



The CPU Rootkit you probably don't know about

Fabian Groffen
Kevin Keijzer

NLUUG, November 28, 2023

Basic Input/Output System origins

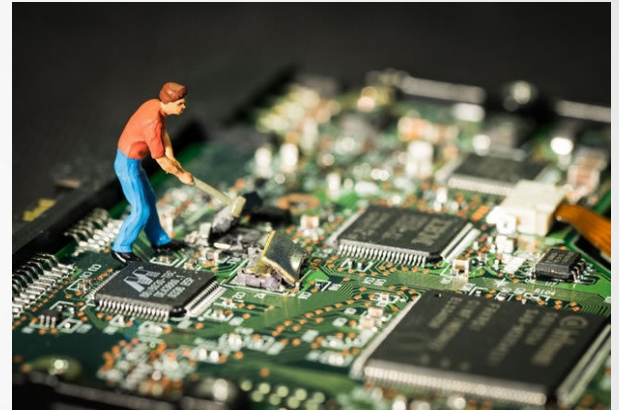
- Modern BIOS stems from the IBM PC
- Reverse-engineered for “PC compatibles”
- First generation BIOSes were very minimal:
 - No GUI
 - Configuration via dipswitches and jumpers
 - Stored on a read-only ROM chip
- All implementations are completely proprietary

Basic Input/Output System current generations

- Mid 90's GUIs appeared, chips became R/W
- 2008: Intel added out-of-band management tools
 - Intel Management Engine
 - Intel Active Management Technology
 - AMD followed suit around 2013 → AMD Platform Security Processor

The x86 boot process

- Flash chip initialization
- RAM detection + training / initialization
- Bus initialization
- Memory mapping
- Search for bootloader
 - e.g. on a storage device's master boot record / EFI system partition
- Load found bootloader

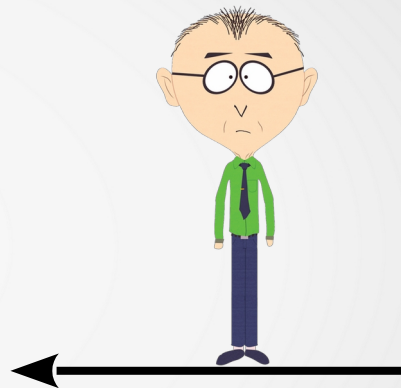


The vendor BIOS / UEFI

- Functionality as for x86 boot process (previously discussed)
- Shows a (vendor) logo, gives interactive menu with function keys
- Allows configuration of low-level hardware
 - Fans
 - Boot devices
 - USB modes, enabled/disabled, etc.
- Facilitates other firmwares, like ME, PSP, PXE boot

Flash chip layout

- Intel systems: firmware descriptor table (FDT)
 0. Flash Descriptor: partition table
 1. BIOS: vendor firmware code BIOS/UEFI
 2. Intel ME: Intel Management Engine firmware
 3. GbE: Intel NIC config including MAC address
 4. Platform Data: possibly unused



Intel Management Engine (ME)

- Separate, independent ARC processor core in the northbridge and later in the CPU
- Runs proprietary RTOS based on MINIX and a Java VM
- Ring -3; full system control, complete memory access
- Runs even when the system is off or suspended
- Cryptographically signed and checked upon every boot
- Part of Active Management Technology and vPro
- Contains network stack and provides out-of-band access
- Numerous serious vulnerabilities have been found
- Essentially a hardware-based rootkit and backdoor



CVE-2017-5712

ID	CVE-2017-5712
Summary	Buffer overflow in Active Management Technology (AMT) in Intel Manageability Engine Firmware 8.x/9.x/10.x/11.0/11.5/11.6 /11.7/11.10/11.20 allows attacker with remote Admin access to the system to execute arbitrary code with AMT execution privilege.

Access	Vector	Complexity	Authentication
	NETWORK	LOW	SINGLE
Impact	Confidentiality	Integrity	Availability
	COMPLETE	COMPLETE	COMPLETE



AMD Platform “Security” Processor

- Separate, independent ARM processor in the main CPU die
- Controls entire boot process; keeps the x86 cores in reset until initialized
- Proprietary OS licensed from a company named TrustZone
- Cryptographically signed and checked upon every boot
- Less researched than the Intel ME, so less is known about it
- Functions (and issues) are similar to the Intel ME though
- Runs in Ring -3
- Multiple security flaws have been found
- Essentially also a hardware-based rootkit

ME/PSP generations

- ME capabilities became gradually worse
- Up to Core 2 era (2009), the firmware could be left out completely
- Starting with Sandy Bridge (2011) and on, parts of it have to remain
 - The system will shut down after 30 minutes without BUP-partition of the ME firmware
- Starting with Skylake (2015) more parts of the ME firmware have to remain, increasing the size from ~90kB to ~300kB
 - The ME requires a kernel and syslib partition in addition to BUP from here on
- Starting with Alder Lake (2021) there is no possibility to strip or disable the ME firmware: it has to remain in full
- The NSA demanded for an option to “soft-disable” the ME, which Intel implemented as **AltMEDisableBit** and later **HAP bit** (both methods are undocumented)
- There is no known way to disable the AMD PSP on any generation of hardware
 - If the PSP is not started, the x86 cores are kept in reset indefinitely

Intel Boot Guard

- Not all vendor BIOSes can run with modified / stripped ME firmware, many of them crash
- 2013: Intel added "Boot Guard", cryptographic signatures checking boot firmware before execution
 - With Boot Guard, the boot firmware has to be signed with an Intel key in order to run at all
 - Boot Guard is present on most systems
 - The vendor can disable Boot Guard during the production process
 - Boot Guard is not possible when the motherboard and CPU are sold separately (i.e. with self-built desktops)

Now, so what?



coreboot

coreboot

- Open source lightweight boot firmware (GPLv2)
- Replaces BIOS / UEFI
- Performs only the minimal hardware initialization
- Booting OS handled by a payload
- Used on all Chromebooks
- Used on all Intel laptops from System76 and Purism
- Around 250 laptops and motherboards are supported

Why use coreboot?

- coreboot allows you to completely control the boot process
- Open source: fully auditable
- System logs from pushing the power button
- coreboot is **FAST**
- Runs without ME (firmware) if the chipset allows it
- Does not implement any artificial restrictions

```
kevin@vanadium: ~/coreboot/util
CPU has 2 cores.
Setting up SMI for CPU
Will perform SMM setup.
CPU: Intel(R) Core(TM)2 Duo CPU   P8600  @ 2.40GHz.
Loading module at 00030000 with entry 00030000. filesize: 0x170 memsize: 0x170
Processing 16 relocs. Offset value of 0x00030000
Attempting to start 1 APs
Waiting for 10ms after sending INIT.
Waiting for 1st SIPI to complete...done.
Waiting for 2nd SIPI to complete...done.
AP: slot 1 apic_id 1.
Loading module at 00038000 with entry 00038000. filesize: 0x1a8 memsize: 0x1a8
Processing 13 relocs. Offset value of 0x00038000
SMM Module: stub loaded at 00038000. Will call 7f7aacdd(00000000)
Installing SMM handler to 0x7fa00000
Loading module at 7fa10000 with entry 7fa103fe. filesize: 0xec8 memsize: 0x4ee8
Processing 45 relocs. Offset value of 0x7fa10000
Loading module at 7fa08000 with entry 7fa08000. filesize: 0x1a8 memsize: 0x1a8
Processing 13 relocs. Offset value of 0x7fa08000
SMM Module: placing jmp sequence at 7fa07c00 rel16 0x03fd
SMM Module: stub loaded at 7fa08000. Will call 7fa103fe(00000000)
Initializing Southbridge SMI...

New SMBASE 0x7fa00000
In relocation handler: cpu 0
New SMBASE=0x7fa00000
Writing SMRR. base = 0x7fa00000, mask=0xffe00800
SMRR not enabled, skip writing SMRR...
Relocation complete.
VMX status: enabled
VMX status: enabled
New SMBASE 0x7f9ffc00
In relocation handler: cpu 1
New SMBASE=0x7f9ffc00
Writing SMRR. base = 0x7fa00000, mask=0xffe00800
Relocation complete.
Initializing CPU #0
CPU: vendor Intel device 1067a
CPU: family 06, model 17, stepping 0a
Enabling cache
```

Payloads

- coreboot only does basic hardware initialization
- coreboot does not load an OS
- coreboot does not have a network stack
- coreboot does not even have a menu
- When coreboot is finished, it executes a payload

Payload examples are:

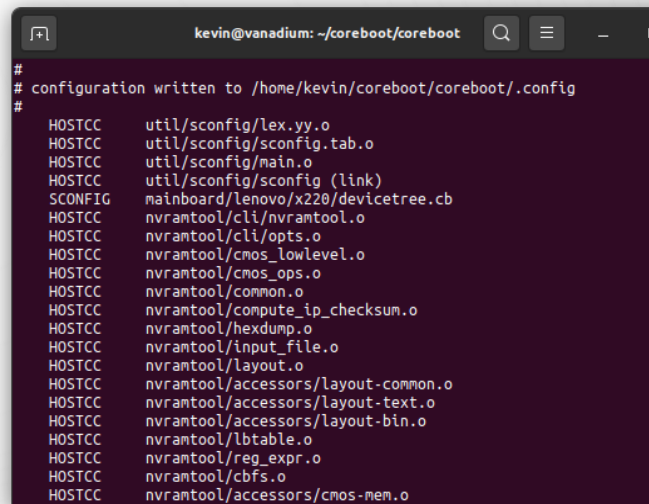
- SeaBIOS
- GRUB
- TianoCore (UEFI)
- Petitboot (Linux)
- coreinfo
- memtest86+

GRUB payload

- GRUB is present on the flash chip on the motherboard
- Even when no storage device is present, it will still start
- It looks for a grub.cfg file on all storage devices
- It can chainload other payloads, like memtest86+, coreinfo, SeaBIOS, etc.
- Acts like a (BIOS) boot menu

Compiling coreboot

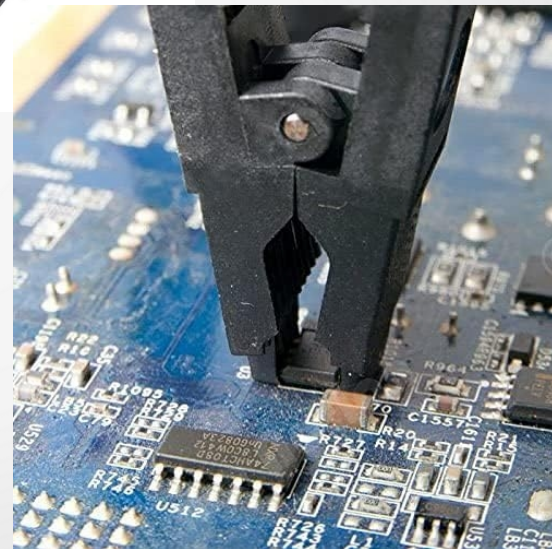
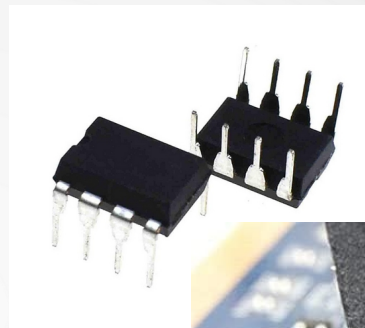
- Compiled for your particular motherboard
- Written mostly in C, with some Assembly and Ada (framebuffer)
- 32-bit code, cross-compiler from coreboot
- Same config system as Linux kernel
- .config file for manual/scripted config
- Internal filesystem: CoreBootFileSystem



```
kevin@vanadium: ~/coreboot/coreboot
# configuration written to /home/kevin/coreboot/coreboot/.config
#
HOSTCC util/sconfig/lex.yy.o
HOSTCC util/sconfig/sconfig.tab.o
HOSTCC util/sconfig/main.o
HOSTCC util/sconfig/sconfig (link)
SCONFIG mainboard/lenovo/x220/devicetree.cb
HOSTCC nvrantool/cli/nvrantool.o
HOSTCC nvrantool/cli/opts.o
HOSTCC nvrantool/cmos_lowlevel.o
HOSTCC nvrantool/cmos_ops.o
HOSTCC nvrantool/common.o
HOSTCC nvrantool/compute_ip_checksum.o
HOSTCC nvrantool/hexdump.o
HOSTCC nvrantool/input_file.o
HOSTCC nvrantool/layout.o
HOSTCC nvrantool/accessors/layout-common.o
HOSTCC nvrantool/accessors/layout-text.o
HOSTCC nvrantool/accessors/layout-bin.o
HOSTCC nvrantool/lbtable.o
HOSTCC nvrantool/reg_expr.o
HOSTCC nvrantool/cbfs.o
HOSTCC nvrantool/accessors/cmos-nem.o
```

Flashing firmware

- Remember: vendor firmware + ME protect the flashchip from writes
- First flash must be done “external”
 - Socketed chips: replace or remove
 - Soldered chips: flash using clip
- Disable Linux kernel protection from writes
- Updates: flash from Linux
`flashrom -p internal ...`



Running coreboot

- Power on system :)
- coreboot launches chosen payload after init stages
- ???
- PROFIT!

- Under Linux, use cbmem (e.g. bootlog) and nvramtool (settings)

coreboot on different Intel platforms

- Up until and including Intel Ivy Bridge (2012), coreboot itself can be 100% FOSS
 - The only exceptions are the other flash regions (**ifd**, **gbe** and **me**), which are not part of coreboot
- Starting with Haswell (2013), there is no more open source code that can initialize RAM, which is a primary feature of coreboot
 - Instead, a blob from Intel, called **MRC.bin** has to be used
 - There are some efforts to reverse-engineer this blob, but it's not production-ready yet
- Starting with Broadwell (2014), Intel uses the Firmware Support Package (**FSP**), which is a large blob handling all hardware initialization
 - The FSP is used by both vendor firmware and coreboot
 - It basically degrades coreboot to a glue layer

Want to run coreboot?

- If you want to run 100% open source coreboot with a stripped ME, these are some easily available options:
 - Laptops:
 - Lenovo ThinkPads: X230, T430, T530
 - Motherboards:
 - ASUS P8Z77-V
 - ASRock B75M-ITX, B75 Pro3-M, H77 Pro4-M
 - Gigabyte GA-B75M-D3H

Desktop CPUs:

Intel 3rd generation
i3/i5/i7

For best performance:

- i7-3770 (**not** K/S/T)
 - 3.9GHz, 8 HT cores

For HTPC use:

- i5-3570S
 - 3.6 GHz, lower TDP

Further reading

- **coreboot website**
 - <https://coreboot.org/>
- **“Intel x86s hide another CPU that can take over your machine”** by Damien Zammit
 - <https://boingboing.net/2016/06/15/intel-x86-processors-ship-with.html>
- **“x86 considered harmful”** by Joanna Rutkowska
 - https://blog.invisiblethings.org/2015/10/27/x86_harmful.html
- **“Rootkit in your laptop”** by Igor Skochinsky
 - https://osresearch.net/PDFs/Rootkit_in_your_laptop.pdf
- **“Intel ME Cleaner: How does it work?”** by Nicola Corna
 - https://github.com/corna/me_cleaner/wiki/How-does-it-work%3F

Thank you

Questions?