h550 project - Security assessment of an AP

Esteban Aguililla Klein

January 23, 2025

Contents

1 Abstract

Closed source firmware cannot be trusted. Therefore, we will show how to hack an access point and search for exploitable vulnerabilities from the hardware to the software. Doing, so we will take a look at how U-boot and BusyBox shells can be used and leveraged to our advantage. Afterwards, we will showcase a level 10[1] known vulnerability (which is in fact a backdoor placed by the constructor) that allows remote code execution and how it was found.

2 Introduction

Nowadays most IOT devices contain some flavor of a vulnerability, backdoor or critical bug. However, those problems in some cases are never fixed as IoT devices are dependent on their vendors for firmware update due their code being proprietary (in most cases). Furthermore, some users might forget (or not know) that their device's software is not updated and by extension, vulnerable. This paper takes the approach of a guide on how to perform basic hardware hacking on everyday embedded devices. What should be taken away from this work is the process which aims to be as generic as possible and not the results as the chosen target is now discontinued and irrelevant. Therefore, we will start by setting the scope of our analysis then, do reconnaissance under those constraints followed by the exploitation of our findings.

3 Scope

The device is an access point from Cisco released in October 2008, aimed at small businesses. This products has been discontinued in 2019 and its sales ended back in 2014.

(a) Product picture (b) Bottom (c) Inside

Figure 1: Pictures of of the WAP4410N

The device can be powered up using a barrel adapter or through PoE which hints at its purpose being ease of batch deployment. On the network side, it support

[1] https://nvd.nist.gov/vuln/detail/CVE-2014-0659

802.11n, 802.11g and 802.11b. With the time period in mind, for 802.11b and g, the only supported encryption protocol was WEP (64/128 bits in this case). Also, firmware updates were handled through the web portal : there is no I/O on the device. On figure 1c, we can see a cable going outside of the device, this is the UART access which was enabled by default.

3.1 Lab setup

The testing setup is made of the WAP4410N, a router and a computer : for simplicity sake, the router is out of scope from this analysis and all its security features are disabled. The computer has a wireless connection to the AP.

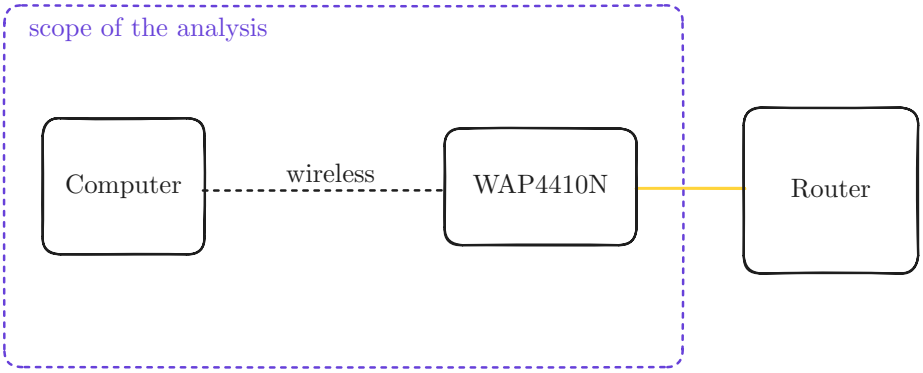

Figure 2: Lab setup

3.1.1 Equipment

The following table contains all the software and hardware required for basic hardware hacking.

software	hardware
Wireshark, nmap GNU Screen, xxd, dd, binwalk squashfs-utils radare2, ghidra	FT232 board USB-A UART adapter Multimeter Dupont Cables

Figure 3: List of used software/hardware

At first, we intended to use a SPI programmer (CH341a) to dump the firmware more easily using flashrom. However, as we will see in a later section, it was not feasible in this context without specialized hardware tools. On the software side, there are no real limitations as most of the state of the art tools are free and open source (with the exception of IDA).

4 Reconnaissance

This is the most important part of any hardware hacking project : the more information about the target we have the easier finding vulnerabilities will become. We can first start by searching on the internet if there are any information on our device : usually checking its fcc id and the nvd is a good starting point.

4.1 Physical

Teardown The device is held together by four 8-points screws under rubber feet which indicates that it wasn't meant to be opened (nor repaired . . .). Once inside, most of the device is empty and everything is available without further need to dismantle it.

IC reconnaissance At this point we can start looking for interesting ICs like flash memory or processors :

- reference : MX29LV640DBTC-90G
- format : 48 TSOP
- manufacturer : Macronix
- size : 8MB

- family : Atheros 9100
- reference : Atheros9132
- manufacturer : Qualcomm
- architecture : MIPS 24K V7.4 32bits
- endianness : big endian
- bogomips : 265.21

The MIPS architecture is very common on embedded devices and very easy to reverse engineer its assembly. However, this makes it impossible for us to use frida as MIPS is only available in the pro version. Then, regarding the NAND flash, as hinted before, reading a 48 TSOP IC requires specialised and expensive hardware. Therefore, using an SPI programmer to dump it is out of the question.

Finding the UART This is the most important step of the physical reconnaissance : the UART allows us to communicate with the device from a computer. To find it, we have to usually look for 3 to 4 pins next to each other.

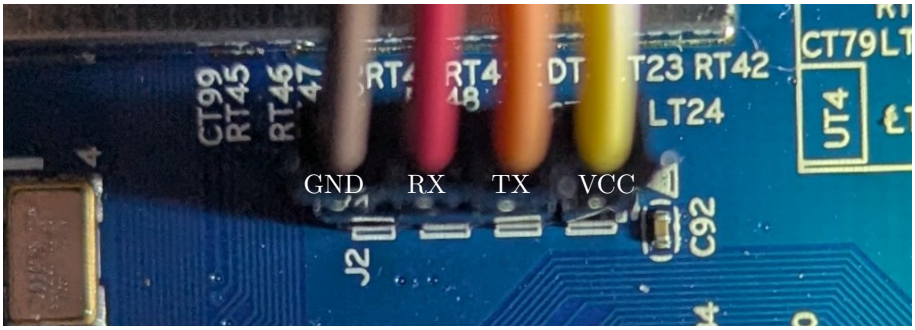

Figure 4: UART connected through Dupont cables with labeled ports

On the device, there are 4 male header pins that might be the UART. We can check using a multimeter. First, we must find the ground, to do so, we can look at any IC datasheet and find a ground pin then, we can try to find a grounded metal to connect to it instead of a pin by checking for continuity. After having found a ground connection we can start taking measurements of our candidate,

pin	R_{VSS}	V	info
1	$8.6k\Omega$	$\approx 3.3V$	VCC
2	$\infty k\Omega$	$\approx 0V - VCC$	TX
3	$\infty k\Omega$	$\approx 0V$	RX
4	0 k Ω	$0V$	GND

Figure 5: Multimeter measurements

The voltage goes from 0V to VCC, at startup because of the boot log outputs a lot of data. The RX has a voltage of 0 as its used to receive data and doesn't send any. The next step is to try connecting the UART to the computer using our TTL,

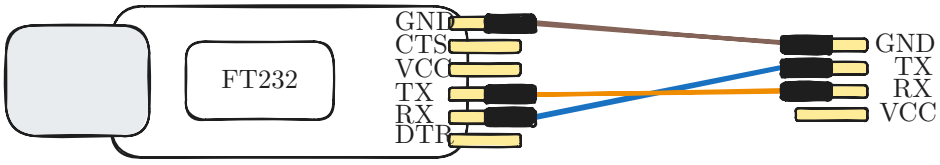

Figure 6: UART to USB-A

Regarding the baudrate, we can try 115200 as its the most common value,

```
screen /dev/ttyUSB0 115200
```

This is indeed the correct baudrate however, if that wasn't the case we would have had to find it by trial and error or by using a logic analyzer.

4.2 Bootloader

4.2.1 Boot log

First, we will take a look at the boot log that is displayed before getting access to any console as analyzing it provides a lot of information about the hardware, software, ... One such information is the bootloader : U-Boot 1.1.4 which is a very common and open source bootloader.

```
arg 1: console=ttyS0,115200
arg 2: root=31:02
arg 3: rootfstype=jffs2
arg 4: init=/sbin/init
arg 5: mtdparts=ar9100-nor0:256k(u-boot),
        64k(u-boot-env),
        6464k(rootfs),
        1280k(uImage),
        64k(nvram),
        64k(calibration)
arg 6: mem=32M
```

From the capture snippet above we see that the rootfs uses jffs2 however, this is not the case.

```
Kernel command line: console=ttyS0,
        115200 root=31:02 rootfstype=squashfs init=/sbin/
            init mtdparts=ar9100-nor0:256k(u-boot),
```

We can see that rootfs uses squasfs which is a read only filesystem as shown in the snippet above : this can be confirmed in the console later. But, how is the device configuration stored persistently ? Usually an embedded device has two filesystems in an overlayfs where the lower is read only and the upper read write (jffs2) for persistency.

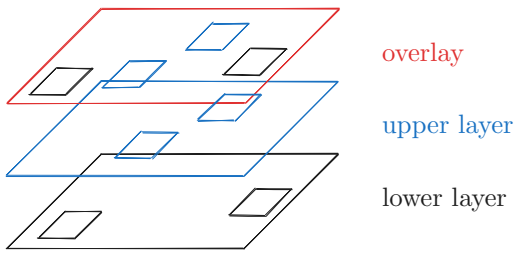

Figure 7: diagram of an overlayfs

However, the access point uses non volatile memory to store persistent device configuration (there is a cli to interact with it /usr/sbin/nvram). Therefore, all the configurations is handled outside of any filesystem in a key-value store in nvram.

4.2.2 U-boot

At some point we are prompted to interrupt the auto boot : by doing so, we get the U-boot console which allows us to get more information on the device,

```
ar7100> bdinfo
flashstart   = 0xBF000000
flashsize    = 0x00800000
flashoffset  = 0x0002F690
```

We now know the memory address and size of the flash storage : this will come handy during the firmware extraction in 5.1.1. There are other useful commands available to us such as tftpboot but in this context, they are not useful.

4.3 Console

If we leave the autoboot alone, we get into a root Linux ash shell. There are device specific CLI commands and a reduced busybox. We can get the firmware version with the fwversion command : 2.0.4.2. As we are already root, we can't really do any privilege escalation.

4.4 Network

We can run a port scan on the device using nmap,

PORT	STATE	SERVICE
80/tcp	open	http
443/tcp	open	https
32764/tcp	open	unknown

The http(s) ports are used for the web administrative portals : when using the http web portal the credential are sent through base64. This isn't really a problem as the http port will most likely be disabled by any competent IT staff in real use cases. However, the port 32764 being open for TCP is not normal and should be explored. To do so, we will try to connect to it using netcat : when sending a random input we get answered with ScMM and the connection closes (if left alone it closes by itself).

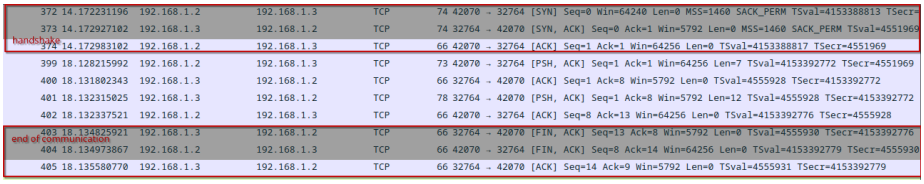

Figure 8: Capture of the exploit traffic

We can analyze the communication in wireshark : the answer we get is 12 bytes long : 53 63 4d 4d ff ff ff ff 00 00 00 00 which translates to ScMM. After googling this string we get a matching vulnerability

5 Exploitation

5.1 Dumping the firmware

In this section we will see how to exploit the device by retrieving its firmware. Then, we will showcase a vulnerability and explain it by reversing the related program from the dumped firmware.

5.1.1 Through the bootloader

This is the "classical" technique used to dump firmware as its very simple (but slow). To perform it, we will use the addresses from 4.2.2 and capturing the output. The size is written in decimal and divided by 4 as it reads by blocks of dwords. The dump uses the xxd format and has to be cleaned. However, this technique didn't work in our case. The reason, is unclear as the output is not entirely unreadable when using binwalk : some possible reasons might be,

- glitchy cable
- this version of U-Boot has a problem with memory display
- out of bound area on the NAND flash
- endianness problem

5.1.2 Through the console

To do so, we have to update the version of busybox : we will download a precompiled busybox binary and serve it to our device through an ftp server. We download it in the /tmp directory as it the only location in a read only filesystem where we can write a file. From there, we use netcat to exfiltrate the mtd partitions to our computer then concatenate them together to make the whole firmware image :

/home/aaaaaa/aaaaaaaaa/aaa/aaaaaaaaaaaa/dump/firmware2.0.4.2.bin

DECIMAL	HEXADECIMAL	DESCRIPTION
158992	0x26D10	CRC32 polynomial table, big endian
327680	0x50000	SquashFS file system, big endian, version: 3.0, compression: unknown, inode count: 794, block size: 65536, image size: 4761817 bytes, created: 2011-05-13 10:54:02
6946816	0x6A0000	uImage firmware image, header size: 64 bytes, data size: 875547 bytes, compression: gzip, CPU: MIPS32, OS: Linux, image type: OS Kernel Image, load address: 0x80002000, entry point: 0x801C2000, creation time: 2011-05-13 10:51:49, image name: "Linux Kernel Image"
8269351	0x7E2E27	PEM private key
8270238	0x7E319E	PEM certificate

5.2 CVE-2014-0659

This CVE corresponds to the "weird" open TCP port that we found in 4.4. We will explain its capabilities by using the script provided by the discoverer on his repo. After adapting it to our situation, we can use it to remotely dump the device configuration. However, the backdoor allows for remote code execution which means that the device is absolutely unsecure and should not be used under any circumstances.

5.2.1 Reverse Engineering

The vulnerability is located in the **usr/sbin/scfgmgr** program : we know this because it was mentioned by the discoverer of the vulnerability. However, to find it naturally, one can look at the process list and analyze all the programs that sound like they don't belong there. The backdoor opens a new child process for each new connection to its socket and starts an alarm of 10s during which you have to send the correct character sequence (this is why the netcat console closes by itself when left alone) then it :

1. Checks if the response starts with the correct code *(ScMM)* and is followed by and identifier (we will call it **id**) and optionally an argument
2. The program matches the **id** and, if it matches with a known **id** it performs the corresponding action *(eg. reboot the device, dump the firmware,...)*
3. The child process terminates

One of those *id*, opens a pipe that calls for **popen** with the user input as argument and returns the output through the socket. In the snippet above,

```
read_ret = (char *)malloc(1);
*read_ret = '\0';
*output = read_ret;
__stream = popen(command,"r");
```

Figure 9: Snippet of vulnerable code

command is a user input and, the resulting stream from the **popen** command is read into the **read_ret** which is pointed by **output** : a function argument. After, returning from the pipe, **output** is written to the socket[2].

5.2.2 Patch

This backdoor was patched in a later version of the firmware however, according to the original discover it was only obfuscated[3]. We cannot verify this as the specific firmware version is not downloadable anymore.

[2] The ghidra project with comments is included in the project repo
[3] https://www.synacktiv.com/ressources/TCP32764_backdoor_again.pdf

6 Conclusion

Through the exploitation of the WAP4410N access point, we identified multiple attack vectors and outlined a simple approach to basic hardware hacking. This process involved, gathering information about the device, gaining access and leveraging the gathered information to exploit it. We highlighted how a device that is meant for usage in a small business (potentially handling critical information) was poorly secured. Issues such as the headers for the UART being soldered, unprotected console access and a backdoor in the firmware underscores the severe risk to the network of utilizing this device. The field of embedded devices is riddled with unsecure firmware which would go unnoticed without security researchers. However, there are too many such devices therefore, there is a need for a push towards transparency and free software such as openwrt. Furthermore, because of the quantity of internet connected device, most of them are not certified and should be strongly considered before connecting them to the internet.

References

Andrew Bellini, Anyone can hack IoT- Beginner's Guide to Hacking Your First IoT Device, https://www.youtube.com/watch?v=YPcOwKtRuDQ&t=3s

Sick codes, Advanced Hardware Hacking with Sick Codes, https://www.youtube.com/watch?v=v8Wxojm6UAw

Eloi Vanderbeken,Matt "hostess" Andreko, SerComm Device - Remote Code Execution (Metasploit), https://www.exploit-db.com/exploits/30915