Dell Firmware Security

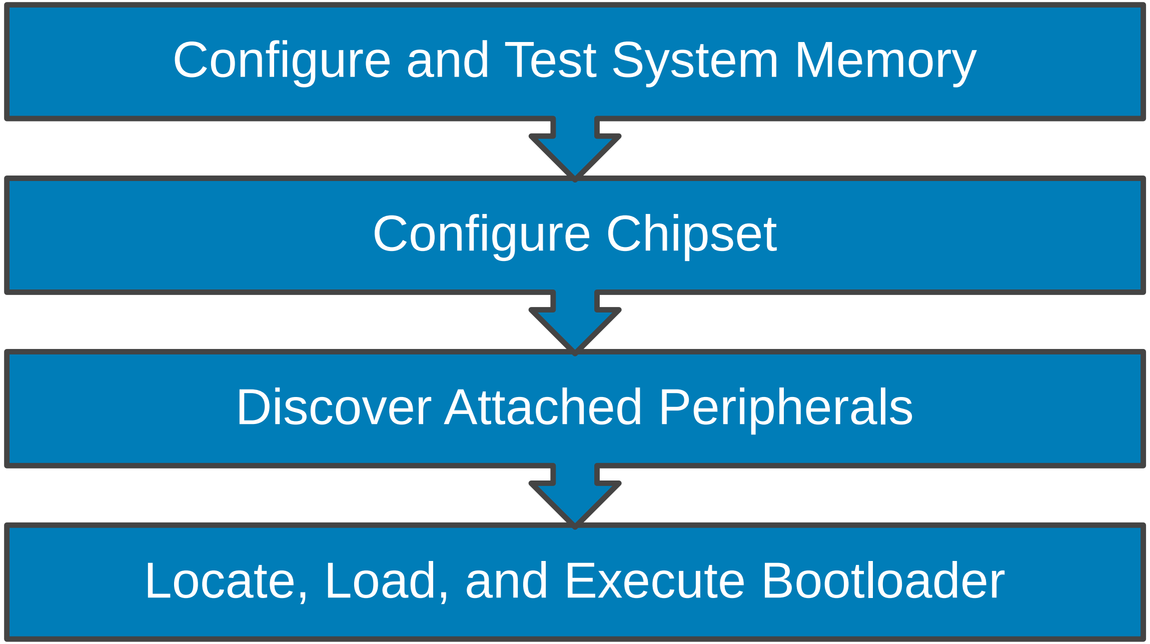
Past, Present, and Future

Justin Johnson
Senior Principal Firmware Engineer
justin.johnson1@dell.com

Dell Security

OpenXT™

What does BIOS do?



Configure and Test System Memory

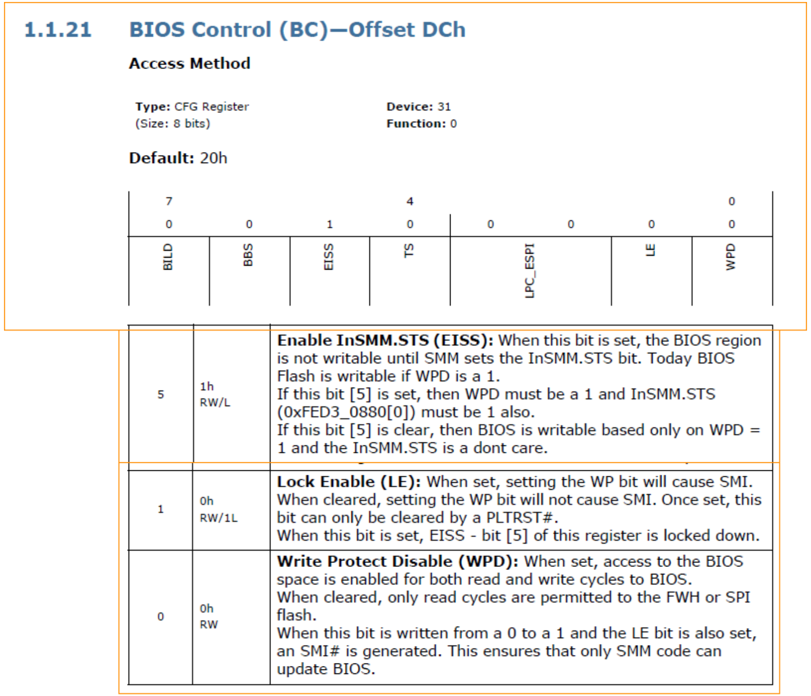
Configure Chipset

Discover Attached Peripherals

Locate, Load, and Execute Bootloader

BIOS Write Protections

- During runtime, block host writes from outside of SMM
- Chipset protection through three bits in BIOS Control register
  - EISS or SMM_BWP
  - BLE
  - WPD or BIOS_WE



1.1.21 BIOS Control (BC)—Offset DCh

Access Method

Type: CFG Register
(Size: 8 bits)

Device: 31
Function: 0

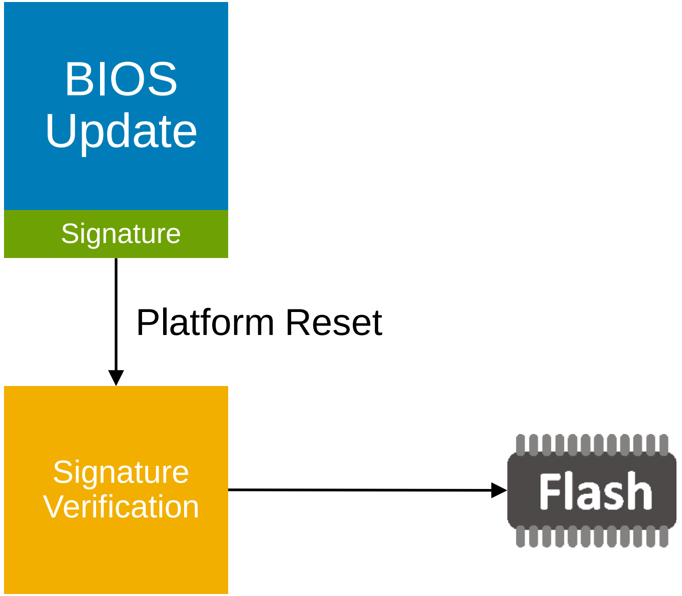
Default: 20h

7			4				0
0	0	1	0	0	0	0	0
BILD	BBS	EISS	TS	LPC_ESPI		LE	WPD

5	1h RW/L	**Enable InSMM.STS (EISS):** When this bit is set, the BIOS region is not writable until SMM sets the InSMM.STS bit. Today BIOS Flash is writable if WPD is a 1. If this bit [5] is set, then WPD must be a 1 and InSMM.STS (0xFED3_0880[0]) must be 1 also. If this bit [5] is clear, then BIOS is writable based only on WPD = 1 and the InSMM.STS is a dont care.
1	0h RW/1L	**Lock Enable (LE):** When set, setting the WP bit will cause SMI. When cleared, setting the WP bit will not cause SMI. Once set, this bit can only be cleared by a PLTRST#. When this bit is set, EISS - bit [5] of this register is locked down.
0	0h RW	**Write Protect Disable (WPD):** When set, access to the BIOS space is enabled for both read and write cycles to BIOS. When cleared, only read cycles are permitted to the FWH or SPI flash. When this bit is written from a 0 to a 1 and the LE bit is also set, an SMI# is generated. This ensures that only SMM code can update BIOS.

BIOS Write Protections

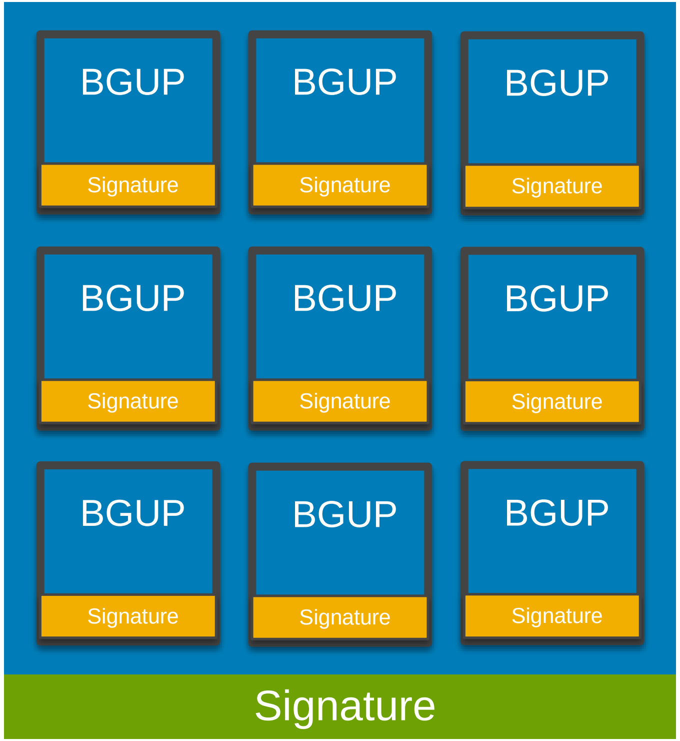
- BIOS updates are digitally signed by Dell
- Updates signed with trusted key are written to flash during reboot cycle
- Follows NIST 800-147 requirements for protected updates



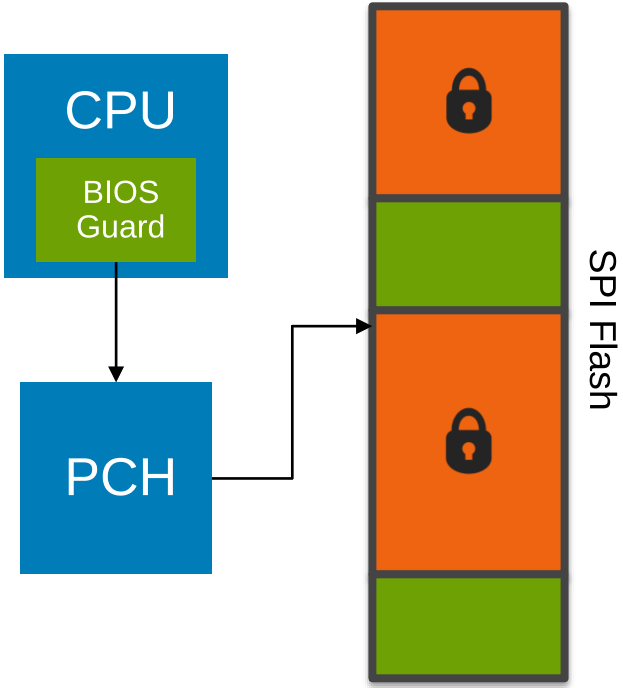
https://csrc.nist.gov/publications/detail/sp/800-147/final

BIOS Write Protections

- BIOSGuard: hardware assisted write protection
- Digitally sign each write to protected flash region
- BIOS update is split into packets and each packet is digitally signed



BIOS Write Protections



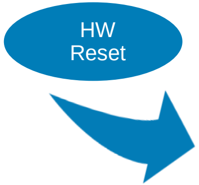
- BIOSGuard has control of PCH flash write enable/disable
- CPU executing from internal RAM
- All flash writes must use BIOSGuard, updates to protected regions must be signed
- Update scripts for unprotected regions (UEFI variables) can be pre-scripted or created on the fly

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

How to trust what’s already on flash?

- Start with a single, immutable piece of code that is trusted and can verify the next piece of code
- Measure each piece of code before execution, propagate measurements for later verification
- Create static root of trust for measurement

Static Root of Trust: Secure Boot Flow

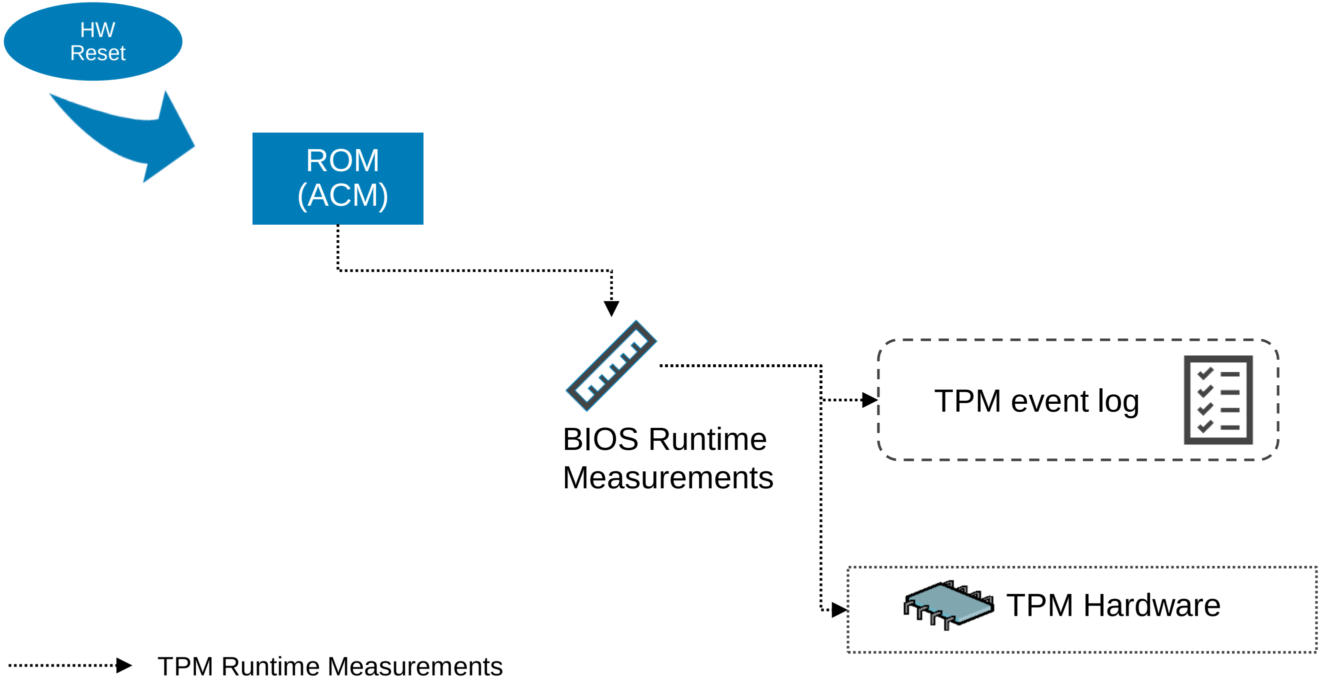


Static Root of Trust: Secure Boot Flow

HW Reset

ROM (ACM)

Static Root of Trust: Secure Boot Flow



HW Reset

ROM (ACM)

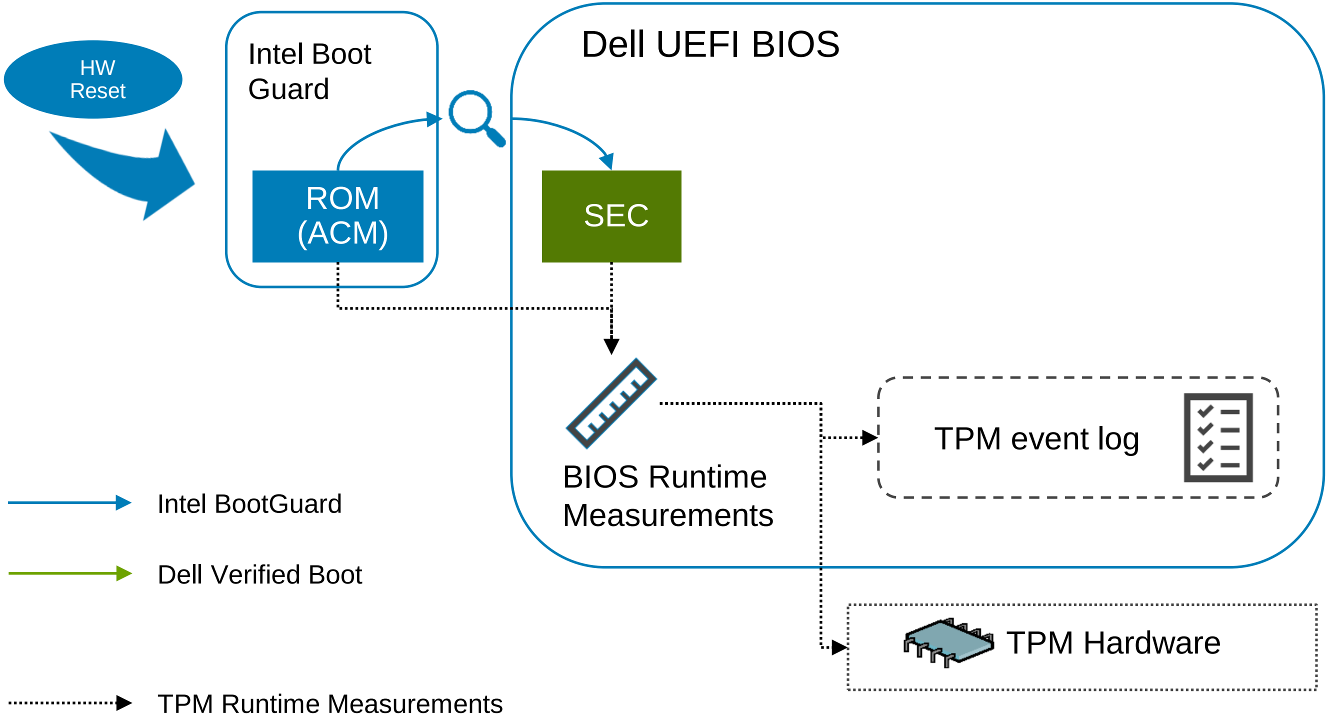
BIOS Runtime Measurements

TPM event log

TPM Hardware

TPM Runtime Measurements

Static Root of Trust: Secure Boot Flow



HW Reset

Intel Boot Guard

ROM (ACM)

Dell UEFI BIOS

SEC

BIOS Runtime Measurements

TPM event log

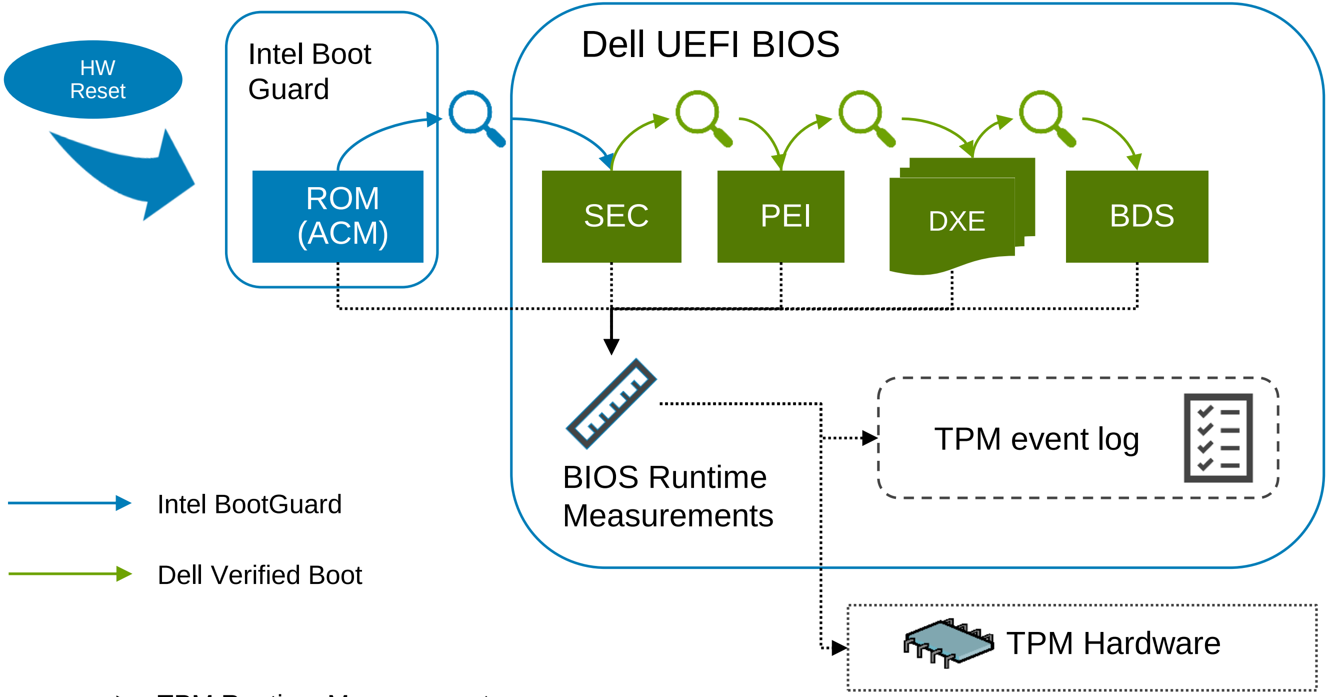
TPM Hardware

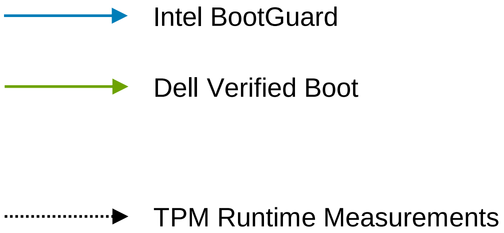
Intel BootGuard

Dell Verified Boot

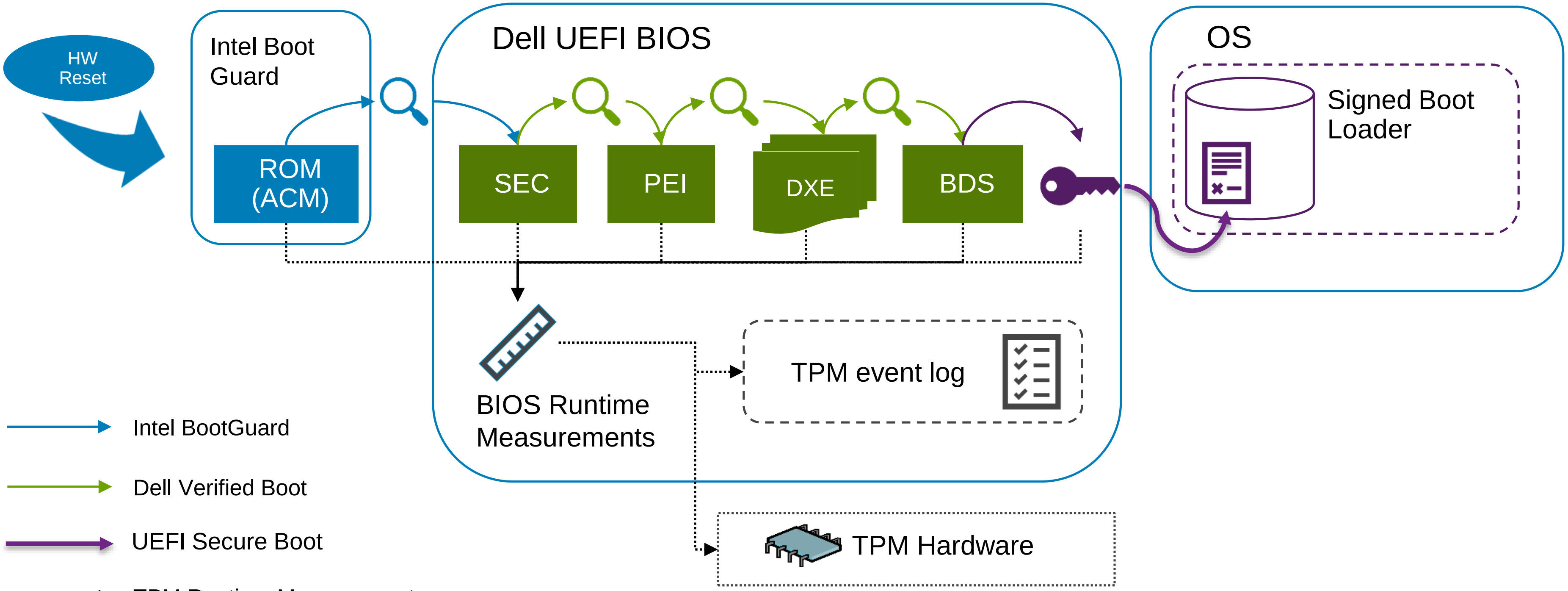
TPM Runtime Measurements

Static Root of Trust: Secure Boot Flow

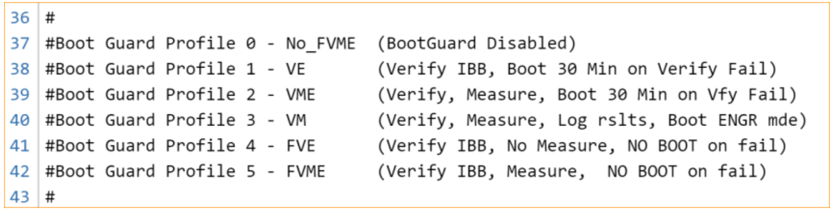




Static Root of Trust: Secure Boot Flow



Static Root of Trust for Measurement

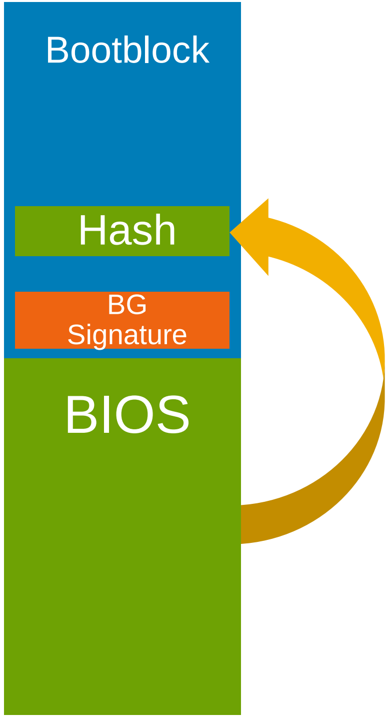


```
#
#Boot Guard Profile 0 - No_FVME  (BootGuard Disabled)
#Boot Guard Profile 1 - VE       (Verify IBB, Boot 30 Min on Verify Fail)
#Boot Guard Profile 2 - VME      (Verify, Measure, Boot 30 Min on Vfy Fail)
#Boot Guard Profile 3 - VM       (Verify, Measure, Log rslts, Boot ENGR mde)
#Boot Guard Profile 4 - FVE      (Verify IBB, No Measure, NO BOOT on fail)
#Boot Guard Profile 5 - FVME     (Verify IBB, Measure,  NO BOOT on fail)
#
```

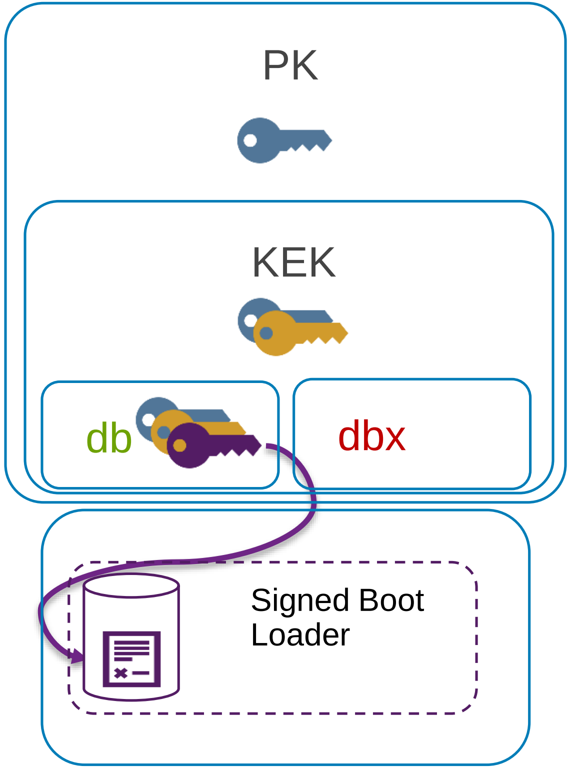
- Intel Boot Guard uses ACM signed by Intel to verify the integrity of initial bootblock of BIOS
- ACM extends measurements to PCR0 – EV_S_CRTM_*
- Bootblock is signed with Dell private key at BIOS build time
- Public key hash is fused into CPU during factory build, ensuring only Dell-signed firmware will be run
- Dell controls the hardware and firmware stack with FVME policy

Chain of Trust

- Ensures that full BIOS is unmodified from original content
- When signing the bootblock, the second-stage BIOS is measured, and the measurement inserted in the bootblock
- After Boot Guard verifies the bootblock, code measures the second stage and compares to the embedded hash
- Execution is blocked if the measurement does not match the embedded measurement
- Measurements extended into PCR0 for later verification – EV_POST_CODE



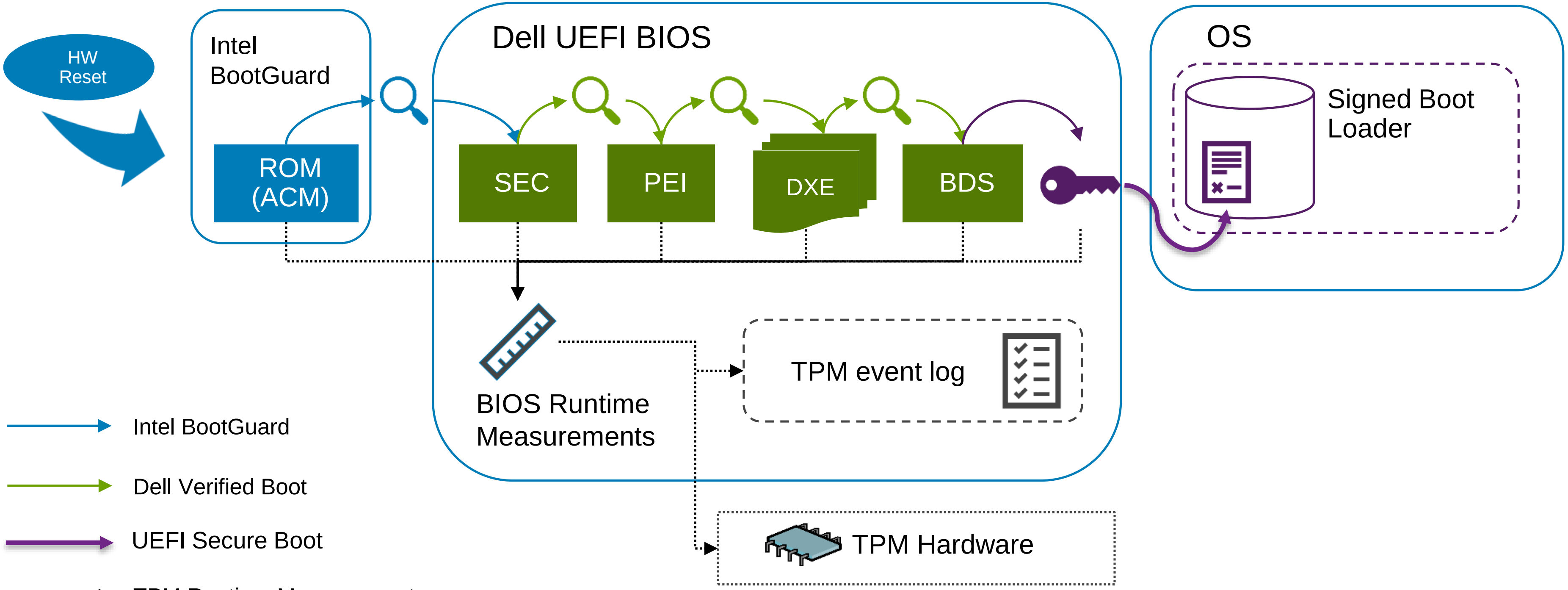
UEFI Secure Boot



- Extends chain of trust to OS bootloader, 3rd party drivers, applications
- Stores certificates in BIOS and verifies the signature of external code
- Dell default configuration includes Microsoft UEFI 3rd party CA certificate; EFI shim bootloader signed by Microsoft
- Customers can enroll their own keys to trust a self-signed bootloader via Audit Mode in UEFI 2.6

http://www.uefi.org/specifications

Static Root of Trust: Secure Boot Flow



Intel BootGuard

Dell Verified Boot

UEFI Secure Boot

TPM Runtime Measurements

TPM Measurements

- PCR0 – BIOS code
  - Measure CRTM code – BootGuard ACM, BIOS code
- PCR1 – BIOS configuration
  - Static SMBIOS structures – hardware configuration
  - User-configurable setup options
- PCR2 – OptionROM code
  - Option ROMs from external devices, embedded ROMs (stored in BIOS SPI flash) are measured in PCR0
- Others as defined by TCG EFI Platform Specification

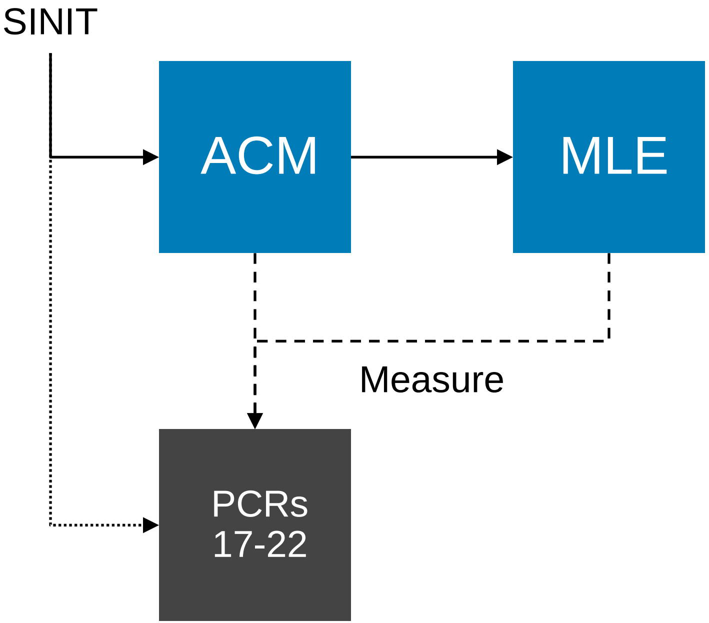
https://trustedcomputinggroup.org/wp-content/uploads/TCG-EFI-Platform-Specification.pdf

Where can it go wrong?

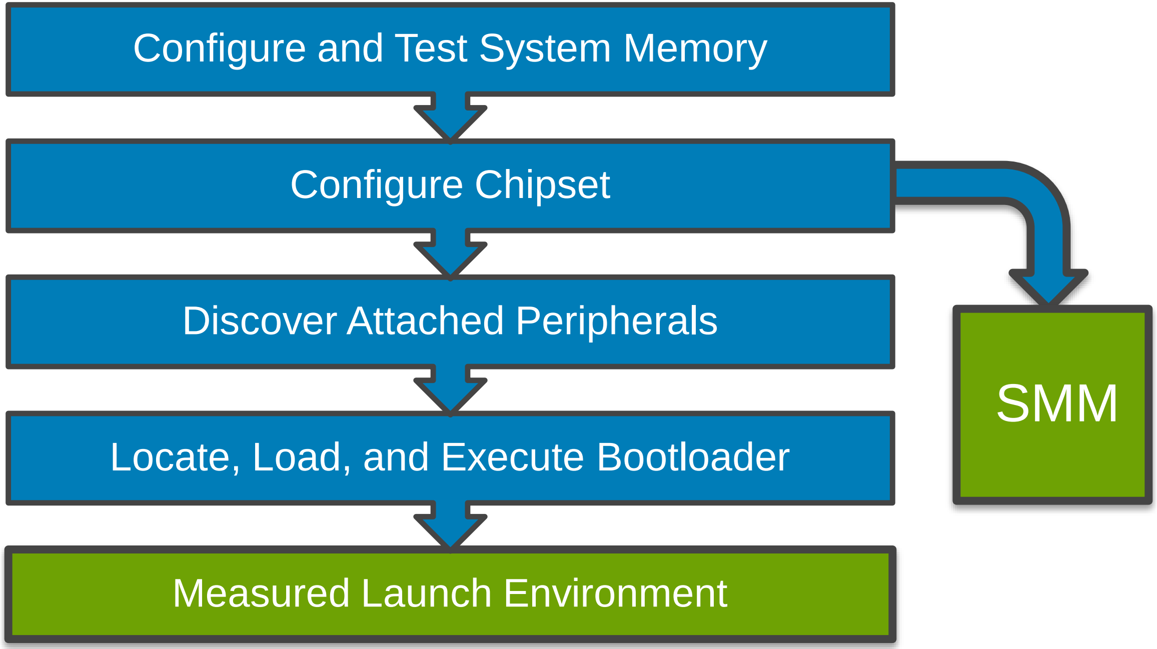
- Trust is only valid from platform reset, must trust all previously executed code, long trust chain from reset vector
- Any change to boot code or configuration results in a different final PCR value – difficult to maintain static root of trust through the lifetime of the platform

Dynamic Root of Trust

- TXT enables late launch of a protected environment, and does not rely on pre-boot code for execution
- Some PCRs can be reset to a known state during runtime, eliminates need to reset entire platform
- Signed code module (SINIT ACM) is loaded into protected memory and measures subsequent software
- Allows for examination of static root of trust PCRs as part of launch control policy



Trusting the protected code



Configure and Test System Memory

Configure Chipset

Discover Attached Peripherals

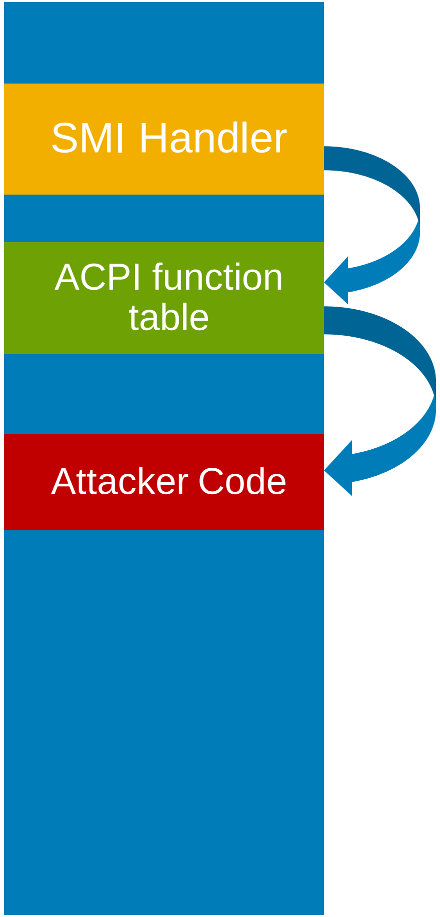
Locate, Load, and Execute Bootloader

Measured Launch Environment

SMM

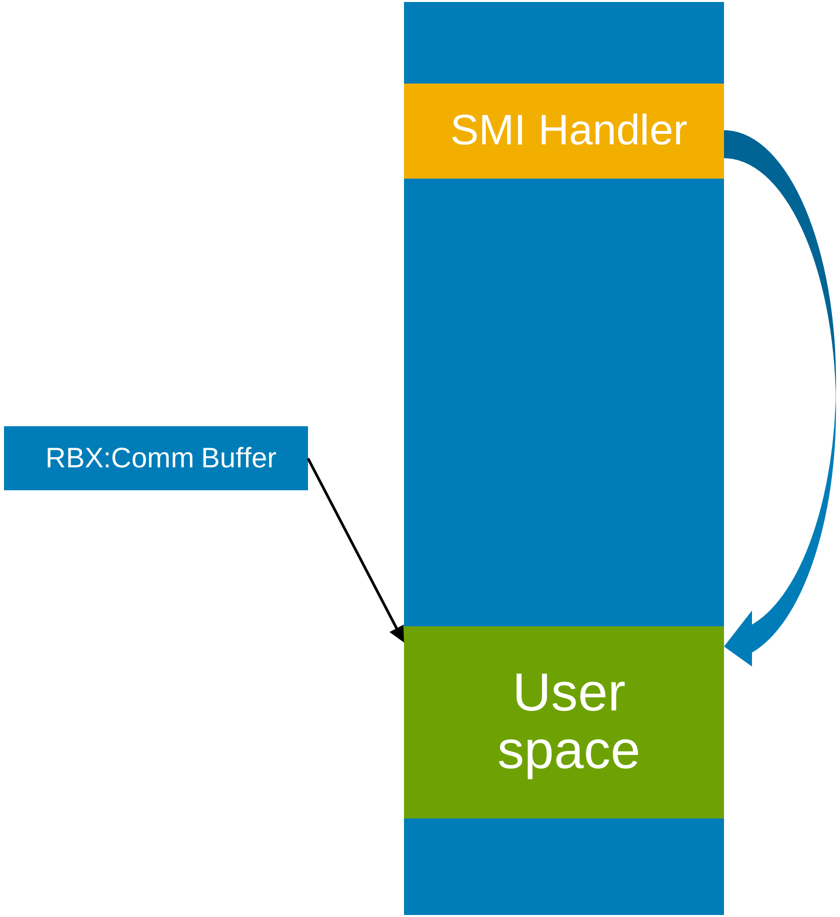
Code execution outside SMRAM

- SMI handler makes a call outside SMRAM, or uses decode table outside protected memory
- Modern chipsets trigger machine check exception if code executed outside range defined by SMRR
- Implement code execution protections normally found in OS memory management



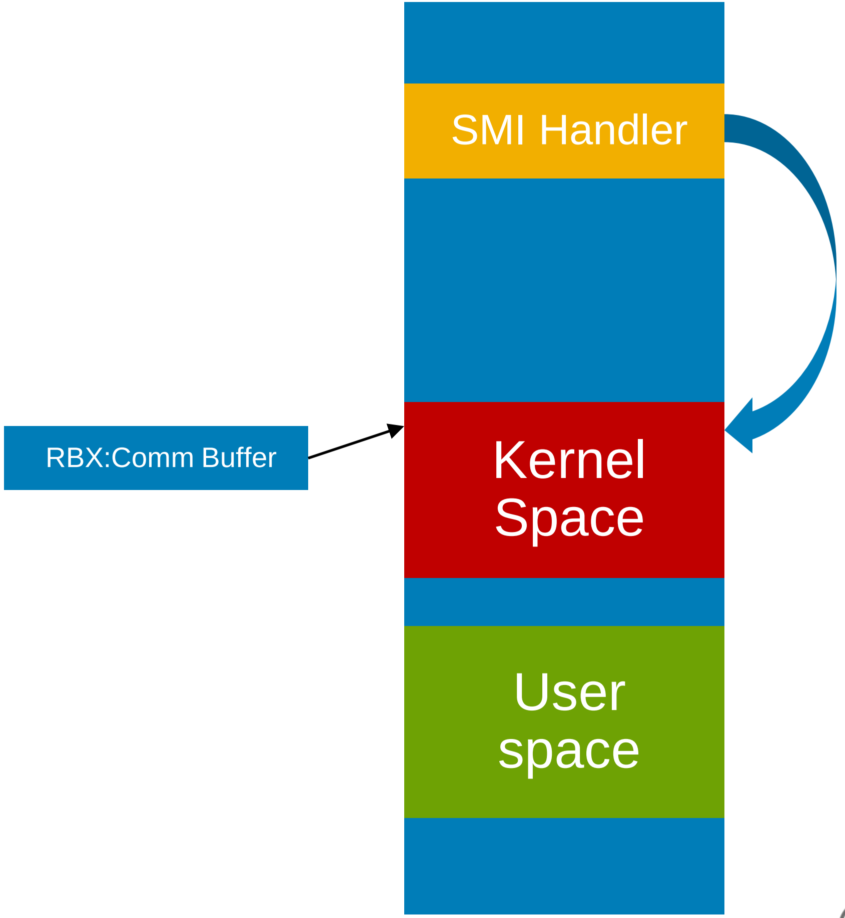
Confused Deputy

- Malicious code uses SMI handler to modify protected kernel or hypervisor memory
- Provide pointer to buffer inside kernel space, SMI handler writes out data to the buffer, corrupting kernel memory
- Solution: use pre-allocated buffer defined during POST for all communication between SMI handler and runtime applications



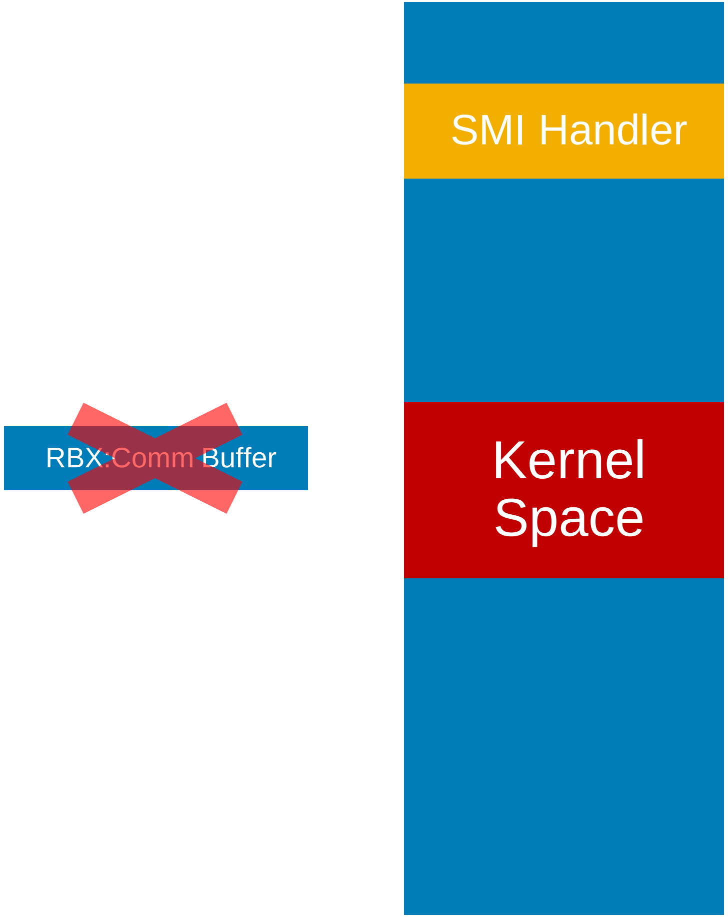
Confused Deputy

- Malicious code uses SMI handler to modify protected kernel or hypervisor memory
- Provide pointer to buffer inside kernel space, SMI handler writes out data to the buffer, corrupting kernel memory
- Solution: use pre-allocated buffer defined during POST for all communication between SMI handler and runtime applications



Confused Deputy

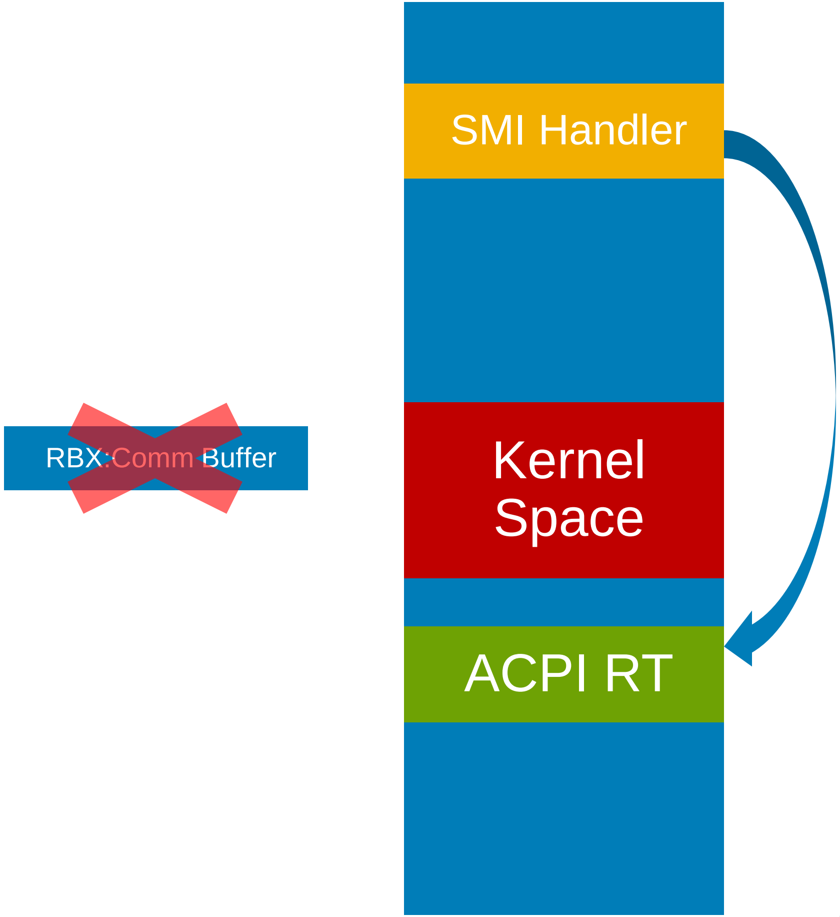
- Malicious code uses SMI handler to modify protected kernel or hypervisor memory
- Provide pointer to buffer inside kernel space, SMI handler writes out data to the buffer, corrupting kernel memory
- Solution: use pre-allocated buffer defined during POST for all communication between SMI handler and runtime applications



Copyright 2018 Dell Technologies – All Rights Reserved

Confused Deputy

- Malicious code uses SMI handler to modify protected kernel or hypervisor memory
- Provide pointer to buffer inside kernel space, SMI handler writes out data to the buffer, corrupting kernel memory
- Solution: use pre-allocated buffer defined during POST for all communication between SMI handler and runtime applications



TOCTOU attack

- Even with fixed comm buffer for SMI, DMA engine can modify buffer contents between check and usage
- Mitigated by copying buffer into TSEG before check and use

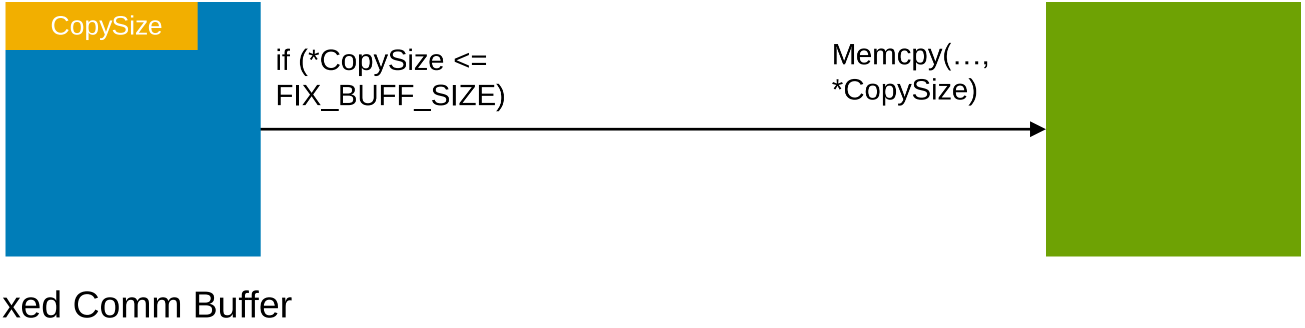
TOCTOU attack

- Even with fixed comm buffer for SMI, DMA engine can modify buffer contents between check and usage
- Mitigated by copying buffer into TSEG before check and use

Fixed Comm Buffer

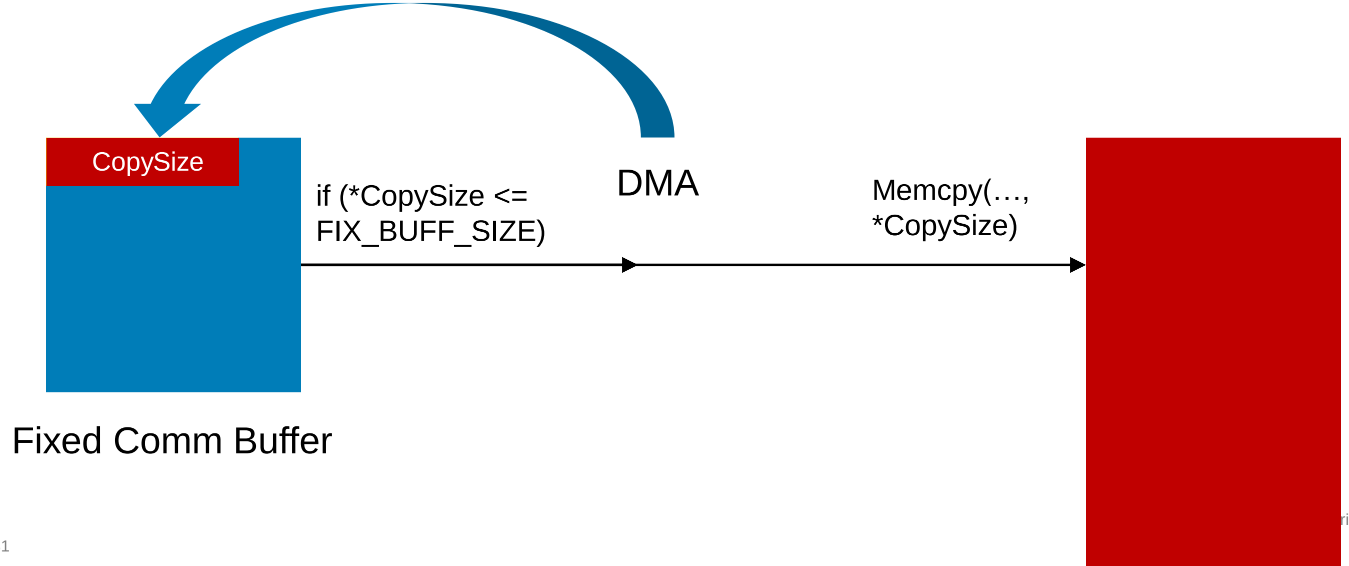
TOCTOU attack

- Even with fixed comm buffer for SMI, DMA engine can modify buffer contents between check and usage
- Mitigated by copying buffer into TSEG before check and use



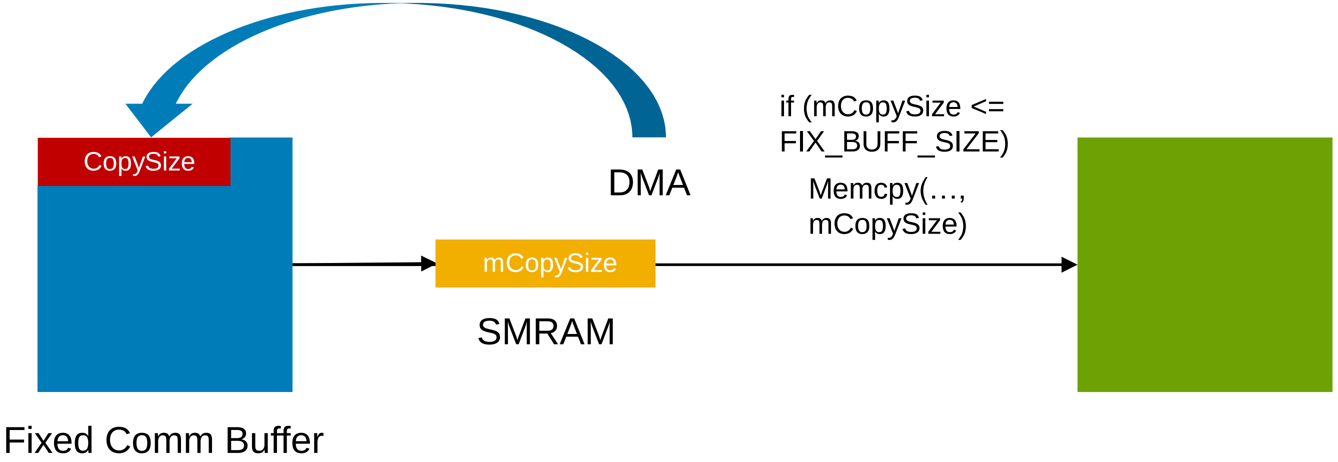
TOCTOU attack

- Even with fixed comm buffer for SMI, DMA engine can modify buffer contents between check and usage
- Mitigated by copying buffer into TSEG before check and use



TOCTOU attack

- Even with fixed comm buffer for SMI, DMA engine can modify buffer contents between check and usage
- Mitigated by copying buffer into TSEG before check and use



Other DMA attacks against static root of trust

- Malicious DMA engine can modify contents of memory after measurement but before use
- Disable DMA for external Thunderbolt devices during POST
- Only allow DMA after SMRAM configuration is locked
- IOMMU during pre-boot environment to protect against even internal devices?

Rise of Windows System Guard

- Microsoft is moving the needle with Virtualization Based Security, requiring OEMs to implement e.g. DRTM, attestation of SMM mitigations
- OpenXT has been doing these things for years, Dell continues to support both environments

http://www.bluehatil.com/files/Hardening%20with%20Hardware.pdf

Conclusions

- Platform security is like an ~~ogre~~ onion, layers of defenses and assurances
- Static- and Dynamic- roots of trust for platform integrity
- OS must be able to extend trust to platform firmware through measurement and attestation

Questions/Discussion

Justin Johnson

justin.johnson1@dell.com