WHITEPAPER

FLASHROM TO HEXEDIT TO ROOT:

A DEF CON 33 IoT Village Hardware Hacking Exercise

By Deral Heiland, Principal Security Researcher, IoT

CONTENTS

Exercise Overview 3

System boot and evaluation 5

Reading firmware from SPI flash chip 10

Flash memory dump review and edit 14

Single-user mode access 18

Mounting needed file systems and removing root password 20

Remove single and boot to root 23

What Was Learned 29

About Rapid7 30

EXERCISE OVERVIEW

The goal of this year's hands-on hardware hacking exercise was to leverage something we learned last year (single-user mode) along with exposing attendees to a few different methods and tools we have not used before, including: reading SPI flash memory with flashrom application and the Tigard multi-protocol, multi-voltage tool. Attendees used these tools and methods to gain access to a smart camera, extract firmware and manipulate that raw flash dump by using a hex editor application to alter command data passed to the kernel during boot up and place the camera into single-user mode.

With single-user mode access, we leveraged various Linux commands to mount the correct partitions and load the needed kernel module to gain access to the flash memory chip and reload the file systems to allow multi-user mode; gaining full file system access.

Once that was done, the user identified the correct shadow file (there is more than one) to delete the root password hash, copied the firmware out with flashrom, and modified it again to remove the single-user mode setting. They then wrote it back on to the camera and rebooted the system to login with root access and no password.

Within this exercise, the attendees were exposed to the following items of interest:

- Universal Asynchronous Receiver / Transmitter (UART)
- U-Boot command to pass data/commands to Kernel Init

- Hexedit: view and edit raw flash dump
- Flashrom: used to read flash memory from SPI flash chip
- Single-user mode and file system recovery

Requirements	
Hardware	• Laptop • KinetCam IoT Camera • Tigard
Software	• Terminal • Hexedit • GTKTerm • Flashrom
Documentation	• This exercise manual

> ***Note:*** *For this exercise we preconfigured the hardware used to simplify the exercise so attendees would not need to conduct any soldering. Since the hardware (IoT camera) had a Serial Peripheral Interface (SPI) flash memory chip, which would require disconnecting from the circuit for us to be able to read and write the memory, we hardwired the circuit so attendees could simply disconnect the power to the chip with a switch and connect the needed SPI reader (Tigard) to the chip using a ribbon cable. More details on the SPI memory chip are discussed later in this exercise writeup. Beside the SPI we also had a CPU which required us to do complex soldering of very small wires directly to the CPU pins. An average attendee at the DEF CON IoT Village event would not have these skills, so we simplified it by doing the soldering for them.*

Detailed images of this prep work are shown along with descriptions and are discussed throughout this exercise.

WARNING
During this exercise, never connect the camera's USB power and the ribbon cable from the Tigard to the breakout board at the same time. Doing so feeds power into the camera from two directions, which will release the magic smoke and permanently damage the camera.

System boot and evaluation

In this section of the exercise, you will validate that the FTDI serial device (Tigard) is connected to the breakout board for Universal Asynchronous Receiver Transmit (UART), and the USB cable for Tigard is connected to the training laptop. You will also be powering up the camera and monitoring the bootup process.

> ***Note:*** *The wiring for UART has been connected (soldered)to the serial pins (73, 74) on the T23 CPU. This camera's circuit board had no headers for connecting to serial and required extremely fine soldering to attach the needed wiring for serial (UART) access to the CPU. This is shown below in Figure 1.*

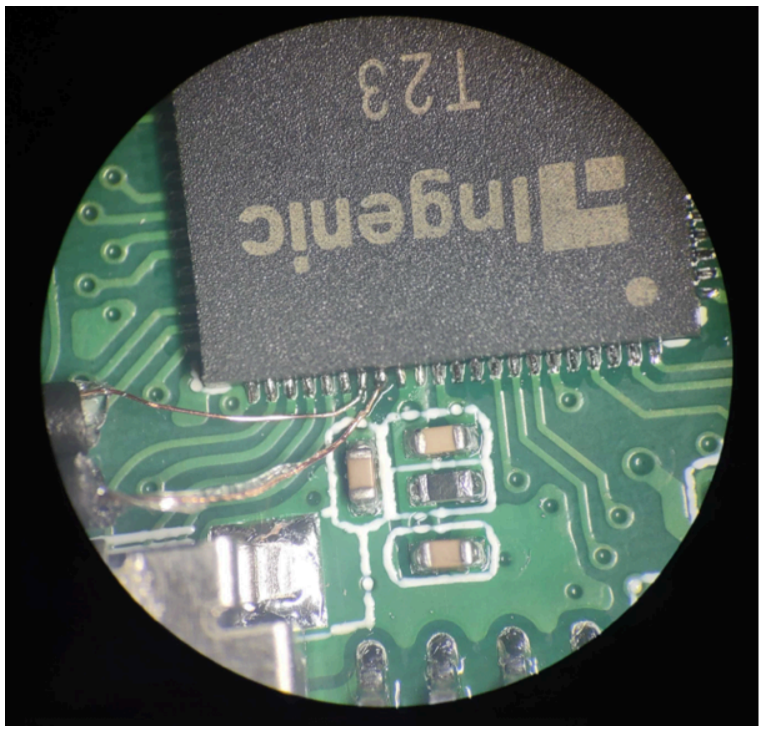


Figure 1: Serial Connections

> ***Note:*** *The Tigard (Figure 2) is a hardware hacking tool designed by Joe FitzPatrick for interacting with and reverse-engineering embedded systems. It is based on the FTDI FT2232H chip, which provides various functionalities for communication and debugging and can be purchased online from various resources.*

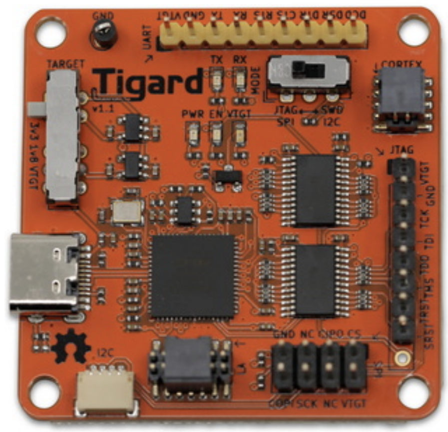


Figure 2: Tigard Debug Board

First, we will make sure the following items are connected correctly as shown below in Figure 3.

- Tigard USB is connected to laptop
- Yellow, Green, Black jumper wires are connected between breakout board and Tigard
- Ribbon cable to breakout board IS NOT CONNECTED to breakout board
- Red breakout board switch is in the on position

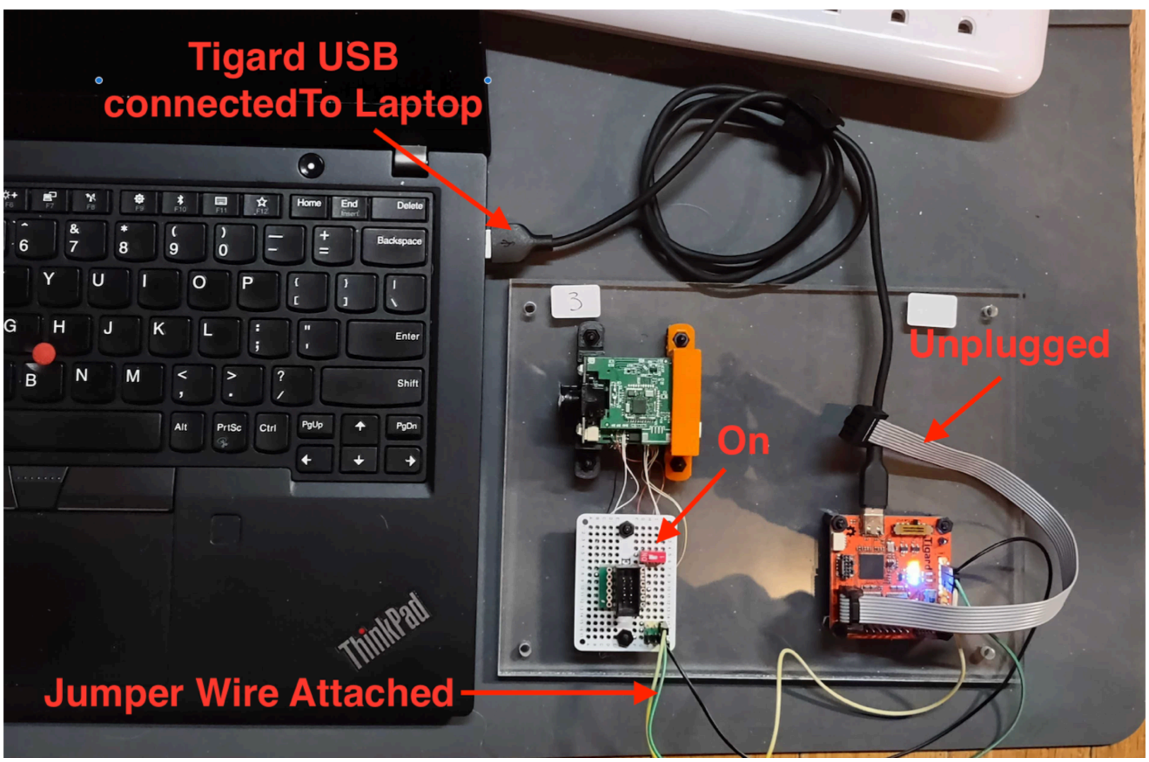


Figure 3: Initial Hardware Setup

Next, we need to start a serial terminal and configure it. For this exercise we used "gtkterm" which was installed on the Linux laptops used during the exercise. To do this, we first opened a terminal by clicking on the terminal icon in the left column of the Linux laptop desktop as shown in Figure 4.

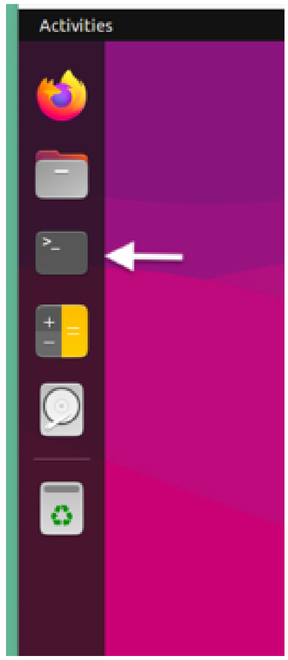


Figure 4: Terminal Icon

Once the command line terminal is open, launch the serial application “gtkterm” as root by running the following command in the terminal and when prompted for the password enter the root password as shown below in Figure 5.

sudo gtkterm

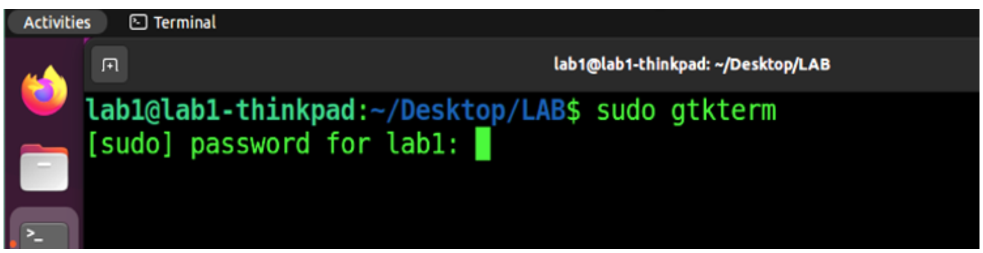


Figure 5: Launch gtkterm

Once gtkterm is running, you will need to configure gtkterm to properly communicate with the camera over UART. This is done by selecting “configuration → port” from the task bar within the gtkterm application as shown below in Figure 6 and make any necessary changes to the configuration setting if needed.

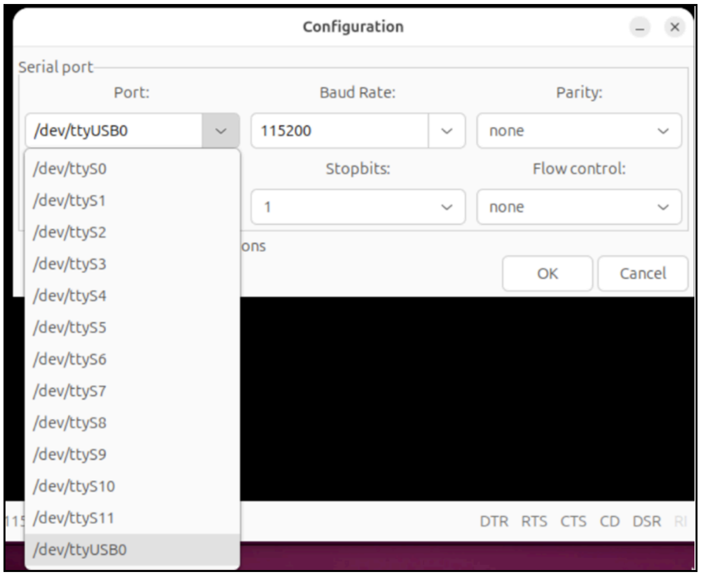


Figure 6: gtkterm Configuration

- Port: /dev/ttyUSB0
- Baud Rate: 115200
- Parity: none
- Bits: 8
- Stopbits: 1
- Flow control: none

Once the gtkterm configuration settings have been completed and confirmed you can now power ON the camera by attaching the USB plug to the camera and turning ON the power strip that the camera is attached to (Figure 7).

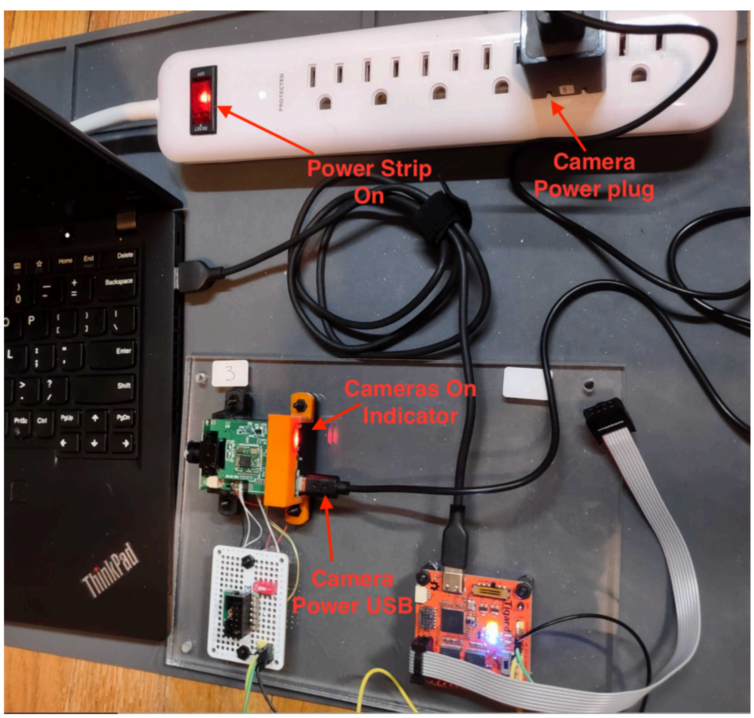
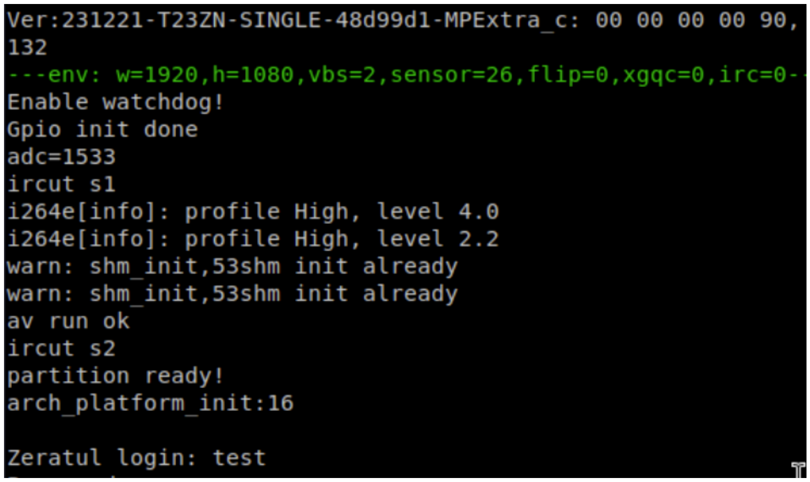


Figure 7: Hardware layout

When the camera powers up you should see activity in the gtkterm terminal screen, similar to what is shown below in Figure 8.



```
Ver:231221-T23ZN-SINGLE-48d99d1-MPExtra_c: 00 00 00 00 90,
132
---env: w=1920,h=1080,vbs=2,sensor=26,flip=0,xgqc=0,irc=0-
Enable watchdog!
Gpio init done
adc=1533
ircut s1
i264e[info]: profile High, level 4.0
i264e[info]: profile High, level 2.2
warn: shm_init,53shm init already
warn: shm_init,53shm init already
av run ok
ircut s2
partition ready!
arch_platform_init:16

Zeratul login: test
```

Figure 8: Camera Booting

If nothing appears, something may be misconfigured. First, double-check all cable connections and settings.

Before proceeding, power off the camera by turning off the power strip and disconnecting USB power from the camera (Figure 9). The power off was done first because it doesn't need to be running for the remainder of this section, and in the next section of the exercise we want to avoid damaging the device.

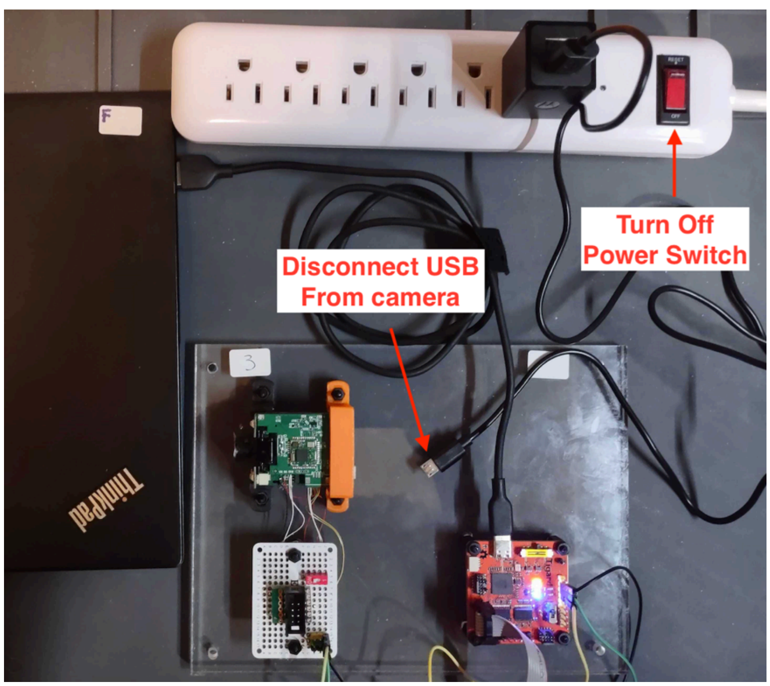


Figure 9: Camera Power Source

Next, examine the boot process within the gtkterm terminal. Looking at that data we can make some basic observations:

1) We do not see the typical U-boat process taking place. Strange! Nothing showing the boot process at all. Typically, we see a U-boot process and then if system console access is locked down that is done when the kernel load starts. But in this case no U-Boot calls are ever displayed in the console.
2) We also see no kernel calls or file systems loading. The cause of this will be identified later in the exercise.
3) We just see the system start running with a few application responses,the login prompt showing up, and maybe some debug messages associated with WiFi.

Reading firmware from SPI flash chip

In this section of the exercise, we will be reading the firmware from the Serial Peripheral Interface (SPI) flash memory chip (XM25QH64) using the open-source application flashrom.

The first thing we want to do is to make sure that the camera is powered off by **turning off the power strip and unplugging the USB power connector from the camera**, if it has not already been done.

Once we have completed removing the power source from the camera, , turn the red power switch on the break out board to the off position and then attach the keyed ribbon cable from the Tigard to the breakout board as shown below in Figure 10.

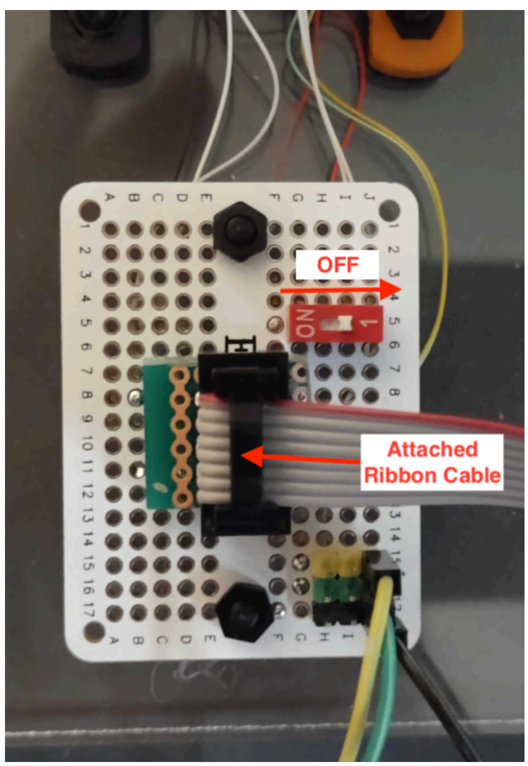


Figure 10: Configure Hardware Connection

Once that is completed, we will also need to start a second terminal by right clicking on the terminal icon in the left column of the desktop as shown in Figure 11 and selecting New Window:

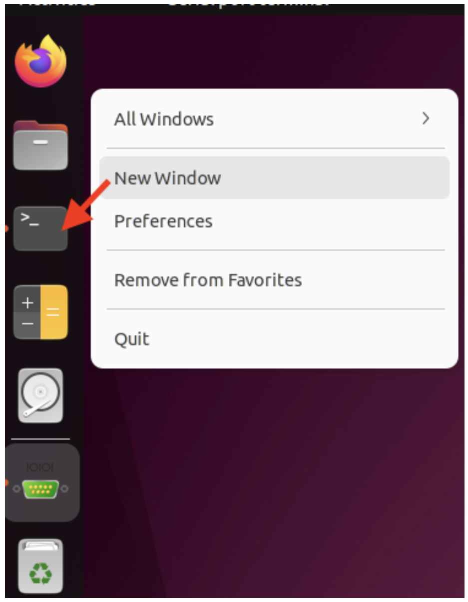


Figure 11: Terminal Launch

Once the new terminal window is open, change directories to the work folder by running the following command (Figure 12).

cd work

Figure 12: Changing Directory to Work

Once we have changed directories to the exercise folder (/desktop/LAB/work/), we can run the following flashrom command to see how the command structure of this tool is designed.

flashrom --help

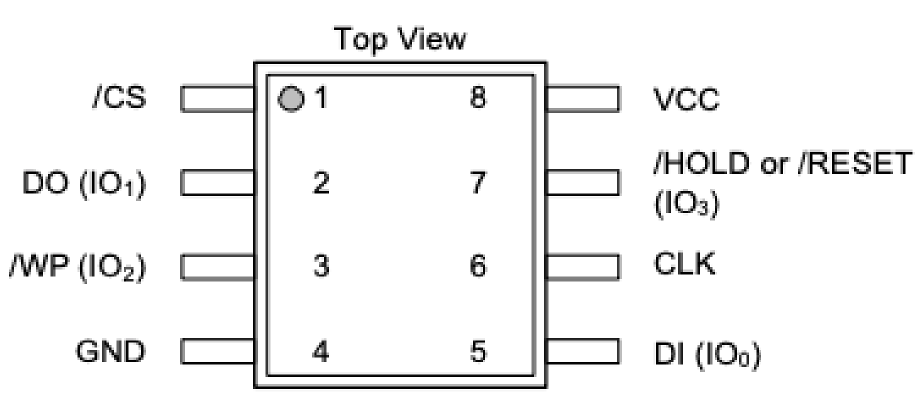
Once the command is run, you should see the following output shown in Figure 13. Scroll around to see the various command syntax instructions.

```
lab7@lab7-ThinkPad-X390:~/Desktop/LAB/work$ flashrom --help
flashrom v1.6.0-devel (git:v1.5.0-50-ged47c871) on Linux 6.8.0-60-generic (x86_64)
flashrom is free software, get the source code at https://flashrom.org

Usage: flashrom [-h|-R|-L|
        -p <programmername>[:<parameters>] [-c <chipname>]
                (--flash-name|--flash-size|
                 [-E|-x|(-r|-w|-v) [<file>]]
                 [(-l <layoutfile>|--ifd| --fmap|--fmap-file <file>) [-i <region>[:
                 [-n] [-N] [-f])]
        [-V[V[V]]] [-o <logfile>]

 -h | --help                        print this help text
 -R | --version                     print version (release)
 -r | --read [<file>]               read flash and save to <file>
 -w | --write [<file>|-]            write <file> or the content provided
                                    on the standard input to flash
 -v | --verify [<file>|-]           verify flash against <file>
                                    or the content provided on the standard input
```

Figure 13: flashrom help

Please read the notes below on reading flash memory from an SPI flash chip and some details about the XM25QH64 flash memory chip before running the flashrom command that allows us to extract the flash memory data.

> ***Notes:*** *The XM25QH64 flash memory chip is an 8 pin 3 volt 64M-BIT Serial Peripheral Interface (SPI) flash memory chip. SPI is a serial communication protocol used to interface and communicate between microcontrollers and peripheral devices such as memory.*
>
> *When trying to read from a SPI flash memory chip we often need to remove the chip from the circuit. The reason behind this is that SPI is not a shared bus protocol. So, when you hook a chip reader to the hardware you will often back power the circuit, which powers up CPU and clock. So, you end up butting heads with the CPU trying to interact with flash memory over the same SPI communication and typically this will cause the reader to fail to read the flash memory. As discussed at the beginning of this exercise writeup, we have rigged the device to allow us (RED switch) to disconnect pin 8 of the flash memory, allowing us to power up the flash memory chip without powering up the circuit board.*



- *VCC is voltage pin*
- *GND is the ground pin*
- *CS is known as chip select or enable*
- *DO is data output*
- *DI is data input*
- *WP is write protect enable*
- *CLK is the clock signal*
- *HOLD RESET*

To read this chip using SPI we need to connect to CS DO DI CLK VCC GND. WP and HOLD/RESET are not needed. The following image shows how we wired up the XM25QH64 to the breakout board so we can do this exercise without requiring us to remove the chip from the circuit board. To avoid cramming too many wires into the chip space area, we tapped the board at other locations for system VCC and GND as shown below.

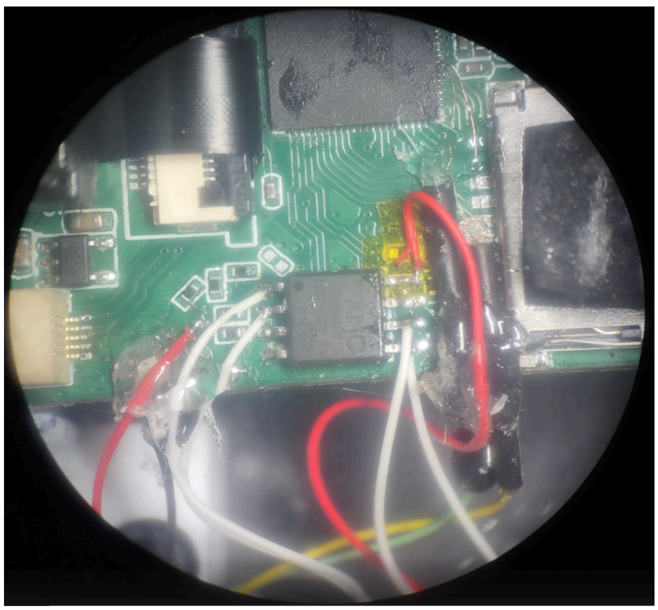


Figure 14: Flash Memory Wiring

OK, now we can proceed with extracting memory from the flash chip by using the application flashrom. To accomplish this run the following flashrom command.

sudo flashrom -p ft2232_spi:type=2232H,port=B,divisor=16 -c XM25QH64C/XM25QH64D -r unaltered-fw.bin

Here is a quick breakdown of the flashrom command's syntax:

- *The -p switch is used to select the programmer used (-p <programmername>[:<parameters>])*
- *The Tigard <programmername> is ft2232_spi*
- *The Tigard <parameters> are type=2232H, ttyUSB1 is port=B, speed of the transfer, divisor=16*

- *The -c selects the chip to be read XM25QH64C*
- *The -r stands for read and the following name unaltered-fw.bin is the name of the file created*

When you run the above command, the response should look something like Figure 15.

```
lab7@lab7-ThinkPad-X390:~/Desktop/LAB/work$ sudo flashrom -p ft2232_spi:type=2232H
-r unaltered-fw.bin
flashrom v1.6.0-devel (git:v1.5.0-50-ged47c871) on Linux 6.8.0-60-generic (x86_64)
flashrom is free software, get the source code at https://flashrom.org

Found XMC flash chip "XM25QH64C/XM25QH64D" (8192 kB, SPI) on ft2232_spi.
===
This flash part has status UNTESTED for operations: WP
The test status of this chip may have been updated in the latest development
version of flashrom. If you are running the latest development version,
please email a report to flashrom@flashrom.org if any of the above operations
work correctly for you with this flash chip. Please include the flashrom log
file for all operations you tested (see the man page for details), and mention
which mainboard or programmer you tested in the subject line.
You can also try to follow the instructions here:
https://www.flashrom.org/contrib_howtos/how_to_mark_chip_tested.html
Thanks for your help!
Reading flash... done.
```

Figure 15: flashrom reading memory

After running the above flashrom command, you should now have a copy of the firmware from the camera stored in the folder: /desktop/LAB/work/ called unaltered-fw.bin.

Flash memory dump review and edit

In this section of the exercise, we will be exploring the raw flash binary with the tool Hexedit, along with steps on altering the data in the binary for the purpose of gaining access to the system via the UART serial communication connection.

Now that the firmware has been extracted using flashrom, let's make a copy of the file unaltered-fw.bin. This is done so we have a good copy of the firmware in case it is needed for system restoration later. We will name the copy single-cpy.bin. This can be done by running the following Linux command in the terminal (Figure 16).

sudo cp unaltered-fw.bin single-cpy.bin

```
lab7@lab7-ThinkPad-X390:~/Desktop/LAB/work$ cp unaltered-fw.bin single-cpy.bin
lab7@lab7-ThinkPad-X390:~/Desktop/LAB/work$ ls -al
total 16392
drwxrwxr-x 2 lab7 lab7    4096 Jun 13 14:51 .
drwxrwxr-x 4 lab7 lab7    4096 Jun  9 11:38 ..
-rw-r--r-- 1 lab7 lab7 8388608 Jun 13 14:51 single-cpy.bin
-rw-r--r-- 1 root root 8388608 Jun 13 14:28 unaltered-fw.bin
```

Figure 16: Making Copy of Binary

Once a copy has been made, open the firmware (copy) using the tool Hexedit for exploring it and making needed changes. To open single-cpy.bin, run the following command within the Linux terminal (Figure 17).

sudo Hexedit single-cpy.bin

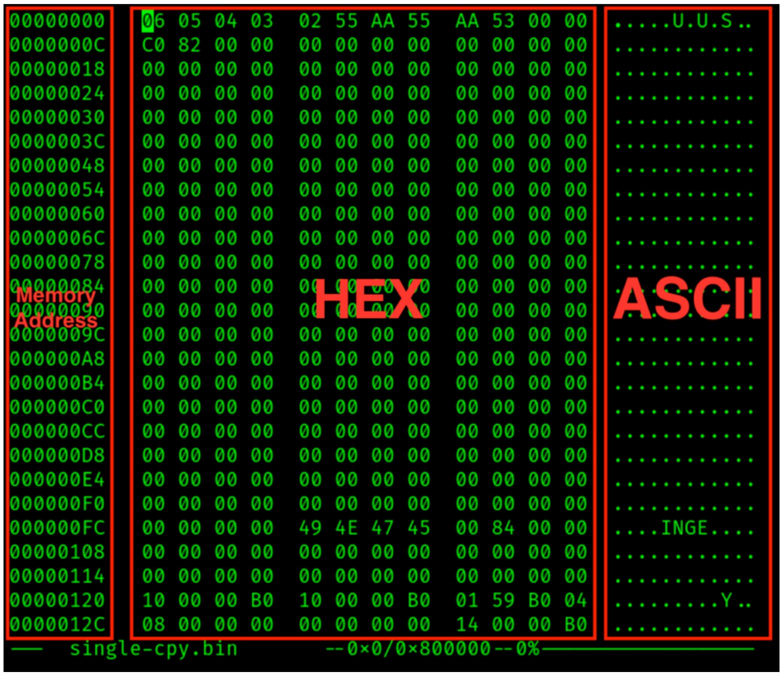


Figure 17: single-cpy.bin Opened in Hexedit

Before going any further, we need to discuss a few basic functions about moving around in Hexedit.

Using the arrow keys, you should be able to move the cursor.

Also, you can switch between hex byte within the center columns, and Ascii on the right-side column and back and forth using the Tab key. These sections are annotated above in Figure 17.

> ***Note:*** *During my initial testing of this camera device, besides noting that U-Boot did not appear to output any interaction to the serial console, I also noticed that if I altered any of the U-Boot bootargs in the raw binary it did not have any usable effect either. After further examination of the firmware, I found CMDLconsole in the binary file and found that changes made to this argument did have a direct effect on the system. CMDLconsole is a kernel console argument within the system boot process that allows settings to be passed to the kernel. This setting can also include single, which is passed directly to init, causing the kernel to load in single-user mode.*
>
> *Single-user mode is a limited Linux boot mode where only the root user is logged in, no networking or services are started. In this mode no password is required. It's typically used for system recovery, maintenance, or password reset and triggered by adding the word single to the kernel command line.*

So, our next step is to locate the CMDLconsole text string in the firmware. To do this, switch your cursor so it is on the Ascii side of the screen on the right (Tab key). Once you have done that, hold down the Control key and hit the s key (CTRL+s) at the same time. This will put you in the Ascii search mode as shown below in Figure 18.

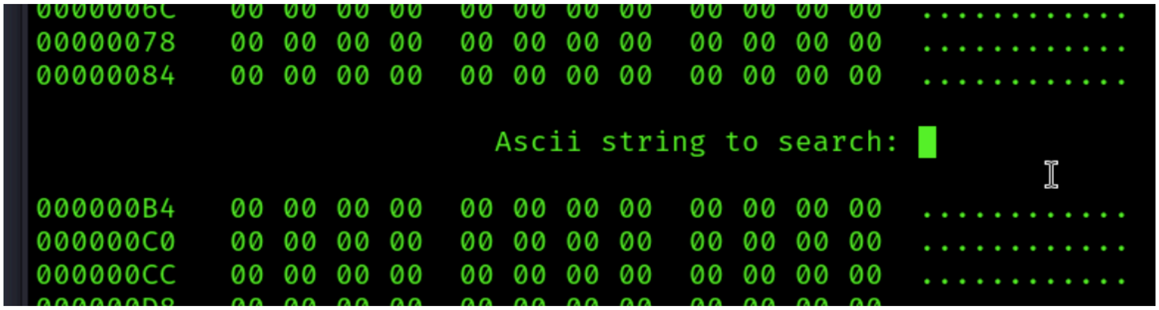


Figure 18: Hexedit Ascii Search Mode

Once the search prompt comes up, enter CMDLconsole and hit enter as shown below in Figure 19.

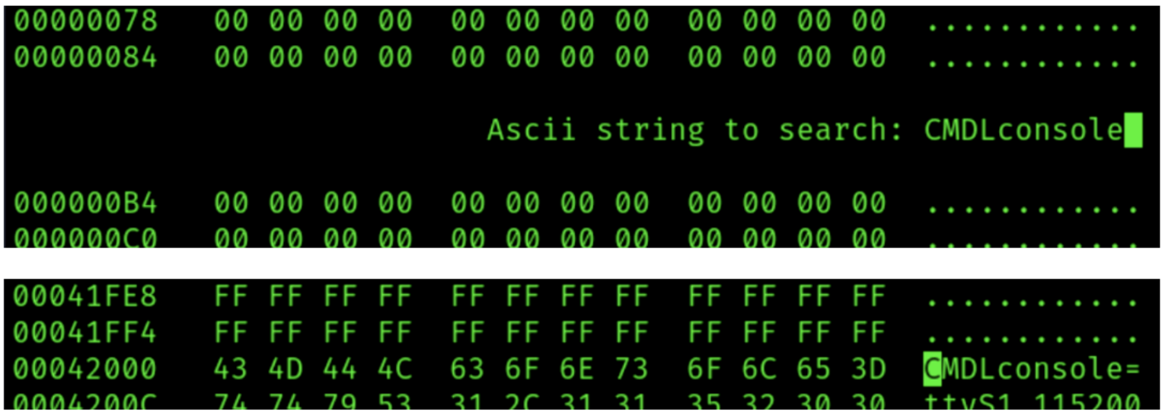


Figure 19: Searching for CMDLconsole String

Next, use your arrow keys to scroll down to the bottom of the CMDLconsole text string. Keep scrolling until you find the word quiet as shown in Figure 20.

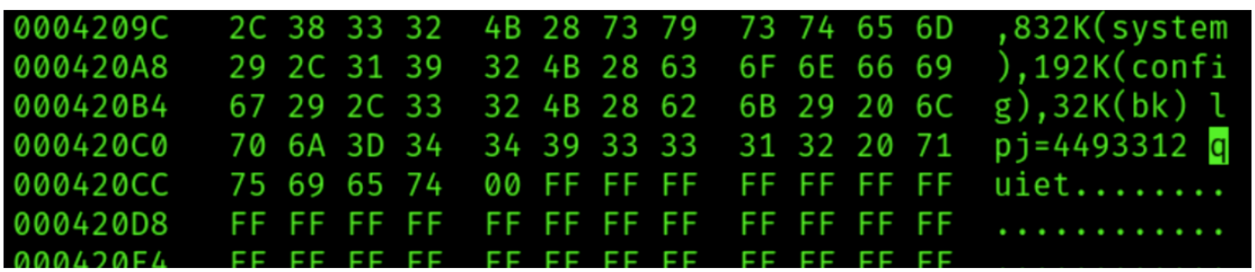


Figure 20: Locating quiet

> ***Note:*** *So, now I think we know why the kernel and operating system load was not very verbose to the screen. Passing quiet to the kernel during load tends to make everything very QUIET. Just so you know I did confirm this by removing it and reloading the firmware. With quiet removed it gets very noisy in the serial console when the kernel and file systems mount and load.*

Next we are going to use the Hexedit application to modify the raw binary file "single-cpy.bin" that we currently have loaded into Hexedit.

Note: *Be careful, if by chance you have accidentally changed anything in the code the best thing is to shut down Hexedit and reload the file. If you saved anything by accident, then make a new copy of single-cpy.bin before restarting this section.*

To shut down Hexedit without saving changes, hold down the Control key and hit the z key (CTRL+z).

If you think everything is good, then proceed by scrolling down to quiet and changing it to single by starting at the q and typing over the word quiet, as shown below in Figure 21.

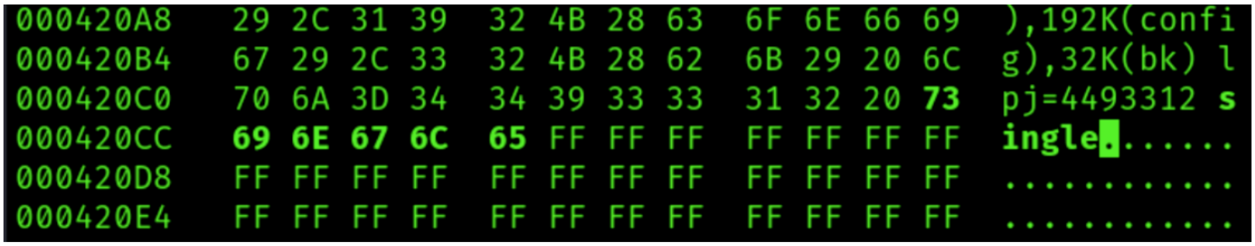


Figure 21: Changing quiet to single

Once you have overwritten quiet with single you will need to tab over to the hex side and add hex 00 at the end just after the 65 as shown in Figure 22.

Note: *The purpose of adding a null byte 00 is to terminate the length of the command data passed to the kernel during load. If not, it is going to keep sending data until it finds a 00 null byte or the whole thing crashes and we don't want either.*

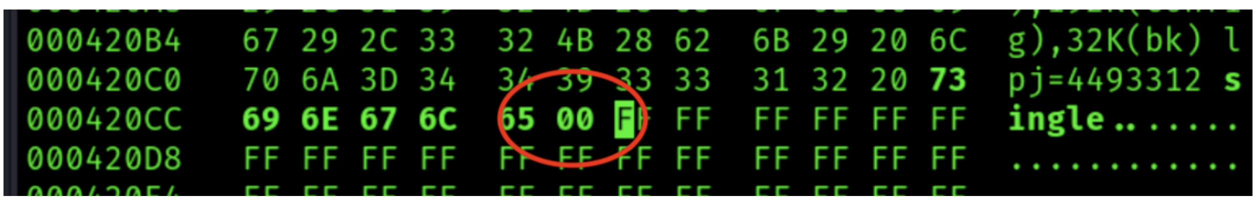


Figure 22: Terminating CMDLconsole with Null Byte

Once the word single is added and terminated with 00 you will need to save these changes to the single-cpy.bin file. This is done by holding down the Control key and hitting the x key (CTRL+x).When prompted, answer yes by hitting the y key as shown below in Figure 23.

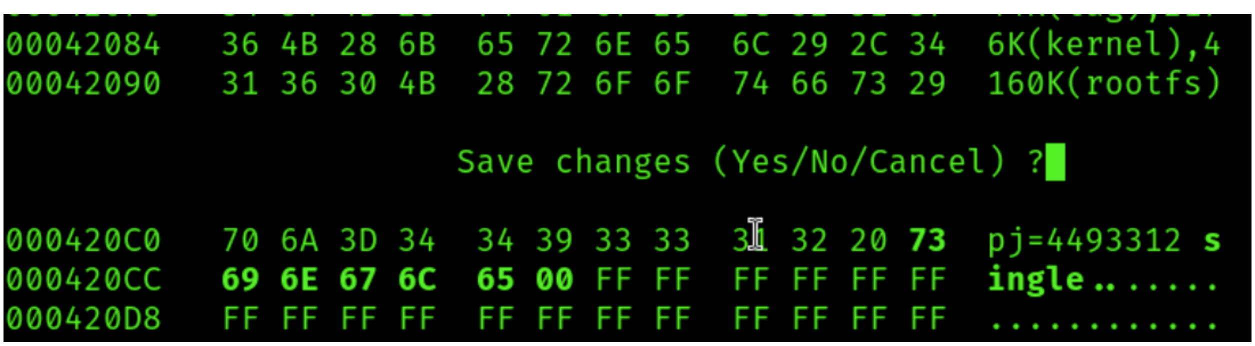


Figure 23: Hexedit Saving Changes to Binary

Single-user mode access

In this section, we will write the altered binary back to the camera's flash memory, reboot the camera, and explore single-user mode. In single-user mode we will have root access but also will have limited system access. This will require us to remount sections of the operating systems and install kernel drivers to allow access to the flash memory chip, so we can load the needed file systems.

The Tigard should still be attached to the breakout board via the ribbon cable, allowing you to write the modified binary (single-cpy.bin) from the previous section back to the camera's flash memory. To do this run the following command in the terminal.

> ***sudo flashrom -p ft2232_spi:type=2232H,port=B,divisor=16 -c XM25QH64C/XM25QH64D -V -w single-cpy.bin***

The response from this will take a little longer to complete. During the write process (-w) the application flashrom will first erase the flash memory, then write the binary, and then it will validate that the binary was written without error by reading it back and comparing it to the original file. Also the -V shows verbose data during the write operation, In conclusion the final output should look similar to Figure 24.

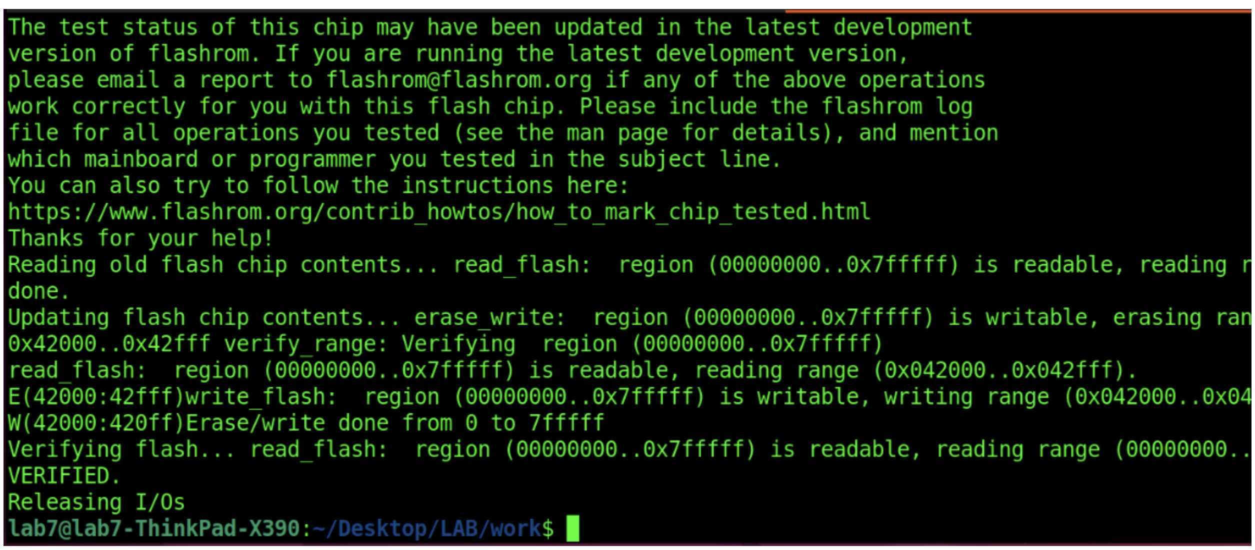


Figure 24: flashrom Writing Changes Back to Flash Chip

Once the binary file has been successfully written back to the flash memory on the camera, we need to do the following before proceeding:

Unplug the ribbon cable on the breakout board and turn the red power switch on. As shown below in Figure 25.

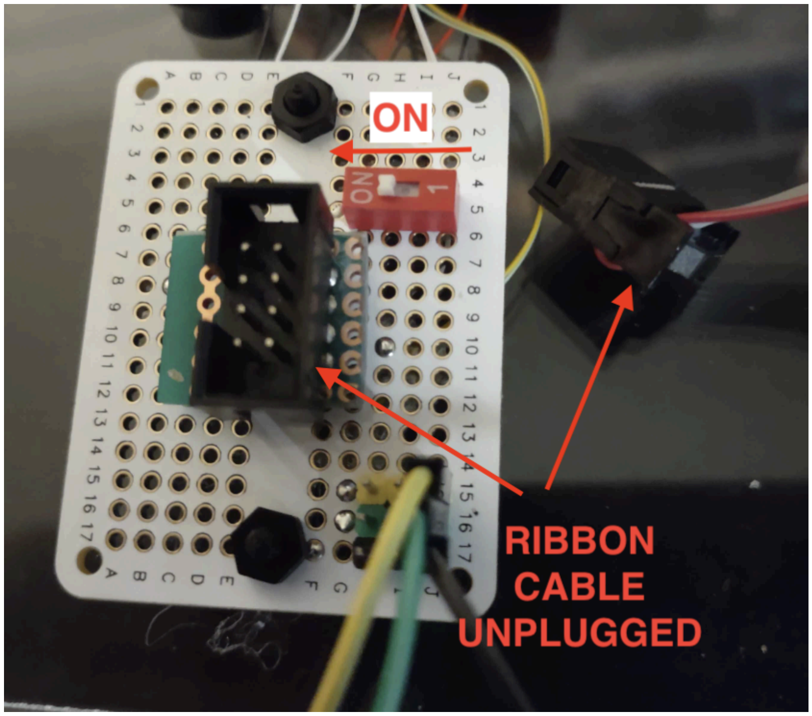


Figure 25: Unplug Ribbon Cable

Once you are 100% sure that the ribbon cable from the Tigard is not attached to the breakout board you can then plug the USB power into the camera (Figure 26), BUT DO NOT TURN ON POWER STRIP YET.

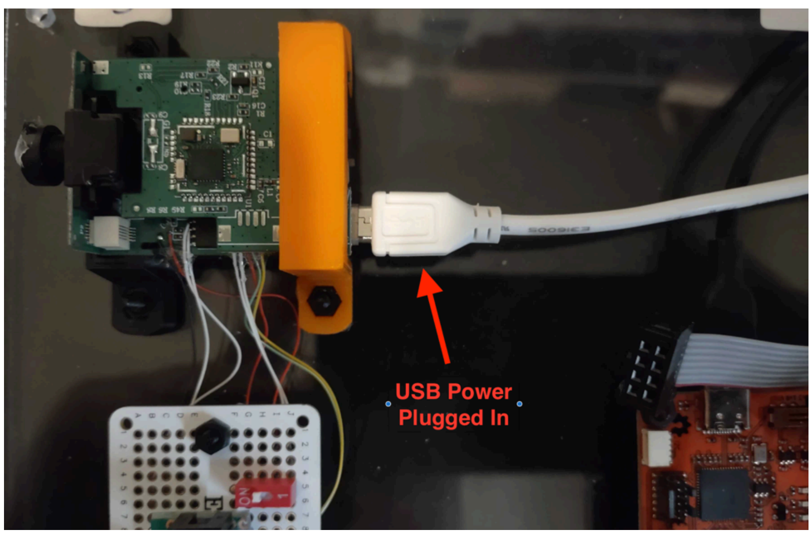


Figure 26: Plug Camera USB Power in

Before turning power (Power Strip) on for the camera we need to bring up the gtkterm, which should still be running. Once you are monitoring gtkterm you can turn on the power strip for the camera. You should see the camera start up and quickly show a prompt that should look similar to the following Figure 27.

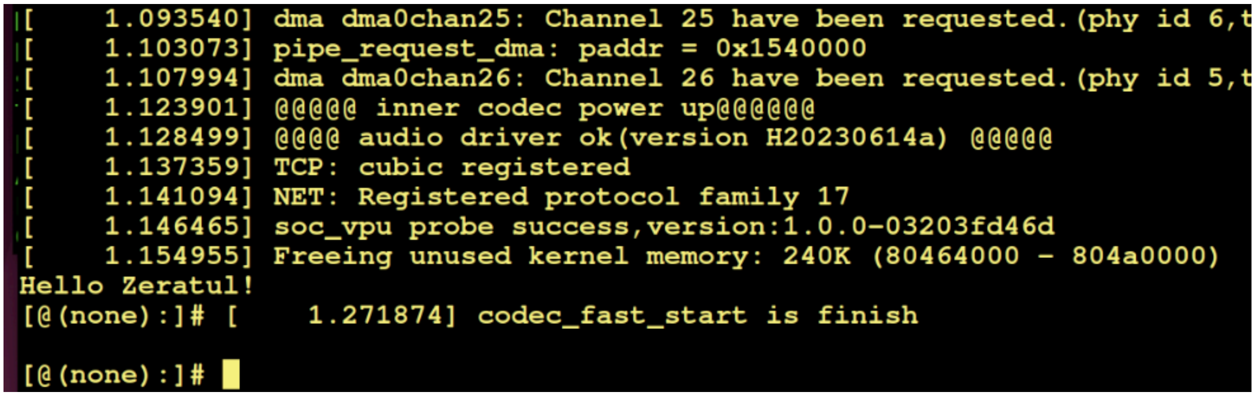


Figure 27: Boot Camera Into Single-User Mode

You should now be in single-user mode with root access to the camera. In the next section, you will be loading the needed file systems and kernel driver and modifying the correct shadow file to allow the system to be booted with a blank root password.

Mounting needed file systems and removing root password

Once the camera is booted into single-user mode you will need to make several modifications to the system to be able to gain full root access. Since we are now in single-user mode it is important to understand that in single-user mode very little is functional and to be able to do further testing and hacking you will need to bring certain features back online, such as file system mounts and certain kernel drivers.

What file systems are needed or should be brought back online? For the most part this is very standard, there are a couple kernel filesystems and, without them, access to other things won't work. To fix this issue you can run the following commands to mount up the needed file systems (proc & sys), so we have access to kernel file systems and objects.

```
mount -t proc proc /proc
mount -t sysfs sysfs /sys
```

Now with /proc and /sys loaded we will need to examine the system start up scripts to see what needs to be loaded up next.

To do this,we will examine the start-up file which the kernel uses to load up needed drivers and file systems. To find these we can typically search the /etc/init.d/ folder for a run control file. For this camera the rcS file is the run control file which is used to start up the system once the kernel is loaded. To view this file, enter the following command:

```
cat /etc/init.d/rcS
```

The output of this file should look similar to the following Figure 28, shown below.

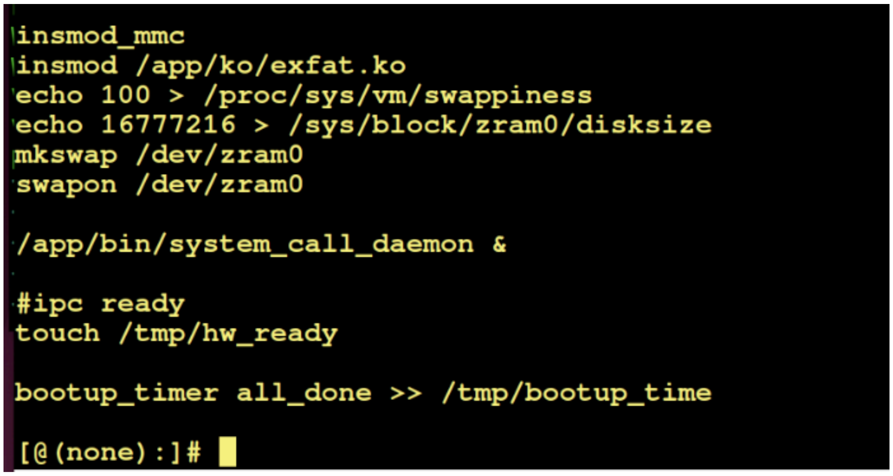


Figure 28: /etc/init.d/rcS. Run Control File

Use your mouse to scroll through the rcS file and examine the various settings. In this case there are several insmod calls, insmod files are kernel driver files. Through various testing, trial and errors, I've identified insmod_sfc as the correct driver for the SPI flash memory driver for this camera.

Also, while examining the rcS file you should see a couple file system mounts for mtdblock4 "/app" and mtdblock5 "/conf", these will be mounted after the flash memory kernel driver is loaded.

So, before the needed file system can be mounted, we need to first load the flash memory kernel driver by running the following two commands.

> ***Note:*** *The export command was used in some of the camera's firmware versions, so it is added here as a precaution. And should not affect your camera device even if not found in the rcS run control.*

export FLASH_TYPE=NOR

insmod_sfc

Once these commands are run, they should return the following results shown below in Figure 29. This indicates that the available file systems are now accessible on the flash memory.

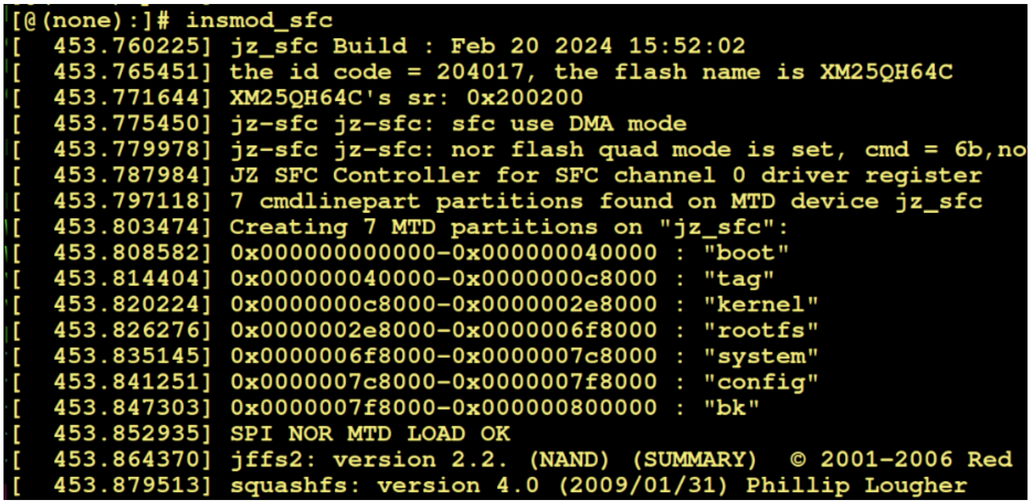


Figure 29: Kernel Drive for Flash Chip Access

Once the insmod_sfc kernel driver is loaded we can next mount the needed file systems which we identified within the rcS run control startup file. This can be done by running the following two commands:

mount -t squashfs /dev/mtdblock4 /app
mount -t jffs2 /dev/mtdblock5 /conf

Once mounted the only thing different we will see on the screen initially is the prompt changed to root as shown below Figure 30.

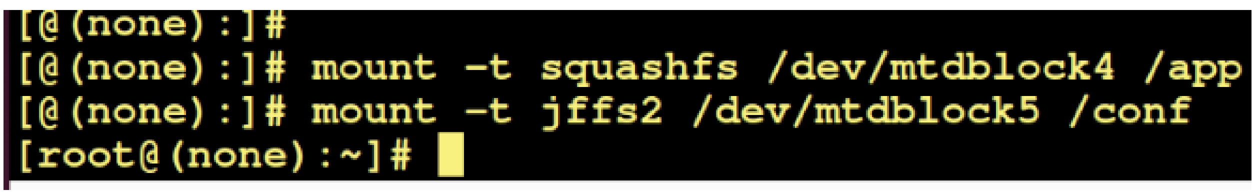


Figure 30: Mounting of /app and /conf File systems

So now we have access to the needed files systems. In this section we will change the root password hash.

> ***Note:*** *While testing this camera I found that the root password hash stored in /etc/shadow could not be changed. Any attempts to change this had no effect on the systems. After deeper examination I determined that a second shadow file found in the /conf folder was overlayed on the system during boot. This allows us to change that shadow file in the /conf folder and successfully remove the root password.*

To change the root password, to a blank password, we will need to alter the shadow file /conf/shadow. Since we do not have a way to interact with the file using an editor program such as vi effectively, we will just do it using the Linux command line.

To accomplish this, let's first view the contents of /conf/shadow by running the following command:

cat /conf/shadow

This should return results that look like Figure 31.

Figure 31: shadow file in /conf folder

To change this shadow file, we will use the echo command along with > redirect to overwrite it. This is done using the following command:

echo "root::0:0:99999:7:::" > /conf/shadow

After you run the above command to overwrite the entries in the shadow file you can next run the cat command to verify the changes you made are correct. This is done by running the following command:

cat /conf/shadow

The results should look like the following Figure 32.

Figure 32: Changed /conf/shadow root password hash

If that looks correct, in the next section, we will make another backup of the firmware and edit it to remove the single mode entry we made earlier and reload it so we can boot the camera to gain root access without a password.

Remove single and boot to root

In this section of the exercise, we will again be reading the firmware from the Serial Peripheral Interface (SPI) flash memory chip (XM25QH64) using the open-source application flashrom.

The first thing we want to do is to make sure that the camera is powered off by turning off the power strip and unplugging the USB power connector from the camera (Figure 33).

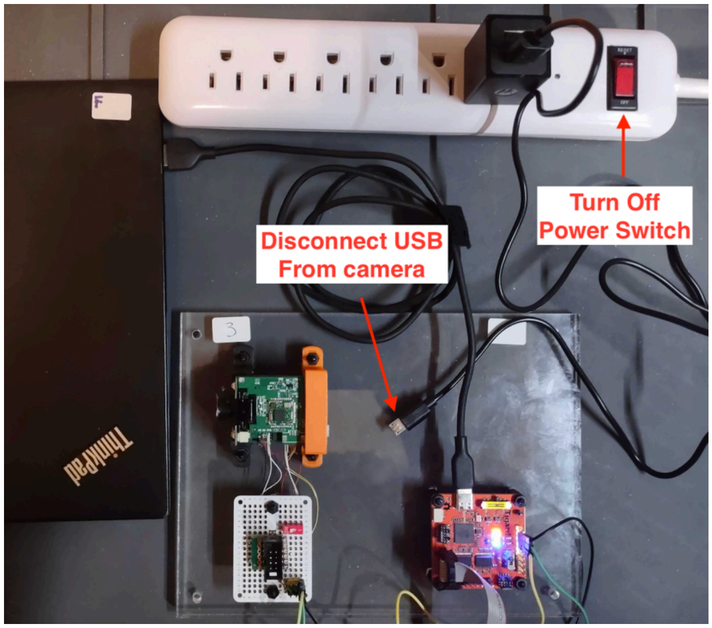


Figure 33: Camera Power Source

Once you have completed removing the power source from the camera, turn the red power switch on the breakout board to the off position and attach the keyed ribbon cable from the Tigard to the breakout board as shown below in Figure 34.

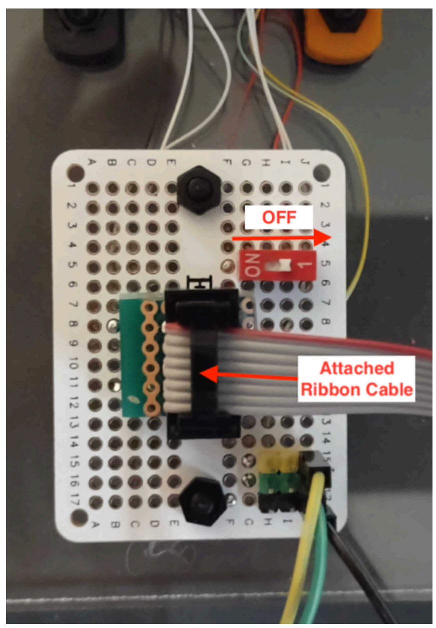


Figure 34: Configure Hardware Connections

Next, open the terminal application. It should still be available from the previous sections. If not viewable on the screen often it can be brought forward by clicking on the terminal icon in the left column as shown in Figure 35.

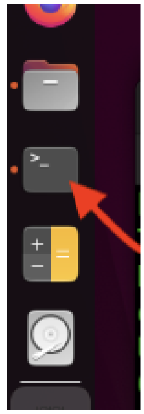


Figure 35: Terminal Launch / Open

Next we will extract another copy of the firmware from the flash memory chip. This copy should have the root password removed. This is being done so we can remove the single from the CMDLconsole command. This can be done by running the following command in the laptops terminal to extract memory from the flash chip on the camera:

sudo flashrom -p ft2232_spi:type=2232H,port=B,divisor=16 -c XM25QH64C/XM25QH64D -r norootpass.bin

When you run the above command, the response should look something like Figure 36.

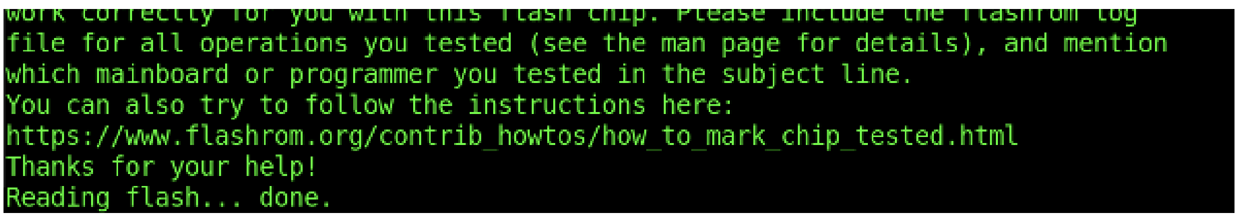


Figure 36: flashrom Reading memory

After running the above flashrom command, you should now have a copy of the firmware from the camera stored in the folder /desktop/LAB/work/ called norootpass.bin (Figure 37).

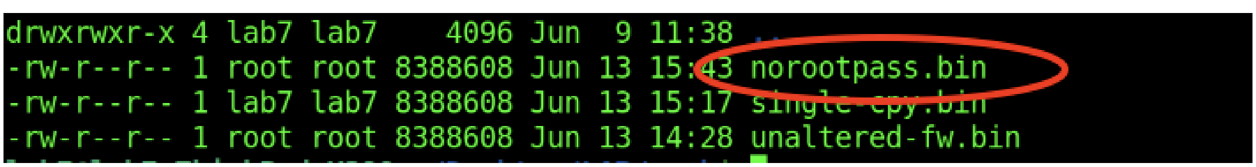


Figure 37: Directory Listing Showing Saved Files

Next, open the file norootpass.bin with Hexedit by running the following command:

sudo Hexedit norootpass.bin

Once the norootpass.bin is open in Hexedit, we can now run another Ascii search for the text string "CMDLconsole" (Figure 38). Make sure you are tabbed into the Ascii side of the screen and hold down the Control key and hit the s key (CTRL+s) at the same time. Once the search comes up enter CMDLconsole and hit enter.

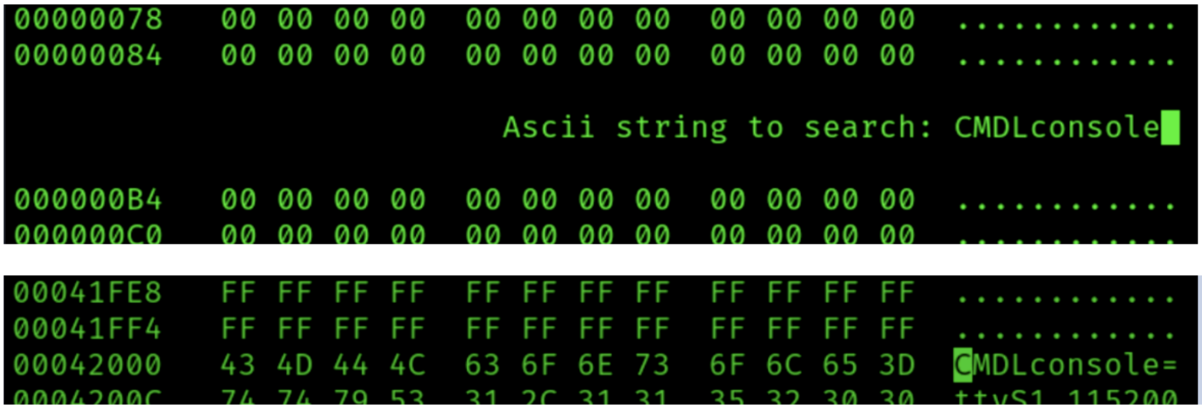


Figure 38: Hexedit Ascii Search for CMDLconsole

Now use your arrow keys to scroll down to the bottom of the text below CMDLconsole. Keep scrolling until you find the word single which you had entered earlier.

Now we need to modify the raw binary with Hexedit and replace single with quiet.

> ***Note:*** *Be careful, if by chance you accidentally changed anything in the code the best thing is to shut down Hexedit and reload the file. If you saved anything by accident then make a new copy of single-cpy.bin using step 3.1. To shut down Hexedit without saving changes hold down the Control key and hit the z key (CTRL+z)*

If you think everything is good, then proceed by scrolling down to single and changing it to quiet, as shown below in Figure 39.

Figure 39: Changing single back to quiet

Once you have overwritten single with quiet you will need to tab over to the hex side and add hex 00 at the end to overwrite the 65 (e) and also replace the following 00 with FF as shown below in Figure 40.

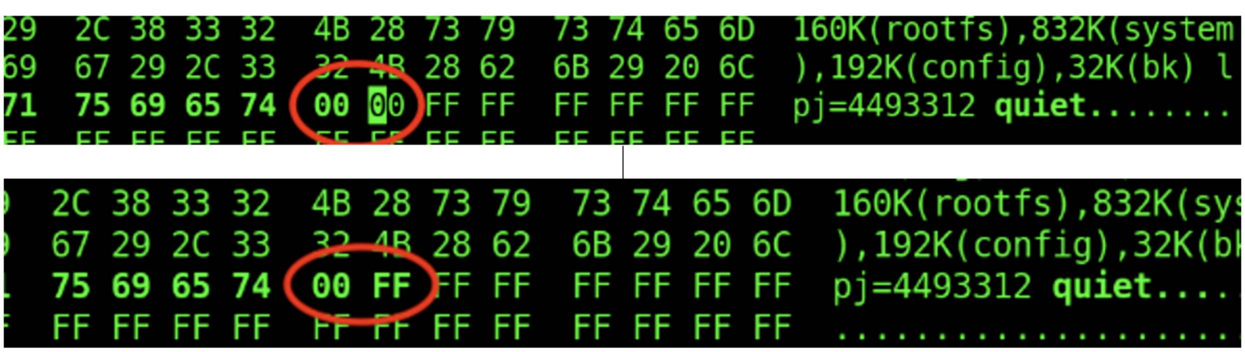


Figure 40: Hexedit Terminating CMDLconsole command with Null 00 byte and fixing the ff

Once quiet is added and terminated with 00 FF you will need to save these changes to the norootpass.bin file. This is done by holding down the Control key and hitting the x key (CTRL+x) and when prompted answer yes by hitting the y key (Figure 41).

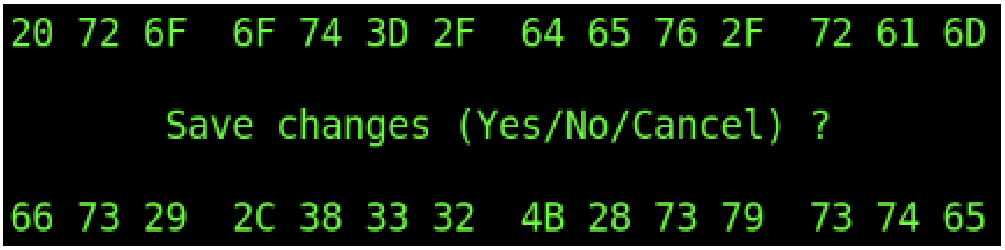


Figure 41: Hexedit Saving Changes to Binary

In this next step, we need to save it back to the camera using flashrom. This can be done by running the following command from within the terminal (Figure 42).

sudo flashrom -p ft2232_spi:type=2232H,port=B,divisor=16 -c XM25QH64C/XM25QH64D -V -w norootpass.bin

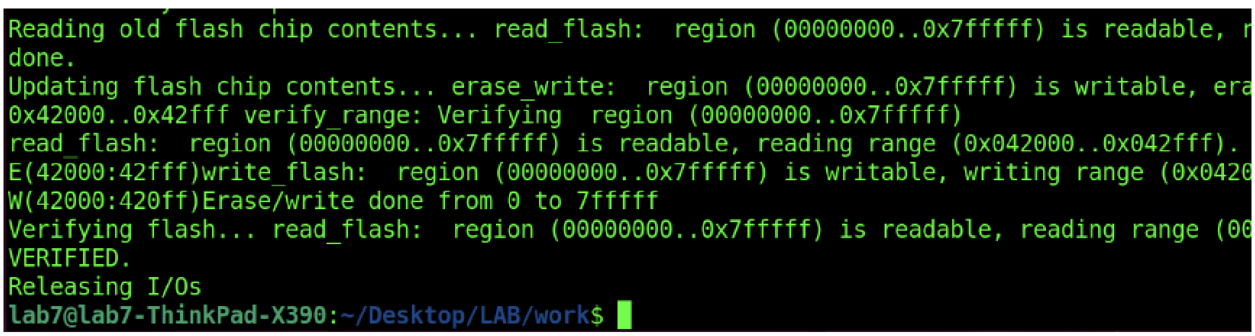


Figure 42: flashrom Writing norootpass.bin back to Flash Chip on Camera

Once the binary file has been successfully written back to the flash memory on the camera, we need to do the following before proceeding.

Unplug the ribbon cable on the breakout board and turn the red power switch on. As shown below in Figure 43.

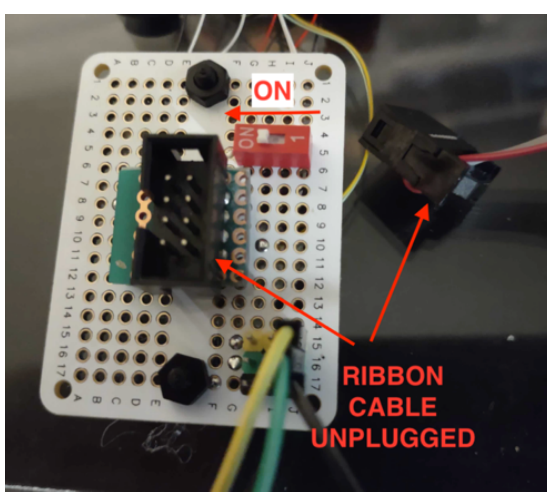


Figure 43: Unplugging ribbon Cable & Chip Power to On

Once you are 100% sure that the ribbon cable from the Tigard is not attached to the breakout board you can then plug the USB power into the camera device, BUT DO NOT TURN ON POWER STRIP YET.

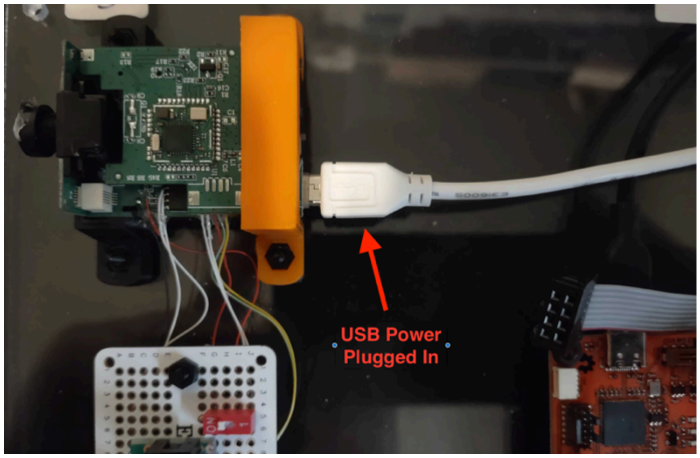


Figure 44: Plugging USB Power into Camera

Before turning power (Power Strip) on for the camera we need to bring up the gtkterm, which should still be running. Once you are monitoring gtkterm, turn on the power on the power strip for the camera.

Once a full screen of data shows up, hit the enter key and you should see a login prompt appear. Enter root and hit return and you should have root access on a fully operational camera system as shown below in Figure 45.

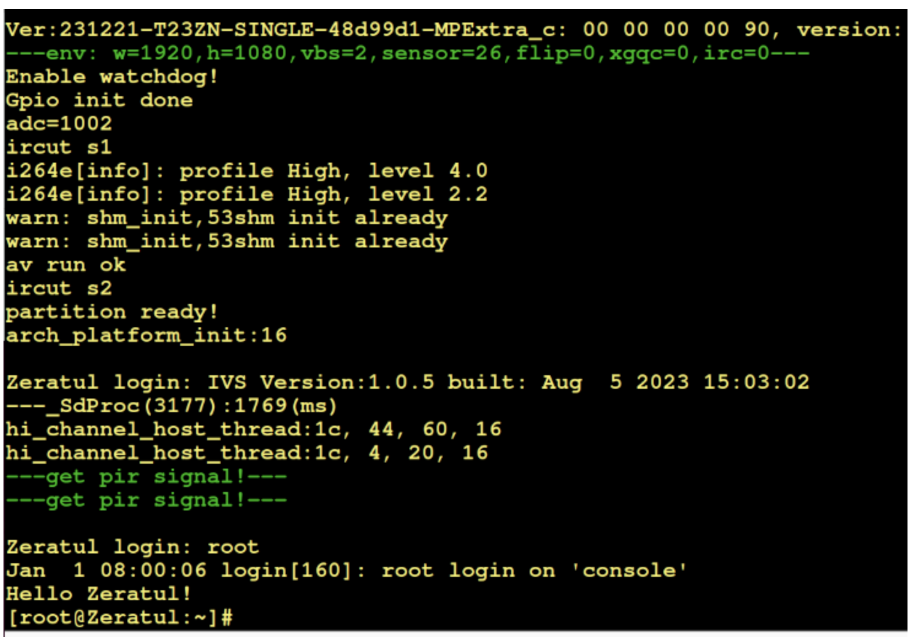


Figure 45: Booting Camera and Gaining Root Access No Password

Next, try running a couple of commands. For example, run the following command to list the files and folders on this system (Figure 46).

ls -al

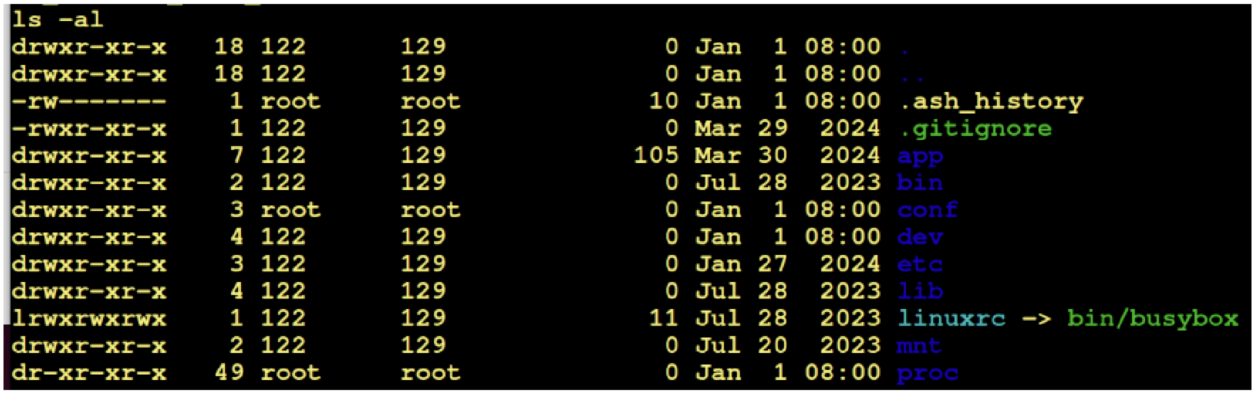


Figure 46: Directory Listing

To restore the system back to its original state, run the following command from the prompt on the camera:

cp /etc/shadow-init /conf/shadow

reboot

WHAT WAS LEARNED

Upon completion of the hands-on hardware hacking exercise at DEF CON 33's IoT Village, attendees were able to understand some new methods and approaches to gaining root access to a typical IoT device. Each year we expand on something learned from the previous year's exercise and then bring something new to the table. This year's exercise gave attendees real, hands-on experience using flashrom to read memory from a flash chip, Hexedit to view and modify raw data, and a deeper insight into how typical embedded device firmware and file systems are constructed as well as how to reconstruct those key components starting from single user mode to bring a system up to operational state.

About Rapid7

Rapid7 is creating a more secure digital future for all by helping organizations strengthen their security programs in the face of accelerating digital transformation. Our portfolio of best-in-class solutions empowers security professionals to manage risk and eliminate threats across the entire threat landscape from apps to the cloud to traditional infrastructure to the dark web. We foster open source communities and cutting-edge research–using these insights to optimize our products and arm the global security community with the latest in attacker methodology. Trusted by more than 11,000 customers worldwide, our industry-leading solutions and services help businesses stay ahead of attackers, ahead of the competition, and future-ready for what's next.