Disabling Intel ME in Firmware

Who Am I?

- Brian Milliron
- Founder of ECR Security
- 8 Years as Penetration Tester
- 10 Years as Security Architect/Network Administrator

Brief Explanation of Intel ME

- Chip within a chip
- Physically located on PCH (Platform Controller Hub)
- Closed source
- Includes Intel AMT for remote administration
- Runs MINIX 3 OS
- Included in all Intel chipsets since 2008
- Runs on Intel Quark 32 bit microprocessor
- Filesystem stored in SPI Flash
- Also includes Boot Guard, TPM, DRM module and more

Intel ME capabilities

- Active even in power off state so long as main power is connected
- Has its own network stack for out of band communication with NIC bypassing the OS completely
- Has full read/write access to all areas of system main memory
- Has full access to system bus
- Operates in SMM or protected system management mode so OS cannot see or interfere with its operations
- Basically God Mode. It can read and write to everything on the system.
- If it recieves a magic packet from the NIC AMT can be activated remotely and power on/off system, load/alter the OS, change BIOS settings, view video output, anything a local user at the terminal could do.

History of Bugs in IME

- In 2009 Invisible Things Lab developed a persistent rootkit which lives inside IME
- In 2010 Vassilios Ververis developed a certificate based authentication bypass to remotely enable, deploy and provision AMT even if AMT is disabled.
- SA-00075 aka CVE-2017-5689 aka Silent Bob is Silent
  - Remotely exploitable authentication bypass vulnerability, which allows attacker to send blank password to AMT and get access to full remote management capabilities (God Mode)
  - AMT must be enabled
  - Listens on ports 623, 664, 16992-16995
  - Affects PCs, servers, firewalls, HSMs and other security appliances. Anything with an Intel chip.
- SA-00086 aka CVE-2017-5705
  - Buffer overflow leading to remote code execution

How Intel Locks Us Into Using IME

- IME firmware located in region of SPI flash that is inaccessible to BIOS/OS
- IME code modules are RSA signed. If signature verification fails system will not boot.
- LZMA and Huffman compressed with hidden directory to prevent reverse engineering
- Additional IME verification module runs every 30 min to check if valid signed IME is present. If not, the computer shuts down

Speculation on Nefarious Possibilities

- NSA rootkit
  - NSA has a history with putting backdoors in enterprise products. Could IME be another example?
  - IME is the perfect backdoor because it operates independently of CPU, cannot be detected by OS level security tools, is always on, can access everything on the system, and has out of band communications for data exfiltration
  - Some of the vulnerabilities could have been intentional to allow NSA remote access while preserving “plausible deniability”
- IME can be used to remotely access encryption keys by reading RAM while system is powered on
- IME can be used to exfiltrate data off air-gapped systems by sending data out over wireless even if the wireless is not enabled or configured
- IME can be used to infect USB devices which then infect other computers

How to Disable IME the Easy Way

- Some vendors allow you to purchase a computer with Intel ME pre-disabled
- Purism
- System 76
- Dell (for gov customers only)

How to Disable IME Yourself

- Igor Skochinsky started the ball rolling by reverse engineering large parts of IME.
- In 2016 Trammel Hudson discovered parts of Intel ME could be overwritten without invalidating the signature checks
- Nicola Corna followed up on the research and created a script to delete most of IME.
- It turns out Intel wasn’t properly implementing integrity verification of the firmware checksums
- You actually can delete ALL of the IME modules except ROMP and BUP
- In 2017 Positive Technologies discovered an undocumented mode called HAP or High Assurance Platform put in at the request of the NSA which effectively disables IME after boot. Turns out the NSA thinks IME is a security risk. Go figure. :)
- Using both techniques together you can disable IME and delete it from firmware to prevent it from reactivating.
- However, since IME is built into the BIOS you need a new BIOS with the modified IME.

Introduction to Coreboot

- Open source BIOS/UEFI firmware
- Supported by Google
- Barebones functionality

- Only initializes hardware then passes control to the OS
- Supports secure boot using VBOOT2 module
- Limited hardware support
  - Works with most Chromebooks
  - Most of the other supported hardware is older models: Intel Ivybridge and Sandybridge or AMD Athlon
  - Some older Macbooks, Thinkpads, and Elitebooks

Hardware Requirements & Cost

- Raspberry Pi $50
- SOIC-8 (or 16) Pomona Clip $10

Additionally for 1.8V chips

- Bi-directional logic level converter $8
- Breadboard $5
- 10nf capacitor $1
- 1.8V/3.3V multi-output linear power supply $20

Preparing Coreboot ROM

- Build coreboot from source
  - https://www.coreboot.org/Build_HOWTO
- Check out submodules
  - git submodule update --init –checkout
- Download and build a payload
  - SeaBIOS (legacy) and Tianocore (UEFI) work well
  - https://www.coreboot.org/SeaBIOS
- Make menuconfig
  - Need to configure Coreboot for your specific hardware
  - Must include proprietary binary blobs specific to the board, ie. video driver, LAN, and this is also where we load the modified IME.
  - Super important to configure the correct location in ROM for the video BIOS or you can get an unbootable system. Always go by the address listed in lspci.

Binary blobs

- The most confusing and difficult part of the process is getting the binary blobs you will need to boot the system.
- You will need the flash descriptor, video BIOS or VBT, Intel ME (cleaned), PCH Reference Code, and Memory Reference Code (MRC). You might also need an Intel Firmware Support Package (FSPS or FSPM).
- First you need a copy of the stock ROM that comes with your PC. The best source is the manufacturer website but you can also copy it from hardware using flashrom.
- Coreboot comes with a helper script extractblobs.sh that should do the messy part for you. If it doesn’t work you need to run the cbfstool binary manually.
- “cbfstool MyBIOS.bin print” will show you all the binaries embedded in your BIOS which you can then extract.

BIOS structure

- MyBIOS.bin: 8192 kB, bootblocksize 2864, romsize 8388608, offset 0x700000
- alignment: 64 bytes, architecture: x86
-

Name	Offset	Type	Size
cmos_layout.bin	0x700000	cmos_layout	1164
pci8086,0406.rom	0x7004c0	optionrom	65536
spd.bin	0x710500	(unknown)	4096
cpu_microcode_blob.bin	0x711540	microcode	70720
fallback/romstage	0x722a00	stage	54210
fallback/ramstage	0x72fe00	stage	96382
config	0x7476c0	raw	6075
fallback/vboot	0x748ec0	stage	15980
fallback/refcode	0x74cd80	stage	75578
fallback/payload	0x75f500	payload	62878
u-boot.dtb	0x76eb00	(unknown)	5318
mrc.bin	0x79ffc0	(unknown)	222876

Extracting binary blobs

- Cbfstool MyBIOS.bin extract -r BOOT_STUB -n fallback/refcode -f refcode.elf -m x86
- This will give you the PCH reference code.
- Cbfstool MyBIOS.bin extract -r BOOT_STUB -n mrc.bin -f mrc.bin
- This gets you the MRC. In order to find the video bios you need to look up the pci address first.
- $ lspci -vnn
- 01:00.0 VGA compatible controller [0300]: VIA Technologies, Inc. UniChrome Pro IGP ["'1106:3344'"] (rev 01) (prog-if 00 [VGA controller])
- Cbfstool MyBIOS.bin extract -r BOOT_STUB -n pci1106,3344.rom -f pci1106,3344.rom
- This gets you the video bios. If you have an embedded ethernet BIOS you'll follow the same steps to extract it.
- Finally you need to run ifdtool to get the rest.
- Ifdtool -x MyBIOS.bin
- This outputs a number of files named "flashregion_#"
- Number 0 will be your flashdescriptor and 2 will be the Intel ME

Neutering IME

- Now comes the fun part!
- Just run the me_cleaner.py script on the IME module you extracted from the BIOS image.
  - python me_cleaner.py -S -O modified_image.bin original_dump.bin
- First you probably want to check that your platform is supported
  - https://github.com/corna/me_cleaner/wiki/me_cleaner-status

Building Final Coreboot ROM

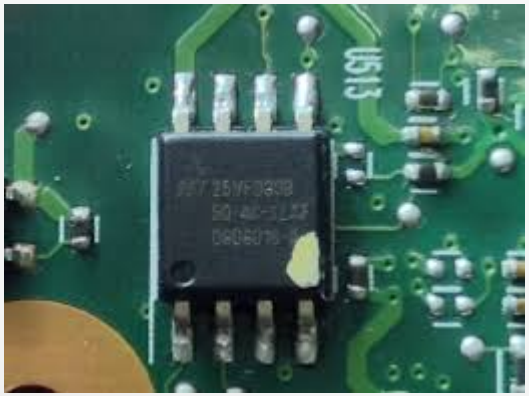
- Now that you have a disabled IME binary you are ready to build the ROM you will be flashing onto firmware
- Make a final check of your config file
  - Make sure coreboot points to the location of all your binary blobs
  - Verify the PCI address of the VGA is correct
  - Make sure coreboot knows the path for the BIOS payload
  - Verify your mainboard model and vendor are correct
  - Take a look at all the other options
- Build!

Setting Up Flashrom on RPi

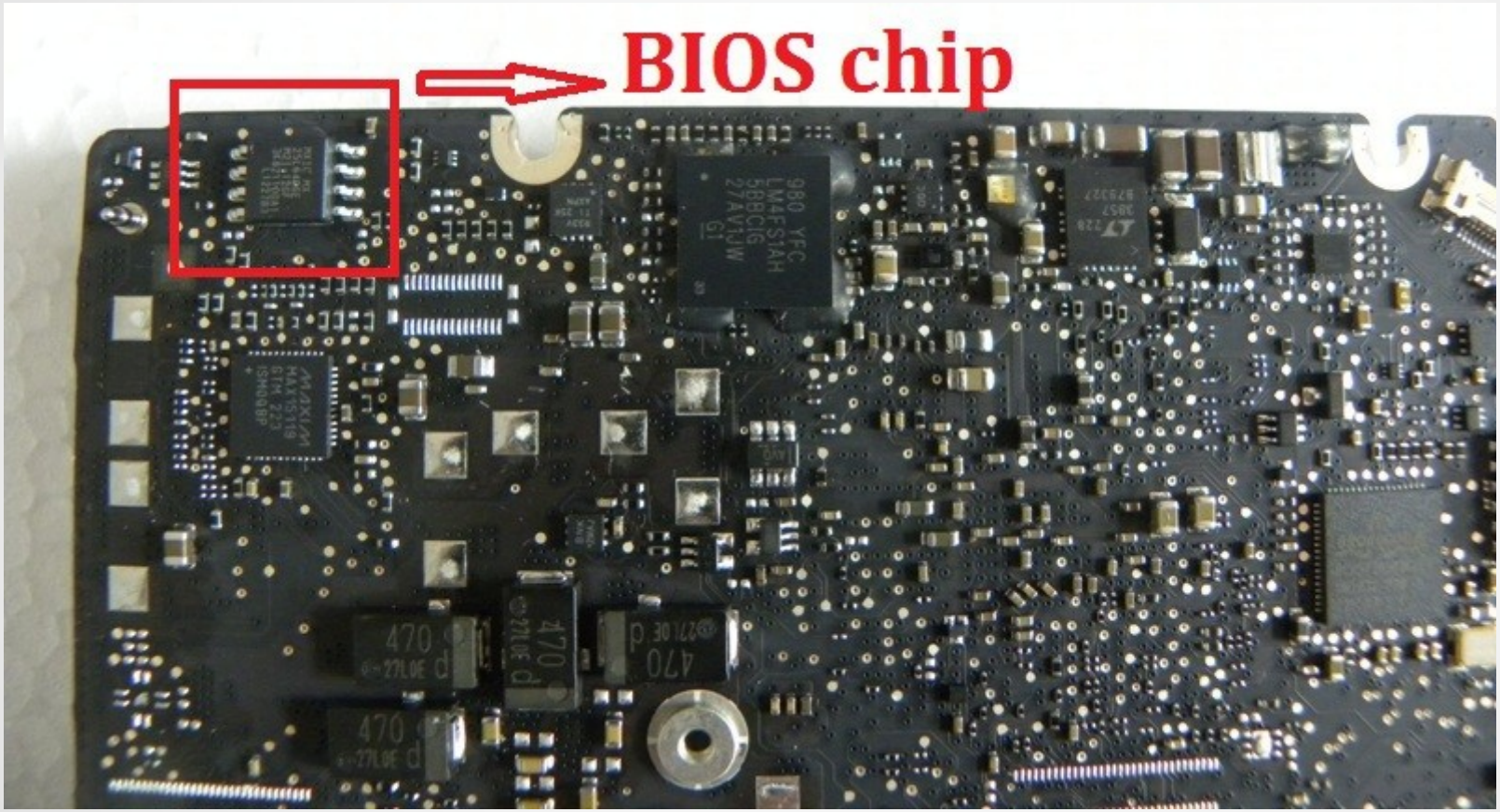
- Now that you have a coreboot ROM ready it’s time to flash
- Check to see if your hardware is supported
  - https://flashrom.org/Supported_hardware
- sudo apt-get install flashrom
  - Sometimes you need to build from source to be able to write to newer chips since the packaged version may be out of date.
- Run raspi-config and enable SPI then reboot

Locating BIOS Chip

- Typically located near CMOS battery
- Most often 8 pin
- Look for any chip with a large G (Gigabyte) or W (Winbond)



Locating your BIOS chip



Identifying BIOS Chip

- Manufacturer’s logo (W for Winbond)
- Model number printed on chip
  - 25 Family
  - X Dual SPI
  - 80 8Mb capacity
  - VAIZ SOIC-8 208 mil

Looking Up Pinout and Voltage Requirements

- You will need to find the datasheet for your specific BIOS chip
- Voltage should be either 3.3V or 1.8V. It may list a voltage range such as 2.7-3.6V That means it's a 3.3V chip. Voltages are rarely exact.
- Pinout Diagram

4. PIN CONFIGURATION SOIC 208-MIL

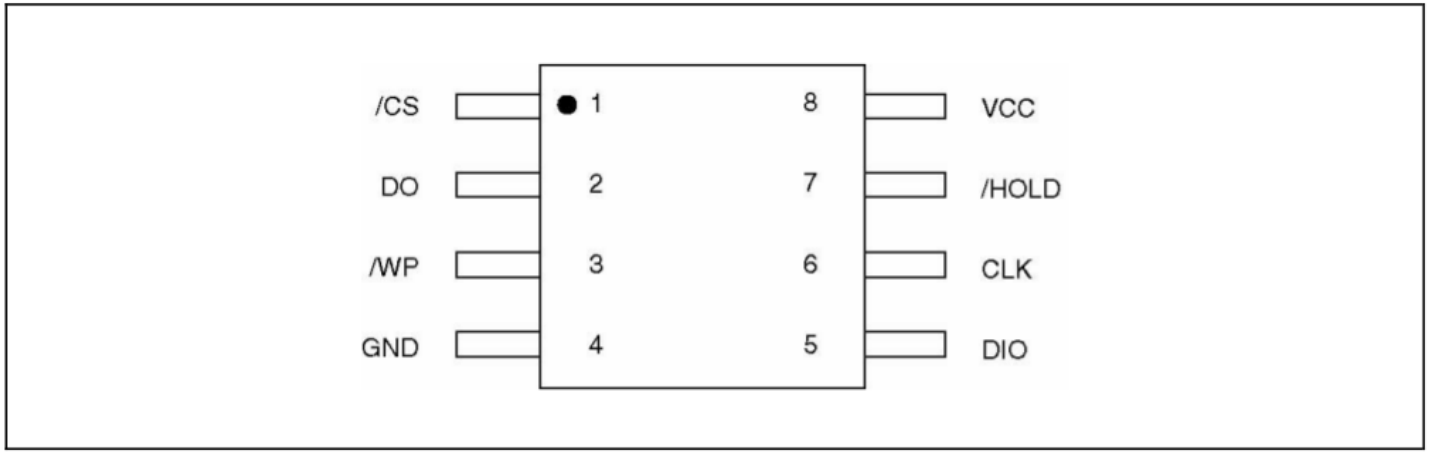


Figure 1b。W25X40 and W25X80 Pin Assignments, 8-pin SOIC (Package Code SS)

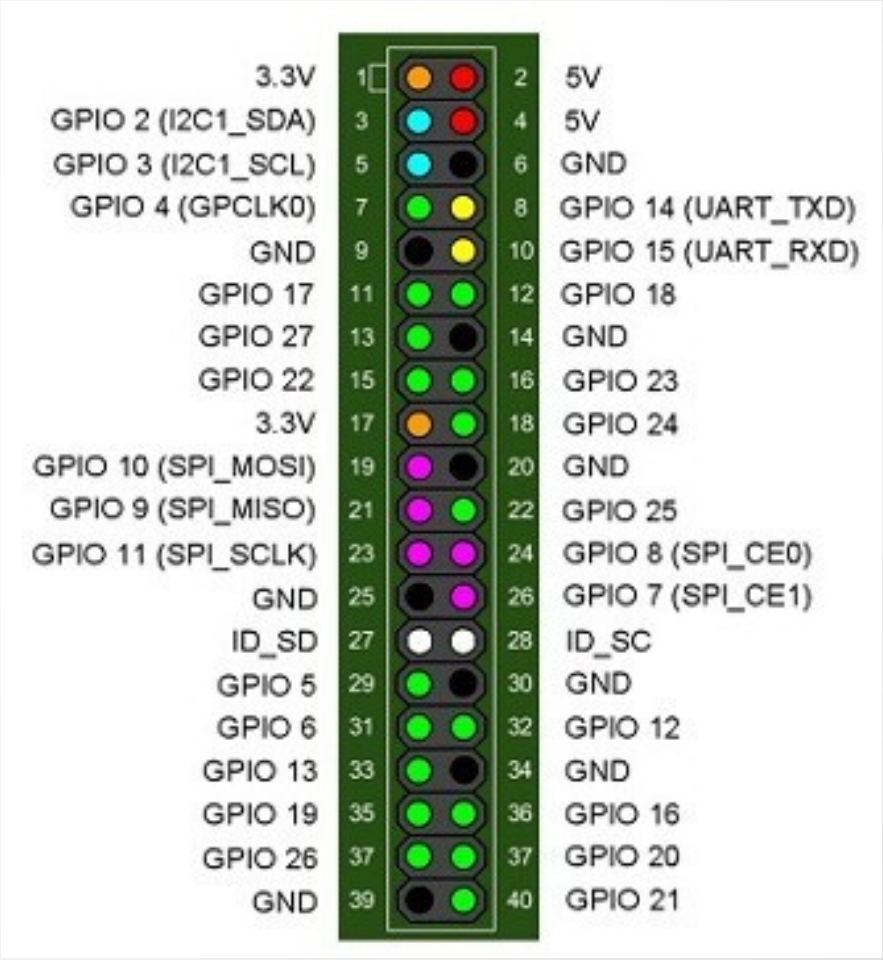
Understanding the datasheet

- The pinout has a dot next to pin 1. If you look at the physical chip you will see a dimple next to a pin. That's pin 1.
- The pins may be labeled differently. Some manufacturers call pin 2 SO and pin 5 SI or SIO. CLK may be called SCLK and WP may be ACC.
- If you read the datasheet there should be a description of what each pin functions as.
- CS is Chip Select, DO is Data Output, WP is Write Protect, GND is ground, DIO is Data Input, CLK is Clock, HOLD is Hold, VCC is the power pin. If the labeling is different just match function for function.

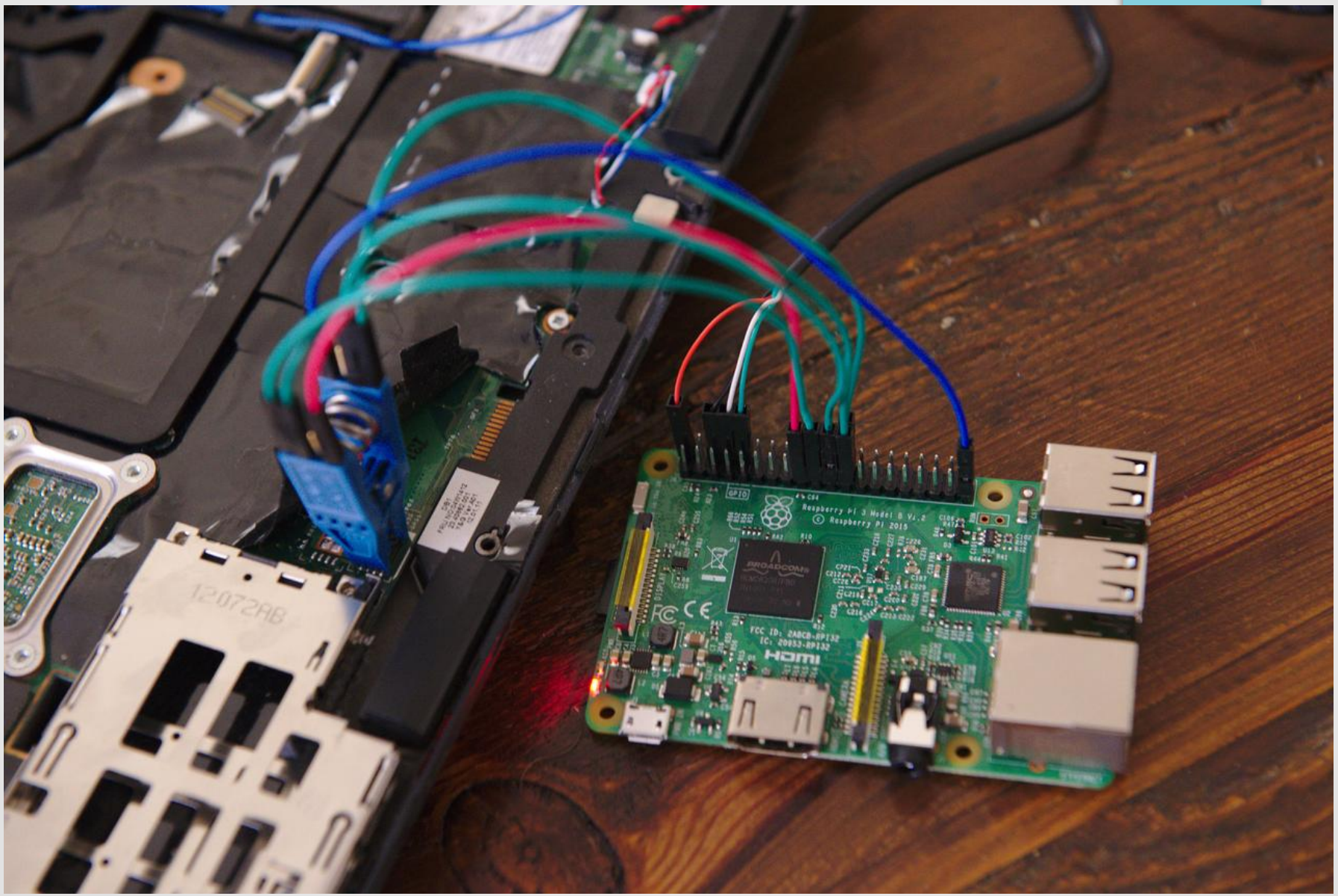
Connecting the RPi to the BIOS Chip

- RPi — BIOS
- Pin 24 CE0 — pin 1 CS
- Pin 21 MISO — pin 2 DO
- Pin 25 GNF — pin 4 GND
- Pin 19 MOSI — pin 5 DIO
- Pin 23 SCLK — pin 6 CLK
- — pin 8 VCC*
- Pin 17 3.3V — pin 7 HOLD*
- — and pin 3 WP*

*(You will need to splice 3 wires onto the one or use a breadboard to bridge.)

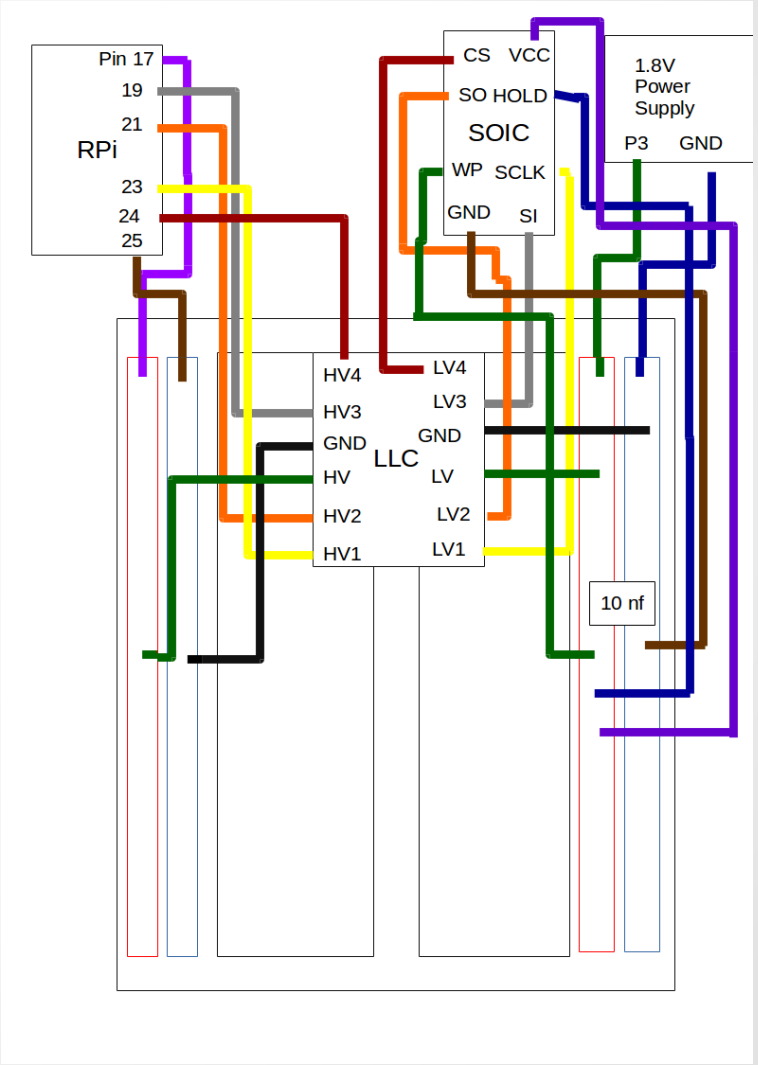


Rpi Connected to BIOS Chip

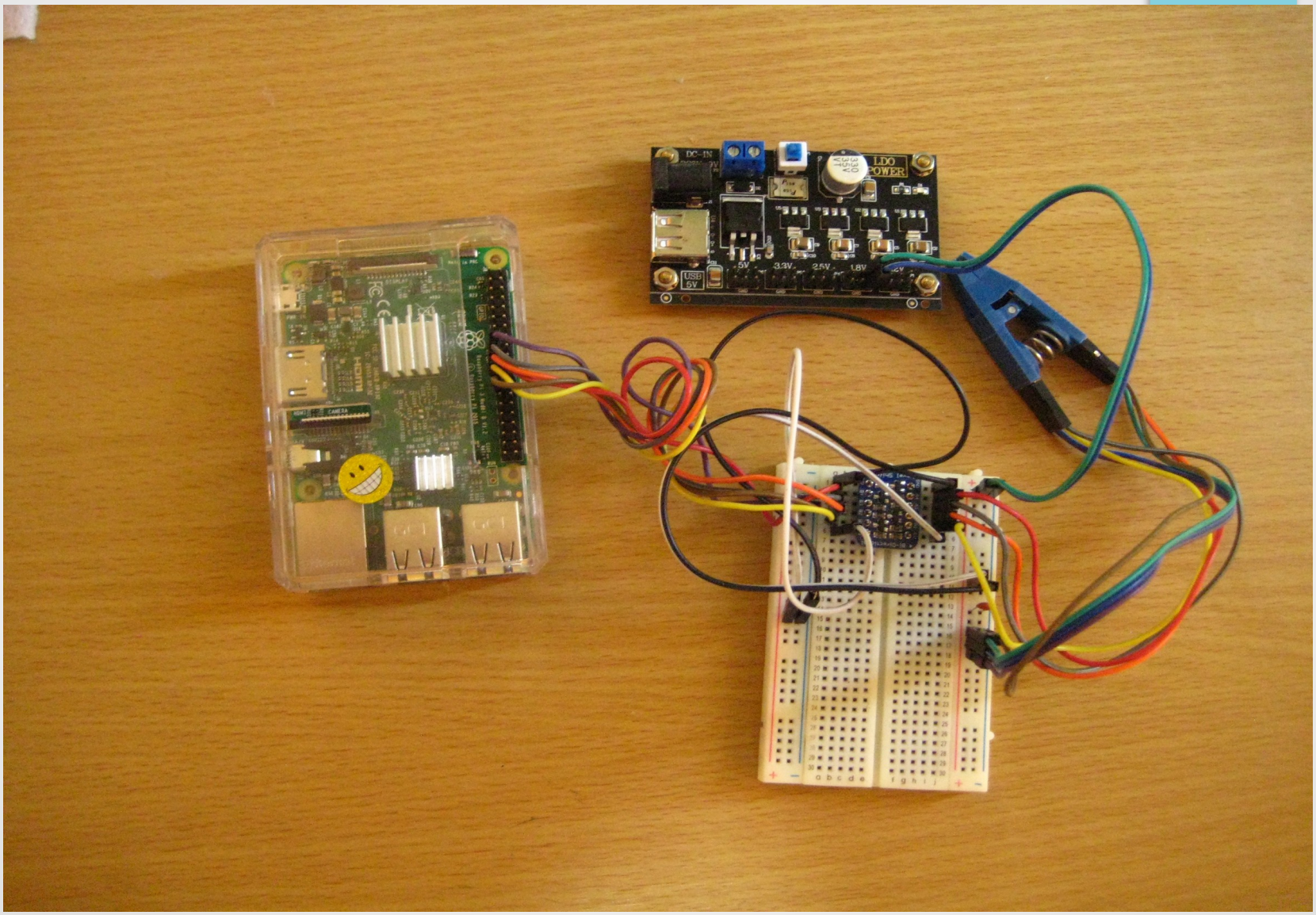


Building a Circuit to Step Down Voltage

- If you have a 1.8V BIOS chip you will need to step down the voltage or else fry your BIOS.
- In this diagram the Raspberry Pi is on the upper left, the Logic Level Converter is mounted on a breadboard in the lower center, and the SOIC-8 clip and power supply are in the upper right.
- The 10nf capacitor should be mounted on the breadboard power rails between the power supply output and the SOIC power pins (1,3,4, and 8)
- This removes fluctuations in the voltage that can otherwise cause noise in the data signal.

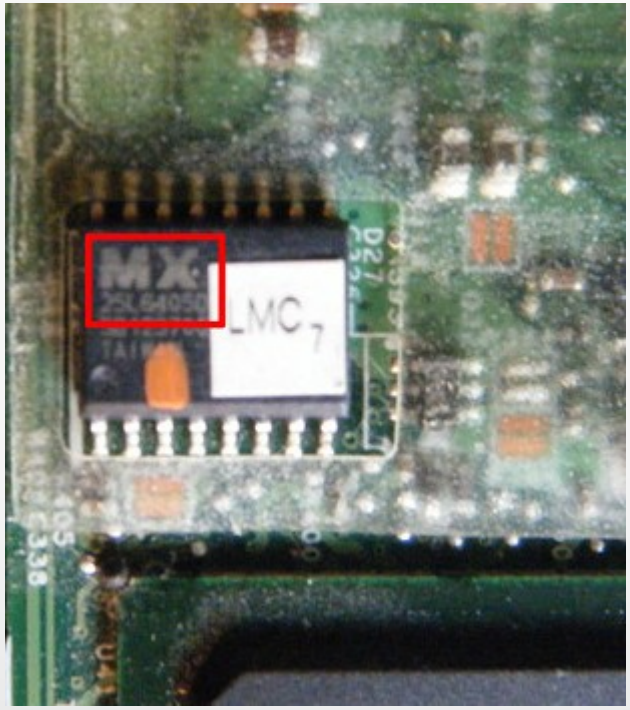


Step Down Circuit



If Your BIOS Chip has 16 pins

- If you have a BIOS chip with 16 pins, just use the datasheet to choose the correct pins for CS, VCC, GND, HOLD, CLK, WP, DO, and DI and connect them to the appropriate GPIO pins on the Raspberry Pi. The rest most likely should be connected to VCC, but may be left floating or connected to GND. Check the docs to be sure.



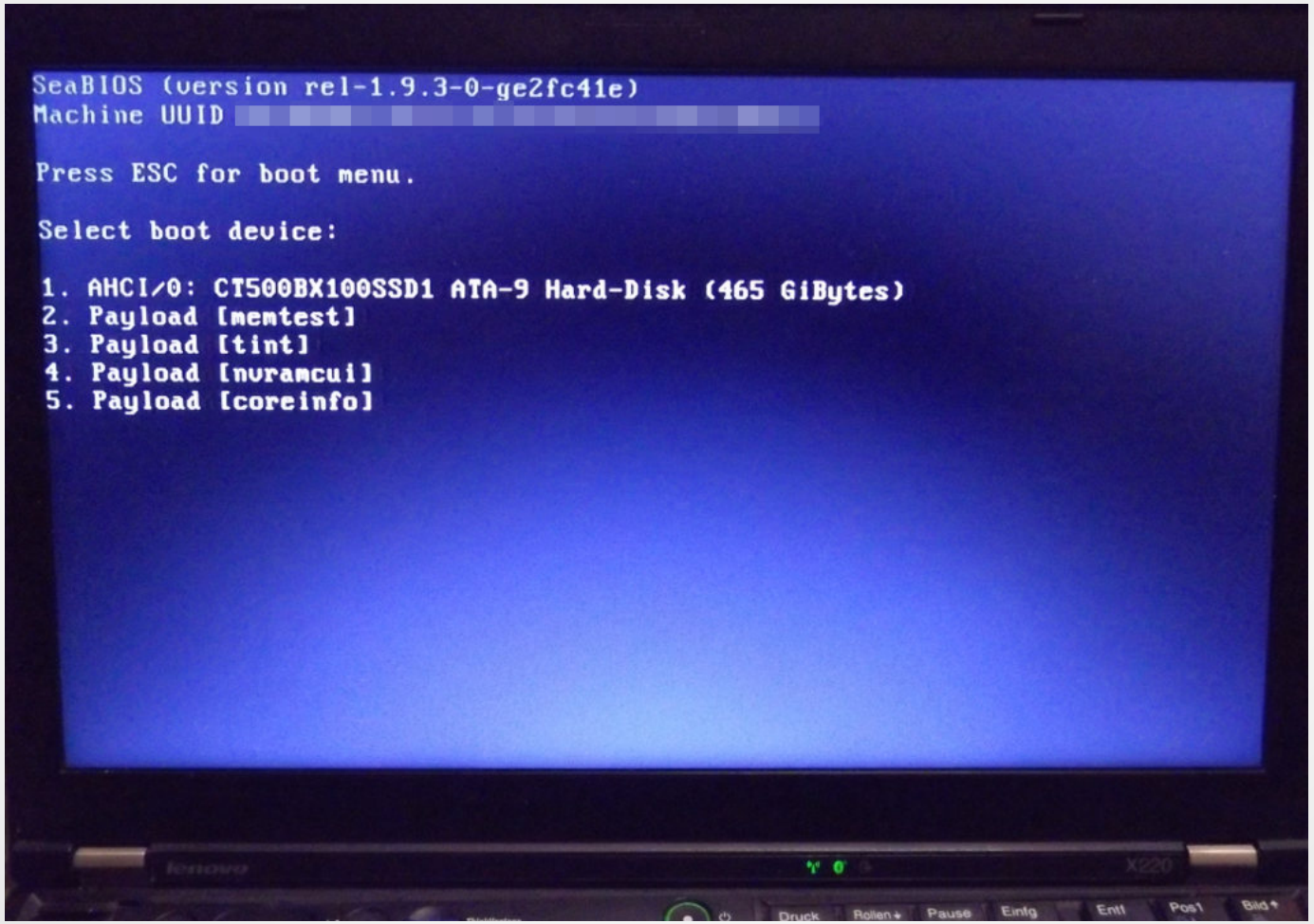
Testing the Connection

- Before you physically connect the SOIC clip to the BIOS chip, make sure to disconnect the AC adapter and battery from the laptop. The Pi should be the only source of power. (unless you are using the step down circuit)
- Once its connected start by reading the existing BIOS
  - flashrom -p linux_spi:dev=/dev/spidev0.0,spispeed=1000 -r flashtest1.bin
- Do this 3 times and then md5sum each output file to compare. If the hash comes back the same for all 3 you have a good connection, if not remove and replace the SOIC clip and make sure all the wires are firmly seated.

Writing the Coreboot ROM to BIOS

- Once you have tested the connection and are getting reliable reads now you are ready to write.
  - flashrom -p linux_spi:dev=/dev/spidev0.0,spispeed=1000 -w coreboot.rom
- If there were no errors then you should see the SeaBIOS (or TianoCore) screen when you plug your laptop back in and boot it up.

Success!


SeaBIOS (version rel-1.9.3-0-ge2fc41e)
Machine UUID
Press ESC for boot menu.
Select boot device:
1. AHCI/0: CT500BX100SSD1 ATA-9 Hard-Disk (465 GiBytes)
2. Payload [memtest]
3. Payload [tint]
4. Payload [nvramcui]
5. Payload [coreinfo]

What Happens if it Doesn't Work

- Just because the screen is blank doesn’t mean it’s not working. There could be a problem with the VGA BIOS. Wait a bit and see if the Linux load screen comes up. Coreboot also comes with a serial output capability for troubleshooting so you can see what the error is.
- Try reflashing. If that doesn’t work you can just flash the original BIOS back.
- Check all the settings in the coreboot config file.
- Check all the wiring connections with a multimeter.
- If flashrom can’t detect your chip you can specify manually with the -m flag, though this usually indicates a poor electrical connection.
- Flashrom developers can be reached for support questions on the #flashrom channel on freenode.net
- The coreboot support mailing list is at
  - http://www.coreboot.org/mailman/listinfo/coreboot

Resources

- http://io.netgarage.org/me/
- https://www.ssh.com/vulnerability/intel-amt/
- https://www.cs.cmu.edu/~davide/bad_thing.html
- https://hardenedlinux.github.io/firmware/2016/11/17/neutralize_ME_firmware_on_sandybridge_and_ivybridge.html
- https://puri.sm/posts/purism-librem-laptops-completely-disable-intel-management-engine/
- https://blog.system76.com/post/168050597573/system76-me-firmware-updates-plan
- https://media.ccc.de/v/34c3-8782-intel_me_myths_and_reality
- http://blog.ptsecurity.com/2017/08/disabling-intel-me.html
- https://github.com/corna/me_cleaner/
- https://libreboot.org/docs/install/rpi_setup.html
- https://mrchromebox.tech/
- https://tylercipriani.com/blog/2016/11/13/coreboot-on-the-thinkpad-x220-with-a-raspberry-pi/

Contact Info

If you have comments, feedback or want to work together please contact me at:

research@ecrsecurity.com

If you want to hire me.

ECR Security

http://www.ecrsecurity.com