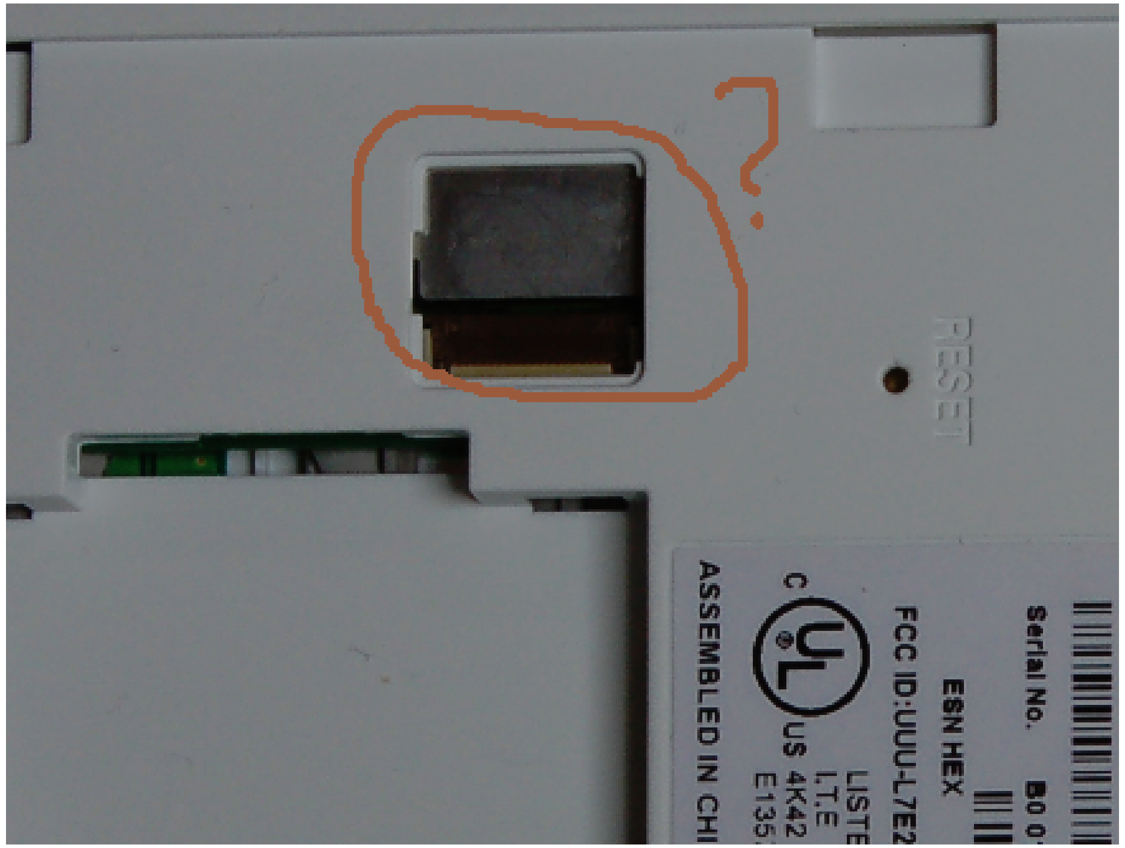
INTEL ME:
MYTHS AND REALITY

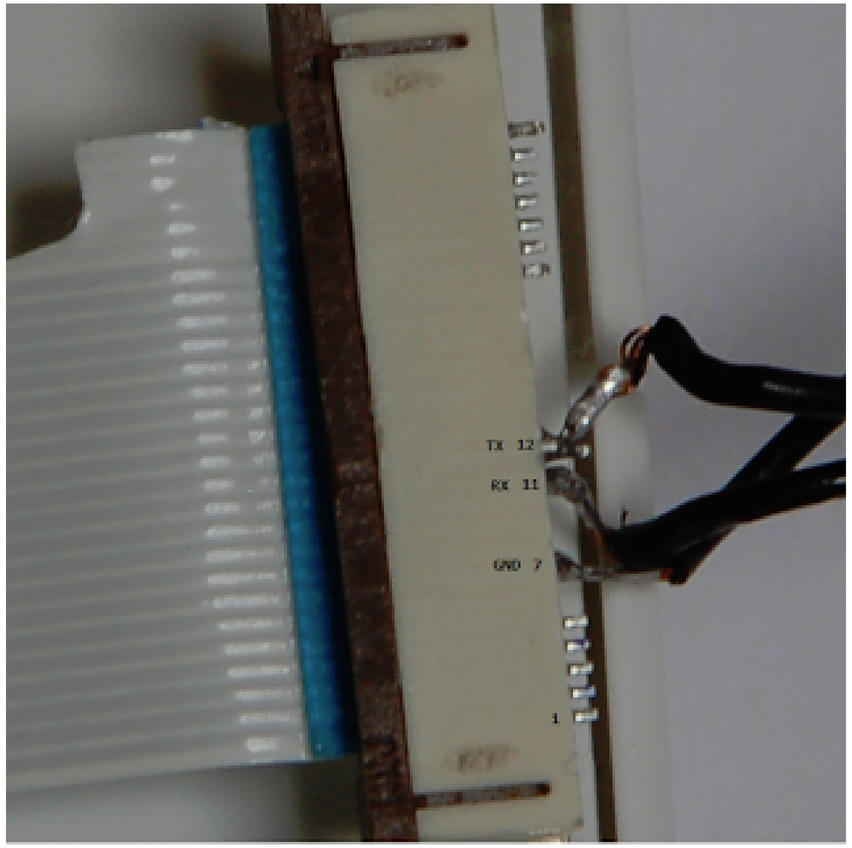
ABOUT SPEAKERS

Igor Skochinsky

- Hobby reverse engineer
- Software developer at Hex-Rays since 2008
- NOT a security researcher

Did a bit of e-Reader hacking



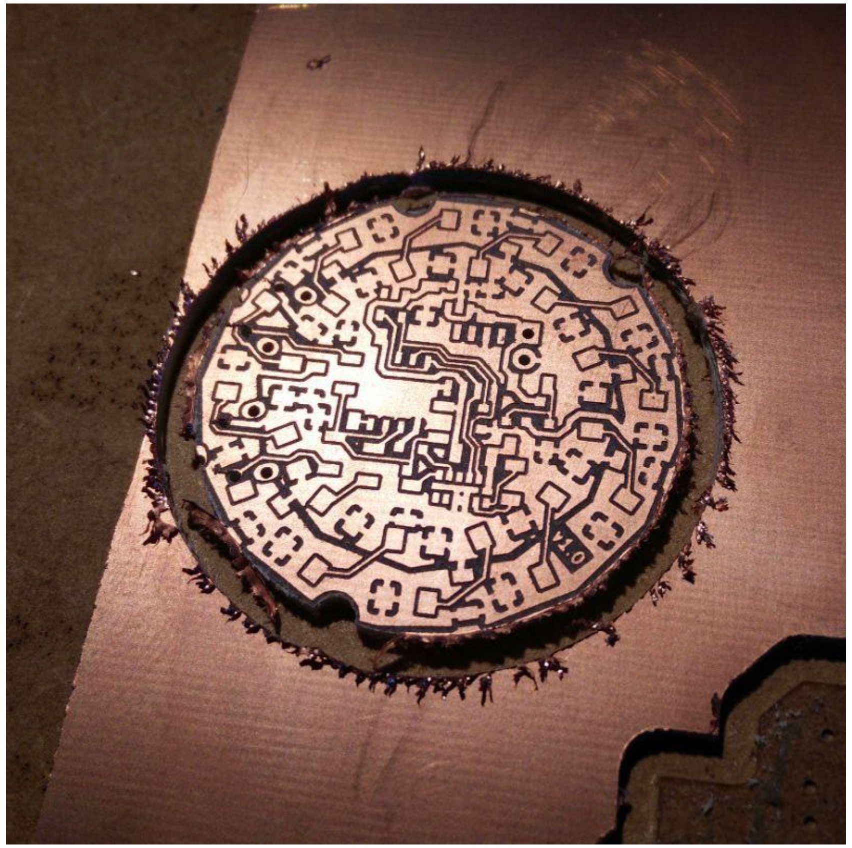


ABOUT SPEAKERS

Nicola Corna

- Hobby reverse engineer
- Interested in DIY hardware
- Electronics Engineering student at the Politecnico di Milano
- NOT a security researcher

PCB milling



DISCLAIMER

The contents of this talk is based on public information sources and our own reverse engineering. We can't be certain that our conclusions are 100% correct and may turn out wrong in future.

We do not have any NDA or "special relationship" with Intel.

INTEL ME

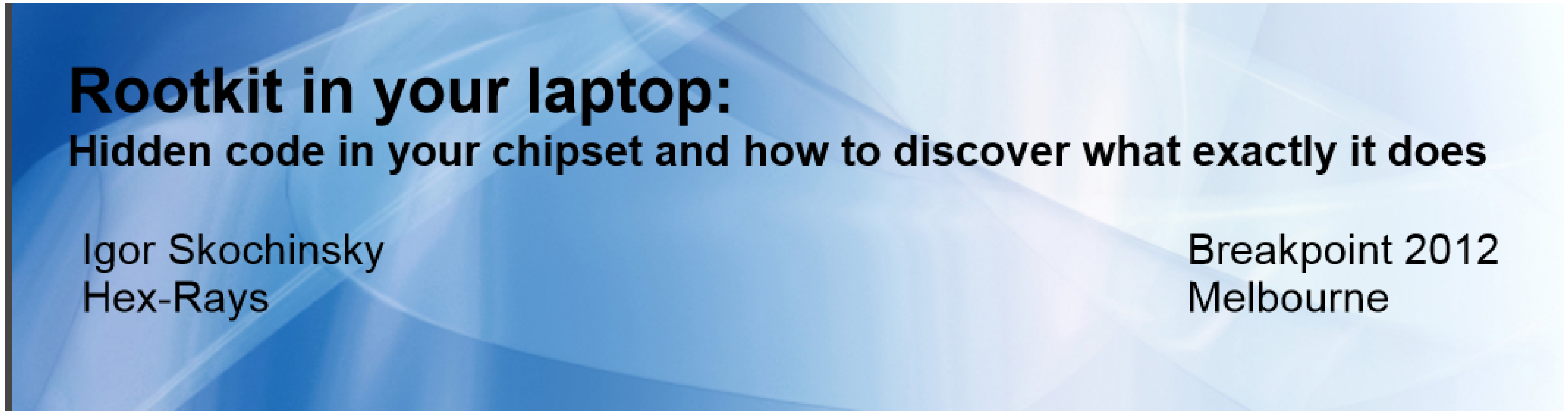
- Originally "manageability engine"
- later just "management engine"
- nowadays "converged security and manageability engine (CSME)"
- or "converged security engine (CSE)"
- "Trusted Execution Engine" (TXE) for mobile/embedded
- "Server Platform Services" (SpS) for servers

INTEL ME

> *Built into many Intel® Chipset–based platforms is a small, low-power computer subsystem [...] performs various tasks [...] must function correctly to get the most performance and capability from your PC*

from Frequently Asked Questions for the Intel® Management Engine Verification Utility

Myth 1: It's a backdoor made for NSA and serves no useful purpose

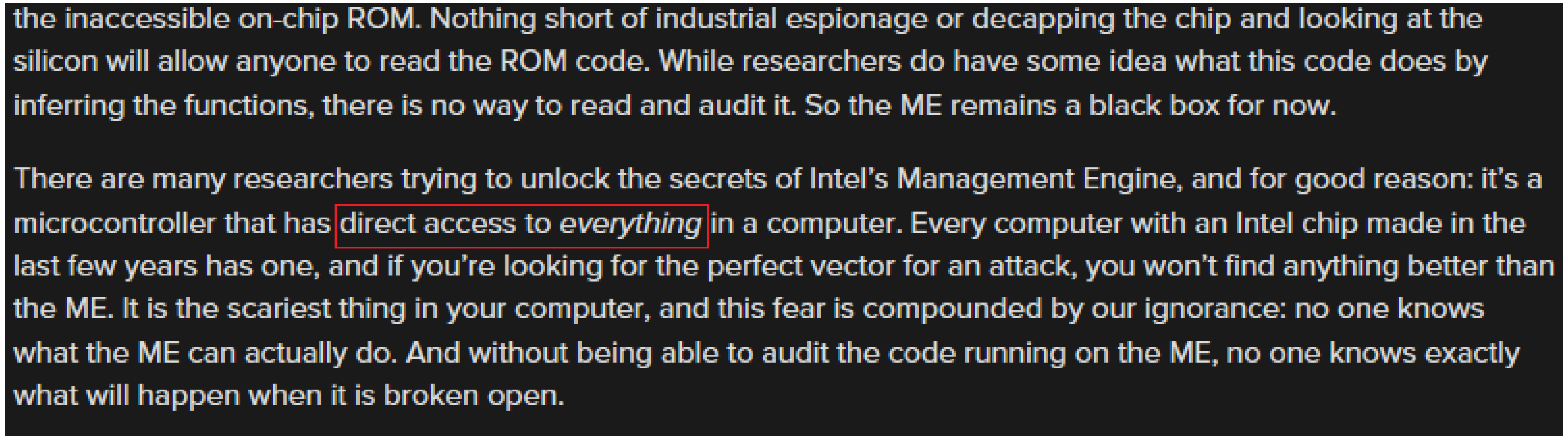


Rootkit in your laptop:

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

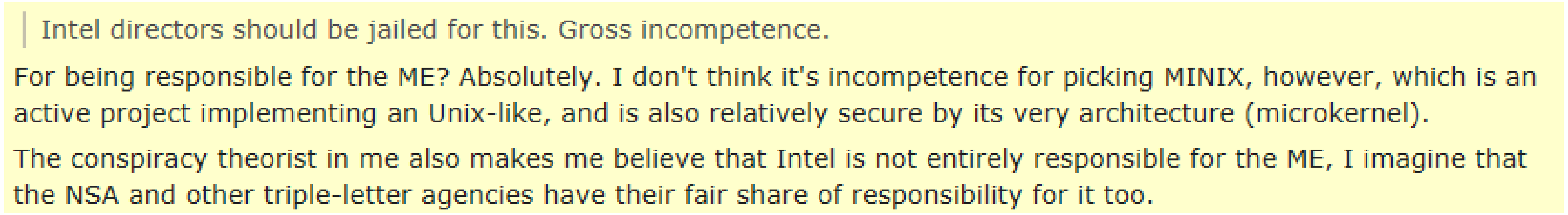
Igor Skochinsky
Hex-Rays

Breakpoint 2012
Melbourne



the inaccessible on-chip ROM. Nothing short of industrial espionage or decapping the chip and looking at the silicon will allow anyone to read the ROM code. While researchers do have some idea what this code does by inferring the functions, there is no way to read and audit it. So the ME remains a black box for now.

There are many researchers trying to unlock the secrets of Intel's Management Engine, and for good reason: it's a microcontroller that has direct access to *everything* in a computer. Every computer with an Intel chip made in the last few years has one, and if you're looking for the perfect vector for an attack, you won't find anything better than the ME. It is the scariest thing in your computer, and this fear is compounded by our ignorance: no one knows what the ME can actually do. And without being able to audit the code running on the ME, no one knows exactly what will happen when it is broken open.

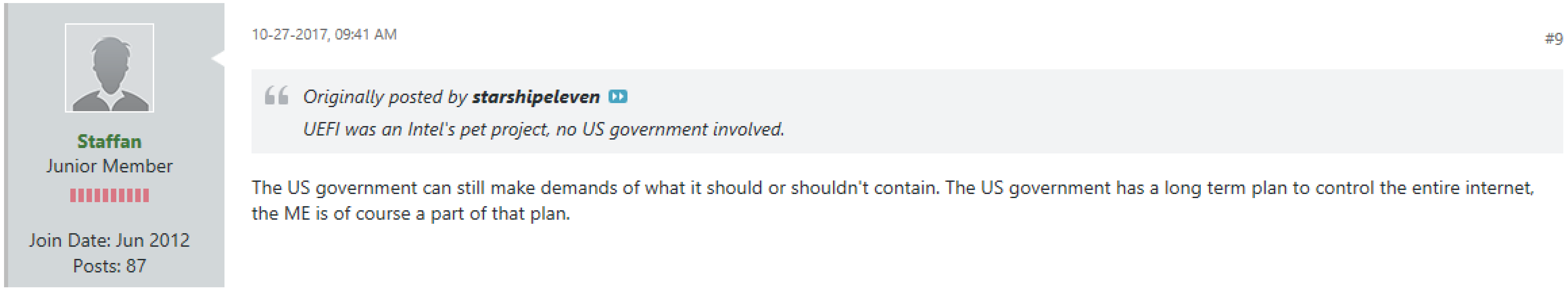


> Intel directors should be jailed for this. Gross incompetence.

For being responsible for the ME? Absolutely. I don't think it's incompetence for picking MINIX, however, which is an active project implementing an Unix-like, and is also relatively secure by its very architecture (microkernel).

The conspiracy theorist in me also makes me believe that Intel is not entirely responsible for the ME, I imagine that the NSA and other triple-letter agencies have their fair share of responsibility for it too.

from Reddit



Staffan
Junior Member

Join Date: Jun 2012
Posts: 87

10-27-2017, 09:41 AM

#9

> *Originally posted by* ***starshipeleven***
> *UEFI was an Intel's pet project, no US government involved.*

The US government can still make demands of what it should or shouldn't contain. The US government has a long term plan to control the entire internet, the ME is of course a part of that plan.

from Phoronix

```
<anon1> How can I "limit my risk" when there is a dedicated, hidden computer on top of my computer that has full access and cannot be disabled?

<anon1> There is no need for a "hardware hack". It's a built-in hardware backdoor.

<anon1> Bypasses any firewall, latches onto any wifi signal, cannot be disabled, etc.
```

(from IRC)

Well, that sounds pretty bad. But is it true?

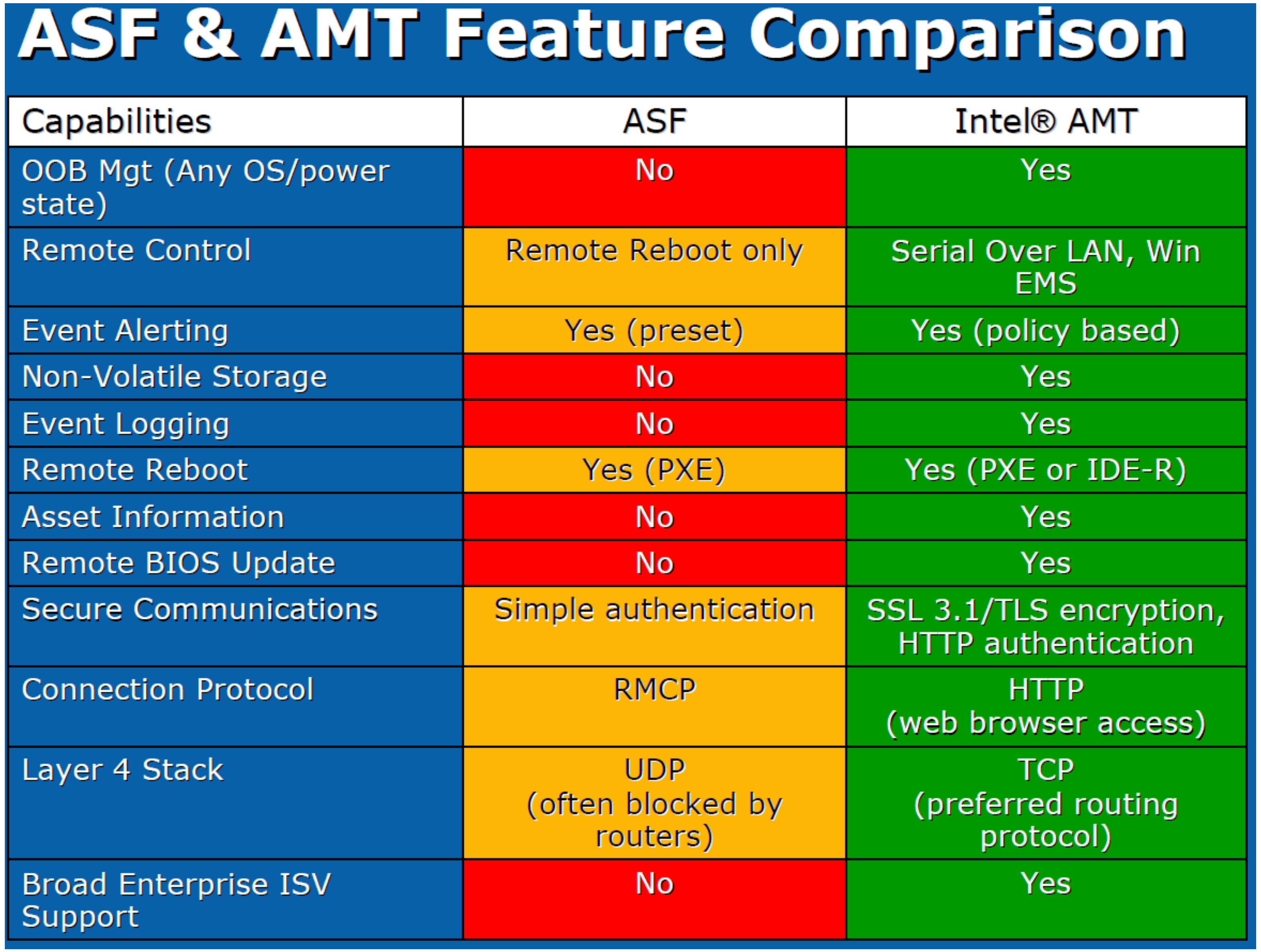
My opinion: pretty unlikely.

Firt, just because you personally don't see a purpose, does not mean there isn't one. ME was initially created to implement AMT to solve real IT problems.

A short history on remote PC management

Technology	Year	Features
KVS/KVM	1995	KBD/Video
Wired for Management	1997	WoL, PXE
Alert on LAN	1998	send alerts to central server
ASF 1.0	2001	UDP-only. no encryption
ASF 2.0	2003	added encryption

2004: AMT announced



ASF & AMT Feature Comparison

Capabilities	ASF	Intel® AMT
OOB Mgt (Any OS/power state)	No	Yes
Remote Control	Remote Reboot only	Serial Over LAN, Win EMS
Event Alerting	Yes (preset)	Yes (policy based)
Non-Volatile Storage	No	Yes
Event Logging	No	Yes
Remote Reboot	Yes (PXE)	Yes (PXE or IDE-R)
Asset Information	No	Yes
Remote BIOS Update	No	Yes
Secure Communications	Simple authentication	SSL 3.1/TLS encryption, HTTP authentication
Connection Protocol	RMCP	HTTP (web browser access)
Layer 4 Stack	UDP (often blocked by routers)	TCP (preferred routing protocol)
Broad Enterprise ISV Support	No	Yes

Short history of AMT/ME

- 2005: AMT/ME 1.0
  - ARC CPU inside Tekoa LAN. IDE-R, SoL, SOAP API.
- 2006: AMT/ME 2.0
  - ME moves into GMCH (North bridge).
- 2007: 3.0(desktop)/ 4.0 (mobile)
  - first variants without AMT. QST, iTPM appears.
  - 4.1 adds TDT (Theft Deterrence Technology) aka Anti-Theft

- 2009: AMT/ME 6.0
  - first PCH platform, ME Gen 2 (ARC600 CPU)
  - KVM support (using VNC protocol)
- 2011: ME 7.1
  - DAL (Dynamic Application Loader) for IPT/OTP
- 2014: ME 10.0
  - Removal of TDT (Anti-Theft) feature
- 2015: ME 11.0
  - first Gen 3 release (x86 core, Minix OS)

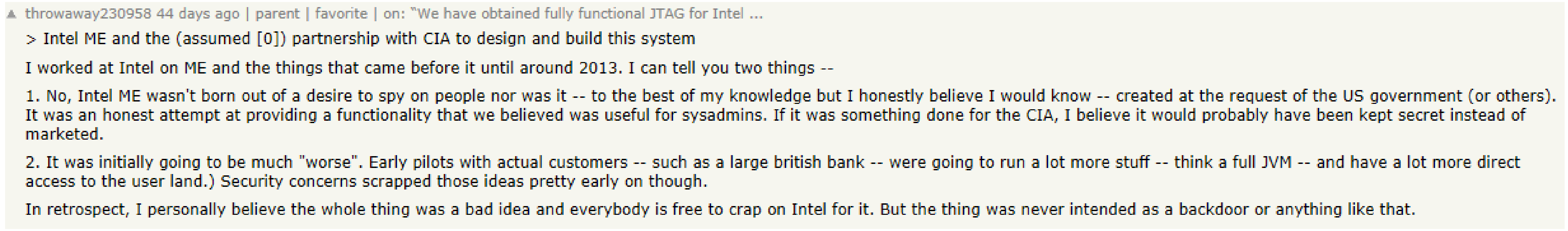
So, initially ME only ran AMT/vPro functionality, but was gradually extended to other, "non-enterprise" features

- TPM (Trusted Platform Module)
- QST (fan control)
- ICC (Integrated Clock Control)
- TDT (Anti-Theft)
- DAL (IPT/OTP)
- SWC (Silicon Workaround Capability)

Many of these features do benefit from being executed on a separate, isolated CPU and not the host OS.

IMO it's logical Intel opted to use the already present ME instead of adding yet another subsystem.

As for the .gov angle, if they had control over Intel, why would they request adding of the infamous HAP bit?



▲ throwaway230958 44 days ago | parent | favorite | on: "We have obtained fully functional JTAG for Intel ...

> Intel ME and the (assumed [0]) partnership with CIA to design and build this system

I worked at Intel on ME and the things that came before it until around 2013. I can tell you two things --

1. No, Intel ME wasn't born out of a desire to spy on people nor was it -- to the best of my knowledge but I honestly believe I would know -- created at the request of the US government (or others). It was an honest attempt at providing a functionality that we believed was useful for sysadmins. If it was something done for the CIA, I believe it would probably have been kept secret instead of marketed.

2. It was initially going to be much "worse". Early pilots with actual customers -- such as a large british bank -- were going to run a lot more stuff -- think a full JVM -- and have a lot more direct access to the user land.) Security concerns scrapped those ideas pretty early on though.

In retrospect, I personally believe the whole thing was a bad idea and everybody is free to crap on Intel for it. But the thing was never intended as a backdoor or anything like that.

from *Hacker News*

Myth 2: It is always on even if the PC is turned off

Reality: yes, but it depends

ME POWER STATES

S0/M0: Host is on, ME is fully functional

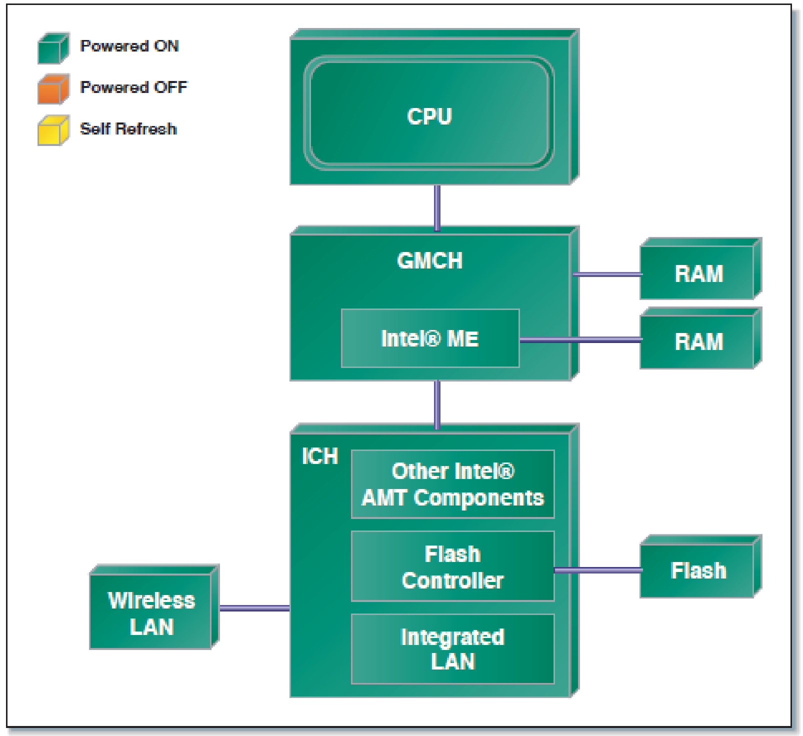


Figure 7.13 An Intel® AMT Computer in S0 and M0 State

source: [APMD09]

ME POWER STATES

M1: Host is suspended, ME is partially functional

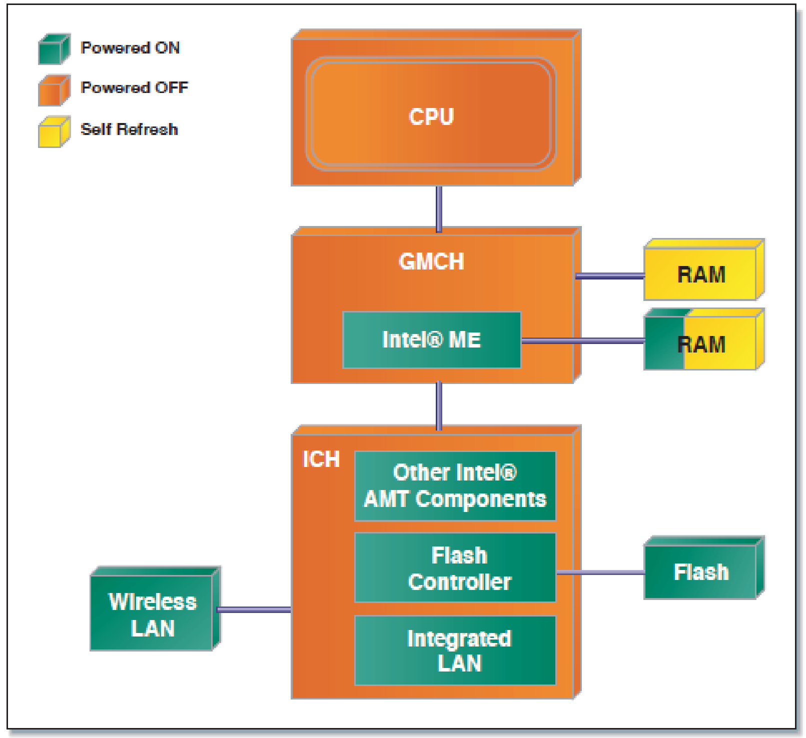


Figure 7.14 An Intel® AMT Computer in S3 and M1 State

source: [APMD09]

ME POWER STATES

M-Off: Host is suspended/off, ME is powered down

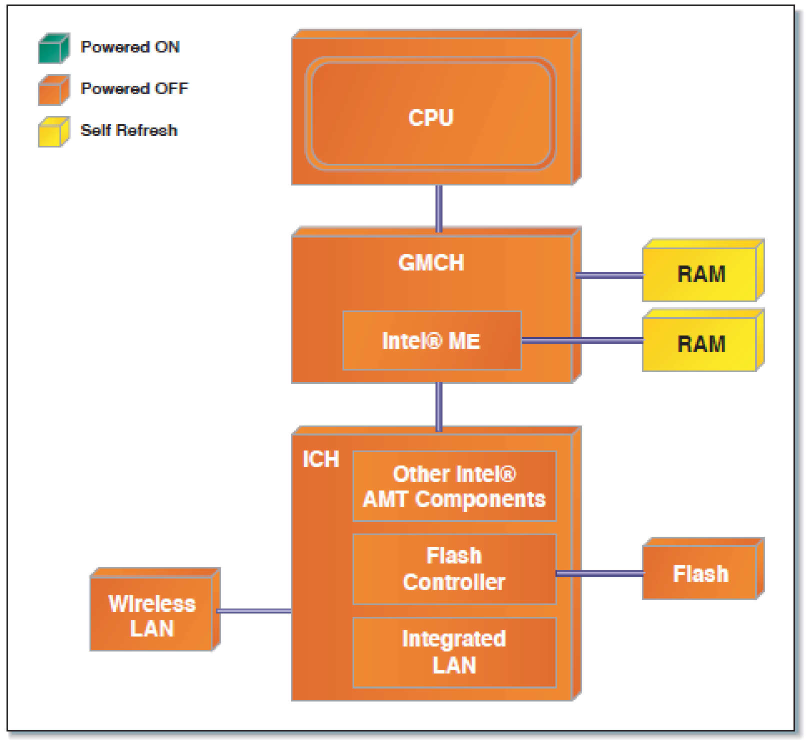


Figure 7.15 An Intel® AMT Computer in S3 and M-Off State

source: [APMD09]

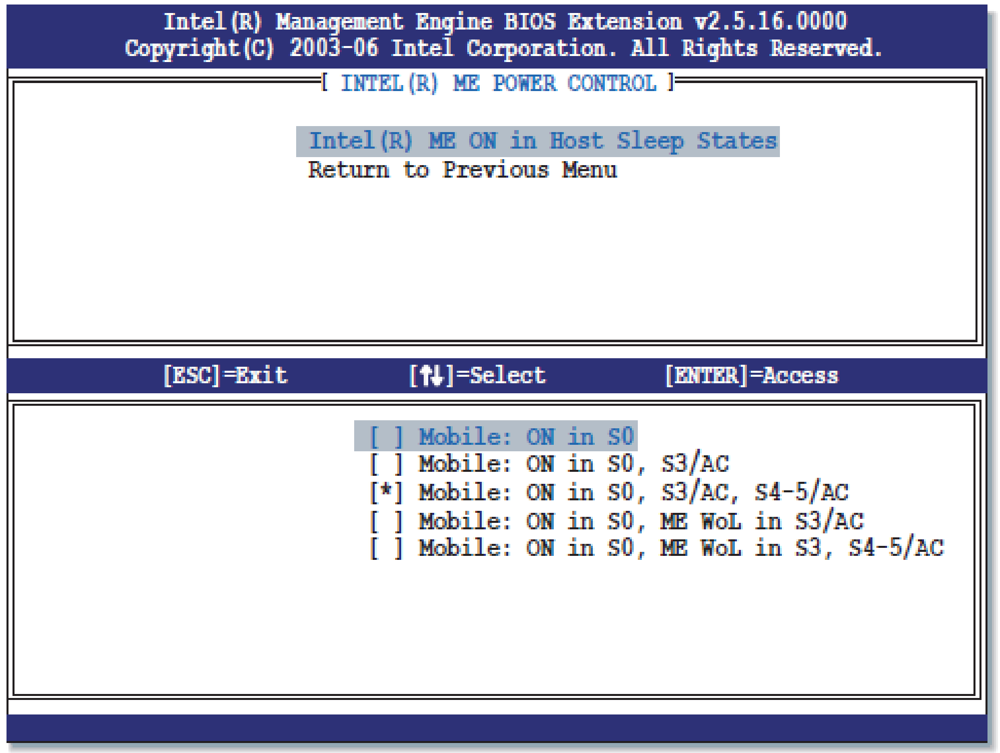
ME POWER STATES

M3: added in more recent platforms.

- Host is in Sx (S3/S4/S5)
- ME is running using a dedicated M3 power rail on the board.
- Host memory/UMA is unavailable (uses internal SRAM only).

Summary

- ME *can* be running when the host is sleeping/hibernating
- shuts down after a timeout (may be configurable)



```
        Intel(R) Management Engine BIOS Extension v2.5.16.0000
    Copyright(C) 2003-06 Intel Corporation. All Rights Reserved.
                   [ INTEL(R) ME POWER CONTROL ]

                   Intel(R) ME ON in Host Sleep States
                   Return to Previous Menu

   [ESC]=Exit          [↑↓]=Select          [ENTER]=Access

              [ ] Mobile: ON in S0
              [ ] Mobile: ON in S0, S3/AC
              [*] Mobile: ON in S0, S3/AC, S4-5/AC
              [ ] Mobile: ON in S0, ME WoL in S3/AC
              [ ] Mobile: ON in S0, ME WoL in S3, S4-5/AC
```

source: [APMD09]

MYTH 3: IT CAN LOCK THE PC WITH A COMMAND SENT OVER THE AIR

Reality: *Was* possible, not anymore

- Need ME with TDT module (4.1-9.x)
- Antitheft needs to be enabled *and* enrolled, with active subscription.

- 3G support added in ME 7.0 (3G module must be connected directly to chipset via SMBus)
- "poison pill" must be signed by Intel to be acccepted by the ME
- Intel Anti-Theft service was shut down in 2015 and TDT removed from firmware in ME 10.0
- current Anti-Theft solutions from Computrace et al. use a BIOS/UEFI module dropping a binary at boot time; no ME involvement

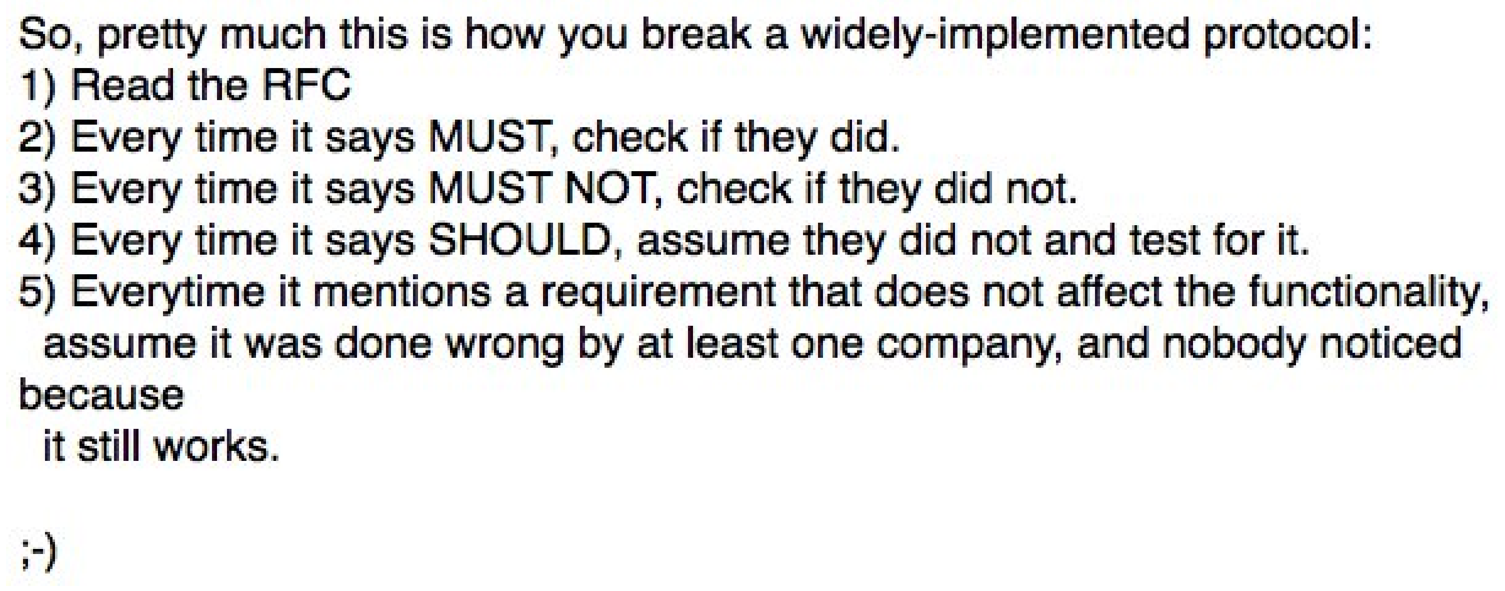
Myth 4: It can read all data on PC/spy on the user

Reality: it's complicated...

- it can read/write host *memory* via DMA engine
- no direct access to other host HW (AFAIK), needs cooperation from host firmware/drivers
- some sensitive memory areas are blocked (SMM/VT-d)
- has access to iGPU for PAVP but I *think* no direct access to frame buffer
- can emulate IDE or USB devices on the host (for IDE-R and KVM)

More info: [PSTR14]

Myth 5: It's a black box which can't be audited because it's closed source



So, pretty much this is how you break a widely-implemented protocol:
1) Read the RFC
2) Every time it says MUST, check if they did.
3) Every time it says MUST NOT, check if they did not.
4) Every time it says SHOULD, assume they did not and test for it.
5) Everytime it mentions a requirement that does not affect the functionality,
  assume it was done wrong by at least one company, and nobody noticed
because
  it still works.

;-)

by @daniel_bilar

Black box auditing

- Plenty of products get audited without source code
- See *Security Evaluation of Intel's Active Management Technology* by Vassilios Ververis [SMT10] - audit by reading docs + experimentation
- The firmware itself is available on the flash and in firmware updates, you "just" need to figure our how to extract and disassemble the code, then make sense of it.

IMO it's better to audit the binary code:

- you see what actually is being executed by the hardware, not what the authors *think* is executed
- no need to account for possible `#ifdefs`/macros/formatting, compiler optimizations etc. (see "goto fail")
- You lose comments and names but those can be actually misleading
- It does take a lot of time :(

The public bugs so far have been found without source code:

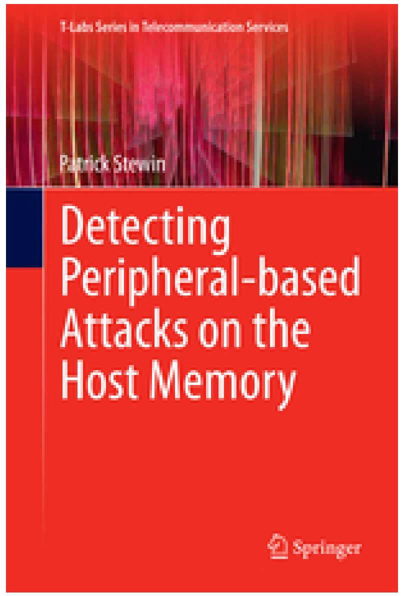
- Vassilios Ververis [SMT10] found several issues in AMT 4.x by monitoring the network and testing the publicly described provisioning scenarios
- INTEL-SA-00075/CVE-2017-5689 [ASM17] was discovered by reading documentation/looking at network traffic
- INTEL-SA-00086/CVE-2017-5705 [PTME17] was found by static analysis of the firmware code

Conclusion: You *can* audit it even without source code (and maybe even without reverse engineering).

Myth 6: you can infect it with a stealthy undetectable rootkit

- There *were* research rootkits for Gen 1 ME, in particular see [IRR09] (BH 2009) and [DAGGER] (Stewin, 2013)
- They used a bug in some older BIOSes which allowed access to the ME UMA to inject code
- requires reinfection on each reboot (modification of firmware in flash is detected and rejected by boot ROM)

- in Gen2 (>=6.0) Intel implemented UMA integrity checking, so any modifications result in shutdown [PSTR14]
- Stewin proposed some approaches on detecting DMA attacks from external hardware (including ME) [DPAHM15]


T-Labs Series in Telecommunication Services
Patrick Stewin
Detecting Peripheral-based Attacks on the Host Memory
Springer

Myth 7: Users can't do anything about it

See next section

WHAT CAN I DO ABOUT IT?

During my experiments with coreboot I asked myself: "Why do I need the Intel ME firmware? Can I remove it?"

The Libreboot F.A.Q. page had the answer:

> *Before version 6.0 (that is, on systems from 2008/2009 and earlier), the ME can be disabled by setting a couple of values in the SPI flash memory. [...]*

ME firmware versions 6.0 and later [...] include “ME Ignition” firmware that performs some hardware initialization and power management. If the ME’s boot ROM does not find in the SPI flash memory an ME firmware manifest with a valid Intel signature, the whole PC will shut down after 30 minutes.

What? It turns on correctly and then it turns off after 30 minutes? Come on...

So the Intel ME firmware is not *technically* required...

coreboot ML, 12 Sep 2016

> *[...] If I just erase the first 4KB of its region [...] coreboot boots up fine and reports that "WARNING: ME has bad firmware". My Linux payload initializes without any complaints. [...]*
>
> *[...] it has been operational for the past few hours [...]*

— Trammel Hudson

coreboot ML, 15 Sep 2016

> *[…] The only piece that must be present for my x230 to function is the 512 KB FTPR partition [probably "Factory Partition and Recovery"], which contains these compressed modules […]*

— Trammel Hudson

coreboot ML, 19 Sep 2016

> *[...] I've built an even more reduced ME firmware that has removed a few modules from the FTPR partition: [these modules] can be replaced with 0xFF and the ME will still initialize the system correctly. This leaves only ROMP, BUP, KERNEL, POLICY and FTCS. [...]*

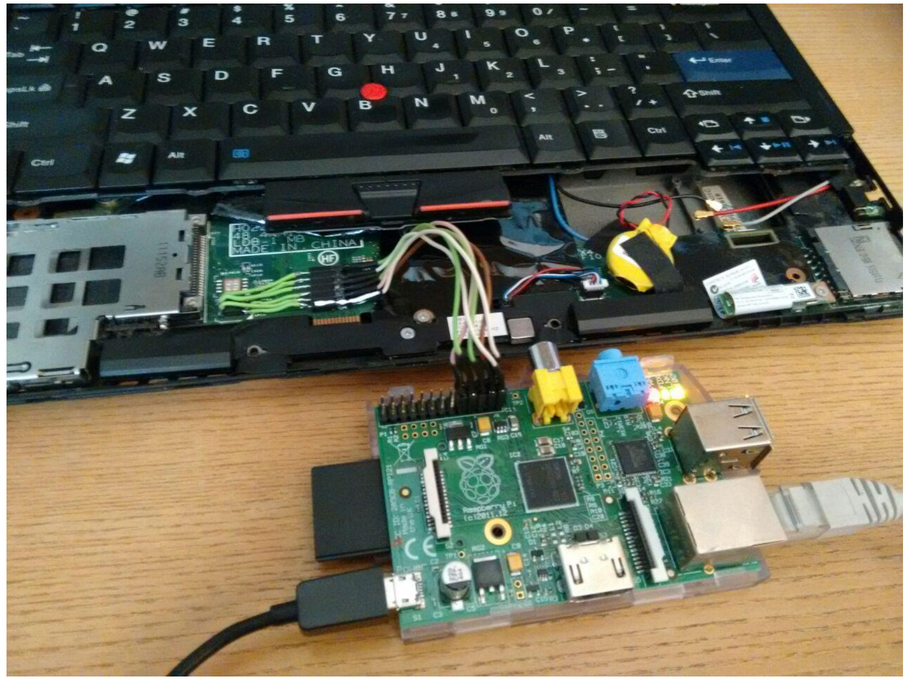
— Trammel Hudson

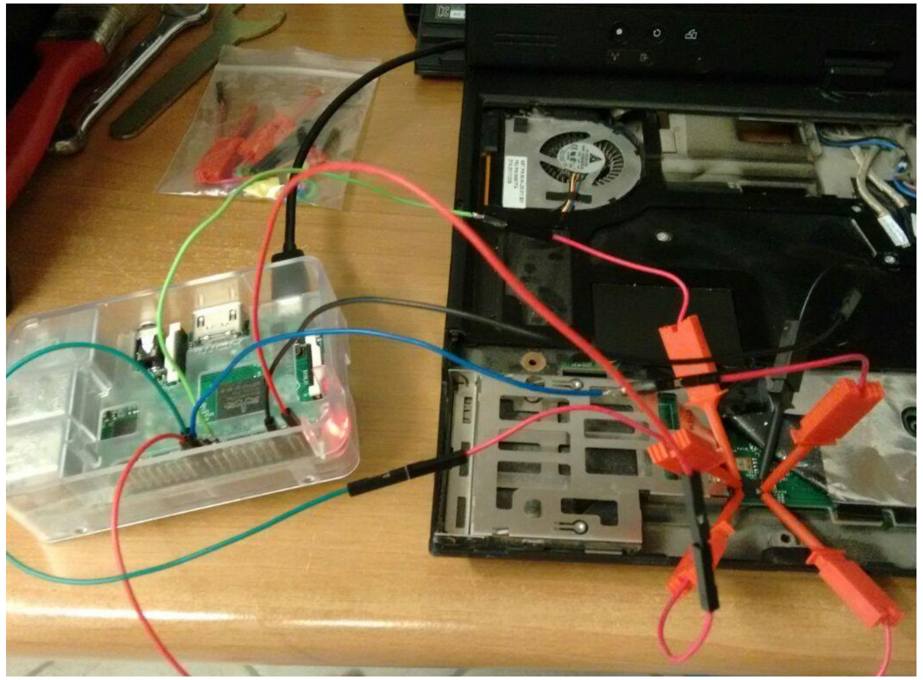
To avoid using an Hex editor I started writing me_cleaner, a python script able to reduce an Intel ME firmware image to the bare minimum and to force Intel ME to shut off just after the hardware initialization.

WHERE'S THE INTEL ME FIRMWARE?

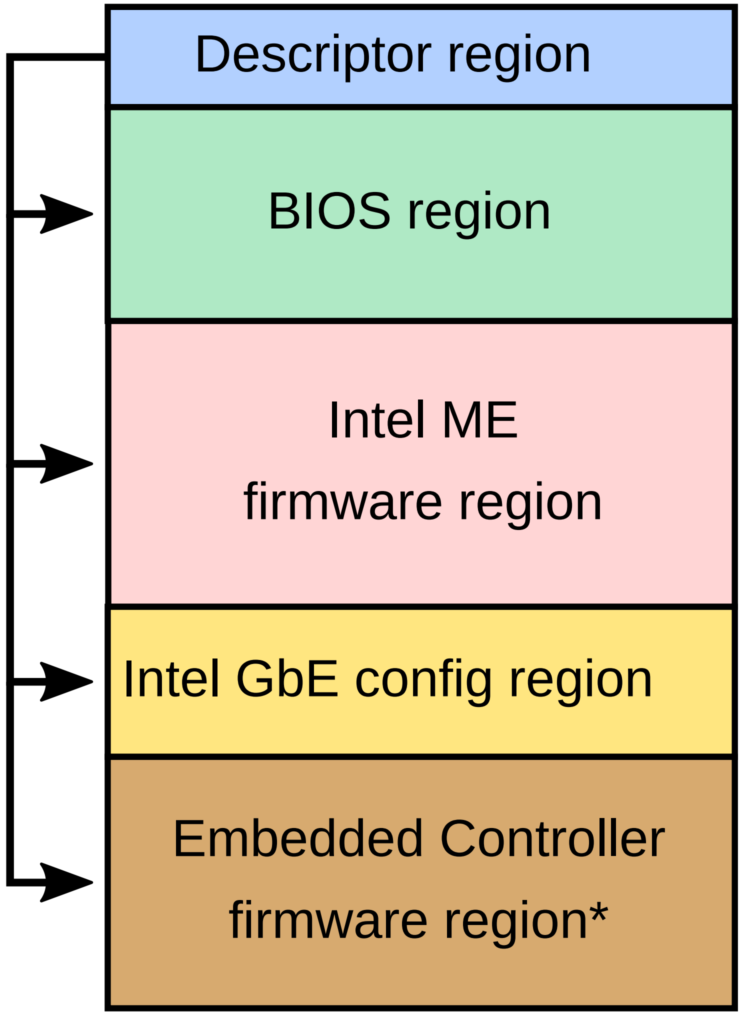
The Intel ME firmware is in the same flash chip of the BIOS/UEFI, in its own region.

Reading and writing it with an external programmer is quite simple.





INTEL FLASH DESCRIPTOR (IFD)



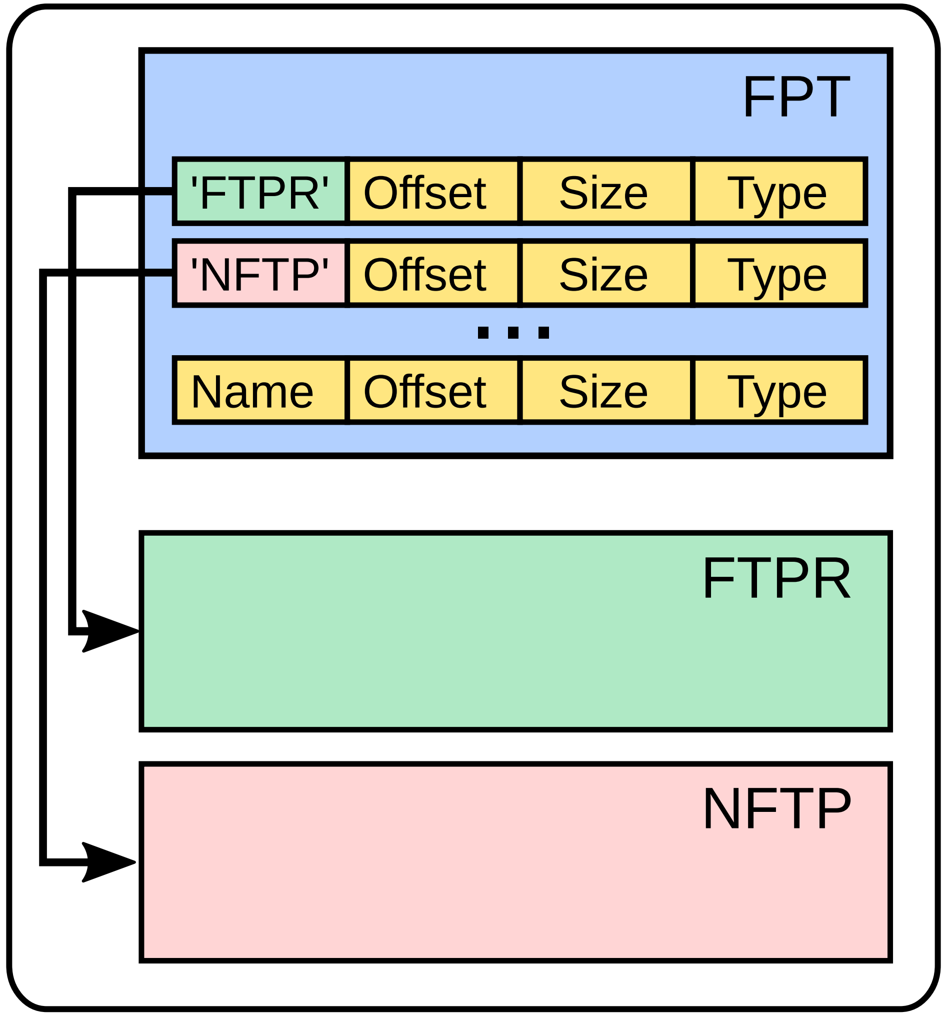
*starting from Skylake

These regions can be analyzed, extracted and modified with the help of ifdtool from the coreboot project.

STEP 1: REMOVE (ALMOST) ALL THE PARTITIONS

The first step was to remove every partition of the Intel ME firmware image except for the FTPR, the fundamental one.

FIRMWARE PARTITION TABLE (FPT)



The removal of the partitions is trivial, thanks to the fact that:

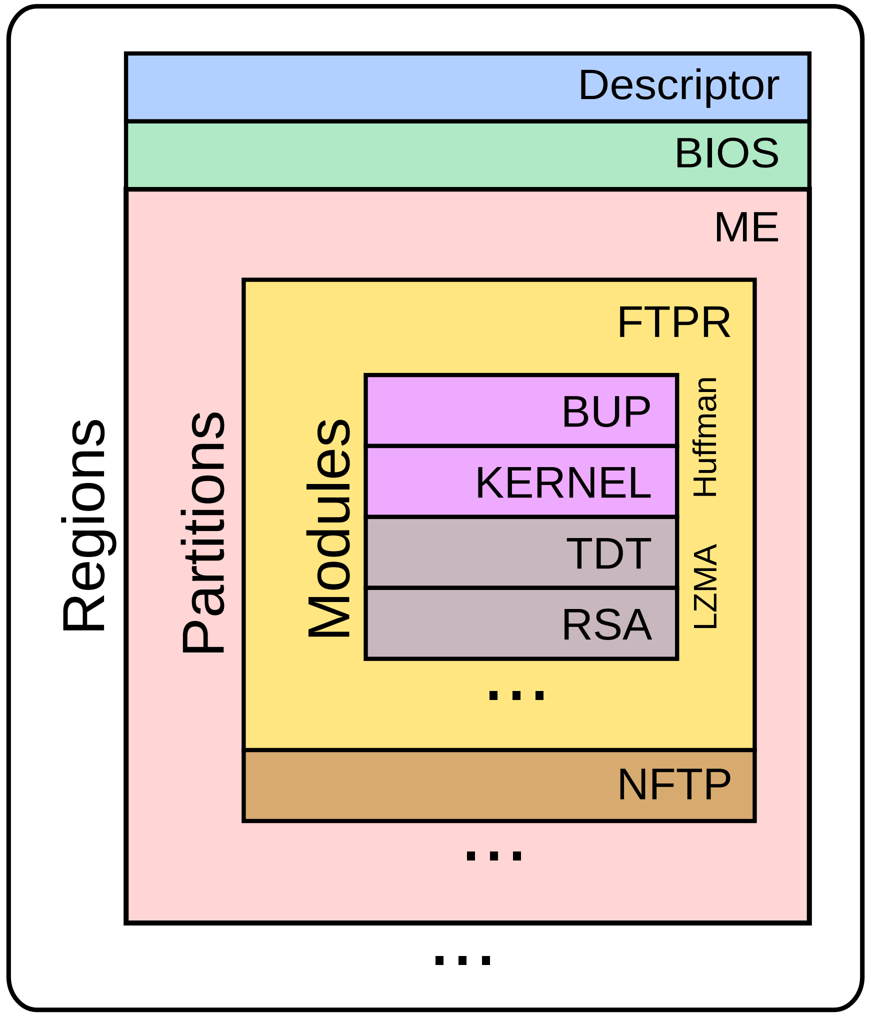
- The FPT is not signed, has just a checksum
- The partitions are individually signed
- The offset and size of each partition are saved in each FPT entry

SUCCESS!

STEP 2: REMOVE THE LZMA MODULES

My next step was to remove all the LZMA modules.

Regions, partitions, modules...



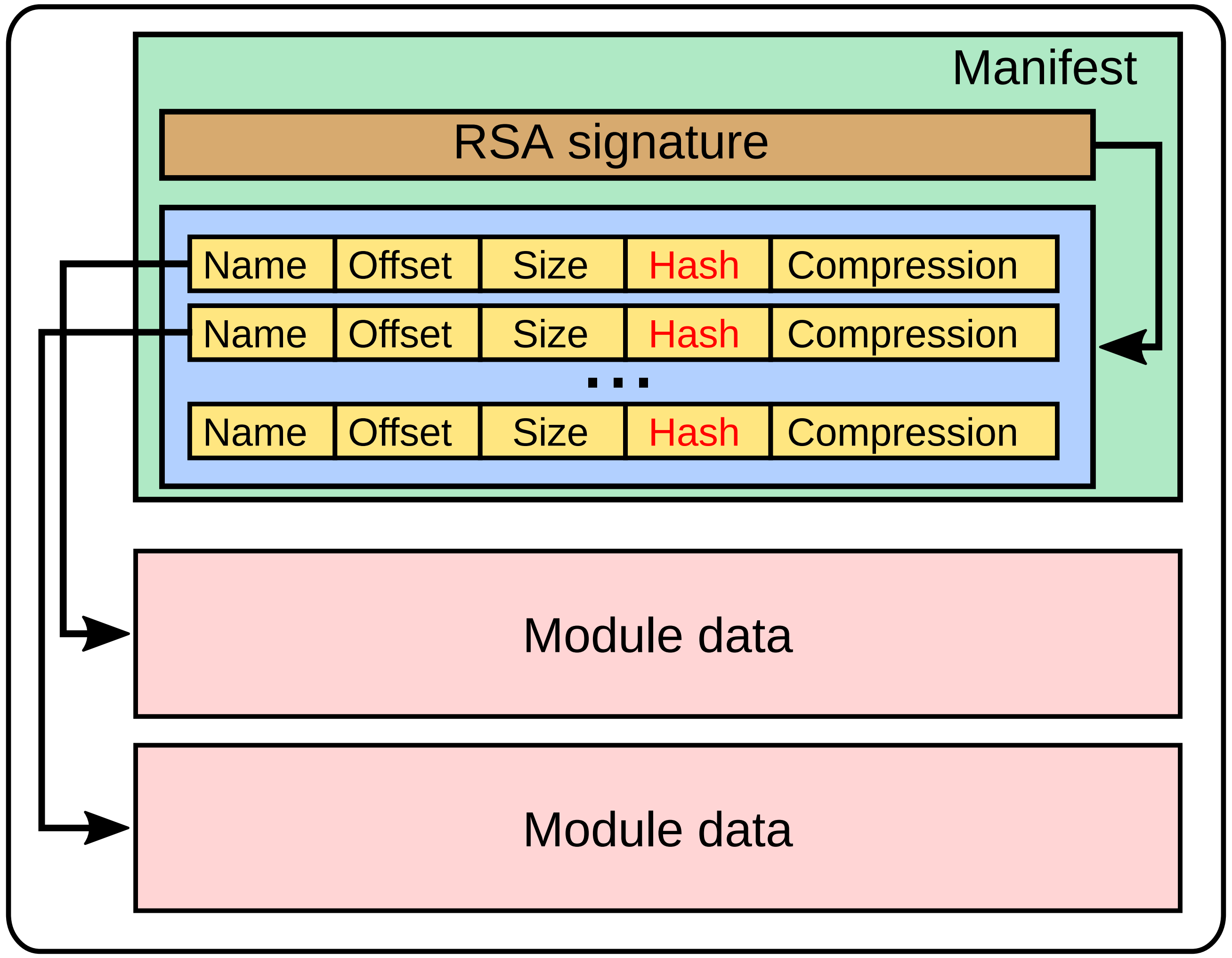
Different types of partitions have different internal structure, but our interest is focused on the "Code" partitions (like the FTPR).

The content of these partitions is organized in modules, however the internal layout changes between different generations.

INTEL ME/TXE/SPS GENERATIONS

	Gen. 1	**Gen. 2**	**Gen. 3**
ME versions	1.x-5.x	6.x-10.x	11.x-12.x
TXE versions		1.x-2.x	3.x
SPS versions	1.x	2.x-3.x	4.x
Years	2005-2008	2008-2015	2015-cur

GENERATION 2

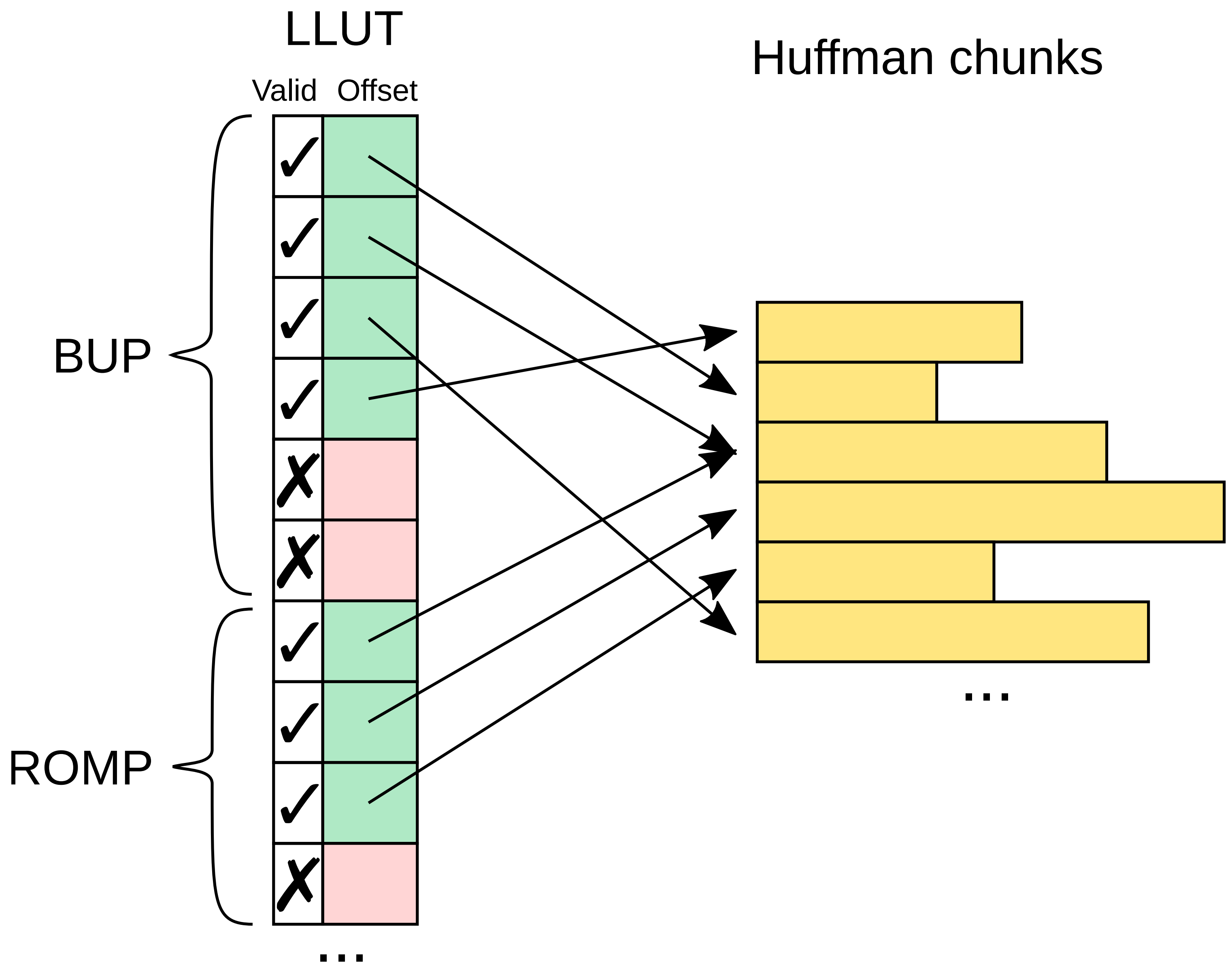


So I tried again: I removed every partition except for FTPR and I kept only the FTPR's Huffman modules (5 modules: ROMP, BUP, KERNEL, POLICY and FTCS).

SUCCESS!

STEP 3: HUFFMAN MODULES

With the help of the source code of unhuffme I understood the structure of the Huffman modules.


LLUT
Valid
Offset
Huffman chunks
BUP
ROMP
✓
✗
...
...

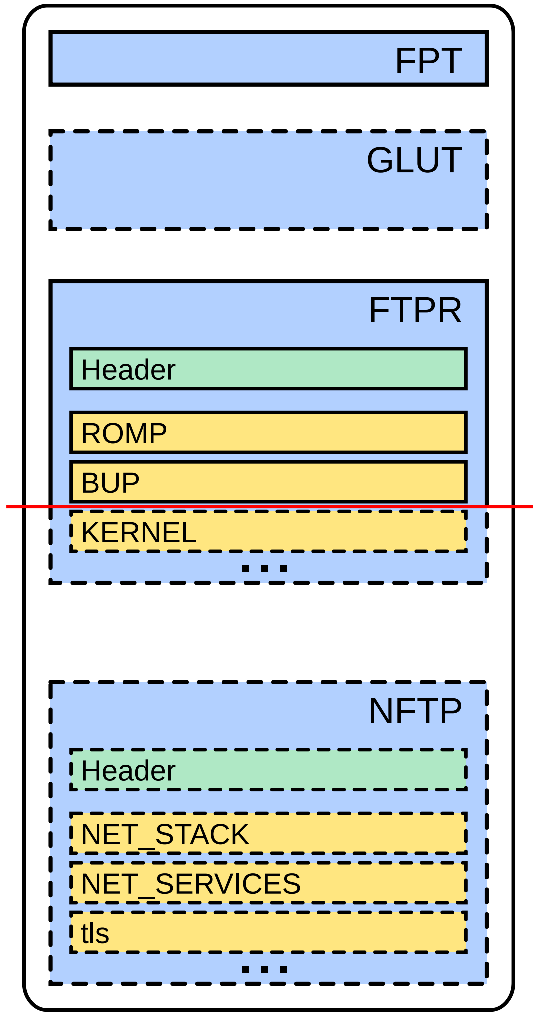
So I updated me_cleaner and I tried to remove the Huffman module with the "less-important" name (FTCS) and I flashed back the result.

SUCCESS!

With the help of intelmetool I discovered which modules were really needed:

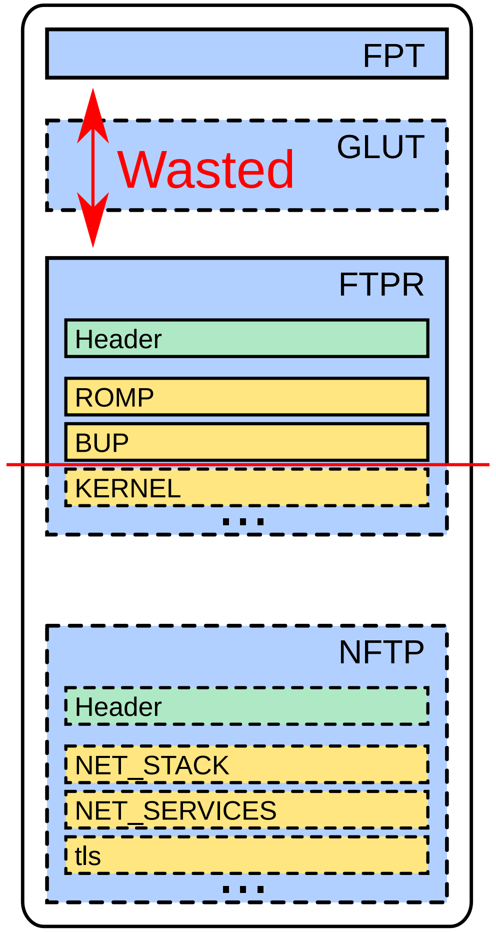
- BUP, where the 30-minutes watchdog is turned off
- ROMP, which seems to contain some sort of configuration data read by BUP

STEP 4: RECOVER THE FREE SPACE

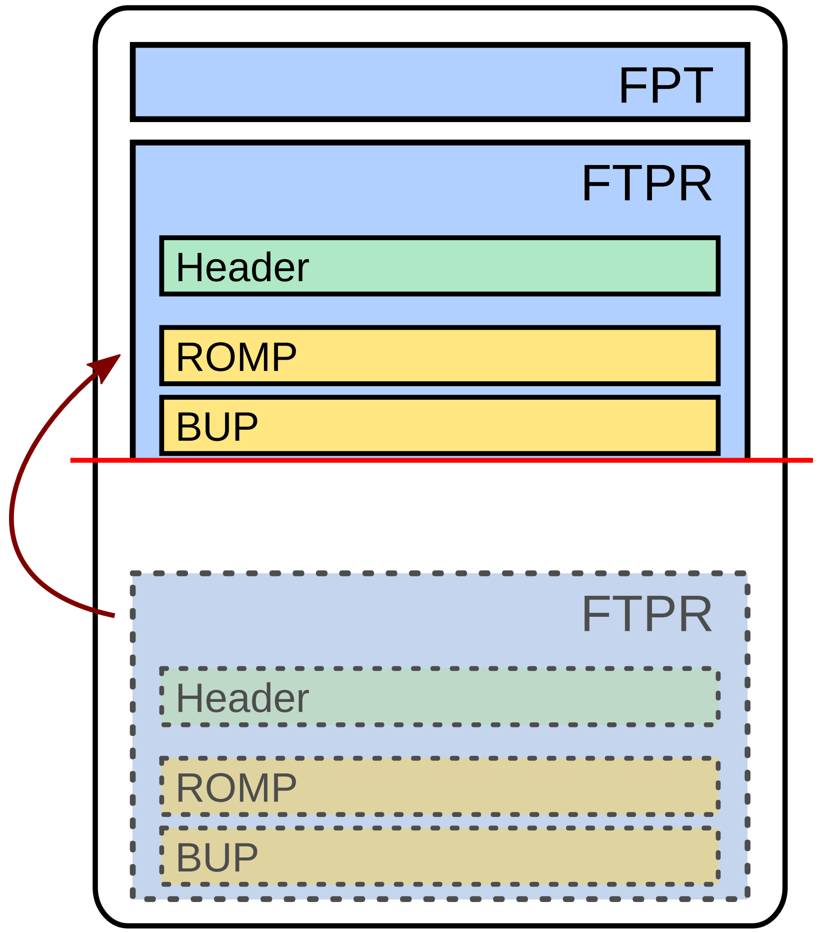


As expected, Intel ME doesn't complain if I remove the unused tail of its firmware.

However there was lot of unused space between the FPT and the FTPR partition, wasted.



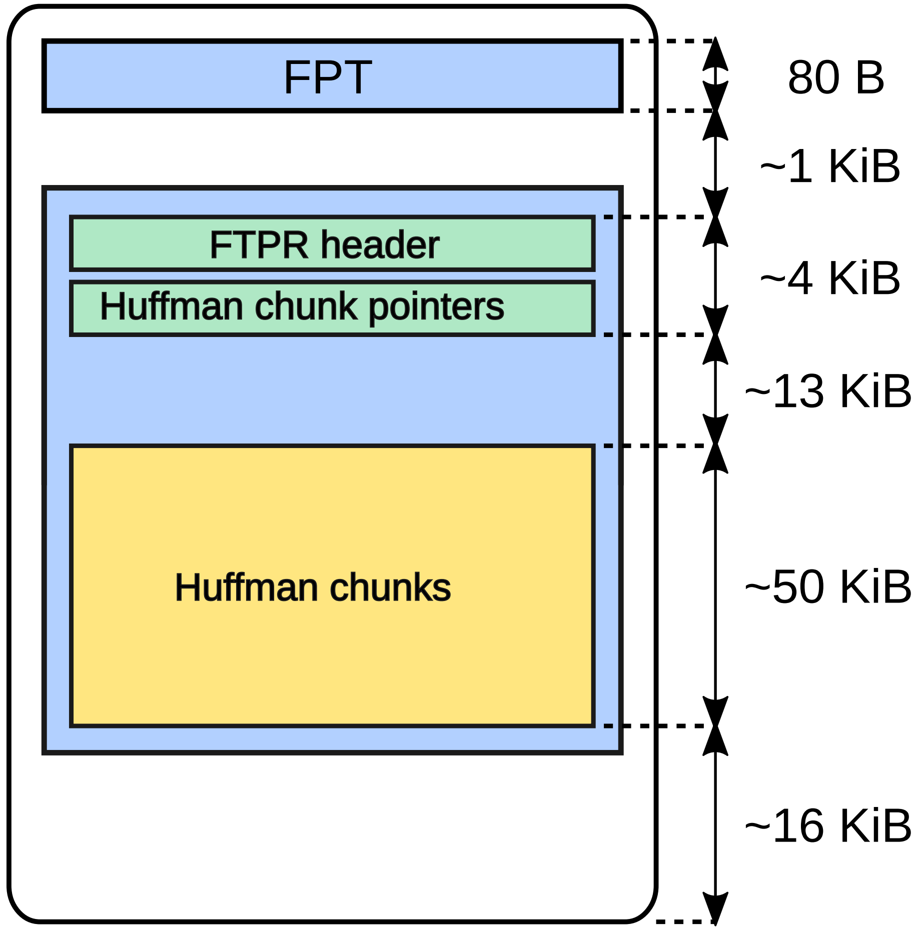
So I had to figure out how to move the partitions around the image.

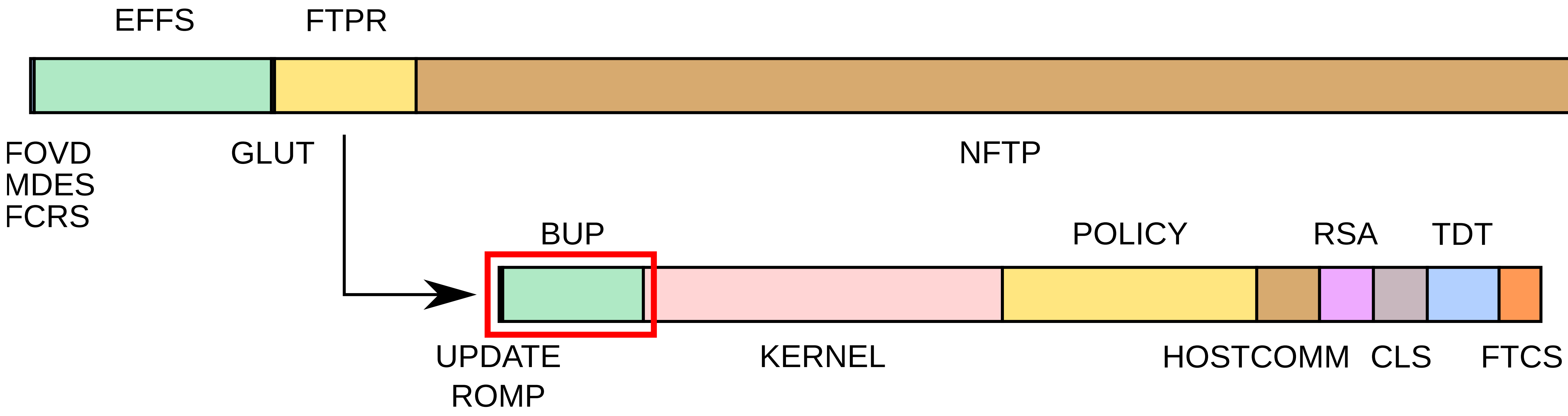


Apparently not all the offsets inside a partition are relative to the partition start, some of them are relative to the beginning of the Intel ME firmware image.

Luckily, they aren't signed, so after many tries (and bricked laptops) I was able to correct them as well.

In the end, the resulting image had only the FTPR partition with just two modules, BUP and ROMP, moved to the lowest address possible. The Intel ME firmware, originally 5 MiB, is now 84 KiB.

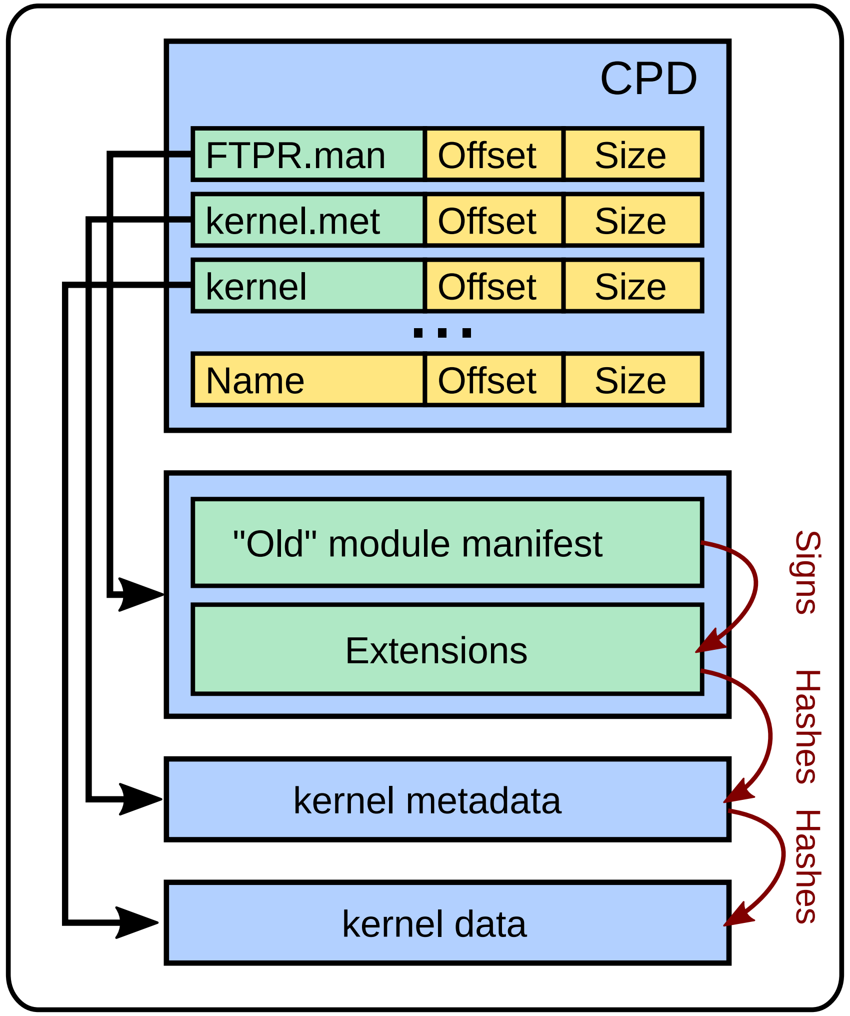



EFFS
FTPR
FOVD
MDES
FCRS
GLUT
NFTP
BUP
POLICY
RSA
TDT
UPDATE
ROMP
KERNEL
HOSTCOMM
CLS
FTCS

GENERATION 3

The Intel ME partition scheme is the same, so I was already able to remove all the partitions (except FTPR) without any modification on me_cleaner.

CODE PARTITION DIRECTORY (CPD)



Each CPD entry can be either:

- the partition manifest (".man"), which is the "old" generation 2 manifest
- a module metadata (".met"), which also contains the hash of the module
- a module

So I started again my search for the "fundamental modules" and I found out that the modules needed for a correct boot were:

- syslib
- rbe
- kernel
- bup

Some weeks after my work on generation 3 the Positive Technologies researchers shared their discoveries of a method to disable Intel ME.

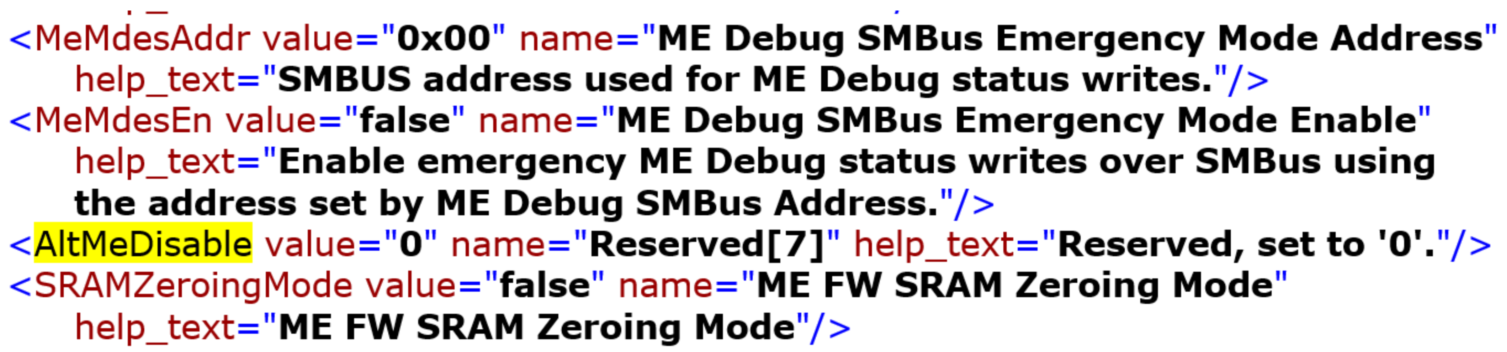
Through the reverse-engineering of the firmware they identified the same modules I found as fundamental, but with an interesting bonus: Intel ME generation 3 has a kill switch.

HAP BIT

The Positive Technologies researchers found an undocumented bit in the descriptor that, once set, forces Intel ME to turn itself off after the initialization of the system.

This bit is called "HAP bit" in the XMLs inside the Intel binaries and, as confirmed by Intel, has been added for the US government's "High Assurance Platform" program.

Moreover Igor Skochinsky found a different bit, the AltMeDisable bit, which should achieve the same result on generation 2.



```
<MeMdesAddr value="0x00" name="ME Debug SMBus Emergency Mode Address"
    help_text="SMBUS address used for ME Debug status writes."/>
<MeMdesEn value="false" name="ME Debug SMBus Emergency Mode Enable"
    help_text="Enable emergency ME Debug status writes over SMBus using
    the address set by ME Debug SMBus Address."/>
<AltMeDisable value="0" name="Reserved[7]" help_text="Reserved, set to '0'."/>
<SRAMZeroingMode value="false" name="ME FW SRAM Zeroing Mode"
    help_text="ME FW SRAM Zeroing Mode"/>
```

FINAL RESULTS

The combination of the HAP/AltMeDisable bit and code removal forces Intel ME to stop just after the initialization of the system but, unlike the sole code removal, is better supported by commercial UEFI/BIOS implementations.

To probe the status of Intel ME I used the software intelmetool; its output with the sole code removal on a Thinkpad X220t (Sandy Bridge) is:

```
[...]
ME: Firmware Init Complete  : NO
[...]
ME: Current Operation Mode  : Normal
ME: Error Code              : Image Failure
ME: Progress Phase          : BUP Phase
[...]
ME: Progress Phase State : M0 kernel load
[...]
```

while, with the addition of the AltMeDisable bit:

```
[...]
ME: Firmware Init Complete  : NO
[...]
ME: Current Operation Mode  : Debug
ME: Error Code              : No Error
ME: Progress Phase          : BUP Phase
[...]
ME: Progress Phase State : [...] straps say ME DISABLED
[...]
```

Thanks to the testing performed by the community, me_cleaner have been reported working on most the PCs from Nehalem to Coffee Lake, both on commercial firmware and coreboot.

The firmware size is greatly reduced:

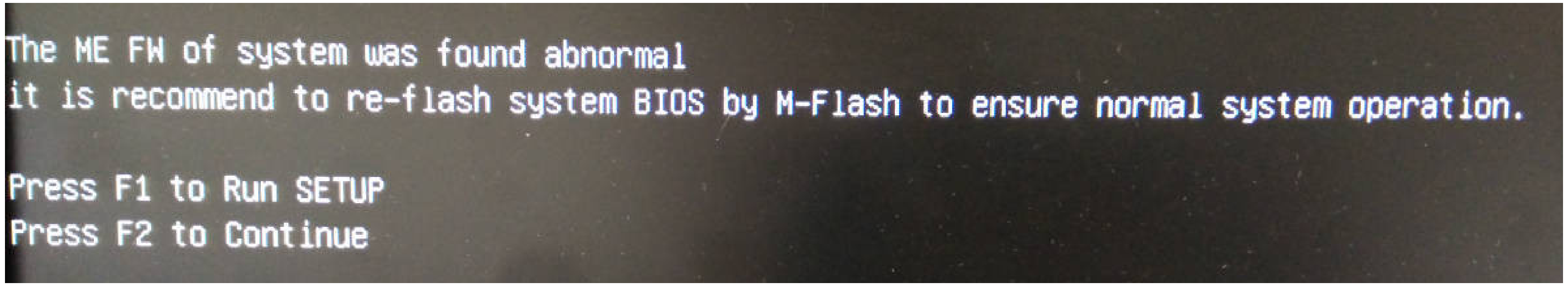
	Original	**Modified**
Generation 2	1.5 MiB / 5 MiB	84 KiB
Generation 3	2 MiB / 6.6 MiB	330 KiB

And many "unwanted features" are now gone, like:

- Intel ME kernel (on generation 2)
- NFTP (AMT/network stack)
- DAL (Dynamic application loader)
- PTT (Platform Trust Technology, firmware TPM)

Some issues can arise:

- "Brick"
- Slight boot delay
- Automatic rollback of the me_cleaner modifications
- Warning messages during the startup



```
The ME FW of system was found abnormal
it is recommend to re-flash system BIOS by M-Flash to ensure normal system operation.

Press F1 to Run SETUP
Press F2 to Continue
```

Moreover some features are now broken, as they depend on parts of Intel ME which have been removed:

- Overclocking (ICC)
- Intel AMT
- Intel PAVP
- Parts of Intel SGX
- Others…

DEMO

```
~/demo
```

CALL FOR ACTION

- Try me_cleaner on your systems (be careful and prepare a way to recover)
- Report both successes and failures
- Investigate actual behavior of ME as compared to documentation
- Go forth and reverse!

ACKNOWLEDGEMENTS

- Invisible Things Lab (invisiblethingslab.com)
- Positive Technologies (ptsecurity.com)
- Trammel Hudson (@qrs, trmm.net)
- Plato Mavropoulos (@platomaniac, win-raid.com)
- Damien Zammit (github.com/zamaudio/intelmetool)
- Roman Sevko (@apple_rom, vpro.by)
- Federico Izzo and Niccolò Izzo
- Gavin Ferris
- Hardened GNU/Linux (hardenedlinux.github.io)

REFERENCES

- [APMD09] Arvind Kumar., Purushottam Goel, and Ylian Saint-Hilare, Active Platform Management Demystified: Unleashing the Power of Intel VPro Technology, 2009, Intel Press, ISBN 9781934053195
- [IRR09] Alexander Tereshkin, Rafal Wojtczuk, Introducing Ring -3 Rootkits. Black Hat USA, 2009, Las Vegas, NV
- [SMT10] Vassilios Ververis, Security Evaluation of Intel's Active Management Technology, Sweden 2010 TRITA-ICT-EX-2010:37

- [DAGGER] Patrick Stewin and Iurii Bystrov, Persistent, Stealthy, Remote-controlled Dedicated Hardware Malware, 2013, 44CON, London, UK, and 30c3, Hamburg, Germany,
- [PSTR14] Xiaoyu Ruan, Platform Embedded Security Technology Revealed: Safeguarding the Future of Computing with Intel Embedded Security and Management Engine, 2014, Apress, ISBN 978-14302-6572-6

- [DPAHM15] Patrick Stewin, Detecting Peripheral-based Attacks on the Host Memory, 2015, Springer, ISBN 978-3-319-13515-1
- [PTME17] Mark Ermolov, Maxim Goryachy, How to Hack a Turned-Off Computer, or Running Unsigned Code in Intel Management Engine, Black Hat Europe 2017, London, UK.

IGOR SKOCHINSKY

@IgorSkochinsky

skochinsky@gmail.com

github.com/skochinsky

NICOLA CORNA

nicola@corna.info

github.com/corna

Q & A

THANK YOU!