ENPM664 Final Report

Team Members:
Brandon Perkins
Steve Routh
Samridha Murali
Sumanth Thyagarajan
Michael Lindsey

05/10/2022

Table of Contents

Executive Summary

Home security has always been a necessity and with the continuing advancement in technology, home security devices from companies like Ring and Nest have become available to homeowners. Unfortunately, these products are usually very expensive and many people look for cheaper alternatives. These alternative devices may be conveniently cheaper and appear to offer a similar level of service compared to name brand devices, but they are what customers pay for and are not usually designed with protecting the devices in mind. They tend to have vulnerabilities that can be exploited, creating an easy attack vector for attackers.

One such device is the Wyze Cam security camera. This device is well known for its “hackability”, as there are documented techniques, including tutorials, on how to download and modify the firmware. The team’s objective is to perform a firmware analysis on the device firmware to identify the software components and potential vulnerabilities that could be exploited. The team will then provide a write up documenting our findings, including description of the vulnerabilities, how they could be exploited, and what they would allow an adversary to do with the device.

The Team

Brandon Perkins
I graduated from Virginia Commonwealth University in 2015 with a Bachelor's degree in Computer Science and have been working in software development and testing focused in cyber security for 7 years now. I have 12 years of experience in linux (7 professional years), 10 years of python experience (7 years professional), and about 4 years of experience in C/C++.

Steve Routh
I currently work for Johns Hopkins University Applied Physics Laboratory (JHU-APL) in Laurel Maryland as a Systems Engineer. Prior to working for JHU-APL, I performed system- and component-level design, integration, testing, and troubleshooting of hardware and software on Naval air and surface platforms. My undergraduate degree is in Electrical and Computer Engineering from Drexel University. I have experience with C, Linux, Python, and Assembly.

Samridha Murali
I graduated with a Bachelor's of Technology in Computer science and Engineering from Manipal Academy of Higher Education with a minor in Network and Security in 2021. I have worked in industry as a software development intern and Software reliability engineer intern. I am proficient in Linux, Python, C, Social engineering.

Sumanth Thyagarajan
I graduated with a Bachelor's of Technology in Computer science and Engineering from SASTRA University, India, in 2018. I have 3 years of experience in cybersecurity as an Identity and Access Management - Software developer. I have sufficient experience with Linux, C, Java, and python.

Michael Lindsey
I graduated with a Bachelor's in Computer Science and a minor in Cybersecurity from the University of Maryland in 2021. I have worked in the industry as a software developer intern and a security engineer intern. I have sufficient experience with Linux, C, and Python. My cybersecurity area of interest is binary exploitation.

Team Collaboration
For communication we will be using a combination of Zoom and Signal. Report collaboration will be done over Google Docs.

Related Works and Background

The Wyze camera is marketed as a security camera. It records video and sound, which may be uploaded to cloud storage and playback service. The device has a built-in speaker and microphone for 2-way communication, WiFi adapter, color night vision, and is compatible with iPhone and Android devices [1].

Wyze has more than 1 million users. Communications requests between mobile devices, Wyze products and AWS are made via https. Each handshake is validated by the camera's own secret key and certificate [2]. Wyze uses AES 128-bit encryption to protect confidentiality of the live stream and playback data. Wyze uses Two-factor authentication to secure accounts, with secondary authentication token or code.

There have been many documented exploits and firmware hijacking attacks performed on the Wyze Cam. Most of the existing attacks and POCs on the Wyze camera focuses on feature unlocking, theft of services, performing Man in the Middle attack [3], and starting new services on the device [4]. The security researchers and firmware developers were able to enable telnet, redirect logs and recordings to NFS, enable RTSP for live streaming , and archive recording [5] [6] [7].

Wyze Cam V3 had an Authentication bypass vulnerability (CVE-2019-9564) [8] and a Remote control execution flaw caused by a stack-based buffer overflow (CVE-2019-12266) [9] vulnerabilities before v4.36.8.32. When these two vulnerabilities are used in combination, malicious actors can gain remote access to the camera's video feed [10]. On March 17, 2022, a new patch with security improvements was released for Wyze Cam V3 [11].

For remote authentication, the client that needs to be authenticated should send an Input/Output Control (I0Ctl) command with ID 0x2710 to the device. To that, the device generates a random value and encrypts it with a 16-byte “enr” (AES encryption key), and sends it to the client. Since the “enr” (key) is known to the client, it decrypts it and sends the decrypted value back to the device in an I0Ctl command with ID 0x2712. If the value matches, the client is authenticated. According to a whitepaper published by BitDefender, when the client sends the 0x2710 command, the device stores the generated random value in memory. When the 0x2710 command is not sent the memory remains NULL. So, when a client sends a 0x2712 command with authentication bytes set to NULL, the device compares NULL with NULL and authenticates the client. After authentication, the device is fully controllable including toggling the camera on/off, enable/disable recording to SD, and motion control (pan/tilt). However, live audio and video feed cannot be read because it is encrypted with the “enr” (key), unless the buffer overflow in the next paragraph is exploited [12]. (CVE-2019-9564)

Buffer overflow vulnerability can be exploited by sending an input of size 0x7f or more with the I0Ctl command with ID 0x2776. It will overwrite the return address of the function. In the request, the length of the buffer is specified in the first byte, then the buffer [12]. This attack could allow remote code execution on the camera device. (CVE-2019-12266)

Furthermore, the content of the SD card can be read through the webserver running on port 80 of the device. When an SD card is inserted, the device creates a symlink in the www directory which is served by the webserver. The SD card also holds the log files, which may include the “enr” (key) and Unique Identification Number (UID) values that could be used to connect remotely. [12] There is no CVE for this vulnerability, but was fixed in a firmware release January 29 2022 [13].

Tools like Trommel and Firmwalker scans through the embedded devices file to identify the potential vulnerable indicators. These tools search in the extracted firmware filesystem for vulnerabilities and things of interest like passwords, configuration files, scripts, URLs, email addresses, web servers, etc [14][15].

The Wyze Cam camera implements a weak encryption algorithm for its communication. The security researcher was able to compromise the device and disclose sensitive information like users' email addresses, passwords, WiFi network names, and WiFi passwords [16].

Based on the DMCA security research exception, it is legally allowed to perform security research on IoT devices and Firmware analysis for classroom purposes. Any vulnerabilities found during this process will be disclosed responsibly to the company/vendors without violating the DMCA [17] [.

Project Description

Project Idea

Our project idea is to perform firmware analysis on an IOT device in the hope of finding vulnerabilities that could be exploited by a malicious attacker. The device that will be performing our analysis on is the Wyze Cam V3. The Wyze Cam V3 is a small IOT camera that allows for live surveillance through the “Wyze” smartphone app. The firmware for the Wyze Cam V3 is hosted on the website of its manufacturer for anyone to download. We plan to download the firmware and perform a variety of analysis techniques on it until we have a solid understanding of the device and/or have identified potentially exploitable vulnerabilities.

Implementation of Project

We will start by performing manual analysis on the firmware. This manual analysis will include using tools like binwalk to dissect the firmware and command-line tools (like find and grep) to search the firmware’s file system for notable artifacts. After a thorough manual analysis, we will use automated analysis tools (like trommel and firmwalker) to identify possible vulnerabilities in the firmware. If there are any vulnerabilities identified by the automated analysis then we will follow-up on any identified vulnerabilities with manual analysis. If there aren’t any vulnerabilities identified by the automated analysis tools then we will perform manual vulnerability analysis (static analysis, dynamic analysis, fuzzing) on custom binaries in the firmware. If we are able to find an exploitable vulnerability then we will develop a proof-of-concept exploit and test it on the Wyze Cam V3.

Required Materials

1. Wyze Cam V3 Firmware
2. Manual Analysis Tools (binwalk, find, grep, etc.)
3. Automated Analysis Tools (Trommel, firmwalker, etc.)
4. Linux Environment (or Linux VM)

Optional Materials (only needed if exploitable vulnerability is found)

5. Wyze Cam V3 (only need if found exploitable vulnerability)
6. Wyze App for Android / iPhone (allows the user to interface with the camera)

Milestones

- Perform preliminary manual analysis on Wyze Cam V3 firmware
- Map out important components of firmware
- Perform automated vulnerability analysis
- Perform manual vulnerability analysis
- **[Optional]** Develop POC exploit for identified vulnerabilities

Timeline

4/12 - Finish manual analysis of firmware and mapping of important components
4/19 - Finish automated vulnerability analysis
4/26 - Finish manual vulnerability analysis and any exploit POC
5/2 - Finish final presentation and final report

Project Results

Hardware Analysis

The hardware analysis focused on the holistic capabilities of the board, specific to the boot processes and instruction set characteristics. This supported analysis of system boot, and binary analysis of the firmware based on the Ingenic T31 System on Chip (SoC) instruction set. As you will see, a memory map has been partially created, identifying sections of memory pertinent to u-boot and Linux. Memory assigned to peripherals was not included due to time constraints and depth of analysis, however is available to be completed in the future.

U-boot was accessed via JTAGULATOR™ (http://www.grandideastudio.com/jtagulator/) and the Linux virtual machines (VM). A noble attempt at soldering wires to six test pads was attempted, with disappointing results, and is left for a future attempt.

Hardware was emulated using Firmadyne software provided on the class VM. Root access was gained on the emulated system. However, root access was not gained on the actual Wyze camera hardware. Attempts to do so will be described in detail.

Physical disassembly

The electronics are housed in a waterproof exterior with rubber gaskets sealing the front face with camera lens, rear USB wiring. The USB port and setup switch face towards the bottom, with rubber protective covers. A speaker is mounted to the upper rear portion of the case and sealed in place with silicone. Silicon is also placed over the USB wire entry point below the speaker. The case is assembled with three recessed phillips-head screws. A hobby screwdriver can be used to remove the screws. Rubber plugs cover the screws in the recessed openings. A plastic white frame covers the openings and presents a finished appearance. These features can be seen in the pictures in Figure A-1.

Figure A-1: Clockwise from left: camera front, left side, rear, and bottom [19]

Three wires connect the electronics to the case: a front light sensor, the rear speaker, and the rear USB cord. Care should be taken to not stress these wires during disassembly. The front sensor and speaker can be disconnected from the board, however the silicon securing the USB cable will need to be removed to relieve strain while analyzing the components. Figure A-2 shows the electronics removed, with the speaker and USB wires running toward the back. Notice the speaker and USB cable connect near each other on opposite sides of the top board. Also notable is that the top board houses most of the integrated circuit components.

The electronics components are mounted on two printed circuit boards (PCB) connected by a wire bus. The boards are folded on top of one another and connected by more screws (also phillips head).

Figure A-2: Case with electronics partially removed [19]

Component Identification and Analysis

The internal electronics consist of two PCBs connected by a wire bus. The boards are held together by the plastic housing and phillips head screws. The front board contains the infrared Light Emitting Diodes (LED), Secure Digital (SD) card reader, switch, six test points, and 20-pin cable connector. This board can't be seen in Figure A-2 because of it's placement between the black front plastic face and the visible rear board.

The rear board is easily seen in Figure A-2, facing upward and provides the insertion points for the Integrated Circuits (IC). Installed on it are the microprocessor, wifi, sound, and optical chips; USB, speaker, light sensor, and 20-pin cable connectors; WiFi antenna; and through-holes for Universal Asynchronous Receiver-Transmitter (UART) connections. For the purpose of component identification, the terms "primary" and "secondary" will be used to identify one board from the other. The board visible in Figure A-2 will be identified as the "primary" board.

Primary Board Analysis

The primary board (Figure A-3) provides insertion points for multiple IC, power, speaker, and 20-wire bus. Clockwise from top center: 128 MB flash memory, Realtek TRL8189FTV 802.11n WiFi processor (green PCB) w/ 26.0 MHz clock and WiFi antenna, Ingenic T31 MIPS32 System on Chip (SoC), 24.0 MHz clock, and power conditioning completing the cycle on the bottom left . The SmartSens SC4335 image sensor and Broadchip BCT8933 audio amplifier are located on reverse side center and top right respectively, as shown in Figures A-4 and A-5. [19] directly references the Ingenic T31 and Realtek 8189 chips but not the other components.

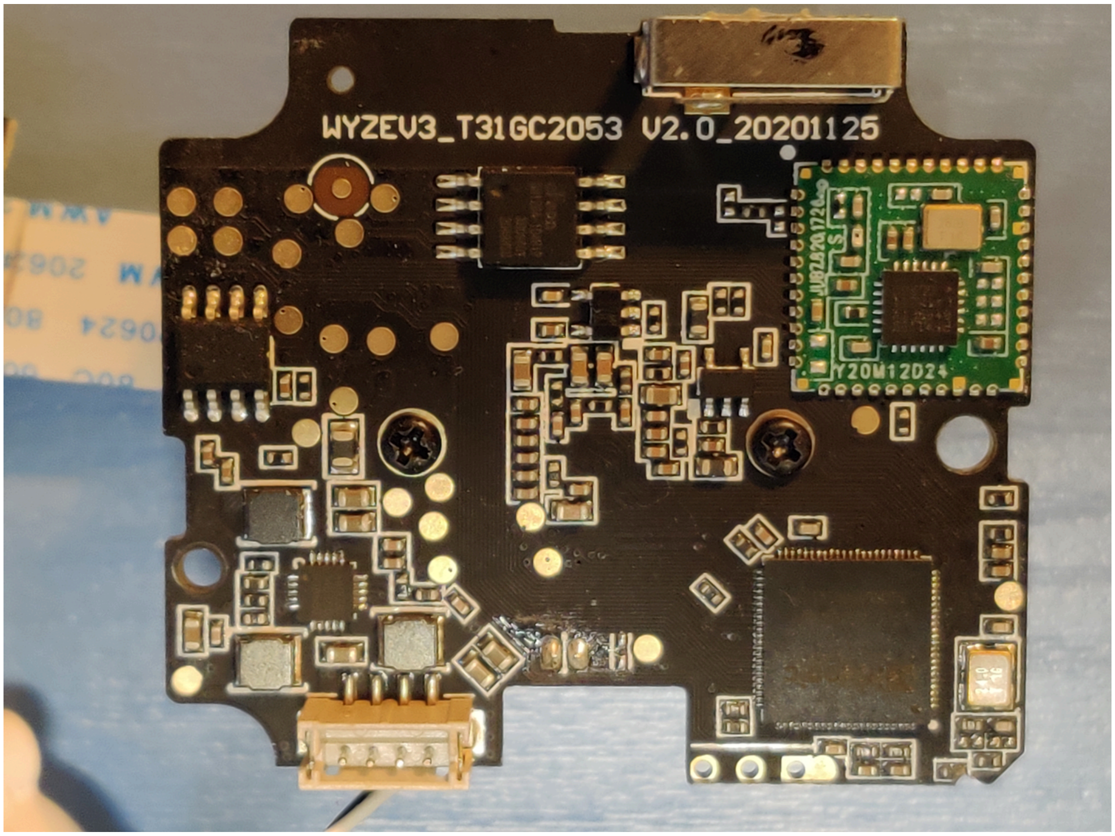


Figure A-3: Primary board (side A)

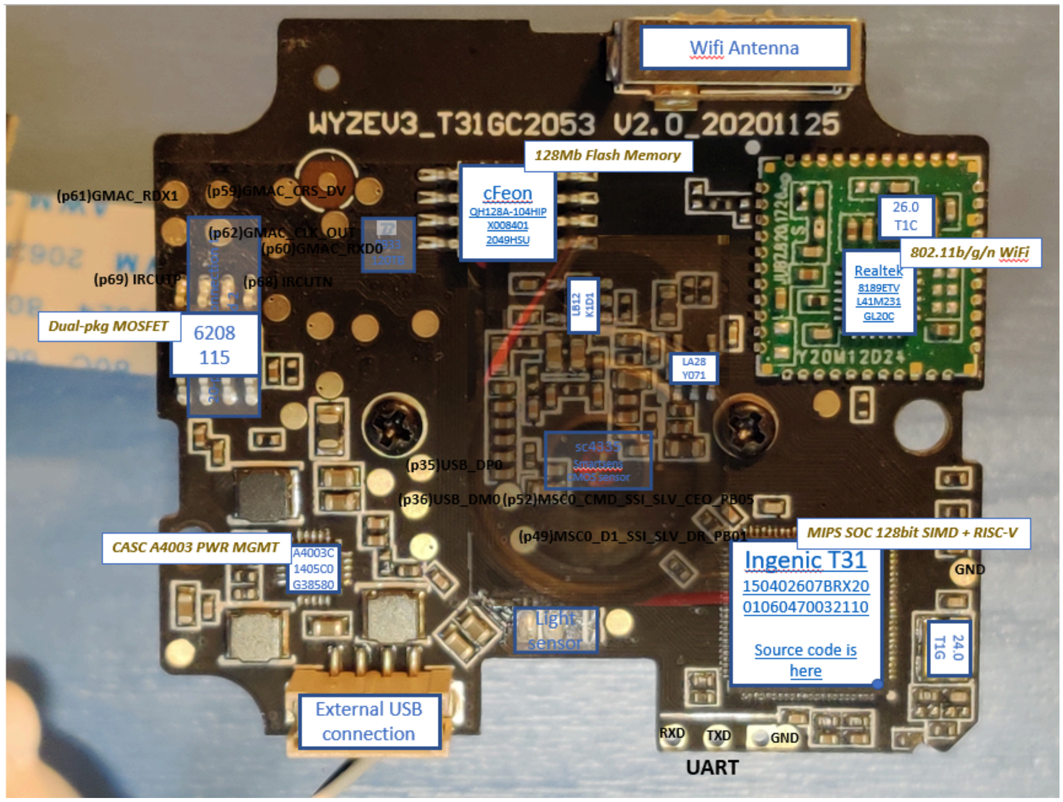


Figure A-4: Primary board (side A, labeled)

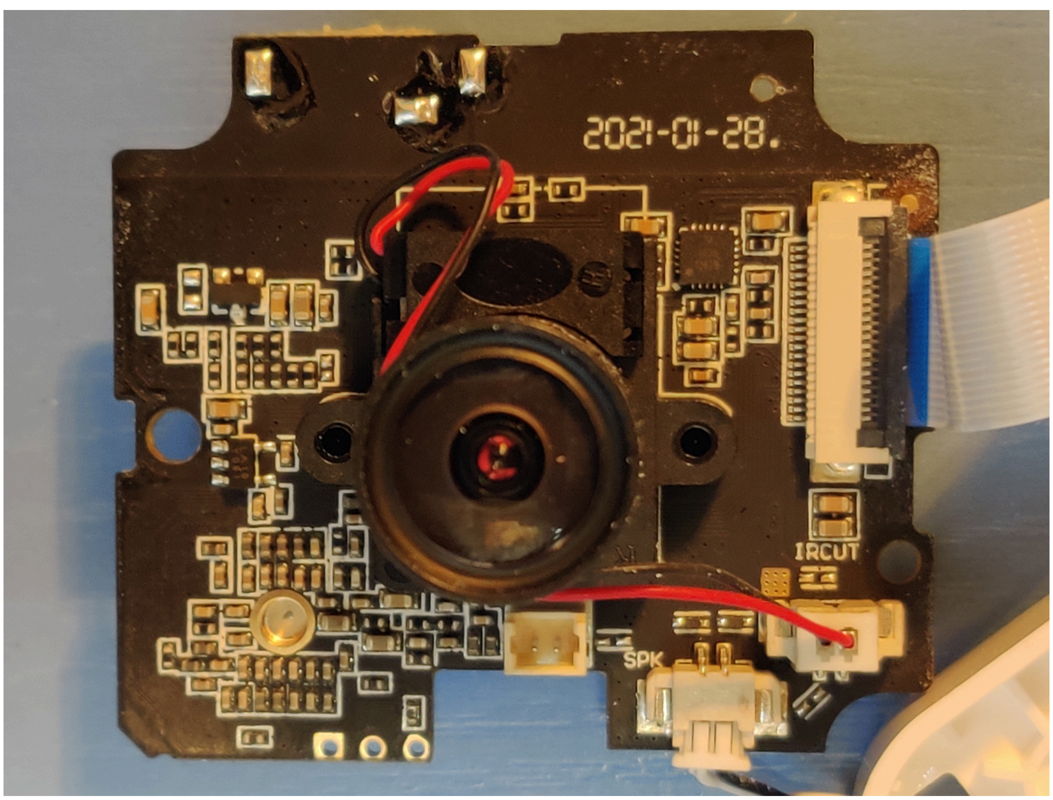


Figure A-5: Primary board (side B)

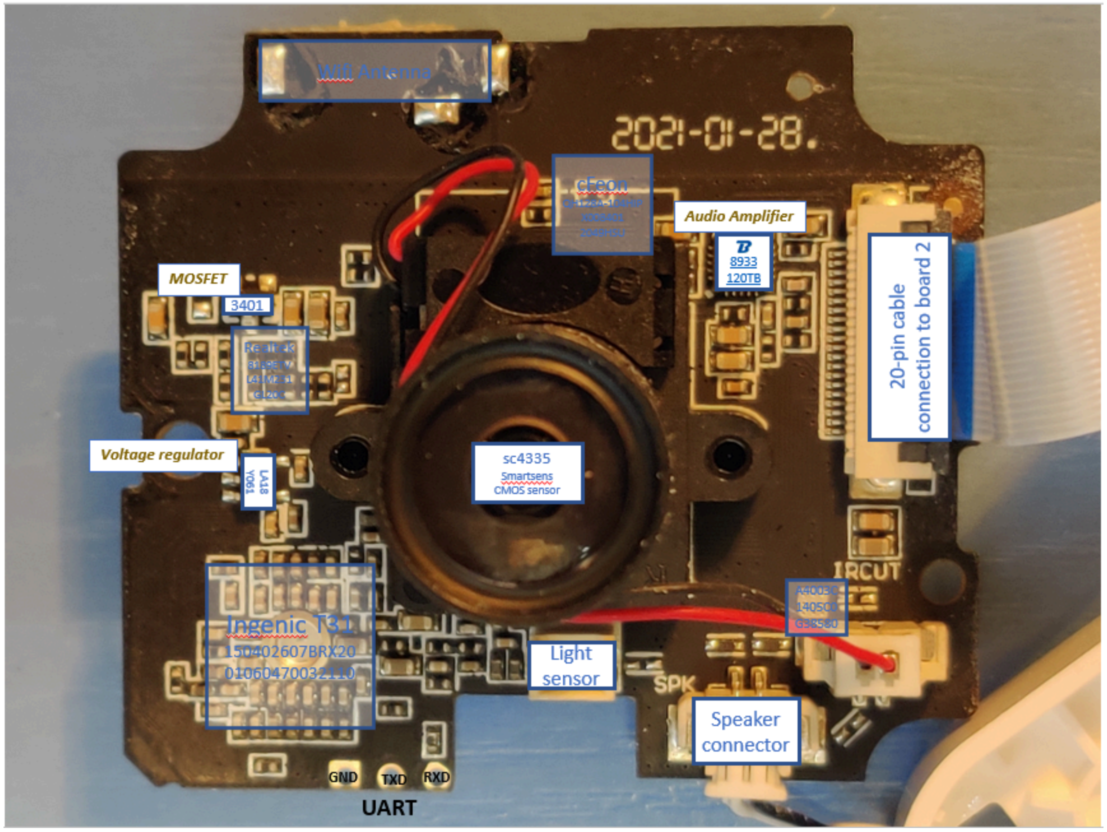


Figure A-6: Primary board (side B, labeled)

Ingenic T31 SoC

The Ingenic T31 is a System on Chip design and is comprised itself of the XBurst1 Central Processing Unit (CPU) (1.5 GHz, dual coprocessors, 128 bit SIMD Engine), integrated 128 MB DDR, video processing, RISC-V core @ 500 MHz, audio codec, UART/SPI/I2C/JTAG interfaces, and onboard encryption services.

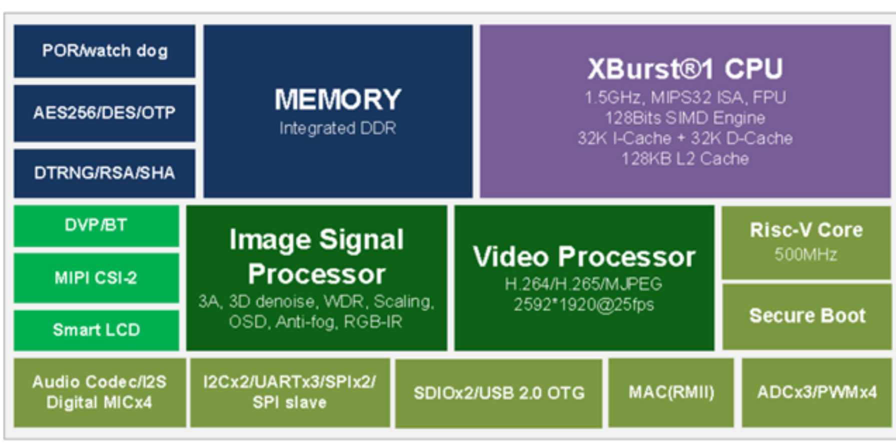


Figure A-7: Ingenic T31 System on Chip [20]

The T31 can be dual-booted (Figure A-8) and claims to be image-stable by 200 ms using auto-exposure and auto-white balance hardware acceleration. Initial boot is assumed to be accomplished by the RISC-V processor [21], which then bootstraps the XBurst1,but this has not been confirmed by other sources.

Security support by the T31 includes Secure Boot and on-board encryption (AES, DES, RSA, SHA, TRNG, OTP).

CPU	Ultra high frequency, up to 1.5GHz . Vector Deep Learning accelerator base on SIMD128 64KB + 128KB L1/L2 Cache RiscV independent lite core
Video Encoder	H.264/H.265/MJPEG encoder Maximum 2592*1920@30fps World class advanced encoder engine Support multiple streaming and various features
Starlight ISP	Dedicated optimizations for low light and surveillance scenarios Upgraded 2D / 3D noise reduction Sharpening enhancement, ROI-AE Advanced WDR, DRC Distortion correction
Memory	Capacity of 512Mbit or 1Gbit
Security	AES/RSA/SHA/TRNG/OTP Support secure boot
AI algorithm	Support deep learning algorithm with high precise and good flexibility Human detection, Facial detection/recognition Cry detection, Vehicle detection, Pets detection
Package	22nm process Package：QFN / BGA
Fast Boot	Support Fast Boot - Dual boot - Fast AE / AWB - ~200ms stable video output
Wide extension	Support 4-channel digital MIC array Support IoT-WIFI / BT / 4 Support SLCD Display Support UVC / UAC
Audio	Integrated Audio Codec Support Rate 8K/12K/16K/24K/32/44.1K/48K/96K Support I2S Interface Echo cancellation
Connectivity & Peripherals	WDT, ADC, UART, I2C, SPI, GPIO, SDIO, PWM, USB-OTG, GMAC

Figure A-8: Ingenic T31 Specification [20]

The T31 SoC uses a robust set of interfaces, some of which are used for internal communications or with other chips on the PCB. This includes UART, JTAG, I2C, and SPI. These interfaces are identified in Figure A-9.

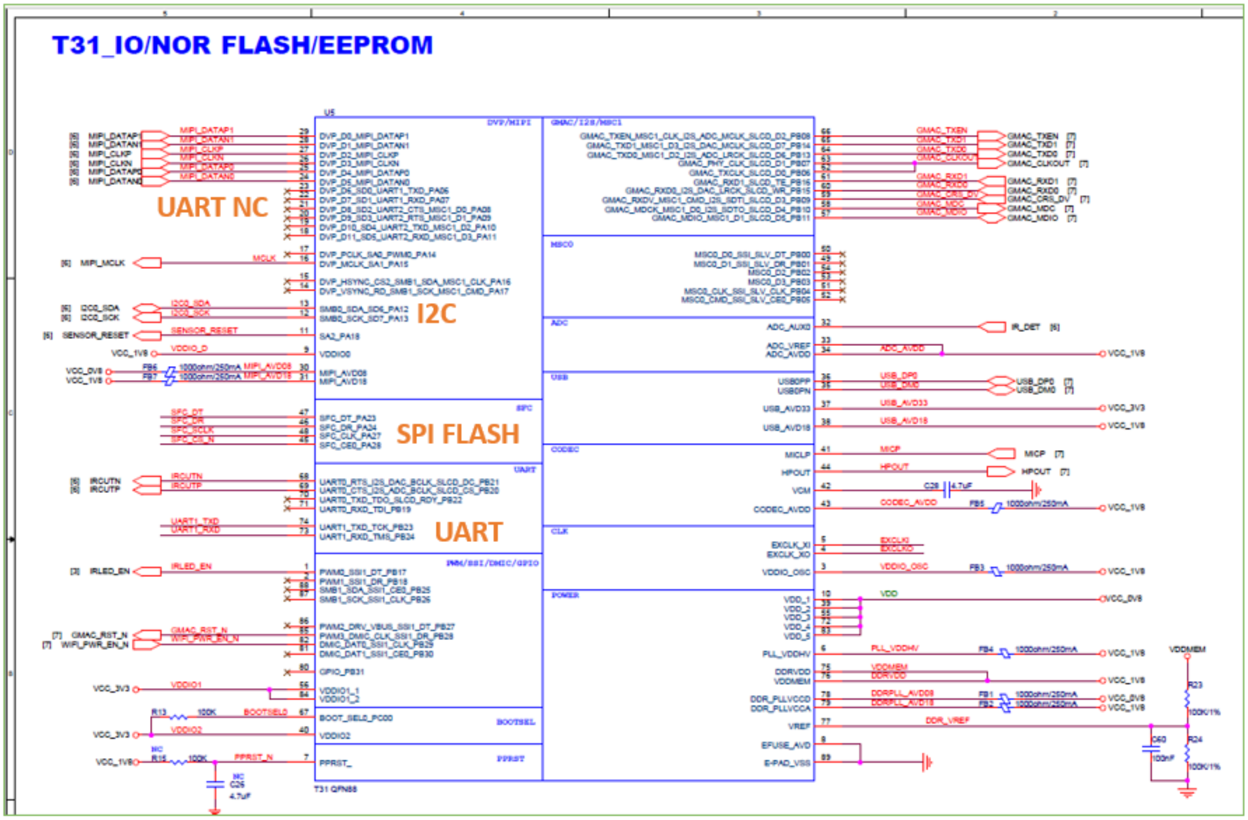


Figure A-9: T31 pinout assignments

Of particular note are pins 73 and 74, which lead to through-holes on the PCB. The other interfaces (except JTAG) are assigned pinouts, but don't lead to through-holes and therefore require a little more work to access. This is shown in Figure A-10.

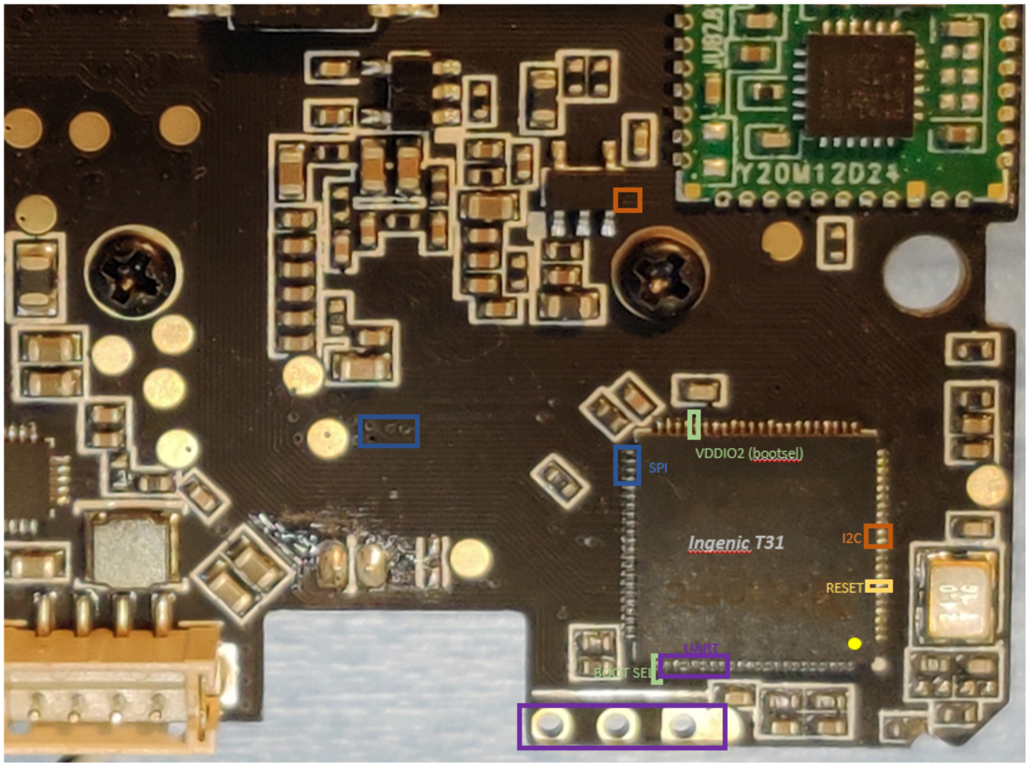


Figure A-10: T31 pin assignments which may be externally probed

The pins of note are color coded, and were visually traced to the color coded through-holes (in the case of the UART pins), or via's (as in the case of the SPI pins). The Reset and Boot Select pins are noted (for possible future use) but not traced. The UART through-holes are outlined in purple at the bottom of Figure A-10. The SPI and I2C pins lead to vias on the PCB.

Using this diagram and [24], the data flows were traced to the different components on the PCB as shown in Figure A-10.

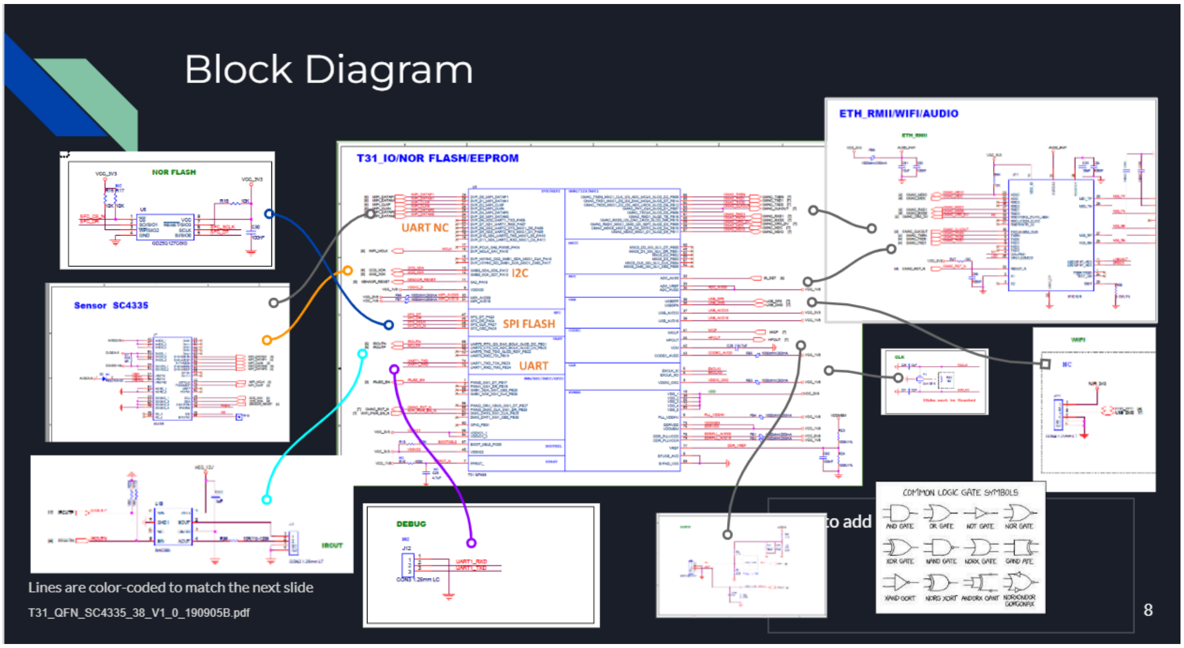


Figure A-11: Wyze Cam v3 Block Diagram

This diagram can be easily translated to the physical components as shown in Figure A-12. Figure A-12 provides the legend for the color codes. The dashed lines identify assumed communications paths and have not been verified. Verification and validation of these traces are left for future work. Although a JTAG interface is identified in the programmer's manual [22], it is not present on the schematic. I'm assuming that is because it is used internally to the T31 SoC, but this too should be verified.

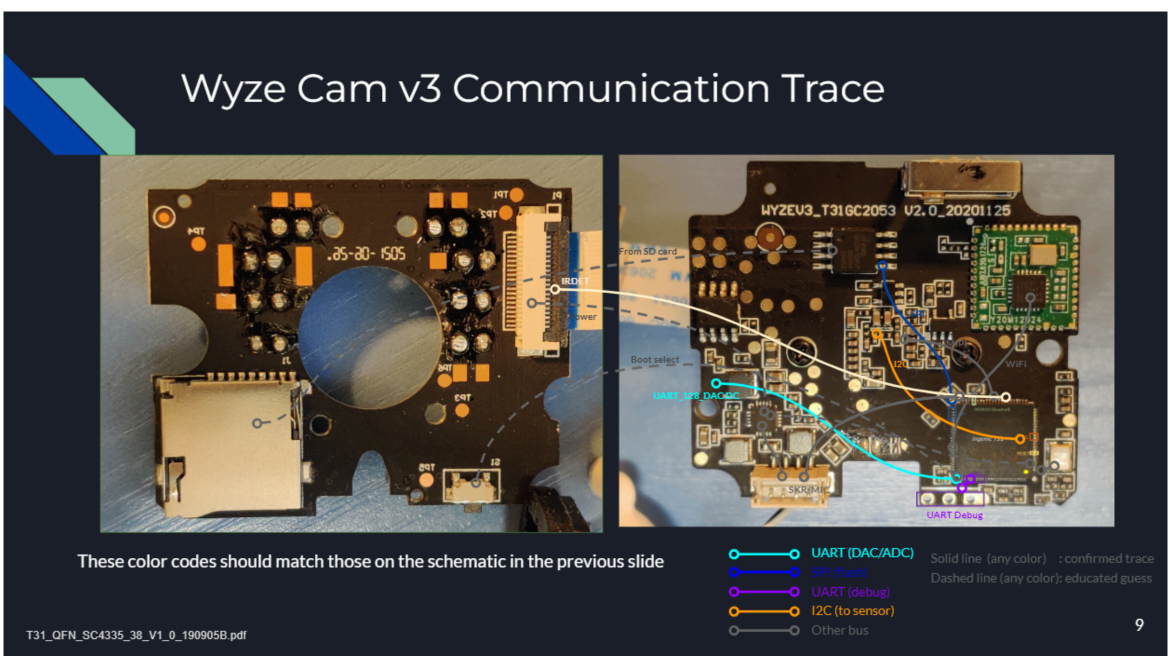


Figure A-12: Pinout traces from the T31 to other locations

XBurst1

The XBurst1 core CPU is based on the MIPS 32 bit revision 1 (MIPS32 Release 3) Reduced Instruction Set Computer (RISC) architecture. It has a 9-stage pipeline. It has 32 registers, each 32 bits wide. The Data Cache (D-Cache) and Instruction Cache (I-Cache) are each 32KB in size, which implies a Harvard architecture. It has a Memory Management Unit (MMU) that is 32 bits wide, supports page sizes of 4KB to 16MB for any entry, and can address 4GB of address space [22]. The MIPS DSP ASE Revision 2, MIPS MT ASE, SmartMIPS ASE, MIPS DSP Extension and trace logic are not implemented. Vectored inputs are implemented [22].

Additional details for the T-series processors can be found in Figure A-13.

Features	JZ4750	JZ4760	JZ4770 JZ4775 X1000	JZ4780	M200	T-series
MIPS32-R1 ISA	Yes	Yes	Yes	Yes	Yes	Yes
MIPS32-R2 Integer Instructions	No	Yes	Yes	Yes	Yes	Yes
MIPS32-R2 Floating point ISA	No	Yes	Yes	Yes	Yes	Yes
Ingenic MXU1	Yes	Yes	Yes	Yes	Yes	No
Ingenic MXU2	No	No	No	No	No	Yes
L1 I-cache	16kB	16kB	16kB	32kB	32kB	32kB
L1 D-cache	16kB	16kB	16kB	32kB	32kB	32kB
L2 cache (unified cache)	No	No	256kB[1)]	512kB[1)]	512kB[2)]	ref-soc
Ingenic PMON	No	Yes	Yes	Yes	Yes	Yes
CoreSceduler (for MP-cores)	No	No	No	Yes	Yes	No
SMP support	No	No	No	Yes	No	No
Big-Little cores support	No	No	No	No	Yes	No
CP0.ErrCtl.WST	No	No	Ye	Yes	Yes	No

Notes:

1) 128-byte cache line, 4-way set association, WT only
2) 32-byte cache line, 8-way set association, WT & WB

Figure A-13: Ingenic processor specific notes [22]

Registers

The XBurst1 has 32 registers, each 32-bits wide.

Six kernel scratch registers are used for temporary storage of information and implemented at register 2,3,4,5,6 and 7. CP0 Register 15, Select 0, contains the company ID, processor ID, and revision [22].

The CPU number is identified in CP0 Register 15, Select 1. CP0 Register 12, Select 0, contains the operating mode of the CPU (kernel or user) and coprocessor information. Supervisor mode is not implemented [22].

Debug registers are CP0 Register 23 Select 0 and Select 6. Debug exception and save information is included in CP0 Register 24 Select 0 and 31 Select 0 [22].

Memory Management Unit

The XBurst1 contains an on-chip MMU which performs address translation. The MMU is 32 bits wide, supports page sizes of 4KB to 16MB for any entry, and can address 4GB of address space. A virtual memory map is shown in Figure A-14. User space (kuseg) is from 0x0000 0000 to 0x7FFF FFFF. This virtual address space may or may not be identical to the physical address space, depending on the status of the configuration registers. When kuseg does address a virtual space, the address is extended by an 8-bit ASID field to form a unique virtual address. kseg0 and kseg1 translated from virtual to physical by subtracting 0x8000 0000 or 0xA000 0000 from the virtual address. In kernel mode, the first three bits of the address determine which kseg is selected [22].

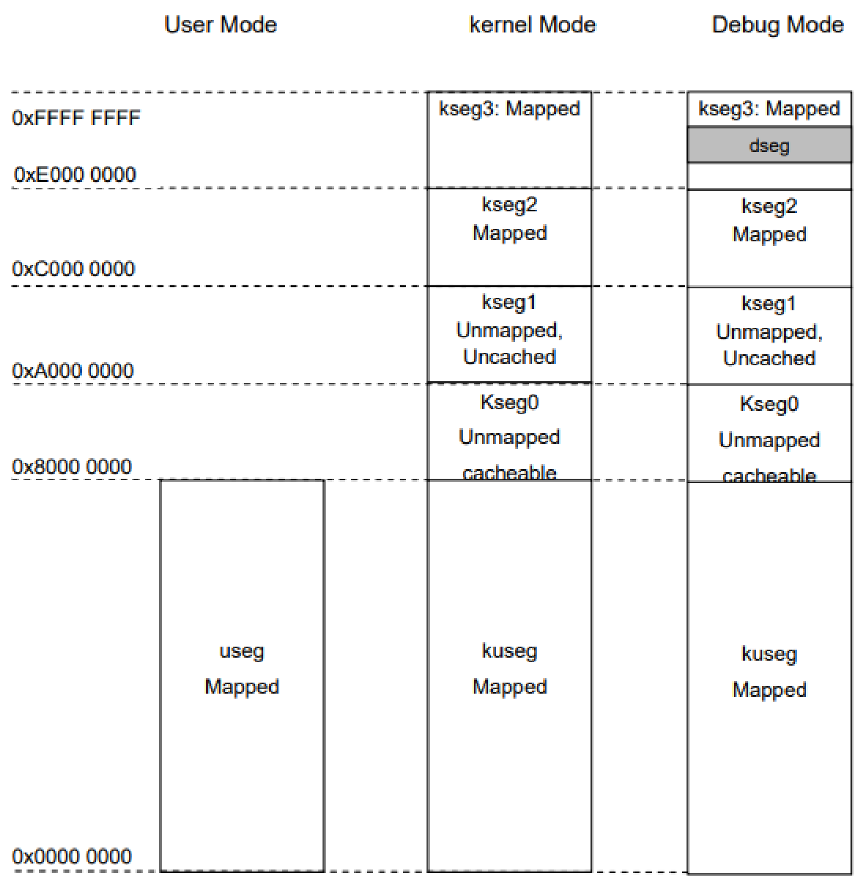


Figure A-14: Virtual Memory Map [22]

A mapping of the u-boot and kernel space was created from the u-boot output, binwalk output, and Ingenic T31 documentation. This memory map is shown in Figure A-15. It is incomplete, however, and should be updated to include peripherals and other missing information. The Linux Entry points (there are two shown) need to be resolved; one was observed during u-boot, the other came from binwalk [22].

end (upper) address reserved for U-boot	0x8400 0000
start (lower) address reserved for U-boot (U-boot executable code)	0x83F9 0000 (436 kbytes)
Global data	0x81F8 EF64 (124 bytes)
Board info	0x81F8 EFE0 (32 bytes)
Heap	0x81F8 F000 (32772 kbytes)
Stack	0x81F6 EF48 (2276 bytes)
Boot parameters	0x81F6 E664 (128 kbytes)
Lower address of Linux kernel stored in flash	0x8060 0000
Squashfs (top)	0x8056 0000
Squashfs (bottom)	0x8052 0000
Linux entry point (from uImage header, see binwalk output)	0x8041 6900
Start (lower address) of onboard U-boot image stored in flash (from u-boot output)	0x8001 0000
Top of useg/ kuseg	0x8000 0000

User-defined physical RAM map	0x0000 0000 to 0x0600 0000 (600 MB)
End of "determined" usable RAM after init	0x005B 0000 (additional 770kB)
Start of "determined" usable RAM after init	0x0057 1000
unidentified	0x0056 1000 to 0x0057 1000
End of "determined" usable physical RAM	0x0056 1000 (25 MB)
Start of "determined" usable physical RAM	0x0001 0000

Figure A-15: U-boot and Linux Kernel Memory Map (incomplete)

JTAG

JTAG operates in either MIPS or ACC mode. Mapped/unmapped address space details can be found in [22] for the debug modes [22].

Instruction Set Architecture (ISA)

The XBurst1 is based off of the MIPS32 revision 1 (MIPS 32 Release 3) architecture. It implements the MIPS32 instruction set to address the need by video, graphical, image, and signal processing. It also uses SIMD extensions. The XBurst ISA is called the MIPS extension/enhanced Unit2 (MXU2). It supports 8, 16, 32, and 64 bit signed and unsigned integers; 32 bit single precision and 64 bit double precision floating points. It uses 32 general purpose registers, vr0 through vr31, each 128 bits wide, and two control registers (MIP and MCSR). It allows operations on byte, halfword, word, doubleword, and vector sizes. The instruction format, in general, is [23]

Instruction vrd, vrs [, vrt]

Where:
vrd is the destination register
vrs is the source register / operand 1
vrt is operand 2

Secondary Board Analysis

The secondary board consists of the infrared Light Emitting Diodes (LED), Secure Digital (SD) card reader, switch, six test points, and 20-pin cable connector. The most interesting thing about this board are the six test points, visibly labeled TP1 through TP6. An attempt was made to solder wires to this board, which damaged the PCB. Particularly, the solder unintentionally bled over to a pad next to TP6. A continuity test showed they were the same point. So I cut/scraped the solder between TP6 and the square pad hoping to break the connection. I achieved my goal. I realized that I damaged the board when under test it wasn't behaving as expected (multiple resets in a never-ending loop). When I bought a new camera and performed a continuity test between TP6 and the square pad next to it, I realized they are the same point by design. The soldering performed is shown in Figure A-16.

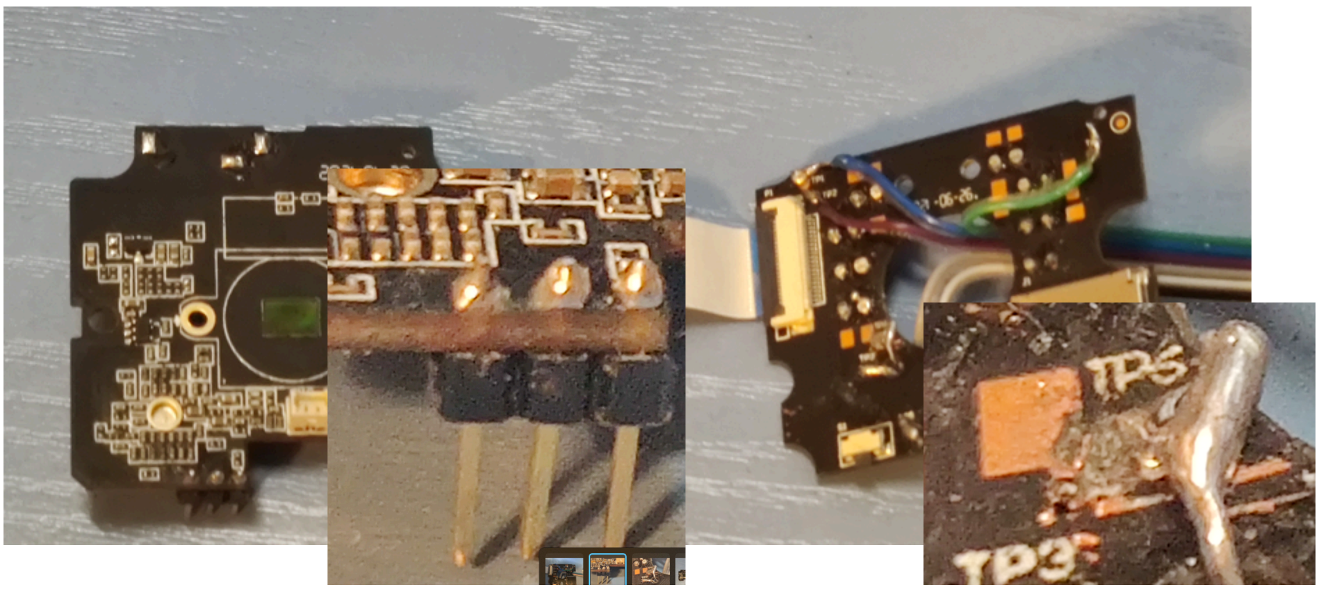


Figure A-16: Soldering the test points on boards 1 and 2

Other mistakes with this board, contributing to rendering it unusable, were: the 20-pin cable connecting the boards was crimped due to rough handling; while under test, the GND pin on the primary board (the outermost pin) was connected to +3.3V. The pin assignments, from left to right as in Figure A-16, are: GND, transmit, receive. For the TP# points, TP5 is confirmed GND (via continuity test with the WiFi antenna on the primary board).

UART Access

UART access is available by the three through-holes located at the bottom of the primary board. The test setup used JTAGULATOR as a means to both identify the TXD and RXD pins, and as a UART passthrough allowing a serial connection to the Wyze camera. The test setup is shown in Figure A-17. Clips were used instead of soldered connections.

Walking through the process, the first step is to identify the ground connections on both the unit under test (UUT) (which is the Wyze camera) and JTAGULATOR. A continuity test identified the GND through-hole by touching one probe to the through-hole and the other probe to the WiFi antenna on the UUT. A black clip was connected from the GND through-hole on the UUT to the GND pin on the JTAGULATOR. Another continuity test was performed, this time one probe touching a GND pin on the JTAGULATOR, and the other probe touching the WiFi antenna. Continuity was confirmed. A yellow clip was attached to the middle through-hole and to the channel 1 (ch1) pin on the JTAGULATOR. A red clip was attached to the innermost through-hole, and then connected to ch0 on the JTAGULATOR. Careful not to connect either the red or yellow wire to the VDJ pin (just about the GND pin) on JTAGULATOR or the board may become damaged. Finally, test that the yellow and red connectors are not grounded.

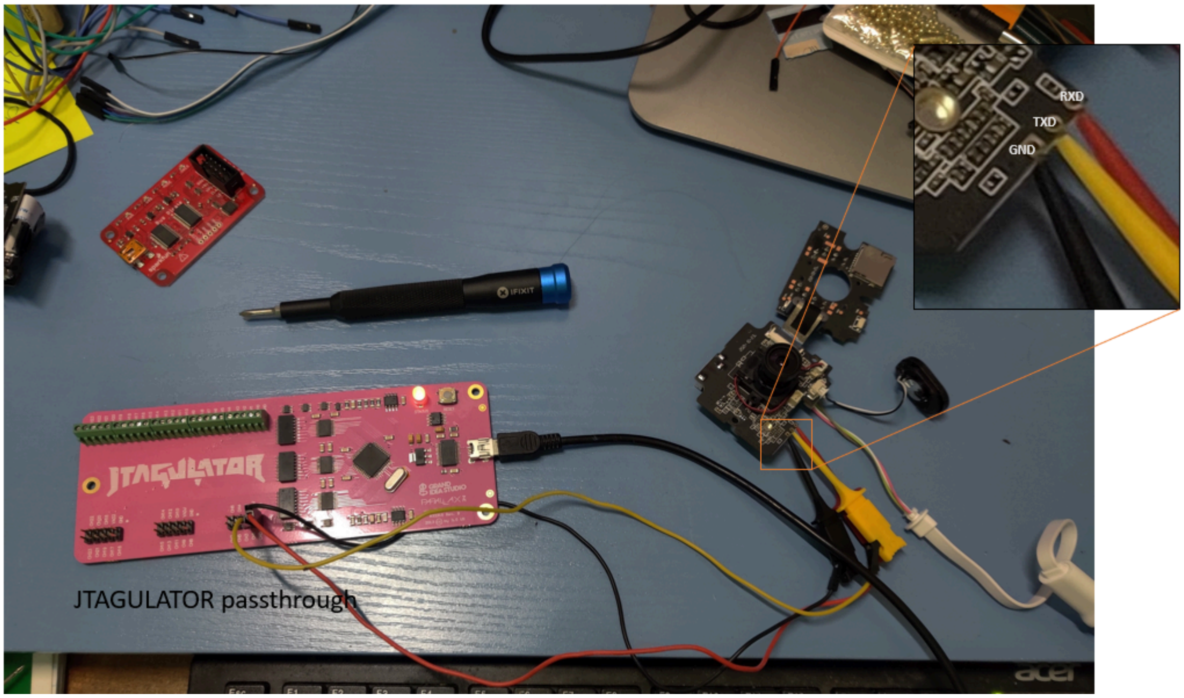


Figure A-17: Setup for UART access

First connect the UUT to power using the white USB connector. Then connect the JTAGULATOR to power using the built-in USB port. This completes the physical connections.

Now spin up your VM. Disconnect power from the UUT (or JTAGULATOR) and reconnect. A pop-up window like the one shown in Figure A-18 should appear. Select your VM from the list and close the pop-up.

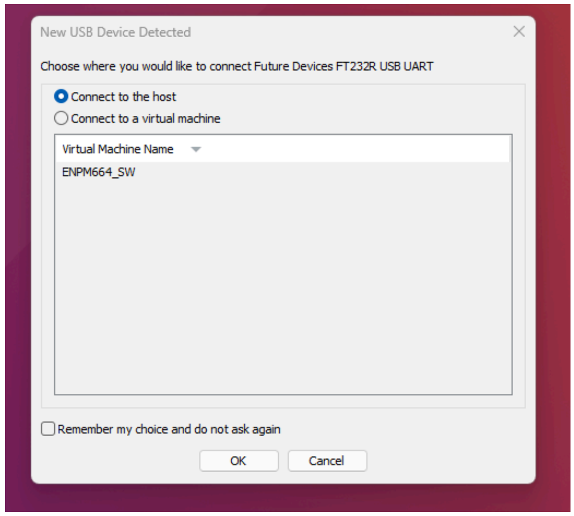


Figure A-18: Connection pop-up for the JTAGULATOR connection

If the pop-up disappears after 10 seconds or so, you can either repeat the disconnect/connect procedure described above or (if using VMWare) goto the VM drop-down menu, select “Removable Devices”, then “Future Devices FTR232R USB UART”, and then “Connect (Disconnect from Host) as in Figure A-19. This will connect the JTAGULATOR to your VM.

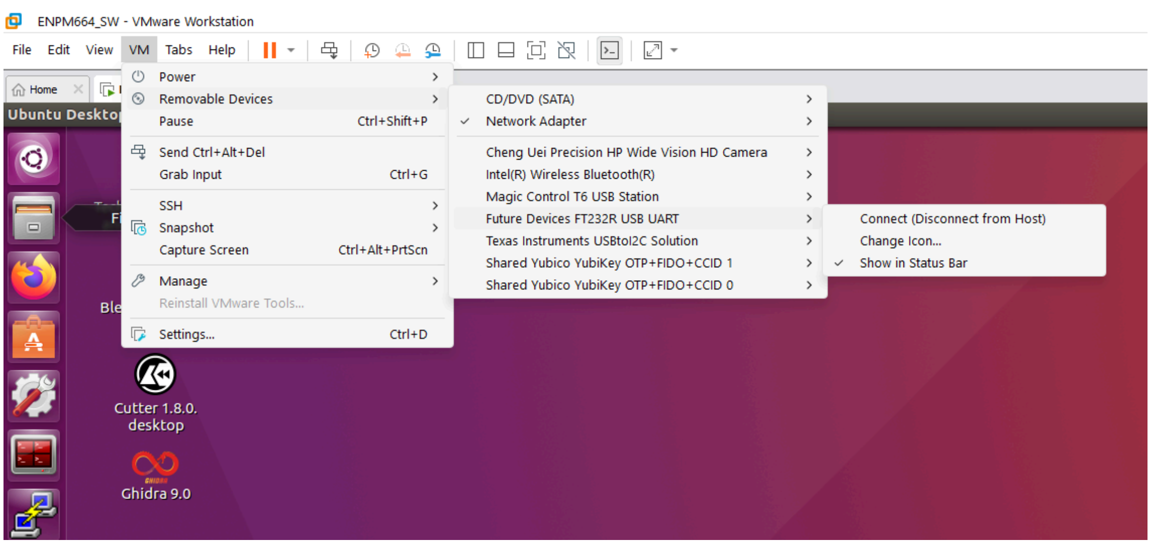


Figure A-19: Connecting JTAGULATOR to your VM (Ubuntu VM on VMWare shown)

Next we need to establish a serial connection to the JTAGULATOR. To do so, either connect to using Putty software, or from the command line using screen. But first we must identify the communications port on which the VM is connected to the JTAGULATOR. To do so, open a terminal on your VM and type “dmesg | grep tty”. If the JTAGULATOR is connected to the VM, we should see it in the Linux response as in Figure A-20.

```
esslp@ubuntu:~$ dmesg | grep tty
[    0.004000] console [tty0] enabled
[   35.191514] usb 2-2.1: FTDI USB Serial Device converter now attached to ttyUSB0
```

Figure A-20: Finding the COM port of the JTAGULATOR.

If the JTAGULATOR becomes disconnect, either intentionally or through some other means, the response to the “dmesg | grep tty” will include multiple “connected” and “disconnected” messages with timestamps. In the example of Figure A-21, the last timestamp at 70314.004053 confirms that the FTDI USB serial device is connected to ttyUSB0.

```
esslp@ubuntu:~$ dmesg | grep tty
[    0.004000] console [tty0] enabled
[   35.191514] usb 2-2.1: FTDI USB Serial Device converter now attached to ttyUSB0
[  392.958707] ftdi_sio ttyUSB0: usb_serial_generic_read_bulk_callback - urb stopped: -32
[  392.980615] ftdi_sio ttyUSB0: error from flowcontrol urb
[  392.980758] ftdi_sio ttyUSB0: FTDI USB Serial Device converter now disconnected from ttyUSB0
[ 2469.276297] usb 2-2.1: FTDI USB Serial Device converter now attached to ttyUSB0
[ 3241.046750] ftdi_sio ttyUSB0: usb_serial_generic_read_bulk_callback - urb stopped: -32
[ 3241.074141] ftdi_sio ttyUSB0: error from flowcontrol urb
[ 3241.074819] ftdi_sio ttyUSB0: FTDI USB Serial Device converter now disconnected from ttyUSB0
[69823.724198] usb 2-2.1: FTDI USB Serial Device converter now attached to ttyUSB0
[69833.415164] ftdi_sio ttyUSB0: FTDI USB Serial Device converter now disconnected from ttyUSB0
[70134.004053] usb 2-2.1: FTDI USB Serial Device converter now attached to ttyUSB0
```

Figure A-21: Example output from “dmesg | grep tty”

The connection should also be listed in the /dev directory, as shown in Figure A-22.

```
esslp@ubuntu:~$ ls -l /dev | grep tty | tail -n10
crw-rw----  1 root   4,  67 May  8 18:49 ttyS3
crw-rw----  1 root   4,  94 May  8 18:49 ttyS30
crw-rw----  1 root   4,  95 May  8 18:49 ttyS31
crw-rw----  1 root   4,  68 May  8 18:49 ttyS4
crw-rw----  1 root   4,  69 May  8 18:49 ttyS5
crw-rw----  1 root   4,  70 May  8 18:49 ttyS6
crw-rw----  1 root   4,  71 May  8 18:49 ttyS7
crw-rw----  1 root   4,  72 May  8 18:49 ttyS8
crw-rw----  1 root   4,  73 May  8 18:49 ttyS9
crw-rw----+ 1 root 188,   0 May  9 15:04 ttyUSB0
esslp@ubuntu:~$
```

Figure A-22: Inspecting the /dev directory on Linux

Once the communications port is identified, the next step is to establish a serial connection. I used Putty, but you can use the tool of your choice. Use a connection speed of 115200 baud and enter the connection port into the “Serial line” text box. Select the “serial” radio button. Then click “Open”. These settings are shown in Figure A-23.

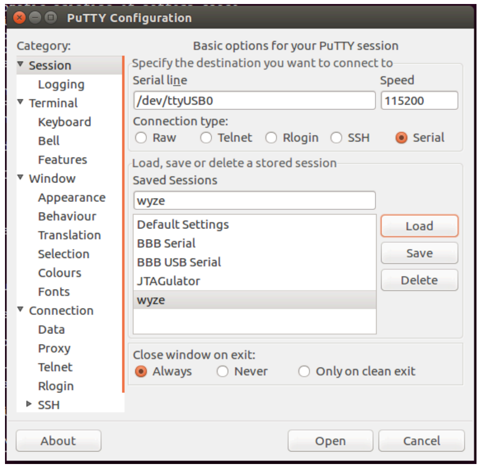


Figure A-24: Serial connection settings

The terminal window should look like something similar to the top of Figure A-25. Type ‘h’ for help.

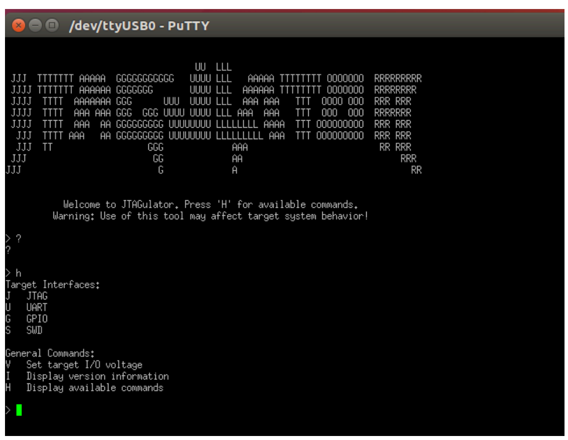


Figure A-25: JTAGULATOR terminal

Type ‘U’ for UART. Again, type ‘h’ for help. To set the voltage, type ‘v’. Set the voltage to 3.3V by typing ‘3.3’ and then enter. JTAGULATOR will warn you that VADJ pins on the PCB should not be used for this configuration. To identify the TXD and RXD pins, type ‘u’ then enter. Enter 0 for the starting channel and 1 for the ending channel. No pins are known, so type ‘N’ or leave the answer to “Are any pins already known?” as default. JTAGULATOR is letting us know it will test two permutations: TXD on through-hole 0 and RXD on through-hole one, then TCD on through-hole 1 and RXD on through-hole zero. The next prompt asks for a text string. Leave this blank by typing enter. Leave the delay as 10ms (or enter 10 if it is not already set). Leave “ignore non-printable characters?” to the default of No. And then press the spacebar to start. These entries are shown in Figure A-26.

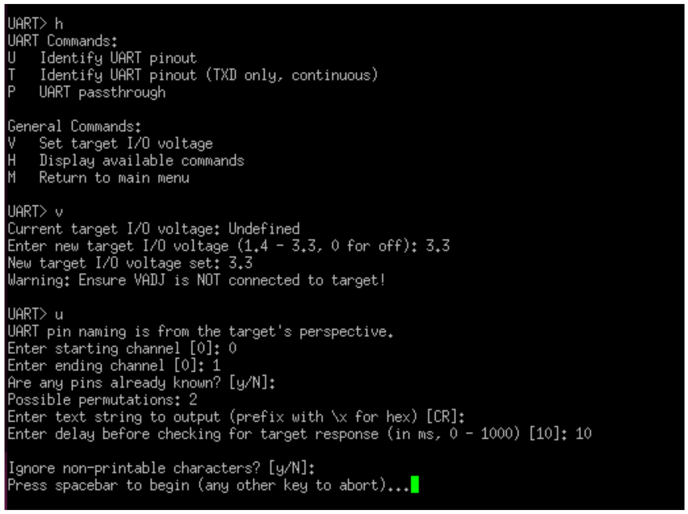


Figure A-26: JTAGULATOR entries for identifying the transmit and receive through-holes

The response should look similar to Figure A-27. The correct configuration is the longest set of data. In this example the TXD through-hole is associated with the yellow wire (the middle through-hole) and RXD is associated with the innermost through-hole (the red wire) as shown in Figure A-17.

Next, type ‘p’ and then enter. The terminal will prompt you for the TXD pin, RXD pin, and baud rate. The baud rate should be set to 115200 and set the “enable local echo?” to ‘n’. Press enter twice. “WCVC login: “ should be shown on the terminal, indicating a successful pass-through connection from the VM to the camera.

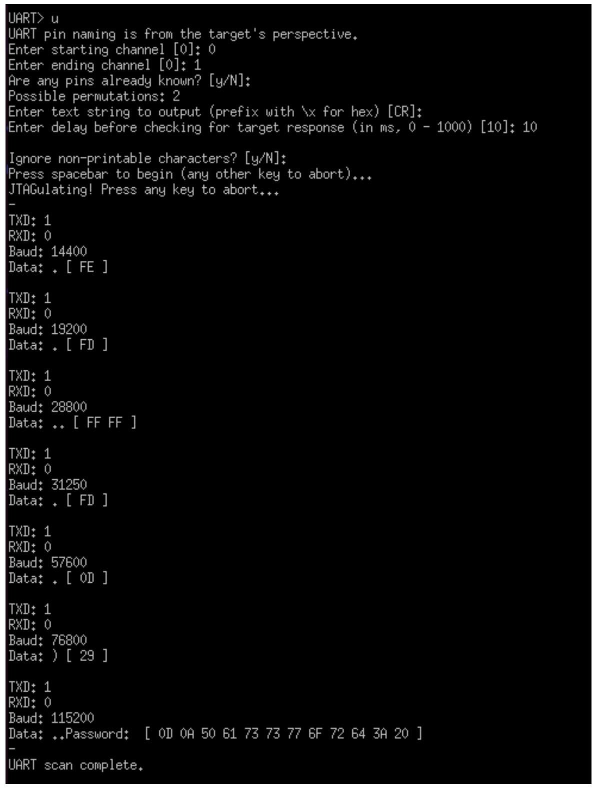


Figure A-27: JTAGULATOR output

Analysis of the U-boot Output

After having established a serial connection with the camera by leveraging the UART interface, power-on (or disconnect and reconnect power to) the camera. The camera will output data to the terminal during the boot process similar to Figure A-28. This information was used to inform the memory map of Figure A-15.

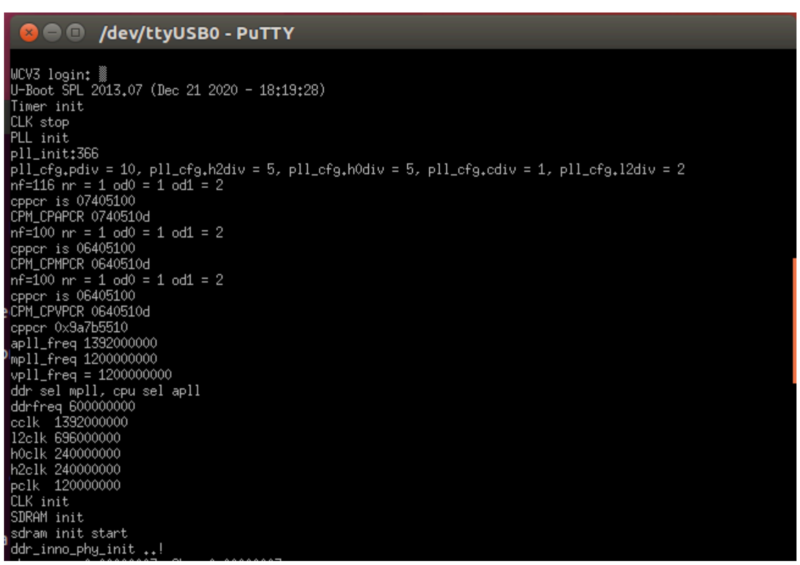


Figure A-28: Memory test portion of the U-boot output

The U-boot version, SPL 2013.07 (Dec 21 2020 - 18:19:28), is shown in the first line of Figure A-28. Figure A-29 shows the rest of the U-boot output during the memory test.

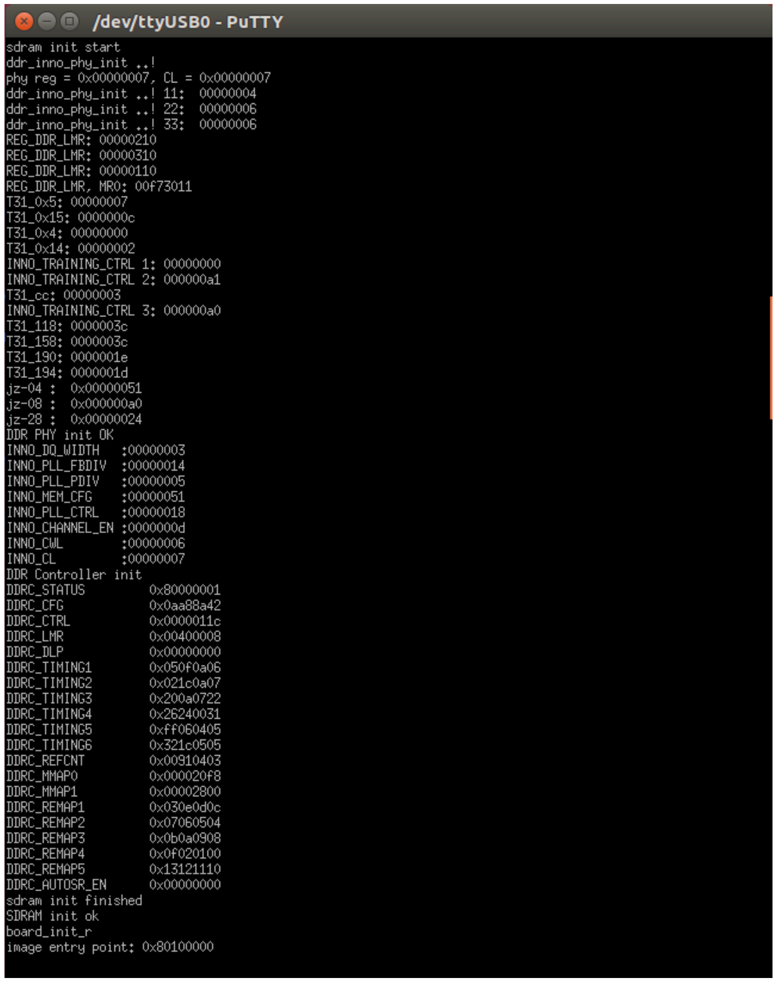
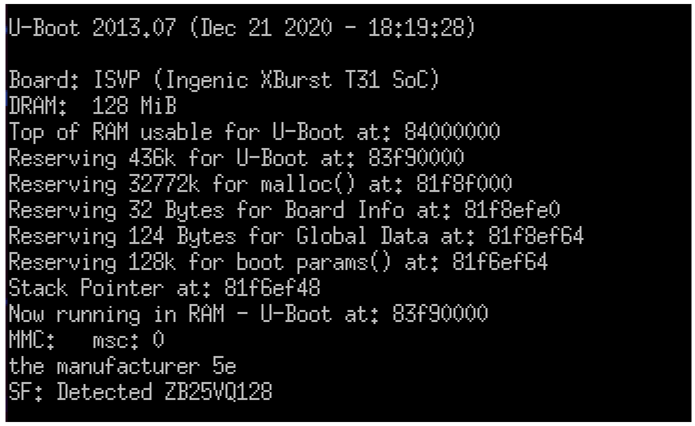


Figure A-29: U-boot memory test

After the memory test, the processor type is identified as the T31. This presumably identifies when the T31 XBurst1 core is booted and configured. Virtual memory addresses are shown in Figure A-3. The stack pointer is initialized to 0x81f6_ef48. Memory is reserved for U-boot from 0x83f9_0000 to 0x8400_0000. The “image entry point”, as shown in Figure A-29, presumably represents start of the U-boot image that will execute next from the onboard flash memory.



```
U-Boot 2013.07 (Dec 21 2020 - 18:19:28)

Board: ISVP (Ingenic XBurst T31 SoC)
DRAM:  128 MiB
Top of RAM usable for U-Boot at: 84000000
Reserving 436k for U-Boot at: 83f90000
Reserving 32772k for malloc() at: 81f8f000
Reserving 32 Bytes for Board Info at: 81f8efe0
Reserving 124 Bytes for Global Data at: 81f8ef64
Reserving 128k for boot params() at: 81f6ef64
Stack Pointer at: 81f6ef48
Now running in RAM - U-Boot at: 83f90000
MMC:   msc: 0
the manufacturer 5e
SF: Detected ZB25VQ128
```

Figure A-30: U-boot image memory allocations

Next the GPIO assignments are listed, as shown in Figure A-31. This is also where the SD card, if inserted, would be recognized. If a suitable binary is on the SD card, named demo_wcv3.bin, then the flash process will begin.

```
In:    serial
Out:   serial
Err:   serial
misc_init_r before change the wifi_enable_gpio
gpio_request lable = wifi_enable_gpio gpio = 57
misc_init_r after gpio_request the wifi_enable_gpio ret is 57
misc_init_r after change the wifi_enable_gpio ret is 0
misc_init_r before change the yellow_gpio
gpio_request lable = yellow_gpio gpio = 38
misc_init_r after gpio_request the yellow_gpio ret is 38
misc_init_r after change the yellow_gpio ret is 0
misc_init_r before change the blue_gpio
gpio_request lable = blue_gpio gpio = 39
misc_init_r after gpio_request the blue_gpio ret is 39
misc_init_r after change the blue_gpio ret is 1
gpio_request lable = night_gpio gpio = 49
misc_init_r after gpio_request the night_gpio ret is 49
misc_init_r after change the night_gpio ret is 0
gpio_request lable = 850_light_gpio gpio = 47
misc_init_r after gpio_request the 850_light_gpio ret is 47
misc_init_r after change the 850_light_gpio ret is 0
gpio_request lable = SPK_able_gpio gpio = 63
misc_init_r after gpio_request the SPK_able_gpio ret is 63
misc_init_r after change the SPK_able_gpio ret is 0
gpio_request lable = TF_en_gpio gpio = 50
misc_init_r after gpio_request the TF_en_gpio ret is 50
misc_init_r after change the TF_en_gpio ret is 0
gpio_request lable = TF_cd_gpio gpio = 59
misc_init_r after gpio_request the TF_cd_gpio ret is 59
misc_init_r after change the TF_cd_gpio ret is 0
gpio_request lable = SD_able_gpio gpio = 48
misc_init_r after gpio_request the SD_able_gpio ret is 48
misc_init_r after change the SD_able_gpio ret is 0
misc_init_r before change the wifi_enable_gpio
gpio_request lable = wifi_enable_gpio gpio = 57
misc_init_r after gpio_request the wifi_enable_gpio ret is 57
misc_init_r after change the wifi_enable_gpio ret is 1
Hit any key to stop autoboot:  0
Card did not respond to voltage select!
SD card is not insert
gpio_request lable = sdupgrade gpio = 51
the manufacturer 5e
SF: Detected ZB25VQ128

The upgrade flag could not be found!
the manufacturer 5e
SF: Detected ZB25VQ128
```

Figure A-31: GPIO allocations

After this, the Linux kernel is booted. The top few lines of this output are shown in Figure A-32. Shown are the architecture (MIPS), location of the onboard (“legacy”) image at 0x8060_0000, the Linux kernel version, size of the kernel, and the address of the kernel 0x8041_6900. The “entry point” is presumably

```
## Booting kernel from Legacy Image at 80600000 ...
   Image Name:   Linux-3.10.14__isvp_swan_1.0__
   Image Type:   MIPS Linux Kernel Image (lzma compressed)
   Data Size:    1897077 Bytes = 1.8 MiB
   Load Address: 80010000
   Entry Point:  80416900
   Verifying Checksum ... OK
   Uncompressing Kernel Image ... OK
```

Figure A-32: Start of boot for Linux kernel

The remaining U-boot output is shown in Figure A-33 through Figure A-37. There is future work needed to complete the memory map from the U-boot output, debugging, and other sources. Figure A-15 shows the current memory map.

```
/dev/ttyUSB0 - PuTTY
Starting kernel ...

[    0.000000] Initializing cgroup subsys cpu
[    0.000000] Initializing cgroup subsys cpuacct
[    0.000000] Linux version 3.10.14__isvp_swan_1.0__ (xiao@xiao-virtual-machine) (gcc version 4.7.2 (Ingen
ic r2.3.3 2016.12) ) #19 PREEMPT Fri Jul 2 20:31:54 CST 2021
[    0.000000] bootconsole [early0] enabled
[    0.000000] CPU0 RESET ERROR PC:74116E1D
[    0.000000] CPU0 revision is: 00d00100 (Ingenic Xburst)
[    0.000000] FPU revision is: 00b70000
[    0.000000] CCLK:1392MHz L2CLK:696Mhz H0CLK:200MHz H2CLK:200Mhz PCLK:100Mhz
[    0.000000] Determined physical RAM map:
[    0.000000]  memory: 00561000 @ 00010000 (usable)
[    0.000000]  memory: 0003f000 @ 00571000 (usable after init)
[    0.000000] User-defined physical RAM map:
[    0.000000]  memory: 06000000 @ 00000000 (usable)
[    0.000000] Zone ranges:
[    0.000000]   Normal   [mem 0x00000000-0x05ffffff]
[    0.000000] Movable zone start for each node
[    0.000000] Early memory node ranges
[    0.000000]   node   0: [mem 0x00000000-0x05ffffff]
[    0.000000] Primary instruction cache 32kB, 8-way, VIPT, linesize 32 bytes.
[    0.000000] Primary data cache 32kB, 8-way, VIPT, no aliases, linesize 32 bytes
[    0.000000] pls check processor_id[0x00d00100],sc_jz not support!
[    0.000000] MIPS secondary cache 128kB, 8-way, linesize 32 bytes.
[    0.000000] Built 1 zonelists in Zone order, mobility grouping off.  Total pages: 24384
[    0.000000] Kernel command line: console=ttyS1,115200n8 mem=96M@0x0 rmem=32M@0x6000000 init=/linuxrc roo
tfstype=squashfs root=/dev/mtdblock2 rw mtdparts=jz_sfc:256K(boot),1984K(kernel),3904K(rootfs),3904K(app),1
984K(kback),3904K(aback),384K(cfg),64K(para)
[    0.000000] PID hash table entries: 512 (order: -1, 2048 bytes)
[    0.000000] Dentry cache hash table entries: 16384 (order: 4, 65536 bytes)
[    0.000000] Inode-cache hash table entries: 8192 (order: 3, 32768 bytes)
[    0.000000] Memory: 90912k/98304k available (4158k kernel code, 7392k reserved, 1349k data, 252k init, 0
k highmem)
[    0.000000] SLUB: HWalign=32, Order=0-3, MinObjects=0, CPUs=1, Nodes=1
[    0.000000] Preemptible hierarchical RCU implementation.
[    0.000000] NR_IRQS:358
[    0.000000] clockevents_config_and_register success.
[    0.000014] Calibrating delay loop... 1391.00 BogoMIPS (lpj=6955008)
[    0.087831] pid_max: default: 32768 minimum: 301
[    0.092688] Mount-cache hash table entries: 512
[    0.097599] Initializing cgroup subsys debug
[    0.101854] Initializing cgroup subsys freezer
[    0.108074] regulator-dummy: no parameters
[    0.112259] NET: Registered protocol family 16
[    0.127949] bio: create slab <bio-0> at 0
[    0.133397] jz-dma jz-dma: JZ SoC DMA initialized
[    0.138470] SCSI subsystem initialized
[    0.142320] usbcore: registered new interface driver usbfs
[    0.147871] usbcore: registered new interface driver hub
[    0.153280] usbcore: registered new device driver usb
[    0.158483]  (null): set:249  hold:250 dev=100000000 h=500 l=500
[    0.164561] media: Linux media interface: v0.10
[    0.169138] Linux video capture interface: v2.00
[    0.173922] Advanced Linux Sound Architecture Driver Initialized.
[    0.181323] Switching to clocksource jz_clocksource
[    0.186254] cfg80211: Calling CRDA to update world regulatory domain
[    0.193117] jz-dwc2 jz-dwc2: cgu clk gate get error
[    0.198036] DWC IN OTG MODE
[    0.201440] dwc2 dwc2: Keep PHY ON
[    0.204820] dwc2 dwc2: Using Buffer DMA mode
[    0.209133] dwc2 dwc2: Core Release: 3.00a
[    0.213326] dwc2 dwc2: DesignWare USB2.0 High-Speed Host Controller
[    0.219651] dwc2 dwc2: new USB bus registered, assigned bus number 1
[    0.226724] hub 1-0:1.0: USB hub found
[    0.230456] hub 1-0:1.0: 1 port detected
[    0.234550] dwc2 dwc2: DWC2 Host Initialized
[    0.238961] NET: Registered protocol family 2
[    0.243735] TCP established hash table entries: 1024 (order: 1, 8192 bytes)
```

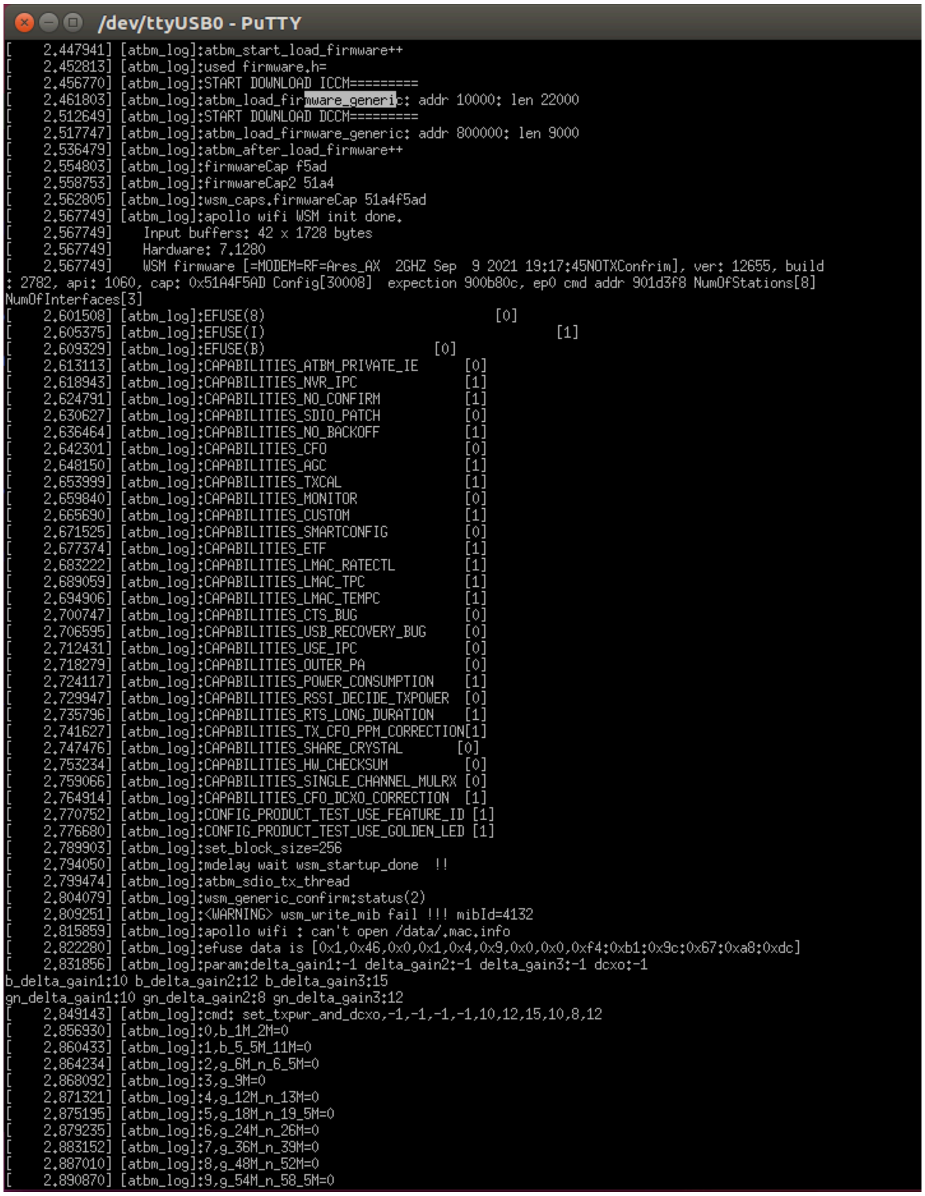
Figure A-33: Terminal output while the Linux kernel boot (1 of 5)

/dev/ttyUSB0 - PuTTY

```
[    0.230456] hub 1-0:1.0: 1 port detected
[    0.234550] dwc2 dwc2: DWC2 Host Initialized
[    0.238961] NET: Registered protocol family 2
[    0.243735] TCP established hash table entries: 1024 (order: 1, 8192 bytes)
[    0.250743] TCP bind hash table entries: 1024 (order: 0, 4096 bytes)
[    0.257216] TCP: Hash tables configured (established 1024 bind 1024)
[    0.263664] TCP: reno registered
[    0.266882] UDP hash table entries: 256 (order: 0, 4096 bytes)
[    0.272824] UDP-Lite hash table entries: 256 (order: 0, 4096 bytes)
[    0.279339] NET: Registered protocol family 1
[    0.283940] RPC: Registered named UNIX socket transport module.
[    0.289868] RPC: Registered udp transport module.
[    0.294685] RPC: Registered tcp transport module.
[    0.299403] RPC: Registered tcp NFSv4.1 backchannel transport module.
[    0.306270] freq_udelay_jiffys[0].max_num = 10
[    0.310693] cpufreq  udelay  loops_per_jiffy
[    0.315146] dwc2 dwc2: ID PIN CHANGED!
[    0.318953] 12000     59956   59956
[    0.322172] 24000     119913  119913
[    0.325628] 60000     299784  299784
[    0.329058] 120000    599569  599569
[    0.332590] 200000    999282  999282
[    0.336202] 300000    1498924         1498924
[    0.339830] 600000    2997848         2997848
[    0.343614] 792000    3957159         3957159
[    0.347247] 1008000   5036385         5036385
[    0.351043] 1200000   5995696         5995696
[    0.358478] squashfs: version 4.0 (2009/01/31) Phillip Lougher
[    0.365032] jffs2: version 2.2. © 2001-2006 Red Hat, Inc.
[    0.370768] msgmni has been set to 177
[    0.375481] io scheduler noop registered
[    0.379404] io scheduler cfq registered (default)
[    0.385314] jz-uart.1: ttyS1 at MMIO 0x10031000 (irq = 58) is a uart1
[    0.392916] console [ttyS1] enabled, bootconsole disabled
[    0.392916] console [ttyS1] enabled, bootconsole disabled
[    0.406546] brd: module loaded
[    0.411040] loop: module loaded
[    0.414820] zram: Created 2 device(s) ...
[    0.419007] logger: created 256K log 'log_main'
[    0.424254] jz TCU driver register completed
[    0.429016] the id code = 5e4018, the flash name is ZB25VQ128
[    0.435006] JZ SFC Controller for SFC channel 0 driver register
[    0.441124] 8 cmdlinepart partitions found on MTD device jz_sfc
[    0.447249] Creating 8 MTD partitions on "jz_sfc":
[    0.452191] 0x000000000000-0x000000040000 : "boot"
[    0.457536] 0x000000040000-0x000000230000 : "kernel"
[    0.463048] 0x000000230000-0x000000600000 : "rootfs"
[    0.468514] 0x000000600000-0x0000009d0000 : "app"
[    0.473768] 0x0000009d0000-0x000000bc0000 : "kback"
[    0.479150] 0x000000bc0000-0x000000f90000 : "aback"
[    0.484593] 0x000000f90000-0x000000ff0000 : "cfg"
[    0.489800] 0x000000ff0000-0x000001000000 : "para"
[    0.495137] SPI NOR MTD LOAD OK
[    0.498418] tun: Universal TUN/TAP device driver, 1.6
[    0.503658] tun: (C) 1999-2004 Max Krasnyansky <maxk@qualcomm.com>
[    0.510130] usbcore: registered new interface driver zd1201
[    0.515934] usbcore: registered new interface driver r8152
[    0.521628] usbcore: registered new interface driver asix
[    0.527289] usbcore: registered new interface driver usb-storage
[    0.533606] usbcore: registered new interface driver usbserial
[    0.539650] usbcore: registered new interface driver ch37x
[    0.545357] usbcore: registered new interface driver pl2303
[    0.551118] usbserial: USB Serial support registered for pl2303
[    0.557266] usbcore: registered new interface driver emi26 - firmware loader
[    0.564776] jzmmc_v1.2 jzmmc_v1.2.0: vmmc regulator missing
[    0.570808] jzmmc_v1.2 jzmmc_v1.2.0: register success!
[    0.576240] jzmmc_v1.2 jzmmc_v1.2.1: vmmc regulator missing
[    0.582162] jzmmc_v1.2 jzmmc_v1.2.1: register success!
[    0.587638] hidraw: raw HID events driver (C) Jiri Kosina
```

Figure A-34: Terminal output while the Linux kernel boot (2 of 5)

Figure A-35: Terminal output while the Linux kernel boot (3 of 5)



```
/dev/ttyUSB0 - PuTTY
[    2.447941] [atbm_log]:atbm_start_load_firmware++
[    2.452813] [atbm_log]:used firmware.h=
[    2.456770] [atbm_log]:START DOWNLOAD ICCM=========
[    2.461803] [atbm_log]:atbm_load_firmware_generic: addr 10000: len 22000
[    2.512649] [atbm_log]:START DOWNLOAD DCCM=========
[    2.517747] [atbm_log]:atbm_load_firmware_generic: addr 800000: len 9000
[    2.536479] [atbm_log]:atbm_after_load_firmware++
[    2.554803] [atbm_log]:firmwareCap f5ad
[    2.558753] [atbm_log]:firmwareCap2 51a4
[    2.562805] [atbm_log]:wsm_caps.firmwareCap 51a4f5ad
[    2.567749] [atbm_log]:apollo wifi WSM init done.
[    2.567749]    Input buffers: 42 x 1728 bytes
[    2.567749]    Hardware: 7.1280
[    2.567749]    WSM firmware [=MODEM=RF=Ares_AX  2GHZ Sep  9 2021 19:17:45NOTXConfrim], ver: 12655, build
: 2782, api: 1060, cap: 0x51A4F5AD Config[30008]  expection 900b80c, ep0 cmd addr 901d3f8 NumOfStations[8]
NumOfInterfaces[3]
[    2.601508] [atbm_log]:EFUSE(8)                          [0]
[    2.605375] [atbm_log]:EFUSE(I)                                  [1]
[    2.609329] [atbm_log]:EFUSE(B)                  [0]
[    2.613113] [atbm_log]:CAPABILITIES_ATBM_PRIVATE_IE      [0]
[    2.618943] [atbm_log]:CAPABILITIES_NVR_IPC              [1]
[    2.624791] [atbm_log]:CAPABILITIES_NO_CONFIRM           [1]
[    2.630627] [atbm_log]:CAPABILITIES_SDIO_PATCH           [0]
[    2.636464] [atbm_log]:CAPABILITIES_NO_BACKOFF           [1]
[    2.642301] [atbm_log]:CAPABILITIES_CFO                  [0]
[    2.648150] [atbm_log]:CAPABILITIES_AGC                  [1]
[    2.653999] [atbm_log]:CAPABILITIES_TXCAL                [1]
[    2.659840] [atbm_log]:CAPABILITIES_MONITOR              [0]
[    2.665690] [atbm_log]:CAPABILITIES_CUSTOM               [1]
[    2.671525] [atbm_log]:CAPABILITIES_SMARTCONFIG          [0]
[    2.677374] [atbm_log]:CAPABILITIES_ETF                  [1]
[    2.683222] [atbm_log]:CAPABILITIES_LMAC_RATECTL         [1]
[    2.689059] [atbm_log]:CAPABILITIES_LMAC_TPC             [1]
[    2.694906] [atbm_log]:CAPABILITIES_LMAC_TEMPC           [1]
[    2.700747] [atbm_log]:CAPABILITIES_CTS_BUG              [0]
[    2.706595] [atbm_log]:CAPABILITIES_USB_RECOVERY_BUG     [0]
[    2.712431] [atbm_log]:CAPABILITIES_USE_IPC              [0]
[    2.718279] [atbm_log]:CAPABILITIES_OUTER_PA             [0]
[    2.724117] [atbm_log]:CAPABILITIES_POWER_CONSUMPTION    [1]
[    2.729947] [atbm_log]:CAPABILITIES_RSSI_DECIDE_TXPOWER  [0]
[    2.735796] [atbm_log]:CAPABILITIES_RTS_LONG_DURATION    [1]
[    2.741627] [atbm_log]:CAPABILITIES_TX_CFO_PPM_CORRECTION[1]
[    2.747476] [atbm_log]:CAPABILITIES_SHARE_CRYSTAL       [0]
[    2.753234] [atbm_log]:CAPABILITIES_HW_CHECKSUM          [0]
[    2.759066] [atbm_log]:CAPABILITIES_SINGLE_CHANNEL_MULRX [0]
[    2.764914] [atbm_log]:CAPABILITIES_CFO_DCXO_CORRECTION  [1]
[    2.770752] [atbm_log]:CONFIG_PRODUCT_TEST_USE_FEATURE_ID [1]
[    2.776680] [atbm_log]:CONFIG_PRODUCT_TEST_USE_GOLDEN_LED [1]
[    2.789903] [atbm_log]:set_block_size=256
[    2.794050] [atbm_log]:mdelay wait wsm_startup_done  !!
[    2.799474] [atbm_log]:atbm_sdio_tx_thread
[    2.804079] [atbm_log]:wsm_generic_confirm:status(2)
[    2.809251] [atbm_log]:<WARNING> wsm_write_mib fail !!! mibId=4132
[    2.815859] [atbm_log]:apollo wifi : can't open /data/.mac.info
[    2.822280] [atbm_log]:efuse data is [0x1,0x46,0x0,0x1,0x4,0x9,0x0,0x0,0xf4:0xb1:0x9c:0x67:0xa8:0xdc]
[    2.831856] [atbm_log]:param:delta_gain1:-1 delta_gain2:-1 delta_gain3:-1 dcxo:-1
b_delta_gain1:10 b_delta_gain2:12 b_delta_gain3:15
gn_delta_gain1:10 gn_delta_gain2:8 gn_delta_gain3:12
[    2.849143] [atbm_log]:cmd: set_txpwr_and_dcxo,-1,-1,-1,-1,10,12,15,10,8,12
[    2.856930] [atbm_log]:0,b_1M_2M=0
[    2.860433] [atbm_log]:1,b_5_5M_11M=0
[    2.864234] [atbm_log]:2,g_6M_n_6_5M=0
[    2.868092] [atbm_log]:3,g_9M=0
[    2.871321] [atbm_log]:4,g_12M_n_13M=0
[    2.875195] [atbm_log]:5,g_18M_n_19_5M=0
[    2.879235] [atbm_log]:6,g_24M_n_26M=0
[    2.883152] [atbm_log]:7,g_36M_n_39M=0
[    2.887010] [atbm_log]:8,g_48M_n_52M=0
[    2.890870] [atbm_log]:9,g_54M_n_58_5M=0
```

Figure A-36: Terminal output while the Linux kernel boot (4 of 5)

Figure A-37: Terminal output while the Linux kernel boot (5 of 5)

Hardware Emulation

The hardware was emulated using Firmadyne software. Firmadyne uses QEMU to emulate the underlying hardware. This walkthrough won't describe the process of installing and configuring the Firmadyne software on your VM.

The commands entered for a successful emulation were modeled after the website's "Usage" section available at [24]. The figures that follow will support the step-by-step procedure below.

The binary must first be extracted from the camera firmware zip file. The zip file is available on Wyze's website. Note the directory where the commands are being executed. It's recommended to execute these commands from the Firmadyne home directory as defined in the configuration file. To extract the binary, type the command at the top of Figure A-38.

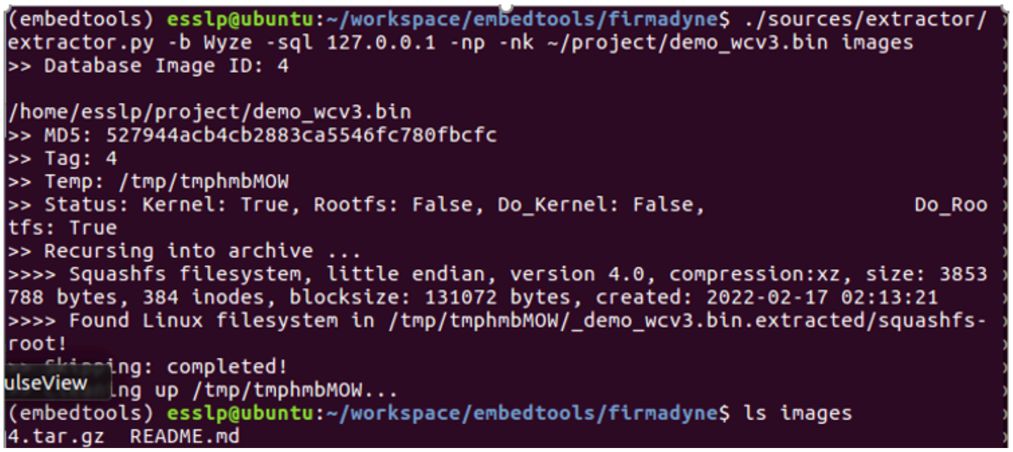


Figure A-38: Extracting the Wyze cam v3 binary

At the end of the extraction process, a compressed tarball should have been created under the images folder. Note the Database Image ID. You will be using it later in other commands. The '-b' flag can be any string that represents the brand name of the device. The '-np' and '-nk' flags represent no kernel and no parallel operation. The command seemed to work, so we moved on and didn't question the flags. Observe how in Figure A-38 the file system is identified as Squashfs and little endian.

The next two commands identify the architecture and store in the SQL database the value and other select information from the firmware. These two commands are shown in Figure A-39. Notice that a password is requested. If the installation instructions were followed, it should be "firmadyne".

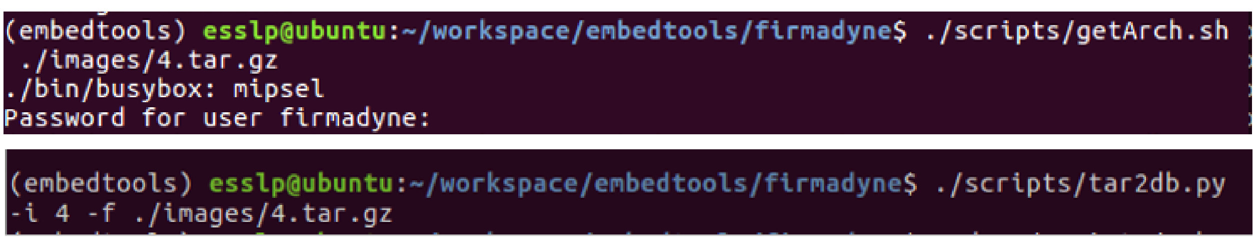


Figure A-39: Get the architecture of the firmware

Type the next command as shown in FigureA-40. This will create the QEMU image.

```
(embedtools) esslp@ubuntu:~/workspace/embedtools/firmadyne$ sudo ./scripts/makeI
mage.sh 4
Querying database for architecture... Password for user firmadyne:
mipsel
----Running----
----Copying Filesystem Tarball----
----Creating QEMU Image----
Formatting '/home/esslp/workspace/embedtools/firmadyne//scratch//4//image.raw',
fmt=raw size=1073741824
----Creating Partition Table----

Welcome to fdisk (util-linux 2.27.1).
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.

Device does not contain a recognized partition table.
Created a new DOS disklabel with disk identifier 0xfa4e1074.

Command (m for help): Created a new DOS disklabel with disk identifier 0xd2d9a6e
3.

Command (m for help): Partition type
   p   primary (0 primary, 0 extended, 4 free)
   e   extended (container for logical partitions)
Select (default p): Partition number (1-4, default 1): First sector (2048-209715
1, default 2048): Last sector, +sectors or +size{K,M,G,T,P} (2048-2097151, defau
lt 2097151):
Created a new partition 1 of type 'Linux' and of size 1023 MiB.

Command (m for help): The partition table has been altered.
Syncing disks.

----Mounting QEMU Image----
add map loop0p1 (253:0): 0 2095104 linear 7:0 2048
xt Editor ting Filesystem----
          .42.13 (17-May-2015)
Discarding device blocks: done
Creating filesystem with 261888 4k blocks and 65536 inodes
Filesystem UUID: a010f049-caf1-4d79-82e8-d5f5512406f3
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376
```

```
----Making QEMU Image Mountpoint----
----Mounting QEMU Image Partition 1----
----Extracting Filesystem Tarball----
----Creating FIRMADYNE Directories----
----Patching Filesystem (chroot)----
Creating /etc/TZ!
Warning: Recreating device nodes!
Removing /etc/scripts/sys_resetbutton!
----Setting up FIRMADYNE----
----Unmounting QEMU Image----
umount: /home/esslp/workspace/embedtools/firmadyne/scratch/4/image: target is bu
sy
        (In some cases useful info about processes that
         use the device is found by lsof(8) or fuser(1).)
```

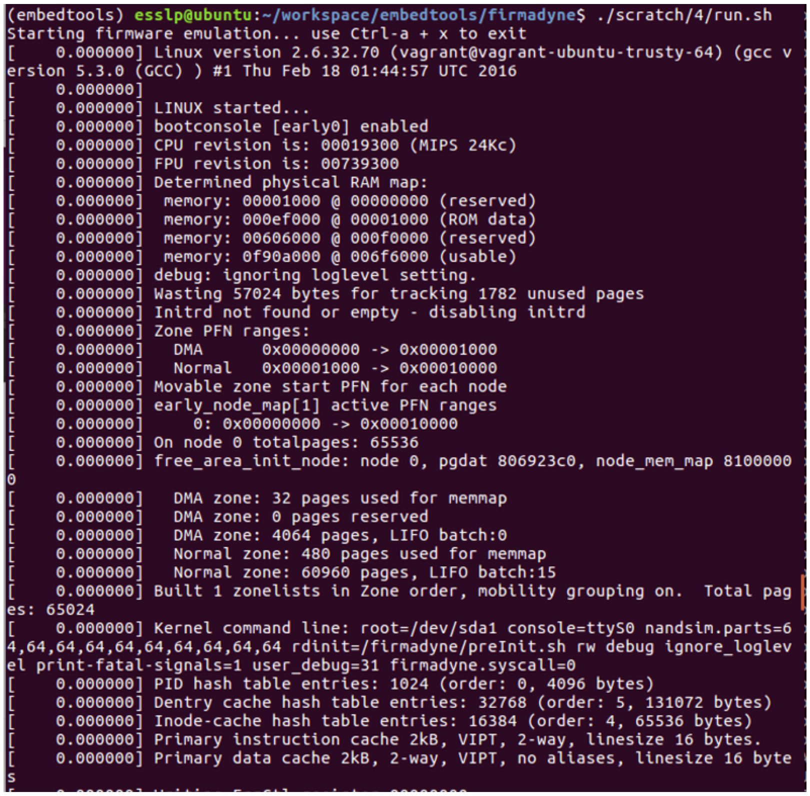
Figure A-41: Create the QEMU image

One last setup command gathers network information and saves it to the database. See Figure A-42.

```
(embedtools) esslp@ubuntu:~/workspace/embedtools/firmadyne$ ./scripts/inferNetwo
rk.sh 4
Querying database for architecture... Password for user firmadyne:
mipsel
Running firmware 4: terminating after 60 secs...
qemu-system-mipsel: terminating on signal 2 from pid 14647
Inferring network...
Interfaces: []
Done!
```

Figure A-42: Gather network information

If the setup was successful, the next command should run the emulation. Your output should be similar to that in Figures A-43 through A-49.



```
(embedtools) esslp@ubuntu:~/workspace/embedtools/firmadyne$ ./scratch/4/run.sh
Starting firmware emulation... use Ctrl-a + x to exit
[    0.000000] Linux version 2.6.32.70 (vagrant@vagrant-ubuntu-trusty-64) (gcc v
ersion 5.3.0 (GCC) ) #1 Thu Feb 18 01:44:57 UTC 2016
[    0.000000]
[    0.000000] LINUX started...
[    0.000000] bootconsole [early0] enabled
[    0.000000] CPU revision is: 00019300 (MIPS 24Kc)
[    0.000000] FPU revision is: 00739300
[    0.000000] Determined physical RAM map:
[    0.000000]  memory: 00001000 @ 00000000 (reserved)
[    0.000000]  memory: 000ef000 @ 00001000 (ROM data)
[    0.000000]  memory: 00606000 @ 000f0000 (reserved)
[    0.000000]  memory: 0f90a000 @ 006f6000 (usable)
[    0.000000] debug: ignoring loglevel setting.
[    0.000000] Wasting 57024 bytes for tracking 1782 unused pages
[    0.000000] Initrd not found or empty - disabling initrd
[    0.000000] Zone PFN ranges:
[    0.000000]   DMA      0x00000000 -> 0x00001000
[    0.000000]   Normal   0x00001000 -> 0x00010000
[    0.000000] Movable zone start PFN for each node
[    0.000000] early_node_map[1] active PFN ranges
[    0.000000]     0: 0x00000000 -> 0x00010000
[    0.000000] On node 0 totalpages: 65536
[    0.000000] free_area_init_node: node 0, pgdat 806923c0, node_mem_map 8100000
0
[    0.000000]   DMA zone: 32 pages used for memmap
[    0.000000]   DMA zone: 0 pages reserved
[    0.000000]   DMA zone: 4064 pages, LIFO batch:0
[    0.000000]   Normal zone: 480 pages used for memmap
[    0.000000]   Normal zone: 60960 pages, LIFO batch:15
[    0.000000] Built 1 zonelists in Zone order, mobility grouping on.  Total pag
es: 65024
[    0.000000] Kernel command line: root=/dev/sda1 console=ttyS0 nandsim.parts=6
4,64,64,64,64,64,64,64,64,64 rdinit=/firmadyne/preInit.sh rw debug ignore_loglev
el print-fatal-signals=1 user_debug=31 firmadyne.syscall=0
[    0.000000] PID hash table entries: 1024 (order: 0, 4096 bytes)
[    0.000000] Dentry cache hash table entries: 32768 (order: 5, 131072 bytes)
[    0.000000] Inode-cache hash table entries: 16384 (order: 4, 65536 bytes)
[    0.000000] Primary instruction cache 2kB, VIPT, 2-way, linesize 16 bytes.
[    0.000000] Primary data cache 2kB, 2-way, VIPT, no aliases, linesize 16 byte
s
```

Figure A-43: Emulation of the firmware (1 of 7)

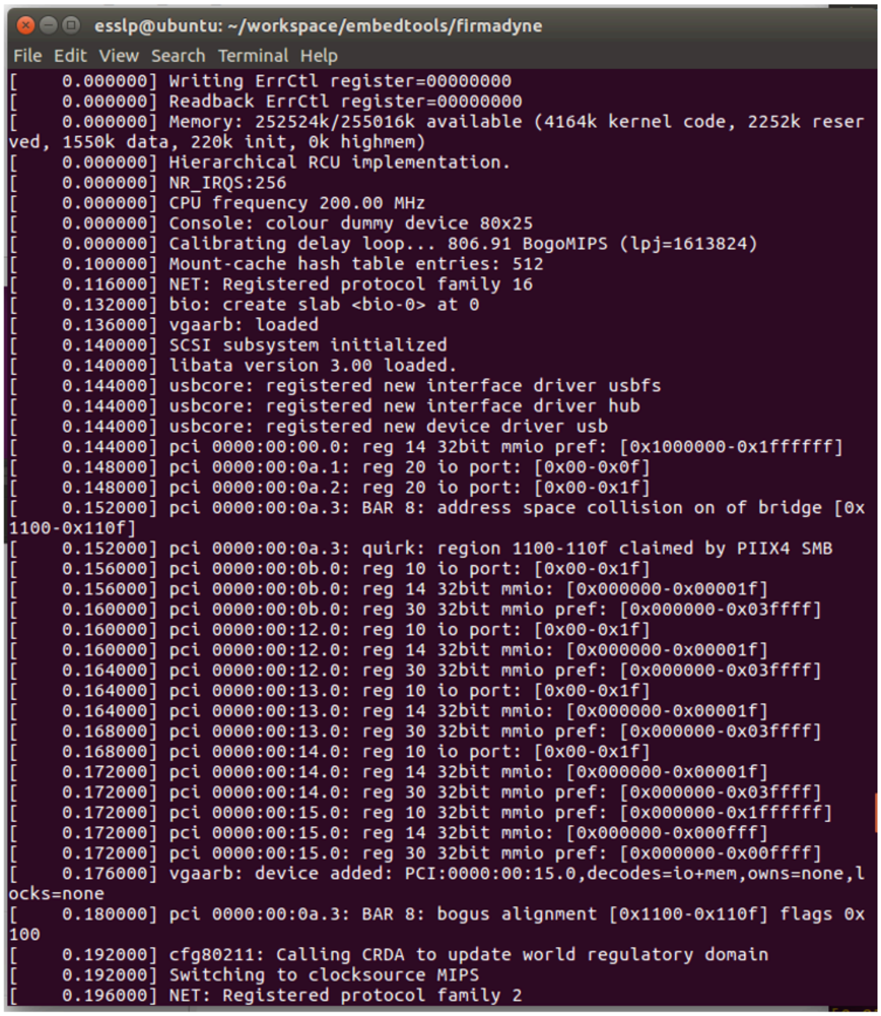
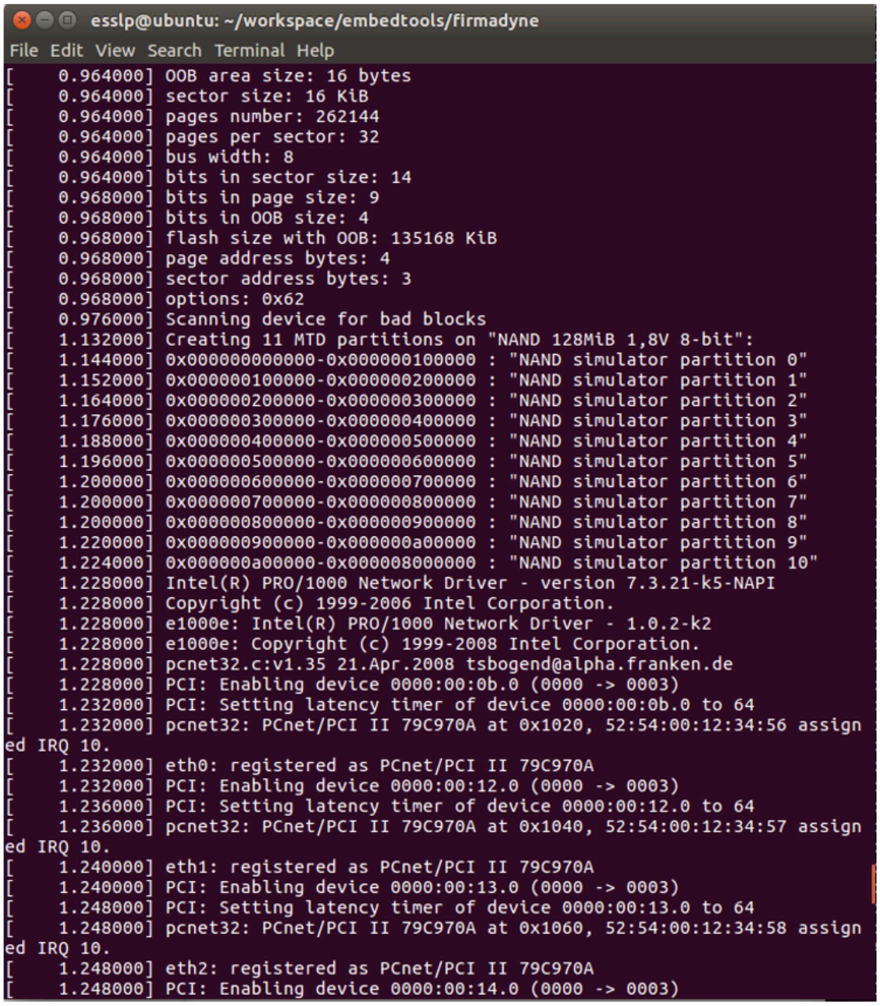

```
esslp@ubuntu: ~/workspace/embedtools/firmadyne
File Edit View Search Terminal Help
[    0.000000] Writing ErrCtl register=00000000
[    0.000000] Readback ErrCtl register=00000000
[    0.000000] Memory: 252524k/255016k available (4164k kernel code, 2252k reser
ved, 1550k data, 220k init, 0k highmem)
[    0.000000] Hierarchical RCU implementation.
[    0.000000] NR_IRQS:256
[    0.000000] CPU frequency 200.00 MHz
[    0.000000] Console: colour dummy device 80x25
[    0.000000] Calibrating delay loop... 806.91 BogoMIPS (lpj=1613824)
[    0.100000] Mount-cache hash table entries: 512
[    0.116000] NET: Registered protocol family 16
[    0.132000] bio: create slab <bio-0> at 0
[    0.136000] vgaarb: loaded
[    0.140000] SCSI subsystem initialized
[    0.140000] libata version 3.00 loaded.
[    0.144000] usbcore: registered new interface driver usbfs
[    0.144000] usbcore: registered new interface driver hub
[    0.144000] usbcore: registered new device driver usb
[    0.144000] pci 0000:00:00.0: reg 14 32bit mmio pref: [0x1000000-0x1ffffff]
[    0.148000] pci 0000:00:0a.1: reg 20 io port: [0x00-0x0f]
[    0.148000] pci 0000:00:0a.2: reg 20 io port: [0x00-0x1f]
[    0.152000] pci 0000:00:0a.3: BAR 8: address space collision on of bridge [0x
1100-0x110f]
[    0.152000] pci 0000:00:0a.3: quirk: region 1100-110f claimed by PIIX4 SMB
[    0.156000] pci 0000:00:0b.0: reg 10 io port: [0x00-0x1f]
[    0.156000] pci 0000:00:0b.0: reg 14 32bit mmio: [0x000000-0x00001f]
[    0.160000] pci 0000:00:0b.0: reg 30 32bit mmio pref: [0x000000-0x03ffff]
[    0.160000] pci 0000:00:12.0: reg 10 io port: [0x00-0x1f]
[    0.160000] pci 0000:00:12.0: reg 14 32bit mmio: [0x000000-0x00001f]
[    0.164000] pci 0000:00:12.0: reg 30 32bit mmio pref: [0x000000-0x03ffff]
[    0.164000] pci 0000:00:13.0: reg 10 io port: [0x00-0x1f]
[    0.164000] pci 0000:00:13.0: reg 14 32bit mmio: [0x000000-0x00001f]
[    0.168000] pci 0000:00:13.0: reg 30 32bit mmio pref: [0x000000-0x03ffff]
[    0.168000] pci 0000:00:14.0: reg 10 io port: [0x00-0x1f]
[    0.172000] pci 0000:00:14.0: reg 14 32bit mmio: [0x000000-0x00001f]
[    0.172000] pci 0000:00:14.0: reg 30 32bit mmio pref: [0x000000-0x03ffff]
[    0.172000] pci 0000:00:15.0: reg 10 32bit mmio pref: [0x000000-0x1ffffff]
[    0.172000] pci 0000:00:15.0: reg 14 32bit mmio: [0x000000-0x000fff]
[    0.172000] pci 0000:00:15.0: reg 30 32bit mmio pref: [0x000000-0x00ffff]
[    0.176000] vgaarb: device added: PCI:0000:00:15.0,decodes=io+mem,owns=none,l
ocks=none
[    0.180000] pci 0000:00:0a.3: BAR 8: bogus alignment [0x1100-0x110f] flags 0x
100
[    0.192000] cfg80211: Calling CRDA to update world regulatory domain
[    0.192000] Switching to clocksource MIPS
[    0.196000] NET: Registered protocol family 2
```

Figure A-44:Emulation of the firmware (2 of 7)

Figure A-45:Emulation of the firmware (3 of 7)



```
esslp@ubuntu: ~/workspace/embedtools/firmadyne
File Edit View Search Terminal Help
[    0.964000] OOB area size: 16 bytes
[    0.964000] sector size: 16 KiB
[    0.964000] pages number: 262144
[    0.964000] pages per sector: 32
[    0.964000] bus width: 8
[    0.964000] bits in sector size: 14
[    0.968000] bits in page size: 9
[    0.968000] bits in OOB size: 4
[    0.968000] flash size with OOB: 135168 KiB
[    0.968000] page address bytes: 4
[    0.968000] sector address bytes: 3
[    0.968000] options: 0x62
[    0.976000] Scanning device for bad blocks
[    1.132000] Creating 11 MTD partitions on "NAND 128MiB 1,8V 8-bit":
[    1.144000] 0x000000000000-0x000000100000 : "NAND simulator partition 0"
[    1.152000] 0x000000100000-0x000000200000 : "NAND simulator partition 1"
[    1.164000] 0x000000200000-0x000000300000 : "NAND simulator partition 2"
[    1.176000] 0x000000300000-0x000000400000 : "NAND simulator partition 3"
[    1.188000] 0x000000400000-0x000000500000 : "NAND simulator partition 4"
[    1.196000] 0x000000500000-0x000000600000 : "NAND simulator partition 5"
[    1.200000] 0x000000600000-0x000000700000 : "NAND simulator partition 6"
[    1.200000] 0x000000700000-0x000000800000 : "NAND simulator partition 7"
[    1.200000] 0x000000800000-0x000000900000 : "NAND simulator partition 8"
[    1.220000] 0x000000900000-0x000000a00000 : "NAND simulator partition 9"
[    1.224000] 0x000000a00000-0x000008000000 : "NAND simulator partition 10"
[    1.228000] Intel(R) PRO/1000 Network Driver - version 7.3.21-k5-NAPI
[    1.228000] Copyright (c) 1999-2006 Intel Corporation.
[    1.228000] e1000e: Intel(R) PRO/1000 Network Driver - 1.0.2-k2
[    1.228000] e1000e: Copyright (c) 1999-2008 Intel Corporation.
[    1.228000] pcnet32.c:v1.35 21.Apr.2008 tsbogend@alpha.franken.de
[    1.228000] PCI: Enabling device 0000:00:0b.0 (0000 -> 0003)
[    1.232000] PCI: Setting latency timer of device 0000:00:0b.0 to 64
[    1.232000] pcnet32: PCnet/PCI II 79C970A at 0x1020, 52:54:00:12:34:56 assign
ed IRQ 10.
[    1.232000] eth0: registered as PCnet/PCI II 79C970A
[    1.232000] PCI: Enabling device 0000:00:12.0 (0000 -> 0003)
[    1.236000] PCI: Setting latency timer of device 0000:00:12.0 to 64
[    1.236000] pcnet32: PCnet/PCI II 79C970A at 0x1040, 52:54:00:12:34:57 assign
ed IRQ 10.
[    1.240000] eth1: registered as PCnet/PCI II 79C970A
[    1.240000] PCI: Enabling device 0000:00:13.0 (0000 -> 0003)
[    1.248000] PCI: Setting latency timer of device 0000:00:13.0 to 64
[    1.248000] pcnet32: PCnet/PCI II 79C970A at 0x1060, 52:54:00:12:34:58 assign
ed IRQ 10.
[    1.248000] eth2: registered as PCnet/PCI II 79C970A
[    1.248000] PCI: Enabling device 0000:00:14.0 (0000 -> 0003)
```

Figure A-46:Emulation of the firmware (4 of 7)

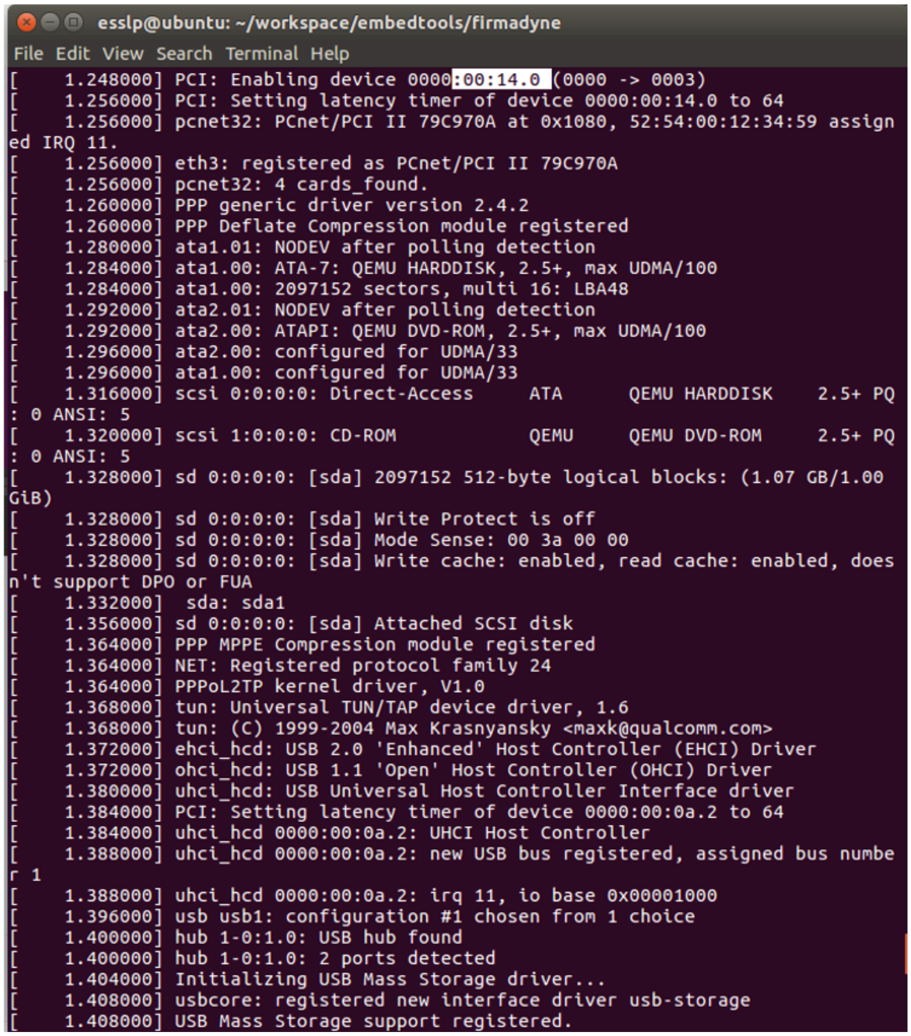
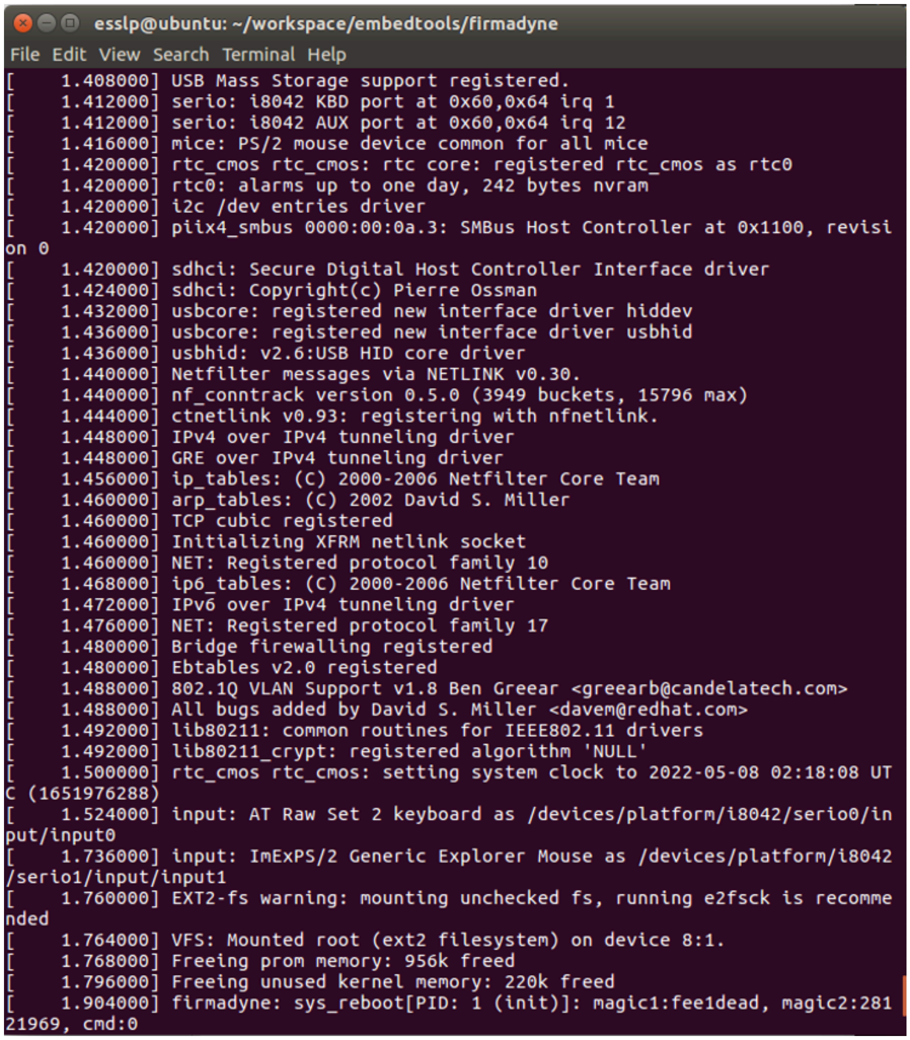


Figure A-47:Emulation of the firmware (5 of 7)



```
esslp@ubuntu: ~/workspace/embedtools/firmadyne
File Edit View Search Terminal Help
[    1.408000] USB Mass Storage support registered.
[    1.412000] serio: i8042 KBD port at 0x60,0x64 irq 1
[    1.412000] serio: i8042 AUX port at 0x60,0x64 irq 12
[    1.416000] mice: PS/2 mouse device common for all mice
[    1.420000] rtc_cmos rtc_cmos: rtc core: registered rtc_cmos as rtc0
[    1.420000] rtc0: alarms up to one day, 242 bytes nvram
[    1.420000] i2c /dev entries driver
[    1.420000] piix4_smbus 0000:00:0a.3: SMBus Host Controller at 0x1100, revisi
on 0
[    1.420000] sdhci: Secure Digital Host Controller Interface driver
[    1.424000] sdhci: Copyright(c) Pierre Ossman
[    1.432000] usbcore: registered new interface driver hiddev
[    1.436000] usbcore: registered new interface driver usbhid
[    1.436000] usbhid: v2.6:USB HID core driver
[    1.440000] Netfilter messages via NETLINK v0.30.
[    1.440000] nf_conntrack version 0.5.0 (3949 buckets, 15796 max)
[    1.444000] ctnetlink v0.93: registering with nfnetlink.
[    1.448000] IPv4 over IPv4 tunneling driver
[    1.448000] GRE over IPv4 tunneling driver
[    1.456000] ip_tables: (C) 2000-2006 Netfilter Core Team
[    1.460000] arp_tables: (C) 2002 David S. Miller
[    1.460000] TCP cubic registered
[    1.460000] Initializing XFRM netlink socket
[    1.460000] NET: Registered protocol family 10
[    1.468000] ip6_tables: (C) 2000-2006 Netfilter Core Team
[    1.472000] IPv6 over IPv4 tunneling driver
[    1.476000] NET: Registered protocol family 17
[    1.480000] Bridge firewalling registered
[    1.480000] Ebtables v2.0 registered
[    1.488000] 802.1Q VLAN Support v1.8 Ben Greear <greearb@candelatech.com>
[    1.488000] All bugs added by David S. Miller <davem@redhat.com>
[    1.492000] lib80211: common routines for IEEE802.11 drivers
[    1.492000] lib80211_crypt: registered algorithm 'NULL'
[    1.500000] rtc_cmos rtc_cmos: setting system clock to 2022-05-08 02:18:08 UT
C (1651976288)
[    1.524000] input: AT Raw Set 2 keyboard as /devices/platform/i8042/serio0/in
put/input0
[    1.736000] input: ImExPS/2 Generic Explorer Mouse as /devices/platform/i8042
/serio1/input/input1
[    1.760000] EXT2-fs warning: mounting unchecked fs, running e2fsck is recomme
nded
[    1.764000] VFS: Mounted root (ext2 filesystem) on device 8:1.
[    1.768000] Freeing prom memory: 956k freed
[    1.796000] Freeing unused kernel memory: 220k freed
[    1.904000] firmadyne: sys_reboot[PID: 1 (init)]: magic1:fee1dead, magic2:281
21969, cmd:0
```

Figure A-48: Emulation of the firmware (6 of 7)

```
[    2.016000] firmadyne: do_execve: /firmadyne/console
[    2.020000] OFFSETS: offset of pid: 0x100 offset of comm: 0x1f0
mdev is ok......

|_____2020_WYZE_CAM_V3_@HUALAI_____|

mount: mounting /dev/mtdblock3 on /system failed: Invalid argument

WCV3 login:
```

Figure A-49: Emulation of the firmware (7 of 7)

The emulation will mount the filesystem onto your VM. You may search the filesystem manually, and change any value you wish. Just use chmod to change the permissions first. This is what we did with the shadow file. The shadow file contains the password hash for each user. By changing the hash, we can control the firmware for debugging sessions and further analysis. This activity is left for future action. Figure 47 shows the command-line perl script used to create the hash (SHA512 with ‘wyzecam3’ as salt).

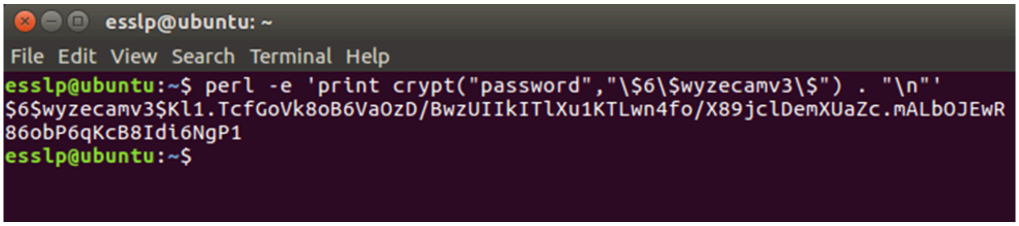


Figure A-50: Creating a password using SHA512

The output of the perl script, starting at $6 and ending at the newline (‘...P1’), was copied and inserted into the shadow file. The old root hash was retained and renamed ‘oldroot’. The shadow file was saved to its original location (Figure A-51).

Figure A-51: Updating the shadow file

For the new password to work, restart the emulation. Then type in ‘root’ for username, and your new password (we used ‘password’). We gained root access to the emulated firmware, shown in Figure A-52.

Figure A-52: Root access achieved in the emulation

This is as far as we got with the emulation. Future efforts to analyze the firmware using emulation are left for future action.

In the meantime, we've provided the hash to a program called hashcat. Hashcat is a password cracking utility freely available. We need to tell hashcat the hash format (1800 UNIX/SHA512) and a mode. The mode we chose was brute force with a rule set. The rule set developed was based on previous Wyze camera passwords that are publicly known. The three passwords are: 'WYom2020', 'WYom20200', and 'ismart12' for user root. We setup a ruleset which requires a lowercase or uppercase 'w', and another rule which requires the last two characters to be a number. We also told hashcat to specifically try some other characteristics, including: the year 2020 or 2021 on the end; the year 2020 or 2021 on the end followed by another number; and look for occurrences of 'v3' somewhere in the string.

The PC on which it is running has an AMD 1700 microprocessor (first generation Ryzen 7), with a separate Radeon RX5700XT graphics card (generation 5). The host operating system is openSuse 'Tumbleweed'. The amdgpu driver was installed, needed for hashcat to recognize the graphics card. Different versions of Ubuntu (Ubuntu 14, 16 and 18; the latest Kali version; the latest popOS). Also attempted was using a laptop, and leveraging the onboard discrete NVIDIA graphics. None of these configurations didn't work (the libraries required for the driver were either deprecated or not available to that version of Ubuntu), and the laptop overheated. Cloud GPUs were also considered, however they were too expensive. The amdgpu drivers are compatible with three flavors of Linux: Ubuntu, Red Hat, and OpenSuse. We downloaded the latest OpenSuse (Tumbleweed) operating system image and installed it. We then installed the amdgpu drivers. Even though some warning were issued during the install (again, deprecated libraries), hashcat recognized the GPU (type 'hashcat -I' into the command line)!

Thus far, hashcat has been running for more than 7 days. It has not yet cracked the password. Occasional crashes have occurred, however hashcat can be configured to resume where it left off.

Linux Kernel Static Source Code Analysis

We acquired the latest version of the T31 chip's SDK on Github. We also located the latest DLinux kernel source code from the WyzeCam website for the V3 camera and downloaded that as well (Figures B-1 and B-2).

Figure B-1: Ingenic T31 SDK

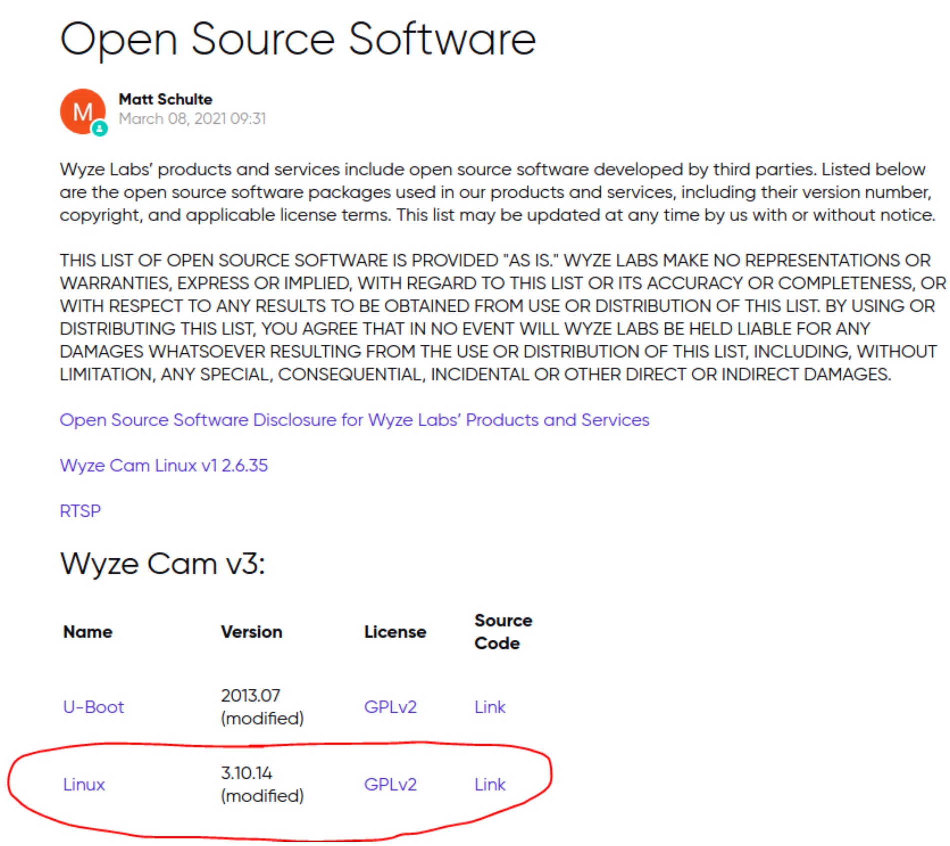


Figure B-2: WyzeCam V3 Source code

The WyzeCam source code indicated that the Linux Kernel version is 3.10.14 and the T31 Chip was determined to be 3.10.14 as well during the firmware analysis. We used the CVE Details (Figures 3-B and 4-B). data source website to look up all CVEs related to the 3.10.14 kernel. The site listed 48 potential CVEs for this kernel version, however we focused only on the most critical CVEs based on CVSS scores due to time constraints presented to us and the large amount of source code that needed to be inspected. The threshold for the cutoff was a CVSS score of 4.9 which still allowed us to inspect 17 CVEs.

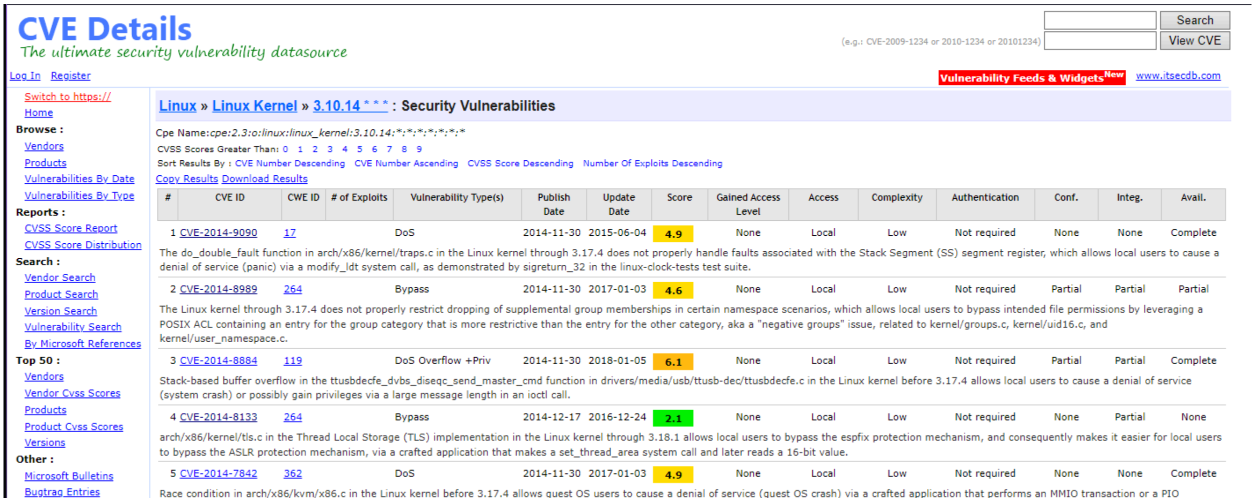


Figure B-3: CVE page filtered for Linux Kernel 3.10.14

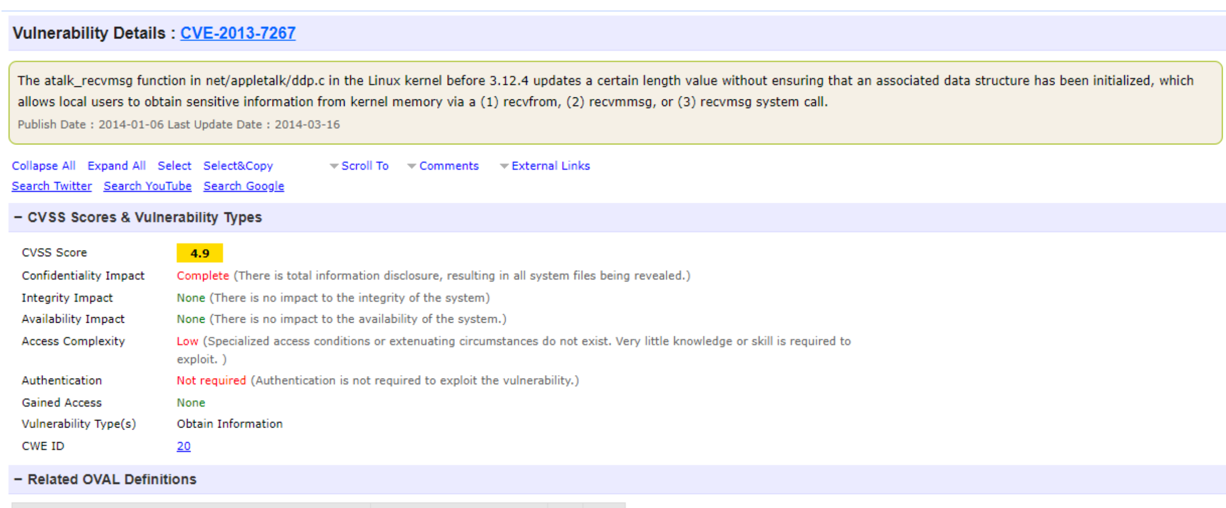


Figure B-4: Page for CVE-2013-7287 as a sample

Before performing the actual code analysis, a diff was run between the T31 kernel files and the WyzeCam Kernel files in order to measure the differences between the two as well as check if any of the functions affected by CVEs had been modified between the two (Figure B-5). None of the differences found from the diff were related to the CVEs we inspected. This indicates that for any CVEs that were not patched and if the WyzeCam V3 T31 chip were to be updated with a newer version, it would still not address any CVEs that had not been patched.

```
diff -r kernel/arch/mips/configs/isvp_swan_defconfig wyze_kernel/arch/mips/configs/isvp_swan_defconfig
1653,1654c1653
< CONFIG_PWM=y
< # CONFIG_JZ_PWM is not set
---
> # CONFIG_PWM is not set
Binary files kernel/arch/mips/xburst/core/mxu-v2-ex.obj and wyze_kernel/arch/mips/xburst/core/mxu-v2-ex.obj differ
diff -r kernel/arch/mips/xburst/soc-t31/chip-t31/isvp/common/board_base.c wyze_kernel/arch/mips/xburst/soc-t31/chip-t31/isvp/common/board_base.c
110,112d109
< #ifdef CONFIG_SERIAL_JZ47XX_UART2
<       DEF_DEVICE(&jz_uart2_device, 0, 0),
< #endif
diff -r kernel/arch/mips/xburst/soc-t31/chip-t31/isvp/common/spi_bus.c wyze_kernel/arch/mips/xburst/soc-t31/chip-t31/isvp/common/spi_bus.c
422,442d421
<               .name           = "XM25QH128C",
<               .pagesize       = 256,
<               .sectorsize     = 4 * 1024,
<               .chipsize       = 16384 * 1024,
<               .erasesize      = 32 * 1024,
<               .id             = 0x204018,
<
<               .block_info     = flash_block_info,
<               .num_block_info = ARRAY_SIZE(flash_block_info),
<
<               .addrsize       = 3,
<               .pp_maxbusy     = 3,            /* 4ms */
<               .se_maxbusy     = 400,          /* 400ms */
<               .ce_maxbusy     = 8 * 10000,    /* 80s */
<
<               .st_regnum      = 3,
< #ifdef CONFIG_SPI_QUAD
<               .quad_mode = &flash_quad_mode[1],
< #endif
<       },
<       {
701,721d679
<
<               .block_info     = flash_block_info,
<               .num_block_info = ARRAY_SIZE(flash_block_info),
<
<               .addrsize       = 3,
```

Figure B-5: Sample output from diff.

The source code analysis starts with identifying the patch for each CVE. The patch is located at the bottom of each CVE Details entry in the form of a github link to the commit that addresses the CVE (Figure B-6). The commits show the code changes for each file modified to address the CVE (Figure B-7). We first check the name of the function that is modified and see if it appears in the diff. We compared the patched files against the files in the T31 SDK Kernel directory and the WyzeCam kernel source files. If all of the necessary code changes from the patches exist, then it is considered fully patched. If they do not, then we determine to what extent it has been patched and whether or not the vulnerability is a threat to the WyzeCam device.

http://secunia.com/advisories/56036
SECUNIA 56036

http://secunia.com/advisories/55882
SECUNIA 55882

https://github.com/torvalds/linux/commit/f3d3342602f8bcbf37d7c46641cb9bca7618eb1c CONFIRM

http://www.openwall.com/lists/oss-security/2013/12/31/7
MLIST [oss-security] 20131231 Re: CVE request: Linux kernel: net: memory leak in recvmsg handlermsg_name & msg_namelen logic

https://bugzilla.redhat.com/show_bug.cgi?id=1039845 CONFIRM

http://www.kernel.org/pub/linux/kernel/v3.x/ChangeLog-3.12.4 CONFIRM

http://git.kernel.org/?p=linux/kernel/git/torvalds/linux-2.6.git;a=commit;h=f3d3342602f8bcbf37d7c46641cb9bca7618eb1c CONFIRM

http://www.ubuntu.com/usn/USN-2128-1
UBUNTU USN-2128-1

http://www.ubuntu.com/usn/USN-2139-1
UBUNTU USN-2139-1

Figure B-5: Link to commit that patches CVE

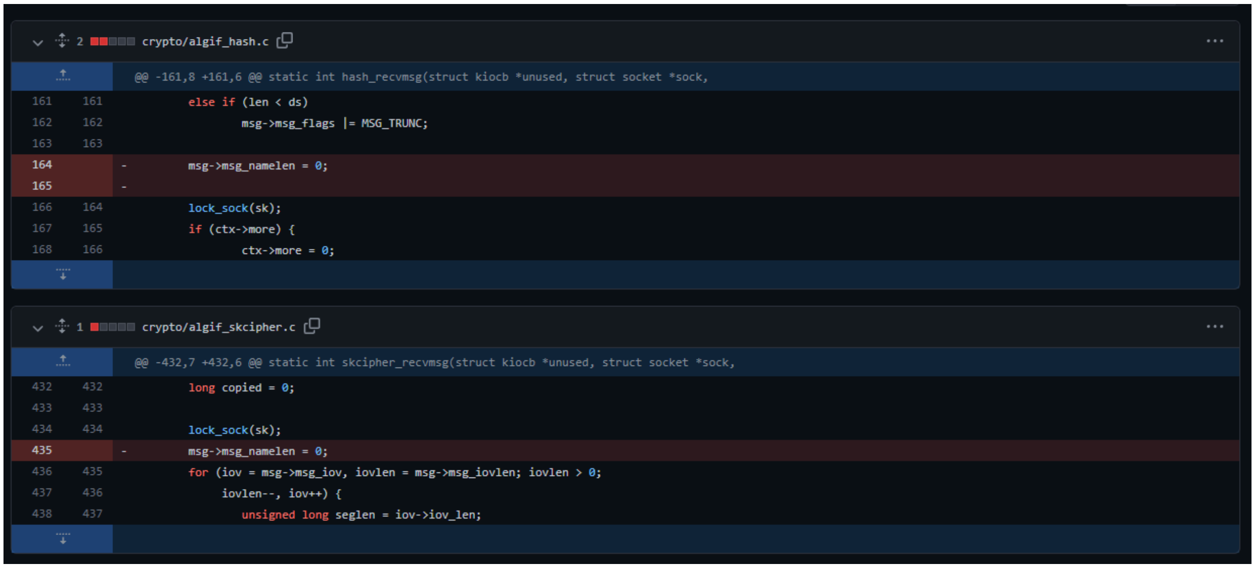


Figure B-6: Code changes in commit. Red means removed, Green (not shown) means added.

Out of the 17 CVEs we inspected, only one has been fully patched: CVE-2012-6638 [25] which has a CVSS score of 7.8. This was the highest rated CVE for this kernel version due to the simplicity of the exploit and how easy it is to reproduce on a vulnerable system by anyone with minimal skill level required. The exploit involves flooding the target system with a specific combination of packets (SYN+FIN) until it is rendered completely unavailable, causing a severe denial of service attack (DoS). (Table B-1)

CVE	CVSS Score	Description
CVE-2012-6638 [25]	7.8	Allows attacker to execute DoS attack by flooding target with SYN+FIN packets [25]

Table B-1: Patched CVEs

10 of the 17 CVEs we inspected were not patched nor appeared to be modified in any way. 3 of them are not applicable however as they require certain functions and systems that the WyzeCam does not utilize, such as KVM [33], Phonet [35], and L2TP [34]. 5 of the applicable CVEs [28] [29] [30] [31] [32] requires local network access at the minimum to the device. This means that an attacker would only be able to utilize these vulnerabilities in targeted attacks where they have access to the network. This limits targets to home networks or small private businesses that cannot afford better security camera options. So as long as the users practice good network security, these CVEs cannot be exploited. The remaining 2 CVEs [26] [27] that are applicable and can be exploited remotely involve modifying properties of packets and share the third highest CVSS score of 7.1. (Table B-2)

CVE	CVSS Score	Description
CVE-2013-3563 [26]	7.1	Allows **remote** attackers to perform DoS attacks using large IPv6 UDP packet sizes [26]
CVE-2013-4348 [27]	7.1	Allows **remote** attackers to perform DoS attacks using small values in the IHL field of a packet with IPIP encapsulation. [27]
CVE-2013-7263 [28]	4.9	Allows **local** users to obtain sensitive info from kernel stack memory using IPV4/V6 systems calls: recvmsg, recvfrom, and recvmmsg [28]
CVE-2013-7281 [29]	4.9	Allows **local** users to obtain sensitive info from the kernel stack memory using 802.15.4 (wireless) system calls: recvmsg, recvfrom, recvmmsg [29]
CVE-2013-6378 [30]	4.4	Allows **local** users to perform DoS attack by using root privileges for a zero-length write operation [30]
CVE-2013-4515 [31]	4.9	Allows **local** users to leak kernel information by exploiting an uninitialized array through IOCTL_BCM_GET_DEVICE_PRINTER system call [31]
CVE-2013-4516 [32]	4.9	Allows **local** users to leak kernel information by exploiting an uninitialized array through TIOCGICOUNT system call [32]
CVE-2013-4587 [33]	7.2	KVM vulnerability. **NOT APPLICABLE** [33]
CVE-2013-7264 [34]	4.9	L2TP vulnerability. **NOT APPLICABLE** [34]
CVE-2013-7265 [35]	4.9	Phonet Packet protocol vulnerability. **NOT APPLICABLE** [35]

Table B-2: CVEs the WyzeCam kernel has not been patched for

7 of the 17 vulnerabilities are considered not patched however, there does appear to be signs of modifications in the functions in the kernel source code related to these CVEs. The amount of changes made is minimal and it is unknown what these modifications are for. However, given the large number of source code files that had to be changed to patch the CVE, it is unlikely that these modifications address the CVE. Fortunately, only one CVE can potentially impact this device. With a CVSS score of 4.9, CVE-2013-7270 [41] allows local users to obtain sensitive information from kernel memory through recvfrom, recvmmsg, and recvmsg system calls through raw packets (af_packet). The other CVEs cover networking protocols [36] [37] [38] [39] [40] that would not be used by the device. (Table B-3)

CVE	CVSS Score	Description
CVE-2013-7270 [41]	4.9	Allows **local** users to obtain sensitive information from kernel memory through recvfrom, recvmmsg, recvmsg system calls related to raw packets (af_packet) [41]
CVE-2013-7266 [36]	4.9	ISDN Protocol. **NOT APPLICABLE** [36]
CVE-2013-7267 [37]	4.9	AppleTalk Protocol. **NOT APPLICABLE** [37]
CVE-2013-7268 [38]	4.9	IPX Protocol. **NOT APPLICABLE** [38]
CVE-2013-7269 [39]	4.9	Netrom Protocol. **NOT APPLICABLE** [39]
CVE-2013-7271 [40]	4.9	X25 Protocol. **NOT APPLICABLE** [40]

Table B-3: Modified CVEs

The ability to exploit these vulnerabilities requires knowledge of the device's existence and in most cases also requires local access to the network. For several of these vulnerabilities, significant modifications would have to be made to the firmware, such as installing utilities that can take advantage of one of the many unpatched/modified CVEs, followed by the reselling of the device to potential targets. Considering this, the threat of the CVEs that were covered in this code inspection is low-to-medium and an attacker would have an easier time loading their own custom software into the camera before reselling the device to potential targets.

Firmware Analysis

We downloaded the latest version of Wyze camera firmware (3_4.36.8.32). The downloaded firmware is a zipped file. On running “binwalk -e” on the zip file, it didn’t mount the file systems, as they were xz compressed. On manually extracting the zipped folder using nautilus file explorer, we can find one empty folder and a blob. The output of running binwalk on blob is show in figure C-1

> binwalk demo_wcv3.bin

```
image name: "jz_fw"
image type: Firmware Image
OS: Linux
CPU: MIPS

image name: "Linux-3.10.14__isvp_swan_1.0__"
image type: OS Kernel Image
OS: Linux
CPU: MIPS

Squashfs filesystem 1 : little endian, version 4.0, compression:xz, size: 3853788 bytes, 384 inodes, blocksize: 131072 bytes

Squashfs filesystem2 :  little endian, version 4.0, compression:xz, size: 3815722 bytes, 194 inodes, blocksize: 131072 bytes, created
```

> file demo_wcv3.bin

```
demo_wcv3.bin: u-boot legacy uImage, jz_fw, Linux/MIPS, Firmware Image (Not compressed), 9846784 bytes, Thu Feb 17 02:13:24 2022, Loa
```

Figure C-1: Binwalk output

From the binwalk output, we can infer that the bootloader is Bootloader - U Boot legacy uimage, and the firmware image is jz_fw. They are packed without any compression techinques. The OS kernel image is Linux-3.10.14__isvp_swan_1.0__ and it is compressed using LZMA compression technique. The instruction set is mips and the system is little endian. There are 2 file system images, both are XZ compressed and are little endian. Binwalk with -e flag didn't set up the file systems. On running binwalk with no flags, it gave the position of each component within the binary, so it is possible to cut each file system image from blob using the dd command line tool. We can use “sudo mount” to mount on the file system, from the file system image.

<u>File system 1 :</u>

On mounting we can see that the file directory has a linux structure as shown in the figure 2. On mounting the file system image using “sudo mount” the file system is read only. There are a total of 349 files in the directory.

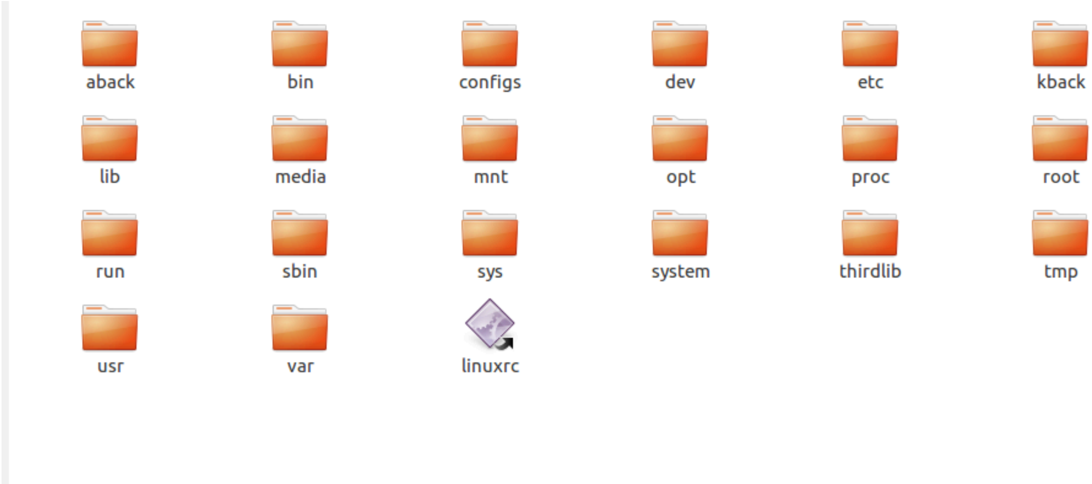


Figure C-2: Structure of file system 1

Once the file system image is mounted we can run the scanning tools on it. On running trommel, we were able to find that the firmware is using BusyBox v1.33.1. It is associated with 18 CVEs (CVE-2018-0099, CVE-2017-5671 ,CVE-2017-16544 ,CVE-2017-15874 ,CVE-2017-15873, CVE-2017-14116 ,CVE-2017-14115 ,CVE-2016-6301 ,CVE-2016-5791 ,CVE-2016-214 ,CVE-2016-2147 ,CVE-2014-9645 ,CVE-2013-1813 ,CVE-2011-5325 ,CVE-2011-2716, CVE-2006-5050 ,CVE-2006-1058 ,CVE-2005-2136) according to trommel false positives may exist. On running firmwalker, we were able to find that there is no trace of SSH, SSL, database, openSSL related files in this file directory. There were 5 IP addresses in the filesystem (4.36.8.32 ,1.2.3.2 ,4.3.24.7 ,3.4.4.3 ,8.8.8.8), but on testing them, they were not vulnerable IP addresses. On etc folder, we can find a shadow file. Shadow file had the root user's hash.

```
root:$6$wyzecamv3$8gyTEsAkm1d7wh12Eup5MMcxQwuA1n1FsRtQLUW8dZGo1b1pGRJg
tSieTI02VPeFP9f4DodbIt2ePOLzwP0WI0:0:0:99999:7:::
```

Figure C-3: Hash of root password

The hash has $6 in the beginning as shown in figure 3, this shows it is hashed using SHA512 hashing algorithm. Following it, there is wyzecmav3, which according to the format of SHA512 hash algorithm is the salt used to hash the password [42]. Following it we have the hash value of salted password. The numbers following the hash tell other details like time to reset, password expiry time etc, which are of no interest currently.

File system 2:

On mounting we can see that the file directory has a linux structure as shown in the figure 4. On mounting the file system image using “sudo mount” the file system is read only. There are a total of 169 files in that directory.

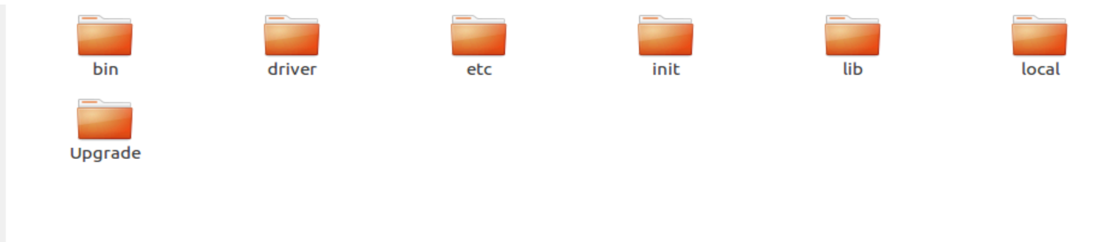


Figure C-4: Structure of file system 2

On running trommel and firmwaker on the mounted file system, binary files cacert.pem, hl_client, and iCamera seemed promising. On analyzing the hl_client file in the cutter tool, it can be seen in figure 5 and figure 6 that fgets and strcpy are used in the binary. They can lead to dangerous consequences, so it is better to avoid them.

```
(fcn) sym.imp.fgets  16
  char * sym.imp.fgets  (char *s, int size, FILE *stream);
lui t7, 0x4a                          ; 'J'
lw t9, 0x6068(t7)
jr t9
addiu t8, t7, 0x6068                  ; 'h`'
```

Figure C-5: fget() function

```
(fcn) sym.imp.strcpy  16
  char * sym.imp.strcpy  (char *dest, const char *src);
lui t7, 0x4a                          ; 'J'
lw t9, 0x6040(t7)
jr t9
addiu t8, t7, 0x6040                  ; '@`'
```

Figure C-6: strcpy() function

<u>Changing root hash and repacking the firmware:</u>

When a file system image is mounted using ‘sudo mount’, the file system is read only and cannot be edited. To get around this, we can use the sasquatch tool. On running the sasquatch tool on file system image, it creates a directory from the file system image, which is read and write-able. Since we know the type of hashing algorithm (SHA512) and the salt (wyzecamv3) used to produce the hash, we can pick a password of our choice, find the corresponding salted hash value and then rewrite it in the shadow file. We have taken ‘esslp’ to be the password, and replaced the original hash with the salted hash of ‘esslp’. The salted hash of esslp is shown in figure 7.

```
esslp@ubuntu:~/workspace/embedtools/squashfs-root/etc$ cat shadow
root:$6$wyzecamv3$Jgd8pQHFFZxPg5C.a1hN17sKDeU4aIBc8IpNhPk/iCFsNcvogroptl66jPO0au4IK2NRqxvQvHFqgzlCA6hty1:0:0:99999:7:::
esslp@ubuntu:~/workspace/embedtools/squashfs-root/etc$
```

Figure C-7: Salted hash of password ‘esslp’

To recreate the filesystem image we have used a command line utility mksquashfs. The output of the running the tool on the modified file directory is shown in figure 8

```
(embedtools) esslp@ubuntu:~/workspace/embedtools/firmware-mod-kit/other-scripts$ mksquashfs /home/ess
lp/workspace/embedtools/squashfs-root /home/esslp/wyze-cam-firmware-analysis/firmware_recreate/filesy
s1.sqsh -comp xz -b 131072
Parallel mksquashfs: Using 2 processors
Creating 4.0 filesystem on /home/esslp/wyze-cam-firmware-analysis/firmware_recreate/filesys1.sqsh, bl
ock size 131072.
[=====================================================================================-] 136/136 100%

Exportable Squashfs 4.0 filesystem, xz compressed, data block size 131072
        compressed data, compressed metadata, compressed fragments, compressed xattrs
        duplicates are removed
Filesystem size 3763.47 Kbytes (3.68 Mbytes)
        32.94% of uncompressed filesystem size (11424.91 Kbytes)
Inode table size 2176 bytes (2.12 Kbytes)
        15.31% of uncompressed inode table size (14212 bytes)
Directory table size 3230 bytes (3.15 Kbytes)
        52.95% of uncompressed directory table size (6100 bytes)
Number of duplicate files found 2
Number of inodes 384
Number of files 64
Number of fragments 14
Number of symbolic links  287
Number of device nodes 0
Number of fifo nodes 0
Number of socket nodes 0
Number of directories 33
Number of ids (unique uids + gids) 1
Number of uids 1
        esslp (1000)
Number of gids 1
        esslp (1000)
(embedtools) esslp@ubuntu:~/workspace/embedtools/firmware-mod-kit/other-scripts$
```

Figure C-8: blob of modified file system 1

We can recreate the firmware back, using command line utility dd and cat. From the binwalk output as shown in figure 9,we can see that the file system 1 starts from location 2031680 (decimal).

```
(embedtools) esslp@ubuntu:~/wyze-cam-firmware-analysis/demo_wcv3_4.36.8.32$ binwalk demo_wcv3.bin

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             uImage header, header size: 64 bytes, header CRC: 0x75A4CF47, created: 2022-02-17 02:13:24, image size: 9846784 bytes, Data Address: 0x0, Entry Point: 0x0, data CRC: 0x1B1540
5A, OS: Linux, CPU: MIPS, image type: Firmware Image, compression type: none, image name: "jz_fw"
64            0x40            uImage header, header size: 64 bytes, header CRC: 0xA3BC7407, created: 2021-07-02 12:31:59, image size: 1897077 bytes, Data Address: 0x80010000, Entry Point: 0x80416900, data
 CRC: 0xB0B2FE38, OS: Linux, CPU: MIPS, image type: OS Kernel Image, compression type: lzma, image name: "Linux-3.10.14__isvp_swan_1.0__"
128           0x80            LZMA compressed data, properties: 0x5D, dictionary size: 67108864 bytes, uncompressed size: -1 bytes
2031680       0x1F0040        Squashfs filesystem, little endian, version 4.0, compression:xz, size: 3853788 bytes, 384 inodes, blocksize: 131072 bytes, created: 2022-02-17 02:13:21
6029376       0x5C0040        Squashfs filesystem, little endian, version 4.0, compression:xz, size: 3815722 bytes, 194 inodes, blocksize: 131072 bytes, created: 2022-02-17 02:13:24

(embedtools) esslp@ubuntu:~/wyze-cam-firmware-analysis/demo_wcv3_4.36.8.32$ ls
```

Figure C-9: Binwalk output of modified firmware

Using dd command, we can copy everything before file system 1 into a new file, this file now contains binary of bootloader, firmware and OS. Similarly using dd we copy file system 2 from offset 6029376 into a file, which now will contain an image of file system 2. We have all the 3 parts in the required format as shown in image 10 , we can combine them using the cat command line tool and write it into a new file, which will be the modified firmware image.

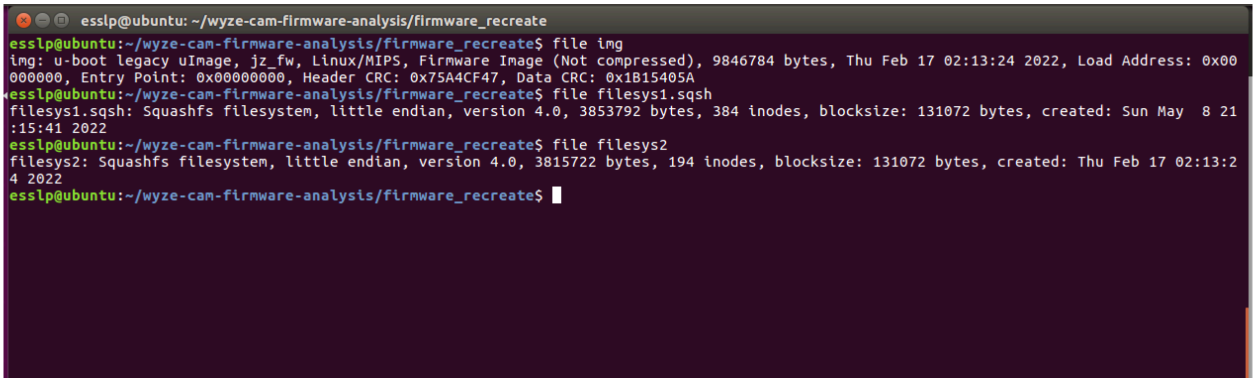


Figure C-10: Details of individual blobs

The modified firmware has same format as the original format, as shown in figure 11

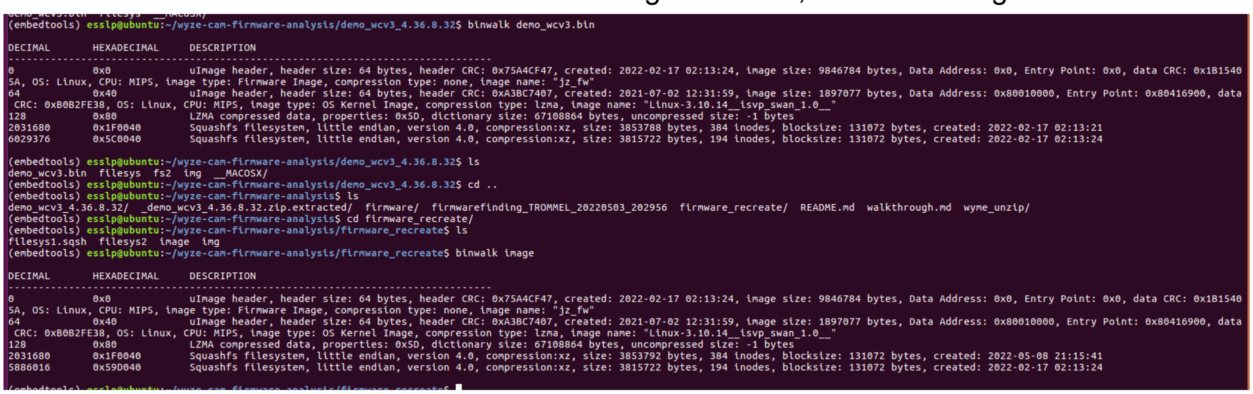


Figure C-11: comparison of original and modified firmware image

iCamera Analysis

The iCamera binary is of interest to us because it is one of the few custom binaries executed at the startup of the Wyze Cam V3. The idea is to perform static analysis on the iCamera binary in the hope of finding possible security vulnerabilities. The methodology that we take in this project is to search for commonly vulnerable libc functions and check if they are used safely. In this section we explore the iCamera binary and outline possible vulnerable function calls that an attacker could use to exploit the Wyze Cam V3.

```
analyst@hub:~/Wyze/squashfs-root-0$ ls
bin  driver  etc  init  lib  local  Upgrade
analyst@hub:~/Wyze/squashfs-root-0$ file ./bin/iCamera
./bin/iCamera: ELF 32-bit LSB executable, MIPS, MIPS32 rel2 version 1 (SYSV), dynamically linked, interpreter /lib/ld-uClibc.so.0, stripped
analyst@hub:~/Wyze/squashfs-root-0$
```

Figure D-1: Running file command on iCamera

The iCamera binary is located in the bin directory of the second squashfs filesystem.

```
analyst@hub:~/Wyze/squashfs-root-0$ cat ./init/app_init.sh | grep -C 10 iCamera
    mkdaemon 20 assis /system/bin/assis
    #/system/bin/assis > /dev/null 2>&1 &
    #while [ true ]; do
    #    pidof assis > /dev/null
    #    if [ $? -eq 0 ]; then
    #        break;
    #    fi
    #done
    mkdaemon 20 hl_client /system/bin/hl_client
    mkdaemon 20 sinker /system/bin/sinker
    mkdaemon 0 iCamera /system/bin/iCamera
    mkdaemon 20 dumpload /system/bin/dumpload
    mkdaemon -1 timesync /system/bin/timesync
    mkdaemon 0 audiocardprocess /system/bin/audiocardprocess

        #/system/bin/assis &
        #/system/bin/hl_client &
        #/system/bin/sinker &
        #/system/bin/iCamera &
else
        sleep 0.5
        echo "########################"
        echo "#   IS DEBUG STATUS    #"
        echo "########################"
fi
analyst@hub:~/Wyze/squashfs-root-0$
```

Figure D-2: The app_init.sh startup script

The iCamera binary is executed at startup by the Wyze Cam V3 in the ./init/app_init.sh script.

```
analyst@hub:~/Wyze/squashfs-root-0$ head -n 40 ./init/app_init.sh
#!/bin/sh
mkdaemon() {
    # dmon options
    #   --stderr-redir  Redirects stderr to the log file as well
    #   --max-respawns  Sets the number of times dmon will restart a failed process
    #   --environ       Sets an environment variable. Used to remove buffering on stdout
    #
    # dslog options
    #   --priority      The syslog priority. Set to DEBUG as these are just the stdout of the
    #   --max-files     The number of logs that will exist at once
    #
    max_respawns=$1
    shift
    daemon_name=$1
    shift
    dmon \
      --stderr-redir \
      --max-respawns $max_respawns \
      --environ "LD_PRELOAD=libsetunbuf.so" \
      $@ \
      -- dslog \
        --priority DEBUG \
        --facility USER \
        $daemon_name
}
############# Setting register and insert wifi ko ############
insmod /system/driver/tx-isp-t31.ko isp_clk=220000000
insmod /system/driver/exfat.ko
insmod /system/driver/audio.ko spk_gpio=-1 alc_mode=0 mic_gain=0
#insmod /system/driver/avpu.ko
insmod /system/driver/sinfo.ko
insmod /system/driver/mmc_detect_test.ko
insmod /system/driver/sample_pwm_core.ko
insmod /system/driver/sample_pwm_hal.ko
insmod /system/driver/speaker_ctl.ko
insmod /system/driver/ch34x.ko

ubootddr=`sed -n '30p' /proc/jz/clock/clocks | cut -d ' ' -f 7`

if [[ "540.000MHz" == $ubootddr ]]; then
analyst@hub:~/Wyze/squashfs-root-0$
```

Figure D-3: Showing app_init.sh loads kernel modules

We know that iCamera is executed with root privileges because the app_init.sh script also loads kernel modules. Because loading kernel modules requires root privileges we can assume that app_init.sh is run with root privileges and therefore iCamera is run with root privileges.

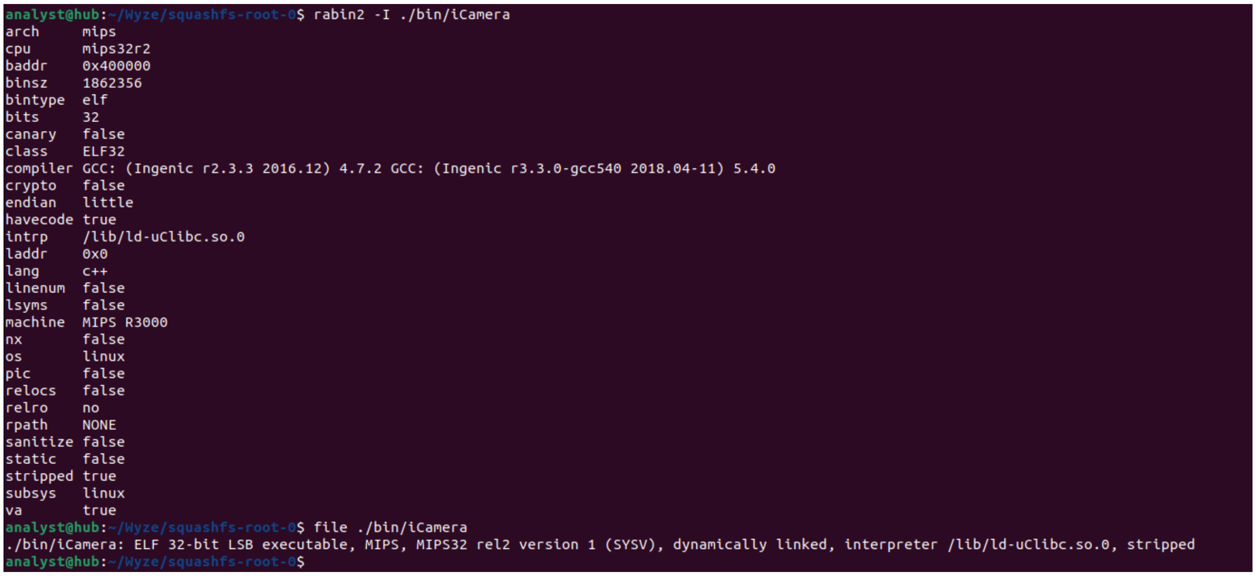


Figure D-4: Rabin2 output

Based on the file command and rabin2 we can see that iCamera is a 32-bit little endian mips ELF binary that is dynamically linked and stripped of symbols. Rabin2 also indicates that iCamera was written in C++.

Network Function Calls

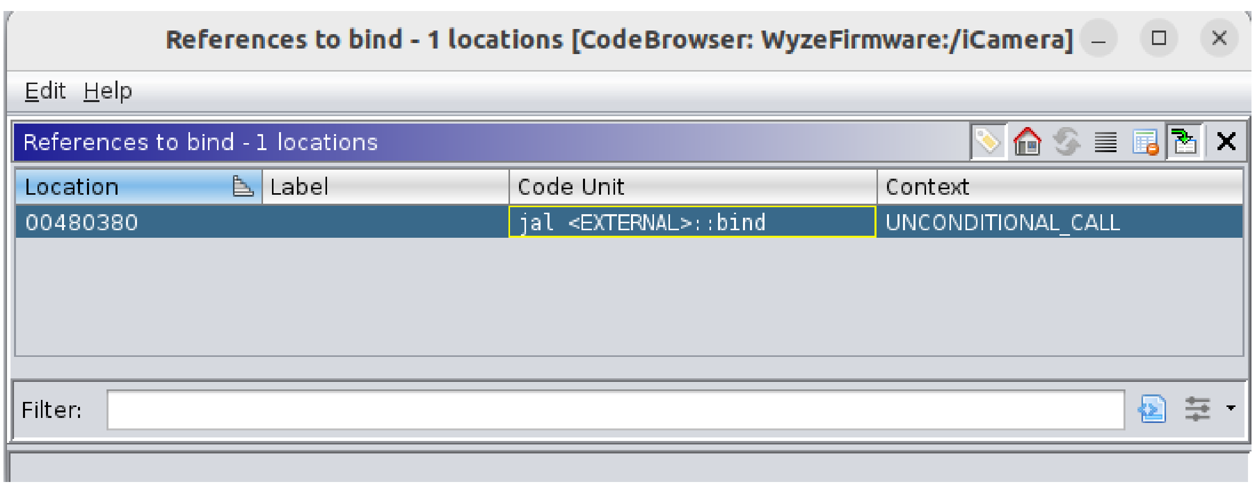


Figure D-5: There is 1 reference found to the bind() libc function

```
__fd = socket(1,1,0);
if (__fd < 0) {
  perror("[av_recv] Error: failed to create audio receiver unix domain socket");
}
else {
  memset(auStack176,0,0x6e);
  auStack176._0_2_ = 1;
  auStack176._3_4_ = 0x657a7977;
  uStack169 = 0x6475612d;
  uStack165 = 0x622d6f69;
  uStack161 = 0x74737469;
  uStack157 = 0x6d616572;
  uStack153 = 0x6365722d;
  uStack149 = 0x65766965;
  cStack145 = 'r';
  cStack144 = '\0';
  iVar3 = bind(__fd,(sockaddr *)auStack176,0x6e);
  if (iVar3 < 0) {
    __s = "[av_recv] Error: failed to bind audio receiver unix domain socket";
  }
  else {
```

Figure D-6: Ghidra decompilation of reference to bind()

After some basic reverse-engineering we can assume that a pseudocode version of the bind statement looks something like the following:

> fd = socket(AF_UNIX, SOCK_STREAM, 0);
> bind(fd, “wyze-audio-bitstream-receiver”, 0x6e);

This tells us that iCamera is binding a unix domain socket and not a network socket. Unix domain sockets are used for interprocess communication so this would not be a good attack vector since we can’t access this socket remotely.

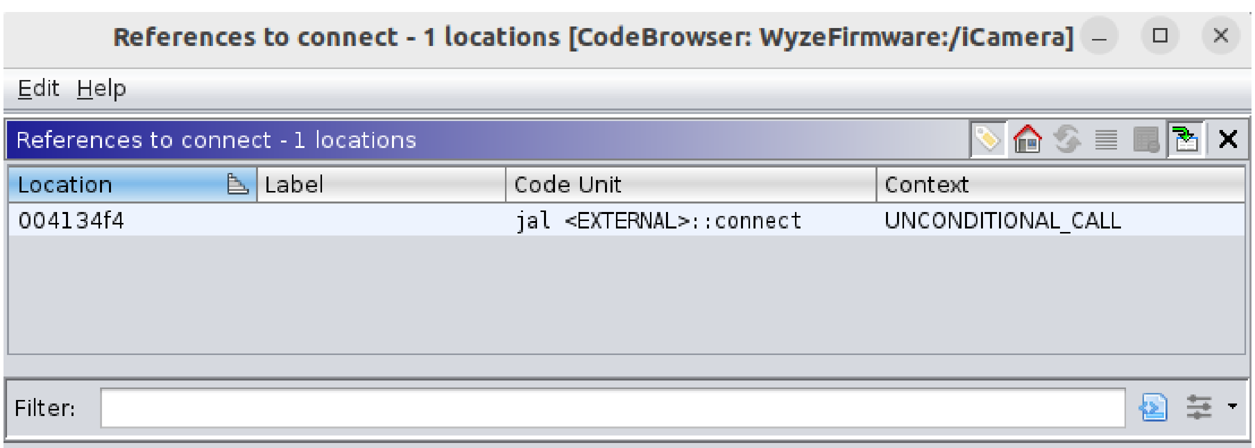


Figure D-7: There is 1 reference found to the connect() libc function

```
__fd = socket(1,2,0);
if (__fd < 0) {
  piVar2 = __errno_location();
  strerror(*piVar2);
  uVar4 = 0xc2;
  pcVar3 = "iot creat socket error: %d,%s\n";
}
else {
  memset(local_80,0,0x6e);
  local_80[0].sa_family = 1;
  strncpy(local_80[0].sa_data,"/tmp/hualaiclient.domain",0x6b);
  iVar1 = connect(__fd,local_80,0x6e);
  if (-1 < iVar1) {
    return __fd;
  }
  close(__fd);
  piVar2 = __errno_location();
  strerror(*piVar2);
  uVar4 = 0xcf;
  pcVar3 = "iot socket connect: %d,%s\n";
}
```

Figure D-8: Ghidra decompilation of reference to connect()

After some basic reverse-engineering we can assume that a pseudocode version of the connect statement looks something like this:

> fd = socket(AF_UNIX, SOCK_DGRAM, 0);
> connect(fd, “/tmp/hualaiclient.domain”, 0x6b);

The iCamera binary is connecting to a local unix domain socket. This isn’t a good attack vector because the connection isn’t over a network socket so we can’t access it remotely.

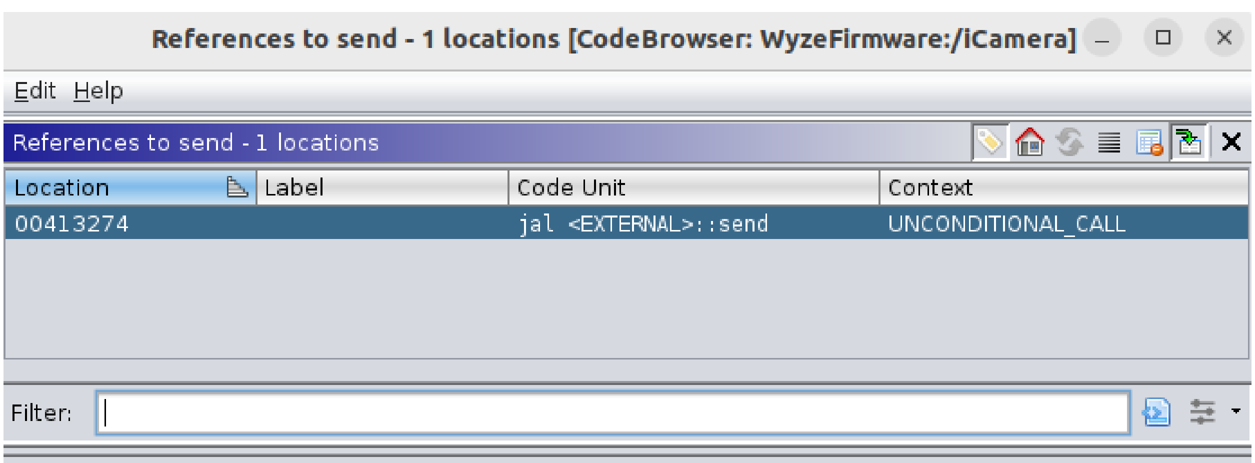


Figure D-9: There is 1 reference found to the send() libc function

```
__fd = stored_unix_fd;
memset(&local_2020,0,0x2000);
local_2020 = param_1;
local_201c = param_3;
memcpy(auStack8216,param_2,param_3);
FUN_0044882c("connectivity/mqtt","{\"sendCmd\":%d}",param_1,param_4);
if (__fd < 0) {
  print_debug("[iCamera]",6,"iot.c","iot_send",0x86,"(%s): SocketFd invalid\n");
  sVar1 = -1;
}
else {
  sVar1 = send(__fd,&local_2020,local_201c + 8,0);
  print_debug("[iCamera]",6,"iot.c","iot_send",0x82,"(%s):%s  (ret%d, len:%d)\n");
}
```

Figure D-10: Ghidra decompilation of reference to send()

After some more reverse engineering we found that the value of stored_unix_fd (which I have renamed for clarity) is the unix domain socket returned from the connect() function call to “/tmp/hualaiclient.domain”. Therefore since this send() doesn’t involve the network we don’t consider it for our attack surface.

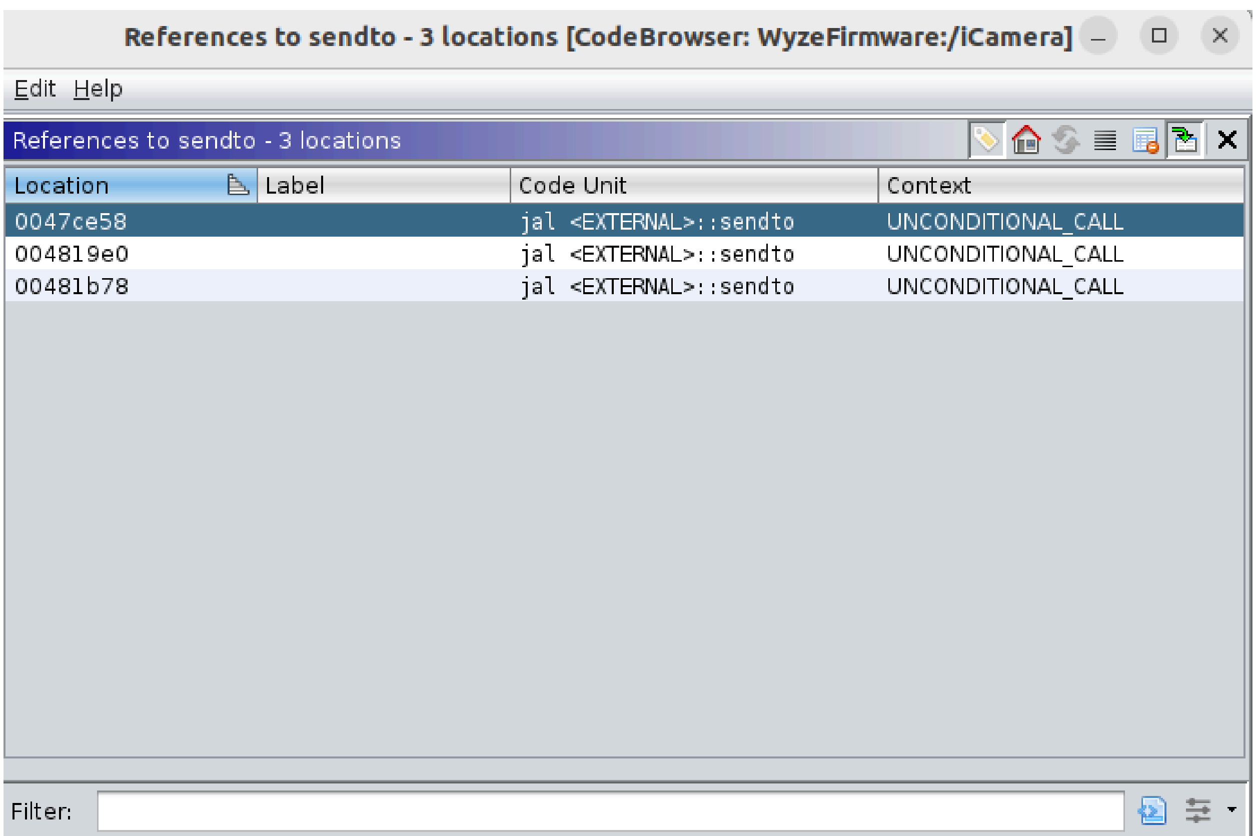


Figure D-11: There are 3 references found to the sendto() libc function

```
DAT_0060d050 = socket(2,3,ppVar3->p_proto);
if ((int)DAT_0060d050 < 0) {
  perror("socket error");
  uVar9 = 0xffffffff;
}
else {
  __uid = getuid();
  setuid(__uid);
  setsockopt(DAT_0060d050,0xffff,0x1002,&local_2c,4);
  DAT_0060d058 = 0;
  DAT_0060d05c = 0;
  DAT_0060d060 = 0;
  DAT_0060d054 = 2;
  iVar4 = inet_addr(param_1);
  if (iVar4 == 0xffffffff) {
    printf("[ping:%d]err: %s only support ip addr, (ex:192.xx.xx.x)\n",0x11e,"ping_process");
    uVar9 = 0xffffffff;
  }
  else {
    DAT_0060d058 = inet_addr(param_1);
    DAT_0060d064 = getpid();
    while (DAT_0060d048 < DAT_005d50f8) {
      while( true ) {
        DAT_0060d048 = DAT_0060d048 + 1;
        DAT_0060e086 = (undefined2)DAT_0060d048;
        DAT_0060e081 = 0;
        DAT_0060e082 = 0;
        DAT_0060e084 = (undefined2)DAT_0060d064;
        DAT_0060e080 = 8;
        gettimeofday((timeval *)&DAT_0060e088,(__timezone_ptr_t)0x0);
        uVar11 = 0;
        puVar5 = (ushort *)&DAT_0060e080;
        do {
          uVar1 = *puVar5;
          puVar5 = puVar5 + 1;
          uVar11 = uVar11 + uVar1;
        } while (puVar5 != (ushort *)0x60e0c0);
        iVar2 = ((int)uVar11 >> 0x10) + (uVar11 & 0xffff);
        DAT_0060e082 = ~((short)iVar2 + (short)((uint)iVar2 >> 0x10));
        sVar6 = sendto(DAT_0060d050,&DAT_0060e080,0x40,0,(sockaddr *)&DAT_0060d054,0x10);
        if (sVar6 < 0) break;
        usleep(30000);
```

Figure D-12: Ghidra decompilation of reference to sendto()

After doing some reverse engineering it looks like iCamera is sending a ICMP packet in the pseudocode form of:

```
fd = socket(AF_INET, SOCK_RAW, IPPROTO_ICMP);
…
sendto(fd, ?, 0x40, 0, ?, 0x10);
```

From static analysis alone it is hard to tell what the exact data is in the ICMP packet and to where the ICMP packet is being sent. Luckily we were able to spin up a hotspot and view the traffic from the Wyze Cam V3 in wireshark.

No.	Time	Source	Destination	Protocol	Length	Info
7009	94.366421774	192.168.12.195	192.168.12.1	ICMP	98	Echo (ping) request id=0xc800, seq=256/1, ttl=64 (reply in 7010)
7010	94.366488668	192.168.12.1	192.168.12.195	ICMP	98	Echo (ping) reply id=0xc800, seq=256/1, ttl=64 (request in 7009)
7011	94.399275571	192.168.12.195	192.168.12.1	ICMP	98	Echo (ping) request id=0xc800, seq=512/2, ttl=64 (reply in 7012)
7012	94.399335304	192.168.12.1	192.168.12.195	ICMP	98	Echo (ping) reply id=0xc800, seq=512/2, ttl=64 (request in 7011)
7013	94.446828160	192.168.12.195	192.168.12.1	ICMP	98	Echo (ping) request id=0xc800, seq=768/3, ttl=64 (reply in 7014)
7014	94.446902413	192.168.12.1	192.168.12.195	ICMP	98	Echo (ping) reply id=0xc800, seq=768/3, ttl=64 (request in 7013)

Figure D-13: ICMP packet being sent to/from Wyze Cam V3

```
▸ Frame 7009: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface ap0, id 0
▸ Ethernet II, Src: WyzeLabs_14:c3:87 (d0:3f:27:14:c3:87), Dst: IntelCor_89:4b:c2 (88:b1:11:89:4b:c2)
▸ Internet Protocol Version 4, Src: 192.168.12.195, Dst: 192.168.12.1
▾ Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0x9e53 [correct]
    [Checksum Status: Good]
    Identifier (BE): 51200 (0xc800)
    Identifier (LE): 200 (0x00c8)
    Sequence Number (BE): 256 (0x0100)
    Sequence Number (LE): 1 (0x0001)
    [Response frame: 7010]
    Timestamp from icmp data: May  5, 2022 16:11:50.924040000 EDT
    [Timestamp from icmp data (relative): 0.433465071 seconds]
  ▾ Data (48 bytes)
      Data: 000000000000000000000000000000000000000000000000000000000000000000000000…
      [Length: 48]
```

Figure D-14: Payload of sniffed ICMP packet

The Wyze Cam V3 has the ip address 192.168.12.195 and the hotspot access point (my laptop) has the ip address 192.168.12.1. This means that the Wyze Cam V3 is sending an ICMP packet to an access point with a data field of all zeros. It looks like it may be a form of a keep-alive message. This is interesting but probably doesn’t lead to a vulnerability.

```
if (DAT_005d6794 < 0) {
  perror("video uds fd is not valid\n");
  uVar1 = 0xffffffff;
}
else {
  memset(auStack152,0,0x6e);
  auStack152._0_2_ = 1;
  auStack152._3_4_ = 0x657a7977;
  uStack145 = 0x3632682d;
  uStack141 = 0x69622d78;
  uStack137 = 0x72747374;
  uStack133 = 0x6d6165;
  _DAT_006747f8 = FUN_00525528(param_1[7],param_1[8],1000,0);
  DAT_006747f4 = 0;
  uVar1 = param_1[1];
  if (uVar1 != 0) {
    uVar2 = 0;
    do {
      __n = uVar1 - uVar2;
      if (0x3ff0 < __n) {
        __n = 0x3ff0;
      }
      uVar3 = __n + uVar2;
      DAT_006747f2 = uVar3 < uVar1 ^ 1;
      DAT_006747f0 = (ushort)(uVar2 == 0);
      memcpy(&DAT_00674800,(void *)(*param_1 + uVar2),__n);
      sendto(DAT_005d6794,&DAT_006747f0,__n + 0x10,0x40,(sockaddr *)auStack152,0x6e);
      DAT_006747f4 = DAT_006747f4 + 1;
      uVar1 = param_1[1];
      uVar2 = uVar3;
    } while (uVar3 < uVar1);
```

Figure D-15: Ghidra decompilation of second reference to sendto()

Some basic reverse engineering results in the simplified pseudocode:

```
fd = socket(AF_UNIX, SOCK_STREAM, 0);
…
sendto(fd, ?, ?, 0x40, “wyze_h2bx_bitstream”, 0x6e);
```

Although we don’t know what it is sending (or the length), we know iCamera is sending data to a unix domain socket. It looks like this specific unix domain socket is responsible for video. This is mildly interesting but likely won’t lead to a vulnerability.

The last reference to the sendto() function call was very similar to the previous one except it communicated over a unix domain socket to the address “wyze-g711-bitstream”. Once again it probably won’t lead to a vulnerability.

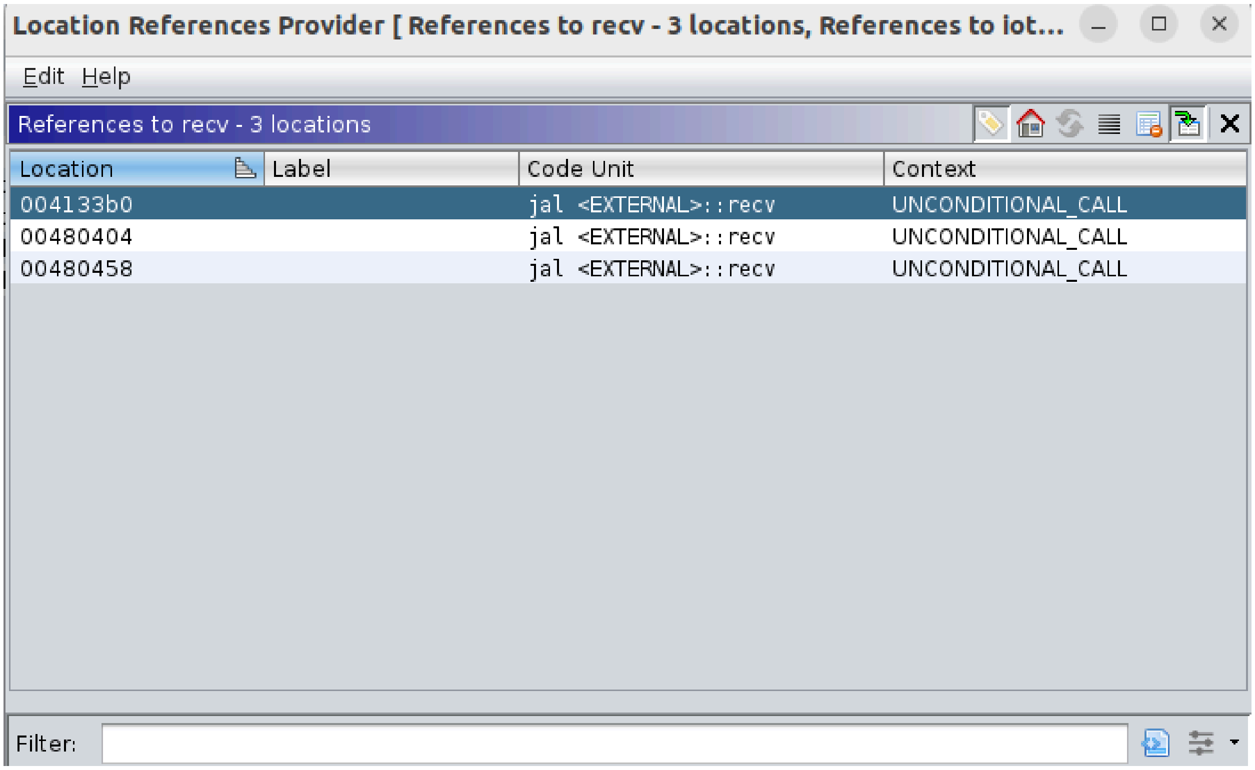


Figure D-16: There are 3 references found to the recv() libc function

The first reference communicates over a unix domain socket that we discussed earlier so it isn't of real interest to us.

The second and third reference receive over the unix domain socket that bind was called on earlier ("wyze-audio-bitstream-receiver"). This isn't of any interest to us from a vulnerability research perspective.

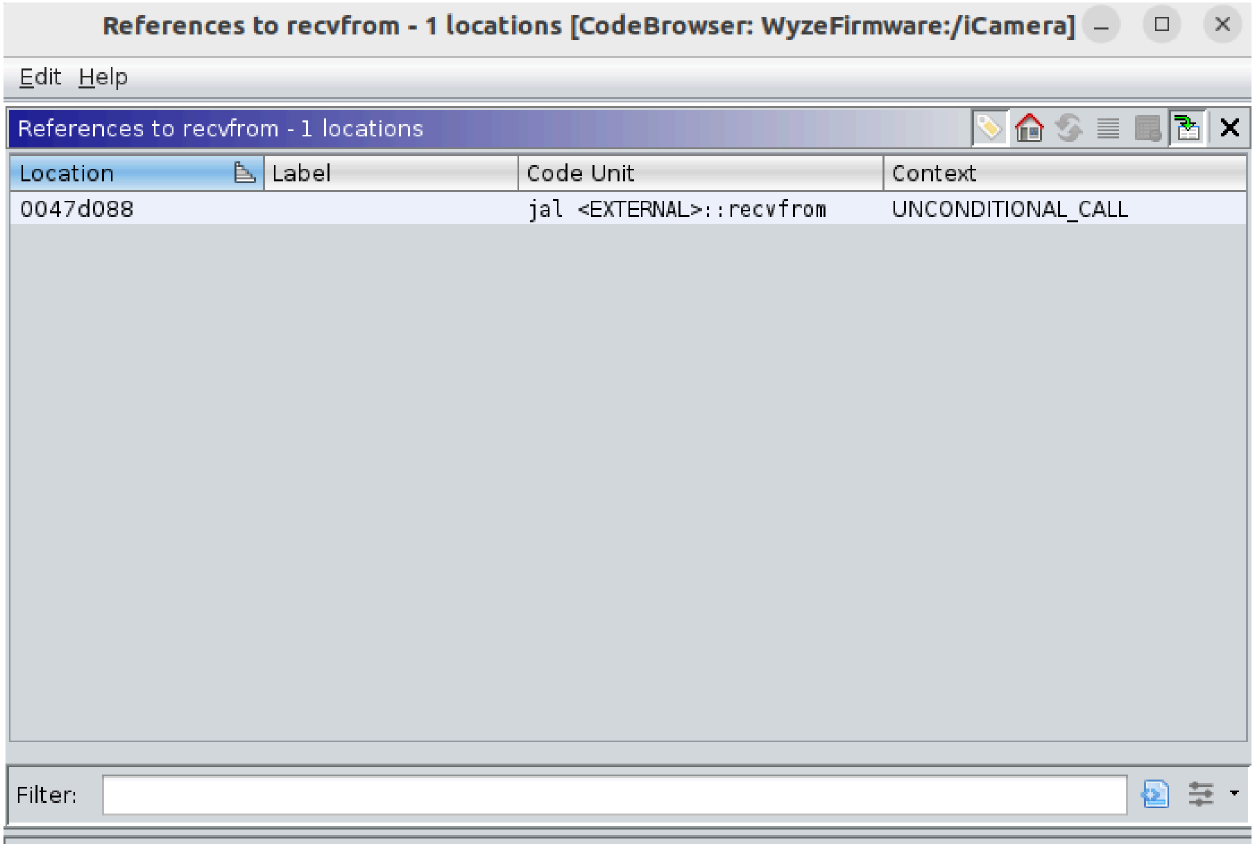


Figure D-17: There is 1 reference found to the recvfrom() libc function

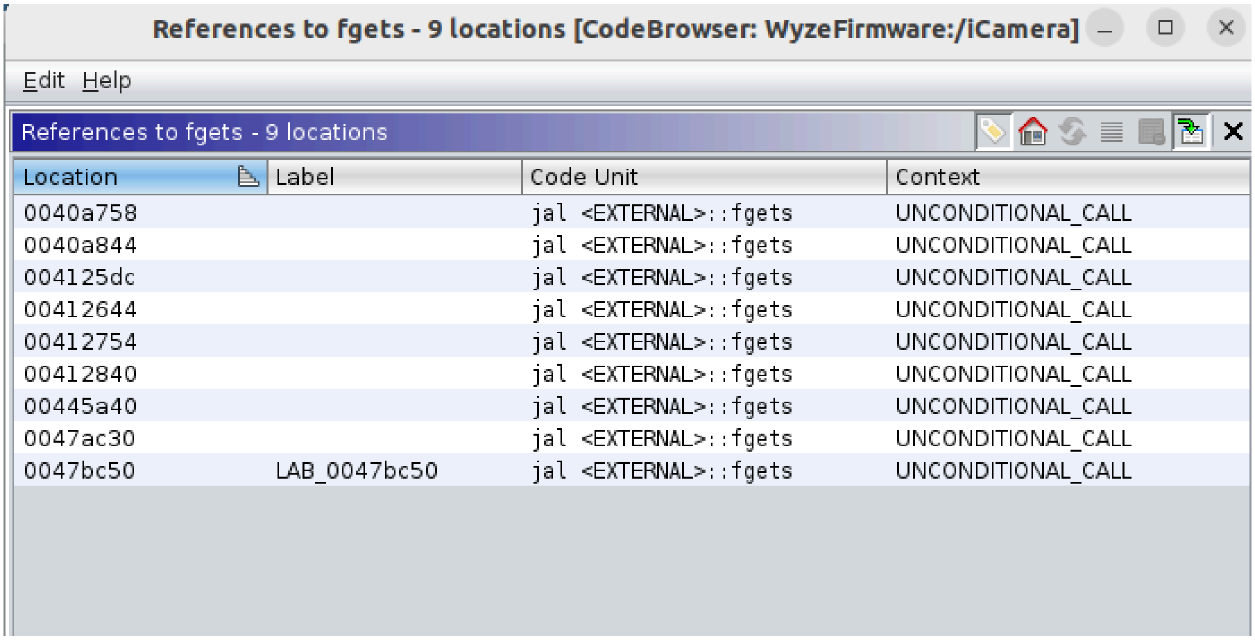
```
            sVar6 = sendto(DAT_0060d050,&DAT_0060e080,0x40,0,(sockaddr *)&DAT_0060d054,0x10);
            if (sVar6 < 0) break;
            usleep(30000);
            if (DAT_005d50f8 <= DAT_0060d048) goto LAB_0047ce88;
          }
          perror("sendto error");
        }
LAB_0047ce88:
        local_30 = 0x10;
        pfVar7 = &local_b8;
        do {
          pfVar7->fds_bits[0] = 0;
          pfVar7 = (fd_set *)pfVar7->fds_bits;
        } while (&local_38 != (timeval *)pfVar7);
        iVar2 = 0;
        while ((DAT_0060d04c < DAT_0060d048 && (iVar2 <= DAT_005d50f4))) {
          local_b8.fds_bits[DAT_0060d050 >> 5] =
               local_b8.fds_bits[DAT_0060d050 >> 5] | 1 << (DAT_0060d050 & 0x1f);
          local_38.tv_sec = 1;
          local_38.tv_usec = 0;
          iVar8 = select(DAT_0060d050 + 1,&local_b8,(fd_set *)0x0,(fd_set *)0x0,&local_38);
          if (iVar8 == -1) break;
          if (iVar8 == 0) {
            iVar2 = iVar2 + 1;
          }
          else if ((local_b8.fds_bits[DAT_0060d050 >> 5] >> (DAT_0060d050 & 0x1f) & 1U) != 0) {
            sVar6 = recvfrom(DAT_0060d050,&DAT_0060d068,0x1000,0,(sockaddr *)&DAT_0060e068,&local_30
                            );
```

Figure D-18: Ghidra decompilation of reference to recvfrom()

Based on the recvfrom() function call's proximity to the sendto() function call that sent out the ICMP packets, and the fact that this recvfrom() uses the same socket as the sendto() call, we reasonably conclude that this recvfrom is receiving the ICMP response packets. This is a possible attack vector depending on how iCamera handles the buffer that the ICMP packet is loaded into, and if it performs any checks validating the ICMP packet.

Buffer Overflow

There are no references to gets() in iCamera.


References to fgets - 9 locations [CodeBrowser: WyzeFirmware:/iCamera]

Edit Help

References to fgets - 9 locations

Location	Label	Code Unit	Context
0040a758		jal <EXTERNAL>::fgets	UNCONDITIONAL_CALL
0040a844		jal <EXTERNAL>::fgets	UNCONDITIONAL_CALL
004125dc		jal <EXTERNAL>::fgets	UNCONDITIONAL_CALL
00412644		jal <EXTERNAL>::fgets	UNCONDITIONAL_CALL
00412754		jal <EXTERNAL>::fgets	UNCONDITIONAL_CALL
00412840		jal <EXTERNAL>::fgets	UNCONDITIONAL_CALL
00445a40		jal <EXTERNAL>::fgets	UNCONDITIONAL_CALL
0047ac30		jal <EXTERNAL>::fgets	UNCONDITIONAL_CALL
0047bc50	LAB_0047bc50	jal <EXTERNAL>::fgets	UNCONDITIONAL_CALL

Figure D-19: There are 9 references found to the fgets() libc function

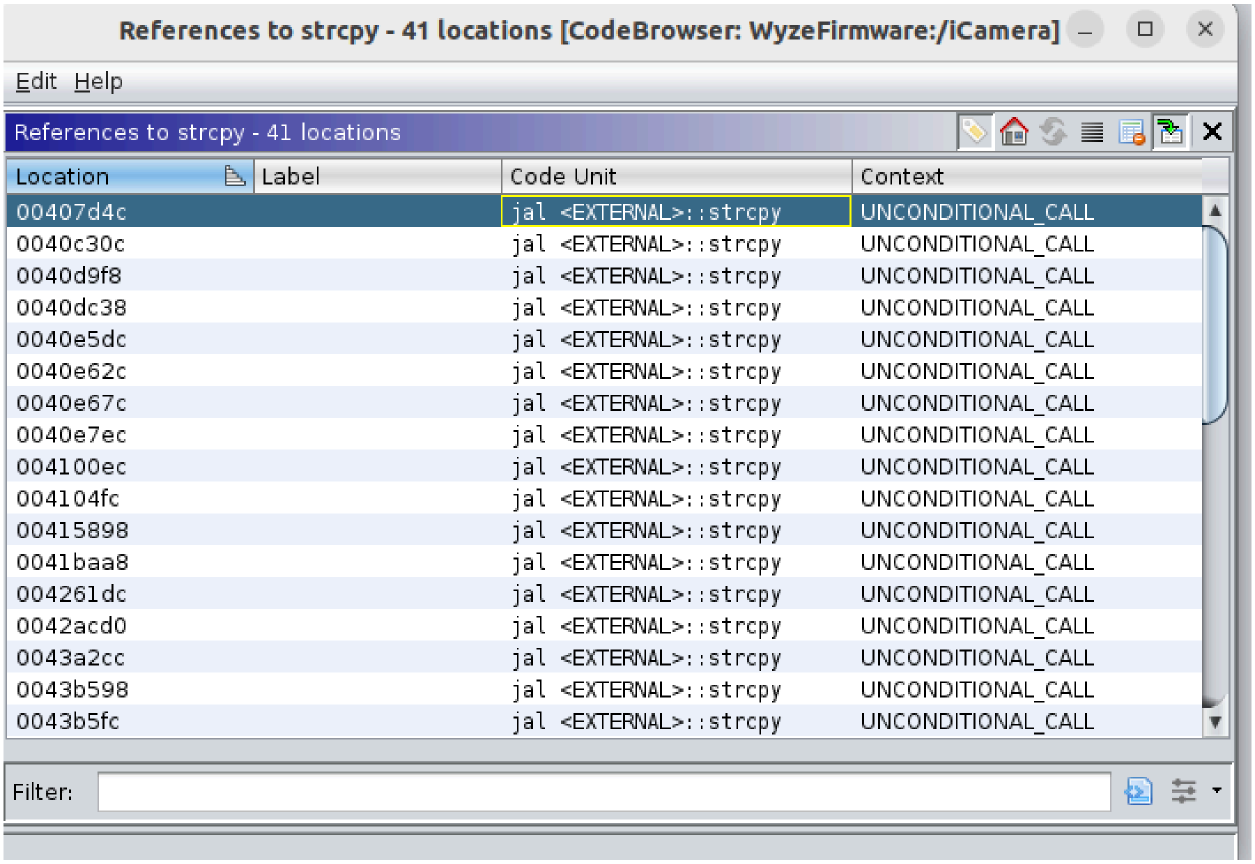
All 9 references to fgets() are used safely. All calls to fgets use a size “n” that is less than or equal to the size of the buffer that fgets is reading into.

Ex: buffer is size 108 and n is 100 so there is no risk of a buffer overflow.

```
  FILE *__stream;
  char *pcVar1;
  size_t sVar2;
  int iVar3;
  char acStack136 [108];

  __stream = fopen("/proc/net/wireless","r");
  iVar3 = 0;
  if (__stream == (FILE *)0x0) {
    print_debug("[iCamera]",5,"iot_msg_process.c","get_wifi_level",0x183,"Error opening file\n");
  }
  else {
LAB_004125d8:
    pcVar1 = fgets(acStack136,100,__stream);
```

Figure D-20: Ghidra decompilation of reference to fgets()



Location	Label	Code Unit	Context
00407d4c		jal <EXTERNAL>::strcpy	UNCONDITIONAL_CALL
0040c30c		jal <EXTERNAL>::strcpy	UNCONDITIONAL_CALL
0040d9f8		jal <EXTERNAL>::strcpy	UNCONDITIONAL_CALL
0040dc38		jal <EXTERNAL>::strcpy	UNCONDITIONAL_CALL
0040e5dc		jal <EXTERNAL>::strcpy	UNCONDITIONAL_CALL
0040e62c		jal <EXTERNAL>::strcpy	UNCONDITIONAL_CALL
0040e67c		jal <EXTERNAL>::strcpy	UNCONDITIONAL_CALL
0040e7ec		jal <EXTERNAL>::strcpy	UNCONDITIONAL_CALL
004100ec		jal <EXTERNAL>::strcpy	UNCONDITIONAL_CALL
004104fc		jal <EXTERNAL>::strcpy	UNCONDITIONAL_CALL
00415898		jal <EXTERNAL>::strcpy	UNCONDITIONAL_CALL
0041baa8		jal <EXTERNAL>::strcpy	UNCONDITIONAL_CALL
004261dc		jal <EXTERNAL>::strcpy	UNCONDITIONAL_CALL
0042acd0		jal <EXTERNAL>::strcpy	UNCONDITIONAL_CALL
0043a2cc		jal <EXTERNAL>::strcpy	UNCONDITIONAL_CALL
0043b598		jal <EXTERNAL>::strcpy	UNCONDITIONAL_CALL
0043b5fc		jal <EXTERNAL>::strcpy	UNCONDITIONAL_CALL

Figure D-21: There are 41 references found to the strcpy() libc function

```
int iVar1;
undefined4 uVar2;

DAT_005dc054 = 3;
iVar1 = FUN_0040da8c((char *)&DAT_005dc058,3);
if (iVar1 < 0) {
  DAT_005dc05c = 0;
  DAT_005dc060 = 0;
  DAT_005dc064 = 0;
  DAT_005dc068 = 0;
  DAT_005dc06c = 0;
  DAT_005dc070 = 0;
  DAT_005dc074 = 0;
  DAT_005dc058 = 0;
  DAT_005dc054 = 0;
  uVar2 = 0xffffffff;
}
else {
  strcpy(param_1,(char *)&DAT_005dc058);
  uVar2 = 0;
}
```

Figure D-22: Ghidra decompilation of reference to strcpy()

This reference to strcpy() could be vulnerable depending if the src pointer &DAT_005dc058 can be manipulated by an attacker. It is difficult to determine its vulnerability based on static analysis alone and would probably need a closer look with dynamic analysis.

Most of the strcpy() calls seemed safe as the src parameter appears bounded, and unable to be influenced by an attacker. There were a few strcpy() calls (like the one above) that copied from a memory address, or parameter, whose contents are difficult to determine using static analysis. Dynamic analysis would be useful in determining if these strcpy() calls are vulnerable to a buffer overflow.

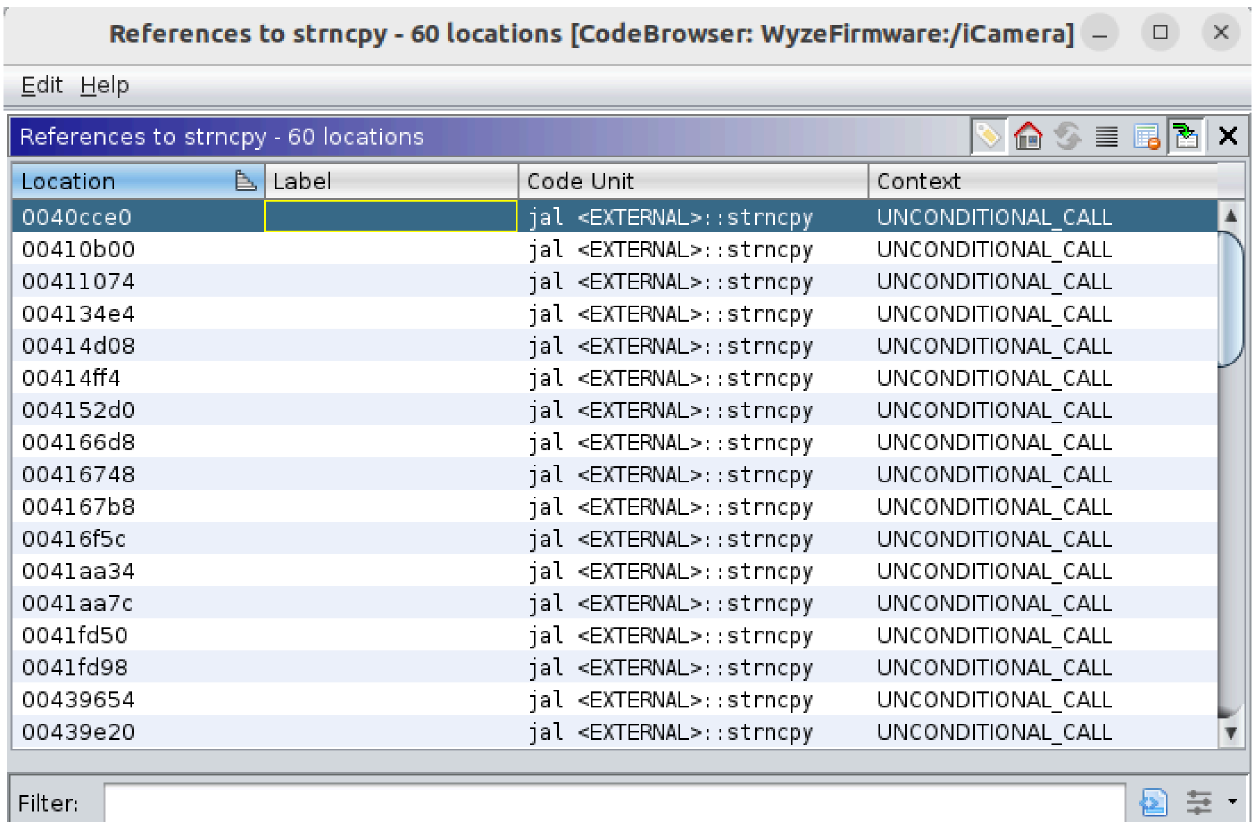


Figure D-23: There are 60 references found to the strncpy() libc function

```
char acStack272 [64];
char acStack208 [64];
char acStack144 [64];
undefined auStack80 [64];
undefined4 local_10;

memset(acStack272,0,0x104);
FUN_0047a824(acStack272);
strncpy(acStack208,"        ",0x40);
strncpy(acStack144,"    ",0x40);
FUN_0047a92c(auStack80);
local_10 = 2;
FUN_0048b8ec(acStack272);
return;
```

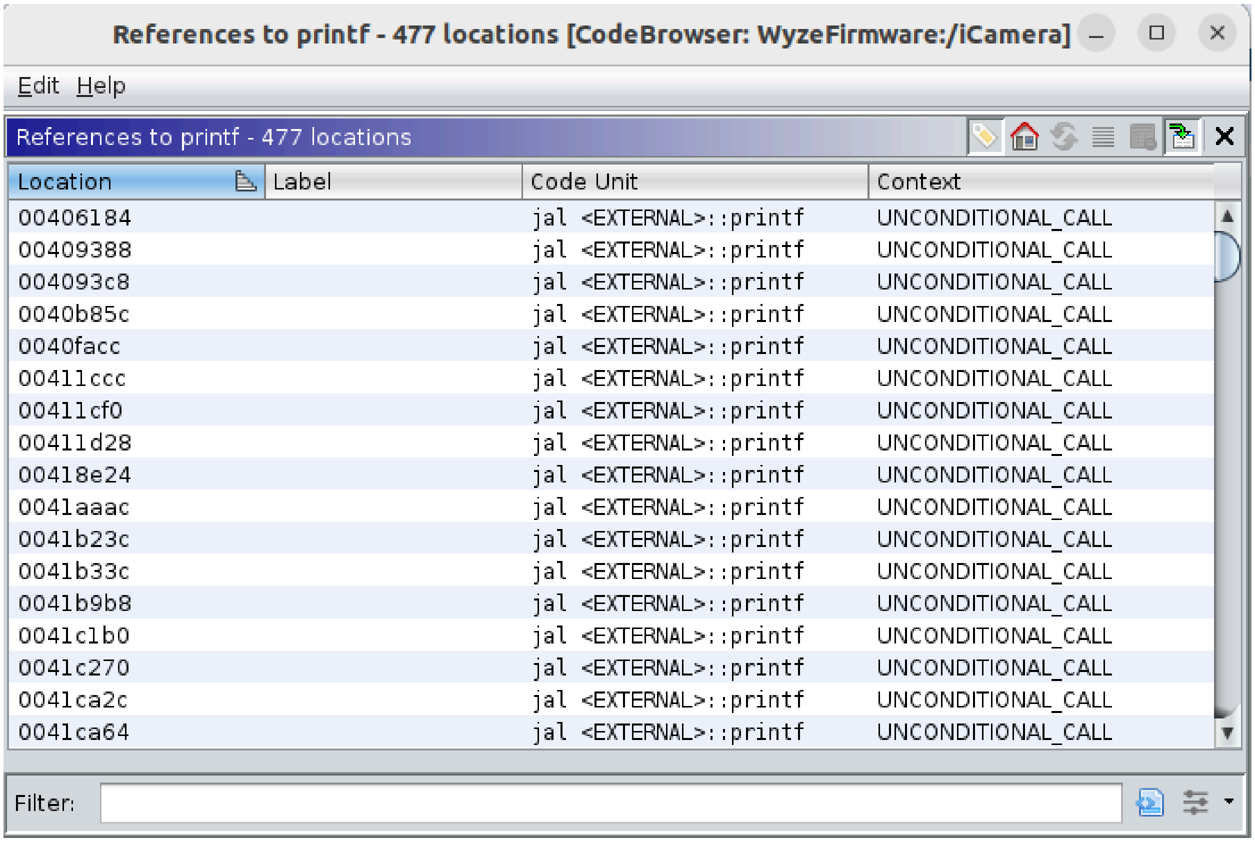
Figure D-24: Ghidra decompilation of reference to strncpy()

Most of the calls to strncpy() were clearly bound correctly (like the example above). It was easy to tell that these strncpy() calls weren't vulnerable to a buffer overflow.

```
local_b0 = param_1[0x2b];
if (local_b0 != 0) {
  strncpy((char *)auStack208,(char *)((int)param_1 + 0x8e),local_b0);
  FUN_0047164c(auStack208);
  return;
}
uVar10 = 0xe9;
pcVar4 = "NOT HAVE rtmp user name id !\n";
```

Figure D-25: Ghidra decompilation of second reference to strncpy()

Other strncpy() calls made it hard to tell if they were bound correctly (like the example above) because the value for "n" was a memory address whose contents are hard to determine only using static analysis. This is another example of when dynamic analysis would likely provide a more concrete answer.



Location	Label	Code Unit	Context
00406184		jal <EXTERNAL>::printf	UNCONDITIONAL_CALL
00409388		jal <EXTERNAL>::printf	UNCONDITIONAL_CALL
004093c8		jal <EXTERNAL>::printf	UNCONDITIONAL_CALL
0040b85c		jal <EXTERNAL>::printf	UNCONDITIONAL_CALL
0040facc		jal <EXTERNAL>::printf	UNCONDITIONAL_CALL
00411ccc		jal <EXTERNAL>::printf	UNCONDITIONAL_CALL
00411cf0		jal <EXTERNAL>::printf	UNCONDITIONAL_CALL
00411d28		jal <EXTERNAL>::printf	UNCONDITIONAL_CALL
00418e24		jal <EXTERNAL>::printf	UNCONDITIONAL_CALL
0041aaac		jal <EXTERNAL>::printf	UNCONDITIONAL_CALL
0041b23c		jal <EXTERNAL>::printf	UNCONDITIONAL_CALL
0041b33c		jal <EXTERNAL>::printf	UNCONDITIONAL_CALL
0041b9b8		jal <EXTERNAL>::printf	UNCONDITIONAL_CALL
0041c1b0		jal <EXTERNAL>::printf	UNCONDITIONAL_CALL
0041c270		jal <EXTERNAL>::printf	UNCONDITIONAL_CALL
0041ca2c		jal <EXTERNAL>::printf	UNCONDITIONAL_CALL
0041ca64		jal <EXTERNAL>::printf	UNCONDITIONAL_CALL

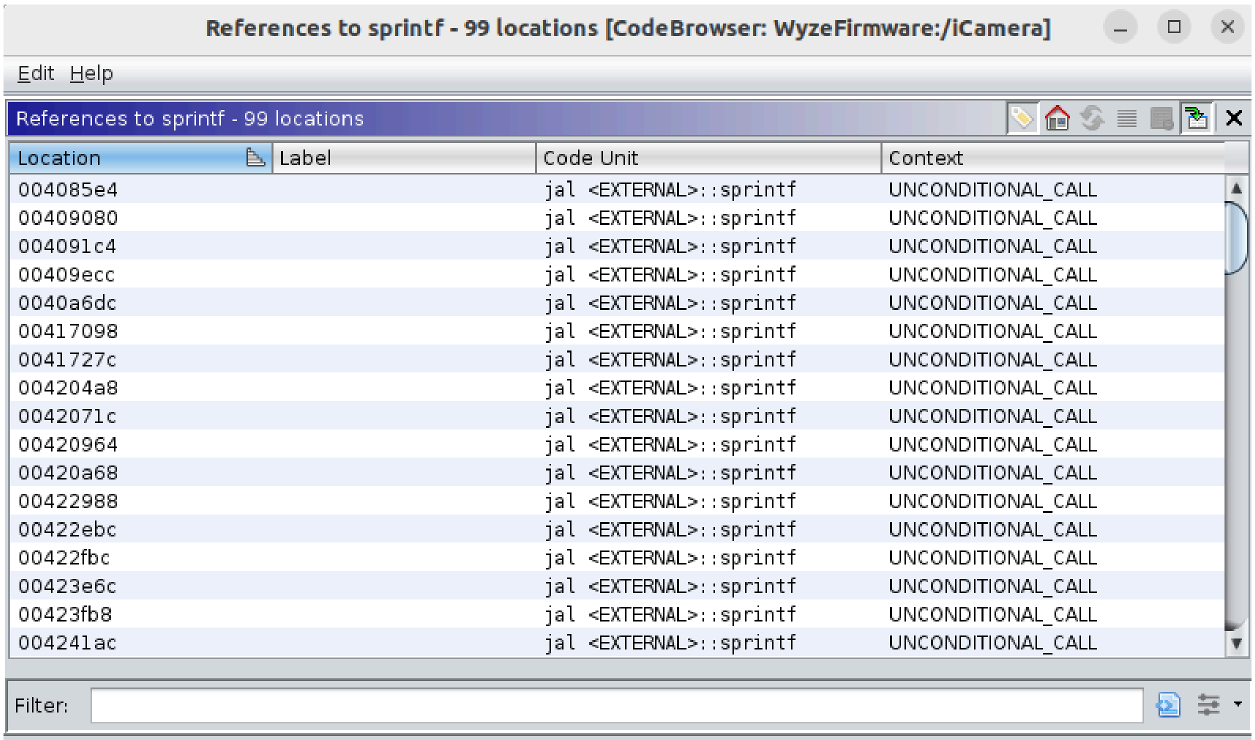
Figure D-26: There are 477 references found to the printf() libc function

```
int iVar1;

iVar1 = IMP_IVS_DestroyGroup(0);
if (iVar1 == 0) {
  printf("[%s]dbg: IMP_IVS_DestroyGroup(%d) ok, ret:%d!\n","IVS-MOTION",0,0);
  return 0;
}
printf("[%s]err: IMP_IVS_DestroyGroup(%d) fail, ret:%d!\n","IVS-MOTION",0,iVar1);
return iVar1;
```

Figure D-27: Ghidra decompilation of reference to printf()

All references to printf() include a format string as the first parameter (like example above), so it appears that there are no printf format string vulnerabilities.



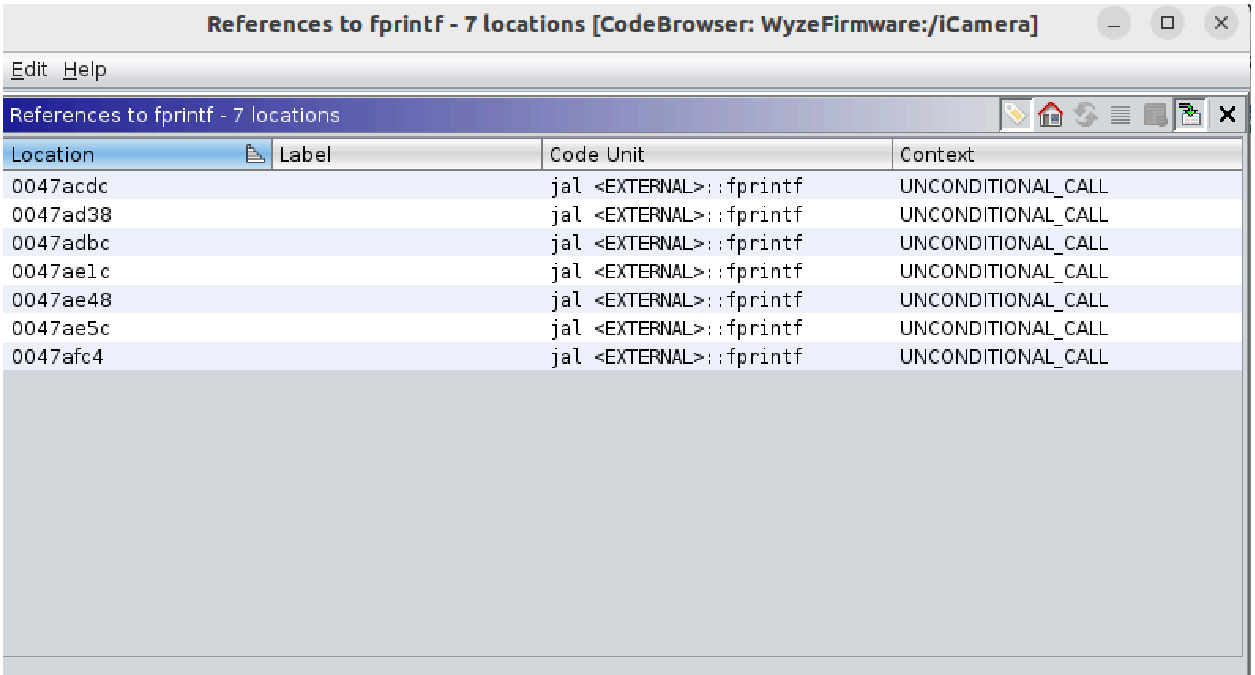
Location	Label	Code Unit	Context
004085e4		jal <EXTERNAL>::sprintf	UNCONDITIONAL_CALL
00409080		jal <EXTERNAL>::sprintf	UNCONDITIONAL_CALL
004091c4		jal <EXTERNAL>::sprintf	UNCONDITIONAL_CALL
00409ecc		jal <EXTERNAL>::sprintf	UNCONDITIONAL_CALL
0040a6dc		jal <EXTERNAL>::sprintf	UNCONDITIONAL_CALL
00417098		jal <EXTERNAL>::sprintf	UNCONDITIONAL_CALL
0041727c		jal <EXTERNAL>::sprintf	UNCONDITIONAL_CALL
004204a8		jal <EXTERNAL>::sprintf	UNCONDITIONAL_CALL
0042071c		jal <EXTERNAL>::sprintf	UNCONDITIONAL_CALL
00420964		jal <EXTERNAL>::sprintf	UNCONDITIONAL_CALL
00420a68		jal <EXTERNAL>::sprintf	UNCONDITIONAL_CALL
00422988		jal <EXTERNAL>::sprintf	UNCONDITIONAL_CALL
00422ebc		jal <EXTERNAL>::sprintf	UNCONDITIONAL_CALL
00422fbc		jal <EXTERNAL>::sprintf	UNCONDITIONAL_CALL
00423e6c		jal <EXTERNAL>::sprintf	UNCONDITIONAL_CALL
00423fb8		jal <EXTERNAL>::sprintf	UNCONDITIONAL_CALL
004241ac		jal <EXTERNAL>::sprintf	UNCONDITIONAL_CALL

Figure D-28: There are 99 references found to the sprintf() libc function

```
sprintf(acStack224,(char *)&PTR_DAT_0053dae4,*param_1);
FUN_0047f468("setup","name",acStack224,acStack124);
sprintf(acStack224,"%d",param_1[3]);
FUN_0047f468("setup","TimeGap",acStack224,acStack124);
sprintf(acStack224,(char *)&PTR_DAT_0053dae4,param_1[2]);
FUN_0047f468("setup","endTime",acStack224,acStack124);
sprintf(acStack224,(char *)&PTR_DAT_0053dae4,param_1[1]);
FUN_0047f468("setup","beginTime",acStack224,acStack124);
sprintf(acStack224,"%d",param_1[4]);
FUN_0047f468("setup","Timezone",acStack224,acStack124);
```

Figure D-29: Ghidra decompilation of reference to sprintf()

Some of the references to sprintf() above could be vulnerable depending if the second argument can be manipulated by an attacker. It is difficult to determine if it's vulnerable based on static analysis alone and would probably need a closer look with dynamic analysis.


References to fprintf - 7 locations [CodeBrowser: WyzeFirmware:/iCamera]

Edit Help

References to fprintf - 7 locations

Location	Label	Code Unit	Context
0047acdc		jal <EXTERNAL>::fprintf	UNCONDITIONAL_CALL
0047ad38		jal <EXTERNAL>::fprintf	UNCONDITIONAL_CALL
0047adbc		jal <EXTERNAL>::fprintf	UNCONDITIONAL_CALL
0047ae1c		jal <EXTERNAL>::fprintf	UNCONDITIONAL_CALL
0047ae48		jal <EXTERNAL>::fprintf	UNCONDITIONAL_CALL
0047ae5c		jal <EXTERNAL>::fprintf	UNCONDITIONAL_CALL
0047afc4		jal <EXTERNAL>::fprintf	UNCONDITIONAL_CALL

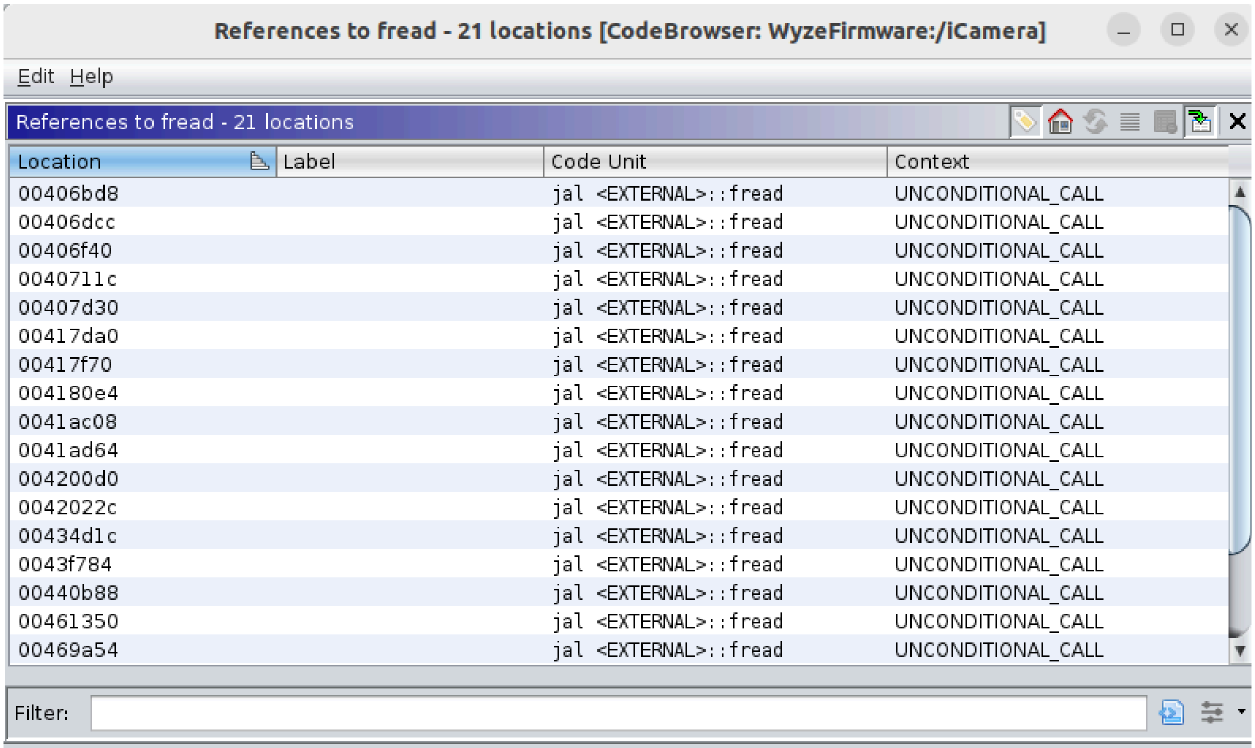
Figure D-30: There are 7 references found to the fprintf() libc function

```
else {
  fwrite("network={\n",1,10,__stream);
  fprintf(__stream,"\tssid=\"%s\"\n",param_1);
  fwrite("\tkey_mgmt=WPA-PSK\n",1,0x12,__stream);
  fwrite("\tpairwise=CCMP TKIP\n",1,0x14,__stream);
  fwrite("\tgroup=CCMP TKIP WEP104 WEP40\n",1,0x1e,__stream);
  fprintf(__stream,"\tpsk=\"%s\"\n",param_2);
  pcVar2 = "\tscan_ssid=1\n";
  sVar3 = 0xd;
}
```

Figure D-31: Ghidra decompilation of reference to fprintf()

All calls to fprintf() are used safely as they all include a hardcoded format string as the second argument (like the example above).

File Access



Location	Label	Code Unit	Context
00406bd8		jal <EXTERNAL>::fread	UNCONDITIONAL_CALL
00406dcc		jal <EXTERNAL>::fread	UNCONDITIONAL_CALL
00406f40		jal <EXTERNAL>::fread	UNCONDITIONAL_CALL
0040711c		jal <EXTERNAL>::fread	UNCONDITIONAL_CALL
00407d30		jal <EXTERNAL>::fread	UNCONDITIONAL_CALL
00417da0		jal <EXTERNAL>::fread	UNCONDITIONAL_CALL
00417f70		jal <EXTERNAL>::fread	UNCONDITIONAL_CALL
004180e4		jal <EXTERNAL>::fread	UNCONDITIONAL_CALL
0041ac08		jal <EXTERNAL>::fread	UNCONDITIONAL_CALL
0041ad64		jal <EXTERNAL>::fread	UNCONDITIONAL_CALL
004200d0		jal <EXTERNAL>::fread	UNCONDITIONAL_CALL
0042022c		jal <EXTERNAL>::fread	UNCONDITIONAL_CALL
00434d1c		jal <EXTERNAL>::fread	UNCONDITIONAL_CALL
0043f784		jal <EXTERNAL>::fread	UNCONDITIONAL_CALL
00440b88		jal <EXTERNAL>::fread	UNCONDITIONAL_CALL
00461350		jal <EXTERNAL>::fread	UNCONDITIONAL_CALL
00469a54		jal <EXTERNAL>::fread	UNCONDITIONAL_CALL

Figure D-32: There are 21 references found to the fread() libc function

Most of the calls to fread() were clearly bound correctly. It was easy to tell that these fread() calls weren't vulnerable to a buffer overflow.

```
if (local_30 < 0xc8000) {
  sVar5 = (&DAT_00607b84)[(int)param_1 * 0x4b] - (&DAT_00607b80)[(int)param_1 * 0x4b];
  if (0x1000 < (int)sVar5) {
    sVar5 = 0x1000;
  }
  sVar5 = fread(__s + 5,1,sVar5,(FILE *)(&DAT_00607b74)[(int)param_1 * 0x4b]);
```

Figure D-33: Ghidra decompilation of reference to fread()

The fread() call above makes it hard to tell if it is bound correctly because the value for "n" and the value for the buffer are memory addresses whose contents are hard to determine only using static analysis. Dynamic analysis would likely provide a more concrete answer.

System

```
memset(acStack280,0,0x100);
sprintf(acStack280,"ps | grep %s | grep -v grep > /tmp/process",param_1);
print_debug("[iCamera]",5,"binding.c","find_if_process_rum",0x122,"find_if_process_rum buf:%s \n")
;
system(acStack280);
memset(acStack280,0,0x100);
```

Figure D-34: Ghidra decompilation of reference to system()

There is a possibility for local privilege escalation if we had a shell on the Wyze Cam V3. The full path isn't specified for ps and grep when they are passed to the system() command. We may be able to use a path trick to force the iCamera binary into using a malicious version of ps or grep.

There are many other cases in iCamera where the full path of a program is not being specified when calling system().

Notable Findings

- A call to recvfrom() is used to read in ICMP packets over a raw socket. Memory corruption could occur if the processing of the packet isn't handled properly.
- Some strcpy() and strncpy() calls have arguments that may be unsafe, but their value is difficult to determine through static analysis. Some of these function calls should be further evaluated with dynamic analysis.
- It is difficult to determine if a proper format string is used for some sprintf() calls, but it is unlikely that these calls are vulnerable to a format string exploit.
- Some fread() calls have arguments that may be unsafe, but their value is difficult to determine through static analysis. Some of these function calls should be further evaluated with dynamic analysis.
- There are multiple calls to system() where the full path of a program isn't specified. This could lead to local privilege escalation if the PATH of iCamera was hijacked.

Firmware Visual Analysis

Generally, firmware updates are downloaded in compressed form to save space. To analyze the firmware, we must first determine whether it is encrypted or compressed. The visual analysis of the binary is one of the techniques that can be used to analyze unknown binary files. Based on the generated pattern image we can determine the instruction set and architecture of the embedded system, identify vulnerability, find the difference between two firmware, perform security audits, and can be used to determine the security posture of the embedded system. We used binwalk, binvis, pixd, bin2bm in this project to generate image patterns of the firmware[53].

Binwalk - Entropy

Entropy is a measure of the information density of the file and they are represented as a number of bits per character[54]. If the entropy is very high meaning that there is a high chance that the file is compressed or encrypted and cannot be used as it is for further analysis.

In binwalk, -E switch is used to find the entropy of the firmware[55].

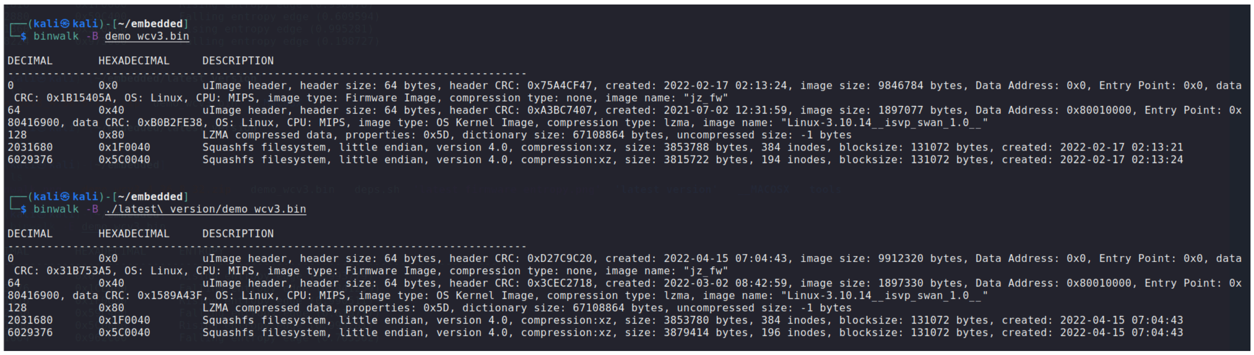


Figure E-1: binwalk -B signature of demo_wcv3_4.36.8.32 and demo_wcv3_4.36.9.131

From the above E-1 image, we can see that the kernel version remains the same. Due to the security fix, the size of the firmware is increased in the latest firmware.

Architecture: MIPS
Endianness: little
Kernel: Linux-3-10.14__isvp_swan_1.0
Compression type: LZMA
Image name: jz_fw

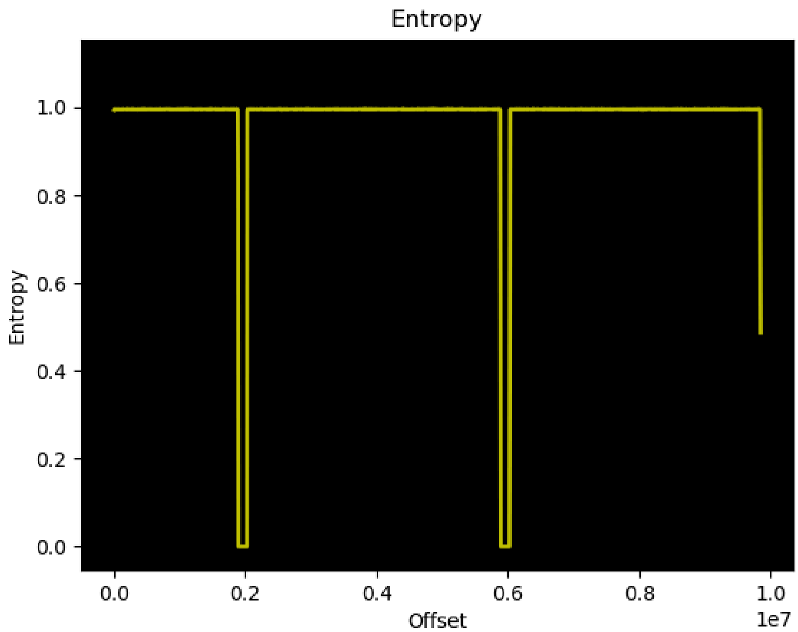


Figure E-2: binwalk -E demo_wcv3_4.36.8.32

The image E-2 is generated on executing binwalk -E demo_wcv3_4.36.8.32 command and On executing binwalk -E demo_wcv3_4.36.8.32 command image E-3 gets generated. Based on the analysis, we could see that the entropy of the firmware image is near 1 which means that the firmware is highly compressed. More numbers 0x00 were together in the firmware and it was seen in the same firmware twice, due to this the firmware experienced a low entropy.

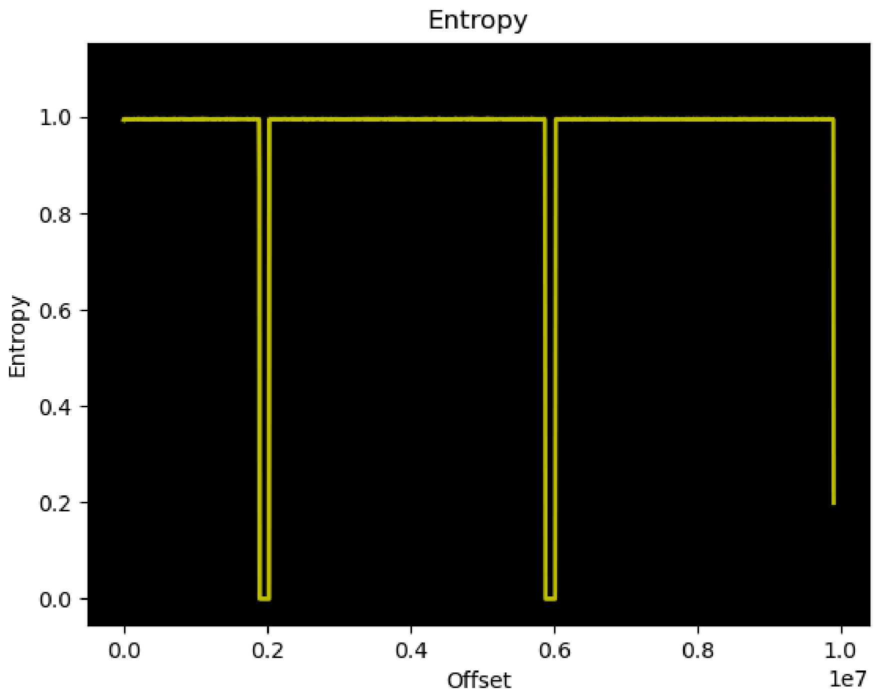
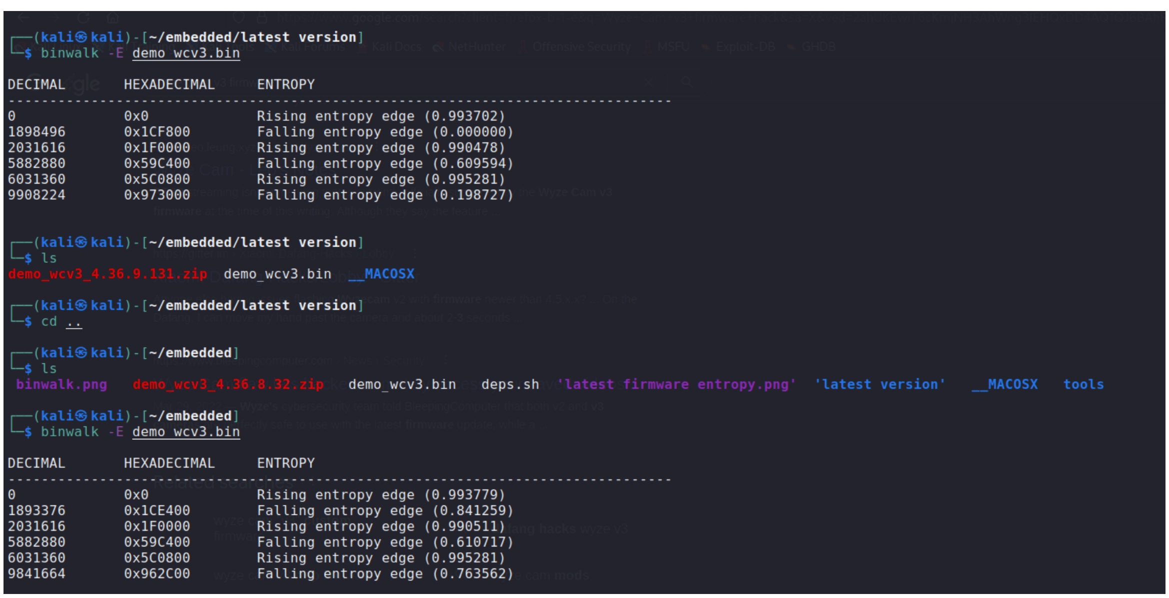


Figure E-3: binwalk -E demo_wcv3_4.36.9.131

When both the firmware are compared, we could infer that the entropy of the latest firmware is less than the previous version.



```
┌──(kali㉿kali)-[~/embedded/latest version]
└─$ binwalk -E demo_wcv3.bin

DECIMAL       HEXADECIMAL     ENTROPY
--------------------------------------------------------------------------------
0             0x0             Rising entropy edge (0.993702)
1898496       0x1CF800        Falling entropy edge (0.000000)
2031616       0x1F0000        Rising entropy edge (0.990478)
5882880       0x59C400        Falling entropy edge (0.609594)
6031360       0x5C0800        Rising entropy edge (0.995281)
9908224       0x973000        Falling entropy edge (0.198727)

┌──(kali㉿kali)-[~/embedded/latest version]
└─$ ls
demo_wcv3_4.36.9.131.zip  demo_wcv3.bin  __MACOSX

┌──(kali㉿kali)-[~/embedded/latest version]
└─$ cd ..

┌──(kali㉿kali)-[~/embedded]
└─$ ls
 binwalk.png   demo_wcv3_4.36.8.32.zip   demo_wcv3.bin   deps.sh  'latest firmware entropy.png'  'latest version'   __MACOSX   tools

┌──(kali㉿kali)-[~/embedded]
└─$ binwalk -E demo_wcv3.bin

DECIMAL       HEXADECIMAL     ENTROPY
--------------------------------------------------------------------------------
0             0x0             Rising entropy edge (0.993779)
1893376       0x1CE400        Falling entropy edge (0.841259)
2031616       0x1F0000        Rising entropy edge (0.990511)
5882880       0x59C400        Falling entropy edge (0.610717)
6031360       0x5C0800        Rising entropy edge (0.995281)
9841664       0x962C00        Falling entropy edge (0.763562)
```

Figure E-4: Entropy comparison

Pixd

Pixd is a tool based on hexdump and hexd, which uses a color palette to do the visualization of the firmware data[56]. This tool can only be used to find the type of the architecture, address, and its color code, determine the region where it has 0x00 values (black region), and can also be used for comparing two firmware.

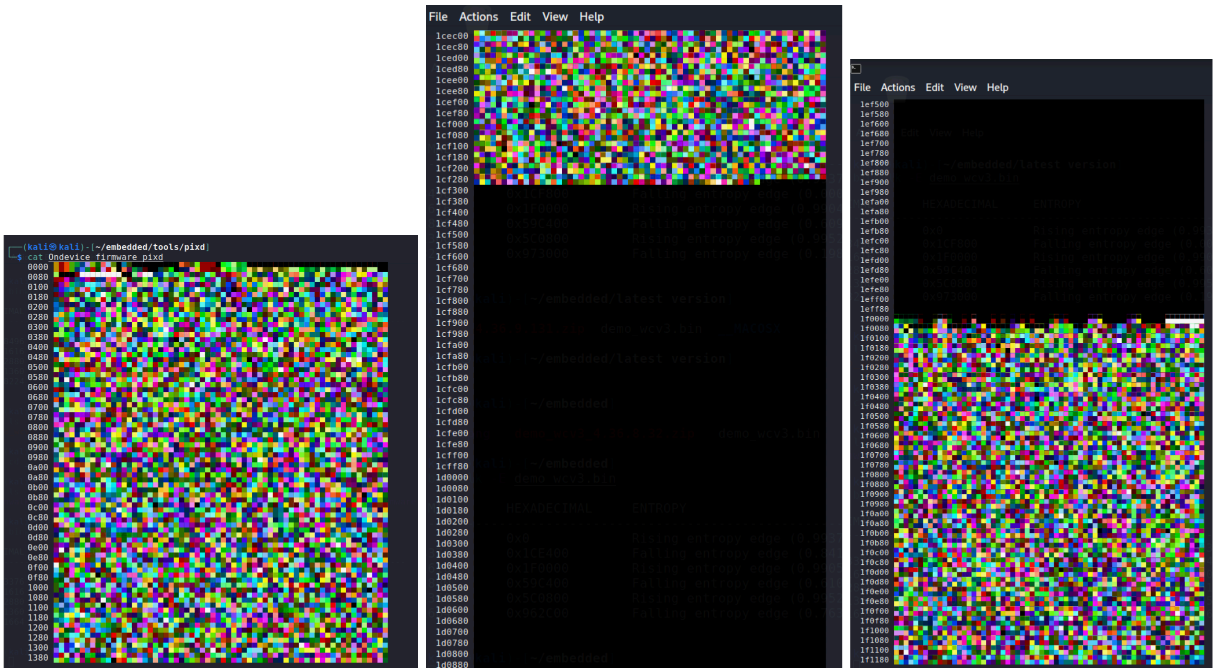


Figure E-5a: pixd for demo_wcv3_4.36.8.32 firmware

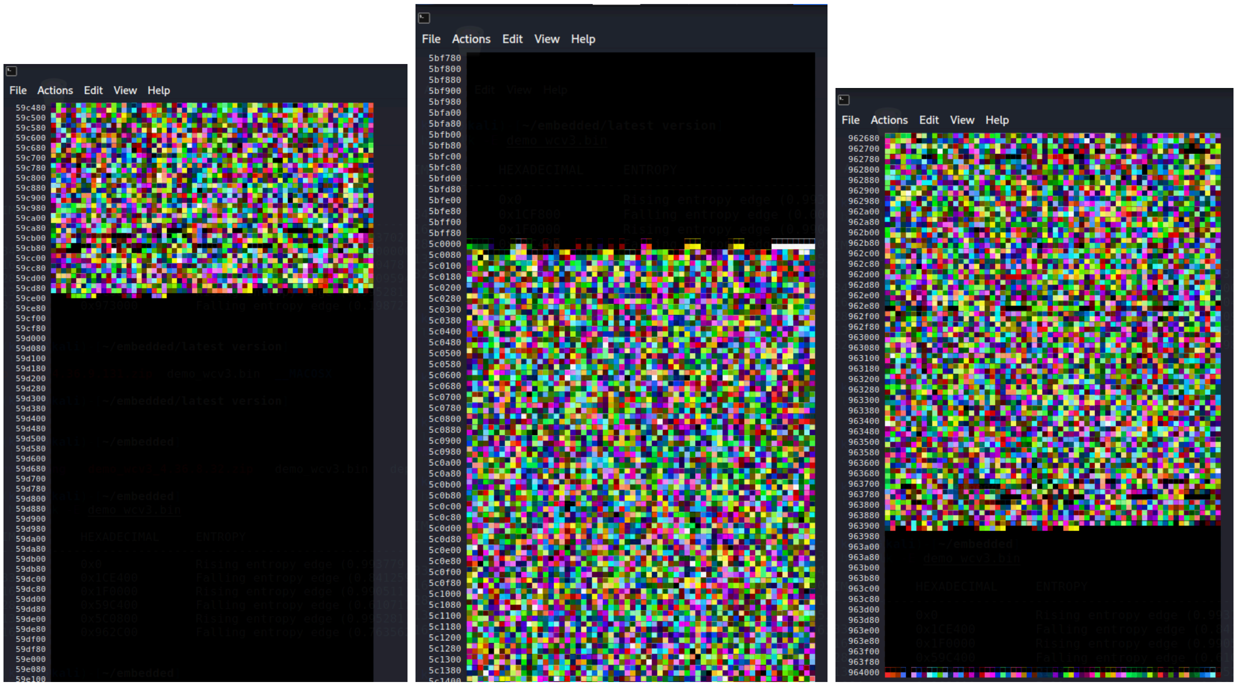


Figure E-5b: pixd for demo_wcv3_4.36.8.32 firmware

Figure E-5a,5b shows the output from executing the pixd command on firmware. There are 3 black regions on the generated output image.

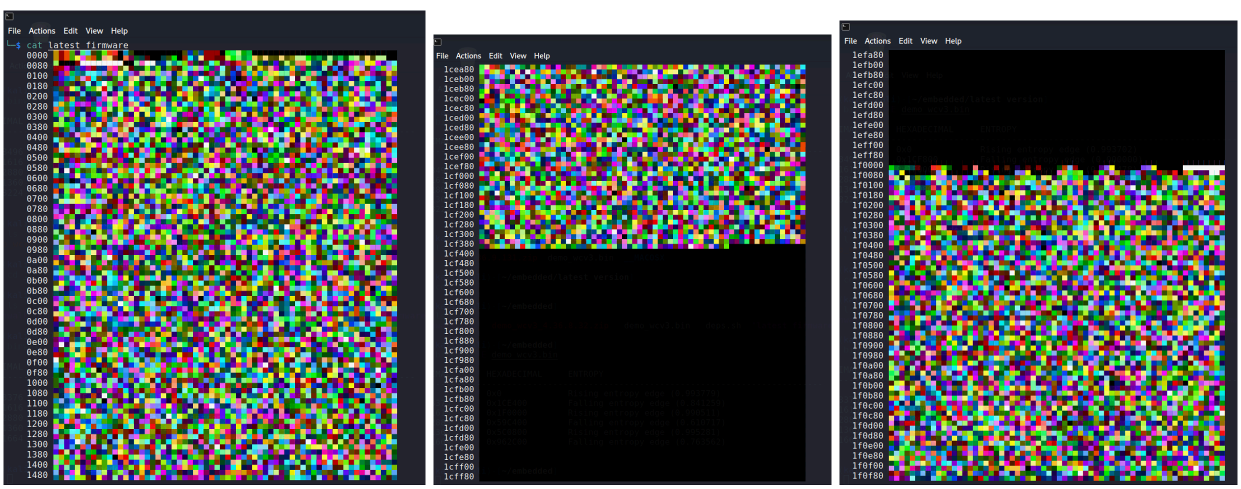


Figure E-6a: pixd for demo_wcv3_4.36.9.131 firmware

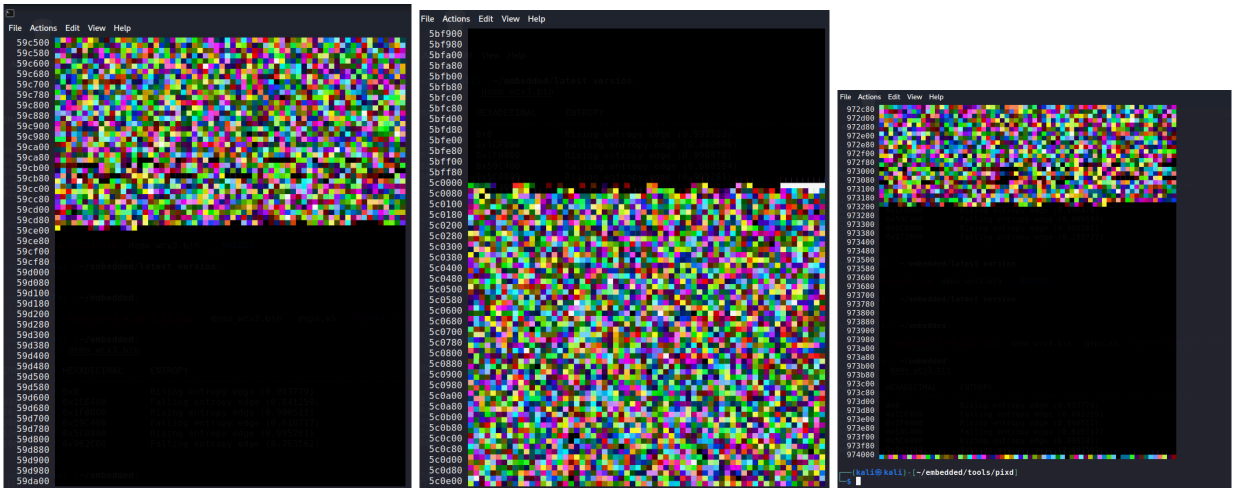


Figure E-6b: pixd for demo_wcv3_4.36.9.131 firmware

Figure E-6a,6b shows the output from executing the pixd command on firmware. There are 3 black regions on the generated output image.

Analyzing the image visually, we can conclude that both are compressed and have different values in their file.

Binvis

Binvis, is a tool used to visualize the files. This tool uses space-filling curves to generate the image [57]. The Pink region on the generated image represents high entropy and the black region represents low entropy. Since they generate unique patterns, they can also be used to find if the firmware is modified[58].

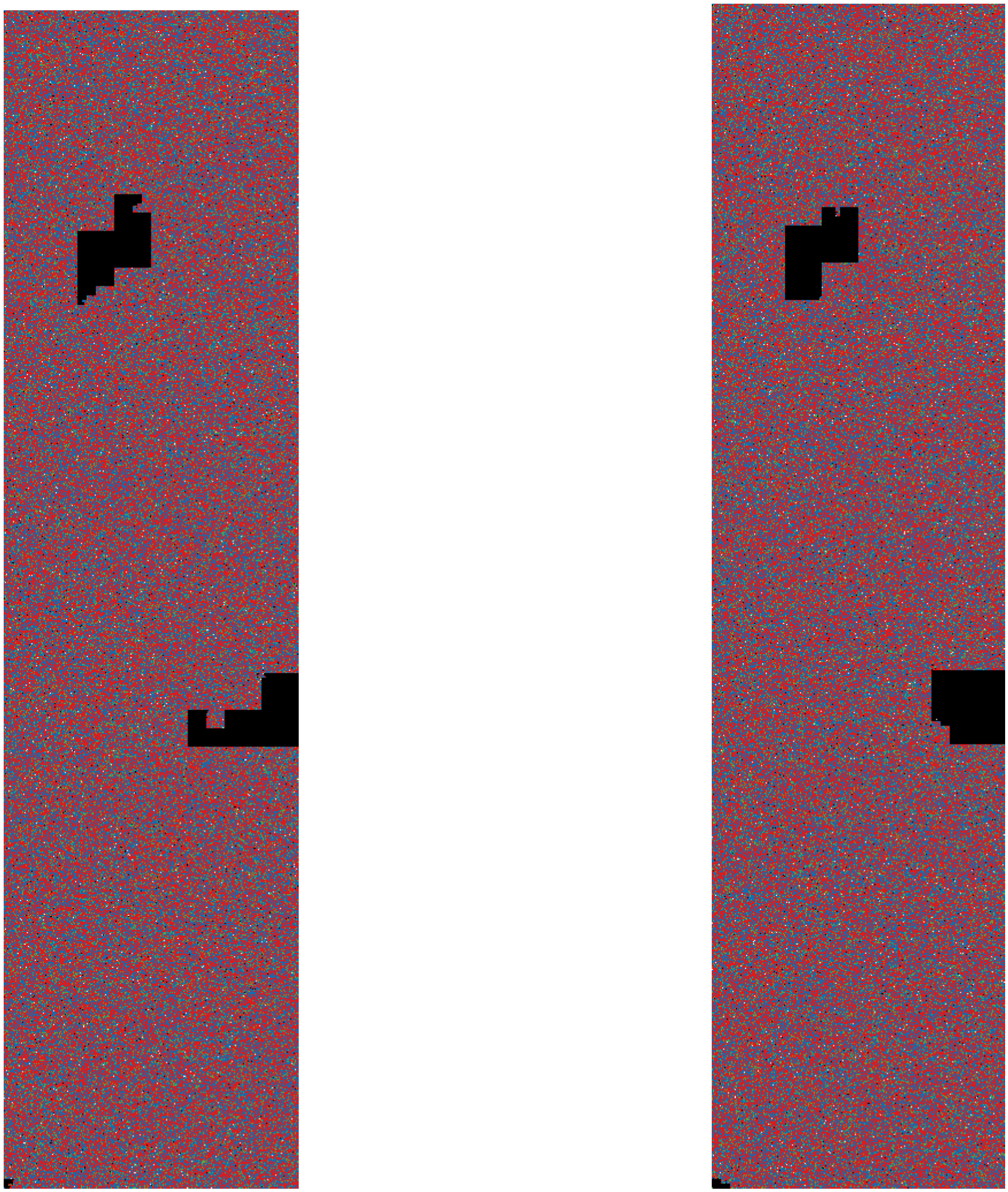


Figure E-7: binvis for demo_wcv3_4.36.8.32 and binvis demo_wcv3_4.36.9.131

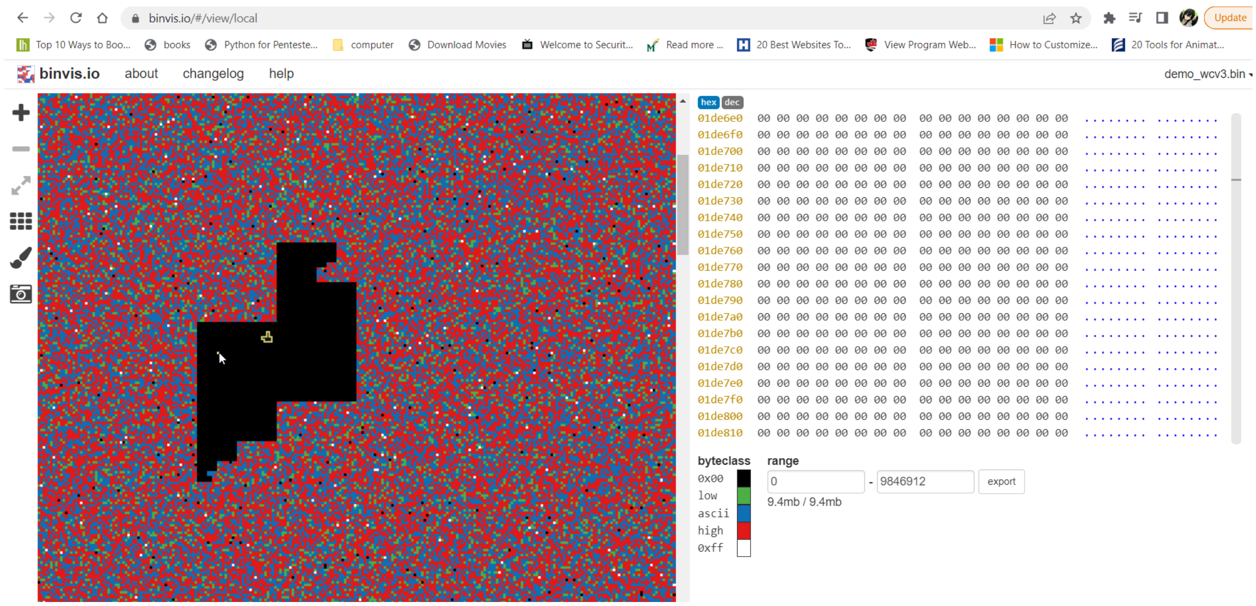


Figure E-8: binvis for demo_wcv3_4.36.8.32

This tool also shows us the hex value, address, and entropy. As can be seen in the above image the black region represents a 0x00 value and also this region has the lowest entropy.

Bin2bmp

Among the list of visualizing tools, bin2bmp is a tool that is developed in python[59]. This tool also converts binary data into graphical form. The analysis of the binary can be difficult as it requires scaling, and ther eis possibility that the image can get distorted.

Figure E-9: binvis for demo_wcv3_4.36.8.32 and binvis demo_wcv3_4.36.9.131

Port Scanning using nmap

Port scanning is a technique that is used to find the open ports of a particular device. One of the most common free and open-source tools used for port scanning is nmap [60]. This tool helps us determine the OS, service running on the open ports, version of the service, protocol type, vulnerable ports, and many others.

We connected the camera to the network by performing an initial setup. We can determine the IP address of the camera using a command like nmap, fping, ping. The IP address can also be found using the Wyze IOS application.

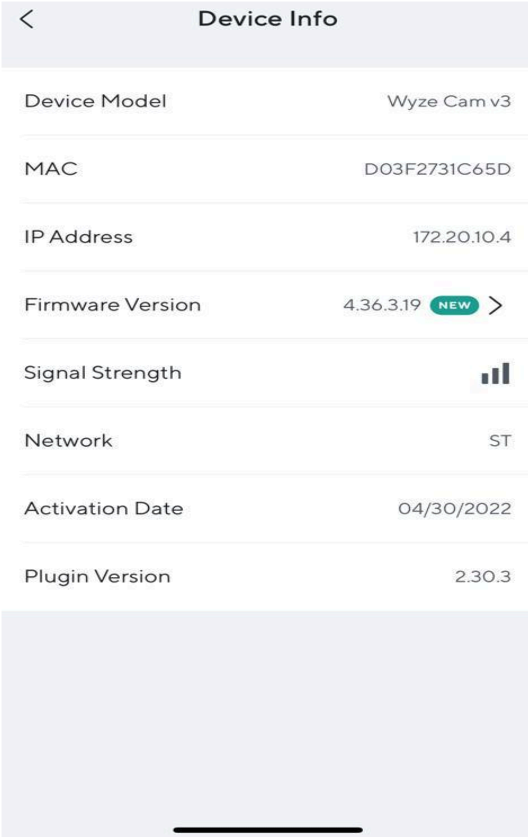


Figure E-10: Wyze cam application - device info

As can be seen from the figure E-10, we found that the camera has an IP address of 172.20.10.4 from the device info page and executed below command to find open tcp ports.

.\nmap.exe -p- -T4 -Pn -vv 172.20.10.4

```
# Nmap 7.92 scan initiated Mon May  9 18:26:44 2022 as: C:\\Users\\suman\\Downloads\\nmap-7.92-win32\\nmap-7.92\\nmap.exe -p- -T4 -Pn -vv -oN wyze_cam_3_on_device_tcp 172.20.10.4
Nmap scan report for 172.20.10.4
Host is up, received arp-response (0.014s latency).
Scanned at 2022-05-09 18:26:55 Eastern Daylight Time for 82s
All 65535 scanned ports on 172.20.10.4 are in ignored states.
Not shown: 65535 closed tcp ports (reset)
MAC Address: D0:3F:27:31:C6:5D (Wyze Labs)

Read data files from: C:\Users\suman\Downloads\nmap-7.92-win32\nmap-7.92
# Nmap done at Mon May  9 18:28:17 2022 -- 1 IP address (1 host up) scanned in 93.45 seconds
```

Figure E-11: nmap - TCP - before update

From the above image, we can conclude that there are no open TCP ports on the device that are used for communication

As we already know that most streaming services use UDP for their communication, we executed the below command to find open UDP ports.

.\nmap.exe -T4 -vv -sU 172.20.10.4

```
# Nmap 7.92 scan initiated Mon May  9 18:40:37 2022 as: C:\\Users\\suman\\Downloads\\nmap-7.92-win32\\nmap-7.92\\nmap.exe -T4 -vv -sU -oN wyze_cam_3_on_device_udp 172.20.10.4
Warning: 172.20.10.4 giving up on port because retransmission cap hit (6).
Increasing send delay for 172.20.10.4 from 100 to 200 due to 11 out of 12 dropped probes since last increase.
Increasing send delay for 172.20.10.4 from 200 to 400 due to 11 out of 11 dropped probes since last increase.
Increasing send delay for 172.20.10.4 from 400 to 800 due to 11 out of 11 dropped probes since last increase.
Nmap scan report for 172.20.10.4
Host is up, received arp-response (0.0066s latency).
Scanned at 2022-05-09 18:40:48 Eastern Daylight Time for 1075s
Not shown: 979 closed udp ports (port-unreach)
PORT      STATE         SERVICE       REASON
53/udp    open|filtered domain        no-response
68/udp    open|filtered dhcpc         no-response
120/udp   open|filtered cfdptkt       no-response
135/udp   open|filtered msrpc         no-response
3456/udp  open|filtered IISrpc-or-vat no-response
4444/udp  open|filtered krb524        no-response
5555/udp  open|filtered rplay         no-response
16947/udp open|filtered unknown       no-response
17207/udp open|filtered unknown       no-response
21247/udp open|filtered unknown       no-response
21261/udp open|filtered unknown       no-response
21556/udp open|filtered unknown       no-response
23608/udp open|filtered unknown       no-response
24279/udp open|filtered unknown       no-response
26720/udp open|filtered unknown       no-response
41446/udp open|filtered unknown       no-response
42056/udp open|filtered unknown       no-response
49154/udp open|filtered unknown       no-response
49178/udp open|filtered unknown       no-response
49213/udp open|filtered unknown       no-response
57172/udp open|filtered unknown       no-response
MAC Address: D0:3F:27:31:C6:5D (Wyze Labs)

Read data files from: C:\Users\suman\Downloads\nmap-7.92-win32\nmap-7.92
# Nmap done at Mon May  9 18:58:43 2022 -- 1 IP address (1 host up) scanned in 1086.11 seconds
```

Figure E-12: nmap - UDP - before update

We could notice that there are few ports in open|filtered status. We cannot concretely conclude that these ports are open for communication as we don’t have a mechanism to check UDP connection is established or not.

Since the device is still running the demo_wcv3_4.36.8.32 version of firmware, there is a chance that a new port might open during an update and if new services are added to the device. We could not capture the firmware update packets in Wireshark [61] as it requires a network adapter in monitor mode.

After updating the firmware to demo_wcv3_4.36.9.131, we executed nmap command to find if there is any change in the open ports.

```
# Nmap 7.92 scan initiated Tue May 10 06:50:11 2022 as: C:\\Users\\suman\\Downloads\\nmap-7.92-win32\\nmap-7.92\\nmap.exe -p- -T4 -Pn -vv -oN wyze_cam_3_Latest 172.20.10.4
Nmap scan report for 172.20.10.4
Host is up, received arp-response (0.0083s latency).
Scanned at 2022-05-10 06:50:24 Eastern Daylight Time for 64s
All 65535 scanned ports on 172.20.10.4 are in ignored states.
Not shown: 65535 closed tcp ports (reset)
MAC Address: D0:3F:27:31:C6:5D (Wyze Labs)

Read data files from: C:\Users\suman\Downloads\nmap-7.92-win32\nmap-7.92
# Nmap done at Tue May 10 06:51:28 2022 -- 1 IP address (1 host up) scanned in 76.78 seconds
```

Figure E-13: nmap - TCP - After update

After a successful update, we could see that no new TCP ports were opened but, on the UDP scan, we could see it has detected a few more ports. Also, a few ports were closed after the update. The image below shows the additional ports that are in open|filtered status after the firmware update.

```
# Nmap 7.92 scan initiated Tue May 10 07:16:09 2022 as: C:\\Users\\suman\\Downloads\\nmap-7.92-win32\\nmap-7.92\\nmap.exe -T4 -vv -sU -oN wyze_cam_3_latest_udp_try 172.20.10.4
Increasing send delay for 172.20.10.4 from 50 to 100 due to 11 out of 12 dropped probes since last increase.
Increasing send delay for 172.20.10.4 from 100 to 200 due to 11 out of 13 dropped probes since last increase.
Increasing send delay for 172.20.10.4 from 400 to 800 due to 11 out of 11 dropped probes since last increase.
Warning: 172.20.10.4 giving up on port because retransmission cap hit (6).
Nmap scan report for 172.20.10.4
Host is up, received arp-response (0.0048s latency).
Scanned at 2022-05-10 07:16:21 Eastern Daylight Time for 1038s
Not shown: 958 closed udp ports (port-unreach)
PORT      STATE         SERVICE        REASON
67/udp    open|filtered dhcps          no-response
512/udp   open|filtered biff           no-response
1007/udp  open|filtered unknown        no-response
1067/udp  open|filtered instl_boots    no-response
1069/udp  open|filtered cognex-insight no-response
1234/udp  open|filtered search-agent   no-response
3296/udp  open|filtered rib-slm        no-response
5353/udp  open|filtered zeroconf       no-response
5555/udp  open|filtered rplay          no-response
6001/udp  open|filtered X11:1          no-response
8181/udp  open|filtered unknown        no-response
8900/udp  open|filtered jmb-cds1       no-response
9199/udp  open|filtered unknown        no-response
16548/udp open|filtered unknown        no-response
17331/udp open|filtered unknown        no-response
17490/udp open|filtered unknown        no-response
17573/udp open|filtered unknown        no-response
18081/udp open|filtered unknown        no-response
18228/udp open|filtered unknown        no-response
19039/udp open|filtered unknown        no-response
19504/udp open|filtered unknown        no-response
19792/udp open|filtered unknown        no-response
20019/udp open|filtered unknown        no-response
21167/udp open|filtered unknown        no-response
22055/udp open|filtered unknown        no-response
22105/udp open|filtered unknown        no-response
22109/udp open|filtered unknown        no-response
23965/udp open|filtered unknown        no-response
24511/udp open|filtered unknown        no-response
```

```
24910/udp open|filtered unknown        no-response
31365/udp open|filtered unknown        no-response
34570/udp open|filtered unknown        no-response
37144/udp open|filtered unknown        no-response
38498/udp open|filtered unknown        no-response
40724/udp open|filtered unknown        no-response
40847/udp open|filtered unknown        no-response
45247/udp open|filtered unknown        no-response
49174/udp open|filtered unknown        no-response
49209/udp open|filtered unknown        no-response
61550/udp open|filtered unknown        no-response
62154/udp open|filtered unknown        no-response
MAC Address: D0:3F:27:31:C6:5D (Wyze Labs)

Read data files from: C:\Users\suman\Downloads\nmap-7.92-win32\nmap-7.92
# Nmap done at Tue May 10 07:33:39 2022 -- 1 IP address (1 host up) scanned in 1050.34 seconds
```

Figure E-14: nmap - UDP scan - After update

It was challenging to analyze the open ports with the limited timeline, so we left it for future action.

Binary Analysis of jz_fw.bin

We removed the Linux filesystem from the binary, leaving only a binary firmware package called ‘jz_fw.bin’. This binary was examined using a graphical version of Radare2 called Cutter. Cutter requires the binary, as well as other clues, to help it disassemble the code. The information provided to Cutter is shown in Figure A-53.

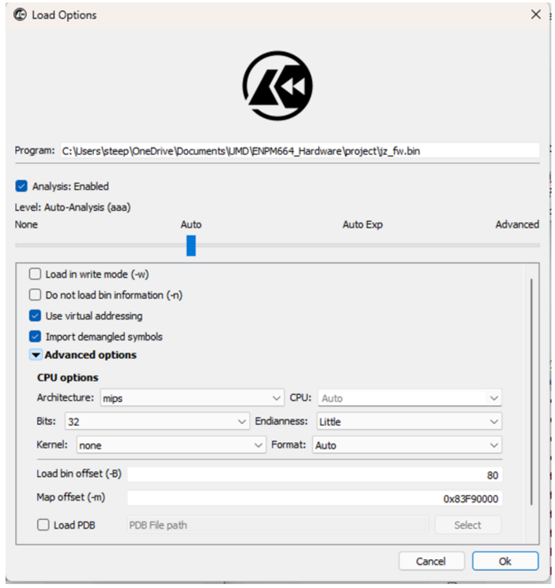


Figure A-53: Cutter setup

The ‘load bin offset’ was taken from the binwalk output, and the ‘map offset’ was retrieved from the memory map of Figure A-15. A sample of the disassembled code is shown in Figure A-54. Cutter did understand many of the instructions, identifying them as simply “invalid”. Additionally, it was noted that some of the function addresses were well above the limit of 0x8400_0000 as shown in the memory map in Figure A-15. Larger functions (~1000 instructions or longer) were interpreted as nop sleds, or branches to empty functions.

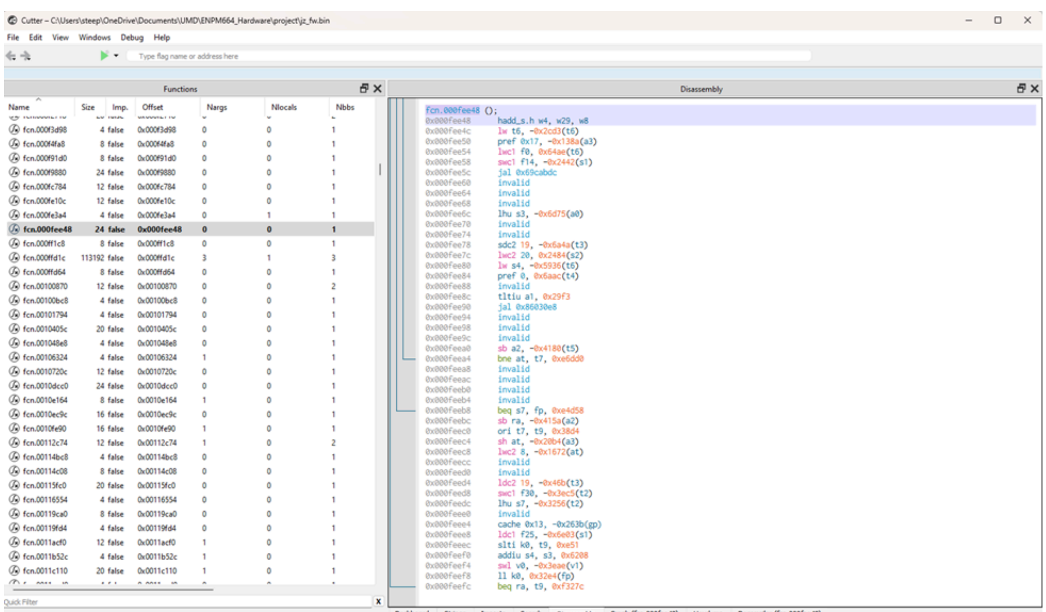


Figure A-54: Example of Cutter disassembly of jz_fw.bin

Based on these issues, we provided different parameters to Cutter and re-ran the analysis. Variations of parameters attempted include:

Architecture: mips, mips.gnu
Endianness: little, big
Kernel: Linux, none
Format: bootimg, Auto

Providing different parameters did result in different output. However, none of these input changes resolved the issues.

Angr-Management is a similar tool used for binary analysis, and requires very similar parameter inputs. The inputs were varied as;

Architecture: MIPS32, MIPS32/64
Endianness: little, big

Providing different parameters resulted in different output. However, none of these input changes resolved the issues. See Figure A-55 for an example of the output.

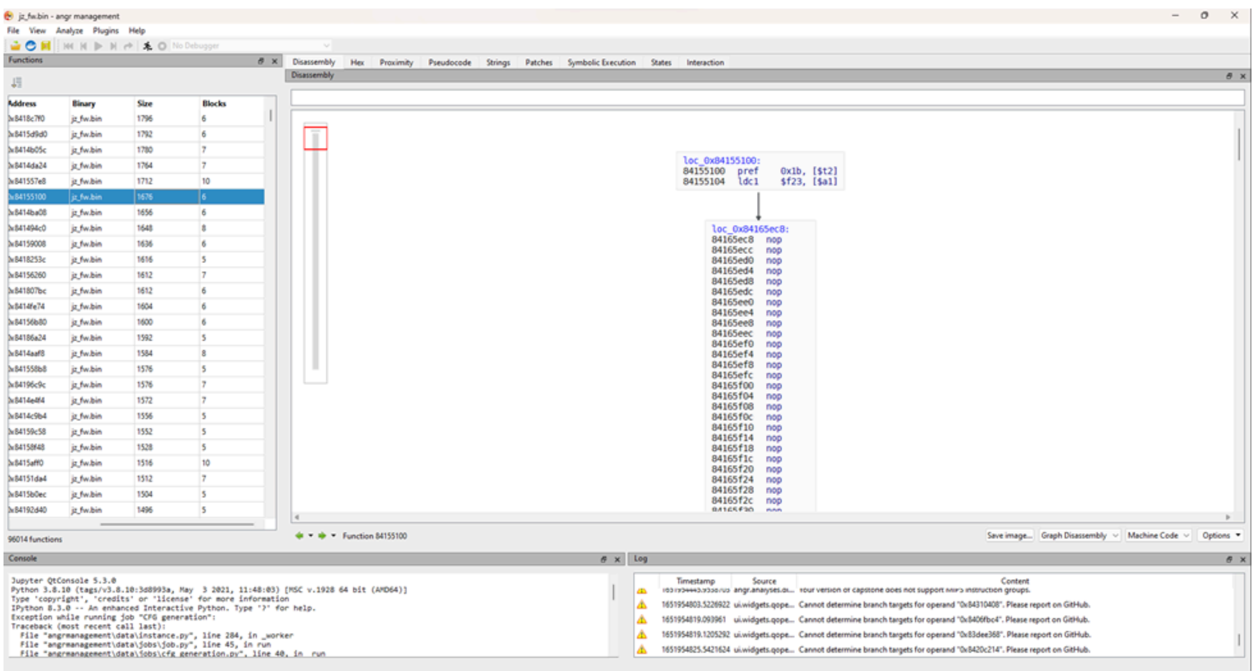


Figure A-55: Example output from Angr-Management

We also tried Radare2 using the Command Line Interface (CLI). This provided the most flexibility while improving the granularity of the inputs. ‘e’ Variables describing the architecture can be defined from the command line. See Figure A-56 for an example.

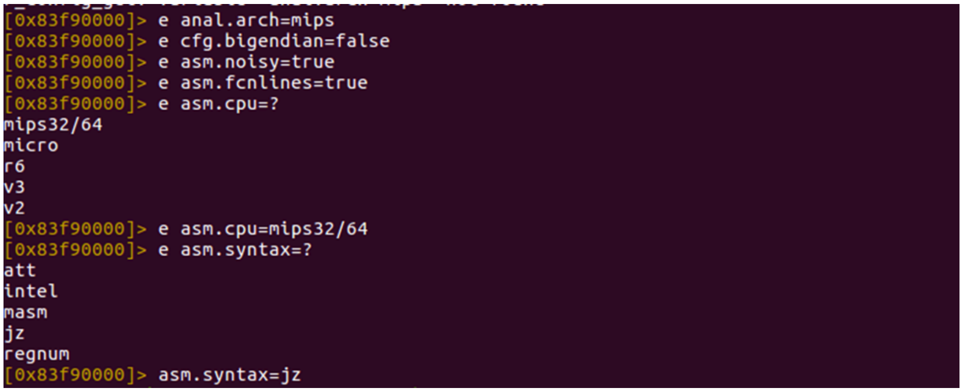


Figure A-56: Example showing command line instructions to define the CPU architecture

Help can be provided for a particular variable, as shown in Figure A-57 for asm.syntax (assembly syntax).

Figure A-57: Getting help setting ‘e’ variables in radare2

Figure A-58 shows additional variable used to define the architecture prior to performing analysis (aaa).

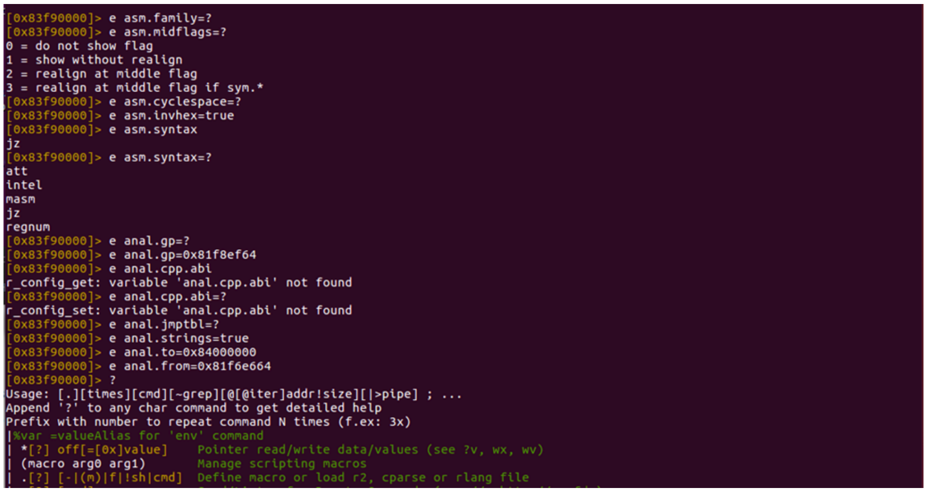


Figure A-58: Additional examples of setting up ‘e’ variables in radare2

The MIPS architecture proved to be a challenge to analyze using these tools. Other tools are available, like Ghidra or IDA, but weren’t attempted due to time constraints. Pulling the thread on these binary analysis tools is left for future action.

Conclusions

During our analysis we uncovered several possible issues related to poor coding practice and existing CVEs. Tools like binwalk, radare2, Cutter, and Ghidra were useful in performing this analysis. The goal of this project was to uncover vulnerabilities in the device, and we feel we have achieved this objective. Our stretch goal was to exploit these vulnerabilities, and unfortunately we did not get that far. We leave that for further research.

The two possible vulnerabilities that stood out to us in the iCamera binary are the recvfrom() call that reads ICMP packets over a raw socket, and the lack of full path specification for the programs passed to the system() call. If the recvfrom() call is truly vulnerable then remote code execution may be possible, and as a result a shell could be achieved on the Wyze Cam V3. If the system() call was also truly vulnerable to PATH hijacking then we could theoretically escalate our privileges to a root shell. Although it is possible that the other notable findings (strcpy, strncpy, fread, sprintf) could lead to a vulnerability, we determined that it is unlikely because even if they do use some of their parameters unsafely, they don’t appear to interact with attacker controlled input.

Our firmware analysis showed that many of the vulnerable CVEs would either need direct physical network access, access to the firmware within the supply chain, or inject utilities to take advantage of one of the many unpatched CVEs. The risk of such an attack was assessed as low-to-medium.

Additional analysis can be performed based on the work described in this paper. Several items are missing from the memory map (GPIO, peripherals…) that should be added by a future effort. The Ingenic T31 SoC and the XBurst1 deserve closer scrutiny, as well as the role of the RISC-V processor in the boot process. The u-boot process occurs very quickly, and the provided time to interrupt the process was minimal. Attempts to interrupt the process failed. There may be other approaches that have a higher likelihood of success. Firmadyne successfully emulates the hardware, so the binary can be executed and analyzed on a laptop. We showed that we can change the root password but we could not flash the repacked firmware to the hardware. This should be relatively easy to investigate given more time than we have for this paper. The information we provided to setup the amdgpu drivers and hashcat should enable the ability to crack the linux password, maybe with a more powerful gpu or cheap cloud service. Wyze’s passwords have been 8-10 characters in the past, making this a doable effort. Various firmware visual analysis tools were used and results were compared. It was also found that the visual analysis can aid in the process of firmware analysis. We also investigated if any new ports are opened after updating the firmware using nmap.

References

[1] “Wyze cam v3,” *Wyze*. [Online]. Available: https://wyze.com/wyze-cam.html. [Accessed: 11-Mar-2022].

[2] “Security & trust,” *Wyze*. [Online]. Available: https://wyze.com/wyze-security-and-trust. [Accessed: 11-Mar-2022].

[3] “Disclosure,” *Networkcamerabug.info*. [Online]. Available: https://networkcamerabug.info/. [Accessed: 11-Mar-2022].

[4] FiveLeavesLeft, “WyzeCameraLiveStream: Hack to allow live streaming from wyze cameras to vlc or mpv on your desktop,” *Github*. [Online]. Available: https://github.com/FiveLeavesLeft/WyzeCameraLiveStream. [Accessed: 11-Mar-2022].

[5] mrlt8, “docker-wyze-bridge: RTMP/RTSP/HLS bridge for Wyze cams in a docker container,” *Github*. [Online]. Available: https://github.com/mrlt8/docker-wyze-bridge. [Accessed: 11-Mar-2022].

[6] HclX, “WyzeHacks: Hacks I discovered allowing Wyze camera owners to do customizations,” *Github*. [Online]. Available: https://github.com/HclX/WyzeHacks. [Accessed: 11-Mar-2022].

[7] Gwendolyn, “Wyze Cam RTSP,” *Wyze.com*, 05-Apr-2022. [Online]. Available: https://support.wyze.com/hc/en-us/articles/360026245231-Wyze-Cam-RTS. [Accessed: 10-May-2022].

[8] Mitre, “CVE-2019-9564,” *Mitre.org*. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-9564. [Accessed: 06-Apr-2022].

[9] Mitre, “CVE-2019-12266,” *Mitre.org*. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-12266. [Accessed: 06-Apr-2022].

[10] C. Davenport, “Wyze knew for years that hackers could remotely access its cameras, but didn’t tell anyone,” *XDA*, 31-Mar-2022. [Online]. Available: https://www.xda-developers.com/wyze-security-vulnerability-2022/. [Accessed: 05-Apr-2022].

[11] Gwendolyn, “Release Notes & Firmware,” *Wyze.com*. [Online]. Available: https://support.wyze.com/hc/en-us/articles/360024852172-Release-Notes-Firmware. [Accessed: 07-Apr-2022].

[12] Bitdefender, *Vulnerabilities Identified in Wyze Cam IoT Device*. Bitdefender. [Online]. Available: https://www.bitdefender.com/files/News/CaseStudies/study/413/Bitdefender-PR-Whitepaper-WCam-creat5991-en-EN.pdf?clickid=yiwUoVT27xyIRgqWdXzvzy4eUkGQNnUhNTJdyg0&irgwc=1&MPid=10078. [Accessed: 07-Apr-2022].

[13] B. Toulas, “Wyze Cam flaw lets hackers remotely access your saved videos,” *BleepingComputer*, 29-Mar-2022. [Online]. Available: https://www.bleepingcomputer.com/news/security/wyze-cam-flaw-lets-hackers-remotely-access-your-saved-videos/. [Accessed: 08-Apr-2022].

[14] Certcc, “Trommel: TROMMEL: Sift Through Embedded Device Files to Identify Potential Vulnerable Indicators,” *Github*. [Online]. Available: https://github.com/CERTCC/trommel. [Accessed: 06-Apr-2022].

[15] C. Smith, “Firmwalker,” *Github*. [Online]. Available: https://github.com/craigz28/firmwalker. [Accessed: 06-Apr-2022].

[16] “r/wyzecam - Wyze still vulnerable to krack?,” *reddit*. [Online]. Available: https://www.reddit.com/r/wyzecam/comments/aaao44/wyze_still_vulnerable_to_krack/. [Accessed: 10-May-2022].

[17] Gullo, K., Rodriguez, K., Romero, C., Reitman, Rainey, Tsukayama, H., Kelley, J., Mir, R., Greenberg, W., Jue, A., & Rathi, M., “Coders’ rights project Reverse Engineering FAQ,” *Electronic Frontier Foundation*. [Online]. Available: https://www.eff.org/issues/coders/reverse-engineering-faq. [Accessed: 08-Apr-2022].

[18] A. Robertson, “The US Copyright Office just struck a blow supporting the right to repair,” *The Verge*, 27-Oct-2021. [Online]. Available: https://www.theverge.com/2021/10/27/22747310/us-copyright-office-dmca-section-1201-exemption-rulemaking-report. [Accessed: 08-Apr-2022].

[19] Dongguan Dongdian Testing Service Co., Ltd, *Report No.: DDT-R21050704-1E2*. [Online]. Available: https://fcc.report/FCC-ID/2AUIUWYZEC3B/5289930.pdf. [Accessed: 08-May-2022].

[20] “Ingenic Semiconductor_M200 M150 JZ4780 JZ4775 JZ4760B,” *Ingenic*. [Online]. Available: http://www.ingenic.com.cn/en/?product/id/20.html. [Accessed: 15-Apr-2022].

[21] J.-L. A. (CNXSoft), “Ingenic T31 AI video processor combines MIPS & RISC-V cores,” *CNX Software - Embedded Systems News*, 26-Apr-2020. [Online]. Available: https://www.cnx-software.com/2020/04/26/ingenic-t31-ai-video-processor-combines-xburst-1-mips-and-risc-v-lite-cores/. [Accessed: 08-May-2022].

[22] *XBurst1 CPU Core Programming Manual*. Ingenic, 2014.

[23] *XBurst1 Instruction Set Architecture MIPS extension/enhanced Unit 2 Programming Manual*. Ingenic, 2017.

[24] Firmadyne, “firmadyne: Platform for emulation and dynamic analysis of Linux-based firmware,” *Github*. [Online]. Available: https://github.com/firmadyne/firmadyne. [Accessed: 15-Apr-2022].

[25] “CVE-2012-6638,” *CVE Details*, 15-Feb-2014. [Online]. Available: https://www.cvedetails.com/cve/CVE-2012-6638/. [Accessed: 30-Apr-2022].

[26] “CVE-2013-4563,” *CVE Details*, 20-Nov-2013. [Online]. Available: http://www.cvedetails.com/cve/CVE-2013-4563. [Accessed: 30-Apr-2022].

[27] “CVE-2013-4348,” *CVE Details*, 04-Nov-2013. [Online]. Available: http://www.cvedetails.com/cve/CVE-2013-4348. [Accessed: 30-Apr-2022].

[28] “CVE-2013-7263,” *CVE Details*, 06-Jan-2014. [Online]. Available: http://www.cvedetails.com/cve/CVE-2013-7263. [Accessed: 30-Apr-2022].

[29] “CVE-2013-7281,” *CVE Details*, 08-Jan-2014. [Online]. Available: http://www.cvedetails.com/cve/CVE-2013-7281. [Accessed: 30-Apr-2022].

[30] “CVE-2013-6378,” *CVE Details*, 27-Nov-2013. [Online]. Available: http://www.cvedetails.com/cve/CVE-2013-6378. [Accessed: 30-Apr-2022].

[31] “CVE-2013-4515,” *CVE Details*, 12-Nov-2013. [Online]. Available: http://www.cvedetails.com/cve/CVE-2013-4515. [Accessed: 30-Apr-2022].

[32] “CVE-2013-4516,” *CVE Details*, 12-Nov-2013. [Online]. Available: http://www.cvedetails.com/cve/CVE-2013-4516. [Accessed: 30-Apr-2022].

[33] “CVE-2013-4587,” *CVE Details*, 14-Dec-2013. [Online]. Available: http://www.cvedetails.com/cve/CVE-2013-4587. [Accessed: 30-Apr-2022].

[34] “CVE-2013-7264,” *CVE Details*, 06-Jan-2014. [Online]. Available: http://www.cvedetails.com/cve/CVE-2013-7264. [Accessed: 30-Apr-2022].

[35] “CVE-2013-7265,” *CVE Details*, 06-Jan-2014. [Online]. Available: http://www.cvedetails.com/cve/CVE-2013-7265. [Accessed: 30-Apr-2022].

[36] “CVE-2013-7266,” *CVE Details*, 06-Jan-2014. [Online]. Available: http://www.cvedetails.com/cve/CVE-2013-7266. [Accessed: 30-Apr-2022].

[37] “CVE-2013-7267,” *CVE Details*, 06-Jan-2014. [Online]. Available: http://www.cvedetails.com/cve/CVE-2013-7267. [Accessed: 30-Apr-2022].

[38] “CVE-2013-7268,” *CVE Details*, 06-Jan-2014. [Online]. Available: http://www.cvedetails.com/cve/CVE-2013-7268. [Accessed: 30-Apr-2022].

[39] “CVE-2013-7269,” *CVE Details*, 06-Jan-2014. [Online]. Available: http://www.cvedetails.com/cve/CVE-2013-7269. [Accessed: 30-Apr-2022].

[40] “CVE-2013-7271,” *CVE Details*, 06-Jan-2014. [Online]. Available: http://www.cvedetails.com/cve/CVE-2013-7271. [Accessed: 30-Apr-2022].

[41] “CVE-2013-7270,” *CVE Details*, 06-Jan-2014. [Online]. Available: http://www.cvedetails.com/cve/CVE-2013-7270. [Accessed: 30-Apr-2022].

[42] “Understanding /etc/shadow file format on Linux,” Cyberciti.biz. [Online]. Available: https://www.cyberciti.biz/faq/understanding-etcshadow-file/. [Accessed: 10-May-2022].

[43] *SDIO Product Specification, IEEE 802.11 b/g/n 2.4 GHz 1T1R WiFi Module BT58189-2 Single Module*

[44] *Single Chip IEEE 802.11 b/g/n 1T1R WLAN With SDIO Interface Datasheet Rev 0.6,* Realtek

[45] *BCT8933 High Power Low THD+N Class T Audio Amplifier,* data sheet, Broadchip

[46] *A4003 3CH Power Management IC,* data sheet, Aerosemi

[47] *EN25QH128A (2T) 128 Megabit 3V Serial Flash Memory with 4Kbyte Uniform Sector*, EON

[48] *T31_QFN_SC4335_38 Schematic Revision 1.0,* Ingenic Semiconductor Co., LTD

[49] *XBurst Instruction Set Architecture MIPS eXternsion/enhanced Unit Programming Manual, Release Date: June 2 2017,* Ingenic Semiconductor Co., LTD

[50] *XC6219/XC6211 Series 300mA High Speed LDO Regulators with ON/OFF Switch,* TOREX

[51] *SC4335 Product Flier CMOS Image Sensor, SmartPixel-2 Series,* data sheet, SMARTSENS

[52] *Mipsdis: MIPS disassembler in the browser,* May 4 2017, [Online] Available: https://blog.loadzero.com/blog/announcing-mipsdis, Jason McSweeney [Accessed: May 3 2022]

[53] *Abhijith-Soman, "How to do firmware visual analysis," Payatu. [Online]. Available: https://payatu.com/firmware-visual-analysis. [Accessed:*

10-May-2022].

[54] *J. Walker, “Pseudorandom number sequence test program,” Fourmilab.ch. [Online]. Available: https://www.fourmilab.ch/random/. [Accessed: 10-May-2022].*

[55] *Binwalk: Firmware analysis tool.*

[56] *FireFly, pixd: Colourful visualization tool for binary files. .*

[57] *“Corte.Si,” Corte.si. [Online]. Available: https://corte.si/. [Accessed: 10-May-2022].*

[58] *“Binvis.Io,” Binvis.io. [Online]. Available: http://binvis.io/#/. [Accessed: 10-May-2022].*

[59] *“Python-datavis,” SourceForge. [Online]. Available: https://sourceforge.net/projects/bin2bmp/. [Accessed: 10-May-2022].*

[60] *“Nmap: The network mapper - Free Security Scanner,” Nmap.org. [Online]. Available: https://nmap.org/. [Accessed: 10-May-2022].*

[61] *“Wireshark · Go Deep,” Wireshark.org. [Online]. Available: https://www.wireshark.org/. [Accessed: 10-May-2022].*