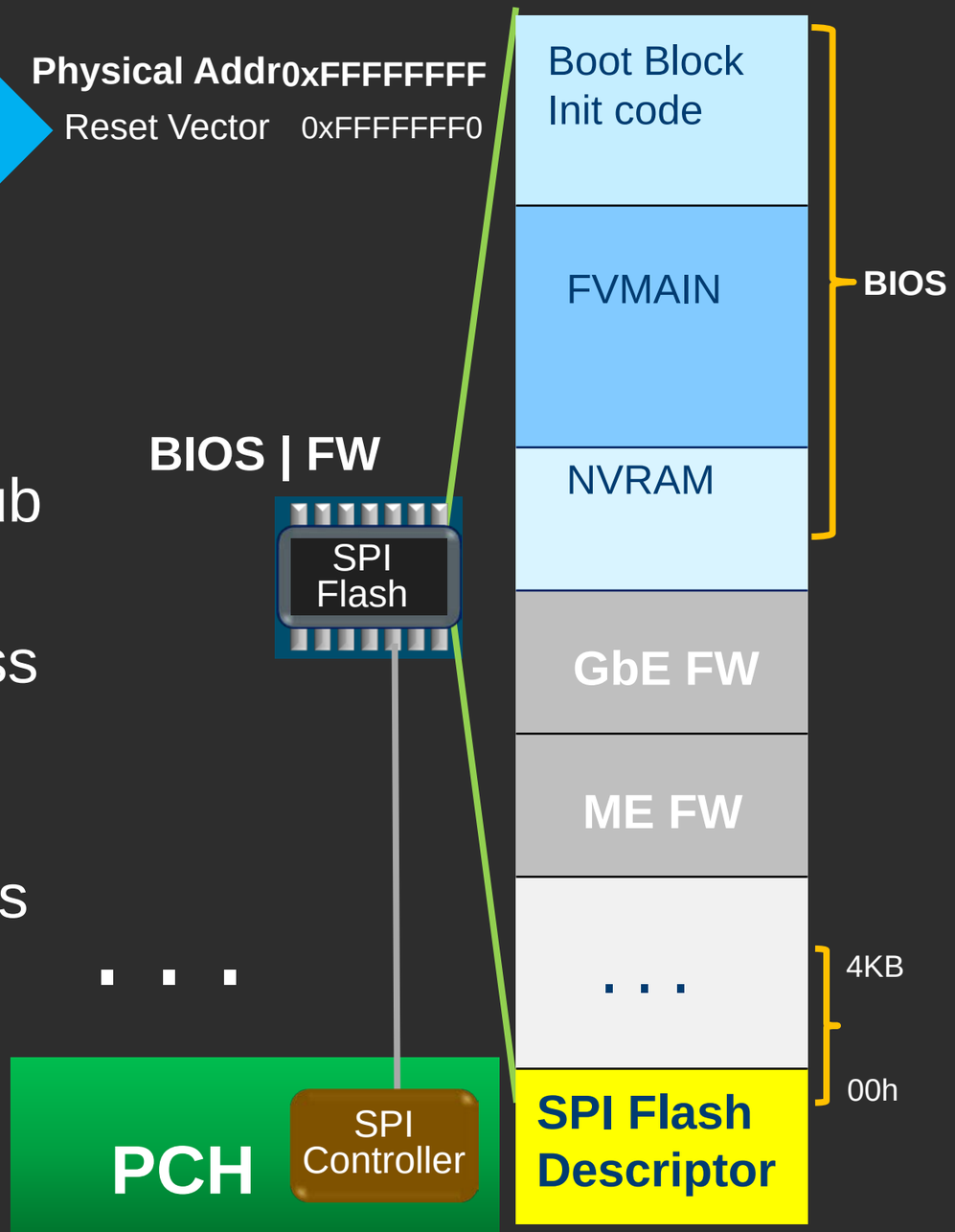
BIOS Firmware uses SPI Flash

Serial Peripheral Interface (SPI)

- Access controlled by the Peripheral Controller Hub (PCH)
- Flash Memory - Direct access to physical address space is programed through SPI MMIO registers
- SPI Flash Descriptor - Access Control table defines which device (CPU, ME, GbE) can access which regions

Flash Descriptor has to be write-protected

ME – Manageability Engine
GbE – Gigabit Ethernet



tianocore

System Flash Security

Chipset (SPI controller) based protections

- SMM based BIOS Write Protection: write-protects entire BIOS/Firmware region from software other than SMI handler firmware executing in SMM
- SPI Protected Range registers(PR0-PR4) : read/write protection of SPI flash regions based on Flash Linear Address for program register access
- Flash Descriptor based access control: defines read/write access to each flash region by each Host Device

Firmware may use SPI flash chips write protection(WP#)

PR0-PR4 defined in SPI MMIO

Lock SPI - BIOS Range is not protected - Threats

- BIOS Write Protections often still not properly enabled on many systems
- SMM based write protection of entire BIOS region is often not used: BIOS_CONTROL[SMM_BWP]
- If SPI Protected Ranges (mode agnostic) are used (defined by **PR0-PR4** in SPI MMIO), they often don't cover entire BIOS & NVRAM
- Some platforms use SPI device specific write protection but only for boot block/startup code or SPI Flash descriptor region

Mitigations:

- Set `BIOS_CONTROL[SMM_BWP] <- 1`
- Program SPI flash protected ranges (**PRx**) to cover BIOS range

References: Persistent BIOS Infection (used flashrom on legacy BIOS), Evil Maid Just Got Angrier, BIOS Chronomancy, A Tale Of One Software Bypass Of Windows 8 Secure

tianocore

FIRMWARE SECURE UPDATE

Solving Firmware Update

Reliable update story

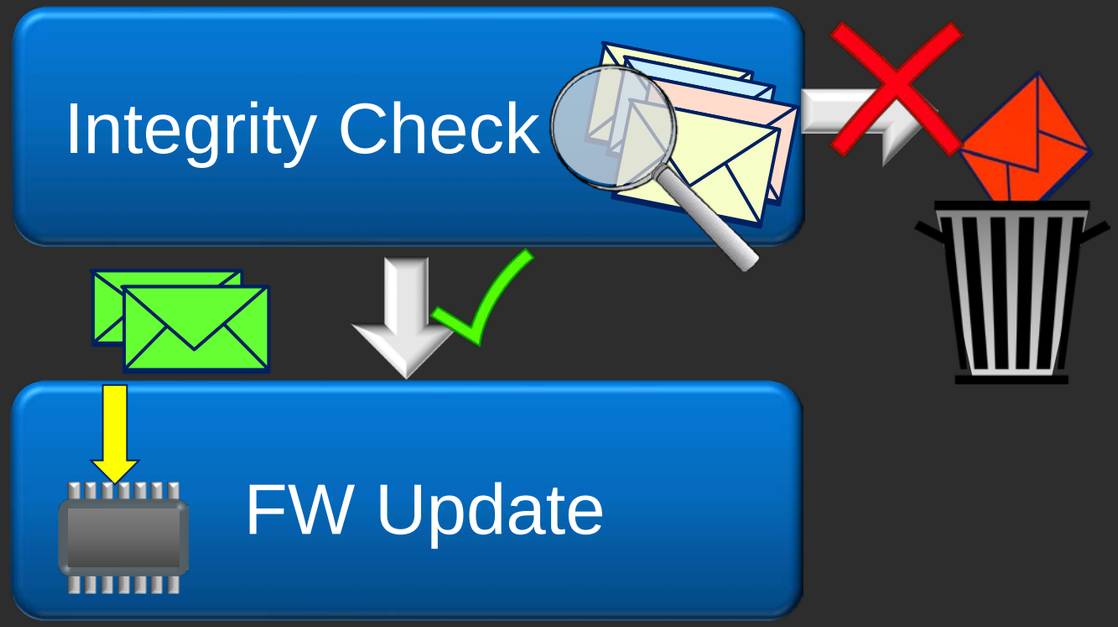
- Fault tolerant
- Scalable & repeatable

Refrences [6] at: UEFI, Open Platforms and ppt

Solving Firmware Update

Reliable update story

- Fault tolerant
- Scalable & repeatable



Refrences [6] at: UEFI, Open Platforms and ppt

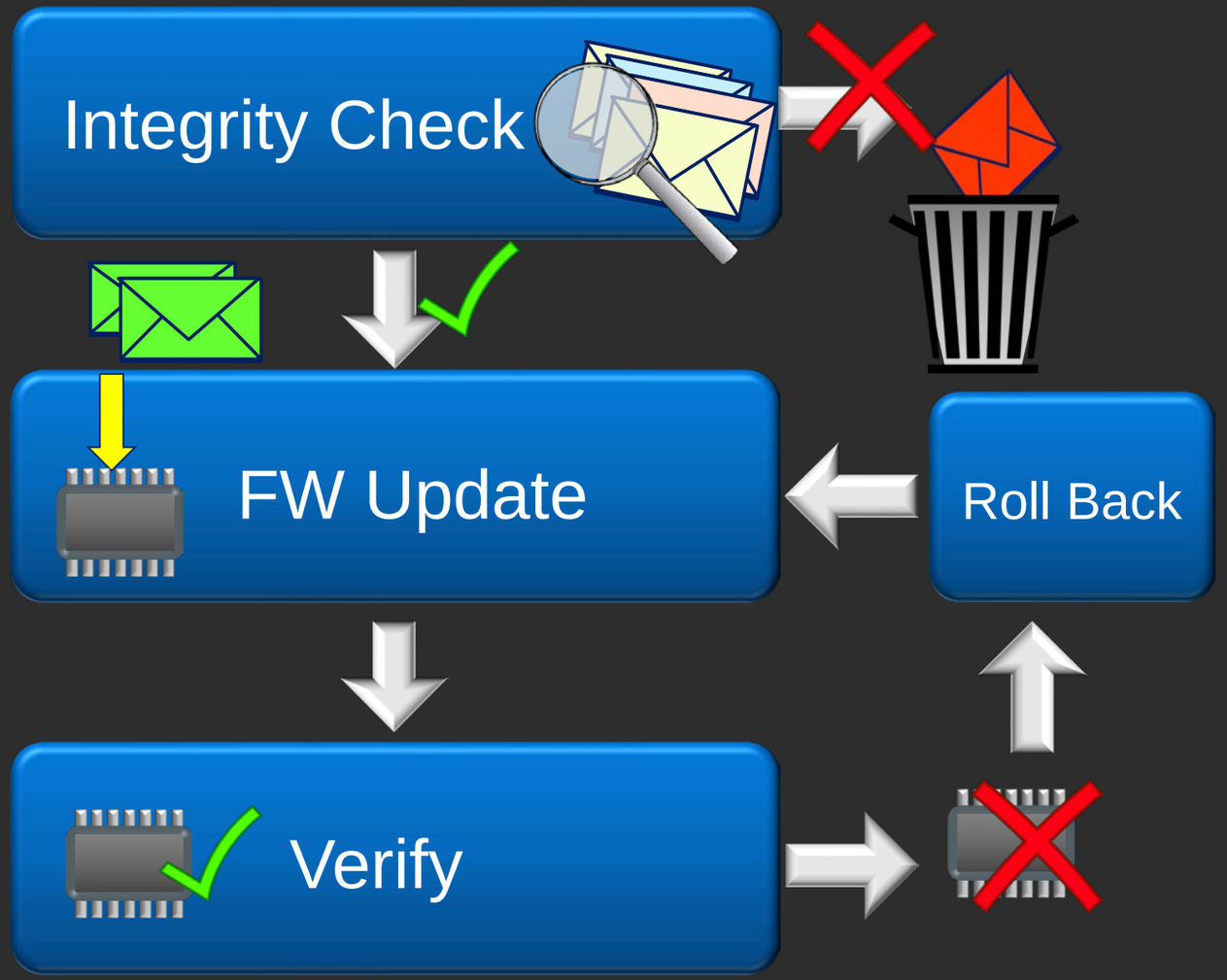
Solving Firmware Update

Reliable update story

- Fault tolerant
- Scalable & repeatable

How can UEFI Help?

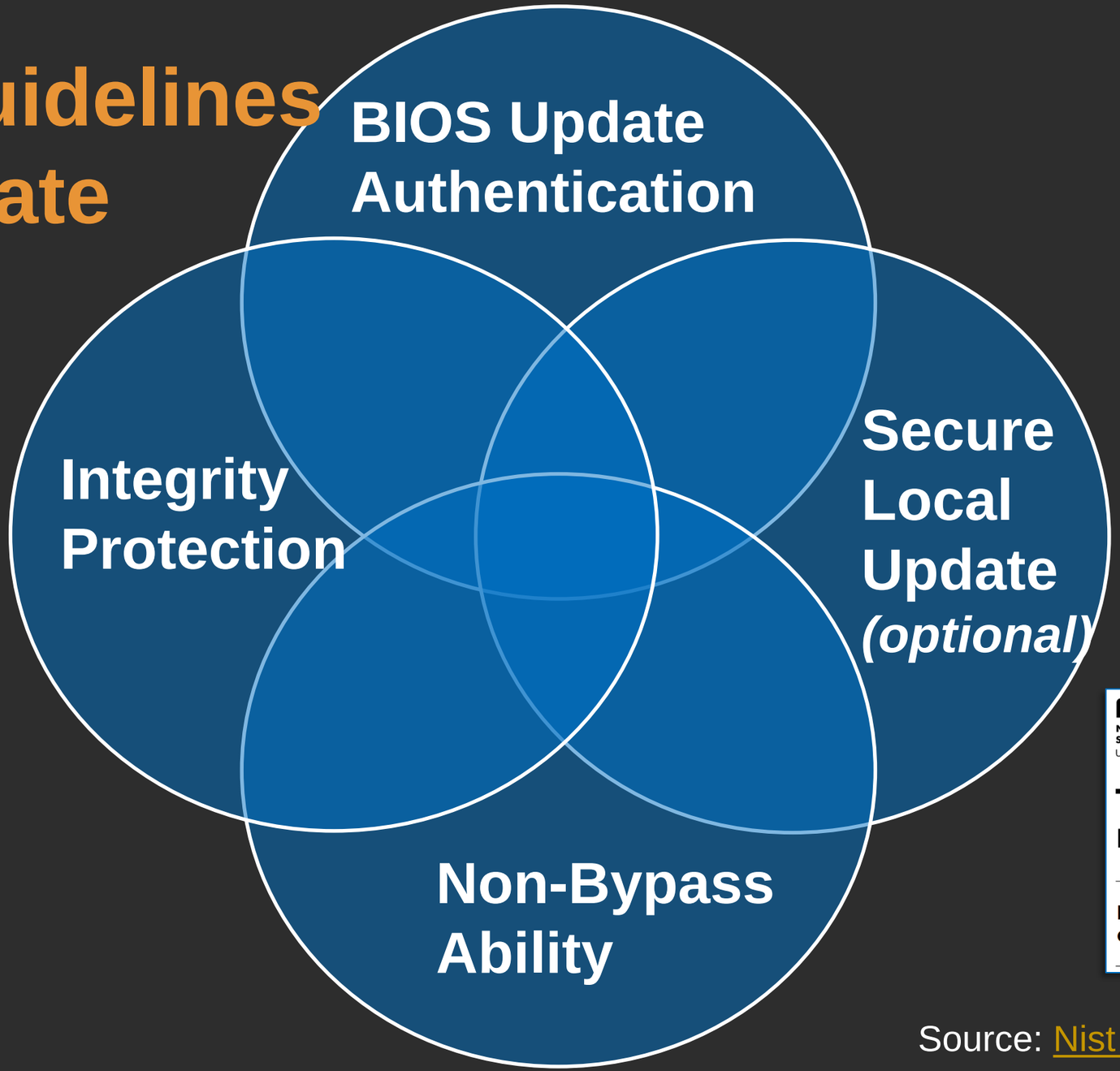
- Capsule model for binary delivery
- Bus / Device Enumeration
- Managing updates via
  - EFI System Resource Table
  - Firmware Management Protocol
  - Capsule Signing

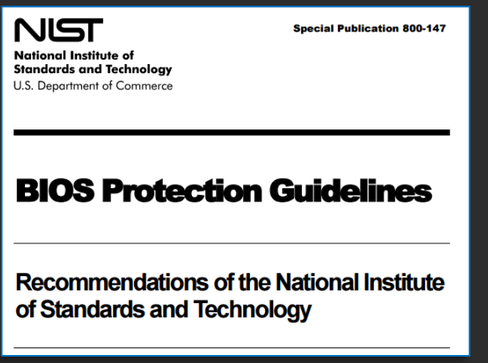


Refrences [6] at: UEFI, Open Platforms and ppt

tianocore

Security Guidelines - BIOS Update




NIST
National Institute of Standards and Technology
U.S. Department of Commerce

Special Publication 800-147

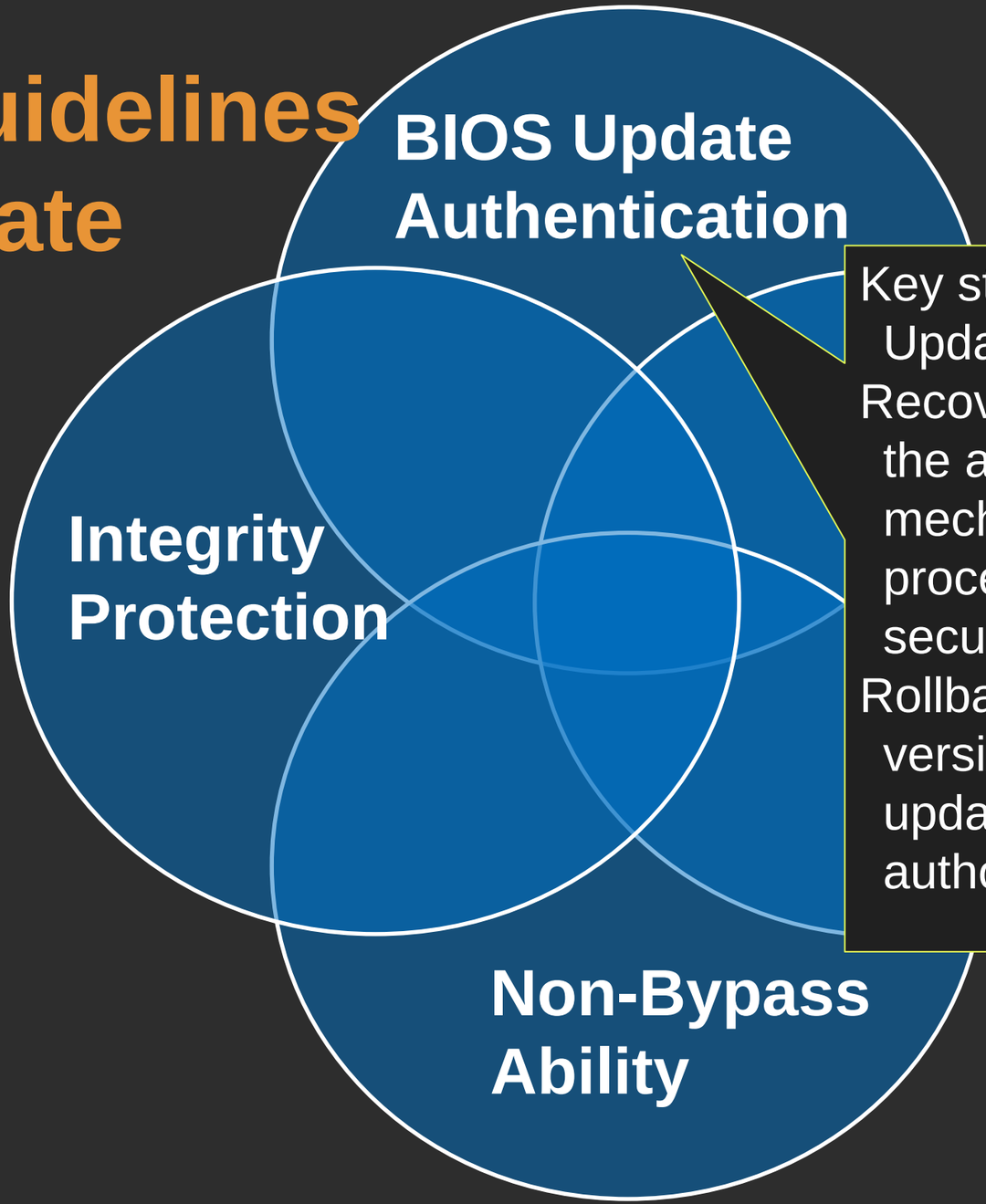
BIOS Protection Guidelines

Recommendations of the National Institute of Standards and Technology

Source: Nist SP 800-147 .pdf

tianocore

Security Guidelines - BIOS Update



Key storage in Root of Trust for Update (RTU)

Recovery mechanisms shall also use the authenticated update mechanism unless the recovery process meets the guidelines for a secure local update.

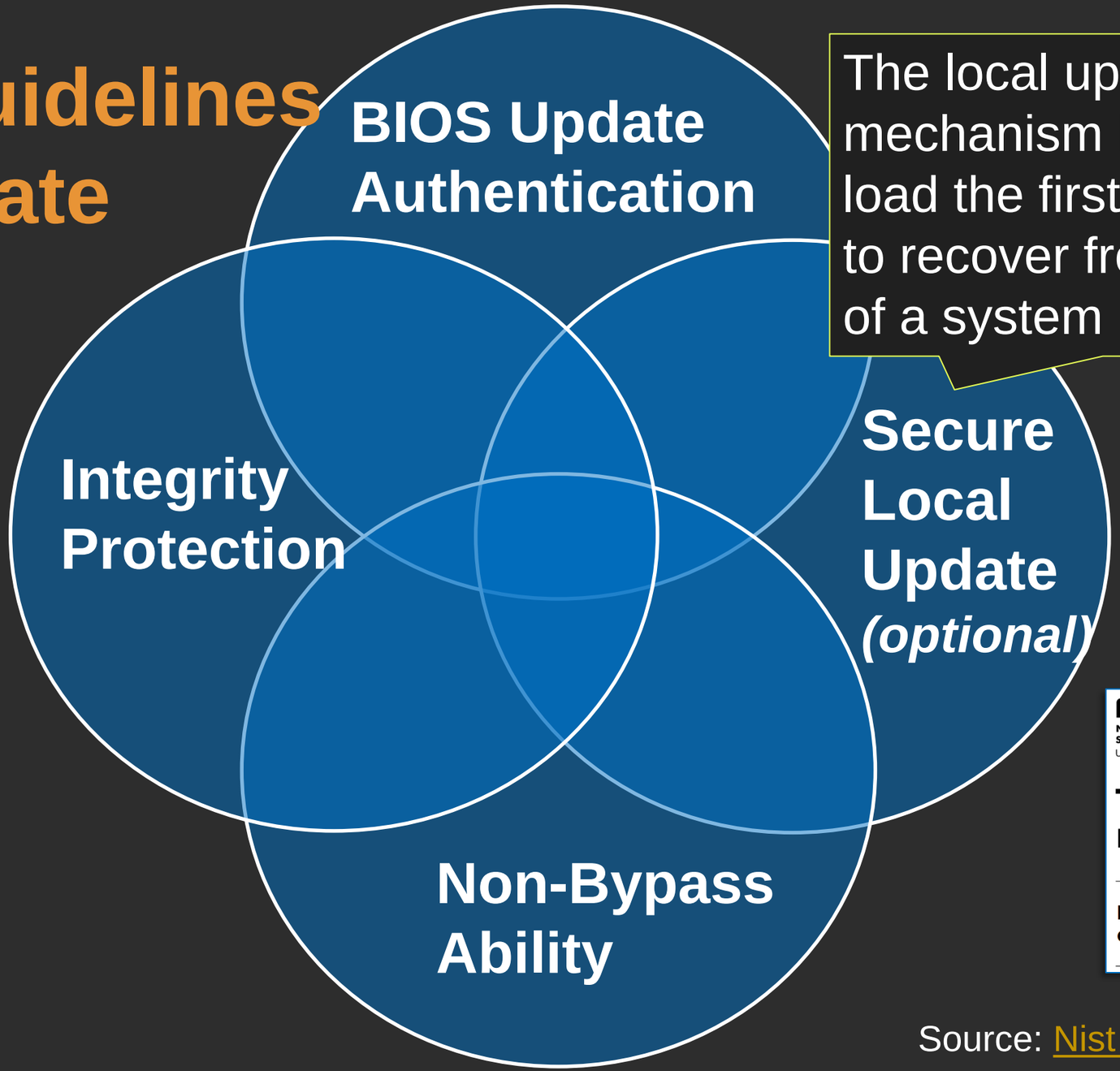
Rollbacks of the BIOS to an earlier version are permitted only if the update or rollback has been authorized by the organization.

Recommendations of the National Institute of Standards and Technology

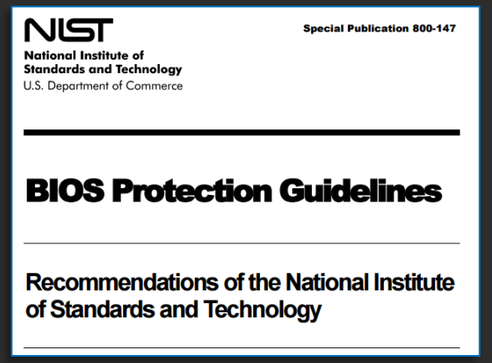
Source: Nist SP 800-147 .pdf

tianocore

Security Guidelines - BIOS Update



The local update mechanism be used only to load the first BIOS image or to recover from a corruption of a system BIOS.


NIST
National Institute of Standards and Technology
U.S. Department of Commerce

Special Publication 800-147

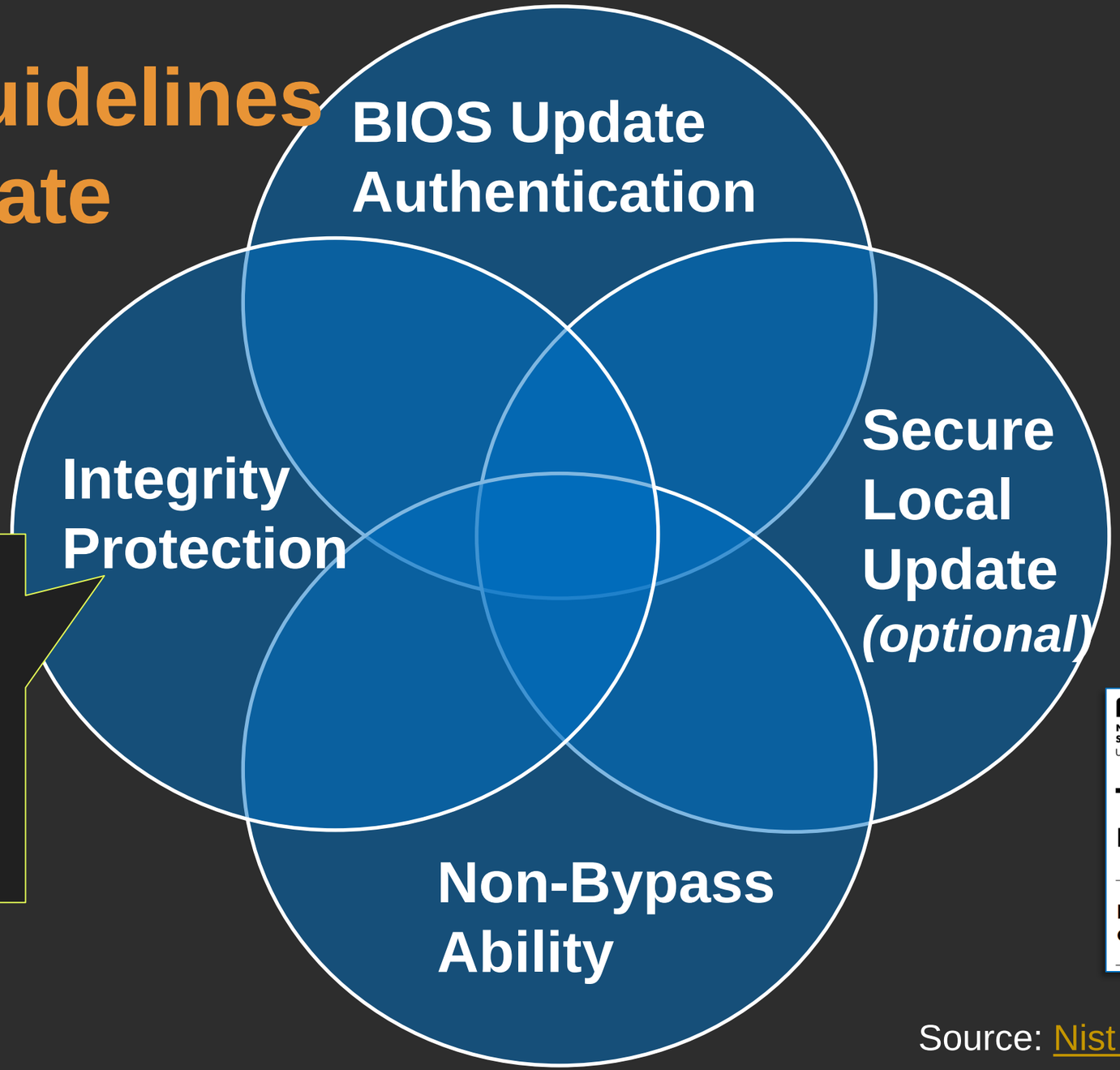
BIOS Protection Guidelines

Recommendations of the National Institute of Standards and Technology

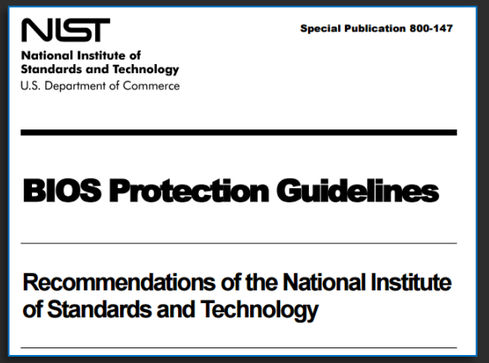
Source: Nist SP 800-147 .pdf

tianocore

Security Guidelines - BIOS Update



The RTU and the system BIOS shall be protected from unintended modification.


NIST
National Institute of Standards and Technology
U.S. Department of Commerce

Special Publication 800-147

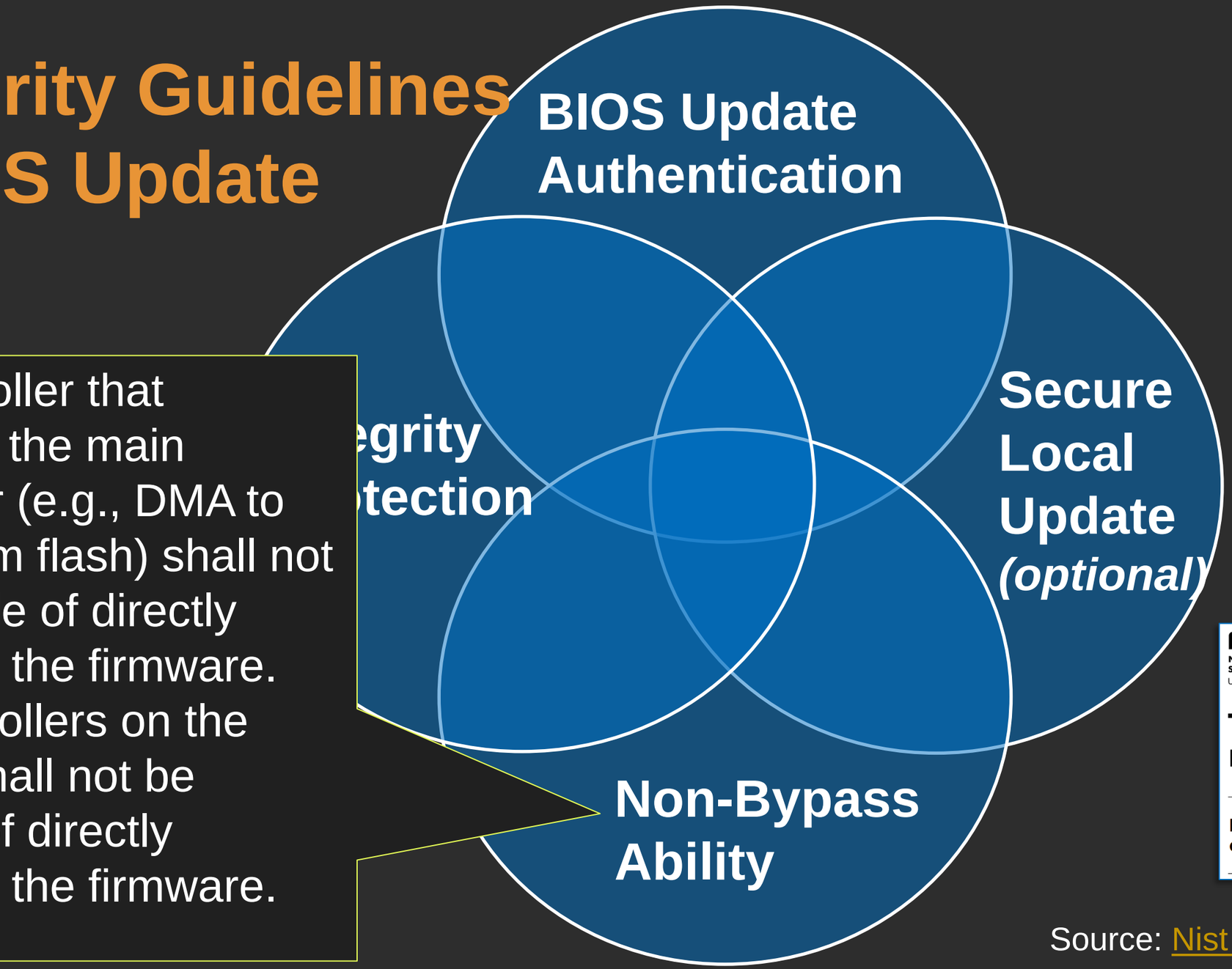
BIOS Protection Guidelines

Recommendations of the National Institute of Standards and Technology

Source: Nist SP 800-147 .pdf

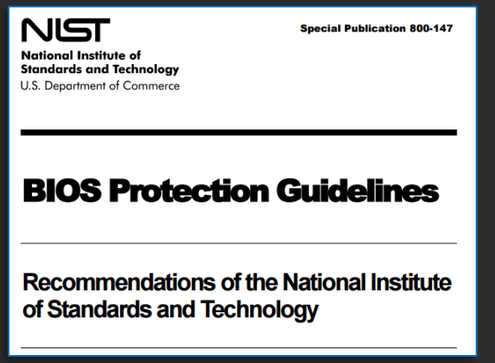
tianocore

Security Guidelines - BIOS Update



Bus Controller that bypasses the main processor (e.g., DMA to the system flash) shall not be capable of directly modifying the firmware.

Microcontrollers on the system shall not be capable of directly modifying the firmware.


NIST
National Institute of Standards and Technology
U.S. Department of Commerce

Special Publication 800-147

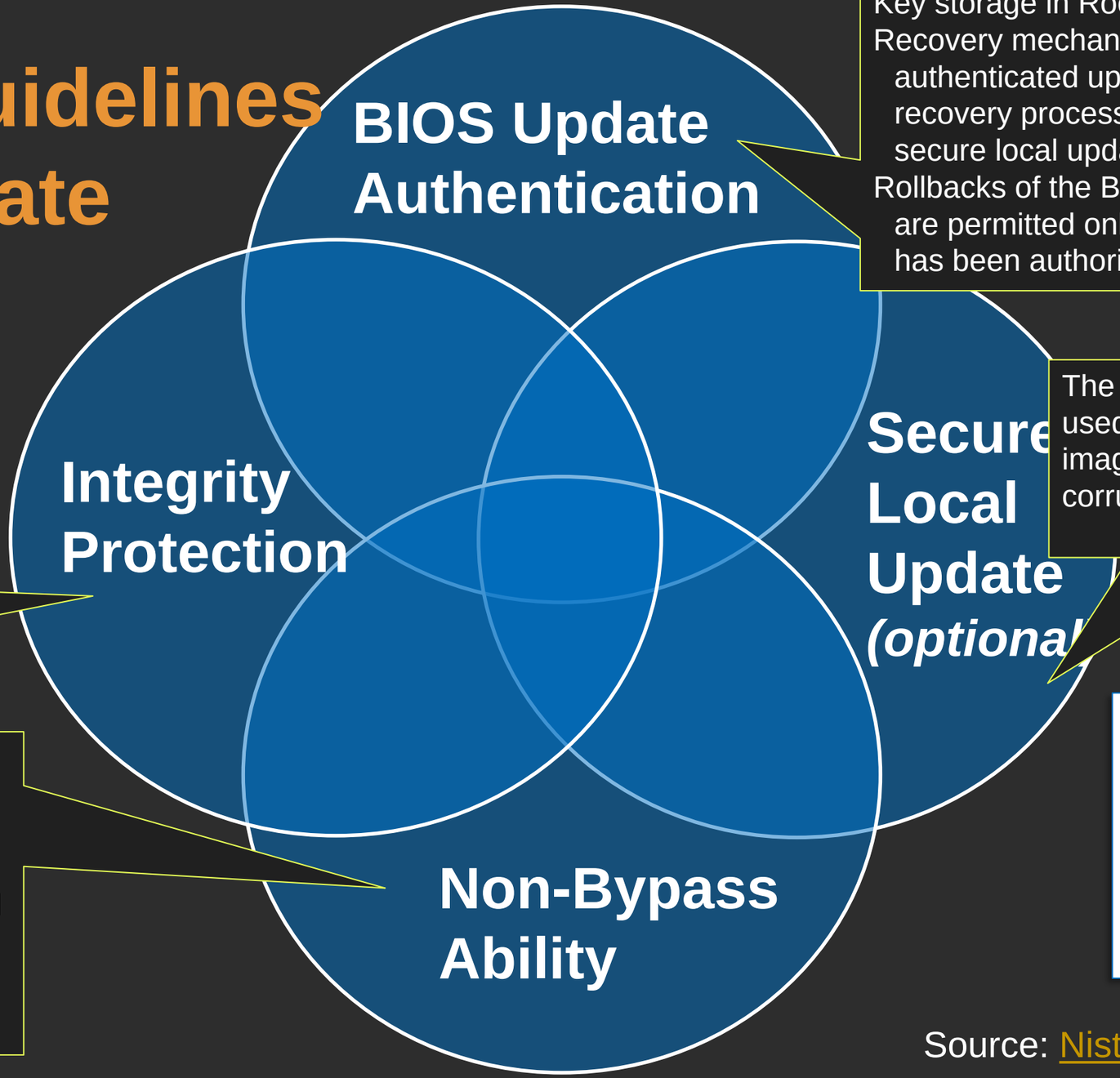
BIOS Protection Guidelines

Recommendations of the National Institute of Standards and Technology

Source: Nist SP 800-147 .pdf

tianocore

Security Guidelines - BIOS Update

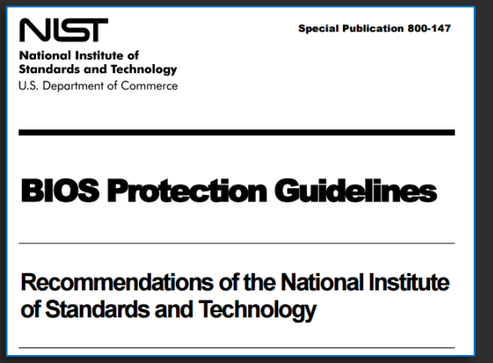


Key storage in Root of Trust for Update (RTU)
Recovery mechanisms shall also use the authenticated update mechanism unless the recovery process meets the guidelines for a secure local update.
Rollbacks of the BIOS to an earlier version are permitted only if the update or rollback has been authorized by the organization.

The local update mechanism be used only to load the first BIOS image or to recover from a corruption of a system BIOS.

The RTU and the system BIOS shall be protected from unintended modification.

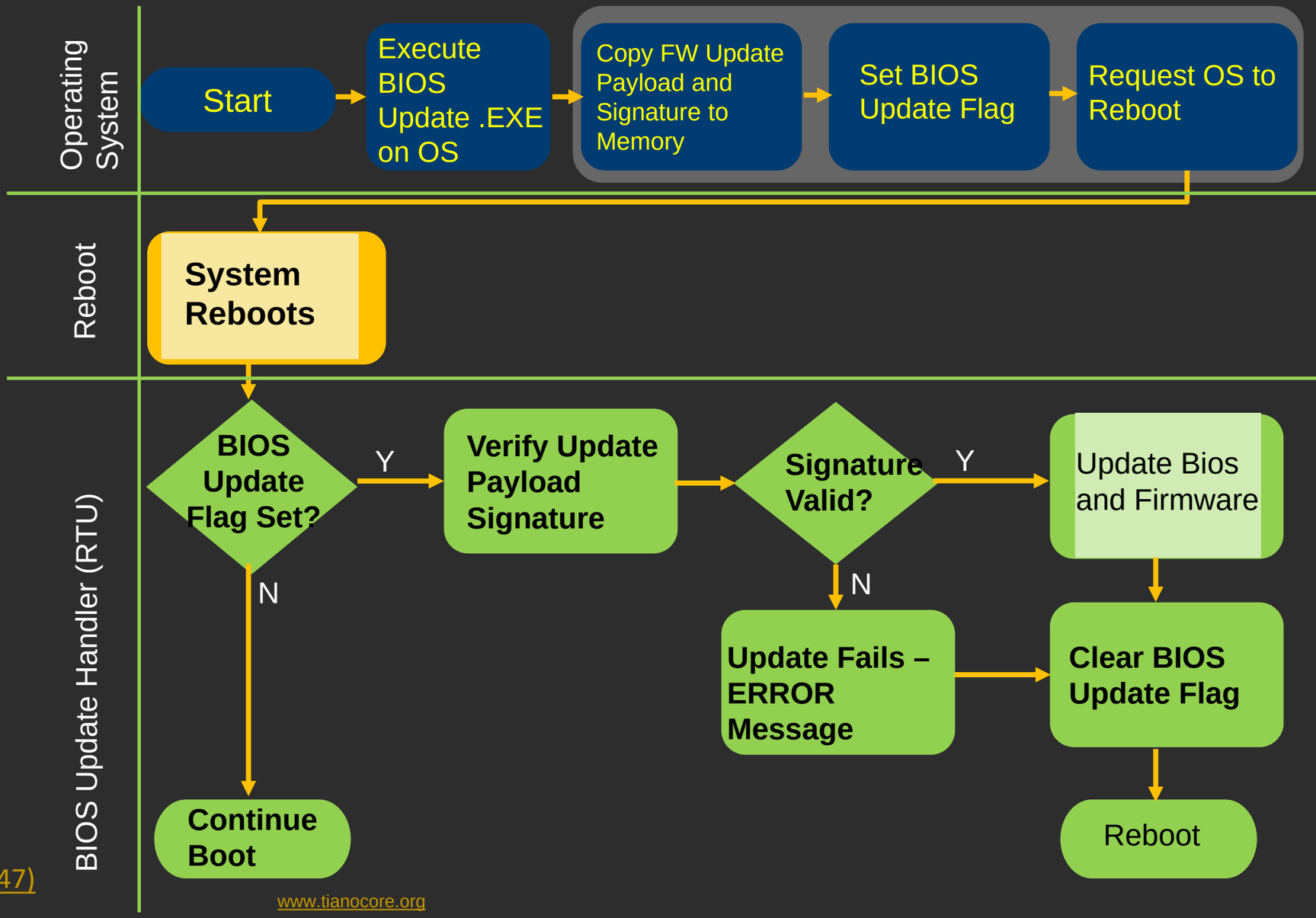
Bus controller that bypasses the main processor (e.g., DMA to the system flash) shall not be capable of directly modifying the firmware.
Microcontrollers on the system shall not be capable of directly modifying the firmware.


NIST
National Institute of Standards and Technology
U.S. Department of Commerce
Special Publication 800-147
BIOS Protection Guidelines
Recommendations of the National Institute of Standards and Technology

Source: Nist SP 800-147 .pdf

Signed Firmware Update

- RTU protected by flash locking mechanisms at the hardware level
- BIOS key store includes the full public key used to verify the signature of all BIOS and firmware updates
- Capsule Update with UEFI FMP



RTU - BIOS Root of Trust for Update
FMP- Firmware Management Protocol

Source: Dell Signed Firmware Update (NIST 800-147)

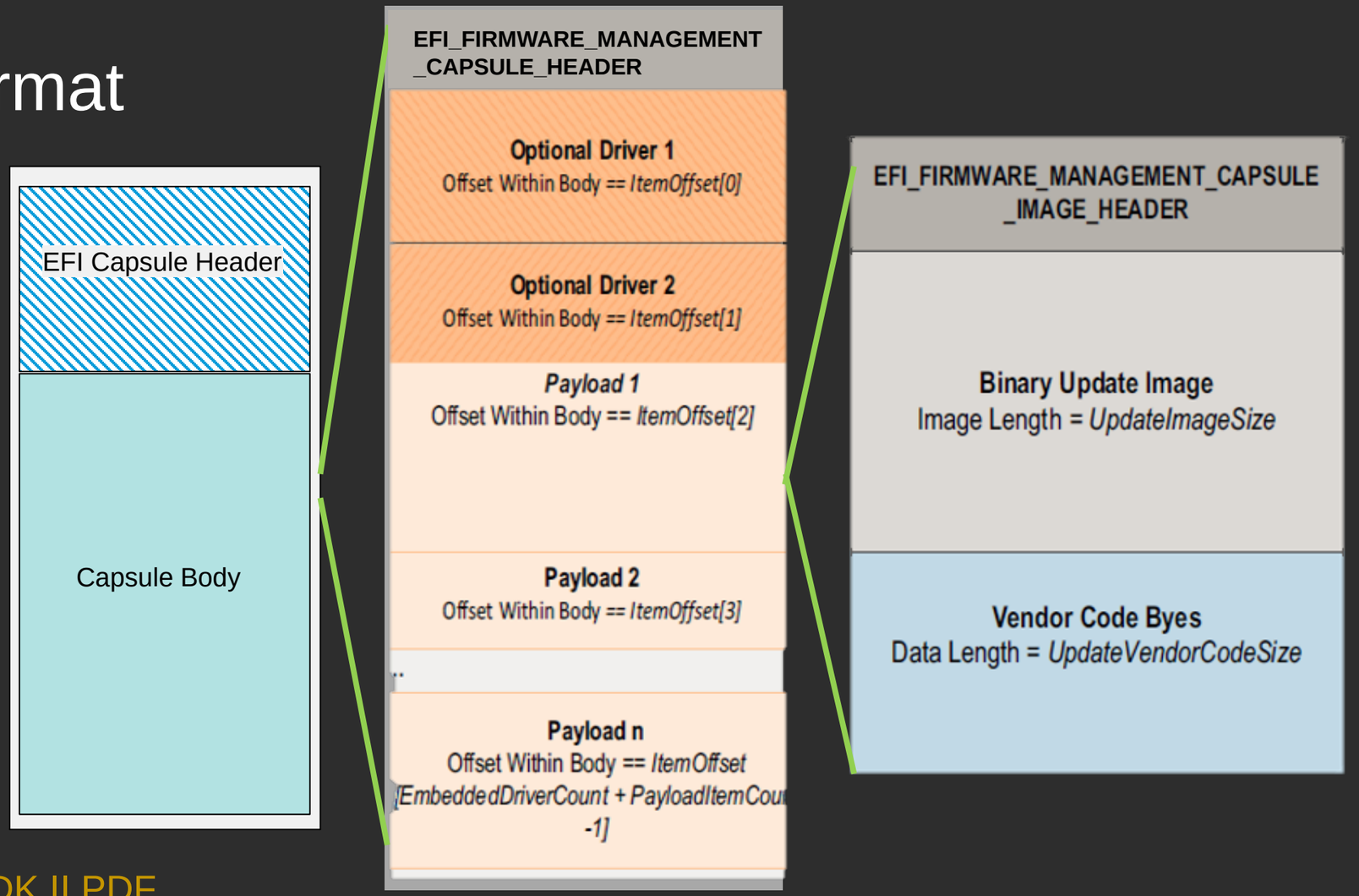
UEFI Capsule Update – Firmware Management Protocol (FMP)

FMP capsule image format

- Update FMP drivers
- FMP payloads

  binary update image and optional vendor code

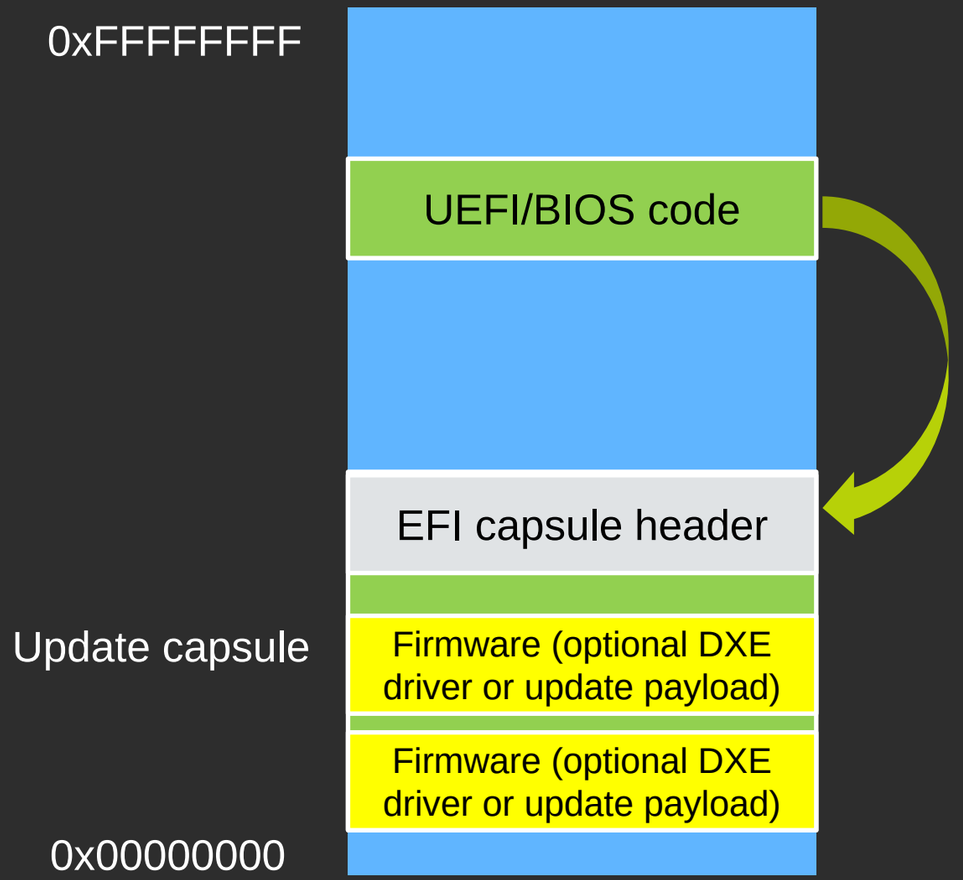
The platform may consume a FMP protocol to update the firmware image



Source: Capsule Update & Recovery EDK II PDF

UEFI Firmware Secure “Capsule” Update

Capsule update is a runtime service used to update UEFI FW

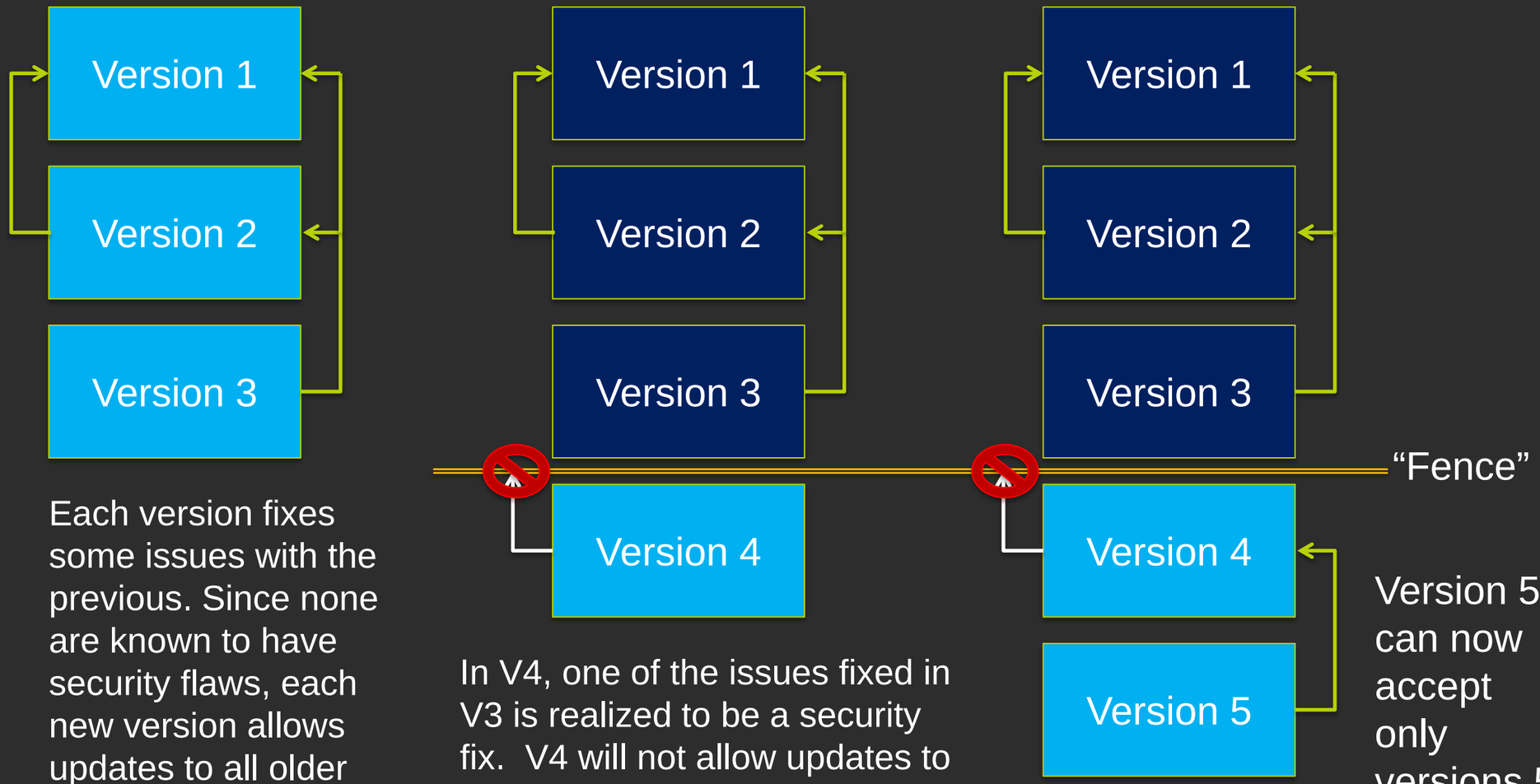


1. Update is initiated by update application/OS run-time
2. Update application stores update “capsule” in DRAM or HDD on ESP (e.g. **`\EFI\CapsuleUpdate`**)
3. Upon reboot or S3 resume, FW finds and parses update capsule
4. After FW verifies digital signature of the capsule, FW writes new BIOS FV(s) to SPI flash memory

Source: UEFI Spec Version 2.4 Facilitates Secure Update UEFI Summerfest – July 15-19, 2013

ESP = EFI System Partition

Firmware Update Rollback Protection



Each version fixes some issues with the previous. Since none are known to have security flaws, each new version allows updates to all older versions.

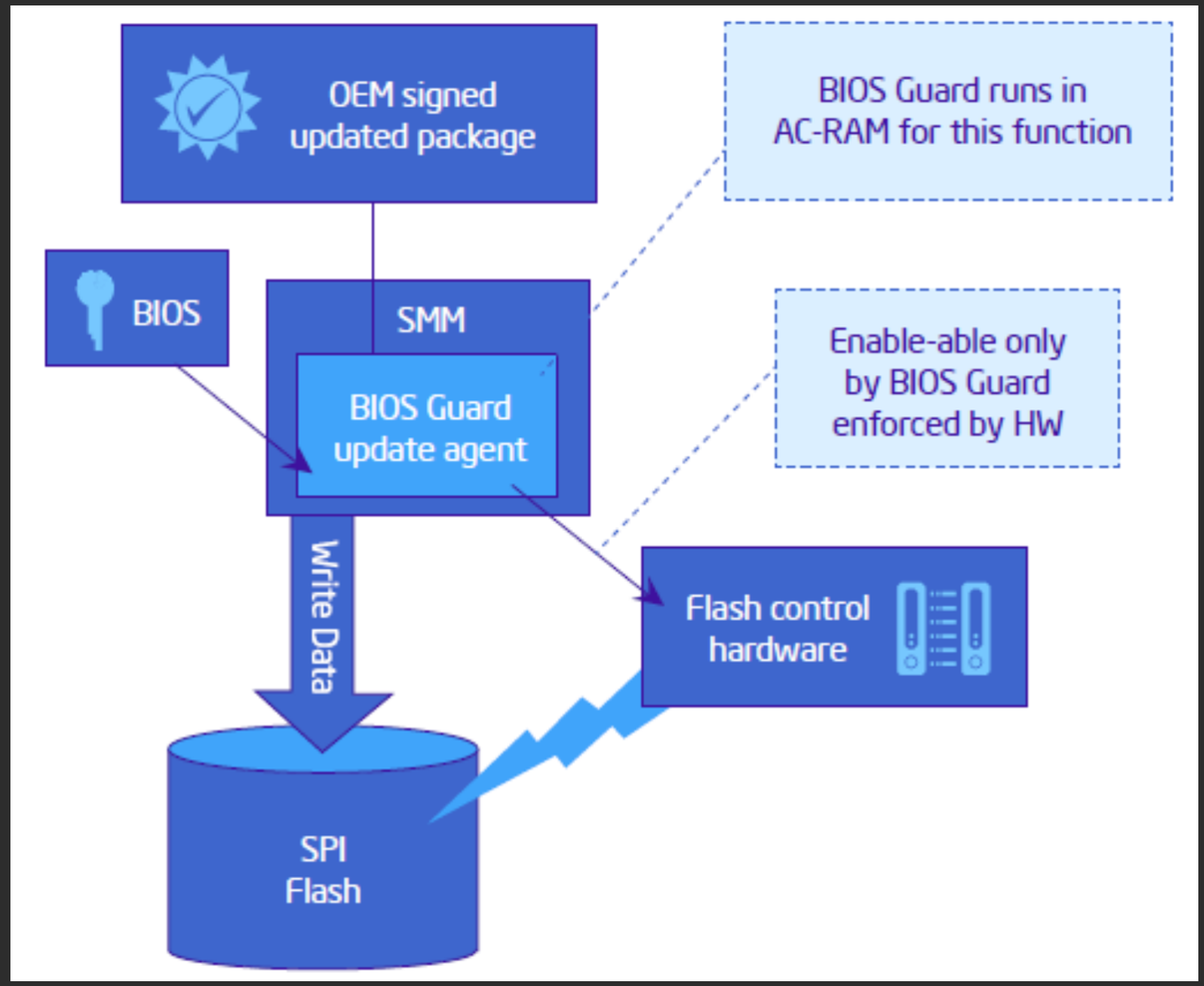
In V4, one of the issues fixed in V3 is realized to be a security fix. V4 will not allow updates to earlier versions, even V3 since it allows update to V2.

Version 5 can now accept only versions 5 and 4.

Hardware based System Firmware Update

BIOS Guard address SMM vulnerabilities by strengthening the update trust boundary

Example: Intel® Platform Protection Technology w/ BIOS Guard



AC-RAM- authenticated code module (ACM) RAM

Source: http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/security-technologies-4th-gen-core-retail-paper.pdf

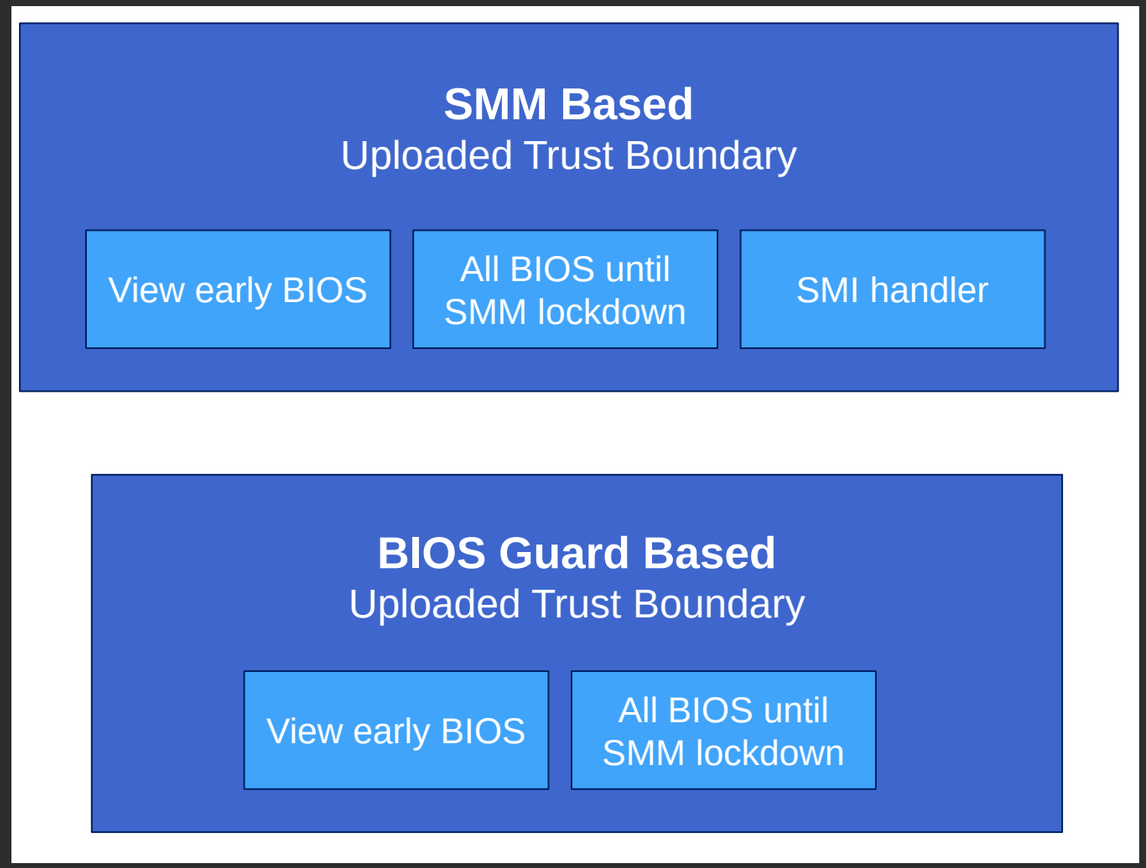
SMM BIOS Update Trust Boundary

- For runtime BIOS Update (e.g. on server platforms), all complex SMI handlers code is in the trust boundary of the firmware update
- Different systems have different SMI handlers which makes it difficult to ensure consistent security level of SMI code across all system and security level of firmware update
- BIOS Guard reduces SMI handler attack surface, using **one** signed BIOS Guard Authenticated Code Module (ACM)
- Platforms enabling BIOS Guard only need to use **one** module for a given processor generation

BIOS Guard Based Firmware Update

- BIOS Guard can update contents of the BIOS region in system SPI flash and EC firmware on EC flash memory
- BIOS Guard module is Authenticated Code Module (ACM) executing in internal processor AC RAM
- When BIOS Guard is enabled, only BIOS Guard module is able to write to system SPI flash memory
- BIOS Guard verifies the signature of a firmware update package signed by a platform manufacturer prior to writing to system SPI flash memory

Trust Boundary with BIOS Guard



EC – Embedded Controller

tianocore

When Is Secure Boot Actually Secure?

When all platform manufacturers…

- protect the UEFI BIOS from programmable SPI writes by malware,
- allow only signed UEFI BIOS updates,
- protect authorized update software,
- correctly program and protect SPI Flash descriptor,
- protect Secure Boot persistent configuration variables in NVRAM,
- implement authenticated variable updates,
- protect variable update API,
- disable Compatibility Support Module (Legacy BIOS),
- don’t allow unsigned legacy Option ROMs,
- configure secure image verification policies,
-

and don’t’ introduce a single bug in all of this, of course.

tianocore

Summary

Platform Firmware Security – Why is it important?

Why is platform firmware Security important	Prevent low level attacks that could “brick” the system
UEFI boot flow with the threat model	Identify where UEFI firmware is vulnerable and define a Threat Model
Security technologies overview	Boot Guard, Secure Boot and NIST Secure Updates provide mitigations to some hacking methods

tianocore

TOOLS AND RESOURCES

How to test firmware for security

tianocore

AUTOMATION:
Automated Test Generation Tools

Build a Security Mindset

“Finding BIOS Vulnerabilities with Symbolic Execution and Virtual Platforms”: Excite Link

tianocore

What is Excite?

Goal

- ***Automatically*** detect security issues in UEFI BIOS
  - *Complement other approaches: security code reviews, static analysis, Chipsec …*

Approach

- Apply open source **fuzzing** & **symbolic execution** tools to Intel Architecture ***binaries***

Versions

- **Excite :** Symbolic Execution using CRETE and Simics
- **Excite DBI** (Dynamic Binary Instrumentation): Fuzzing using AFL and Intel Pin Tool

AFL - American Fuzzy Lop (fuzzer)
Pin – Intel tool to determine trace
CRETE - https://github.com/SVL-PSU/crete-dev

tianocore

What is Fuzzing?

Technique of providing variety of invalid, unexpected, or random data as inputs to target code to try to make it misbehave

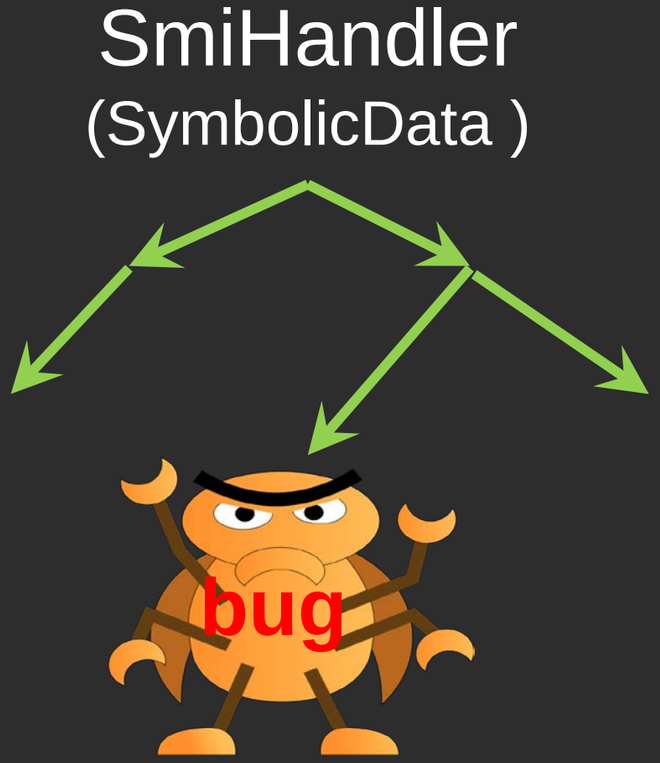
- Some inputs are non-intuitive, harder for developers to discover

Why fuzz UEFI?

- Find security issues early
  - Complement manual code reviews
  - Re-run as software evolves
  - New vulnerability detection capability

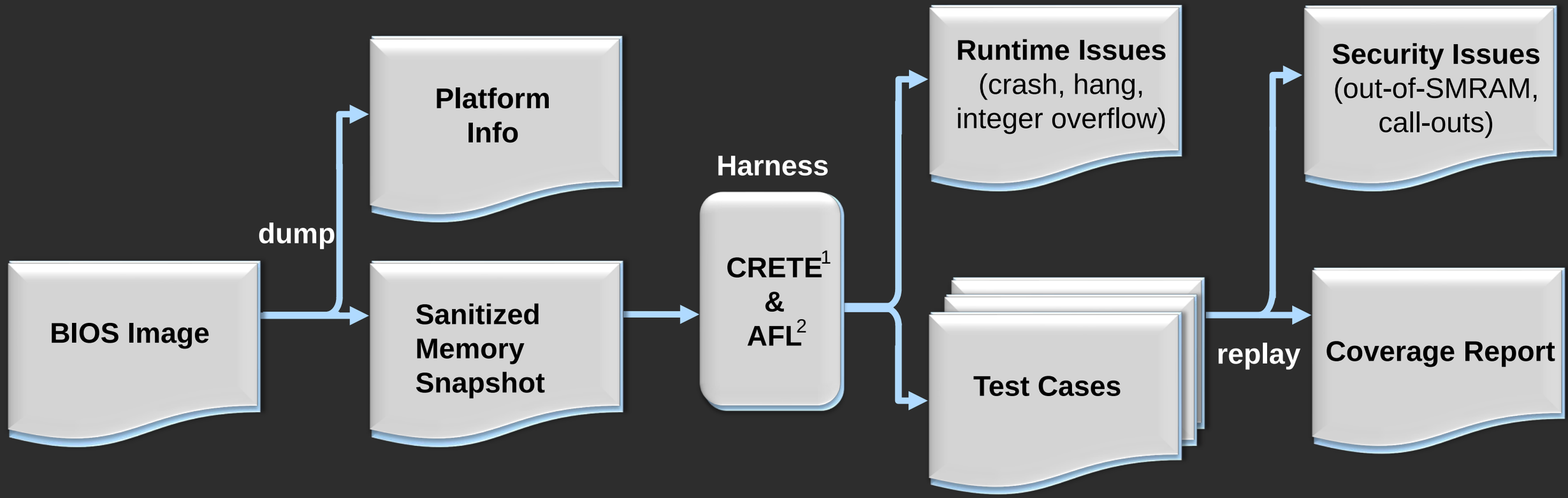
Symbolic Execution (S²E)– SMM Example

- Searching for SMM security vulnerabilities
- Symbolic execution generates test cases that cover the computation tree of the handler and induce vulnerabilities.
- Those test cases are replayed in Simics which detects illegal accesses and SMM callouts.
  (e.g. read memory outside SMRAM)



Symbolic execution for BIOS security
https://www.usenix.org/system/files/conference/woot15/woot15-paper-bazhaniuk.pdf

Excite – Common Flow



[1] Selective Symbolic Execution (S²E | CRETE)
[2] American Fuzzy Lop (fuzzier)

tianocore

Excite Benefits

Extend fuzzing beyond interface level

- UEFI code (SMM, STM, UEFI PE modules) accessible to AFL/CRETE, Pin, gdb.
- Record/analyze code flow, call trace, memory access data
- Individual functions accessible, new inputs created, can test different flows

Quantify code coverage reporting

STM -SMI Transfer Monitor
https://firmware.intel.com/content/smi-transfer-monitor-stm

tianocore

KNOWN ISSUE:

Tools for Testing Known Issue

tianocore

CHIPSEC

Raising the Bar for Platform Security

New Attacks

Test Modules for OEMs/IBVs

End-user Risk

Empowering End-Users to Make a Risk Decision

Source: CHIPSEC-csw14 PDF

tianocore

What is CHIPSEC?

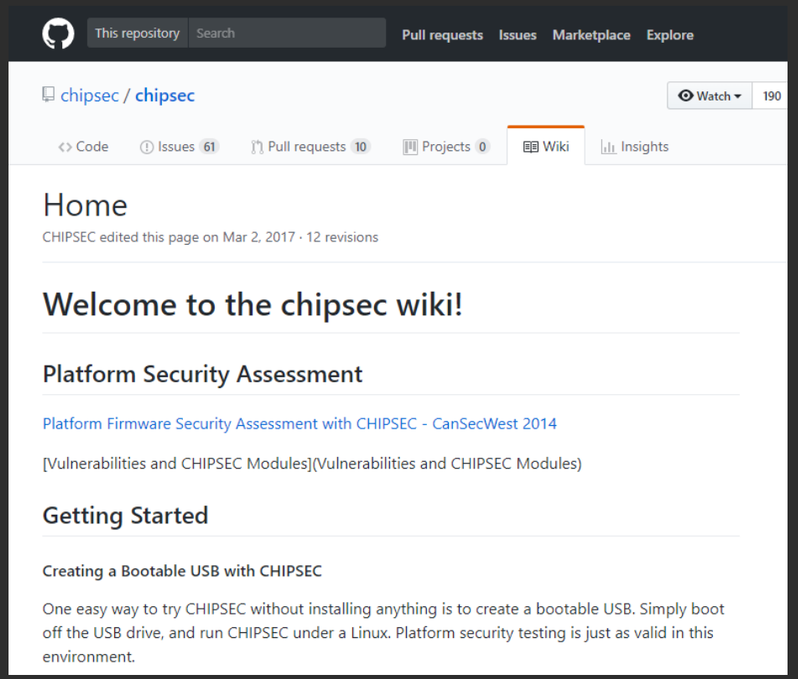
CHIPSEC
Platform Security
Assessment Framework

Framework for Platform Security Assessment

- Tests for known security issues
- Tools for investigation of platform properties

Open Source (GPL v2 License)

- Released in 2014 – https://github.com/chipsec/chipsec
- Currently active community

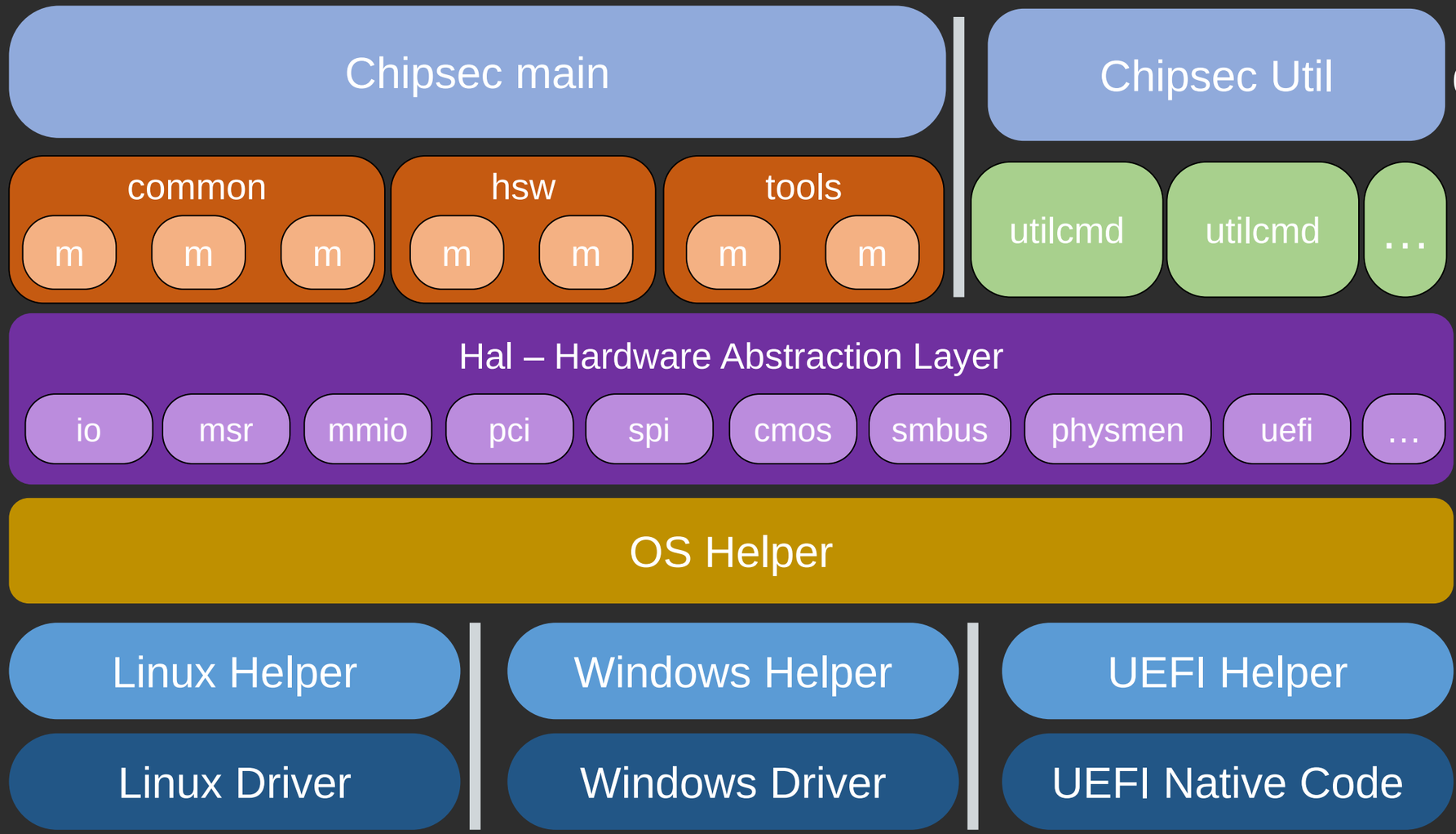


- Top referring sites: Google, Github, pcworld.com, community.spiceworks.com, anonhq.com, Twitter, securingtomorrow.mcafee.com, intelsecurity.com, cylance.com,

tianocore

Chipsec Framework Overview

CHIPSEC



Source: https://cansecwest.com/slides/2014/Platform%20Firmware%20Security%20Assessment%20wCHIPSEC-csw14-final.pdf

tianocore

Example Chipsec Check - SPI Controller LOCK

CHIPSEC
Platform Security
Assessment Framework

- `FLOCKDN` – Reg that Locks critical SPI controller registers
  - Flash Region Access Permissions
  - Secondary Flash Region Access Permissions
  - Flash Protected Range Registers
  - Other Settings
- Without setting this bit some SPI protections may be modified or bypassed
- Chipsec provides the `common.spi_lock` test to verify this setting

FLOCKDN – Flash Configuration Lock down Reg

tianocore

Sample Issue Discovered

Is BIOS correctly protected?

CHIPSEC
Platform Security Assessment Framework

```
[*] running module: chipsec.modules.common.spi_lock
[x][ ================================================================
[x][ Module: SPI Flash Controller Configuration Lock
[x][ ================================================================
[*] HSFS = 0x6008 << Hardware Sequencing Flash Status Register (SPIBAR + 0x4)
    [00] FDONE              = 0 << Flash Cycle Done
    [01] FCERR              = 0 << Flash Cycle Error
    [02] AEL                = 0 << Access Error Log
    [03] BERASE             = 1 << Block/Sector Erase Size
    [05] SCIP               = 0 << SPI cycle in progress
    [13] FDOPSS             = 1 << Flash Descriptor Override Pin-Strap Status
    [14] FDV                = 1 << Flash Descriptor Valid
    [15] FLOCKDN            = 0 << Flash Configuration Lock-Down
```

```
[-] FAILED: SPI Flash Controller configuration is not locked
```

tianocore

Known Threats and Chipsec modules

CHIPSEC

Issue	CHIPSEC Module	References
SMRAM Locking	common.smm	CanSecWest 2006
BIOS Keyboard Buffer Sanitization	common.bios_kbrd_buffer	DEFCON 16
SMRR Configuration	common.smrr	ITL 2009, CanSecWest 2009
BIOS Protection	common.bios_wp	BlackHat USA 2009, CanSecWest 2013, Black Hat 2013, NoSuchCon 2013
SPI Controller Locking	common.spi_lock	Flashrom, Copernicus
BIOS Interface Locking	common.bios_ts	PoC 2007

tianocore

Known Threats and Chipsec modules

CHIPSEC

Issue	CHIPSEC Module	References
BIOS Interface Locking	common.bios_ts	PoC 2007
Secure Boot variables with keys and configuration are protected	common.secureboot.variables	UEFI 2.4 Spec , All Your Boot Are Belong To Us (here & here)
Memory remapping attack	remap	Preventing and Detecting Xen Hypervisor Subversions
DMA attack against SMRAM	smm_dma	Programmed I/O accesses: a threat to VMM?, System Management Mode Design and Security Issues
SMI suppression attack	common.bios_smi	Setup for Failure: Defeating Secure Boot

Known Threats and Chipsec modules

Issue	CHIPSEC Module	References
SMI suppression attack	common.bios_smi	Setup for Failure: Defeating Secure Boot
ETC . . .		(See https://github.com/chipsec/chipsec)

tianocore

CHIPSEC Modules

CHIPSEC
Platform Security
Assessment Framework

Modules encapsulate the main functionality of CHIPSEC:

1. Tests for known vulnerabilities in firmware
2. Tests for insufficient or incorrectly configured hardware protections
3. Hardware/firmware-level security tools
   - Fuzzing tools for firmware interfaces/formats
   - Manual security checkers (e.g. TE checker, DMA dumper)
   - Reside in `modules/tools` directory are not launched automatically (only through `-m` command-line option)
   - PoC exploit modules demonstrating vulnerabilities

CHIPSEC Modules

- All modules reside in `chipsec/modules` directory
- Modules can be specific to one or more platforms or common for all supported platforms
  - Modules in `modules/<platform_code>` directory will only be executed on `<platform_code>` platform
  - Modules in `modules/common` directory will always be executed
- Modules can implement `is_supported` function which can further check for supported platforms, OS environments (legacy vs UEFI boot), etc.

tianocore

Shift-Left Firmware Security Testing

- Host-based Firmware Analyzer (HBFA)

tianocore

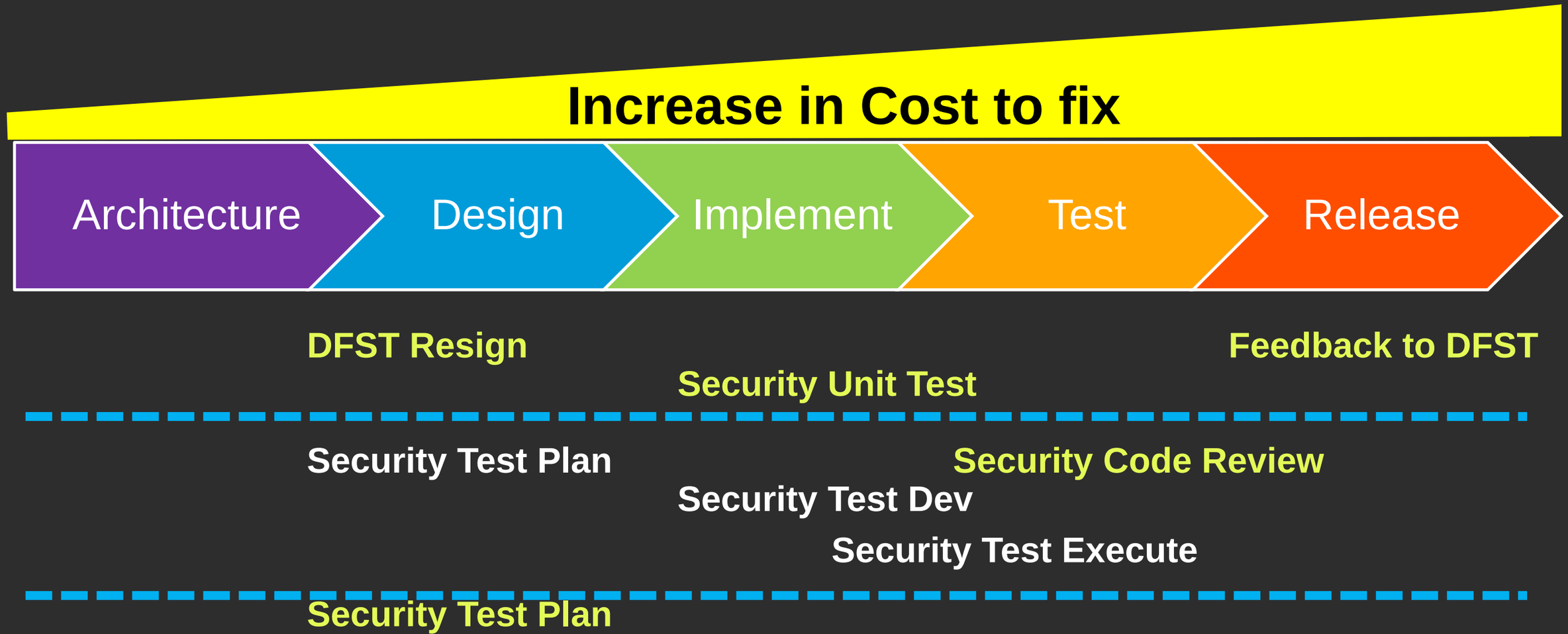
Common Tools for Firmware Security Testing

CHIPSEC: open source framework for analyzing the security of platform firmware and hardware configuration at runtime, based on the Unified Extensible Firmware Interface (UEFI) specification.

Symbolic Execution and Virtual Platforms: Intel's Excite project uses a combination of symbolic execution, fuzzing, and concrete testing to find vulnerabilities in firmware running on Wind River* Simics* virtual platforms.

Great tools… but they're based on integration testing, so issues are more expensive to detect and mitigate. Can we improve testing before integration?

Shift Left – Design for Security Test (DFST)



How can we improve security test efficiency?

Design for Security Test (DFST)

Engineers should recognize that ***reducing risk is not an impossible task, even under financial and time constrains.***

All it takes in many cases is ***a different perspective on the design problem.***

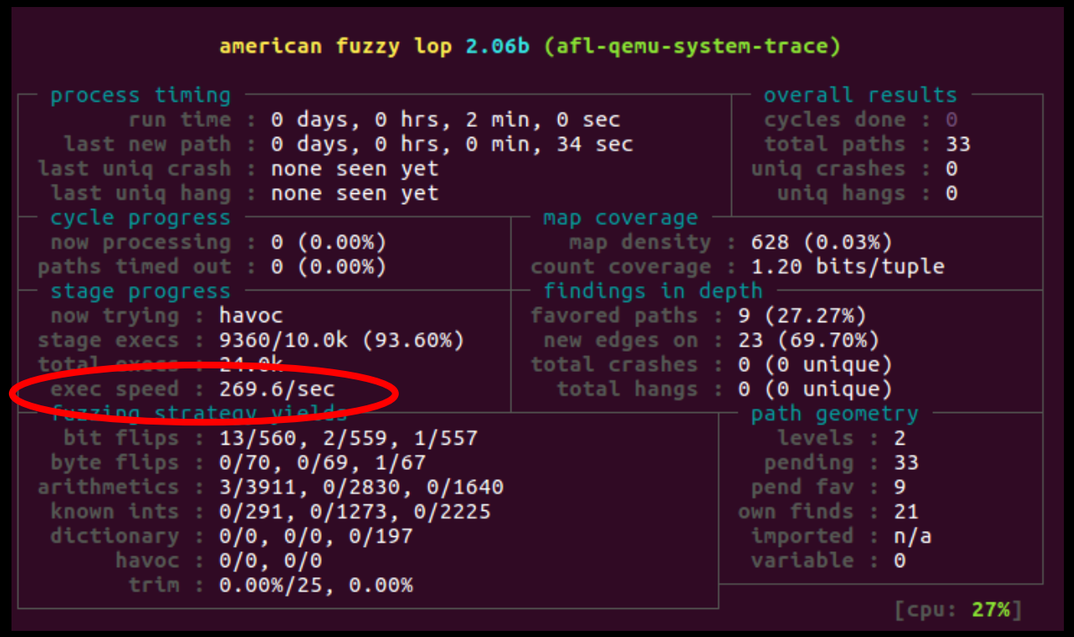
-- Mike Martin & Roland Schinizinger

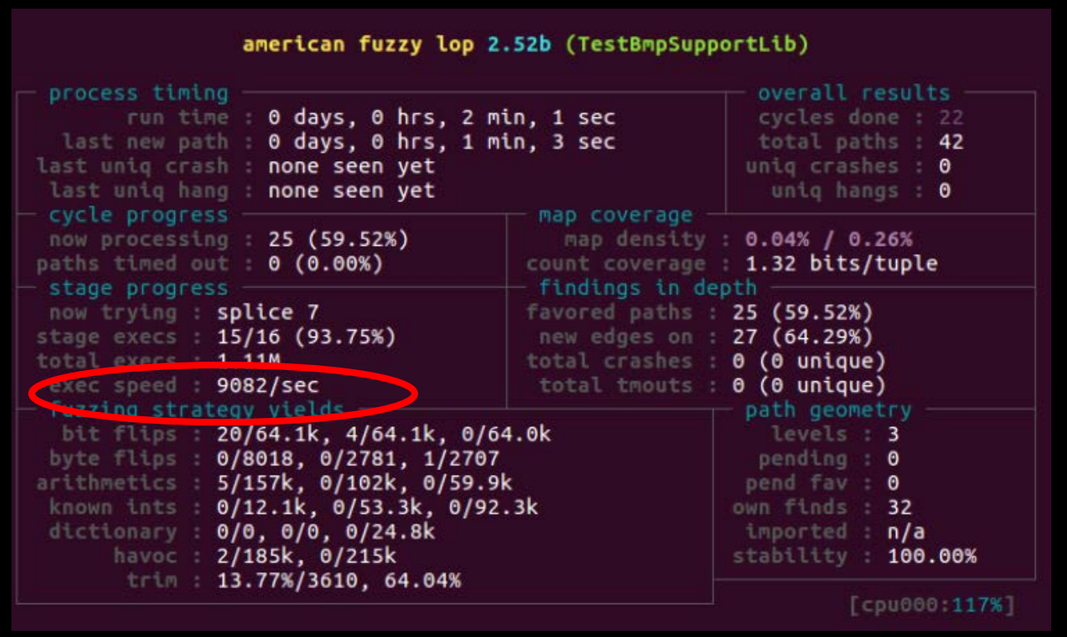
tianocore

Host-based Firmware Analyzer

Test for BIOS code under OS: Pure software logic

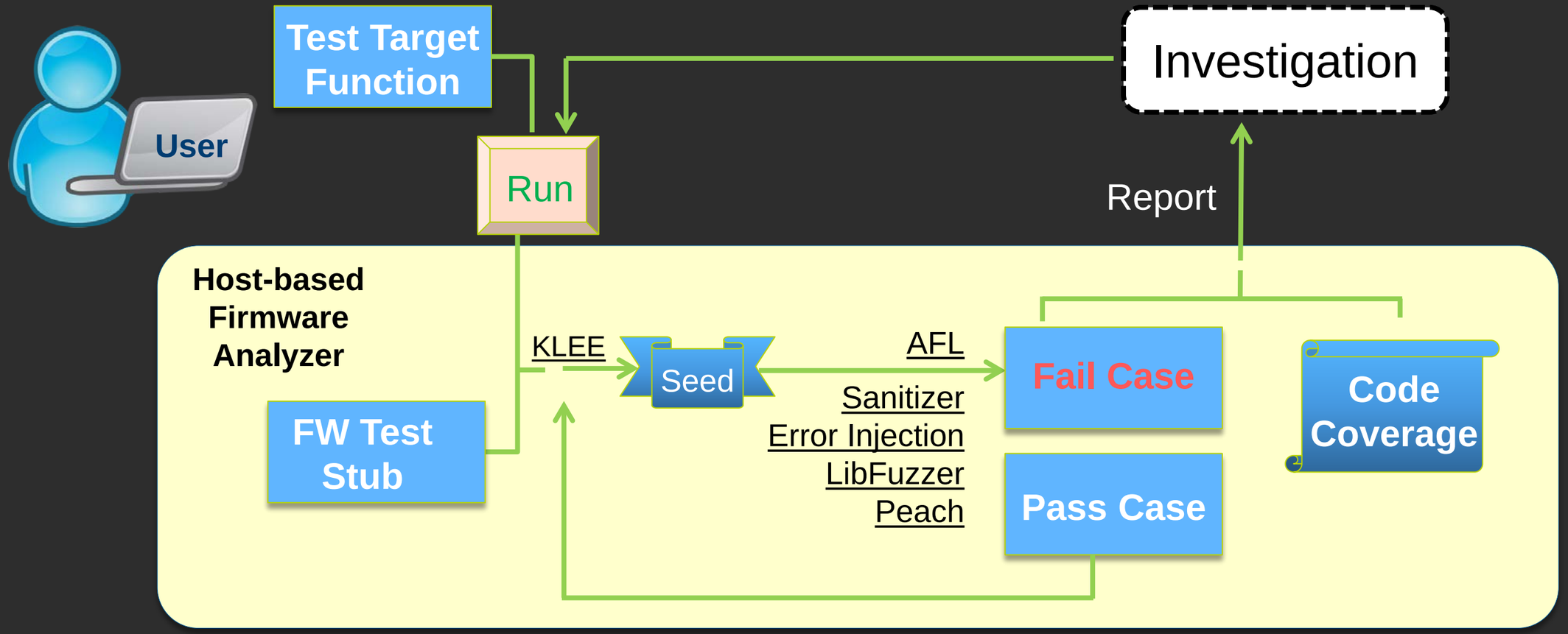
Benefit: 30X faster than UEFI based running env(OVMF).





tianocore

Host-based Firmware Analyzer Architecture



Implementing testing UEFI Firmware under OS environment

tianocore

Host-based Firmware Analyzer

OS based environment utilizing best-in-class test tools

GUI and command-line interfaces

Fuzzing testing (AFL, libFuzzer, Peach)

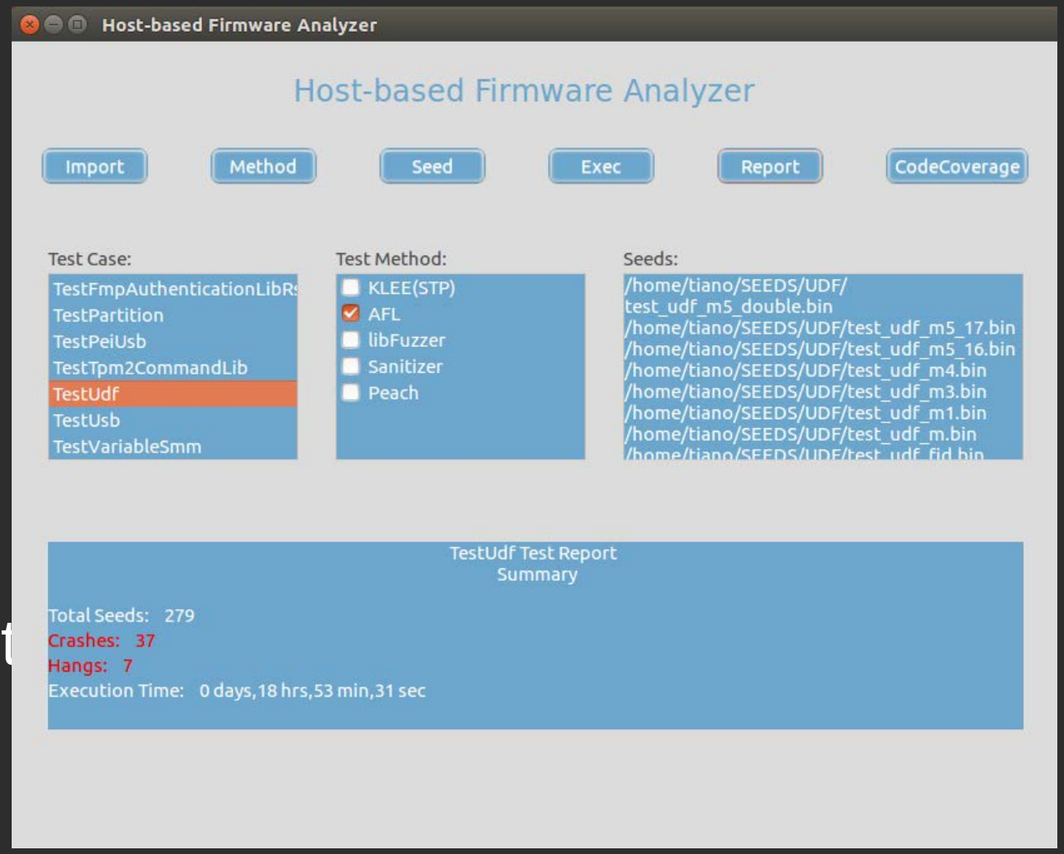
Symbolic execution (KLEE/STP)

Address Sanitizer & Code Coverage

Automated unit test execution

Instrumentation methods for fault injection and t

Database of unit test cases



tianocore

HBFA - Summary

Host Based Firmware Analyzer (HBFA)

- Isolate test environment – run test under OS (like OS application)
- Integrate with OS based fuzz/Symbolic tool, such as AFL, KLEE, Peach.
- Provide an infrastructure to build Pure Software Logic function of BIOS.

Benefit

- Easy for Secure Fuzzy Test.
- Easy for test automation and case reuse.
- Easy for Code Coverage , debugging with OS Debugger tool

tianocore

Expectation

Developer :

- Easily enable fuzz test in unit test Phase.

Benefit:

- Reduce the total cost of a project.
- Help to detect and fix security issue quickly
- Reduce the effort for security unit test
- QA can focus on other security test. (eg, system level or cost much time to set up env)

Release on tianocore.org wiki page hbfa

EDK II Security Common Paper List

Industry Standards

NIST: https://csrc.nist.gov/publications/sp800
TCG: http://trusted.computinggroup.com/
SideChannel: Link-software-IDZ

General

- uefi.org/Intel-UEFI-ThreatModel.pdf
- tianocore.org/A_Tour_Beyond_BIOS_Security_Design_Guide_in_EDK_II.pdf
- A-tour-beyond-bios-mitigate-buffer-overflow-in-UEFI-Gitbook
- uefi.org/Intel_An Introduction to Platform.pdf

Memory Protection

A-tour-beyond-bios Memory Protection in UEFI BIOS – gitbook

SSM Communication

- uefi.org/SMM Protection in EDKII PDF
- tianocore.org/A_Tour_Beyond_BIOS Secure SMM Communication PDF

DMA

Using_IOMMU_for_DMA_Protection_in_UEFI.pdf

Capsule / Recovery – tianocore.org/A_Tour_Beyond_BIOS Capsule Update and Recovery in EDK II.pdf

Variable

- tianocore.org/A_Tour_Beyond_BIOS Implementing UEFI Authenticated Variables in SMM w/ EDK II.pdf
- Variable Confidentiality PDF – with Confidentiality Annex pdf

S3 – tianocore.org/A_Tour_Beyond_BIOS Implementing S3 resume w/ EDK II .pdf

TPM2 - A_Tour_Beyond_BIOS Implementing TPM2 Support in EDK II .pdf

Profile - tianocore.org/A_Tour_Beyond_BIOS Implementing Profiling in EDK II .pdf

STM/VMM

- A_Tour_Beyond_BIOS Launching STM to Monitor SMM in EDK II .pdf
- A_Tour_Beyond_BIOS Launching VMM in EDK II.pdf
- A_Tour_Beyond_BIOS Supporting SMM Resource Monitor using the EDK II .pdf

Standalone MM - A_Tour_Beyond_BIOS Launching Standalone SMM Drivers in PEI using EDK II .pdf

tianocore

Summary

Platform Firmware Security – Why is it important?

Why is platform firmware Security important	Prevent low level attacks that could “brick” the system
UEFI boot flow with the threat model	Identify where UEFI firmware is vulnerable and define a Threat Model
Security technologies overview	Boot Guard, Secure Boot and NIST Secure Updates provide mitigations to some hacking methods
Tools and resources on how to test firmware for security	Ongoing tools to validate security mechanism

tianocore

SUMMARY

- Why is platform firmware Security important
- UEFI boot flow with the threat model
- Security technologies overview
- Tools and resources on how to test firmware for security

tianocore

Questions?

Return to Main Training Page

Return to Training Table of contents for next presentation link

tianocore

ACKNOWLEDGEMENTS