presented by

An Introduction to Platform Security

Spring 2018 UEFI Seminar and Plugfest
March 26-30, 2018
Presented by Brent Holtsclaw and John Loucaides (Intel)

Legal Notice

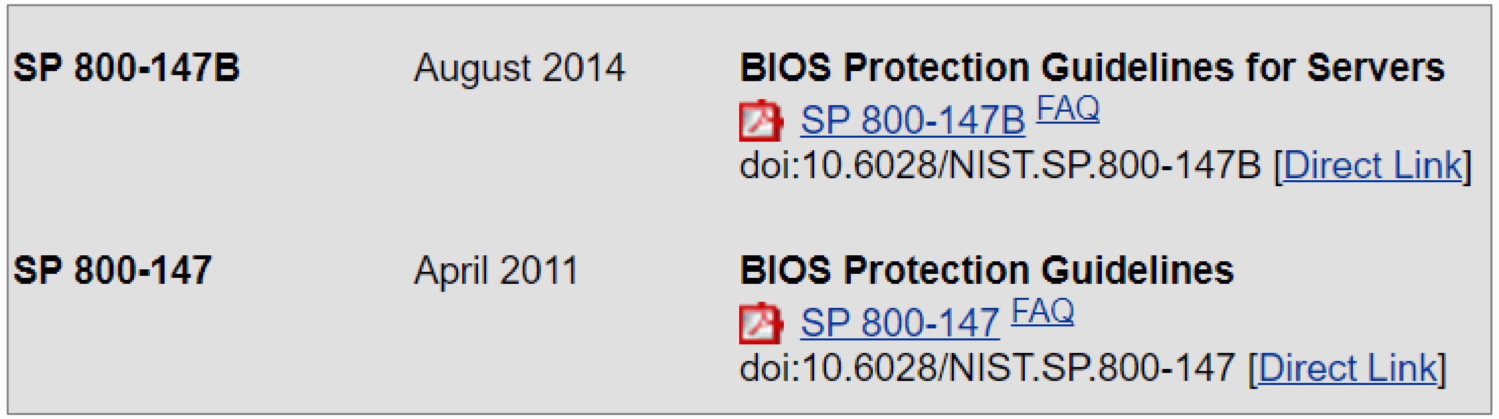
Building a Threat Model…

Note: Contents are meant as examples. This does not represent an exhaustive analysis.

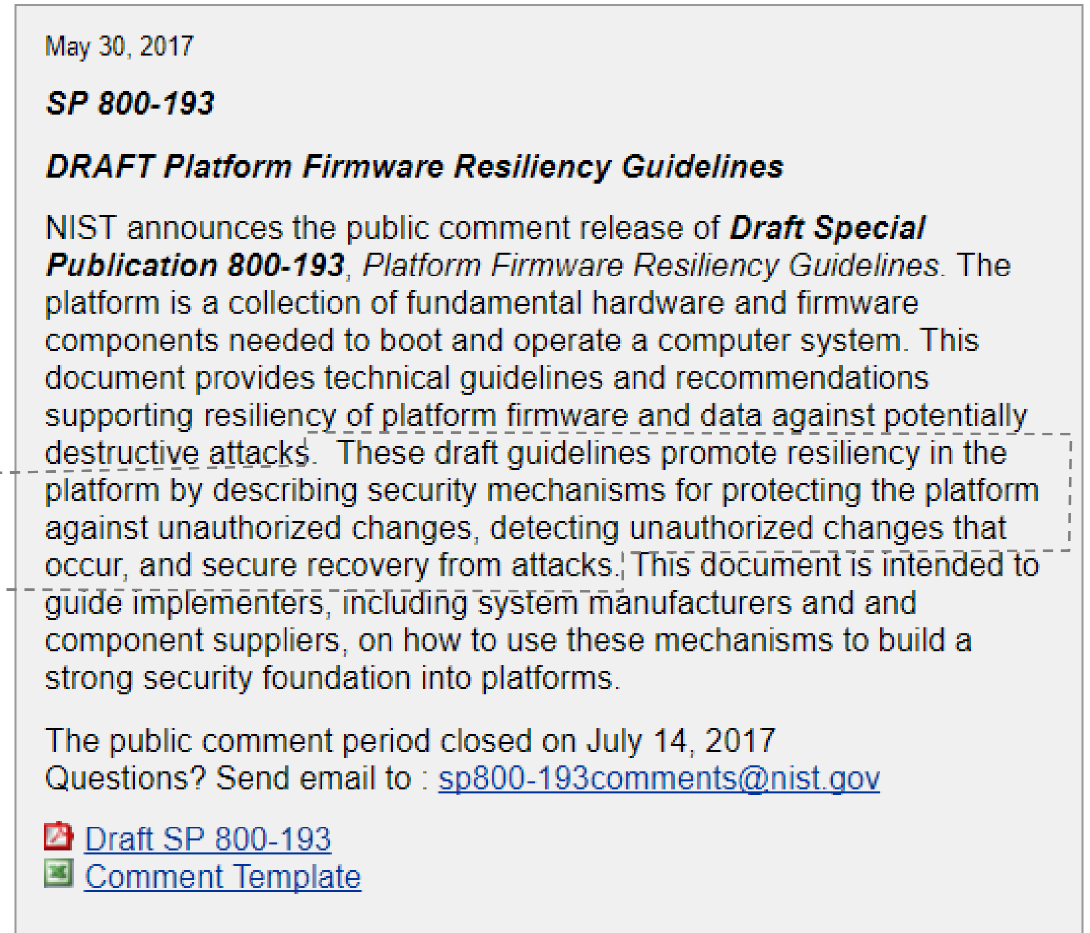
Why Attack Firmware?

- **Persistent Compromise**
  - Update firmware image with malicious content
- **Stealthy Compromise**
  - System Management Mode (SMM) code injection
- **Bypass of Security Features**
  - Hypervisor / Virtual Machine Monitor (VMM) Bypass
- **Denial of Service**
  - Corrupt/Delete critical configuration settings

NIST National Institute of Standards and Technology
Information Technology Laboratory
SEARCH:
Search
CONTACT
SITE MAP
Computer Security Division CSD
Computer Security Resource Center CSRC



SP 800-147B	August 2014	**BIOS Protection Guidelines for Servers** SP 800-147B FAQ doi:10.6028/NIST.SP.800-147B [Direct Link]
SP 800-147	April 2011	**BIOS Protection Guidelines** SP 800-147 FAQ doi:10.6028/NIST.SP.800-147 [Direct Link]



May 30, 2017

SP 800-193

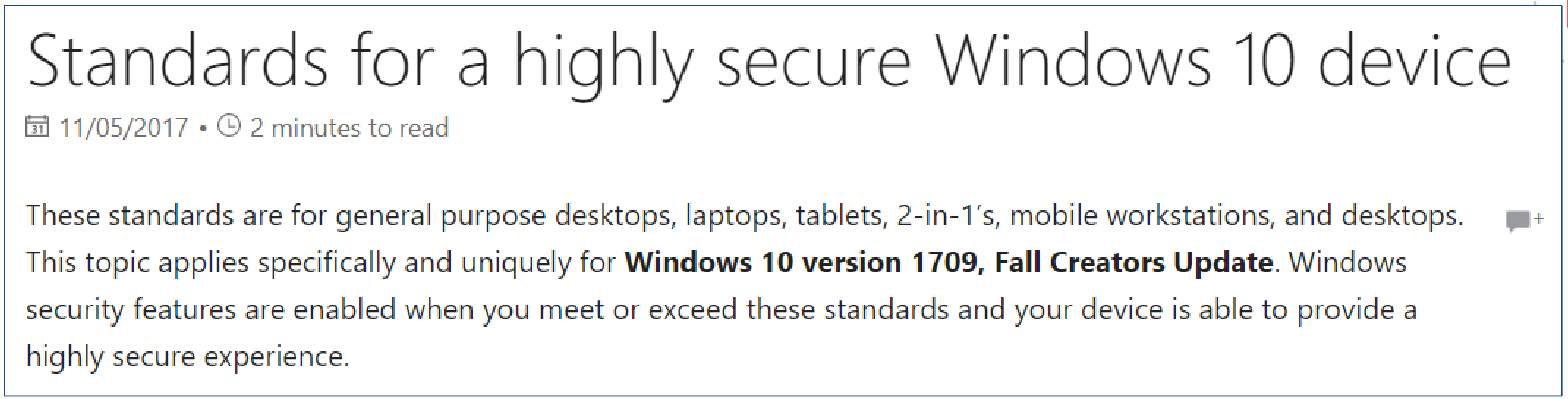
DRAFT Platform Firmware Resiliency Guidelines

NIST announces the public comment release of ***Draft Special Publication 800-193***, *Platform Firmware Resiliency Guidelines*. The platform is a collection of fundamental hardware and firmware components needed to boot and operate a computer system. This document provides technical guidelines and recommendations supporting resiliency of platform firmware and data against potentially destructive attacks. These draft guidelines promote resiliency in the platform by describing security mechanisms for protecting the platform against unauthorized changes, detecting unauthorized changes that occur, and secure recovery from attacks. This document is intended to guide implementers, including system manufacturers and and component suppliers, on how to use these mechanisms to build a strong security foundation into platforms.

The public comment period closed on July 14, 2017
Questions? Send email to : sp800-193comments@nist.gov

Draft SP 800-193
Comment Template

"These draft guidelines promote resiliency in the platform by describing security mechanisms for protecting the platform against unauthorized changes, detecting unauthorized changes that occur, and secure recovery from attacks."



Standards for a highly secure Windows 10 device

11/05/2017 • 2 minutes to read

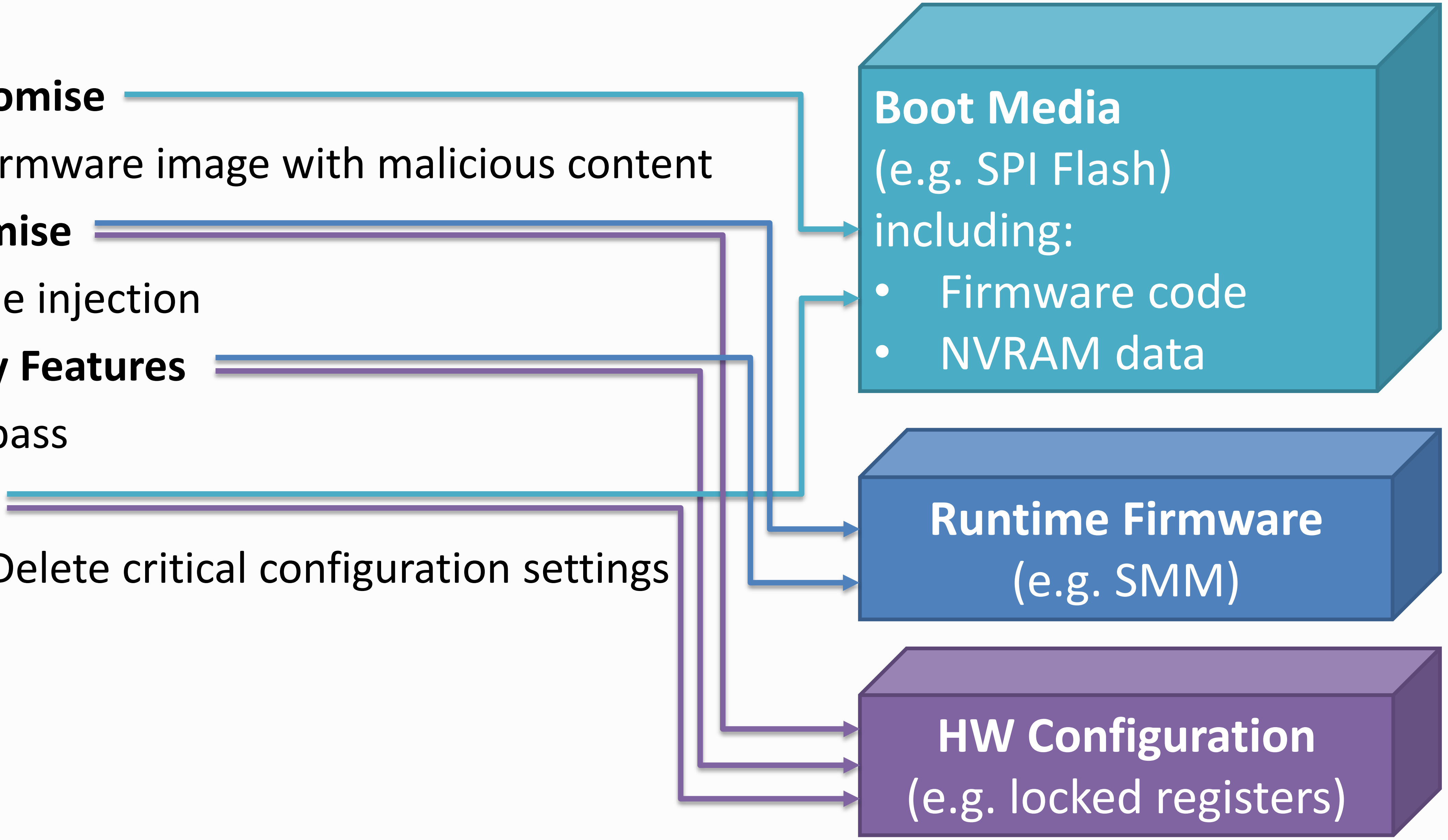
These standards are for general purpose desktops, laptops, tablets, 2-in-1's, mobile workstations, and desktops. This topic applies specifically and uniquely for **Windows 10 version 1709, Fall Creators Update**. Windows security features are enabled when you meet or exceed these standards and your device is able to provide a highly secure experience.

https://docs.microsoft.com/en-us/windows-hardware/design/device-experiences/oem-highly-secure

Firmware-related features
Systems must have firmware that implements Unified Extension Firmware Interface (UEFI) version 2.4+ Systems must have firmware that implements UEFI Class 2 or UEFI Class 3 System's firmware must support UEFI Secure Boot and must have UEFI Secure Boot enabled by default System's firmware must implement Secure MOR revision 2 Systems must support the Windows* UEFI Firmware Capsule Update specification

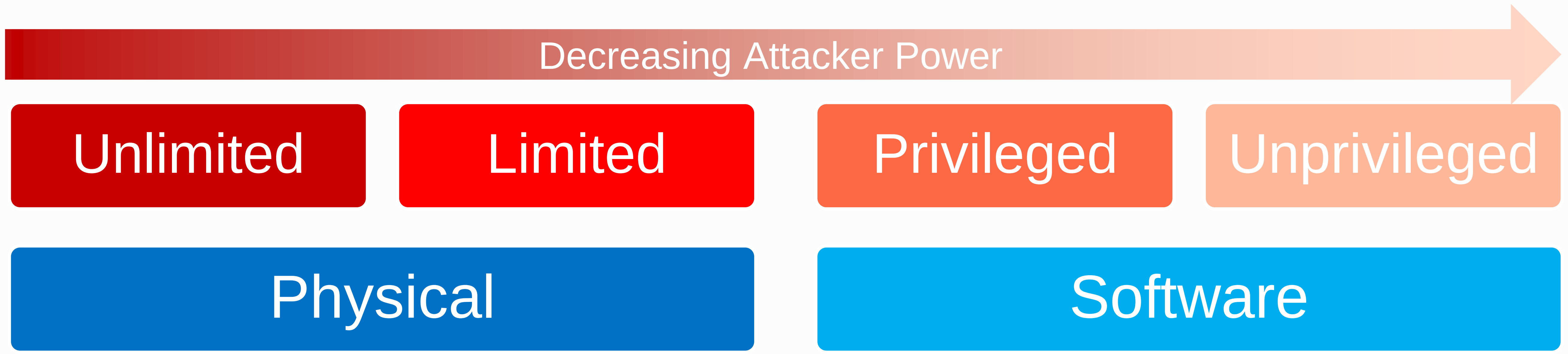
Attacks and Platform Assets

- **Persistent Compromise**
  - Update firmware image with malicious content
- **Stealthy Compromise**
  - SMM code injection
- **Bypass of Security Features**
  - VMM Bypass
- **Denial of Service**
  - Corrupt/Delete critical configuration settings



These are examples. Not an exhaustive list.

Classes of Attacker

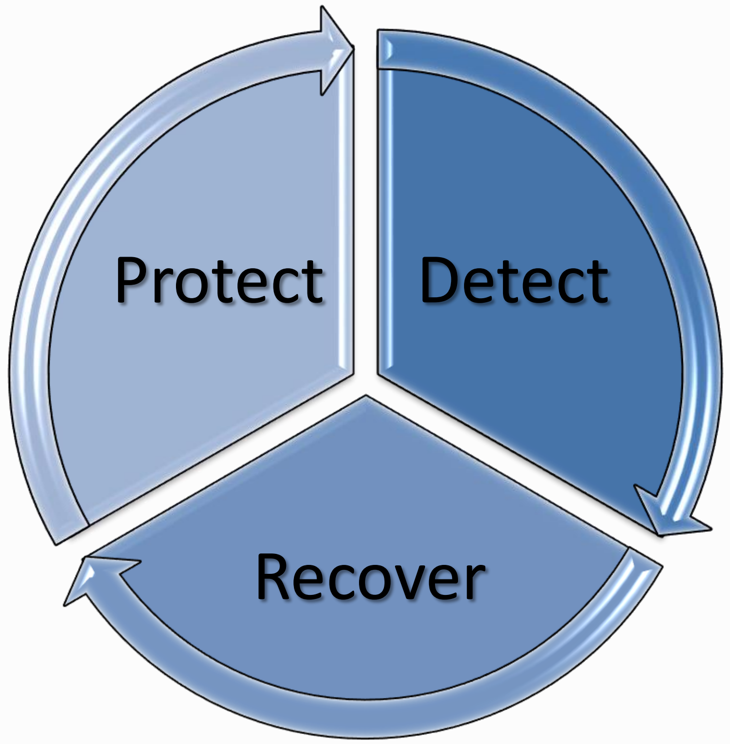


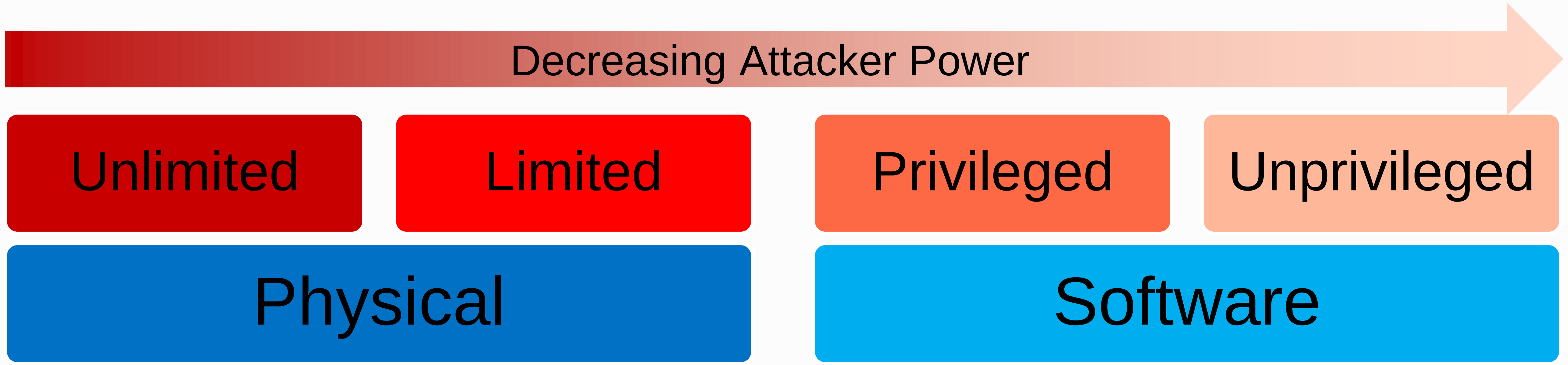
Resilient Defense

Boot Media

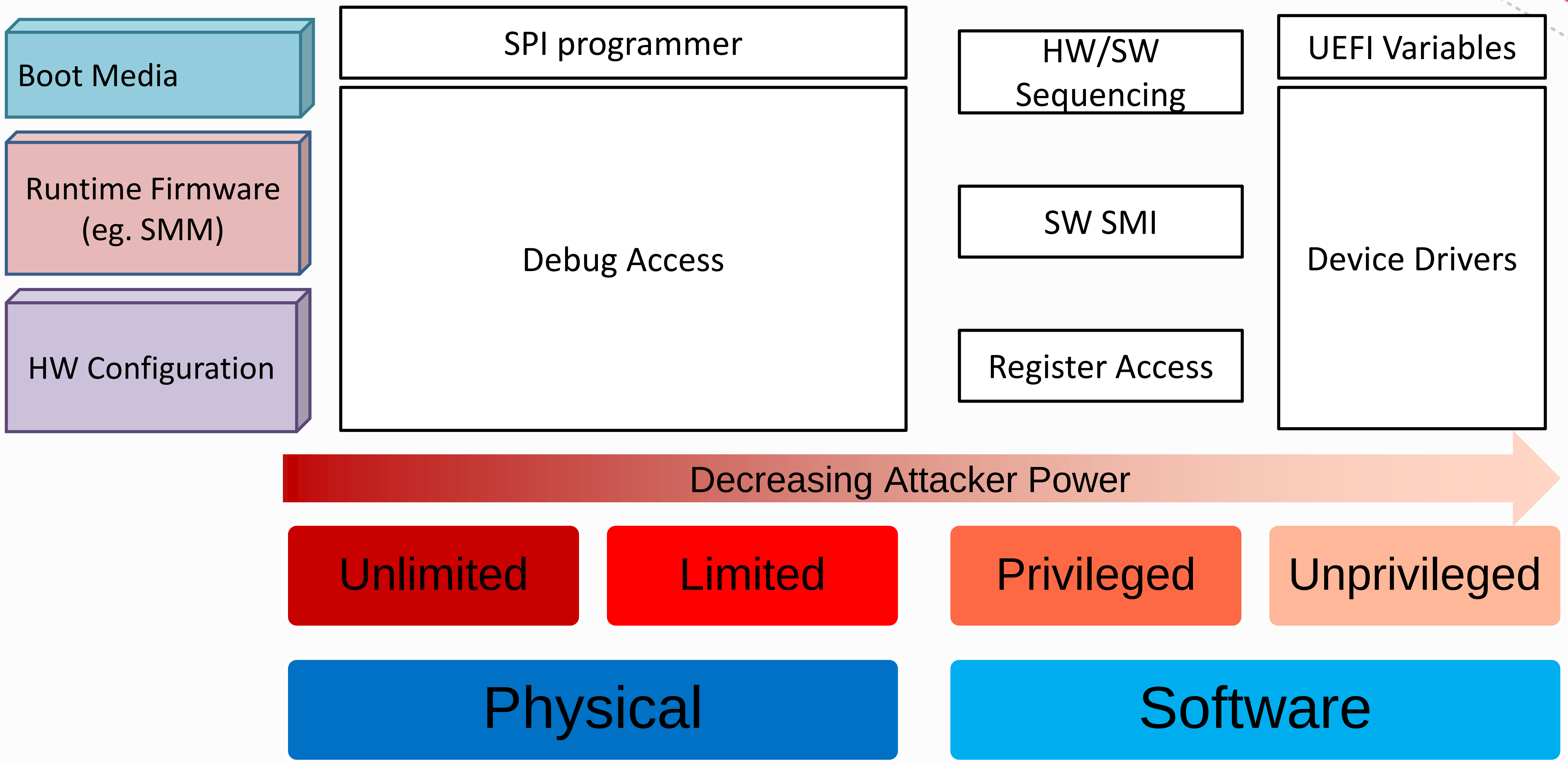
Runtime Firmware (eg. SMM)

HW Configuration



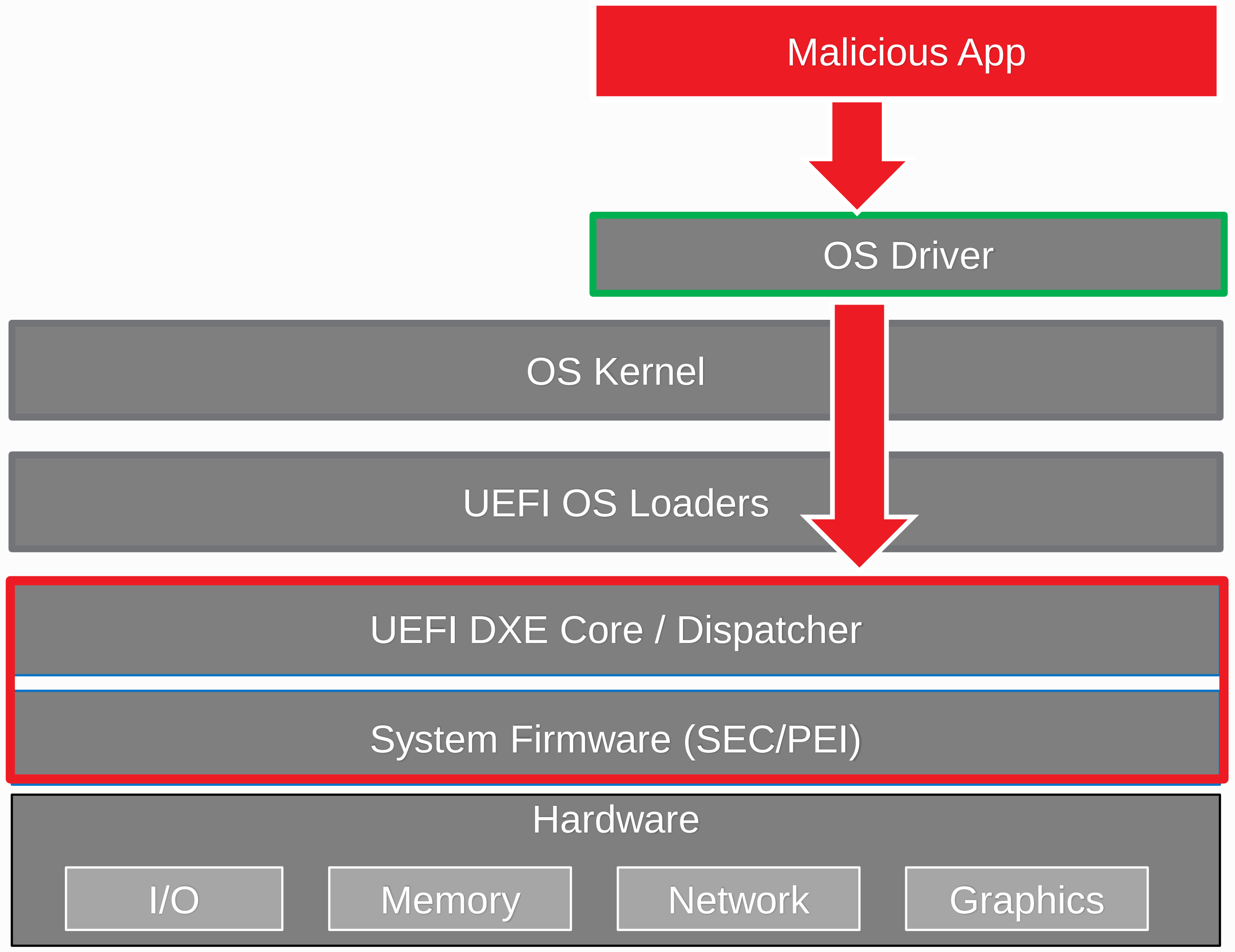


Attack Surface (Interfaces to access/attack assets)



Note: Contents are meant as examples. This does not represent an exhaustive analysis.

Do Firmware Attacks Require Kernel Privileges?



A matter of finding legitimate signed kernel driver which can be used on behalf of user-mode exploit as a *confused deputy*.

RWEverything driver signed for Windows 64bit versions (co-discovered with researchers from MITRE)

Securing the Platform

Defending the Boot Media Asset

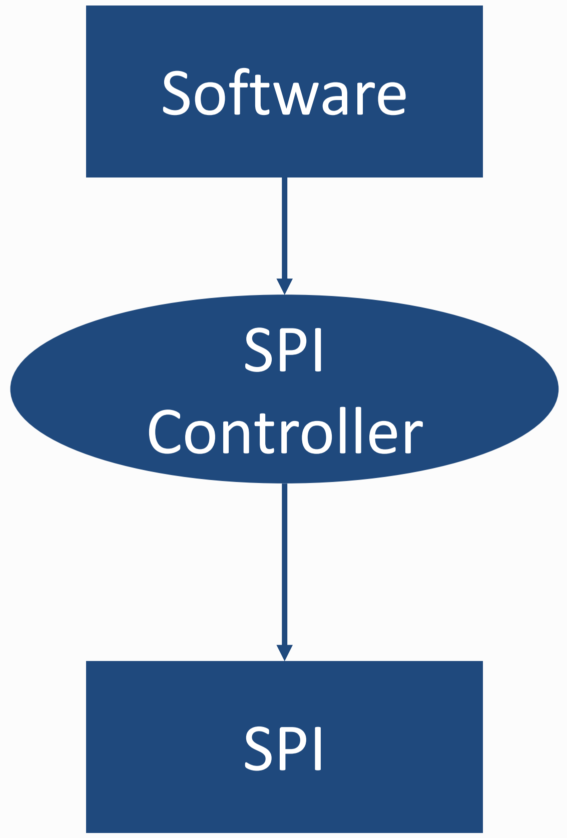
Boot Media Resiliency

Attack	Protect Mechanism	Detect Mechanism	Recover Mechanism
Direct Write to Boot Media (eg. unlocked SPI, Speed Racer, etc.)	• SPI Controller Config • SMM-based Protection • TCB reduction	• UEFI Secure (Verified) Boot • Measured Boot • HW Root of Trust	• Capsule Update and Recovery • Independent hardware

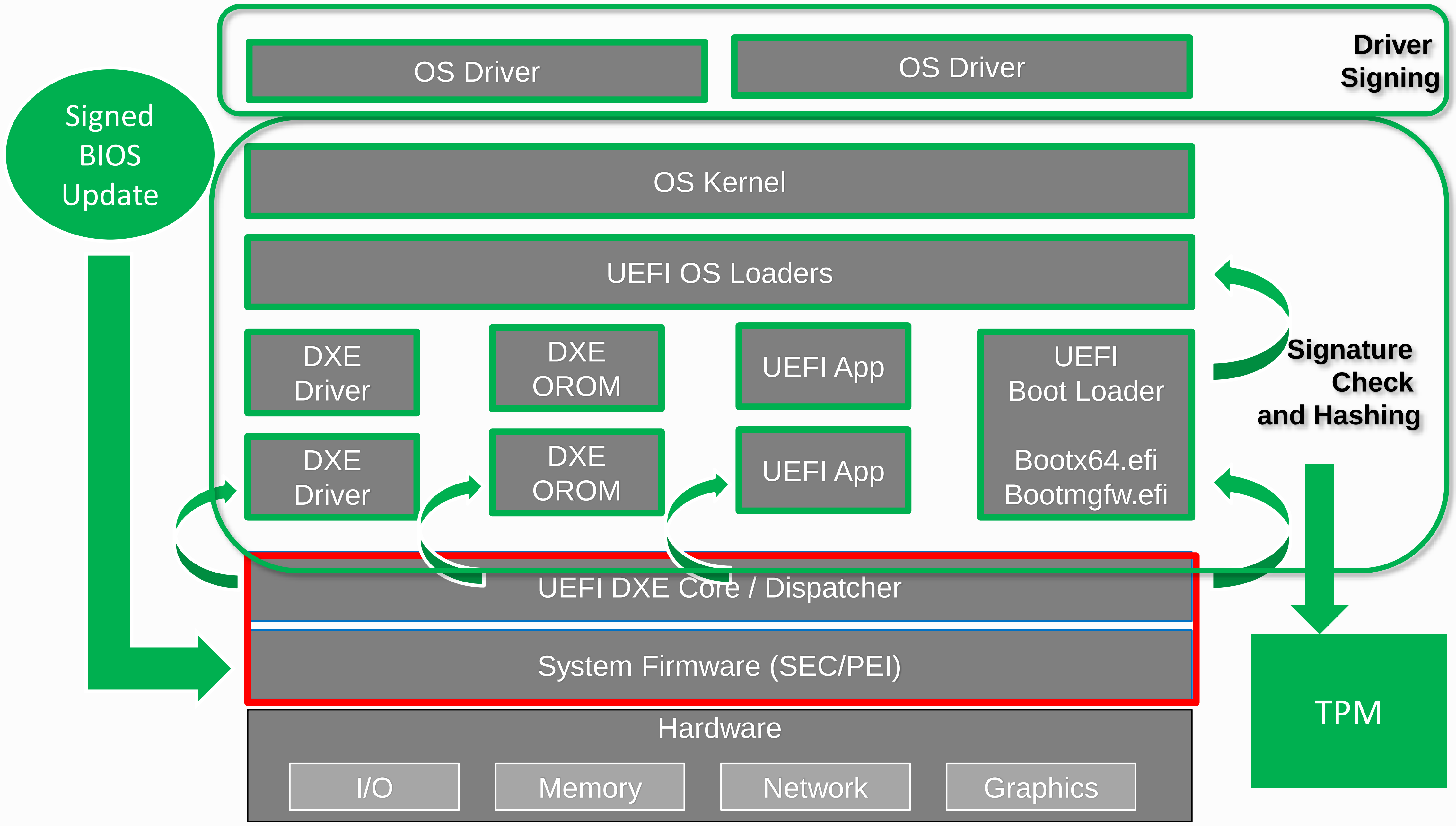
These are examples. Not an exhaustive list.

Boot Media Protections

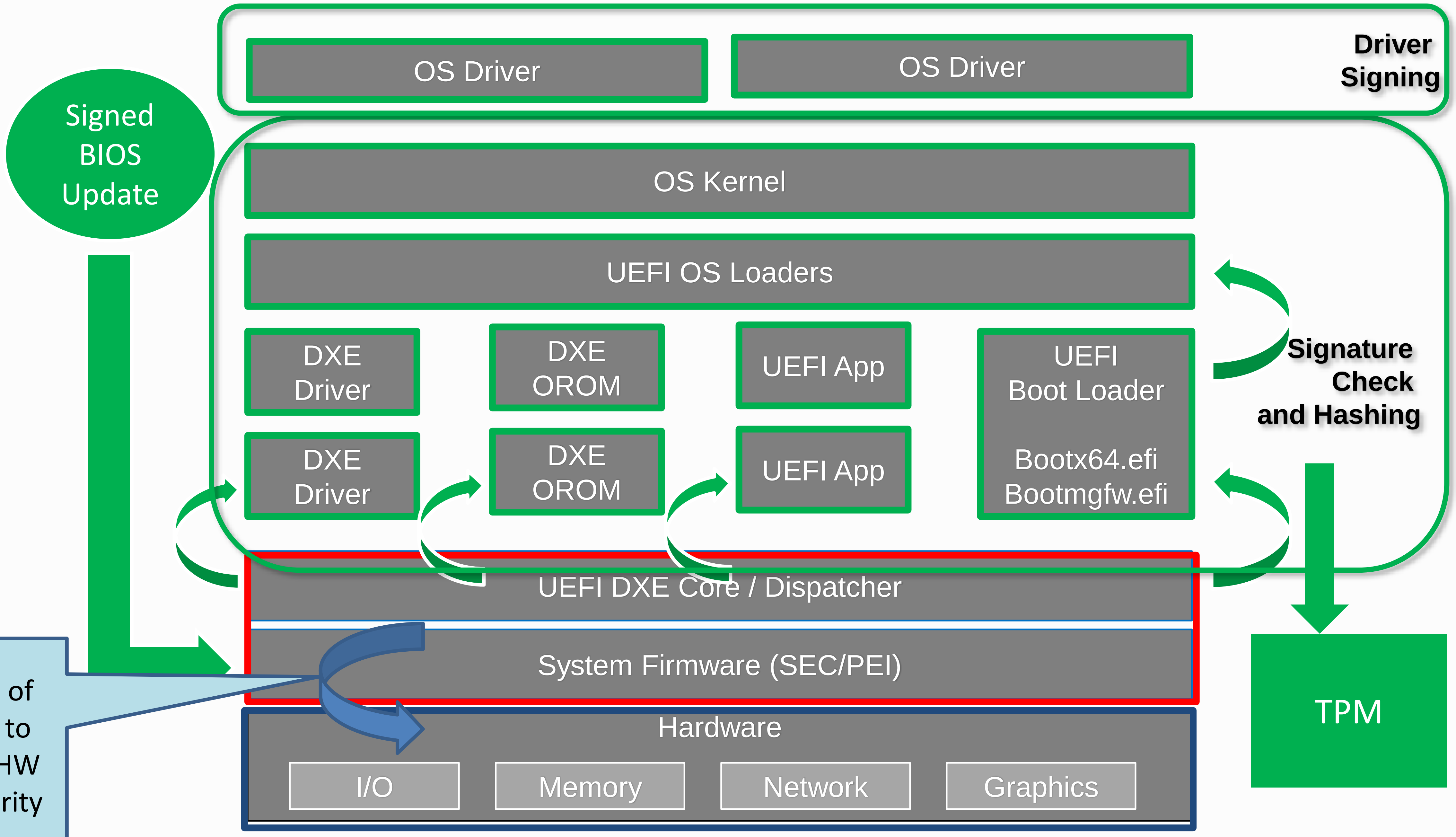
- SPI Controller
  - Descriptor regions and permissions
  - Protected Range Registers
- SMM-Based BIOS Write Protection
  - SMI when enabling write access
  - Enable write access from SMM
- Reducing the TCB



Detection: Verified and Measured Boot



Detection: Hardware Root of Trust



Move the root of trust from FW to HW by having HW check FW integrity

Securing the Platform

Defending the Runtime Firmware Assets

Runtime Firmware Resiliency

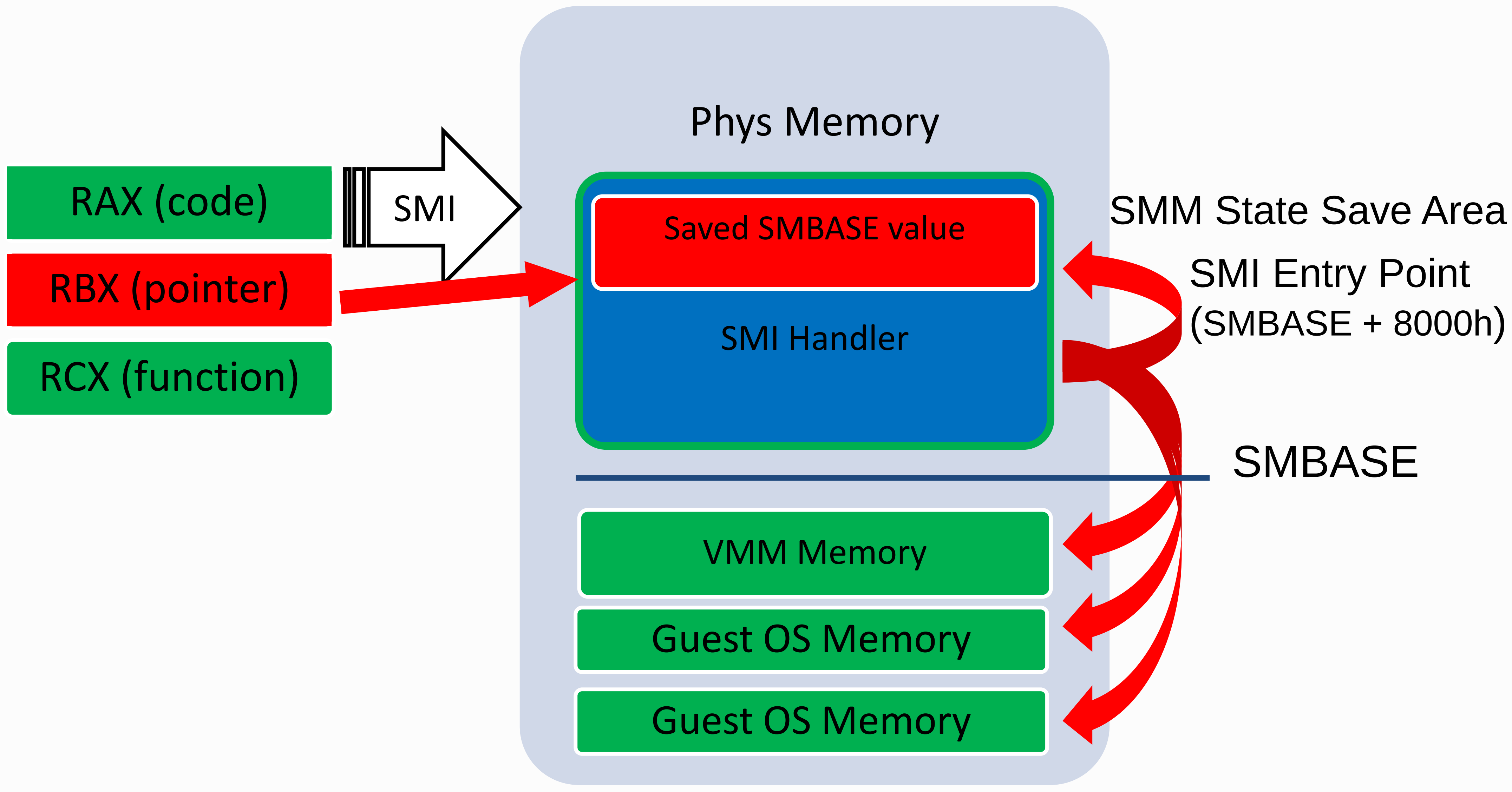
Attack	Protect Mechanism	Detect Mechanism	Recover Mechanism
Call-Outs	• Limited Page Table Access • No Execute Pages • Hardware Check	• Debugger • Fuzzing	• Firmware Update
Confused Deputy	• Limited Page Table Access	• Fuzzing (e.g. CHIPSEC)	• Firmware Update
Malicious DMA	• TSEG • IOMMU		• Reboot • Firmware Update

These are examples. Not an exhaustive list.

System Management Mode (SMM)

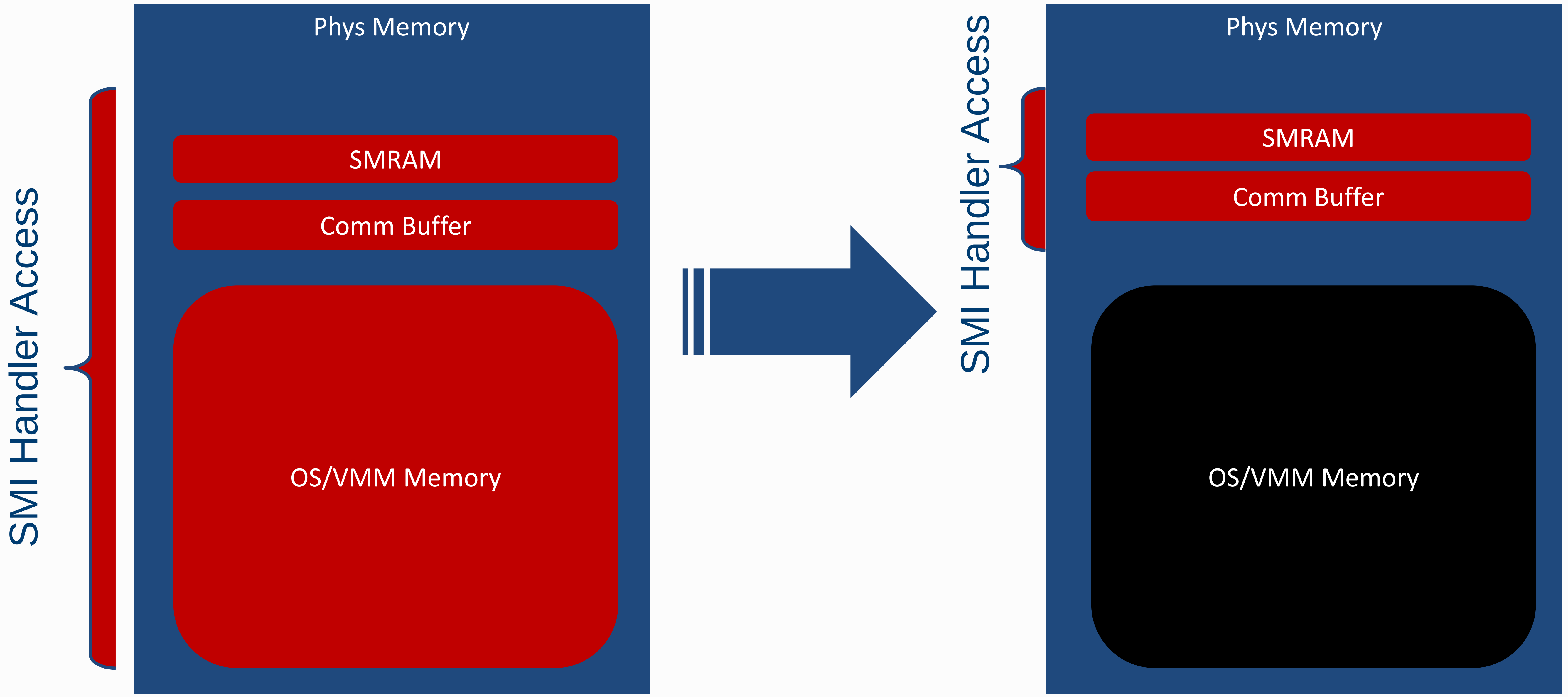
- CPU enters System Management Mode (SMM) upon receiving System Management Interrupt (SMI#) from the chipset or other logical CPU
- CPU (OS) state is saved in SMRAM upon entry to SMM and restored upon exit from SMM. SMRAM is a range of DRAM reserved by BIOS and protected from other runtime code.
- CPU exits SMM to the interrupted OS when SMI handler executes `RSM` instruction (“Resume from SMM”)

SMI “Confused Deputy” Attacks



Attacker can target SMM itself or bypass VMM protections, writing to VMM or other Guest VM memory

SMI Handler Memory Map Restriction



Finding SMM “Pointer” Vulnerabilities

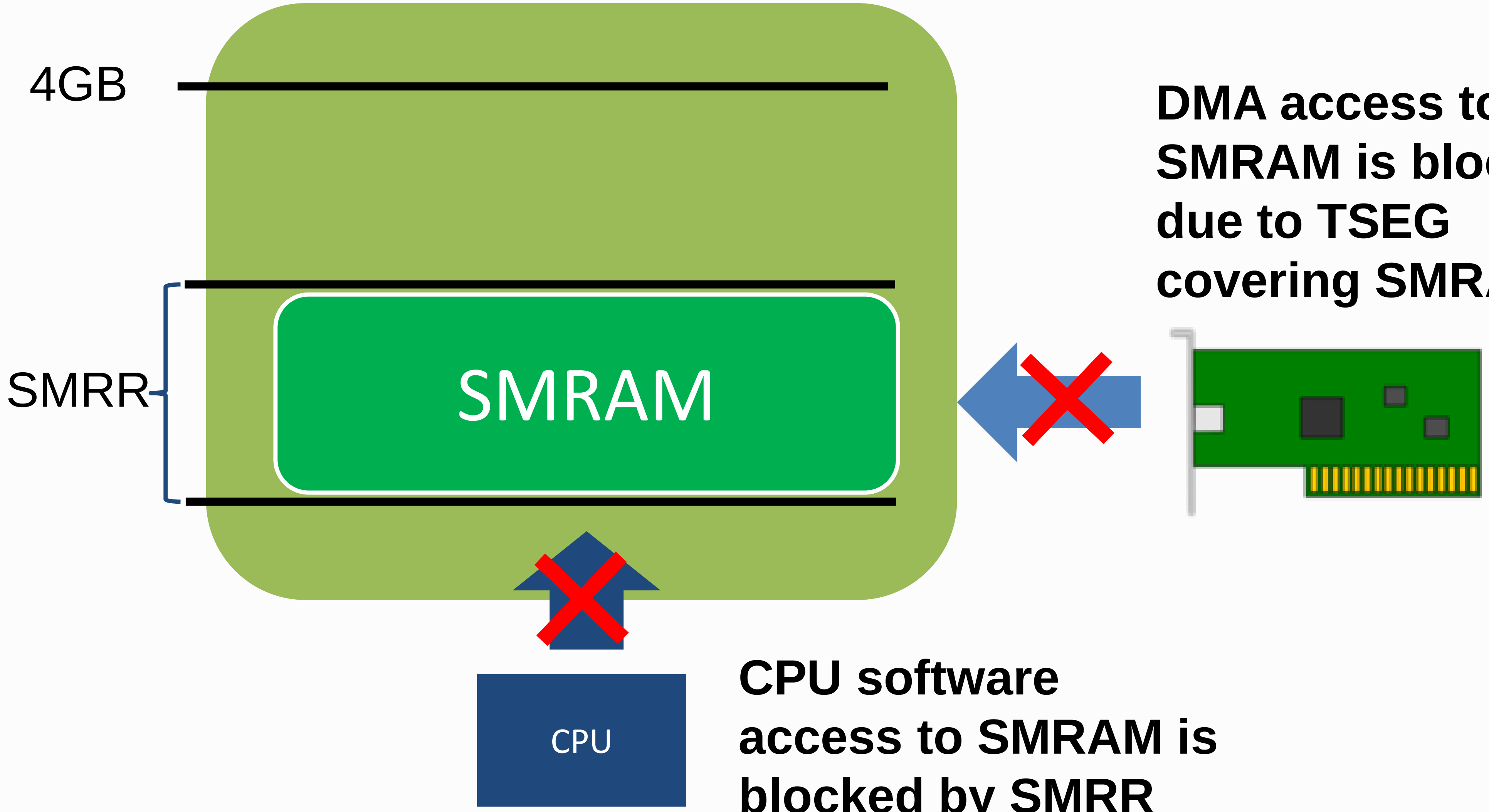
```
[x][ =======================================================================
[x][ Module: Testing SMI handlers for pointer validation vulnerabilities
[x][ =======================================================================
...
[*] Allocated memory buffer (to pass to SMI handlers) : 0x00000000DAAC3000
[*] >>> Testing SMI handlers defined in 'smm_config.ini'..
...

[*] testing SMI# 0x1F (data: 0x00) SW SMI 0x1F
[*] writing 0x500 bytes at 0x00000000DAAC3000
    > SMI 1F (data: 00)
      RAX: 0x5A5A5A5A5A5A5A5A
      RBX: 0x00000000DAAC3000
      RCX: 0x0000000000000000
      RDX: 0x5A5A5A5A5A5A5A5A
      RSI: 0x5A5A5A5A5A5A5A5A
      RDI: 0x5A5A5A5A5A5A5A5A
    < checking buffers contents changed at 0x00000000DAAC3000 +[29,32,33,34,35]
[!] DETECTED: SMI# 1F data 0 (rax=5A5A5A5A5A5A5A5A rbx=DAAC3000 rcx=0 rdx=...)

[-] <<< Done: found 2 potential occurrences of unchecked input pointers
```

https://www.youtube.com/watch?v=z2Qf45nUeaA

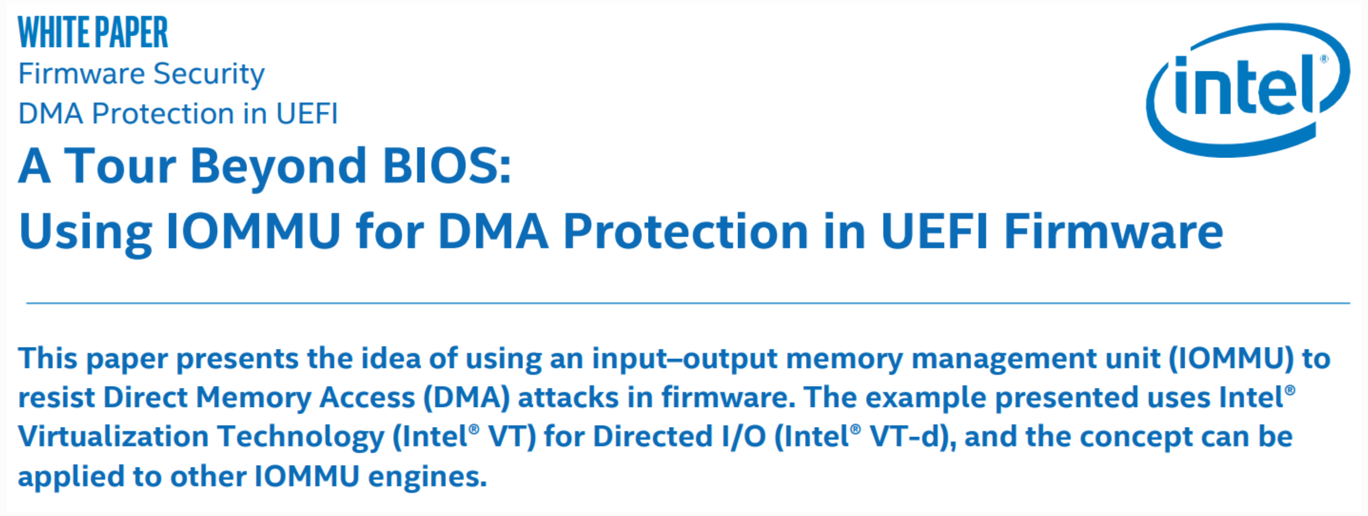
Software/DMA Access to SMRAM



DMA access to SMRAM is blocked due to TSEG covering SMRAM

CPU software access to SMRAM is blocked by SMRR

Preboot DMA Protection



WHITE PAPER
Firmware Security
DMA Protection in UEFI

A Tour Beyond BIOS:
Using IOMMU for DMA Protection in UEFI Firmware

This paper presents the idea of using an input–output memory management unit (IOMMU) to resist Direct Memory Access (DMA) attacks in firmware. The example presented uses Intel® Virtualization Technology (Intel® VT) for Directed I/O (Intel® VT-d), and the concept can be applied to other IOMMU engines.

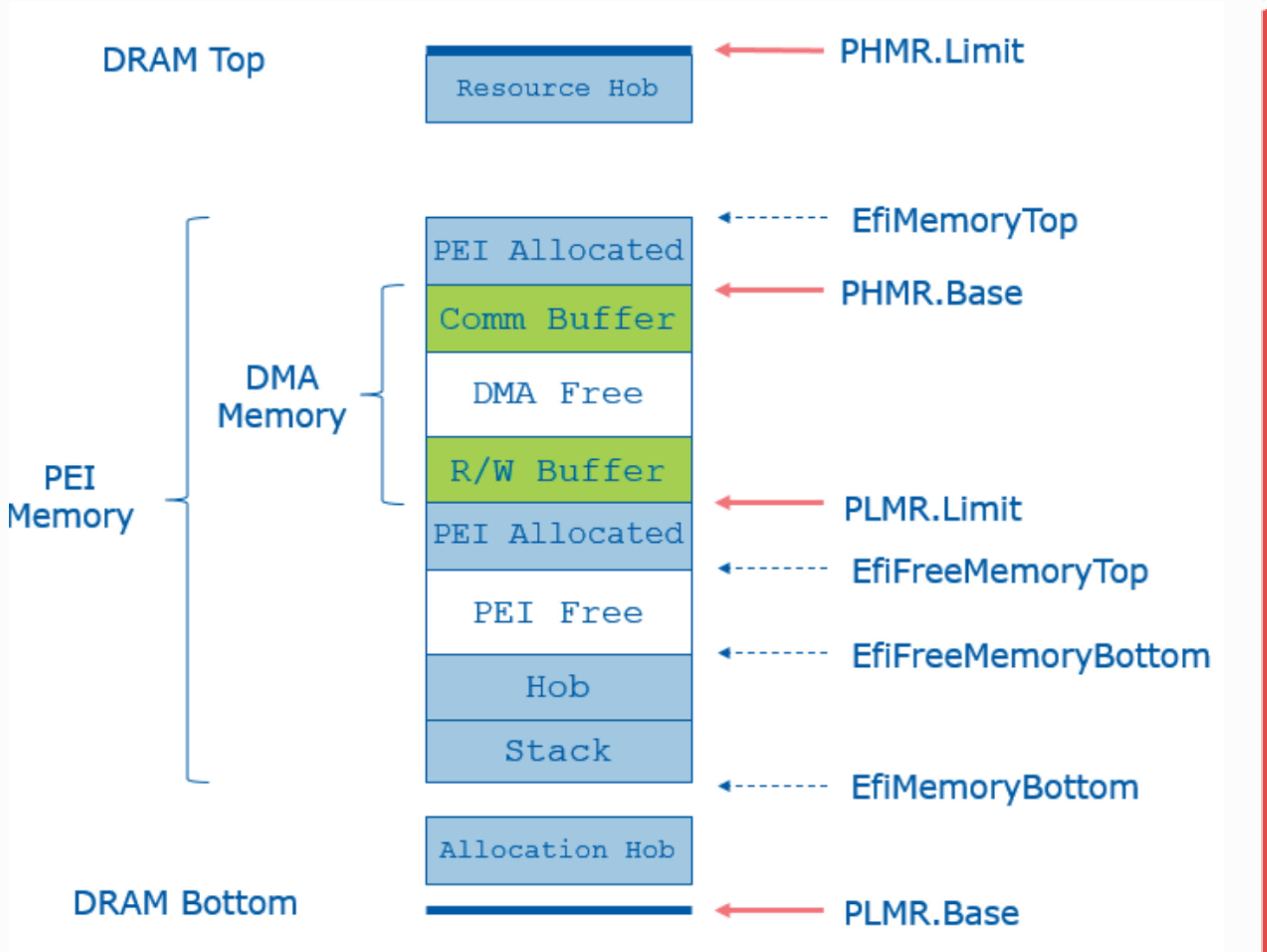


Figure 9 - Memory Map Example

https://firmware.intel.com/sites/default/files/Intel_WhitePaper_Using_IOMMU_for_DMA_Protection_in_UEFI.pdf

Securing the Platform

Defending the HW Configuration Assets

Hardware Configuration Resiliency

Attack	Protect Mechanism	Detect Mechanism	Recover Mechanism
Memory Reconfiguration (e.g. Remap)	• Configuration Guidance & Locking	• CHIPSEC	• Reboot • Firmware Update
Controller Reconfiguration (e.g. SPI)	• Configuration Guidance & Locking	• CHIPSEC	• Reboot • Firmware Update
Feature Enable/Disable or Reconfiguration (e.g. IOMMU, instructions, etc)	• Configuration Guidance & Locking	• CHIPSEC	• Reboot • Firmware Update

These are examples. Not an exhaustive list.

CHIPSEC: Platform Security Assessment Framework

CHIPSEC is a framework for analyzing the security of PC platforms including hardware, system firmware (e.g. BIOS/UEFI), and the configuration of platform components.

Research → Testing → Risk Assessment

Research

- Access to hardware from the OS
- Reusable Python based framework

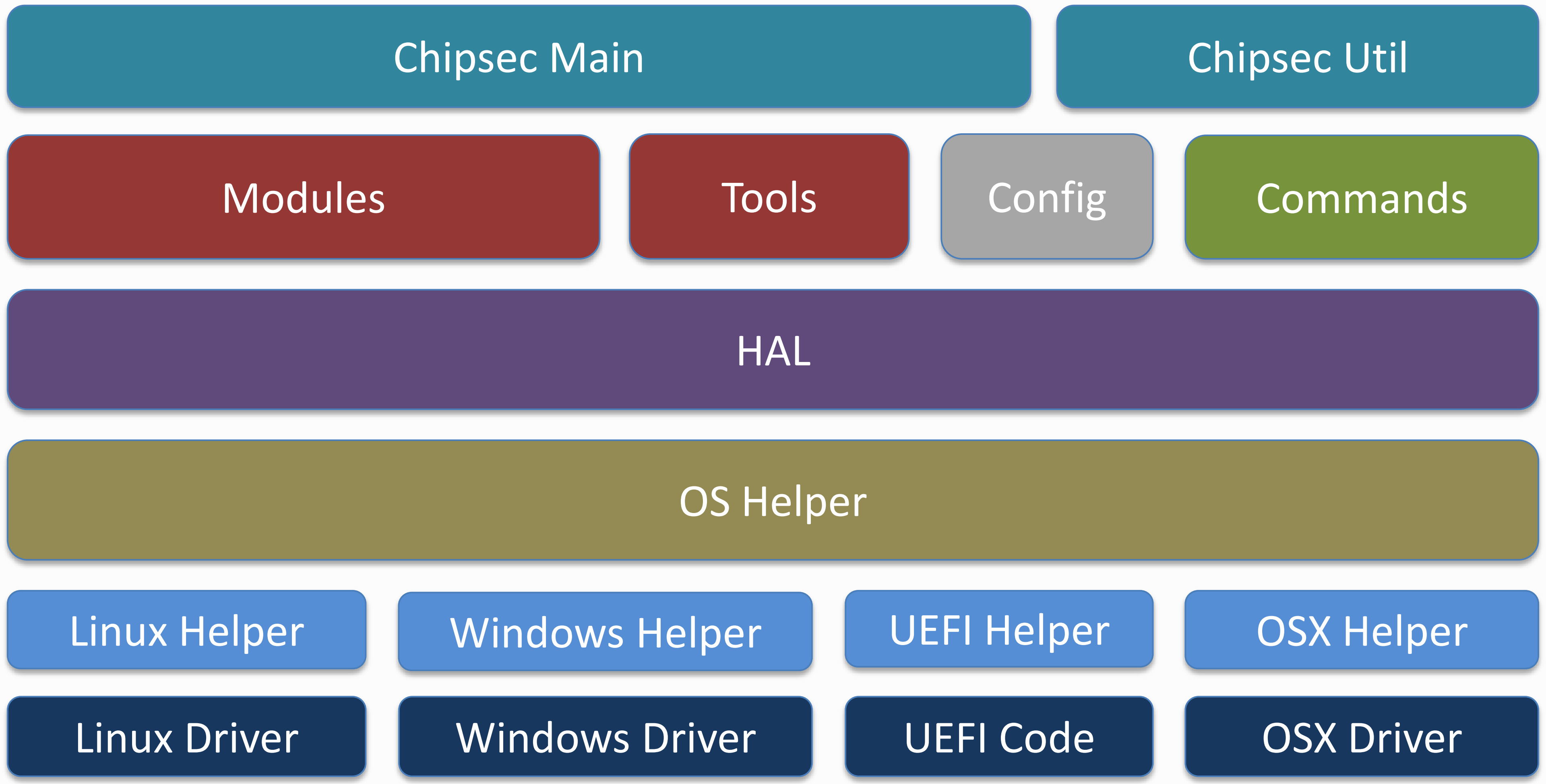
Testing/Validation

- Implement test modules that support multiple platforms
- Ability to provide both positive and negative test cases

Risk Assessment

- Evaluate new systems for vulnerabilities and mitigations
- Evaluate the state of existing systems

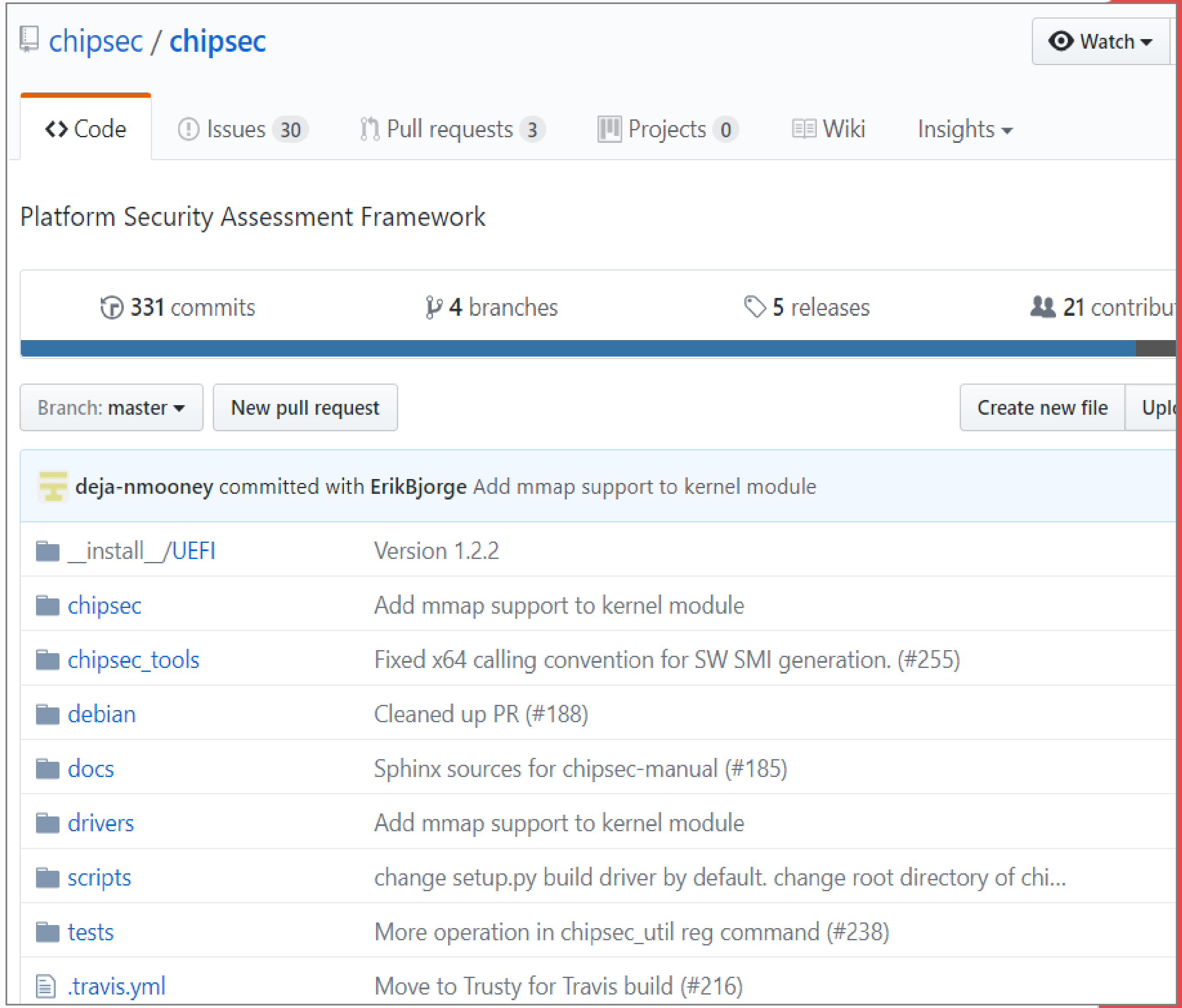
CHIPSEC Architecture



*Other names and brands may be claimed as the property of others.

Testing Against Known Issues

- CHIPSEC - Framework for Platform Security Assessment
  - Tests for known security issues (ex: locking SPI ROM at runtime)
  - Runs under Microsoft Windows, Linux, Mac OS X, and the UEFI Shell
  - chipsec@intel.com
- Open Source (GPLv2 License)
  - https://github.com/chipsec/chipsec
  - Released in 2014
  - Part of Intel's Linux UEFI Validation (LUV) suite: https://01.org/linux-uefi-validation



Platform Security Assessment Framework

deja-nmooney committed with ErikBjorge Add mmap support to kernel module

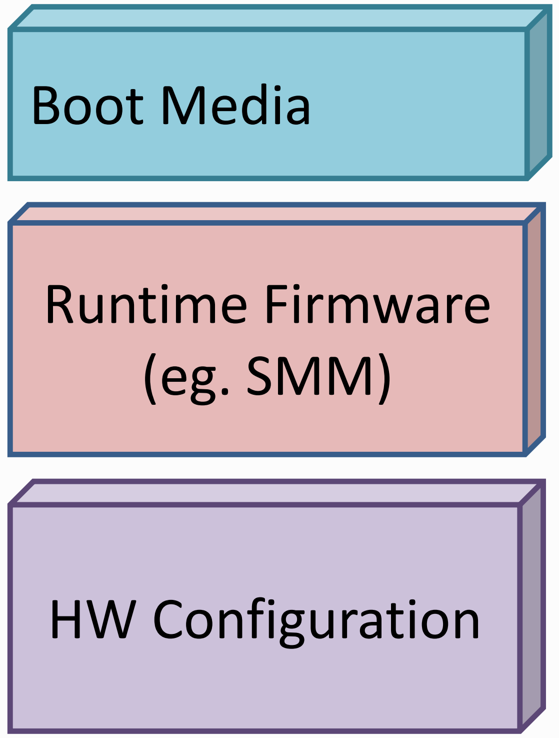
__install__/UEFI	Version 1.2.2
chipsec	Add mmap support to kernel module
chipsec_tools	Fixed x64 calling convention for SW SMI generation. (#255)
debian	Cleaned up PR (#188)
docs	Sphinx sources for chipsec-manual (#185)
drivers	Add mmap support to kernel module
scripts	change setup.py build driver by default. change root directory of chi...
tests	More operation in chipsec_util reg command (#238)
.travis.yml	Move to Trusty for Travis build (#216)

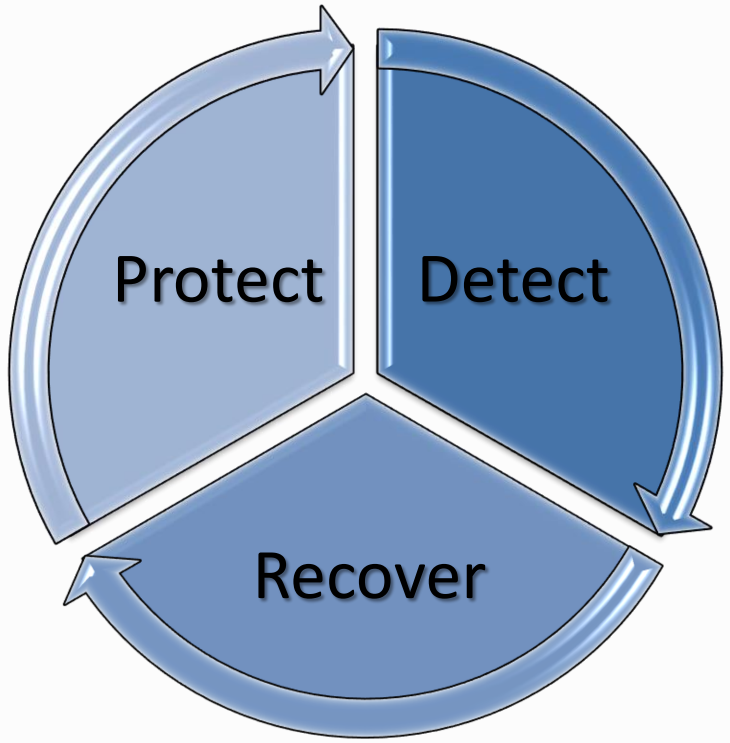
Examples: Checking Locks with CHIPSEC

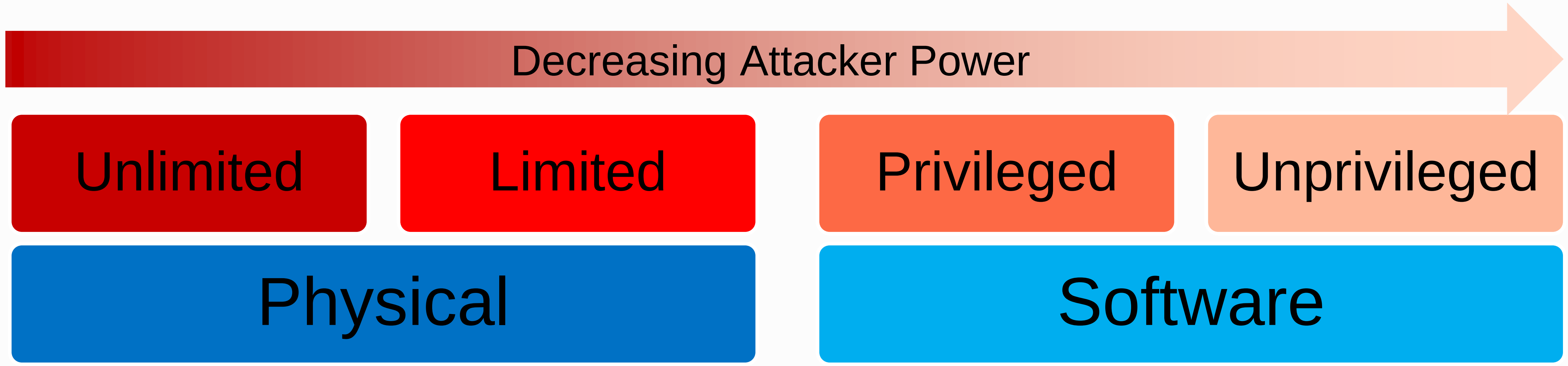
HW Configuration	Test
Memory Controller	memconfig
SPI Descriptor	spi_access
SPI Controller	spi_lock
BIOS Write Protection	bios_wp
Debug Enable/Disable	debug_interface
Architectural Features	ia32cfg

These are examples. Not an exhaustive list.

Resilient Defense







Thanks for attending the Spring 2018 UEFI Plugfest

For more information on the UEFI Forum and UEFI Specifications, visit http://www.uefi.org

presented by