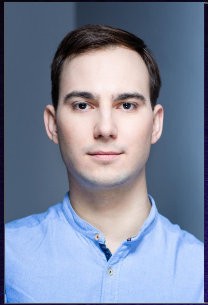
binarly

Broken Trust: Firmware Bypass Chains, BMC Persistence, and EDR Evasion

Alex Matrosov, Fabio Pagani

@DistrictCon 1

Binarly REsearch Team

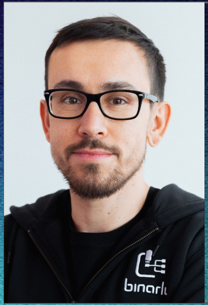


Anton
Ivanov

@ant_av7

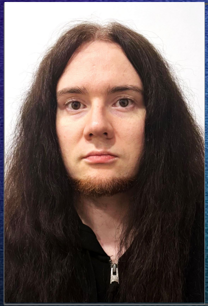
Alex
Matrosov

@matrosov



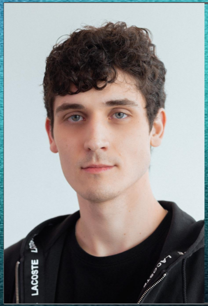
Fabio
Pagani

@pagabuc



Sam L.
Thomas

@xorpse



Yegor
Vasilenko

@yeggorv

Introduction

The Invisible Foundation: Firmware is Everywhere

PERSONAL COMPUTING

Laptops, desktops

Enterprise servers

CORE INFRASTRUCTURE

Enterprise servers

Network appliances

CRITICAL SYSTEMS

ATMs

Voting machines

The Scale of Code in Modern Firmware

Project	# Lines of ASM*	# Lines of C*
SQLite	438k	183k
Linux kernel 6.18 (defconfig)	19.9M	6.5M
BMC (uboot + kernel + libs)	25.9M	8.9M
Laptop UEFI Firmware	30.1M	7.1M

* Counted using scc on the output of IDA Pro's 'Create ASM File' and 'Create C File'.

Firmware: A Reality Check

- Billions of heterogeneous devices across the entire computing stack
- Large codebase: millions of lines of C code
- Testing is non-trivial, highly hardware-dependent
- What can go wrong?

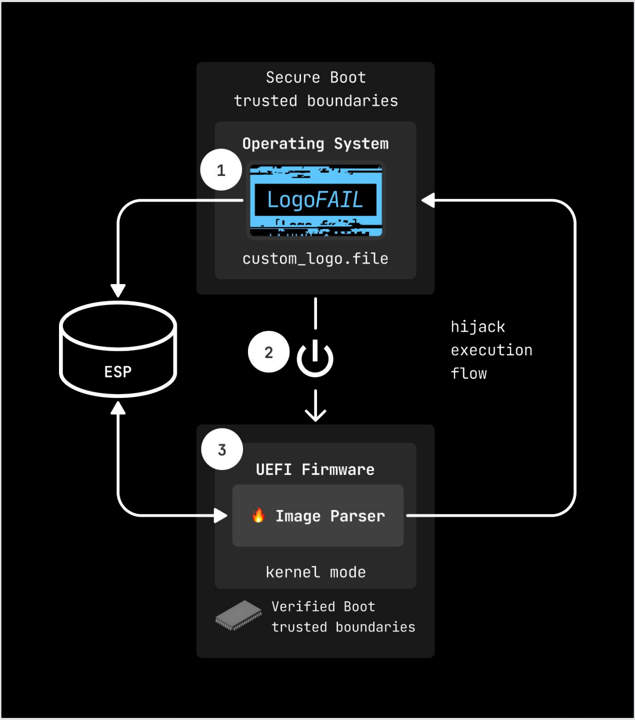
LogoFAIL

SUMMARY

- Majority of UEFI firmware contains image parsers
- Vendors allow customization of the logo displayed during boot
- Image parsers written in C, found crashes after seconds of fuzzing

DEVELOPING A POC

1. From the OS, store a malformed image on the ESP
2. Reboot the system
3. UEFI firmware parses the malformed image
4. Integer overflow to Heap overflow to DXE arbitrary code execution



https://www.binarly.io/blog/finding-logofail-the-dangers-of-image-parsing-during-system-boot
https://www.binarly.io/blog/inside-the-logofail-poc-from-integer-overflow-to-arbitrary-code-execution

Unknown Vulnerabilities Threatening the UEFI Ecosystem

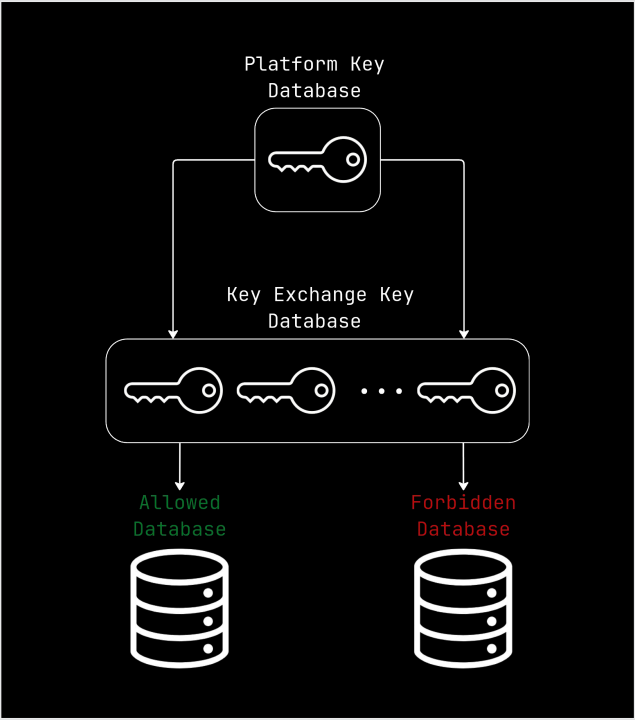
Vulnerability	CVSS Score	CWEs
DoubleGetVariable	8.2 (High)	CWE-787: Out-of-bounds Write
GetSetVariable	6.0 (Medium)	CWE-125: Out-of-bounds Read
PointerViaVariable (Memory Write)	8.2 (High)	CWE-787: Out-of-bounds Write CWE-822: Untrusted Pointer Dereference
PointerViaVariable (Function Call)	8.2 (High)	CWE-822: Untrusted Pointer Dereference CWE-829: Inclusion of Untrusted Functionality
SmmCommBuffer (Memory Write)	8.2 (High)	CWE-787: Out-of-bounds Write CWE-822: Untrusted Pointer Dereference
SmmCommBuffer (Callout)	8.2 (High)	CWE-822: Untrusted Pointer Dereference CWE-829: Inclusion of Untrusted Functionality
…		

Exploiting UEFI SMM Vulnerabilities For Persistent Implants, Nika Korchok Wakulich, OOTB2025BKK, https://www.youtube.com/watch?v=OzM2aGsiD1s

Secure Boot Vulnerabilities Impacting the Chain of Trust

Recent vulnerabilities impacting Secure Boot:

- PKfail
- Hydroph0bia (CVE-2025-4275)
- Broken dbx[1]
- Vulnerable signed module:
    - CVE-2025-3052 (Binarly)
    - CVE-2024-7344 (ESET)



[1] https://www.binarly.io/blog/from-trust-to-trouble-the-supply-chain-implications-of-a-broken-dbx

PKfail

Oh, hi! I am a private key, that’s been available on GitHub for 6 months! 🤷‍♂️

```
Version: 3 (0x2)
Serial Number:
    55:fb:ef:87:81:23:00:84:47:17:0b:b3:cd:87:3a:f4
Signature Algorithm: sha256WithRSAEncryption
Issuer: CN=DO NOT TRUST - AMI Test PK
Validity
    Not Before: Nov  8 23:32:53 2017 GMT
    Not After : Nov  8 23:32:52 2021 GMT
Subject: CN=DO NOT TRUST - AMI Test PK
Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
        Public-Key: (2048 bit)
        Modulus:
            00:e7:36:7b:20:92:ba:7f:aa:a3:f6:0e:49:08:87:
            f5:1c:11:33:ba:5d:f8:9b:5c:ed:c7:90:e4:f3:41:
...
```

```
$ openssl pkcs12 -in FW_priKey.pfx -nodes
Enter Import Password:
```

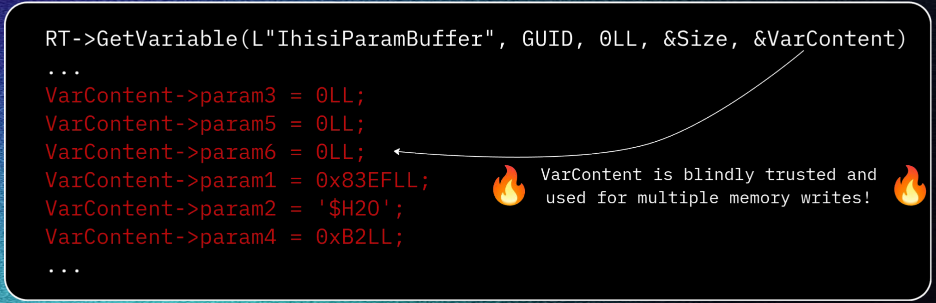
```
$ cat AmiTestKey.sdl | grep password -C3
TOKEN
        Name  = "FW_PFX_Password"
        Value  = "abcd"
        Help  = "Specifies the password to use when opening a PFX -
Private Key container file."
        TokenType = Expression
        TargetMAK = Yes
End
```

```
$ openssl x509 -noout -text -in FW_pubKey.cer  | rg "Issuer:|Subject:"
        Issuer: CN=DO NOT TRUST - AMI Test PK
        Subject: CN=DO NOT TRUST - AMI Test PK
```

https://www.binarly.io/pkfail

CVE-2025-3052

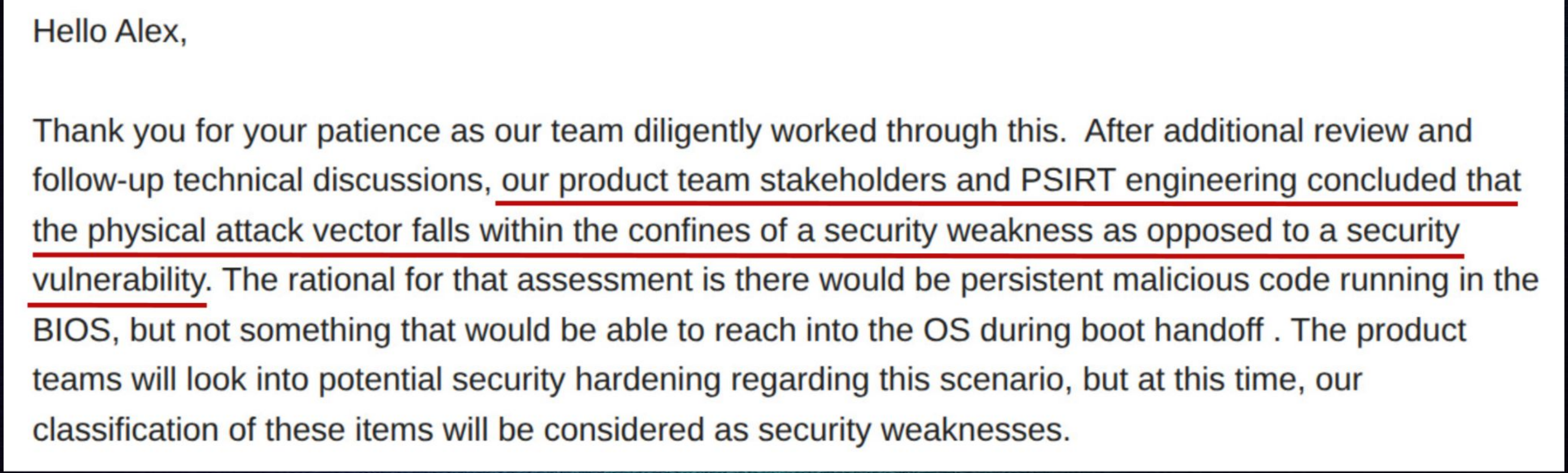
- Vulnerability found in module signed with Microsoft's third-party UEFI certificate ("*Microsoft Corporation UEFI CA 2011*")
- Secure Boot can be bypassed on any device trusting this key
- Microsoft added 14 new hashes to *dbx* as a mitigation during Patch Tuesday



```
RT->GetVariable(L"IhisiParamBuffer", GUID, 0LL, &Size, &VarContent)
...
VarContent->param3 = 0LL;
VarContent->param5 = 0LL;
VarContent->param6 = 0LL;
VarContent->param1 = 0x83EFLL;
VarContent->param2 = '$H20';
VarContent->param4 = 0xB2LL;
...
```

🔥 VarContent is blindly trusted and used for multiple memory writes! 🔥

Physical Attacks are (Sometimes) Out of Scope

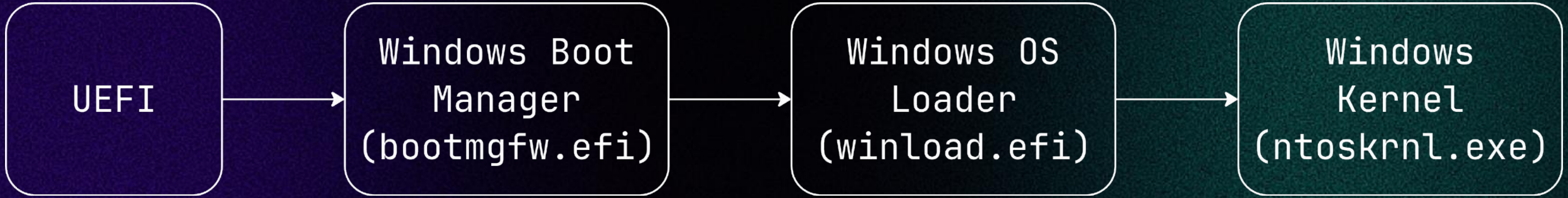


Hello Alex,

Thank you for your patience as our team diligently worked through this. After additional review and follow-up technical discussions, our product team stakeholders and PSIRT engineering concluded that the physical attack vector falls within the confines of a security weakness as opposed to a security vulnerability. The rational for that assessment is there would be persistent malicious code running in the BIOS, but not something that would be able to reach into the OS during boot handoff . The product teams will look into potential security hardening regarding this scenario, but at this time, our classification of these items will be considered as security weaknesses.

The Anatomy of UEFI Bootkits

High-Level Overview of the Windows Boot Process



UEFI

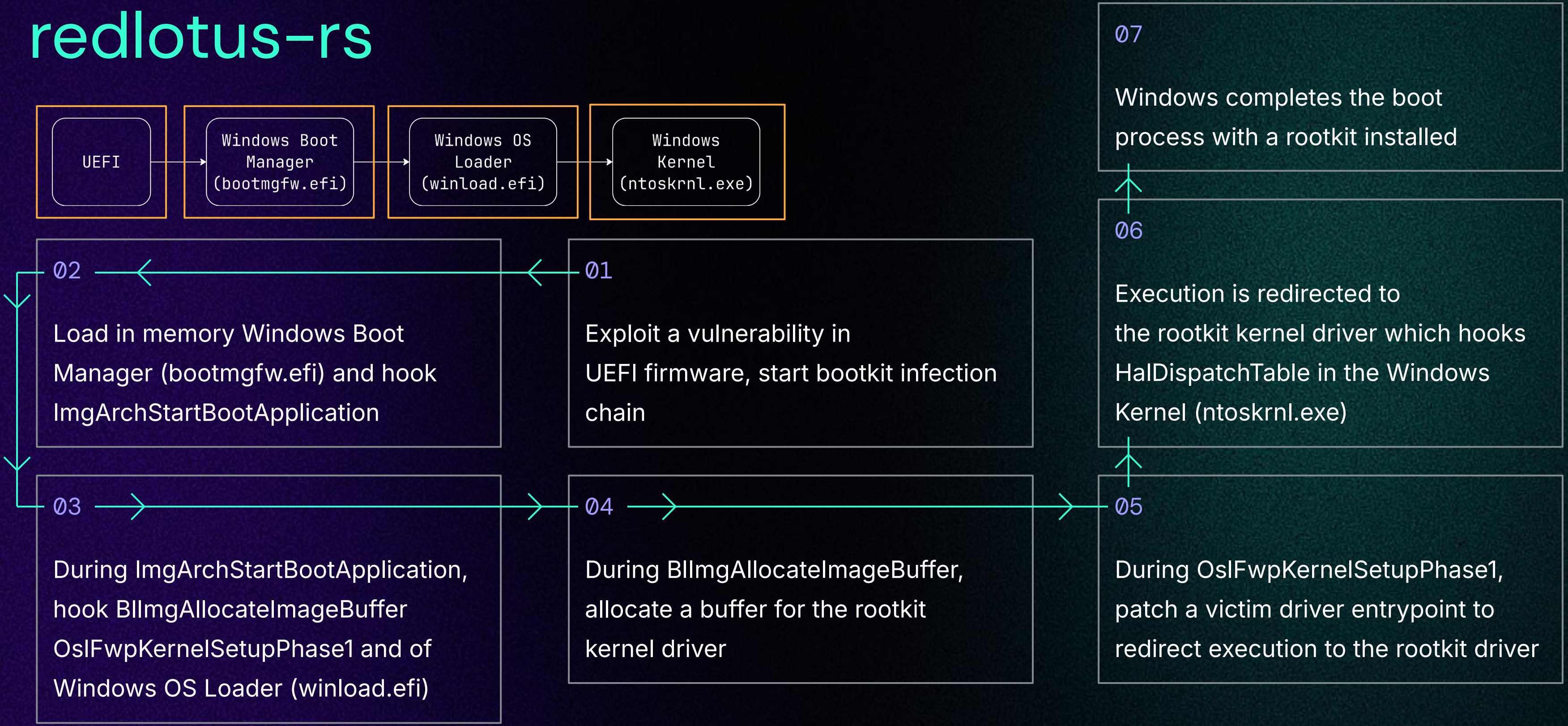
Windows Boot Manager (bootmgfw.efi)

Windows OS Loader (winload.efi)

Windows Kernel (ntoskrnl.exe)

https://www.binarly.io/blog/uefi-bootkit-hunting-in-depth-search-for-unique-code-behavior

The Anatomy of an UEFI Bootkit: redlotus-rs


UEFI
Windows Boot Manager (bootmgfw.efi)
Windows OS Loader (winload.efi)
Windows Kernel (ntoskrnl.exe)

01

Exploit a vulnerability in UEFI firmware, start bootkit infection chain

02

Load in memory Windows Boot Manager (bootmgfw.efi) and hook ImgArchStartBootApplication

03

During ImgArchStartBootApplication, hook BlImgAllocateImageBuffer OslFwpKernelSetupPhase1 and of Windows OS Loader (winload.efi)

04

During BlImgAllocateImageBuffer, allocate a buffer for the rootkit kernel driver

05

During OslFwpKernelSetupPhase1, patch a victim driver entrypoint to redirect execution to the rootkit driver

06

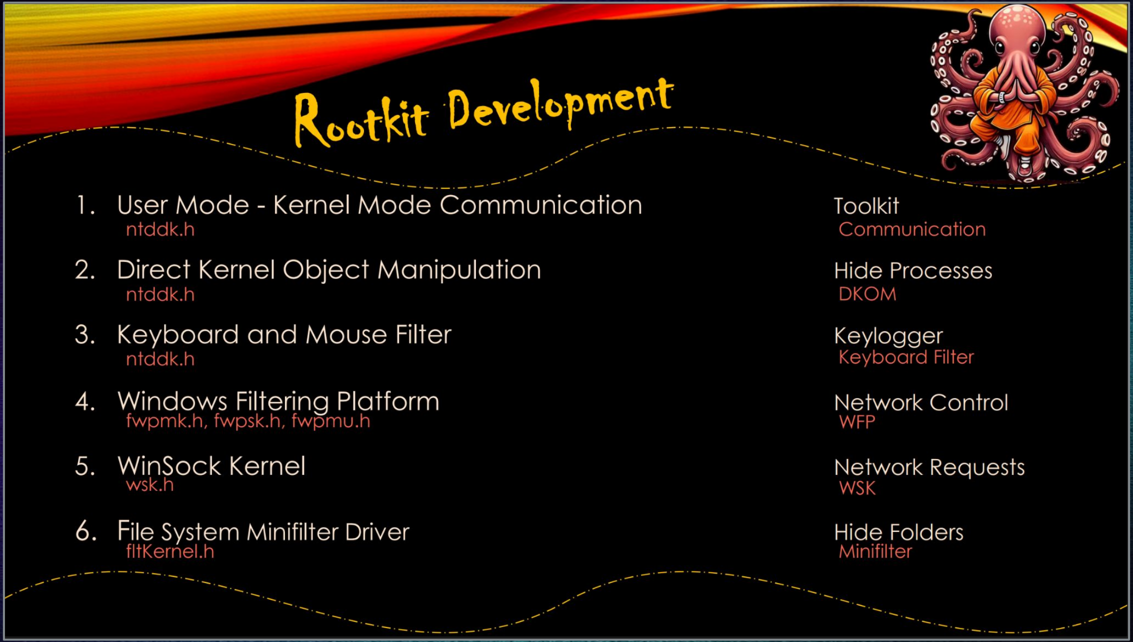
Execution is redirected to the rootkit kernel driver which hooks HalDispatchTable in the Windows Kernel (ntoskrnl.exe)

07

Windows completes the boot process with a rootkit installed

Combining a Secure Boot Bypass with a Bootkit on Windows 11

The Sky's the Limit



Rootkit Development

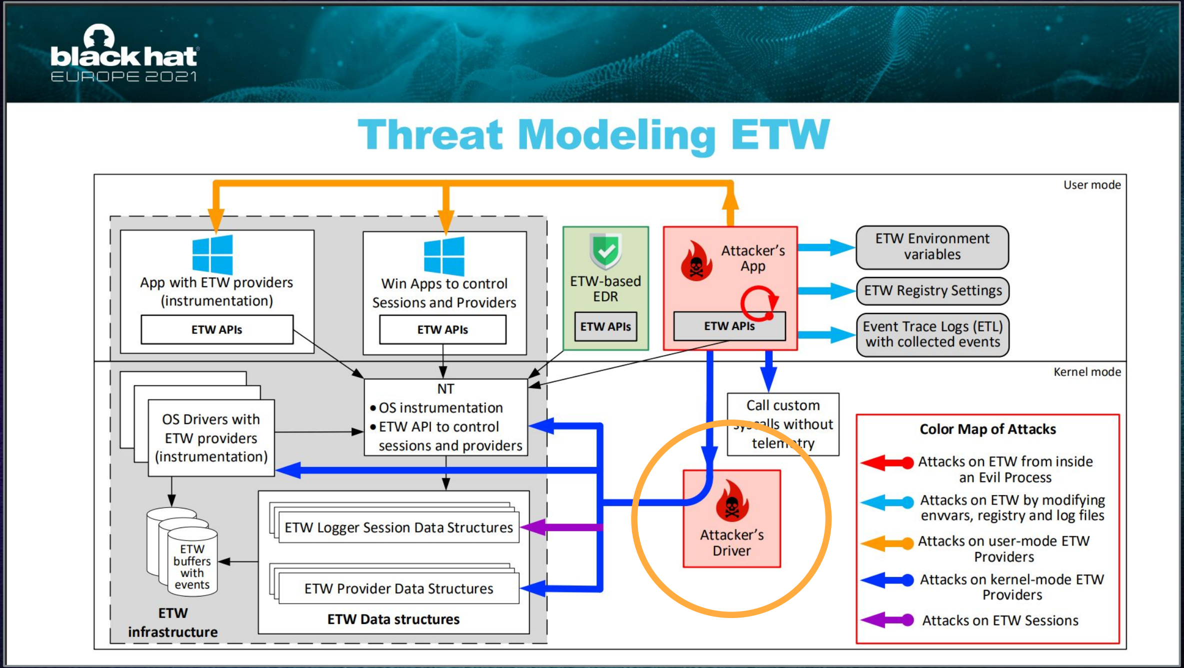
1. User Mode - Kernel Mode Communication ntddk.h	Toolkit Communication
2. Direct Kernel Object Manipulation ntddk.h	Hide Processes DKOM
3. Keyboard and Mouse Filter ntddk.h	Keylogger Keyboard Filter
4. Windows Filtering Platform fwpmk.h, fwpsk.h, fwpmu.h	Network Control WFP
5. WinSock Kernel wsk.h	Network Requests WSK
6. File System Minifilter Driver fltKernel.h	Hide Folders Minifilter

Infecting the Boot to Own the Kernel, Alejandro Vazquez, Maria San Jose, DEF CON 33, https://github.com/TheMalwareGuardian/Abyss

Event Tracing for Windows

- Event Tracing for Windows (ETW) is a native, high-performance Windows telemetry framework that records detailed kernel and user-mode system activity.
- EDR solutions leverage ETW to gain deep visibility into process execution, file operations, registry changes, and network activity using trusted OS-level signals.
- ETW is ideal for EDR because it provides telemetry, enabling real-time detection and forensic analysis without degrading system performance.

Event Tracing for Windows



Threat Modeling ETW

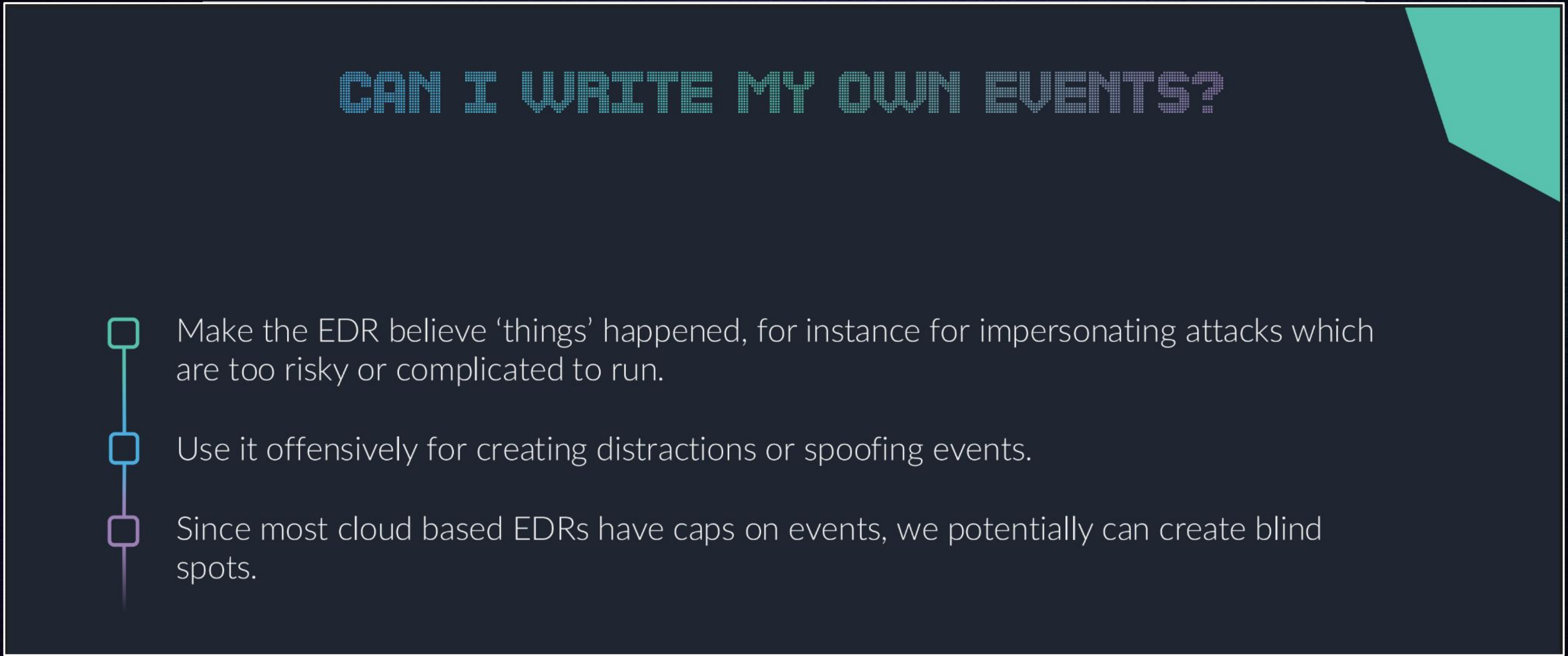
User mode
App with ETW providers (instrumentation)
ETW APIs
Win Apps to control Sessions and Providers
ETW APIs
ETW-based EDR
ETW APIs
Attacker's App
ETW APIs
ETW Environment variables
ETW Registry Settings
Event Trace Logs (ETL) with collected events
Kernel mode
OS Drivers with ETW providers (instrumentation)
NT
• OS instrumentation
• ETW API to control sessions and providers
Call custom syscalls without telemetry
Attacker's Driver
ETW buffers with events
ETW Logger Session Data Structures
ETW Provider Data Structures
ETW Data structures
ETW infrastructure

Color Map of Attacks

- 🟥 Attacks on ETW from inside an Evil Process
- 🟦 Attacks on ETW by modifying envvars, registry and log files
- 🟧 Attacks on user-mode ETW Providers
- 🟦 Attacks on kernel-mode ETW Providers
- 🟪 Attacks on ETW Sessions

Veni, No Vidi, No Vici: Attacks on ETW Blind EDR Sensors, Binarly REsearch Team, BH EU 2021

Event Tracing for Windows



CAN I WRITE MY OWN EVENTS?

- Make the EDR believe ‘things’ happened, for instance for impersonating attacks which are too risky or complicated to run.
- Use it offensively for creating distractions or spoofing events.
- Since most cloud based EDRs have caps on events, we potentially can create blind spots.

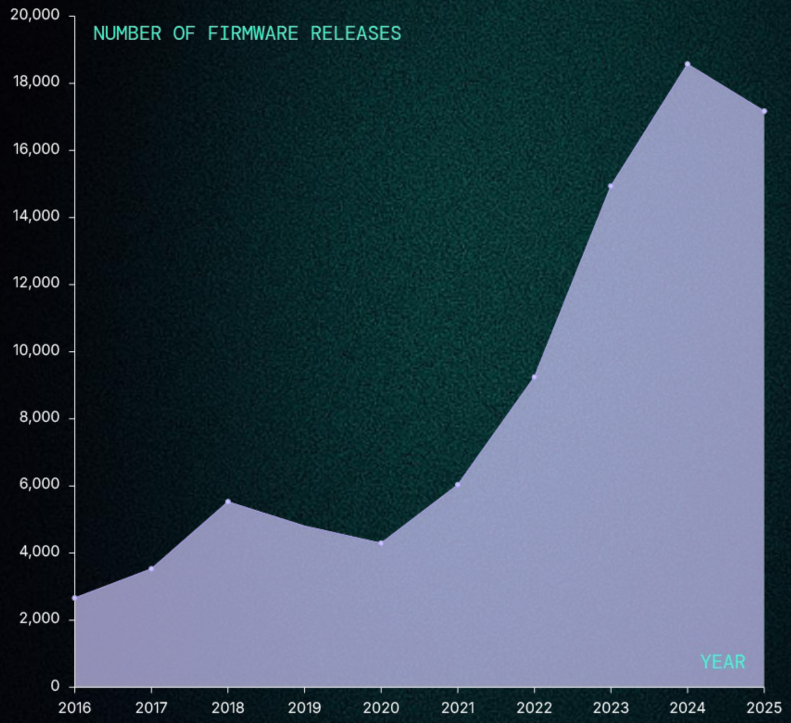
I'm in your logs now, deceiving your analysts and blinding your EDR, Olaf Hartong, BH US 2025 binarly

A Look Inside the UEFI Ecosystem

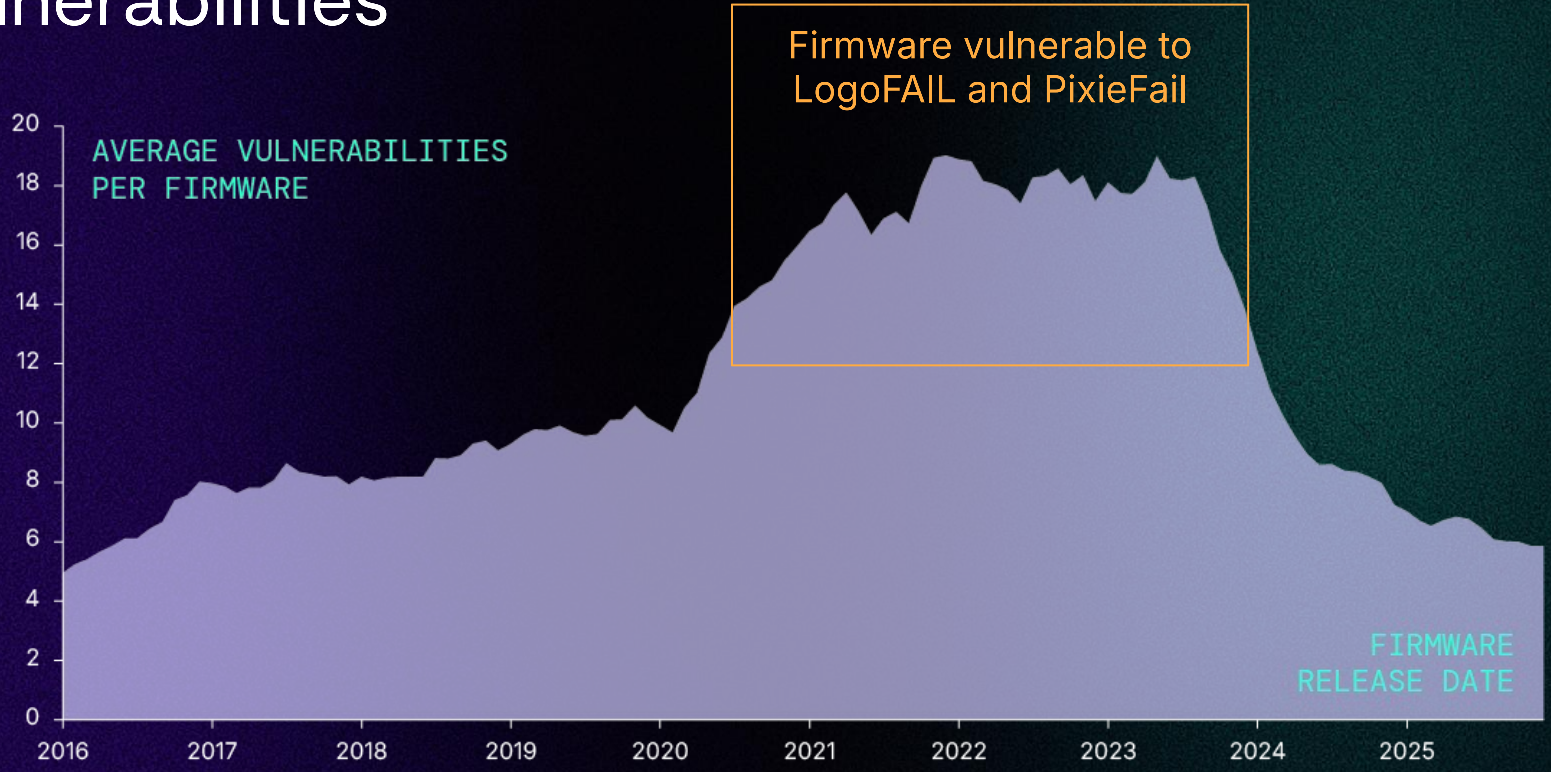
Binarly’s Dataset of UEFI Firmware

Dataset with 80,000 UEFI firmware images:

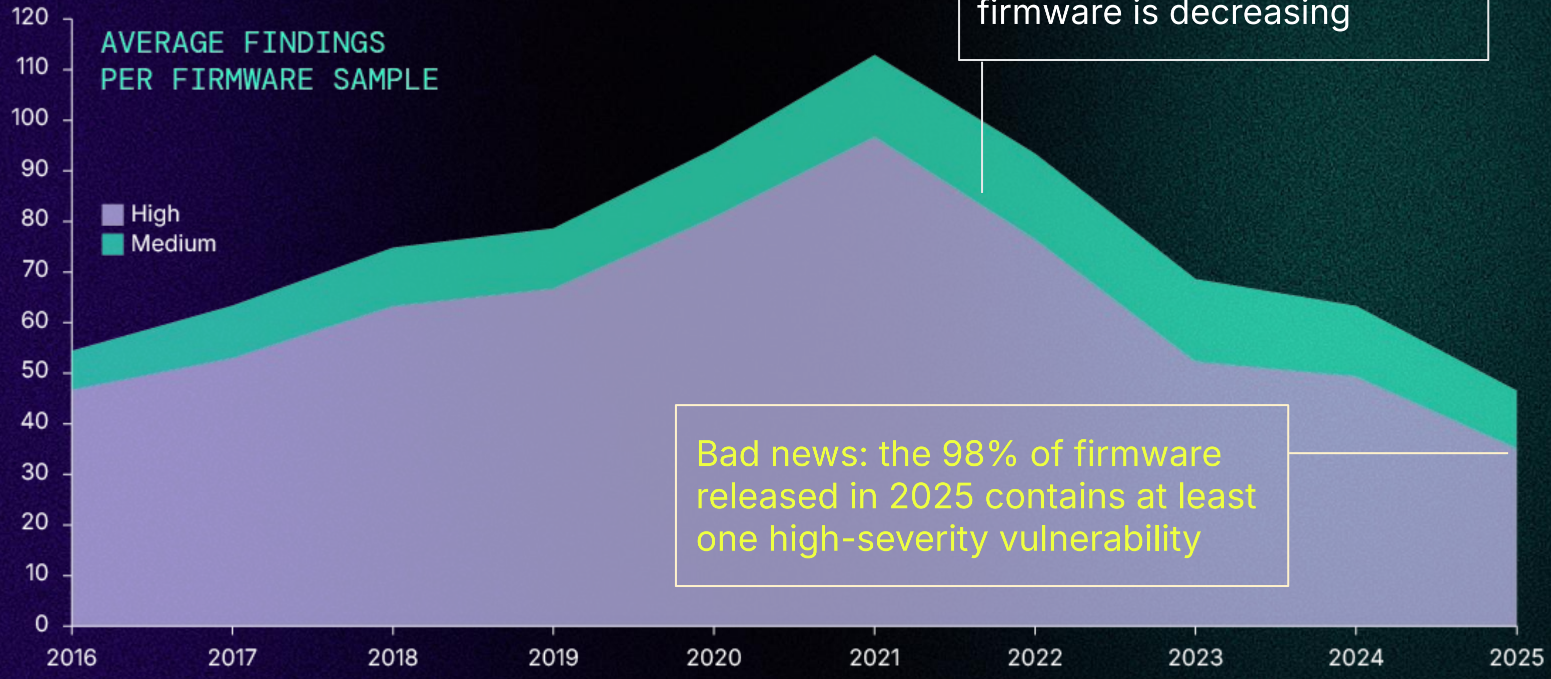
- Spanning over 10 years
- Includes every major vendor (Lenovo, Dell, HP, Intel..)
  - Tracking around 10,000 of recent device models
  - At least one firmware released in the past 4 years
  - 25% can be considered EOL (no firmware released in the last 2 years)



Impact on Known Firmware Vulnerabilities



Impact on Unknown Firmware Vulnerabilities



Good news: the average number of vulnerabilities per firmware is decreasing

Bad news: the 98% of firmware released in 2025 contains at least one high-severity vulnerability

Latest From the Trenches

ESET Research has discovered HybridPetya, on the VirusTotal sample sharing platform. It is a copycat of the **infamous Petya/NotPetya malware**, adding the capability of **compromising UEFI-based systems** and weaponizing CVE-2024-7344 to **bypass UEFI Secure Boot** on outdated systems.

ESET Research

Introducing HybridPetya: Petya/NotPetya copycat with UEFI Secure Boot bypass

UEFI copycat of Petya/NotPetya exploiting CVE-2024-7344 discovered on VirusTotal

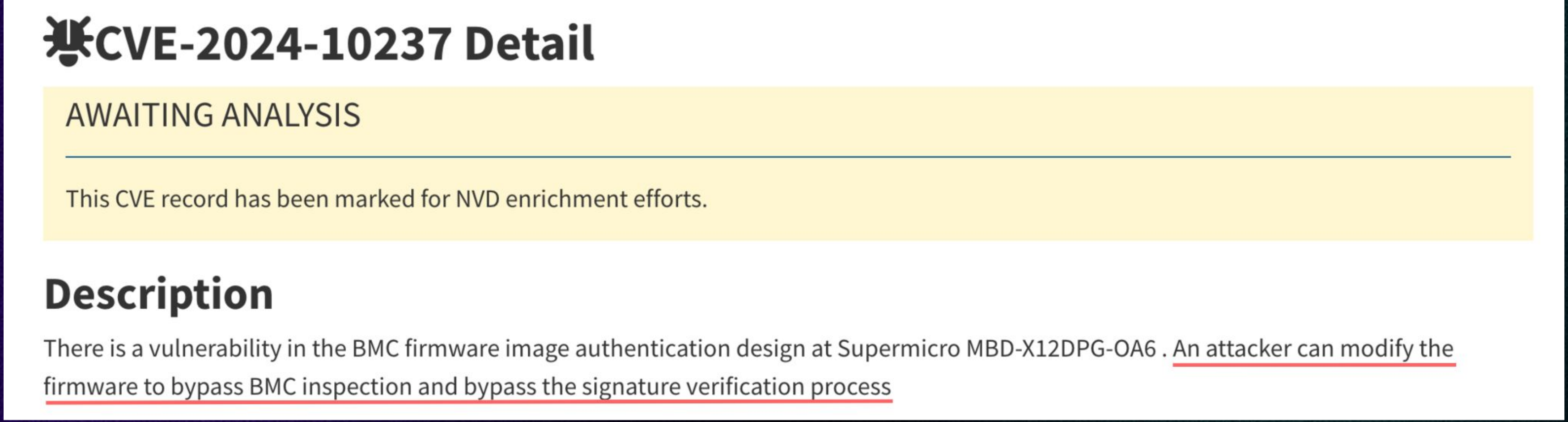
Martin Smolár

12 Sep 2025 • 14 min. read

BMC REsearch

The long chain of Supermicro BMC firmware fixes

- It all started with CVE-2024-10237



CVE-2024-10237 Detail

AWAITING ANALYSIS

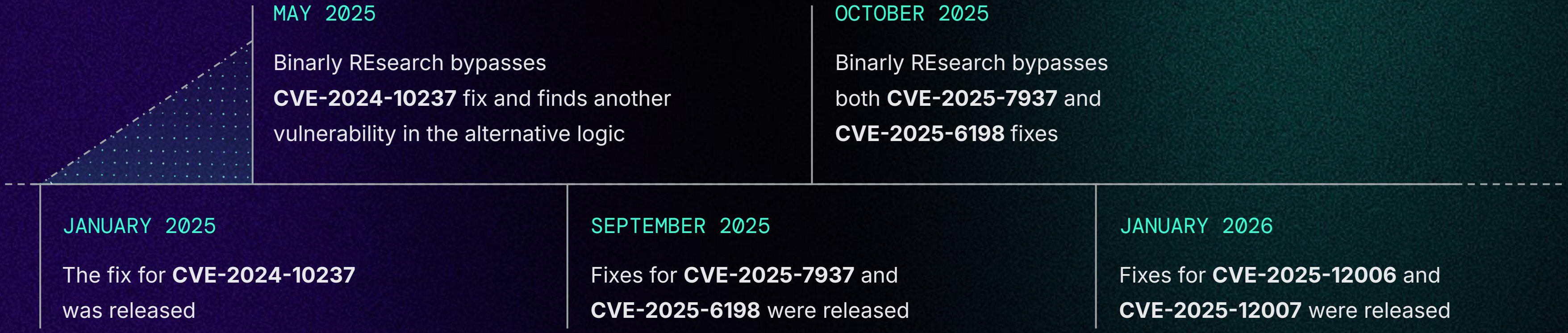
This CVE record has been marked for NVD enrichment efforts.

Description

There is a vulnerability in the BMC firmware image authentication design at Supermicro MBD-X12DPG-OA6 . An attacker can modify the firmware to bypass BMC inspection and bypass the signature verification process

The long chain of Supermicro BMC firmware fixes

- It all started with CVE-2024-10237
- It took Supermicro one year and three release cycles to resolve the issues
- Fixes for CVE-2025-12006 and CVE-2025-12007 were released in January 2026



JANUARY 2025

The fix for **CVE-2024-10237** was released

MAY 2025

Binarly REsearch bypasses **CVE-2024-10237** fix and finds another vulnerability in the alternative logic

SEPTEMBER 2025

Fixes for **CVE-2025-7937** and **CVE-2025-6198** were released

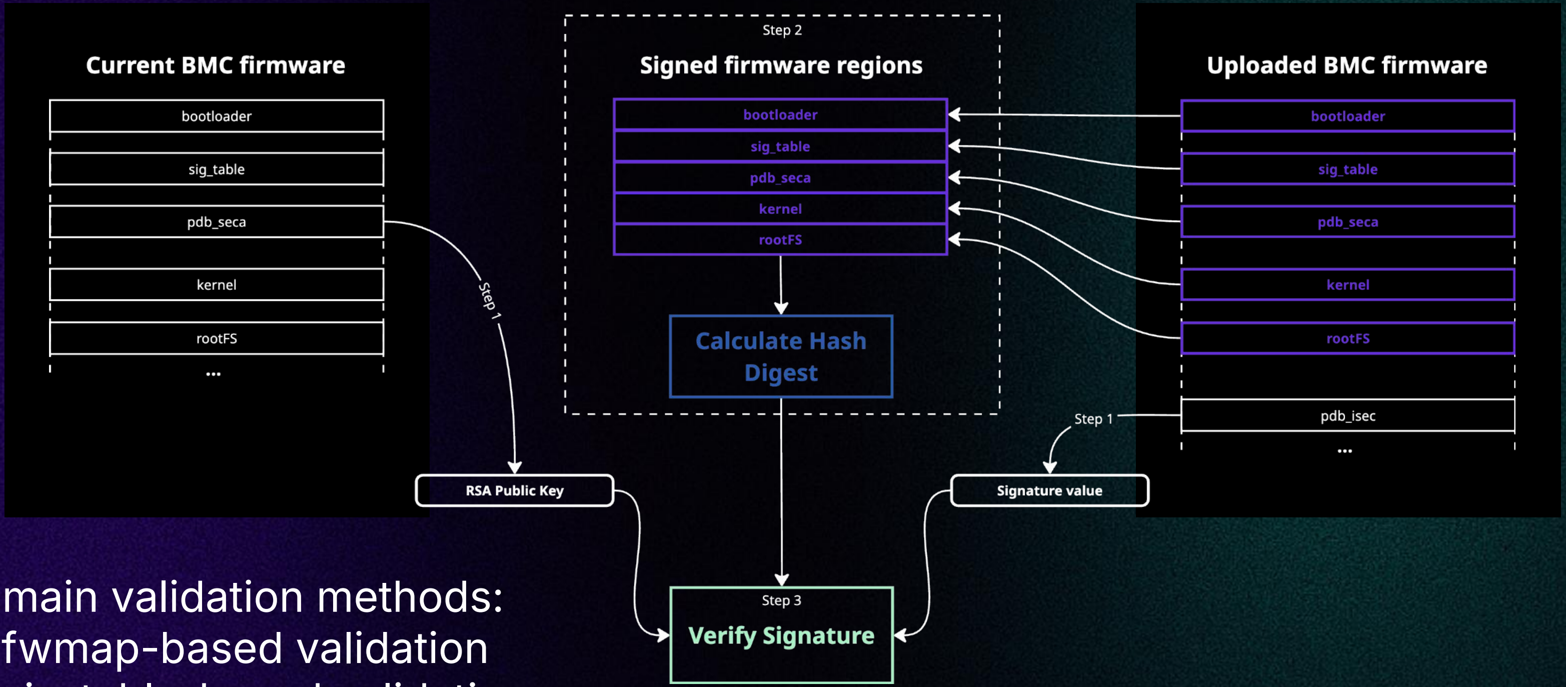
OCTOBER 2025

Binarly REsearch bypasses both **CVE-2025-7937** and **CVE-2025-6198** fixes

JANUARY 2026

Fixes for **CVE-2025-12006** and **CVE-2025-12007** were released

Supermicro BMC validation



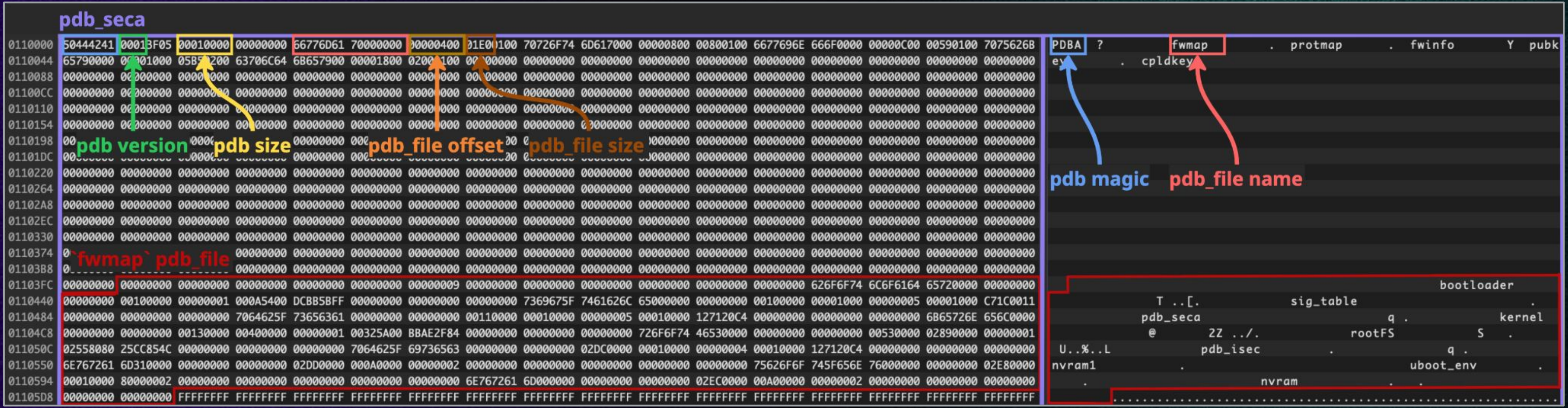
Two main validation methods:

- fwmap-based validation
- sig_table-based validation

fwmap-based validation

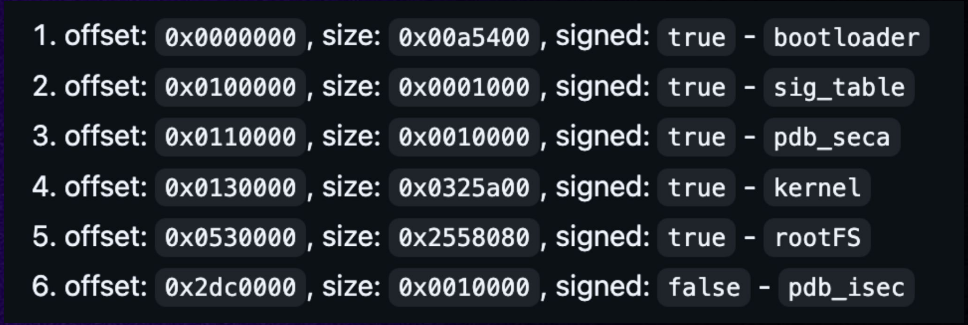
fwmap table contains information about the firmware regions:

- offset
- size
- attributes (e.g. whether the region is signed or not)



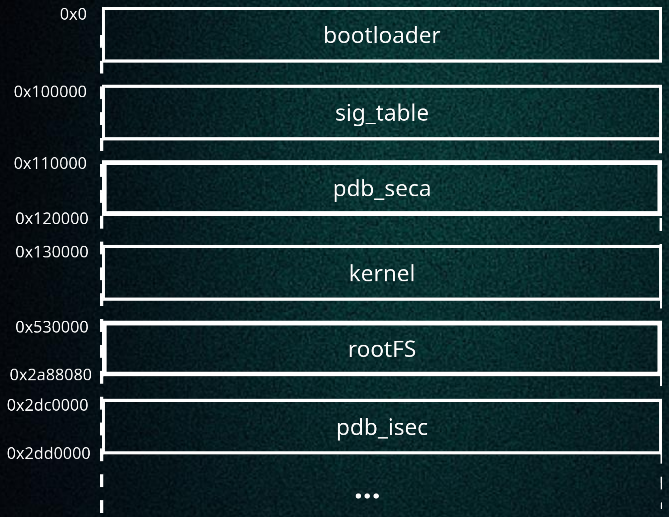
```
0110000 50444241 00013F05 00010000 00000000 66776D61 70000000 00000400 01E00100 70726F74 6D617000 00000800 00800100 6677696E 666F0000 00000C00 00590100 7075626B  PDBA  ?        fwmap        .  protmap      .  fwinfo        Y  pubk
0110044 65790000 00001000 05B50200 63706C64 6B657900 00001800 02000100 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000  ey        .  cpldkey
0110088 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
01100CC 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0110110 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0110154 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0110198 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
01101DC 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0110220 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0110264 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
01102A8 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
01102EC 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0110330 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0110374 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
01103B8 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
01103FC 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000009 00000000 00000000 00000000 00000000 00000000 626F6F74 6C6F6164 65720000 00000000                                                  bootloader
0110440 00000000 00100000 00000001 000A5400 DCBB5BFF 00000000 00000000 00000000 7369675F 7461626C 65000000 00000000 00100000 00001000 00000005 00001000 C71C0011               T ..[.              sig_table                    .
0110484 00000000 00000000 00000000 7064625F 73656361 00000000 00000000 00110000 00010000 00000005 00010000 127120C4 00000000 00000000 00000000 6B65726E 656C0000             pdb_seca                         q .               kernel
01104C8 00000000 00000000 00130000 00400000 00000001 00325A00 BBAE2F84 00000000 00000000 00000000 726F6F74 46530000 00000000 00000000 00530000 02890000 00000001               @       2Z ../.             rootFS             S   .
011050C 02558080 25CC854C 00000000 00000000 00000000 7064625F 69736563 00000000 00000000 02DC0000 00010000 00000004 00010000 127120C4 00000000 00000000 00000000   U..%..L            pdb_isec          .                q .
0110550 6E767261 6D310000 00000000 00000000 02DD0000 000A0000 00000002 00000000 00000000 00000000 00000000 00000000 75626F6F 745F656E 76000000 00000000 02E80000  nvram1          .                         uboot_env         .
0110594 00010000 80000002 00000000 00000000 00000000 00000000 00000000 6E767261 6D000000 00000000 00000000 02EC0000 00A00000 00000002 00000000 00000000 00000000      .                nvram              .  .
01105D8 00000000 00000000 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF          ............................................................
```

fwmap from Supermicro X12STW-F



1. offset: `0x0000000`, size: `0x00a5400`, signed: `true` - `bootloader`
2. offset: `0x0100000`, size: `0x0001000`, signed: `true` - `sig_table`
3. offset: `0x0110000`, size: `0x0010000`, signed: `true` - `pdb_seca`
4. offset: `0x0130000`, size: `0x0325a00`, signed: `true` - `kernel`
5. offset: `0x0530000`, size: `0x2558080`, signed: `true` - `rootFS`
6. offset: `0x2dc0000`, size: `0x0010000`, signed: `false` - `pdb_isec`

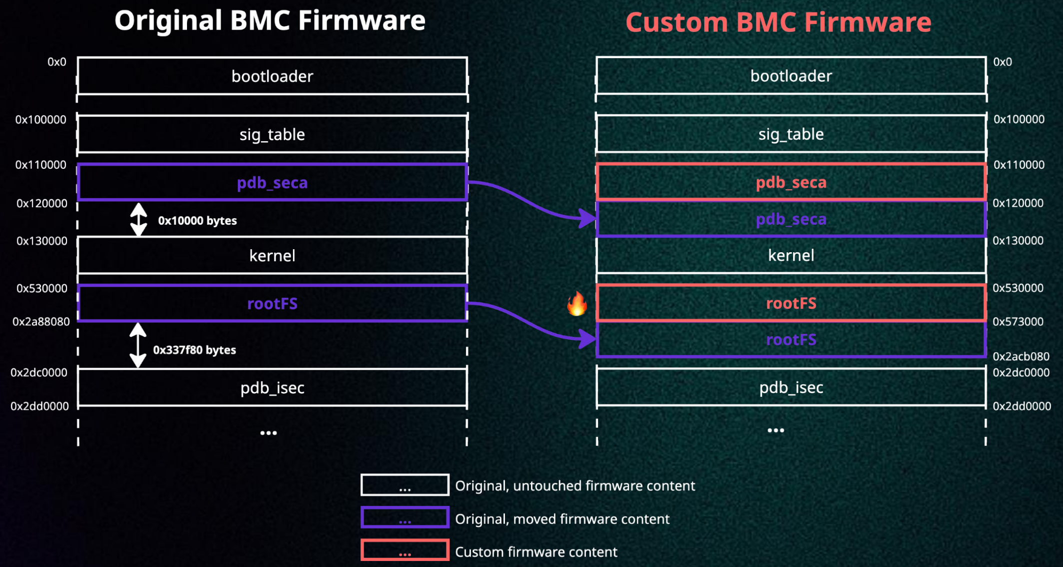
Original BMC Firmware



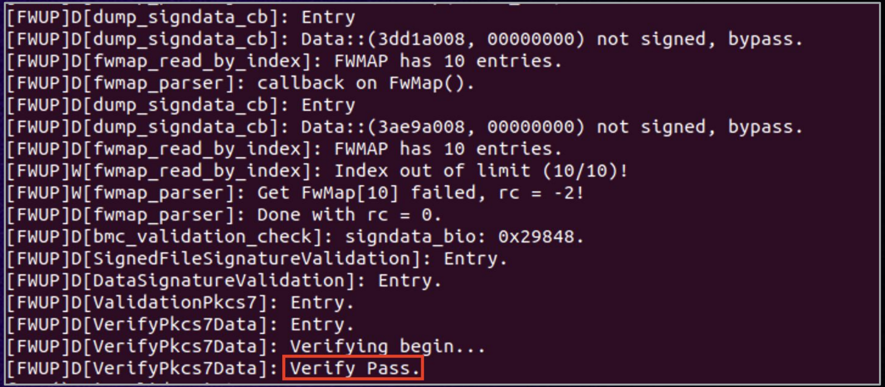
CVE-2024-10237: PoC

Custom *fwmap*

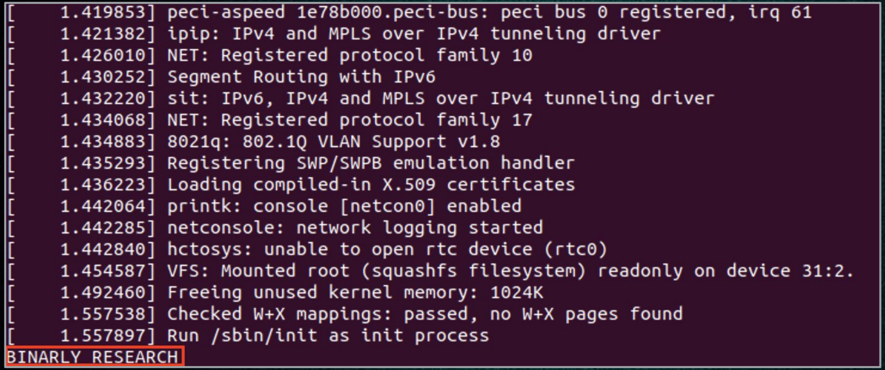
1. offset: `0x0000000`, size: `0x00a5400`, signed: `true` - `bootloader`
2. offset: `0x0100000`, size: `0x0001000`, signed: `true` - `sig_table`
3. offset: `0x0120000`, size: `0x0010000`, signed: `true` - `pdb_seca`
4. offset: `0x0130000`, size: `0x0325a00`, signed: `true` - `kernel`
5. offset: `0x0573000`, size: `0x2558080`, signed: `true` - `rootFS`
6. offset: `0x2dc0000`, size: `0x0010000`, signed: `false` - `pdb_isec`



CVE-2024-10237: Demo



```
[FWUP]D[dump_signdata_cb]: Entry
[FWUP]D[dump_signdata_cb]: Data::(3dd1a008, 00000000) not signed, bypass.
[FWUP]D[fwmap_read_by_index]: FWMAP has 10 entries.
[FWUP]D[fwmap_parser]: callback on FwMap().
[FWUP]D[dump_signdata_cb]: Entry
[FWUP]D[dump_signdata_cb]: Data::(3ae9a008, 00000000) not signed, bypass.
[FWUP]D[fwmap_read_by_index]: FWMAP has 10 entries.
[FWUP]W[fwmap_read_by_index]: Index out of limit (10/10)!
[FWUP]W[fwmap_parser]: Get FwMap[10] failed, rc = -2!
[FWUP]D[fwmap_parser]: Done with rc = 0.
[FWUP]D[bmc_validation_check]: signdata_bio: 0x29848.
[FWUP]D[SignedFileSignatureValidation]: Entry.
[FWUP]D[DataSignatureValidation]: Entry.
[FWUP]D[ValidationPkcs7]: Entry.
[FWUP]D[VerifyPkcs7Data]: Entry.
[FWUP]D[VerifyPkcs7Data]: Verifying begin...
[FWUP]D[VerifyPkcs7Data]: Verify Pass.
```

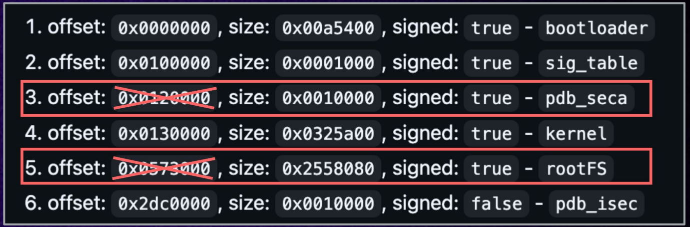


```
[    1.419853] peci-aspeed 1e78b000.peci-bus: peci bus 0 registered, irq 61
[    1.421382] ipip: IPv4 and MPLS over IPv4 tunneling driver
[    1.426010] NET: Registered protocol family 10
[    1.430252] Segment Routing with IPv6
[    1.432220] sit: IPv6, IPv4 and MPLS over IPv4 tunneling driver
[    1.434068] NET: Registered protocol family 17
[    1.434883] 8021q: 802.1Q VLAN Support v1.8
[    1.435293] Registering SWP/SWPB emulation handler
[    1.436223] Loading compiled-in X.509 certificates
[    1.442064] printk: console [netcon0] enabled
[    1.442285] netconsole: network logging started
[    1.442840] hctosys: unable to open rtc device (rtc0)
[    1.454587] VFS: Mounted root (squashfs filesystem) readonly on device 31:2.
[    1.492460] Freeing unused kernel memory: 1024K
[    1.557538] Checked W+X mappings: passed, no W+X pages found
[    1.557897] Run /sbin/init as init process
BINARLY RESEARCH
```

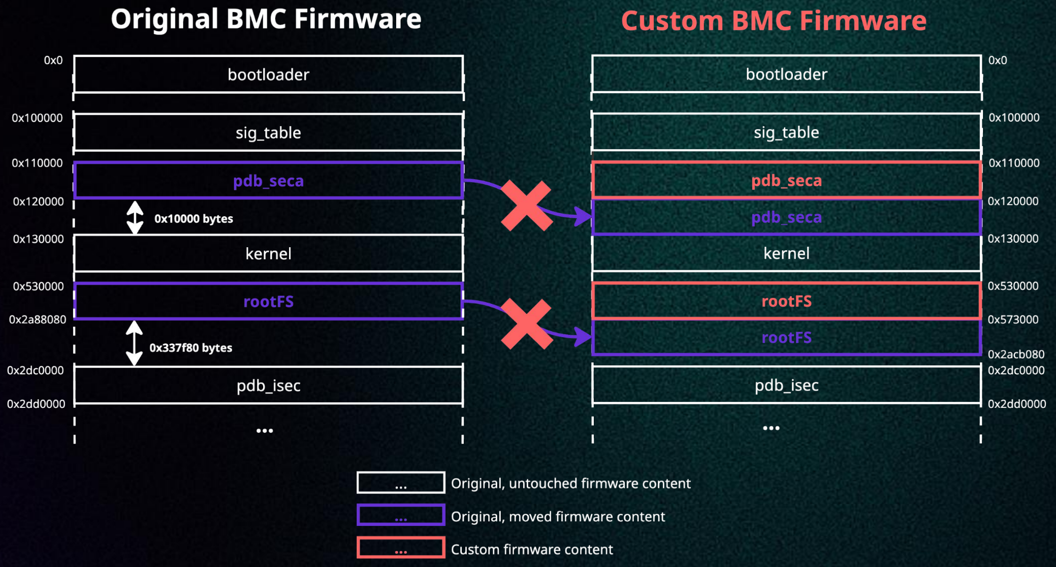
CVE-2024-10237: Supermicro’s Patch

- No custom region offsets in *fwmap*, only **whitelisted** offsets can be used
- Only certain regions can have ***is_signed*** flag

Custom *fwmap*



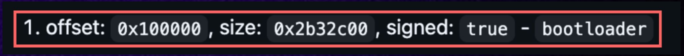
1. offset: `0x0000000`, size: `0x00a5400`, signed: `true` - `bootloader`
2. offset: `0x0100000`, size: `0x0001000`, signed: `true` - `sig_table`
3. offset: ~~`0x0120000`~~, size: `0x0010000`, signed: `true` - `pdb_seca`
4. offset: `0x0130000`, size: `0x0325a00`, signed: `true` - `kernel`
5. offset: ~~`0x0573000`~~, size: `0x2558080`, signed: `true` - `rootFS`
6. offset: `0x2dc0000`, size: `0x0010000`, signed: `false` - `pdb_isec`



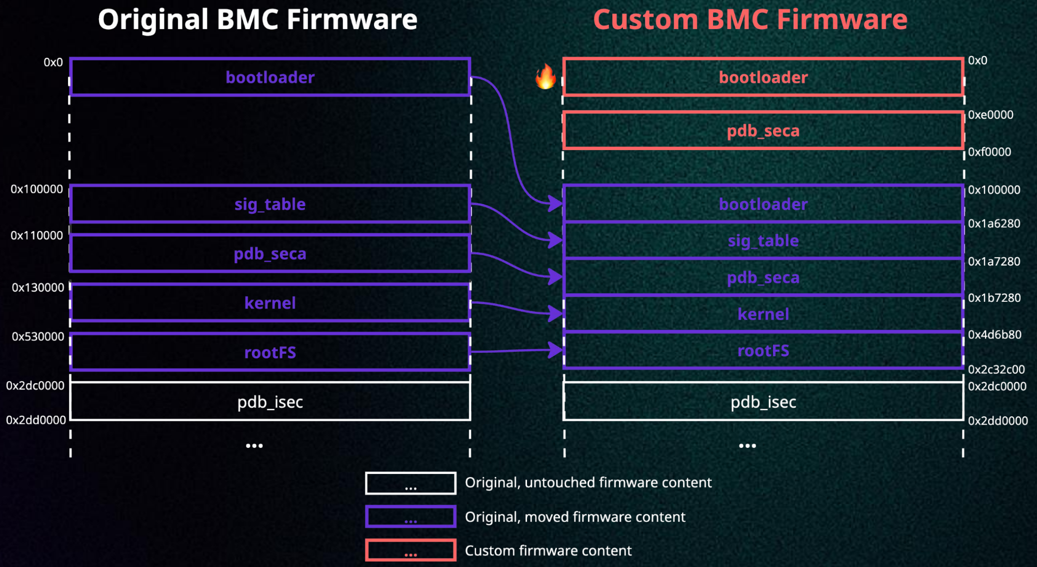
CVE-2024-10237: Bypassing the patch

- Move all the signed regions at whitelisted offset 0x100000
- Add entry in the custom *fwmap* and name it bootloader

Custom *fwmap*



```
1. offset: 0x100000 , size: 0x2b32c00 , signed: true - bootloader
```



Original BMC Firmware

0x0 bootloader

0x100000 sig_table

0x110000 pdb_seca

0x130000 kernel

0x530000 rootFS

0x2dc0000 pdb_isec 0x2dd0000

...

Custom BMC Firmware

bootloader 0x0

pdb_seca 0xe0000 0xf0000

bootloader 0x100000

sig_table 0x1a6280

pdb_seca 0x1a7280

kernel 0x1b7280

rootFS 0x4d6b80 0x2c32c00

pdb_isec 0x2dc0000 0x2dd0000

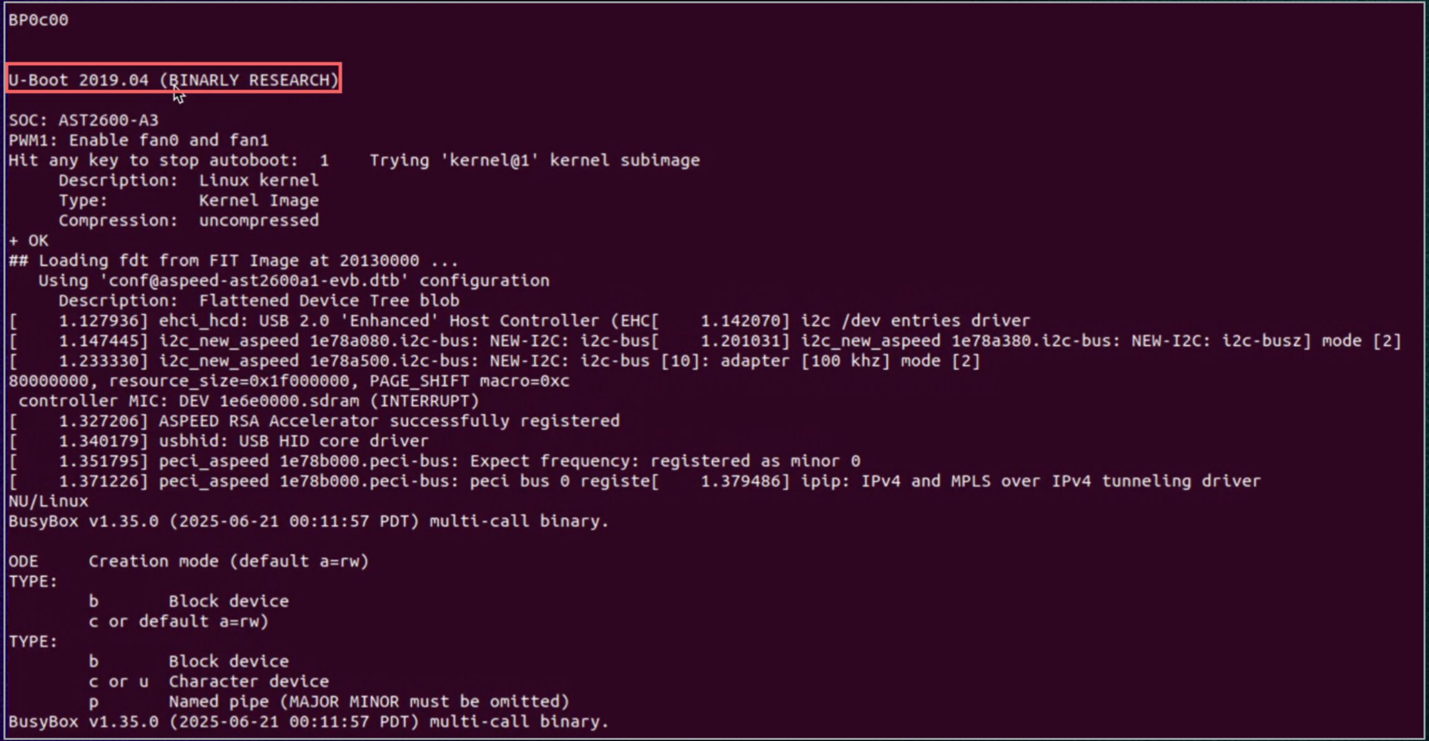
...

... Original, untouched firmware content

... Original, moved firmware content

... Custom firmware content

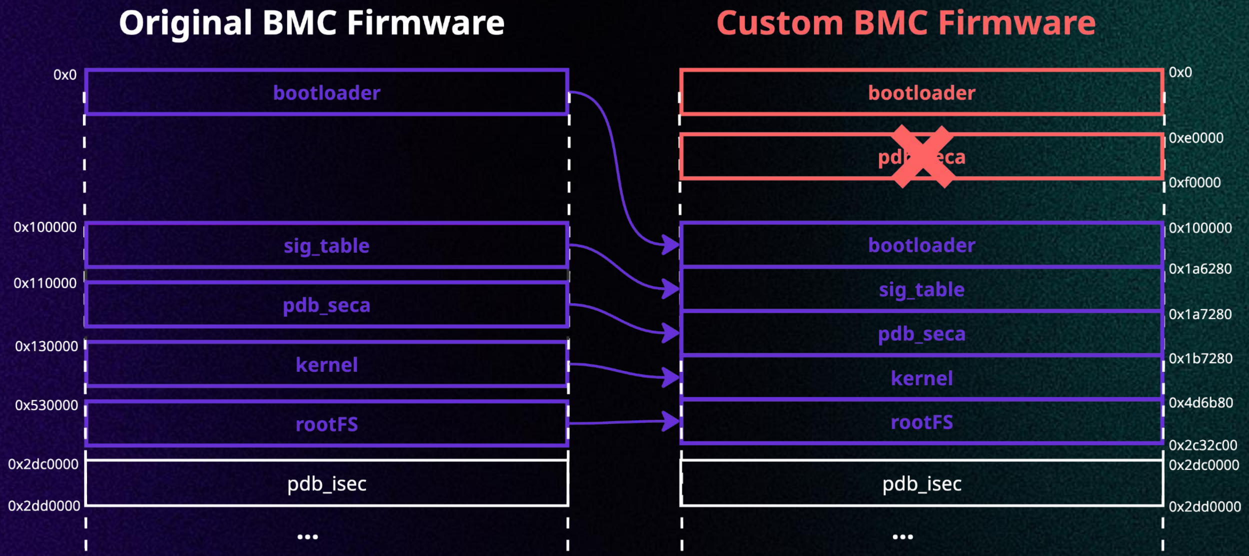
CVE-2025-7937: Demo



```
BP0c00


U-Boot 2019.04 (BINARLY RESEARCH)

SOC: AST2600-A3
PWM1: Enable fan0 and fan1
Hit any key to stop autoboot:  1    Trying 'kernel@1' kernel subimage
     Description:  Linux kernel
     Type:         Kernel Image
     Compression:  uncompressed
+ OK
## Loading fdt from FIT Image at 20130000 ...
   Using 'conf@aspeed-ast2600a1-evb.dtb' configuration
     Description:  Flattened Device Tree blob
[    1.127936] ehci_hcd: USB 2.0 'Enhanced' Host Controller (EHC[    1.142070] i2c /dev entries driver
[    1.147445] i2c_new_aspeed 1e78a080.i2c-bus: NEW-I2C: i2c-bus[    1.201031] i2c_new_aspeed 1e78a380.i2c-bus: NEW-I2C: i2c-busz] mode [2]
[    1.233330] i2c_new_aspeed 1e78a500.i2c-bus: NEW-I2C: i2c-bus [10]: adapter [100 khz] mode [2]
80000000, resource_size=0x1f000000, PAGE_SHIFT macro=0xc
 controller MIC: DEV 1e6e0000.sdram (INTERRUPT)
[    1.327206] ASPEED RSA Accelerator successfully registered
[    1.340179] usbhid: USB HID core driver
[    1.351795] peci_aspeed 1e78b000.peci-bus: Expect frequency: registered as minor 0
[    1.371226] peci_aspeed 1e78b000.peci-bus: peci bus 0 registe[    1.379486] ipip: IPv4 and MPLS over IPv4 tunneling driver
NU/Linux
BusyBox v1.35.0 (2025-06-21 00:11:57 PDT) multi-call binary.

ODE     Creation mode (default a=rw)
TYPE:
        b       Block device
        c or default a=rw)
TYPE:
        b       Block device
        c or u  Character device
        p       Named pipe (MAJOR MINOR must be omitted)
BusyBox v1.35.0 (2025-06-21 00:11:57 PDT) multi-call binary.
```

CVE-2025-7937: Supermicro’s patch

- Checks that offset of processed *pdb_seca* is 0x110000
- *fwmap* must contain a region where offset <= *pdb_seca offset* < *offset + size*



Original BMC Firmware

0x0 bootloader

0x100000 sig_table

0x110000 pdb_seca

0x130000 kernel

0x530000 rootFS

0x2dc0000 pdb_isec

0x2dd0000

...

Custom BMC Firmware

0x0 bootloader

0xe0000 ~~pdb_seca~~

0xf0000

0x100000 bootloader

0x1a6280 sig_table

0x1a7280 pdb_seca

0x1b7280 kernel

0x4d6b80 rootFS

0x2c32c00

0x2dc0000 pdb_isec

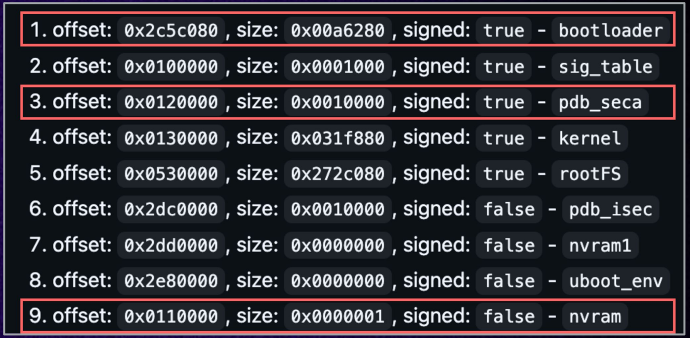
0x2dd0000

...

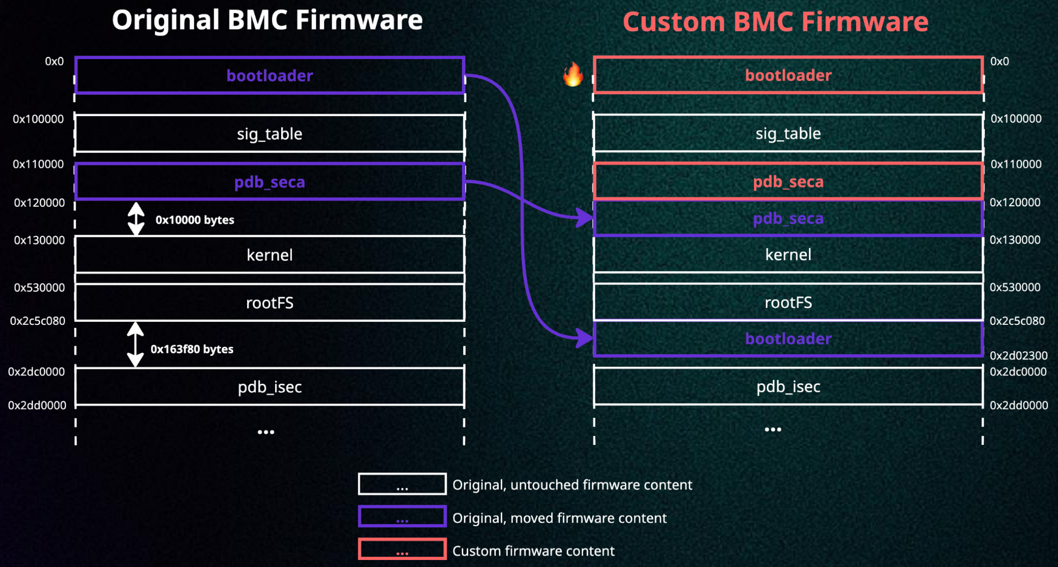
CVE-2025-7937: Bypassing the second patch

- Previously implemented checks were removed :)
  - We can add *fwmap* entries at custom offsets again!

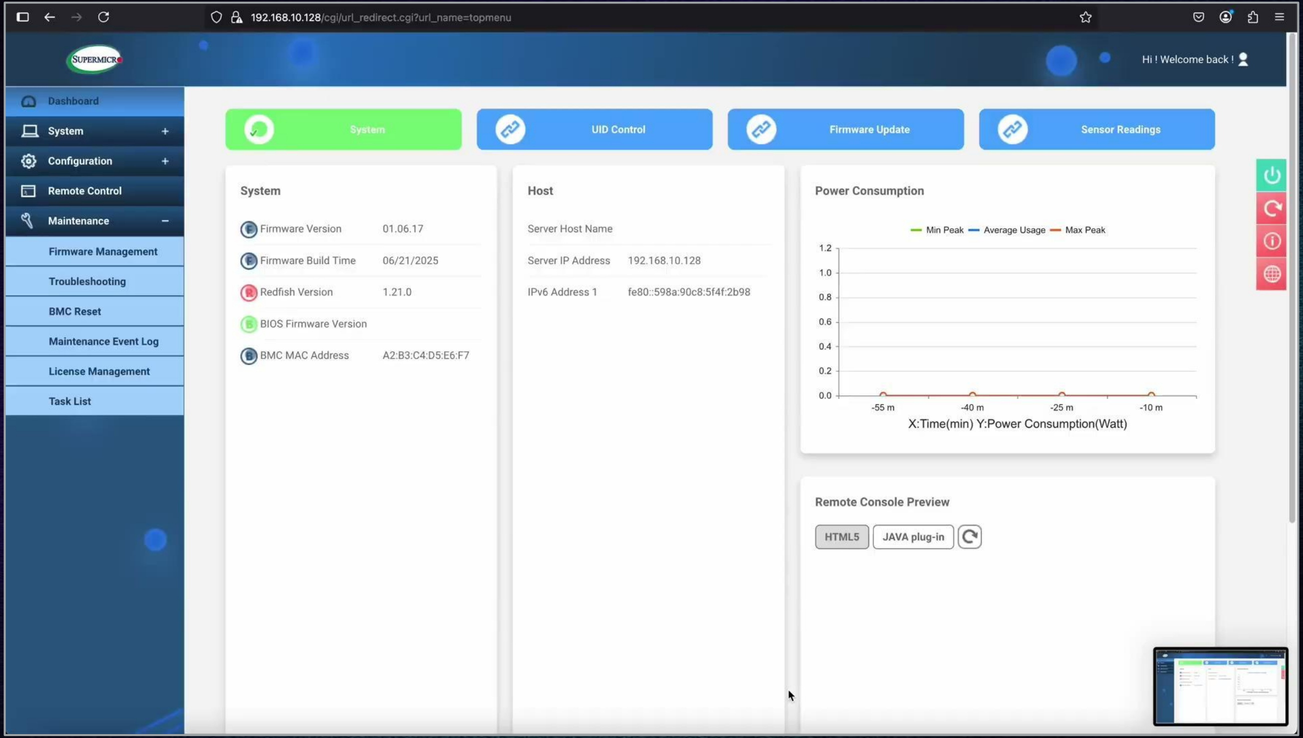
Custom *fwmap*



```
1. offset: 0x2c5c080 , size: 0x00a6280 , signed: true - bootloader
2. offset: 0x0100000 , size: 0x0001000 , signed: true - sig_table
3. offset: 0x0120000 , size: 0x0010000 , signed: true - pdb_seca
4. offset: 0x0130000 , size: 0x031f880 , signed: true - kernel
5. offset: 0x0530000 , size: 0x272c080 , signed: true - rootFS
6. offset: 0x2dc0000 , size: 0x0010000 , signed: false - pdb_isec
7. offset: 0x2dd0000 , size: 0x0000000 , signed: false - nvram1
8. offset: 0x2e80000 , size: 0x0000000 , signed: false - uboot_env
9. offset: 0x0110000 , size: 0x0000001 , signed: false - nvram
```



CVE-2025-12006: Demo



192.168.10.128/cgi/url_redirect.cgi?url_name=topmenu

Hi ! Welcome back !

System

Firmware Version 01.06.17

Firmware Build Time 06/21/2025

Redfish Version 1.21.0

BIOS Firmware Version

BMC MAC Address A2:B3:C4:D5:E6:F7

Host

Server Host Name

Server IP Address 192.168.10.128

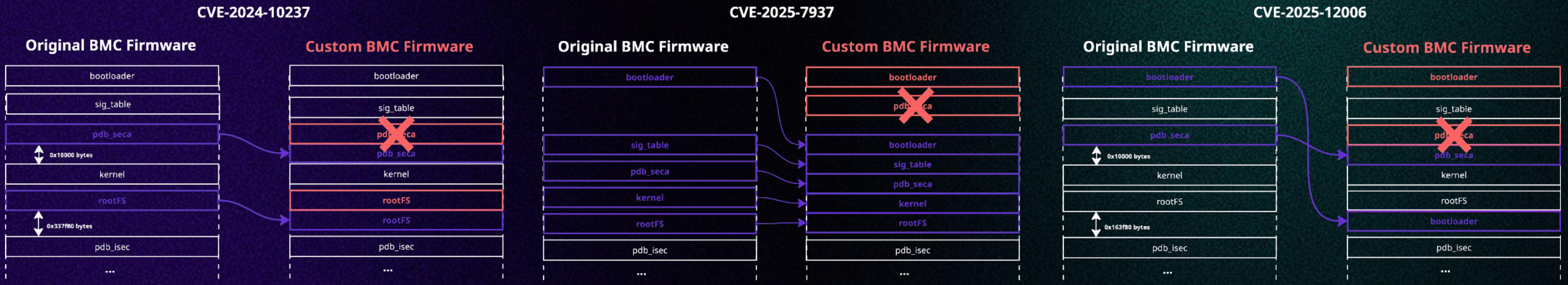
IPv6 Address 1 fe80::598a:90c8:5f4f:2b98

Power Consumption

Remote Console Preview

CVE-2025-12006: Supermicro’s final patch

- Offset of parsed pdb_seca should be equal to 0x110000
- For pdb_seca region defined in *fwmap*:
  - offset should be 0x110000
  - size should be 0x10000
  - it should have is_signed attribute
- Other fwmap regions should be located at only allowed offsets
  - For some regions, their size and attributes are also checked



CVE-2025-12006: Supermicro’s final patch

Fixes provided with the latest firmware release mitigate the issues, **but**:

- For both X12STW-F (*fwmap*) and X13SEM-F (*sig_table*), RSA keys used for image signing were not rotated
  - Firmware downgrade is not possible due to other changes, but may arise in the future
- For X13SEM-F, the required validation logic was added to the *libipmi.so* library, but before it was executed in the *OP-TEE* environment
  - Potential attackers with root privileges to the BMC system could bypass the introduced checks

What about *sig_table*-based validation?

- Similar logic, similar problems – CVE-2025-6198, CVE-2025-12007
- Blogpost coming soon, stay tuned!

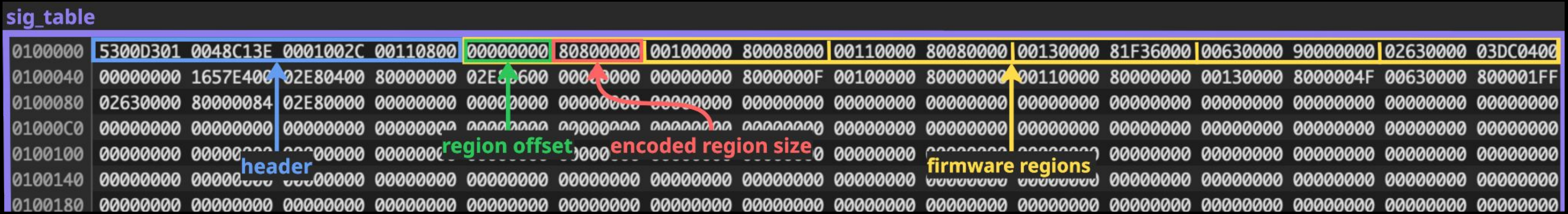
Conclusions

- Firmware is ubiquitous, complex and not tested enough
- Number of bugs in the UEFI ecosystem are declining, still almost every firmware out there has 1+ high-severity bug
- Bugs in UEFI can impact the boot process and OS integrity
- BMC firmware validation is not a trivial task

Backup Slides

sig_table-based validation

- Similar to *fwmap*, contains information about **signed** firmware regions:
  - offset
  - size
- Always located at fixed offset **0x100000**

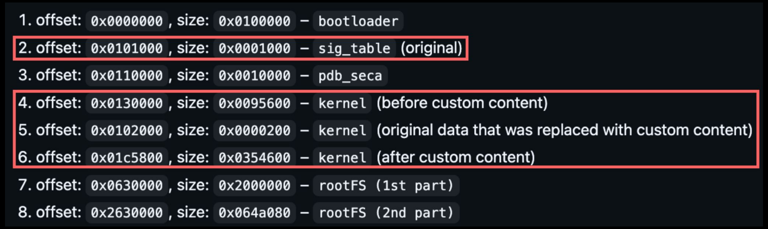


sig_table

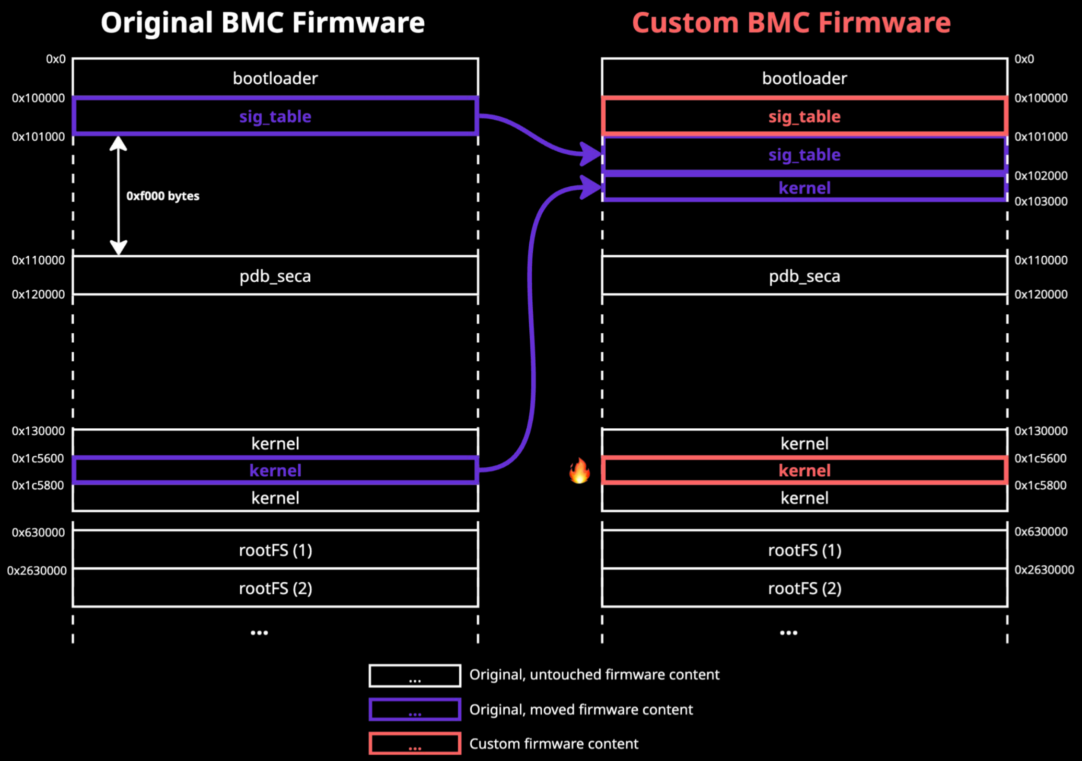
```
0100000  5300D301 0048C13E 0001002C 00110800 00000000 80800000 00100000 80008000 00110000 80080000 00130000 81F36000 00630000 90000000 02630000 03DC0400
0100040  00000000 1657E400 02E80400 80000000 02E80600 00000000 00000000 8000000F 00100000 80000000 00110000 80000000 00130000 8000004F 00630000 800001FF
0100080  02630000 80000084 02E80000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
01000C0  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0100100  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0100140  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0100180  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

CVE-2025-6198 – PoC

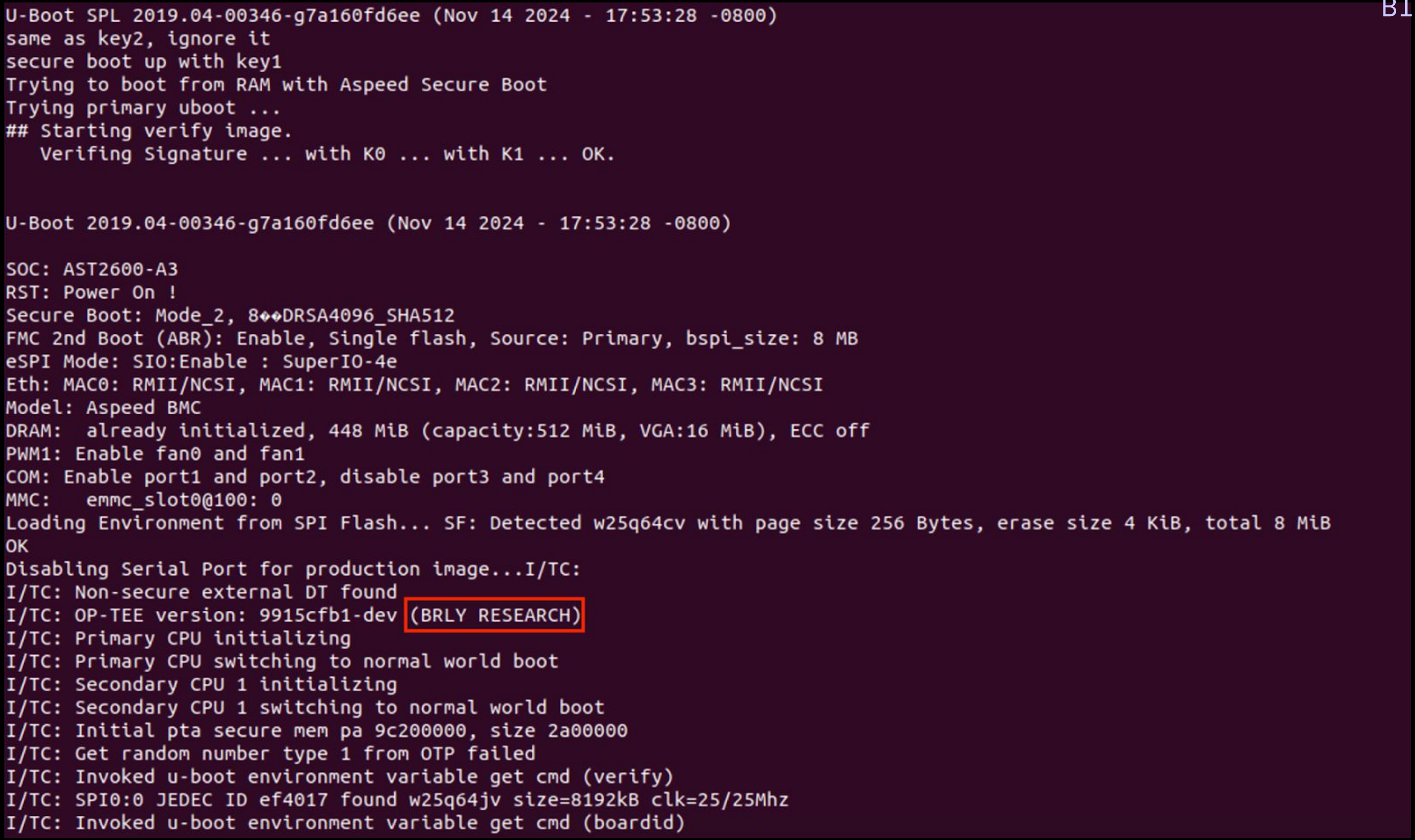
Custom *sig_table*



1. offset: `0x0000000`, size: `0x0100000` – `bootloader`
2. offset: `0x0101000`, size: `0x0001000` – `sig_table` (original)
3. offset: `0x0110000`, size: `0x0010000` – `pdb_seca`
4. offset: `0x0130000`, size: `0x0095600` – `kernel` (before custom content)
5. offset: `0x0102000`, size: `0x0000200` – `kernel` (original data that was replaced with custom content)
6. offset: `0x01c5800`, size: `0x0354600` – `kernel` (after custom content)
7. offset: `0x0630000`, size: `0x2000000` – `rootFS (1st part)`
8. offset: `0x2630000`, size: `0x064a080` – `rootFS (2nd part)`



CVE-2025-6198 – exploitation demo

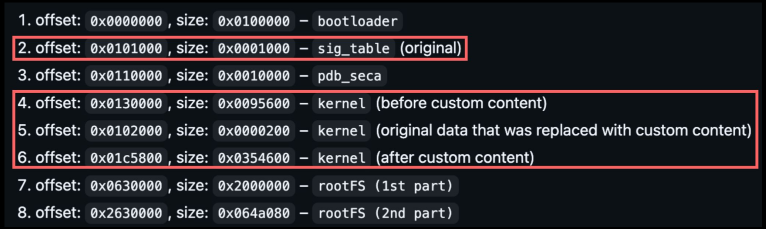


```
U-Boot SPL 2019.04-00346-g7a160fd6ee (Nov 14 2024 - 17:53:28 -0800)
same as key2, ignore it
secure boot up with key1
Trying to boot from RAM with Aspeed Secure Boot
Trying primary uboot ...
## Starting verify image.
   Verifing Signature ... with K0 ... with K1 ... OK.


U-Boot 2019.04-00346-g7a160fd6ee (Nov 14 2024 - 17:53:28 -0800)

SOC: AST2600-A3
RST: Power On !
Secure Boot: Mode_2, 8��DRSA4096_SHA512
FMC 2nd Boot (ABR): Enable, Single flash, Source: Primary, bspi_size: 8 MB
eSPI Mode: SIO:Enable : SuperIO-4e
Eth: MAC0: RMII/NCSI, MAC1: RMII/NCSI, MAC2: RMII/NCSI, MAC3: RMII/NCSI
Model: Aspeed BMC
DRAM:  already initialized, 448 MiB (capacity:512 MiB, VGA:16 MiB), ECC off
PWM1: Enable fan0 and fan1
COM: Enable port1 and port2, disable port3 and port4
MMC:   emmc_slot0@100: 0
Loading Environment from SPI Flash... SF: Detected w25q64cv with page size 256 Bytes, erase size 4 KiB, total 8 MiB
OK
Disabling Serial Port for production image...I/TC:
I/TC: Non-secure external DT found
I/TC: OP-TEE version: 9915cfb1-dev (BRLY RESEARCH)
I/TC: Primary CPU initializing
I/TC: Primary CPU switching to normal world boot
I/TC: Secondary CPU 1 initializing
I/TC: Secondary CPU 1 switching to normal world boot
I/TC: Initial pta secure mem pa 9c200000, size 2a00000
I/TC: Get random number type 1 from OTP failed
I/TC: Invoked u-boot environment variable get cmd (verify)
I/TC: SPI0:0 JEDEC ID ef4017 found w25q64jv size=8192kB clk=25/25Mhz
I/TC: Invoked u-boot environment variable get cmd (boardid)
```

CVE-2025-6198 – the fix

- Check that the offset of parsed *sig_table* is **0x100000**
- *sig_table* must contain a region where *offset <= sig_table offset < offset + size*

~~***Custom sig_table***~~

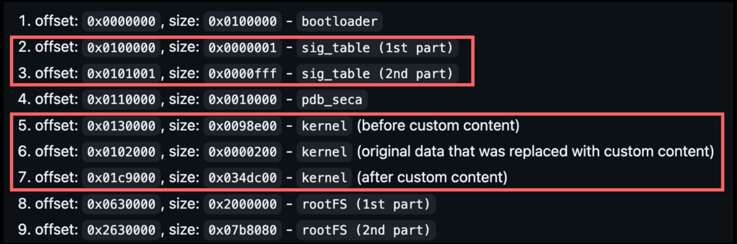


1. offset: `0x0000000`, size: `0x0100000` – `bootloader`
2. offset: `0x0101000`, size: `0x0001000` – `sig_table` (original)
3. offset: `0x0110000`, size: `0x0010000` – `pdb_seca`
4. offset: `0x0130000`, size: `0x0095600` – `kernel` (before custom content)
5. offset: `0x0102000`, size: `0x0000200` – `kernel` (original data that was replaced with custom content)
6. offset: `0x01c5800`, size: `0x0354600` – `kernel` (after custom content)
7. offset: `0x0630000`, size: `0x2000000` – `rootFS (1st part)`
8. offset: `0x2630000`, size: `0x064a080` – `rootFS (2nd part)`

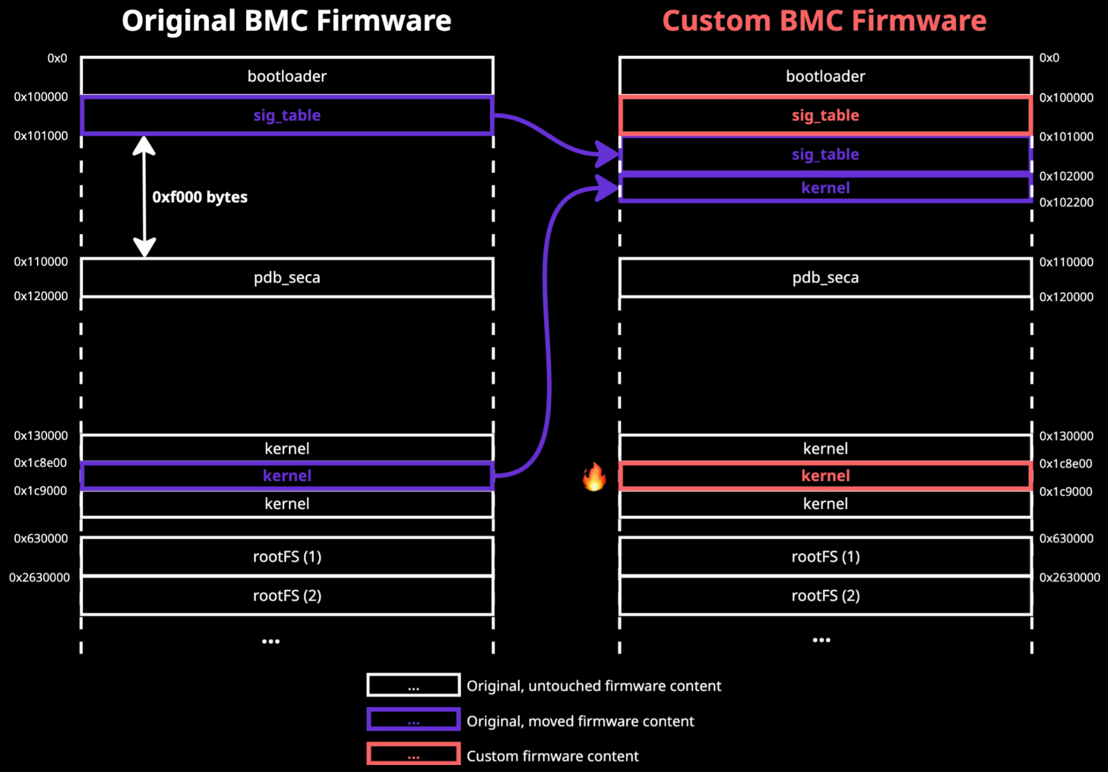
CVE-2025-6198 – fix bypass

BINARLY 2026

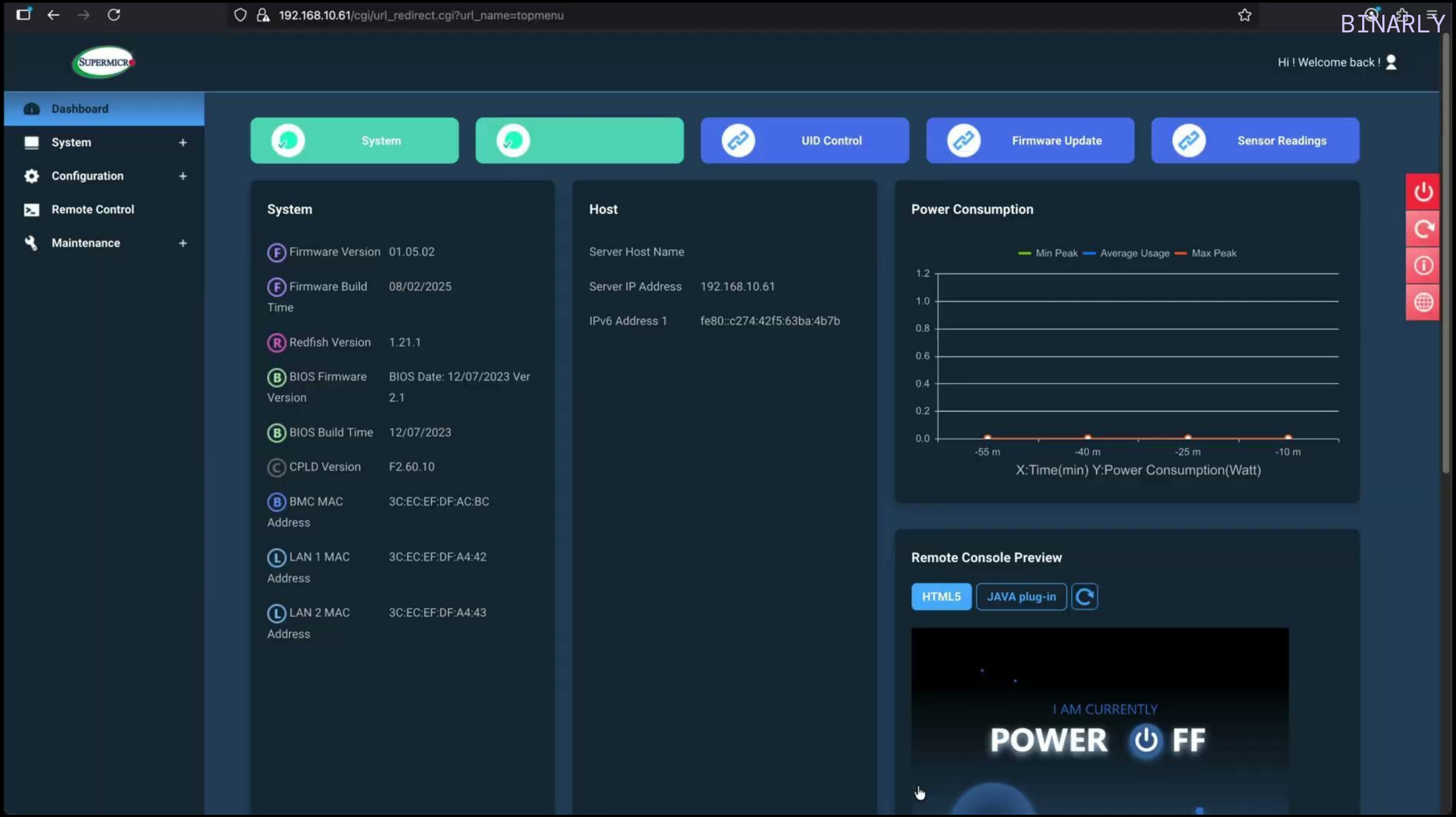
Custom *sig_table*



1. offset: `0x0000000`, size: `0x0100000` - `bootloader`
2. offset: `0x0100000`, size: `0x0000001` - `sig_table (1st part)`
3. offset: `0x0101001`, size: `0x0000fff` - `sig_table (2nd part)`
4. offset: `0x0110000`, size: `0x0010000` - `pdb_seca`
5. offset: `0x0130000`, size: `0x0098e00` - `kernel` (before custom content)
6. offset: `0x0102000`, size: `0x0000200` - `kernel` (original data that was replaced with custom content)
7. offset: `0x01c9000`, size: `0x034dc00` - `kernel` (after custom content)
8. offset: `0x0630000`, size: `0x2000000` - `rootFS (1st part)`
9. offset: `0x2630000`, size: `0x07b8080` - `rootFS (2nd part)`



CVE-2025-12007 – exploitation demo



192.168.10.61/cgi/url_redirect.cgi?url_name=topmenu

System

Firmware Version 01.05.02

Firmware Build Time 08/02/2025

Redfish Version 1.21.1

BIOS Firmware Version BIOS Date: 12/07/2023 Ver 2.1

BIOS Build Time 12/07/2023

CPLD Version F2.60.10

BMC MAC Address 3C:EC:EF:DF:AC:BC

LAN 1 MAC Address 3C:EC:EF:DF:A4:42

LAN 2 MAC Address 3C:EC:EF:DF:A4:43

Host

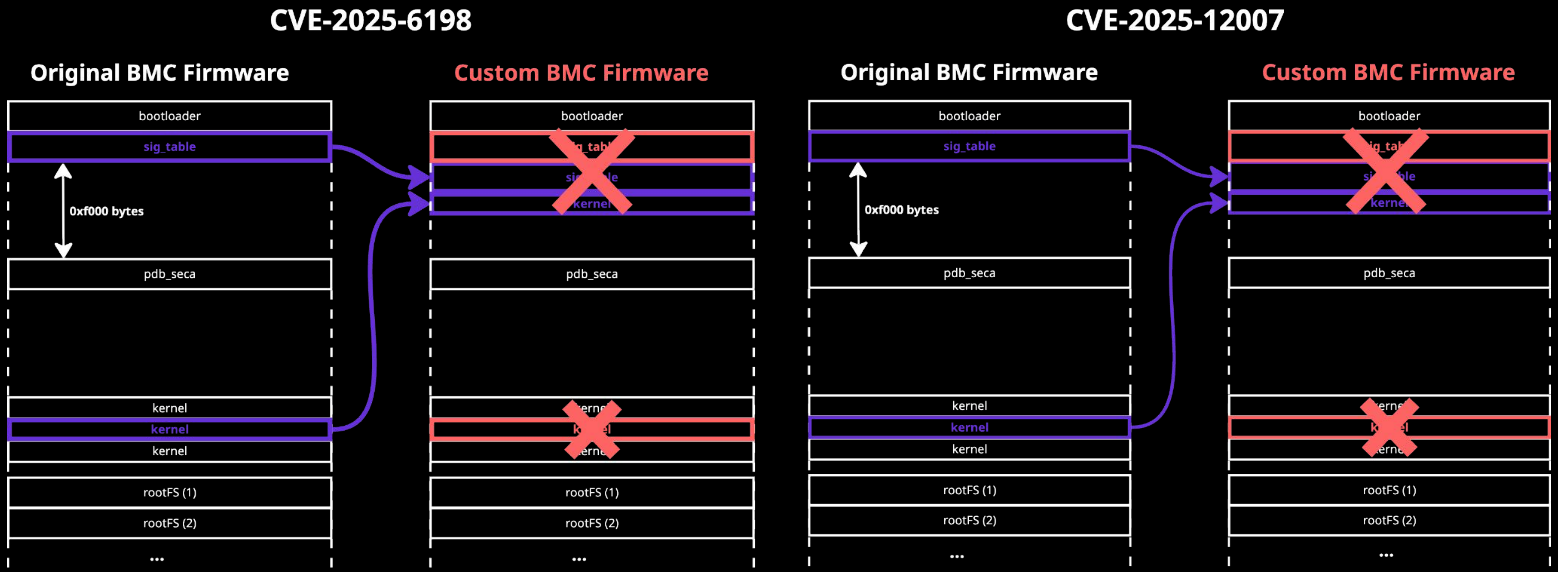
Server Host Name

Server IP Address 192.168.10.61

IPv6 Address 1 fe80::c274:42f5:63ba:4b7b

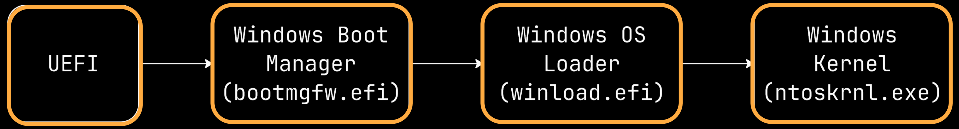
CVE-2025-12007 – the fix

- Only two allowed offsets for *sig_table* entries:
  - 0x0
  - 0x3FB0000 (location of region containing image cryptographic signature)

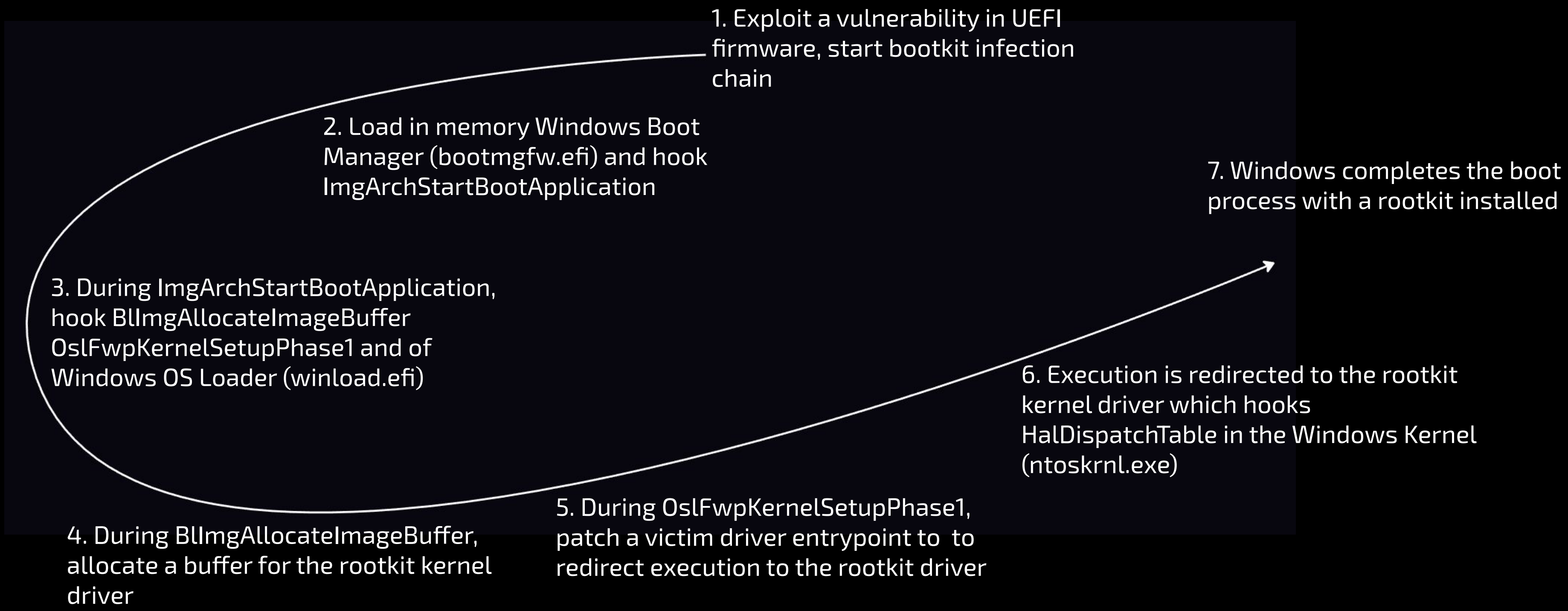


The Anatomy of a UEFI Bootkit: *redlotus-rs*

BINARLY 2026



UEFI → Windows Boot Manager (bootmgfw.efi) → Windows OS Loader (winload.efi) → Windows Kernel (ntoskrnl.exe)



1. Exploit a vulnerability in UEFI firmware, start bootkit infection chain

2. Load in memory Windows Boot Manager (bootmgfw.efi) and hook ImgArchStartBootApplication

3. During ImgArchStartBootApplication, hook BlImgAllocateImageBuffer OslFwpKernelSetupPhase1 and of Windows OS Loader (winload.efi)

4. During BlImgAllocateImageBuffer, allocate a buffer for the rootkit kernel driver

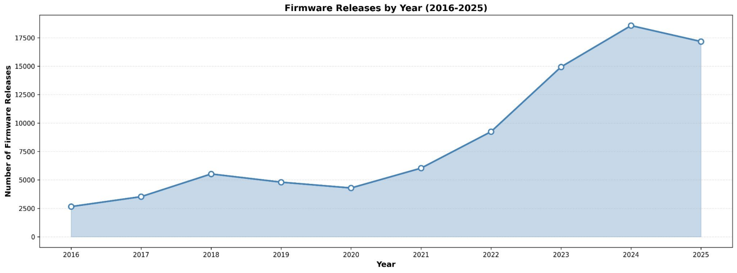
5. During OslFwpKernelSetupPhase1, patch a victim driver entrypoint to to redirect execution to the rootkit driver

6. Execution is redirected to the rootkit kernel driver which hooks HalDispatchTable in the Windows Kernel (ntoskrnl.exe)

7. Windows completes the boot process with a rootkit installed

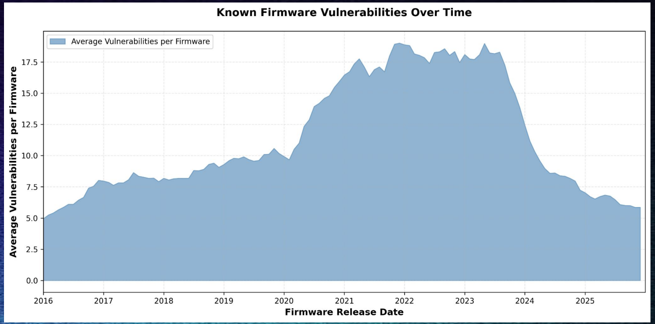
OG Plots

Slide 24



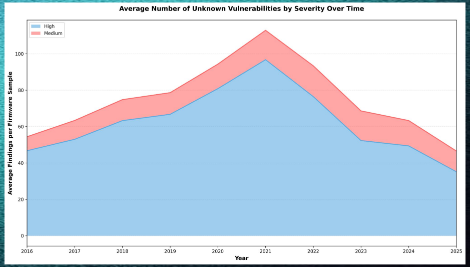
Firmware Releases by Year (2016-2025)

Slide 25



Known Firmware Vulnerabilities Over Time

Slide 26



Average Number of Unknown Vulnerabilities by Severity Over Time

BONUS !

Images from slide 4