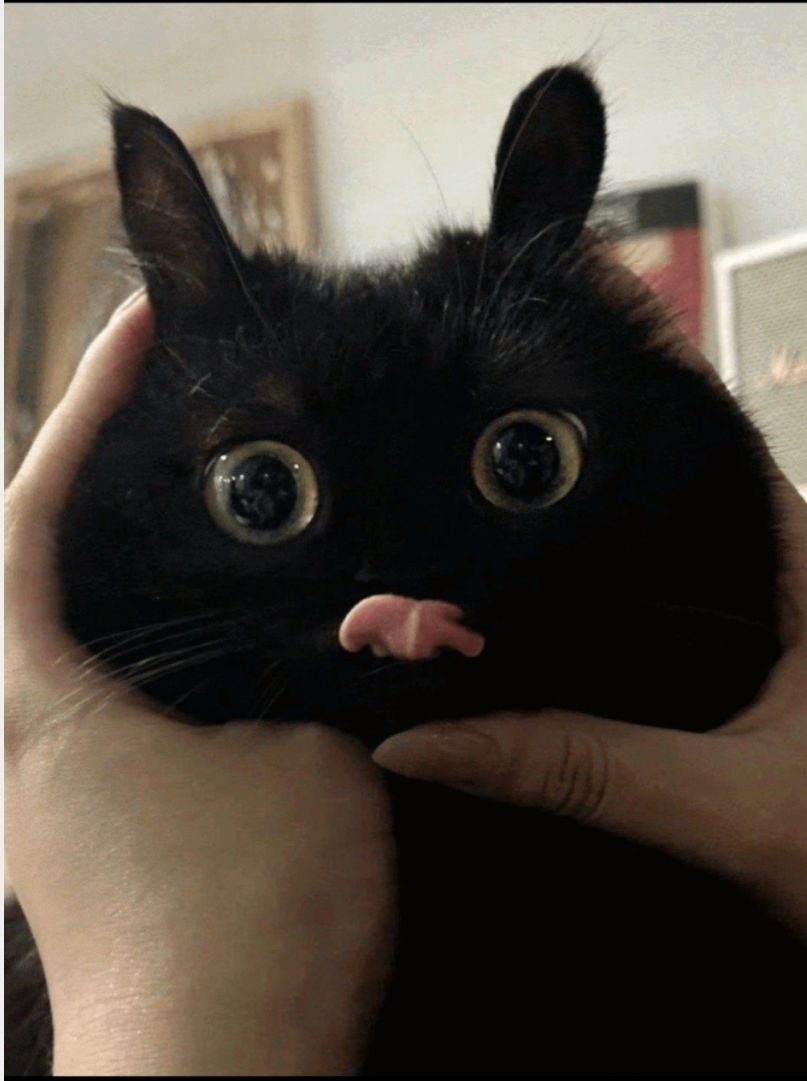
Extracting Secrets from IoT Devices

By Isaiah Davis-Stober (@net_code)

What I’m going to cover

- What is firmware?
- The IoT epidemic
- How to acquire firmware samples
- Dynamic and static analysis



Who am I?

Experience:

I’ve been tearing apart (and sometimes putting back together) electronics since I was instantiated, and tearing apart IoT devices for the last couple of years

Things I like:

- Misusing computers
- Cats
- Terrible bash scripts
- Lock picking/bypassing
- Tearing apart whitelabel IoT devices

What is “IoT”?

It’s the Internet of Things babyyyyy

What is Firmware?

- Lowest level of software
- Interacts directly with the hardware

Familiar operating systems

Key differences between embedded Linux and other operating systems I hope you are familiar with

Why embedded is so different

- Usually a different chip architecture
- Often relies on hardware to be connected to work correctly

Why firmware so interesting

- Lowest level means if you control it you are god
- It is often neglected/ignored

Why is it a fun target?

- Low level is fun
- I don’t have to fight an AV
- No one configures things correctly

IoT epidemic and why it’s stupid

- Every product is the minimum viable product
- Toasters were not meant to have touch screens

Quick rant about “smart” devices

Privacy concerns

I just want it to work, why does it need an account?

Firmware acquisition

- Download
- Extraction

Downloading firmware

- Manufacture website
- Open S3 buckets
- Some very nice person who is distributing a copy in a legally questionable way
- FTP servers

For example: ftp://ftp2.dlink.com/PRODUCTS/DIR-882/REVA/

Security to look out for when getting firmware

- Encryption
- Bootloader security

Bootloader

Common bootloaders

- U-Boot
- Barebox
- RedBoot
- RT-Thread

Extracting firmware

- UART
- JTAG
- Flash extraction (chip on and chip off)

UART (Universal Asynchronous Receiver/Transmitter)

RX -> TX

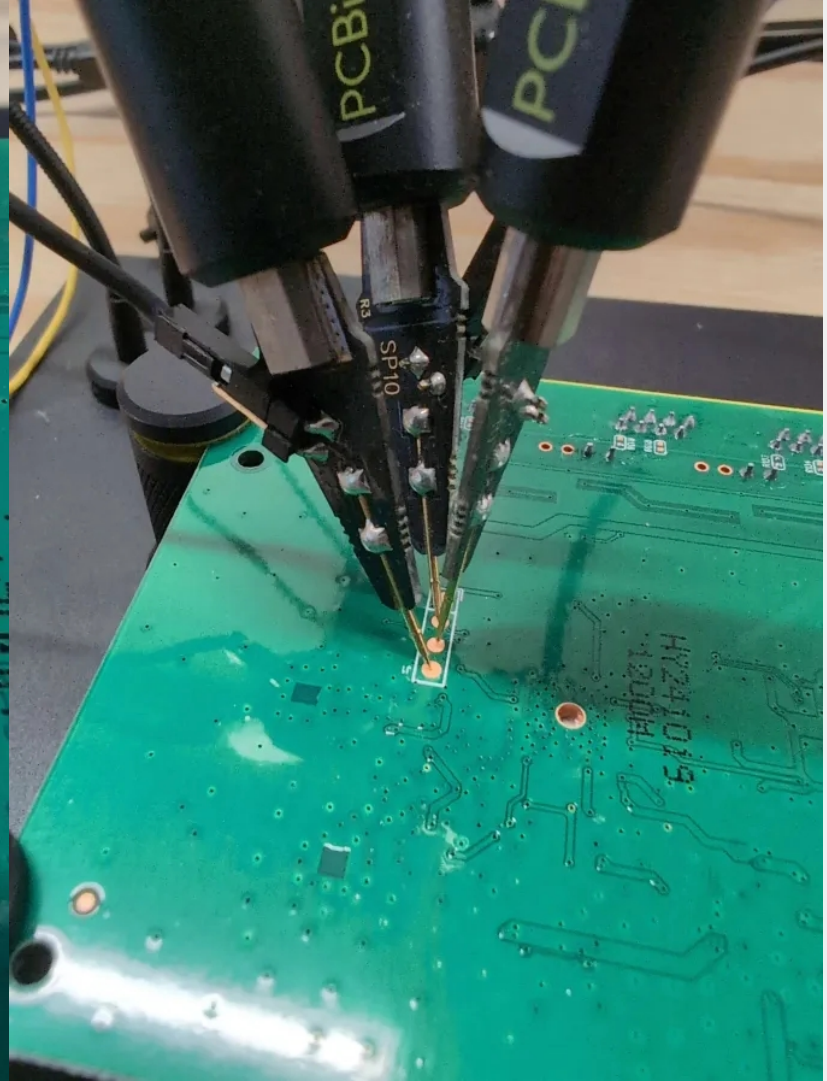
TX - > RX

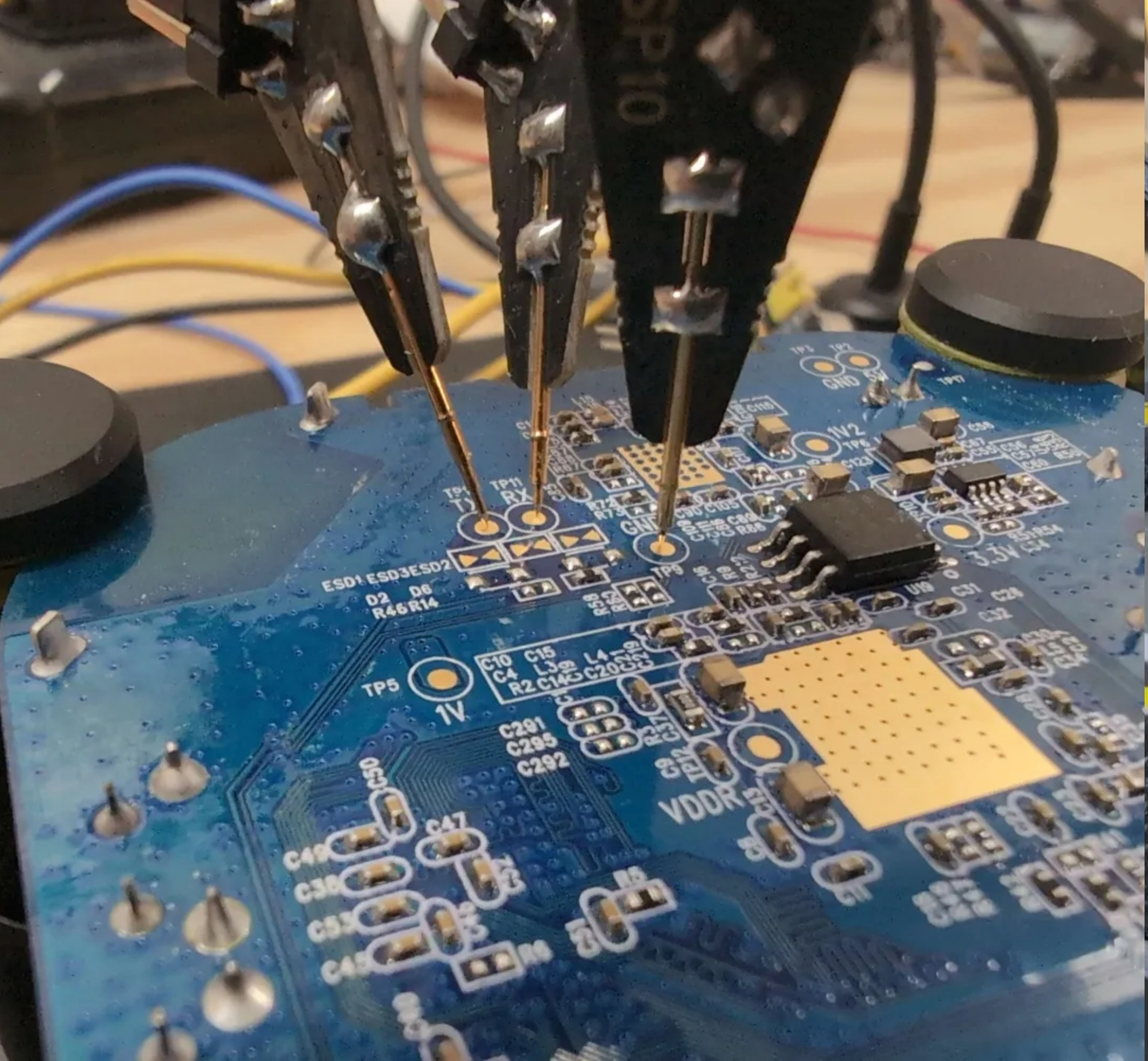
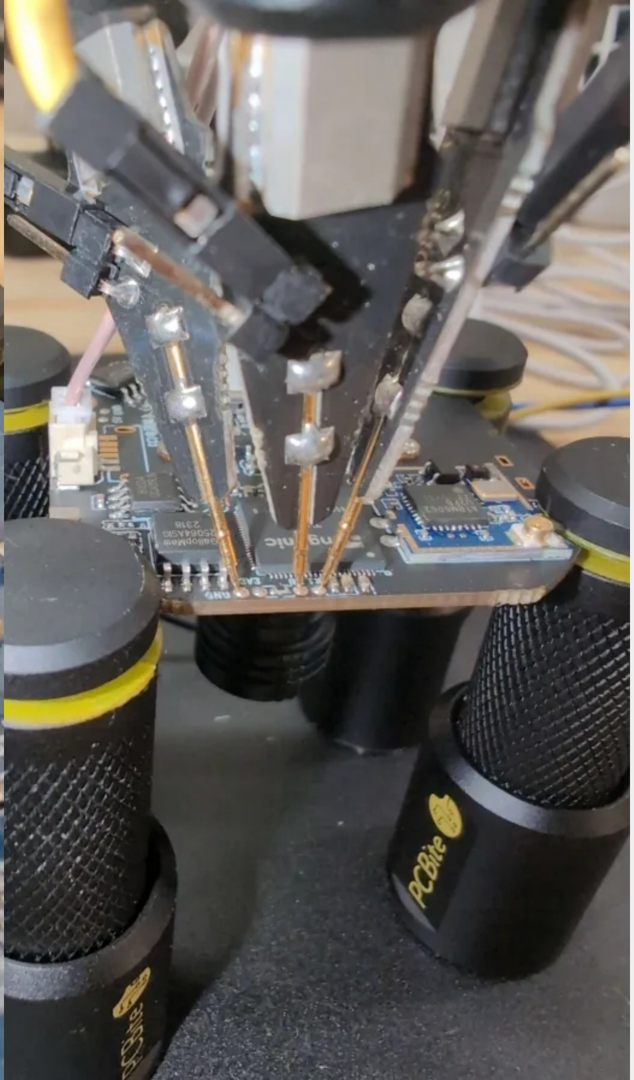
GND -> GND

Common baud rates:

- 115200
- 57600
- 38400
- 9600


J1


SP10
R3


TP11
RX
GND 5V
1V2
3.3V
ESD1 ESD3 ESD2
TP5
1V
VDDR

PCBite

JTAG (Joint Test Action Group)

- Debugging interface similar to UART
- More in depth

Checking boot log for information

```
---RealTek(RTL8196E)at 2015.01.06-18:13-0800 v1.6 [16bit](400MHz)
bootbank is 1, bankmark FFFFFFF0
check_image_header  return_addr:05010000 bank_offset:00000000
no sys signature at 00010000!
no sys signature at 00020000!
no rootfs signature at 000E0000!
no rootfs signature at 000F0000!
Jump to image start=0x80500000...
return_addr = 05030000 ,boot bank=1, bank_mark=0xfffffff0...
decompressing kernel:
Uncompressing Linux... done, booting the kernel.
done decompressing kernel.
start address: 0x80003480
CPU revision is: 0000cd01
Determined physical RAM map:
 memory: 02000000 @ 00000000 (usable)
Zone PFN ranges:
  Normal   0x00000000 -> 0x00002000
Movable zone start PFN for each node
early_node_map[1] active PFN ranges
    0: 0x00000000 -> 0x00002000
Built 1 zonelists in Zone order, mobility grouping on.  Total pages: 8128
Kernel command line: console=ttyS0,38400 root=/dev/mtdblock1
icache: 16kB/16B, dcache: 8kB/16B, scache: 0kB/0B
NR_IRQS:48
PID hash table entries: 128 (order: 7, 512 bytes)
console handover: boot [early0] -> real [ttyS0]
Dentry cache hash table entries: 4096 (order: 2, 16384 bytes)
Inode-cache hash table entries: 2048 (order: 1, 8192 bytes)
Memory: 26420k/32768k available (2566k kernel code, 6348k reserved, 459k data, 108k init, 0k highmem)
Calibrating delay loop... 398.95 BogoMIPS (lpj=1994752)
Mount-cache hash table entries: 512
net_namespace: 524 bytes
NET: Registered protocol family 16
bio: create slab <bio-0> at 0
```

Flash dumping

- Types of flash
- Tools
- Methods

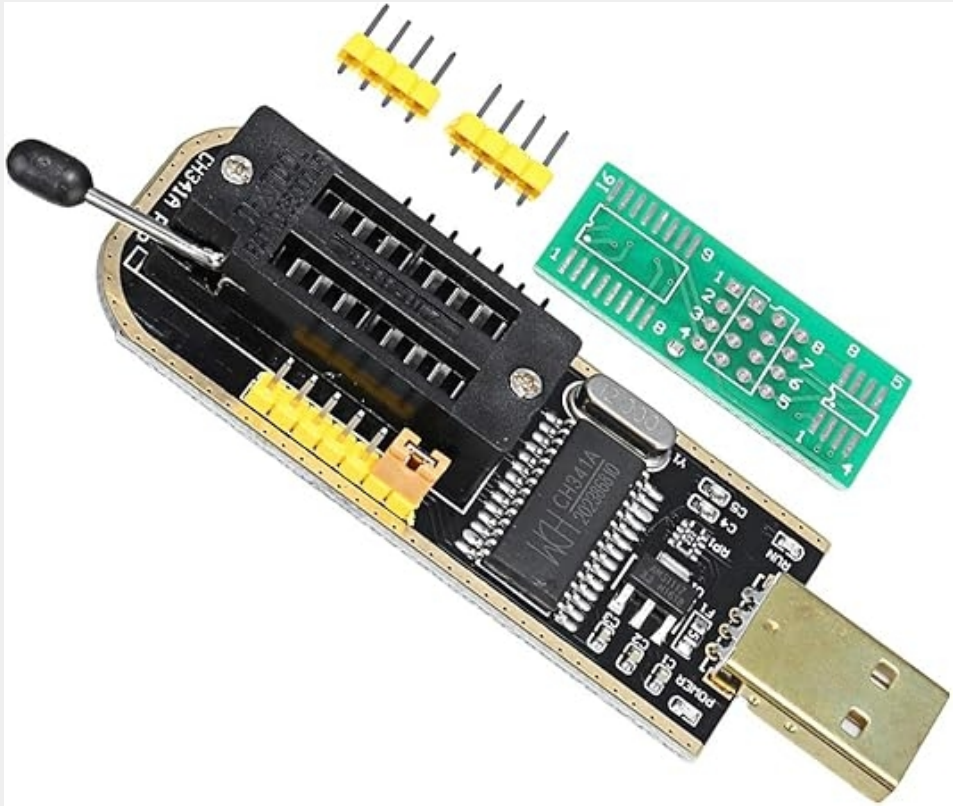
Types of flash interfaces

- SPI (Serial Peripheral Interface)
- QSPI (Quad Serial Peripheral Interface)
- I2C (Inter-Integrated Circuit)

Flashdump

- Easiest tool for using the CH341a programmer which is $14 on Amazon
- Works on just about every chip I’ve tried*

*dumping, it cannot always write them



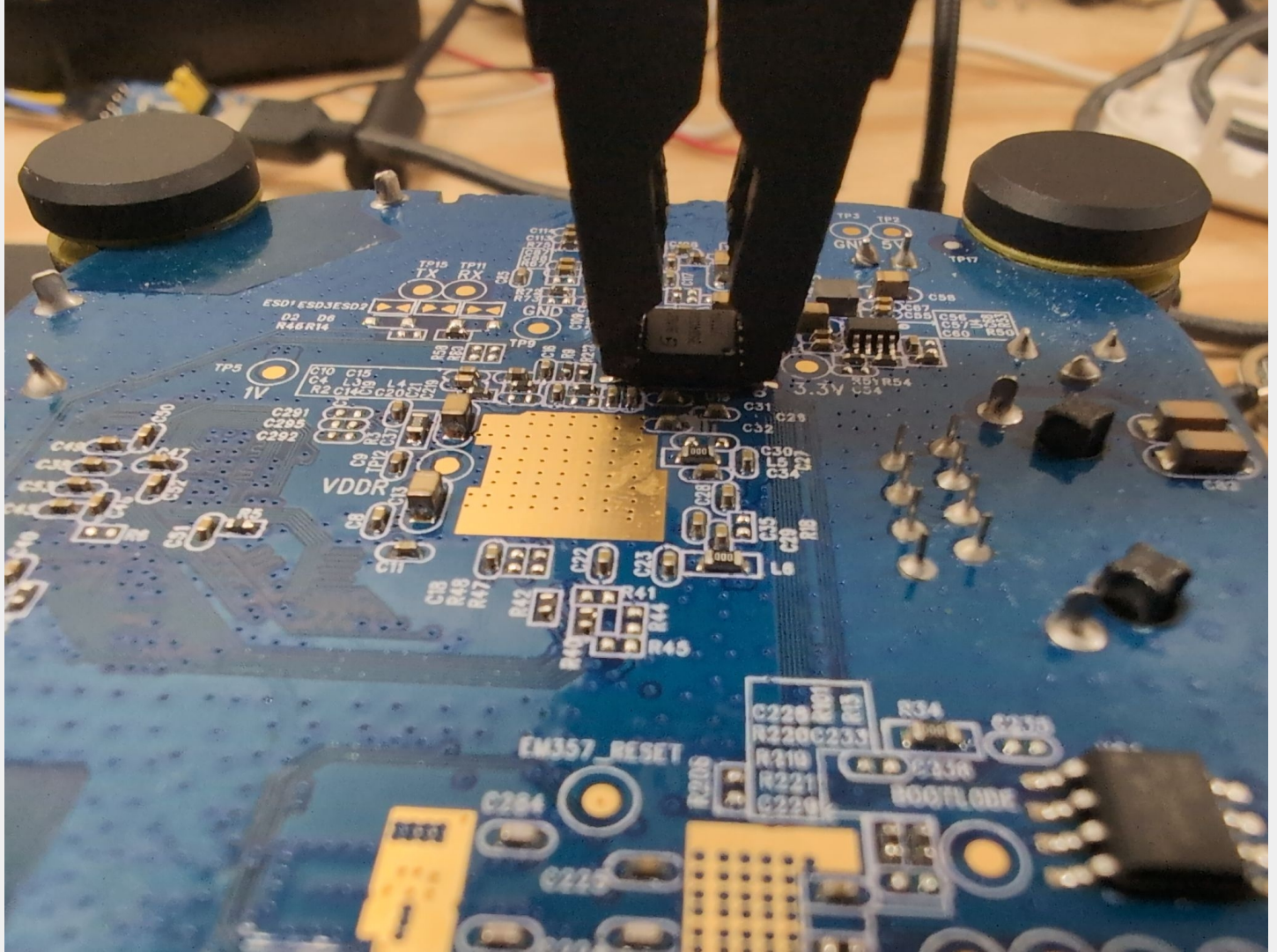
Methods

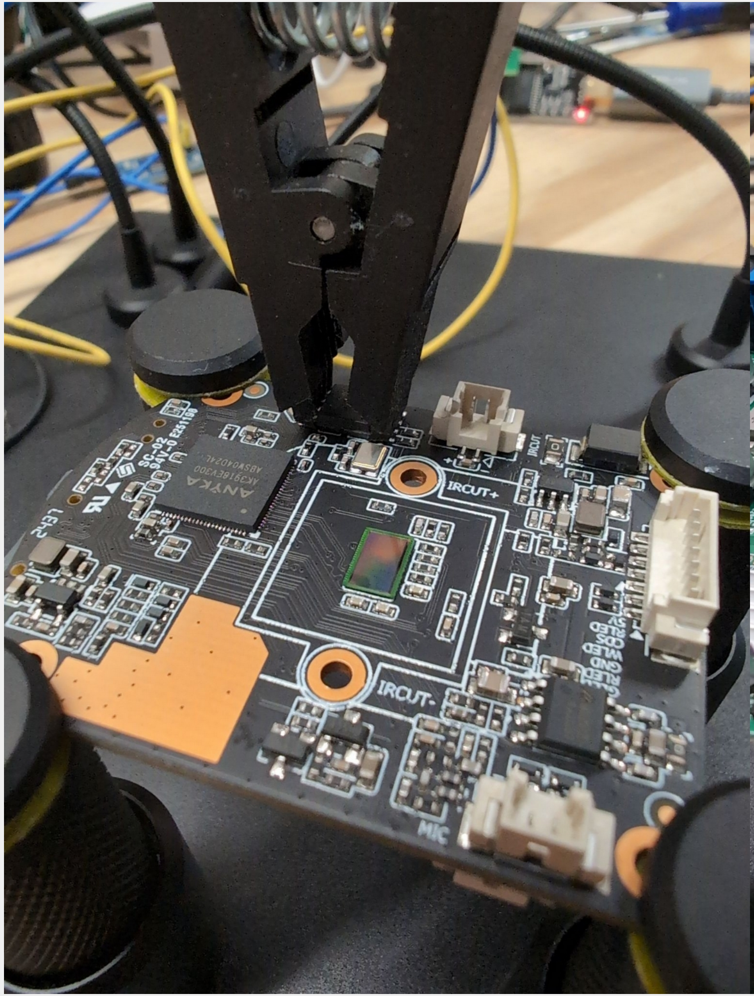
- Chip on
- Chip off

Chip on flash dumping

Dumping the firmware straight off of the flash module, without taking it off the board

- SOP8 clip
- CH341a and flashdump


TP15
TX
TP11
RX
TP3
GND
TP2
5V
TP17
GND
TP9
TP5
1V
3.3V
VDDR
EM357_RESET
BOOTLOADR

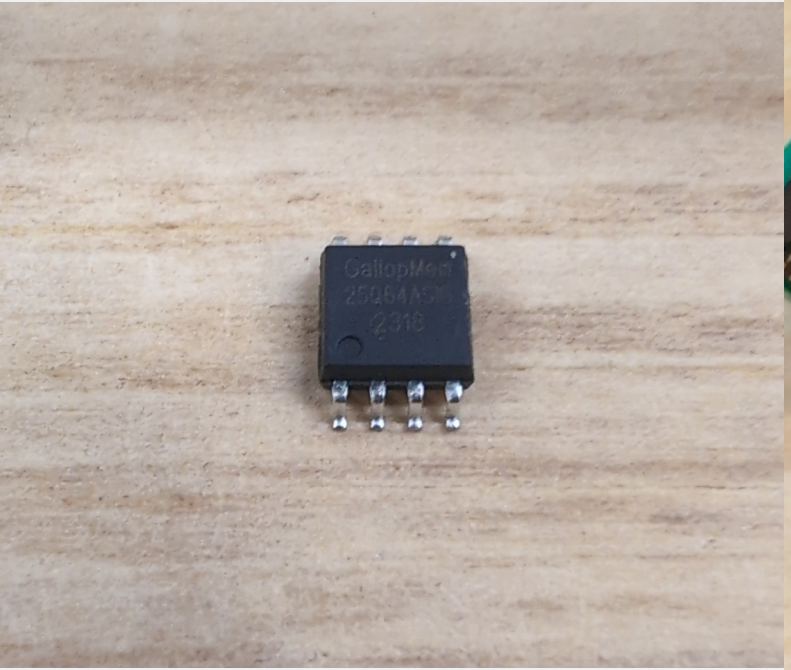

ANYKA
IRCUT+
IRCUT-
MIC

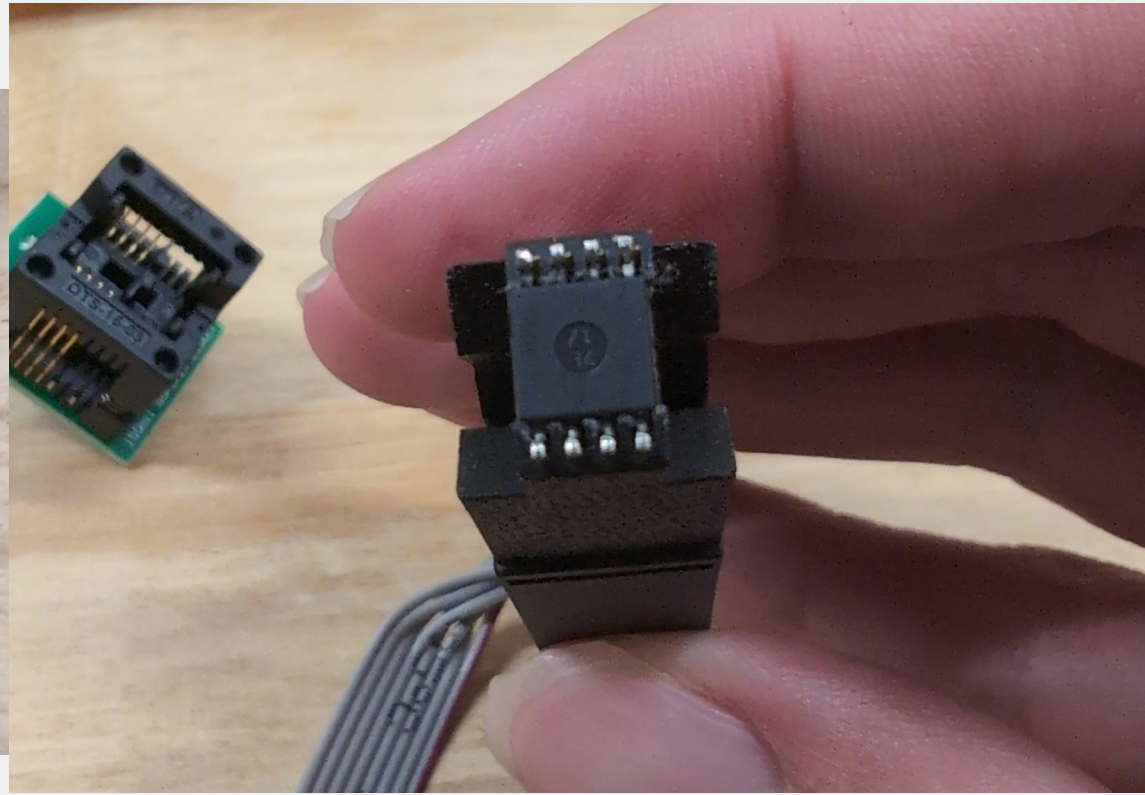

Chip off flash dumping

- Dumping the firmware straight off of the flash module, but this time by removing the flash module
- Flashdump

- Xgecu for more complicated chips


ATBM6062
U2FB
CE19Y02
T23
GallopMem
25Q64ASIG
2318
1F7K
HY2407230A


GallopMem
25064A
2316



Using Flashdump

- Amazing for SPI flash

XGecu

- Amazing for everything else

Main use

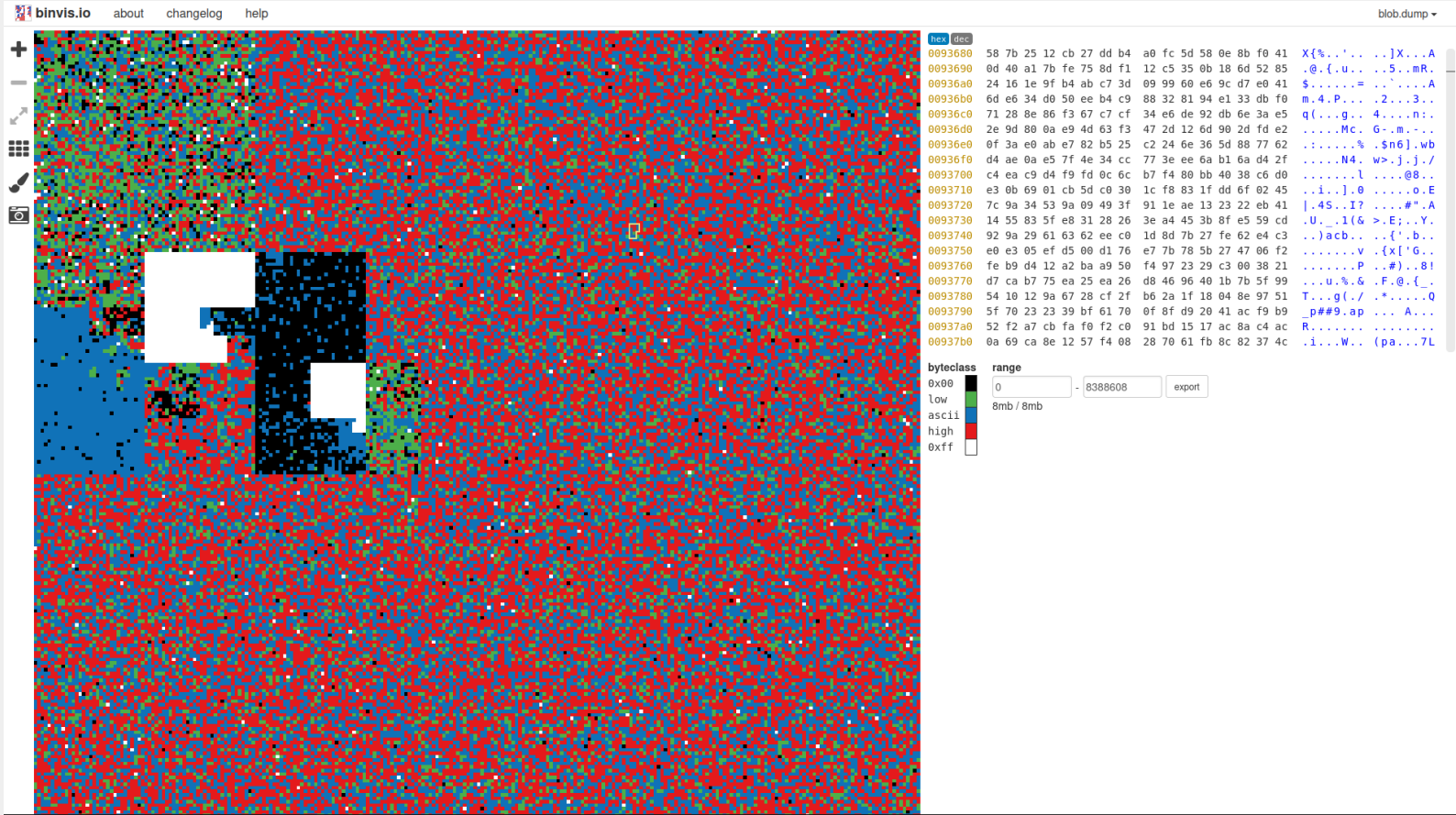
Analyzing firmware

- Static vs dynamic analysis
- Common embedded systems filesystems

Checking if the dump is encrypted

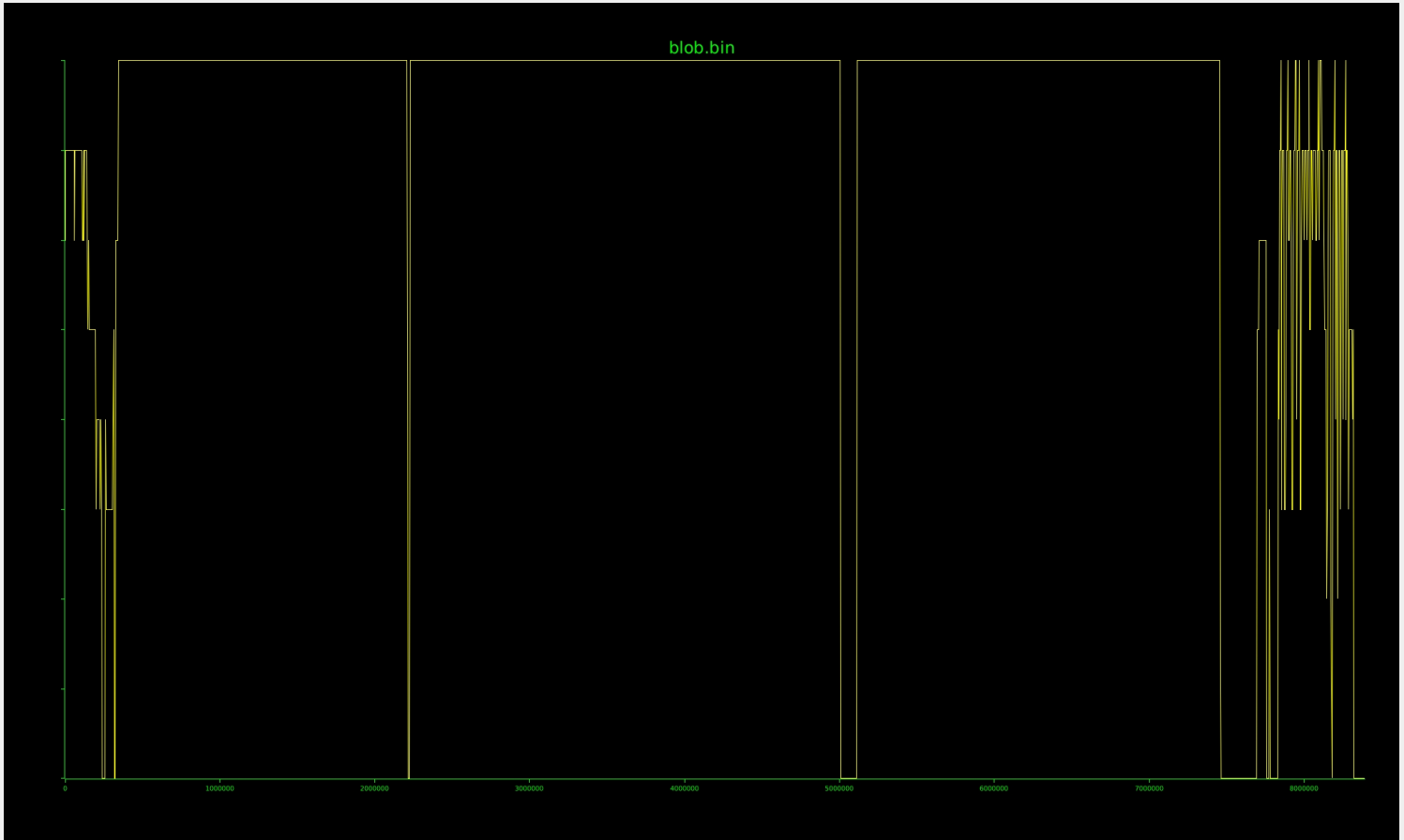
- Check entropy with binwalk
- Checking what different parts of the file look like with binvis.io

binvis.io


binvis.io
about
changelog
help
blob.dump
hex
dec
0093680 58 7b 25 12 cb 27 dd b4 a0 fc 5d 58 0e 8b f0 41 X{%..'.. ..]X...A
0093690 0d 40 a1 7b fe 75 8d f1 12 c5 35 0b 18 6d 52 85 .@.{.u.. ..5..mR.
00936a0 24 16 1e 9f b4 ab c7 3d 09 99 60 e6 9c d7 e0 41 $.....= ..`....A
00936b0 6d e6 34 d0 50 ee b4 c9 88 32 81 94 e1 33 db f0 m.4.P... .2...3..
00936c0 71 28 8e 86 f3 67 c7 cf 34 e6 de 92 db 6e 3a e5 q(...g.. 4....n:.
00936d0 2e 9d 80 0a e9 4d 63 f3 47 2d 12 6d 90 2d fd e2 .....Mc. G-.m.-..
00936e0 0f 3a e0 ab e7 82 b5 25 c2 24 6e 36 5d 88 77 62 .:.....% .$n6].wb
00936f0 d4 ae 0a e5 7f 4e 34 cc 77 3e ee 6a b1 6a d4 2f .....N4. w>.j.j./
0093700 c4 ea c9 d4 f9 fd 0c 6c b7 f4 80 bb 40 38 c6 d0 ....l ....@8..
0093710 e3 0b 69 01 cb 5d c0 30 1c f8 83 1f dd 6f 02 45 ..i..].0 .....o.E
0093720 7c 9a 34 53 9a 09 49 3f 91 1e ae 13 23 22 eb 41 |.4S..I? ....#".A
0093730 14 55 83 5f e8 31 28 26 3e a4 45 3b 8f e5 59 cd .U._.1(& >.E;..Y.
0093740 92 9a 29 61 63 62 ee c0 1d 8d 7b 27 fe 62 e4 c3 ..)acb.. ..{'.b..
0093750 e0 e3 05 ef d5 00 d1 76 e7 7b 78 5b 27 47 06 f2 ....v .{x['G..
0093760 fe b9 d4 12 a2 ba a9 50 f4 97 23 29 c3 00 38 21 ....P ..#)..8!
0093770 d7 ca b7 75 ea 25 ea 26 d8 46 96 40 1b 7b 5f 99 ...u.%.& .F.@.{_.
0093780 54 10 12 9a 67 28 cf 2f b6 2a 1f 18 04 8e 97 51 T...g(./ .*.....Q
0093790 5f 70 23 23 39 bf 61 70 0f 8f d9 20 41 ac f9 b9 _p##9.ap ... A...
00937a0 52 f2 a7 cb fa f0 f2 c0 91 bd 15 17 ac 8a c4 ac R....... ....
00937b0 0a 69 ca 8e 12 57 f4 08 28 70 61 fb 8c 82 37 4c .i...W.. (pa...7L
byteclass
0x00
low
ascii
high
0xff
range
0
8388608
export
8mb / 8mb

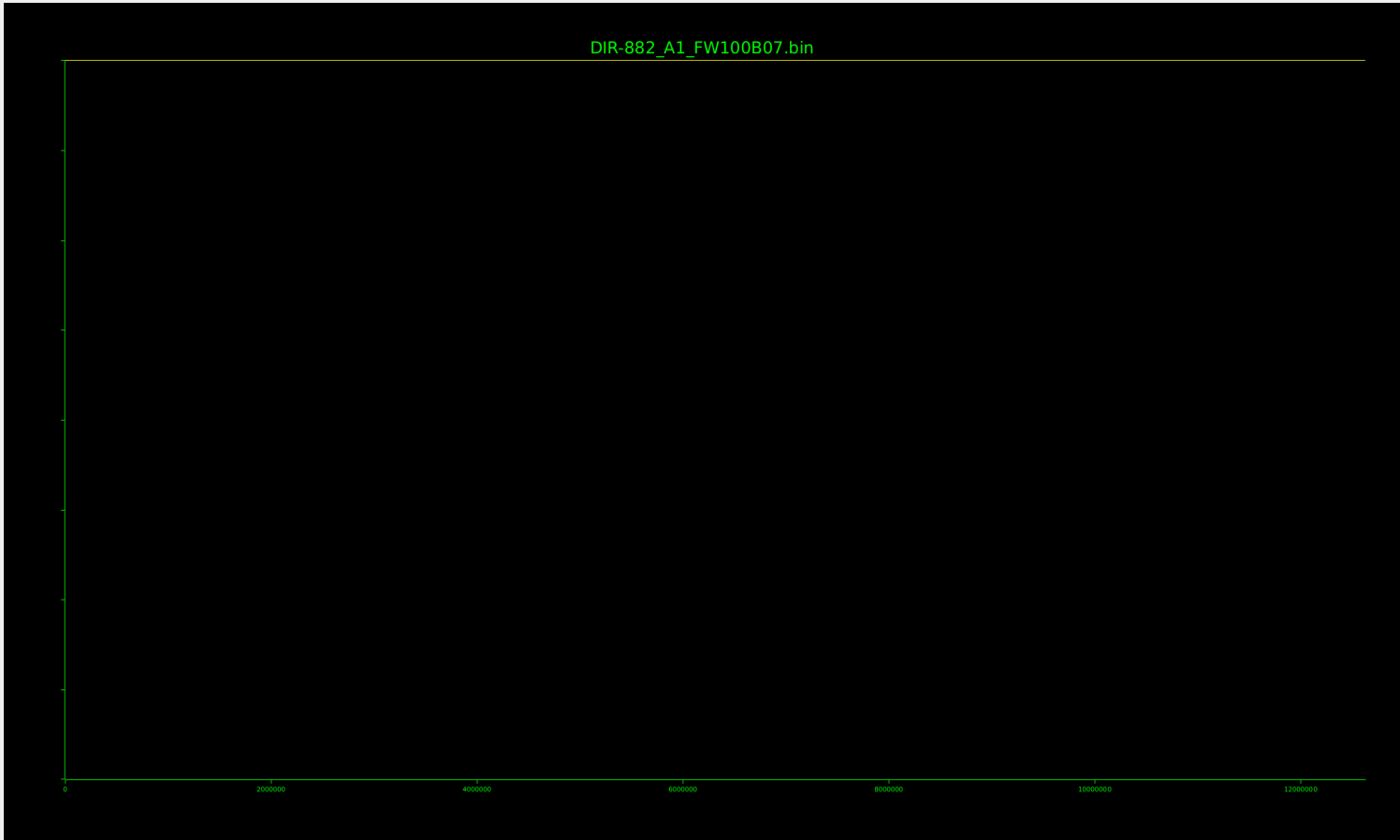
Binwalk -E

Good



Binwalk -E

Not good



Static analysis

- Binwalk
- Strings
- Ghidra
- Xxd

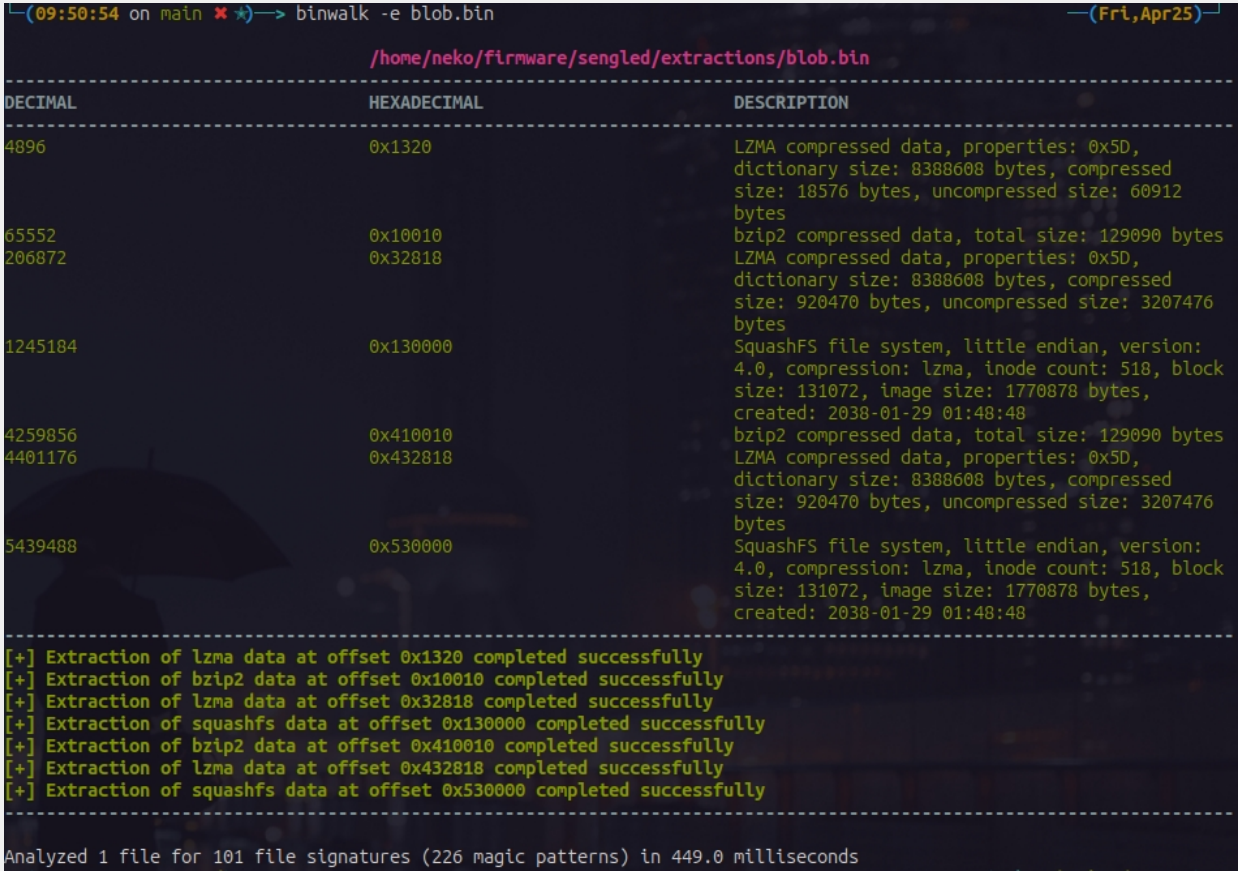
- FACT (Firmware Analysis and Comparison Toolkit)

Determining the type of system you’re working with

- CPU architecture
  - MIPS
  - ARM
- What the firmware might be based on
- Kernel version
- If BusyBox, what version

Extracting the filesystems and directory structure

- BinwalkV3 is awesome



Common File systems

Read only

- SquashFS

Writable

- JFFS2 (Journaling Flash File System v2)

Extracting those common filesystems

SquashFS

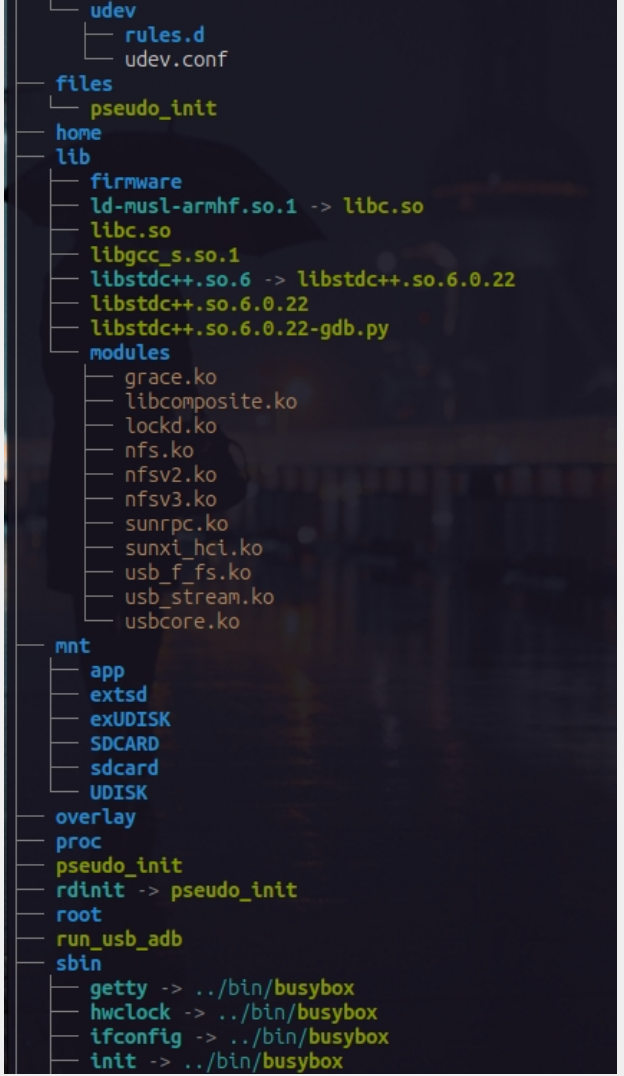
- UnsquashFS
- Sasquatch (patch for unsquashfs)

JFFS2

- Jefferson

Directory layout

- I like exa -T, and you can pry that from my cold dead hands



Determining how the filesystems and directories are mounted

- fstab
- inittab
- Whatever other init script they are using

File: **inittab.sh**

```
1  # /etc/inittab
2  #
3  # Copyright (C) 2001 Erik Andersen <andersen@codepoet.org>
4  #
5  # Note: BusyBox init doesn't support runlevels.  The runlevels field is
6  # completely ignored by BusyBox init. If you want runlevels, use
7  # sysvinit.
8  #
9  # Format for each entry: <id>:<runlevels>:<action>:<process>
10 #
11 # id        == tty to run on, or empty for /dev/console
12 # runlevels == ignored
13 # action    == one of sysinit, respawn, askfirst, wait, and once
14 # process   == program to run
15
16 # Startup the system
17 ::sysinit:/sbin/swapoff -a
18 # ::sysinit:/bin/mount -t tmpfs tmpfs /dev
19 ::sysinit:/bin/mkdir -p /dev/pts
20 ::sysinit:/bin/mkdir -p /dev/shm
21 ::sysinit:/bin/mount -a
22 ::sysinit:/bin/hostname -F /etc/hostname
23
24 # now run any rc scripts
25 ::sysinit:/etc/init.d/rcS
26
27 # Put a getty on the serial port
28 #console::respawn:-/bin/sh
29 console::respawn:/sbin/getty -L console 115200 vt100 # GENERIC_SERIAL
30
31 # Stuff to do for the 3-finger salute
32 #::ctrlaltdel:/sbin/reboot
33
34 # Stuff to do before rebooting
35 ::shutdown:/bin/umount -a -r
```

Locating particular files

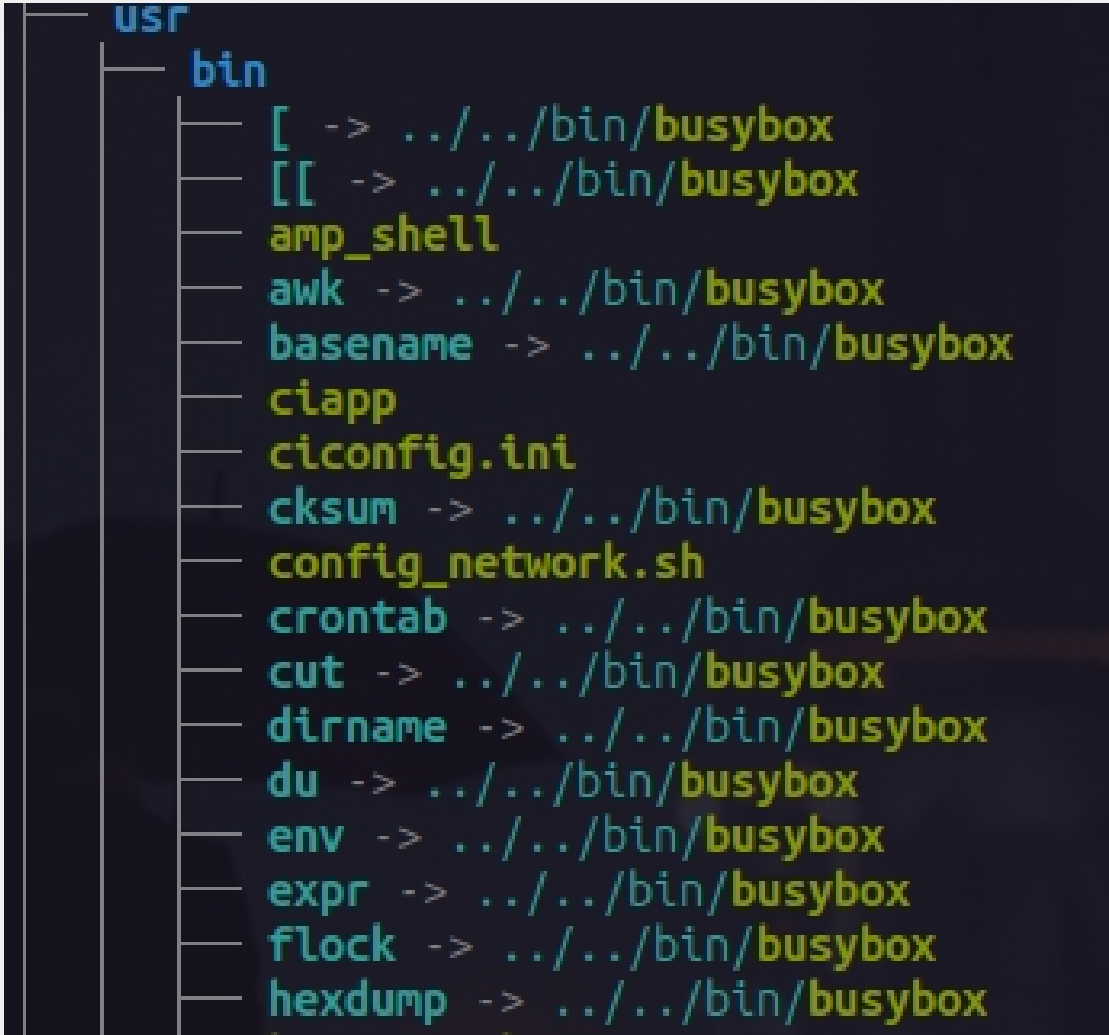
- grep
- find
- rg (ripgrep)
- Exa -T
- Guessing

BusyBox

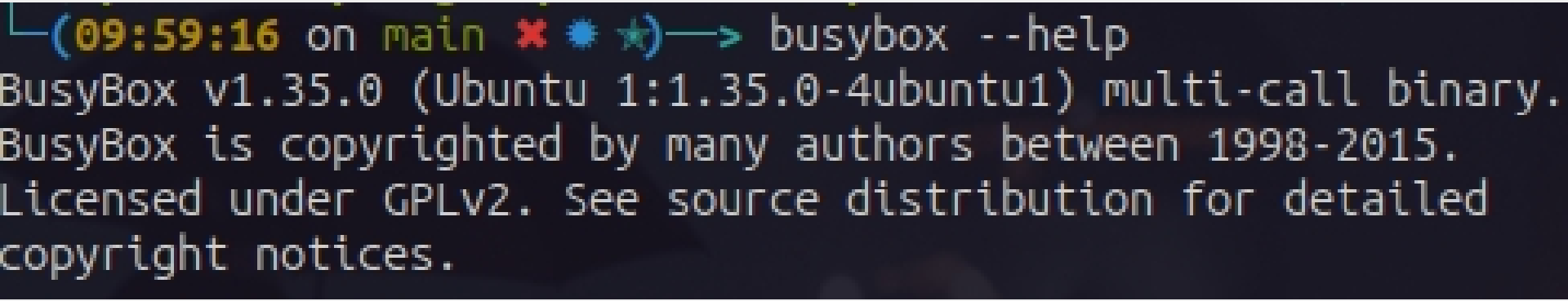
- A useful utility for small embedded Linux machines
- Single utility
- Does all of the things
- Swiss Army Knife of Embedded Linux

Just symlink

- It’s that easy

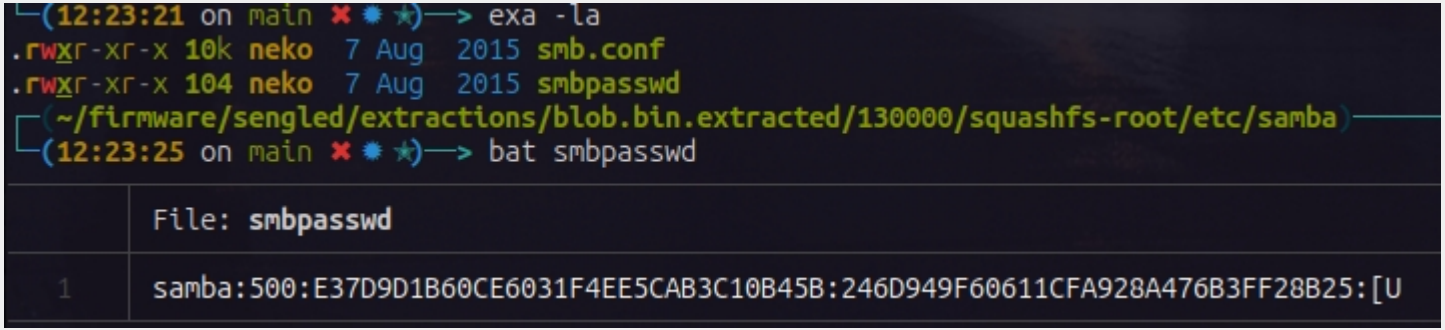


BusyBox –help



Looking for configuration files

- .conf
- .ini



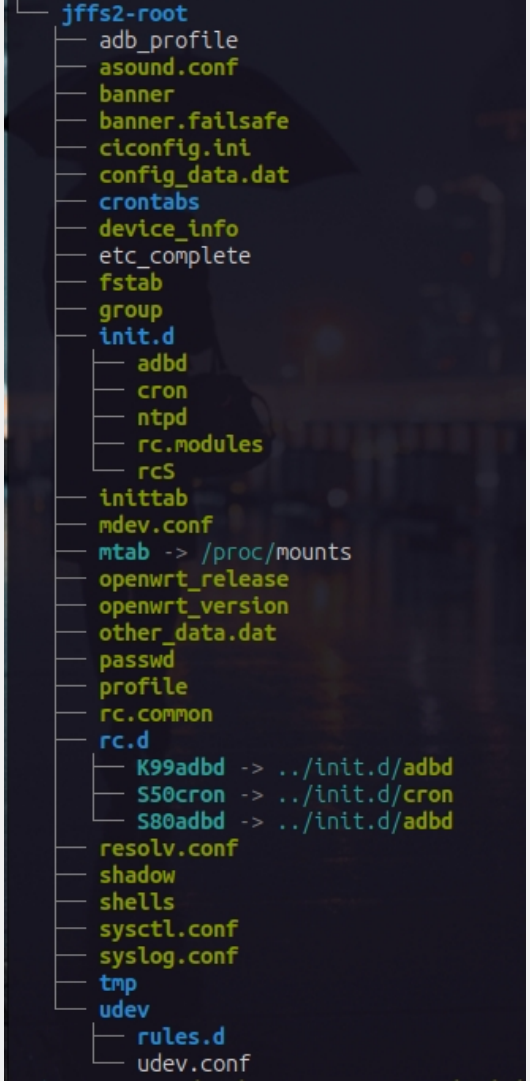
This one is “shirley/”

Finding misconfigurations in said conf files

- It helps to be somewhat familiar with the default configuration file for the service (if possible)
- Just read it
- Not every misconfiguration is actually exploitable or useful

Finding hard coded credentials

- Passwd file
- Shadow file
- Configuration files
- Custom application/script strings



Locating custom utilities and scripts

- A basic understand of Linux and common utilities goes a long way here in not wasting your time

Checking out custom scripts

- Usually as simple as reading them

File: **pseudo_init**

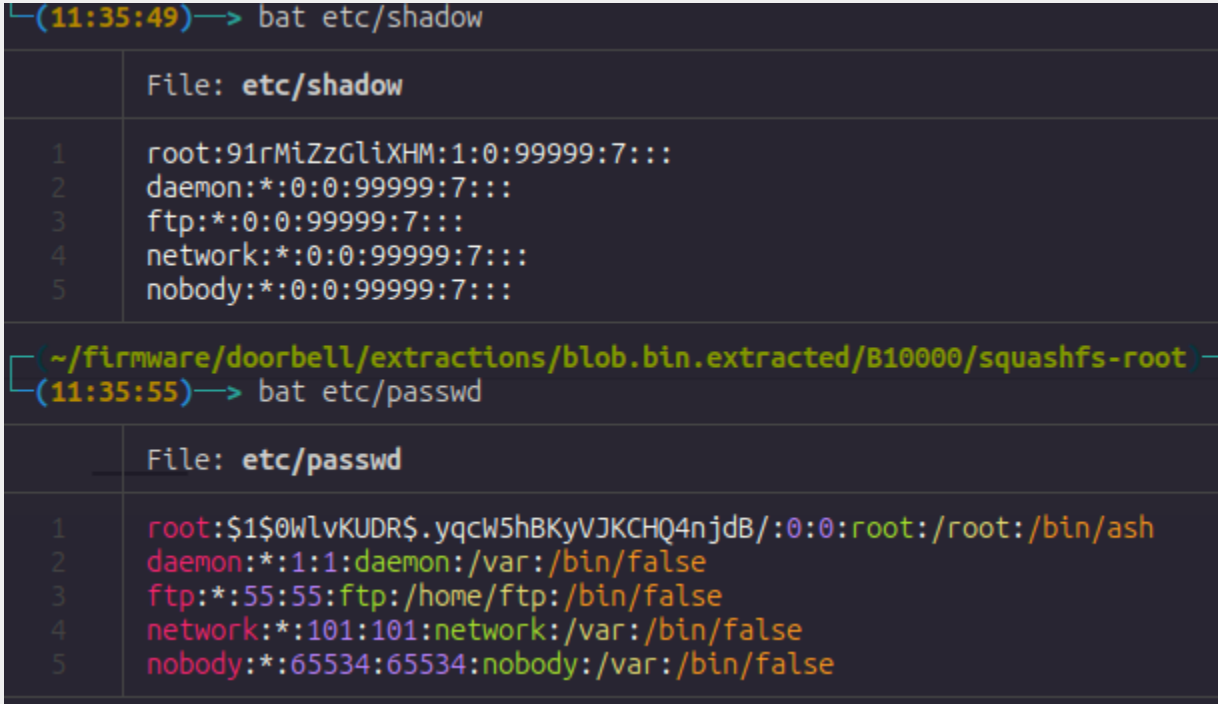
```
#!/bin/sh

MOUNT_ETC=1
MOUNT_OVERLAY=0

################################## functions ##################################

#mkfs_jffs2() <device in /dev/by-name>
mkfs_jffs2() {
    ! [ -x /usr/sbin/mkfs.jffs2 ] \
        && ! [ -x /sbin/mkfs.jffs2 ] \
        && echo "Not Found /usr/sbin/mkfs.jffs2 or /sbin/mkfs.jffs2" \
        && return 1

    # format to jffs2
    local erase_block=$(/bin/cat /proc/mtd \
        | /bin/grep "$(basename $1)" \
        | /usr/bin/awk '{print $3}')
    /bin/mkdir -p /tmp/jffs2.dir/tmp
    mkfs.jffs2 -p -e 0x${erase_block} -d /tmp/jffs2.dir \
        -o /tmp/jffs2.img >/dev/null || return 1
    /bin/dd if=/tmp/jffs2.img of=$1 || return 1
    /bin/rm -rf /tmp/jffs2.img /tmp/jffs2.dir
    return 0
}

mkfs_ubifs() {
    mkfs.ubifs -x lzo -y "$1"
}

mount_etc() {
    local etc_update=0
    # if enable ota, do update
    [ -f /etc/init.d/rc.ota-upgrade ] \
        && source /etc/init.d/ota-upgrade

    #local root_dev="$(readlink /dev/by-name/rootfs)"
    local root_dev="$(readlink /dev/by-name/rootfs_data)"

    # if mount failed, format.
    case "${root_dev}" in
        /dev/mtdblock*)
            # /bin/echo "mount jffs2"
            /bin/mount -t jffs2  -o rw,sync /dev/by-name/rootfs_data /etc
            # /bin/echo "mount jffs2 over"

            [ -e /etc/etc_complete -a ! -e /etc/etc_need_update ] \
                && return
            # /etc/etc_complete and /etc/etc_need_update both exist, that means we just need to update
            [ -e /etc/etc_complete -a -e /etc/etc_need_update ] && /bin/echo "do etc update" && etc_update=1

            cp -arf /etc/ciconfig.ini /tmp/
            cp -arf /etc/config_data.dat /tmp/
            cp -arf /etc/other_data.dat /tmp/
            /bin/umount /etc
```

Common things you might find in custom scripts

- Embedded credentials
- Possible endpoints
- OTA (Over The Air) update process
- telnetd
- DropBear SSH
- Poor error handling
- Understanding of how the filesystems are setup
- True enlightenment

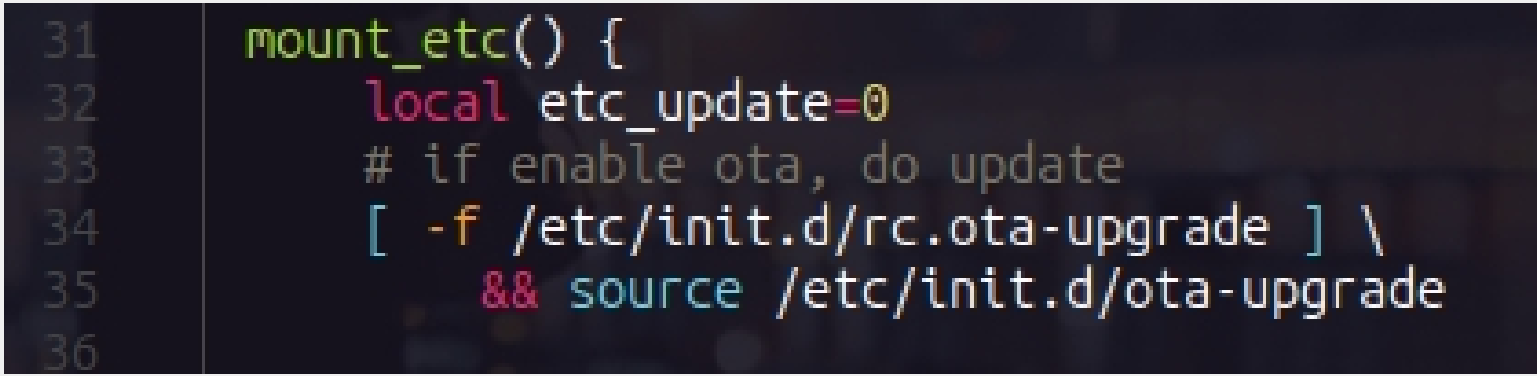
Embedded credentials

- Often hashed (but not always)
  - Google the hash



Over The Air updates

- A possible attack vector if there is no signing or security efforts put into the update process



telnetd

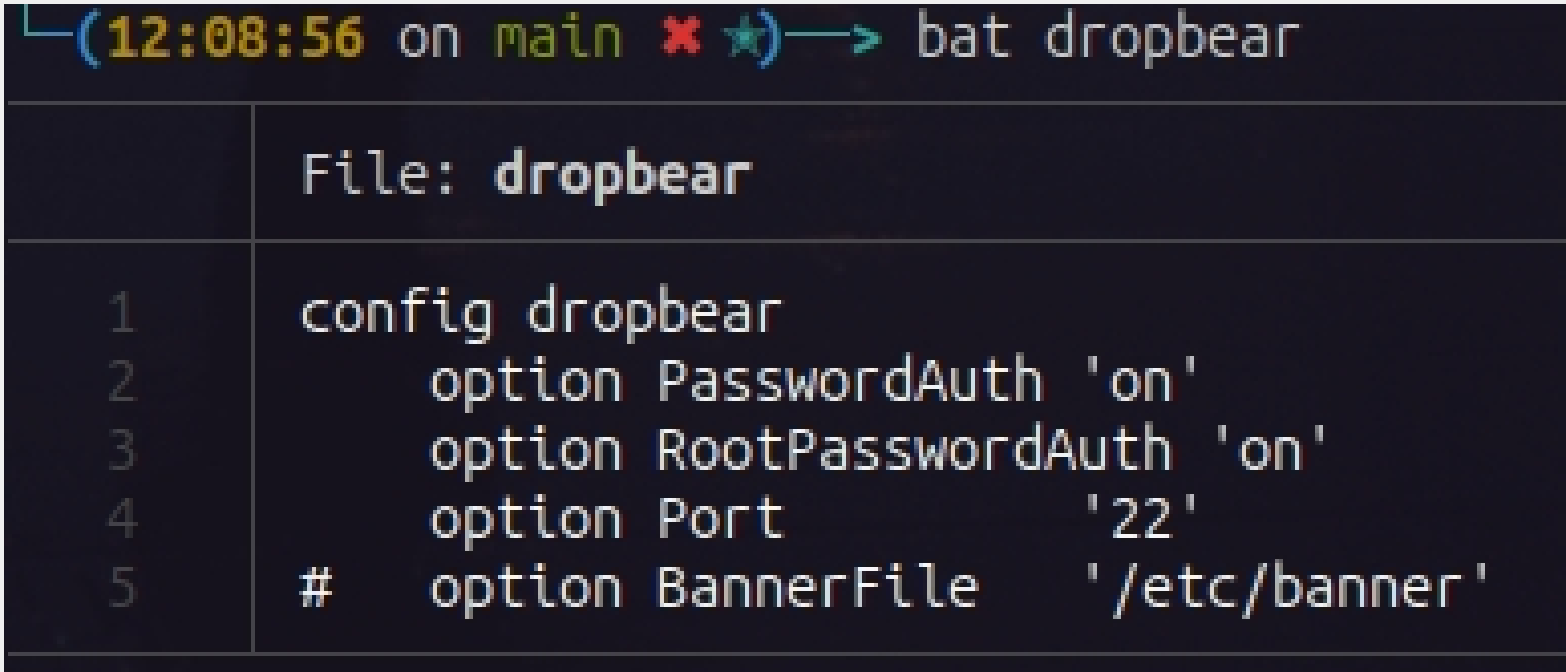
- It’s telnet, what more do you want?
- It usually is on the box and is configured, but is often not actually started at startup

```
426  #/usr/sbin/telnetd &
427  /sbin/syslogd &
428  #hardcode but fast
429  #mount_etc_hardcode
430  #set_parts_by_name_hardcode
431  #mount_usr
432
433  exec /sbin/init
```

```
426  /usr/sbin/telnetd &
427  /sbin/syslogd &
428  #hardcode but fast
429  #mount_etc_hardcode
430  #set_parts_by_name_hardcode
431  #mount_usr
432
433  exec /sbin/init
```

SSH

- Often Dropbear
- Often super out of date



Appalling script error handling

- It’s always bash
- No one ever does proper error handling in bash

Weird and wacky filesystem setups

- Explain how JFFS2 and SquashFS can be layered and can be used to do some funky stuff

Checking out custom utilities

- Strings
- Ghidra
- xxd

Common things to find in custom utilities

- Embedded credentials
- URLs/URIs
- IPs and ports
- API keys

```
└─(11:24:11)──> bat ciconfig.ini
```

File: **ciconfig.ini**

```
1   [wlan]
2   keepalive_interval=15
3   dtim_interval=6
4   suspend_pm=2
5   ap_ssid=
6   ap_pwd=
7
8   [server]
9   web_server_url=https://api.v2.gdxp.com;
10  cmd_server_ip=120.24.87.105;
11  cmd_server_port=8888;
12  udp_server_ip=120.24.87.105;
13  udp_server_port=25050;
14  stun_server_ip=8.218.91.142;
15  stun_server_port=17051;
16  p2p_server_ip=47.242.63.121;
17  p2p_server_port=17051;
18  push_server_url=http://120.24.87.105:58720;
19  oss_from_id=9;
20  oss_endpoint=http://oss-cn-shenzhen.aliyuncs.com;
21  oss_bucket=sz-aiwit;
```

File: **base_conf.ini**

```
1   [auth]
2   key=1111111111111111
3   [USERINFO]
4   #压力测试使用/正常填写Off即可
5   PressureTest=Off
6   #新产测
7   NewfactoryTest=On
8   #定时重启
```

F.A.C.T (Firmware Analysis and Comparison Toolkit)

- Great for basic static analysis of firmware samples
- You will still have to explore custom utilities and scripts yourself
- Really helpful to compare a new sample to see if you can expect something you've seen before

- https://fkie-cad.github.io/FACT_core/

Dynamic analysis

- Emulators
- Watching what the device is doing over UART/JTAG

Getting onto a device and seeing what’s up on it

- Exploring the filesystem on a live device
- Testing applications to see what they do

UART shell

- RX, TX, and GND
- Guess the baud rate
- Use the credentials that you found in the copy of the firmware you already extracted
- Done

Other shell options

- JTAG has some ways of getting a shell
- SSH
- Telnet

Network traffic

- How do we get it?
- What’s in it?

Network setup

- Dedicated vlan
- MitM
- Victory

Analyzing network traffic

- TCPdump
- Wireshark

SSL certificate abuse

- Lack of certificate pinning

Demonstrations (hopefully)

Ways firmware can be better secured

- Locking down the bootloader
- Stop shipping things in debug mode
- Encryption at rest

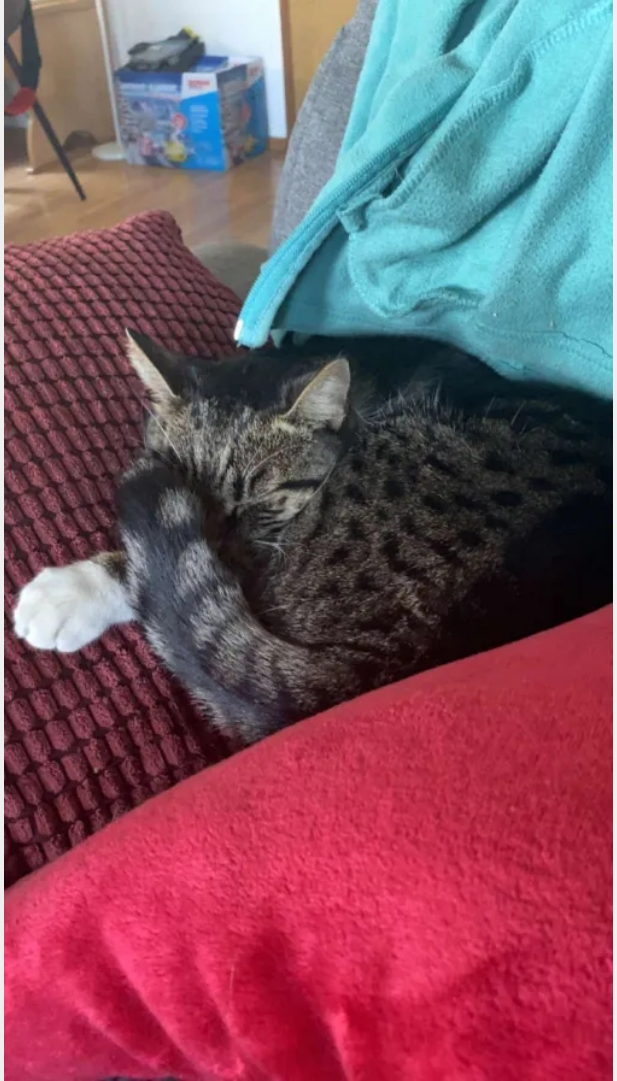
Possible impacts

Initial foothold

- Routers
- IP cameras

Botnets

- Mirai
- Bashlite
- Mozi



Questions?

Thanks for listening

Thank you for listening to me talk about this stuff for an hour, I hope you all enjoyed