👋 Linux Plumbers Conference 2025 🎉

The Future of Platform Security Measurement in Linux

Linux Plumbers Conference 2025

$ whoami

Maciej Pijanowski

Engineering Manager

🔑 `A766 C895 6989 5C0B 86D5 98D0 9963 C36A AC3B 2B46`

✉️ maciej.pijanowski@3mdeb.com

🐦 @macpijan

🔗 LinkedIn

🌐 3mdeb.com

💻 GitHub

Reach out for collaborations or inquiries!

Agenda

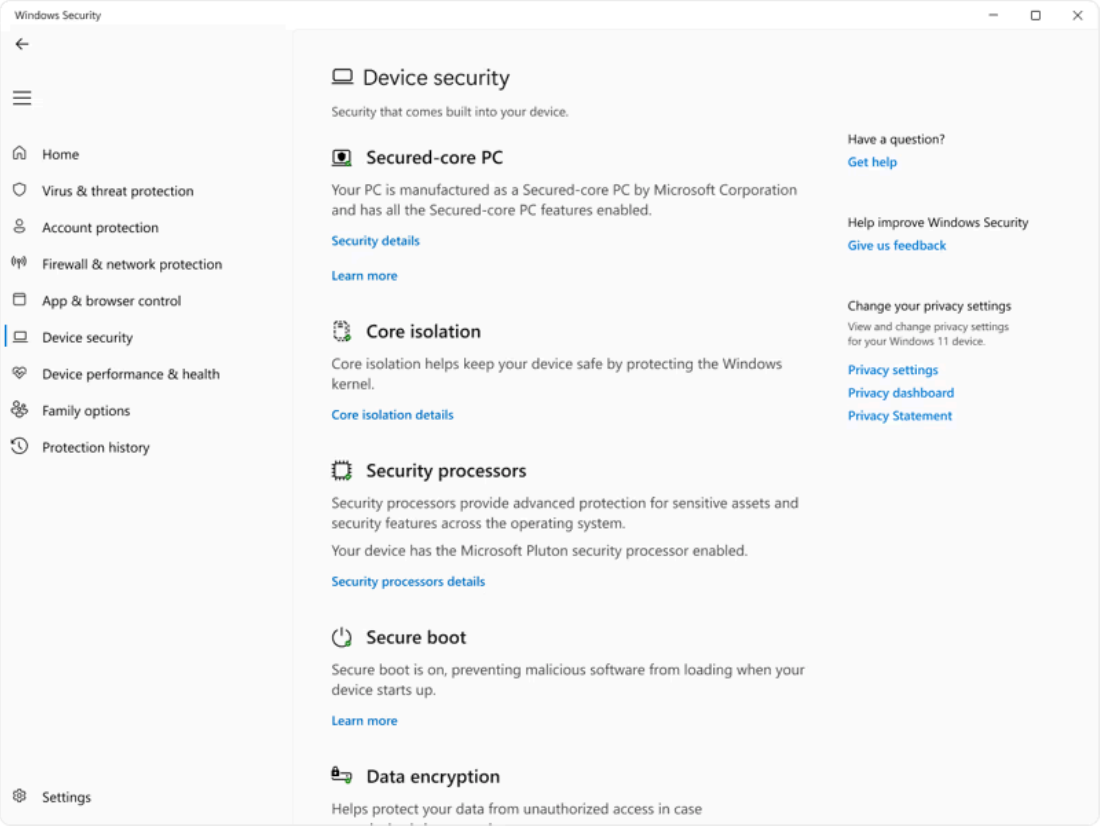
- Why should we care?
- Overview across ecosystem
- fwupd / HSI
- Problems and possible improvements

Why platform security measurement matters

- Firmware is the new attack surface
  - Runs before OS, OS security relies on it (e.g. UEFI Secure Boot)
- Complex security landscape
  - Dozens of complex security features, must be configured correctly
- User awareness gap
  - Users don’t know how secure are their platforms
- Enterprise compliance
  - IT policies mandate specific security configurations

There is a need for OS-enforced firmware quality assessment presenting simple metrics to end user.

Windows



Device security

Security that comes built into your device.

Secured-core PC

Your PC is manufactured as a Secured-core PC by Microsoft Corporation and has all the Secured-core PC features enabled.

Security details

Learn more

Core isolation

Core isolation helps keep your device safe by protecting the Windows kernel.

Core isolation details

Security processors

Security processors provide advanced protection for sensitive assets and security features across the operating system.

Your device has the Microsoft Pluton security processor enabled.

Security processors details

Secure boot

Secure boot is on, preventing malicious software from loading when your device starts up.

Learn more

Data encryption

Helps protect your data from unauthorized access in case

Have a question?

Get help

Help improve Windows Security

Give us feedback

Change your privacy settings

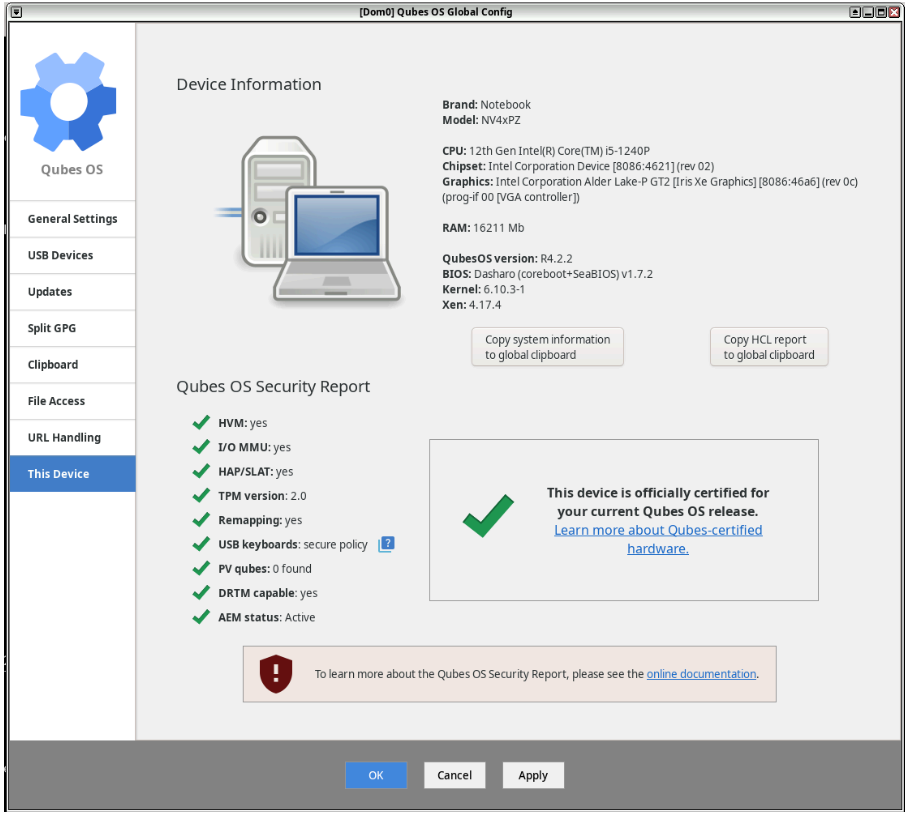
View and change privacy settings for your Windows 11 device.

Privacy settings

Privacy dashboard

Privacy Statement

QubesOS


[Dom0] Qubes OS Global Config
Qubes OS
General Settings
USB Devices
Updates
Split GPG
Clipboard
File Access
URL Handling
This Device
Copy system information to global clipboard
Copy HCL report to global clipboard
OK
Cancel
Apply

Device Information

Brand: Notebook
Model: NV4xPZ

CPU: 12th Gen Intel(R) Core(TM) i5-1240P
Chipset: Intel Corporation Device [8086:4621] (rev 02)
Graphics: Intel Corporation Alder Lake-P GT2 [Iris Xe Graphics] [8086:46a6] (rev 0c) (prog-if 00 [VGA controller])

RAM: 16211 Mb

QubesOS version: R4.2.2
BIOS: Dasharo (coreboot+SeaBIOS) v1.7.2
Kernel: 6.10.3-1
Xen: 4.17.4

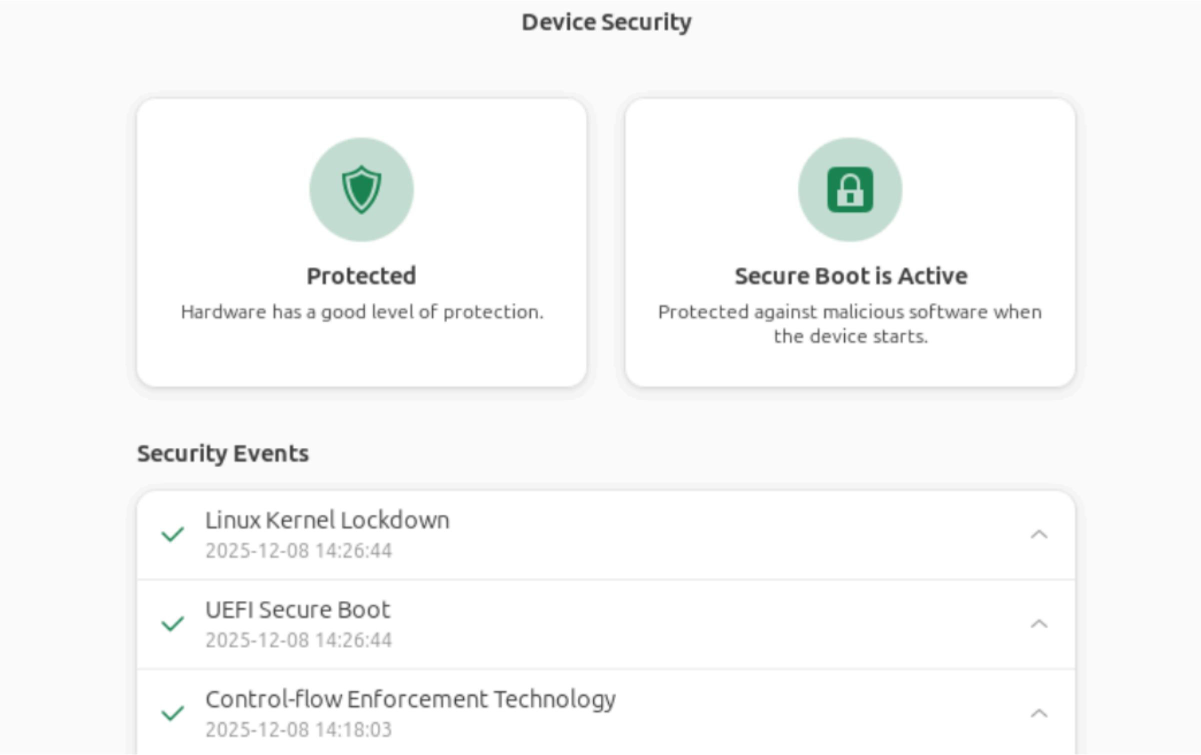
Qubes OS Security Report

- ✓ **HVM:** yes
- ✓ **I/O MMU:** yes
- ✓ **HAP/SLAT:** yes
- ✓ **TPM version:** 2.0
- ✓ **Remapping:** yes
- ✓ **USB keyboards:** secure policy ?
- ✓ **PV qubes:** 0 found
- ✓ **DRTM capable:** yes
- ✓ **AEM status:** Active

✓ **This device is officially certified for your current Qubes OS release.** Learn more about Qubes-certified hardware.

! To learn more about the Qubes OS Security Report, please see the online documentation.

Linux (GNOME)



Device Security

Protected

Hardware has a good level of protection.

Secure Boot is Active

Protected against malicious software when the device starts.

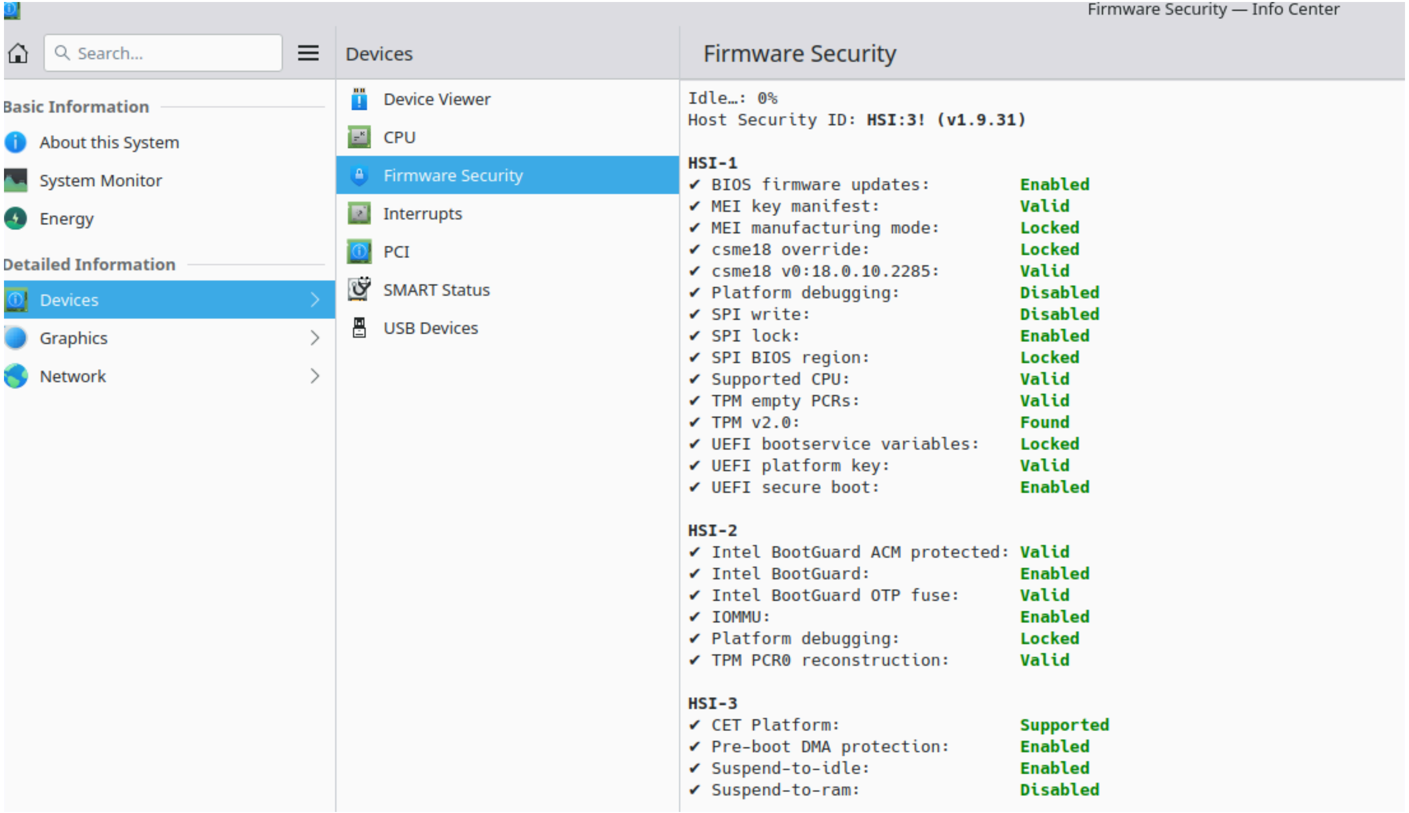
Security Events

✓ Linux Kernel Lockdown
2025-12-08 14:26:44

✓ UEFI Secure Boot
2025-12-08 14:26:44

✓ Control-flow Enforcement Technology
2025-12-08 14:18:03

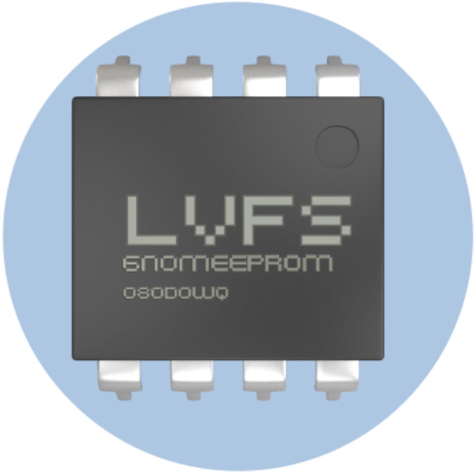
Linux (KDE)



```
Idle…: 0%
Host Security ID: HSI:3! (v1.9.31)

HSI-1
✔ BIOS firmware updates:           Enabled
✔ MEI key manifest:                Valid
✔ MEI manufacturing mode:          Locked
✔ csme18 override:                 Locked
✔ csme18 v0:18.0.10.2285:          Valid
✔ Platform debugging:              Disabled
✔ SPI write:                       Disabled
✔ SPI lock:                        Enabled
✔ SPI BIOS region:                 Locked
✔ Supported CPU:                   Valid
✔ TPM empty PCRs:                  Valid
✔ TPM v2.0:                        Found
✔ UEFI bootservice variables:      Locked
✔ UEFI platform key:               Valid
✔ UEFI secure boot:                Enabled

HSI-2
✔ Intel BootGuard ACM protected: Valid
✔ Intel BootGuard:                 Enabled
✔ Intel BootGuard OTP fuse:        Valid
✔ IOMMU:                           Enabled
✔ Platform debugging:              Locked
✔ TPM PCR0 reconstruction:         Valid

HSI-3
✔ CET Platform:                    Supported
✔ Pre-boot DMA protection:         Enabled
✔ Suspend-to-idle:                 Enabled
✔ Suspend-to-ram:                  Disabled
```

What is fwupd?

- A tool for applying firmware updates from the Linux Vendor Firmware Service (LVFS)
- Beyond updates: evaluates system security through HSI scoring
- Presents users with automated security reports
- Users generate reports: `fwupdmgr security` or `fwupdtool security`



Host Security ID (HSI)

- A proposal of standardized metric to quantify platform security
- Developed by:
  - Richard Hughes (Red Hat)
  - Mario Limonciello (AMD)
  - Alex Bazhaniuk (Eclypsium)
  - Alex Matrosov (Binarly)
- **Important:** Specification is under active development
  - Incomplete, subject to change, may have errors
- https://fwupd.github.io/libfwupdplugin/hsi.html

HSI overview

Hierarchical framework with multiple levels

- **HSI-0:** HSI-1 requirements not met
- **HSI-1:** Least restrictive - non-permanent features
  - BIOS update capability, TPM presence, SPI write protection, UEFI Secure Boot
- **HSI-2:** Hardware-based firmware verification
  - "Fusing" - irreversible hardware changes enforcing firmware authorization
- **HSI-3:** Advanced protections
  - CPU control-flow integrity, DMA protection, low-power state requirements
- **HSI-4:** Memory protection
  - Supervisor Mode Access Prevention (SMAP), memory encryption
- **HSI-5:** Out-of-band attestation (planned, not yet implemented)

Inputs for HSI

fwupd uses several different interfaces to have an overview of platform's security

Method	Interface	Tool/Path	Example Checks
sysfs (kernel) 🟢	sysfs	`/sys/class/` , `/sys/kernel/security/`	IOMMU, lockdown, TPM
CPUID 🟢	instruction	Direct CPU instruction, `/proc/cpuinfo`	CET, TME support
procfs 🟢	procfs	`/proc/sys/` , `/proc/cmdline` , `/proc/swaps`	Kernel tainted, swap
ACPI Tables 🟢	sysfs	`/sys/firmware/acpi/tables/`	DMAR (DMA protection)
EFI Variables 🟢	sysfs	`/sys/firmware/efi/efivars/`	SecureBoot, PK
MSR 🟡	devfs	`/dev/cpu/0/msr`	Platform debugging (DCI), TME
MTD 🟡	devfs	`/dev/mtd0`	Flash descriptor
PCI Config Space 🔴	sysfs	`/sys/bus/pci/devices/ … /config`	ME HFSTS (BootGuard), BCR (SPI)

Proper user-space interfaces 🟢

- Sysfs
  - Read `/sys/class/tpm/tpm0/tpm_version_major` for TPM version
  - Read `/sys/power/mem_sleep` for available suspend modes
- ACPI tables
  - Read `/sys/firmware/acpi/tables/DMAR` and check DMA protection flag
- UEFI variables
  - Read `SecureBoot` EFI variable

MSR 🟡

- Current flow
  - Open `/dev/cpu/0/msr`
  - Read buffer at register's offset (e.g. `IA32_DEBUG_INTERFACE`, `IA32_TME_ACTIVATION`)
  - Parse bit fields to inspect configuration (e.g. debug interface and memory encryption)
- Problems
  - Requires root permissions and `msr` kernel module
  - Low-level hardware knowledge in userspace (bit parsing)
- Possible improvements
  - Expose as sysfs entries for Intel CPUs as well
  - AMD exposes some security properties, e.g.:
    - `/sys/bus/pci/devices/<BDF>/debug_lock_on`
    - `/sys/bus/pci/devices/<BDF>/tsme_status`
    - AMD PSP patchset

Parsing Intel Flash Descriptor (IFD) 🟡

- Current flow
  - Open `/dev/mtd0`
  - Parse IFD structure
  - Check if descriptor region is write-protected by parsing bit fields
- Problems
  - Requires root permissions
  - Parsing of low-level IFD structures
  - Multiple IFD layout versions have to be supported by the tool
  - Low-level hardware knowledge in userspace (bit parsing)
- Possible improvements
  - Parsing done once by kernel
  - Expose parsed IFD and access permissions as sysfs entries

Parsing PCI config space (BCR) 🔴

- Current flow
  - Find Intel PCH device
  - Open `/sys/bus/pci/devices/<BDF>/config`
  - Read at offset `0xDC` (`BIOS_CNTL - BIOS Control Register`)
  - Parse bits
    - `Write Protect Disable`, `BIOS Lock Enable`, `SMM BIOS Write Protect`
- Problems
  - Low-level hardware knowledge in userspace (bit parsing)
- Possible improvements
  - Parsing done once by kernel
  - Expose flash security flags as sysfs entries

Parsing PCI config space (ME) 🔴

- Current flow
  - Open `/sys/bus/pci/devices/0000:00:16.0/config`
  - Read 6 HFSTS registers at different offsets:
    - HFSTS1 at `0x40` - Manufacturing mode, operation mode
    - HFSTS2 at `0x48` - System state, error codes
    - HFSTS3 at `0x60` - Firmware SKU
    - HFSTS4 at `0x64` - Flash operation status
    - HFSTS5 at `0x68` - ACM (Authenticated Code Module) status
    - HFSTS6 at `0x6C` - BootGuard config, OTP fuse

Parsing PCI config space (ME) 🔴 #2

- Problems
  - 6x 32-bit registers
  - Version-dependent layouts (CSME 11-17 vs 18+)
  - **Breaks when ME disabled** (false negatives - Intel Boot Guard still works)
- Possible improvements
  - Parsing done once by kernel
  - Expose ME and Intel Boot Guard configuration status in sysfs
  - AMD: `/sys/bus/pci/devices/<BDF>/fused_part`
    - reports whether the CPU has been fused to prevent tampering

Going further: firmware security interface?

- Centralized security posture API
- Reusable across tools (not just fwupd)
- No need for root privileges to check security status
- Simplified implementation for userspace tools (vendor abstraction)
- A "similar" pattern already exists: `/sys/devices/system/cpu/vulnerabilities/`
  - translates low-level details into user-readable `PASS / FAIL` information
  - https://docs.kernel.org/admin-guide/hw-vuln/

```
cat /sys/devices/system/cpu/vulnerabilities/meltdown
Not affected
```

Going further: firmware security interface?

```
/sys/firmware/security/
├── flash/
│   └── descriptor
│       ├── locked              # "0" or "1" - descriptor region write-locked
│       └── version             # "1", "2", or "3" - IFD version
├── srtm/                       # Vendor-agnostic HW RoT interface
│   ├── technology              # "bootguard", "psb", "trustzone", "secureboot"
│   ├── verified_boot/
│   │   ├── enabled             # 0 or 1
│   │   └── key_hash            # SHA256 of root public key
│   ├── vendor_specific/        # Vendor extensions
│   │   ├── intel_bootguard/
│   │   │   ├── acm_protected   # 0 or 1
│   │   │   └── btg_profile     # "production", "debug"
│   │   ├── amd_psb/
│   │   └── arm_xyz/
│   └── status                  # "active", "disabled", "not_provisioned"
├── drtm/                       # Vendor-agnostic HW RoT interface
```

Move (some of) the checks done by `fwupd HSI` into kernel?

Going further: firmware security interface?

Use Case: Verify platform is using your Intel Boot Guard key

Intel Boot Guard:

```
# Kernel reads Key Manifest from FIT
cat /sys/firmware/security/srtm/verified_boot/key_hash
a7f3d2c1b8e9... (your provisioned key hash)
```

AMD Platform Secure Boot:

```
# Kernel queries PSP root key from fuses
cat /sys/firmware/security/srtm/verified_boot/key_hash
3c8d9f2e1a7b... (your provisioned key hash)
```

- Attestation: prove platform uses specific key
- Supply chain security: verify OEM provisioned correct key

Q&A

Parsing PCI config space (ME) 🔴 #3

Example: Reading Intel Boot Guard OTP Fuse Status

```
const guint hfs_cfg_addrs[] = {0x0, 0x40, 0x48, 0x60, 0x64, 0x68, 0x6c}
```

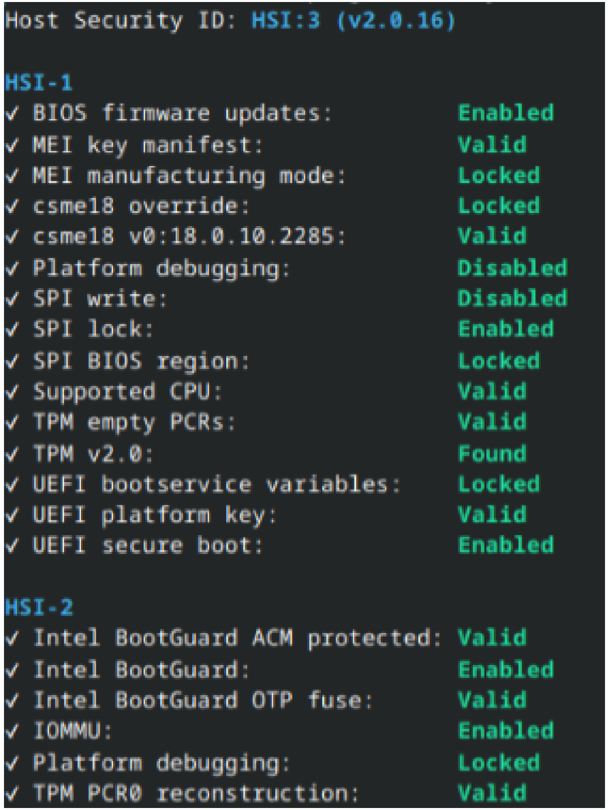
```
struct FuMeiCsme18Hfsts6 {
    _reserved0: u21,               // bits 0-20: reserved/unused
    _manufacturing_lock: u1,       // bit 21
    _reserved1: u8,                // bits 22-29: reserved
    fpf_soc_configuration_lock: u1, // bit 30: ⭐ OTP fuse lock status
    _sx_resume_type: u1,           // bit 31
}
```

```
if (!fu_mei_csme18_hfsts6_get_fpf_soc_configuration_lock(hfsts6)) {
    // OTP fuse check FAILS - user sees ❌ in HSI report
    fwupd_security_attr_set_result(attr, FWUPD_SECURITY_ATTR_RESULT_NOT_VALID);
    fwupd_security_attr_add_flag(attr, FWUPD_SECURITY_ATTR_FLAG_ACTION_CONTACT_OEM);
}
```

Intel Boot Guard

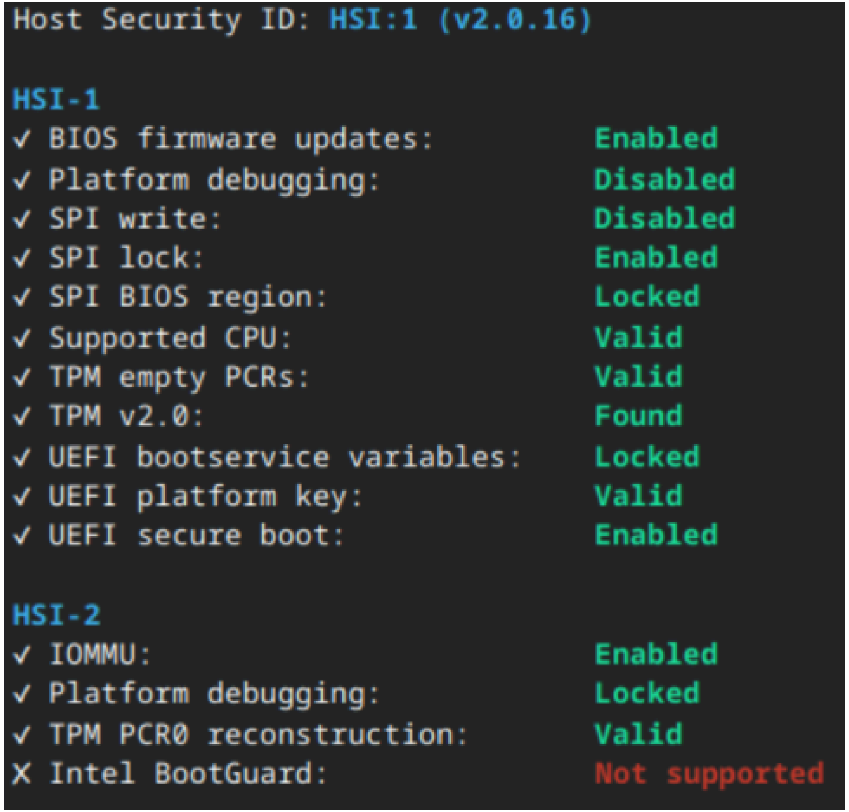
- Hardware-based boot integrity protection
- Prevents the machine from running firmware images not released (signed) by the system vendor
- It forms a Root of Trust for Verification (RTV) and Static Root of Trust for Measurement (S-RTM) by fusing cryptographic keys into hardware

Intel Boot Guard and Management Engine



```
Host Security ID: HSI:3 (v2.0.16)

HSI-1
✓ BIOS firmware updates:          Enabled
✓ MEI key manifest:               Valid
✓ MEI manufacturing mode:         Locked
✓ csme18 override:                Locked
✓ csme18 v0:18.0.10.2285:         Valid
✓ Platform debugging:             Disabled
✓ SPI write:                      Disabled
✓ SPI lock:                       Enabled
✓ SPI BIOS region:                Locked
✓ Supported CPU:                  Valid
✓ TPM empty PCRs:                 Valid
✓ TPM v2.0:                       Found
✓ UEFI bootservice variables:     Locked
✓ UEFI platform key:              Valid
✓ UEFI secure boot:               Enabled

HSI-2
✓ Intel BootGuard ACM protected: Valid
✓ Intel BootGuard:                Enabled
✓ Intel BootGuard OTP fuse:       Valid
✓ IOMMU:                          Enabled
✓ Platform debugging:             Locked
✓ TPM PCR0 reconstruction:        Valid
```

ME enabled



```
Host Security ID: HSI:1 (v2.0.16)

HSI-1
✓ BIOS firmware updates:          Enabled
✓ Platform debugging:             Disabled
✓ SPI write:                      Disabled
✓ SPI lock:                       Enabled
✓ SPI BIOS region:                Locked
✓ Supported CPU:                  Valid
✓ TPM empty PCRs:                 Valid
✓ TPM v2.0:                       Found
✓ UEFI bootservice variables:     Locked
✓ UEFI platform key:              Valid
✓ UEFI secure boot:               Enabled

HSI-2
✓ IOMMU:                          Enabled
✓ Platform debugging:             Locked
✓ TPM PCR0 reconstruction:        Valid
✗ Intel BootGuard:                Not supported
```

ME disabled

Alternative ways of checking Intel Boot Guard configuration

ME HFSTS registers cached in SMBIOS

```
Handle 0x0031, DMI type 219, 106 bytes
OEM-specific Type
  Header and Data:
    DB 6A 31 00 01 04 01 55 02 00 90 00 81 00 60 30
    00 00 00 00 00 00 00 03 1F D6 02 00 00 00 00 02
    00 00 00 80 00 00 00 00 00 00 00 00 00 00 00 00
    00 00 00 00 00 00 00 00 03 00 00 00 80 00 00 00
    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    00 04 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    00 00 00 00 00 00 00 00 00 00
  Strings:
    MEI1
    MEI2
    MEI3
    MEI4
```

See this issue for details.