

Jailbreaking an Electric Vehicle in 2023

WHAT IT MEANS TO HOTWIRE TESLA'S X86-BASED SEAT HEATER

Christian Werling

Niclas Kühnapfel

Hans Niklas Jacob

Oleg Drokin

TU Berlin

Independent



Tesla's Infotainment Now AMD-Powered

TESMANIAN

Model S Model 3 Model X ▾ Model Y ▾ Mach-E Gift Card SALE News

Europe

Tesla to Soon Start Delivering Model 3 & Y with AMD Ryzen Chips to Europe, Parts Catalog Hints

Dec 6, 2021 By Eva Fox 3 Comments



2

Our Previous AMD Research



[CS.CR] 2 Sep 2019

Insecure Until Proven Updated: Analyzing AMD SEV's Remote Attestation

Robert Buhren
robert.buhren@sect.tu-berlin.de
Technische Universität Berlin
Security in Telecommunications

Christian Werling
christian.werling@student.hpi.de
Hasso Plattner Institute, Potsdam
Security in Telecommunications

Jean-Pierre Seifert
jpseifert@sect.tu-berlin.de
Technische Universität Berlin
Security in Telecommunications

ABSTRACT

Cloud computing is one of the most prominent technologies to host Internet services that unfortunately leads to an increased risk of data theft. Customers of cloud services have to trust the cloud providers, as they control the building blocks that form the cloud. This includes the hypervisor enabling the sharing of a single hardware platform among multiple tenants. Executing in a higher-privileged CPU mode, the hypervisor has direct access to the memory of virtual machines. While data at rest can be protected using well-known disk encryption methods, data residing in main memory is still threatened by a potentially malicious cloud provider.

AMD Secure Encrypted Virtualization (SEV) claims a new level of protection in such cloud scenarios. AMD SEV encrypts the main memory of virtual machines with VM-specific keys, thereby denying the higher-privileged hypervisor access to a guest's memory.

1 INTRODUCTION

Cloud computing is one of the core foundations of today's Internet landscape. The manifold advantages such as on-demand resource allocation or high availability of services have led to a wide usage of this technology. However, outsourcing the processing of enterprise data comes at a risk. The technical infrastructure that forms the cloud is owned by the cloud provider and thus under his full control. This includes the server hardware, as well as the software components that allow the co-location of multiple virtual machines on a single host.

Therefore security concerns impede the deployment of confidential data and applications in cloud scenarios [14, 19]. The potential threats range from misconfiguration of software components over cloud provider admin access to foreign government access [8].

To counter these threats, the research community, as well as industry, proposed new approaches to allow secure cloud computing

faulTPM: Exposing AMD fTPMs' Deepest Secrets

Hans Niklas Jacob*, Christian Werling*, Robert Buhren, Jean-Pierre Seifert†
Technische Universität Berlin – SecT
†also: Fraunhofer SIT
{ hnj, cwerling, robert.buhren, jpseifert }@sect.tu-berlin.de

One Glitch to Rule Them All: Fault Injection Attacks Against AMD's Secure Encrypted Virtualization

Robert Buhren
robert.buhren@sect.tu-berlin.de
Technische Universität Berlin - SECT

Thilo Krachenfels
tkrachenfels@sect.tu-berlin.de
Technische Universität Berlin - SECT

Hans Niklas Jacob
hnj@sect.tu-berlin.de
Technische Universität Berlin - SECT
Jean-Pierre Seifert
jpseifert@sect.tu-berlin.de
Technische Universität Berlin - SECT
Fraunhofer SIT

21

EM-Fault It Yourself: Building a Replicable EMFI Setup for Desktop and Server Hardware

Niclas Kühnapfel*, Robert Buhren*, Hans Niklas Jacob*, Thilo Krachenfels*,
Christian Werling*, Jean-Pierre Seifert*†
* Technische Universität Berlin, Chair of Security in Telecommunications, Germany
† Fraunhofer SIT, Germany

a cloud data
in a hosted
confidentiality of
memory with a
ried out by a dedi-
the memory con-
IV, SEV Encrypted
SEV-SNP), expand
introduce softwar-
ship tracking [3, 29],
note attestation fea-
ct instantiation of
y encryption keys
ature, AMD CPUs
AMD Secure Pro-
te root-of-trust for
ated VM life-cycle
The AMD-SP uses

conditions of
ality. Altering
lucing energy
ay destabilize
s occur. As a
glitching was
n [3]-[5] and
s and CPUs.
ce under test
(EMFI) are
the DUT by
ckly chang-
e, both tech-
to the power
sive attacks.
decapsulated
non-shielding
FI and EMFI
ns and change

blems: Firstly,
age glitching,
results more

Why Jailbreak a Car?

Many reasons:

- to "look around" (curiosity)
- to replace its software
- **to activate soft-locked features**



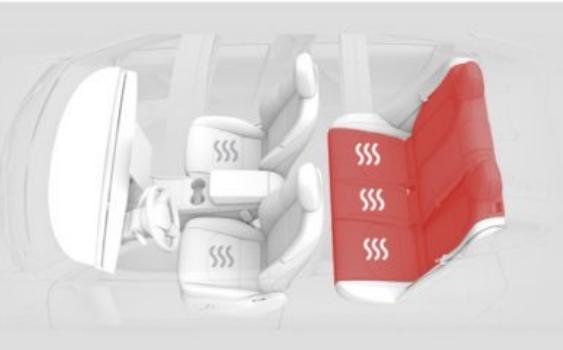
8:48 ⓘ

tesla.com

TESLA

◀ Back to Vehicle Profile

Upgrades



Rear Heated Seats

Enhance your passengers riding experience with rear heated seats

Buy for \$300

Outline

1

Analyzing Boot and Firmware Security

2

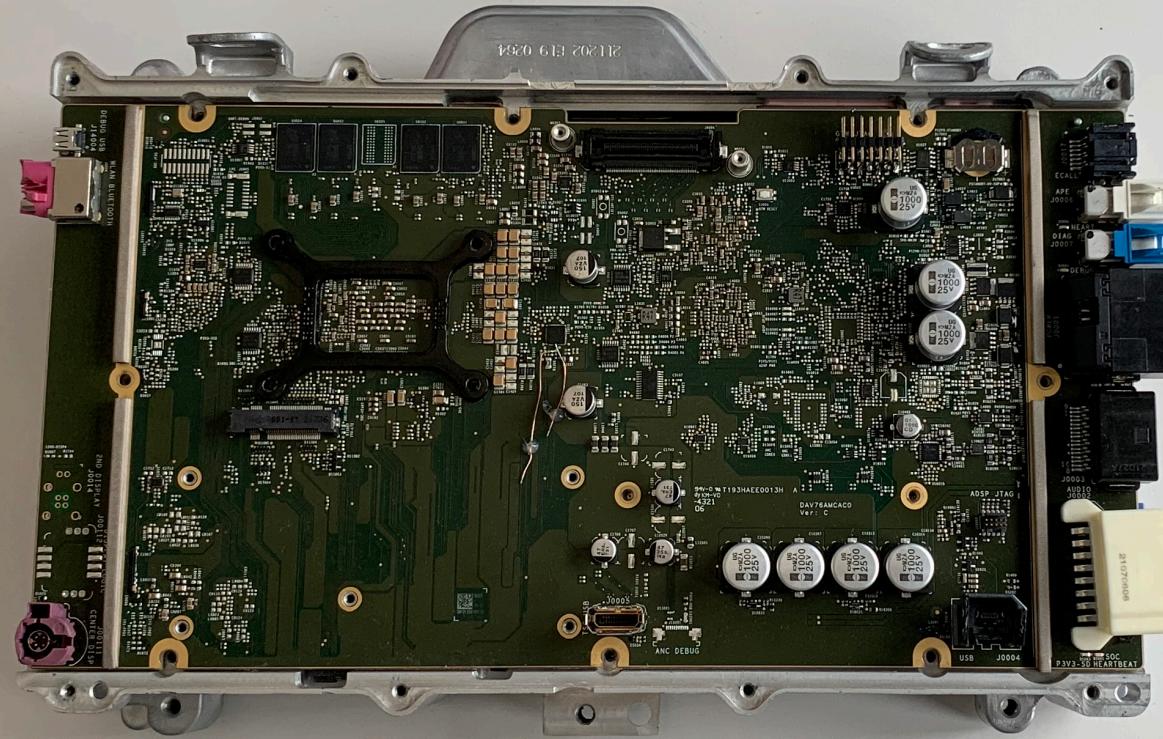
Hotwiring the Infotainment system

3

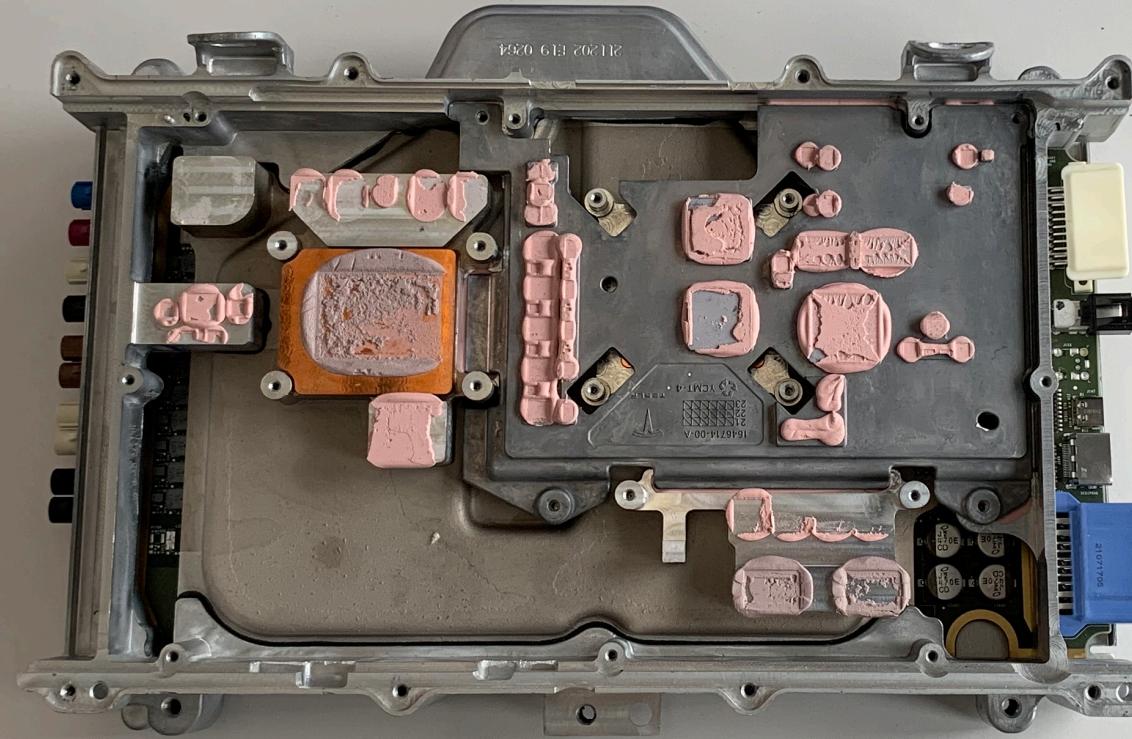
Extracting Secrets from the Tesla



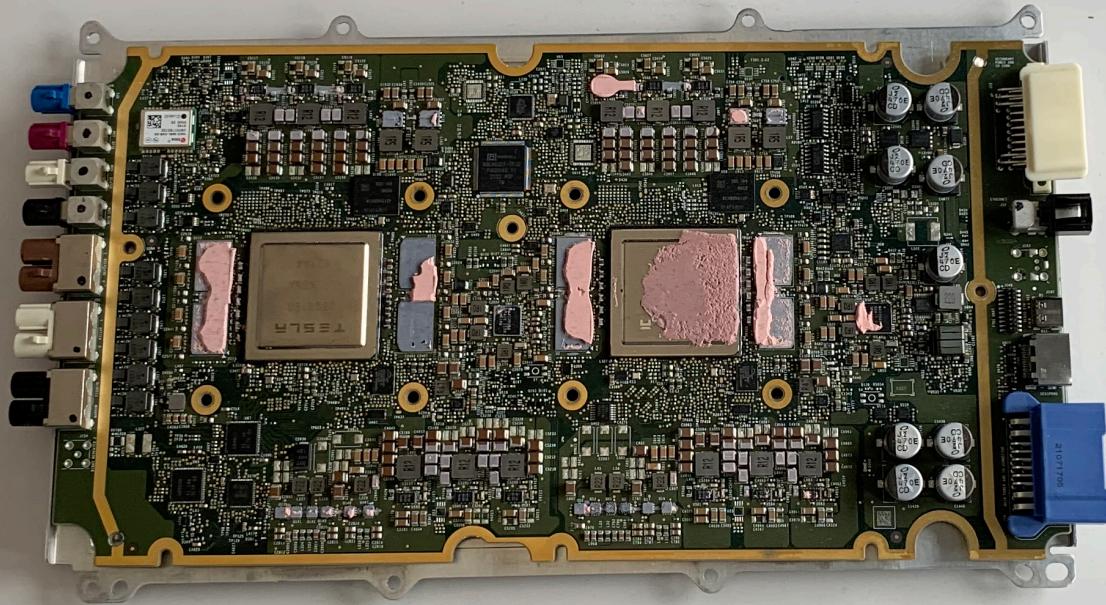
Model 3 Car Computer



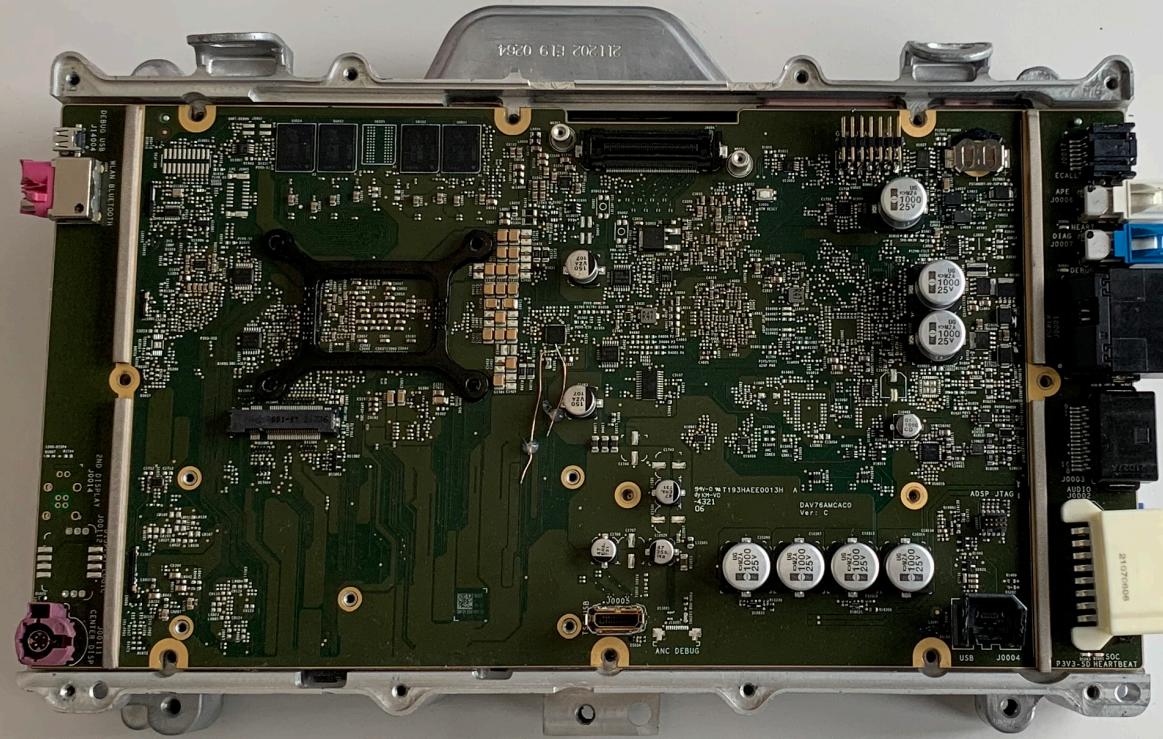
Infotainment and Connectivity ECU (ICE)



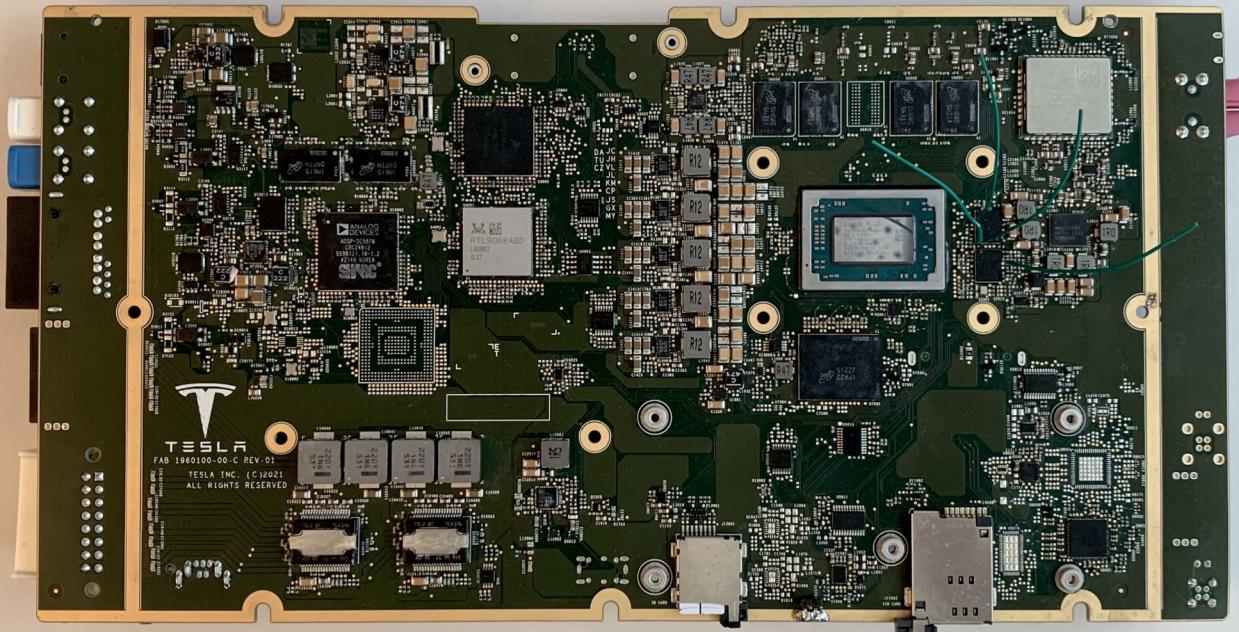
Cooling chassis



Autopilot v3

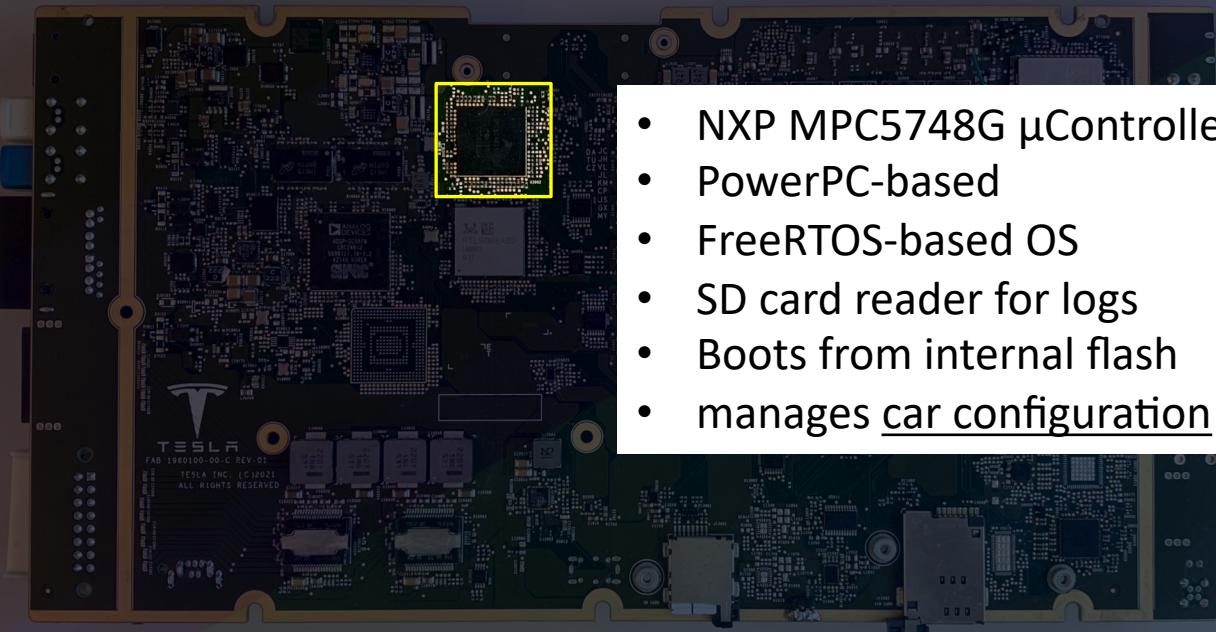


Infotainment and Connectivity ECU (ICE)



ICE (Backside)

Gateway



- NXP MPC5748G µController
- PowerPC-based
- FreeRTOS-based OS
- SD card reader for logs
- Boots from internal flash
- manages car configuration

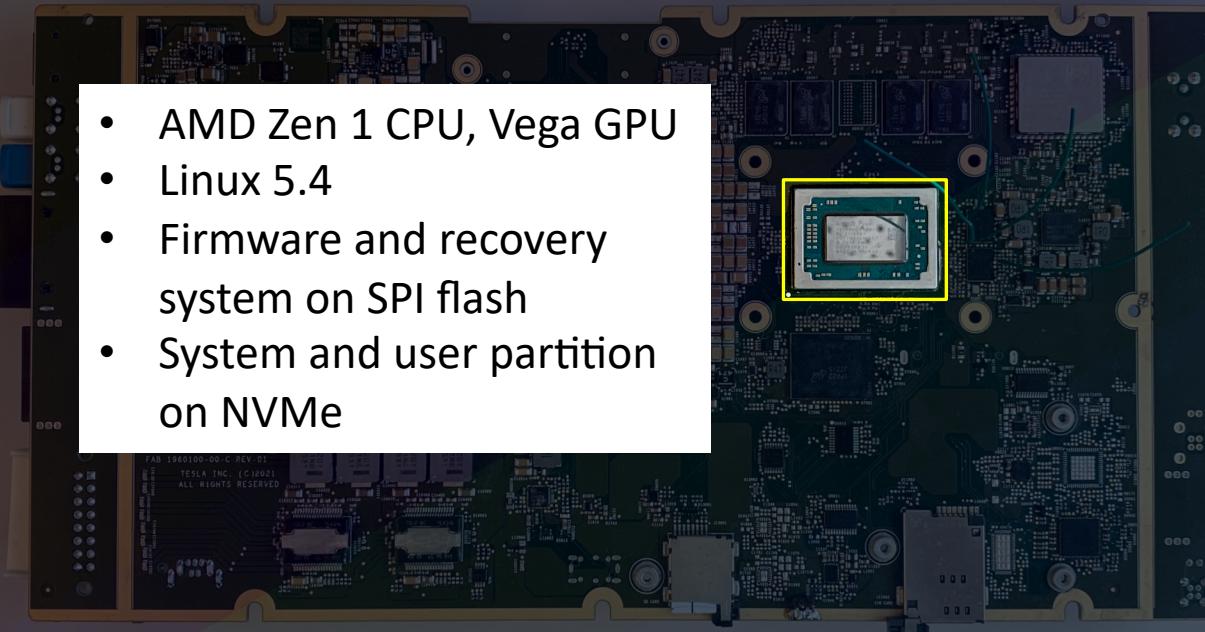
ICE (Backside)

Car configuration

- Stored and managed by the Gateway
- Lists (paid) hardware and software features
 - Car performance
 - Battery capacity (for software-locked batteries)
 - Level of Autopilot: (Enhanced) Autopilot, Full Self-Driving capability
 - Car region
 - Rear seat heaters

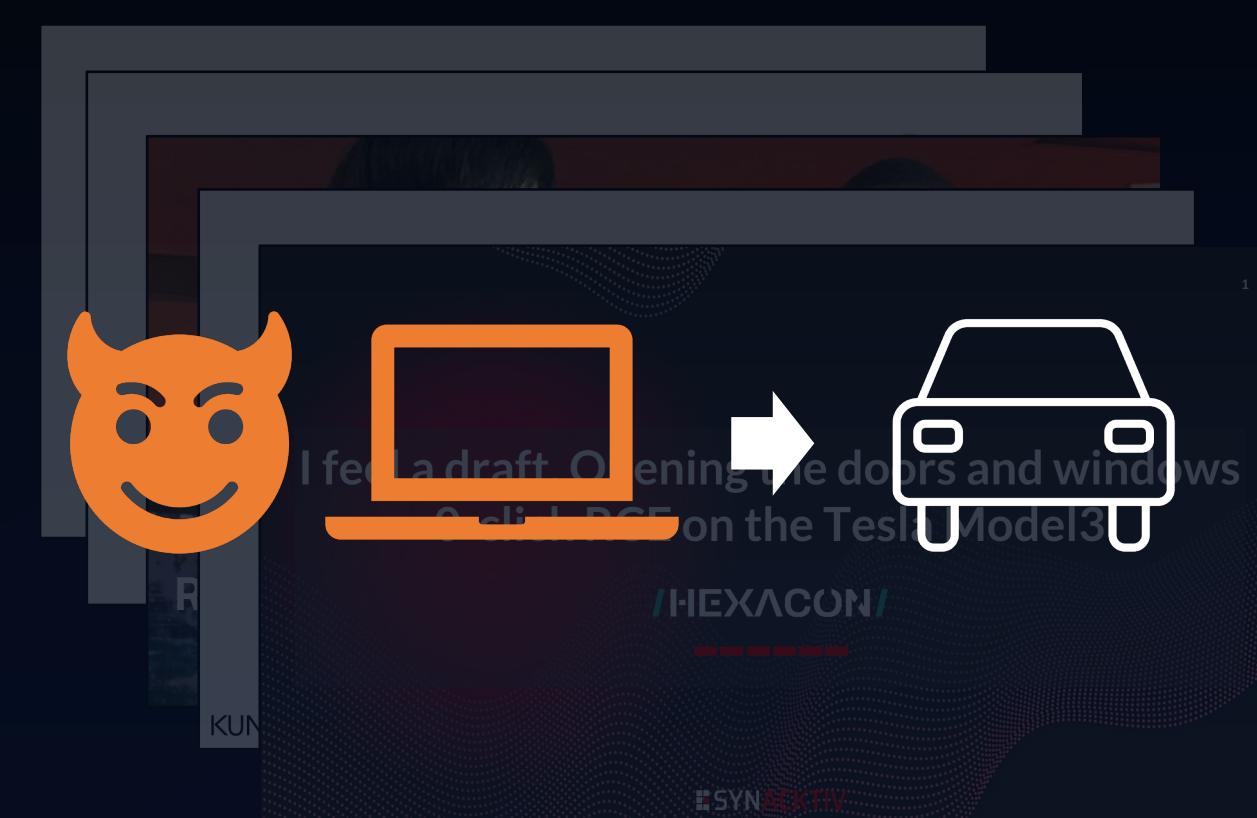
Infotainment APU

- AMD Zen 1 CPU, Vega GPU
- Linux 5.4
- Firmware and recovery system on SPI flash
- System and user partition on NVMe



ICE (Backside)

Previous Tesla Hacking



- Threat model: *Outsider* who is remote or in physical proximity
- Goal: Access/control car
- Software-based vulnerabilities: Can be fixed by Tesla over-the-air

Platform Threats from the *Inside*

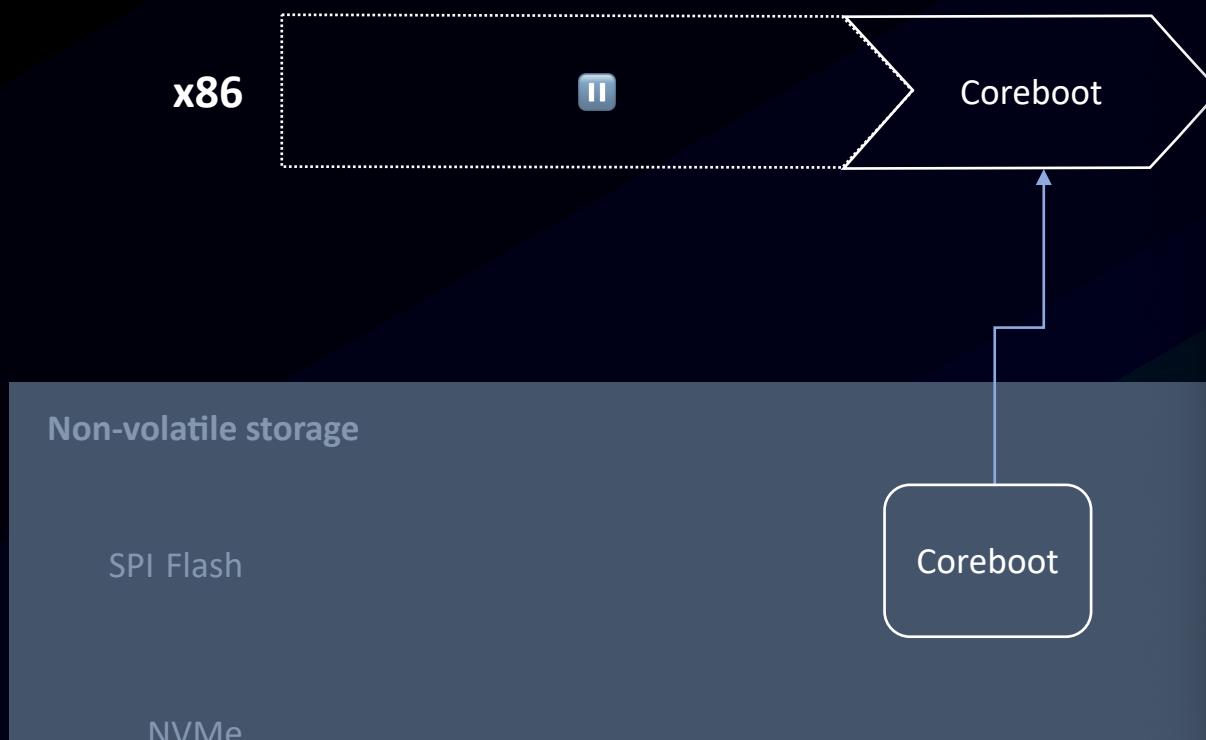
- Threat model: Insider who already has **digital and physical access to the car**
- Goal: Tweak car beyond normal flows
 - activate **soft-locked features** without paying
 - lift repair and regulation restrictions
- Insider not limited to software-based attacks



Verified Boot



Verified Boot

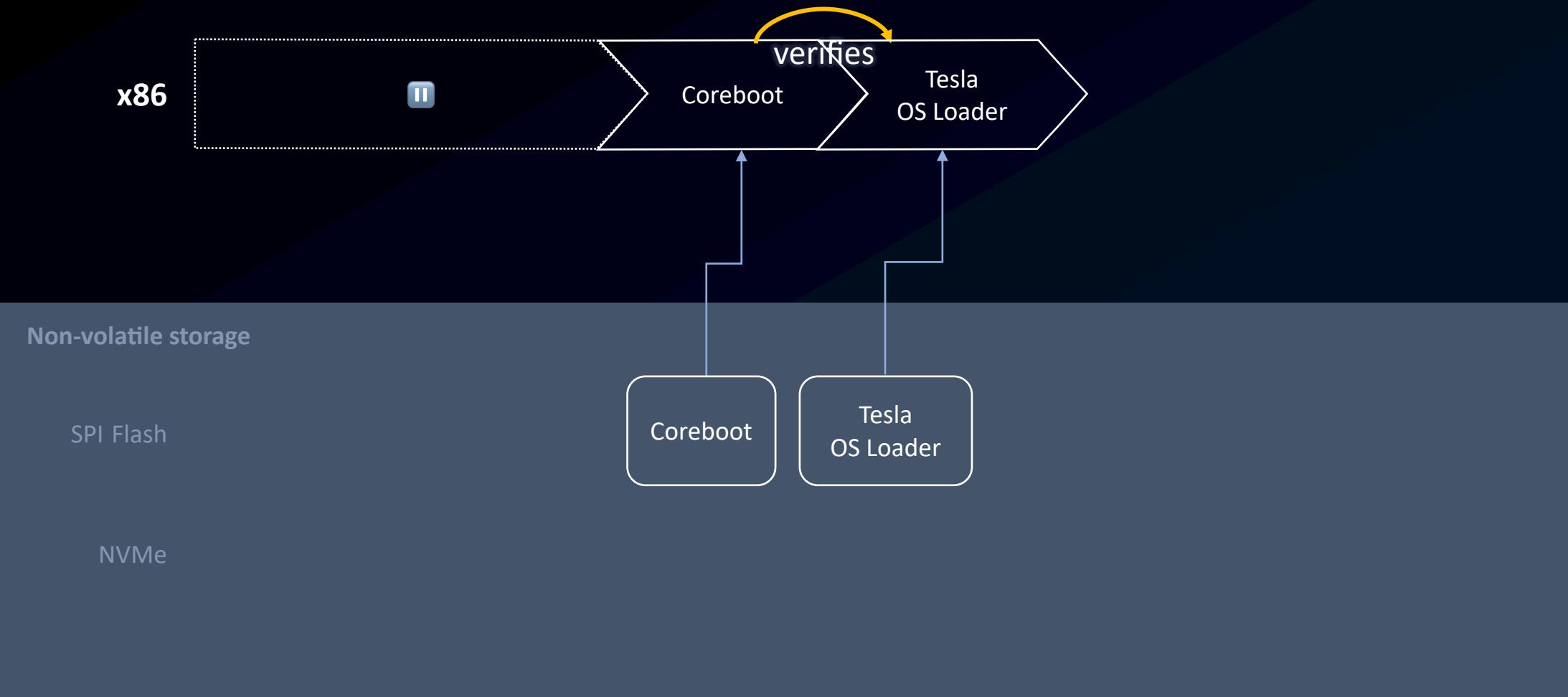


```
cwerling@MacBookProPro:~/Git/tesla-hacking/logs
coreboot-archive/develop/2021.44.25.2-8836-gb025c688348a
Thu Jan 13 14:46:27 UTC 2022 bootblock starting (log level
l: 5)...
PMxC0 STATUS: 0x800 BIT11

coreboot-archive/develop/2021.44.25.2-8836-gb025c688348a
Thu Jan 13 14:46:27 UTC 2022 romstage starting (log level
: 5)...
POST: 0x41
POST: 0x42
POST: 0x43
POST: 0x34
POST: 0x36
POST: 0x92
POST: 0x98
SF size 0x2000000 does not correspond to CONFIG_ROM_SIZE
0x1000000!!
POST: 0x44

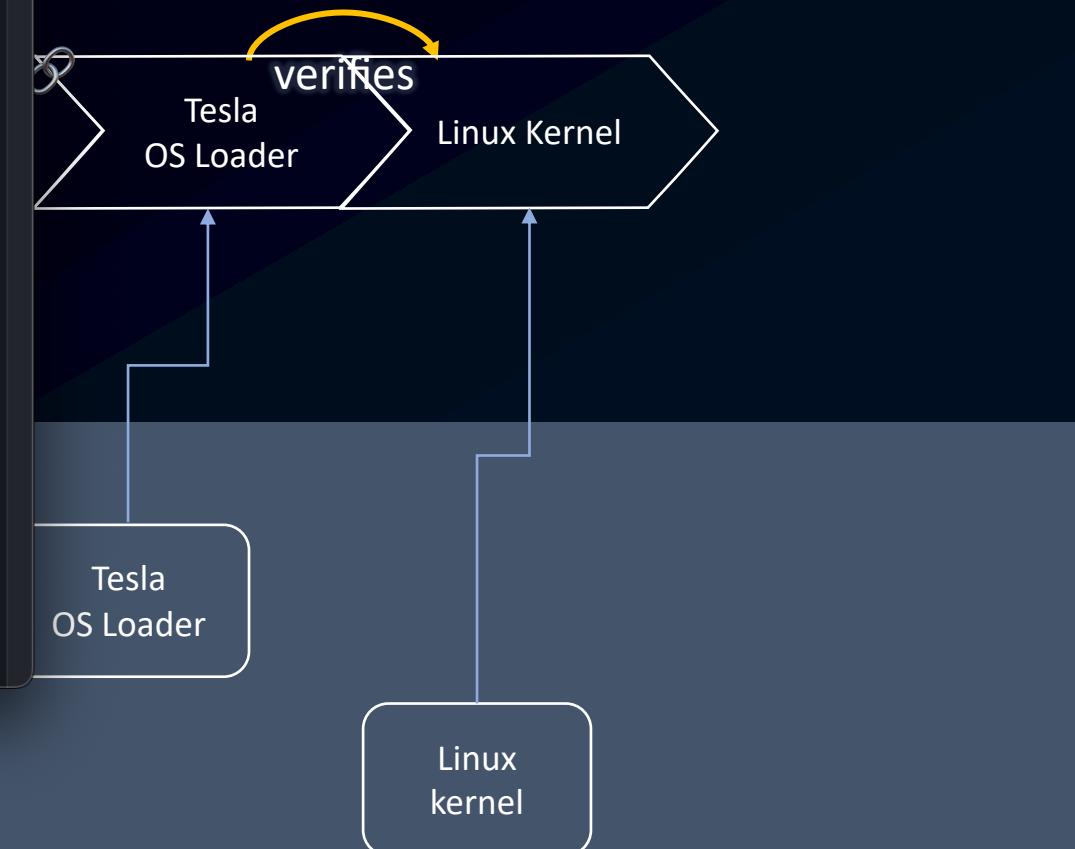
coreboot-archive/develop/2021.44.25.2-8836-gb025c688348a
Thu Jan 13 14:46:27 UTC 2022 ramstage starting (log level
: 5)...
POST: 0x39
POST: 0x80
POST: 0x70
POST: 0x71
Board name: Spinach
```

Verified Boot

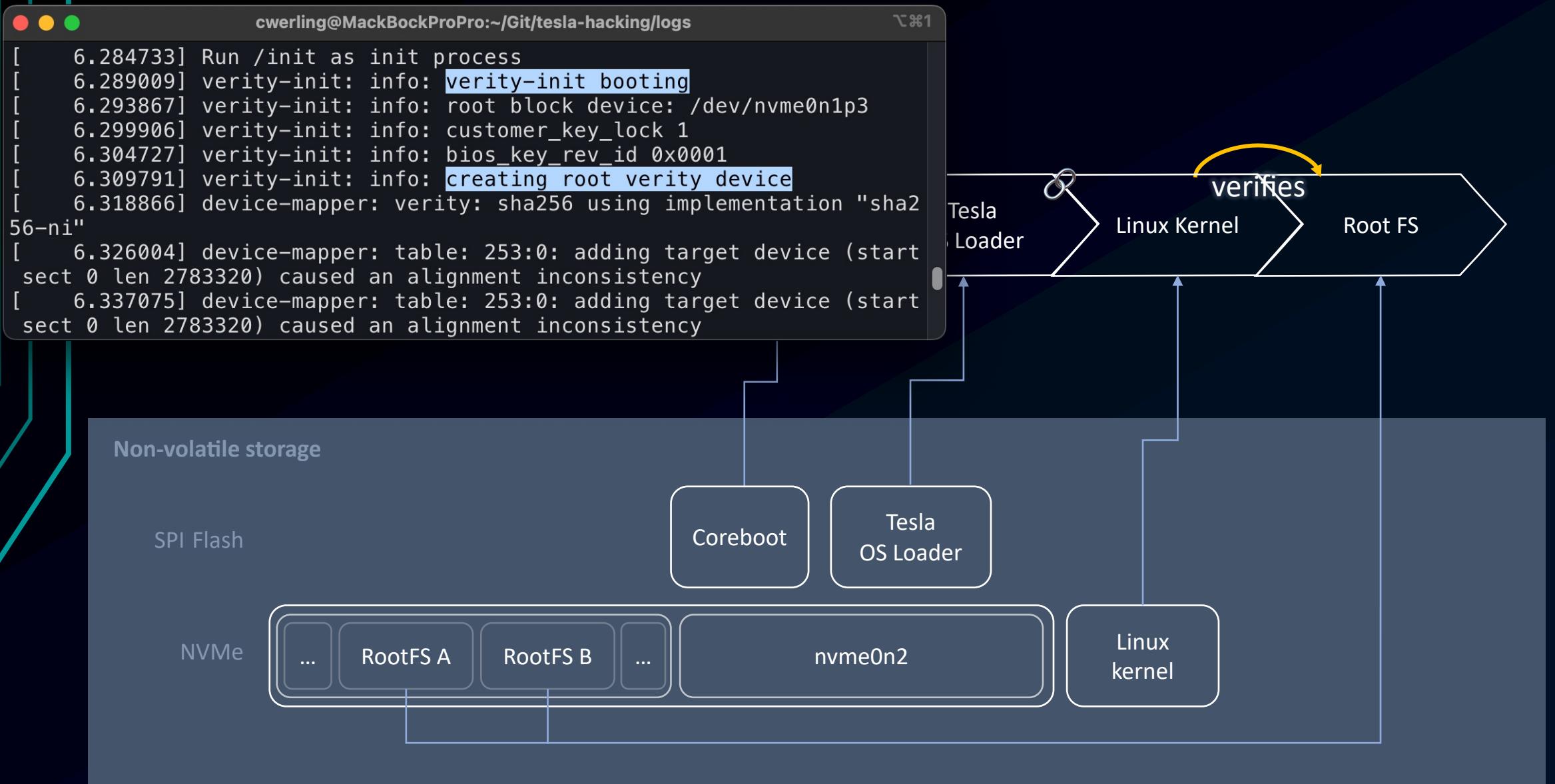


```
cwerling@MacBookProPro:~/Git/tesla-hacking/logs
```

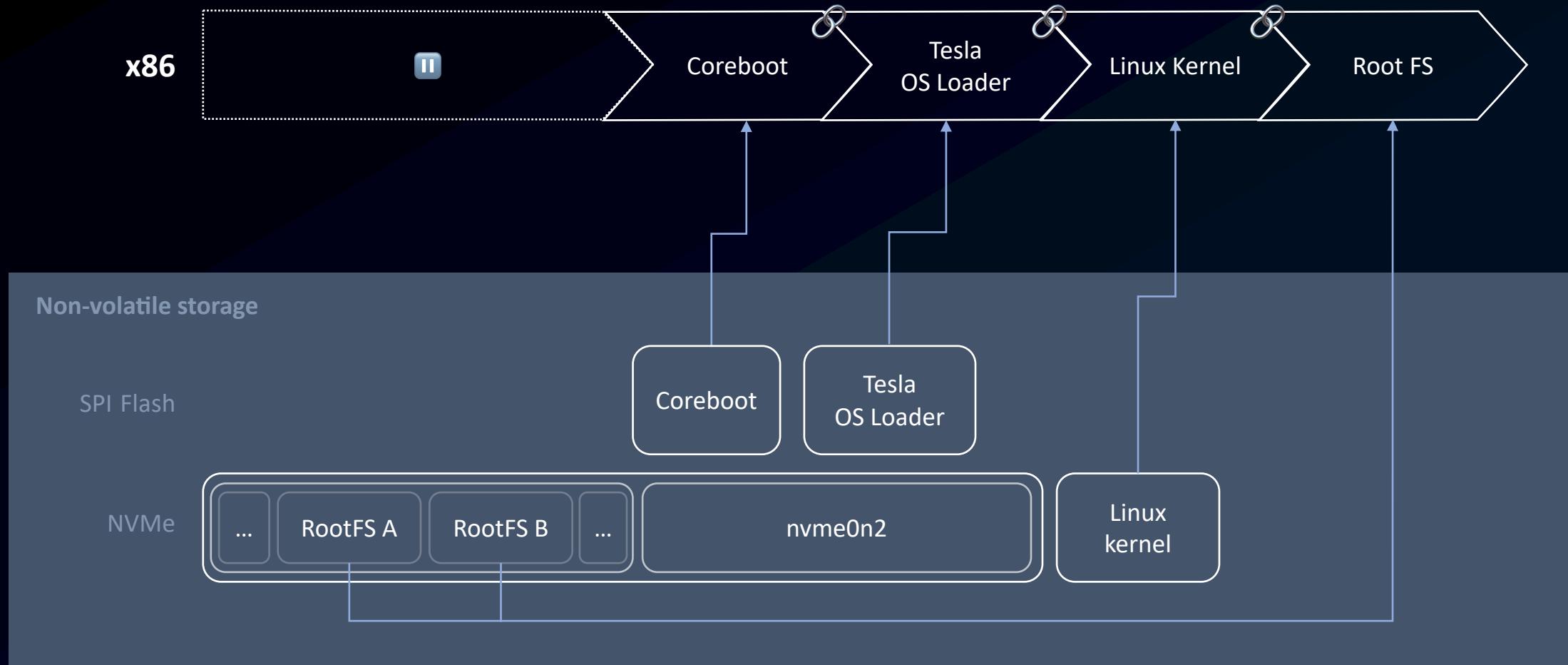
```
[tesla-os-loader] v7
[tesla-os-loader] starting
[tesla-os-loader] Booted from SPI Bank B
[tesla-os-loader] Found NVMe
[tesla-os-loader] BAR address: 0xD0200000
[tesla-os-loader] Attempting to boot online image from the NVMe
[tesla-os-loader] initialized NVMe controller access
[tesla-os-loader] NVMe with boot partition support detected OK
[tesla-os-loader] Reading header from NVMe boot partition ID: 0x1
[tesla-os-loader] Reading container header
[tesla-os-loader] Boot partition header read complete
[tesla-os-loader] Verifying header...
vb2_secdat_kernel_set: secdat_kernel flags updated from 0x0 to 0x6
vb2_verify_keyblock: Checking keyblock signature...
vb2_verify_kernel_preamble: Verifying kernel preamble.
[tesla-os-loader] Successfully verified boot partition header!
[tesla-os-loader] container_type: Vboot bzimage
[tesla-os-loader] image_version: Version 1
[tesla-os-loader] header_crc32: 0x43524254
[tesla-os-loader] Reading boot payload
[tesla-os-loader] Verifying nvme image...
[tesla-os-loader] Successfully verified image!
[tesla-os-loader] Boot payload read complete
[tesla-os-loader] Found bzImage at address: 0x126000
[tesla-os-loader] bzImage size: 0x65c000
[tesla-os-loader] Linux bzImage is valid!
[tesla-os-loader] Loading and jumping into kernel
```



NVMe



Verified Boot



How to get a root shell

Many options:

- Spawn serial shell on boot
- Add SSH key to authorized_keys file
- Add known SSH password

They all require **changes** to the Root file system

Non-volatile storage

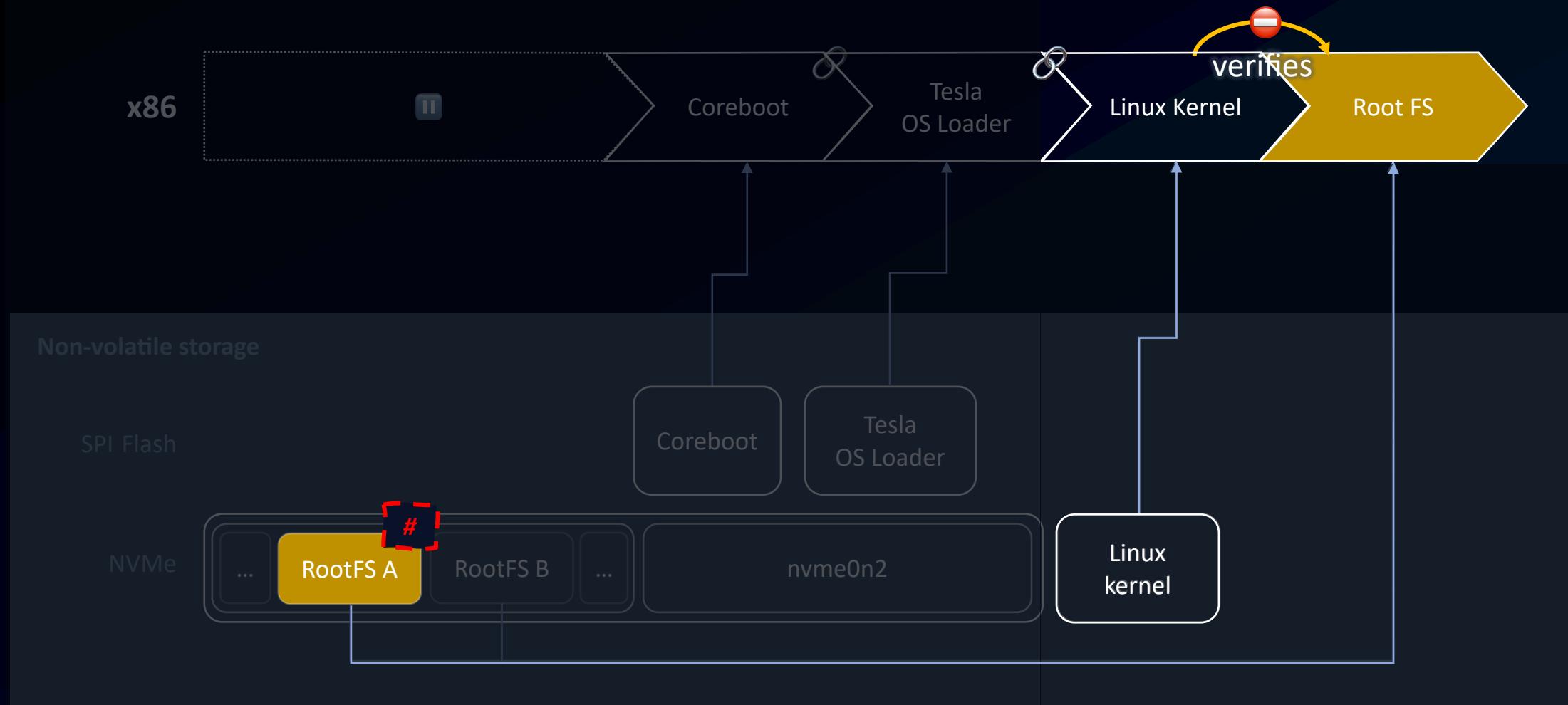
SPI Flash

NVMe



Verified Boot

loaded
rejected



dm-verity

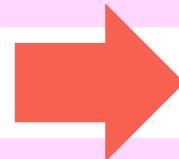
- Integrity checking of block devices
 - When a block is read into memory, it's hashed in parallel
- Merkle tree used to efficiently store and verify hashes of individual block
 - Trusted root file system represented by **root hash**
 - **Intermediate hashes** stored alongside data



dm-verity

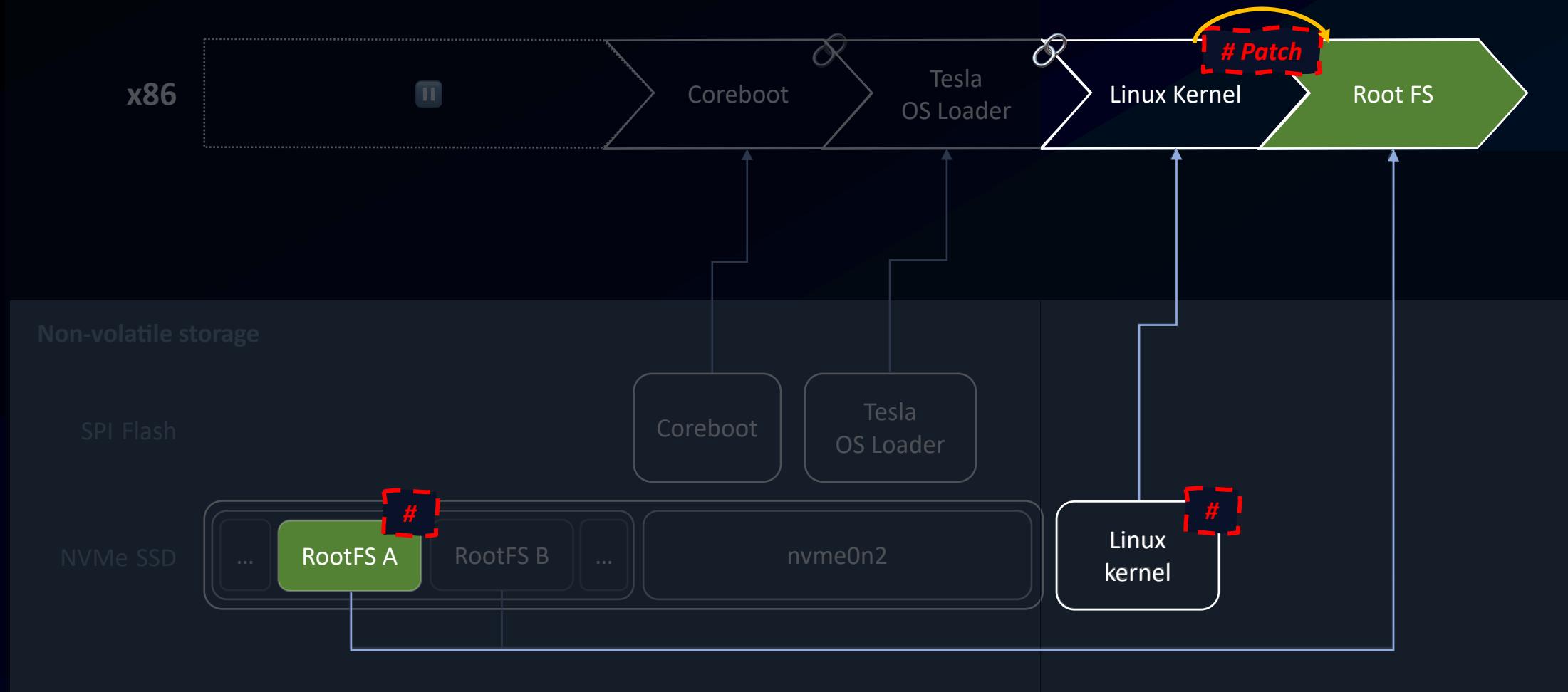
Patch

+ +-390 lines: 00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000 .ELF.....	c46 0201 0100 0000 0000 0000 0000 0000 .ELF.....
00001860: 5dc3 5548 89e5 5348 8d1d da13 2000 4883].UH..SH.... .H.	8d1d da13 2000 4883].UH..SH.... .H.
00001870: ec08 4883 eb08 488b 0348 83f8 ff74 04ff ..H...H..H...t..	0348 83f8 ff74 04ff ..H...H..H...t..
00001880: d0eb ef58 5b5d c350 e86a f6ff ff58 c325 ...X[.]P.j...X.%	e86a f6ff ff58 c325 ...X[.]P.j...X.%
00001890: 7520 2575 2025 7520 256c 7520 256c 7520 u %u %u %lu %lu	256c 7520 256c 7520 u %u %u %lu %lu
000018a0: 2531 3673 2025 3132 3873 2025 3132 3873 %16s %128s %128s	3873 2025 3132 3873 %16s %128s %128s
000018b0: 2025 7500 556e 6b6e 6f77 6e20 6572 726f %u.Unknown erro	6f77 6e20 6572 726f %u.Unknown erro
000018c0: 7200 3200 342e 3100 7265 7374 6172 745f r.2.4.1.restart_	6967 6e6f 7265 5f63 r.2.4.1.ignore_c
000018d0: 6f6e 5f63 6f72 7275 7074 696f 6e20 0075 on_corruption .u	6e20 2020 2020 0075 orruption .u
000018e0: 7365 5f66 6563 5f66 726f 6d5f 6465 7669 se_fec_from_devi	726f 6d5f 6465 7669 se_fec_from_devi
000018f0: 6365 2025 7320 6665 635f 726f 6f74 7320 ce %s fec_roots	635f 726f 6f74 7320 ce %s fec_roots
00001900: 2575 2066 6563 5f62 6c6f 636b 7320 256c %u fec_blocks %l	6c6f 636b 7320 256c %u fec_blocks %l
00001910: 7520 6665 635f 7374 6172 7420 2531 246c u fec_start %1\$1	6172 7420 256c 7520 u fec_start %lu
00001920: 7520 6c69 6e65 6172 2025 3324 7320 3020 u linear %3\$s 0	6572 6974 7920 2575 .0 %lu verity %u
00001930: 2325 3131 2473 2000 7520 2575 2025 6c75 #%11\$s .u %u %lu	7520 2575 2025 6c75 %s %s %u %u %lu
00001940: 2025 6c75 2025 7320 2573 2025 7300 2025 %lu %s %s %s. %	2573 2025 7300 2025 %lu %s %s %s. %
00001950: 7a75 2025 7300 2f75 7372 2f73 6269 6e2f zu %s./usr/sbin/	7372 2f73 6269 6e2f zu %s./usr/sbin/
00001960: 646d 7365 7475 7000 6372 6561 7465 002d dmsetup.create.-	6372 6561 7465 002d dmsetup.create.-
00001970: 7200 2d2d 7461 626c 6500 7265 6d6f 7665 r.--table.remove	6500 7265 6d6f 7665 r.--table.remove
00001980: 002d 2d66 6f72 6365 002d 2d72 6574 7279 .--force.--retry	002d 2d72 6574 7279 .--force.--retry
00001990: 002d 2d64 6566 6572 7265 6400 4553 5550 .--deferred.ESUP	7265 6400 4553 5550 .--deferred.ESUP
+ +-453 lines: 000019a0: 4552 4241 4400 496e 7661 6c69 6420 7375 ERBAD.Invalid su---	241 4400 496e 7661 6c69 6420 7375 ERBAD.Invalid su---



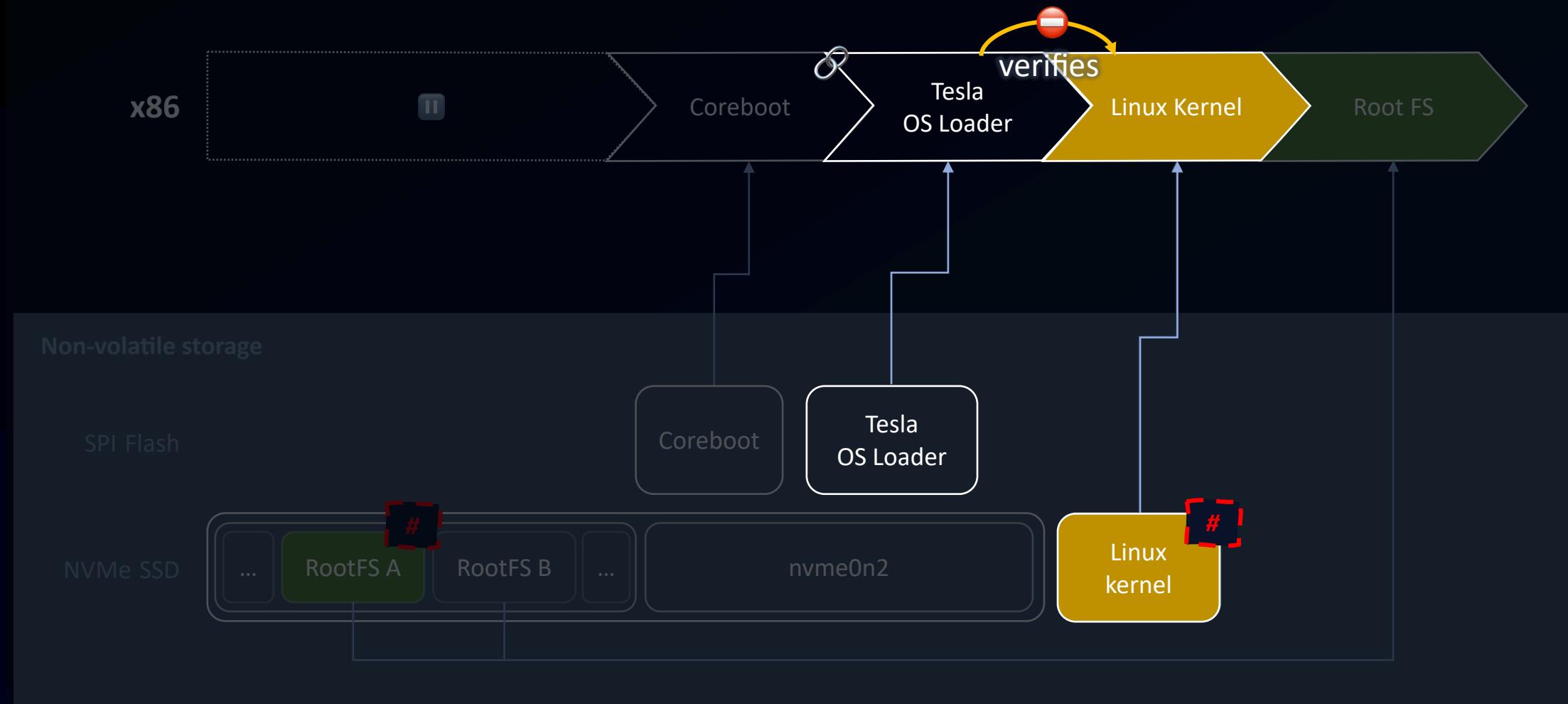
Verified Boot

loaded
rejected



Verified Boot

loaded
rejected



Tesla OS Loader

Listing: tesla-os-loader.bin

```

00101b11 83 ec 08      SUB    ESP,0x8
00101b14 68 61 51      PUSH   s_Verifying_nvme_image..._0011516
00101b19 68 51 4c      PUSH   s_[tesla-os-loader]_%s_00114c51
00101b1e e8 aa dc      CALL   puts
00101b23 83 c4 10      ADD    ESP,0x10
00101b26 8b 45 d8      MOV    EAX,dword ptr [EBP + local_2c]
00101b29 2b 45 ec      SUB    EAX,dword ptr [EBP + local_18]
00101b2c 8b 4d f4      MOV    ECX,dword ptr [EBP + local_10]
00101b2f 8b 55 ec      MOV    EDX,dword ptr [EBP + local_18]
00101b32 01 ca          ADD    EDX,ECX
00101b34 83 ec 08      SUB    ESP,0x8
00101b37 50             PUSH   EAX
00101b38 52             PUSH   EDX
00101b39 e8 58 f0      CALL   FUN_00100b96
00101b3e ff ff          ADD    ESP,0x10
00101b41 89 45 e0      MOV    dword ptr [EBP + local_24],EAX
00101b44 83 7d e0 00      CMP   dword ptr [EBP + local_24],0x0
00101b48 74 3e          JZ    LAB_00101b88
00101b4a 83 ec 0c      SUB    ESP,0xc
00101b4d 68 28 4f      PUSH   s_[tesla-os-loader]_Invalid_boot_
00101b52 e8 76 dc      CALL   puts
00101b57 83 c4 10      ADD    ESP,0x10
00101b5a 83 ec 0c      SUB    ESP,0xc
00101b5d ff 75 e0      PUSH   dword ptr [EBP + local_24]
00101b60 00 00 00          CALL   FUN_00100b96

```

Decompile: FUN_00101838 - (tesla-os-loader.bin)

```

69         puts(s_[tesla-os-loader]_%s_00114c51,s_Verifying_nvme_image..._0011516);
70         local_24 = FUN_00100b96(local_18 + local_10,local_2c - local_18);
71         if (local_24 == 0) {
72             puts(s_[tesla-os-loader]_%s_00114c51,s_Successfully_verified_image!_00114f54);
73             *param_3 = local_2c;
74             puts(s_[tesla-os-loader]_%s_00114c51,s_Boot_payload_read_complete_00115179);
75             return local_10;
76         }
77         puts(s_[tesla-os-loader]_Invalid_boot_i_00114f28);
78         uVar2 = FUN_00100c0a(local_24);
79         FUN_001000f4(uVar2,0x20);
80         puts(&DAT_00114beb);
81     }
82 }
83 }
84 }
85 else {
86     puts(s_[tesla-os-loader]_Invalid_boot_c_00114d78);
87     uVar2 = FUN_00100c0a(local_24);
88     FUN_001000f4(uVar2,0x20);
89     puts(&DAT_00114beb);
90 }
91 }
92 }
93 *param_3 = 0;
94 }
95 }
96 else {
97     puts(s_[tesla-os-loader]_%s_00114c51,s_ERROR:_Could_not_find_or_initial_00114f74);
98 }
99 return 0;
100 }
101 }

```

Tesla OS Loader **#Patch**

Listing: tesla-os-loader.bin

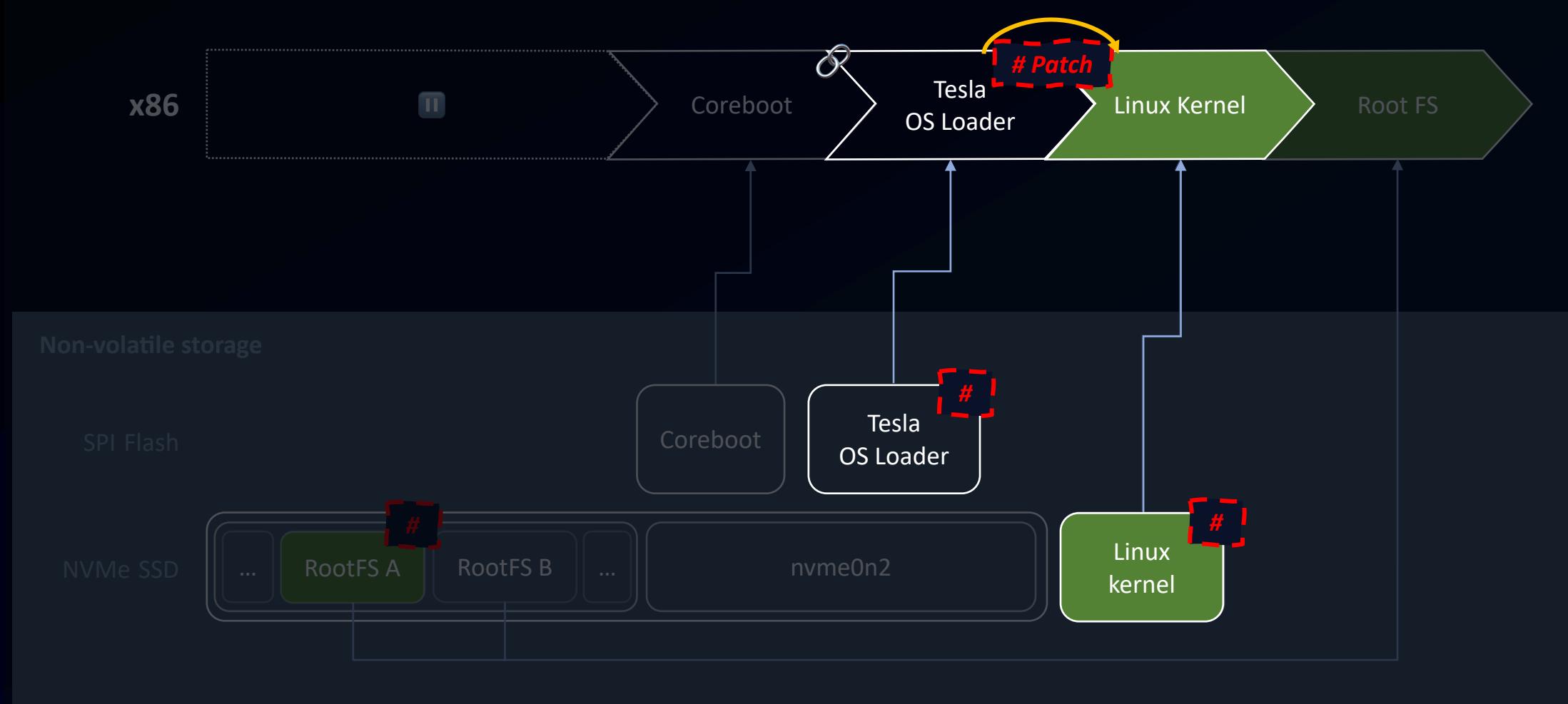
00101b11	83 ec 08	LAB_00101b11
00101b14	68 61 51	SUB ESP,0x8
	11 00	PUSH s_Verifying_nvme_image..._0011516
00101b19	68 51 4c	PUSH s_[tesla-os-loader]_%s_00114c51
	11 00	
00101b1e	e8 aa dc	CALL puts
	00 00	
00101b23	83 c4 10	ADD ESP,0x10
00101b26	8b 45 d8	MOV EAX,dword ptr [EBP + local_2c]
00101b29	2b 45 ec	SUB EAX,dword ptr [EBP + local_18]
00101b2c	8b 4d f4	MOV ECX,dword ptr [EBP + local_10]
00101b2f	8b 55 ec	MOV EDX,dword ptr [EBP + local_18]
00101b32	01 ca	ADD EDX,ECX
00101b34	83 ec 08	SUB ESP,0x8
00101b37	50	PUSH EAX
00101b38	52	PUSH EDX
00101b39	e8 58 f0	CALL FUN_00100b96
	ff ff	
00101b3e	83 c4 10	ADD ESP,0x10
00101b41	89 45 e0	MOV dword ptr [EBP + local_24],EAX
00101b44	83 7d e0 00	CMP dword ptr [EBP + local_24],0x0
00101b48	eb 3e	JMP LAB_00101b88
00101b4a	83 ec 0c	SUB ESP,0xc
00101b4d	68 28 4f	PUSH s_[tesla-os-loader]_Invalid_boot_
	11 00	
00101b52	e8 76 dc	CALL puts
	00 00	
00101b57	83 c4 10	ADD ESP,0x10
00101b5a	83 ec 0c	SUB ESP,0xc
00101b5d	ff 75 e0	PUSH dword ptr [EBP + local_24]

Decompile: FUN_00101838 - (tesla-os-loader.bin)

```
puts(s_[tesla-os-loader]_%s_00114c51,s_Verifying_nvme_image..._0011516);
local_24 = FUN_00100b96(local_18 + local_10,local_2c - local_18);
if (local_24 == 0) {
    puts(s_[tesla-os-loader]_%s_00114c51,s_Successfully_verified_image!_00114f54);
    *param_3 = local_2c;
    puts(s_[tesla-os-loader]_%s_00114c51,s_Boot_payload_read_complete_00115179);
    return local_10;
}
puts(s_[tesla-os-loader]_Invalid_boot_i_00114f28);
uVar2 = FUN_00100c0a(local_24);
FUN_001000f4(uVar2,0x20);
puts(&DAT_00114beb);
}
}
else {
    puts(s_[tesla-os-loader]_Invalid_boot_c_00114d78);
    uVar2 = FUN_00100c0a(local_24);
    FUN_001000f4(uVar2,0x20);
    puts(&DAT_00114beb);
}
}
*param_3 = 0;
}
else {
    puts(s_[tesla-os-loader]_%s_00114c51,s_ERROR:_Could_not_find_or_initial_00114f74);
}
return 0;
```

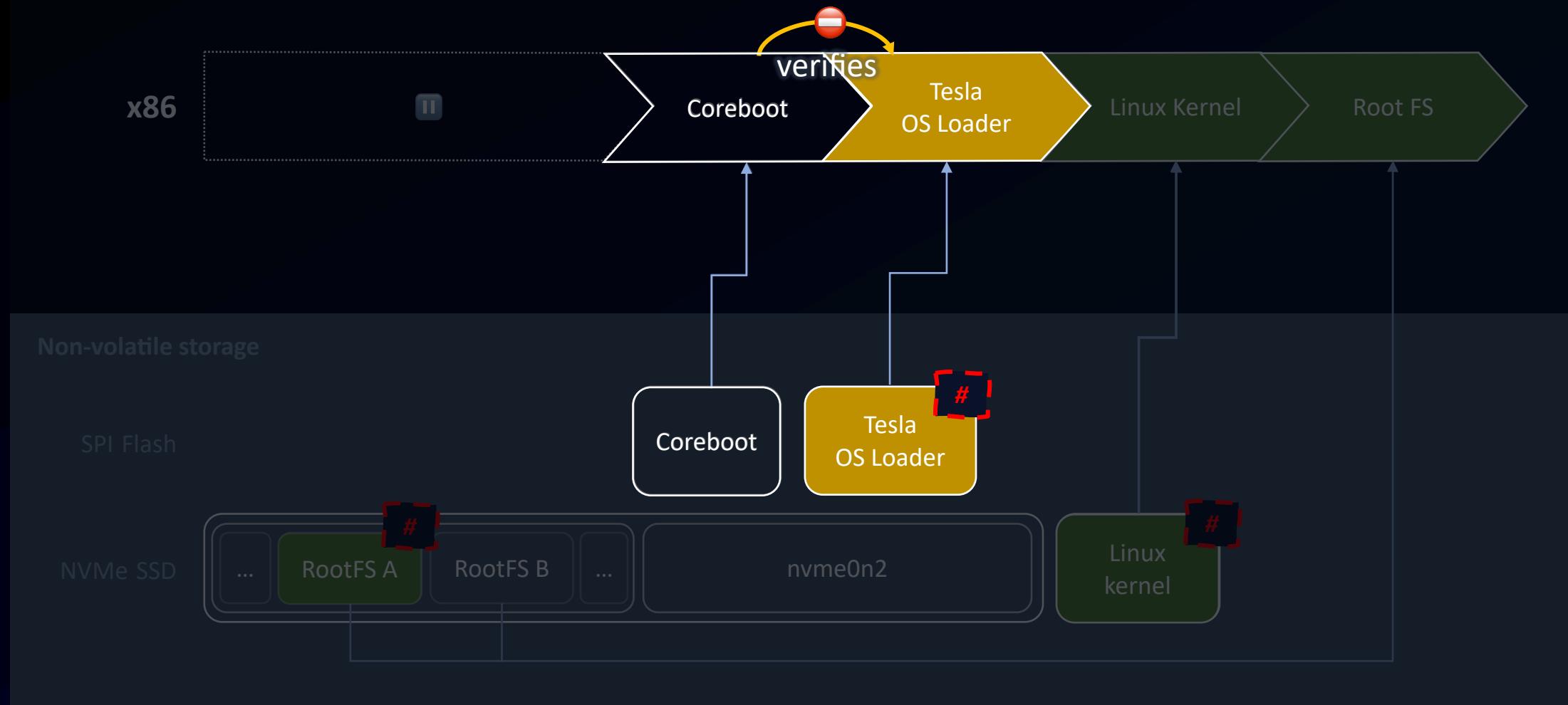
Verified Boot

loaded
rejected



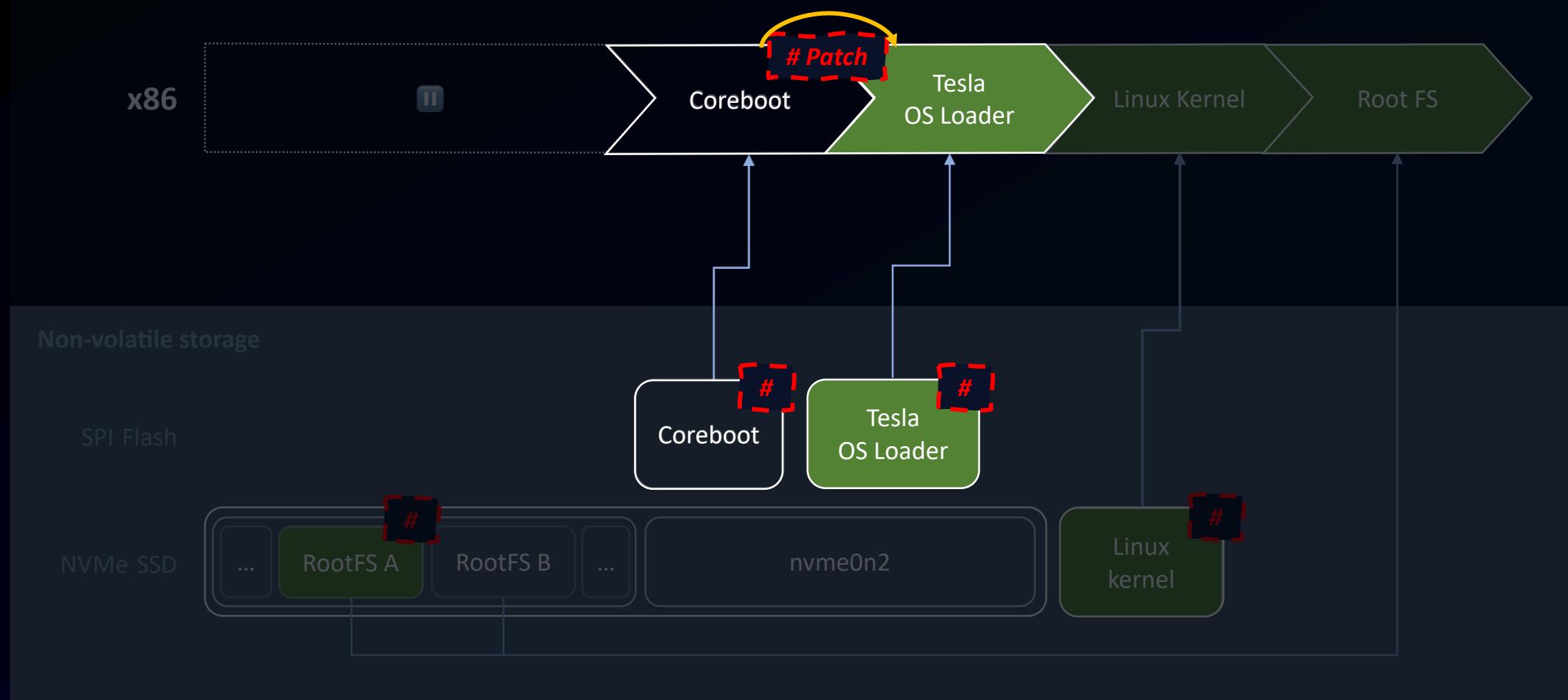
Verified Boot

loaded
rejected



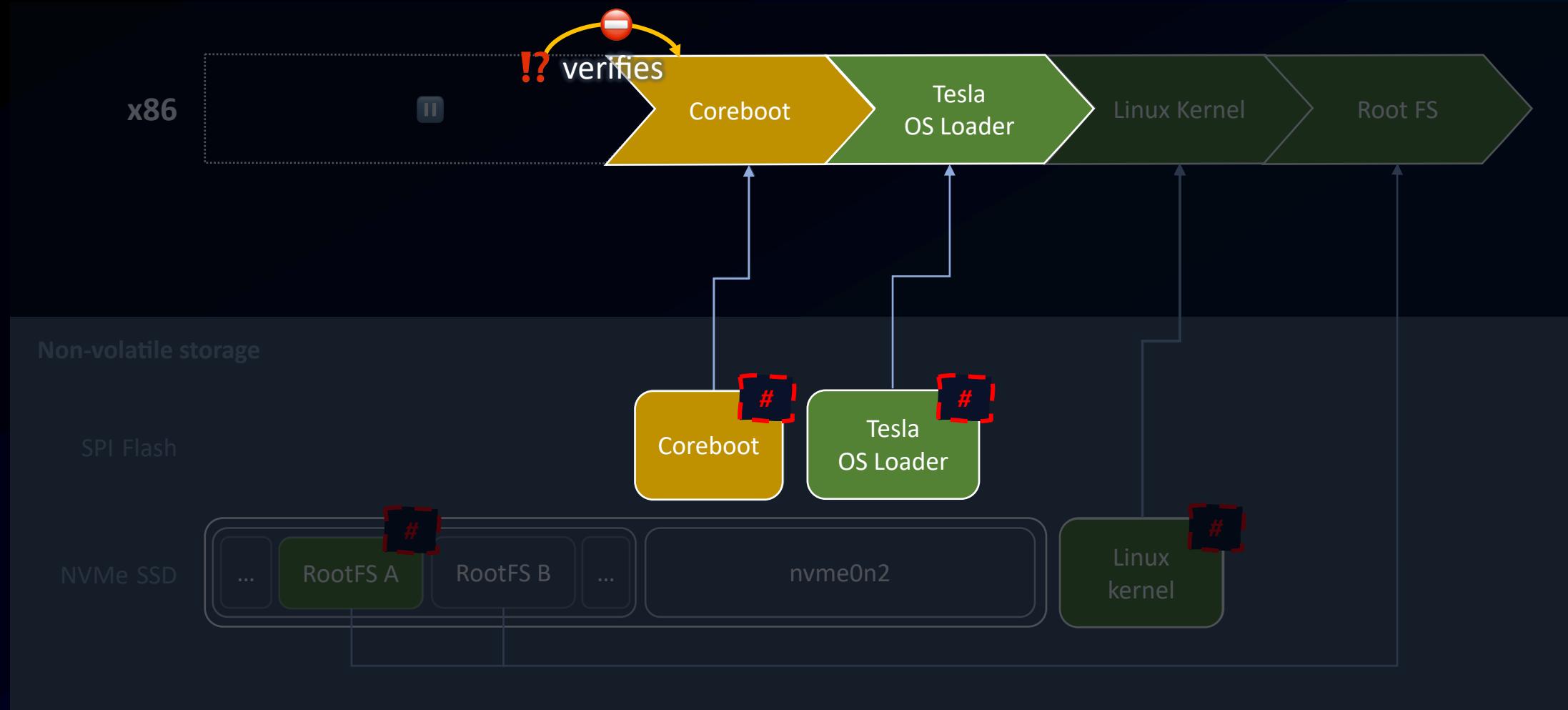
Verified Boot

loaded
rejected



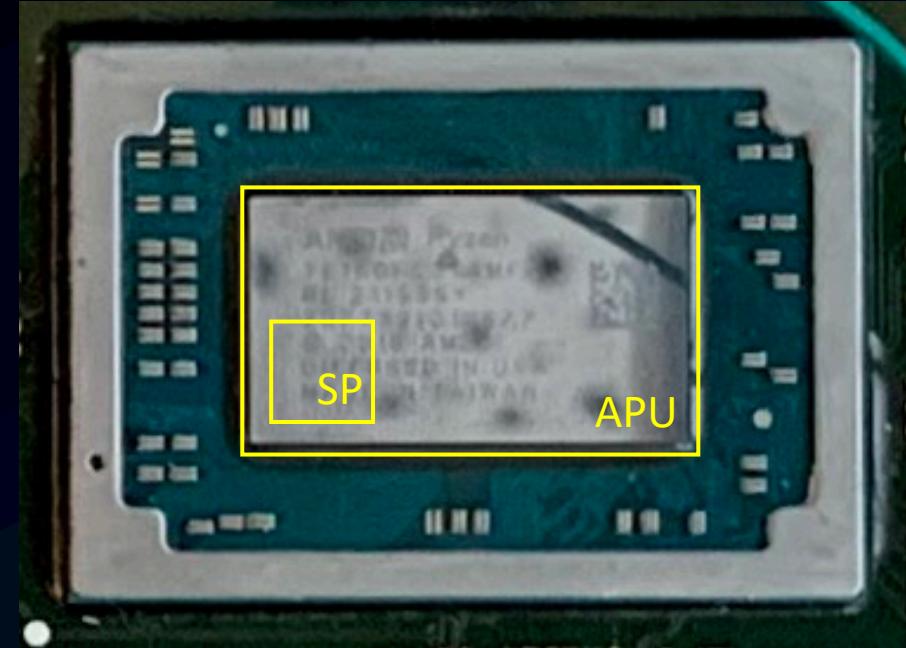
Verified Boot

loaded
rejected



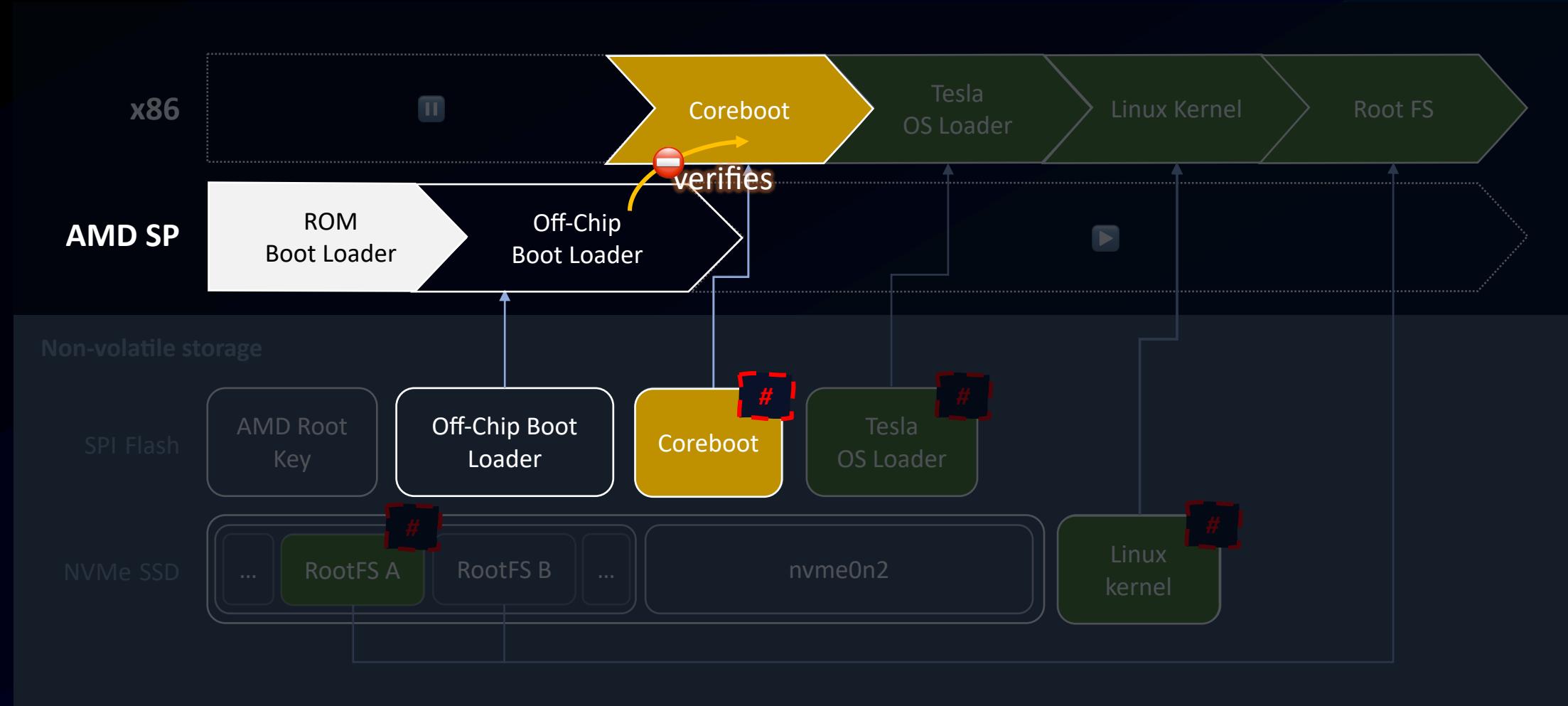
AMD Secure Processor

- ARMv7 µController
- Integrated into CPU SoC
- Highly privileged
- Variety of responsibilities
 - Hardware root of trust
 - Firmware TPM (fTPM) for key management and more
 - (On EPYC Servers) Secure Encrypted Virtualization



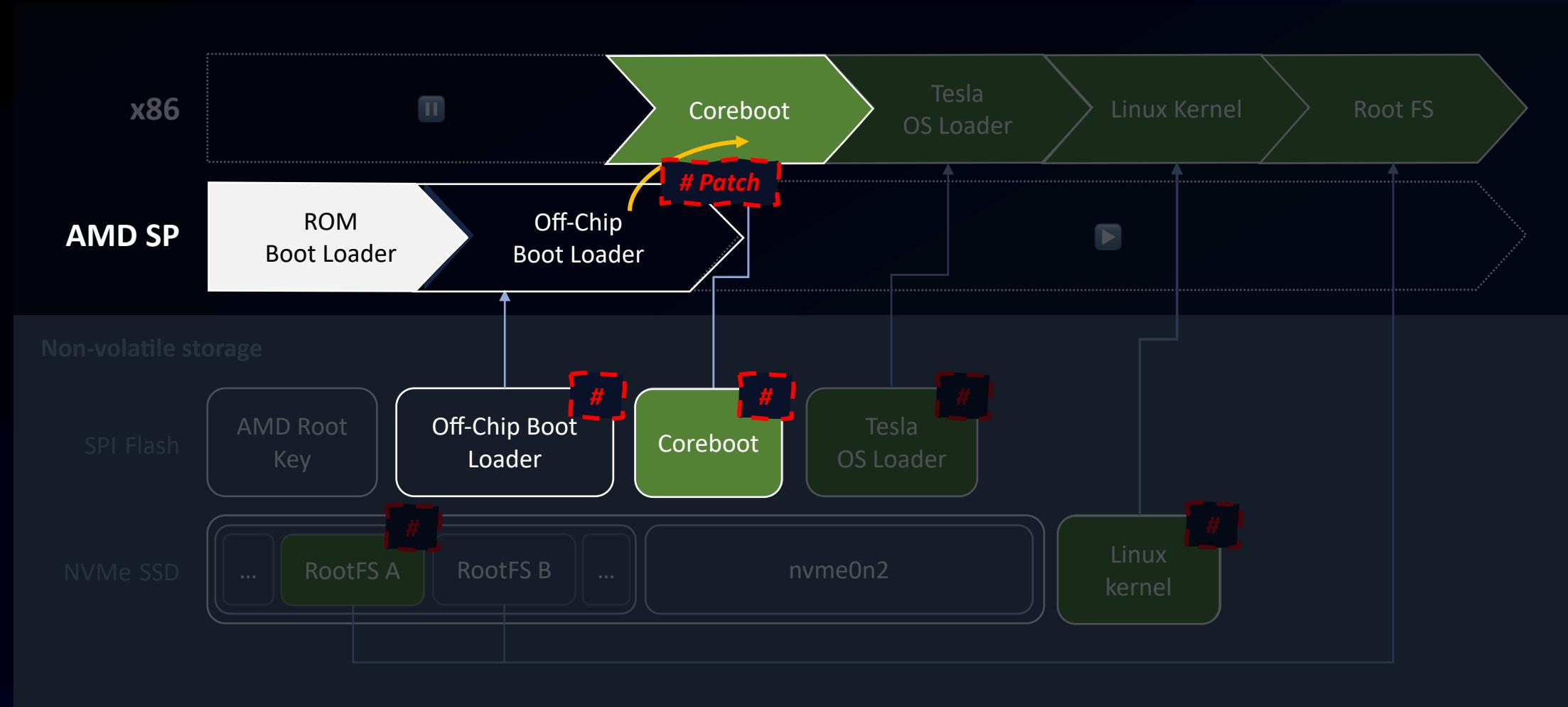
loaded
 rejected

AMD Platform Secure Boot



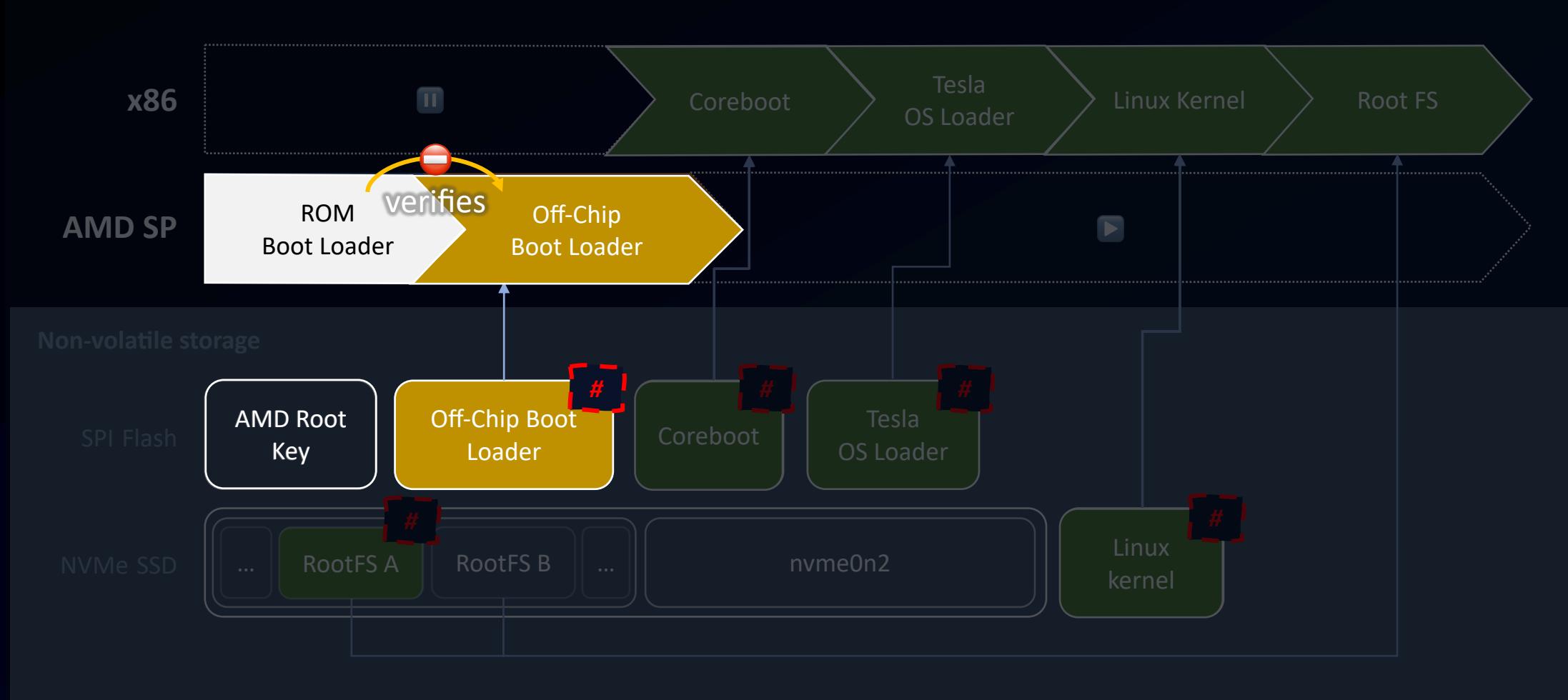
loaded
rejected

AMD Platform Secure Boot

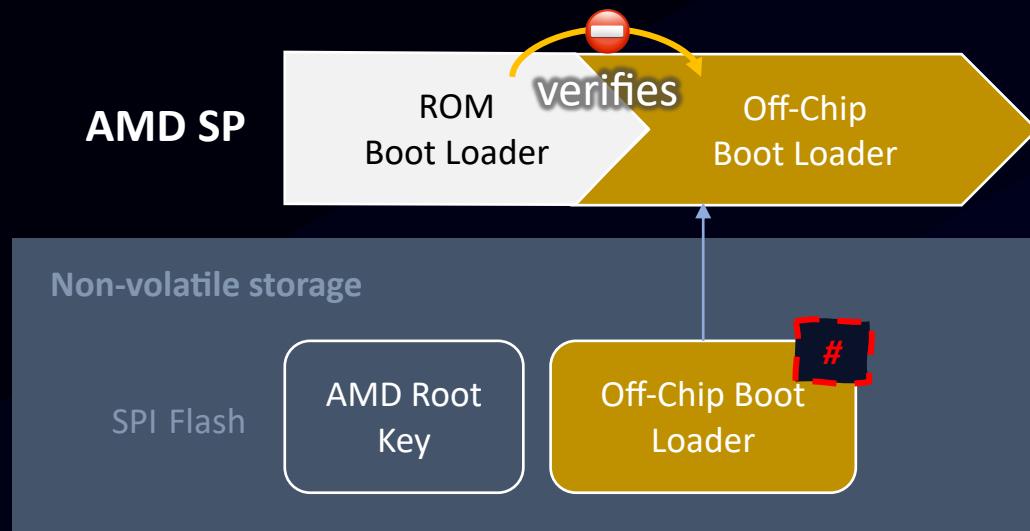


loaded
 rejected

AMD Platform Secure Boot



Previous AMD SP Vulnerabilities



- 2019: Off-Chip Boot Loader Buffer overflow
 - Arbitrary Code Execution ✓
 - **Fixed via firmware updates**
- 2020: ROM Boot Loader Buffer overflow
 - Arbitrary Code Execution ✓
 - Not fixable (ROM)
 - Fixed in new generations (>= Zen 2)
 - **Fixes backported to Tesla's Zen 1 APU**

Tesla's Security Evolution

2014

- Open X servers
- Hardcoded passwords
- Diagnostic Ethernet: root
- No code signing

2023

- Firmware and OS signing
- Chain of trust during boot
- **Root of trust in AMD SoC**

Outline

1

Analyzing Boot and Firmware Security

2

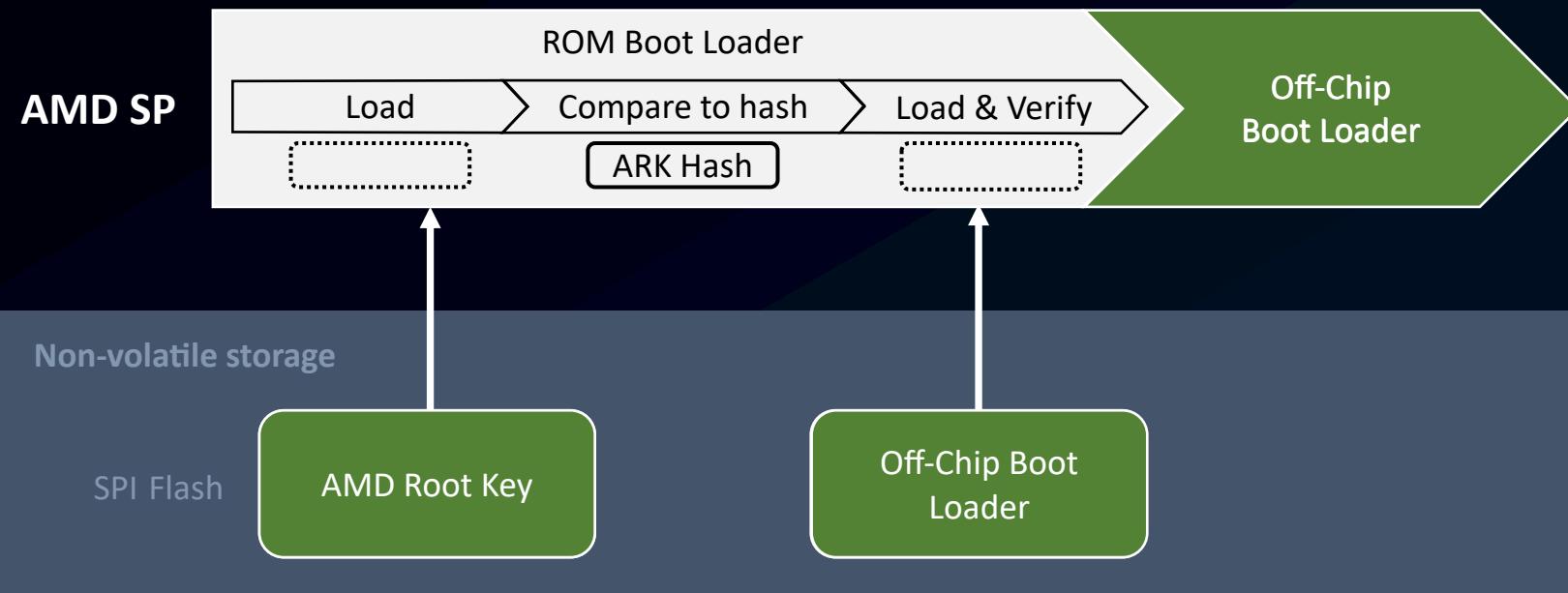
Hotwiring the Infotainment system

3

Extracting Secrets from the Tesla

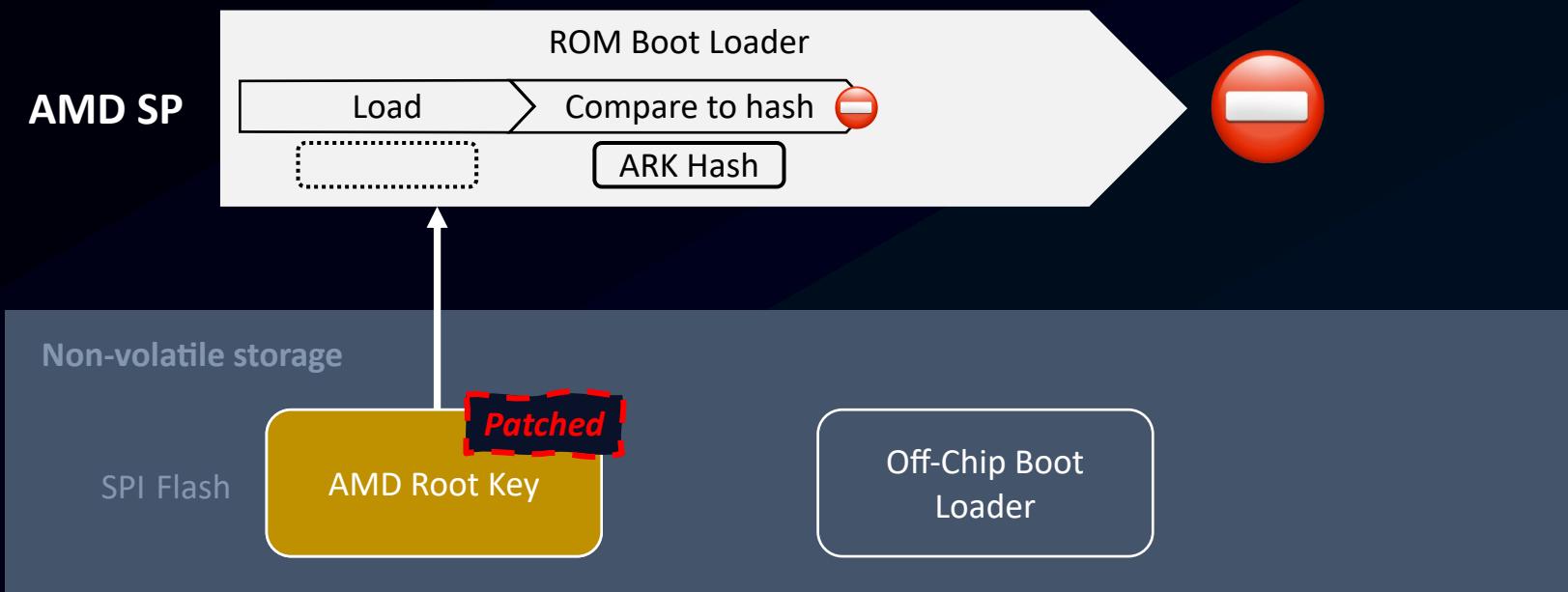
Regular Early Boot Verification

loaded
rejected



loaded
 rejected

Failed Early Boot Verification



Fault Injection Attacks

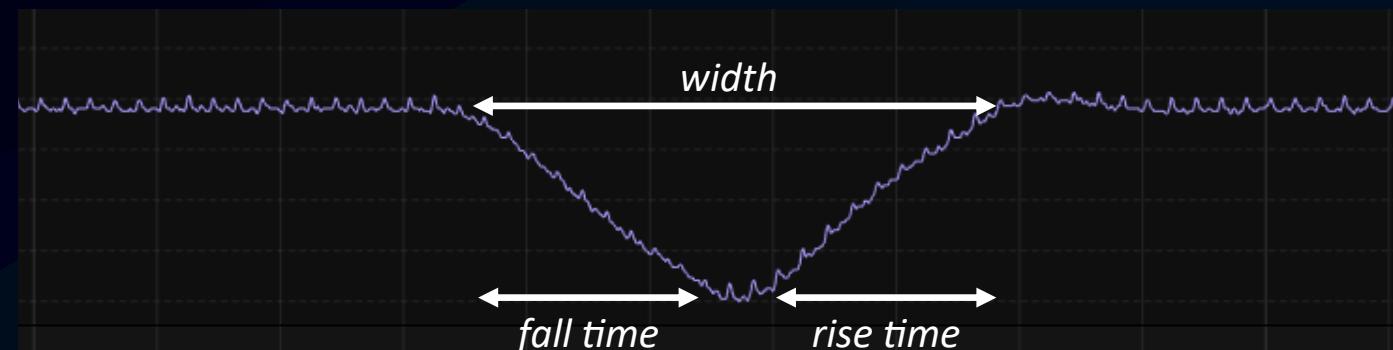


Induce fault by altering the IC's environment:

- Laser, electromagnetic-radiation, clock, supply voltage

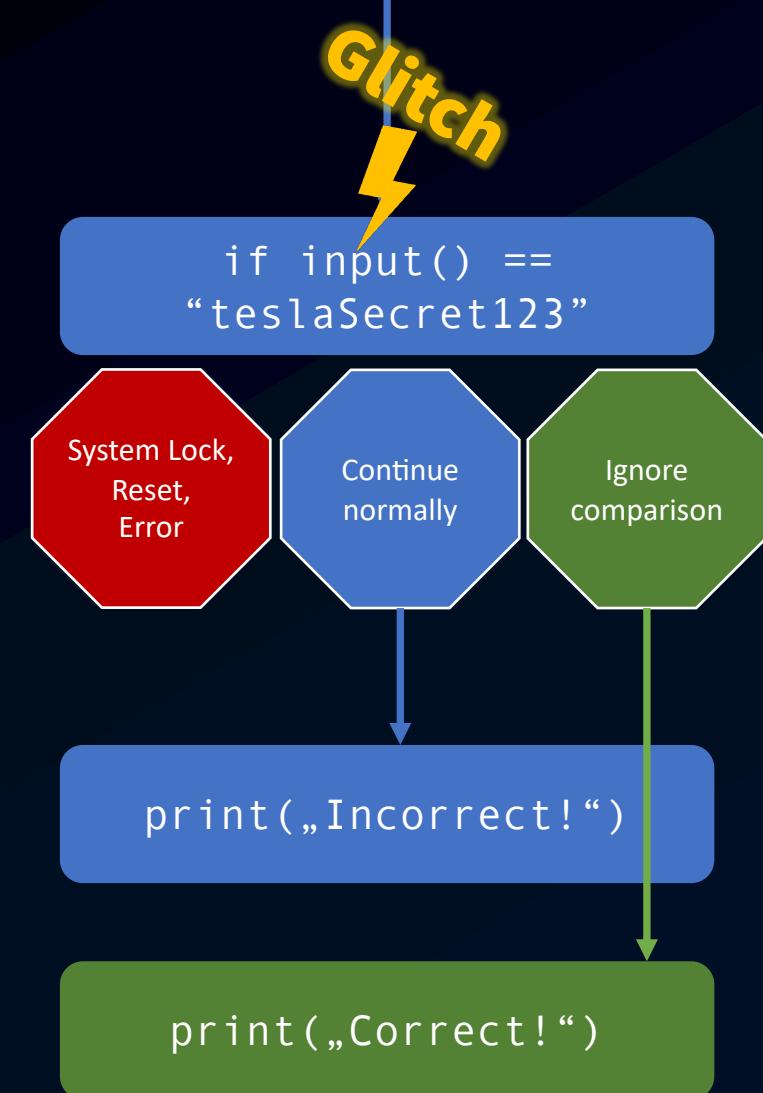
Voltage Glitching:

- Lowering voltage shortly



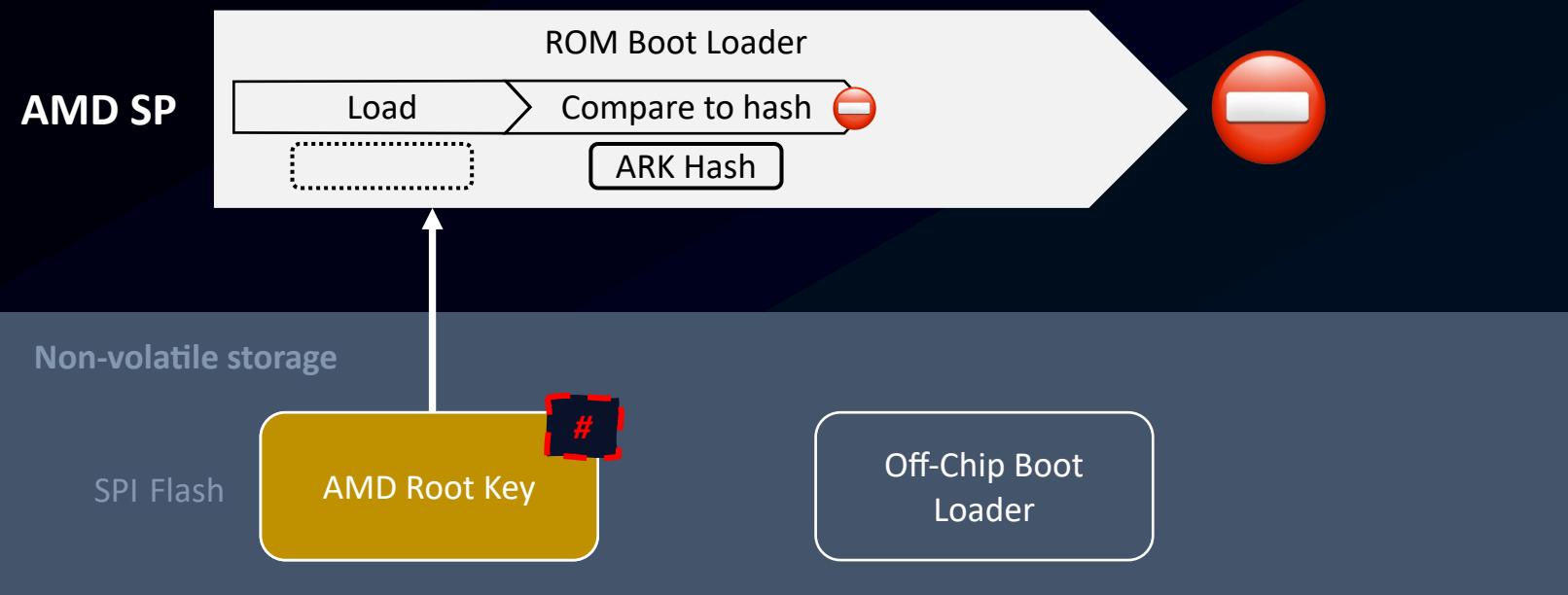
Key Challenges

- **Most faults are “useless”**
- **Trigger:**
Figure out when targeted check happens
- **Parameters:**
Voltage drop steepness, width, minimum
- **Reset/Success:**
Identify failed attacks and retry as fast as possible



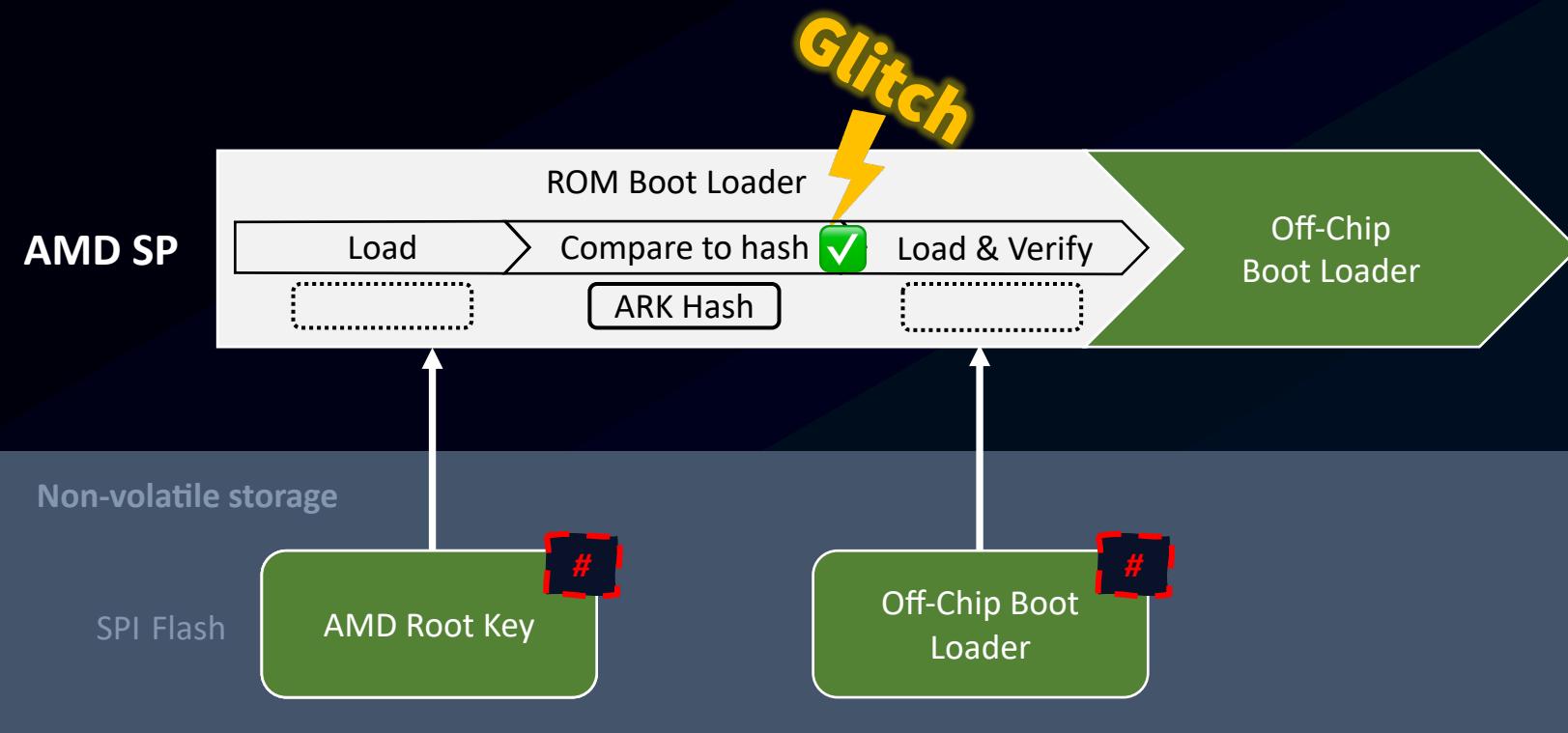
Failed Early Boot Verification

loaded
rejected

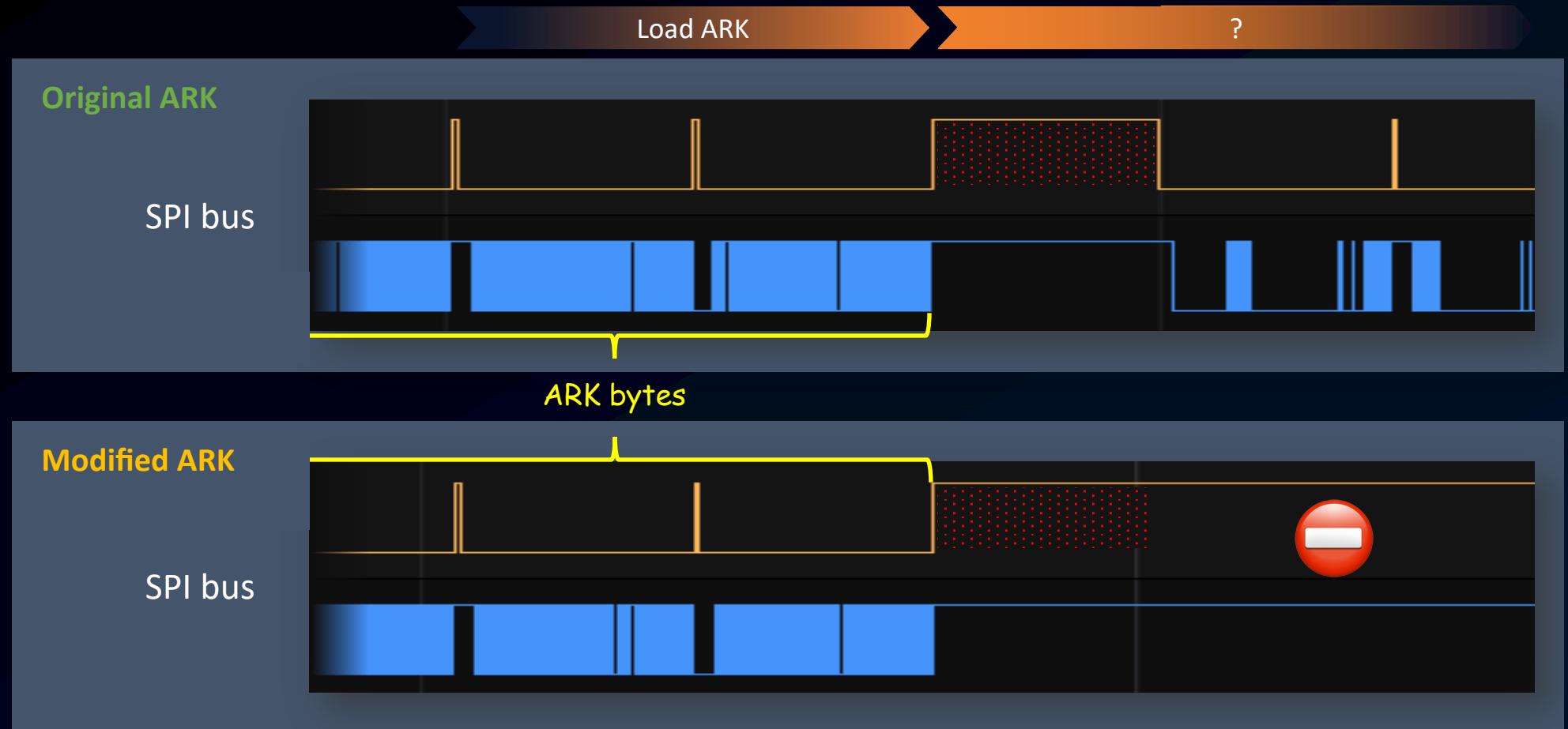


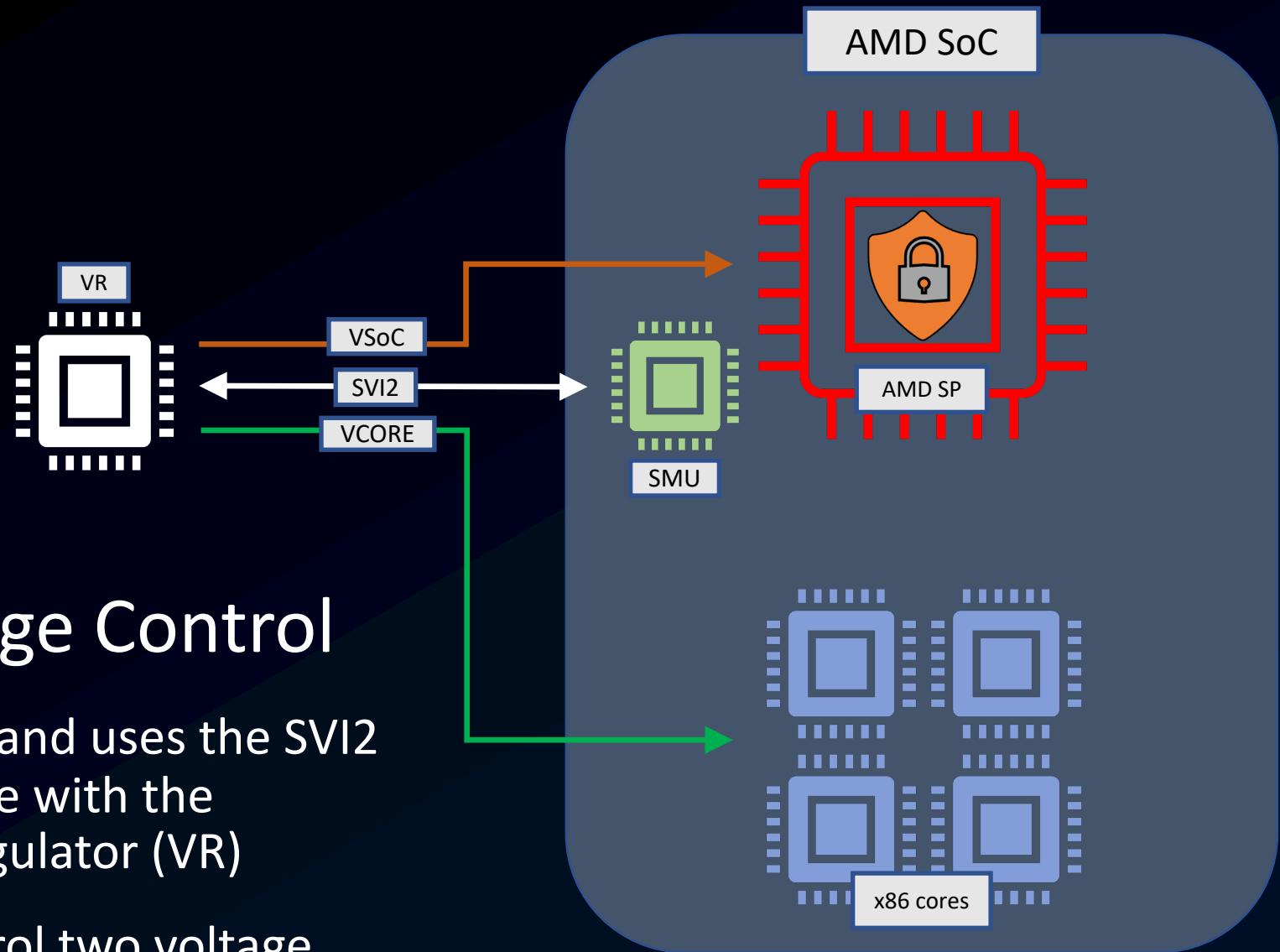
Glitched Early Boot Verification

loaded
rejected



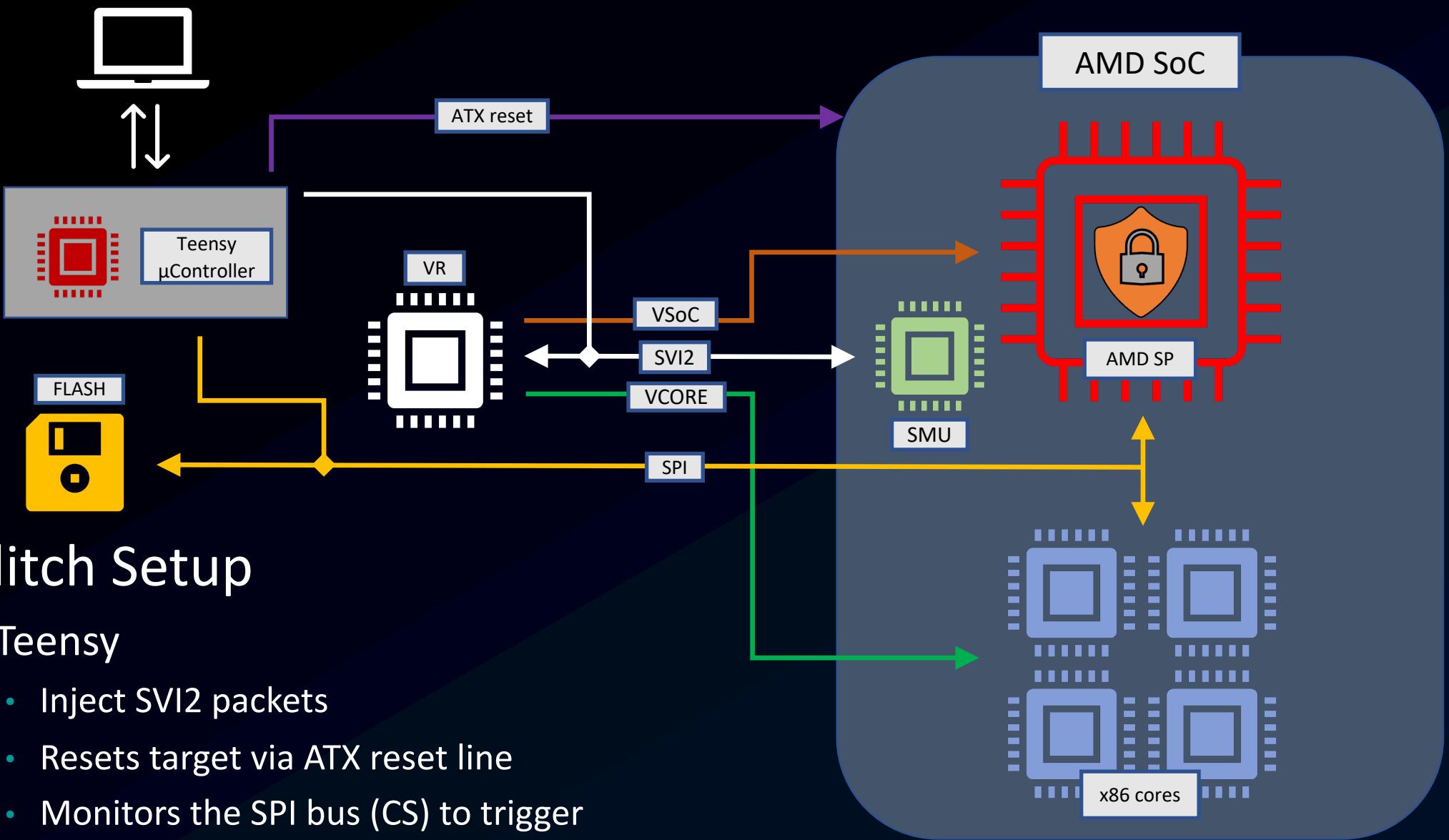
Finding the ARK Verification Time Window





Dynamic Voltage Control

- SMU monitors SoC and uses the SVI2 bus to communicate with the external voltage regulator (VR)
- SVI2 allows to control two voltage domains per VR



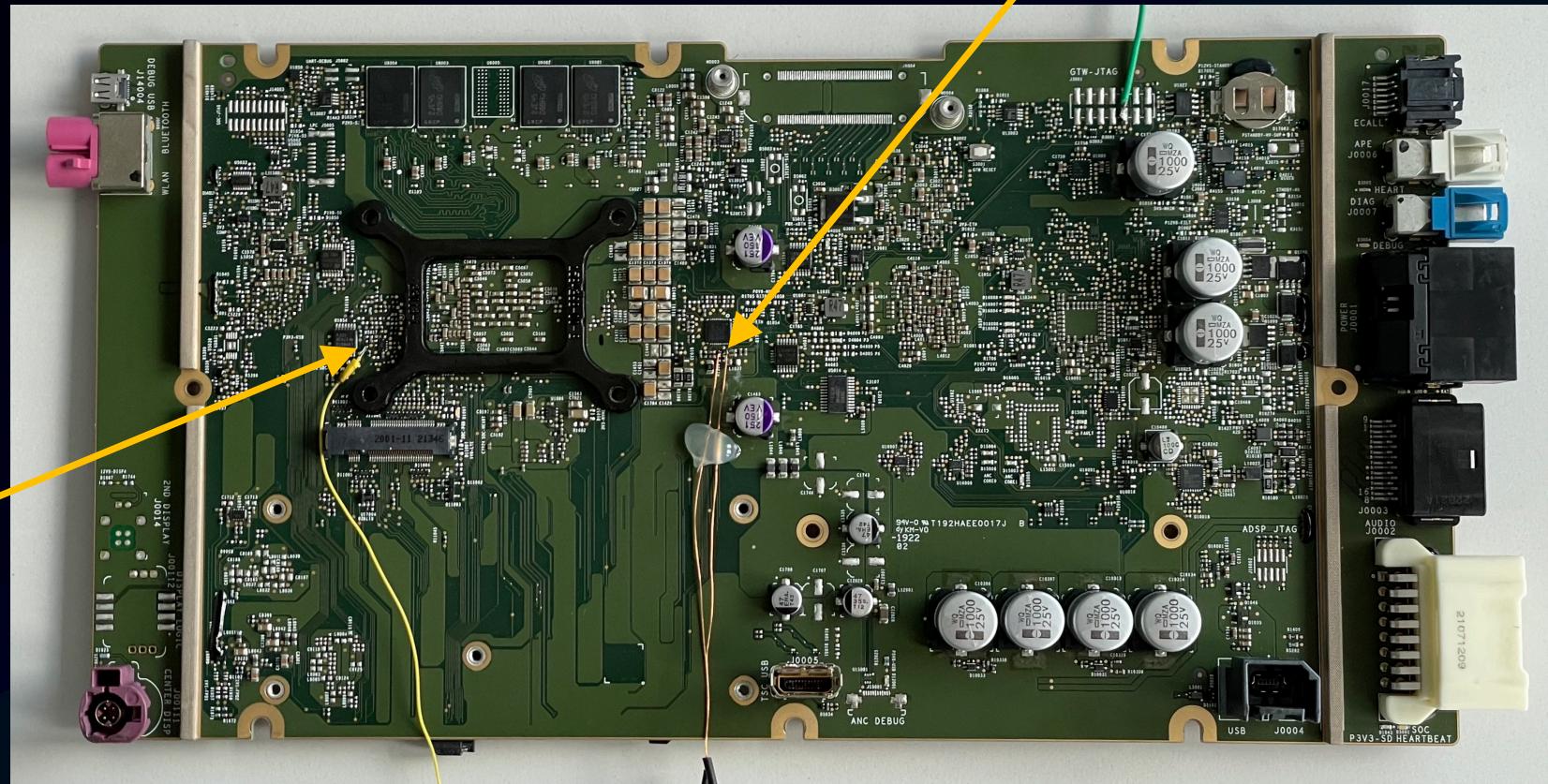
Glitch Setup

- Teensy
 - Inject SVI2 packets
 - Resets target via ATX reset line
 - Monitors the SPI bus (CS) to trigger voltage glitch
- External PC controls glitch parameters

Voltage Glitch Wiring

SVI2 bus (SVD + SVC)

SPI chip-select



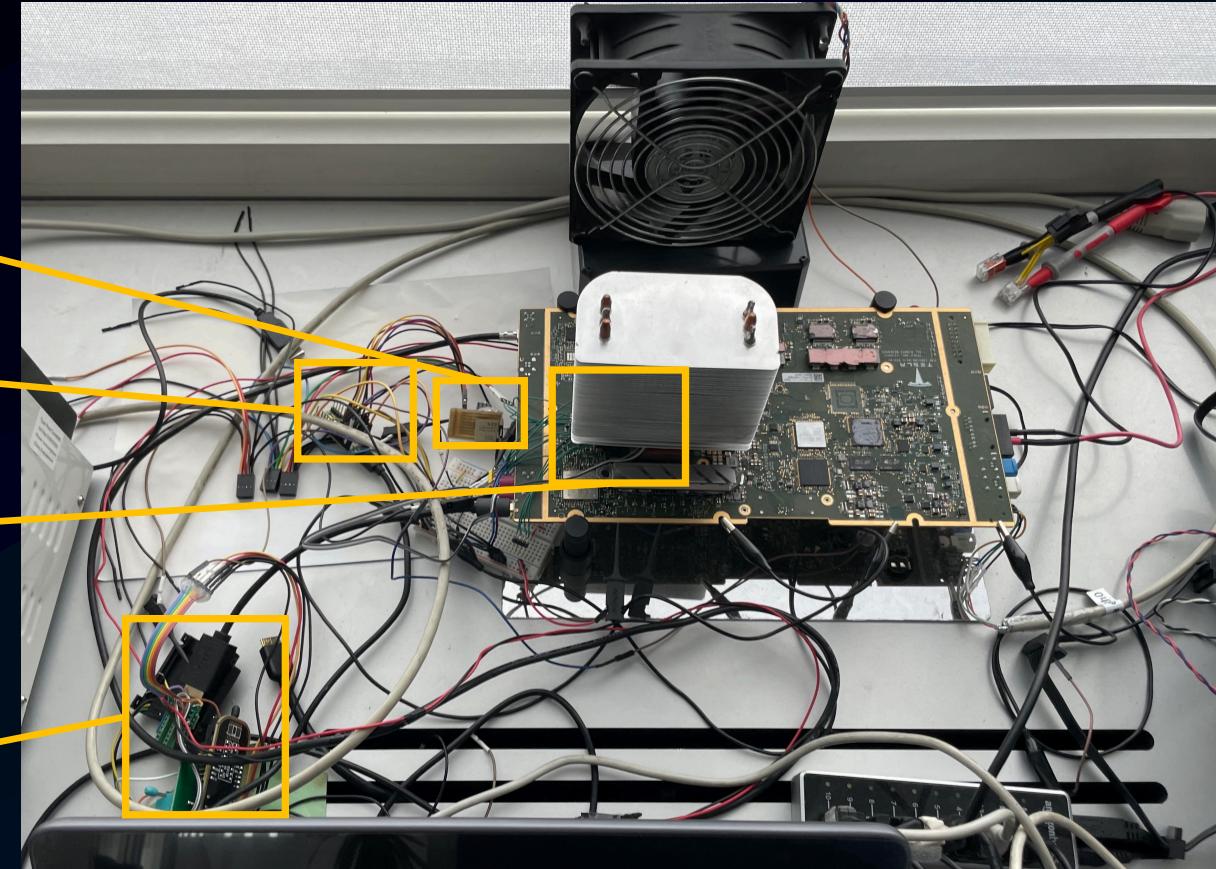
Glitch Setup in Reality

SVI2 bus

Teensy
μController

SPI bus

ATX reset
SPI programmer
Serial output



Voltage Glitch Steps



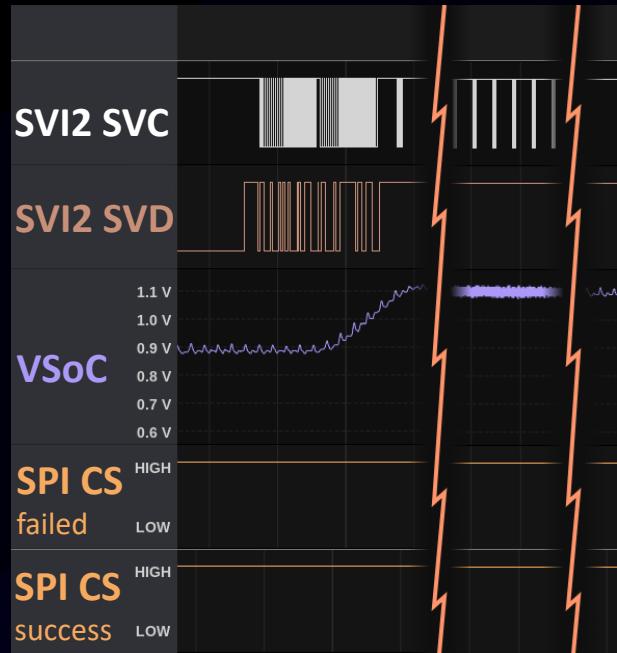
- SVI2 SVC: bus clock
- SVI2 SVD: bus data
- VSoC: target's voltage
- SPI CS: chip-select signal

Voltage Glitch Steps



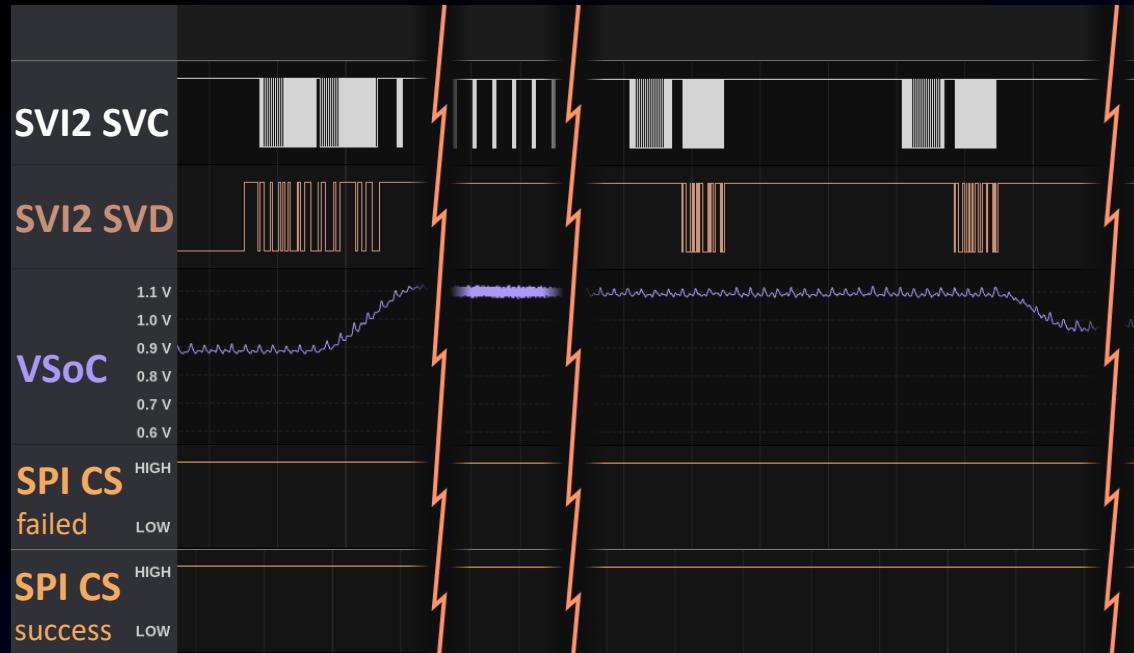
- SoC sets initial voltage
- SVD rising edge triggers attack logic
- VSoC rises

Voltage Glitch Steps



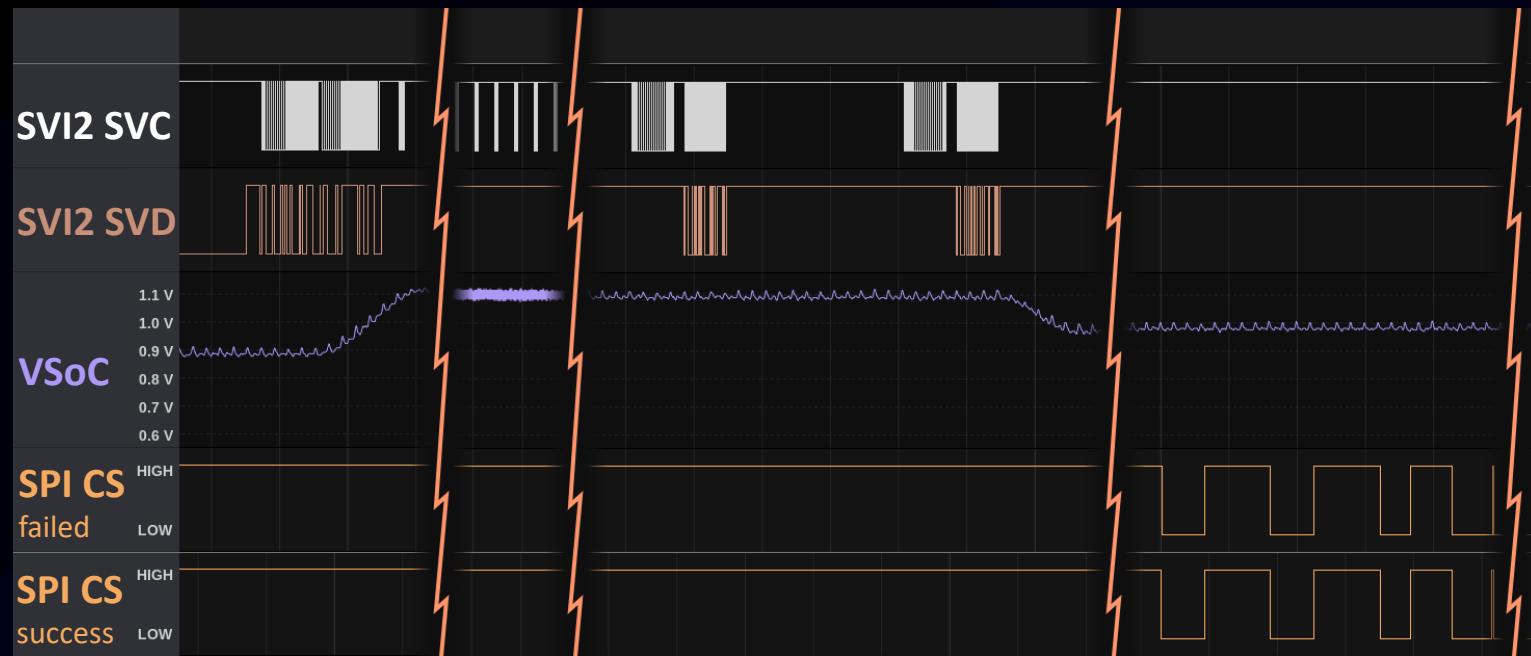
- VR sends telemetry packets
- VSoC stable

Voltage Glitch Steps



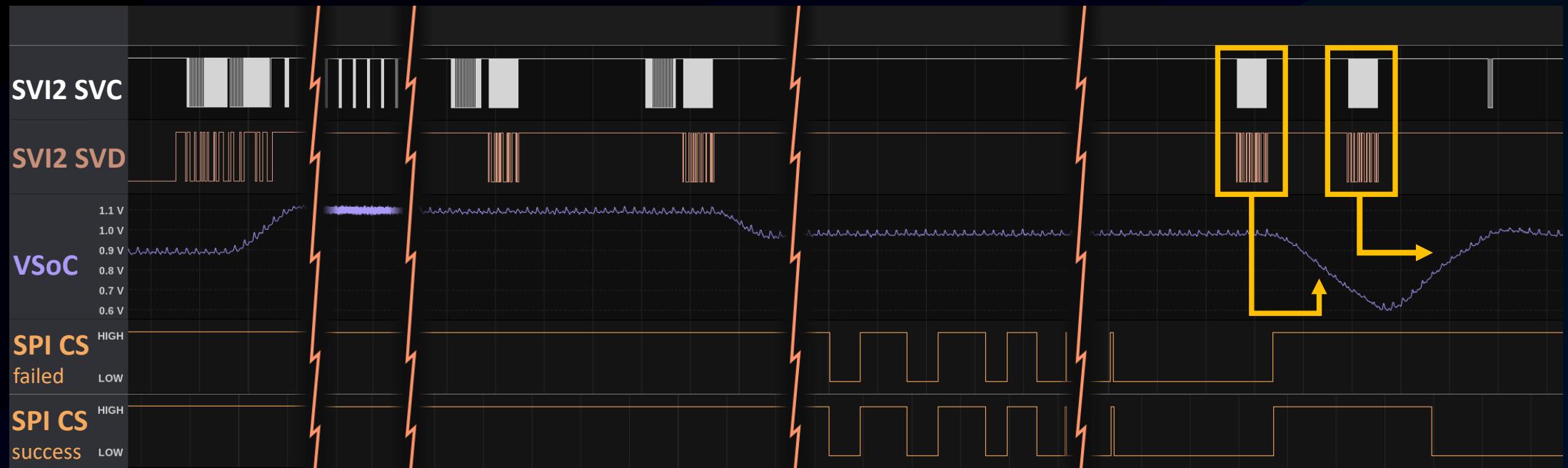
- Teensy injects **SVI2** packets
- Disable telemetry to avoid collisions
- **VSoC** is adjusted

Voltage Glitch Steps



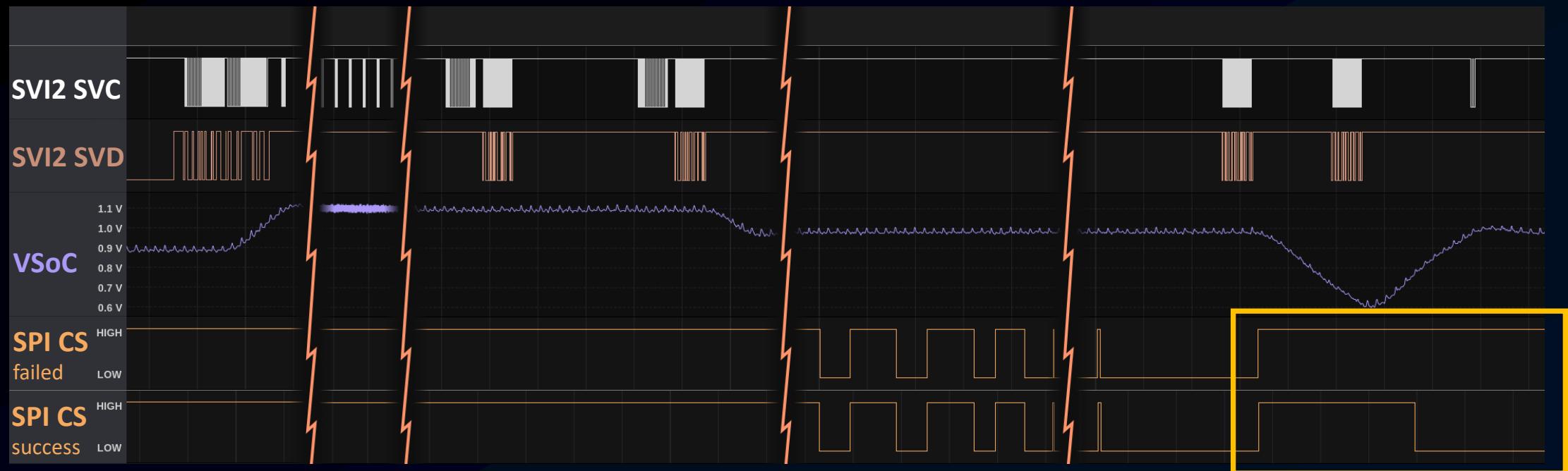
- Teensy starts counting CS edges to trigger glitch on time
- CS becomes active → AMD SP loads data

Voltage Glitch Steps



- Teensy injects two SVI2 packets to create voltage disturbance
- Voltage drop on VSoC (glitch)

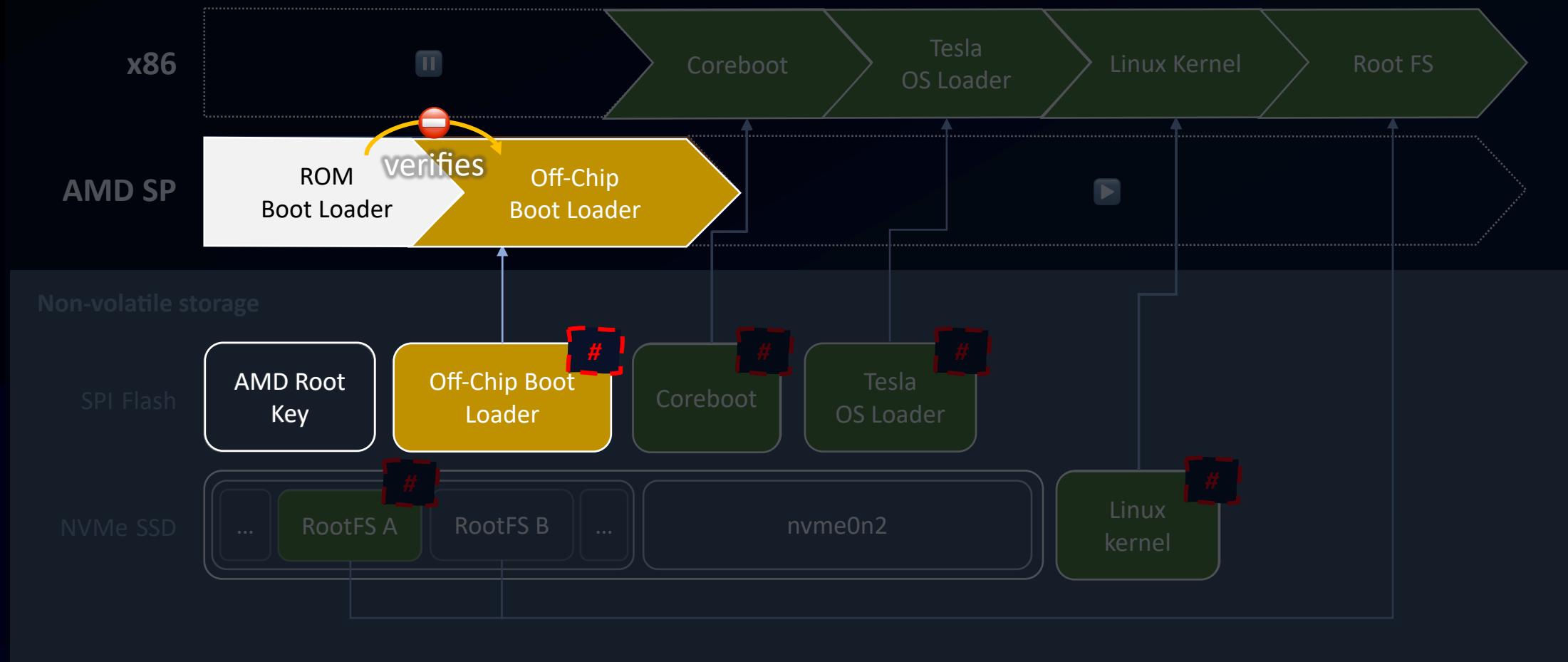
Voltage Glitch Steps



- Teensy monitors **CS** to detect success
- Teensy resets target on fail
- CS inactive (high) → failed attempt
- CS active (low) → successful attempt

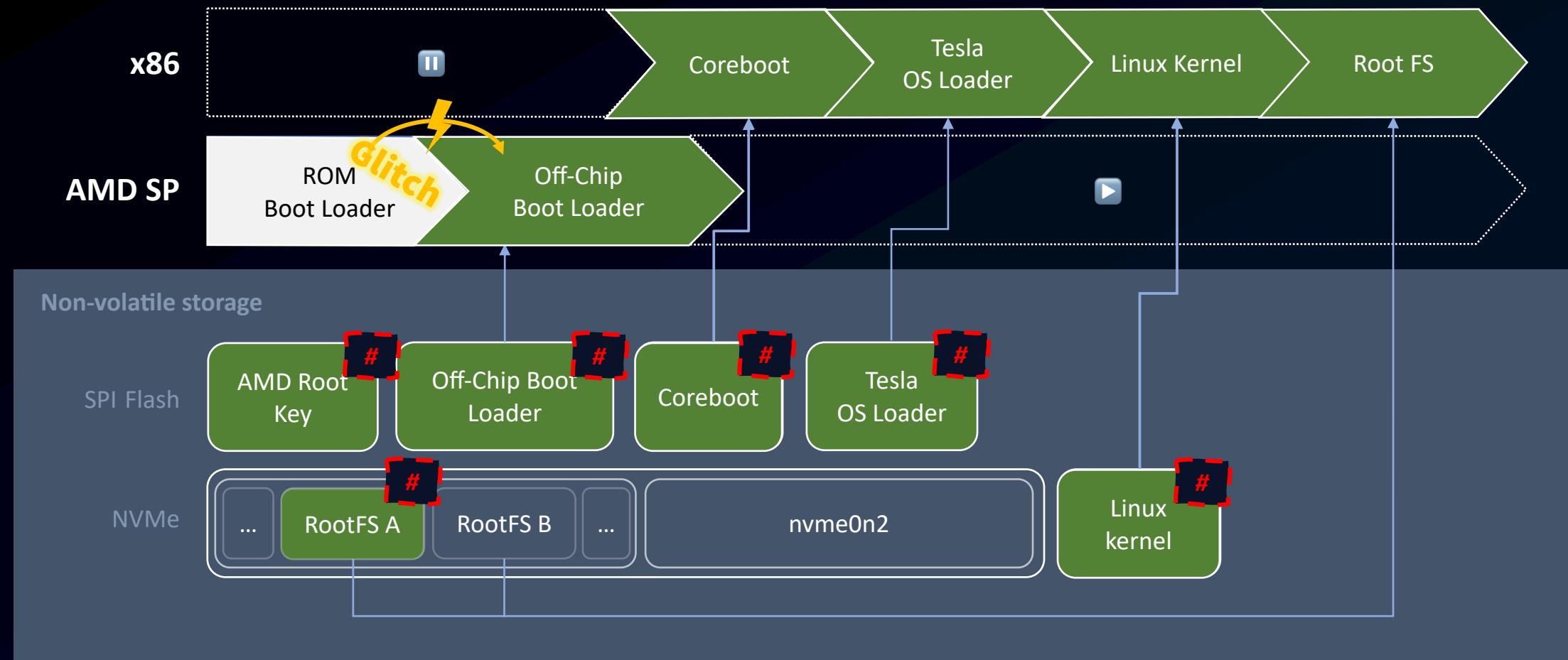
loaded
rejected

AMD Platform Secure Boot



AMD Platform Secure Boot

loaded
rejected



Trying to Activate the Rear Seat Heaters



Finding their Configuration ID

```
● ● ●  
1 {  
2   "accessId": 13,  
3   "codeKey": "rearSeatHeaters",  
4   "content": {  
5     "enums": [  
6       {  
7         "codeKey": "NONE",  
8         "description": "None",  
9         "value": 0  
10      },  
11      {  
12        "codeKey": "KONGSBERG_LOW_POWER",  
13        "description": "Kongsberg low-power heaters",  
14        "value": 1  
15      }  
16    ]  
17  },  
18  "description": "Type of rear seat heaters installed",  
19  "products": [  
20    "Model3",  
21    "ModelY"  
22  ]  
23 }
```

```
deploy@psp-deploy:~/tesla-hacking$ picocom /dev/ttyUSBHUB10 -b 115200 | tee -a $(date +"%Y_%m_%d").log
```

Serial console

```
deploy@psp-deploy:~$ ssh -t root@192.168.90.100 'bash'
```

SSH console

```
deploy@psp-deploy:~/tesla/fi-attack$ python3 start-tesla.py -r ../../tesla-hacking/roms/boot_nvme.bin
```

Attack script

Trying to Activate the Rear Seat Heaters



What About Persistence?

- Sorry, voltage glitch is not persistent
 - Need to glitch on every Infotainment boot
 - But the car configuration survives regular infotainment (re)boot
 - And Infotainment supposedly doesn't reboot very often
- Glitching could be made even smoother by a mod chip/PCB
 - Implementation detail ... 😊
 - *We leave this as an exercise to the interested audience*

Secure Configuration Items

- Demo possible since the rear seat heaters were an “insecure configuration item” in our Gateway firmware version
 - “Secure configuration items” can only be changed with a valid signature
- **“Rear seat heaters were upgraded to be a signed configuration starting in the 2022.44 release”, Tesla told us**
- So being root on the Infotainment is not sufficient
 - Software or hardware vulnerability in Gateway necessary

ADVISORY DETAILS

July 18th, 2023

(Pwn2Own) Tesla Model 3 Gateway Firmware Signature Validation Bypass Vulnerability

[ZDI-23-972](#)

[ZDI-CAN-20734](#)

CVE ID CVE-2023-32156

CVSS SCORE 9.0, (AV:A/AC:L/PR:L/UI:N/S:C/C:H/I:H/A:H)

AFFECTED VENDORS Tesla

AFFECTED PRODUCTS Model 3

VULNERABILITY DETAILS This vulnerability allows network-adjacent attackers to execute arbitrary code on affected Tesla Model 3 vehicles. An attacker must first obtain the ability to execute privileged code on the Tesla infotainment system in order to exploit this vulnerability.

The specific flaw exists within the handling of firmware updates. The issue results from improper error-handling during the update process. An attacker can leverage this vulnerability to execute code in the context of Tesla's Gateway ECU.

ADDITIONAL DETAILS Fixed in 2023.12 firmware release.

Outline

1

Analyzing Boot and Firmware Security

2

Hotwiring the Infotainment system

3

Extracting Secrets from the Tesla

What secrets are there on the Tesla?

CAR CREDENTIALS

- Authenticates car against Tesla servers (Tesla's car VPN)
 - Firmware updates
 - Car configuration
- Bound to Vehicle Identification Number (VIN)
- Used to remotely (de-)authorize services

USER DATA

- Phones connected via Bluetooth
 - Contacts, calendar, call logs ...
- Locations visited
- WiFi passwords
- Spotify and Gmail session cookies

How are these secrets secured?

- Everything used to be cleartext
 - Car Credentials on SD card, on storage
 - User data on cleartext storage partition
- Now there is TPM-based security
 - Car Creds sealed in TPM
 - User data partition encrypted, key sealed in TPM



What we extracted

- We wrote a paper on attacking AMD's fTPM
 - Extracting the TPM's internal state
 - *Unsealing arbitrary TPM objects*
- We extracted the car credentials
→ giving us access to Tesla's server endpoints meant for cars
- We extracted the encrypted user partition's disk encryption keys
→ we have access to user data

ics.CR] 2 May 2023

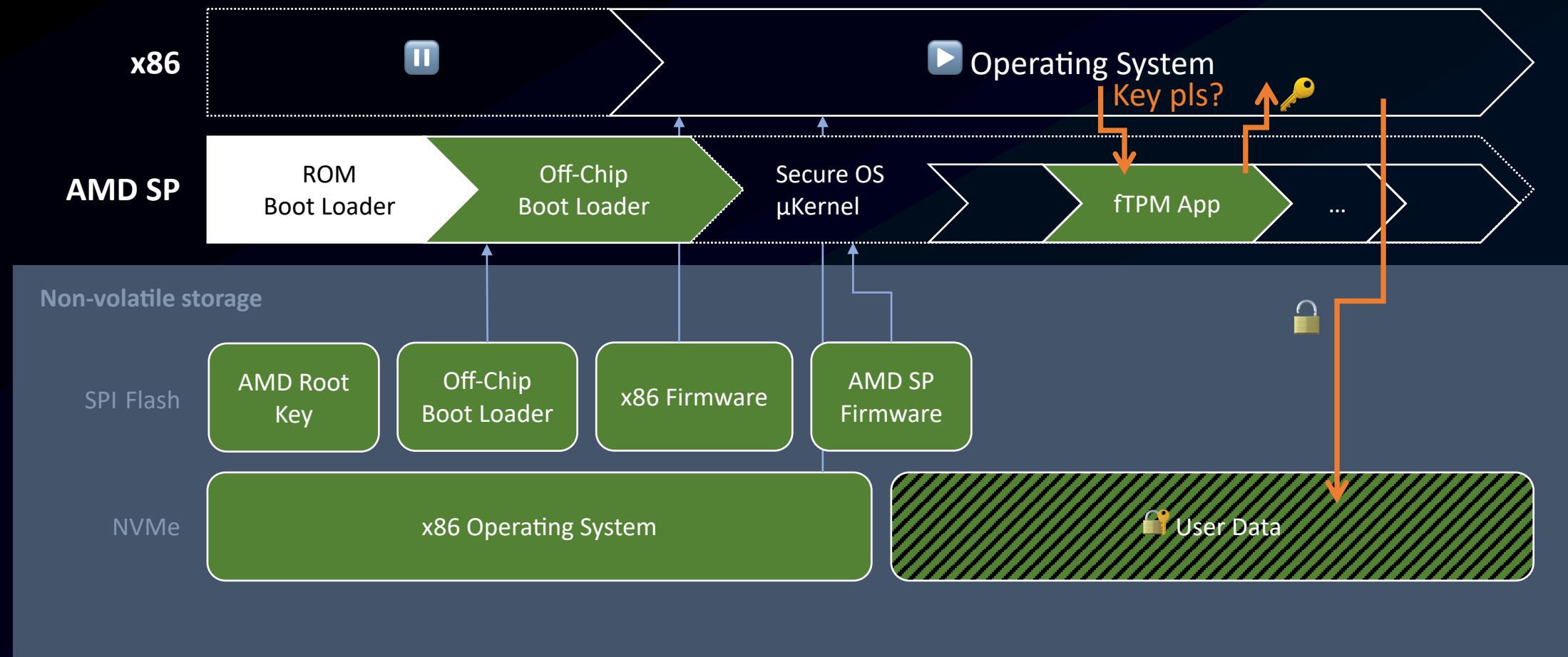
faultTPM: Exposing AMD fTPMs' Deepest Secrets
Hans Niklas Jacob*, Christian Werling*, Robert Buhren, Jean-Pierre Seifert†
Technische Universität Berlin – SecT
†also: Fraunhofer SIT
{ hnj, cwerling, robert.buhren, jpseifert }@sec.tu-berlin.de

Abstract—Trusted Platform Modules (TPMs) constitute an integral building block of modern security features. Moreover, as Windows 11 made a TPM 2.0 mandatory, they are subject to an ever-increasing academic challenge. While discrete TPMs (dTPMs) – as found in higher-end systems – have been susceptible to attacks on their exposed communication interface, more common firmware TPMs (fTPMs) are immune to this attack vector as they do not communicate with the CPU via an exposed bus. In this paper, we analyze a new class of attacks against fTPMs: Attacking their Trusted Execution Environment (TEE) can lead to a full TPM state compromise. We experimentally verify this attack by compromising the AMD Secure Processor (AMD-SP), which constitutes the TEE for AMD's fTPMs. In contrast to previous dTPM sniffing attacks, this vulnerability exposes the complete internal TPM state of this fTPM. It allows us to extract any cryptographic material stored or sealed by the fTPM regardless of authentication mechanisms such as Platform Configuration Register (PCR)

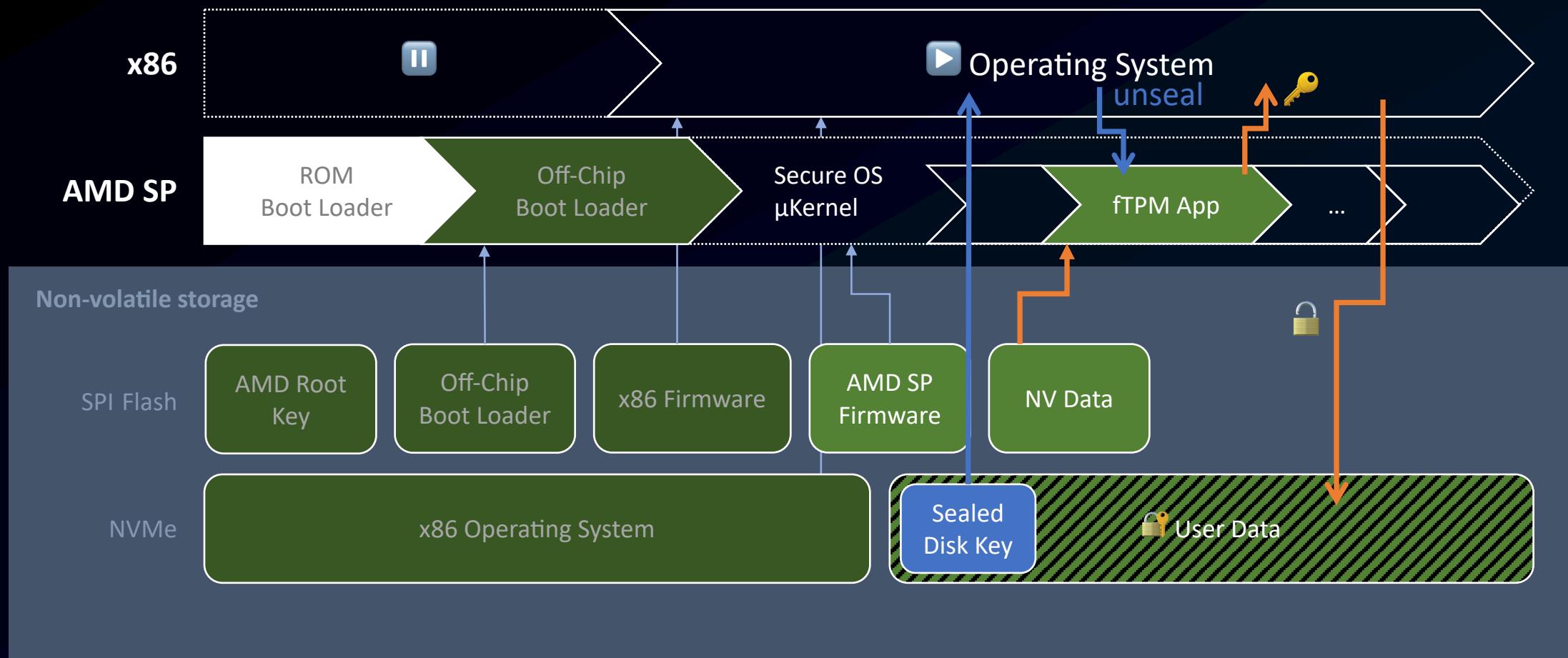
(fTPMs) as an alternative implementation [28], [42]. In contrast to discrete fTPMs, they are implemented through code execution environments (TEE) provided. While dTPMs are still found, fTPMs' omnipresence is still found, affordable and convenient in modern, the ever-increasing integration into cryptographic APIs, e.g., Microsoft Windows [31], for layers and domains.

One of the more promising attacks is Full Disk Encryption (FDE). FDE protects the rest without requiring the user to remember a passphrase. The standard FDE system was booted into the TPM, commonly called the Trusted Platform Module (TPM). Hereby, the system is protected from tampering and configuration changes.

Where in the boot is the fTPM?



Where in the boot is the fTPM?



TPM Objects

- Public Part
 - Metadata
 - Which algorithm (AES, RSA, ECC, ...)
 - When and how can the object be used (policy)
 - Public key (if asymmetric algo.)
- Private Part
 - (Private) key
 - Auth value (for user input policy)
 - Seed value
 - Encrypted, integrity-protected

TPM Object

Public Part

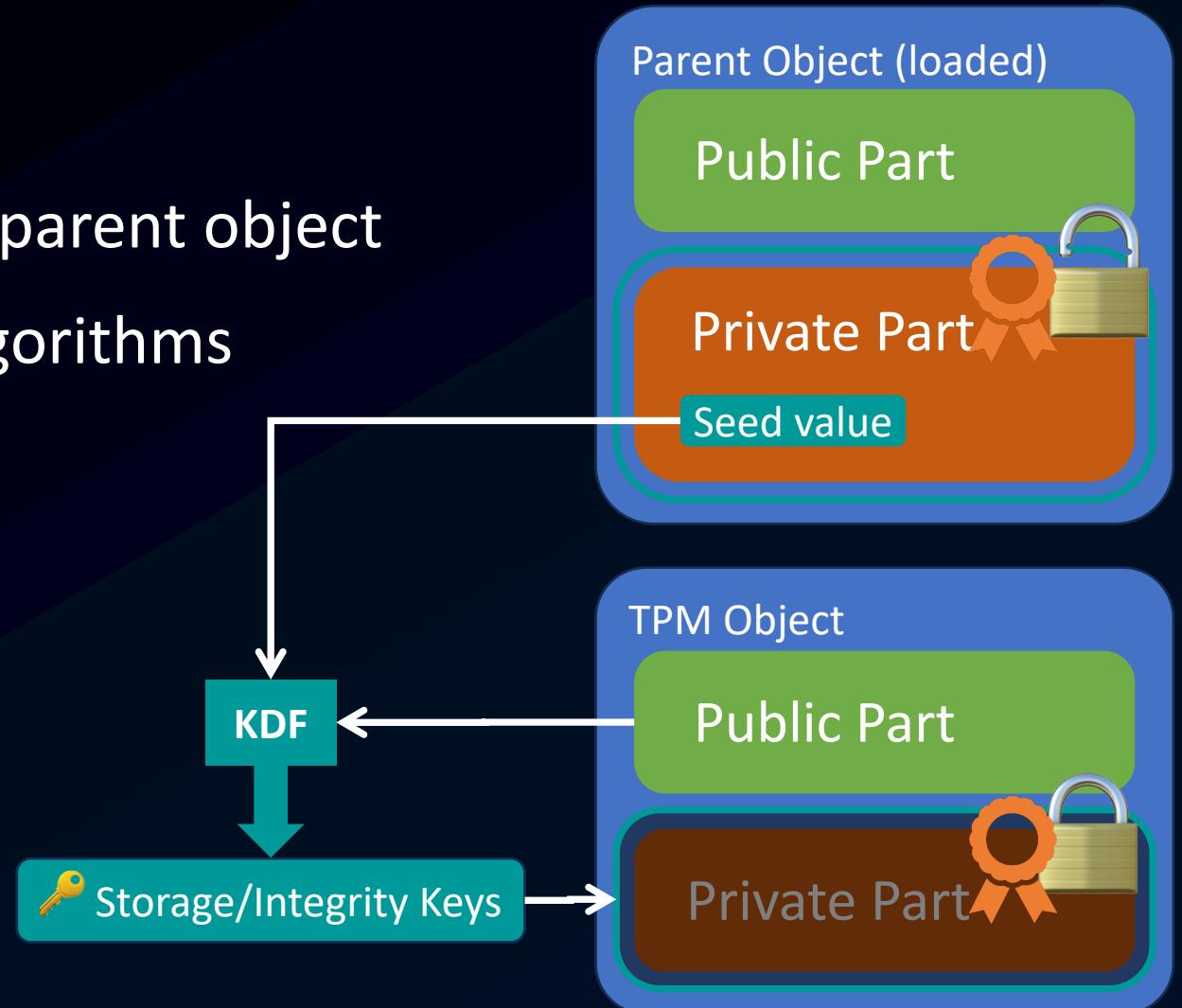
algorithm: RSA
usage: sign=with pin
en/decrypt=never
copy=never
public key: c28e f334 c9...

Private Part

private key: 3175 4088 06...
auth value: hash(PIN 1, 2, 3, 4)
seed value: adf9 8dd3 0e...

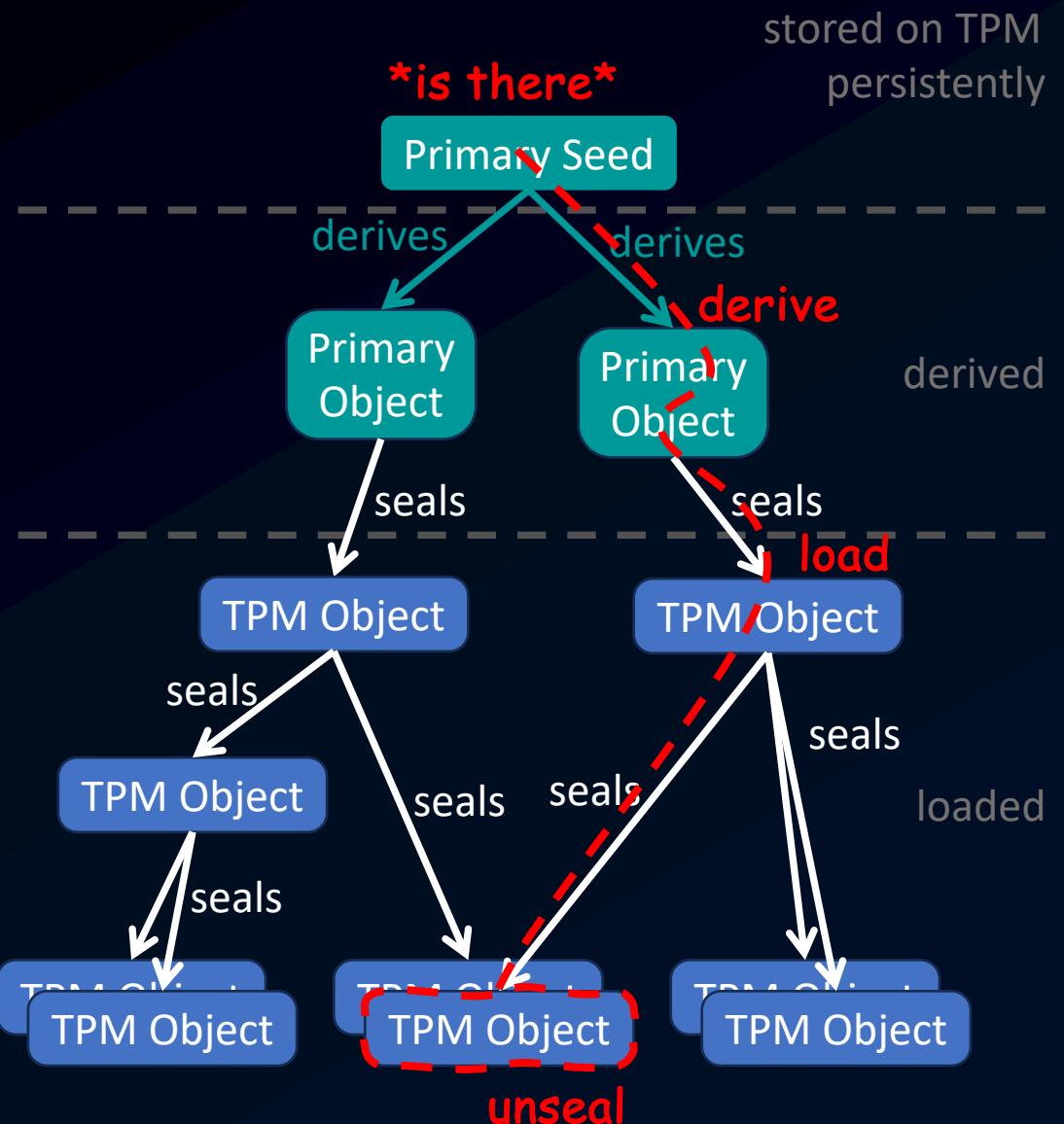
TPM Object Sealing

- Objects are sealed using a parent object
- TPM Spec. gives sealing algorithms



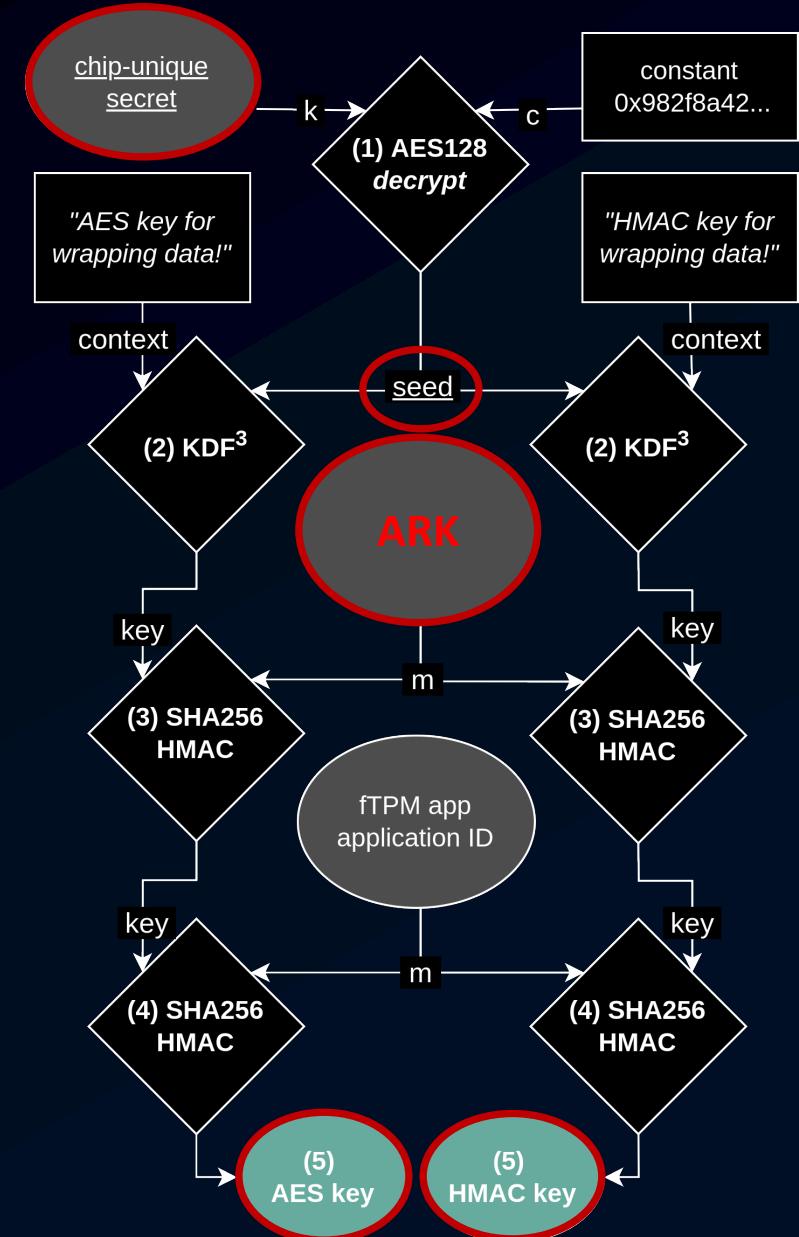
TPM Object Hierarchies

- TPM objects form a forest (multiple trees)
- Roots: Primary objects
 - Derived from one of three primary seeds
- Need to walk hierarchy to unseal/load object

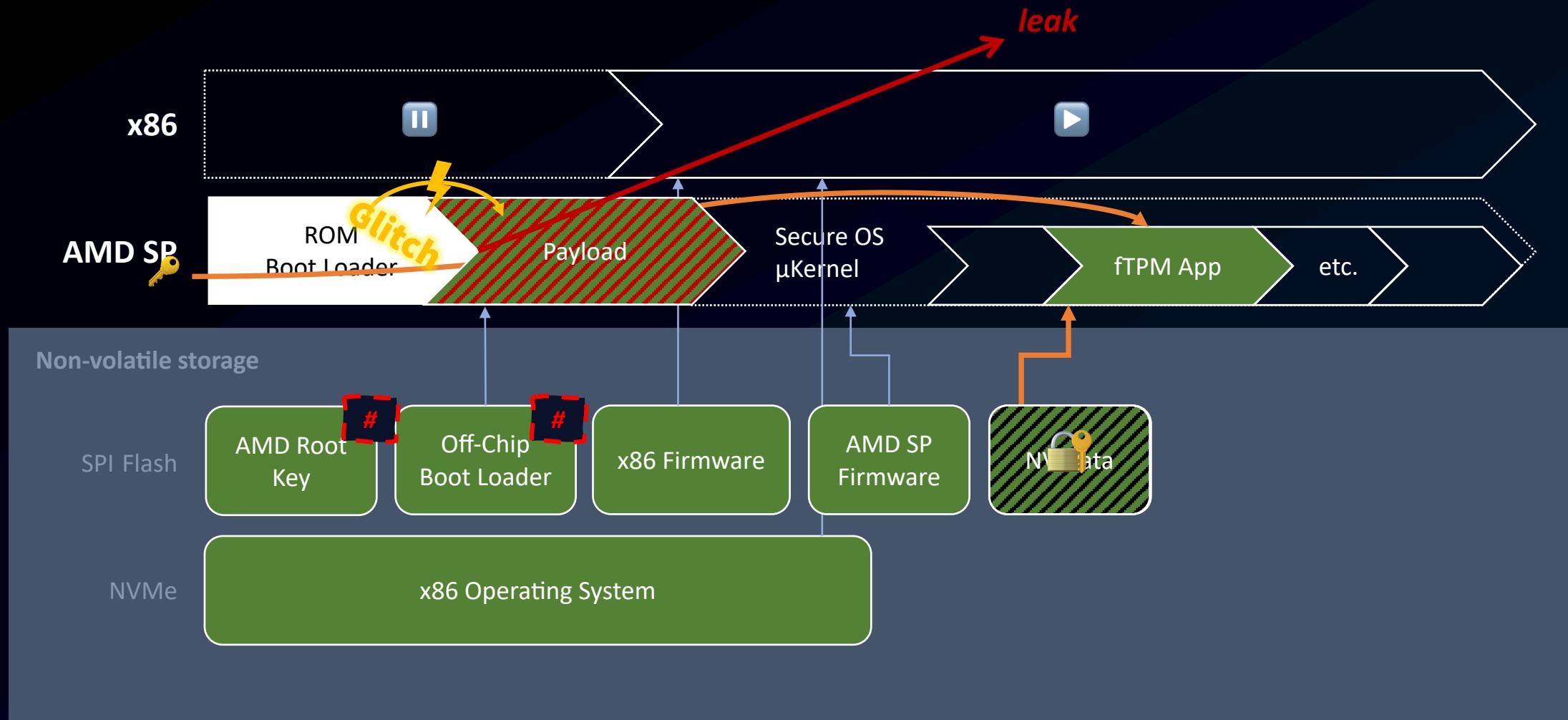


The Non-Volatile fTPM Data

- On SPI flash chip
 - Primary seeds, persistent counters, etc.
- Encrypted and integrity-protected
- We reverse-engineered the key derivation
- **Chip-unique secret** locked in CCP storage
 - Can only be used as AES key
- But we can extract intermediate value

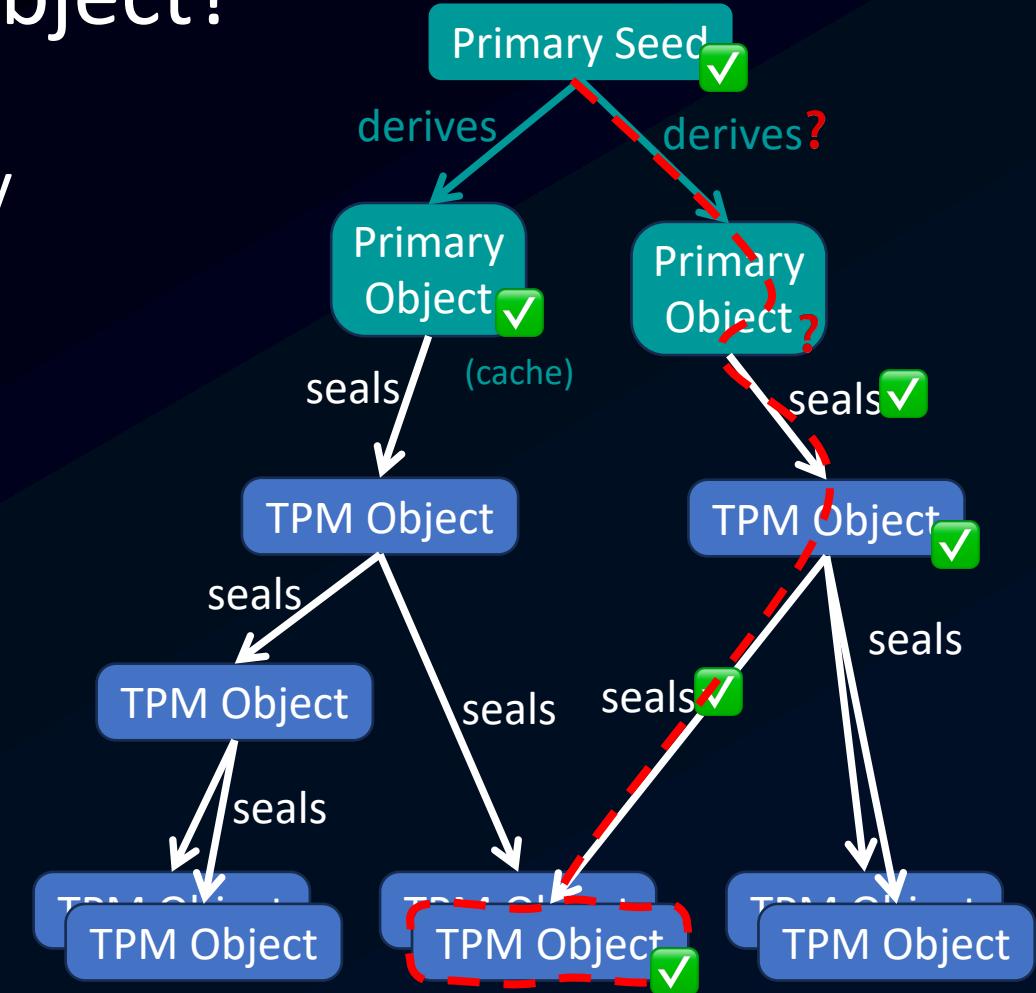


Where in the boot is the fTPM?



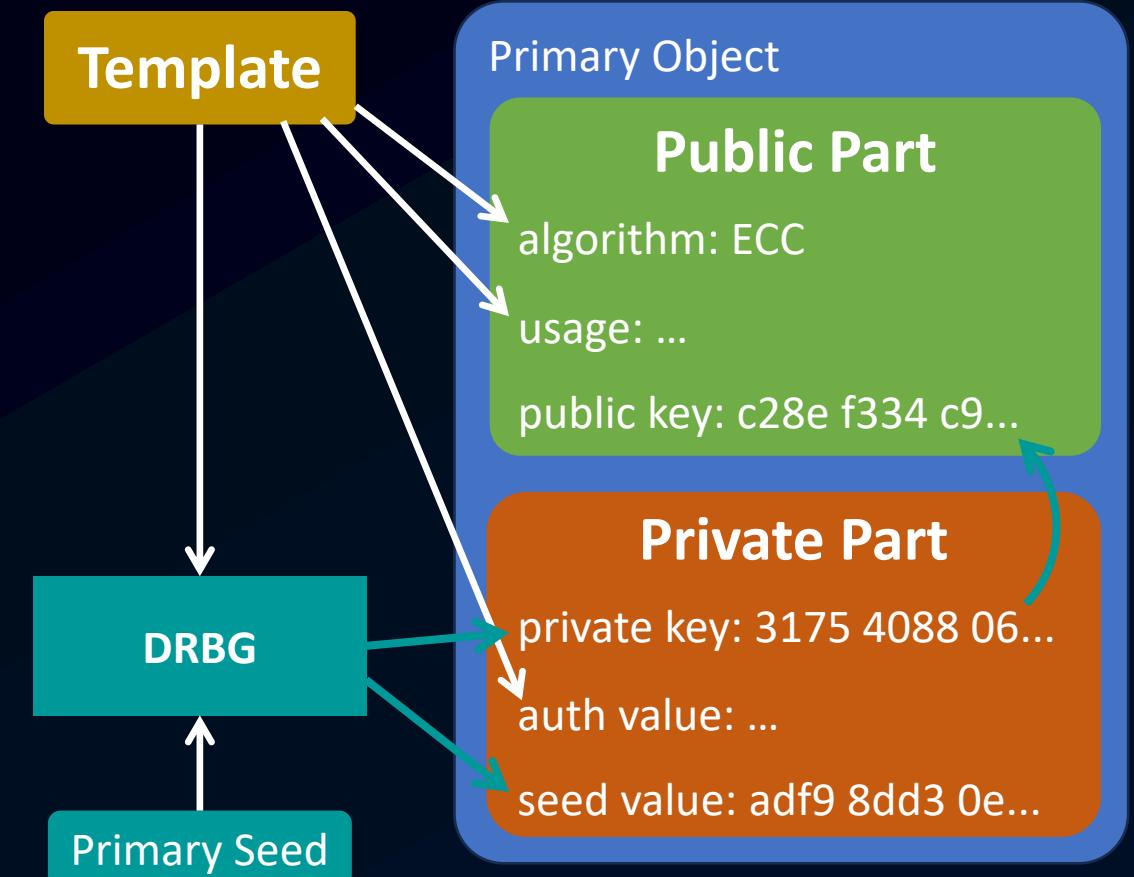
How do we unseal a TPM Object?

- TPM objects are stored externally
- Sealing is defined in TPM spec.
- Primary objects:
 - Some are cached in NV data (see faultTPM)
 - Seeds should be in NV data
 - Derivation only loosely specified!

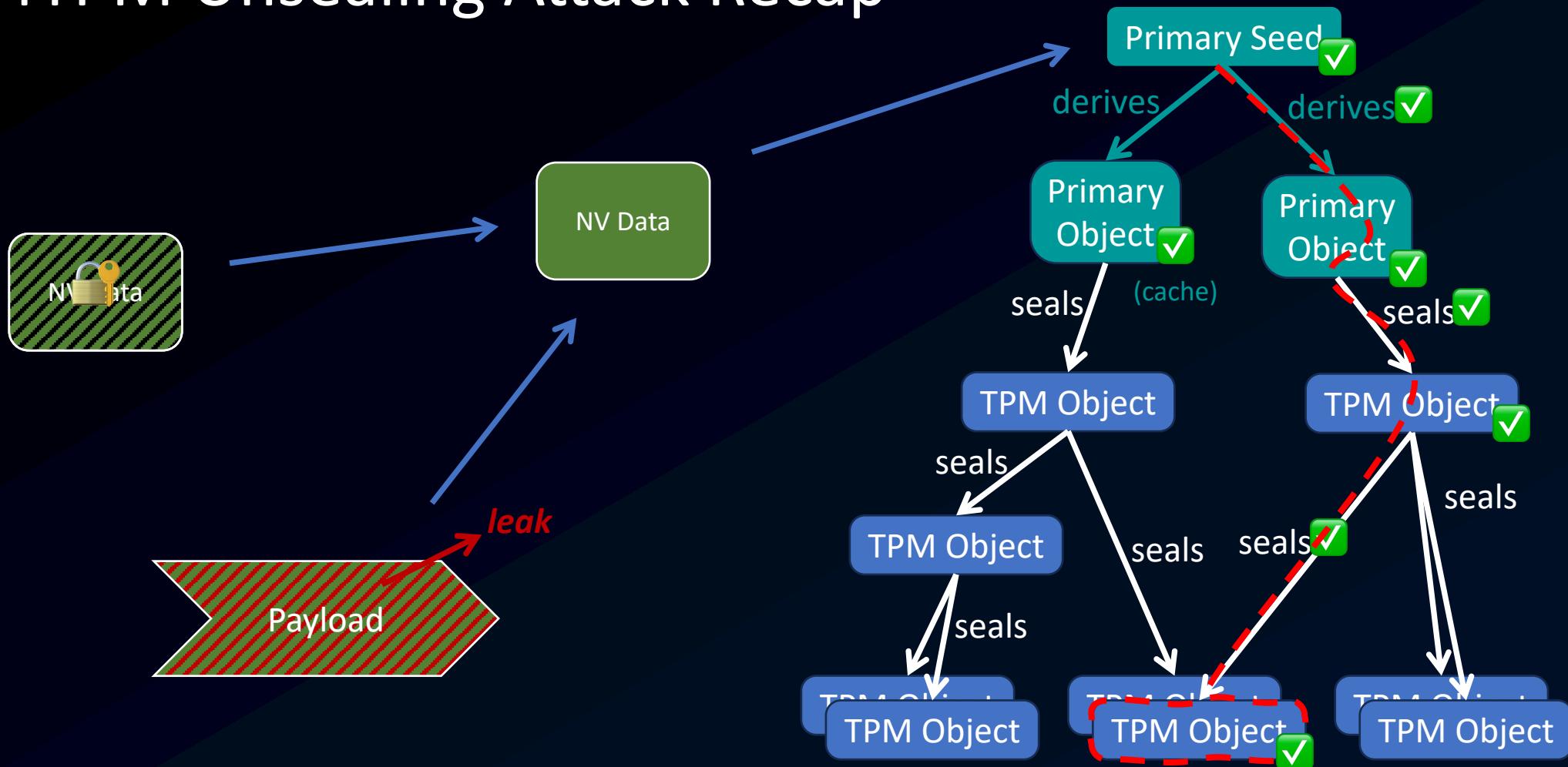


Primary Object Derivation

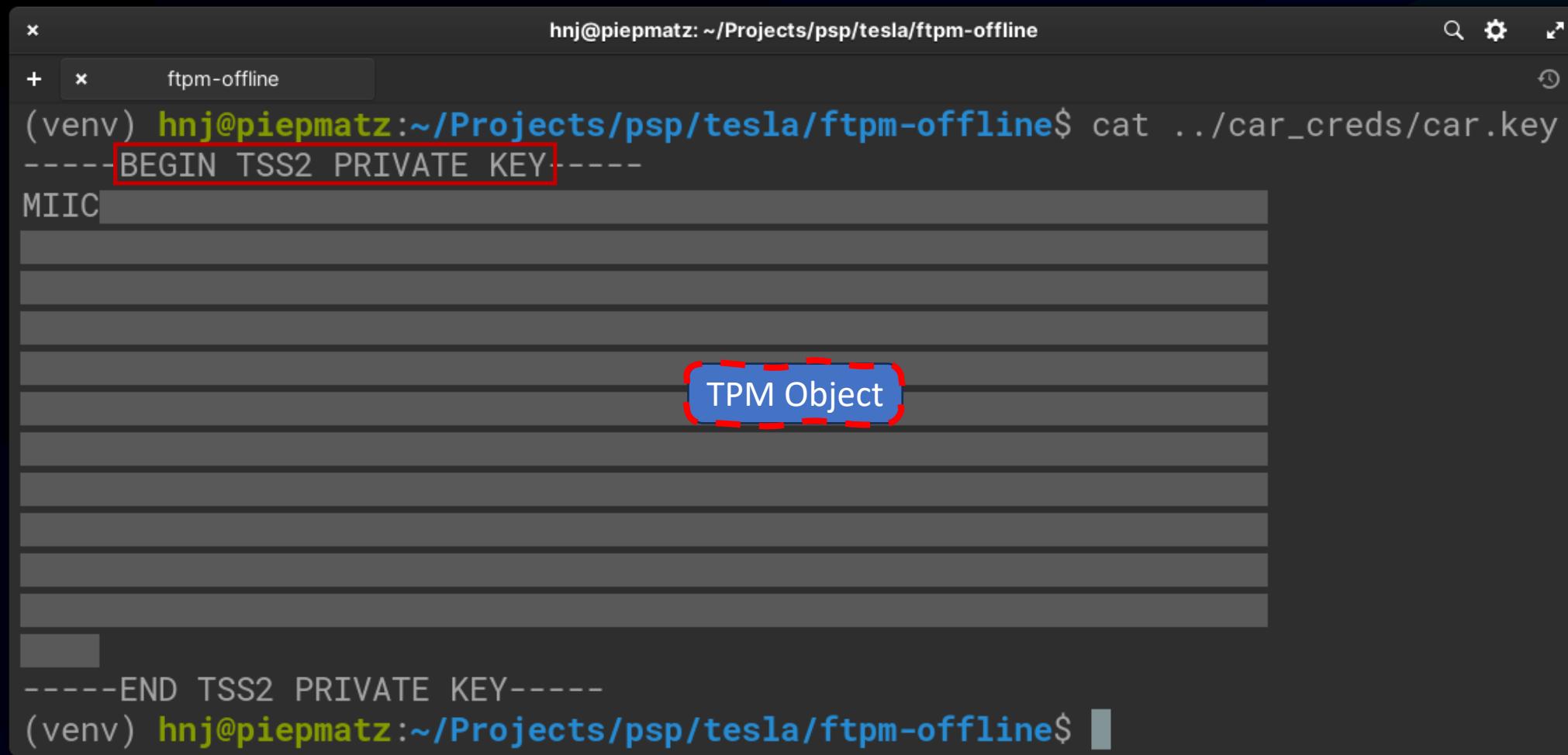
- Most fields come from the input “template”
 - Metadata, Authorization, ...
- Other fields are derived from a **deterministic random bit generator (DRBG)**
 - Seeded with template and seed
 - **Algorithm not specified by spec.**
→ reverse engineering



fTPM Unsealing Attack Recap

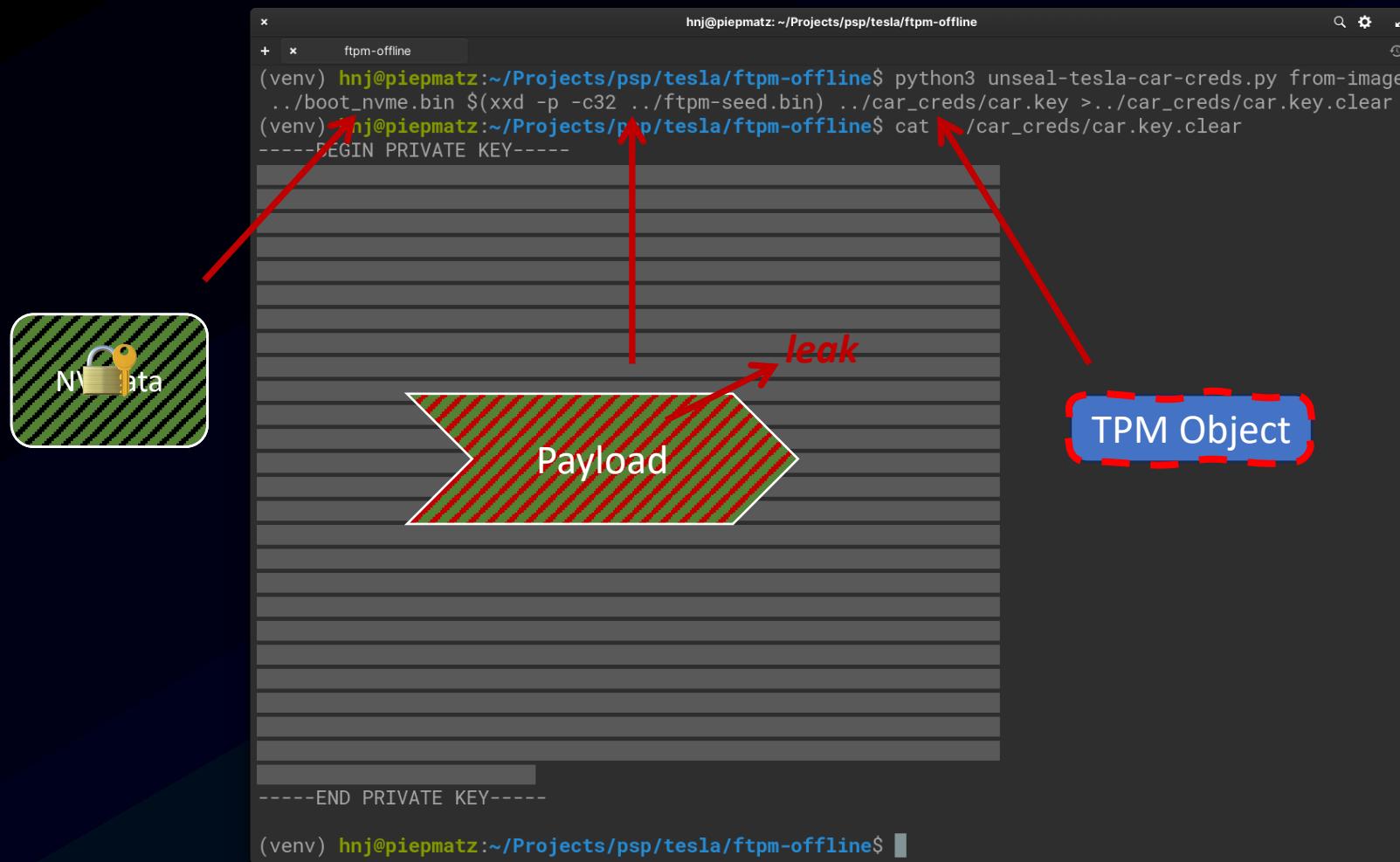


Finding the Car Credentials



```
hnj@piepmatz: ~/Projects/psp/tesla/ftpm-offline
+ *      ftpm-offline
(venv) hnj@piepmatz:~/Projects/psp/tesla/ftpm-offline$ cat ../car_creds/car.key
-----BEGIN TSS2 PRIVATE KEY-----
MIIC
[REDACTED LINES]
TPM Object
[REDACTED LINES]
-----END TSS2 PRIVATE KEY-----
(venv) hnj@piepmatz:~/Projects/psp/tesla/ftpm-offline$
```

Unsealing the Car Credentials



```
hnj@piepmatz:~/Projects/psp/tesla/ftpm-offline
+ x ftpm-offline
(venv) hnj@piepmatz:~/Projects/psp/tesla/ftpm-offline$ echo -e "GET /mothership/vehicles/ [REDACTED] / HTTP/1.0\r\n"
| openssl s_client -connect api-prd.vn.tesla.services:443 -cert ../car_creds/car.crt -verify_quiet -quiet -ign_eof -nocommands -key ../car_creds/car.key.clear
depth=0 CN = api-prd.vn.tesla.services, OU = Tesla Motors, O = Tesla, L = Palo Alto, ST = California, C = US
verify error:num=20:unable to get local issuer certificate
depth=0 CN = api-prd.vn.tesla.services, OU = Tesla Motors, O = Tesla, L = Palo Alto, ST = California, C = US
verify error:num=21:unable to verify the first certificate
HTTP/1.1 200 OK
Date: Wed, 26 Jul 2023 [REDACTED] GMT
Content-Type: application/json; charset=utf-8
Connection: close
Cache-Control: no-cache
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 1; mode=block
X-Content-Type-Options: nosniff
X-Download-Options: noopen
X-Permitted-Cross-Domain-Policies: none
Referrer-Policy: strict-origin-when-cross-origin
X-TXID: [REDACTED]
ETag: [REDACTED]
Cache-Control: max-age=0, private, must-revalidate
X-Request-Id: [REDACTED]
X-Runtime: [REDACTED]
Strict-Transport-Security: max-age=31536000; includeSubDomains
Content-Security-Policy: default-src 'none'

{"id": [REDACTED], "vin": "[REDACTED]", "nickname": "[REDACTED]", "last_seen": [REDACTED], "created_at": [REDACTED], "current_version": "develop/2023.20. [REDACTED]", "current_version_time": null, "active": true, "cell_number": null, "country": "US", "backseat_token": null, "backseat_token_updated_at": null, "radio_config": null, "service_possession": false, "hermes_capable": true, "factory_gated": true, "delivered": true, "model": "3", "use_country": null, "service_state": null, "connection_id": null, "connection_region": "aws:us-west-2", "birthplace": "fremont-factory", "do_not_disturb_until": null, "device_type": "vehicle", "is_customer": true, "state": "asleep", "odin_grablogs": false, "type": "Vehicle"}
(venv) hnj@piepmatz:~/Projects/psp/tesla/ftpm-offline$
```

Using the Car Credentials

Extracting the Disk Encryption Keys

The screenshot shows a terminal window with several tabs and a command-line interface. The terminal is running on a system named 'deploy@psp-deploy'. The user is performing a task related to 'ftp-offline' and 'luksOpen'.

Key parts of the terminal output include:

- Extracting keys from a Luks header:

```
[root@fatbox3 ~]# cryptsetup -v luksOpen --header /tmp/m3/var.luks /tmp/m3/var m3-var --key-file /tmp/m3/var.key
No usable token is available.
```

- Unlocking a key slot:

```
Key slot 0 unlocked.
Command successful.
```

- Opening another Luks header:

```
[root@fatbox3 ~]# cryptsetup -v luksOpen --header /tmp/m3/home.luks /tmp/m3/home m3-home --key-file /tmp/m3/home.key
No usable token is available.
```

- Mounting decrypted partitions:

```
Key slot 0 unlocked.
Command successful.
[root@fatbox3 ~]# blkid /dev/mapper/m3-home
/dev/mapper/m3-home: LABEL="Home" UUID="XXXXXXXXXXXXXX" BLCK_SIZE="4096" TYPE="ext4"
[root@fatbox3 ~]# mount /dev/mapper/m3-home /mnt/home
[root@fatbox3 ~]# mount /dev/mapper/m3-var /mnt/var
[root@fatbox3 ~]# cat /mnt/var/vin
cat: /mnt/var/vin: No such file or directory
[root@fatbox3 ~]# cat /mnt/var/etc/vin
```

- Extracting data from SQLite databases:

```
[root@fatbox3 ~]# sqlite3 /mnt/home/tesla/.Tesla/data/PhonebookV2.db "select * from vcards limit 15"
20971|1||My Number|16+1 |||||||| ||;;|0
20974|4||Alice| 91||||||| ||;;|0
```

A red box highlights the text 'TPM Object' in the bottom left corner of the terminal window.

Outline

1

Analyzing Boot and Firmware Security

2

Hotwiring the Infotainment system

3

Extracting Secrets from the Tesla

Summary

1. We reverse-engineered Tesla's boot security
 - Tesla sets a good example of how it should be done
2. We still rooted the system through voltage glitching
 - This allows to activate some soft-locked features without paying
 - Not software-patchable by anyone
3. We extracted hardware-bound secrets from the TPM using the same attack
 - This can ease independent repairs

Key Takeaways

1. Soft-locking hardware features increases hacking incentives
2. Using battle-tested open-source software like Coreboot and Linux provides a good level of software security
3. But: Consider *hardware* attacks in your threat model, too

Responsible Disclosure(s)

- 2021: Informed AMD about voltage glitching susceptibility
- 2022: Shared faulTPM attack with AMD (based on glitching)
- 2023: Informed Tesla about “AMD jailbreak”
 - Tesla was ‘relieved’ that a single glitch did not yield persistence
 - Did not comment the car_creds extraction

Jailbreaking an Electric Vehicle in 2023

WHAT IT MEANS TO HOTWIRE TESLA'S X86-BASED SEAT HEATER

Christian Werling



Niclas Kühnapfel



Hans Niklas Jacob



Oleg Drokin



cwerling@sect.tu-berlin.de

kuehnafel@tu-berlin.de

hnj@sect.tu-berlin.de

drokin@linuxhacker.ru



All code available at:
github.com/PSPReverse