The Bugs in Your Bootloaders:

Embedded Device Secure Boot Fails and How to Fix Them

Henrik Ferdinand Nölscher @s1ckcc
Product Security Engineering (PSE), Google Cloud
Blackhat Europe 2024

Hi! I'm Ferdi.
I'm part of Google Cloud - Product Security Engineering

We study how low-level attacks can compromise our hardware and firmware

We study how low-level attacks can compromise our hardware and firmware

Our approach:
Perform hardware and firmware penetration tests.
Find exploitable vulnerabilities.
Report vulnerabilities, influence vendors and standards, protect our infrastructure.

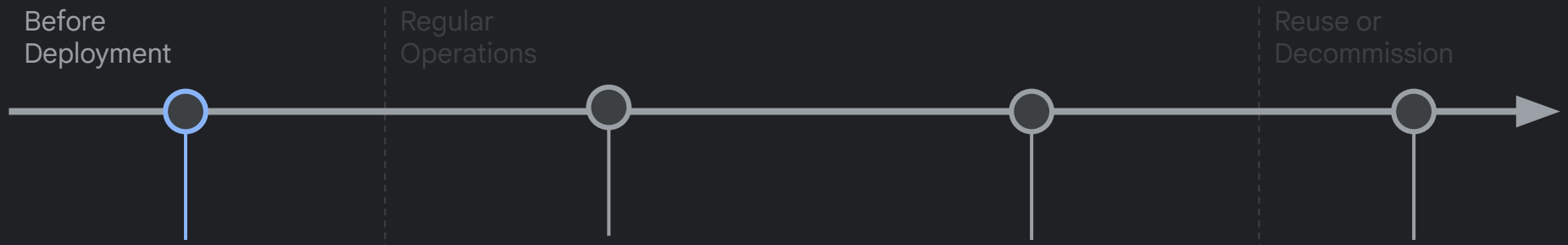
Start with the lowest layers:
Hardware, Firmware, Bootloaders

We study how low-level attacks can compromise our hardware and firmware

Our approach:
Perform hardware and firmware penetration tests.
Find exploitable vulnerabilities.
Report vulnerabilities, influence vendors and standards, protect our infrastructure.

Start with the lowest layers:
Hardware, Firmware, Bootloaders

Our approach:
Find exploitable vulnerabilities.



Supply Chain

Attackers have time-limited access to device before it arrives at its destination

Insider Threat

When deployed, attackers gain time-limited physical access on the device and perform attacks

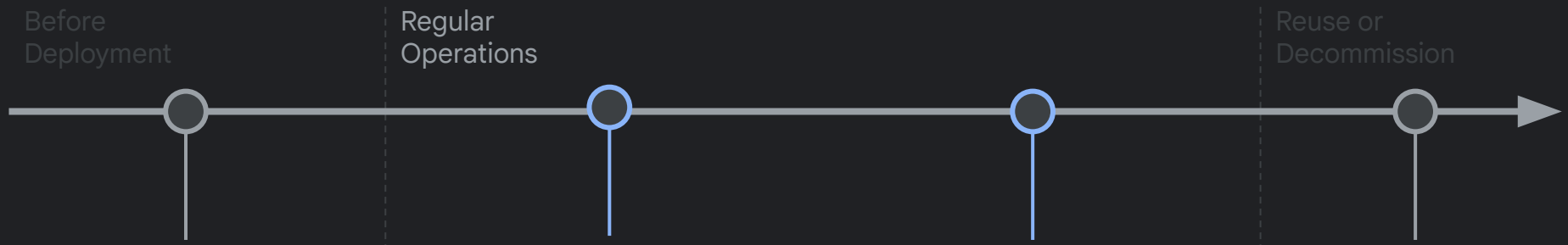
OS-to-Firmware Escape

Attackers gain privilege on the device remotely and infect the firmware

Device Reuse

Attackers try to compromise past or future users

Our approach:
Find exploitable vulnerabilities.



Supply Chain

Attackers have time-limited access to device before it arrives at its destination

Insider Threat

When deployed, attackers gain time-limited physical access on the device and perform attacks

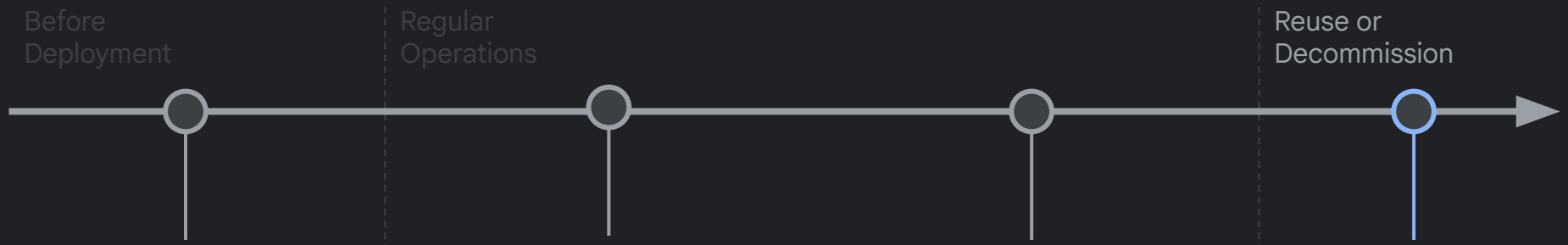
OS-to-Firmware Escape

Attackers gain privilege on the device remotely and infect the firmware

Device Reuse

Attackers try to compromise past or future users

Our approach:
Find exploitable vulnerabilities.



Supply Chain

Attackers have time-limited access to device before it arrives at its destination

Insider Threat

When deployed, attackers gain time-limited physical access on the device and perform attacks

OS-to-Firmware Escape

Attackers gain privilege on the device remotely and infect the firmware

Device Reuse

Attackers try to compromise past or future users

We study how low-level attacks can compromise our hardware and firmware

Our approach:
Perform hardware and firmware penetration tests.
Find exploitable vulnerabilities.
Report vulnerabilities, influence vendors and standards, protect our infrastructure.

Start with the lowest layers:
Hardware, Firmware, Bootloaders

We study how low-level attacks can compromise our hardware and firmware

Our approach:
Perform hardware and firmware penetration tests.
Find exploitable vulnerabilities.
Report vulnerabilities, influence vendors and standards, protect our infrastructure.

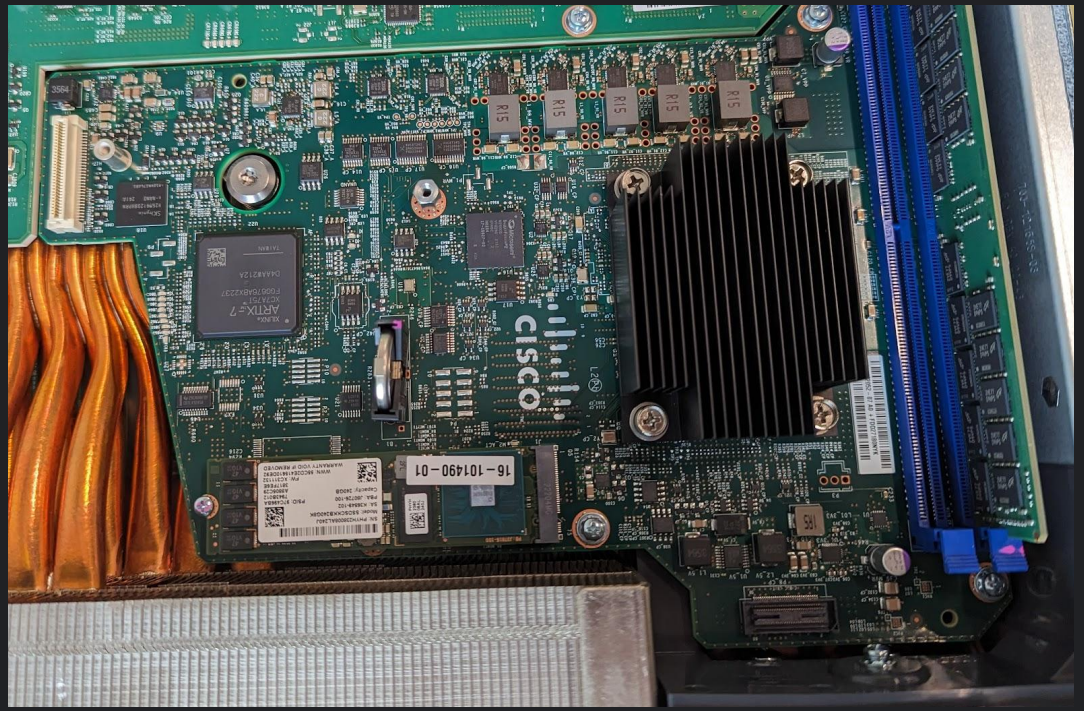
Start with the lowest layers:
Hardware, Firmware, Bootloaders

We study how low-level attacks can compromise our hardware and firmware.

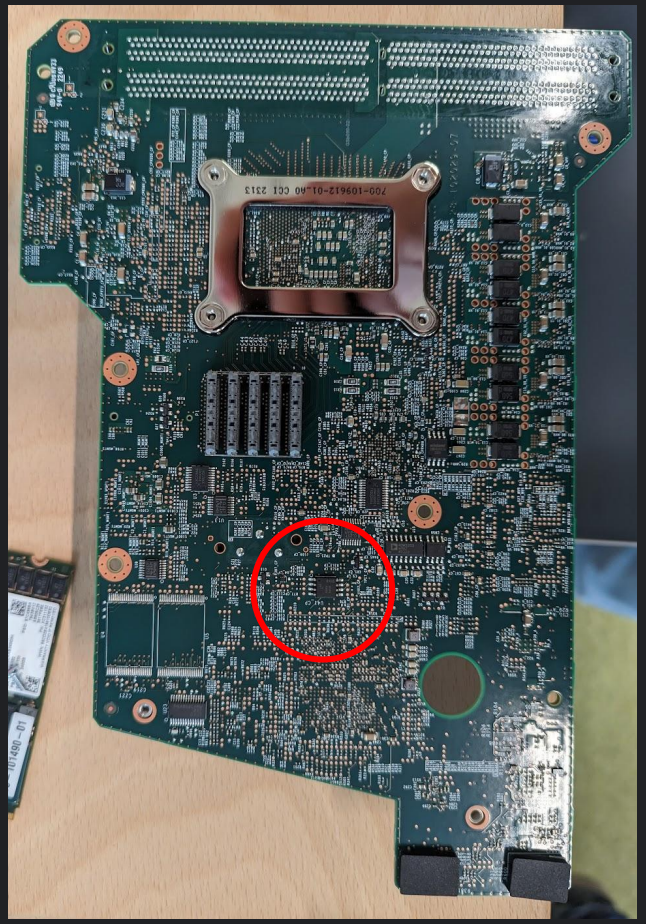
All 15+ reviewed device types that use open source bootloaders were affected by bootloader vulnerabilities.

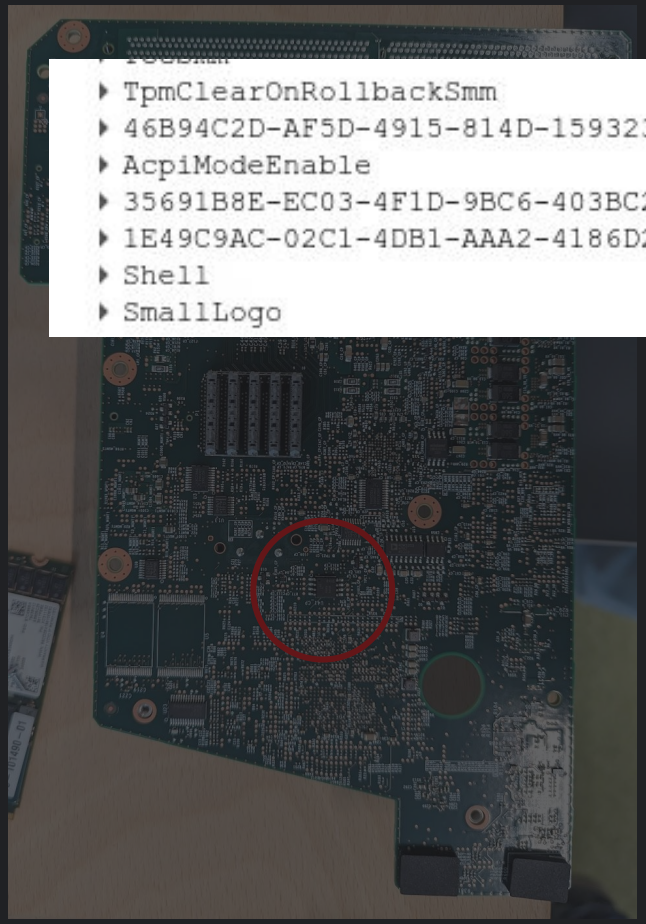
Real-World Bootloader Vulnerabilities: Cisco

The Device

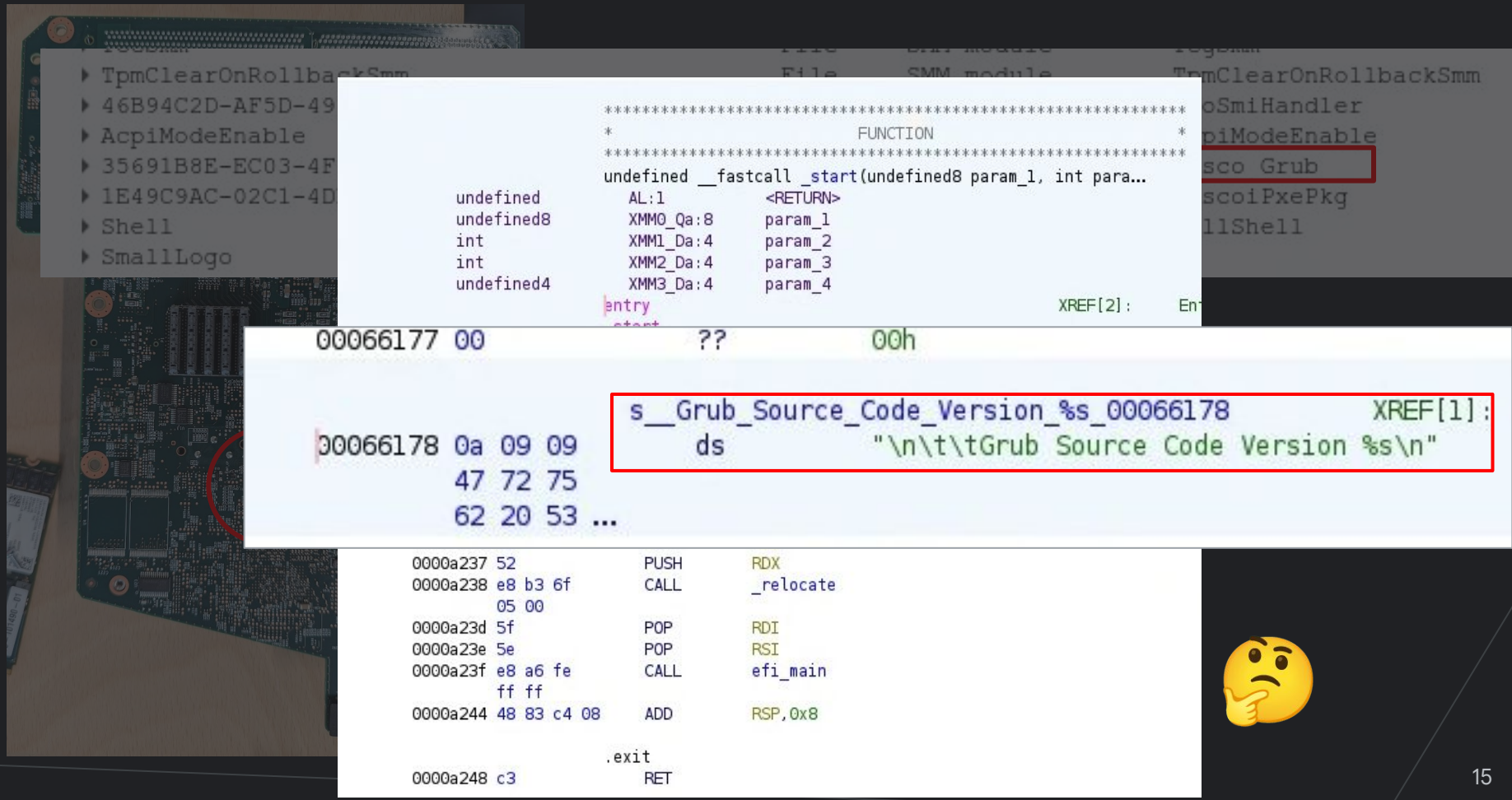


Affected devices: Cisco Nexus N9K Series

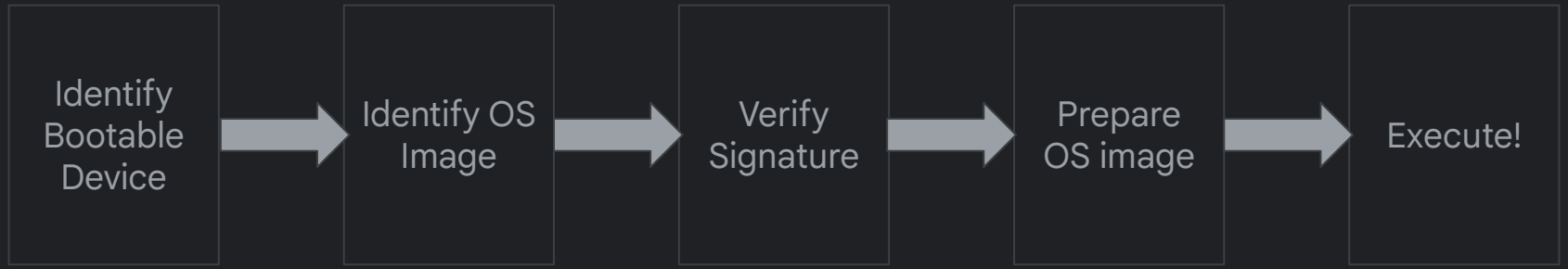




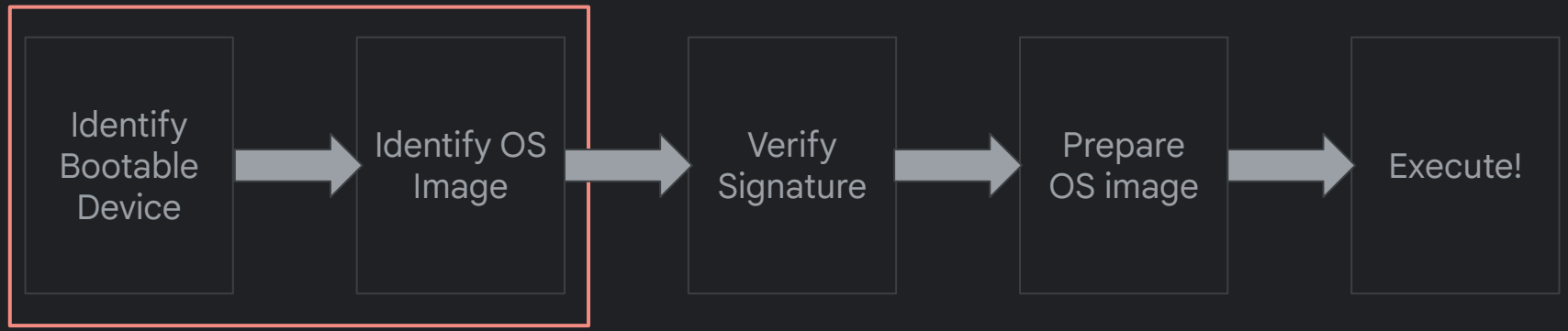
TpmClearOnRollbackSmm	File	SMM module	TpmClearOnRollbackSmm
46B94C2D-AF5D-4915-814D-159323AE8…	File	SMM module	TcoSmiHandler
AcpiModeEnable	File	SMM module	AcpiModeEnable
35691B8E-EC03-4F1D-9BC6-403BC2673…	File	Application	Cisco Grub
1E49C9AC-02C1-4DB1-AAA2-4186D2BE6…	File	Application	CiscoiPxePkg
Shell	File	Application	FullShell
SmallLogo	File	Freeform	


TpmClearOnRollbackSmm
46B94C2D-AF5D-49
AcpiModeEnable
35691B8E-EC03-4F
1E49C9AC-02C1-4D
Shell
SmallLogo
TpmClearOnRollbackSmm
oSmiHandler
piModeEnable
sco Grub
scoiPxePkg
llShell
FUNCTION
undefined __fastcall _start(undefined8 param_1, int para...
undefined AL:1 <RETURN>
undefined8 XMM0_Qa:8 param_1
int XMM1_Da:4 param_2
int XMM2_Da:4 param_3
undefined4 XMM3_Da:4 param_4
entry
XREF[2]:
00066177 00 ?? 00h
s__Grub_Source_Code_Version_%s_00066178 XREF[1]:
00066178 0a 09 09 ds "\n\t\tGrub Source Code Version %s\n"
47 72 75
62 20 53 ...
0000a237 52 PUSH RDX
0000a238 e8 b3 6f CALL _relocate
05 00
0000a23d 5f POP RDI
0000a23e 5e POP RSI
0000a23f e8 a6 fe CALL efi_main
ff ff
0000a244 48 83 c4 08 ADD RSP,0x8
.exit
0000a248 c3 RET

cisco-grub workflow

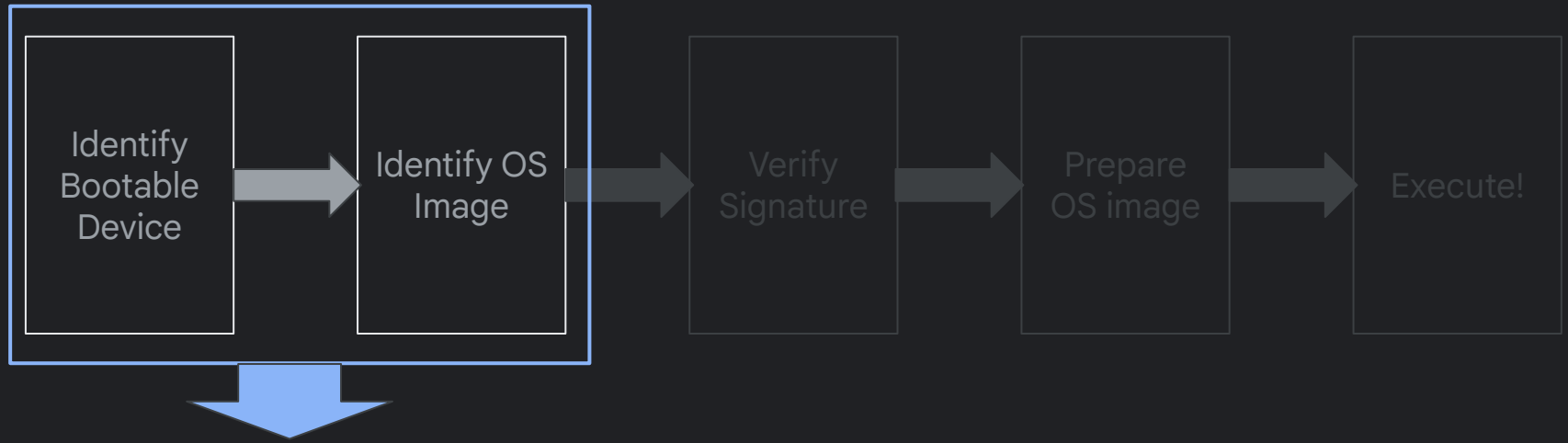


cisco-grub workflow



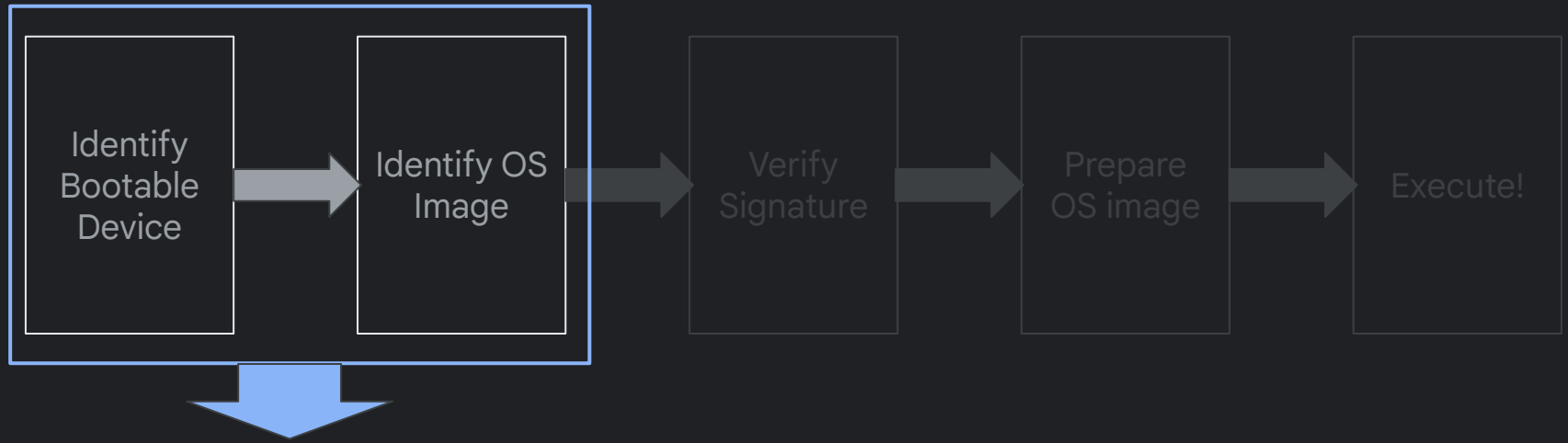
Requires file system interaction

cisco-grub workflow




```
for each storage device:
    for each file system $fs:
        if $fs->mount():
            try_boot($fs)
```

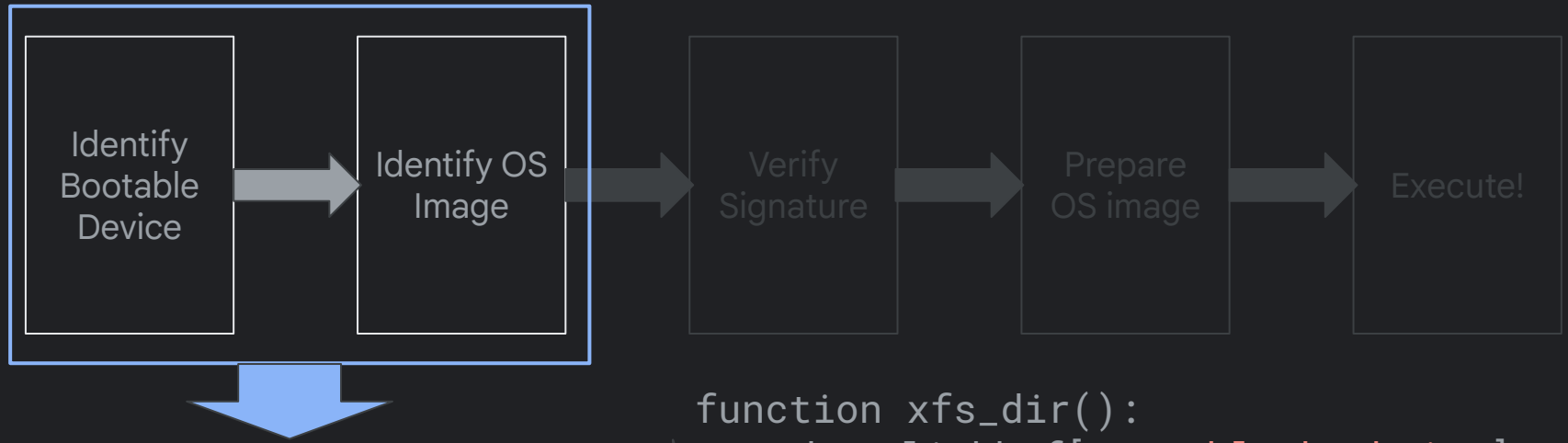
cisco-grub workflow




```
for each storage device:
    for each file system $fs:
        if $fs->mount():
            try_boot($fs)
```

Idea: find bugs in file system backends!

cisco-grub workflow




```
for each storage device:
    for each file system $fs:
        if $fs->mount():
            try_boot($fs)
```

```
function xfs_dir():
    char linkbuf[superblock->bsize];
    di_size = read(...);
    if di_size < (superblock->bsize-1):
        memcpy(linkbuf,...,di_size);
```

Classic buffer overflow!

Exploiting cisco-grub

USB DRIVE ATTACK

1. Craft a malicious XFS partition
2. Write it to a USB drive
3. Plug into device
4. Reboot

OS-TO-FIRMWARE ATTACK

1. Remotely: obtain admin privileges
2. Get a shell (this is a feature)
3. Write malicious XFS partition to disk
4. Reboot

Exploiting cisco-grub

USB DRIVE ATTACK

1. Craft a malicious XFS partition
2. Write it to a USB drive
3. Plug into device
4. Reboot

OS-TO-FIRMWARE ATTACK

1. Remotely: obtain admin privileges
2. Get a shell (this is a feature)
3. Write malicious XFS partition to disk
4. Reboot

5. Wait for cisco-grub to open a well-known file
6. Exploit buffer overflow, bypass signature checks

```
grub > root (hd1,0)
 Filesystem type is xfs, partition type 0x83

grub > cat /asdf/asdfsdf
-> pwnd by OTS-HS <-
!!!! X64 Exception Type - 06(#UD - Invalid Opcode)  CPU Apic ID - 00000000 !!!!
RIP  - 000000007FBFBB12, CS  - 0000000000000038, RFLAGS - 0000000000010202
RAX  - 000000000000BD00, RCX - 00000000BF0351E0, RDX - 0000000000000015
RBX  - 7FFFFFFFFFFFFFFF, RSP - 000000007FBFBB38, RBP - 000000007FBFBBA0
RSI  - 00000000BE78E150, RDI - 00000000BE78B043
R8   - 0000000000000000, R9  - 000000007FBFB87F, R10 - 0000000000000244
R11  - 0000000000000010, R12 - 00000000BEA37F3C, R13 - 0000000000000000
R14  - 0000000000000000, R15 - 00000000BDBBF018
DS   - 0000000000000030, ES  - 0000000000000030, FS  - 0000000000000030
GS   - 0000000000000030, SS  - 0000000000000030
CR0  - 0000000080010033, CR2 - 0000000000000000, CR3 - 00000000BF801000
CR4  - 0000000000000668, CR8 - 0000000000000000
DR0  - 0000000000000000, DR1 - 0000000000000000, DR2 - 0000000000000000
DR3  - 0000000000000000, DR6 - 00000000FFFF0FF0, DR7 - 0000000000000400
GDTR - 00000000BF5DC000 0000000000000047, LDTR - 0000000000000000
IDTR - 00000000BF059018 0000000000000FFF,   TR - 0000000000000000
FXSAVE_STATE - 000000007FBFB790
!!!! Find image based on IP(0x7FBFBB12) (No PDB)  (ImageBase=0000000000E26B54, EntryPoint=0000000000E2BBDF) !!!!
```

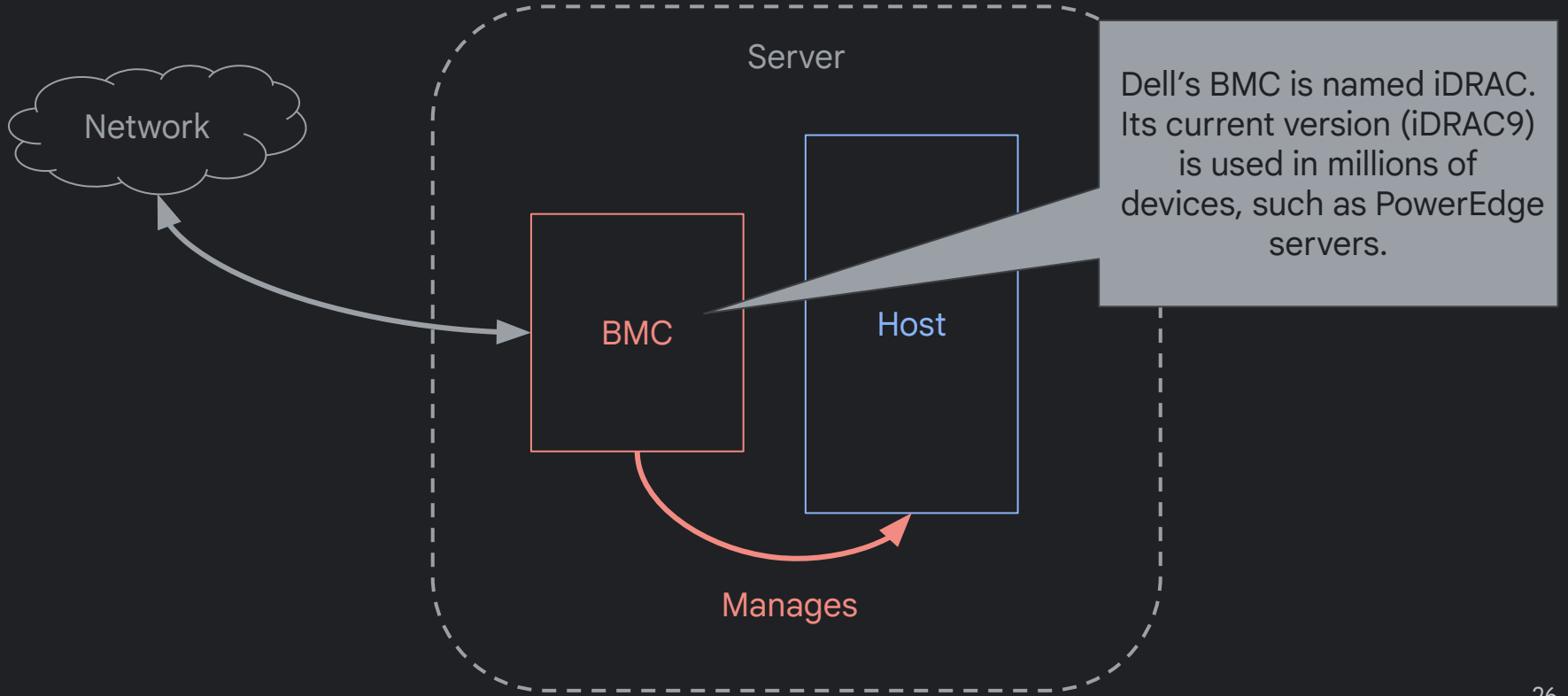
Exploit in action

Exploiting cisco-grub

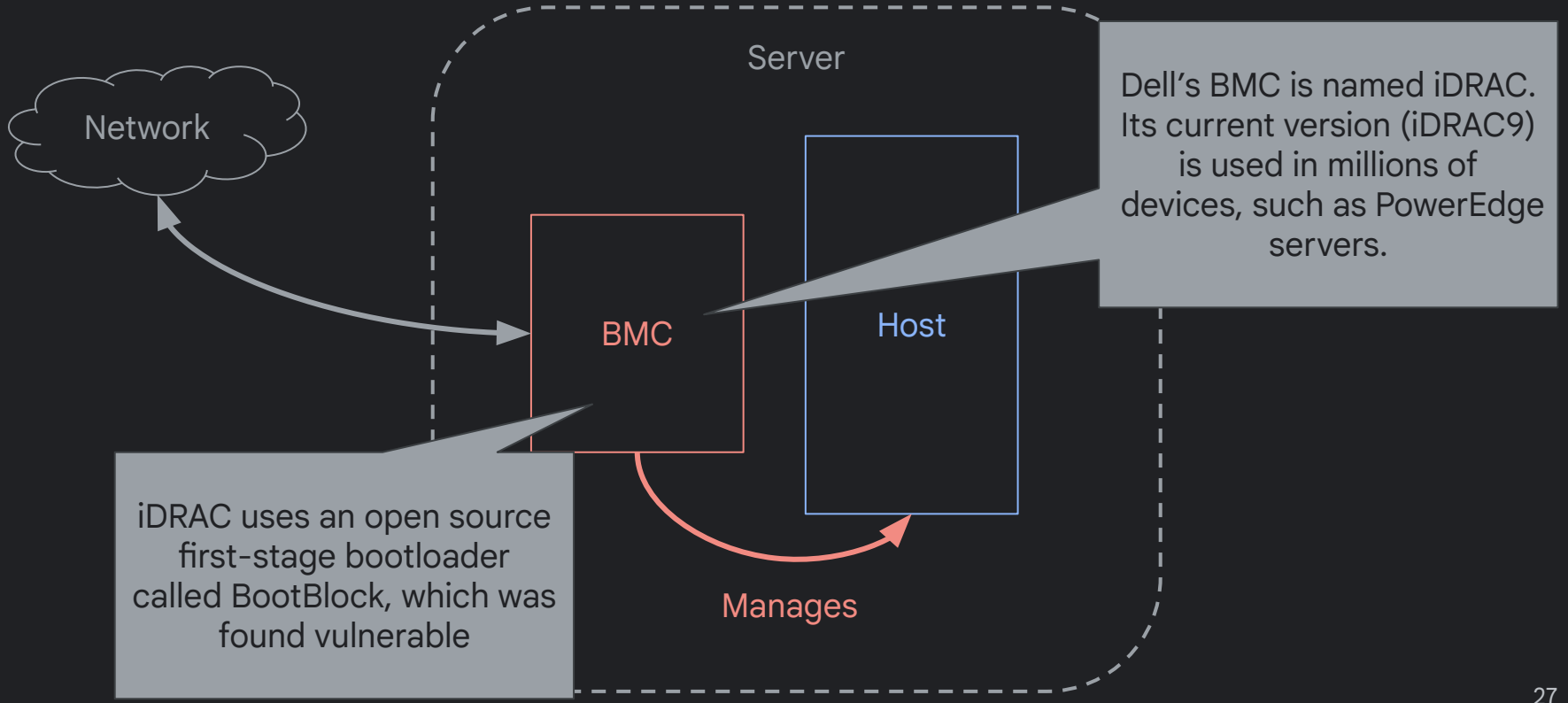
- Code execution in bootloader allows signature verification bypass. If exploited correctly: undetectable, unrecoverable compromise
- XFS vulnerability was fixed in NX-OS 10.4.2
  CVE 2023-4949
- Vulnerable XFS code was reused in Xen tools and Coreboot Filo! CVE-2023-34325

Real World Bootloader Vulnerabilities: Dell RootBlock

What is Dell iDRAC?



What is Dell iDRAC?

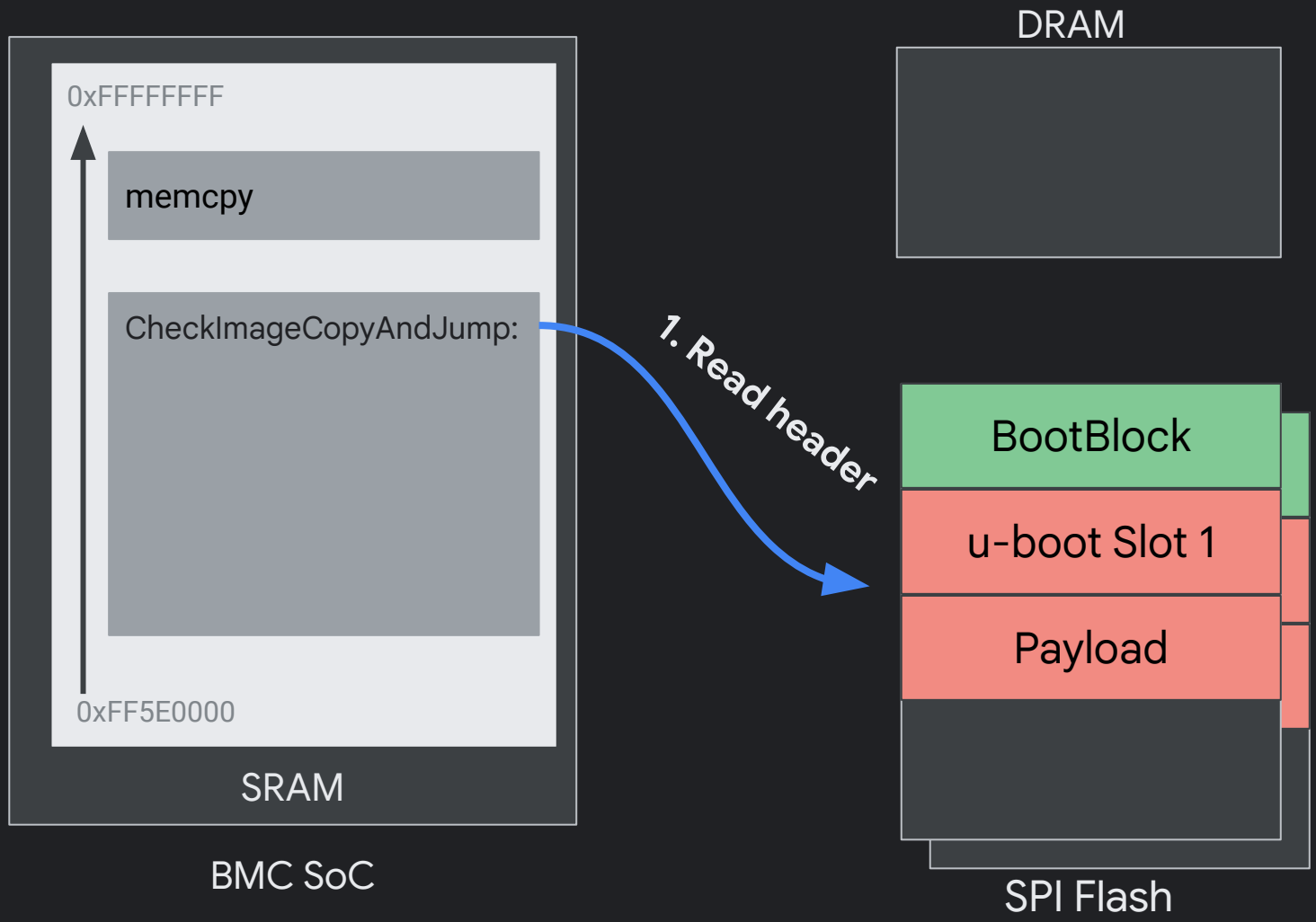


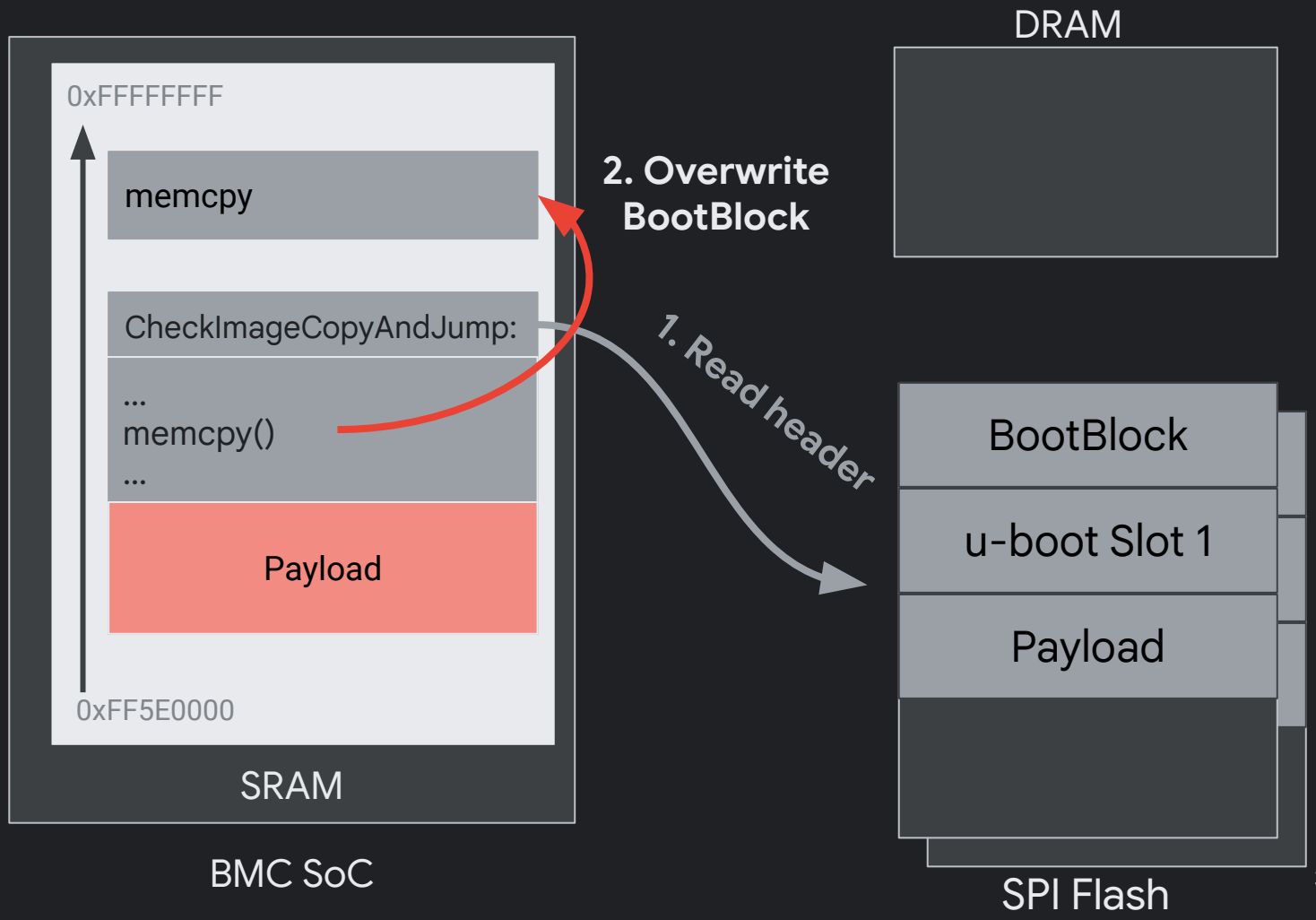
iDRAC First Stage Bootloader: BootBlock

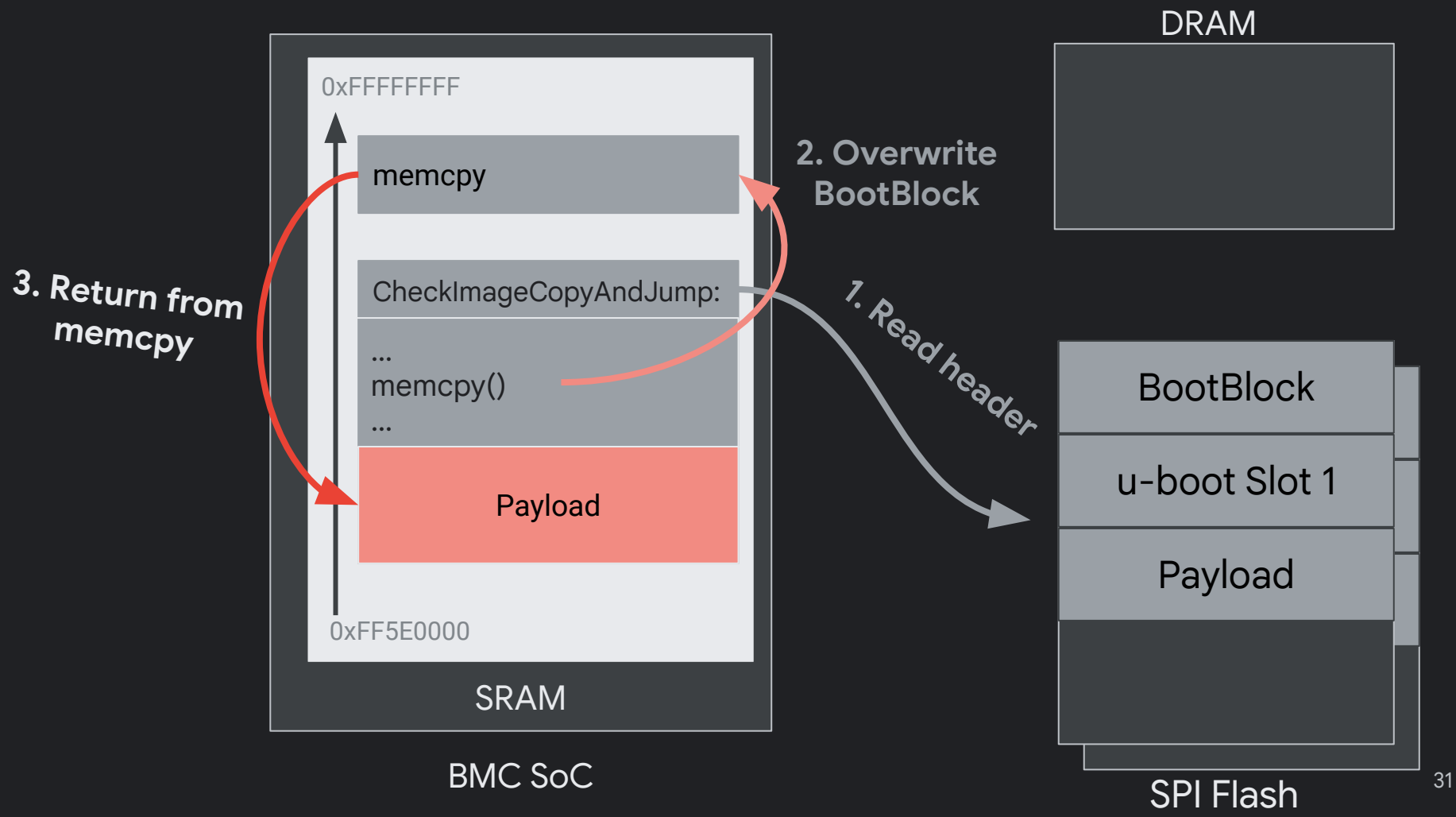
```
function CheckImageCopyAndJump {
    header_t image_header;
    memcpy(&image_header, s                    (image_header);

    if(image_header->size >
        bail();

    memcpy(image_header->dst, spi_flash, image_header->size);

    if(! check_signature(image_header->dst, &image_header))
        bail(); // signature check fail

    execute_uboot_image();
}
```

Attacker controls destination address

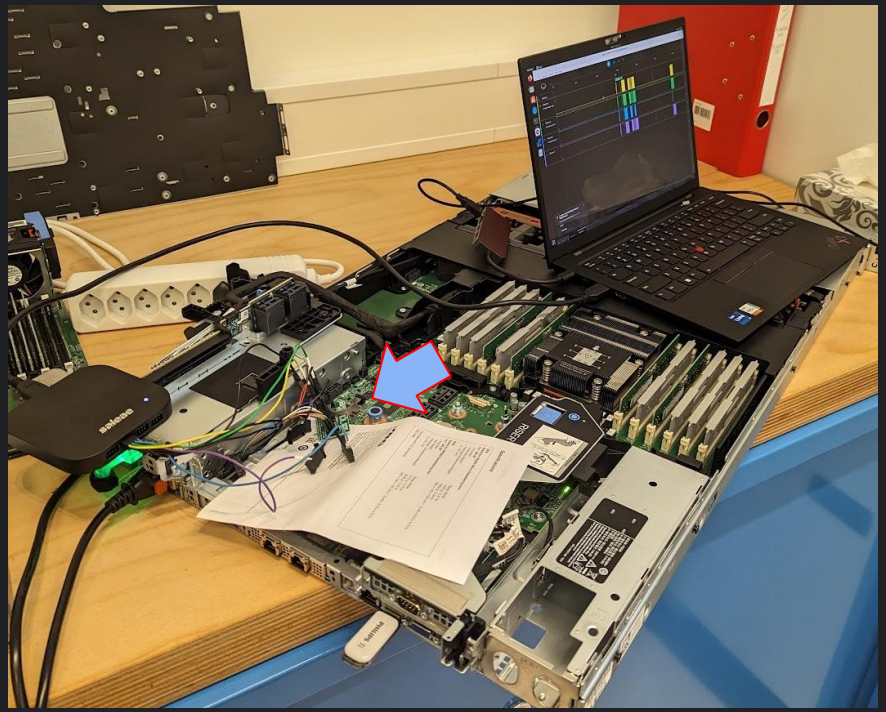
Signature is checked AFTER copying the u-boot image

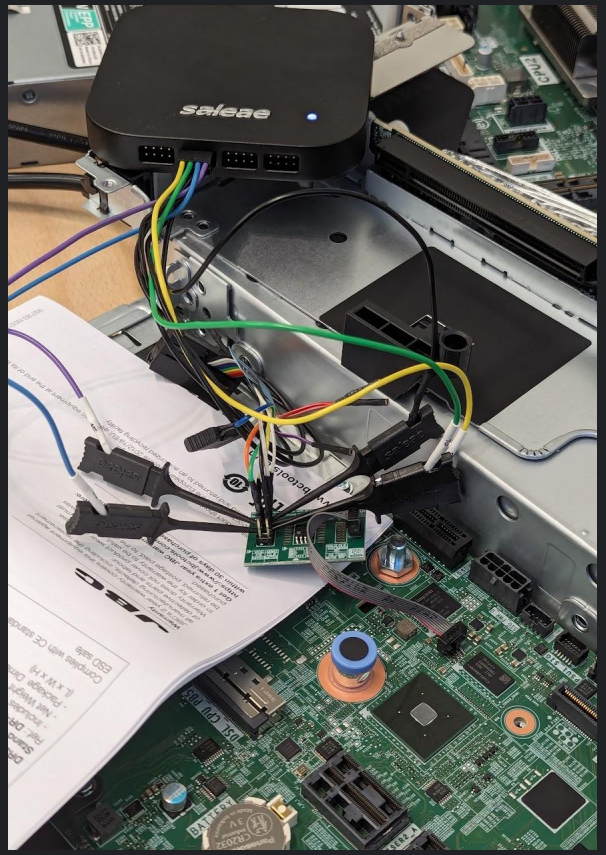

DRAM
0xFFFFFFFF
memcpy
CheckImageCopyAndJump:
0xFF5E0000
SRAM
BMC SoC
1. Read header
BootBlock
u-boot Slot 1
Payload
SPI Flash


DRAM
0xFFFFFFFF
memcpy
CheckImageCopyAndJump:
...
memcpy()
...
Payload
0xFF5E0000
SRAM
BMC SoC
2. Overwrite BootBlock
1. Read header
BootBlock
u-boot Slot 1
Payload
SPI Flash


DRAM
0xFFFFFFFF
memcpy
2. Overwrite BootBlock
CheckImageCopyAndJump:
...
memcpy()
...
Payload
1. Read header
3. Return from memcpy
0xFF5E0000
SRAM
BMC SoC
BootBlock
u-boot Slot 1
Payload
SPI Flash

Exploiting RootBlock

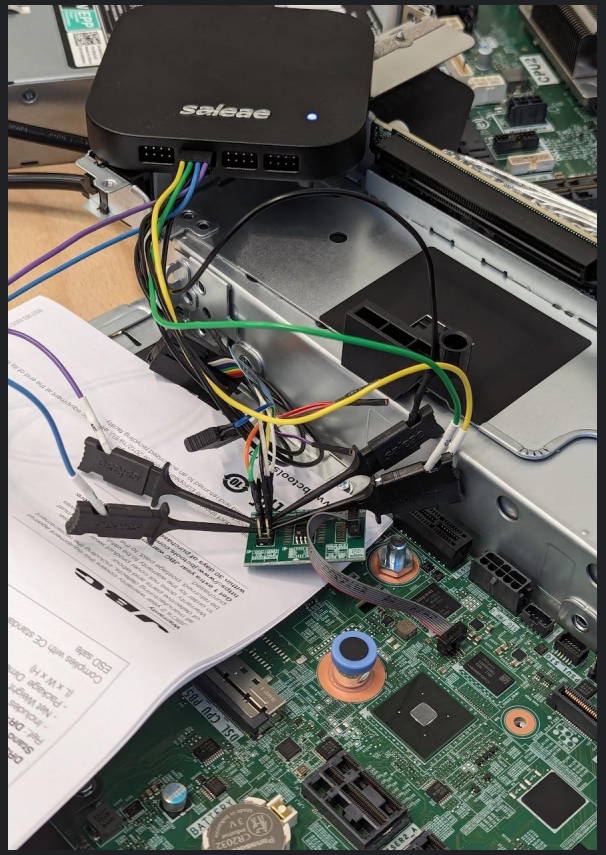




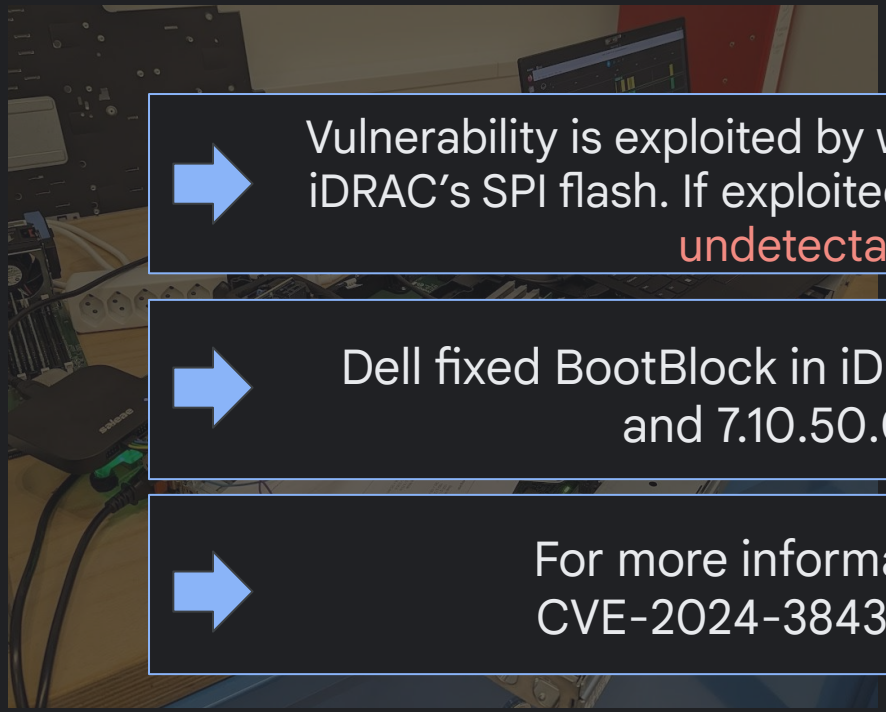
Exploiting RootBlock

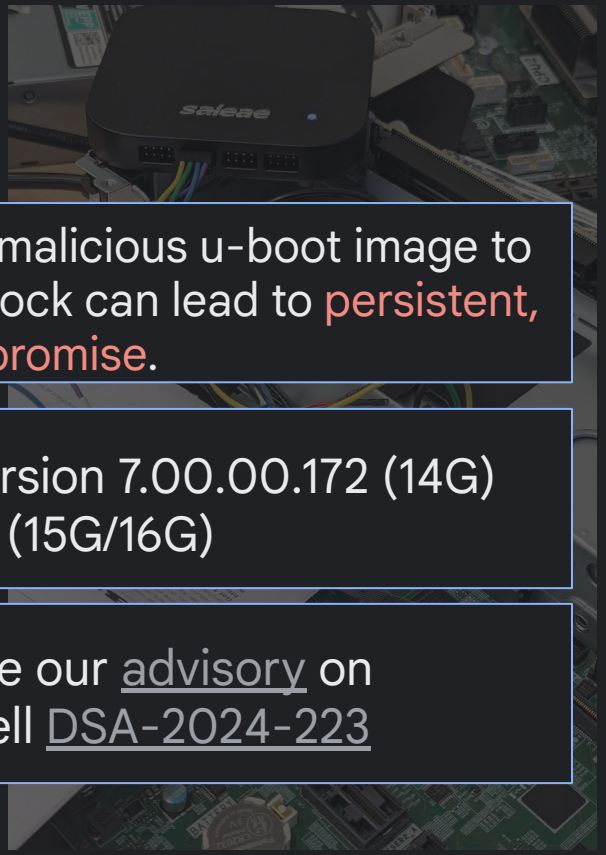
Data

```
U-Boot 2021.04 (Feb 23 2024 - 12:30:26 +0000) -> pwnd by OTS-HS <-

CPU: NPCM750 A1 @ Model: Nuvoton npcm750 Development Board (Device Tree)
DRAM:  464 MiB
12_pl310_init
RNG: NPCM RNG module bind OK
OTP: NPCM OTP module bind OK
AES: NPCM AES module bind OK
SHA: NPCM SHA module bind OK
MMC:   sdhci0@f0842000: 0
Loading Environment from SPIFlash... SF: Detected w25q32jv with page size 2
56 Bytes, erase size 4 KiB, total 4 MiB
*** Warning - bad CRC, using default environment

In:    serial@1000
Out:   serial@1000
Err:   serial@1000
Net:   No ethernet found.
Security is enabled
Hit any key to stop autoboot:  0
80006960: 1c 10 90 e5 08 10 81 e3 1c 10 80 e5 1c 10 90 e5    ................
..
80006970: 04 10 81 e3 1c 10                                  ......
No MDIO bus found
NULL device name!
No such device: <NULL>
```



Exploiting RootBlock





- Vulnerability is exploited by writing a malicious u-boot image to iDRAC's SPI flash. If exploited, RootBlock can lead to persistent, undetectable compromise.
- Dell fixed BootBlock in iDRAC9 Version 7.00.00.172 (14G) and 7.10.50.00_A00 (15G/16G)
- For more information, see our advisory on CVE-2024-38433 and Dell DSA-2024-223

How to fix it: A Recipe


3 Hardened Bootloaders
Modern
Hardware Security
2
1 Better Threat Models

3 Hardened Bootloaders

Modern
Hardware Security
2

1 Better Threat Models

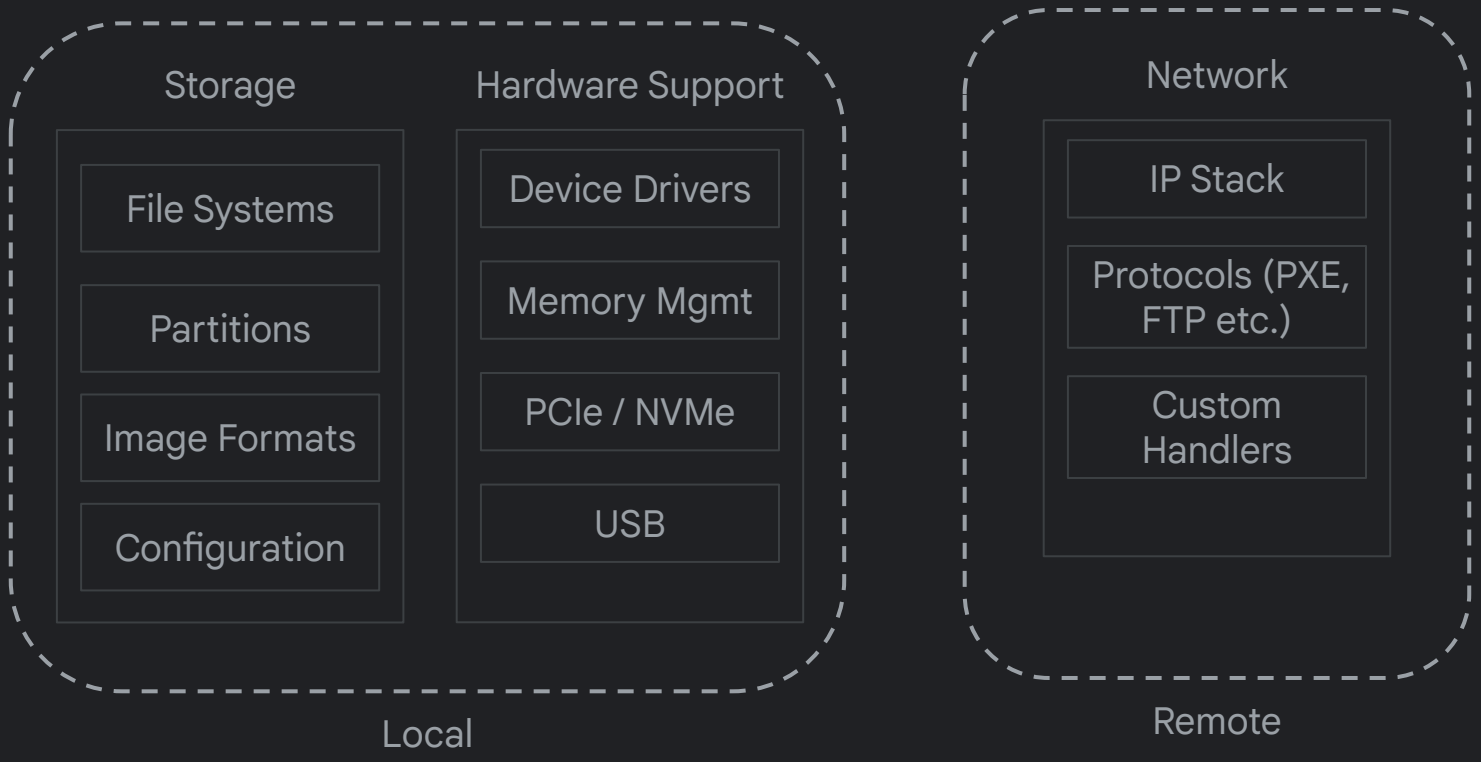
Ingredient #1: Better Threat Models

Do not assume that physical security is guaranteed.
Consider Insider & Supply Chain risk

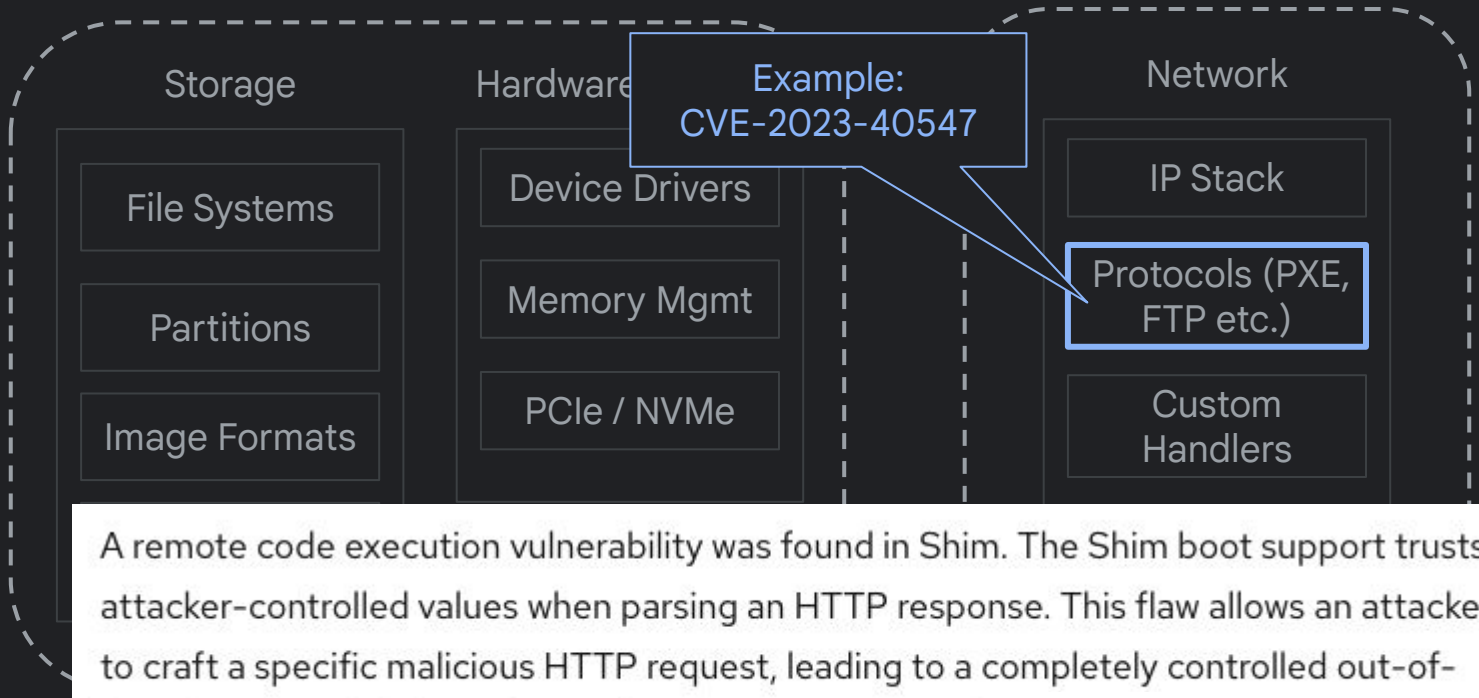
Do not assume that users/workloads can be trusted.

Consider your early-boot attack surface!

Ingredient #1: Bootloader Attack Surface



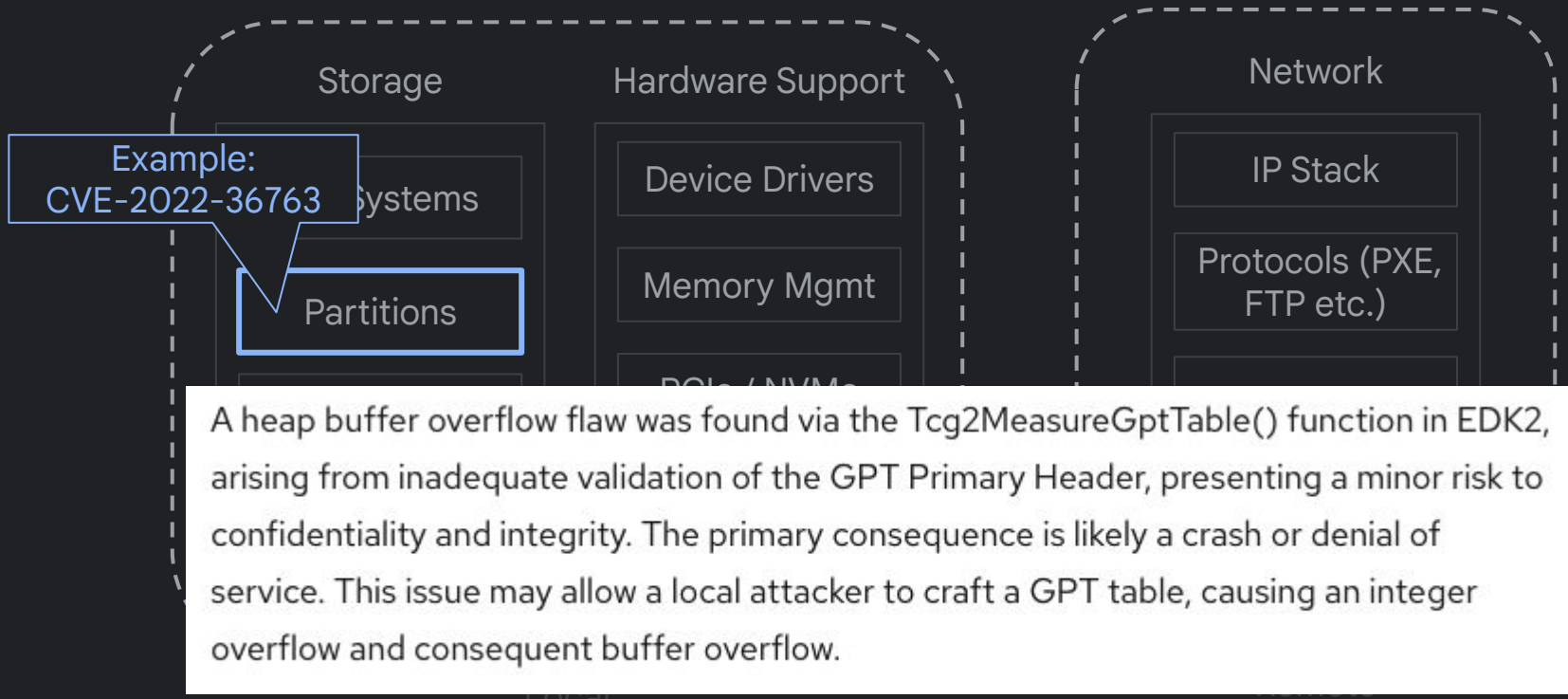
Ingredient #1: Bootloader Attack Surface



A remote code execution vulnerability was found in Shim. The Shim boot support trusts attacker-controlled values when parsing an HTTP response. This flaw allows an attacker to craft a specific malicious HTTP request, leading to a completely controlled out-of-bounds write primitive and complete system compromise.

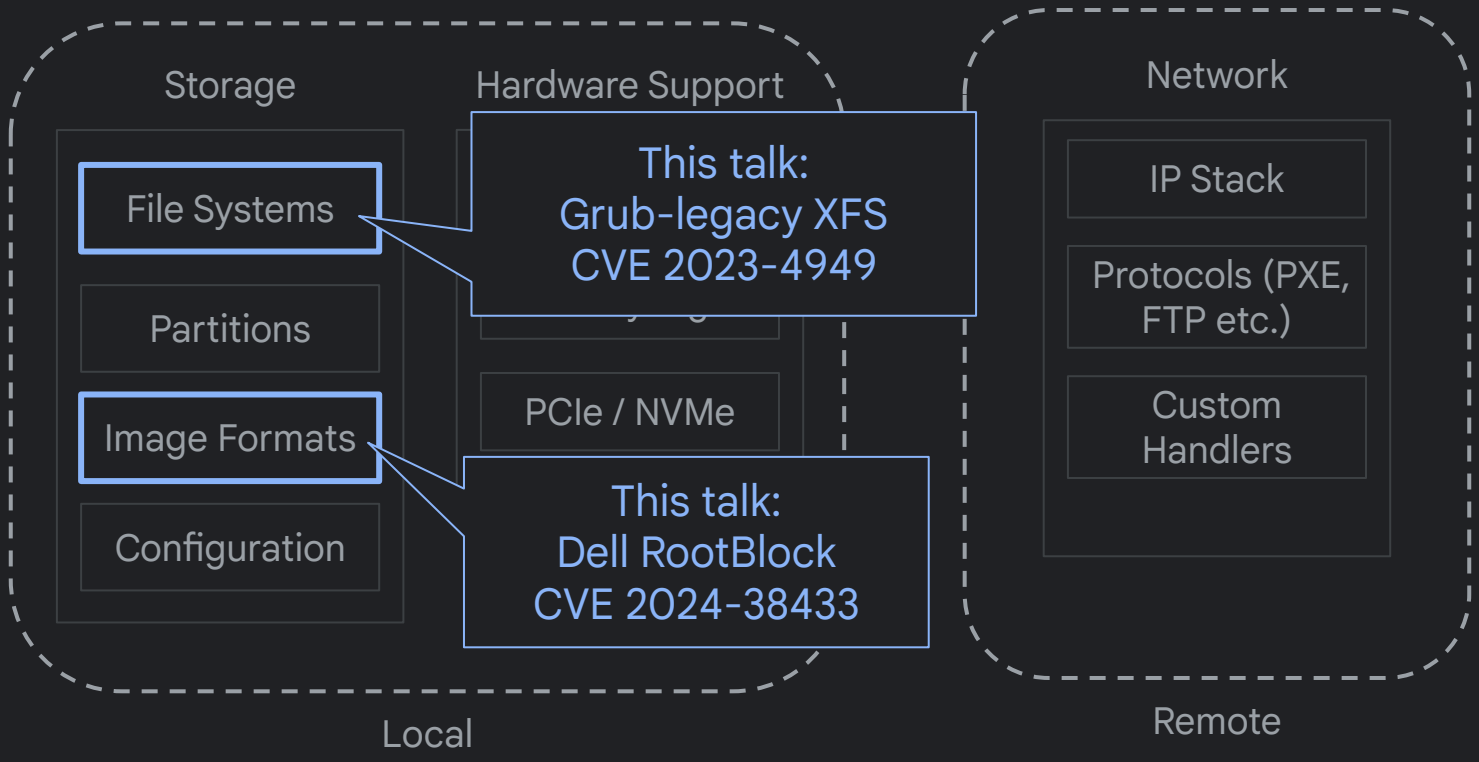
Credit: Bill Demirkapi

Ingredient #1: Bootloader Attack Surface



Credit: Marc Beatove

Ingredient #1: Bootloader Attack Surface



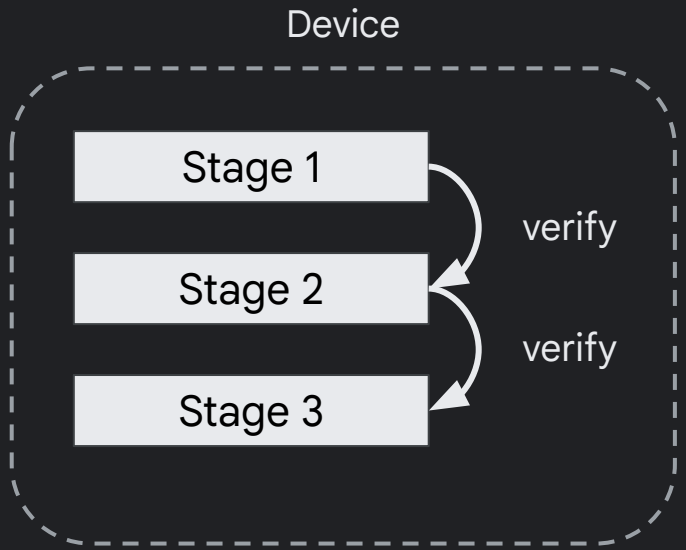

Modern
Hardware Security
2
3
Hardened Bootloaders
1
Better Threat Models

Ingredient #2: Modern Hardware Security

There will always be bugs. A single vulnerability must not lead to full compromise. We need downgrade protection:

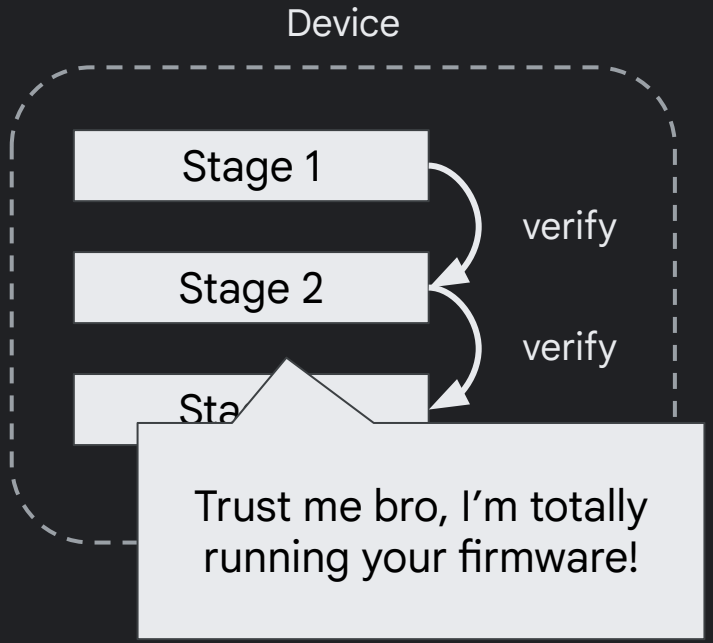
Ingredient #2: Modern Hardware Security

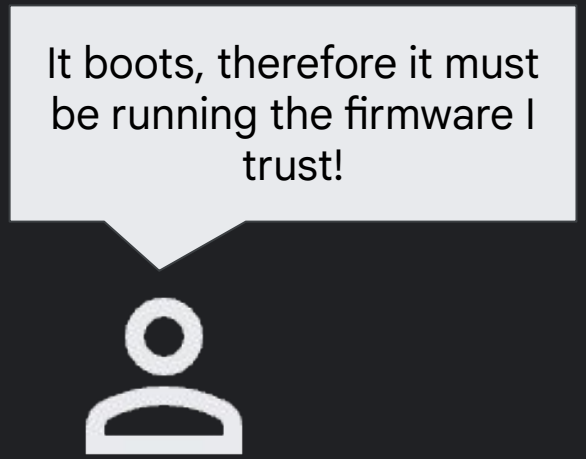
There will always be bugs. A single vulnerability must not lead to full compromise. We need remote attestation:



Ingredient #2: Modern Hardware Security

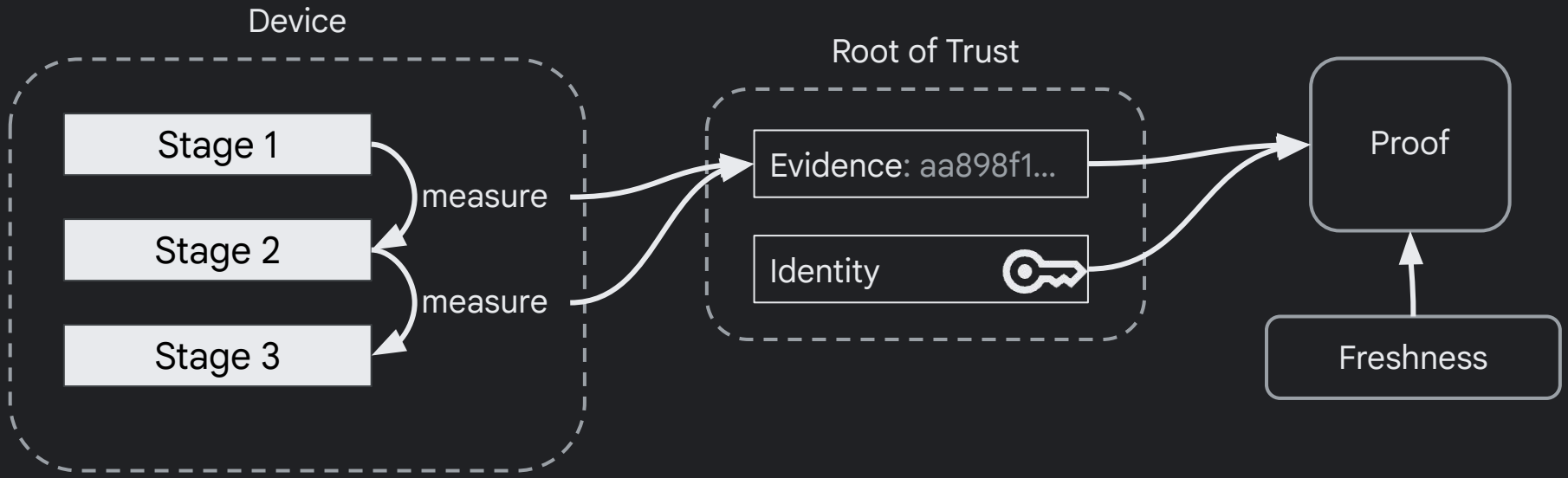
There will always be bugs. A single vulnerability must not lead to full compromise. We need remote attestation:





Ingredient #2: Modern Hardware Security

There will always be bugs. A single vulnerability must not lead to full compromise.
We need remote attestation:




3 Hardened Bootloaders
Modern
Hardware Security
2
1 Better Threat Models

Ingredient #3: Hardened Bootloaders

There will always be bugs. A single vulnerability must not lead to full compromise. We need exploit mitigations:

Classic Exploit Mitigations	Mitigations such as Stack Canaries and Control Flow Integrity could improve defense in depth
Read-Only Sections	RootBlock could have been prevented if SRAM was made read-only before loading untrusted images

Ingredient #3: OSS Bootloader Security State

Bootloader Project	Memory Safe?	Exploit Mitigations?	Fuzzed?	Security Critical?
BootBlock	No	No	No	Yes
grub-legacy	No	No	No	Sometimes
u-boot	No	No	No	Yes
grub2	No	No	Kind of	Yes
shim	No	No	Kind of	Yes
linuxboot	Partially	No	No	Yes
EDK2	No	Optional	Kind of	Yes
Arm Trusted Firmware (ATF)	No	Yes	No	Yes
Caliptra Firmware	Yes	Yes	Yes	Yes

Ingredient #3: OSS Bootloader Security State

Danger territory

Bootloader Project	Memory Safe?	Exploit Mitigations?	Fuzzed?	Security Critical?
BootBlock	No	No	No	Yes
grub-legacy	No	No	No	Sometimes
u-boot	No	No	No	Yes
grub2	No	No	Kind of	Yes
shim	No	No	Kind of	Yes
linuxboot	Partially	No	No	Yes
EDK2	No	Optional	Kind of	Yes
Arm Trusted Firmware (ATF)	No	Yes	No	Yes
Caliptra Firmware	Yes	Yes	Yes	Yes

Ingredient #3: Discovering more vulnerabilities

Code Scanning	Code scanning can catch basic vulnerabilities
LLMs	LLMs were able to find the presented vulnerabilities. However they did not uncover more vulnerabilities, yet
Fuzzing	Integrate fuzzers upstream. Extend fuzzers so that they cover our attack surface. Find and report bugs.

oss-fuzz for Bootloaders

Receive up to 15 000 USD reward for integrating critical open source projects

Are popular bootloaders critical? Yes!

u-boot integration has already been started

Are you a bootloader developer? Please reach out!

Thanks!

Hardened Bootloaders

Hardening bootloaders can increase security for a wide range of devices. Approaches like code review, exploit mitigations and fuzzing work well.

Modern Hardware Security

Hardware should offer more protection against a single compromised component. In many cases, secure boot is not enough.

Better Threat Models

A threat model that considers threats to hardware and firmware is required to improve security in the long term. This can be applied across different bootloaders, vendors or devices.

Advisories

grub-legacy XFS
cisco-grub script execution
Dell RootBlock

Contact

Henrik Ferdinand Nölscher
@s1ckcc
bootloader-bugs@google.com

Google Cloud - Product Security Engineering