Is there an EFI monster inside your apple?

fG! @ CODE BLUE 2015
Who am I?

- An Economist.
- Who loves Human Behavior.
- And politics.
- Oh, and a bit of computers.
Thank you!

Translators!
What's Up Doc?
EFI Monsters?

- Introduction to EFI.
- How to
  - Reverse engineer (U)EFI binaries.
  - Search for (U)EFI rootkits.
ASSUMPTIONS

"Relax! I know this road perfectly! I've been driving it all my life!"
Assumptions

- Reference machine
  - MacBook Pro Retina 10.1.
- 64-bit only OS X versions.
- Sandy Bridge or newer.
Why EFI?

In this corner, we have firewalls, encryption, antivirus software, etc. And in this corner, we have Dave!!
Why EFI?

- BIOS replacement.
- Initially developed by Intel.
- Now UEFI, managed by UEFI consortium.
  - http://www.uefi.org
Why EFI?

- Initializes your machine.
- Access to low level features.
- Modular.
- Feature rich.
- Rather easy development in C.
What evil things can we do?
What evil things can we do?

- **Diskless kernel/userland rootkits**
- Rootkit data stored in the flash chip.
- Unpack and patch kernel on boot.
- RAM only, never touch hard-disk.
- Check Snare’s SyScan 2012 presentation.
What evil things can we do?

- Can be hard to detect.
- With regular available tools.
- And with some anti-forensics.
- For example anti-memory dumping.
What evil things can we do?

- Persistence across operating system installs
- HackingTeam built a UEFI rootkit.
  - https://github.com/hackedteam/vector-edk
  - https://github.com/informationextraction/vector-edk/blob/master/MdeModulePkg/Application/fsbg/fsbg.c
What evil things can we do?

- Attack full-disk encryption
- Install a keylogger.
- Recover FileVault2 password.
What evil things can we do?

- Attack “secure” operating systems
- For example, Tails.
- Recover PGP keys and/or passphrases.
- https://www.youtube.com/watch?v=sNYsfUNegEA.
What evil things can we do?

- Bootloader
  - Redirect to a custom bootloader.
- SMM backdoors
TL;DR
OWN EVERYTHING!
Once upon a time...
there was a...
a zero day!
Cyber-Safe

Mac attack! Nasty bug lets hackers into Apple computers

By Jose Pagliery  @Jose_Pagliery

Mac bug makes rootkit injection as easy as falling asleep
Apple hacker reveals cracker 0day rootkit whacker

Security
Related topics
Apple, Security
A zero day story...

- Firmware related zero day.
- Disclosed a few months ago.
  - https://reverse.put.as/2015/05/29/the-empire-strikes-back-apple-how-your-mac-firmware-security-is-completely-broken/
A zero day story...

- Failure to lock the flash.
- Write to the flash from userland.
- Similar to Thunderstrike but better.
- Thunderstrike requires physical access.
- Prince Harming allows remote attack.
- Hardware-specific, but it’s always there
- Can modify everything
  - SEC, PEI, DXE, BDS, custom drivers, whatever
- Can be written to from the OS
- So awesome. 11/10 A++++ would buy again.
A zero day story...

- Extremely simple to trigger.
- Put machine to sleep.
  - Close, wait for fans to stop, and reopen.
  - Or force sleep with "pmset sleepnow".
A zero day story...

- Sandy Bridge and Ivy Bridge Macs are vulnerable.
- Haswell or newer are not.
- All older machines are vulnerable
  - Core 2 Duo or older.
  - No flash protections at all.
## A zero day story...

- **Available updates:**

<table>
<thead>
<tr>
<th>MacBook Air</th>
<th>MacBook Pro</th>
<th>Mac Mini</th>
<th>Mac Pro</th>
<th>iMac</th>
</tr>
</thead>
<tbody>
<tr>
<td>4,1</td>
<td>8,1</td>
<td>5,1</td>
<td>6,1</td>
<td>12,1</td>
</tr>
<tr>
<td>5,1</td>
<td>9,1</td>
<td>6,1</td>
<td></td>
<td>13,1</td>
</tr>
<tr>
<td>6,1</td>
<td>10,1</td>
<td>7,1</td>
<td></td>
<td>14,1</td>
</tr>
<tr>
<td>7,1</td>
<td>10,2</td>
<td></td>
<td></td>
<td>14,2</td>
</tr>
<tr>
<td></td>
<td>11,1</td>
<td></td>
<td></td>
<td>14,3</td>
</tr>
<tr>
<td></td>
<td>11,2</td>
<td></td>
<td></td>
<td>14,4</td>
</tr>
<tr>
<td></td>
<td>11,4</td>
<td></td>
<td></td>
<td>15,1</td>
</tr>
</tbody>
</table>
A zero day story...

- Reversing and understanding the vulnerability.
  - https://reverse.put.as/2015/07/01/reversing-prince-harmings-kiss-of-death/
- Contains links to relevant EFI documentation.
A zero day story...

- Venamis aka Dark Jedi was also patched.
  - http://events.ccc.de/congress/2014/Fahrplan/events/6129.html

- Slightly more complex, same results.
A zero day story...

- The story doesn’t end here.
- Check ThunderStrike 2 slides.
- Other unpatched vulnerabilities.
- Can be exploited with remote attack vectors.
# Old bugs, new platforms

<table>
<thead>
<tr>
<th>Vulnerability</th>
<th>Private disclosure</th>
<th>Status on OSX</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Snorlax/PrinceHarming</strong></td>
<td>August 2013</td>
<td><strong>Patched June 2015</strong></td>
</tr>
<tr>
<td>VU #577140</td>
<td>July 2015 / May 2015</td>
<td></td>
</tr>
<tr>
<td><strong>Darth Venamis</strong></td>
<td>Sept 2014</td>
<td><strong>Partial Patch June 2015</strong></td>
</tr>
<tr>
<td>VU #976132</td>
<td>Dec 2014</td>
<td></td>
</tr>
<tr>
<td><strong>SpeedRacer/BIOS_CTNL</strong></td>
<td>Dec 2013</td>
<td><strong>Vulnerable</strong></td>
</tr>
<tr>
<td>VU #766164</td>
<td>Aug 2014</td>
<td></td>
</tr>
<tr>
<td><strong>King's Gambit</strong></td>
<td>Dec 2013</td>
<td><strong>Vulnerable</strong></td>
</tr>
<tr>
<td>VU #552286</td>
<td>Aug 2014</td>
<td>(See HITB-GSEC 2015)</td>
</tr>
<tr>
<td><strong>The Sicilian</strong></td>
<td>~May 2013</td>
<td><strong>Vulnerable</strong></td>
</tr>
<tr>
<td>VU #255726</td>
<td>Sep 2013</td>
<td></td>
</tr>
<tr>
<td><strong>Setup UEFI Variable</strong></td>
<td>June 2013</td>
<td><strong>Not vulnerable</strong></td>
</tr>
<tr>
<td>VU #758382</td>
<td>Mar 2014</td>
<td></td>
</tr>
</tbody>
</table>
Reminder: This talk has 1 main point

- Apple has not been as responsive, or as accurate, as other PC vendors in responding to industry-wide notifications of firmware vulnerabilities. Consequently Mac users have been left vulnerable to attacks that have been fixed on other x86-based PCs.
Apple...

POTHEADS

POTHEADS EVERYWHERE
Where is EFI?
Where is EFI?

- Usually stored in a CMOS serial flash.
- Two popular chips
  - Macronix MX25L6406E.
  - Micron N25Q064A.
- SPI compatible.
- Most are 64 Mbits/8 Mbytes.
Where is EFI?

- Newer machines flash chip(s)
  - Winbond W25Q64FV.

- Chip list from EfiFlasher.elf:

<table>
<thead>
<tr>
<th>SST 25VF080</th>
<th>Macronix 25L1605</th>
<th>ST Micro M25P16</th>
<th>WinBond 25X32</th>
</tr>
</thead>
<tbody>
<tr>
<td>SST 25VF016</td>
<td>Macronix 25L3205</td>
<td>ST Micro M25P32</td>
<td>WinBond 25X64</td>
</tr>
<tr>
<td>SST 25VF032</td>
<td>Macronix 25L6436E</td>
<td>Eon M25P32</td>
<td>WinBond 25X128</td>
</tr>
<tr>
<td>SST 25VF064</td>
<td>Atmel 45DB321</td>
<td>Eon M25P16</td>
<td>Numonyx N25Q064</td>
</tr>
</tbody>
</table>
Where is EFI?

- Most chips are 8 pin SOIC.
- SMD or BGA versions used?
  - Retinas 13”?
  - New MacBook 12”?
Where is EFI?

- You can buy the chips bulk and cheap.
- Useful for flashing experiments.
- Good results from Aliexpress.com.
- Around $14 for 10 N25Q064A.
- Around $8 for 10 MX25L640E.
Where is EFI?

- Easy access on some models.
  - Retinas 15” are the easiest.
- Extensive disassembly required on others.
- Still, a MacBook Pro 8,1 can be disassembled in 5 mins or less.
Retina 10,1
Air 7,2
How to dump EFI
How to dump EFI

- **Hardware**
  - The best and most reliable way.
  - Trustable.

- **Software**
  - Possible if chip supported by flashrom.
  - Not (very) trustable.
Hardware

- Any SPI compatible programmer.
  - [http://flashrom.org/Supported_programmers](http://flashrom.org/Supported_programmers)
- I use Trammell Hudson’s SPI flasher.
  - [https://trmm.net/SPI](https://trmm.net/SPI)
Hardware

- Based on Teensy 2.0 or 3.x.
Hardware

- Easy to build.
- Cheap, ~ $30.
- Fast, dumps a 64Mbit flash in 8 mins.
- The Teensy 3 version is even faster.
- It just works!
Flash chip SPI pinout

(WHITE) CS ----- | o
(BROWN) SO ----- | ---- VCC (RED)
(ORANGE) WP ----- | ---- RST (YELLOW)
(BLACK) GND ----- | ---- SCLK (GREEN)

| ---- SI (BLUE) |
Teensy 2.0 pinout

Teensy 2.0

- (BLACK) GND
- (WHITE) CS
- (GREEN) SCLK
- (BLUE) SI
- (BROWN) SO

USB

---

VCC (RED)

---

VCC (YELLOW/ORANGE)
Teensy 2.0 pinout

- Teensy 2 default voltage is 5v.
- Flash chips are 3.3v.
- Requires voltage regulator MCP1825.
Teensy 3.1 pinout
Tips & Tricks

- Shunt WP and RST pins to VCC.
- Different SPI pins names
  - SCLK, SCK, CLK.
  - MOSI, SIMO, SDO, DO, DOUT, SO, MTSR.
  - MISO, SOMI, SDI, DI, DIN, SI, MRST.
  - SS, nCS, CS, CSB, CSN, nSS, STE, SYNC.
Hardware

- How to read entire flash

```
$ time lrx -X -O </dev/cu.usbmodem12341 >/dev/cu.usbmodem12341 Retina-09-07-2015-Secuinside.bin

lrx: ready to receive Retina-09-07-2015-Secuinside.bin
^C
lrx: caught signal 2; exiting

real 6m58.773s
user 0m0.774s
sys 0m1.726s
```

```
$ ls -la Retina-09-07-2015-Secuinside.bin
-rw------- 1 reverser staff 8388608 Jul 9 16:47 Retina-09-07-2015-Secuinside.bin
```
Hardware

- How to write entire 64MB flash

```plaintext
spi
>Help:
i: print ID
r: read 16 bytes from address - r0<enter>
R: read XX bytes from address - RO 10<enter>
d: dump to console
w: write enable interactive
e: erase sector interactive
u: upload
b: upload bios area only
1: flash first ffs
2: flash second ffs
3: flash third ffs
x: download

u
>0 800000
(exit to shell)
# pv new-efi.bin > /dev/cu.usbmodem12341
```
Hardware

- Linux works best to write the flash.
- Some issues with OS X version.
- pv or serial driver issues?
Software

- Requirements
  - Flashrom
  - DirectHW.kext
- Rwmem by Trammell also works.
- Or readphysmem.
Software

- DarwinDumper.
- Contains binary versions of flashrom and DirectHW.kext.
- Kernel extension is not code signed.
- (Still) Whitelisted by Apple.
Software

- http://flashrom.org/Flashrom
- http://www.coreboot.org/DirectHW
- https://bitbucket.org/blackosx/darwindumper/downloads
- https://github.com/osresearch/rwmem
- https://github.com/gdbinit/readphysmem
sh-3.2# kextload DirectHW.kext/

sh-3.2# ./flashrom -r bios_dump.bin -V -p internal
flashrom v0.9.7-r1711 on Darwin 14.4.0 (x86_64)
flashrom is free software, get the source code at http://www.flashrom.org

flashrom was built with libpci 3.1.7, LLVM Clang 6.0 (clang-600.0.56), little endian
Command line (5 args): ./flashrom -r bios_dump.bin -V -p internal
(...)
Found chipset "Intel HM77" with PCI ID 8086:1e57.
This chipset is marked as untested. If you are using an up-to-date version
of flashrom *and* were (not) able to successfully update your firmware with it,
then please email a report to flashrom@flashrom.org including a verbose (-V) log.
Thank you!
SPI Read Configuration: prefetching disabled, caching enabled, OK.
The following protocols are supported: FWH, SPI.

(....)
Probing for Micron/Numonyx/ST N25Q064..3E, 8192 kB: probe_spi_rdid_generic: id1 0x20, id2 0xba17
Found Micron/Numonyx/ST flash chip "N25Q064..3E" (8192 kB, SPI) at physical address 0xff800000.
Chip status register is 0x00.
Chip status register: Status Register Write Disable (SRWD, SRP, ...) is not set
Chip status register: Block Protect 3 (BP3) is not set
Chip status register: Top/Bottom (TB) is top
Chip status register: Block Protect 2 (BP2) is not set
Chip status register: Block Protect 1 (BP1) is not set
Chip status register: Block Protect 0 (BPO) is not set
Chip status register: Write Enable Latch (WEL) is not set
Chip status register: Write In Progress (WIP/BUSY) is not set
(....)
Found Micron/Numonyx/ST flash chip "N25Q064..3E" (8192 kB, SPI). This chip may contain one-time programmable memory. flashrom cannot read and may never be able to write it, hence it may not be able to completely clone the contents of this chip (see man page for details). Reading flash... done.

Restoring MMIO space at 0x10ae098a0
Restoring PCI config space for 00:1f:0 reg 0xd

```
sh-3.2# ls -la bios_dump.bin
-rw-r--r-- 1 root staff 8388608 Jul 8 01:23 bios_dump.bin
```
Software

- AppleHWAccess.kext.
- readphysmem utility.
- Can read bios without external kext.
- Default on Mavericks and Yosemite.
- Not anymore on El Capitan.
Software

- Good enough to play around.
- Mostly useless to chase (U)EFI rootkits.
- Unless it is made by HackingTeam.
  - Their version makes no attempt to hide itself from software dumps.
What is in the flash?
What’s in the flash

Descriptor Region

[0x1000]---

Intel Management Engine (ME) Region

[0x19000]---

BIOS Region

<- a big black box

<- our playground!
What’s in the flash

- Descriptor Region
- GbE Region
- ME Region
- Platform Data Region
- BIOS Region
## What's in the flash

### Structure

<table>
<thead>
<tr>
<th>Name</th>
<th>Image</th>
<th>Type</th>
<th>Subtype</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel image</td>
<td>Intel</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Descriptor region</td>
<td>Region</td>
<td>Descriptor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ME/TXE region</td>
<td>Region</td>
<td></td>
<td>ME/TXE</td>
<td></td>
</tr>
<tr>
<td>BIOS region</td>
<td>Region</td>
<td></td>
<td>BIOS</td>
<td></td>
</tr>
<tr>
<td>7A935409-8468-444A-B1CE-0BF617D8900F</td>
<td>Volume</td>
<td>FFSv2</td>
<td>AppleCRC32 AppleFSO</td>
<td></td>
</tr>
<tr>
<td>7A935409-8468-444A-B1CE-0BF617D8900F</td>
<td>Volume</td>
<td>FFSv2</td>
<td>AppleCRC32 AppleFSO</td>
<td></td>
</tr>
<tr>
<td>7A935409-8468-444A-B1CE-0BF617D8900F</td>
<td>Volume</td>
<td>FFSv2</td>
<td>AppleCRC32 AppleFSO</td>
<td></td>
</tr>
<tr>
<td>E3898A09-5FE3-48E5-9982-2798385A9027</td>
<td>Volume</td>
<td>Unknown</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7A935409-8468-444A-B1CE-0BF617D8900F</td>
<td>Volume</td>
<td>FFSv2</td>
<td>AppleCRC32 AppleFSO</td>
<td></td>
</tr>
<tr>
<td>7A935409-8468-444A-B1CE-0BF617D8900F</td>
<td>Volume</td>
<td>FFSv2</td>
<td>AppleCRC32 AppleFSO</td>
<td></td>
</tr>
<tr>
<td>7A935409-8468-444A-B1CE-0BF617D8900F</td>
<td>Volume</td>
<td>FFSv2</td>
<td>AppleCRC32 AppleFSO</td>
<td></td>
</tr>
<tr>
<td>15302197-29B0-44DC-AC59-887F70E41A6B</td>
<td>Volume</td>
<td>Unknown</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15302197-29B0-44DC-AC59-887F70E41A6B</td>
<td>Volume</td>
<td>Unknown</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FFF128B0-7696-4CBB-A9B5-2747075B4F50</td>
<td>Volume</td>
<td>Unknown</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7A935409-8468-444A-B1CE-0BF617D8900F</td>
<td>Volume</td>
<td>FFSv2</td>
<td>AppleCRC32 AppleFSO</td>
<td></td>
</tr>
<tr>
<td>7A935409-8468-444A-B1CE-0BF617D8900F</td>
<td>Volume</td>
<td>FFSv2</td>
<td>AppleCRC32 AppleFSO</td>
<td></td>
</tr>
<tr>
<td>84ADEEAD-61FF-4D31-B6BA-64F8BF901F5A</td>
<td>Volume</td>
<td>FFSv2</td>
<td>AppleCRC32 AppleFSO</td>
<td></td>
</tr>
<tr>
<td>84ADEEAD-61FF-4D31-B6BA-64F8BF901F5A</td>
<td>Volume</td>
<td>FFSv2</td>
<td>AppleCRC32 AppleFSO</td>
<td></td>
</tr>
</tbody>
</table>

### Messages

```
parseVolume: unknown file system E3898A09-5FE3-48E5-9982-2798385A9027
parseVolume: unknown file system 15302197-29B0-44DC-AC59-887F70E41A6B
parseVolume: unknown file system 15302197-29B0-44DC-AC59-887F70E41A6B
parseVolume: unknown file system FFF128B0-7696-4CBB-A9B5-2747075B4F50
```

Opened: Retina-08-07-2015-after-SyScan-dump-and EFI-update-09.bin
What’s in the flash

Opened: bios_dump.bin

Message:
parseVolume: unknown file system FFF12800-7696-4C88-A985-2747075B4F50
parseVolume: non-UEFI data found in volume's free space
Descriptor region

- Location of other regions.
- Access permissions.
  - OS/BIOS shouldn’t access ME region.
- VSCC configures ME flash access.
Intel ME region

- A CPU inside your CPU 😊.
- Runs Java.
- Can be active with system powered off.
- Out of band network access!
- No access from BIOS and OS.
Intel ME region

- Mostly a blackbox.
- Three presentations by Igor Skochinsky.
- Definitely requires more research!
- Unpacker
  - http://io.smashthestack.org/me/
Intel ME region

- Rootkit in your laptop: Hidden code in your chipset and how to discover what exactly it does
- Intel ME Secrets
- Intel ME: Two years later
- https://github.com/skochinsky/papers
BIOS region

- Contains
  - EFI binaries for different phases.
  - NVRAM.
  - Microcode (not for some models).

- Each on its own firmware volume (FVH).
BIOS region

- Everything is labeled with a GUID.
- No filenames.
- Many GUID can be found in EFI specs.
- Others are vendor specific/private.
- Google and luck are your friends!
GUIDs found in the AMI source

See the following URL for more info and the latest version:
https://github.com/snarez/idaf-efiutils

```
GUIDs = {
    'ACOUSTIC_SETUP_PROTOCOL_GUID': [0xc1d7859d, 0x5719, 0x46c3, 0xa2, 0x98, 0xd0, 0x71, 0xe3, 0x2, 0x64, 0xd1],
    'ADD_BOOT_OPTION_GUID': [0x19d96d3f, 0x6a6a, 0x47d2, 0xb1, 0x95, 0x7b, 0x24, 0x32, 0xda, 0x3b, 0xe2],
    'ADVANCED_FORM_SET_GUID': [0xe14f04fa, 0x8706, 0x4353, 0x92, 0xf2, 0x9c, 0x24, 0x24, 0x74, 0x6f, 0x9f],
    'ACI_BUS_INIT_PROTOCOL_GUID': [0xb2fa4764, 0x3b6f, 0x43d3, 0x91, 0xdf, 0x87, 0xd1, 0x5a, 0x3e, 0x56, 0x68],
    'AHCI_SMM_PROTOCOL_GUID': [0xb2fa5764, 0x3b6f, 0x43d3, 0x91, 0xdf, 0x87, 0xd1, 0x5a, 0x3e, 0x56, 0x68],
    'AMICSM_PCIBUSNUM_XLAT_PROTOCOL_GUID': [0xc5b55c4c0, 0x23d0, 0x43db, 0x92, 0x2c, 0x72, 0x3f, 0x8c, 0x91, 0x5c],
    'AMITSETUP_GUID': [0xc811fa38, 0x42c8, 0x4579, 0xa9, 0xb8, 0x60, 0xe9, 0xe4, 0xdd, 0xfb, 0x34],
    'AMITSE_ADMIN_PASSWORD_VALID_GUID': [0x541d5a75, 0x95ee, 0x43c7, 0x9e, 0x5d, 0x23, 0xb9, 0xdc, 0x48, 0x62, 0x49],
    'AMITSE_AFTER_FIRST_BOOT_OPTION_GUID': [0xc48d61c, 0x9d8e, 0x4ace7, 0xad, 0x39, 0xed, 0xd1, 0xab, 0xb3, 0xb8, 0x30],
    'AMITSE_BOOT_ORDER_CHANGE_GUID': [0xb680c809, 0x4937, 0x93, 0xf1, 0xe5, 0x86, 0x22, 0xfe, 0x50],
    'AMITSE_DRIVER_HEALTH_CTRL_GUID': [0x58279c2d, 0xfb19, 0x466e, 0xb4, 0xe2, 0xcd, 0x43, 0x70, 0x16, 0xdc, 0x25],
}````
PI Boot Phases

Security (SEC) | Pre EFI Initialization (PEI) | Driver Execution Environment (DXE) | Boot Dev Select (BDS) | Transient System Load (TSL) | Run Time (RT) | After Life (AL)

Power on → [ . . . Platform initialization . . . ] → [ . . . OS boot . . . ] → Shutdown
EFI Boot Phases

- Different initialization phases.
- Make resources available to next phase.
- Memory for example.
The PEI/DXE Dispatchers

- PEI and DXE phases have a dispatcher.
- Guarantees dependencies and load order.
- Dependency expressions.
- Available as a section.
<table>
<thead>
<tr>
<th>Name</th>
<th>Action</th>
<th>Type</th>
<th>Subtype</th>
</tr>
</thead>
<tbody>
<tr>
<td>7A9354D9-0468-444A-81CE-0BF617D890DF</td>
<td>Volume</td>
<td>FFSv2</td>
<td></td>
</tr>
<tr>
<td>52C05B14-089B-496C-BC3B-04B50211D680</td>
<td>File</td>
<td>PEI core</td>
<td></td>
</tr>
<tr>
<td>7CA23D91-9C13-4679-A2B7-9DCE98734A2</td>
<td>File</td>
<td>PEI module</td>
<td></td>
</tr>
<tr>
<td>38317FC0-2795-4DE6-B207-6B0CA768CFB1</td>
<td>File</td>
<td>PEI module</td>
<td></td>
</tr>
<tr>
<td>PEI dependency section</td>
<td>Section</td>
<td>PEI dependency</td>
<td></td>
</tr>
<tr>
<td>TE image section</td>
<td>Section</td>
<td>TE image</td>
<td></td>
</tr>
<tr>
<td>34C8CB28F-B61C-45A2-BF2E-89E468BEC63B</td>
<td>File</td>
<td>PEI module</td>
<td></td>
</tr>
<tr>
<td>PEI dependency section</td>
<td>Section</td>
<td>PEI dependency</td>
<td></td>
</tr>
<tr>
<td>TE image section</td>
<td>Section</td>
<td>TE image</td>
<td></td>
</tr>
<tr>
<td>80F1DE13-3C6E-4A78-A082-1AC5FF3750FB</td>
<td>File</td>
<td>PEI module</td>
<td></td>
</tr>
<tr>
<td>BAC57518-0934-423D-BB39-F5FC88840CCF</td>
<td>File</td>
<td>PEI module</td>
<td></td>
</tr>
<tr>
<td>6A890B44-D0DB-5AA8-A301-53FA273E2FD6</td>
<td>File</td>
<td>PEI module</td>
<td></td>
</tr>
<tr>
<td>D072670B-DC2C-4768-B182-99B4A9E5FEDC</td>
<td>File</td>
<td>PEI module</td>
<td></td>
</tr>
<tr>
<td>PEI dependency section</td>
<td>Section</td>
<td>PEI dependency</td>
<td></td>
</tr>
<tr>
<td>TE image section</td>
<td>Section</td>
<td>TE image</td>
<td></td>
</tr>
<tr>
<td>CD2BCEB3-EA11-484B-B687-AFE57D301C0F</td>
<td>File</td>
<td>PEI module</td>
<td></td>
</tr>
<tr>
<td>4AA91D46-051B-54AE-9C5E-8F4A1F22183D</td>
<td>File</td>
<td>PEI module</td>
<td></td>
</tr>
<tr>
<td>A66A4162-0221-456D-A519-05C4E302AB64</td>
<td>File</td>
<td>PEI module</td>
<td></td>
</tr>
</tbody>
</table>

**Information**

- Type: 1Bh
- Full size: 28h (40)
- Header size: 4h (4)
- Body size: 24h (36)
- Parsed expression:
  - PUSH 6CB3C560-C13F-450A-9993-F10FDD23286
  - PUSH CCEE425A-63DE-45AB-BA0F-E9D7AFC50ACB
  - AND
  - END
gFrameworkEfiMpServiceProtocol
Guid
How to reverse EFI
Tools

- UEFITool and UEFIExtract
  - https://github.com/LongSoft/UEFITool
- Snare’s IDA EFI Utils
  - https://github.com/snare/id้า-efiutils/
- UEFI Firmware parser
  - https://github.com/snare/id้า-efiutils/
- CHIPSEC
  - https://github.com/chipsec/chipsec
EFI file types

- Two executable file types.
- PE32/PE32+ (as in Windows).
- TE – Terse Executable.
- 16/32/64 bit code, depending on phase.
TE file format

- TE is just a stripped version of PE.
- Unnecessary PE headers are removed.
- To save space.
- Used by SEC and PEI phase binaries.
TE file format

- IDA unable to correctly disassemble.
- Fails to parse the TE headers.
- Afaik, still not fixed in 6.8.
- Solution is to build your own TE loader.
- [https://github.com/gdbinit/TELoader](https://github.com/gdbinit/TELoader)
Where is libc?
 EFI Services

- No standard libraries to link against.
- Instead there are services.
- Basic functions made available on each phase.
- Access via function pointers.
typedef struct _EFI_PIEI_SERVICES {
    EFI_TABLE_HEADER Hdr;
    EFI_PIEI_INSTALL_PPI InstallPpi;
    EFI_PIEI_REINSTALL_PPI ReInstallPpi;
    EFI_PIEI_LOCATE_PPI LocatePpi;
    EFI_PIEI_NOTIFY_PPI NotifyPpi;
    EFI_PIEI_GET_BOOT_MODE GetBootMode;
    EFI_PIEI_SET_BOOT_MODE SetBootMode;
    EFI_PIEI_GET_HOB_LIST GetHobList;
    EFI_PIEI_CREATE_HOB CreateHob;
    EFI_PIEI_FFS_FIND_NEXT_VOLUME FfsFindNextVolume;
    EFI_PIEI_FFS_FIND_NEXT_FILE FfsFindNextFile;
    EFI_PIEI_FFS_FIND_SECTION_DATA FfsFindSectionData;
    EFI_PIEI_INSTALL_PEI_MEMORY InstallPeiMemory;
    EFI_PIEI_ALLOCATE_PAGES AllocatePages;
    EFI_PIEI_ALLOCATE_POOL AllocatePool;
    EFI_PIEI_COPY_MEM CopyMem;
    EFI_PIEI_SET_MEM CopyMem;
    EFI_PIEI_REPORT_STATUS_CODE CopyMem;
    EFI_PIEI_RESET_SYSTEM ResetSystem;
    EFI_PIEI_CPU_IO_PPI CpuIo;
    EFI_PIEI_PCI_CFG_PPI PciCfg;
} EFI_PIEI_SERVICES;
typedef struct {
    EFI_TABLE_HEADER
    EFI_GET_TIME
    EFI_SET_TIME
    EFI_GET_WAKEUP_TIME
    EFI_SET_WAKEUP_TIME
    EFI_SET_VIRTUAL_ADDRESS_MAP
    EFI_CONVERT_POINTER
    EFI_GET_VARIABLE
    EFI_GET_NEXT_VARIABLE_NAME
    EFI_SET_VARIABLE
    EFI_GET_NEXT_HIGH_MONO_COUNT
    EFI_RESET_SYSTEM
    EFI_UPDATE_CAPSULE
    EFI_QUERY_CAPSULE_CAPABILITIES
    EFI_QUERY_VARIABLE_INFO
} EFI_RUNTIME_SERVICES;
 EFI Services

- Each phase has different services.
- Entrypoint function contains a pointer to the tables.

```c
typedef EFI_STATUS (*EFI_IMAGE_ENTRY_POINT)(
    IN EFI_HANDLE ImageHandle,
    IN EFI_SYSTEM_TABLE *SystemTable
);
```
typedef struct {
    EFI_TABLE_HEADER Hdr;
    CHAR16 *FirmwareVendor;
    UINT32 FirmwareRevision;
    EFI_HANDLE ConsoleInHandle;
    EFI_SIMPLE_TEXT_INPUT_PROTOCOL *ConIn;
    EFI_HANDLE ConsoleOutHandle;
    EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL *ConOut;
    EFI_HANDLE StandardErrorHandle;
    EFI_SIMPLE_TEXT_OUTPUT_PROTOCOL *StdErr;
    EFI_RUNTIME_SERVICES *RuntimeServices; <- EFI_RUNTIME_SERVICES
    EFI_BOOT.Services *BootServices; <- EFI_BOOT_SERVICES
    UINTN NumberOfTableEntries;
    EFI_CONFIGURATION_TABLE *ConfigurationTable;
} EFI_SYSTEM_TABLE;
EFI Services

- Code that you often see in DXE drivers

```assembly
.text:000000000000000240  GetSystemTables proc near ; CODE XREF: start+16
.text:000000000000000240  mov     cs:SystemTable, rdx
.text:000000000000000247  mov     rax, [rdx+60h]
.text:00000000000000024B  mov     cs:BootServices, rax
.text:000000000000000252  mov     rax, [rdx+58h]
.text:000000000000000256  mov     cs:RunTimeServices, rax
.text:00000000000000025D  xor     eax, eax
.text:00000000000000025F  retn
.text:00000000000000025F  GetSystemTables endp
```
Talking about functions...
Calling conventions

- 32-bit binaries use standard C convention
  - Arguments passed on the stack.
  - SEC/PEI phase binaries.
call PeiPerfMeasure ; PEI_PERF_START (&PrivateData.PS, L"PreMem", NULL, mTick);
lea  eax, [ebp+var_C8]
mov  [esp+8], eax
lea  eax, [ebp-268h]
mov  [esp+4], eax
mov  [esp], edi
call PeiDispatcher ; PeiDispatcher (PeiStartupDescriptor, &PrivateData, DispatchData);
cmp  [ebp+var_9B], 1
jz   short loc_FFEA736E
mov  [esp], esi
mov  dword ptr [esp+0Ch], offset aPrivatedata_pe ; "PrivateData.PeiMemoryInstalled == ((BOO"
mov  dword ptr [esp+8], 16Ch
mov  dword ptr [esp+4], offset a_EdkFoundati_4 ; "./Edk/Foundation/Core/Pei/PeiMain/PeiMa"
call PeiDebugAssert ; PEI_ASSERT(&PrivateData.PS, PrivateData.PeiMemoryInstalled == TRUE);
Calling conventions

- 64-bit binaries use Microsoft’s x64
  - First four arguments: RCX, RDX, R8, R9.
  - Remaining on the stack.
  - 32-byte shadow space on stack.
  - First stack argument starts at offset 0x20.
  - DXE phase binaries.
mov    rax, cs:1F688h
mov    dword ptr [rsp+28h], 2 <- 6th
mov    qword ptr [rsp+20h], 0 <- 5th
lea    rdx, qword_1D7A0    <- 2nd
lea    r8, [rbp+var_38]    <- 3rd
mov    rcx, rdi             <- 1st
xor    r9d, r9d             <- 4th
call   qword ptr [rax+118h]
Protocols & PPIS

Follow Protocol
Protocols & PPIs

- The basic services aren’t enough.
- How are more services made available?
- Via Protocols and PPIs.
- Installed (published) by (U)EFI binaries.
- Others can locate and use them.
Protocols & PPIs

- Protocol (and PPI) is a data structure.
- Contains an identification, GUID.
- Optionally, function pointers and data.


#define EFI ACPI S3 SAVE GUID { 0x125f2de1, 0xfb85, 0x440c, 0xa5, 0x4c, 0x4d, 0x99, 0x35, 0x8a, 0x8d, 0x38 }

typedef struct _EFI ACPI S3 SAVE PROTOCOL {
    EFI ACPI GET LEGACY MEMORY SIZE GetLegacyMemorySize;
    EFI ACPI S3 SAVE S3Save;
} EFI ACPI S3 SAVE PROTOCOL;

[ Function Pointers]
typedef EFI STATUS (EFIAPI *EFI ACPI S3 SAVE)(
    IN EFI ACPI S3 SAVE PROTOCOL * This,
    IN VOID LegacyMemoryAddress
);

typedef EFI STATUS (EFIAPI *EFI ACPI GET LEGACY MEMORY SIZE)(
    IN EFI ACPI S3 SAVE PROTOCOL * This,
    OUT UINTN Size
);
Protocols & PPIs

- Protocols exist in DXE phase.
- PPIs exist in PEI phase.
- In practice we can assume they are equivalent.
Sample PPI usage

- First, locate the PPI.

```c
EFI_STATUS Status;
EFI_BOOT_MODE BootMode;
PEI_CAPSULE_PPI *Capsule;

Status = (*PeiServices)->LocatePpi ((const EFI_PEI_SERVICES **)PeiServices,
&gPeiCapsulePpiGuid,
0,
NULL,
(VOID **)&Capsule
);
```
Sample PPI usage

- Second, use it.

```c
if (Status == EFI_SUCCESS) {
    if (Capsule->CheckCapsuleUpdate(((EFI_PEI_SERVICES**)PeiServices) == EFI_SUCCESS)) {
        BootMode = BOOT_ON_FLASH_UPDATE;
        Status = (*PeiServices)->SetBootMode((const EFI_PEI_SERVICES **)PeiServices, BootMode);
        ASSERT_EFI_ERROR (Status);
    }
}
```
```c
#define EFI_BOOT_SCRIPT_SAVE_GUID \ 
{ 0x470e1529, 0xb79e, 0x4e32, 0xa0, 0xfe, 0x6a, 0x15, 0x6d, 0x29, 0xf9, 0xb2 } 

typedef struct _EFI_BOOT_SCRIPT_SAVE_PROTOCOL { 
    EFI_BOOT_SCRIPT_WRITE Write;
    EFI_BOOT_SCRIPT_CLOSE_TABLE CloseTable;
} EFI_BOOT_SCRIPT_SAVE_PROTOCOL;

.data:000000000000009D20 ; EFI_GUID gEfiBootScriptSaveProtocolGuid
/data:000000000000009D20 gEfiBootScriptSaveProtocolGuid dd 470E1529h
.data:000000000000009D20      dw 0B79Eh
.data:000000000000009D20      dw 4E32h
.data:000000000000009D20      db 0A0h, 0FEh, 6Ah, 15h, 6Dh, 29h, 0F9h, 0B2h
```
; locate_bootscript_save_protocol proc near ; CODE XREF: sub_180C+21
    push    rbp
    mov     rbp, rsp
    sub     rsp, 20h
    mov     rax, [rdx+60h] <- BootServices
    lea     rcx, gEfiBootScriptSaveProtocolGuid <- GUID to locate
    lea     r8, Boot_Script_Save_Interface <- store pointer to table
    xor     edx, edx
    call    qword ptr [rax+140h] <- BootServices->LocateProtocol()
    test    rax, rax
    jns     short loc_281
    mov     rcx, 800000000000000014h
    cmp     rax, rcx
    jz      short loc_281
    mov     cs:Boot_Script_Save_Interface, 0

loc_281:     ; CODE XREF: locate_bootscript_save_protocol+25
             ; locate_bootscript_save_protocol+34
    xor     eax, eax
    add     rsp, 20h
    pop     rbp
    retn
locate_bootscript_save_protocol endp
save_script_dispatch_opcode proc near
    ; CODE XREF: sub_2D0F+6C
    ; sub_3C1A+83 ...
    push rbp
    mov rbp, rsp
    sub rsp, 20h
    mov r9, rdx <- EntryPoint
    mov rdx, 80000000000000000000000000000000eh
    mov rax, cs:Boot_Script_Save_Interface
    test rax, rax <- NULL ptr?
    jz short loc_3E1
    movzx edx, cx <- TableName
    mov rcx, rax <- *This
    mov r8d, 8 <- OpCode
    call qword ptr [rax] <- BootScriptSave->Write()
    xor edx, edx

loc_3E1:
    ; CODE XREF: save_script_dispatch_opcode+1F
    mov rax, rdx
    add rsp, 20h
    pop rbp
    retn
save_script_dispatch_opcode endp
Apple EFI customizations

Apple Computer Inc.

NOTICE OF PROPRIETARY PROPERTY

THE INFORMATION CONTAINED HEREIN IS THE PROPRIETARY PROPERTY OF APPLE COMPUTER, INC. THE POSSESSOR AGREES TO THE FOLLOWING

I TO MAINTAIN THE DOCUMENT IN CONFIDENCE
II NOT TO REPRODUCE OR COPY IT
III NOT TO REVEAL OR PUBLISH IN WHOLE OR PART
Apple EFI customizations

- Apple specific modifications.
- To reserved fields.
- Must be taken care of.
- Else bricked firmware.
- UEFITool v0.27+ handles everything.
EFI_FIRMWARE_VOLUME_HEADER

Summary

Describes the features and layout of the firmware volume.

Prototype

typedef struct {
    UINT8             ZeroVector[16];
    EFI_GUID          FileSystemGuid;
    UINT64            FvLength;
    UINT32            Signature;
    EFI_FVB_ATTRIBUTES_2 Attributes;
    UINT16            HeaderLength;
    UINT16            Checksum;
    UINT16            ExtHeaderOffset;
    UINT8             Reserved[1];
    UINT8             Revision;
    EFI_FV_BLOCK_MAP  BlockMap[];
} EFI_FIRMWARE_VOLUME_HEADER;

Parameters

ZeroVector

The first 16 bytes are reserved to allow for the reset vector of processors whose reset vector is at address 0.
Apple EFI customizations

- The first 8 bytes.
- Constant between firmware volumes with the same GUID.
- Changes between versions?
- Unknown meaning, doesn’t seem relevant.
Apple EFI customizations

- Next 4 bytes.
- CRC32 value.
- Of the firmware volume contents.
- By spec, header got its own 16-bit checksum.
<table>
<thead>
<tr>
<th>Name</th>
<th>Action</th>
<th>Type</th>
<th>Subtype</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel image</td>
<td></td>
<td>Image</td>
<td></td>
</tr>
<tr>
<td>Descriptor region</td>
<td></td>
<td>Region</td>
<td>Descriptor</td>
</tr>
<tr>
<td>ME region</td>
<td></td>
<td>Region</td>
<td>ME</td>
</tr>
<tr>
<td>BIOS region</td>
<td></td>
<td>Region</td>
<td>BIOS</td>
</tr>
<tr>
<td>7A9354D9-0468-444A-81CE-0BF617DB90DF</td>
<td>Volume</td>
<td>FFSv2</td>
<td></td>
</tr>
<tr>
<td>7A9354D9-0468-444A-81CE-0BF617DB90DF</td>
<td>Volume</td>
<td>FFSv2</td>
<td></td>
</tr>
<tr>
<td>7A9354D9-0468-444A-81CE-0BF617DB90DF</td>
<td>Volume</td>
<td>FFSv2</td>
<td></td>
</tr>
<tr>
<td>E3B980A9-5FE3-48E5-9B92-279385A9027</td>
<td>Volume</td>
<td>Unknown</td>
<td></td>
</tr>
<tr>
<td>7A9354D9-0468-444A-81CE-0BF617DB90DF</td>
<td>Volume</td>
<td>Unknown</td>
<td></td>
</tr>
<tr>
<td>153D2197-29BD-440C-AC59-887F70E41A6B</td>
<td>Volume</td>
<td>Unknown</td>
<td></td>
</tr>
<tr>
<td>153D2197-29BD-440C-AC59-887F70E41A6B</td>
<td>Volume</td>
<td>Unknown</td>
<td></td>
</tr>
<tr>
<td>FFF12BD-7696-4CB8-A985-274707584F50</td>
<td>Volume</td>
<td>Unknown</td>
<td></td>
</tr>
<tr>
<td>7A9354D9-0468-444A-81CE-0BF617DB90DF</td>
<td>Volume</td>
<td>FFSv2</td>
<td></td>
</tr>
<tr>
<td>52C05B14-8098-496C-BC3B-04B50211D680</td>
<td>File</td>
<td>PEI core</td>
<td></td>
</tr>
<tr>
<td>88F1DE13-3C6E-4A78-A802-1AC5FF3750FB</td>
<td>File</td>
<td>PEI module</td>
<td></td>
</tr>
<tr>
<td>3B17FC0-2795-406E-B207-680CA768CFB1</td>
<td>File</td>
<td>PEI module</td>
<td></td>
</tr>
<tr>
<td>34C8C2B-661C-45A2-8F2E-89E46B6CC63B</td>
<td>File</td>
<td>PEI module</td>
<td></td>
</tr>
<tr>
<td>8A7B8187-6FDD-4CB8-B7BA-DC3E13CB8524</td>
<td>File</td>
<td>PEI module</td>
<td></td>
</tr>
<tr>
<td>27A5159D-5E61-4809-919A-422EB87101EF</td>
<td>File</td>
<td>PEI module</td>
<td></td>
</tr>
<tr>
<td>01359DD9-9446-4560-ADA4-50A711C83ADA</td>
<td>File</td>
<td>PEI module</td>
<td></td>
</tr>
<tr>
<td>EDF59D2E-D5D6-4A63-A298-8FF2FA47D20B</td>
<td>File</td>
<td>PEI module</td>
<td></td>
</tr>
<tr>
<td>53984C6A-1B4A-4174-9512-A65E5BCBB278</td>
<td>File</td>
<td>PEI module</td>
<td></td>
</tr>
<tr>
<td>996D8F2-703F-492C-9A50-10BE832AAEB1</td>
<td>File</td>
<td>PEI module</td>
<td></td>
</tr>
<tr>
<td>320A58FC-5E08-4D92-9255-BB10AEE6A30</td>
<td>File</td>
<td>PEI module</td>
<td></td>
</tr>
<tr>
<td>011878B-0D3E-4D06-BA29-F0989246599</td>
<td>File</td>
<td>PEI module</td>
<td></td>
</tr>
<tr>
<td>779F6D8-7113-4A41-9648-EB1633CD753B</td>
<td>File</td>
<td>PEI module</td>
<td></td>
</tr>
<tr>
<td>233DF097-3218-47B2-9E09-FFE8C2B0022</td>
<td>File</td>
<td>PEI module</td>
<td></td>
</tr>
<tr>
<td>A65EA411-02A1-456D-A510-86E4E32AB664</td>
<td>File</td>
<td>PEI module</td>
<td></td>
</tr>
</tbody>
</table>

ZeroVector:
70 3D 75 55 00 00 00 00
30 50 65 C8 D0 B1 06 00

Filesysten GUID:
7A9354D9-0468-444A-81CE-0BF617DB90DF

Full size: A0000h (655360)
Header size: 48h (72)
Body size: 9FFB0h (655280)
Revision: 1
Attributes: FFFFFE7Fh
Erase polarity: 1

parseVolume: unknown file system E3B980A9-5FE3-48E5-9B92-279385A9027
parseVolume: unknown file system 153D2197-29BD-440C-AC59-887F70E41A6B
parseVolume: unknown file system 153D2197-29BD-440C-AC59-887F70E41A6B
parseVolume: unknown file system FFF12BD-7696-4CB8-A985-274707584F50
parseVolume: unknown file system 7A9354D9-0468-444A-81CE-0BF617DB90DF

<table>
<thead>
<tr>
<th>Name</th>
<th>Action</th>
<th>Type</th>
<th>Subtype</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel image</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Descriptor region</td>
<td></td>
<td>Intel</td>
<td>Descriptor</td>
<td></td>
</tr>
<tr>
<td>ME region</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BIOS region</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7A9354D9-0468-444A-81CE-0BF617DB90DF</td>
<td>Volume</td>
<td>FFSv2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7A9354D9-0468-444A-81CE-0BF617DB90DF</td>
<td>Volume</td>
<td>FFSv2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7A9354D9-0468-444A-81CE-0BF617DB90DF</td>
<td>Volume</td>
<td>FFSv2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>153D2197-29BD-44DC-AC59-887F70E41A6B</td>
<td>Volume</td>
<td>FFSv2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>153D2197-29BD-44DC-AC59-887F70E41A6B</td>
<td>Volume</td>
<td>FFSv2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>153D2197-29BD-44DC-AC59-887F70E41A6B</td>
<td>Volume</td>
<td>FFSv2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FFF12BBD-7696-4CBB-A985-274707584F50</td>
<td>Volume</td>
<td>FFSv2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7A9354D9-0468-444A-81CE-0BF617DB90DF</td>
<td>Volume</td>
<td>FFSv2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PEI core</td>
<td></td>
<td>File</td>
<td>PEI module</td>
<td></td>
</tr>
<tr>
<td>PEI module</td>
<td></td>
<td>File</td>
<td>PEI module</td>
<td></td>
</tr>
<tr>
<td>PEI module</td>
<td></td>
<td>File</td>
<td>PEI module</td>
<td></td>
</tr>
<tr>
<td>PEI module</td>
<td></td>
<td>File</td>
<td>PEI module</td>
<td></td>
</tr>
<tr>
<td>PEI module</td>
<td></td>
<td>File</td>
<td>PEI module</td>
<td></td>
</tr>
<tr>
<td>PEI module</td>
<td></td>
<td>File</td>
<td>PEI module</td>
<td></td>
</tr>
<tr>
<td>PEI module</td>
<td></td>
<td>File</td>
<td>PEI module</td>
<td></td>
</tr>
<tr>
<td>PEI module</td>
<td></td>
<td>File</td>
<td>PEI module</td>
<td></td>
</tr>
<tr>
<td>PEI module</td>
<td></td>
<td>File</td>
<td>PEI module</td>
<td></td>
</tr>
<tr>
<td>PEI module</td>
<td></td>
<td>File</td>
<td>PEI module</td>
<td></td>
</tr>
</tbody>
</table>

Messages:
- parseVolume: unknown file system E3B980A9-5FE3-48E5-9B92-279B385A9027
- parseVolume: unknown file system 153D2197-29BD-44DC-AC59-887F70E41A6B
- parseVolume: unknown file system 153D2197-29BD-44DC-AC59-887F70E41A6B
- parseVolume: unknown file system FFF12BBD-7696-4CBB-A985-274707584F50

Zero Vector: 30 50 65 C8
Apple EFI customizations

- Last 4 bytes.
- Total space used by firmware files.
- Must be updated if there are any modifications to volume free space.
- Bricked firmware if wrong.
<table>
<thead>
<tr>
<th>Name</th>
<th>Action</th>
<th>Type</th>
<th>Subtype</th>
</tr>
</thead>
<tbody>
<tr>
<td>147B4839-5DBE-413F-917F-DFEB687C6312</td>
<td>File</td>
<td>PEI module</td>
<td></td>
</tr>
<tr>
<td>3B42EF57-16D3-44CB-8632-9FDB06B41451</td>
<td>File</td>
<td>PEI module</td>
<td></td>
</tr>
<tr>
<td>FD236AE7-8791-4C84-B29E-29BDEEE1A811</td>
<td>File</td>
<td>PEI module</td>
<td></td>
</tr>
<tr>
<td>B6A2AFF3-767C-5658-C37A-D1CB2EF75543</td>
<td>File</td>
<td>PEI module</td>
<td></td>
</tr>
<tr>
<td>4B62AFF3-667C-545B-B274-A16C2DFBBA0</td>
<td>File</td>
<td>PEI module</td>
<td></td>
</tr>
<tr>
<td>8BCEDD7-E2B5-4168-9BF3-09AF66C03FFE</td>
<td>File</td>
<td>PEI module</td>
<td></td>
</tr>
<tr>
<td>8AC5718-9B39-423D-BB39-F5C8B8404C0F</td>
<td>File</td>
<td>PEI module</td>
<td></td>
</tr>
<tr>
<td>6A89B044-D0DB-5AA8-A301-53FA273E2FD6</td>
<td>File</td>
<td>PEI module</td>
<td></td>
</tr>
<tr>
<td>1ACEEB06-5A6F-4077-A934-865B78C8DC03</td>
<td>File</td>
<td>PEI module</td>
<td></td>
</tr>
<tr>
<td>4B30B764-6C1C-4BF9-95DA-9782918EB398</td>
<td>File</td>
<td>PEI module</td>
<td></td>
</tr>
<tr>
<td>CD2B6EB3-EA11-4848-B867-AFE570D31C0F</td>
<td>File</td>
<td>PEI module</td>
<td></td>
</tr>
<tr>
<td>6ECFCE51-5724-450C-A38A-58553E954422</td>
<td>File</td>
<td>PEI module</td>
<td></td>
</tr>
<tr>
<td>8666BD71-7C79-4BF1-A93B-0668B30D8F9A</td>
<td>File</td>
<td>PEI module</td>
<td></td>
</tr>
<tr>
<td>88B8214F9-4ADB-47DD-AC62-8313C537E9FA</td>
<td>File</td>
<td>PEI module</td>
<td></td>
</tr>
<tr>
<td>616E687C-7CE7-4563-8706-226E02CE2A09</td>
<td>File</td>
<td>PEI module</td>
<td></td>
</tr>
<tr>
<td>6406C7D3-B5E4-4F76-B35A-BF07D1CF58D2</td>
<td>File</td>
<td>PEI module</td>
<td></td>
</tr>
<tr>
<td>ADA7DBBB-2E6F-4FF6-8963-7CDS5C040C52</td>
<td>File</td>
<td>PEI module</td>
<td></td>
</tr>
<tr>
<td>3D17205B-4C49-47E2-B157-864CD30BD0</td>
<td>File</td>
<td>PEI module</td>
<td></td>
</tr>
<tr>
<td>66ACB016-A1D4-4E74-BA7D-EF93A85F112F</td>
<td>File</td>
<td>PEI module</td>
<td></td>
</tr>
<tr>
<td>C3E36D09-8294-49B7-ABA5-D528BE33E828</td>
<td>File</td>
<td>Freeform</td>
<td></td>
</tr>
<tr>
<td>B535ABF6-967D-43F2-B494-A1EBEB21A28E</td>
<td>File</td>
<td>Freeform</td>
<td></td>
</tr>
<tr>
<td>FF48D0C5-2BF0-4980-BF2D-05806B3E79F9</td>
<td>File</td>
<td>PEI module</td>
<td></td>
</tr>
</tbody>
</table>

**Full size:** 34E30h (216624)

**Messages**

parseVolume: unknown file system E3B980A9-5FE3-4BE5-9B92-2798305A9027
parseVolume: unknown file system 153D2197-29BD-440C-AC59-887F8E41A6B
parseVolume: unknown file system 153D2197-29BD-440C-AC59-887F8E41A6B
parseVolume: unknown file system FFF12BBD-7696-4C8B-A985-274707584F50

0xA0000 – 0x34E30 = 0x06B1D0
How to find EFI monsters
How to find EFI monsters

- Dump the flash contents.
  - Via hardware, if possible.
- Have a known good image.
  - A previously certified/trusted dump.
  - Or firmware updates.
How to find EFI monsters

- Firmware updates available from Apple.
- Direct downloads.
- Or combined with OS installer or updates.
- No hashes from Apple available (yet).
How to find EFI monsters

- Only useful for machines with available updates.
- Newly released machines need to wait for a firmware update.
How to find EFI monsters

- Firmware & signatures vault
  - https://github.com/gdbinit/firmware_vault

- Signed by my PGP key.

- Extracted from available Apple updates.

- Soon, the SMC updates.
How to find EFI monsters

- Two file formats used for updates.
  - SCAP (most common).
  - FD (some newer and older models).
- UEFITool can process both.
### SCAP

- EFI Capsule.
- Used to deliver updates.
- Recommended delivery mechanism.
- Composed by firmware volumes.
- Flash dumps parser can be reused.
SCAP

- 1 is the EfiFlasher.efi or also known as UpdateDriverDxe.

- 2 are the BIOS region contents.

- Encapsulated on different GUIDs.
<table>
<thead>
<tr>
<th>Name</th>
<th>Action</th>
<th>Type</th>
<th>Subtype</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>0EB4FC69-29CC-4C6D-92AC-6D476921850F</td>
<td>File</td>
<td>DXE driver</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9BBBD59B-EBBA-48EE-98DD-C295392F1EDB</td>
<td>File</td>
<td>Raw</td>
<td></td>
<td></td>
</tr>
<tr>
<td>283FA2E5-532C-484D-9383-9F93B36F087E</td>
<td>File</td>
<td>Raw</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7A9354D9-0468-444A-81CE-0BF6178D90DF</td>
<td>Volume</td>
<td>FFSv2</td>
<td>AppleCRC32 AppleFSO</td>
<td></td>
</tr>
<tr>
<td>77AD7FDB-DFA2-4302-B898-C72E4CD0F4</td>
<td>File</td>
<td>Volume image</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FB1E2F9C-8E65-448D-A98F-C22943F45CAF</td>
<td>File</td>
<td>Volume image</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AFCCAA0E-E825-441E-A353-1577F1E9D8289</td>
<td>File</td>
<td>Volume image</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5B4C51B3-A7AC-41B9-B345-022C4EE1C001</td>
<td>File</td>
<td>Volume image</td>
<td></td>
<td></td>
</tr>
<tr>
<td>66E86CB8-B7AE-4FB0-9ACA-C833B6ED4AD</td>
<td>File</td>
<td>Volume image</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00858DD9-0E28-4709-A472-F8129EBCBDA7</td>
<td>Section</td>
<td>Compressed</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FC1BCDB8-7D31-49AA-936A-A4600DDDD83</td>
<td>Section</td>
<td>GUID defined</td>
<td></td>
<td></td>
</tr>
<tr>
<td>77777777-EB25-441E-A353-1577F1E9D8289</td>
<td>Volume image section</td>
<td>Volume</td>
<td>AppleCRC32</td>
<td></td>
</tr>
<tr>
<td>153D2197-29BD-44DC-AC59-887F0E41A6B</td>
<td>Compressed section</td>
<td>Compressed</td>
<td>Volume image</td>
<td></td>
</tr>
<tr>
<td>FC1BCDB8-7D31-49AA-936A-A4600DDDD83</td>
<td>Volume image section</td>
<td>GUID defined</td>
<td>Volume image</td>
<td></td>
</tr>
<tr>
<td>04ADEEAD-61FF-4D31-B6BA-64F8BF901F5A</td>
<td>Volume image section</td>
<td>GUID defined</td>
<td>Volume image</td>
<td>AppleCRC32 AppleFSO</td>
</tr>
<tr>
<td>1CEA9D78-200D-49D4-B2A0-062EBE58A872</td>
<td>File</td>
<td>Freeform</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F1143A53-CBEB-4833-A4DC-0826E063EC08</td>
<td>File</td>
<td>Freeform</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BA4F8CAB-E228-4BC2-BCE8-89D5BEBA9C13</td>
<td>File</td>
<td>Volume image</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0AECB734-6EC6-4FD1-A877-EF185E5BFEE</td>
<td>File</td>
<td>Volume image</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Volume free space</td>
<td>Volume free space</td>
<td>Non-empty</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Padding</td>
<td>Padding</td>
<td>Non-empty</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SCAP

- 1 is NVRAM region.
- 2 is Microcode.
- 3 is Boot volume.
- SCAP is signed.
- RSA2048 SHA256.
- Apple backported from UEFI.
- First reported by Trammell Hudson.
% xxd -g 1 MBP101_00EE_B02_LOCKED.scap | tail -40 | head

0810030: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
0810040: ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
0810050: 14 74 71 a7 16 c6 77 49 94 20 84 47 12 a7 35 bf
0810060: cf fd 3e 6b fe 66 ec 5 f4 4b 7e 0e d2 63 98
0810070: 08 a9 8d 10 ac 37 8e 95 1c aa 0e 1c 1d 85 ef 6c
0810080: d5 1c 75 8c 75 18 16 ef 59 9f be da ef 4d 6b 0c

GUID a7717414-c616-4977-9420844712a735bf
How to find EFI monsters

- Compare the flash dump against SCAP.
- Locate all EFI binaries in the dump.
- Checksum against SCAP contents.
How to find EFI monsters

- We also need to verify:
  - New files.
  - Missing files.
  - Free/padding space?
How to find EFI monsters

- Verify NVRAM contents!
- Boot device is stored there.
- HackingTeam had a new variable there.
  - A simple “fuse” to decide to infect or not target system.
BOOLEAN
EFIAPI
CheckfTA()
{
    EFI_STATUS Status = EFI_SUCCESS;
    UINTN VarDataSize;
    UINT8 VarData;

    VarData=0;
    VarDataSize=sizeof(VarData);
    Status=gRT->GetVariable(L"fTA", &gEfiGlobalVariableGuid, NULL, &VarDataSize, (UINTN*)&VarData);

    if(Status!=EFI_SUCCESS || VarData==0)
    {
        #ifdef FORCE_DEBUG
        Print(L"Devo Infettare\n");
        #endif
        return FALSE;
    }

    #ifdef FORCE_DEBUG
    Print(L"NON Devo Infettare\n");
    #endif
    return TRUE;
}
How to find EFI monsters

- Don’t forget boot.efi.
- Not very stealth.
- Always keep in mind that sophistication is not always required!
- If it works, why not?
How to find EFI monsters

- SCAP is used by EfiFlasher.
- We can stitch our own firmware.
- Extract files from SCAP and build it.
- Reflash via SPI.
- Assumption that SCAP is legit.
How to find EFI monsters

- Stitch utility still in TODO list.
- Potential issues:
  - NVRAM contents?
  - Serial numbers?
- Use current dump and just replace binaries?
Conclusions
Conclusions

- (U)EFI rootkits aren’t unicorns.
- Although they are very rare.
- Honestly, we don’t know what’s out there.
- HackingTeam developed one in 2014.
- Although it was too simple and not advanced.
Conclusions

- Chasing them requires hardware assistance.
- Disassembling computers monthly is not scalable/efficient/viable.
- How to deal with this at enterprise level?
Conclusions

- Vendors are usually slow releasing updates.
- If they ever do it.
- Check legbacore.com work.
Conclusions

- SMC is another interesting chip.
- Alex Ionescu and Andrea Barisani did some work in this area.
- Great rootkit possibilities?
Conclusions

- Intel Management Engine (ME).
- Big Pandora Box?
- Security researchers should have easier access to it.
Conclusions

- Option ROMs.
- Still an issue with Apple’s EFI implementation.
- No SecureBoot (signed OptionROMs).
- Check Thunderstrike 2 OptionROM worm.
Footage released of Guardian editors destroying Snowden hard drives

GCHQ technicians watched as journalists took angle grinders and drills to computers after weeks of tense negotiations.

- Watch the footage of the hard drives being destroyed

New video footage has been released for the first time of the moment Guardian editors destroyed computers used to store top-secret documents leaked by the NSA whistleblower Edward Snowden.
In July 2013, GCHQ, Britain’s equivalent of the U.S. National Security Agency, **forced** journalists at the London headquarters of *The Guardian* to completely obliterate the memory of the computers on which they kept copies of top-secret documents provided to them by former NSA contractor and whistle-blower Edward Snowden.
How to Destroy a Laptop with Top Secrets
How did GCHQ do it to the Guardian's copy of Snowden's files?

Mustafa Al-Bassam and Richard Tynan
Conclusions

- Trolling?
- Real?
- Maybe a mix of both.
- Check Apple logic board schematics.
- There’s a ton of interconnected stuff.
Conclusions

- We need trusted hardware solutions.
- If we can’t trust hardware we are wasting a lot of time solving some software problems.
Conclusions

- Bring back physical protections?
- Switches to enable:
  - Flash writes.
  - MIC.
  - Camera.
  - Etc...
Conclusions

Jumper JP4: BIOS Flash Protect

The system BIOS and CMOS Setup Utility are stored in Flash memory on the motherboard, which provides permanent storage, but is rewritable, allowing for BIOS updates. Jumper JP4 controls the protection scheme that prevents accidental damage to or rewriting of the data stored in Flash memory.

<table>
<thead>
<tr>
<th>JP4: BIOS Flash Protect</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short 1-2</td>
<td>Protection mode selected in BIOS CMOS Setup Utility [Default]</td>
</tr>
<tr>
<td>Short 2-3</td>
<td>Protection enabled in hardware</td>
</tr>
<tr>
<td>Open [Remove Cap]</td>
<td>No BIOS Flash Protection</td>
</tr>
</tbody>
</table>
Conclusions

- Acer C720 & C720P Chromebook.
- #7 is a write-protect screw.
Conclusions

- Might require new hardware design?
- NVRAM needs to be writable.
- An independent flash chip for writable regions?
- BOM/space restrictions?
Conclusions

- Apple has a great opportunity here.
- Full control of design and supply chain.
- Can improve designs.
- Can force faster updates.
- Only matched by Chromebook?
FIRMWARE SECURITY IS CRITICAL!
Greetings

- CODE BLUE team, Snare, Trammell, Xeno, Corey, Saure, cr4sh.
https://reverse.put.as
https://github.com/gdbinit
reverser@put.as
@osxreverser
#osxre @ irc.freenode.net

PGP key
https://reverse.put.as/wp-content/uploads/2008/06/publickey.txt

PGP Fingerprint
7B05 44D1 A1D5 3078 7F4C E745 9BB7 2A44 ED41 BF05
A day full of possibilities!

Let's go exploring!
References

- Images from images.google.com. Credit due to all their authors.
- Thunderstrike presentation
  - https://trmm.net/Thunderstrike_31c3
- Thunderstrike 2 presentation
  - https://trmm.net/Thunderstrike_2
- Snare EFI rootkits presentations
- Legbacore.com papers and presentations
  - http://legbacore.com/Research.html
References

- Alex Ionescu. Ninjas and Harry Potter: "Spell"unking in Apple SMC Land
  - http://www.nosuchcon.org/talks/2013/D1_02_Alex_Ninjas_and_Harry_Potter.pdf
- Alex Ionescu. Apple SMC The place to be definitely For an implant
  - https://www.youtube.com/watch?v=nSqpinnjmg
- Andrea Barisani. Daniele Bianco. Practical Exploitation of Embedded Systems
References

- fGI. The Empire Strikes Back Apple – how your Mac firmware security is completely broken
  - https://reverse.put.as/2015/05/29/the-empire-strikes-back-apple-how-your-mac-firmware-security-is-completely-broken/

- fGI. Reversing Prince Harming’s kiss of death
  - https://reverse.put.as/2015/07/01/reversing-prince-harmings-kiss-of-death/

- Cr4sh, Exploiting UEFI boot script table vulnerability
References

- Cr4sh, Building reliable SMM backdoor for UEFI based platforms

- Firmware papers and presentations timeline
  - http://timeglider.com/timeline/5ca2daa6078caaf4

- Archive of OS X/iOS and firmware papers & presentations
  - https://reverse.put.as/papers/

- Intel ATR - Black Hat 2015 / Def Con 23 - Firmware rootkit
  - https://www.youtube.com/watch?v=sJnliPN0104&app=desktop