coreboot

14 Dicembre 2016

Politecnico Open unix Labs

Nicola Corna

nicola@corna.info

github.com/corna

Federico Izzo

federico.izzo42@gmail.com

github.com/Nimayer

Indice

- **Coreboot**
  - **Cos'è?**
  - **Come si installa?**
- **Intel ME**

Coreboot: cos'è?

Coreboot è un progetto volto a sostituire il firmware presente nella maggior parte dei computer.

Potremmo dire che Coreboot è un'implementazione di un BIOS libero, ma non sarebbe completamente vero.

Perchè Coreboot non è propriamente un BIOS

- Un BIOS inizializza l'hardware e fornisce delle chiamate al sistema in esecuzione
- Coreboot inizializza solamente l'hardware
  - Coreboot + SeaBIOS è un BIOS completo

Vantaggi

- FOSS software
  - Sicuro
  - "Hackable"
  - Libero da backdoor nel BIOS
- Molto veloce (0.5/1 s per iniziare a caricare il kernel Linux)
- Scritto quasi completamente in C 32-bit
  - Mentre tutti i BIOS commerciali sono scritti completamente in assembler 16-bit
- Sviluppato secondo la filosofia "*facciamo il minimo indispensabile, poi togliamoci dai piedi*"

Svantaggi

- Supportato da pochi computer
- Difficile da compilare
- Difficile da installare
- Sviluppo e installazione sempre più difficili con le nuove generazioni di processori
  - Intel Boot Guard

Funzionamento

Coreboot è suddiviso in quattro fasi principali:

- Bootblock
- Romstage
- Ramstage
- Payload

Bootblock

In questa primissima fase Coreboot:

- Legge il contenuto della CMOS
- Decide quale modalità eseguire successivamente (*Normal* o *Fallback*)

Romstage

Questa è la fase più critica, in cui Coreboot inizializza la RAM e Intel ME.

- Inizializza le periferiche per il debugging
- Legge le proprietà del chipset
- Legge le configurazioni delle RAM dall'SPD o dall'XMP
- Decide la configurazione migliore e la applica
- Controlla se la RAM funziona
- Alloca la memoria richiesta da Intel ME

Ramstage

In questa fase coreboot inizializza il resto delle periferiche ed esegue il codice del payload.

Coreboot ha terminato il suo scopo ed esce di scena: fino allo spegnimento/sospensione non sarà più eseguito codice di Coreboot.

Payload

L'hardware è ora inizializzato ed è giunto il momento che un altro software continui con l'avvio del computer.

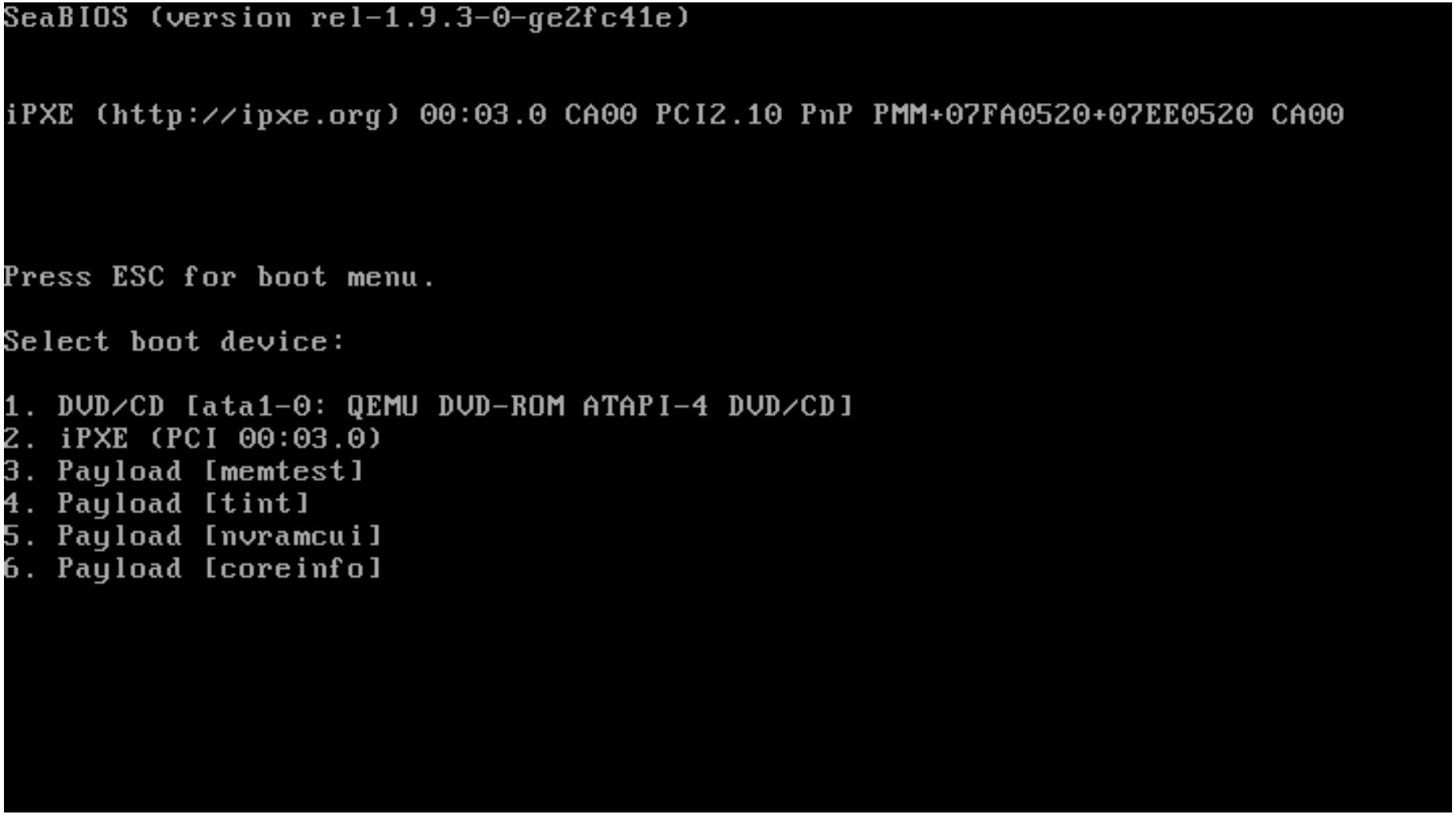
I payload più interessanti sono:

- SeaBIOS
- Tianocore (UEFI)
- GRUB
- Linux

Ci sono poi anche dei payload "secondari" che possono essere lanciati dal menu interattivo di SeaBIOS o GRUB:

- nvramcui
- coreinfo
- Memtest86+
- Tint
- GRUB invaders

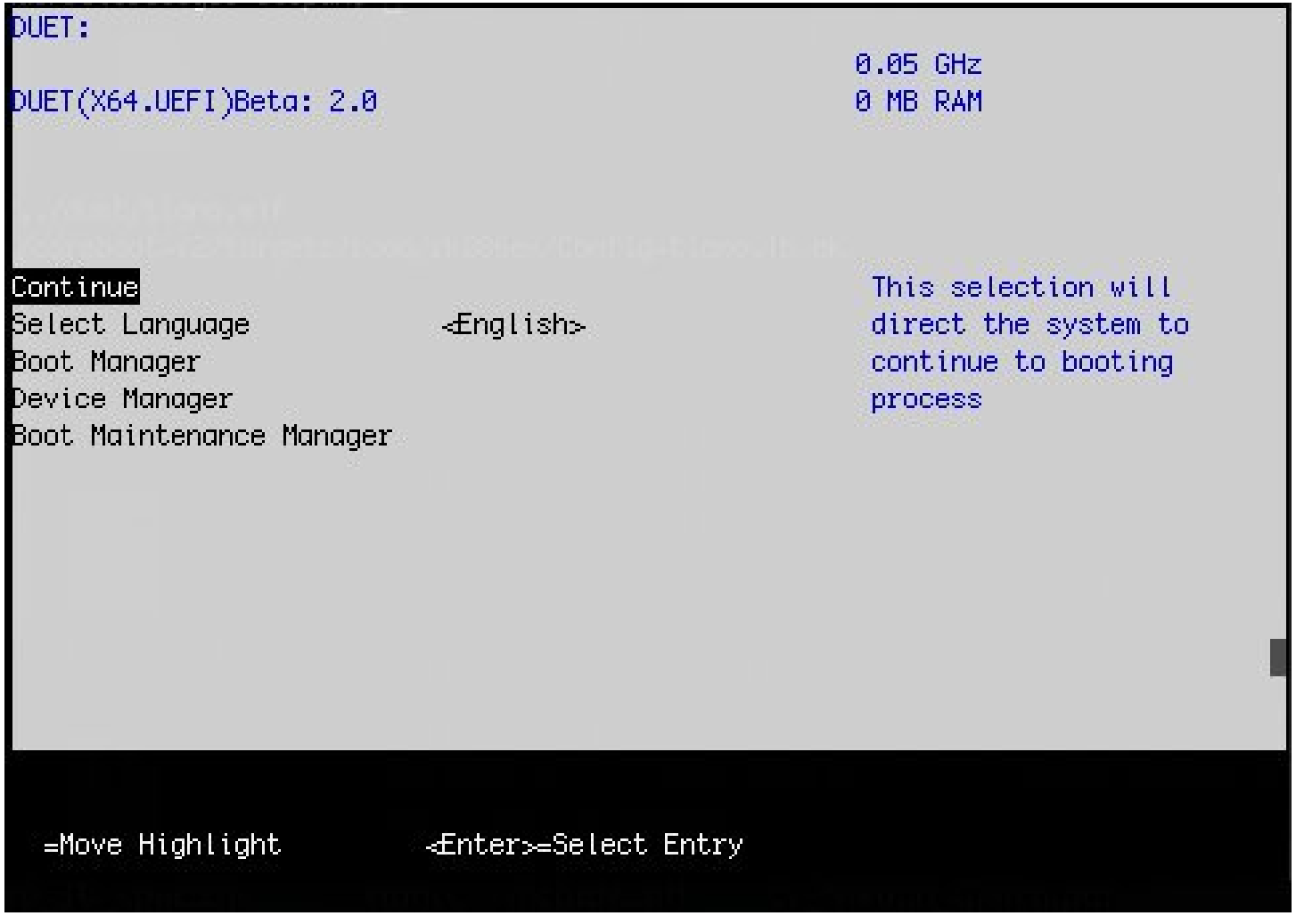
SeaBIOS


SeaBIOS (version rel-1.9.3-0-ge2fc41e)
iPXE (http://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+07FA0520+07EE0520 CA00
Press ESC for boot menu.
Select boot device:
1. DVD/CD [ata1-0: QEMU DVD-ROM ATAPI-4 DVD/CD]
2. iPXE (PCI 00:03.0)
3. Payload [memtest]
4. Payload [tint]
5. Payload [nvramcui]
6. Payload [coreinfo]

Un classico BIOS x86.

Coreboot + SeaBIOS è la soluzione standard, che permette di avere un BIOS "standard".

Tianocore

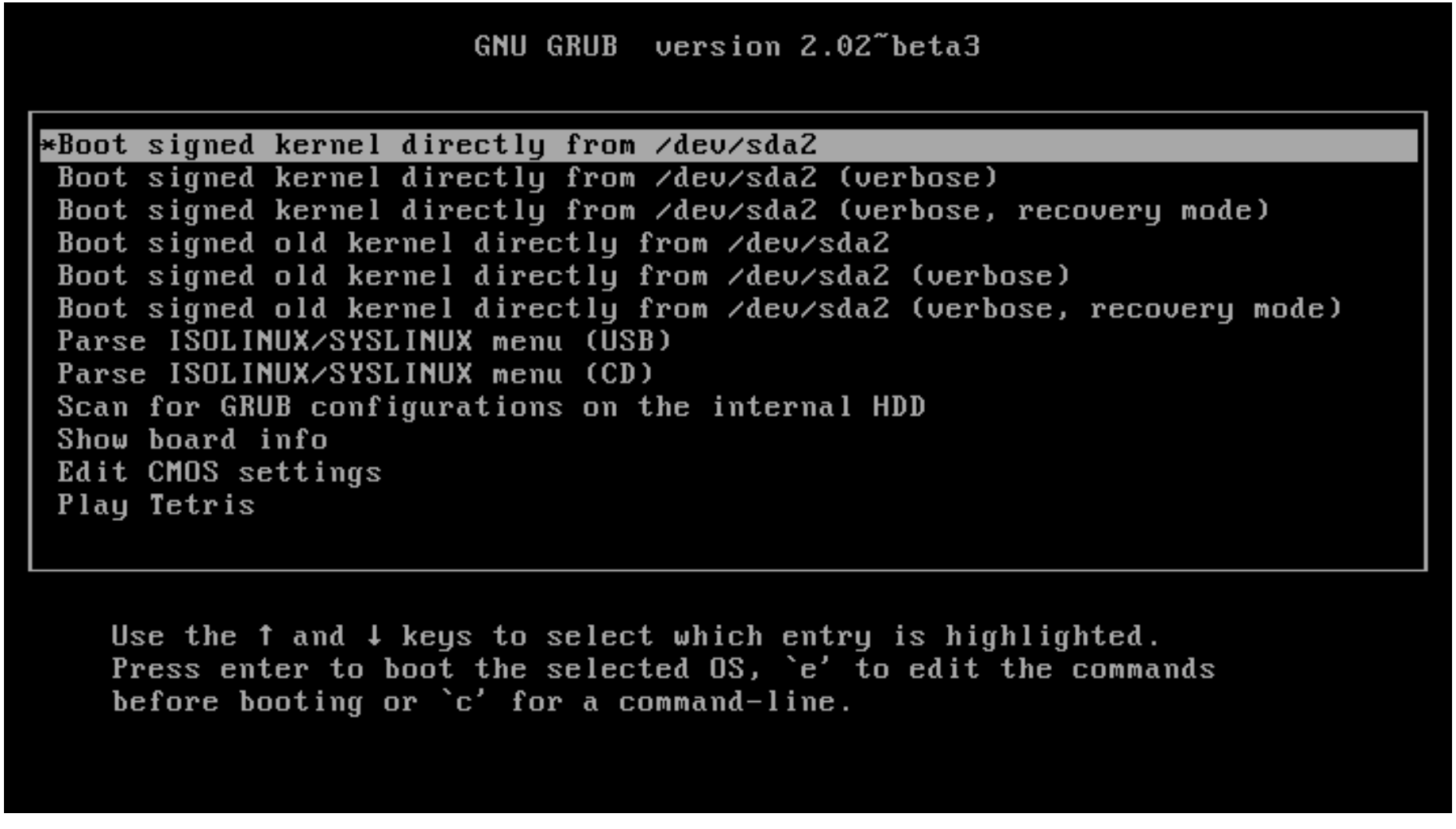

DUET:
0.05 GHz
DUET(X64.UEFI)Beta: 2.0
0 MB RAM
Continue
Select Language
<English>
Boot Manager
Device Manager
Boot Maintenance Manager
This selection will direct the system to continue to booting process
=Move Highlight
<Enter>=Select Entry

Tianocore è la *reference implementation* di UEFI di Intel, rilasciata sotto licenze open.

Duet è uno dei progetti di Tianocore, che permette di avere UEFI con coreboot (se riuscite a farlo funzionare, io non ci riesco).

Tianocore può anche includere SeaBIOS come CSM, in modo da avere un sistema UEFI + BIOS.

GRUB


GNU GRUB version 2.02~beta3
*Boot signed kernel directly from /dev/sda2
Boot signed kernel directly from /dev/sda2 (verbose)
Boot signed kernel directly from /dev/sda2 (verbose, recovery mode)
Boot signed old kernel directly from /dev/sda2
Boot signed old kernel directly from /dev/sda2 (verbose)
Boot signed old kernel directly from /dev/sda2 (verbose, recovery mode)
Parse ISOLINUX/SYSLINUX menu (USB)
Parse ISOLINUX/SYSLINUX menu (CD)
Scan for GRUB configurations on the internal HDD
Show board info
Edit CMOS settings
Play Tetris
Use the ↑ and ↓ keys to select which entry is highlighted.
Press enter to boot the selected OS, `e' to edit the commands
before booting or `c' for a command-line.

GRUB non ha bisogno di presentazione, sapete già cos'è.

La cosa che non sapete è che GRUB può essere direttamente lanciato da Coreboot senza un BIOS. Questo grazie al fatto che Linux non usa le chiamate BIOS legacy.

Rispetto a SeaBIOS abbiamo alcuni vantaggi:

- Velocità
- Sicurezza
- Crypto integrata
  - Aprire volumi LUKS
  - Verificare kernel/initramfs firmati

Linux

```
[    4.040228] hub 4-0:1.0: 5 ports detected
[    4.040627] Initializing USB Mass Storage driver...
[    4.040815] usbcore: registered new interface driver usb-storage
[    4.040821] USB Mass Storage support registered.
[    4.040989] usbcore: registered new interface driver libusual
[    4.041313] mice: PS/2 mouse device common for all mice
[    4.041451] usbcore: registered new interface driver xpad
[    4.041456] xpad: X-Box pad driver
[    4.044607] usbcore: registered new interface driver hiddev
[    4.044729] usbcore: registered new interface driver usbhid
[    4.044734] usbhid: USB HID core driver
[    4.045228] snd_xenon: iobase_phys=0x200ea001600 iobase_virt=0xd000080081074600
[    4.045262] xenon_snd: give me an interrupt, please!
[    4.045271] snd_xenon: irq=40
[    4.045292] snd_xenon: descr_base_virt=0xc00000001836e000, descr_base_phys=0x1836e000
[    4.045510] snd_xenon: driver initialized
[    4.046474] ALSA device list:
[    4.046479]   #0: Xenon AudioPCI at 0x200ea001600 irq 64
[    4.046546] TCP cubic registered
[    4.046555] NET: Registered protocol family 17
[    4.046621] Registering the dns_resolver key type
[    4.417774] ata2.00: 488397168 sectors, multi 16: LBA48 NCQ (depth 0/32)
[    4.441602] ata2.00: configured for UDMA/133
[    4.456622] scsi 1:0:0:0: Direct-Access     ATA      ST9250315AS      0002 PQ: 0 ANSI: 5
[    4.476138] sd 1:0:0:0: [sda] 488397168 512-byte logical blocks: (250 GB/232 GiB)
[    4.476733] sd 1:0:0:0: Attached scsi generic sg1 type 0
[    4.510622] sd 1:0:0:0: [sda] Write Protect is off
[    4.525934] sd 1:0:0:0: [sda] Mode Sense: 00 3a 00 00
[    4.541440] sd 1:0:0:0: [sda] Write cache: enabled, read cache: enabled, doesn't support DPO or FUA
[    4.592956] usb 2-2: new high speed USB device using ehci_hcd and address 3
[    4.593545]  sda: unknown partition table
[    4.626206] sd 1:0:0:0: [sda] Attached SCSI disk
[    4.748153] scsi2 : usb-storage 2-2:1.0
[    4.869613] usb 2-3: new high speed USB device using ehci_hcd and address 4
[    5.359608] usb 4-1: new low speed USB device using ohci_hcd and address 2
[    5.545259] input: SINO WEALTH USB KEYBOARD as /devices/pci0000:00/0000:00:05.0/usb4/4-1/4-1:1.0/input/input0
[    5.566939] generic-usb 0003:258A:0001.0001: input,hidraw0: USB HID v1.10 Keyboard [SINO WEALTH USB KEYBOARD] on usb-0000:00:05.0-1/input0
[    5.569613] Sending DHCP requests .
[    5.610876] input: SINO WEALTH USB KEYBOARD as /devices/pci0000:00/0000:00:05.0/usb4/4-1/4-1:1.1/input/input1
```

Coreboot può lanciare direttamente un kernel linux + initramfs contenuto nella ROM.

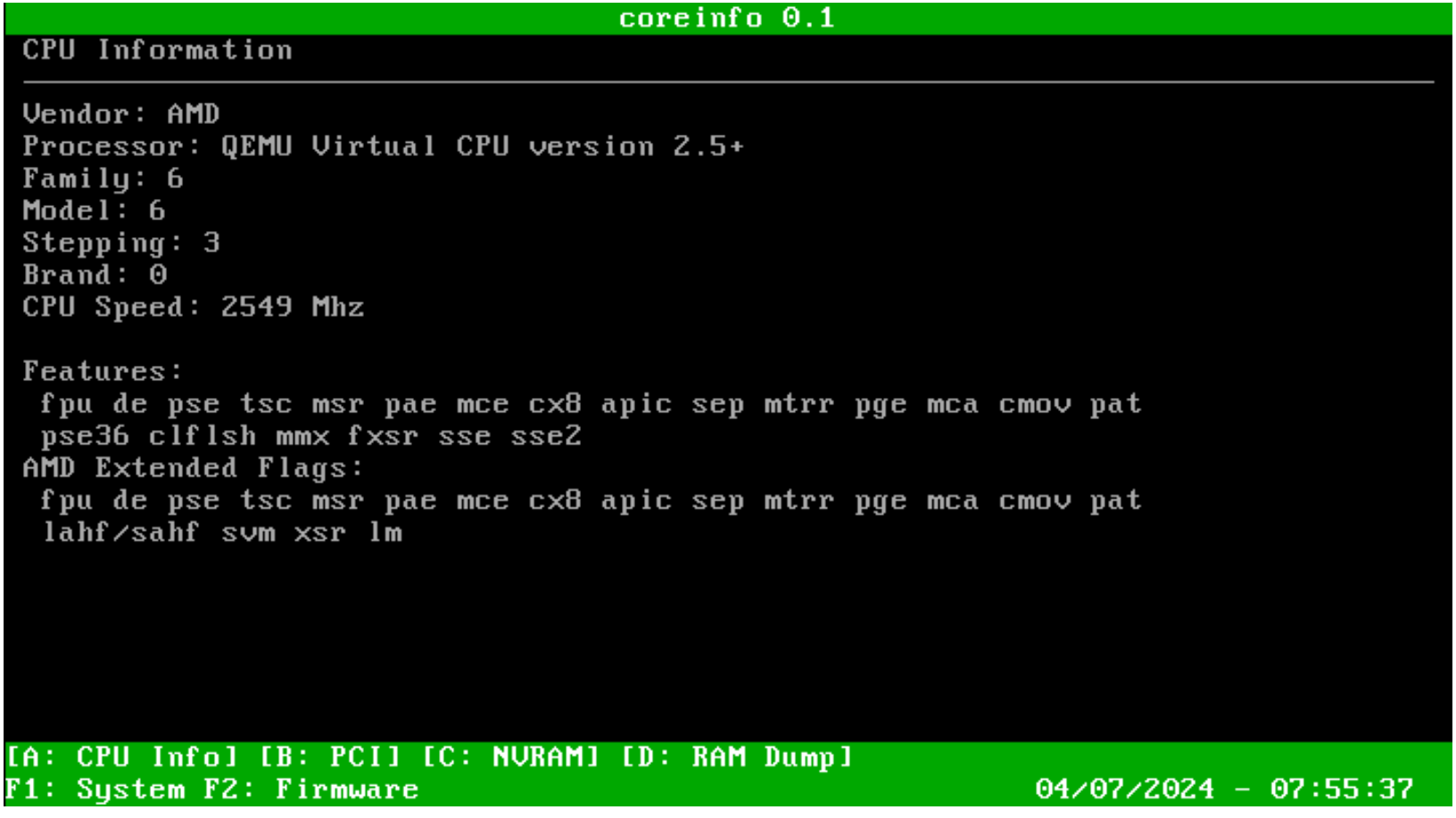
IMO, non particolarmente utile: ad ogni aggiornamento del kernel/initramfs/opzioni del kernel è necessario riflashare.

Inoltre se sbagliate qualcosa e Linux non si avvia, non ci sono molte opzioni disponibili se non riflashare Coreboot con un programmatore esterno.

nvramcui

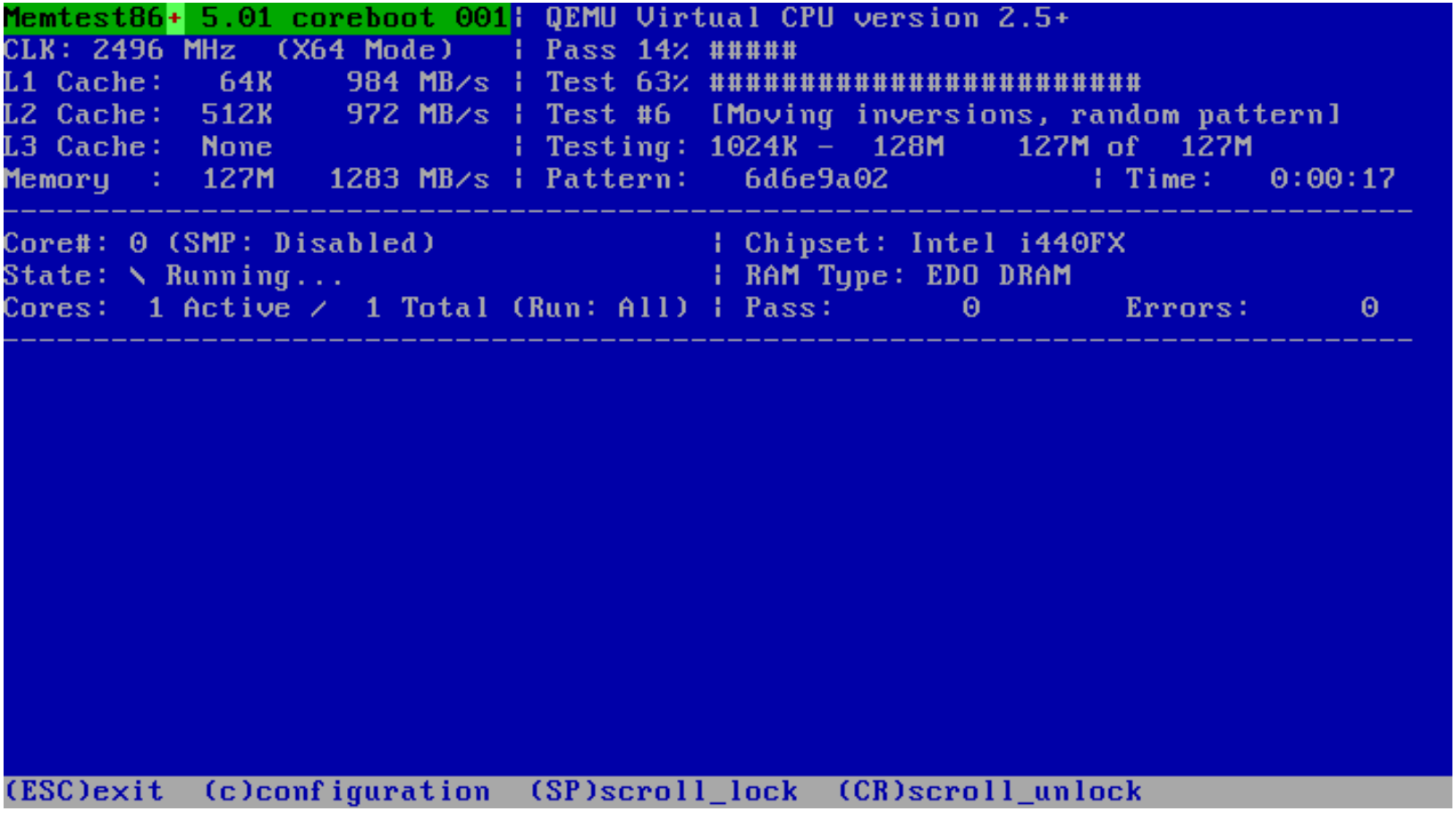
Un'utility per cambiare la configurazione CMOS.

coreinfo



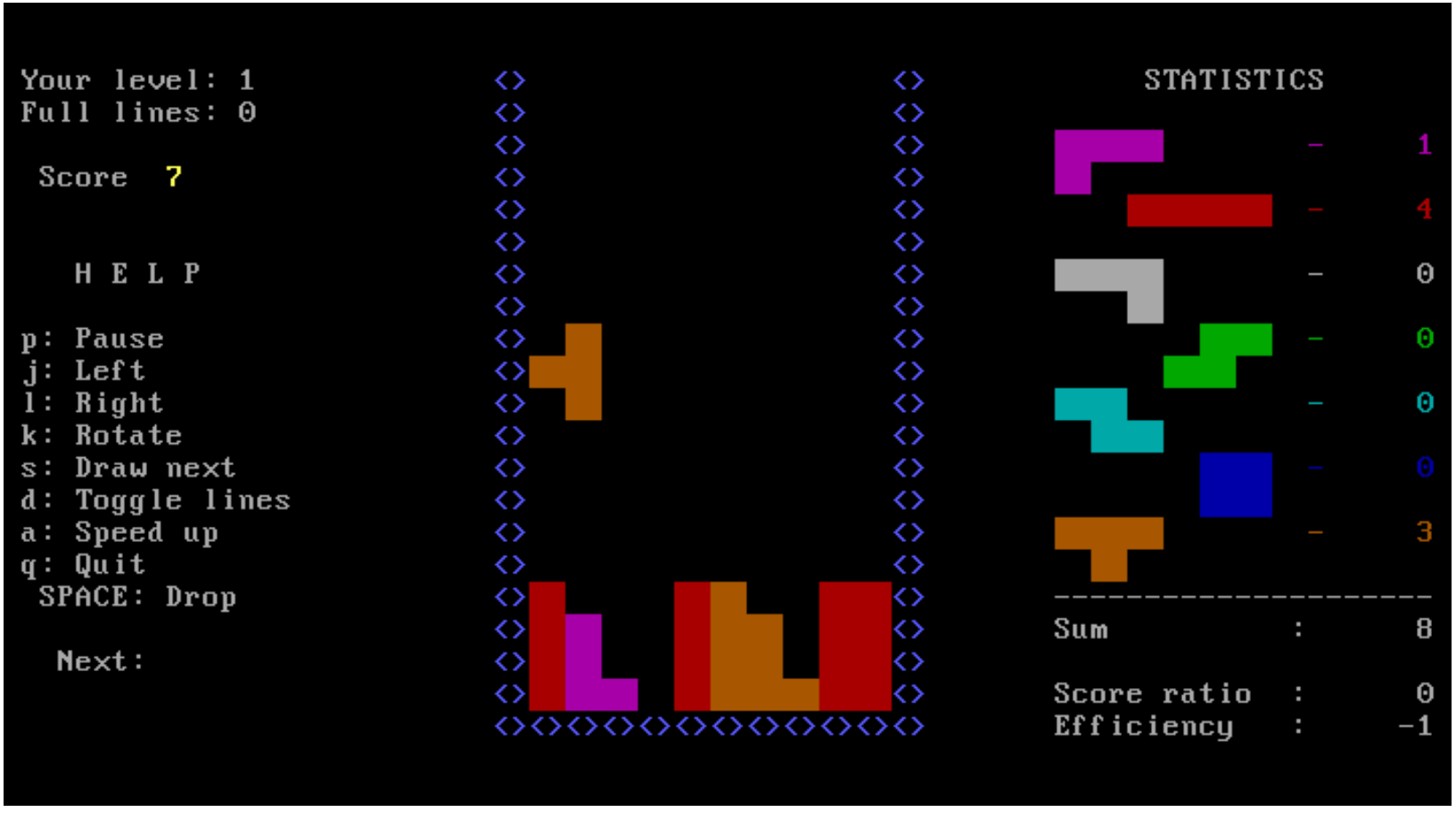
Un'utility per vedere varie informazioni di sistema.

Memtest86+



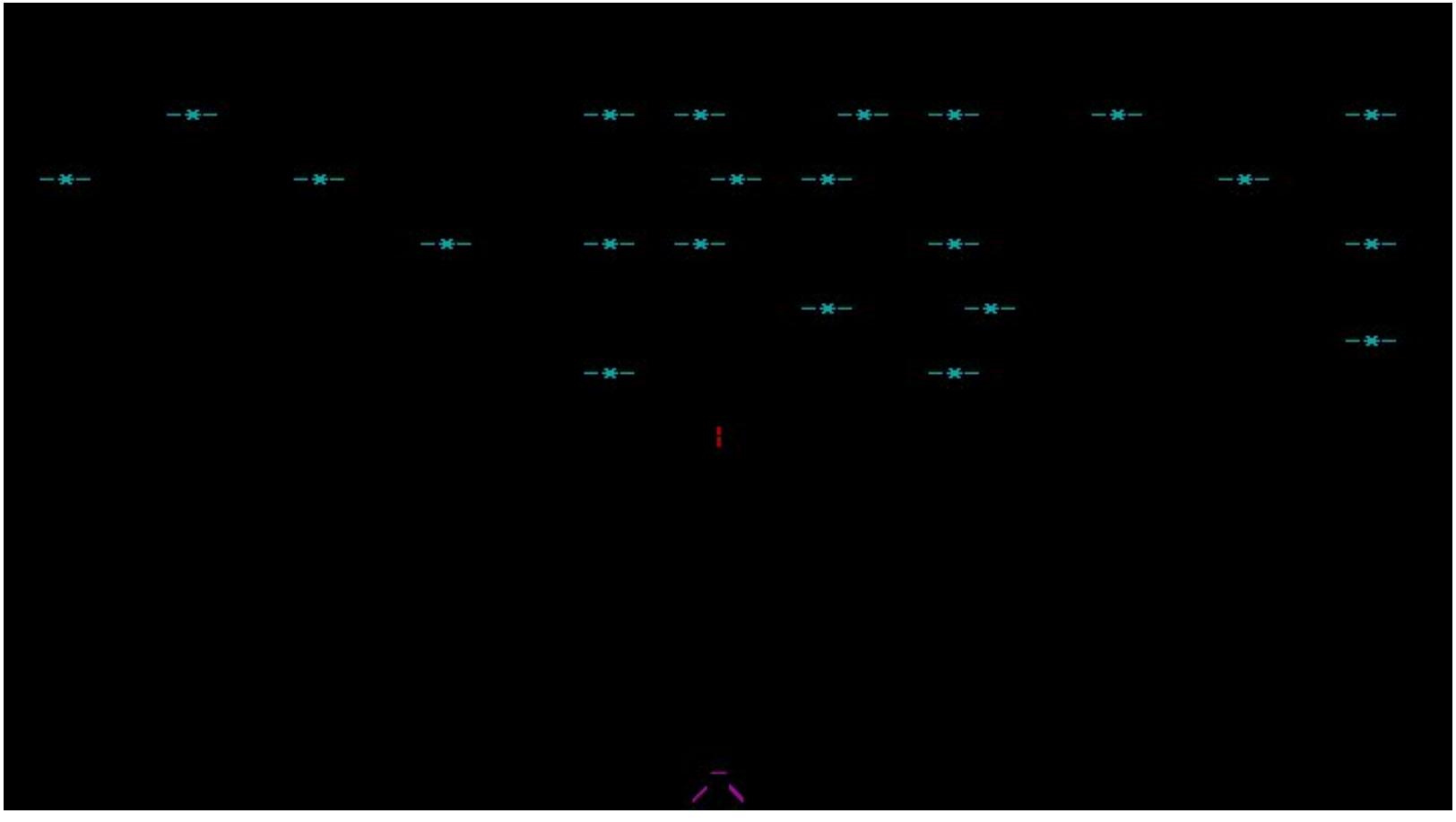
Un software per controllare lo stato della propria RAM.

TinT (Tint is not Tetris)



TETRIS!!!

GRUB invaders



Space invaders!!!

Coreboot: come si installa?

Le fasi dell'installazione sono:

- Preparare l'ambiente di compilazione
- Fare un dump della BIOS originale
- Compilare coreboot
- Flashare l'immagine di coreboot

L'ambiente di compilazione

qui c'é il link alle istruzioni ufficiali, ma seguono un ordine discutibile

Quello che dovrete fare sará:

- Clonare il repository di coreboot

```
$ git clone --recursive http://review.coreboot.org/p/coreboot
$ cd coreboot
```

- Compilare il *cross-compilatore*, coreboot per ora gira a 32bit

```
make crossgcc-i386 CPUS=4
```

- Configurare coreboot

```
make menuconfig
```

Provatelo con QEMU!

É possibile provare coreboot+payload su QEMU prima di passare all'hardware

Lanciate `make menuconfig` per configurare coreboot

e controllate che nel menu *Mainboard* sia selezionato:

- vendor: Emulation
- model: QEMU x86 q35/ich9

Uscite dal menuconfig e usate `make -jN` per compilare

Il file `coreboot.rom` nella sottocartella `build` é la vostra immagine

potete lanciare qemu con

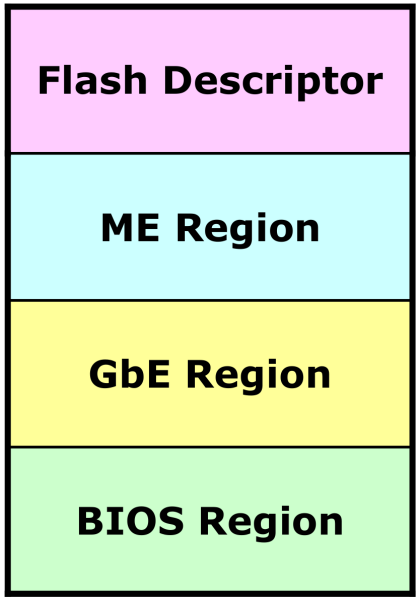
```
qemu-system-x86_64 -M q35 -bios build/coreboot.rom
```

Per compilare un'immagine per il vostro portatile

vi servirá fare un *dump* della flash originale, per estrarre:

- Intel Flash Descriptor
- Firmware Intel ME
- Firmware Gigabit Ethernet
- VBIOS GPU Intel
  (opzionale)

Cosa contiene la flash di un PC Intel:



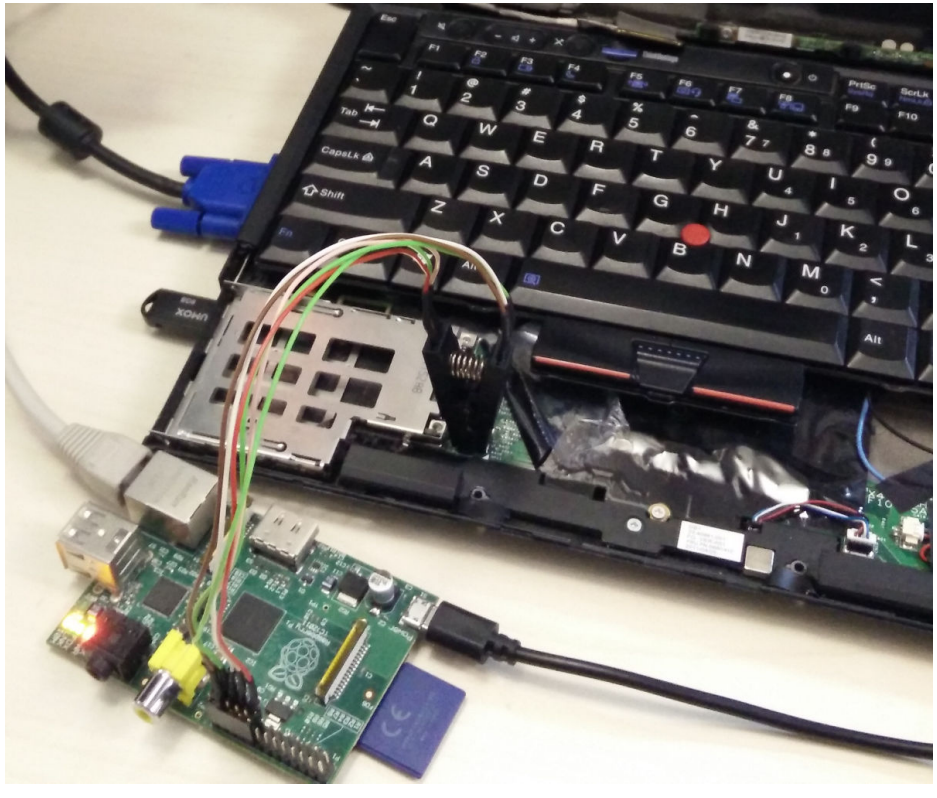
La regione di Intel ME é accessibile **solo da ME stesso**, inoltre le BIOS possono essere **protette da scrittura**.

É possibile peró leggere o scrivere l'intera flash collegandosi fisicamente al chip.

Dumping the hard way

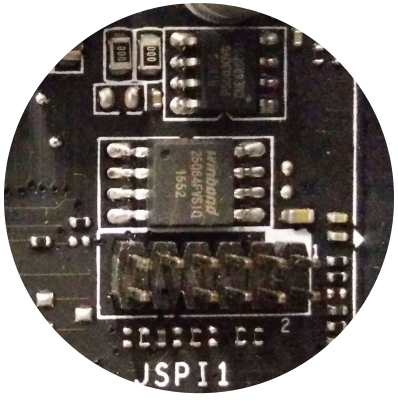
La flash usa il protocollo **SPI**,

Quindi é possibile leggerne il contenuto utilizzando l'interfaccia SPI di un Raspberry Pi o una board equivalente con GPIO a 3.3V.



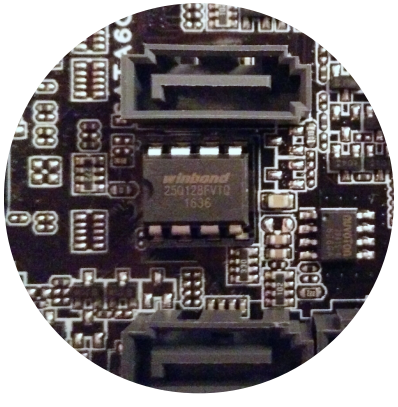
Trovare la flash

SOIC-8



SOIC-16

DIP-8



PLCC-32

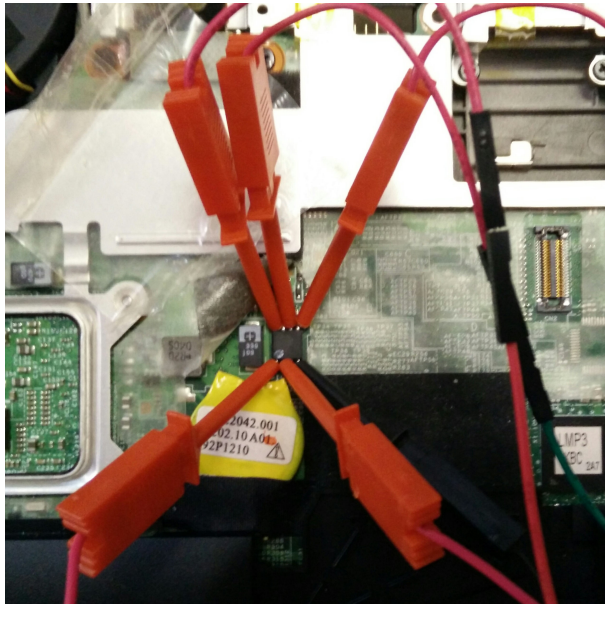
Pinze!

Cercate il datasheet della flash per trovare il **pinout**

Per collegare il chip potete usare delle

testclip SOIC-8

clip SMD

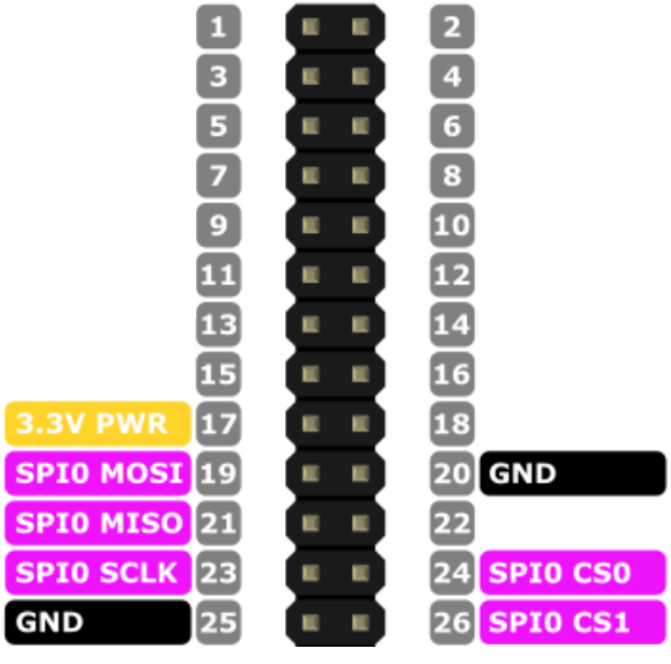


Secondo noi le clip SMD funzionano meglio

Collegare i fili

Prima di tutto **scollegate alimentazione e batteria dal PC**

pin del Raspberry Pi



da collegare in questo ordine

RPi	**Flash**
GND	GND
CS0	CS
SPI0 SCLK	CLK
3.3V PWR	3.3V
SPI0 MISO	MISO
SPI0 MOSI	MOSI

Flashrom

Installate **flashrom** dal repo github o dal vostro package manager

La sintassi per il Raspberry Pi é:

```
flashrom -p linux_spi:dev=/dev/spidev0.0 -r dump.bin
```

Se flashrom non é sicuro su quale sia il vostro chip vi chiederá di specificarlo con l'opzione `-c <chipname>`

(ad esempio, con un X220 dovrete usare `-c W25Q64.V`)

Un consiglio é di fare due dump su due file diversi e confrontarli con `diff` per essere sicuri di avere un dump valido.

Estrarre i blob

Useremo il programma `ifdtool` presente nella sottocartella `util` della repo di coreboot

- Compilate il programma

```
cd coreboot/util/ifdtool
make
```

- Estraete le regioni della flash

```
mkdir extracted_dump
cp dump.bin extracted_dump/
./util/ifdtool/ifdtool -x extracted_dump/dump.bin
```

- Troverete nella cartella le sezioni estratte dal dump della flash,

ovvero BIOS, blob ME, blob GbE, Flash Descriptor

Configurazione

Coreboot usa una configurazione simile al kernel linux

Potete usare `make menuconfig` per aprire la configurazione e la funzione **help** se avete dubbi.

Vi mostreró una configurazione standard, sta alla vostra curiositá scoprire e provare le varie funzioni (hint: normal/fallback).

Configurare coreboot pt.I

Le cose importanti da scegliere sono:

- Mainboard
  - Mainboard vendor: *produttore del vostro pc*
  - Mainboard model: *il modello del vostro pc*
  - Rom chip size: *dimensione della vostra flash*
- Chipset
  - Include microcode in CBFS: *Generate from tree*
  - Add Intel descriptor.bin file: *selezionare il path*
  - Add Intel ME/TXT firmware: *stessa cosa*
  - Add gigabit ethernet firmware: *stessa cosa*

Configurare coreboot pt.II

- Devices
  - Use native graphics initialization: *generalmente funziona*
  - Enable PCIe Clock Power Management: *buona idea*
- Display
  - Keep VESA framebuffer: *modalitá grafica invece di testuale*
- Generic Drivers
  - Enable TPM support
- payload
  - Add a payload: *SeaBIOS o quello che preferite*
  - Secondary Payloads: *vedi qui*

Compilare

Per compilare usate `make -jN`

Il risultato sará l'immagine in `coreboot/build/coreboot.rom`

Flashare coreboot

Per flashare la prima volta coreboot serve lo stesso setup del *dump*

Dalle volte successive é possibile flashare coreboot **da linux**, in quanto coreboot non protegge il BIOS / blob ME da scrittura a meno che non lo vogliate.

Il comando per flashare esternamente tramite Raspberry Pi é:

```
flashrom -c <chipname> -p linux_spi:dev=/dev/spidev0.0 -w coreboot.rom
```

force_I_want_a_brick

Quando avrete un sistema funzionante, potrete aggiornare coreboot da linux tramite il comando:

```
flashrom -c <chipname> -p internal:laptop=force_I_want_a_brick -w coreboot.rom
```

Dopo aver aggiornato coreboot vi consiglio di **spegnere completamente il PC** e riaccenderlo, in modo da caricare la nuova BIOS / blob ME

Approfondimento: CBFS

Il BIOS é organizzato come un filesystem in cui sono memorizzati romstage, ramstage, blob, payload, configurazioni

Il comando `cbfstool` presente in `coreboot/util` o in `build` dopo la compilazione permette di

elencare i file

```
./util/cbfstool/cbfstool build/coreboot.rom print
```

o modificarli

```
./build/cbfstool build/coreboot.rom add -f <file da inserire> -n <nome file> -t
```

Necessario per aggiungere il file `grub.cfg` all'immagine nel caso scegliate GRUB come payload

Approfondimento: cbmem

Coreboot usa un buffer in memoria per salvare:

- log di avvio
- statistiche sul tempo di avvio (tramite *timestamps*)

Il comando `cbmem` permette di mostrare questi dati, compilandolo prima con `make`:

Per mostrare il log

```
sudo ./util/cbmem -c
```

Per mostrare il tempo di avvio di sistema

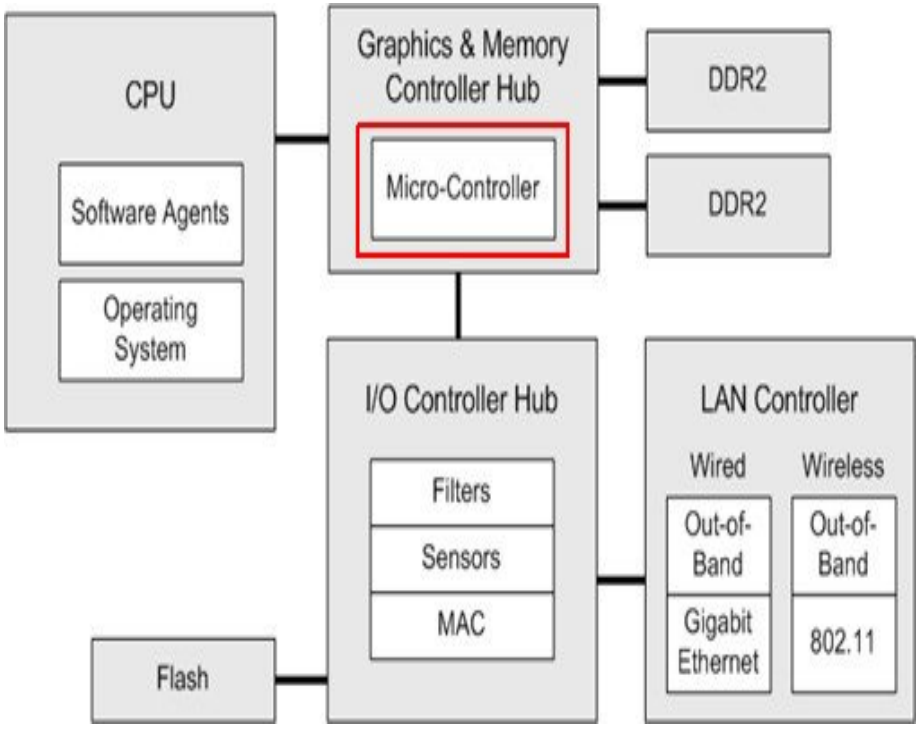
```
sudo ./util/cbmem -t
```

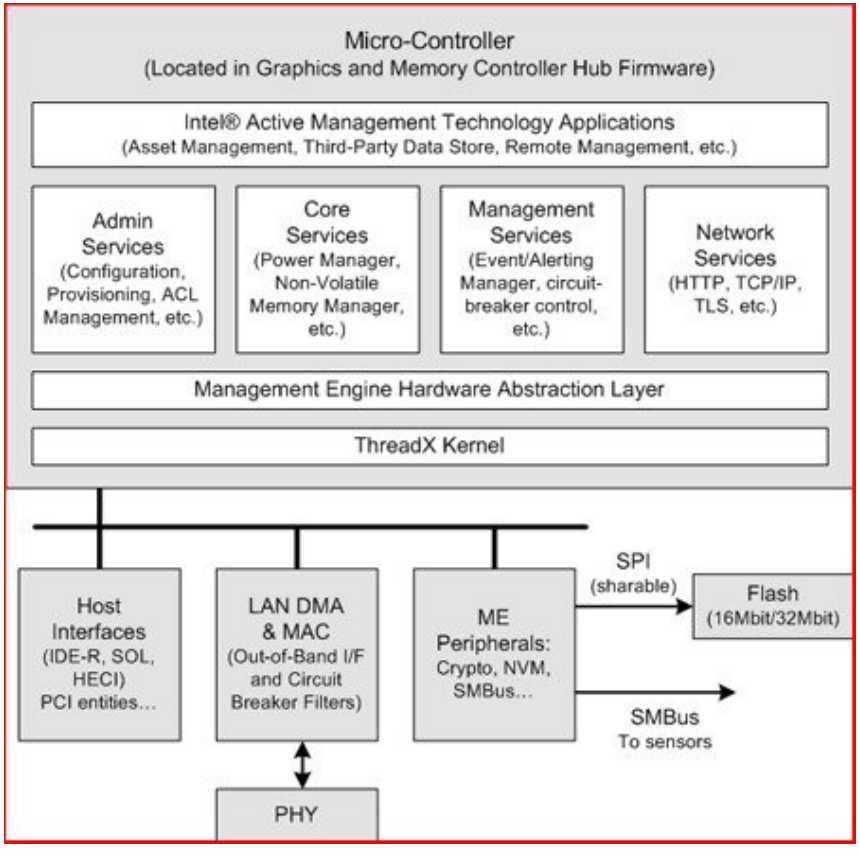
Intel ME

Intel ME è un coprocessore integrato in tutti i processori Intel moderni che costituisce l'hardware principale per Intel AMT (Advanced Management Technology, componente principale di Intel vPro), una serie di servizi:

- controllo remoto
- accensione remota
- KVM
- PAVP per DRM
- ...

Intel ME


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
Out-of-Band
Gigabit Ethernet
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

Intel ME ha accesso a:

- tutta la memoria
- il bus PCI
- alla GPU
- rete cablata e wireless (con MAC e IP separati)
- ...

Il firmware (ovviamente un blob non-free) per Intel ME è contenuto nel chip del BIOS, ed è firmato con chiave RSA Intel.

Molti dei moduli sono inoltre compressi tramite Huffman, con dizionario contenuto in hardware, quindi il loro codice non è accessibile.

Come lo rimuovo?

Prima di Nehalem era possibile rimuovere completamente il firmware di ME e il PC si avviava correttamente.

Da Nehalem in poi se non viene fornito alcun firmware il PC si accende correttamente, ma dopo 30 minuti (controllati tramite timer interno) ME spegne forzatamente il PC (probabilmente come misura di sicurezza per evitare il bypass di Intel Antitheft).

Risultato

In tutti i PC moderni è installata di fatto una backdoor, non rimovibile, con completo accesso a tutte le risorse della macchina, rete inclusa.

Inoltre, grazie al WOL, il sistema non è sicuro nemmeno a PC spento.

Davvero non si può fare nulla?

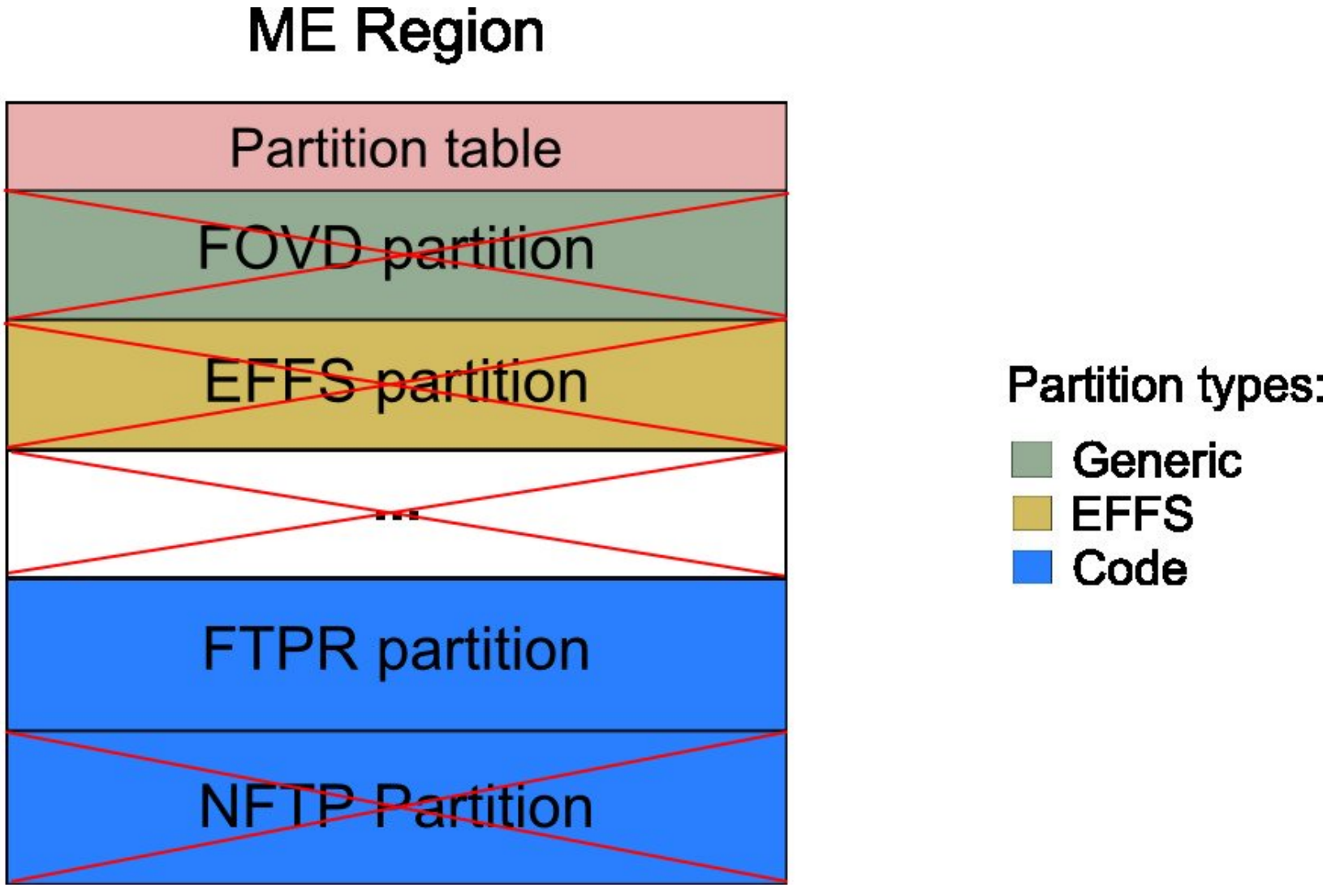
Fortunatamente Igor Skochinsky, un ricercatore indipendente, ha reversato la struttura del firmware di Intel ME e l'ha pubblicata qui:

me.bios.io/ME_blob_format

A Settembre 2016 Trammel Hudson ha scoperto che rimuovendo i primi 4 kB dal firmware di Intel ME del suo X230 con Coreboot, il PC si avviava correttamente e non si spegneva dopo 30 minuti.

Da questa scoperta ha iniziato ad indagare e ha scoperto che è possibile rimuovere:

- tutte le partizioni meno quella fondamentale
- tutti i moduli compressi con LZMA dalla partizione fondamentale


ME Region
Partition table
FOVD partition
EFFS partition
...
FTPR partition
NFTP Partition
Partition types:
Generic
EFFS
Code

Pur non avendo rimosso completamente Intel ME questo processo ne ha limitato fortemente le capacità in quanto rimuove:

- l'accesso di rete (contenuto in una partizione separata, NFTP)
- il PAVP (Protected Audio-Video Path)
- la JVM

A Novembre 2016 io e Federico abbiamo iniziato a fare test con Intel ME nel tentativo di scoprire quanto codice potevamo rimuovere.

A tal proposito ho scritto uno script Python per rimuovere quanto più codice possibile da un firmware ME.

github.com/corna/me_cleaner

Abbiamo quindi confermato le scoperte di Trammel Hudson, in particolare che:

- Rimuovere i primi 4 kB (ovvero la tabella delle partizioni) fa sì che Intel ME utilizzi una tabella delle partizioni interna
  - quindi è meglio lasciarla, in modo da poterla modificare (non essendo firmata)
- In tutte le generazioni da Sandy Bridge a Skylake è possibile rimuovere tutte le partizioni meno quella fondamentale
- In tutte le generazioni da Sandy Bridge a Broadwell è possibile rimuovere tutti i moduli compressi LZMA (circa la metà)
- Le modifiche funzionano anche con BIOS OEM

Ogni partizione è firmata singolarmente, mentre la tabella delle partizioni non è firmata: questo significa che rimuovere completamente le partizioni non invalida alcuna firma, ma rimuovere i moduli si.

Allora perchè il sistema parte comunque senza i moduli LZMA (e quindi con una firma non valida)?

Tralasciando l'ipotesi che Intel non verifichi la firma (decisamente poco probabile), l'ipotesi più probabile è che, per poter accendere il PC, Intel ME richiede che la struttura del firmware sia valida, nulla di più.

Questo significa che è probabile che, costruendo un firmware ad-hoc, senza codice, con firme non valide ma struttura corretta, il PC si accenda correttamente.

Rimuovere il firmware di Intel ME è l'ultimo step per poter avere un PC moderno con certificazione RYF (Respects Your Freedom) dalla FSF!

L'ultima domanda che ci si pone è:

come faccio ad essere sicuro che Intel ME non abbia una ROM al suo interno con un firmware da usare nel caso quello esterno non sia valido?

Fortunatamente il lavoro di Igor Skochinsky ci fornisce la risposta: in alcuni firmware di ME è presente una partizione particolare, la **ROMB** (ROM Bypass), una partizione contenente il codice da utilizzare al posto di quello nella ROM interna.

Igor ha analizzato questa partizione di "*update*" e ha scoperto che contiene:

- funzioni standard C (memcpy, memset, strcpy...)
- routine di ThreadX (il kernel di Intel ME)
- API di basso livello per l'accesso all'hardware

Lo scopo del codice nella ROM interna è solamente quello di:

- inizializzare l'hardware di base
- verificare la partizione FTPR
  - e ora sappiamo che l'unica cosa fondamentale è che la struttura sia valida, la firma non è fondamentale
- caricare la FTPR e eseguirne il modulo BUP (Bringup)

Questa tesi è rafforzata dai documenti "Intel Confidential" accidentalmente pubblicati da alcuni produttori, facilmente raggiungibili da Google , che confermano che la partizione ROMB può essere eseguita al posto del codice nella ROM interna.

Binary input file	Navigate to your **Source Directory** (as specified in Section 2.1) and switch to the **Firmware** subdirectory. Choose the ME FW binary image. ***Note:*** You may choose to build the ME Region only. To do so, **Flash Image | Descriptor Region | Descriptor Map** parameter **Number of Flash components** must be set to **0**. ***Note:*** Loading an ME FW binary image that contains ME ROM Bypass unlocks the **ME Boot from Flash** parameter in **Flash Image | Descriptor Region | PCH Straps | PCH Strap 10**.

ME boot from Flash	false (grayed out)	**false (default)** = No ME Region binary loaded, or ME Region binary does not contain ME ROM bypass image **Note: On B0 and later PCH stepping parts this setting should be set to 'false'**

Provate a googlare "loading an ME FW binary image" o "management engine system tools", virgolette incluse.

Se volete testare lo script sul vostro computer potete seguire questa guida:

http://hardenedlinux.org/firmware/2016/11/17/neutralize_ME_firmware_

More info:

Struttura del firmware di Intel ME

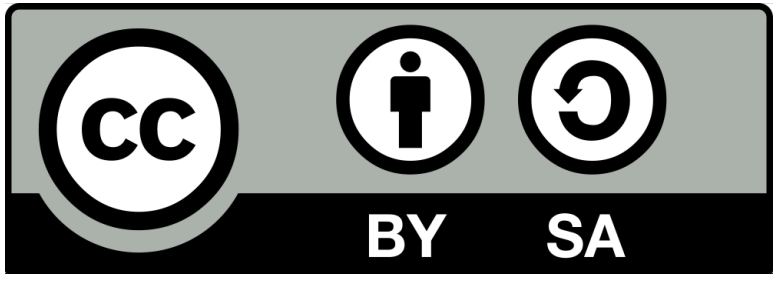
Igor Skochinsky - Rootkit in your laptop - 2012

Igor Skochinsky - Intel ME Secrets - 2014

Trammel Hudson - Coreboot Mailing List

Nicola Corna - Coreboot Mailing List

Thank you!



www.poul.org