# Rootkit in your laptop:

## Hidden code in your chipset and how to discover what exactly it does

Igor Skochinsky
Hex-Rays

Breakpoint 2012
Melbourne

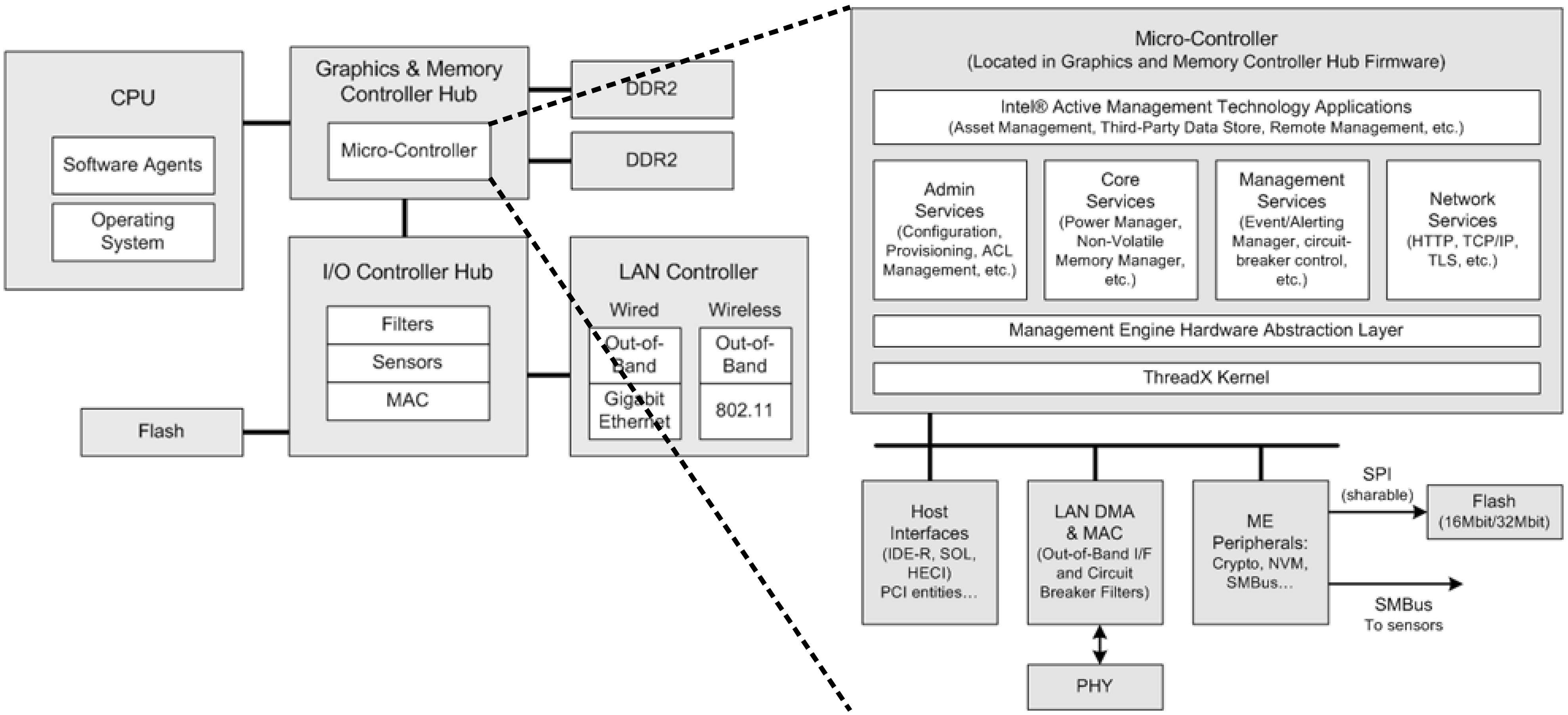
## Outline

- High-level overview of the ME
- Low-level details
- ME security and potential attacks
- Results
- Future work

## ME: High-level overview

- Management Engine (or Manageability Engine) is a dedicated microcontroller on recent Intel platforms
- In first versions it was included in the network card, later moved into the chipset
- Shares flash with the BIOS but is completely independent from the main CPU
- Can be active even when the system is hibernating or turned off (but connected to mains)
- Has a dedicated connection to the network interface; can intercept or send any data without main CPU's knowledge

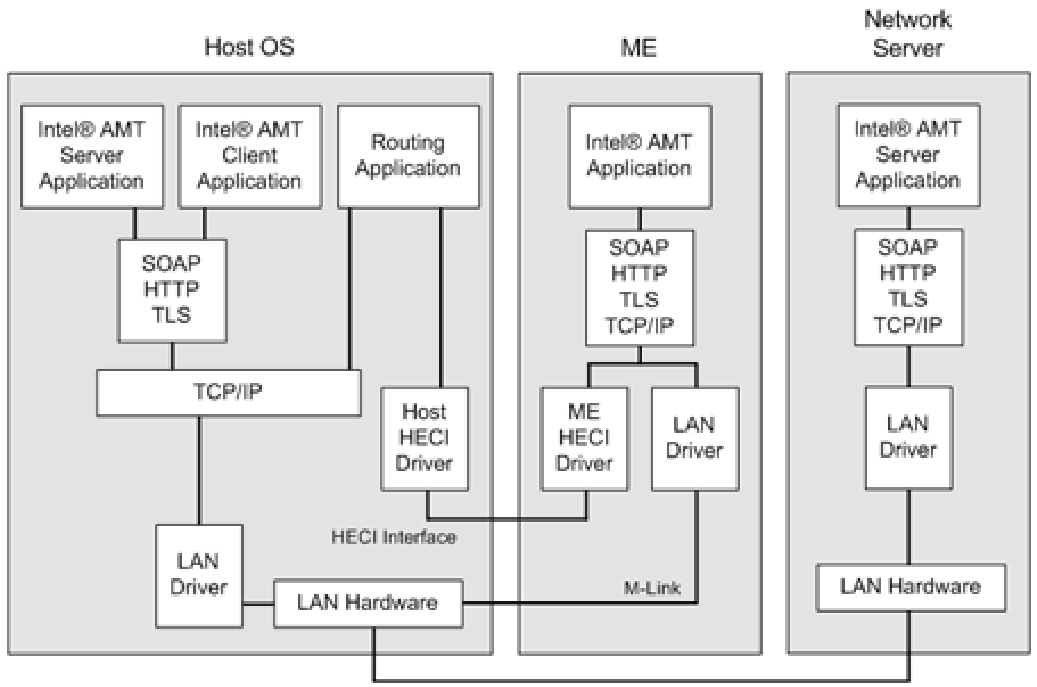
## ME: High-level overview


CPU
Software Agents
Operating System
Graphics & Memory Controller Hub
Micro-Controller
DDR2
DDR2
I/O Controller Hub
Filters
Sensors
MAC
Flash
LAN Controller
Wired
Wireless
Out-of-Band
Gigabit Ethernet
Out-of-Band
802.11
Micro-Controller (Located in Graphics and Memory Controller Hub Firmware)
Intel® Active Management Technology Applications (Asset Management, Third-Party Data Store, Remote Management, etc.)
Admin Services (Configuration, Provisioning, ACL Management, etc.)
Core Services (Power Manager, Non-Volatile Memory Manager, etc.)
Management Services (Event/Alerting Manager, circuit-breaker control, etc.)
Network Services (HTTP, TCP/IP, TLS, etc.)
Management Engine Hardware Abstraction Layer
ThreadX Kernel
Host Interfaces (IDE-R, SOL, HECI) PCI entities...
LAN DMA & MAC (Out-of-Band I/F and Circuit Breaker Filters)
ME Peripherals: Crypto, NVM, SMBus...
SPI (sharable)
Flash (16Mbit/32Mbit)
SMBus To sensors
PHY

Credit: Intel 2009

## ME: High-level overview

Communicating with the Host OS and network



- HECI: Host Embedded Controller Interface; communication using a PCI memory-mapped area
- Network protocol is SOAP based; can be plain HTTP or HTTPS

## ME: High-level overview

### Some of the ME components

- Active Management Technology (AMT): remote configuration, administration, provisioning, repair, KVM
- System Defense: lowest-level firewall/packet filter with customizable rules
- IDE Redirection (IDE-R) and Serial-Over-LAN (SOL): boot from a remote CD/HDD image to fix non-bootable or infected OS, and control the PC console
- Identity Protection: embedded one-time password (OTP) token for two-factor authentication
- Protected Transaction Display: secure PIN entry on a remote server not visible to the host software

## ME: High-level overview

### Intel Anti-Theft

- PC can be locked or disabled if it fails to check-in with the remote server at some predefined interval; if the server signals that the PC is marked as stolen; or on delivery of a "poison pill"
- Poison pill can be sent as an SMS if a 3G connection is available
- Can notify disk encryption software to erase HDD encryption keys
- Reactivation is possible using previously set up recovery password or by using one-time password

## ME: Low-level details

## ME: Low-level details

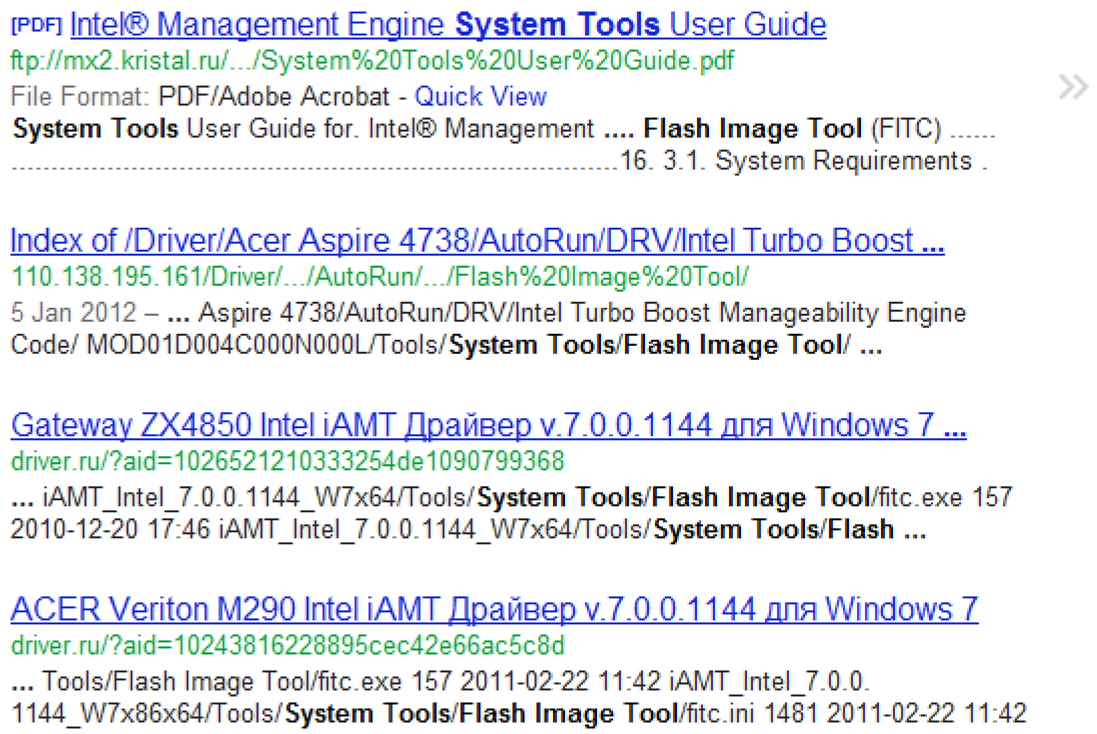
Sources of information

- Intel's whitepapers and other publications (e.g. patents)
- Intel's official drivers and software
  - HECI driver, management services, status checkers
  - AMT SDK, code samples
  - Linux drivers and supporting software; coreboot
- BIOS updates for boards on Intel chipsets
  - Even though ME firmware is usually not updateable using normal means, it's usually still included in the BIOS image
  - Sometimes separate ME firmware updates are available too

## ME: Low-level details

Sources of information

- Intel's ME Firmware kits are not supposed to be distributed to end users
- However, many vendors still put up the whole package instead of just the drivers, or forget to disable the FTP listing



[PDF] Intel® Management Engine **System Tools** User Guide
ftp://mx2.kristal.ru/.../System%20Tools%20User%20Guide.pdf
File Format: PDF/Adobe Acrobat - Quick View
**System Tools** User Guide for. Intel® Management **.... Flash Image Tool** (FITC) ...... ............................................................................16. 3.1. System Requirements .

Index of /Driver/Acer Aspire 4738/AutoRun/DRV/Intel Turbo Boost ...
110.138.195.161/Driver/.../AutoRun/.../Flash%20Image%20Tool/
5 Jan 2012 – **...** Aspire 4738/AutoRun/DRV/Intel Turbo Boost Manageability Engine Code/ MOD01D004C000N000L/Tools/**System Tools/Flash Image Tool**/ **...**

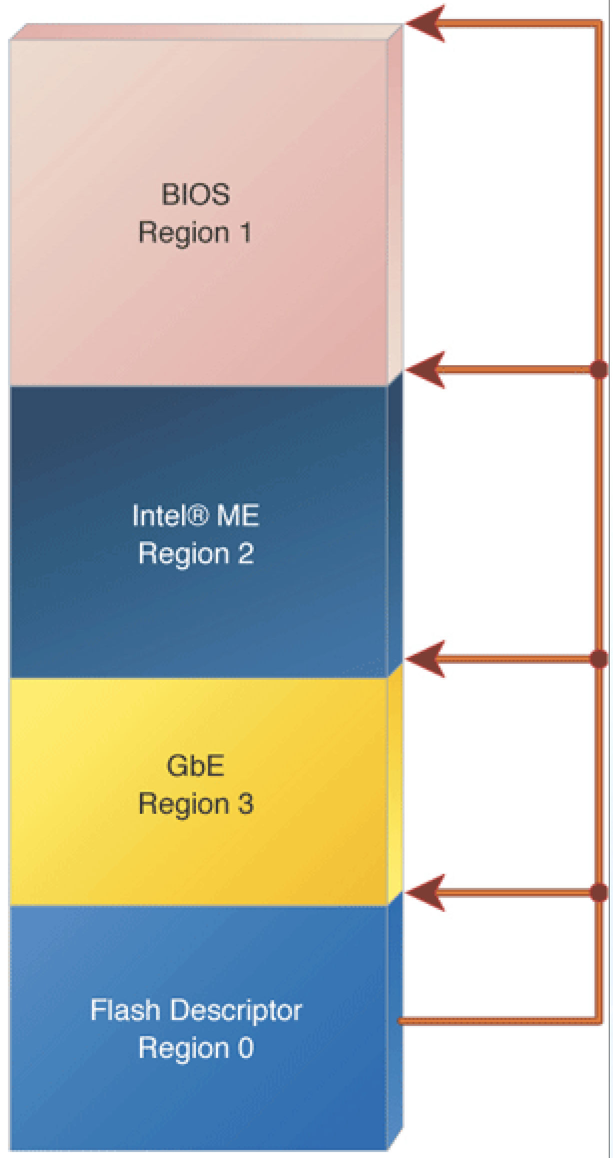
Gateway ZX4850 Intel iAMT Драйвер v.7.0.0.1144 для Windows 7 ...
driver.ru/?aid=1026521210333254de1090799368
**...** iAMT_Intel_7.0.0.1144_W7x64/Tools/**System Tools/Flash Image Tool**/fitc.exe 157 2010-12-20 17:46 iAMT_Intel_7.0.0.1144_W7x64/Tools/**System Tools/Flash** **...**

ACER Veriton M290 Intel iAMT Драйвер v.7.0.0.1144 для Windows 7
driver.ru/?aid=10243816228895cec42e66ac5c8d
**...** Tools/Flash Image Tool/fitc.exe 157 2011-02-22 11:42 iAMT_Intel_7.0.0. 1144_W7x86x64/Tools/**System Tools/Flash Image Tool**/fitc.ini 1481 2011-02-22 11:42

With a few picked keywords you can find the good stuff :)

## ME: Low-level details

- The SPI flash is shared between BIOS, ME and GbE
- For security, BIOS (and OS) should not have access to ME region
- The chipset enforces it using information in the Descriptor region
- The Descriptor region must be at the lowest address of the flash and contain addresses and sizes of other regions, as well as their mutual access permissions.



## ME: Low-level details

- ME region itself is not monolithic
- It consists of several partitions, and the table at the start* describes them

Partition table header

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | "$FPT" | | | | NumEntries | | | | Ver | Entry Type | HdrLen | Checksum | FlashCycleLifetime | | FlashCycleLimit | |
| 10 | UMASize | | | | Flags | | | | | | | | | | | |

Partition table entry

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | Name | | | | Owner | | | | Offset | | | | Length | | | |
| 10 | StartTokens | | | | MaxTokens | | | | ScratchSectors | | | | Flags | | | |

*Starting from ME 3.x the table begins at offset 0x10 (table version 2.0)

## ME: Low-level details

```
===ME Flash Partition Table===
NumEntries: 10
Version:    2.0
EntryType:  10
HeaderLen:  30
Checksum:   9F
FlashCycleLifetime: 7
FlashCycleLimit:    100
UMASize:    16
Flags:      FFFFFE07
    EFFS present:   1
    ME Layout Type: 3
```

```
Partition:       'FOVD'
Owner:           'KRID'
Offset/size:     00000400/00001C00
TokensOnStart:   00000001
MaxTokens:       00000001
ScratchSectors:  00000000
Flags:               0783
    Type:          3 (Generic)
    DirectAccess:  1
    Read:          1
    Write:         1
    Execute:       1
    Logical:       0
    WOPDisable:    0
    ExclBlockUse:  0
```

Partition type (Flags&0x7F):

| 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Code | Block I/O | NVRAM | Generic | Effs | Rom |

## ME: Low-level details

- Code partitions have a header called "manifest"
- It contains versioning info, number of code modules, but also an RSA signature of the whole partition
- Format of the header is very close to TXT AC modules

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | Type | | SubType | | HdrLen | | | | HdrVer | | | | Flags | | | |
| 10 | Vendor | | | | Date | | | | Size | | | | Tag | | | |
| 20 | NumMods | | | | Version | | | | Reserved==> | | | | | | | |
| 70 | <==Reserved | | | | | | | | KeySize | | | | Reserved | | | |
| 80-17F | RsaPubKey | | | | | | | | | | | | | | | |
| 180 | RsaPubExp | | | | RsaSig==> | | | | | | | | | | | |
| 280 | <==RsaSig | | | | PartitionName | | | | | | | | | | | |

## ME: Low-level details

An example code partition header

```
Module Type: 4, Subtype: 0
Header Length:       0xA1 (0x284 bytes)
Header Version:      1.0
Flags:               0x00000000  [production signed] [production flag]
Module Vendor:       0x8086
Date:                20120705
Total Manifest Size: 0xFD (0x3F4 bytes)
Tag:                 $MN2
Number of modules:   2
Version:             8.1.0.1265
Unknown data 1:      [0L, 1L, 2L, 0L, 0L, 0L, 0L, 0L, 0L, 0L, 0L, 0L,
0L, 0L, 0L, 0L, 0L, 0L, 0L]
Key size:            0x40 (0x100 bytes)
Scratch size:        0x01 (0x4 bytes)
RSA Public Key:      [skipped]
RSA Public Exponent: 17
RSA Signature:       [skipped]
Partition name:      MDMV
Unknown data 2:      [0L, 0L]
```

## ME: Low-level details

- The format of module headers depends on the version (header tag $MAN or $MN2)
- Module headers include module name, hash, sizes (compressed and uncompressed), flags and runtime info (load address, entrypoints)
- Modules can be stored uncompressed, or compressed with LZMA or Huffman

```
Header tag:      $MME
Module name:     JOM
Hash:            AC A3 [...] C1 6C
Offset:          0x00015F7A
Data length:     0x00019F6D
LoadBase:        0x200B1000
Flags:           0x0012D42A
    Power Type:      POWER_TYPE_M0_ONLY (1)
    Compression:     COMP_TYPE_LZMA (2)
    API Type:        API_TYPE_KERNEL (2)
```

## ME: Low-level details

- There have been two generations of the processor core, and corresponding changes in firmware layout

| | Gen 1 | Gen 2 |
|---|---|---|
| ME versions | 1.x-5.x | 6.x-8.x |
| Core | ARCTangent-A4 | ARCTangent-A5(?) |
| ISA | ARC (32-bit) | ARCompact (32/16) |
| Manifest tag | $MAN | $MN2 |
| Module header tag | $MOD | $MME |

## ME: Low-level details

- The OS running on the chip is ThreadX RTOS from Express Logic
- OS provides APIs for managing threads (tasks), semaphores, message queues, event flags, timers, memory allocations etc.
- The ME firmware wraps those APIs in a module called KERNEL, and uses it from other modules (via tables of pointers).
- Express Logic provides a demo version (binary only) of ThreadX for ARC, which helps in identifying APIs in ME
- Unfortunately Gen2 uses the Huffman compression (which I have not figured out yet) for the KERNEL :(
- So the going is somewhat slow for the newer firmwares

## ME: communications

- If AMT option is enabled, ME listens for packets on several ports (e.g. 16992 for HTTP and 16993 for HTTPS) for HTTP requests from browsers (for Web UI) or SOAP requests.
- Since it has a separate IP and MAC for the OOB interface, this does not interfere with the host
- ME is also exposed by the chipset as a PCI device to the CPU, and can exchange messages with it using Host Embedded Controller Interface (HECI) protocol over a memory-mapped IO area (MMIO)
- The protocol itself is described in public documentation [DCMI-HI], but the higher-level messages are not well documented
- ME can expose various *clients* to the host, each identified by a unique UUID or a numeric ID, and host can talk to each client independently
- Several core clients have fixed low IDs, the rest gets dynamic numbers

## ME: communications

- An example of enumerating clients (FreeBSD):

```
heci0: <Intel 82G33/G31/P35/P31 Express HECI/MEI Controller> mem
0xd0526100-0xd052610f irq 16 at device 3.0 on pci0
heci0: using MSI
heci0: [ITHREAD]
heci0: found ME client at address 0x02:
heci0:  status = 0x00
heci0:  protocol_name(guid) = BB875E12-CB58-4D14-AE93-8566183C66C7
heci0: found ME client at address 0x03:
heci0:  status = 0x00
heci0:  protocol_name(guid) = A12FF5CA-FACB-4CB4-A958-19A23B2E6881
heci0: found ME client at address 0x06:
heci0:  status = 0x00
heci0:  protocol_name(guid) = 9B27FD6D-EF72-4967-BCC2-471A32679620
heci0: found ME client at address 0x07:
heci0:  status = 0x00
heci0:  protocol_name(guid) = 55213584-9A29-4916-BADF-0FB7ED682AEB
[...]
heci0: found ME client at address 0x27:
heci0:  status = 0x00
heci0:  protocol_name(guid) = 05B79A6F-4628-4D7F-899D-A91514CB32AB
```

## ME: communications

- A list of some of the known clients, gathered from headers and other sources

| Fixed ID | GUID | Name |
|---|---|---|
| | 8e6a6715-9abc-4043-88ef-9e39c6f63e0f | MKHI |
| 8 | 42b3ce2f-bd9f-485a-96ae-26406230b1ff | ICC |
| 9 | d2ea63bc-5f04-4997-9454-8cadf4e3ef8a | Thermal |
| | 309dcde8-ccb1-4062-8f78-600115a34327 | Firmware Update |
| | 05b79a6f-4628-4d7f-899d-a91514cb32ab | Watchdog |
| | 6733a4db-0476-4e7b-b3af-bcfc29bee7a7 | LME |
| | 12f80028-b4b7-4b2d-aca8-46e0ff65814c | PTHI (AMTHI) |
| | 3d98d9b7-1ce8-4252-b337-2eff106ef29f | LMS |
| | 6b5205b9-8185-4519-b889-d98724b58607 | QST |
| | 0f908627d-13bf-4a04-0b91f-0a64e9245323d | CLS |
| | 3c4852d6-d47b-4f46-b05e-b5edc1aa430a | TDT (AT-p) |

## ME: communications

- One of the main users of the HECI interface is the BIOS
- It has to allocate the UMA memory for ME, protect it, and notify ME about it
- It also needs to tell ME about various events, including End-Of-POST (EOP)
- If not disabled at manufacturing time, BIOS can also ask ME to temporarily open its flash region for reading and writing; this functionality is intended to allow ME region updates, and is called Host ME Region Flash Protection Override (HMRFPO)
- An optional module inside BIOS, MEBx (ME BIOS Extension) provides a UI for the user to configure various ME options. It also uses HECI to communicate with ME
- Thus, reverse-engineering BIOS is a good source for info about ME communications

## ME: Security

## ME: Security

- ME includes numerous security features
- Code signing: all code that is supposed to be running on the ME is signed with RSA and is checked by the boot ROM

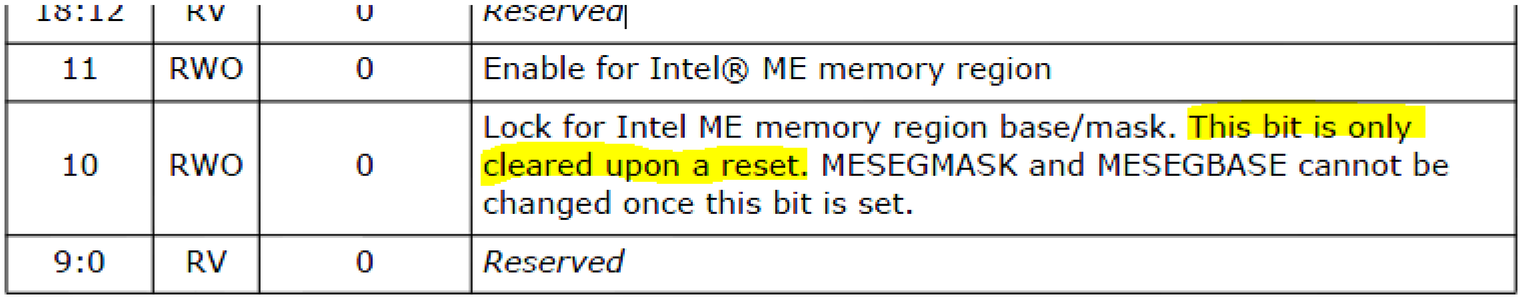
“During the design phase, a Firmware Signing Key (FWSK) public/private pair is generated at a secure Intel Location, using the Intel Code Signing System. The Private FWSK is stored securely and confidentially by Intel. Intel AMT ROM includes a SHA-1 Hash of the public key, based on RSA, 2048 bit modulus fixed. Each approved production firmware image is digitally signed by Intel with the private FWSK. The public FWSK and the digital signature are appended to the firmware image manifest.

At runtime, a secure boot sequence is accomplished by means of the boot ROM verifying that the public FWSK on Flash is valid, based on the hash value in ROM. The ROM validates the firmware image that corresponds to the manifest’s digital signature through the use of the public FWSK, and if successful, the system continues to boot from Flash code.”

From "Architecture Guide: Intel® Active Management Technology", 2009

## ME: Security

- ME requires some RAM to put unpacked code and runtime variables (MCU's own memory is too limited and slow)
- This memory is reserved by BIOS on ME's request and cannot be accessed by the host CPU once locked.



| | | | |
|---|---|---|---|
| 18:12 | RV | 0 | *Reserved* |
| 11 | RWO | 0 | Enable for Intel® ME memory region |
| 10 | RWO | 0 | Lock for Intel ME memory region base/mask. This bit is only cleared upon a reset. MESEGMASK and MESEGBASE cannot be changed once this bit is set. |
| 9:0 | RV | 0 | *Reserved* |

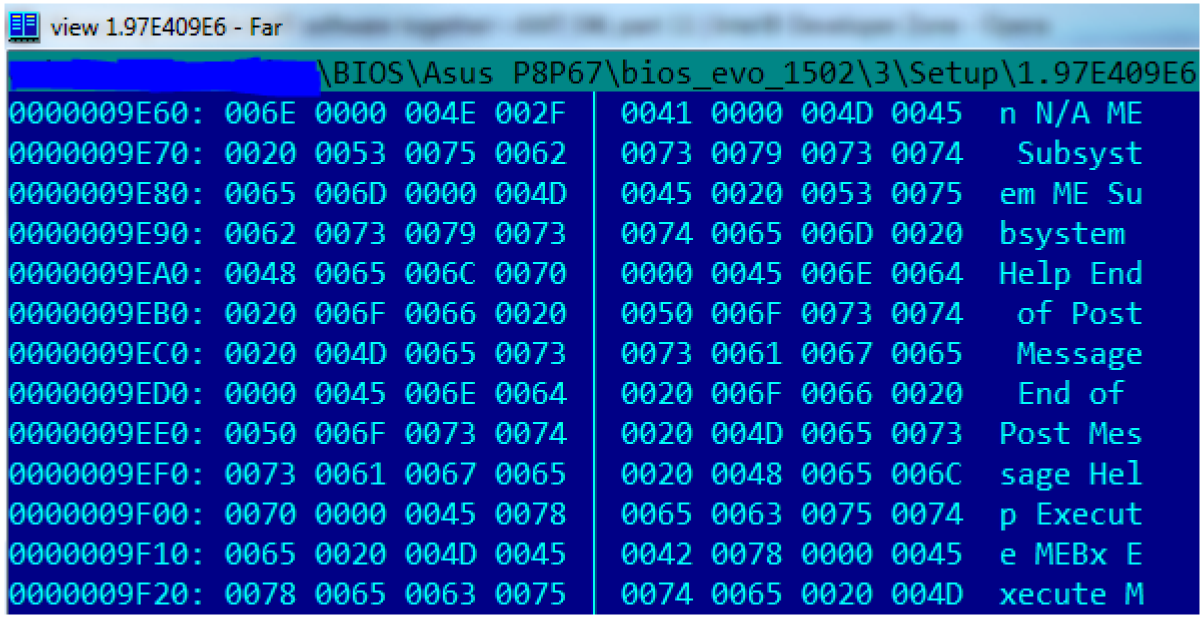
- A memory remapping attack was demonstrated by Invisible Things Lab in 2009, but it doesn't work anymore
- Cold boot attack is probably still possible, though...
- An open question: how does it work with the integrated memory controller on the newer chips?

## ME: Security

- Flash access is limited by the chipset according to the flags in the descriptor region (start of the flash chip), and normally ME region is not accessible to others
- Since the descriptor region itself is marked read-only at end of manufacturing, changing permissions is not trivial
- One obvious solution is to use a hardware flash programmer to write to the chip directly, bypassing CPU and chipset. This might require unsoldering the chip, however
- Another option is the HMRFPO message which asks ME to unlock the flash temporarily, but it's tricky to use because it only works before End-Of-POST

## Getting along with the BIOS

- I decided to find a place where the BIOS sends End-Of-Post to the ME
- Extracted BIOS with 7-zip (UEFI Firmware Filesystem)
- Searched for "Post", both ANSI and Unicode
- A strange file appears...



```
view 1.97E409E6 - Far [redacted]
[redacted]\BIOS\Asus P8P67\bios_evo_1502\3\Setup\1.97E409E6
0000009E60: 006E 0000 004E 002F │ 0041 0000 004D 0045  n N/A ME
0000009E70: 0020 0053 0075 0062 │ 0073 0079 0073 0074   Subsyst
0000009E80: 0065 006D 0000 004D │ 0045 0020 0053 0075  em ME Su
0000009E90: 0062 0073 0079 0073 │ 0074 0065 006D 0020  bsystem
0000009EA0: 0048 0065 006C 0070 │ 0000 0045 006E 0064  Help End
0000009EB0: 0020 006F 0066 0020 │ 0050 006F 0073 0074   of Post
0000009EC0: 0020 004D 0065 0073 │ 0073 0061 0067 0065   Message
0000009ED0: 0000 0045 006E 0064 │ 0020 006F 0066 0020   End of
0000009EE0: 0050 006F 0073 0074 │ 0020 004D 0065 0073  Post Mes
0000009EF0: 0073 0061 0067 0065 │ 0020 0048 0065 006C  sage Hel
0000009F00: 0070 0000 0045 0078 │ 0065 0063 0075 0074  p Execut
0000009F10: 0065 0020 004D 0045 │ 0042 0078 0000 0045  e MEBx E
0000009F20: 0078 0065 0063 0075 │ 0074 0065 0020 004D  xecute M
```

## Getting along with the BIOS

- The file contains bunch of Unicode strings, first in English then in couple of other languages
- The strings refer to the BIOS setup items
- File appears next to a .efi executable, meaning they were two sections of the flash file "Setup".
- So obviously this is a kind of a resource file for the BIOS setup UI
- Turns out that (U)EFI provides a standard way to encode strings and forms for UI, called HII (Human Interface Infrastructure)
- And someone already wrote tools[1] to parse them...

[1] http://marcansoft.com/blog/2009/06/enabling-intel-vt-on-the-aspire-8930g/

## Getting along with the BIOS

- After some hacking of the scripts (apparently there were some updates in the format) dumped a list of strings and forms
- And here's the option we need:

```
Suppress If
 EQ [0xdb<1>] == 0x0
 One Of [0xdc<1>] u'End of Post Message'
 \Help text: u'End of Post Message Help'
  Option 'Disabled' = 0x0 Flags 0x10 Key 0x0
  Option 'Enabled' = 0x1 Flags 0x13 Key 0x0
 End One Of
End If
```

- However, it doesn't seem to be present in the actual UI?

## Getting along with the BIOS

- This setting is a part of a form named 'ME Subsystem'
- Scrolling a bit around, we find:

```
Suppress If
 LIST [0xdb<1>] in (0x0,0x1)
 Reference: 'ME Subsystem' Form ID 0x1a Flags 0x0 Key 0x0
 \Help text: u'ME Subsystem Parameters'
End If
```

- So, the form is not shown if the byte in the Setup variable at offset 0xDB is either 0 *or* 1.
- One solution is to patch the form bytecode, pack the file back into the BIOS (updating the checksums) and flash the new BIOS
- But this is rather involved and risky. Is there an easier way?

## Getting along with the BIOS

- Examining and editing UEFI variables is rather awkward but doable with the EFI shell and command "dmpstore"

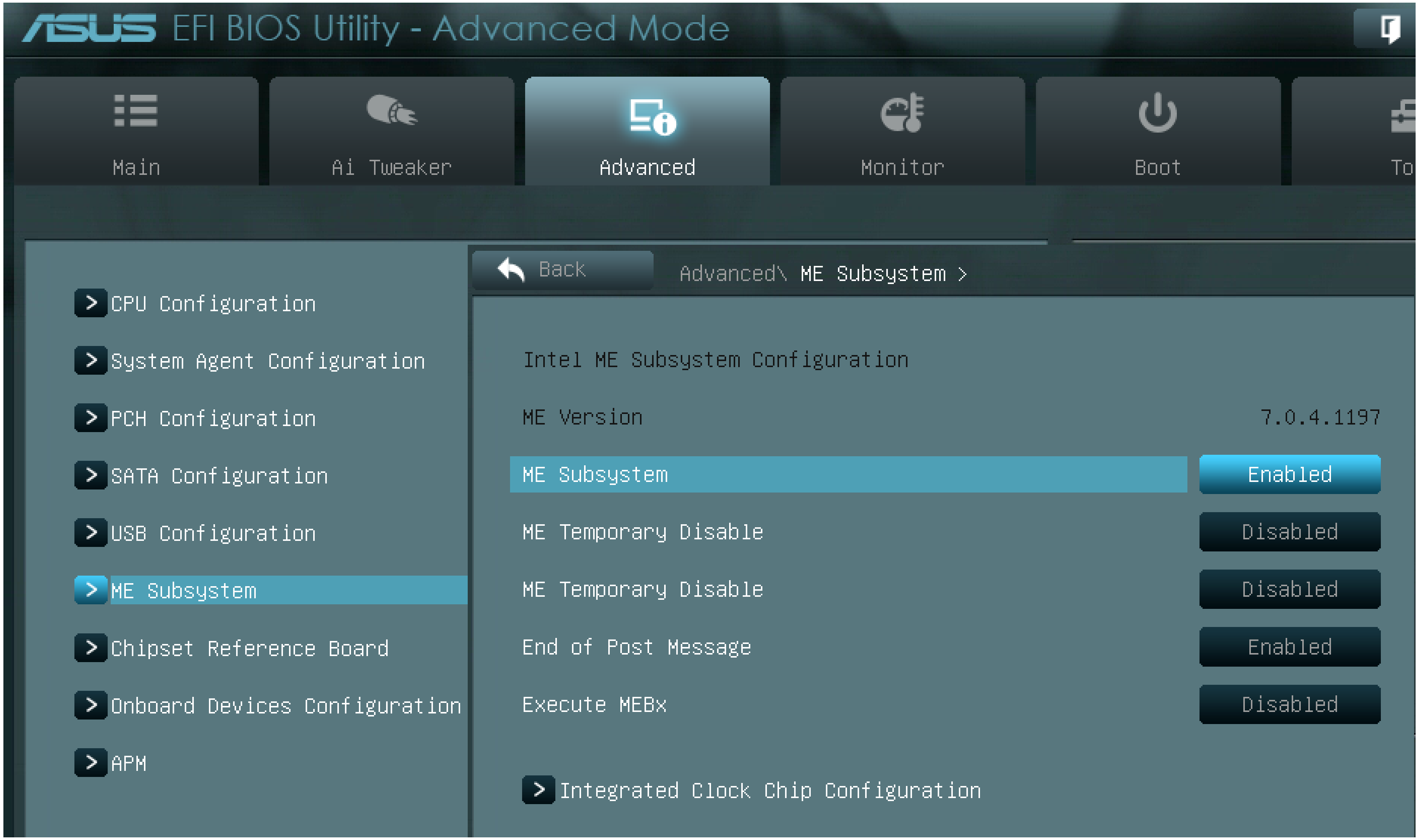
```
              0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
000000d0   00 00 00 00 00 00 01 00  00 00 00 01 01 00 00 01
```

- Changing EFI vars is much easier than patching actual files in the FFS. Also, no need to reflash.
- Since neither 0 nor 1 will show the form, let's put something else in there... for example, 0xFF

```
> dmpstore Setup -s temp.bin
> hexedit temp.bin
> dmpstore Setup -l temp.bin
> exit
```

- Did it work?

## Getting along with the BIOS



EFI BIOS Utility - Advanced Mode

- > CPU Configuration
- > System Agent Configuration
- > PCH Configuration
- > SATA Configuration
- > USB Configuration
- > ME Subsystem
- > Chipset Reference Board
- > Onboard Devices Configuration
- > APM

Back    Advanced\ ME Subsystem >

Intel ME Subsystem Configuration

| | |
|---|---|
| ME Version | 7.0.4.1197 |
| ME Subsystem | Enabled |
| ME Temporary Disable | Disabled |
| ME Temporary Disable | Disabled |
| End of Post Message | Enabled |
| Execute MEBx | Disabled |

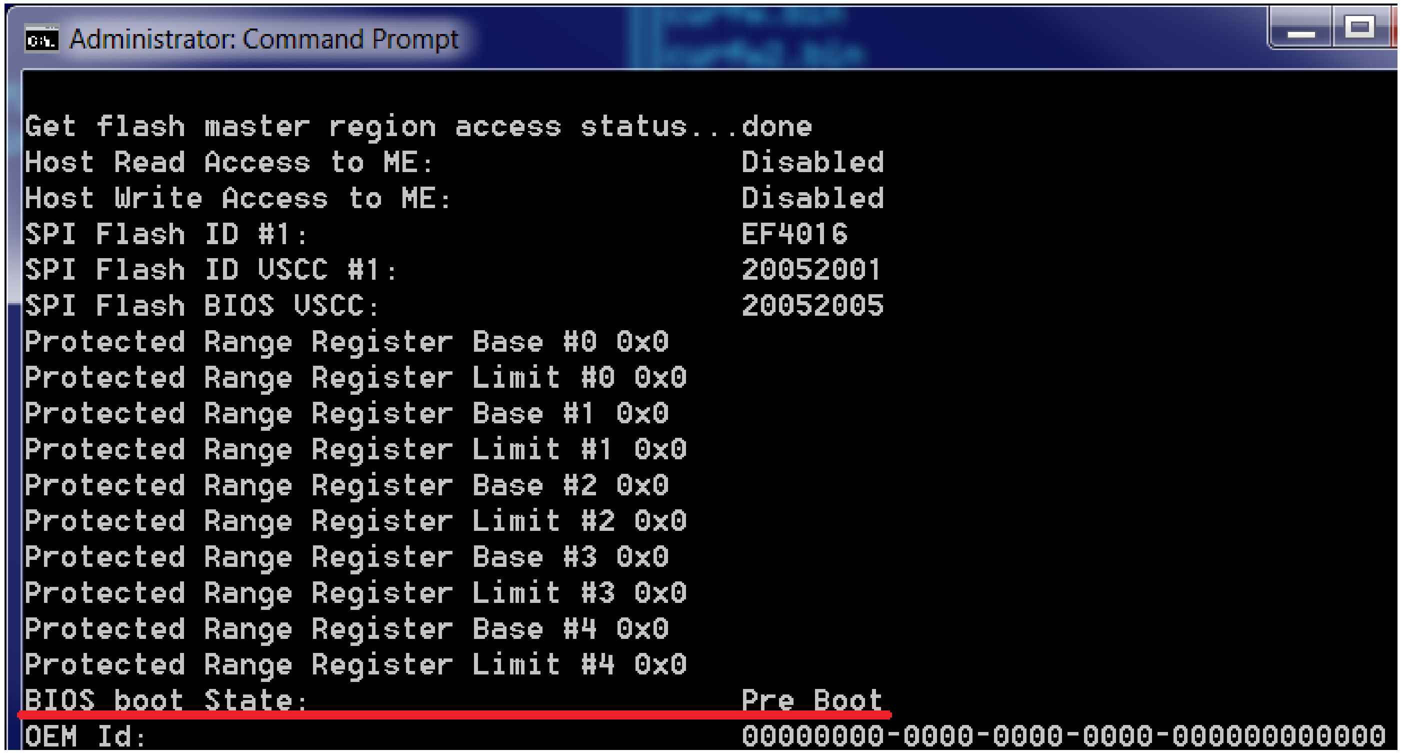
> Integrated Clock Chip Configuration

## Getting along with the BIOS

- Now we can change the ME options and disable End-Of-POST
- One minor issue: when you return from the "ME Subsystem" form, the menu item disappears :)
- This happens because the byte 0xDB gets set to 1 (or 0, if you disable ME) again, triggering the "Suppress If" opcode
- So if you need to go there once more, you need to do the dmpstore/hexedit trick again
- By the way, instead of going through the menus we could directly set the necessary value in the Setup variable (byte at offset 0xDC)

## Getting along with the BIOS

- Rebooting after changing "End of Post Message" to "Disabled":


Administrator: Command Prompt

```
Get flash master region access status...done
Host Read Access to ME:                 Disabled
Host Write Access to ME:                Disabled
SPI Flash ID #1:                        EF4016
SPI Flash ID VSCC #1:                   20052001
SPI Flash BIOS VSCC:                    20052005
Protected Range Register Base #0 0x0
Protected Range Register Limit #0 0x0
Protected Range Register Base #1 0x0
Protected Range Register Limit #1 0x0
Protected Range Register Base #2 0x0
Protected Range Register Limit #2 0x0
Protected Range Register Base #3 0x0
Protected Range Register Limit #3 0x0
Protected Range Register Base #4 0x0
Protected Range Register Limit #4 0x0
BIOS boot State:                        Pre Boot
OEM Id:                                 00000000-0000-0000-0000-000000000000
```

## Getting along with the BIOS

- Okay, we have our ME in desired state, what now?
- The specifics of the HMRFPO message are not available in public documentation
- However, some BIOS updates exist that allow updating ME version from 7.0 to 8.0
- ME cannot update itself to the next major version, so this must be done by external (to the ME) code
- From reading the "Bios ME7 to ME8 update SOP" for MSI boards it's clear that the ME update happens on the first boot of the new BIOS
- So the code must be there somewhere...

## Getting along with the BIOS

- Several days of reversing later...
- Found the new ME partition (stored as a file in the UEFI volume)
- Found the code that does the ME update ("Updating BIOS ME, please wait")
- Found code which seems to talk to ME and send commands not mentioned in documentation
- Found code in ME which handles these messages (probably)
- Converted an AMT SDK sample to send similar commands
- Unfortunately, didn't work on my test hardware (ASUS)
- However, it was a good learning experience!

## A different approach

- After I went through this, I accidentally found a mention that the newest BIOS for my board contains ME 8.0 (this fact was not mentioned in Asus' release notes)
- As a nice side effect, this update completely opens the ME region!
- So now I can read and write the ME region freely (using Intel's FPT)
- I can also analyze the update process in more detail and figure out how it works around the ME lock on the old version

## Poking the flash

- One of the partitions in the ME region is "EFFS"
- It contains in turn other, virtual partitions with tags beginning with "NV" (non-volatile variables) and "BI" (block I/O), used by the software components of ME
- Some of these variables are used to enable and disable various ME features which usually depend on the specific chipset model (a single ME binary is used on many configurations)
- For example, ME on my board includes modules TDT (Anti-Theft) and PAVP (Protected Audio/Video Path), but they're disabled in software
- I tried changing some obvious bits, but it seems it's not that simple...

## Results

- I have not managed to run my own rootkit on the ME (yet)
- However, I've learned a lot about it and I hope to achieve it in future
- Intel seems to have done a good job on security so far, but there's a *lot* of code in there (now up to 5MB, *compressed*)
- I made some tools that should help others in research:
    - ME ROM dumper/extractor
    - ARC processor module for IDA

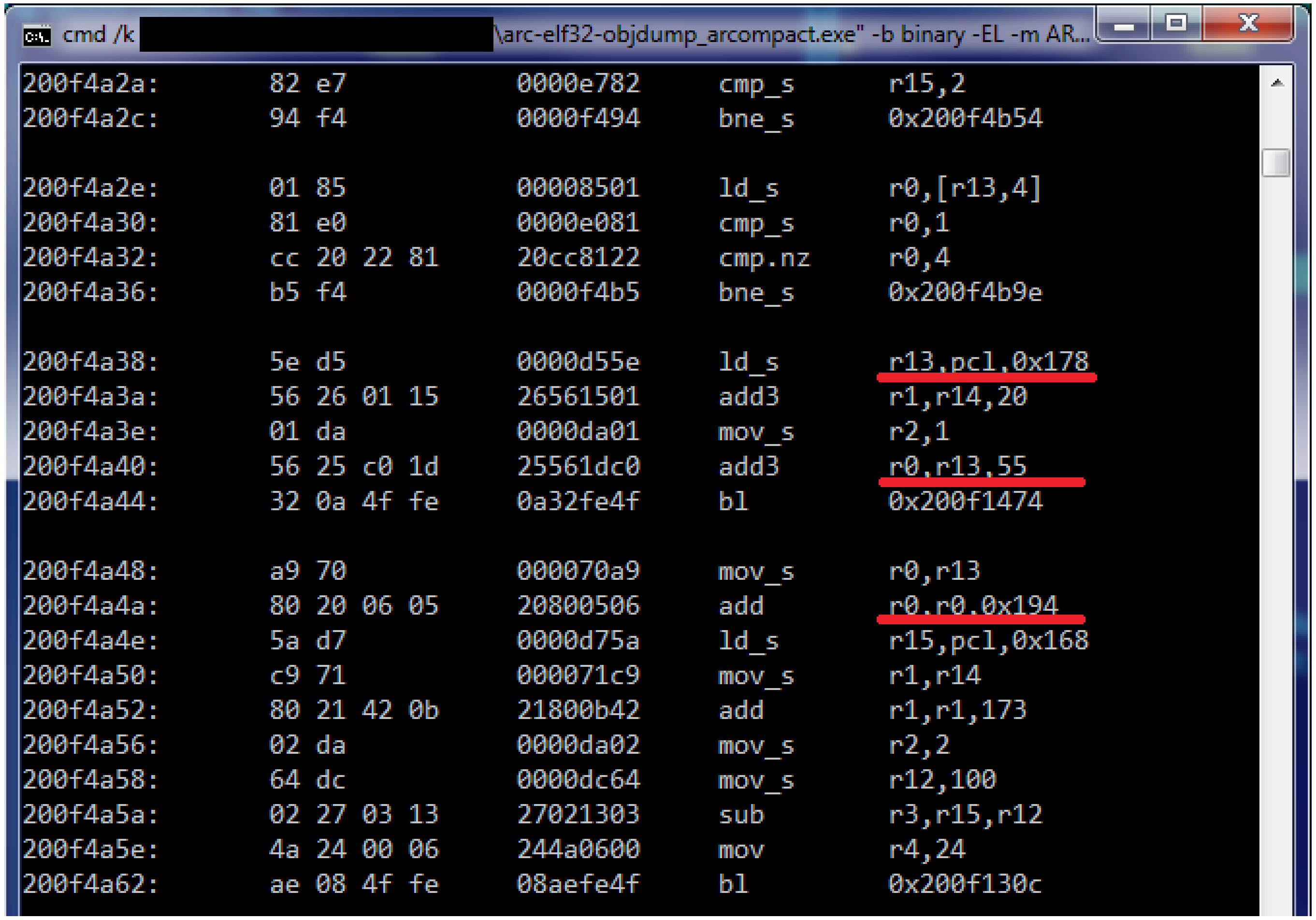
## ME dumper/extractor

- Written in Python
- Supports parsing of the following formats:
  - Full SPI flash image (signature 5A A5 F0 0F)
  - Separate ME region (signature $FPT)
  - Individual ME code module ($MN2 or $MAN)
- Prints detailed header info
- prepares LZMA-compressed modules for easy unpacking with 7-zip

## ME dumper/extractor

```
Header tag:      $MME
Module name:     JOM
Hash:            FB 49 10 CB 04 C8 62 9D BE 53 BB 7A CF 0C
6A D4 1F F9 92 A7 AD 52 2A 55 FE F6 71 74 06 F0 0C 64
Unk34:           0x20157000
Offset:          0x0001198E
Unk38:           0x00029000
Data length:     0x000133CA
Unk44:           0x00029518
Unk48:           0x00029518
LoadBase:        0x20159000
Flags:           0x0012D42A
    Unknown B0:      0
    Power Type:      POWER_TYPE_M0_ONLY (1)
    Unknown B3:      1
    Compression:     COMP_TYPE_LZMA (2)
    Stage:           STAGE 8 (8)
    API Type:        API_TYPE_KERNEL (2)
    Unknown B14:     1
    Unknown B15:     1
    Privileged:      0
    Unknown B17_19: 1
    Unknown B20_21: 1
```

# ARC processor: objdump

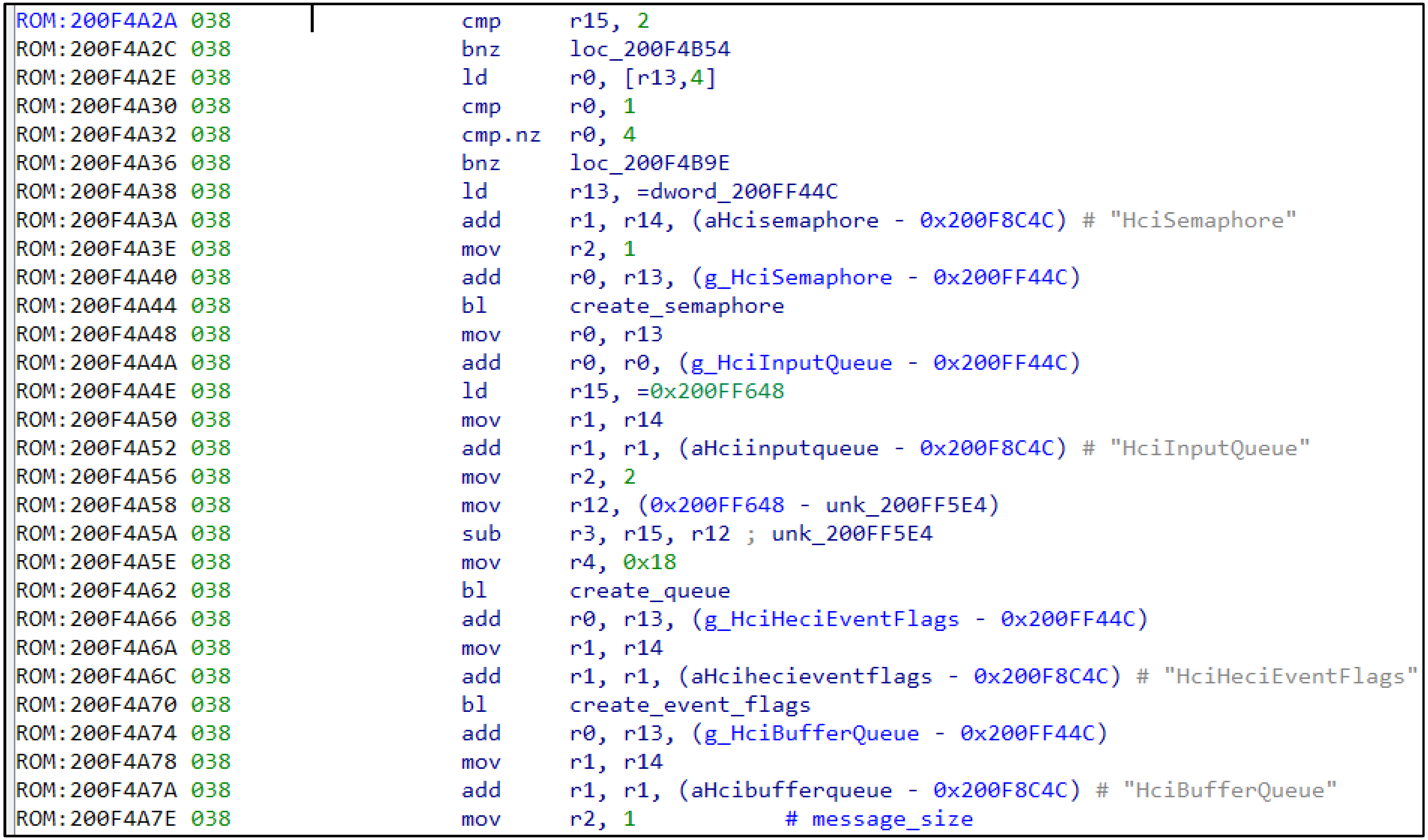


```
cmd /k [redacted]\arc-elf32-objdump_arcompact.exe" -b binary -EL -m AR...
```

```
200f4a2a:       82 e7           0000e782    cmp_s     r15,2
200f4a2c:       94 f4           0000f494    bne_s     0x200f4b54

200f4a2e:       01 85           00008501    ld_s      r0,[r13,4]
200f4a30:       81 e0           0000e081    cmp_s     r0,1
200f4a32:       cc 20 22 81     20cc8122    cmp.nz    r0,4
200f4a36:       b5 f4           0000f4b5    bne_s     0x200f4b9e

200f4a38:       5e d5           0000d55e    ld_s      r13,pcl,0x178
200f4a3a:       56 26 01 15     26561501    add3      r1,r14,20
200f4a3e:       01 da           0000da01    mov_s     r2,1
200f4a40:       56 25 c0 1d     25561dc0    add3      r0,r13,55
200f4a44:       32 0a 4f fe     0a32fe4f    bl        0x200f1474

200f4a48:       a9 70           000070a9    mov_s     r0,r13
200f4a4a:       80 20 06 05     20800506    add       r0,r0,0x194
200f4a4e:       5a d7           0000d75a    ld_s      r15,pcl,0x168
200f4a50:       c9 71           000071c9    mov_s     r1,r14
200f4a52:       80 21 42 0b     21800b42    add       r1,r1,173
200f4a56:       02 da           0000da02    mov_s     r2,2
200f4a58:       64 dc           0000dc64    mov_s     r12,100
200f4a5a:       02 27 03 13     27021303    sub       r3,r15,r12
200f4a5e:       4a 24 00 06     244a0600    mov       r4,24
200f4a62:       ae 08 4f fe     08aefe4f    bl        0x200f130c
```

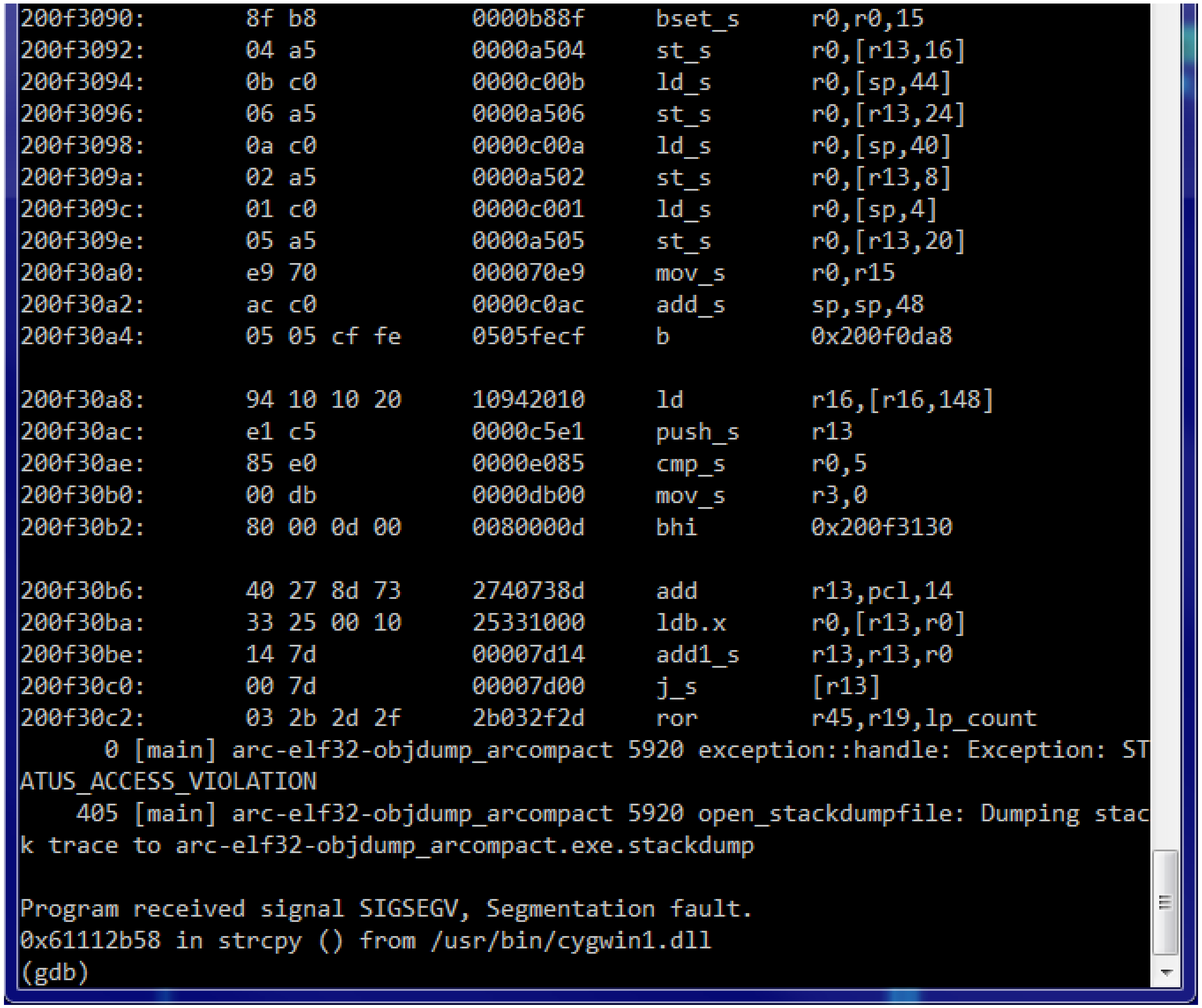
What are the results of underlined instructions?

## ARC processor: IDA



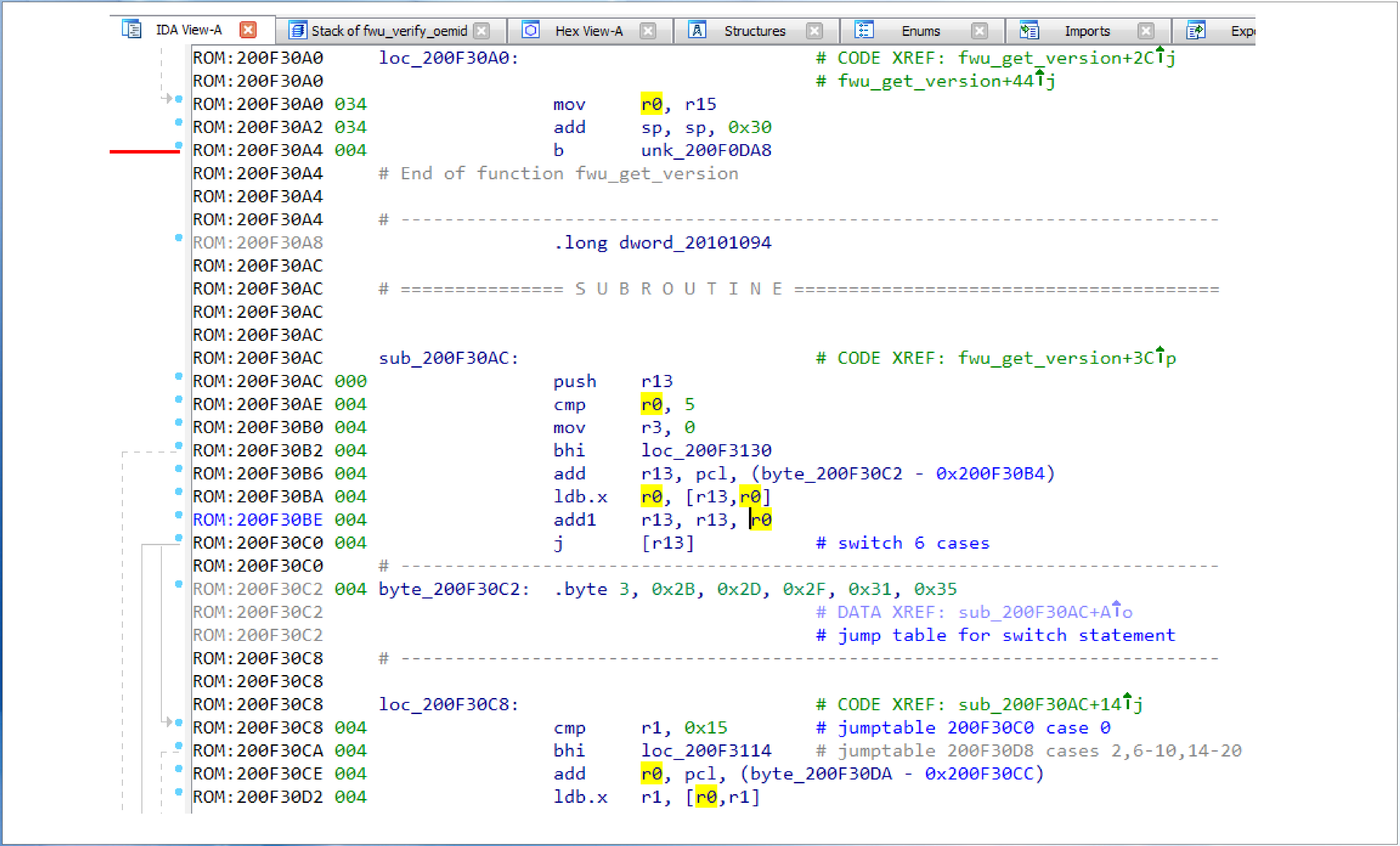
```
ROM:200F4A2A 038                cmp     r15, 2
ROM:200F4A2C 038                bnz     loc_200F4B54
ROM:200F4A2E 038                ld      r0, [r13,4]
ROM:200F4A30 038                cmp     r0, 1
ROM:200F4A32 038                cmp.nz  r0, 4
ROM:200F4A36 038                bnz     loc_200F4B9E
ROM:200F4A38 038                ld      r13, =dword_200FF44C
ROM:200F4A3A 038                add     r1, r14, (aHcisemaphore - 0x200F8C4C) # "HciSemaphore"
ROM:200F4A3E 038                mov     r2, 1
ROM:200F4A40 038                add     r0, r13, (g_HciSemaphore - 0x200FF44C)
ROM:200F4A44 038                bl      create_semaphore
ROM:200F4A48 038                mov     r0, r13
ROM:200F4A4A 038                add     r0, r0, (g_HciInputQueue - 0x200FF44C)
ROM:200F4A4E 038                ld      r15, =0x200FF648
ROM:200F4A50 038                mov     r1, r14
ROM:200F4A52 038                add     r1, r1, (aHciinputqueue - 0x200F8C4C) # "HciInputQueue"
ROM:200F4A56 038                mov     r2, 2
ROM:200F4A58 038                mov     r12, (0x200FF648 - unk_200FF5E4)
ROM:200F4A5A 038                sub     r3, r15, r12 ; unk_200FF5E4
ROM:200F4A5E 038                mov     r4, 0x18
ROM:200F4A62 038                bl      create_queue
ROM:200F4A66 038                add     r0, r13, (g_HciHeciEventFlags - 0x200FF44C)
ROM:200F4A6A 038                mov     r1, r14
ROM:200F4A6C 038                add     r1, r1, (aHcihecieventflags - 0x200F8C4C) # "HciHeciEventFlags"
ROM:200F4A70 038                bl      create_event_flags
ROM:200F4A74 038                add     r0, r13, (g_HciBufferQueue - 0x200FF44C)
ROM:200F4A78 038                mov     r1, r14
ROM:200F4A7A 038                add     r1, r1, (aHcibufferqueue - 0x200F8C4C) # "HciBufferQueue"
ROM:200F4A7E 038                mov     r2, 1            # message_size
```

# ARC processor: objdump



```
200f3090:       8f b8           0000b88f        bset_s      r0,r0,15
200f3092:       04 a5           0000a504        st_s        r0,[r13,16]
200f3094:       0b c0           0000c00b        ld_s        r0,[sp,44]
200f3096:       06 a5           0000a506        st_s        r0,[r13,24]
200f3098:       0a c0           0000c00a        ld_s        r0,[sp,40]
200f309a:       02 a5           0000a502        st_s        r0,[r13,8]
200f309c:       01 c0           0000c001        ld_s        r0,[sp,4]
200f309e:       05 a5           0000a505        st_s        r0,[r13,20]
200f30a0:       e9 70           000070e9        mov_s       r0,r15
200f30a2:       ac c0           0000c0ac        add_s       sp,sp,48
200f30a4:       05 05 cf fe     0505fecf        b           0x200f0da8

200f30a8:       94 10 10 20     10942010        ld          r16,[r16,148]
200f30ac:       e1 c5           0000c5e1        push_s      r13
200f30ae:       85 e0           0000e085        cmp_s       r0,5
200f30b0:       00 db           0000db00        mov_s       r3,0
200f30b2:       80 00 0d 00     0080000d        bhi         0x200f3130

200f30b6:       40 27 8d 73     2740738d        add         r13,pcl,14
200f30ba:       33 25 00 10     25331000        ldb.x       r0,[r13,r0]
200f30be:       14 7d           00007d14        add1_s      r13,r13,r0
200f30c0:       00 7d           00007d00        j_s         [r13]
200f30c2:       03 2b 2d 2f     2b032f2d        ror         r45,r19,lp_count
      0 [main] arc-elf32-objdump_arcompact 5920 exception::handle: Exception: ST
ATUS_ACCESS_VIOLATION
    405 [main] arc-elf32-objdump_arcompact 5920 open_stackdumpfile: Dumping stac
k trace to arc-elf32-objdump_arcompact.exe.stackdump

Program received signal SIGSEGV, Segmentation fault.
0x61112b58 in strcpy () from /usr/bin/cygwin1.dll
(gdb)
```

# ARC processor: IDA



```
ROM:200F30A0      loc_200F30A0:                               # CODE XREF: fwu_get_version+2C↑j
ROM:200F30A0                                                  # fwu_get_version+44↑j
ROM:200F30A0 034                  mov     r0, r15
ROM:200F30A2 034                  add     sp, sp, 0x30
ROM:200F30A4 004                  b       unk_200F0DA8
ROM:200F30A4      # End of function fwu_get_version
ROM:200F30A4
ROM:200F30A4      # ---------------------------------------------------------------------------
ROM:200F30A8                      .long dword_20101094
ROM:200F30AC
ROM:200F30AC      # =============== S U B R O U T I N E =======================================
ROM:200F30AC
ROM:200F30AC
ROM:200F30AC      sub_200F30AC:                               # CODE XREF: fwu_get_version+3C↑p
ROM:200F30AC 000                  push    r13
ROM:200F30AE 004                  cmp     r0, 5
ROM:200F30B0 004                  mov     r3, 0
ROM:200F30B2 004                  bhi     loc_200F3130
ROM:200F30B6 004                  add     r13, pcl, (byte_200F30C2 - 0x200F30B4)
ROM:200F30BA 004                  ldb.x   r0, [r13,r0]
ROM:200F30BE 004                  add1    r13, r13, r0
ROM:200F30C0 004                  j       [r13]           # switch 6 cases
ROM:200F30C0      # ---------------------------------------------------------------------------
ROM:200F30C2 004 byte_200F30C2:   .byte 3, 0x2B, 0x2D, 0x2F, 0x31, 0x35
ROM:200F30C2                                                  # DATA XREF: sub_200F30AC+A↑o
ROM:200F30C2                                                  # jump table for switch statement
ROM:200F30C8      # ---------------------------------------------------------------------------
ROM:200F30C8
ROM:200F30C8      loc_200F30C8:                               # CODE XREF: sub_200F30AC+14↑j
ROM:200F30C8 004                  cmp     r1, 0x15        # jumptable 200F30C0 case 0
ROM:200F30CA 004                  bhi     loc_200F3114    # jumptable 200F30D8 cases 2,6-10,14-20
ROM:200F30CE 004                  add     r0, pcl, (byte_200F30DA - 0x200F30CC)
ROM:200F30D2 004                  ldb.x   r1, [r0,r1]
```

## ARC processor module for IDA

- Supports ARCTangent-A4 (older, 32-bit only instructions) and ARCompact (newer, mixed 32/16-bit ISA)
- Tracks changes to SP register and creates local variables
- Handles switch tables
- Tracks register values to find more cross-references
- Inlines constant pool loads (PC-relative) for convenience
- In general, makes life not constant pain
- Not production quality yet, but hopefully will appear in the next version of IDA

## Future work

- Dynamic Application Loader
  - New feature in 7.1/8.0 firmware: load **Java applets** and run them inside ME
  - Used for things like PIN entry UI and remote authentication
  - The applets provided by Intel are signed, but it's one more vector of entry...
- EFFS parsing and modifying
  - Most of the ME state is stored there
  - If we can modify flash, we can modify EFFS
  - Critical variables are protected from tampering but the majority isn't
  - Complicated format because of flash wear leveling

## Future work

- Huffman compression
  - Used in newer firmwares for compressing the kernel and some other modules
  - Couldn't find decompression code; some whitepapers mention hardware decompression...
  - Still, Huffman is a pretty simple protocol, so should be doable from just the compressed data
- ME ↔ Host protocols
  - Most modules use different message format
  - A lot of undocumented messages; some modules seem to be not mentioned anywhere
  - Some client software has very verbose debugging messages in their binaries...

## Future work

- BIOS RE
  - In early boot stages ME accepts some things which are not possible later
  - Reversing BIOS modules that talk to ME is a good source of info
  - Even better would be to run custom code early
  - Big room for improvement in tools
- Simulation and fuzzing
  - Open Virtual Platform (www.ovpworld.org) has modules for ARC600 and ARC700 (ARCompact-based)
  - They claim that it's easy to extend the models with emulation for custom hardware
  - The simulator has GDB stub for debugging
  - Debugging and fuzzing should be possible

## References and links

http://software.intel.com/en-us/articles/architecture-guide-intel-active-management-technology/

http://software.intel.com/sites/manageability/AMT_Implementation_and_Reference_Guide/

http://theinvisiblethings.blogspot.com/2009/08/vegas-toys-part-i-ring-3-tools.html

http://download.intel.com/technology/itj/2008/v12i4/paper[1-10].pdf

http://web.it.kth.se/~maguire/DEGREE-PROJECT-REPORTS/100402-Vassilios_Ververis-with-cover.pdf

http://www.thefengs.com/wuchang/work/courses/cs592/cs592_spring2007/

http://www.stewin.org/papers/dimvap15-stewin.pdf

http://www.stewin.org/techreports/pstewin_spring2011.pdf

http://www.stewin.org/slides/pstewin-SPRING6-EvaluatingRing-3Rootkits.pdf

http://marcansoft.com/blog/2009/06/enabling-intel-vt-on-the-aspire-8930g/

http://flashrom.org/trac/flashrom/browser/trunk/Documentation/mysteries_intel.txt

http://review.coreboot.org/gitweb?p=coreboot.git;a=blob;f=src/southbridge/intel/bd82x6x/me.c

http://download.intel.com/technology/product/DCMI/DCMI-HI_1_0.pdf

## Thank you!

**Questions?**

igor@hex-rays.com
skochinsky@gmail.com