DEFHAWK

Keeping an Eye on your Eye

Disassembling and Reassembling Smart Camera Hardware to breach YOUR Security

Presenters:
Ayushman Agrawal Hingorani
Azim Javed

DEFHAWK

DEFHAWK

Contents

- What is a “Smart Camera”
- Disassembling the Smart Camera
- Extracting Firmware
- Reading/Understanding the Firmware
- Reassembling the Smart Camera
- Notes & Future Discussions

What is a Smart Camera?

- Well, techically, a “Smart Camera” is any Camera with WiFi/Wireless Network Connectivity
- More broadly, any IoT Device that has an Image Sensor
- For us however, we can focus on mass market hardware, which largely relies on WiFi and Local Hotspots

DEFHAWK

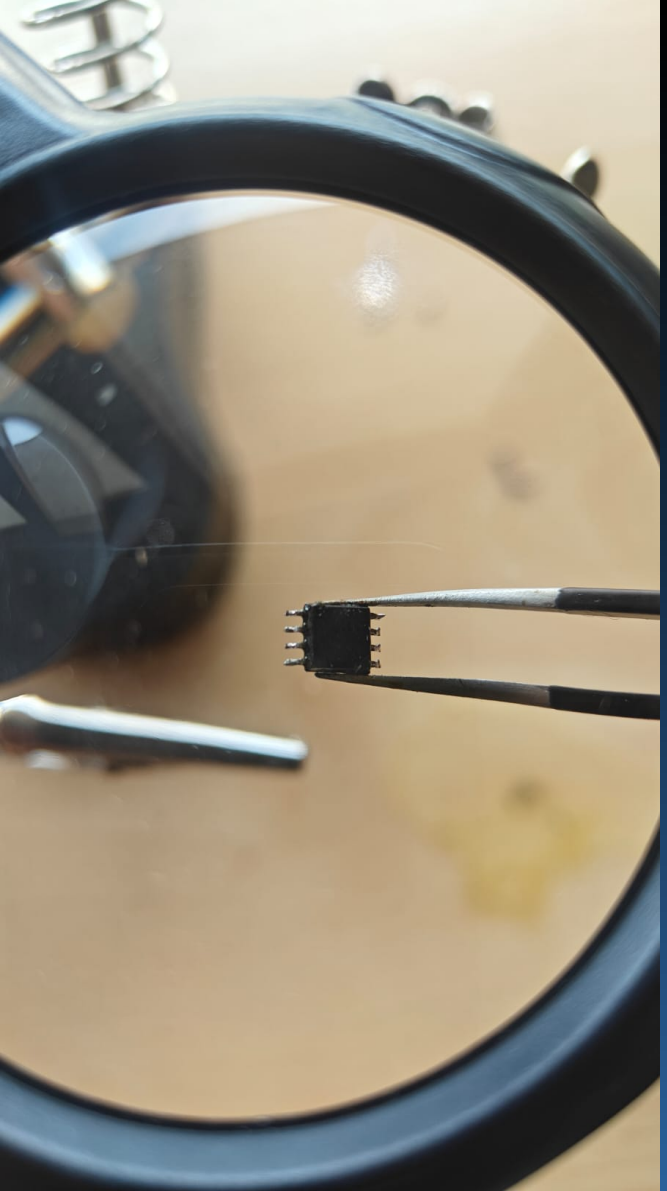
Disassembling The Camera

1	Pop Open
	1) Most Cameras are Pressure Fit 2) Some might also have screws

2	Figure Out Storage Chip
	1) Usually firmware stored in SPI-NOR Chip 2) Some also store in NAND (Higher End Cameras)

3A	SPI NOR
	1) Usually SOP8 2) Can be read via flashrom + CH341A

3B	SPI NAND
	1) Can be any form factor, WSON 8 is what we have come across 2) Can be read via IMSProg + CH341A



DEFHAWK

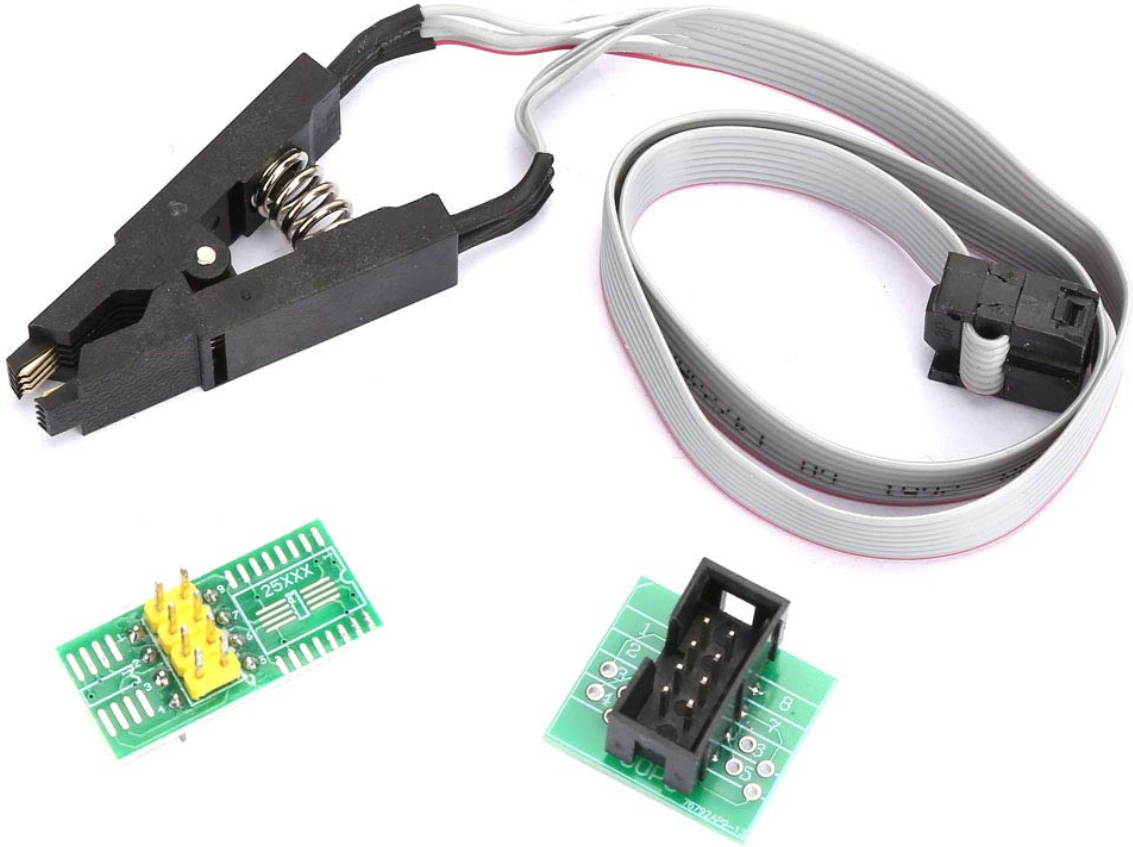
SOP-8

1) Most Common
2) In cheaper brands, may be “knockoffs” [I.e chinese clones] of well known chips, but would function identical software wise to the knockoffs
3) 90% of the chips can be read using a program known as “Flashrom” along with a CH341A Programmer
4) However...

Why do this?

- The next step, after you've disassembled the camera is to actually extract the firmware
- In the past, this could easily be done via exposed UART Ports
- However, recent revisions and newer cameras are coming with the RX line disabled either physically or via firmware configuration
- Hence, the most reliable way to actually extract the firmware is either by using a clamp OR desoldering the chip from the motherboard

DEFHAWK



SOIC 8 Clip

Extracting Firmware

What You Need

VIA CLIP

- SOP 8 Clip
- CH341A Programmer
- Flashrom (Ideally on a non-virtualized Linux Machine)

VIA Break out Board

- SMD Rework Station
- Solder Paste
- CH341A Programmer
- Flashrom (Ideallly on a non-virtualized Linux Machine)

DEFHAWK

Some Important Tips

1) Always know your chip voltage, and check what voltage your CH341A is working on
2) When desoldering, use ample solder paste AND protect surroundings using Kapton Tape
3) If working on critical/expensive hardware, do not cheap out; Get the best possible tools budget allows



DEFHAWK

Example: Using a breakout board, with a 1.8V limiter board on top of the CH341A

Reading Firmware

Reading the Firmware

1) Establish size of firmware, along with initial reconnaissance
2) Perform binwalk to further understand possible partitions
3) Use dd to carve out specific partitions
4) Load the entire firmware, or specific partitions, into a binary analysis tool such as Ghidra (amongst others)

Important Points

- Always make a backup of the firmware dump
- You can use tools apart from Ghidra depending on your needs and comfort
- A good initial recon (mainly via google/internet searching) is important to ensure a smooth journey into actually understanding the firmware

Some Observations

- When loading into Ghidra, the following settings are the most common:
  - Language: MIPS; Variant: Default; Endian: Little; Compiler: Default;
- File systems being used can include:
  - SquashFS
  - JFFS2
  - Proprietary Filesystems like MiniFS (TP Link) and Anksys FS

Case Study: MiniFS

- Developed by TP Link for their routers, can also be found on their IPTV Smart Cameras

```
binwalk -A firmware.bin

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
9420          0x24CC          MIPSEL instructions, function epilogue
10232         0x27F8          MIPSEL instructions, function epilogue
11310         0x2C2E          MIPSEL16e instructions, function epilogue
13924         0x3664          MIPSEL instructions, function epilogue
14064         0x36F0          MIPSEL instructions, function epilogue
14140         0x373C          MIPSEL instructions, function epilogue
41236         0xA114          MIPSEL instructions, function epilogue
```

```
binwalk -B firmware.bin

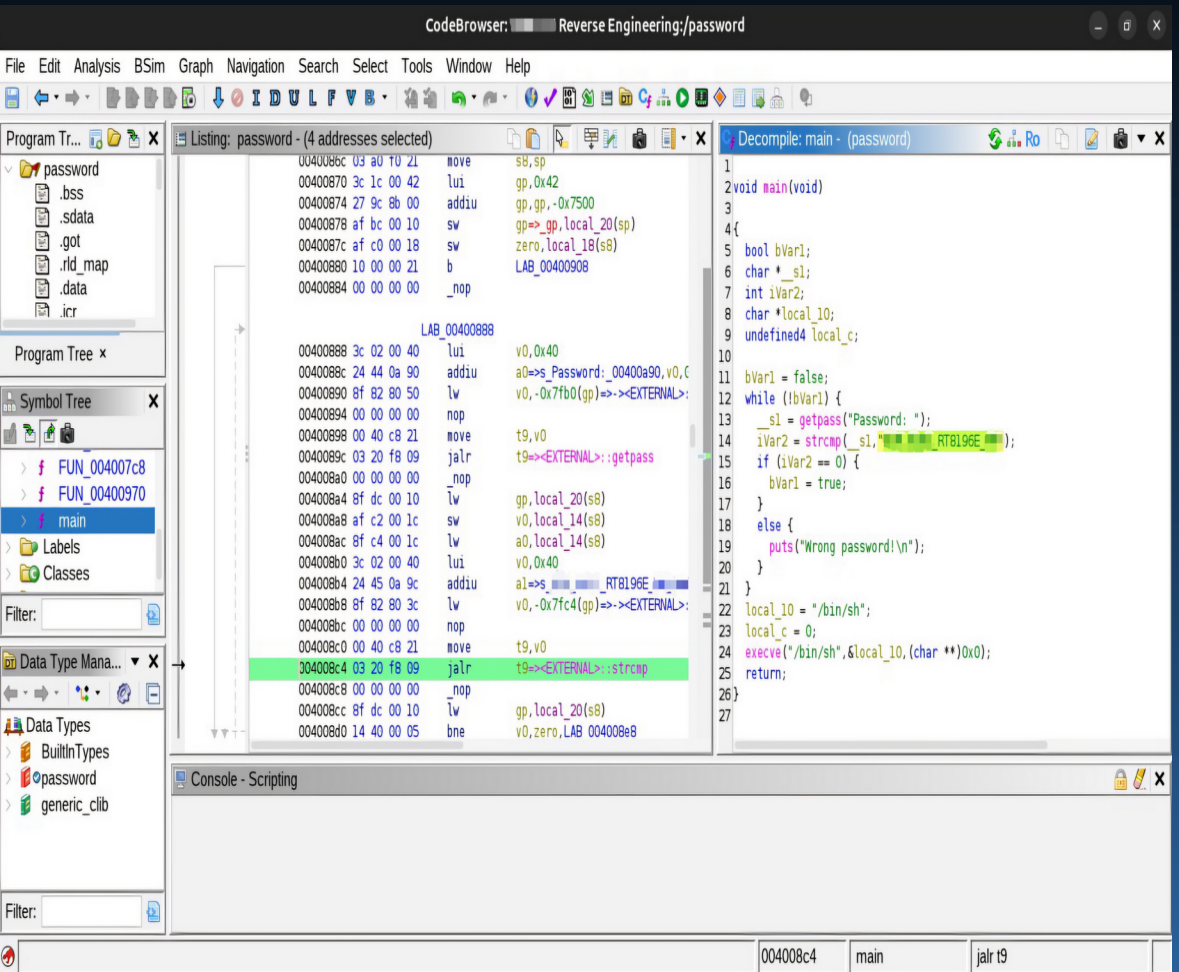
DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
41248         0xA120          U-Boot version string, "U-Boot 1.1.3 (Jul 22 2021
- 16:38:33)"
53248         0xD000          IMG0 (VxWorks) header, size: 1392864
53376         0xD080          LZMA compressed data, properties: 0x6E, dictionary
size: 8388608 bytes, uncompressed size: 2455600 bytes
869684        0xD4534         LZMA compressed data, properties: 0x5D, dictionary
size: 8388608 bytes, uncompressed size: -1 bytes
915873        0xDF9A1         LZMA compressed data, properties: 0x5D, dictionary
size: 8388608 bytes, uncompressed size: -1 bytes
934810        0xE439A         LZMA compressed data, properties: 0x5D, dictionary
size: 8388608 bytes, uncompressed size: -1 bytes
963514        0xEB3BA         LZMA compressed data, properties: 0x5D, dictionary
size: 8388608 bytes, uncompressed size: -1 bytes
1077639       0x107187        LZMA compressed data, properties: 0x5D, dictionary
size: 8388608 bytes, uncompressed size: -1 bytes
1104257       0x10D981        LZMA compressed data, properties: 0x5D, dictionary
size: 8388608 bytes, uncompressed size: -1 bytes
1128073       0x113689        LZMA compressed data, properties: 0x5D, dictionary
size: 8388608 bytes, uncompressed size: -1 bytes
1153431       0x119997        LZMA compressed data, properties: 0x5D, dictionary
size: 8388608 bytes, uncompressed size: -1 bytes
1183988       0x1210F4        LZMA compressed data, properties: 0x5D, dictionary
size: 8388608 bytes, uncompressed size: -1 bytes
1223806       0x12AC7E        LZMA compressed data, properties: 0x5D, dictionary
size: 8388608 bytes, uncompressed size: -1 bytes
1248762       0x130DFA        LZMA compressed data, properties: 0x5D, dictionary
size: 8388608 bytes, uncompressed size: -1 bytes
1273152       0x136D40        LZMA compressed data, properties: 0x5D, dictionary
size: 8388608 bytes, uncompressed size: -1 bytes
1299124       0x13D2B4        LZMA compressed data, properties: 0x5D, dictionary
size: 8388608 bytes, uncompressed size: -1 bytes
1332094       0x14537E        LZMA compressed data, properties: 0x5D, dictionary
size: 8388608 bytes, uncompressed size: -1 bytes
1358336       0x14BA00        LZMA compressed data, properties: 0x5D, dictionary
size: 8388608 bytes, uncompressed size: -1 bytes
1385844       0x152574        LZMA compressed data, properties: 0x5D, dictionary
size: 8388608 bytes, uncompressed size: -1 bytes
1412126       0x158C1E        LZMA compressed data, properties: 0x5D, dictionary
size: 8388608 bytes, uncompressed size: -1 bytes
1441339       0x15FE3B        LZMA compressed data, properties: 0x5D, dictionary
size: 8388608 bytes, uncompressed size: -1 bytes
```

DEFHAWK

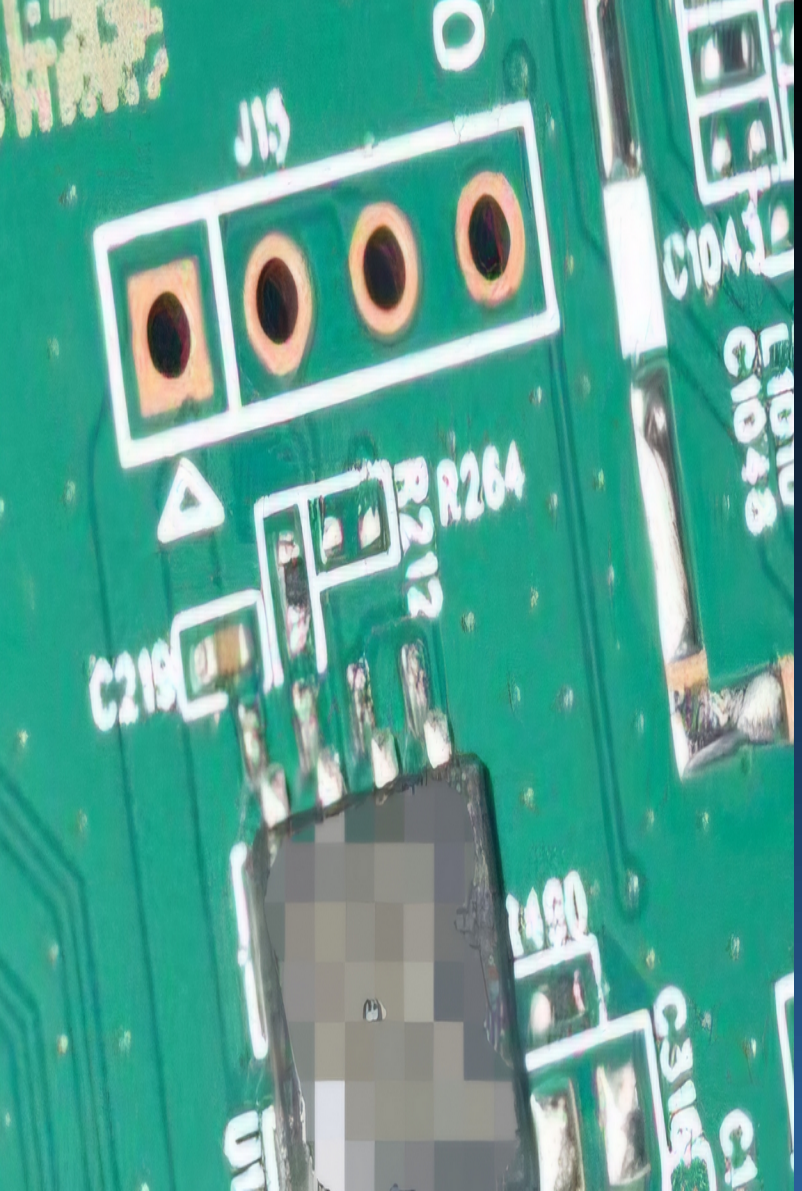
1) Magic Signature: MiniFS
2)Built on top of VxWorks
3) Uses a custom Proprietary Bootloader
4) Even though it comes up as “U-Boot”, running strings firmware.bin | grep boot returns the proprietary name
5) The LZMA Compressed Data seen is actually a part of the original IMG0
6) For more information, lets talk!

DEFHAWK

Case Study: When UART Fails



1) Used SOIC8 Clips along with CH341A to dump the firmware
2) Loaded the firmware into Ghidra
3) In one of our targets, an ELF binary named "password" was found which is suspected to be the root-console password utility
4) It prompts the console user with a password input. The input is compared against a hard-coded string. If the password matches, it spawns /bin/sh and grants root shell access. Otherwise, it prompts with "Wrong Password!"



DEFHAWK

1)Maybe, this could lead us to get root on the UART shell, so why not try that!
2)But…. manufacturer had intentionally disabled UART communication.
3)R212 and C219 were disconnected.
4)Checked it all with a logic analyzer.
5)Showed literally no UART traffic, communication channel was never established.

An Interesting Paper

- “Reverse Engineered MiniFS File System”
- By Dmitrii Belimov, Evgenii Vinogradov
- Link : https://arxiv.org/abs/2407.05064
- Talks about extraction of firmware, and tries to recreate the structure of the firmware
- Falls short on actually reverse engineering the firmware in full, showing us gaps still exist

Reassembling

Basically, you reverse it

- You basically follow the steps you did in disassembling, in reverse.
- This means soldering the chip back to the board (incase you desoldered it) and then putting all the pieces back together like legos

Tips

- The more carefully you do it, the more “professional” it becomes; Remember: You can’t just disassemble and destroy everything. Sometimes you need it to actually survive the ordeal so you can further test it
- Incase you are flashing custom firmware, remember to keep backups. Incase the new firmware doesn’t work, you will need to repeat the steps and flash back the old firmware.

Challenges

- Damaged Electronic Components
  - Excessive Heat
  - Accidental desolder of other components
  - Electrostatic Discharge
  - Wrong pin order/polarity
- Broke Shell
  - Excessive Force
  - Anti-tamper mechanisms
  - Act of God (Really)