Firmware Biopsy

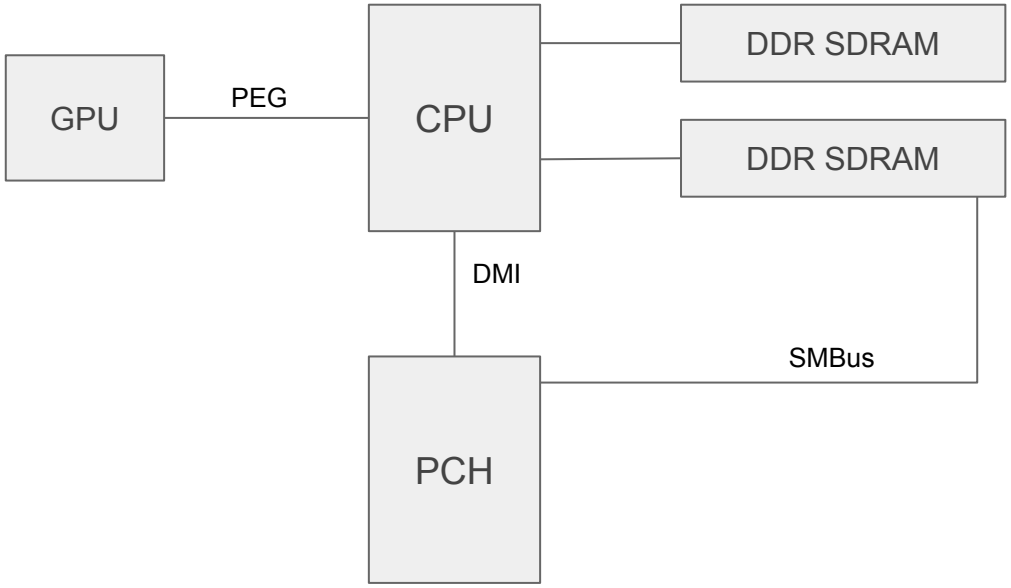
tweek <tweek@google.com>
Enterprise Infrastructure Protection

Agenda

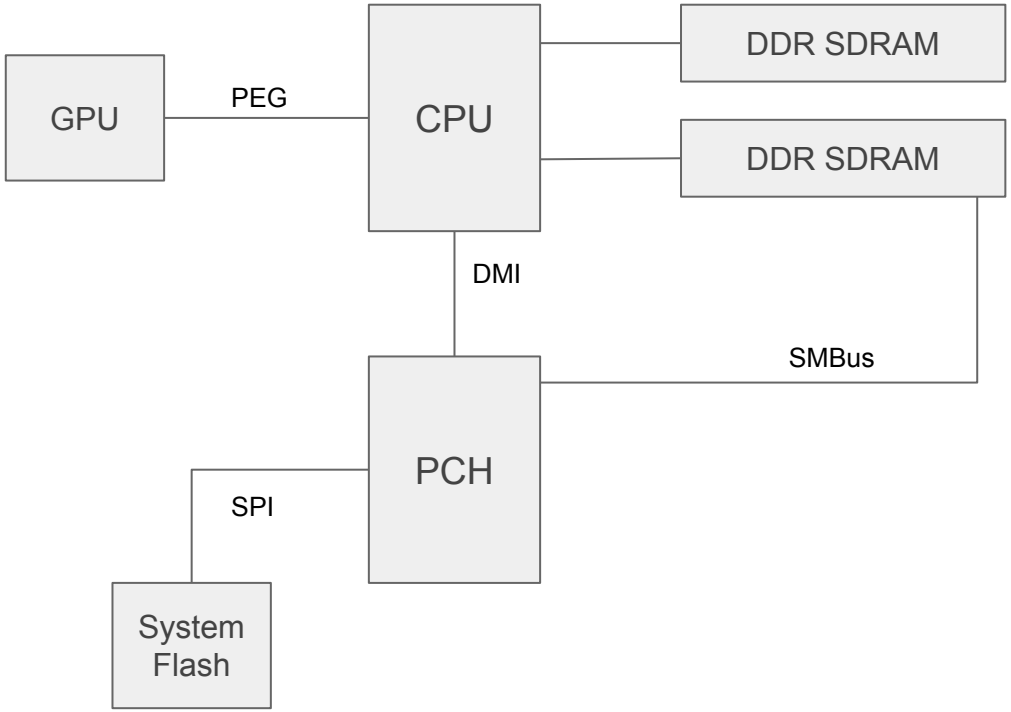
- Context
- Read Primitives
- More Read Primitives
- Collection at Scale
- Findings
- Q & A

Context

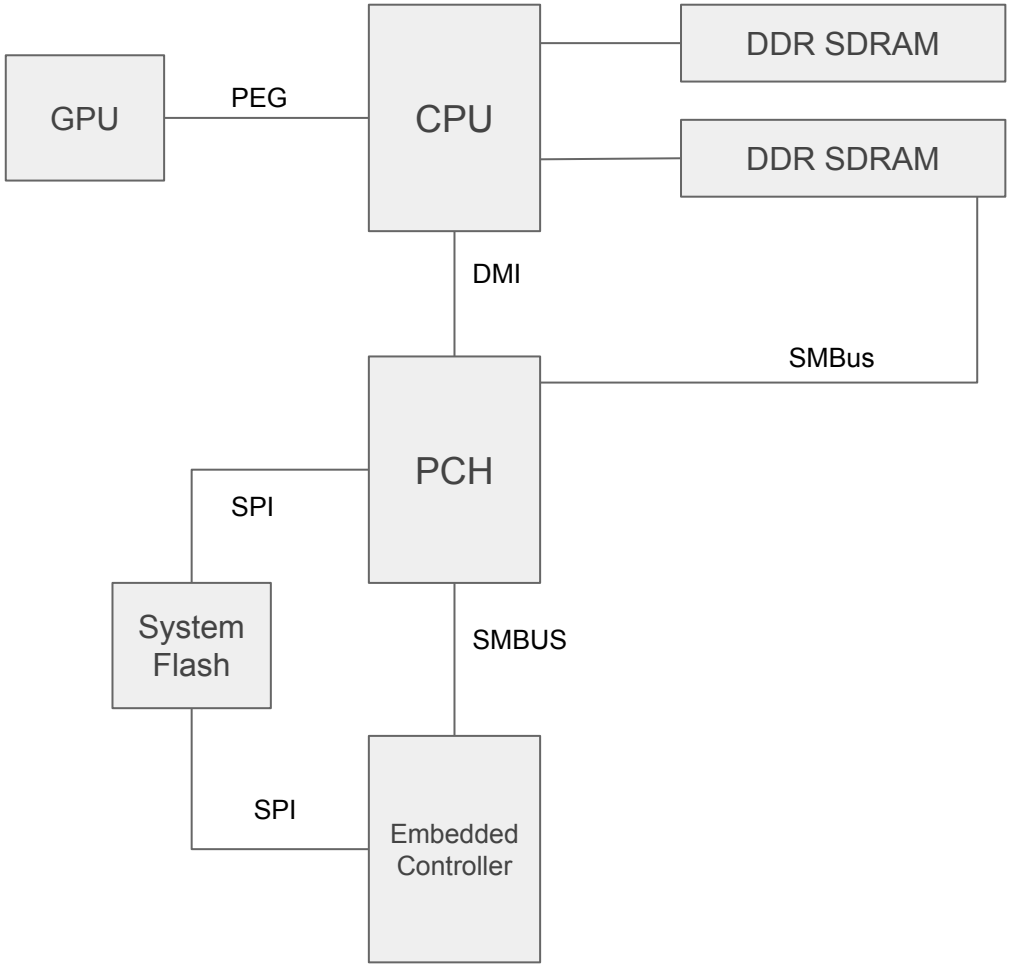
x86 in 2016 (Skylake)



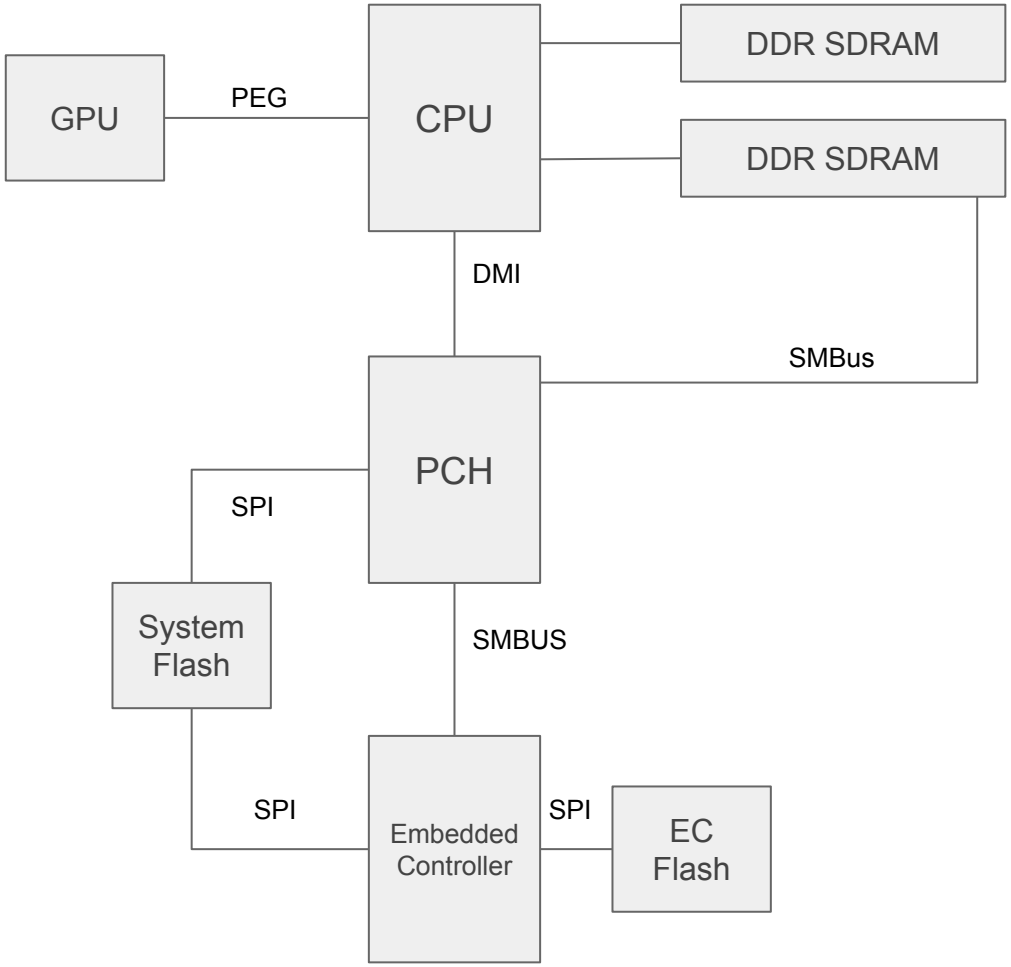
x86 in 2016 (Skylake)



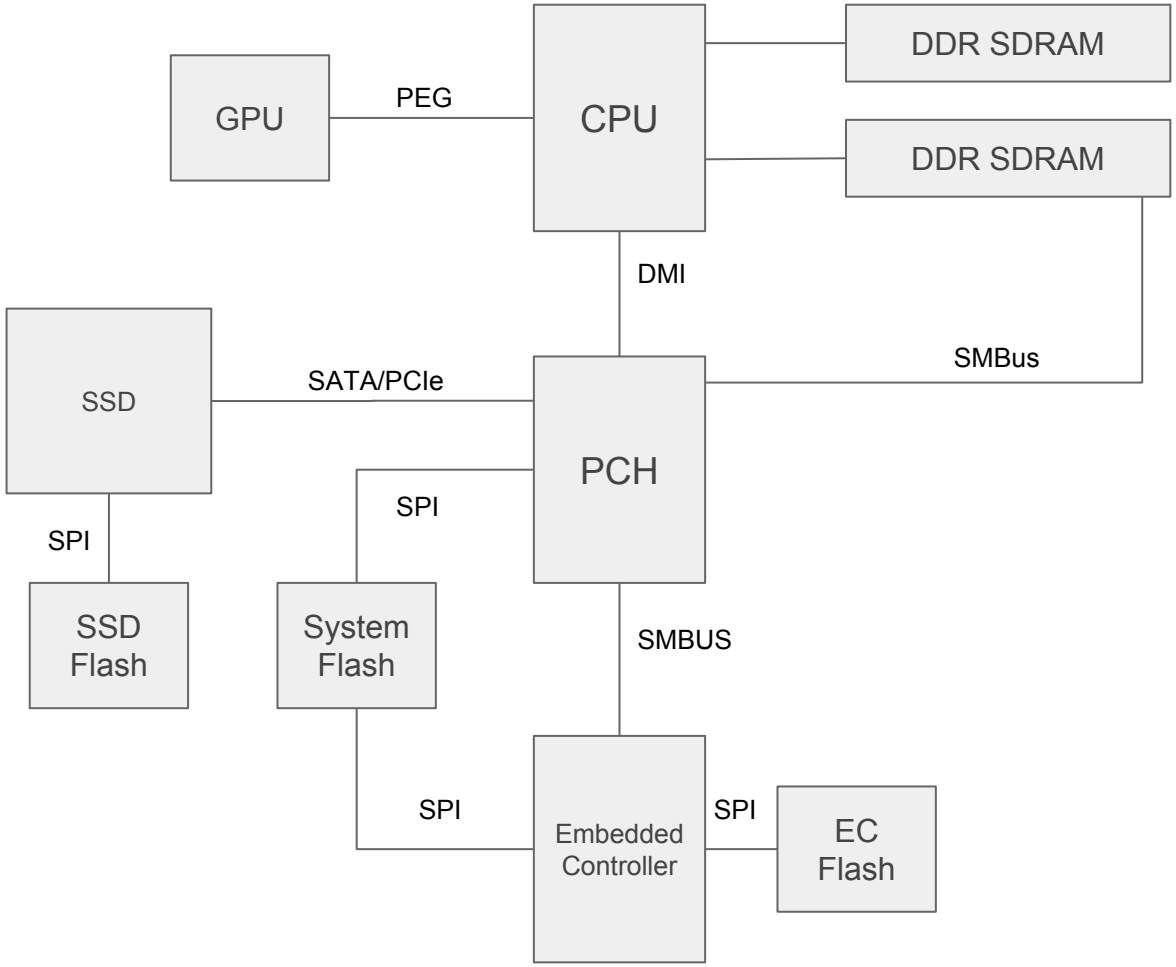
x86 in 2016 (Skylake)



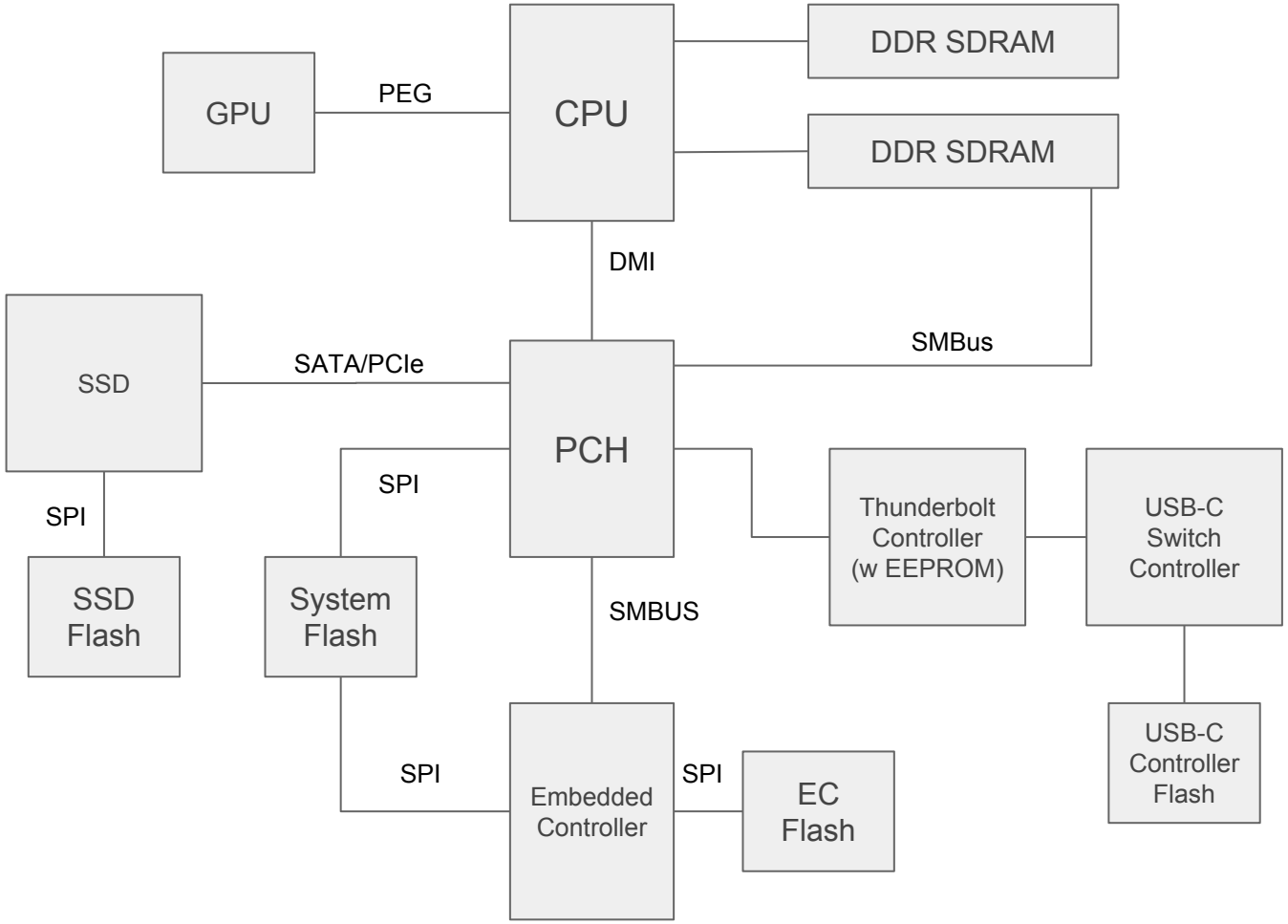
x86 in 2016 (Skylake)



x86 in 2016 (Skylake)

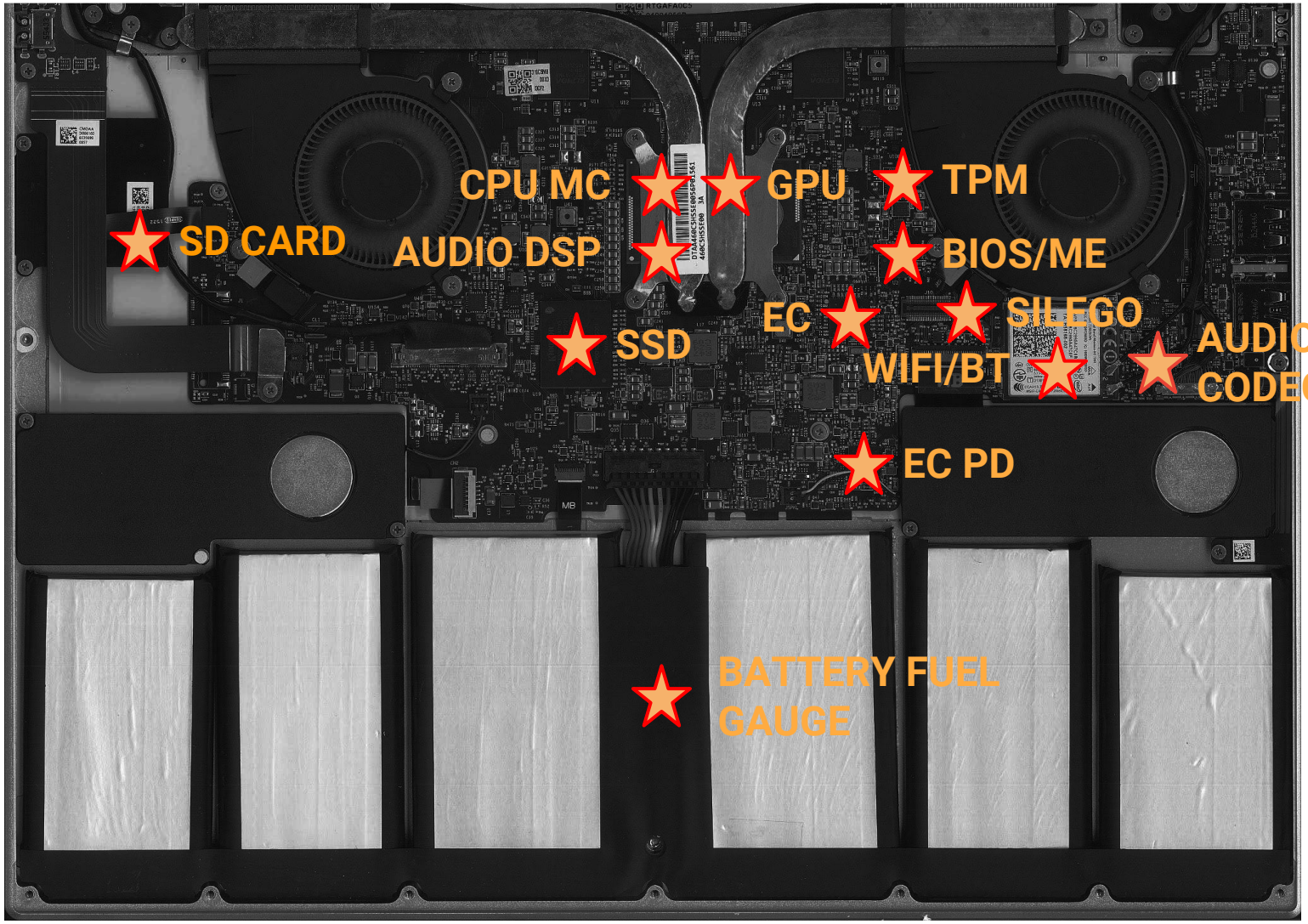


x86 in 2016 (Skylake)



Firmware on Pixel 2

Firmware on Pixel 2



Research

Firmware / Topic	Published Research
PCI Option ROM	Heasman (2007) Snare (2013) Kovah & LegbaCore (2015)
Hard Drive	Goodspeed et al. (2013)
Network Controller	Triulzi (2008)
x86 Modes and Design flaws	Rutkowska (2009-2015) Cr4sh SMM research (2015-2016)

What is happening in the wild?

- State-sponsored attackers exploiting firmware implants
  - Equation Group, IRATEMONK, DEITYBOUNCE

- Non-state-sponsored attackers picking up
  - Hacking Team

Why is this attractive to attackers?

- High initial investment, but lasts for a long time
- Very low chance of detection
- Remote deployment or hardware interception is still easy

What do defenders want?

Increase costs of performing firmware attacks

- Removing trivial to find security flaws
- Increasing chance of detection in the wild
- Reduce length of time you can expect capability will last before being disclosed

Ultimately, protecting our users and their data.

Improving the state of detection

Increase knowledge & visibility

- Where is firmware running?
- What firmware is running?
- Is that the firmware intended to be run by the vendor?
- Does this firmware contains known vulnerability?

How to verify that a fleet of devices is running the original vendor firmware?

Read Primitives

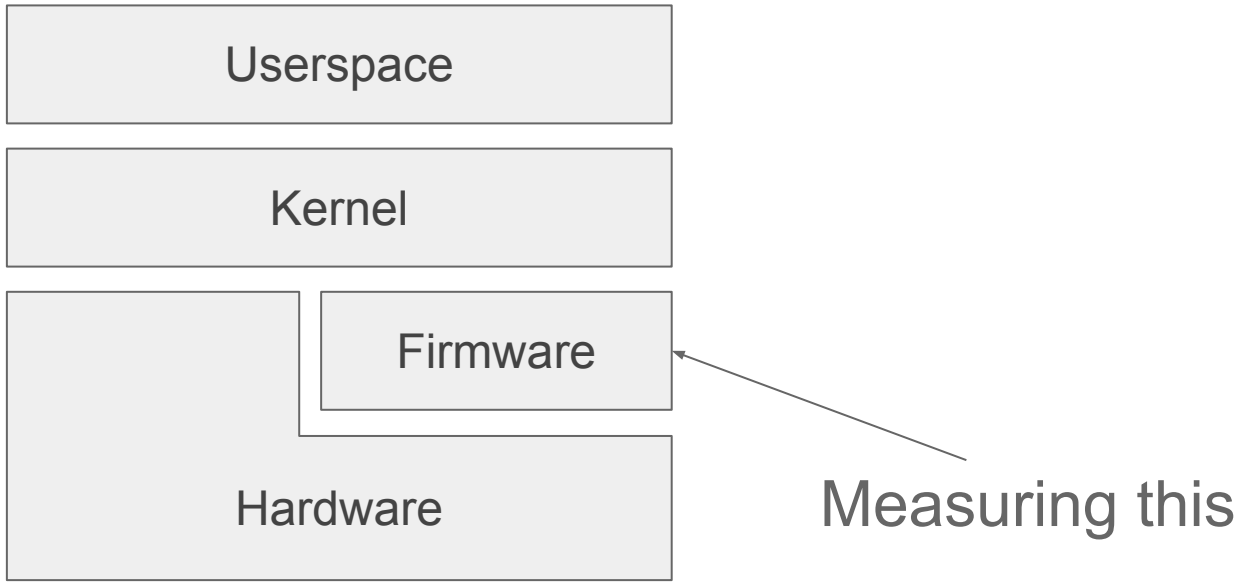
Read Primitive

- Method to extract a copy of the running firmware
  - Reliable
  - Generic
  - Complete
- Physical vs Software
  - Trade-off between integrity and scalability of measurement
  - Physical: hook onto pins = easiest, not practical for internal flash
- Limited read primitive
  - Hash of firmware
  - Partial copy

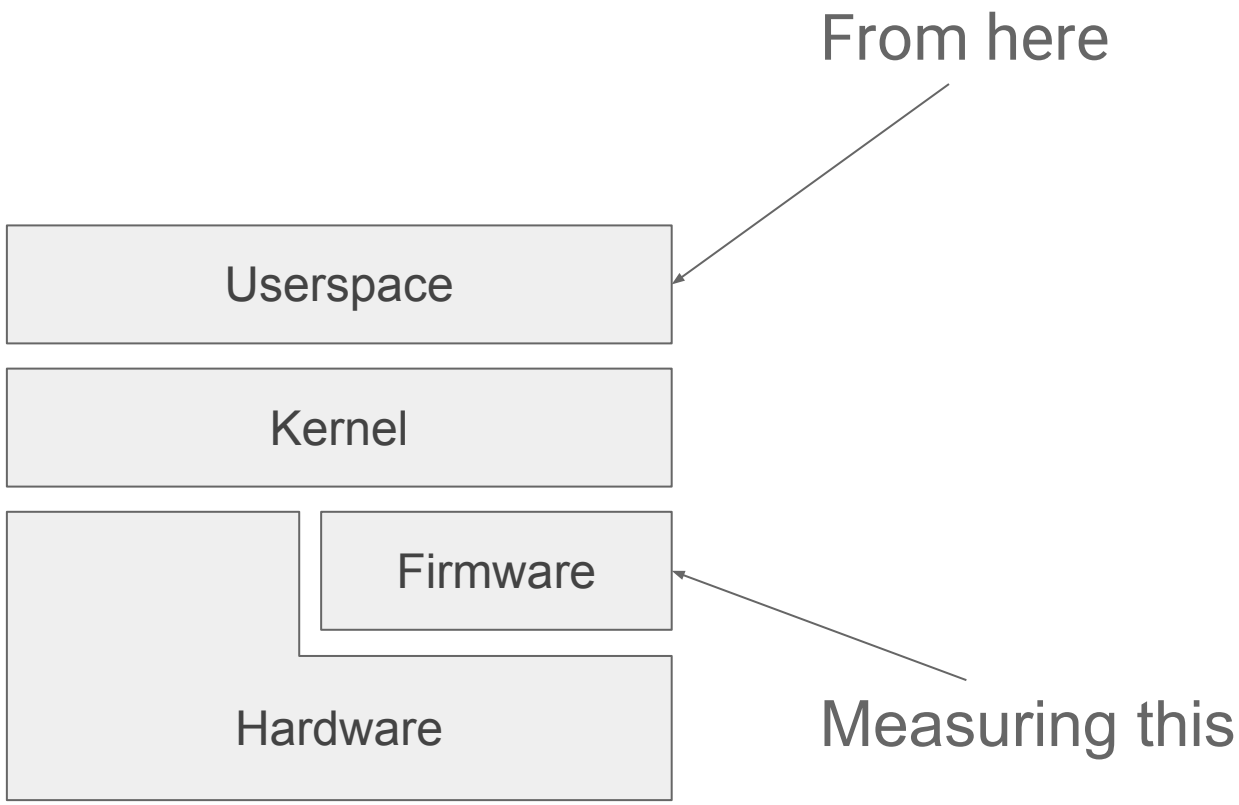
Read Primitive (cont’d)

- Detection method more than prevention
- PCR of TPM
  - Similar objective
  - Partial measure of boot environment
  - Limited to boot path
  - Preventative method

Software Read Primitive Flaw



Software Read Primitive Flaw



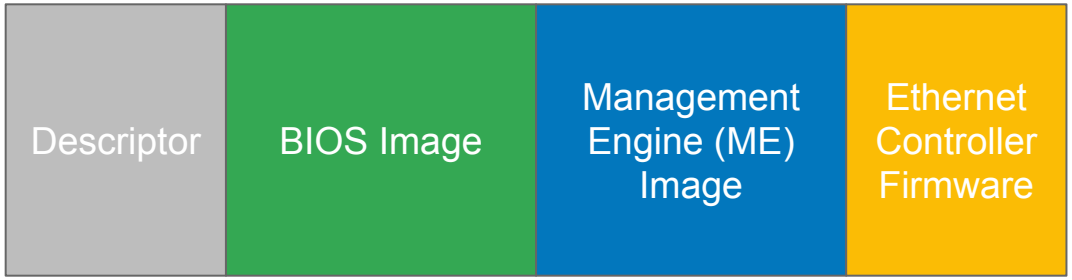
One solution

- Similar flaw in today live forensic
  - Investigate the OS from the running kernel
- Increase the number and type of measures
  - For a specific firmware => have two or more read primitives
  - Increase the cost of hiding for an attacker

BIOS/UEFI Read Primitives

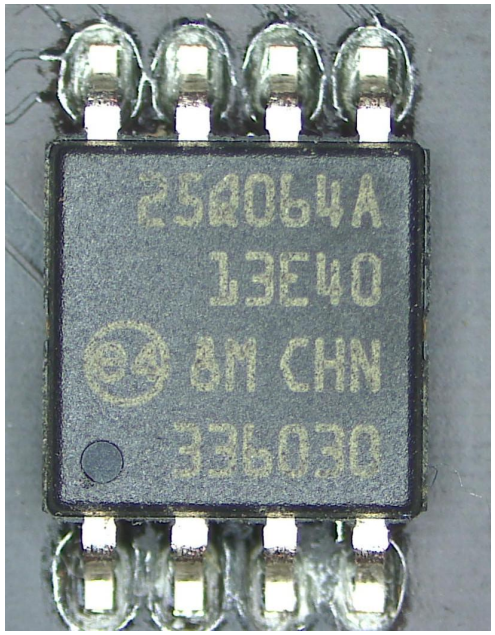
BIOS/UEFI

- The most well-known firmware
- Stored on the SPI flash

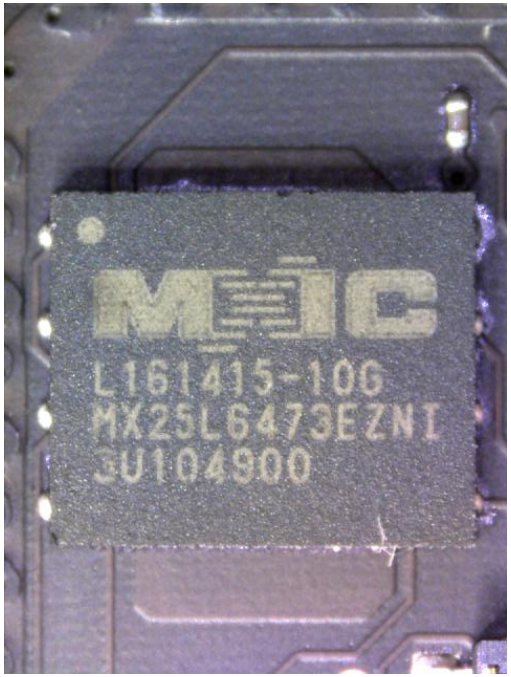


- Descriptor defines access control between regions
- All latest chipset generation follow a specific Intel standard for their format

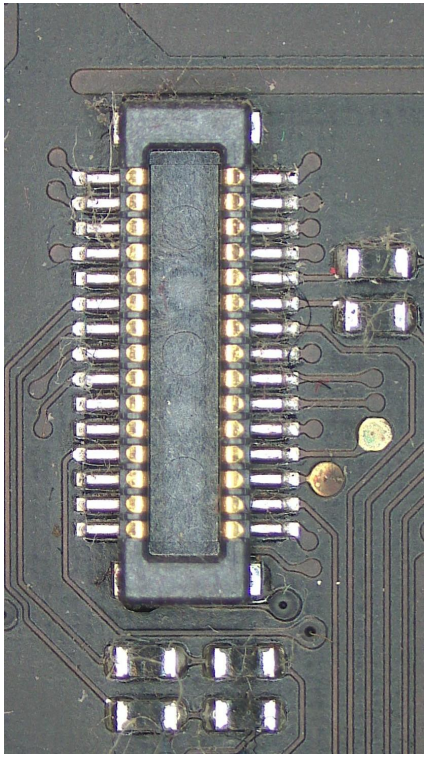
SPI Flash



8-PIN SOP

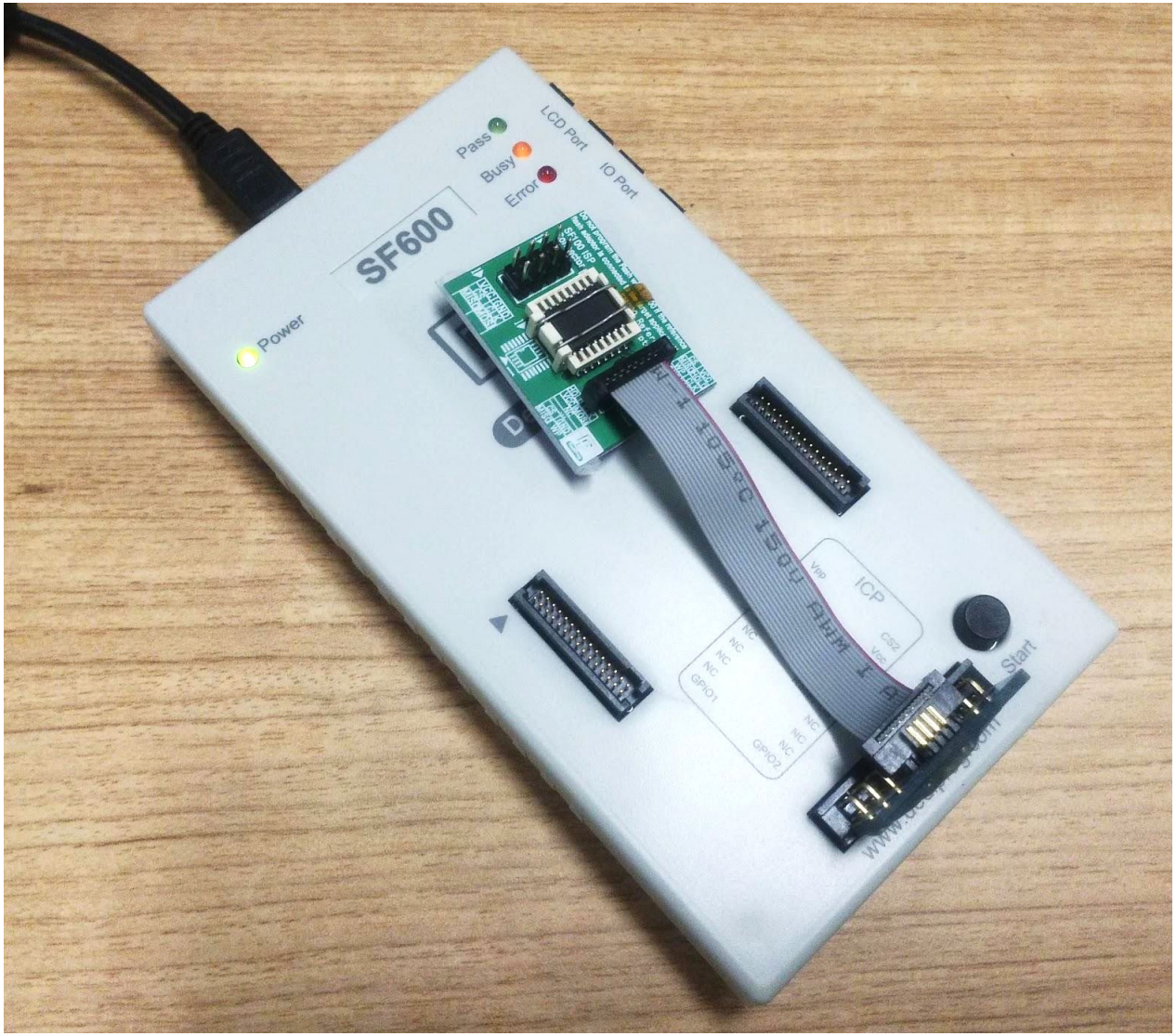


8-PIN WSON



Debug Header

Hardware Acquisition



BIOS/UEFI Read Primitive (SPIBAR)

- PCI device exposed by the PCH
- Interact with the flash using memory access
- Used by Flashrom and Chipsec
- Multiple modes
  - Software sequencing:
    Deprecated, forward white-listed operations to the flash
  - Hardware sequencing:
    PCH offers standard “API” to interact with flash

BIOS/UEFI Read Primitive (SPIBAR)

21.1 Serial Peripheral Interface Memory Mapped Configuration Registers

The SPI Host Interface registers are memory-mapped in the RCRB (Root Complex Register Block) Chipset Register Space with a base address (SPIBAR) of 3800h and are located within the range of 3800h to 39FFh. The address for RCRB are in the RCBA Register (see Section 12.1.40). The individual registers are then accessible at SPIBAR + Offset as indicated in the following table.

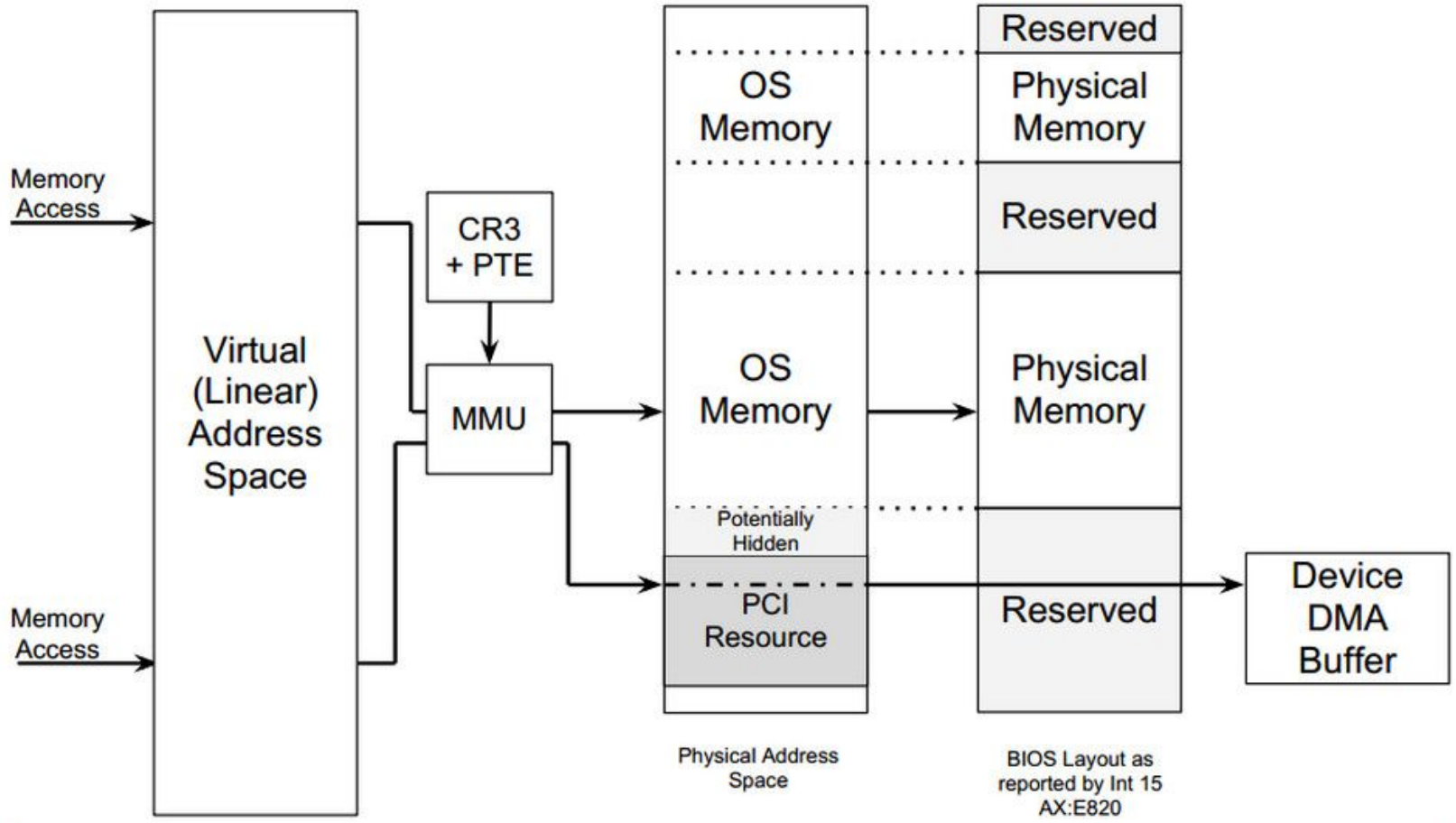
These memory mapped registers must be accessed in byte, word, or DWord quantities.

Table 21-1. Serial Peripheral Interface (SPI) Register Address Map (SPI Memory Mapped Configuration Registers) (Sheet 1 of 2)

SPIBAR + Offset	Mnemonic	Register Name	Default
00h–03h	BFPR	BIOS Flash Primary Region	00000000h
04h–05h	HSFS	Hardware Sequencing Flash Status	0000h
06h–07h	HSFC	Hardware Sequencing Flash Control	0000h
08h–0Bh	FADDR	Flash Address	00000000h
0Ch–0Fh	—	Reserved	00000000h
10h–13h	FDATA0	Flash Data 0	00000000h
14h–4Fh	FDATAN	Flash Data N	00000000h
50h–53h	FRAP	Flash Region Access Permissions	00000202h
54h–57h	FREG0	Flash Region 0	00000000h

[1]

Memory-mapped I/O



[2]

/dev/mem

- CONFIG_STRICT_DEVMEM ?
- Access to MMIO for uid 0 is allowed
- OSX and Windows requires extra driver for such access

SPIBAR example

```
tweek@tweek-z420-l:~$ sudo lspci -xxx -v -s 00:1f.0
00:1f.0 ISA bridge: Intel Corporation C600/X79 series chipset LPC Controller (rev 05)
        Subsystem: Hewlett-Packard Company Device 1589
        Flags: bus master, medium devsel, latency 0
        Capabilities: [e0] Vendor Specific Information: Len=0c <?>
        Kernel driver in use: lpc_ich
00: 86 80 41 1d 07 01 10 02 05 00 01 06 00 00 80 00
10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
20: 00 00 00 00 00 00 00 00 00 00 00 00 3c 10 89 15
30: 00 00 00 00 e0 00 00 00 00 00 00 00 00 00 00 00
40: 01 04 00 00 80 00 00 00 01 05 00 00 11 00 00 00
50: f8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
60: 8b 87 84 83 d0 00 00 00 85 80 8b 8a ff 00 00 00
70: 78 f0 78 f0 78 f0 78 f0 78 f0 78 f0 78 f0 78 f0
80: 10 00 0e 3f 01 06 fc 00 01 08 fc 00 00 00 00 00
90: 00 00 00 00 00 0f 00 00 00 00 00 00 00 00 00 00
a0: 18 0a 20 00 48 3d 06 00 00 47 00 00 00 03 00 80
b0: 00 00 00 00 00 00 00 00 00 10 01 00 00 00 00 00
c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
d0: 33 22 11 00 67 45 00 00 cf ff 00 00 2a 00 00 00
e0: 09 00 0c 10 00 00 00 00 91 02 64 0c 00 00 00 00
f0: 01 c0 d1 fe 00 00 00 00 87 0f 07 08 00 00 00 00
```

SPIBAR example

```
tweek@tweek-z420-l:~$ sudo lspci -xxx -v -s 00:1f.0
00:1f.0 ISA bridge: Intel Corporation C600/X79 series chipset LPC Controller (rev 05)
        Subsystem: Hewlett-Packard Company Device 1589
        Flags: bus master, medium devsel, latency 0
        Capabilities: [e0] Vendor Specific Information: Len=0c <?>
        Kernel driver in use: lpc_ich
00: 86 80 41 1d 07 01 10 02 05 00 01 06 00 00 80 00
10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
20: 00 00 00 00 00 00 00 00 00 00 00 00 3c 10 89 15
30: 00 00 00 00 e0 00 00 00 00 00 00 00 00 00 00 00
40: 01 04 00 00 80 00 00 00 01 05 00 00 11 00 00 00
50: f8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
60: 8b 87 84 83 d0 00 00 00 85 80 8b 8a ff 00 00 00
70: 78 f0 78 f0 78 f0 78 f0 78 f0 78 f0 78 f0 78 f0
80: 10 00 0e 3f 01 06 fc 00 01 08 fc 00 00 00 00 00
90: 00 00 00 00 00 0f 00 00 00 00 00 00 00 00 00 00
a0: 18 0a 20 00 48 3d 06 00 00 47 00 00 00 03 00 80
b0: 00 00 00 00 00 00 00 00 00 10 01 00 00 00 00 00
c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
d0: 33 22 11 00 67 45 00 00 cf ff 00 00 2a 00 00 00
e0: 09 00 0c 10 00 00 00 00 91 02 64 0c 00 00 00 00
f0: 01 c0 d1 fe 00 00 00 00 87 0f 07 08 00 00 00 00
```

SPIBAR is at: 0xfed1c000 + 0x3800 (constant) = 0xfed1f800

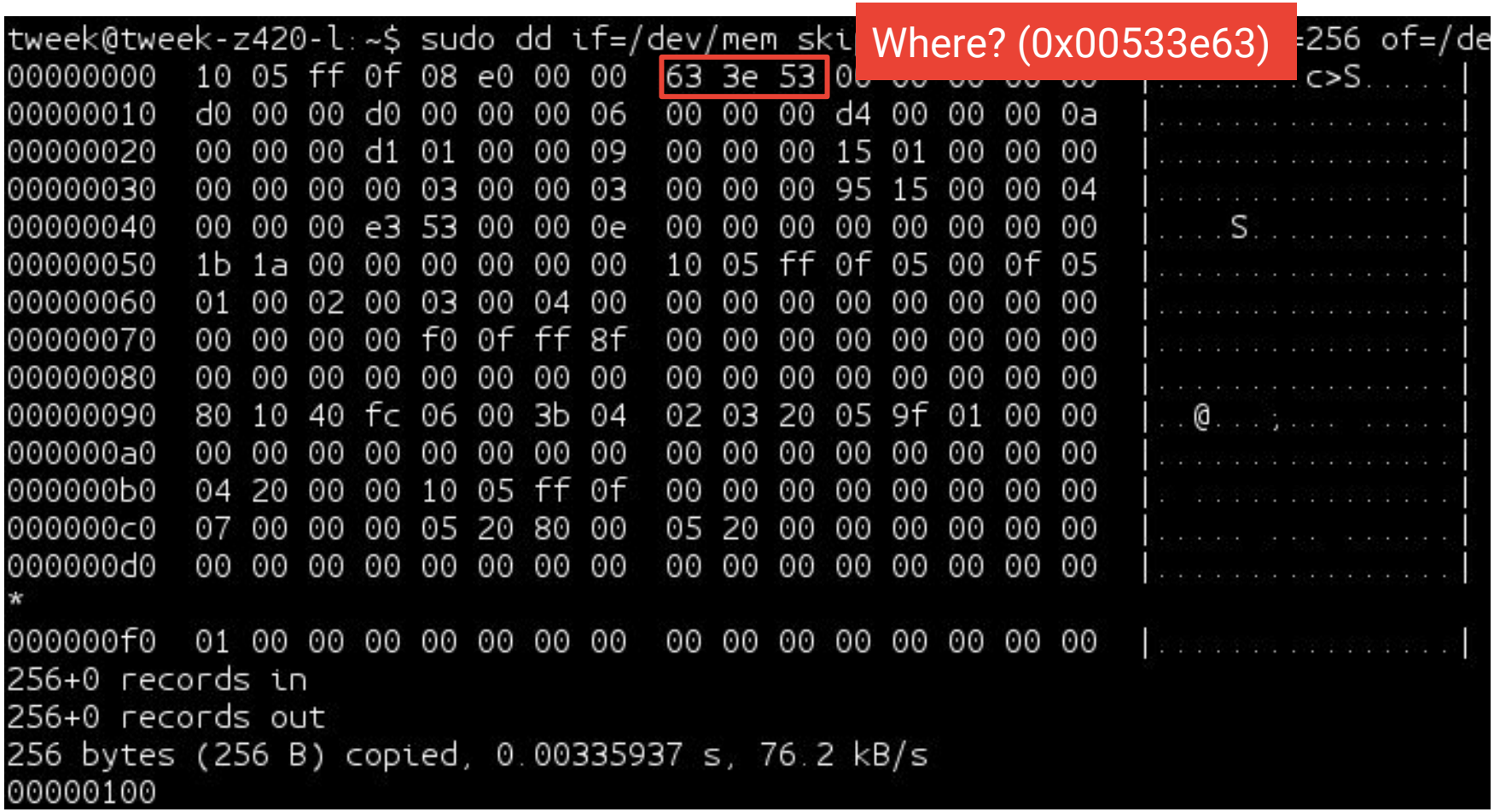
SPIBAR example

```
tweek@tweek-z420-l:~$ cat /proc/iomem  | tail
fed08000-fed08fff : pnp 00:0b
fed1c000-fed3ffff : reserved
  fed1c000-fed1ffff : pnp 00:0b
    fed1f410-fed1f414 : iTCO_wdt
fed45000-fedfffff : pnp 00:00
fee00000-fee00fff : Local APIC
ff000000-ffffffff : reserved
  ff000000-ffffffff : pnp 00:0b
100000000-82fffffff : System RAM
380000000000-38007fffffff : PCI Bus 0000:00
```

SPIBAR example

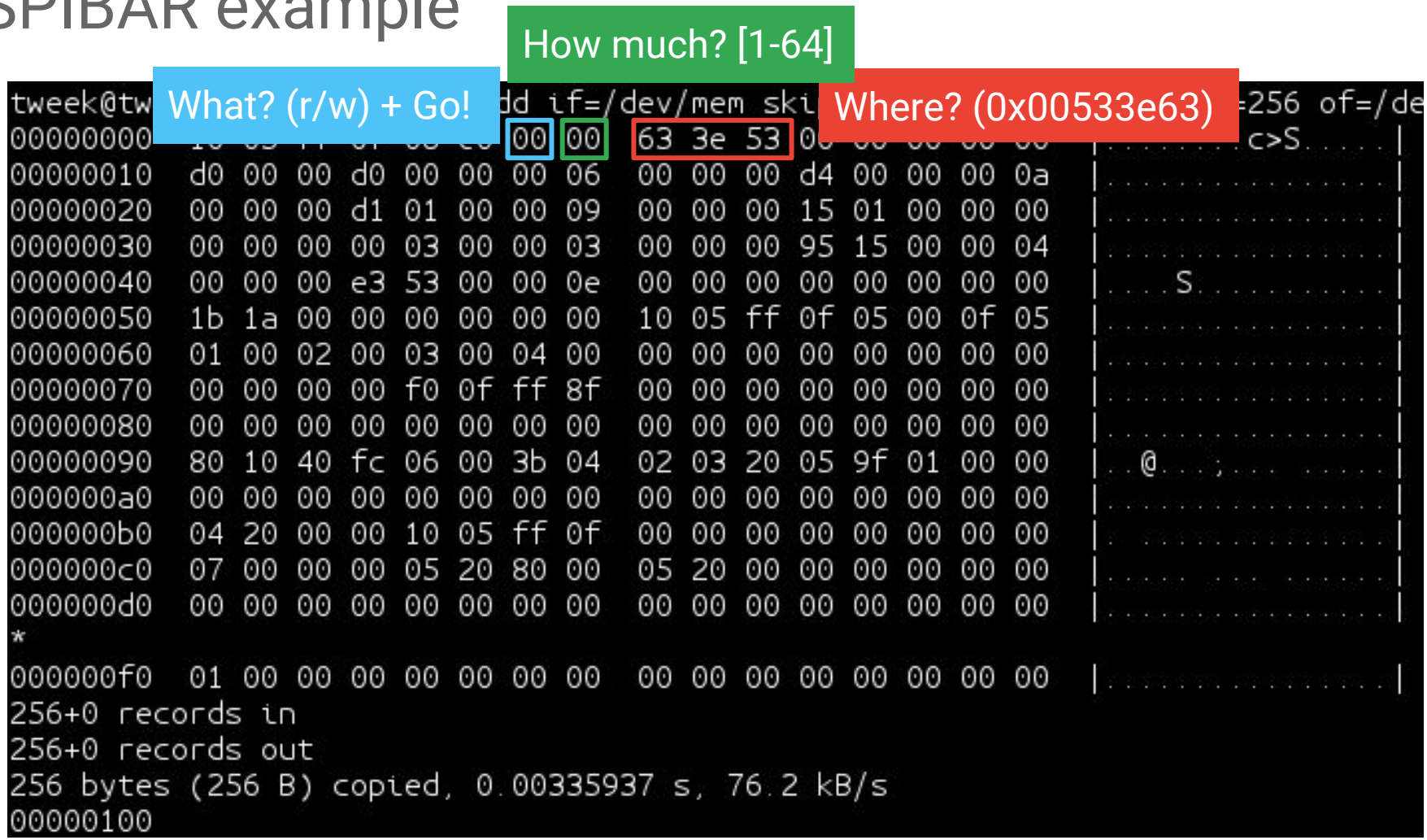
```
tweek@tweek-z420-l:~$ sudo dd if=/dev/mem skip=4275173376 bs=1 count=256 of=/de
00000000  10 05 ff 0f 08 e0 00 00  63 3e 53 00 00 00 00 00  |........c>S.....|
00000010  d0 00 00 d0 00 00 00 06  00 00 00 d4 00 00 00 0a  |................|
00000020  00 00 00 d1 01 00 00 09  00 00 00 15 01 00 00 00  |................|
00000030  00 00 00 00 03 00 00 03  00 00 00 95 15 00 00 04  |................|
00000040  00 00 00 e3 53 00 00 0e  00 00 00 00 00 00 00 00  |....S...........|
00000050  1b 1a 00 00 00 00 00 00  10 05 ff 0f 05 00 0f 05  |................|
00000060  01 00 02 00 03 00 04 00  00 00 00 00 00 00 00 00  |................|
00000070  00 00 00 00 f0 0f ff 8f  00 00 00 00 00 00 00 00  |................|
00000080  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000090  80 10 40 fc 06 00 3b 04  02 03 20 05 9f 01 00 00  |..@...;... .....|
000000a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000000b0  04 20 00 00 10 05 ff 0f  00 00 00 00 00 00 00 00  |. ..............|
000000c0  07 00 00 00 05 20 80 00  05 20 00 00 00 00 00 00  |..... ... ......|
000000d0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
000000f0  01 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
256+0 records in
256+0 records out
256 bytes (256 B) copied, 0.00335937 s, 76.2 kB/s
00000100
```

SPIBAR example

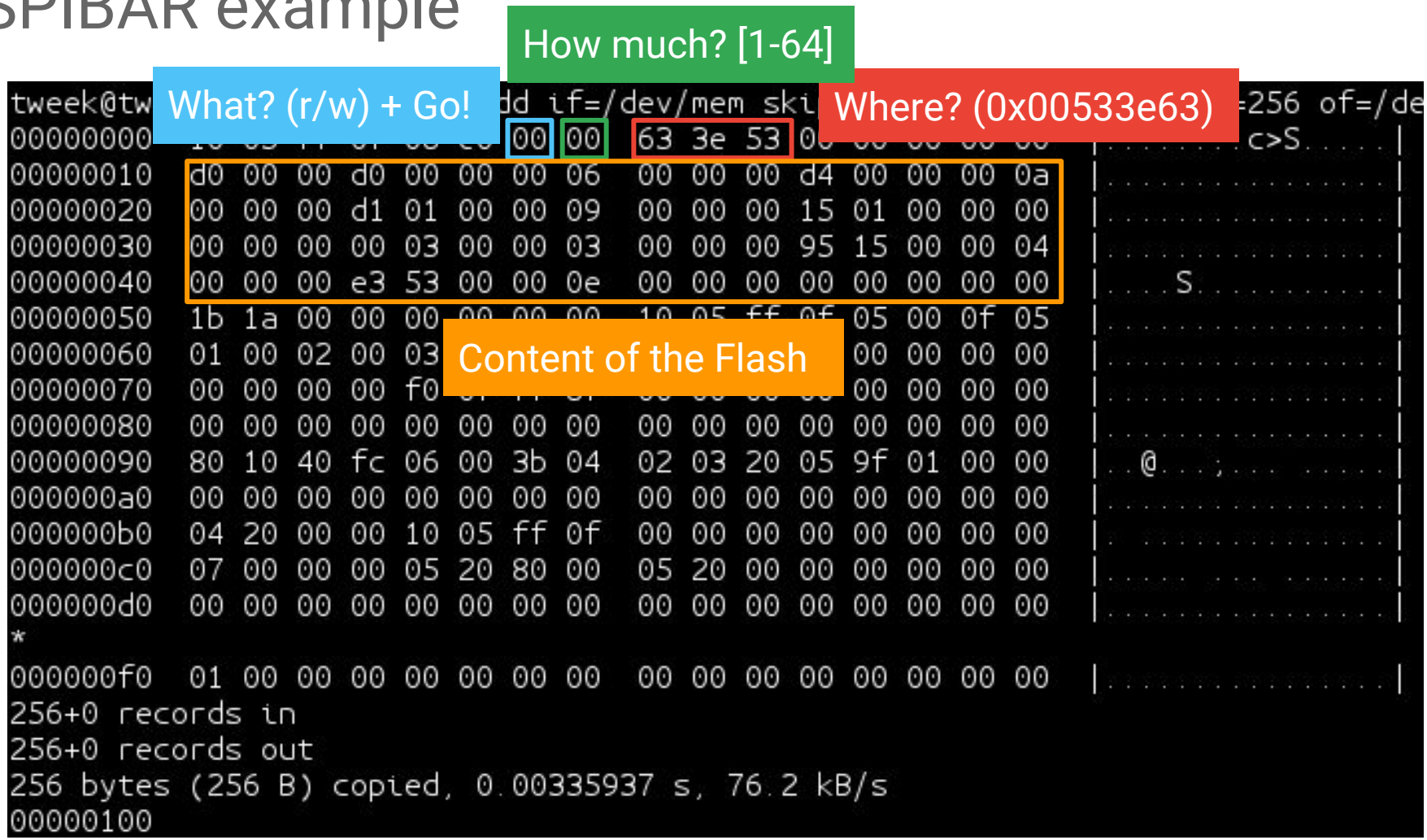


SPIBAR example

SPIBAR example



SPIBAR example



BIOS/UEFI Read Primitive (0xFF000000)

- 16MB forwarded to the PCH
- “For security reasons, the processor will positively decode this range to DMI. This positive decode ensures any overlapping ranges will be ignored. This ensures that the boot vector and BIOS execute off the PCH.” - Intel Skylake datasheet

PCH caching?

More Read Primitives

PCI Option ROM

- Stored on the PCI device
- Executed by CPU when the device is initialised
- By design, execution of unknown code
- Leveraged by Thunderstrike

GPU Read Primitives

- Multiple memory areas
  - VRAM
  - PCI Option ROM
  - GPU firmware
- Documentation from Nouveau project
  - Describes low-level interface of cards
  - Highly dependent on card generation

Embedded Controller

- Manage battery, fans, sensors
- No standard interface
  - ACPI define two IO port
  - Index I/O for extra reads
- Moving proprietary tech from BIOS to EC
  - Lenovo’s ThinkEngine
  - Apple’s SMC
- Chrome OS
  - Open Source EC
  - Read primitive available using flashrom (in dev mode)

Collection at Scale

Chipsec

- From Intel Advanced Threat research, published in 2014
- https://github.com/chipsec/chipsec
- Allow inspection of hardware/firmware
- By default, requires kernel driver
- /dev/mem is enough for PCI memory access
- Port to OSX for similar functionalities

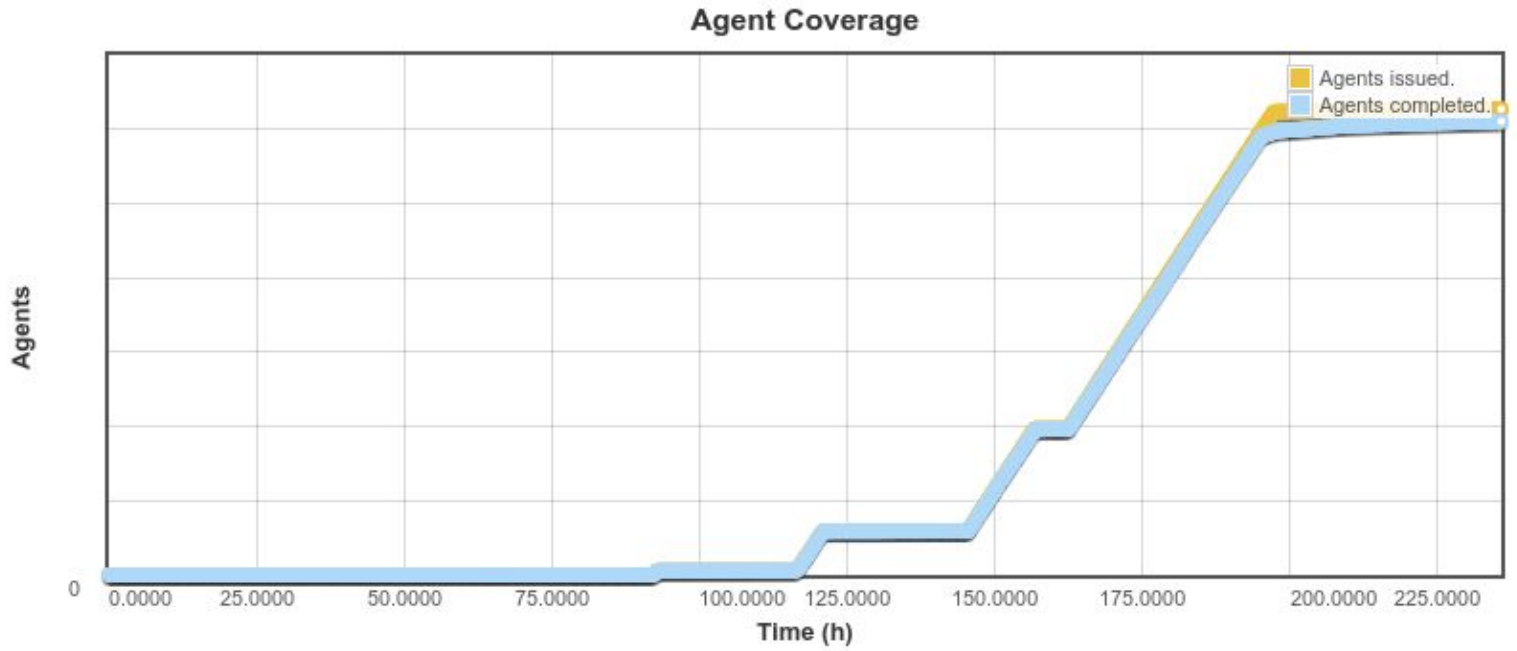
GRR

- Google's IR tool
- Open Source, https://github.com/google/grr
- Highly customizable
  - Integrate Sleuthkit for live disk forensic
  - Integrate Rekall for memory forensic
- Stable
- Design for low-impact (memory footprint) on client

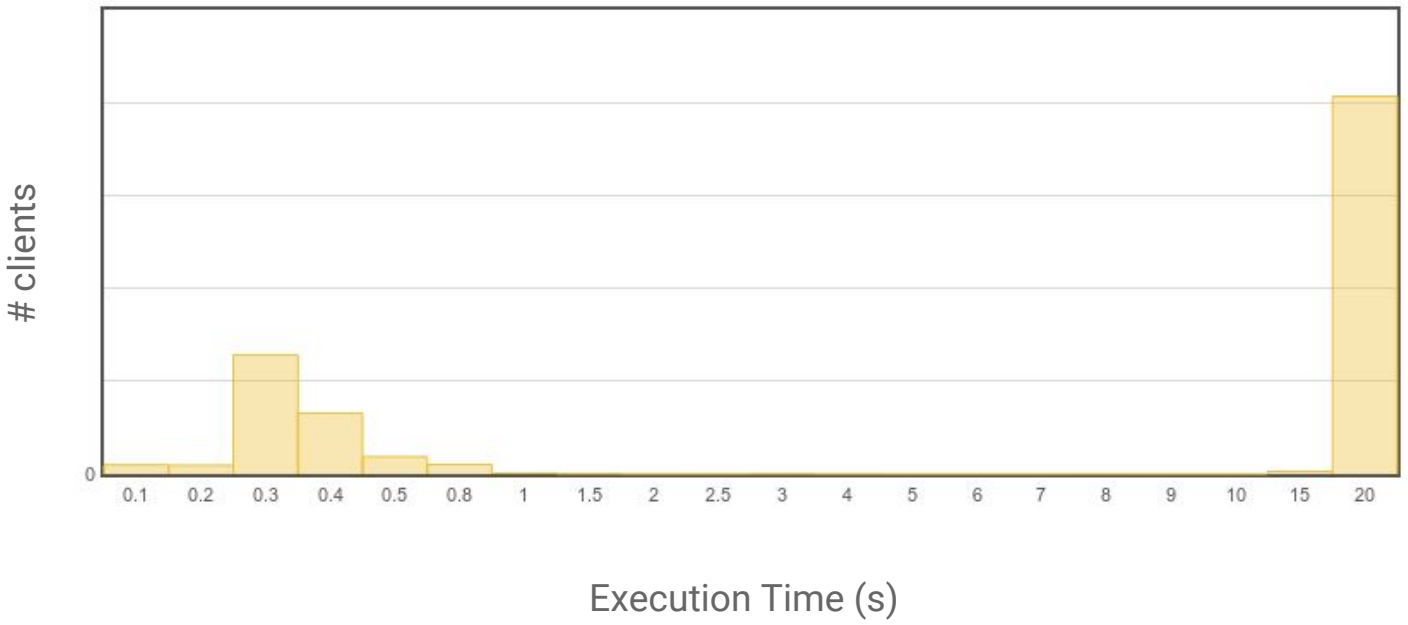
GRR Chipsec

- Integrate Chipsec to GRR
- Open Source since April
- Implemented as a GRR component
- Able to dump the SPI flash image
- Able to inspect hardware/firmware status
  - Quickly extend the functionality in case of incident or public release

GRR Chipsec - BIOS collection



GRR Chipsec - BIOS collection



What can go wrong?

- Unsupported platform
  - Older generation only supports software sequencing
  - Unsupported hardware by Chipsec
  - Execution on a VM
- Lack of space to ...
  - Load Chipsec
  - Dump the flash image

Analysis

Comparison

- With what?
  - Previous versions from the same host
  - Official version
  - Other machine with the same BIOS version
  - Different read primitives

Granularity

- Considering one blob and hash
  - Lots of noise
  - E.g., BIOS contains variable areas, all flash images will be different
- Deconstructing the blob
  - Vendor specific format
  - Extra care to consider “in-between” regions
  - Some regions will still be out of analysis
  - May need to run control flow analysis to uncover similar code

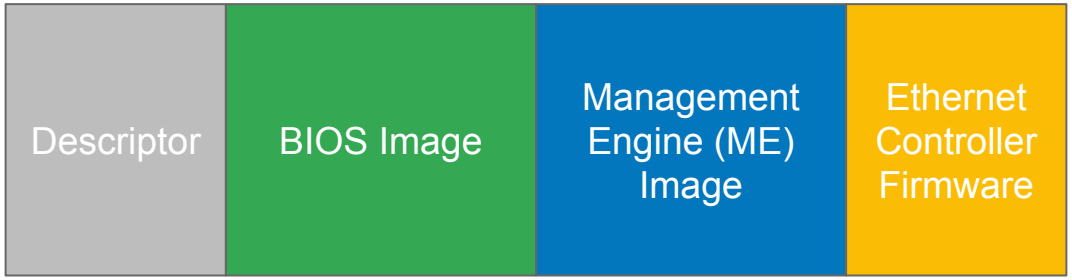
Implementation

- Leverage existing parsing code
  - UEFI: UEFITools, uefi-firmware-parser
  - ME: me-tools, unhuffme
- Separate server to receive collected images and compare with official versions
- Using manually rules to match / ignore false positives, per vendor/BIOS version

Findings

Unexpected Flash Descriptor content

- Descriptor has access control info for each regions
- When running in OS, CPU should only be able to read certain regions



- Found some flashes with full access to other regions

Unexpected Management Engine images

- While collecting and analysing BIOS:
  - Able to dump the ME part of the flash image
  - While the flash descriptor explicitly forbid such operation ??
- ME is usually not readable (Mac excepted)
- Similar machines (manufacturer, BIOS version) did not expose such behaviour

SPI FDOPSS

- Pin strap on the PCH
- If (de)asserted, override flash protection
- Some vendors allow overwrite of this bit using a jumper
- Some vendors connect this pin to the Embedded Controller

SPI FDOPSS

21.1.2 HSFS—Hardware Sequencing Flash Status Register (SPI Memory Mapped Configuration Registers)

Memory Address: SPIBAR + 04h
Default Value: 0000h
Attribute: RO, R/WC, R/W
Size: 16 bits

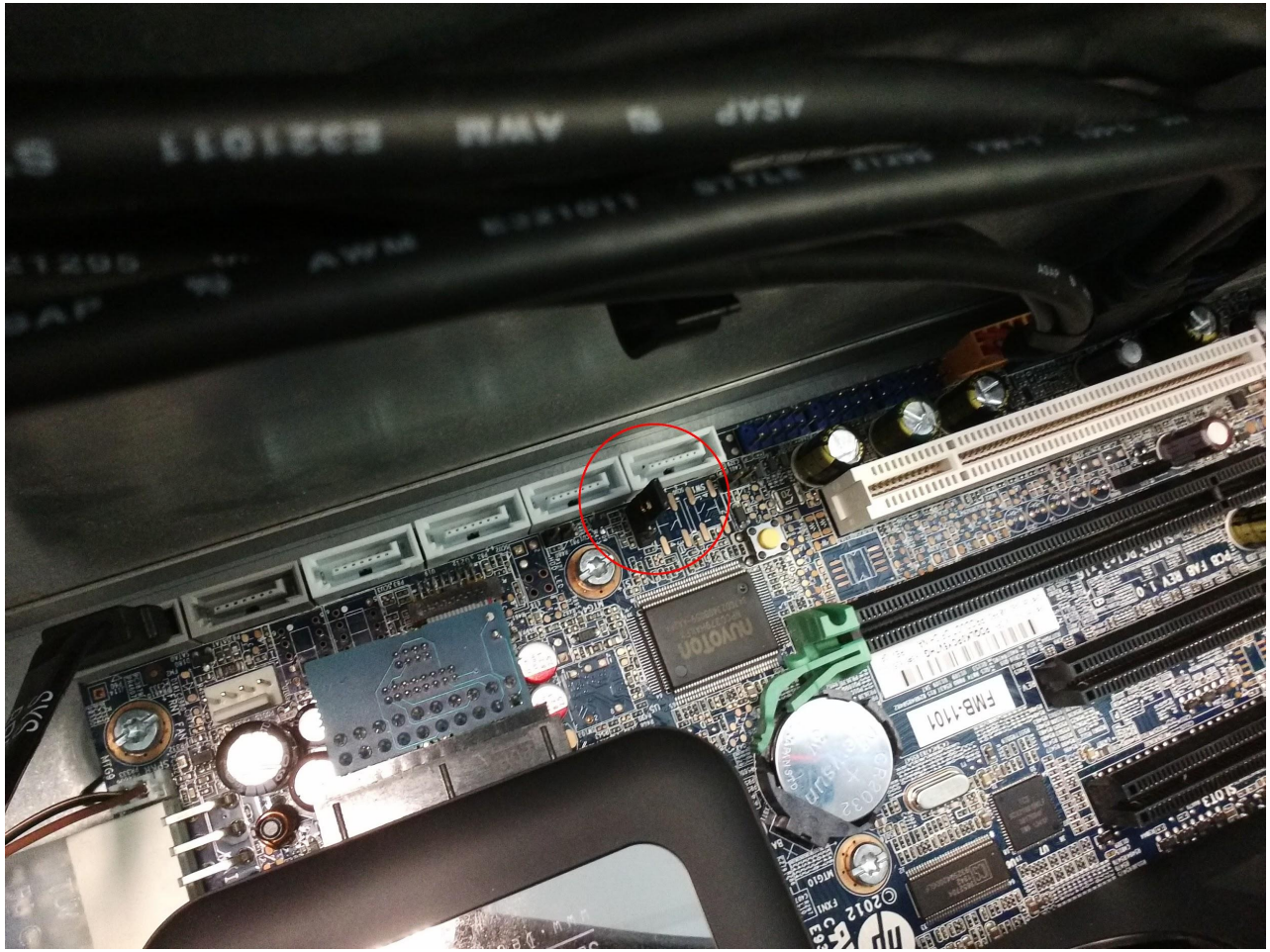
Bit	Description
15	**Flash Configuration Lock-Down (FLOCKDN)—**R/W/L. When set to 1, those Flash Program Registers that are locked down by this FLOCKDN bit cannot be written. Once set to 1, this bit can only be cleared by a hardware reset due to a global reset or host partition reset in an Intel ME enabled system.
14	**Flash Descriptor Valid (FDV)—**RO. This bit is set to a 1 if the Flash Controller read the correct Flash Descriptor Signature. If the Flash Descriptor Valid bit is not 1, software cannot use the Hardware Sequencing registers, but must use the software sequencing registers. Any attempt to use the Hardware Sequencing registers will result in the FCERR bit being set.
13	**Flash Descriptor Override Pin Strap Status (FDOPSS)**—RO. This bit indicates the condition of the Flash Descriptor Security Override / Intel ME Debug Mode pin strap. 0 = The Flash Descriptor Security Override / Intel ME Debug Mode strap is set using external pull-up on HDA_SDO 1 = No override
12:6	Reserved
5	**SPI Cycle In Progress (SCIP)—**RO. Hardware sets this bit when software sets the Flash Cycle Go

[1]

SPI FDOPSS

- Use Chipsec module of GRR to verify if that bit is set
- 4 lines of Python (hack) to read a specific hardware register
- Can also be implemented as a Chipsec module:
  - chipsec/modules/common/spi_fdopss.py

SPI FDOPSS



Conclusion

- Context shows firmware attacks are to be considered
- Bring some visibility to the x86 platform
- First tooling for enterprise-wide hunting

Huge thanks to the team: Ben, Darren, Jan, Parth and Mario.

Thank you!

References

1. Intel 9 Series Chipset Family Platform Controller Hub (PCH) datasheet
2. Diagram by Michael Cohen