RAPID7

Hands-On IoT Hacking: From Memory Manipulation to Root Access

By Deral Heiland - Principal Security Researcher, IoT.

December 2023

Table of Contents

Summary of Exercise 4

Part 1: Setup and U-Boot Access 5

Part 2: Extract Firmware 11

Part 3: Carve Firmware from SD card 15

Part 4: Extract File Systems and Crack Root Password 18

Part 5: Identify and Modify Memory and Gain Root Access 21

About Rapid7 29

Rapid7 was back this year at DEF CON 31, participating once again at the IoT Village with our hands-on hardware hacking exercise that teaches attendees various concepts and methods for IoT hacking. And just in time for the holiday season, I'm gifting our readers with an in-depth write-up of the exercise we ran, along with some expanded context based on the questions and discussion we had at this year's event.

This year's exercise focused on the following key areas and applications:

- Universal Asynchronous Receiver/Transmitter (UART)
- U-Boot console commands
- Relocaddr
- Alter boot image memory.
- XMUART
- Binwalk
- Linux dd command
- John-the-ripper
- Hexedit
- XM Smart Home Camera

Shout Out: While working on building this exercise I used the following online source of information which covered this actual method for bypassing the filters on an IP Camera using XM530 CPU. The camera in Andrzej Szombierski's writeup was a different brand/model and offsets addresses were different, but for the most part it is 100% the same method. So, I want to give credit where credit is due.

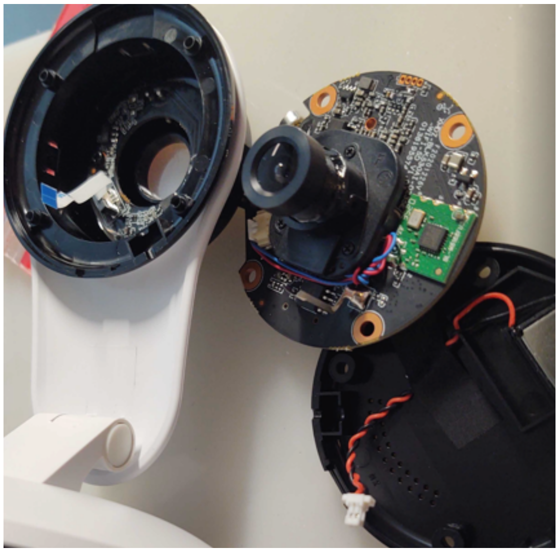
So let's get started!

Summary of Exercise

The goal of this hands-on hardware hacking exercise is to gain root access to an IP Camera over Universal Asynchronous Receiver/Transmitter (UART) by manipulating U-Boots running memory to disable a filter, which prevents the user from making certain changes to the U-Boot environment variables.

To do this, the user will interact with the device via U-Boot console over a UART connection on the device's circuit board. With UART access, the user will extract the firmware onto an SD card using U-Boot commands, move it to the laptop and use binwalk to extract the file system, and then extract the password hashes from the passwd file and crack the root password.

Once that is done, the user will also evaluate the binary to determine the memory offset of the U-Boot filter data, which will then be combined with the base address of U-Boot in memory to determine the location of the filter and alter its running memory. This will allow the user to make needed changes to U-Boot environment variables to enable a working UART console, allowing them to login with the cracked root password on the IP Camera.



Part 1: Setup and U-Boot Access

The first step is to identify the camera's Universal Asynchronous Receiver Transmitter (UART) port and connect an FTDI device to that, plug in power via USB on the camera, and then boot the system and observe the boot up and initial operations of the system.

Note: Future Technology Devices International Limited (aka FTDI) is a hardware product designed to connect TTL Logic Serial communication to a USB bus.

The FTDI hardware to the UART header connection on the camera device is shown below in the training unit we used at DEF CON. Figure 1 below shows the proper UART pinout for each device. When connecting UART, it is important to note that wiring crosses over as TXD to RXD, and vice-versa.

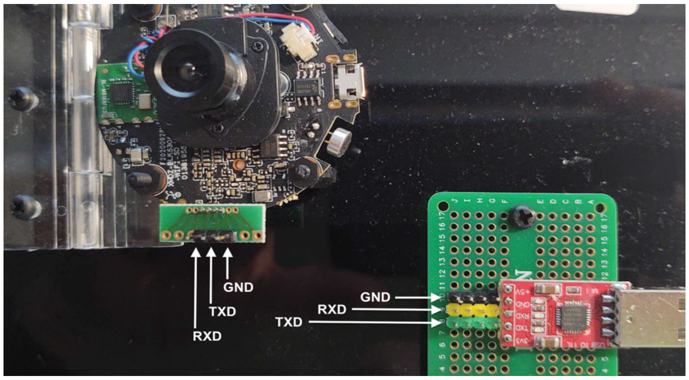


Figure 1: FTDI hardware to UART header connection

Once UART wiring is attached, you will also need to attach the USB cables between the FTDI and the laptop, and then the IP camera and the power plug. Everything should be hooked as shown below in Figure 2.

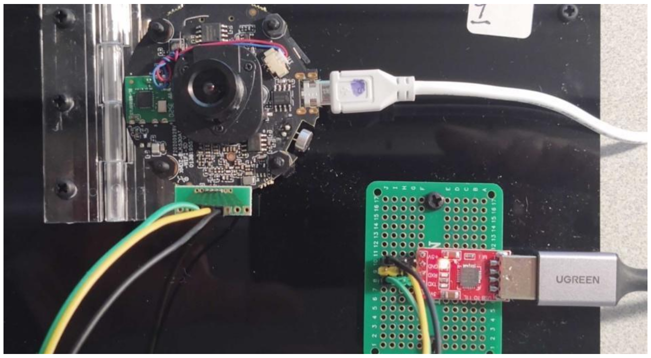


Figure 2: USB and wiring connections

In the IoT Village exercise lab we used gtkterm installed on an Ubuntu 22 Linux laptop, but any serial terminal can be used including the screen command on Linux systems. To run gtkterm first, open the command line terminal and launch the serial application "gtkterm" as root by running the following command in the terminal and when prompted, enter the correct password. Running as root allows proper communication access to the FTDI device.

sudo gtkterm

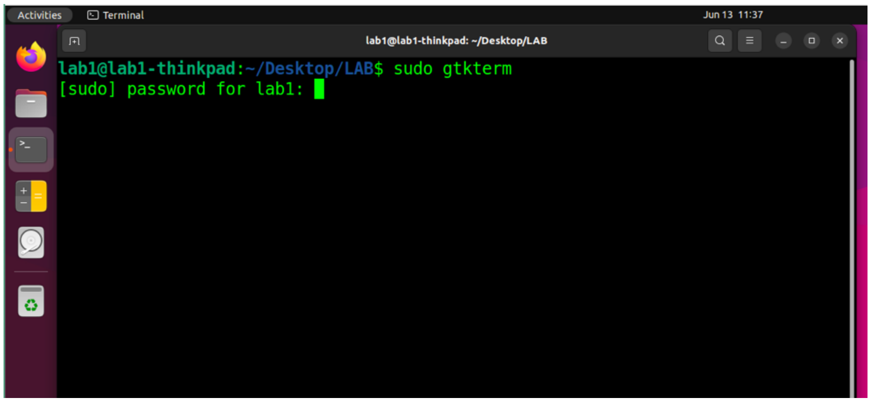


Figure 3: Launch gtkterm

Once gtkterm is running, you will need to configure gtkterm to properly communicate with the IP camera over UART. This is done by selecting "configuration -> port" from the task bar within the gtkterm application as shown in Figure 4 below, then making any necessary changes to the configuration setting so they match the settings shown in Table 1.

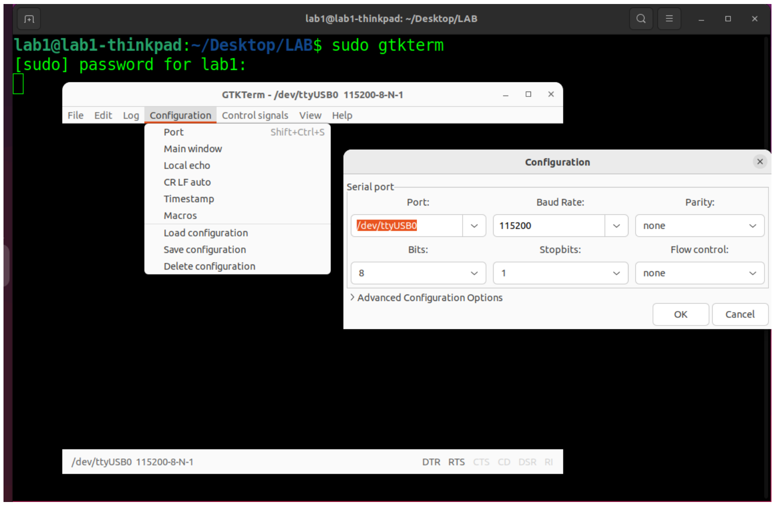


Figure 4: Configure gtkterm

Table 1: gtkterm Configuration Settings	
Port:	/dev/ttyUSB0
Baud Rate:	115200
Parity:	none
Bits:	8
Stopbits:	1
Flow control:	none

Once the gtkterm settings have been confirmed you can power up the camera device. At this point you should see the camera boot up as shown below in Figure 5.

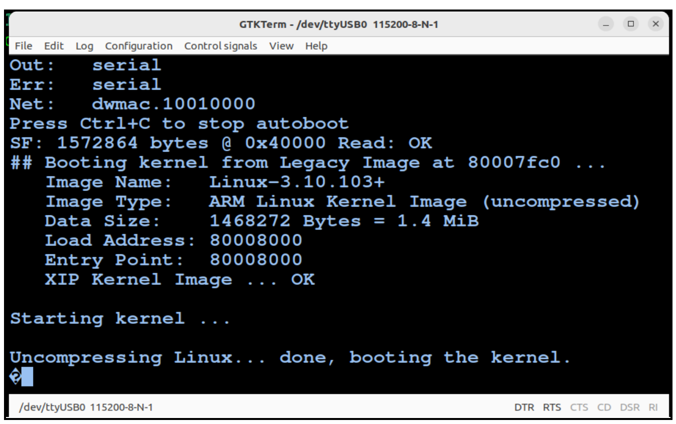


Figure 5: Camera power-up boot

If you do not see the camera device booting up, then you need to re-examine previous steps to make sure UART wiring is correct and that gtkterm configuration settings are correct.

Examining the boot process displayed on the gtkterm screen, you can see that once the device echoes “booting the kernel” that the console displays no more usable information and does not respond to any interaction from your keyboard.

You should also see “Press Ctrl+C to stop autoboot”. This allows you to stop the boot process before kernel loads. By stopping autoboot you will gain access to the U-Boot console, which will allow you to run several other commands.

To stop the autoboot process and drop into the U-Boot console, you need to first power off the device, wait 5-10 seconds, and then power it back on to restart the system.

On system restart, hit the key combination of CTRL and C (Ctrl+C) a couple of times before the “Starting kernel” displays. You will need to do this quickly; if you fail to halt the boot process then power off and try again. Once you are successful in stopping autoboot, you should be in the U-Boot console as shown below in Figure 6.

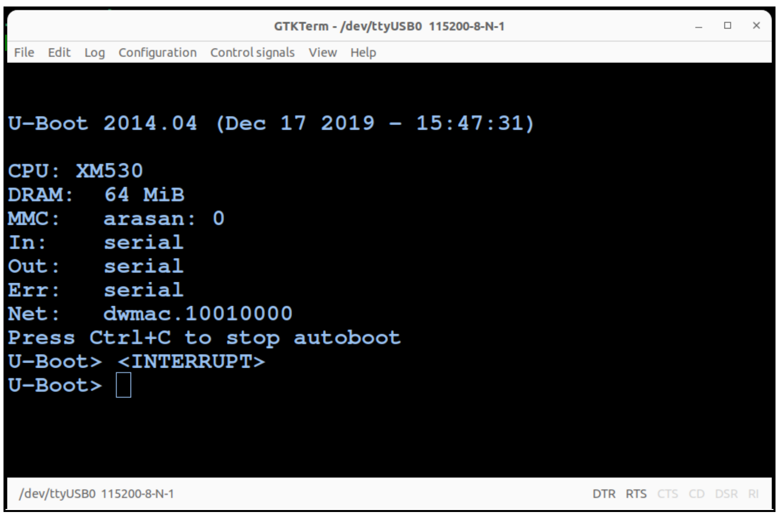


Figure 6: U-Boot console

Next, I recommend running a few simple U-Boot console commands so you can see some of the different commands and environment variables information available within the U-Boot console.

To see the various U-Boot commands available, enter a question mark and then press return. This will show all the U-Boot console commands available (Figure 7). You may have to scroll up and down to see them.

?

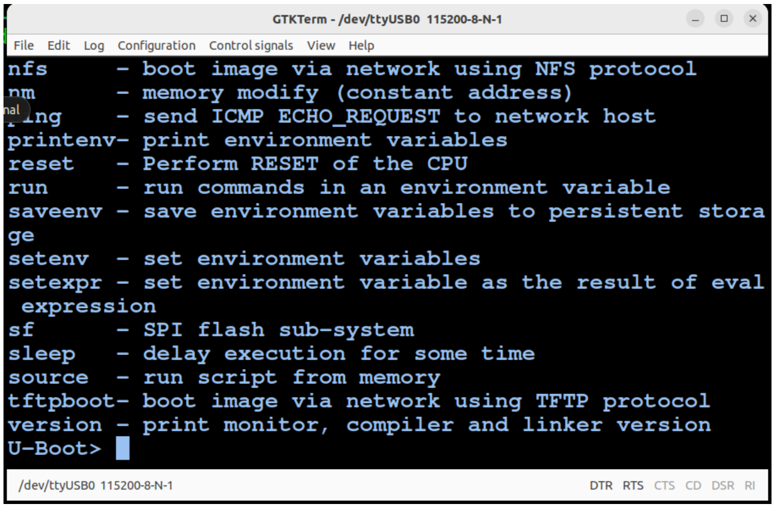


Figure 7: U-Boot command list

Next, enter the following command to show the U-Boot environment variables (Figure 8). These variables are used to define the device's boot process.

printenv

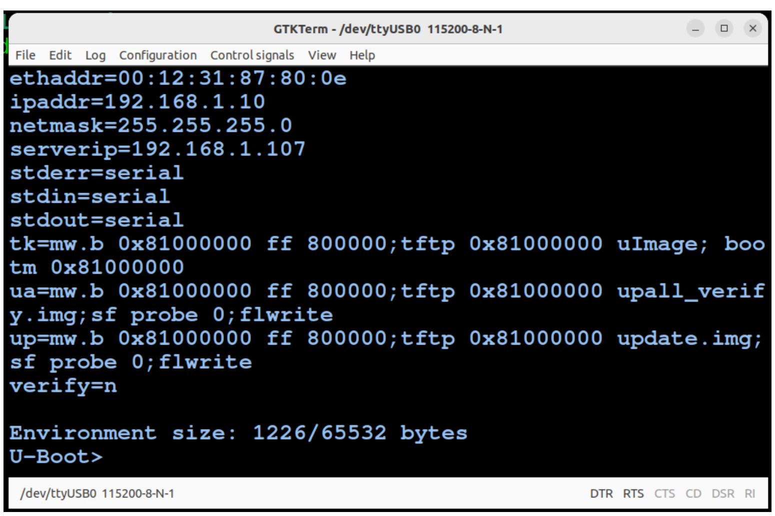


Figure 8: U-Boot environment variables

The typical method for unlocking the console on this brand of camera is to add xmuart = 0 to the U-Boot environment variables. Why is that? Turns out this device is running a XM530 processor, which was shown during the device's boot up in Figure 6.

The XM530 processor's devices typically manage the UART console with xmuart setting in the devices' U-Boot environment variables. Much of this information is published online and can be located with some Google searches.

To make this change to the environment variables within the U-Boot console you will need to run the following command:

setenv xmuart 0

Once the command is run and before saving any settings to memory you will need to validate that it was properly added. This is done by running printenv to show the U-Boot environment variables and examining the output. So, run the following command and examine the output for your changes:

printenv

Note: Any new environment variables added should initially show up at the bottom of the list.

As you can see, there is no xmuart=0 environment variable added to the U-Boot environment variables. So, it appears that this XM smart home camera system has some extra restriction to prevent you from setting the xmuart in the U-Boot environment variables, which should enable the UART console. Therefore, we need to go find the various pieces of data to help us make the needed changes to gain access. We will come back to the xmuart setting later.

Part 2: Extract Firmware

The first thing you need to do is get a copy of the firmware so you can find the data needed to activate the console and login with root credentials.

- Bypass data for xmuart
- Root password for the device

To extract firmware from this camera we are going to take advantage of the fact that the camera has an SD card slot. To do this, you will need to place an SD card in the camera's SD card slot. Install the SD card as shown below in Figure 9.

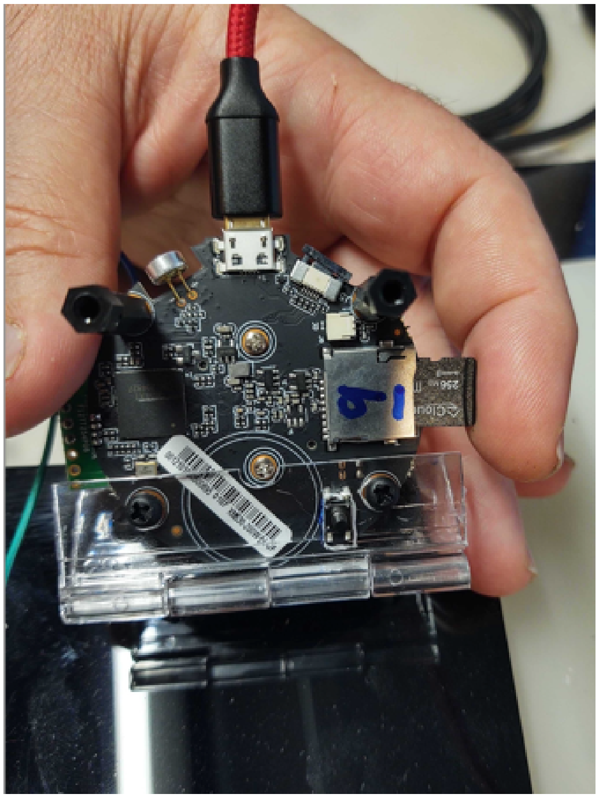


Figure 9: Camera SD card installation

Insert the SD card slowly until it clicks and stays in place, as shown below in Figure 10.

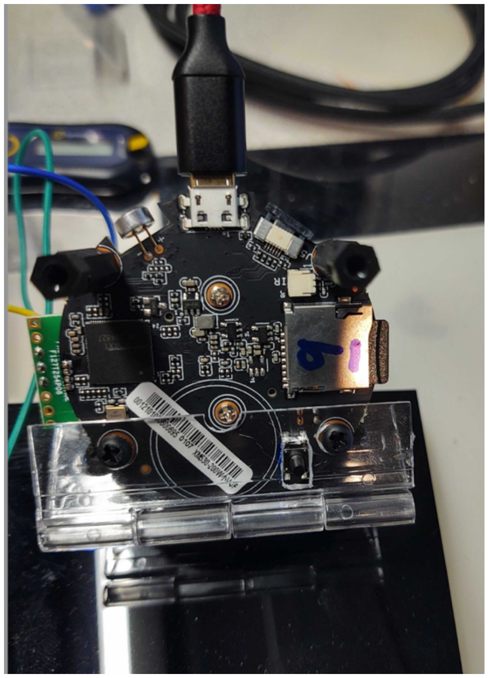


Figure 10: SD card inserted in camera

In the following steps you will read the flash memory into RAM and then copy that RAM section off to the installed SD card.

The first command you will use is sf. The sf command is used to access SPI flash. Details of this command can be seen by using help in the U-Boot console. Run the following command so you can see the various options available for the sf command. The output should look like Figure 11.

help sf

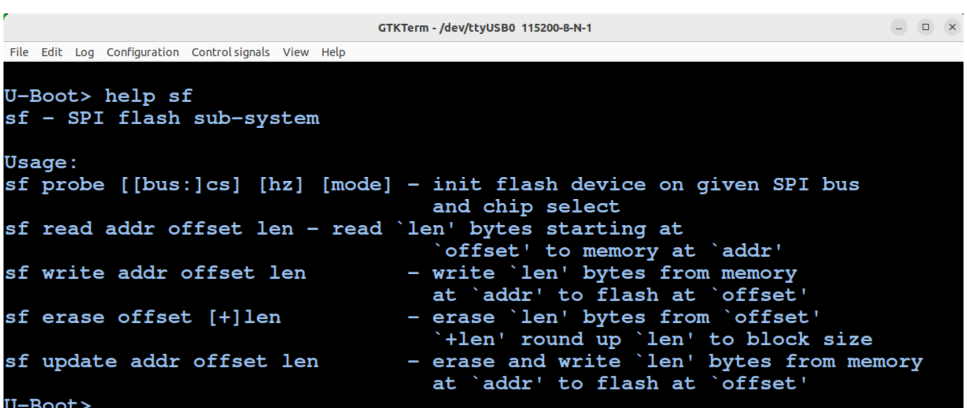


Figure 11: Help SF

The first step you need to do, which allows you to read the flash memory, is to initialize communication to the SPI flash so you can access it. This is done by running the following probe command:

sf probe

This sf probe command will not return a response.

Once the probe command is finished you will run the following command to read the SPI flash memory.

sf read 81000000 0 800000

The meaning of the command's syntax is, starting at address 0 within the camera's flash memory and read hex 0x800000 bytes of data from flash into the camera's RAM starting at address hex 0x81000000.

sf read 81000000 0 800000

This command should return the results shown below in Figure 12.

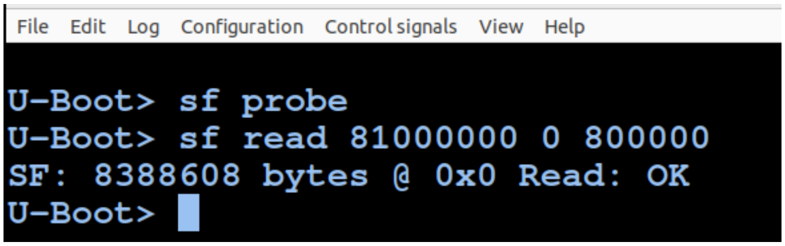


Figure 12: Read SPI flash into RAM

You may wonder how we know how big the flash memory is and where to write it to in memory. The two areas in which I typically gather this data are:

1) Flash memory size (8MB) comes from the data sheet for the flash memory chip. In this case, the flash memory is a Winbond 25Q64JV. A quick look at the datasheet shows it is 64Mb.

64Mb / 8 bits = 8MB

1. GENERAL DESCRIPTIONS

The W25Q64JV (64M-bit) Serial Flash memory provides a storage solution for systems with limited space, pins and power. The 25Q series offers flexibility and performance well beyond ordinary Serial Flash devices. They are ideal for code shadowing to RAM, executing code directly from Dual/Quad SPI (XIP) and storing voice, text and data. The device operates on 2.7V to 3.6V power supply with current consumption as low as 1µA for power-down. All devices are offered in space-saving packages.

2) For the memory location, often a quick review of the system boot or review of the U-Boot environment variables will reveal the answer. In this case, a view of the environment variables shows several examples of firmware loads from an alternate TFTP source being written to 0x81000000.

```
cramfsaddr=0x60040000
da=mw.b 0x81000000 ff 800000;tftp 0x81000000 u-boot.bin.img;sf probe 0;flwri
te
dc=mw.b 0x81000000 ff 800000;tftp 0x81000000 custom-x.cramfs.img;sf probe 0;
flwrite
dd=mw.b 0x81000000 ff 800000;tftp 0x81000000 mtd-x.jffs2.img;sf probe 0;flwr
ite
dr=mw.b 0x81000000 ff 800000;tftp 0x81000000 romfs-x.cramfs.img;sf probe 0;f
lwrite
du=mw.b 0x81000000 ff 800000;tftp 0x81000000 user-x.cramfs.img;sf probe 0;fl
write
dw=mw.b 0x81000000 ff 800000;tftp 0x81000000 web-x.cramfs.img;sf probe 0;flw
rite
```

Your next step is to write this data out to the SD card you installed in the camera.

The following command syntax will read hex 0x4000 blocks (512 bytes per block) from RAM memory starting at location hex 0x81000000 and writing it to the SD card starting at memory address 0.

mmc write 81000000 0 4000

This command should return the results shown below in Figure 13.

```
File  Edit  Log  Configuration  Control signals  View  Help

U-Boot> mmc write 81000000 0 4000

MMC write: dev # 0, block # 0, count 16384 ... 16384 blocks write: OK
U-Boot>
```

Figure 13: Write memory to SD card

Part 3: Carve Firmware from SD card

In this section you will be carving the firmware from the SD card using Linux command dd and writing it to a binary file on the laptop.

To do this you need to remove the SD card from the camera. This is easily done by slightly pushing in until you feel a click, then releasing the pressure. The SD card should slide out of the SD card holder on the camera. Once that is done you will need to connect an SD card reader via USB to the laptop and install this Micro SD card into the reader. During the IoT Village exercise we used an inexpensive Dynex SD card reader, but any SD card reader should work fine. This is similar to the camera; just insert the micro SD card as shown in Figure 14 until it clicks into your reader.

Figure 14: Dynex SD card reader

Next, you will need to identify what the device ID is for the SD card. To do this use the Disks application on your Linux machine by clicking on the Disks icon (Figure 15).

Figure 15: Disks icon

Once the application is open, locate the correct mounted drive. In the case of our IoT Village exercise, since we used 256MB SD cards, the drive showed up as a 251MB drive. In this example the device was identified as /dev/sdc as shown below with the red arrow in Figure 16.

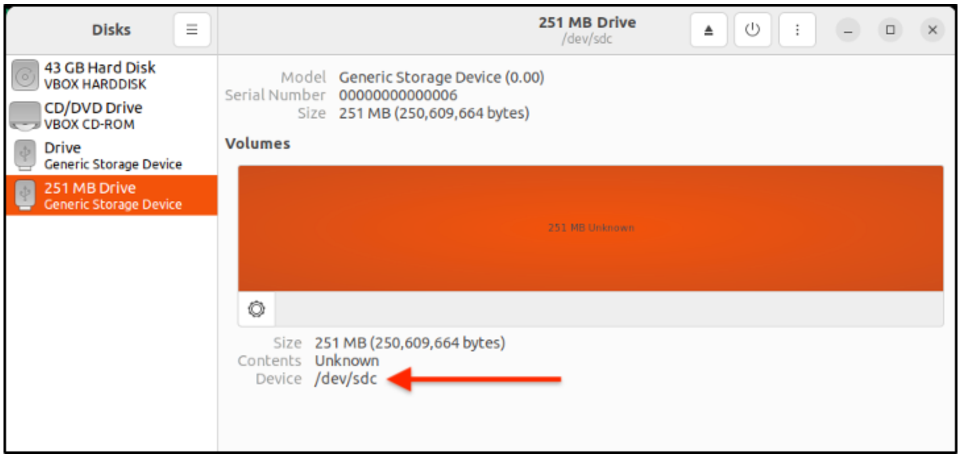


Figure 16: Disks application 251MB drive

This could typically show as either /dev/sdb or /dev/sdc or /dev/sdd — and on a newer version of Ubuntu it could also show up as /dev/sda. Make sure you identify the correct one or the following operation will fail.

Since the firmware image is much smaller than the full SD card size, you can speed things up by just carving out only the firmware (8MB) we wrote to the SD card. Unlike the U-Boot command we will be using decimal, not hexadecimal, when defining memory size during this step.

You will need to open another Terminal window. Once the new Terminal is open, run the following command. Make sure the if= (input file) is set to the correct device id (/dev/sdb, /dev/sdc, or /dev/sdd) identified above.

```
sudo dd if=/dev/sdb of=image.bin  bs=512 count=16384
```

The syntax of the dd command is. (if=) is input file, (of=) is output file, (bs=) is block size, (count=) is number of blocks.

dd if=/dev/sdb of=image.bin bs=512 count=16384

So, the above command will carve off the first 16385 blocks of data from the SD card, which is 8MB, each block is 512 bytes in size, and save it to a file called image.bin.

Once the command completes you should see the following response (Figure 17).

```
lab1@lab1-thinkpad:~/Desktop/LAB$ sudo dd if=/dev/sdc of=image.bin  bs=512 count=16384
16384+0 records in
16384+0 records out
8388608 bytes (8.4 MB, 8.0 MiB) copied, 0.724653 s, 11.6 MB/s
lab1@lab1-thinkpad:~/Desktop/LAB$
```

Figure 17: dd command carve out firmware from SD card

Next, run the following Linux directory list command to show that the image.bin file was created during the above step. It should show the file to be 8388608 (8MB) in size.

ls -al

```
lab1@lab1-thinkpad:~/Desktop/LAB$ ls -al
total 8200
drwxrwxr-x 2 lab1 lab1    4096 Jun 14 14:27 .
drwxr-xr-x 3 lab1 lab1    4096 Jun  1 17:06 ..
-rw-r--r-- 1 root root 8388608 Jun 14 14:27 image.bin
lab1@lab1-thinkpad:~/Desktop/LAB$
```

Figure 18: Directory listing

Part 4: Extract File Systems and Crack Root Password

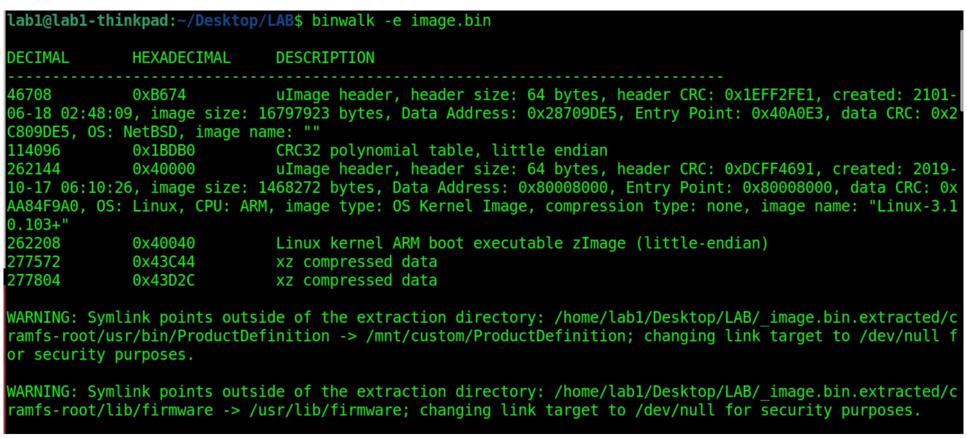
Now that you have extracted the firmware from the camera, you will need to extract the embedded Linux file system from the binary file image.bin. Once the file system is extracted, you will then locate the embedded Linux passwd file and use it to crack the root password hash so you can eventually login to the camera with root privileges.

The next step in this process is to run the following binwalk command with -e switch to extract the filesystems.

binwalk -e image.bin

Note: [Binwalk](#) is an open-source command line tool produced by [RefirmLabs](#), which is currently owned by Microsoft. Binwalk is used for analyzing, firmware extraction, and extraction of associated file systems and files from binary firmware images.

Once the Binwalk is run, the output should look something like the following image (Figure 19) and should also create a folder called _image.bin.extracted .



```
lab1@lab1-thinkpad:~/Desktop/LAB$ binwalk -e image.bin

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
46708         0xB674          uImage header, header size: 64 bytes, header CRC: 0x1EFF2FE1, created: 2101-
06-18 02:48:09, image size: 16797923 bytes, Data Address: 0x28709DE5, Entry Point: 0x40A0E3, data CRC: 0x2
C809DE5, OS: NetBSD, image name: ""
114096        0x1BDB0         CRC32 polynomial table, little endian
262144        0x40000         uImage header, header size: 64 bytes, header CRC: 0xDCFF4691, created: 2019-
10-17 06:10:26, image size: 1468272 bytes, Data Address: 0x80008000, Entry Point: 0x80008000, data CRC: 0x
AA84F9A0, OS: Linux, CPU: ARM, image type: OS Kernel Image, compression type: none, image name: "Linux-3.1
0.103+"
262208        0x40040         Linux kernel ARM boot executable zImage (little-endian)
277572        0x43C44         xz compressed data
277804        0x43D2C         xz compressed data

WARNING: Symlink points outside of the extraction directory: /home/lab1/Desktop/LAB/_image.bin.extracted/c
ramfs-root/usr/bin/ProductDefinition -> /mnt/custom/ProductDefinition; changing link target to /dev/null f
or security purposes.

WARNING: Symlink points outside of the extraction directory: /home/lab1/Desktop/LAB/_image.bin.extracted/c
ramfs-root/lib/firmware -> /usr/lib/firmware; changing link target to /dev/null for security purposes.
```

Figure 19: Binwalk command output

If you poke around the folder that was created (_image.bin.extracted), you will see that a lot of files and file system data was created. Since the goal is to find the etc/passwd file and run john-the-ripper against the passwd file to crack the root password, we can save some time and go straight to the correct passwd file in the folder _image.bin.extracted/cramfs-root/etc. So, to do this just change directory into this folder using the following commands:

cd _image.bin.extracted/cramfs-root/etc/

ls -al

Once you have changed into that directly and run the directory file list command you should see the following files, including the passwd file as shown below in Figure 20.

```
lab1@lab1-thinkpad:~/Desktop/LAB$ cd _image.bin.extracted/cramfs-root/etc/
lab1@lab1-thinkpad:~/Desktop/LAB/_image.bin.extracted/cramfs-root/etc$ ls -al
total 36
drwxr-xr-x  4 lab1 lab1 4096 Jun 14 14:55 .
drwxrwxr-x 16 lab1 lab1 4096 Jun 14 14:55 ..
-rwxr--r--  1 lab1 lab1   95 Dec 31  1969 fstab
-rwxr--r--  1 lab1 lab1    9 Dec 31  1969 group
-rw-r--r--  1 lab1 lab1   20 Dec 31  1969 hosts
drwxr-xr-x  2 lab1 lab1 4096 Jun 14 14:55 init.d
-rwxr-xr-x  1 lab1 lab1  206 Dec 31  1969 inittab
lrwxrwxrwx  1 lab1 lab1    9 Jun 14 14:55 localtime -> /dev/null
-rwxr--r--  1 lab1 lab1   59 Dec 31  1969 passwd
drwxr-xr-x  3 lab1 lab1 4096 Jun 14 14:55 ppp
lrwxrwxrwx  1 lab1 lab1    9 Jun 14 14:55 resolv.conf -> /dev/null
lab1@lab1-thinkpad:~/Desktop/LAB/_image.bin.extracted/cramfs-root/etc$
```

Figure 20: _image.bin.extracted/cramfs/etc file listing

Next, run the following command to show the contents of the passwd file:

cat passwd

You should see the stored root password hash as shown below in Figure 21.

```
lab1@lab1-thinkpad:~/Desktop/LAB/_image.bin.extracted/cramfs-root/etc$ cat passwd
root:$1$RYIwEiRA$d5iRRVQ5ZeRTrJwGjRy.B0:0:0:root:/:/bin/sh
lab1@lab1-thinkpad:~/Desktop/LAB/_image.bin.extracted/cramfs-root/etc$
```

Figure 21: Contents of passwd file

You can now run john-the-ripper to crack the password hash for root, which is stored in the file called passwd. This can be done using the following command:

john passwd

For the IoT Village exercises the root password was previously cracked, so john returned "No password hashes left to crack" as shown below in Figure 22. So, if this is your first attempt at cracking this password it could take some time.

```
lab1@lab1-thinkpad:~/Desktop/LAB/_image.bin.extracted/cramfs-root/etc$ john passwd
Loaded 1 password hash (md5crypt [MD5 32/64 X2])
No password hashes left to crack (see FAQ)
lab1@lab1-thinkpad:~/Desktop/LAB/_image.bin.extracted/cramfs-root/etc$
```

Figure 22: John returned results

If the password has been previously cracked it can be retrieved from the john-the-ripper database by running the following command:

john -show passwd

This command should return all previously cracked passwords for account hashes listed in the passwd file. In this case, root was previously cracked and the password was found to be xmhdipc as shown below in Figure 23.

```
lab1@lab1-thinkpad:~/Desktop/LAB/_image.bin.extracted/cramfs-root/etc$ john -show passwd
root:xmhdipc:0:0:root:/:/bin/sh

1 password hash cracked, 0 left
```

Figure 23: John show cracked passwords

Part 5: Identify and Modify Memory and Gain Root Access

In the final section of this exercise, you will be identifying the offset address for the U-Boot's xmuart. This xmuart appears to be a filter that prevents xmuart from being set within the U-Boot environment variables. To find this you will first examine the extracted firmware and record the offset to the beginning of xmuart. Then you will identify the relocation address for the beginning of the U-Boot code in RAM on the camera.

The first step in accomplishing this task is to open the image.bin file using the application [hexedit](). Other hex/ascii editor programs are available and should also work fine for this step. Run the following command to open the image.bin file using the hexedit application. Once the image.bin file has been opened in hexedit it should look like Figure 24.

hexedit image.bin

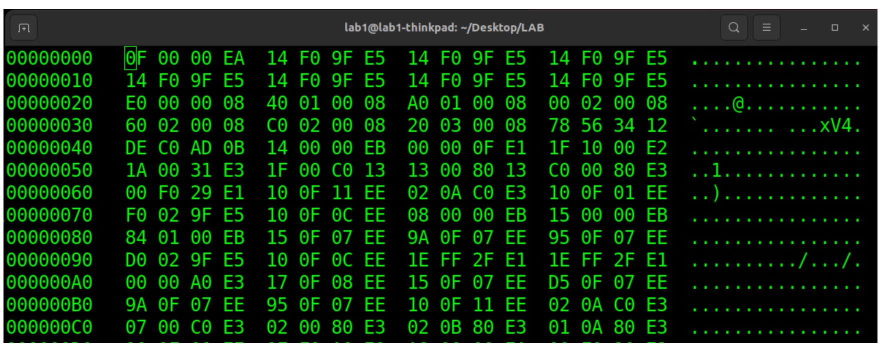


Figure 24: Hexedit of image.bin

The layout of information displayed in the hexedit application is shown below in Figure 25.

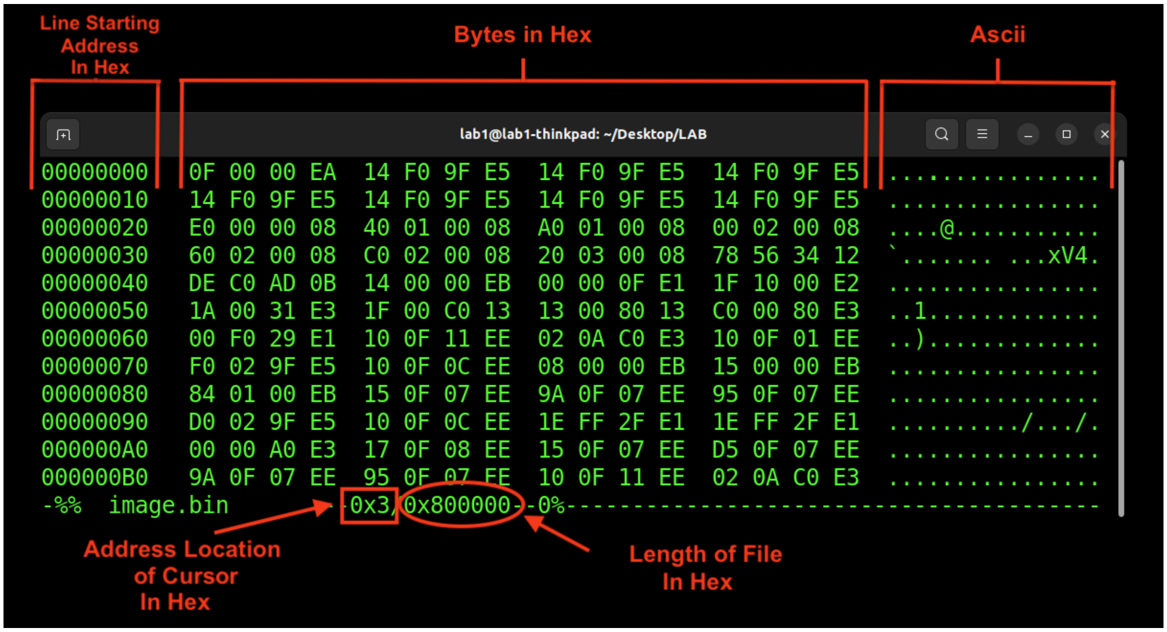


Figure 25: Hexedit layout

Before going any further, let us take a quick look at some basic functions with hexedit.

Using the arrow keys, you should be able to move the cursor. As you move the cursor around you should see the address location changes at the bottom of the screen.

Also, you can switch between Hex bytes within the center 4 columns and Ascii on the right side column, and back and forth using the Tab key.

The next step is to locate the offset address of the xmuart filter. To do this, switch your cursor so it is on the Ascii side of the screen on the right (Tab key). Once you have that set, hold down the Control key and hit the s key (CTRL+s) at the same time. This should put you in the Ascii search mode as shown below in Figure 26.

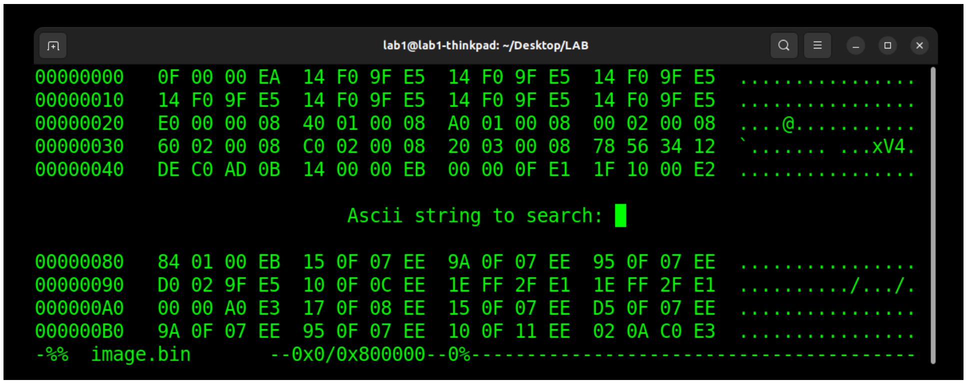


Figure 26: Hexedit Ascii search mode

Next, enter xmuart in the search window and hit enter as shown below in Figure 27.

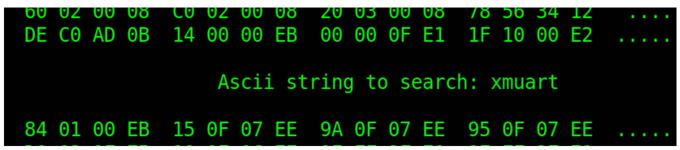


Figure 27: xmuart search entry

This will search through the files and find the first occurrence of the word xmuart. In this case there is only one occurrence of the word xmuart throughout the binary file. So, no chance of getting the wrong offset by accident. Hexedit should locate xmuart and it should look like what is shown below in Figure 28.

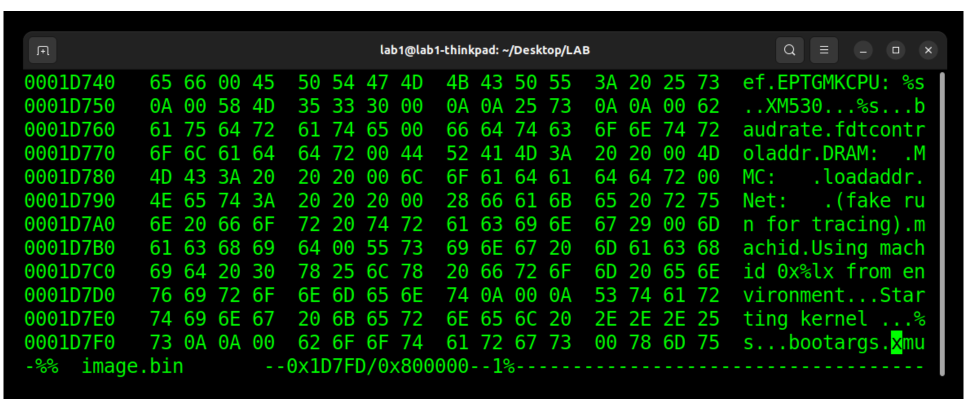


Figure 28: Hexedit search for xmuart

The offset number we are looking for is the hex location of the cursor, which is displayed at the bottom of the hexedit screen and should be 0x1D7FD as shown above in Figure 28.

In the next part of this exercise, you need to return to the U-Boot console that we had accessed using gtkterm. Once you have gtkterm back in the forefront of your desktop, you will need to run the following bdinfo command within the U-Boot console. The returned results should look like Figure 29.

bdinfo

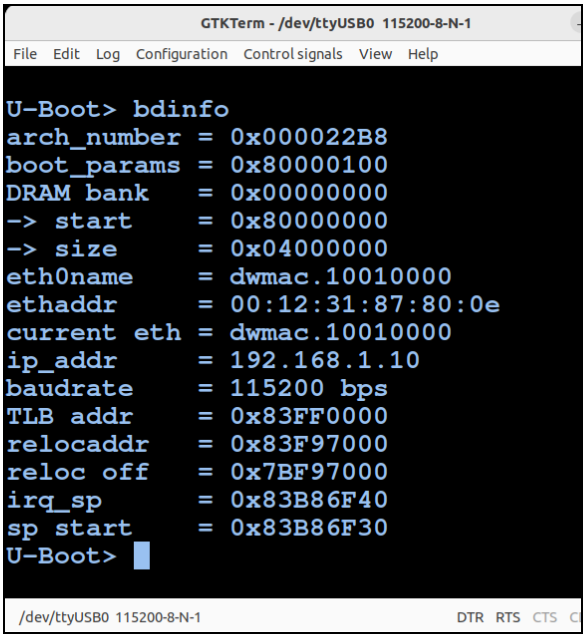


Figure 29: U-Boot bdinfo command

Look at the list of data returned and find the relocaddr. This is the location of the running U-Boot image and should be 0x83F97000.

Note: To give you a better understanding of the staged boot loader process, the typical U-Boot process goes something like this: When the device powers up, ROM code executes first (Primary Program Loader); when this ROM code runs it loads a First-Stage Bootloader into Static RAM (SRAM). This code cannot be loaded into primary DRAM because DRAM has not yet been initialized. This First-Stage Bootloader will initialize DRAM and then load the Second-stage bootloader (U-Boot) into DRAM at the relocation address (relocaddr) and execute it.

So now we need to do some math. To find the location of xmuart in running memory we need to add the offset address from the hexedit image.bin to the relocaddr.

- Xmuart offset : 0x1D7FD
- Relocaddr : 0x83F97000

You can use a calculator to do this, or if desired you can do it from a Linux command line interface (CLI) by running the following command from a Terminal command line to get the answer shown in Figure 30.

printf 0x%x $((0x83F97000+0x1d7fd))

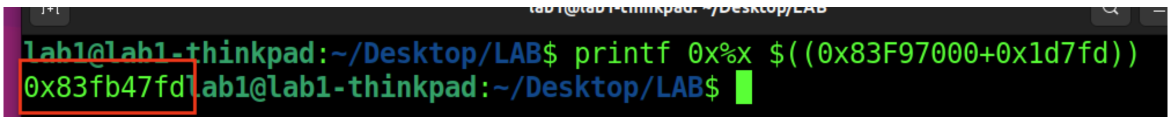


Figure 30: Some Hex math

The next step is to look at U-Boot's running memory and see if your calculations were correct. By using the U-Boot console md command, you can read memory and see if you are correct.

First run the following command to see the command options for md.

help md

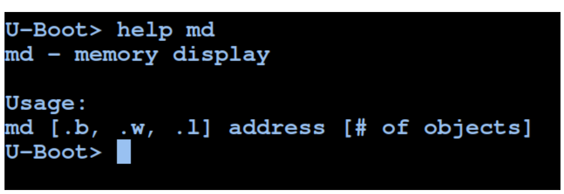


Figure 31: help md

The switches .b .w .l are for defining the output size from the memory read command md.

- .b = byte (8 bits)
- .w = word (16 bits)
- .l = long (32 bits)

Now read the memory starting from the offset address and read 32 bytes of data using the following command. The first 6 bytes should be xmuart.

md.b 0x83fb47fd 32

You may need to expand the width of the terminal application window to be able to view the information because it may be wrapping on the console. The data should look like Figure 32 below.

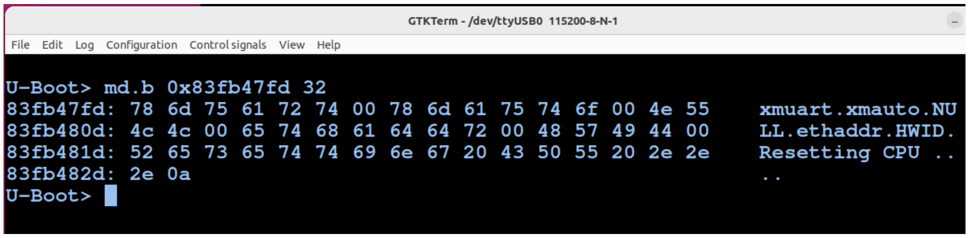


Figure 32: md.b read 32 bytes

If that all looks good, then the next step is to alter xmuart in memory so the filter does not work, and you will then be able to make the needed entry into U-Boot environment variables. To do this, run the following memory write command. This command will write hex 41, which is an “A”, to the starting memory address where xmuart is stored in running memory, changing xmuart to Amuart.

mw.b 0x83fb47fd 41

Once you have run the above command, rerun the following memory read command and see if the xmuart has been changed to Amuart. If successful, the results should look like Figure 33.

md.b 0x83fb47fd 32

```
U-Boot> mw.b 0x83fb47fd 41
U-Boot> md.b 0x83fb47fd 32
83fb47fd: 41 6d 75 61 72 74 00 78 6d 61 75 74 6f 00 4e 55    Amuart.xmauto.NU
83fb480d: 4c 4c 00 65 74 68 61 64 64 72 00 48 57 49 44 00    LL.ethaddr.HWID.
83fb481d: 52 65 73 65 74 74 69 6e 67 20 43 50 55 20 2e 2e    Resetting CPU ..
83fb482d: 2e 0a                                              ..
U-Boot>
```

Figure 33: Memory write an "A"

If it looks correct, you should now be able to modify the environment variables and add xmuart=0.

To make this change to the environment variables within the U-Boot console you will need to run the following command:

setenv xmuart 0

Once the above command is run and before saving the environment variables to memory you will need to validate that it was properly added. This is done by running printenv to show the U-Boot environment variables and examining the output. Run the following command and examine the output for your changes:

printenv

Note: Any new environment variables added should initially show up at the bottom of the list.

DO YOU SEE xmuart=0 ? You should, and it should look like Figure 34 below.

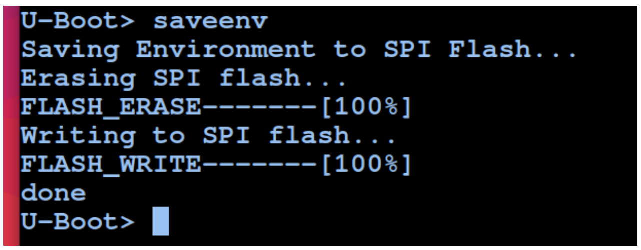
```
stdout=serial
tk=mw.b 0x81000000 ff 800000;tftp 0x81000000 uImage; bootm 0x81000000
ua=mw.b 0x81000000 ff 800000;tftp 0x81000000 upall_verify.img;sf probe 0;flwrite
up=mw.b 0x81000000 ff 800000;tftp 0x81000000 update.img;sf probe 0;flwrite
verify=n
xmuart=0

Environment size: 1237/65532 bytes
U-Boot>
/dev/ttyUSB0 115200-8-N-1                                    DTR RTS CTS CD DSR
```

Figure 34: printenv results showing xmuart=0

If xmuart=0 is properly written into the U-Boot environment, then the next step is to save those changes into memory. This is done by running the following command:

saveenv

When the saveenv command runs you should see output that looks like Figure 35 below.



```
U-Boot> saveenv
Saving Environment to SPI Flash...
Erasing SPI flash...
FLASH_ERASE-------[100%]
Writing to SPI flash...
FLASH_WRITE-------[100%]
done
U-Boot>
```

Figure 35: saveenv

The final step — if you did everything correctly — is to reset the devices and login to the UART console as root. You can restart the system by running the following or by powering the camera off and back on.

reset

You should now see the system booting up past the loading kernel prompts. Once it has booted up completely just hit enter a couple of times and you should get a login prompt.

You can now login with root using the password you gained from the john-the-ripper.

```
eth2      mp_arx:start
eth2      mp_arx:Indicating Receive Packet to network start

(none) login: (none) login: root

Password:
Apr 12 22:30:17 login[429]: root login on 'ttyAMA0'

~ #
~ # pwd
/
~ #
~ # ls
bin       dev       home      linuxrc   proc      sbin      tmp       var
boot      etc       lib       mnt       root      sys       usr
~ #
~ #
```

CONGRATS, YOU NOW HAVE ROOT ACCESS!!!

Did you enjoy this exercise? If so, check out our hands-on IoT hacking exercises from previous DEF CON events, as well as other IoT blogs here.

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.