Bachelor thesis
Computing Science

Radboud University

A security analysis of the TP-Link TL-WR802N Router

Author:
Ciske Harsema
s1010048

First supervisor/assessor:
dr. Buhan, I.R. (Ileana)
ileana.buhan@ru.nl

Second assessor:
dr. ir. Poll, E. (Erik)
e.poll@cs.ru.nl

January 6, 2022

Abstract

With the rise of cheap internet connected devices, there has also been a rise of incidents related to such devices. While not very powerful in isolation, the combined power of these devices can lead to significant real-world impact. One clear example of this impact are record breaking Distributed Denial of Service attacks, performed by infected devices part of botnets, through malware such as the Mirai family. Devices are frequently infected through common security vulnerabilities, such as default credentials. There is a clear need to improve the security by eliminating such vulnerabilities. As vendors have been slow to react, external research highlighting vulnerabilities is a useful tool to both put pressure on vendors to address the issues, but also help inform consumers in making safer choices when purchasing such devices. In this research project we investigate the state of security for one such device: the TP-Link TR-WR802N Router. The goal of this research project is to ascertain to what degree the device is at risk from vulnerabilities commonly found in similar devices, and document the results in a report.

Contents

1 Introduction **3**

2 Preliminaries **5**

3 Scope **7**

4 Research **9**
4.1 Physical analysis 9
4.1.1 Interpreting the boot log 14
4.2 Wireless scanning 15
4.3 Obtaining the firmware 15
4.3.1 Extracting firmware from the device 17
4.3.2 Interpreting extracted firmware 17
4.4 Initial filesystem analysis 18
4.4.1 Cracking the admin password 20
4.4.2 Overview of dependencies 21
4.5 Known vulnerabilities 21
4.5.1 Open Source CVEs 21
4.5.2 GDPR system 21
4.5.3 Command injections 24
4.5.4 Buffer overflows 24
4.6 Comparison of firmware versions 30
4.6.1 Firmware header 31
4.6.2 Bootloader 32
4.6.3 Kernel 33
4.6.4 Filesystem 35
4.7 Decrypting config files 40
4.8 Generation of sensitive information 50
4.8.1 Initial device parameters 51
4.8.2 WPS Web Interface 51
4.8.3 Dropbear 58
4.9 Firmware update process 63
4.10 Dynamic analysis 72

4.11 Findings 75
4.11.1 Summary to research questions 75
4.11.2 Overview of vulnerabilities 76

5 Related Work **79**

6 Future work **81**

7 Conclusions **82**

A Appendix **88**
A.1 TL-WR802N Boot log 88
A.2 iwlist scan results 108
A.3 TCP portscan results 109
A.4 UDP portscan results 112
A.5 Third party dependencies 113
A.6 List of closed source binaries 113
A.7 Config decompression tool 114
A.8 Firmware filesystem compare script 116

Chapter 1

Introduction

Research has shown that the state of IoT security is lacking, with some researchers even going as far as calling it a mess[17]. The state of router security is also worrisome, as shown by numerous results that found firmware vulnerabilities in 83%[5] of investigated routers.

Botnets pose a serious threat to internet connected users and systems[9]. Malware families such as Mirai have abused default credentials in IoT devices in order to infect vulnerable devices, turning them into bots[10]. These botnets have been used to launch devastating DDoS attacks, eventually exceeding 1 Tbps, that caused outages in several important services, resulting in substantial real-world impact[10, 7].

TP-Link is a major producer of network products, including over at least 80 routers[28]. One of their products is the TR-WR802N (EU, v4) router, introduced in 2019[29]. Prior research has shown that TP-Link routers too suffer from vulnerabilities[15]. As such, the subject of this research is to conduct a security analysis of the TP-Link TR-WR802N router, with the purpose of gaining a better understanding of the state of security for this particular device. The scope of the research, along with specific research questions, is elaborated in Chapter 3.

Performing such security analyses is not just a useful tool for informing potential customers. By collaboration with vendors any found vulnerabilities can be patched, resulting in safer devices. As new devices are routinely introduced, it is also important that these analyses continue to occur.

The remainder of this research paper is structured as follows:

- Chapter 2 covers the preliminary knowledge needed to understand this thesis.
- Chapter 3 covers the scope of the security analysis, along with concrete research questions.

- Chapter 4 covers the investigation conducted to answer the research questions, and presents the findings.
- Chapter 5 covers how this research relates to the broader academic field.
- Chapter 6 covers directions for additional follow up research.
- Chapter 7 covers the conclusion of the research, presenting the key findings.

Chapter 2

Preliminaries

Embedded devices roughly speaking consist of two parts: the hardware, and the software it runs. The software in this context is frequently referred to as firmware, as it is closely integrated with the hardware in question, as opposed to more general purpose software than can run on a large number of devices with various hardware support. This is not to say the firmware may not also contain more general purpose software, but by enlarge the nature of it means it runs fairly directly on the hardware; there are not many abstraction layers.

The hardware can vary to a large degree, such as devices with wireless capability requiring hardware to provide this capability, but there are some common elements that are also relevant for the security analysis.

Most notable is the existence of some kind of processor unit, which executes the software instructions. Various architecture families exist, such as x86, ARM, and MIPS. The processor can be integrated with other components, such as input/output ports, memory, and co-processors, into a single chip. Such a chip is called a System on a Chip (SoC).

The WR802N device in question contains a MIPS based processor. MIPS is a Reduced Instruction Set Computer (RISC), which compared to a Complex Instruction Set Computer (CISC) is less complex. This is achieved through means such as less instructions, typically of lesser complexity, fixed size instruction encodings, and a load/store architecture. A load/store architecture means that outside of dedicated load and store instructions, no instruction uses memory operands. Instead all operands are either registers, or constant values (called immediate values). This reduced complexity allows for a simpler hardware implementation, which makes it well suited for a constrained and low power context such as embedded devices.

Besides a processor, it is also common to find a flash storage chip that provides persistent storage. It is used to store the firmware, as well as persistent backing for file systems. The size of the flash storage varies greatly, but is usually on the order of megabytes for devices comparable to the WR802N.

The PCB connecting all the components might also expose debug pads and/or pins (called an interface) for protocols such as JTAG and UART.

UART is a basic protocol for serial data transfer. It consists of a ground pin (GND), a transmit pin (TX), a receive pin (RX), and sometimes also a voltage pin (VCC), typically 3.3 or 5 volts. Voltage changes of the RX/TX pins are used to signal logical zeros and logical ones, allowing a single bit to be communicated. These voltage changes occur with a certain rate, called the baud rate. Both the sender and the receiver must use the same baud rate in order to correctly interpret the signals.

Some devices that expose a UART interface use it for debugging purposes. It might be used to output debugging information, such as real-time log outputs, or even provide an interactive shell to the device, similar to an SSH session.

The contents of firmware vary, but typically contain three key components: a bootloader, a kernel, and one or more file systems. Once the device powers on, control is handed to the bootloader. The bootloader is responsible for setting up the device, and loading the kernel. Once the kernel is loaded, the file system(s) can be mounted. These file systems can contain configuration files, applications, and various other resources.

Chapter 3

Scope

As with any analysis, it is important to define the scope of the analysis taking place. A complete analysis would involve a detailed investigation of not only the entirety of the firmware, but also the hardware components contained in the device. This not only requires a broad range of knowledge, and specialized equipment to analyze the hardware components, but also a significant amount of time. In the context of a Bachelor Thesis, conducted by a sole researcher, this is simply not feasible. A decision therefore has to be made what is included in the security analysis, and what is omitted.

To aid us with this decision we first define our attacker model, which documents the presumed capabilities of an attacker/threat actor. This has significant impact on the scope of the research, as a nation state actor has far more capabilities (in terms of personal, knowledge, and equipment) compared to an individual attacker that only has access to knowledge and methods in the public domain, and does not possess expensive equipment. Our chosen threat actor is an individual attacker, with capabilities that closely mimic those of the researcher. The motivation for this is to make the findings of this research representable for real-world scenarios. The capabilities of the attacker are: access to relevant knowledge in the public domain, physical access to the device during research, possession of configured credentials during research, and access to cheap equipment such as UART USB adapters and multimeters. Notable limitations to the attacker capabilities are: attacks that require active hardware manipulation, and attacks on the web interface outside of aspects that directly interact with the device, such as firmware updates and WiFi pin code generation. We also exclude any hardware based side channel attacks on cryptography, such as (differential) power analysis. The goal of the attacker is to gain access to devices, so that they can be taken over and infected with malware, turning the devices into bots under the control of the attacker.

The focus is placed on investigating common security vulnerabilities, in addition to enumerating basic security features and dependencies. This results in the following research questions, henceforth referred to as **RQ n**, where **n** is replaced by the respective number listed below:

1. Does the hardware expose debug ports, such as UART or JTAG, that can be used to obtain debug output?
2. Is it possible to obtain the firmware of the device, either through debug interfaces, vulnerabilities, or other means?
3. Is it possible to flash custom firmware onto the device, not signed by the vendor?
4. What network services are running on the device by default?
5. Does a comparison of available firmware versions reveal any changes relevant to security, such as (undocumented) vulnerability patches?
6. Have prior publicly known vulnerabilities been addressed by the vendor, or does a known vulnerability remain?
7. Which software products, and specifically which versions, are used by the firmware?
8. Does the firmware expose sensitive information, such as keys, certificates, password, or other such information?
9. How is sensitive information, such as the WiFi pin code, generated?
10. How does the firmware update process work from a security perspective?

While investigating the above research questions, any security relevant information is also documented.

Not only do these research questions give an immediate overview of the current state of security, and susceptibility to common attack vectors, but it also aids in retaining a future overview of potential susceptibility. This is achieved by identifying and enumerating the dependencies of the device. It is possible that some of these dependencies are also used in other products or devices, for instance popular open source software. If at a later time a vulnerability is discovered in a specific version of a dependency, one can retroactively see the device in question is also vulnerable without first having to re-conduct a complete security analysis.

Chapter 4

Research

4.1 Physical analysis

Initial inspection of the device shows there are no screws or glue that hold the case together. In order to open the device the blue top cover, shown in Figure 4.1 must be pried open.

Figure 4.1: TL-WR802N

Afterwards the PCB can be removed from the plastic casing; there are no screws or glue to hold it in place. The front of the PCB, with part of the heat sink removed, is shown in Figure 4.2. The back of the PCB, with some leads soldered to pads, is shown in Figure 4.3.

At the front of the PCB we can see the SoC in Figure 4.4, which is a Mediatek MT7628NN chip. This chip is specifically designed for AP/router devices, and integrates amongst other features 2.4GHz WiFi 4 (n) and Ethernet[30].

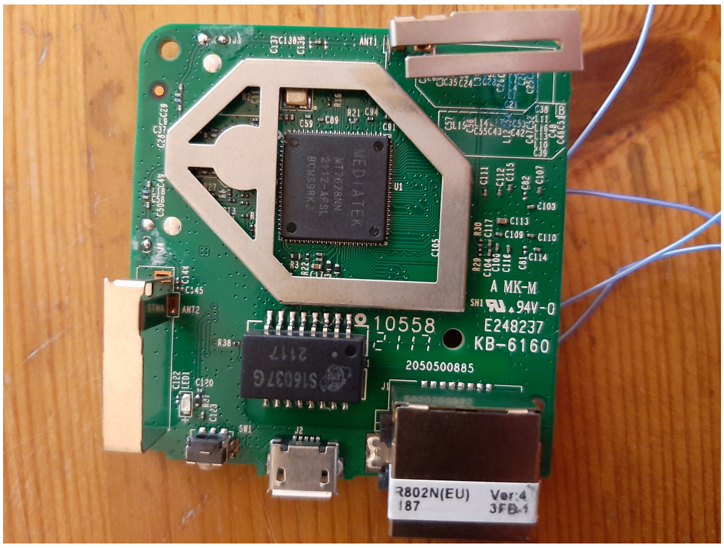


Figure 4.2: Front of the PCB

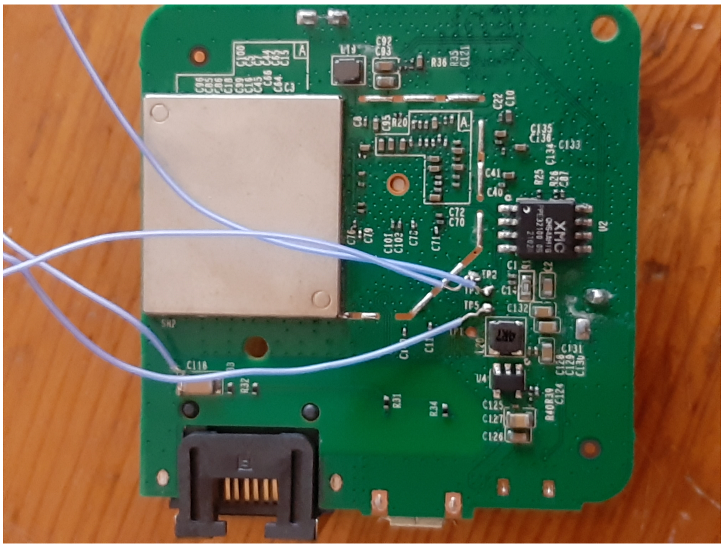


Figure 4.3: Back of the PCB

It contains a single 32-bit MIPS 24KEc core that runs at 575/580MHz[30]. Also noteworthy is the mention of UART(3) under the I/O section, which is a possible hint that the PCB could expose a UART interface.

Figure 4.5 shows an XMC QH64AH16 flash chip, which presumably contains the firmware. Other TL-WR802N boards have been observed with Winbond flash chips rather than XMC flash chips, but these parts seem interchangeable.

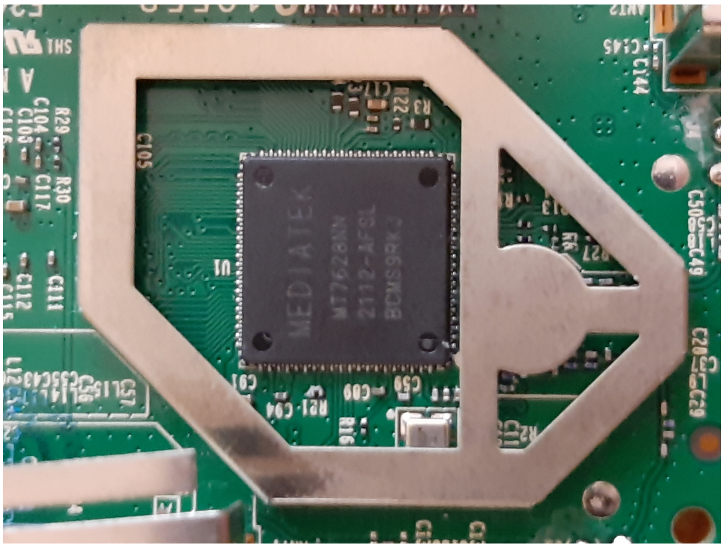


Figure 4.4: MT7628NN SoC

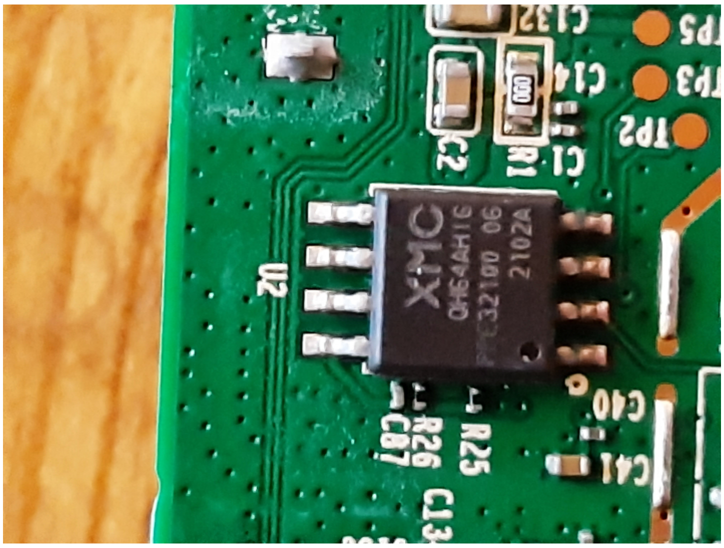


Figure 4.5: XMC QH64AH16 Flash memory chip

At the back of the PCB we observe pads labeled TP1, TP2, TP3, and TP5, as shown in Figure 4.6. Somewhat curiously, no pad labeled TP4 exists. Since we know the SoC support UART, and no other pads or pins on the PCB are labeled with anything resembling a UART interface, a hypothesis is made that the TP label stands for "Test Pad", and that these pads are related to UART. In order to test this hypothesis, we can either use a device called a logic analyzer to try and identify the interface automatically, or we can attempt to do this manually using a multimeter[8, 2].

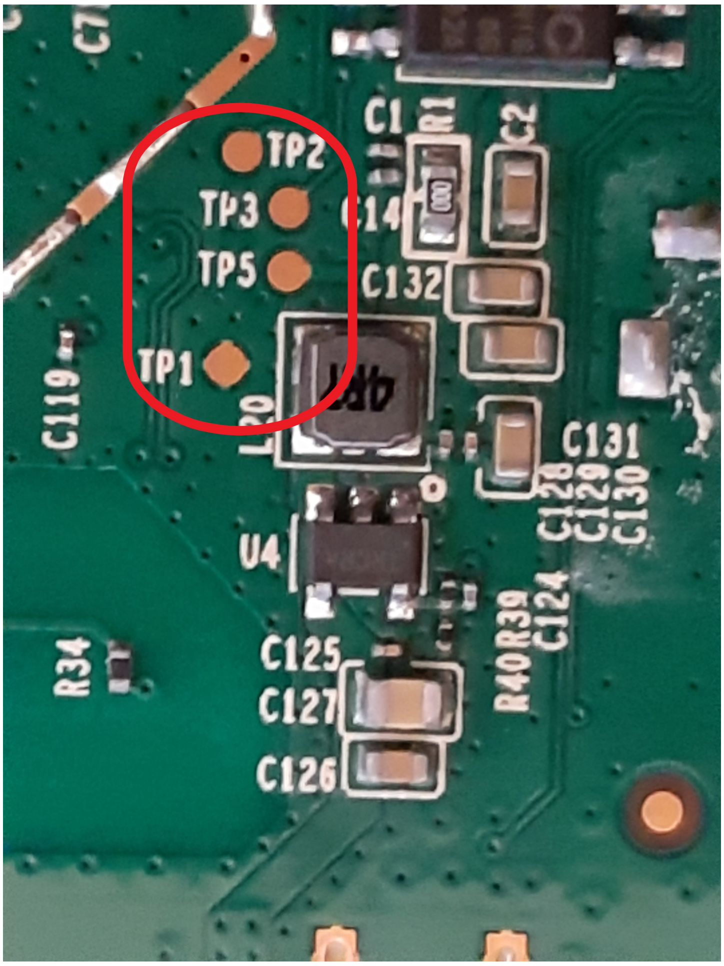


Figure 4.6: TP pads

After checking with a multimeter in continuity mode, it appears TP1 is GND (ground). Switching to DC mode and power cycling the device, we then observe that TP3 and TP5 fluctuate between 0 and 5 Volts as the device boots. This indicates that TP3 and TP5 are possibly TX (transmit) and RX (receive). TP2 appears to have no function, which would be consistent with the Mediatek specifications of the SoC supporting UART(3), which is likely a variant of UART only featuring TX/RX/GND, and no Vcc.

The next step is to confirm that TP1, TP3 and TP5 indeed expose a UART interface. We solder leads to the pads, and connect them to the GND, TX and RX pins of a TTL USB UART adapter. As we do not know whether TP3 and TP5 are TX and RX, or RX and TX, and do not know at which baud rate the UART interface operates, we begin identifying this by trial and error. As long as we guess RX and TX correctly, we should at least observe output, even if scrambled due to an incorrect choice of baud rate, giving us only two options to test. For the baud rate we assume that a common value such as 115200 or 9600 is used, but scripts do exist that can help to automate the process[2].

With the guess that TP1 is GND, TP5 is RX, TP3 is TX, and the baud rate is 115200 we observe log output from the boot process of the device after power cycling, and are eventually dropped in an interactive shell as *root*. This means we effectively have full control over the device; there is no authentication whatsoever. A snippet of the output is shown below. See Appendix A.1 for the full output. It is now clear that the device does indeed expose debug information through the UART interface, providing a positive answer to **RQ 1**.

```
$ sudo screen /dev/ttyUSB0 115200

U-Boot 1.1.3 (Jun 23 2020 - 17:33:43)


Board: Ralink APSoC DRAM:  64 MB

relocate_code Pointer at: 83fb8000

flash manufacture id: 20, device id 70 17

Warning: un-recognized chip ID, please update bootloader!

============================================

Ralink UBoot Version: 4.3.0.0

--------------------------------------------

ASIC 7628_MP (Port5<->None)

DRAM component: 512 Mbits DDR, width 16

DRAM bus: 16 bit

Total memory: 64 MBytes

Flash component: SPI Flash

Date:Jun 23 2020  Time:17:33:43
```

4.1.1 Interpreting the boot log

From the boot log we can infer several important facts. First it is clear that U-Boot 1.1.3 is used as the bootloader, specifically the Ralink 4.3.0.0 variant adopted from the mainline 1.1.3 version. U-Boot is an open source bootloader, commonly used in embedded Linux systems. As expected, after decompressing the kernel, we see Linux boot, and then greet us with a version string, showing it is running a 2.6.36 kernel.

```
Linux version 2.6.36 (jenkins@mobile-System) (gcc version 4.6.3
    (Buildroot 2012.11.1) ) #1 Tue Jun 23 17:35:59 CST 2020
```

Throughout the boot process numerous memory addresses are printed that could aid in reverse engineering. One such example is Memory Technology Device (MTD) output, which reveals the memory partitioning of the 8 MiB address space.

```
mtd .name = raspi, .size = 0x00800000 (8M) .erasesize = 0
    x00010000 (64K) .numeraseregions = 0
Creating 7 MTD partitions on "raspi":
0x000000000000-0x000000020000 : "boot"
0x000000020000-0x000000160000 : "kernel"
0x000000160000-0x0000007c0000 : "rootfs"
mtd: partition "rootfs" set to be root filesystem
0x0000007c0000-0x0000007d0000 : "config"
0x0000007d0000-0x0000007e0000 : "romfile"
0x0000007e0000-0x0000007f0000 : "rom"
0x0000007f0000-0x000000800000 : "radio"
```

Lastly, we observe Dropbear, a popular open source lightweight SSH2 server, is used to provide SSH access to the device. 1024 bit RSA and DSS (DSA) keypairs are generated, which can be used to authenticate. The SSH server is then started on port 22, using the file `/var/tmp/dropbear/dropbearpwd` for password authentication.

```
[ util_execSystem ] 141:  prepareDropbear cmd is "dropbearkey -t
     rsa -f /var/tmp/dropbear/dropbear_rsa_host_key"

Will output 1024 bit rsa secret key to '/var/tmp/dropbear/
    dropbear_rsa_host_key'
Generating key, this may take a while...
[ util_execSystem ] 141:  prepareDropbear cmd is "dropbearkey -t
     dss -f /var/tmp/dropbear/dropbear_dss_host_key"

Will output 1024 bit dss secret key to '/var/tmp/dropbear/
    dropbear_dss_host_key'
Generating key, this may take a while...
```

```
[ util_execSystem ] 141: prepareDropbear cmd is "dropbear -p 22
    -r /var/tmp/dropbear/dropbear_rsa_host_key -d /var/tmp/
    dropbear/dropbear_dss_host_key -A /var/tmp/dropbear/
    dropbearpwd"
```

4.2 Wireless scanning

During the wireless analysis of the device in default configuration we first scan for wireless networks on another device using `iwlist wlp59s0 scan`. The full scan output is shown in Appendix A.2. From the output we observe that the device is broadcasting a network under ESSID `TP-Link_B488`, and uses WPA2 Version 1, CCMP/PSK for the wireless security.

The default wireless password, which is a sequence of eight numeric digits, is printed on a label attached to the case of the device. It is unclear at this point how this default password was generated. Assuming it is generated as a sequence of eight independent digits with uniform probability, there are a total of 10^8 possible combinations. This would mean an entropy of $\log_2(10^8) \approx 26.6$ bits, which offers little protection against an attacker that has access to GPU accelerated password cracking, conducting a bruteforce/-dictionary attack as outlined by numerous researchers [14, 11].

After connecting to the wireless network we note from our default gateway and DNS parameters that the IP address of the device is `192.168.0.1`. We now conduct TCP and UDP portscans with Nmap, using commands `nmap -sV -sC -oA scan -p1-65535 192.168.0.1` and `nmap -sV -sC -oA scan-udp -sU 192.168.0.1` respectively. The full output of the TCP scan is shown in Appendix A.3, whereas the UDP scan output is shown in Appendix A.4. From the output we conclude that the device is running a DNS service on port 53, UPnP on port 1900, DHCP on port 67, SSH on port 22, and HTTP webserver on port 80, which provides our answer to **RQ 4**.

4.3 Obtaining the firmware

In order to obtain the firmware of the device we have two main approaches. We can either try to extract the firmware from the device itself, or obtain a copy from an external source, such as the vendor. In our case the vendor lists three firmware updates [31], which after extraction contain a `.bin` file of roughly 8 MiB. While uncertain at this point, this is a strong indication that the update files contain a full copy of the firmware, as the size roughly equals the in memory size observed from the boot log.

We download the latest version, at the time of writing `200623`, and extract the zip archive[1]. Using binwalk [50] we conduct a `--signature` scan:

[1] SHA1 checksum of `.bin` file is `13d5b9583472eeedc72a779bb86977dfdad0221`

```
DECIMAL          HEXADECIMAL      DESCRIPTION
--------------------------------------------------------------------------------

82048            0x14080          U-Boot version string, "U-Boot
    1.1.3 (Jun 23 2020 - 17:33:43)"
132096           0x20400          LZMA compressed data, properties:
    0x5D, dictionary size: 8388608 bytes, uncompressed size:
    3286220 bytes
1442304          0x160200         Squashfs filesystem, little endian
    , version 4.0, compression:xz, size: 4481884 bytes, 672
    inodes, blocksize: 131072 bytes, created: 2020-06-23 09:41:13
```

We can see the bootloader is detected, as is a blob of LZMA compressed data, which is likely the compressed Linux kernel. In addition we see a Squashfs filesystem, so we attempt to `--extract` it. This not only extracts the Squashfs filesystem, but also places the LZMA blob in a separate file. Conducting another signature scan on it using binwalk reveals it is indeed the Linux kernel:

```
DECIMAL          HEXADECIMAL      DESCRIPTION
--------------------------------------------------------------------------------

818514           0xC7D52          PGP RSA encrypted session key -
    keyid: 801000 205242C RSA Encrypt-Only 1024b
2482280          0x25E068         Linux kernel version 2.6.36
2482452          0x25E114         CRC32 polynomial table, little
    endian
2522064          0x267BD0         DES SP2, little endian
2522576          0x267DD0         DES SP1, little endian
2541392          0x26C750         CRC32 polynomial table, little
    endian
2792924          0x2A9DDC         xz compressed data
2844540          0x2B677C         Neighborly text, "
    NeighborSolicits6InDatagrams"
2844560          0x2B6790         Neighborly text, "
    NeighborAdvertisementsorts"
2848027          0x2B751B         Neighborly text, "neighbor %.2x%.2
    x.%pM lostrename link %s to %s"
3113520          0x2F8230         Certificate in DER format (x509 v3
    ), header length: 4, sequence length: 12160
3280896          0x321000         ASCII cpio archive (SVR4 with no
    CRC), file name: "/dev", file name length: "0x00000005", file
     size: "0x00000000"
3281012          0x321074         ASCII cpio archive (SVR4 with no
    CRC), file name: "/dev/console", file name length: "0
    x0000000D", file size: "0x00000000"
3281136          0x3210F0         ASCII cpio archive (SVR4 with no
    CRC), file name: "/root", file name length: "0x00000006",
    file size: "0x00000000"
3281252          0x321164         ASCII cpio archive (SVR4 with no
    CRC), file name: "TRAILER!!!", file name length: "0x0000000B
```

```
", file size: "0x00000000"
```

4.3.1 Extracting firmware from the device

While optional at this point as we presumably already have access to the complete firmware, we will still attempt to extract it from the device itself to see if it possibly contains more information, and ascertain the presence of any security measures.

Gaining access to a U-Boot shell would allow us to dump arbitrary memory over the serial interface through a command such as `md` (memory dump). There seems to be a wide variety amongst embedded devices using U-Boot however when it comes to accessing this shell. Some devices boot straight to it, others provide a countdown during which a key has to be pressed, or pressing/holding a certain key during the boot process will cause the device to boot into the U-Boot shell. Unfortunately none of these methods worked for the device in question, causing this avenue to be abandoned.

An alternative approach is to de-solder the flash chip itself, physically removing it from the PCB, and placing it in an external flash programmer. This programmer is then used to extract the memory contents. Afterwards the flash chip can be re-soldered onto the PCB. As this is a destructive method that can potentially damage the board, or even outright brick it if done incorrectly, this avenue was not explored further, favoring other methods instead.

Next we consider a SPI (Serial Peripheral Interface) based method, which is outlined by Chantzis et al. [2]. We connect a SOIC clip to the pins of the flash chip, which is subsequently connected to a Bus Pirate [52]. This Bus Pirate acts as a USB to serial adapter, and allows us to interface with the flash chip as if we had de-soldered it and placed it in an external programmer. Finally we use Flashrom [53], and issue a read operation using the Bus Pirate as an adapter. This produces an 8 MiB file containing the device firmware.

4.3.2 Interpreting extracted firmware

Now that the firmware has been extracted, we must verify we did so correctly. It is possible that the memory content extracted is encrypted, or otherwise protected. We again use binwalk to conduct a signature scan:

```
DECIMAL        HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
81536          0x13E80         U-Boot version string, "U-Boot
    1.1.3 (Jun 23 2020 - 17:33:43)"
```

```
131584        0x20200       LZMA compressed data, properties:
    0x5D, dictionary size: 8388608 bytes, uncompressed size:
    3286220 bytes
1441792       0x160000      Squashfs filesystem, little endian
    , version 4.0, compression:xz, size: 4481884 bytes, 672
    inodes, blocksize: 131072 bytes, created: 2020-06-23 09:41:13
```

The output is nearly identical to our first signature scan of the downloaded firmware, but seems shifted by 512 (0x200) bytes. After creating a copy of the downloaded firmware, skipping the first 512 bytes, and comparing this to the extracted firmware using the diff mode in binwalk we observe that the first 0x7c0000 bytes are identical. After this the downloaded firmware file ends, but the extracted file contains a further 0x40000 bytes.

Taking our knowledge regarding the mtd memory layout into account, we know that the memory sections up to 0x7c0000 contain the bootloader, kernel, and root filesystem. This indicates that the firmware extraction was successful, and that the downloaded firmware only contains those sections, preceded by a 512 byte header.

The remaining 0x40000 bytes in the extracted firmware contain the config, romfile, rom, and radio sections. Except for the config section, these are nearly entirely filled with 0x00 or 0xff bytes. The config section does not contain any immediately readable data. The entropy graph of the firmware, generated by binwalk and shown in Figure 4.7, reflects these observations. The first two entropy spikes are the kernel and root filesystem sections, which are both compressed. The last spike is the config section, followed by the remaining sections that clearly have very little entropy. The fact that the config section has an entropy of close to 1 suggest it is either compressed or encrypted. Figure 4.8 shows the entropy of the config section in isolation. Only about 60 percent of the section is filled with data; the remainder is filled with zeros, explaining the low entropy. The section does not appear to be compressed with any conventional compression algorithm, which likely means it is encrypted.

We conclude that the answer to **RQ 2** is yes. It is possible to obtain the firmware, either by extracting it from the device, or downloading it from the vendor. Extracting it from the device does provide access to the config section, but we currently do not know how to decrypt it.

4.4 Initial filesystem analysis

Now that we can extract the root filesystem, we begin our analysis of its contents. Under `/bin` we find all the common system utilities such as `ls` and `cp`, which are just symbolic links to one `busybox` binary that implements all this functionality. Busybox is a popular open source choice on embedded

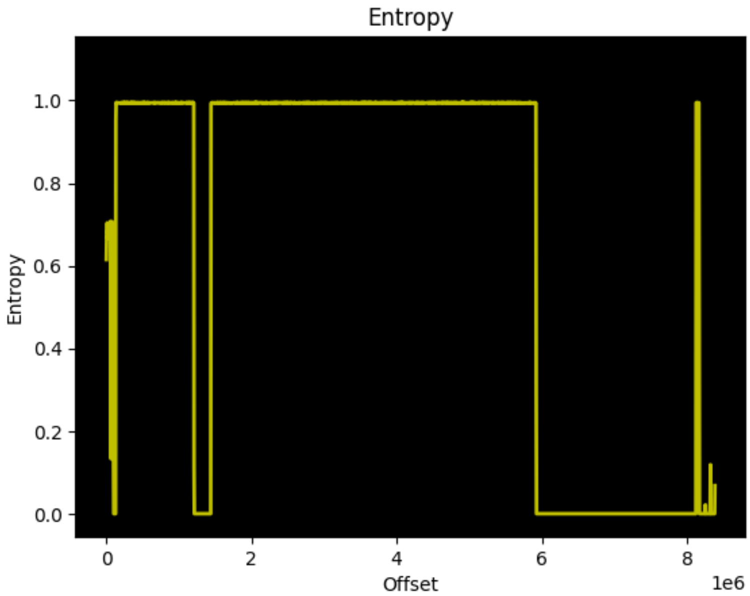


Figure 4.7: Entropy of extracted firmware binary

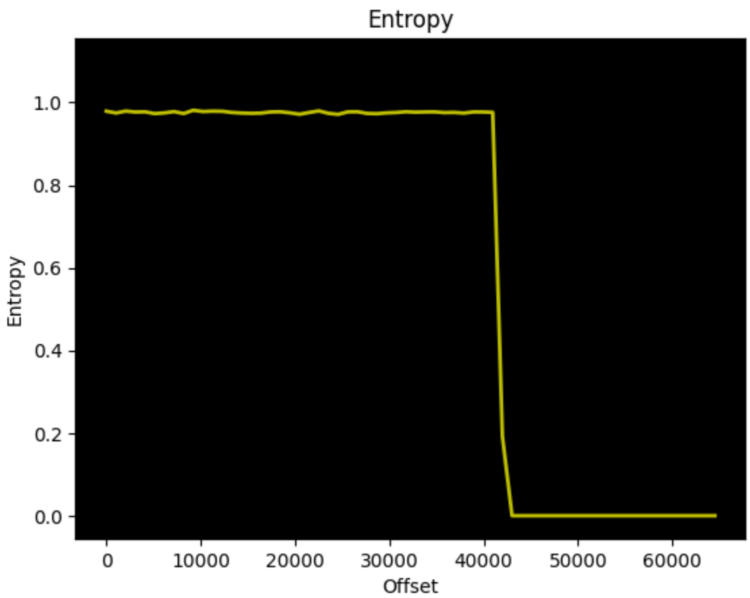


Figure 4.8: Entropy of config section

devices as it eliminates much of the overhead of having many small applications. Using the `strings` tool on this binary, we are able to identify the version as v1.19.2[2].

Under /sbin, /usr/bin, and /usr/sbin we find more binaries, as well as more symbolic links to busybox.

In /web we find all the files related to the web interface, and in /lib we find common system libraries. Under /lib/modules we find various kernel modules. Some of these modules have source code included in the GPL code released, but not all. As these are firmware/drivers related to networking[3] they are considered out of scope for this research.

Lastly, under /etc we find various configuration files, as well as several .dat files that seem to originate from mtk_ApSoC_4320/wireless/lib/firmware in the GPL code released. This again suggests binaries related to networking firmware, and thus considered out of scope. More interestingly are default_config.xml and reduced_data_model.xml. Both files are not plain XML files, but rather have an entropy near 1. This likely means they are encrypted, possibly with the same encryption scheme as the config mtd section in the extracted firmware. Especially interesting is /etc/passwd.bak, which contains an entry for the admin account, along with a hashed password. In a further attempt to discover any sensitive information such as credentials, crypto keys, or certificates, we run firmwalker [51]. This yielded no results not already noted however.

4.4.1 Cracking the admin password

In /etc/passwd.bak we observe that the admin account has a password hash of $1$$iC.dUsGpxNNJGeOm1dFio. From the structure we can infer that this is an unsalted MD5 hash[4]. In an attempt to crack the password we use John the ripper to launch a bruteforce attack against this hash. Nearly instantaneously the password (1234) is cracked:

```
Created directory: /home/ubuntu/.john
Loaded 1 password hash (md5crypt [MD5 32/64 X2])
Press 'q' or Ctrl-C to abort, almost any other key for status
1234             (admin)
1g 0:00:00:00 100% 2/3 3.030g/s 15496p/s 15496c/s 15496C/s
    1234..qwerty
Use the "--show" option to display all of the cracked passwords
    reliably
Session completed
```

[2] BusyBox v1.19.2 (2020-06-23 17:38:24 CST)

[3] raeth.ko, rt_rdm.ko, and mt_wifi.ko

[4] See the crypt(3) manpage

4.4.2 Overview of dependencies

Cross referencing all application and library binaries to the GPL open source code package by the vendor [32], as well as conducting internet searches on binary names or detected strings, reveals a significant portion of the binaries originate from open source projects. Appendix A.5 shows the list of all open source dependencies, their versions, and release dates, providing an answer to **RQ 7**. Appendix A.6 shows the list of presumably closed source binaries.

For the web interface, it seems `/web/js/cryptoJS.min.js` is a minimized version of the crypto-js library [33]. The version used is unknown. Investigating `/web/js/encrypt.js` reveals it appears to be a concatenation of TrippleDES code found online [35], and `jsbn` by Tom Wu [39].

4.5 Known vulnerabilities

4.5.1 Open Source CVEs

Looking at the list of open source dependencies, shown in Appendix A.5, we observe that many of the release dates are over a decade old. Some of these dependencies simply do not have a more recent version, such as `bpalogin`. Others do have way more recent version, such as `busybox`, with stable releases in late 2021.

Conducting a CVE search reveals several dependencies have documented CVEs, namely `busybox`, `uClibc`, `dropbear`, `openssl`, `iptables`, `U-Boot`, and `Linux` [37, 40, 43, 44, 45, 46, 47]. While it is possible that security fixes have been (selectively) back-ported, as appears the case for at least one `pppd` vulnerability (CVE-2020-8597, see Section 4.5.4), it is unclear to what extend this has occurred.

4.5.2 GDPR system

Prior research results have found several vulnerabilities in the GDPR system of the TP-Link Archer [24]. These vulnerabilities include weak entropy leading to bruteforce attacks recovering plaintext credentials. Inspecting `/web/js/tpEncrypt.js` reveals this device uses the same vulnerable key generation:

```
AES.prototype.genKey = function() {
  var key = (new Date().getTime() + "" + Math.random()
      *1000000000).substr(0, KEY_LEN);
  var iv = (new Date().getTime() + "" + Math.random()
      *1000000000).substr(0, IV_LEN);
  this.key = key;
  this._keyUtf8 = CryptoJS.enc.Utf8.parse(key);
  this.iv = iv;
```

```
  this._ivUtf8 = CryptoJS.enc.Utf8.parse(iv);
  return {
    key: key,
    iv: iv
  }
};
```

Other vulnerabilities relate to the generation of RSA parameters in `httpd`, which if fails falls back to hardcoded values:

```
undefined4 http_rsa_getPubKey(char **param_1,undefined4 *param_2
    ,int param_3) {
  int iVar1;

  if ((g_rsa_n_hex_str == (undefined4 *)0x0) || (param_3 !=
      DAT_004315f0)) {
    iVar1 = FUN_00413260(param_3);
    if (iVar1 != 0) {
      g_rsa_n_hex_str = &g_hardcoded_rsa_n;
      g_rsa_e_hex_str = g_hardcoded_rsa_e;
      g_rsa_d_hex_str = g_hardcoded_rsa_d;
    }
  }
  *param_1 = g_rsa_e_hex_str;
  *param_2 = g_rsa_n_hex_str;
  return 0;
}
```

As noted, failure can occur due to `malloc` failing to allocate memory:

```
undefined4 FUN_00413260(int param_1) {
  void *__ptr;
  void *pvVar1;
  size_t __size;

  __size = ((int)(((uint)(param_1 >> 0x1f) >> 0x1e) + param_1)
      >> 2) + 1;
  DAT_004315f0 = param_1;
  __ptr = malloc(__size);
  g_rsa_n_hex_str = __ptr;
  if (__ptr != (void *)0x0) {
    pvVar1 = malloc(__size);
    g_rsa_e_hex_str = pvVar1;
    if (pvVar1 != (void *)0x0) {
      g_rsa_d_hex_str = malloc(__size);
      if (g_rsa_d_hex_str != (void *)0x0) {
        rsa_gdpr_generate_key(param_1,__ptr,pvVar1,
            g_rsa_d_hex_str);
        return 0;
      }
      free(__ptr);
      __ptr = pvVar1;
    }
```

```
    free(__ptr);
  }
  fputs("Malloc RSA key buffer failed!\n",stderr);
  return 0xffffffff;
}
```

Furthermore we see the `b64_decode` function is unchanged, as this Base64 decode function still truncates once it encounters a space character. This matters as extra spaces are used during a replay attack to increase the data length and a sequence value, but due to the truncation does not lead to invalid decoding. The function is shown below:

```
int b64_decode(byte *param_1,int param_2,byte *input,size_t
    input_len) {
  byte bVar1;
  int iVar2;
  byte *pbVar3;
  uint uVar4;
  uint uVar5;
  byte *pbVar6;

  iVar2 = 0;
  if (input != (byte *)0x0) {
    if ((int)input_len < 0) {
      input_len = strlen((char *)input);
    }
    while ((0 < (int)input_len && ((*(ushort *)(__ctype_b + (
        uint)*input * 2) & 0x20) != 0))) {
      input_len = input_len - 1;
      input = input + 1;
    }
```

And finally, we also see `aes_tmp_decrypt_buf_nopadding_new` has not been modified, which suffers from a minor implementation flaw regarding padding:

```
if (-1 < iVar1) {
  AES_cbc_encrypt(param_1,param_2,iVar2 + 0x10,&AStack288,
      auStack360,0);
  __n = remove_padding(param_2,iVar2,0x10);
  if (__n != 0xffffffff) {
    iVar2 = iVar2 - __n;
    memset(param_2 + iVar2,0,__n);
  }
  *param_3 = iVar2;
  return 0;
}
```

We conclude none of these vulnerabilities published some 8 months ago at the time of writing have been fixed. While this is understandable for the firmware version investigated, as it was released prior to these vulnerabilities being published, there has not been a new firmware release to address these vulnerabilities either.

4.5.3 Command injections

A command injection vulnerability has been found in earlier TP-Link devices [25], which revolves around a vulnerable `execFormatCmd` function in `httpd`. No such function exists in our `httpd` binary, nor are any calls to `vsprintf` made, which was the root cause of the vulnerability. As such, this vulnerability is no longer present.

Another command injection works by restoring a config file [42]. A config file is first exported, and then the `Description` key of the `DeviceInfo` section is modified to include an injected command. A script is used to convert between the encrypted and plaintext XML formats. After uploading and restoring the modified config, the injected command is executed.

Attempting to re-create this exploit was somewhat troublesome. Exporting and converting the encrypted config to plaintext was easy, but the `Description` key was not present by default. Luckily we had already decrypted `reduced_data_model.xml` which shows it was present, and had the following default value:

```
TP-Link Wireless N Nano Router WR802N
```

So to inject a command we add the following line under the `DeviceInfo` section of our exported config:

```
<Description val="TP-Link Wireless N Nano Router WR802N`cp /etc/
    passwd /var/pwned`" />
```

With this injection we utilize backticks (command substitution), as the string is presumably passed to a `system` call. Using the provided script to re-encrypt the modified config did not work out of the box. After attempting to restore it, we are greeted with an MD5 checksum mismatch error. This was due to the script first computing the MD5 checksum, and then padding the output length to a multiple of 8, rather than first padding and then computing the checksum. After this tweak the new config was accepted, but did not seem to have triggered the command injection. It turned out the command injection is triggered by `upnpd`, which was not running as the device was configured in Access Point mode for testing. Turning the device into the default Router mode did start `upnpd`, and successfully triggered the command injection, copying `/etc/passwd` to `/var/pwned`, and thus providing proof that the command injection still works.

4.5.4 Buffer overflows

Multiple buffer overflows have been found in TP-Link routers [21, 27, 26, 41]. We will now check whether these buffer overflows are still present in this device.

The first overflow occurs in `httpd` in a function called `ipAddrDispose` [21]. This function no longer seems to exist, nor any function resembling that name. The vulnerability has also been fixed, with updated firmware released on 2019-03-12, which predates the release date of the firmware version we are investigating. Whether it was not susceptible to begin with, or has been fixed due to a patch since, it appears this vulnerability no longer exists.

The second overflow was found quite recently [27], only 6 months ago at the time of writing. The vulnerability exists in the `dm_fillObjByStr` function (line 67), found in `libcmm.so`, shown below:

```
1  int dm_fillObjByStr(int param_1,uint param_2,void *param_3,char
       *param_4,uint param_5) {
2    bool bVar1;
3    int iVar2;
4    char *end_idx;
5    char *sep_idx;
6    int iVar3;
7    char *pcVar4;
8    undefined4 uVar5;
9    uint uVar6;
10   char key_buf [64];
11   char val_buf [1304];
12   char **local_30;
13   size_t key_len;
14
15   uVar6 = param_2 & 0xffff;
16   iVar2 = 0;
17   if (*param_4 != '\0') {
18     iVar2 = dm_getObjNode(uVar6);
19     if (iVar2 == 0) {
20       cdbg_printf(8,"dm_fillObjByStr",0x794,"Get object(oid = %u
             ) info node failed.",uVar6);
21       iVar2 = 1;
22     } else if (param_5 < *(ushort *)(iVar2 + 6)) {
23       cdbg_printf(8,"dm_fillObjByStr",0x79e,
24                   "object(%s, oid = %u) buf will exceed, object
                        size is %u, but buf is %u",
25                   *(undefined4 *)(iVar2 + 8),uVar6,(uint)*(
                        ushort *)(iVar2 + 6),param_5);
26       iVar2 = 0x2650;
27     } else {
28       end_idx = strchr(param_4,L'\n');
29       bVar1 = false;
30       do {
31         if ((end_idx == (char *)0x0) && (bVar1)) {
32           return 0;
33         }
34         sep_idx = strchr(param_4,L'=');
35         if (sep_idx == (char *)0x0) {
36           cdbg_printf(8,"dm_fillObjByStr",0x7aa,
37                       "String format error: name and value is
```

```
                        separated by \"=\"\n");
          return 0x232b;
        }
        key_len = (int)sep_idx - (int)param_4;
                    /* copy KEY from 'KEY=val$' into key_buf */
        strncpy(key_buf,param_4,key_len);
        key_buf[key_len] = '\0';
        if ((*(byte *)(iVar2 + 2) & 0x10) == 0) {
          iVar3 = dm_checkAccessPermissions((short)uVar6,param_3
              ,key_buf,param_1);
          pcVar4 = key_buf;
          if (iVar3 != 0) {
            uVar5 = 0x7b9;
            end_idx = "Parameter(%s) deny access by %d";
            goto LAB_00088f04;
          }
        }
        local_30 = dm_getParamNode(uVar6,key_buf);
        if (local_30 == (char **)0x0) {
          cdbg_printf(8,"dm_fillObjByStr",0x7c0,"Get parameter %
              s\'s infomation failed.",key_buf);
          return 0x232d;
        }
        if ((*(ushort *)(local_30 + 3) & 1) == 0) {
          cdbg_printf(8,"dm_fillObjByStr",0x7c6,"Parameter(%s) 
              deny to be written.",key_buf);
          return 0x2329;
        }
        if (end_idx == (char *)0x0) {
          bVar1 = true;
                    /* copy VAL from 'key=VAL$' into val_buf */
          strcpy(val_buf,sep_idx + 1);
        } else {
                    /* copy VAL from 'key=VAL\n$' into val_buf
                        */
          strncpy(val_buf,sep_idx + 1,(size_t)(end_idx + (-1 - (
              int)sep_idx)));
          key_buf[(int)(end_idx + (-1 - (int)sep_idx) + 0x40)] =
                '\0';
          param_4 = end_idx + 1;
          if (end_idx[1] == '\0') {
            bVar1 = true;
            end_idx = (char *)0x0;
          } else {
            end_idx = strchr(param_4,L'\n');
          }
        }
        iVar3 = dm_setParamNodeString(local_30 ,val_buf);
        uVar5 = 0x7e4;
      } while (iVar3 == 0);
      pcVar4 = *local_30;
      end_idx = "Set parameter %s\'s value to object error.";
LAB_00088f04:
      key_len = iVar3;
```

```
85         cdbg_printf(8,"dm_fillObjByStr",uVar5,end_idx,pcVar4);
86         iVar2 = key_len;
87       }
88     }
89     return iVar2;
90   }
```

The param_4 parameter seems to contain a string of the form `key=val`, where the value either continues to the end of the string, or until a newline character is found. The value part of this parameter is isolated, and copied to a local buffer of 1304 bytes called `val_buf` using a call to `strncpy`. As no length checks occur, this creates a potential buffer overflow, which was found by CVE-2021-29302.

Inspecting the function further reveals that in the case where the parameter does not include a newline character, the `strcpy` alternative, shown in line 64, also suffers from a similar buffer overflow. This minor –but non-discussed– variant is closely related to the prior CVE, but unless also addressed could remain as a separate vulnerability.

Looking at how the key is processed, we observe a similar situation where the key part is extracted, and copied into a local buffer using `strncpy`, as shown in line 41. The length of the string copy is again determined by the length of the key part, without any length checks. This makes it possible to overflow the destination buffer `key_buf`, which is only 64 bytes in size, creating yet another buffer overflow not discussed in the CVE.

This means that not only the device is most likely still susceptible to the published vulnerability, but also that several closely related buffer overflow vulnerabilities exist in the same function.

The third overflow was found in the `/admin/syslog` endpoint handler in `httpd` [26]. A call to `httpGetEnv` retrieves a parameter string, which is then copied into a local buffer using `strcpy` without any length validation, resulting in a potential buffer overflow.

The `/admin/syslog` endpoint no longer seems to exist in our device binary, nor does the `httpGetEnv` function. As such, we presume the device is not susceptible to this buffer overflow.

While `httpGetEnv` no longer exists, a function called `http_parser_getEnv` does. Quickly inspecting some call sites of this function looking for nearby calls to string copy functions revealed the following function:

```
int FUN_0040b640(int param_1,undefined4 param_2,undefined4
    param_3,undefined4 param_4) {
  undefined4 test_arg;
  char *pcVar1;
  size_t sVar2;
```

```
  int iVar3;
  int *piVar4;
  undefined1 *puVar5;
  undefined arg_buf [64];
  char msg_buf [2052];

  puVar5 = &_mips_gp0_value;
  test_arg = http_parser_getEnv("testarg");
  http_tool_jsEscape(test_arg,arg_buf);
  sprintf(msg_buf,"var testarg=\"%s\";\n",arg_buf,param_4,puVar5
      );
  piVar4 = *(int **)(param_1 + 0x84);
  if (((piVar4 == (int *)0x0) || (*piVar4 != 1)) ||
     (pcVar1 = strstr(*(char **)(param_1 + 0xc),"/cgi_gdpr"),
         pcVar1 == (char *)0x0)) {
    iVar3 = http_io_output(param_1,msg_buf);
    iVar3 = -(uint)(iVar3 == -1);
  } else {
    pcVar1 = (char *)http_buf_getptr(piVar4[7],0);
    sVar2 = strlen(msg_buf);
    strncpy(pcVar1,msg_buf,sVar2);
    iVar3 = *(int *)(*(int *)(param_1 + 0x84) + 0x1c);
    sVar2 = strlen(msg_buf);
    *(size_t *)(iVar3 + 0xc) = *(int *)(iVar3 + 0xc) + sVar2;
    sVar2 = strlen(msg_buf);
    http_buf_incrpos(iVar3,sVar2);
    iVar3 = 0;
  }
  return iVar3;
}
```

While a completely different function compared to the previously discovered vulnerability, it suffers from a similar vulnerability pattern. The `testarg` parameter is returned by a call to `http_parser_getEnv`, and assigned to `test_arg`. A subsequent call to `http_tool_jsEscape` is made with `test_arg` as the source parameter, and local buffer `arg_buf` as the destination parameter. The implementation of this function is shown below:

```
void http_tool_jsEscape(char *src,char *dest) {
  char c;

  while( true ) {
    c = *src;
    src = src + 1;
    if (c == '\0') break;
    if (((c == '\'') || (c == '\\')) || (c == '\"')) {
      *dest = '\\';
      dest = dest + 1;
    }
    *dest = c;
    dest = dest + 1;
  }
```

```
  *dest = '\0';
  return;
}
```

The function essentially behaves as a strcpy, except that the characters ', \, and " are copied as \', \\, and \" respectively. Most importantly: there is no length limiting or validation. This means that the function will happily overflow our arg_buf buffer, which is only 64 bytes large, as long as the –escaped– string length of the test parameter exceeds 63 characters. This means a test parameter of a mere 32 characters is potentially enough to trigger the buffer overflow.

After the escape call, the escaped output is copied into another local buffer, called msg_buf. While this buffer is 2052 bytes large, there are again no length checks, and thus could result in a potential second buffer overflow.

Presumably the device is not vulnerable to the discovered overflow in the /admin/syslog handler, but by investigating this we have seemed to stumble onto a novel buffer overflow by accident. It remains to be seen if this overflow can be actively exploited, resulting in denial of service attack, or even remote code execution, which is left as future work.

The fourth buffer overflow is known under CVE-2020-8597, and occurs in pppd versions 2.4.2 through 2.4.8 [41]. Inspecting the strings of our pppd binary reveals the version to be 2.4.5, and thus potentially vulnerable. The overflow occurs on a local buffer called rhostname in functions eap_request and eap_response. The relevant snippet of eap_request is shown below:

```
/* Not so likely to happen. */
if (vallen >= len + sizeof (rhostname)) {
        dbglog("EAP: trimming really long peer name down");
        BCOPY(inp + vallen, rhostname, sizeof (rhostname) - 1);
        rhostname[sizeof (rhostname) - 1] = '\0';
} else {
        BCOPY(inp + vallen, rhostname, len - vallen);
        rhostname[len - vallen] = '\0';
}
```

Note that BCOPY is just a macro for memcpy, but reverses the source and destination parameters. Here inp is an input parameter of the function, acting as the source, and rhostname is the destination. While it appears a length validation occurs, it is flawed, as len + sizeof(rhostname) can overflow and wrap around back to below vallen. This means that the subsequent copy, which uses len - vallen as the length, could result in a buffer overflow. To fix the vulnerability, the length check has been changed as shown below:

```
if (len - vallen >= sizeof (rhostname)) {
```

```
  // ...
} else {
  // ...
}
```

Investigating our pppd binary in Ghidra reveals that FUN_00431b48 is eap_request. The relevant snippet is once again shown below:

```
if (len - 0x16 < 0x100) {
  memcpy(rhostname, inp + 0x16, len - 0x16);
  auStack644[len + 0x52] = 0;
} else {
  dbglog("EAP: trimming really long peer name down");
  (**(code **)(local_2a0 + -0x7720))(rhostname, inp + 0x16, 0xff);
  uStack285 = 0;
}
```

While the code looks slightly different as vallen and sizeof(rhostname) have been replaced by constants 0x16 and 0x100 respectively, and the BCOPY macros have been expanded to their memcpy form, we can still see the resemblance. More importantly, we can see that the length check uses the fixed, non-vulnerable, form. Even though the version reports 2.4.5, which according to the CVE is vulnerable, it appears security patches have been back-ported. Looking at the relevant code from the pppd binary of the prior firmware version 190428 reveals the equivalent of the relevant snippet is as follows:

```
memcpy(rhostname, inp + 0x12, len - 0x12);
auStack644[len + 0x56] = 0;
```

Clearly the trimming of long peer names was not yet implemented, whether securely or not. This complete lack of length validation makes it trivially vulnerable to a buffer overflow. Based on this we conclude the device is not vulnerable to this CVE as of the latest firmware (200623), but was susceptible in prior firmware versions.

4.6 Comparison of firmware versions

The vendor lists three available firmware updates for the device [31]. Besides a release date, we are also provided with a short changelog, as summarized below:

- TL-WR802N(EU)_V4_200623, released 2020-08-05.
  1. Optimize online detect function on AP mode.
  2. Improve the stability and security of device.
  3. Optimize remote management function.

4. Optimize the security and change the login mode.

- `TL-WR802N(EU)_V4_190428`, released 2019-05-16.
  1. Support 32 characters on username and password.
  2. Improve the stability and security of device.

- `TL-WR802N(EU)_V4_190218`, released 2019-04-29.
  1. Add Reboot Schedule function.
  2. Add display of port status.
  3. Add Lock to AP functions in RE/Client mode.
  4. Add the support for https.
  5. Add the support for static gateway configuration in AP mode.
  6. Optimize NTP and PPPoE function.
  7. Fix the issue that L2TP connection is unstable in some scenarios.
  8. Fix the CSRF/XSRF vulnerability.

We will now compare the contents of the firmware updates, and cross reference that to the published changelog. All three updates have a similar structure, as detected by binwalk and our earlier investigation of the firmware:

- 0x000000-0x000200: Firmware header
- 0x000200-0x020200: Bootloader
- 0x020200-0x160200: Compressed kernel
- 0x160200-0x??????: Filesystem

4.6.1 Firmware header

Isolating the firmware headers and comparing them using binwalk yields the differences shown in Figures 4.9, 4.10, and 4.11. Green indicates identical values across all files, red different values across all files, and blue different values across some files.

Cross referencing the differences to public information [38] regarding the firmware header format reveals that the changes at 0x40-0x50 are an MD5 checksum, and that the changes at 0x94-0x98 are some unknown field, perhaps another type of checksum over the header. The field at 0x78-0x7c determines the compressed header length, which has a maximum difference of 397 bytes, indicating only minor changes occurred in kernel. Finally the field at 0x80-0x84 determines the filesystem length, from which we conclude

```
OFFSET      190218-header.bin
--------------------------------------------------------------------------------
0x00000000  03 00 00 00 76 65 72 2E 20 32 2E 30 00 FF FF FF |....ver..2.0....|
0x00000010  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF |................|
0x00000020  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF |................|
0x00000030  FF FF FF FF 08 02 00 04 00 00 00 01 00 00 00 04 |................|
0x00000040  8A 31 25 A5 DC B5 4A E1 94 DC EF 85 80 26 37 F9 |.1%...J......&7.|
0x00000050  00 00 00 00 FF FF FF FF FF FF FF FF FF FF FF FF |................|
0x00000060  FF FF FF FF FF FF FF FF 00 00 00 80 50 C1 00 80 |............P...|
0x00000070  00 7C 02 00 00 02 04 00 00 10 6D 9C 00 14 00 00 |.|........m.....|
0x00000080  00 44 10 00 00 00 00 00 00 01 82 04 55 AA 03 11 |.D..........U...|
0x00000090  A5 00 09 01 83 4A 9A BB FF FF FF FF FF FF FF FF |.....J..........|
0x000000A0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF |................|
0x000000B0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF |................|
0x000000C0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF |................|
0x000000D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
0x000000E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
0x000000F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
0x00000100  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
0x00000110  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
0x00000120  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
0x00000130  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
0x00000140  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
0x00000150  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
0x00000160  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
0x00000170  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
0x00000180  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
0x00000190  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
0x000001A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
0x000001B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
0x000001C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
0x000001D0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF |................|
0x000001E0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF |................|
0x000001F0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF |................|
```

Figure 4.9: 190218 Firmware header hexdump

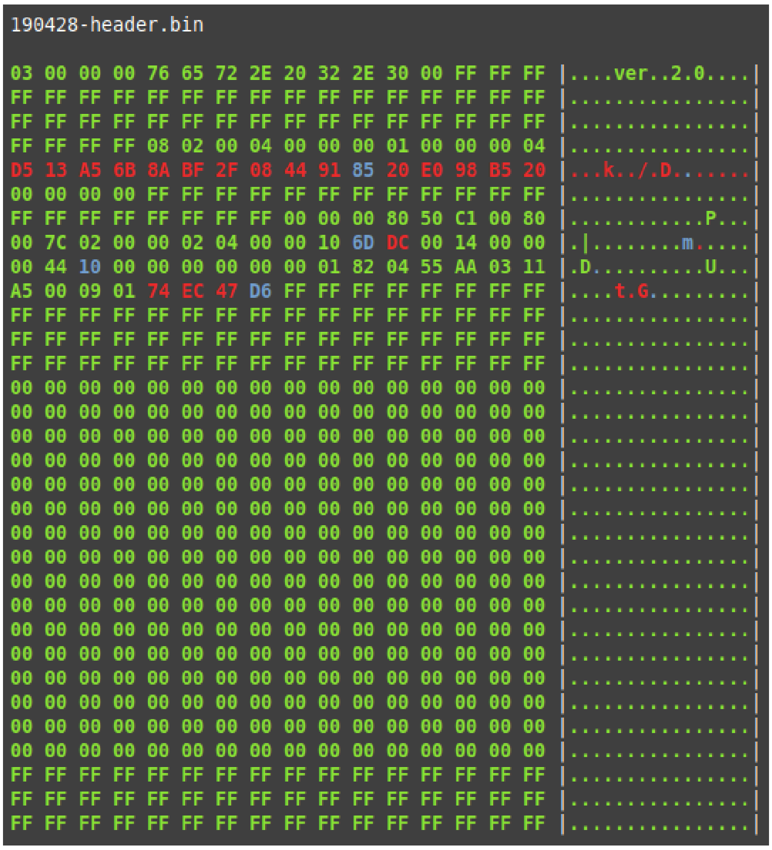
version 200623 caused a slight increase in filesystem length, whereas version 190428 did not.

Interestingly the header contains fields for two 1024 bit signatures [38], but in both the firmware header and the kernel header these fields are zeroed out, indicating the firmware is not signed.

4.6.2 Bootloader

Using the same binwalk compare method to compare the bootloader sections we see only minor changes in offsets 0x13e80-0x13ea0, 0x14610-0x14630, and 0x15f40-0x14f50. Closer inspection reveals these changes are only of strings, as shown below:

- Version 190218
    1. `U-Boot 1.1.3 (Feb 18 2019 - 09:53:38)`
    2. `Date:Feb 18 2019` and `Time:09:53:38`
    3. `I_MTK-1169`
- Version 190428



```
190428-header.bin

03 00 00 00 76 65 72 2E 20 32 2E 30 00 FF FF FF |....ver..2.0....|
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF |................|
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF |................|
FF FF FF FF 08 02 00 04 00 00 00 01 00 00 00 04 |................|
D5 13 A5 6B 8A BF 2F 08 44 91 85 20 E0 98 B5 20 |...k../.D.......|
00 00 00 00 FF FF FF FF FF FF FF FF FF FF FF FF |................|
FF FF FF FF FF FF FF FF 00 00 00 80 50 C1 00 80 |............P...|
00 7C 02 00 00 02 04 00 00 10 6D DC 00 14 00 00 |.|........m.....|
00 44 10 00 00 00 00 00 00 01 82 04 55 AA 03 11 |.D..........U...|
A5 00 09 01 74 EC 47 D6 FF FF FF FF FF FF FF FF |....t.G.........|
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF |................|
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF |................|
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF |................|
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF |................|
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF |................|
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF |................|
```

Figure 4.10: 190428 Firmware header hexdump

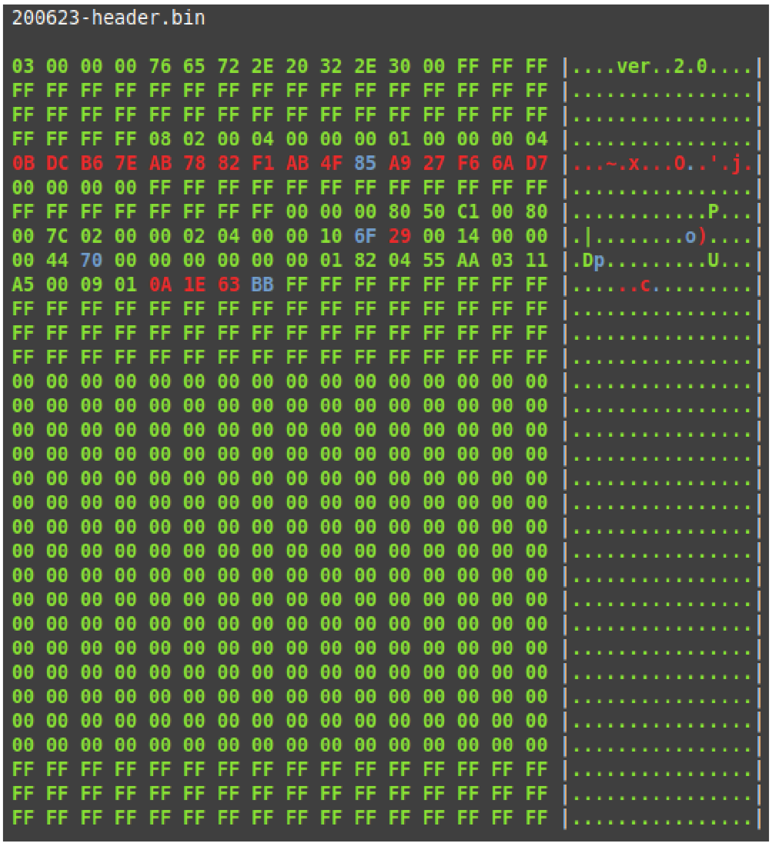
1. `U-Boot 1.1.3 (Apr 28 2019 - 17:31:18)`
2. `Date:Apr 28 2019` and `Time:17:31:18`
3. `I_MTK-1201`

- Version 200623
  1. `U-Boot 1.1.3 (Jun 23 2020 - 17:33:43)`
  2. `Date:Jun 23 2020` and `Time:17:33:43`
  3. `I_MTK-1440`

Presumably these time and date strings were automatically updated during the build process, as the bootloader otherwise remains unchanged.

4.6.3 Kernel

Comparing the uncompressed kernels of 190218 to 190428 reveals that again a date string has changed from `Mon Feb 19 09:55:55 CST 2019` to `Sun`



```
200623-header.bin

03 00 00 00 76 65 72 2E 20 32 2E 30 00 FF FF FF |....ver..2.0....|
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF |................|
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF |................|
FF FF FF FF 08 02 00 04 00 00 00 01 00 00 00 04 |................|
0B DC B6 7E AB 78 82 F1 AB 4F 85 A9 27 F6 6A D7 |...~.x...O..'.j.|
00 00 00 00 FF FF FF FF FF FF FF FF FF FF FF FF |................|
FF FF FF FF FF FF FF FF 00 00 00 80 50 C1 00 80 |............P...|
00 7C 02 00 00 02 04 00 00 10 6F 29 00 14 00 00 |.|........o)....|
00 44 70 00 00 00 00 00 00 01 82 04 55 AA 03 11 |.Dp.........U...|
A5 00 09 01 0A 1E 63 BB FF FF FF FF FF FF FF FF |......c.........|
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF |................|
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF |................|
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF |................|
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF |................|
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF |................|
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF |................|
```

Figure 4.11: 200623 Firmware header hexdump

`Apr 28 17:33:31 CST 2019`. The only other changes are three occurrences of `6A102E` changing to `C572F0`, possibly also some kind of timestamp, but inconsequential change.

Comparing 190428 to 200623 reveals a substantial amount of changes, which is likely caused by the inclusion of new data that unsynchronized the diff tool, causing it to report more changes than actually present. Comparing the strings found in both kernels reveals that besides the expected date string changes, the newer kernel also includes some new strings:

```
tplink/internet_status
tcp_min_snd_mss
TCPWqueueTooBig
```

Both are still 2.6.36 kernels, but perhaps TP-Link has made some modifications, updated modules/drivers, or compiled using different settings. The extend of these changes are unknown, but presumably relatively minor, given how the compressed kernels did not have a large variation in size.

4.6.4 Filesystem

To compare the filesystem contents, we first extract all filesystems using binwalk, and then write a Python3 script, shown in Appendix A.8, to automate the comparison. This script not only detects added and removed files, but also detects changes to files by comparing SHA256 hashes.

The output from comparing 190218 to 190428 is shown below:

```
Comparing version 190218 to 190428
- Old version has 471 files
- New version has 471 files
- Files added: 0
- Files removed: 0
- Files modified: 23
  - /bin/busybox
  - /etc/reduced_data_model.xml
  - /lib/libcmm.so
  - /lib/libcrypto.so.0.9.8
  - /lib/modules/kmdir/kernel/drivers/net/wireless/mt_wifi_ap/
    mt_wifi.ko
  - /usr/bin/cli
  - /usr/bin/cos
  - /usr/bin/dropbearmulti
  - /usr/bin/httpd
  - /usr/bin/tdpd
  - /usr/bin/tmpd
  - /usr/bin/traceroute
  - /web/frame/login.htm
  - /web/help/ChangeLoginPwdHelpRpm.htm
  - /web/help/ManageCtrl_h.htm
  - /web/help/QsChangeLoginPwdHelpRpm.htm
  - /web/js/help.js
  - /web/js/lib.js
  - /web/js/oid_str.js
  - /web/js/str.js
  - /web/main/manageCtrl.htm
  - /web/main/password.htm
  - /web/main/qspassword.htm
- Files kept: 448
```

We see some changes to various binaries, and some changes related to the web interface, as expected from the changelog reporting increased username and password length limits. Comparing `busybox` reveals only a partial version string change of `2019-02-18 09:58:11` to `2019-04-28 17:35:48`. A similar story applies to `libcrypto.so.0.9.8`, `traceroute`, and `mt_wifi.jk` In `reduced_data_model.xml` we only see minor changes, namely 14 blocks of 8 bytes. These changes occurring in blocks is explained by DES crypto in ECB mode, as will be shown when investigating the config files.

In `libcmm.so`, `cli`, `cos`, `httpd`, `tdpd`, and `tmpd` we see only minor changes, indicating likely small modifications made to the source code. Quick manual inspection of function FUN_00403f3c of cli revealed some of these changes are minor tweaks, likely related to the mentioned stability improvements:

```
cstr_strncpy(g_cliUsrAccCfg,0x4279b8,0x10); -> cstr_strncpy(
    g_cliUsrAccCfg,0x4279b8,0x21);
cstr_strncpy(0x427bc8,0x4279c8,0x10);       -> cstr_strncpy(0
    x427bd9,0x4279d9,0x21);
cstr_strncpy(0x427bd8,0x4279d8,0x10);       -> cstr_strncpy(0
    x427bfa,0x4279fa,0x21);
cstr_strncpy(0x427be8,0x4279e8,0x10);       -> cstr_strncpy(0
    x427c1b,0x427a1b,0x21);
cstr_strncpy(0x427bf8,0x4279f8,0x10);       -> cstr_strncpy(0
    x427c3c,0x427a3c,0x21);
cstr_strncpy(0x427c08,0x427a08,0x10);       -> cstr_strncpy(0
    x427c5d,0x427a5d,0x21);
cstr_strncpy(0x427c18,0x427a18,0x40);       -> cstr_strncpy(0
    x427c7e,0x427a7e,0x40);
```

It is possible other changes, especially those to libcmm.so and httpd are security improvements, though it could also be related to the increase of password and username size. A tool such as BinDiff [57] could prove valuable, but for the sake of scope reduction this is left as future research. In dropbearmulti we see more changes, but this could again be due to the diff tool desynchronizing. Comparing the version string stills lists both as version 2012.55.

Inspecting the changes to the web interface reveals that the old password and username limit of 15 characters has indeed been increased to 32. Notably an additional check is now performed on the password to ensure it only contains ASCII characters, using a function defined by /web/js/lib.js:

```
checkAscii: function(str, unalert) {
        for (var i = 0, l = str.length; i < l; i++)
                if (str.charCodeAt(i) < 33 || str.charCodeAt(i)
                    > 126) return $.alert(90201, unalert);
        return 0;
},
```

The output from comparing 190428 to 200623 is shown below:

```
Comparing version 190428 to 200623
- Old version has 471 files
- New version has 478 files
- Files added: 7
  - /web/help/cwmp_h.htm
  - /web/img/login/icons.png
  - /web/img/login/info.png
  - /web/img/login/ok.png
  - /web/img/login/unview.png
```

```
    – /web/img/login/view.png
    – /web/img/login/wrong.png
– Files removed: 0
– Files modified: 48
    – /bin/busybox
    – /etc/reduced_data_model.xml
    – /lib/libcmm.so
    – /lib/libcrypto.so.0.9.8
    – /lib/libos.so
    – /lib/libupnp.so
    – /lib/modules/kmdir/kernel/drivers/net/raeth/raeth.ko
    – /lib/modules/kmdir/kernel/drivers/net/wireless/mt_wifi_ap/
        mt_wifi.ko
    – /usr/bin/ated_tp
    – /usr/bin/cli
    – /usr/bin/cos
    – /usr/bin/dhcpc
    – /usr/bin/dhcpd
    – /usr/bin/dnsProxy
    – /usr/bin/dropbearmulti
    – /usr/bin/ebtables
    – /usr/bin/httpd
    – /usr/bin/ntpc
    – /usr/bin/tddp
    – /usr/bin/tdpd
    – /usr/bin/tmpd
    – /usr/bin/traceroute
    – /usr/bin/wlNetlinkTool
    – /usr/sbin/dhcp6c
    – /usr/sbin/pppd
    – /web/frame/login.htm
    – /web/frame/menu.htm
    – /web/help/NetworkCfgHelpRpm_AP.htm
    – /web/help/RestoreDefaultCfgHelpRpm.htm
    – /web/js/help.js
    – /web/js/language.js
    – /web/js/lib.js
    – /web/js/oid_str.js
    – /web/js/vlancfg.js
    – /web/main/backNRestore.htm
    – /web/main/cwmp.htm
    – /web/main/lanEditAP.htm
    – /web/main/manageCtrl.htm
    – /web/main/password.htm
    – /web/main/portTrigger.htm
    – /web/main/qsWl5G.htm
    – /web/main/qspassword.htm
    – /web/main/status.htm
    – /web/main/wlAcl.htm
    – /web/main/wlAcl5G.htm
    – /web/main/wlAclMssid.htm
    – /web/main/yandexDns.htm
    – /web/qr.htm
– Files kept: 423
```

After investigation we again see the changes to `busybox`, `traceroute`, and `libcrypto.so.0.9.8` are only due to the bump of a version string. In `raeth.ko` only a single byte is changed, and `wlNetlinkTool` only seems to bump numbers 5105/5134 to 5106/5135. In `ntpc` we observe some minor changes that seem to suggest a change in NTP server IPs. The rest of the non-web files contain more extensive modifications, which again are left as future research.

Looking at the web interface changes, some of the changes are just cosmetic changes to the login page. We do also see some fundamental changes however that relate to security. One of these changes is that when visiting the login page for the first time, the user is asked to create a new password, rather than keeping the default password, as controlled by `/web/frame/login.htm`:

```
if (isFirstLogin === "1") {
                createPwd();
        } else {
                PCSubWin();
}
```

The `createPwd` function is shown below:

```
function createPwd() {
        if (isFirstLogin === "0" || $.hasClass($.id("createBtn")
            , "disabled")) {
                return;
        }
        var re = /[^\x00-\x19\x21-\xff]/;
        var password = $.id("change-pcPassword").value;
        var confirmPassword = $.id("confirm-pcPassword").value;
        var userName = "admin";

        if (re.test(password)) {
                $.alert(ERR_USER_PWD_HAS_SPACE);
                return false;
        }
        if (password === "") {
                $.alert(ERR_USER_PWD_EMPTY);
                return false;
        }
        if ($.asc(password, true)) {
                $.alert(ERR_USER_PWD_ASCII);
                return false;
        }

        if (password.length > 32) {
                alert(localString[lang].CHANGE_PWD_TITLE);
                return false;
        }

        if (password !== confirmPassword) {
```

```
                $.alert(ERR_USER_NAME_PWD_CONFLICT);
                return false;
        }

        if(!checkSpace(password) || !checkLength(password) || !
            checkChar(password))
        {
                return false;
        }

        $.act(ACT_CGI, "/cgi/auth", null, null, {
                name: userName,
                oldPwd: "admin",
                pwd: password
        });

        if (!$.exe(null, null, 0)) {
                doLogin(userName, password);
        }
}
```

The new password has to meet certain requirements, such as not containing spaces or non-ASCII characters, and have a length between 6 and 32 characters. Notably it must also meet certain complexity requirements, which in this case means contain at least characters from two different characters classes, where classes are letters, digits, and special characters. A lot of the checks seem duplicated, likely due to incremental development adding new features and requirements. The username is also forced to `admin`, rather than being chosen by the user. Some of the helper functions are shown below:

```
function checkSpace(value) {
        var re = /[^\x00-\x19\x21-\xff]/;
        if (re.test(value)) return false;
        return true;
}

function checkLength(value) {
        if (value.length > 5 && value.length < 33) return true;
        return false;
}

function checkChar(value) {
        var patternNum = /[0-9]/g;
        var patternLetter = /[A-Za-z]/g;
        var patternSign = /[\`\~\!\@\#\$\%\^\&\*\(\)\-\=\_
            \+\[\]\{\}\;\:\'\"\\\|\/\?\.\,\<\>\x20]/g;
␣␣␣␣␣␣␣␣var␣level␣=␣0;

␣␣␣␣␣␣␣␣if␣(patternNum.test(value))␣level++;
␣␣␣␣␣␣␣␣if␣(patternLetter.test(value))␣level++;
␣␣␣␣␣␣␣␣if␣(patternSign.test(value))␣level++;
```

```
␣␣␣␣␣␣␣␣return␣level␣>␣1;
}
```

Other changes of note include the addition of HTML string encoding, as performed by the following function in `/web/js/lib.js`:

```
htmlEncodeStr: function(str) {
        if(str ==undefined || str=="")   return str;
        return str.replace(/[<>&"]/g,function(c){return {'<':'&
            lt;','>':'&gt;','&':'&amp;','"':'&quot;'}[c];});
},
```

Here we see the `<>&`¨ characters are replaced by their HTML encoded equivalent. This function is used in the following files:

- `/web/main/wlAcl.htm`
- `/web/main/wlAcl5G.htm`
- `/web/main/wlAclMssid.htm`
- `/web/main/yandexDns.htm`

The usage pattern for this function is the same across all these files. Instead of `cell.innerHTML` being assigned with `description`, it is now assigned with `htmlEncodeStr(description)`, serving as a rudimentary input sanitation. Note that the function only partially HTML encodes input, as characters `'` and `/` are not replaced.

4.7 Decrypting config files

Going back to the presumably encrypted config files, we conduct some searches on the internet to see if other TP-Link devices have similar constructions. Based on several results, this appears to be the case [23, 24]. We will now attempt to mimic this path to key recovery for this device.

The first step is to check if `libcmm.so` contains string references to the XML files, which appears to be the case:

```
$ strings lib/libcmm.so | grep -F ".xml"
/etc/default_config.xml
/etc/reduced_data_model.xml
/var/tmp/wsc_upnp/WFAWLANConfigSCPD.xml
/var/tmp/wsc_upnp_5G/WFAWLANConfigSCPD_5G.xml
/var/tmp/wsc_upnp/WFADeviceDesc.xml
/var/tmp/wsc_upnp_5G/WFADeviceDesc_5G.xml
<SCPDURL>WFAWLANConfigSCPD.xml</SCPDURL>
```

Next we load `libcmm.so` into Ghidra, a reverse engineering tool published by the NSA [54]. After the analysis has finished, we see a function called `dm_decryptFile` exists in the export symbols. Decompilation of this function into pseudo-C yields:

```
int dm_decryptFile(uint param_1,undefined4 param_2,uint param_3,
    int param_4) {
  int iVar1;
  undefined auStack40 [8];
  int local_20;

  memcpy(auStack40,&DAT_000c9a60,8);
  if (param_3 < param_1) {
    cdbg_printf(8,"dm_decryptFile",0xbbb,
                "Buffer exceeded, decrypt buf size is %u, but dm
                    file size is %u",param_3,param_1);
    local_20 = 0;
  } else {
    local_20 = cen_desMinDo(param_2,param_1,param_4,param_3,
        auStack40,0);
    iVar1 = local_20;
    if (local_20 == 0) {
      cdbg_printf(8,"dm_decryptFile",0xbc2,"DES decrypt error\n"
          );
    } else {
      do {
        local_20 = iVar1;
        if (((undefined *)(param_4 + local_20))[-1] != '\0')
          break;
        iVar1 = local_20 + -1;
      } while (local_20 != 0);
      *(undefined *)(param_4 + local_20) = 0;
    }
  }
  return local_20;
}
```

The function is clearly using DES as expected, and we see 8 bytes being copied into a local buffer `auStack40`, which originate from `DAT_000c9a60`. This symbol resides at address 0x000c9a60 in the `.data` section of the ELF binary, and could possibly be our decryption key. Inspecting the contents reveals the data is identical to the decryption key found for the TP-Link Archer C20 [24], as shown in Figure 4.12.
At this point we assume the config is encrypted with DES in ECB mode, using key `478DA50FF9E3D2CB`. To confirm this we attempt to decrypt `/etc/default_config.xml` using CyberChef [34] and the recipe shown below[5]:

```
From_Hexdump()
```

[5] A hexdump of the file is used as input

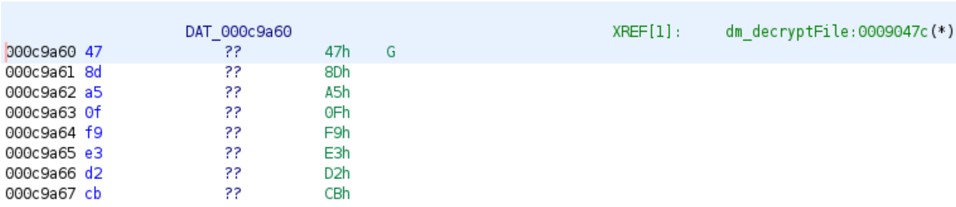


Figure 4.12: Config decryption key

```
DES_Decrypt({'option':'Hex','string':'478DA50FF9E3D2CB '},{'
    option':'Hex','string':''},'ECB','Raw','Raw')
```

This successfully decrypts the config file, as well as `/etc/reduced_data_model.xml`. In the decrypted default config we find several entries for password fields, one of which is populated with a plain text password:

```
<StorageService>
  <UserAccount instance=1 >
    <Enable val=1 />
    <Username val=admin />
    <Password val=admin />
    <X_TP_Reference val=0 />
    <X_TP_SupperUser val=1 />
  </UserAccount>
  <UserAccount instance=2 >
    <Enable val=0 />
    <Username val="" />
    <Password val="" />
    <X_TP_Reference val=0 />
    <X_TP_SupperUser val=0 />
  </UserAccount>
```

Attempting to decrypt the config mtd section from the firmware flashdump does yield successful decryption after cutting off the trailing zeroes, but the output appears gibberish, as reflected by the entropy shown in Figure 4.13. Decompressing using wel-known compression algorithms such as deflate did not recover meaningful output. This likely means the section is encrypted with a different key, or possibly even a different scheme altogether.
Assuming `libcmm.so` is responsible for encrypting and decrypting this config section, we resume our analysis of it in the hopes of finding the code responsible. After looking at the defined strings, we note "Write user config to flash error." at address 0x000c216c. This string is cross-referenced by the `rsl_sys_restoreCfg` function, shown below:

```
undefined4 rsl_sys_restoreCfg(void *param_1,int param_2) {
  bool bVar1;
  int iVar2;
```

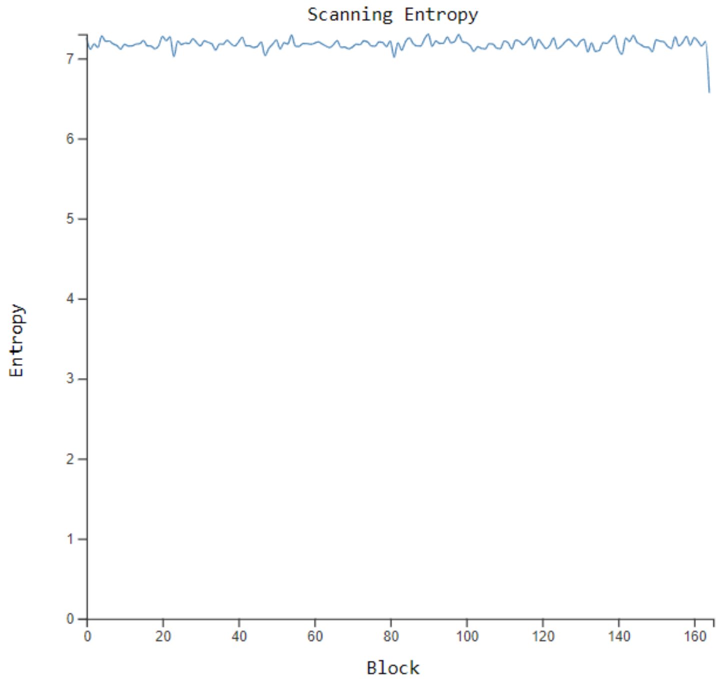


Figure 4.13: Entropy of decrypted config section

```
  uint uVar3;
  int iVar4;
  uint __n;
  uint uVar5;
  int *piVar6;
  undefined4 uVar7;
  int *piVar8;
  char *pcVar9;
  void *pvVar10;
  undefined4 uVar11;
  uint uVar12;

  iVar2 = cmem_getSharedBuffSize();
  pvVar10 = (void *)((int)param_1 + param_2);
  memset(pvVar10,0,iVar2 - param_2);
  uVar3 = cen_desMinDo(param_1,param_2,pvVar10,iVar2 - param_2,&
      DAT_000ef2c0,0);
  if (uVar3 == 0) {
    uVar7 = 0x7e8;
    pcVar9 = "DES decrypt error\n";
LAB_0002c76c:
    uVar11 = 1;
    cdbg_printf(8,"rsl_sys_restoreCfg",uVar7,pcVar9);
  } else {
    if (uVar3 < 0x11) {
      cdbg_printf(8,"rsl_sys_restoreCfg",0x7fc,"File size %d is 
```

```
            too small\n",uVar3);
      } else {
        uVar3 = uVar3 - 0x10;
        uVar12 = iVar2 - 0x41830;
        if (uVar12 < uVar3) {
          uVar7 = 0x806;
LAB_0002c874:
          cdbg_printf(8,"rsl_sys_restoreCfg",uVar7,
                      "Compress data is too long, available size 
                          is %d bytes, now is %d bytes",uVar12,
                      uVar3);
          return 0x1194;
        }
        iVar4 = cen_md5VerifyDigest(pvVar10,(void *)((int)pvVar10
            + 0x10),uVar3);
        if (iVar4 == 0) {
          uVar7 = 0x80d;
          pcVar9 = "Config file MD5 check fail\n";
        } else {
          memcpy(param_1,(void *)((int)pvVar10 + 0x10),uVar3);
          pvVar10 = (void *)((int)param_1 + uVar3);
          memset(pvVar10,0,iVar2 - uVar3);
          __n = cen_uncompressBuff(param_1,pvVar10,iVar2 - uVar3);
          if (__n != 0) {
            uVar5 = dm_restoreCfg(pvVar10,__n,1);
            if (uVar5 == 0) {
              dm_cleanupCfg();
              uVar5 = dm_restoreCfg(pvVar10,__n,0);
              if (uVar5 == 0) {
                if (__n < 0xfff1) {
                  memset(param_1,0,uVar3);
                  *(uint *)((int)pvVar10 + -0x10) = __n;
                  *(undefined4 *)((int)pvVar10 + -0xc) = 0
                      x98765432;
                  *(undefined4 *)((int)pvVar10 + -4) = 0;
                  *(undefined4 *)((int)pvVar10 + -8) = 0;
                  *(undefined *)((int)pvVar10 + __n) = 0;
LAB_0002c908:
                  piVar6 = (int *)((int)pvVar10 + -0x10);
                  iVar2 = 0;
                  uVar3 = 0;
                  piVar8 = piVar6;
                  while (bVar1 = uVar3 != *piVar6 + 0x10U >> 2,
                      uVar3 = uVar3 + 1, bVar1) {
                    iVar4 = *piVar8;
                    piVar8 = piVar8 + 1;
                    iVar2 = iVar2 + iVar4;
                  }
                  *(int *)((int)pvVar10 + -8) = -iVar2;
                  iVar2 = oal_sys_writeCfgFlash(piVar6,*piVar6 + 0
                      x10U);
                  if (iVar2 == 0) {
                    return 0;
                  }
```

```c
                cdbg_printf(8,"rsl_sys_restoreCfg",0x8bb,"Write 
                    user config to flash error.");
                return 0;
              }
              memcpy(param_1,pvVar10,__n);
              pvVar10 = (void *)((int)param_1 + __n);
              memset(pvVar10,0,iVar2 - __n);
              uVar3 = cen_compressBuff(param_1,__n,(int)param_1
                  + iVar2 + -0x8000,pvVar10);
              if (uVar3 != 0) {
                if (uVar12 < uVar3) {
                  uVar7 = 0x887;
                  goto LAB_0002c874;
                }
                memset(param_1,0,__n);
                *(undefined4 *)((int)pvVar10 + -0xc) = 0
                    x98765432;
                *(uint *)((int)pvVar10 + -0x10) = uVar3;
                *(undefined4 *)((int)pvVar10 + -4) = 1;
                *(undefined4 *)((int)pvVar10 + -8) = 0;
                *(undefined *)((int)pvVar10 + uVar3) = 0;
                goto LAB_0002c908;
              }
              uVar7 = 0x87f;
              pcVar9 = "compress data error!";
              goto LAB_0002c76c;
            }
            uVar7 = 0x85e;
          } else {
            uVar7 = 0x855;
          }
          pcVar9 = "Set config into DM error\n";
          goto LAB_0002c76c;
        }
        uVar7 = 0x82c;
        pcVar9 = "uncompress data error!\n";
      }
      cdbg_printf(8,"rsl_sys_restoreCfg",uVar7,pcVar9);
    }
    uVar11 = 0x1195;
  }
  return uVar11;
}
```

We see a similar DES decryption code, but this time using a key at address 0x000ef2c0. Inspecting this address reveals the encryption key is `478DA50BF9E3D2CF`, as shown in Figure 4.14. The rest of the function then uses the first 16 bytes of the decrypted `pvVar10` buffer as an MD5 checksum for the remainder of the buffer. If the checksum is correct, the remainder of the buffer is decompressed, and written to flash using `oal_sys_writeCfgFlash`. Attempting to decrypt the config flash memory section using this new key did not work. After inspecting `oal_sys_writeCfgFlash`, shown below, we

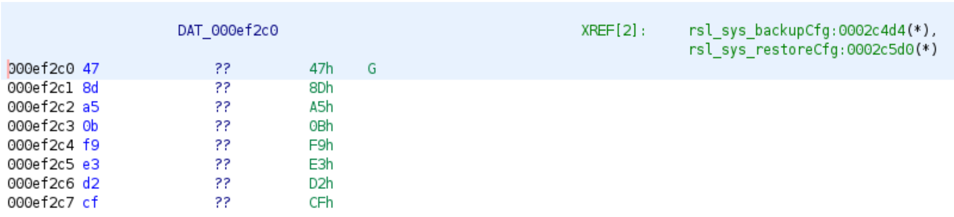


Figure 4.14: Exported config decryption key

see why:

```
undefined4 oal_sys_writeCfgFlash(int *param_1,int param_2) {
  int iVar1;
  undefined4 uVar2;
  uint uVar3;
  int local_60;
  undefined auStack92 [36];
  undefined auStack56 [44];

  local_60 = 0;
  memset(auStack56,0,0x21);
  memset(auStack92,0,0x21);
  memcpy(auStack56,gKey,0x20);
  memcpy(auStack92,gIv,0x20);
  local_60 = param_2 + -0x10;
  iVar1 = aes_cbc_encrypt_intface_bypart(param_1 + 4,&local_60,
      auStack56,auStack92);
  if (iVar1 < 0) {
    cdbg_printf(8,"oal_sys_writeCfgFlash",0x6a0,"Encrypt flash
        error %d",iVar1);
  }
  param_1[3] = param_1[3] + 2;
  *param_1 = local_60;
  uVar3 = local_60 + 0x10;
  if (uVar3 < 0x10001) {
    iVar1 = FUN_00097d0c(0x7c0000,uVar3,param_1);
    uVar2 = 0;
    if (iVar1 < 0) {
      cdbg_printf(8,"oal_sys_writeCfgFlash",0x6b0,"Write config
          error\n");
      uVar2 = 0x232a;
    }
  } else {
    cdbg_printf(8,"oal_sys_writeCfgFlash",0x6aa,"Config file
        length too long - %d\n",uVar3);
    uVar2 = 0x1194;
  }
  return uVar2;
}
```

Before writing the config to flash, it is encrypted using AES in CBC mode.

This likely means the config section we obtained from the flashdump is encrypted with this scheme. Thankfully we see two symbols, `gKey` and `gIv`, being copied into local buffers `auStack56` and `auStack92` respectively. Their contents are ASCII strings `1528632946736109` and `1528632946736539`, as shown in Figure 4.15.

Figure 4.15: Flash memory config encryption parameters

Using these parameters in a new CyberChef recipe, shown below, does yield successful decryption[6].

```
From_Hexdump()
AES_Decrypt({'option':'UTF8','string':'1528632946736109'},{'
    option':'UTF8','string':'1528632946736539'},'CBC','Raw','Raw
    ',{'option':'Hex','string':''},{'option':'Hex','string':''})
```

This decrypted config contains several pieces of sensitive information, such as the WLAN configuration, including SSID and the 8 digit password (Pre-SharedKey) in plaintext, as shown below:

```
<WLANConfiguration instance=1 >
    <__apLastStatus val=3 />
    <Enable val=1 />
    <Name val=wlan0 />
    <Channel val=10 />
    <AutoChannelEnable val=1 />
    <X_TP_PreSSID val=TP-Link />
    <SSID val=TP-Link_FC66 />
    <BeaconType val=11i />
    <X_TP_MACAddressControlRule val=deny />
    <X_TP_Band val=2.4GHz />
    <X_TP_Bandwidth val=Auto />
    <Standard val=n />
    <WEPKeyIndex val=1 />
    <BasicEncryptionModes val=None />
    <BasicAuthenticationMode val=None />
```

[6] There are some leading and trailing garbage bytes in the output, but the XML is clearly readable

```
<WPAEncryptionModes val=TKIPandAESEncryption />
<WPAAuthenticationMode val=PSKAuthentication />
<IEEE11iEncryptionModes val=AESEncryption />
<IEEE11iAuthenticationMode val=PSKAuthentication />
<X_TP_PreSharedKey val=15265258 />
<SSIDAdvertisementEnabled val=1 />
<TransmitPower val=100 />
<RegulatoryDomain val="DE " />
<DeviceOperationMode val=InfrastructureAccessPoint />
<WMMEnable val=1 />
<X_TP_ShortGIEnable val=1 />
<WPS>
    <Enable val=1 />
    <DevicePassword val=15265258 />
    <ConfigurationState val=Configured />
</WPS>
```

Presumably the `rsl_sys_restoreCfg` function is responsible for importing an XML configuration that has been exported via the web interface, which apparently uses a different (but hardcoded) encryption key compared to the other XML config files. In order to test this we use the web interface to export a config, and attempt to decrypt it with this new DES key. This does appear successful, and the output is partly readable XML, as shown in Figure 4.16, but as expected we need an additional decompression step to recover the proper XML.

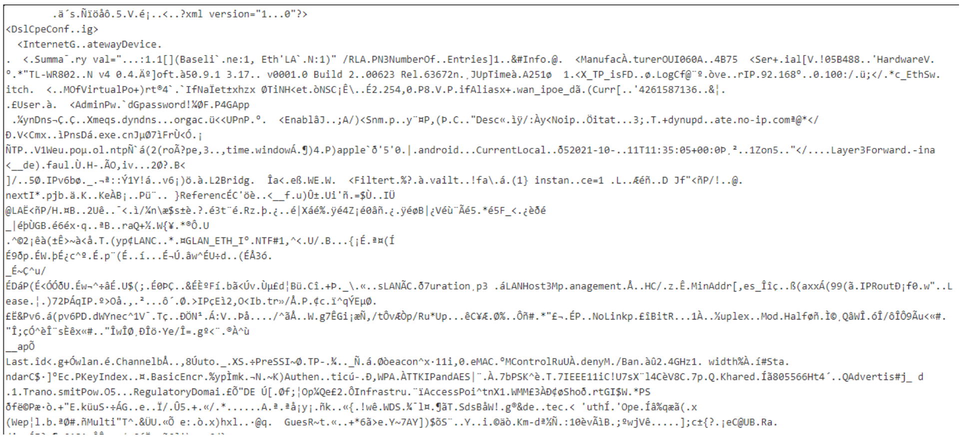


Figure 4.16: Decrypted, but compressed, exported config

Initial trial and error skipping the first 16 bytes, and using raw inflate does not decompress correctly. As such we investigate `cen_uncompressBuff` to see how the decompression works. This function is exported by `/lib/libcutil.so`, so we load that binary into Ghidra next. The decompilation of the function is shown below:

```
uint cen_uncompressBuff(void *src, undefined *dest, uint len) {
```

```
bool bVar1;
byte bVar2;
uint uVar3;
int iVar4;
int iVar5;
byte *pbVar6;
char *__s;
uint uVar7;
uint local_38;
byte *local_34;
byte *local_30;
undefined4 local_28;

local_38 = 0;
if ((src == (void *)0x0) || (dest == (undefined *)0x0)) {
  __s = "Invalid params!";
} else {
  memcpy(&local_38,src,4);
  uVar3 = local_38;
  if (local_38 <= len) {
    uVar7 = 0;
    if (local_38 != 0) {
      local_34 = (byte *)((int)src + 5);
      local_28 = 0;
      *dest = *(undefined *)((int)src + 4);
      local_30 = dest + 1;
      uVar7 = 1;
      while (uVar7 < uVar3) {
        iVar4 = FUN_00019540(&local_34);
        if (iVar4 == 0) {
          uVar7 = uVar7 + 1;
          bVar2 = *local_34;
          local_34 = local_34 + 1;
          *local_30 = bVar2;
          local_30 = local_30 + 1;
        } else {
          iVar5 = FUN_0001959c(&local_34);
          iVar4 = FUN_0001959c(&local_34);
          bVar2 = *local_34;
          local_34 = local_34 + 1;
          pbVar6 = local_30 + -(bVar2 + 1 + (iVar4 + -2) * 0
              x100);
          iVar4 = iVar5 + 2;
          while (bVar1 = 0 < iVar4, iVar4 = iVar4 + -1, bVar1)
              {
            bVar2 = *pbVar6;
            pbVar6 = pbVar6 + 1;
            *local_30 = bVar2;
            local_30 = local_30 + 1;
          }
          uVar7 = uVar7 + iVar5 + 2;
        }
      }
    }
```

```
      if (uVar7 == local_38) {
        return uVar7;
      }
      printf("Length is not match depackLen = %d\tsrcDataLen = %
          d",uVar7);
      return 0;
    }
    __s = "Invalid file or file is too long!";
  }
  puts(__s);
  return 0;
}
```

Outside of calls to FUN_00019540 and FUN_0001959c (both small functions) there do not seem to be any non-standard dependencies, or even global data references. In FUN_00019540 we have no external calls or data references, and in FUN_0001959c we only have calls to FUN_00019540, again no data references. This means the entire decompression algorithm is fairly small, and as such we wrote a simple C decompression tool based on the decompiled pseudo C, shown in Appendix A.7.

Running this decompression tool on the decrypted config successfully recovers the readable XML. Again this config contains sensitive information, even including the password used to authenticate to the web interface[7]:

```
<X_TP_UserCfg>
    <AdminPwd val=password! />
</X_TP_UserCfg>
```

We conclude for the answer to **RQ 8** that not only admin credentials are easily recovered from `/etc/passwd.bak`, but that configuration files containing potentially sensitive information (such as credentials) are encrypted with a weak DES/ECB encryption scheme, using a hardcoded key that is shared across various TP-Link products. Several hardcoded encryption keys and parameters are used, but all can be extracted from the firmware, thus offering little protection.

4.8 Generation of sensitive information

In this research section we direct our focus on the generation of sensitive information, such as credentials. Any weakness here could allow attackers to guess or derive this information, thus compromising security. We therefore investigate how the initial device parameters, WPS key generation in the web interface, and dropbear initialization work.

[7] Prior configs did not include this key because no password was configured yet

4.8.1 Initial device parameters

We do not know how the default WiFi pin, which is printed on a label attached to the case, is generated. If a weak scheme is used that does not have sufficient entropy, such as mainly using the SSID, MAC, or serial number, it could be possible to predict the WiFi pin codes, as observed in earlier products [36].

Looking at one of our own devices, we observe the following data:

- SSID: `TP-Link_B488`
- MAC: `60-A4-B7-05-B4-88`
- PIN: `80556640`
- Serial: `22150C3004751`

Immediately we spot the pattern where the SSID is constructed as `TP-Link_XXYY`, where `XX` and `YY` are the last two bytes of the MAC address. Unlike earlier products however, the PIN (password) is not a segment of the MAC. No immediate relation between the PIN code and other data could be inferred, and looking for cross references to the string `TP-Link` in binaries did not reveal any code responsible for the generation of these default pin codes.

4.8.2 WPS Web Interface

The WPS subsection of the Wireless section in the web interface offers two buttons: one to restore the PIN, and one to generate a new PIN. These buttons are defined in `/web/main/wlQss.htm`, shown below:

```
function getNewPIN(obj) {
        $.addLoading(obj);
        $.act(ACT_OP, ACT_OP_WLAN_GET_NEW_PIN, wpsObj.__stack);
        $.exe(function(ret){if (!ret) $.reload();});
}

function restorePIN(obj) {
        $.addLoading(obj);
        $.act(ACT_OP, ACT_OP_WLAN_RESTORE_PIN, wpsObj.__stack);
        $.exe(function(ret){if (!ret) $.reload();});
}

<input type="button" class="button_XL_T" value="Restore PIN"
    onclick="restorePIN(this)" id="restore"/>
<input type="button" class="button_XL_T" value="Generate New PIN
    " onclick="getNewPIN(this)" id="genNew"/>
```

Due to the Javascript abstractions it is not immediately clear how these requests work. Analyzing traffic with Burp Suite [55] reveals requests go through an encrypted channel via the `/cgi_gdpr` endpoint, as shown in Figure 4.17.

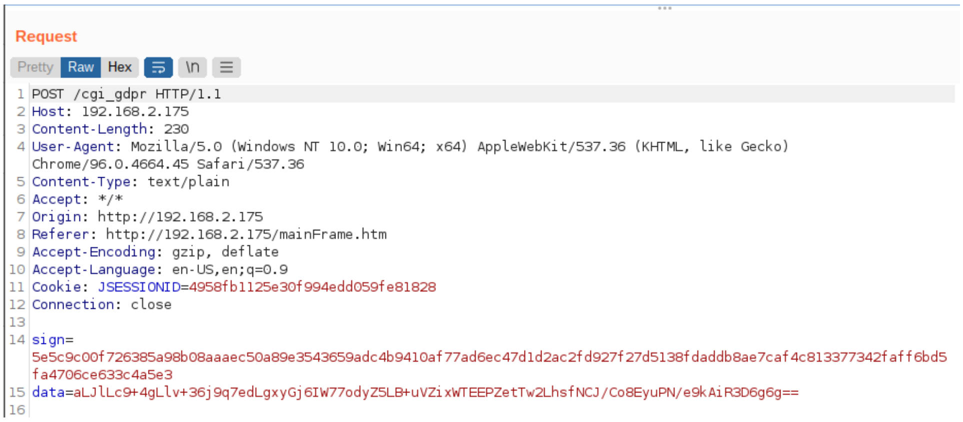


Figure 4.17: Capture of new PIN request

This GDPR encrypted channel has been investigated for other products before [24], and appears to work identically here. Opening up the developer tools of the browser used, and executing `$.encrypt.encryptManager.encryptor.aes` in the console allows us to recover the AES key and IV used to encrypt the request, as shown in Figure 4.18.

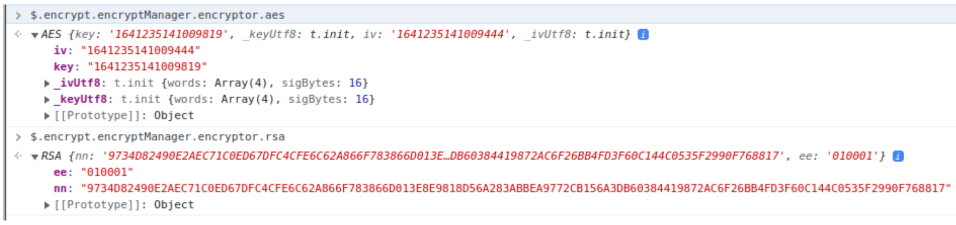


Figure 4.18: Recovery of AES key and IV, as well as RSA public parameters

Using an online AES decryption service[8] in 128-bit/CBC mode allows us to decrypt the base64 encoded data string from the request in Figure 4.17, using the AES key and IV shown in Figure 4.18. The decrypted request data is shown below:

```
7[ACT_WLAN_GET_NEW_PIN#1,1,0,0,0,0#0,0,0,0,0,0]0,0
```

Further analysis of `/usr/bin/httpd` reveals `http_cgi_gdpr_main` is the handler for the `/cgi_gdpr` endpoint. After some expected AES decryption and RSA signature verification code, we see a switch statement with 9 cases (1-8,

[8] `https://www.devglan.com/online-tools/aes-encryption-decryption`

and a default case). Given that `/web/js/lib.js` defines 9 `ACT_X` cases, as shown below, there is a good chance these are directly related.

```
var ACT_GET = 1;
var ACT_SET = 2;
var ACT_ADD = 3;
var ACT_DEL = 4;
var ACT_GL = 5;
var ACT_GS = 6;
var ACT_OP = 7;
var ACT_CGI = 8;
var ACT_SIG = 9;
```

The earlier Javascript code made calls using `ACT_OP`, which has value 7, and shows up at the beginning of the decrypted request data. We thus assume that case 7 is the handler for this particular act type, shown below:

```
case 7:
  uVar12 = rdp_action(acStack12376,&local_30bc);
  if (0 < (int)uVar12) goto LAB_00411c60;
  iVar11 = strcmp(acStack12376,"ACT_REBOOT");
  uVar6 = uVar12;
  if (iVar11 == 0) {
    param_1[7] = 0;
  }
  break;
```

A call to `rdp_action` is made that seems to do most of the heavy lifting, which is defined in `/lib/libcmm.so`, and shown below:

```
int rdp_action(void *param_1,undefined4 param_2) {
  int iVar1;
  size_t sVar2;
  int iVar3;
  char *__s;
  int iVar4;
  undefined1 *puVar5;

  iVar1 = dm_acquireLock("rdp_action");
  if (iVar1 == 0) {
    puVar5 = l_actStringTable;
    iVar1 = 1;
    do {
      puVar5 = (undefined1 *)((int)puVar5 + 4);
      __s = *(char **)puVar5;
      sVar2 = strlen(__s);
      iVar3 = memcmp(param_1,__s,sVar2 + 1);
      iVar4 = iVar1 + 1;
      if (iVar3 == 0) {
        iVar3 = (**(code **)(g_rsl_actFuncTable + iVar1 * 4))(
            param_2);
```

```
          if (iVar3 != 0) {
            cdbg_perror("rdp_action",0xed,iVar3);
            dm_unLock();
            return iVar3;
          }
          if (iVar1 - 8U < 2) {
            rsl_saveCfg();
          }
          dm_unLock();
          return 0;
        }
        iVar1 = iVar4;
      } while (iVar4 != 0x1a);
      cdbg_printf(8,"rdp_action",0xe6,"Wrong action %s\n",param_1)
          ;
      iVar1 = 1;
      dm_unLock();
    }
    else {
      cdbg_printf(8,"rdp_action",0xde,"Can\'t get lock, return %d
          .\n",iVar1);
    }
    return iVar1;
}
```

The code appears to be iterating over a table called `l_actStringTable`, and calling functions through function pointers defined in `g_rsl_actFuncTable`. Inspecting `l_actStringTable`, shown in Figure 4.19, appears to indicate it is a table of pointers.

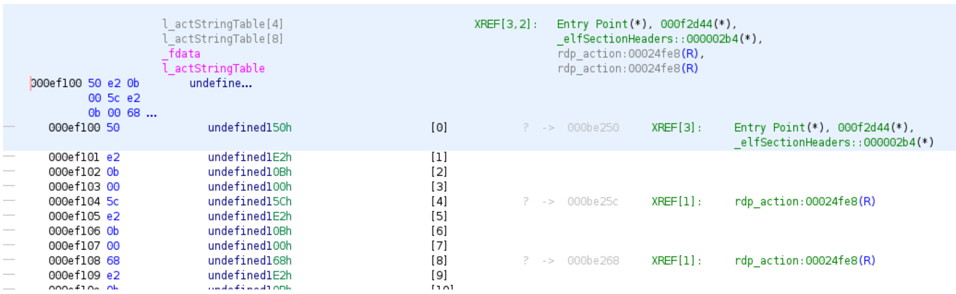


Figure 4.19: Raw string table

Inspecting the data at these pointer location reveals familiar strings, shown in Figure 4.20.

We see the string `ACT_WLAN_GET_NEW_PIN` that appeared in our decrypted request, as well as `ACT_WLAN_RESTORE_PIN`, presumably used when clicking the restore PIN button. Inspecting `g_rsl_actFuncTable` reveals it is also a table of pointers, similar to the act string table. We assume the code computes the index of the act string supplied by the request in the act string table, and uses this index to call into the act function table. Manually

000be250	ACT_NULL	"ACT_NULL"	ds
000be25c	ACT_REBOOT	"ACT_REBOOT"	ds
000be268	ACT_FACTORY_RESET	"ACT_FACTORY_RESET"	ds
000be27c	ACT_ERASE_RADIO_FLASH	"ACT_ERASE_RADIO_FLASH"	ds
000be294	ACT_DHCP_RENEW	"ACT_DHCP_RENEW"	ds
000be2a4	ACT_DHCP_RELEASE	"ACT_DHCP_RELEASE"	ds
000be2b8	ACT_PPP_CONN	"ACT_PPP_CONN"	ds
000be2c8	ACT_PPP_DISCONN	"ACT_PPP_DISCONN"	ds
000be2d8	ACT_WLAN_GET_NEW_PIN	"ACT_WLAN_GET_NEW_PIN"	ds
000be2f0	ACT_WLAN_RESTORE_PIN	"ACT_WLAN_RESTORE_PIN"	ds
000be308	ACT_WLAN_UPDATE_ASSOC	"ACT_WLAN_UPDATE_ASSOC"	ds
000be320	ACT_WLAN_WPS_PBC	"ACT_WLAN_WPS_PBC"	ds
000be334	ACT_WLAN_WPS_PIN	"ACT_WLAN_WPS_PIN"	ds
000be348	ACT_WLAN_SCAN	"ACT_WLAN_SCAN"	ds
000be358	ACT_NTP_REQUEST	"ACT_NTP_REQUEST"	ds
000be368	ACT_L2TP_CONN	"ACT_L2TP_CONN"	ds
000be378	ACT_L2TP_DISCONN	"ACT_L2TP_DISCONN"	ds
000be38c	ACT_PPTP_CONN	"ACT_PPTP_CONN"	ds
000be39c	ACT_PPTP_DISCONN	"ACT_PPTP_DISCONN"	ds
000be3b0	ACT_WAN_TYPE_DETECT	"ACT_WAN_TYPE_DETECT"	ds
000be3c4	ACT_BPA_CONN	"ACT_BPA_CONN"	ds
000be3d4	ACT_BPA_DISCONN	"ACT_BPA_DISCONN"	ds
000be3e4	ACT_OP_IPPING	"ACT_OP_IPPING"	ds
000be3f4	ACT_OP_TRACERT	"ACT_OP_TRACERT"	ds
000be404	ACT_IPV6_DISCONN	"ACT_IPV6_DISCONN"	ds
000be418	ACT_IPV6_CONN	"ACT_IPV6_CONN"	ds
000be428	ACT_END	"ACT_END"	ds

Figure 4.20: Corresponding string table strings

cross-referencing the indices for the pin generation and restore entries leads us to functions `rsl_wlan_getNewPIN` and `rsl_wlan_restorePIN`, proving our assumption correct.

We begin with analyzing `rsl_wlan_restorePIN`, shown below:

```
uint rsl_wlan_restorePIN(ushort *param_1) {
  undefined4 uVar1;
  ushort auStack104 [18];
  uint auStack68 [13];
  uint local_10;

  memset(auStack104,0,0x54);
  local_10 = rsl_getObj(0x31,param_1,0x54,auStack104);
  uVar1 = 0x1acd;
  if (local_10 == 0) {
    local_10 = FUN_00059fa0(auStack68);
    if (local_10 == 0) {
      local_10 = rsl_setObj(0x31,param_1,auStack104,2);
      uVar1 = 0x1ad9;
      if (local_10 == 0) {
        return 0;
      }
    } else {
      uVar1 = 0x1ad3;
```

```
    }
  }
  cdbg_perror("rsl_wlan_restorePIN",uVar1,local_10);
  return local_10;
}
```

A call to `rsl_getObj` obtains the current config node from the data model, which after modification is written back using `rsl_setObj`. It is not immediately obvious, but `FUN_00059fa0` performs the modification, and is shown below:

```
int FUN_00059fa0(uint *param_1) {
  bool bVar1;
  int iVar2;
  undefined3 extraout_var;
  ushort local_2d8;
  undefined2 local_2d6;
  undefined2 local_2d4;
  undefined2 local_2d2;
  undefined2 local_2d0;
  undefined2 local_2ce;
  undefined2 local_2cc;
  undefined auStack712 [20];
  uint local_2b4;

  local_2d8 = 0;
  local_2d6 = 0;
  local_2d4 = 0;
  local_2d2 = 0;
  local_2d0 = 0;
  local_2ce = 0;
  local_2cc = 0;
  memset(auStack712,0,0x2b4);
  iVar2 = dm_getObj(6,&local_2d8,0x2b4,auStack712);
  if (iVar2 == 0) {
    bVar1 = wlan_checkPIN(local_2b4);
    if ((CONCAT31(extraout_var,bVar1) == 0) || (local_2b4 == 0))
        {
      local_2b4 = 0xbc6146;
    }
    *param_1 = local_2b4;
  } else {
    cdbg_perror("wlan_getRestoredPIN",0x13ef,iVar2);
  }
  return iVar2;
}
```

Here we see an additional call to `dm_getObj` to obtain the PIN, which is subsequently checked. If the pin is not valid[9], or zero, the pin is replaced with 0xbc6146 (12345670).

[9]Based on the WPS PIN checksum

Ghidra fails to properly decompile the code in these past two functions, obscuring some of the data movements behind `extraout` variables, and not correctly identifying that `auStack104` and `auStack68` are part of the same data structure/buffer. Nevertheless we managed to infer the hidden data flows, and conclude that resetting the PIN resets it to the value as obtained from the data model. Noteworthy is the hardcoded fallback to 12345670 as the PIN code in case of invalid data.

Inspecting `rsl_wlan_getNewPIN`, shown below, reveals a similar structure where first `rsl_getObj` is called to retrieve the relevant config code from the data model, which is then modified and subsequently written back using `rsl_setObj`.

```
uint rsl_wlan_getNewPIN(ushort *param_1) {
  int iVar1;
  undefined4 uVar2;
  char *pcVar3;
  uint uVar4;
  ushort auStack5096 [18];
  undefined4 local_13c4;
  undefined auStack5012 [3920];
  int local_444;

  memset(auStack5012,0,0x1380);
  iVar1 = FUN_000590f4(param_1,auStack5012);
  if (iVar1 == 0) {
    uVar2 = 0x1a99;
    if (local_444 != 0) {
      uVar2 = (**(code **)(local_444 + 0x78))(8);
      memset(auStack5096,0,0x54);
      uVar4 = rsl_getObj(0x31,param_1,0x54,auStack5096);
      if (uVar4 == 0) {
        local_13c4 = uVar2;
        uVar4 = rsl_setObj(0x31,param_1,auStack5096,2);
        if (uVar4 == 0) goto LAB_0005ad10;
        uVar2 = 0x1aad;
      } else {
        uVar2 = 0x1aa6;
      }
      cdbg_perror("rsl_wlan_getNewPIN",uVar2,uVar4);
      goto LAB_0005ad10;
    }
    pcVar3 = "Failed to get Oal Funcs\n";
  } else {
    uVar2 = 0x1a92;
    pcVar3 = "Failed to get Wlan Cfg\n";
  }
  uVar4 = 1;
  cdbg_printf(8,"rsl_wlan_getNewPIN",uVar2,pcVar3);
LAB_0005ad10:
  FUN_00057910((int)auStack5012);
```

```
  return uVar4;
}
```

Ghidra again seems to have issues accurately decompiling the code, but the assignment of `uVar2` to `local_13c4` immediately before the `rsl_setObj` call is presumably what updates the PIN code. This `uVar2` variable is assigned from `(**(code **)(local_444 + 0x78))(8);`. Sadly Ghidra does not detect what value `local_444` has prior to this computed call. It is most likely set as a side effect from calling `FUN_000590f4`, but manual inspection did not reveal any obvious static value. As such we are unable to progress, and have hit the limit of static analysis. Dynamic analysis through emulation of the device, or running a gdb debug server on the device could reveal the computed call location. Because neither of those methods are trivial to set up, we abandon this avenue for the sake of scope reduction.

While we do not know how the function called is implemented, we do observe it is called with a single 8 parameter. It is unlikely to be a coincidence that the WPS PIN to be generated consists out of 8 digits, but without the implementation we cannot say for sure whether the scheme is secure. At the very least we did not find proof that information such as MAC addresses or SSIDs are used during the generation of a new random PIN, though it should be said that absence of proof is not proof of absense.

4.8.3 Dropbear

Cross referencing the dropbear strings found in the boot log to binaries reveals matches in `/lib/libcmm.so`. Specifically we see `FUN_0009b3b8` is the main source of these boot log entries:

```
void FUN_0009b3b8(void) {
  int iVar1;
  char *pcVar2;
  ushort local_e8;
  undefined2 local_e6;
  undefined2 local_e4;
  undefined2 local_e2;
  undefined2 local_e0;
  undefined2 local_de;
  undefined2 local_dc;
  char acStack216 [200];

  local_e8 = 0;
  local_e6 = 0;
  local_e4 = 0;
  local_e2 = 0;
  local_e0 = 0;
  local_de = 0;
  local_dc = 0;
  sleep(5);
```

```
  memset(acStack216,0,200);
  pcVar2 = acStack216;
  iVar1 = dm_getObj(8,&local_e8,200,pcVar2);
  if (iVar1 != 0) {
    pcVar2 = "get OID_USER_CFG error.\n";
    cdbg_printf(8,"prepareDropbear",0x11a);
  }
  FUN_0009b1d0((int)acStack216);
  util_execSystem((int)"prepareDropbear","dropbearkey -t rsa -f 
      %s",
                  "/var/tmp/dropbear/dropbear_rsa_host_key",
                      pcVar2);
  util_execSystem((int)"prepareDropbear","dropbearkey -t dss -f 
      %s",
                  "/var/tmp/dropbear/dropbear_dss_host_key",
                      pcVar2);
  util_execSystem((int)"prepareDropbear","dropbear -p %d -r %s -
      d %s -A %s",0x16,
                  "/var/tmp/dropbear/dropbear_rsa_host_key");
  os_threadExit(0);
  return;
}
```

The `dropbearkey` utility, part of Dropbear, is used to generate the RSA and DSS key pairs. Function `FUN_0009b1d0` is responsible for generating the `dropbearpwd` file:

```
undefined4 FUN_0009b1d0(int param_1) {
  undefined4 uVar1;
  FILE *__stream;
  size_t sVar2;
  byte *pbVar3;
  int iVar4;
  char *__s;
  char acStack112 [36];
  byte local_4c [32];
  undefined local_2c;
  char *local_28;

  memset(local_4c,0,0x21);
  if ((*(char *)(param_1 + 0x44) == '\0') || (*(char *)(param_1
      + 0x65) == '\0')) {
    cdbg_printf(8,"setDropbearLogin",0xe3,"uname = %s, pswd = %s
        \n",param_1 + 0x44,param_1 + 0x65);
    uVar1 = 1;
  } else {
    __stream = fopen("/var/tmp/dropbear/dropbearpwd","wb+");
    uVar1 = 1;
    if (__stream != (FILE *)0x0) {
      fprintf(__stream,"username:%s\n",param_1 + 0x44);
      local_28 = (char *)(param_1 + 0x65);
      __s = acStack112;
      sVar2 = strlen(local_28);
```

```
      iVar4 = 0;
      cen_md5MakeDigest(local_4c,local_28,sVar2);
      memset(acStack112,0,0x21);
      do {
        pbVar3 = local_4c + iVar4;
        iVar4 = iVar4 + 1;
        sprintf(__s,"%02x",(uint)*pbVar3);
        __s = __s + 2;
      } while (iVar4 != 0x10);
      memcpy(local_4c,acStack112,0x21);
      local_2c = 0;
      fprintf(__stream,"password:%s\n",local_4c);
      fclose(__stream);
      uVar1 = 0;
    }
  }
  return uVar1;
}
```

Here we see that the file is opened, and that a `username:USER` line is written, where `USER` is the username. Next a `password:PASS` line is written, where `PASS` is a hex-string of the MD5 checksum of the password. The username and password are based on the parameter `param_1`, which is ultimately obtained from a data model using a call to `dm_getObj`, as we saw before. Plotting the function call graph of `dm_init`, as shown in Figure 4.21, reveals a path to `dm_setObj`, which first loads and then restores a config file, that is apparently XML data. Based on our prior research regarding config files and related `dm_X` functions we know this data model data is thus sourced from the (encrypted XML) configs, which include the plaintext SSH credentials. Further analysis of `dm_loadCfg`, shown below, reveals that the config is loaded from flash memory. Loading from flash memory simply means reading the config mtd section at address 0x7c0000-0x7d0000, and decrypting it using the AES parameters as previously discovered. The `oal_sys_readCfgFlash` function is responsible for this. If this fails, the config is loaded from `/etc/default_config.xml` instead.

```
uint dm_loadCfg(void) {
  bool bVar1;
  char **ppcVar2;
  char *pcVar3;
  int iVar4;
  char *pcVar5;
  uint uVar6;
  char *pcVar7;
  undefined4 uVar8;
  char *pcVar9;
  char **ppcVar10;
  char *local_20 [3];

  local_20[0] = (char *)0x0;
```

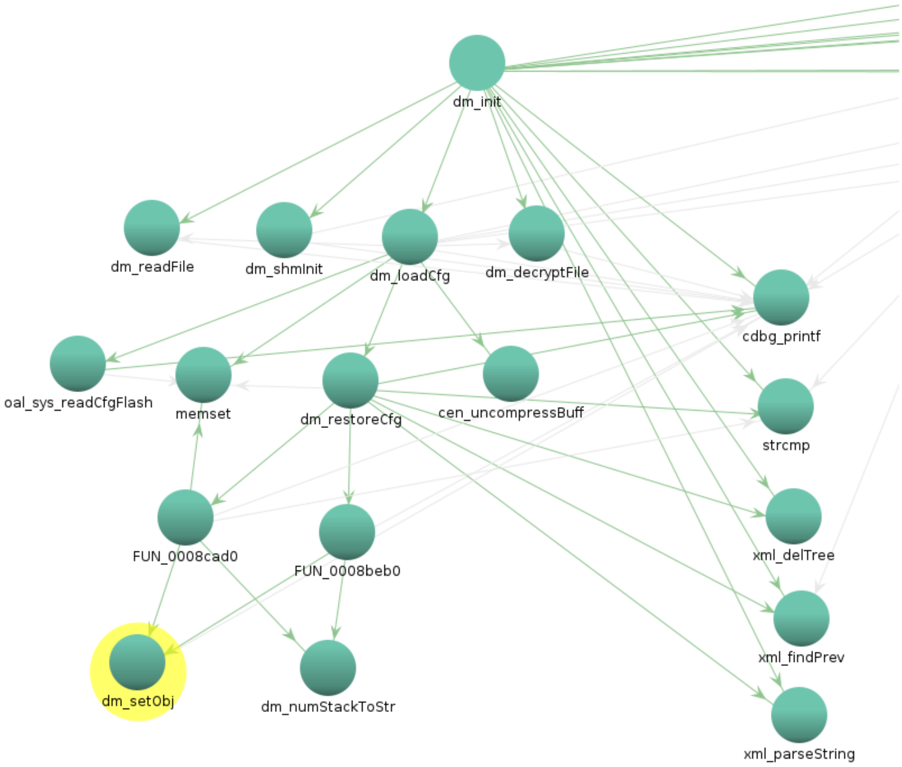


Figure 4.21: Data model call graph

```
ppcVar2 = (char **)cmem_attachSharedBuff();
if (ppcVar2 == (char **)0x0) {
  cdbg_printf(8,"dm_loadCfg",0x921,"attach to big shared 
      buffer failed.");
  return 0x232a;
}
pcVar3 = (char *)cmem_getSharedBuffSize();
local_20[0] = (char *)0x10;
iVar4 = oal_sys_readCfgFlash((uint *)ppcVar2,(uint *)local_20)
    ;
if (iVar4 == 0) {
  pcVar5 = *ppcVar2;
  if (local_20[0] == pcVar5) {
    pcVar7 = ppcVar2[1];
    if (pcVar7 != (char *)0x98765432) {
      uVar8 = 0x93c;
      pcVar9 = "software version is not match, in config, 
          version = %u";
      goto LAB_00090674;
    }
    pcVar7 = (char *)0x0;
    uVar6 = 0;
    ppcVar10 = ppcVar2;
```

```
      while (bVar1 = uVar6 != (uint)(pcVar5 + 0x10) >> 2, uVar6
          = uVar6 + 1, bVar1) {
        pcVar9 = *ppcVar10;
        ppcVar10 = ppcVar10 + 1;
        pcVar7 = pcVar7 + (int)pcVar9;
      }
      if (pcVar7 == (char *)0x0) {
        if (ppcVar2[3] == (char *)0x0) {
          ppcVar10 = ppcVar2 + 4;
        } else {
          if (pcVar3 + -0x41830 < pcVar5) {
            cdbg_printf(8,"dm_loadCfg",0x9e3,
                        "Compress data is too long, available 
                            size is %d bytes, now is %d bytes",
                        pcVar3 + -0x41830,pcVar5);
            cmem_detachSharedBuff(ppcVar2);
            return 0x1194;
          }
          ppcVar10 = (char **)((int)(ppcVar2 + 4) + (int)pcVar5)
              ;
          memset(ppcVar10,0,((int)pcVar3 - (int)pcVar5) - 0x10);
          local_20[0] = (char *)cen_uncompressBuff(ppcVar2 + 4,
              ppcVar10,(int)pcVar3 - (int)pcVar5);
          if (local_20[0] == (char *)0x0) {
            cdbg_printf(8,"dm_loadCfg",0x9f7,"Depack config from
                flash failed!\n");
            cmem_detachSharedBuff(ppcVar2);
            return 0x232a;
          }
          *(char *)((int)ppcVar10 + (int)local_20[0]) = '\0';
        }
        goto LAB_00090894;
      }
      cdbg_printf(8,"dm_loadCfg",0x94d,"checksum is not right");
    } else {
      uVar8 = 0x935;
      pcVar9 = "length is not match, in config, length = %u byte
          , but read %u byte";
      pcVar7 = pcVar5;
LAB_00090674:
      cdbg_printf(8,"dm_loadCfg",uVar8,pcVar9,pcVar7);
    }
    iVar4 = 0x264a;
  } else {
    cdbg_printf(8,"dm_loadCfg",0x954,"Read config from flash 
        failed. ret = %d",iVar4);
  }
  pcVar5 = dm_readFile("/etc/default_config.xml",pcVar3,ppcVar2)
      ;
  if (pcVar5 == (char *)0x0) {
    cdbg_printf(8,"dm_loadCfg",0x9be,"Read default config file 
        failed, ret = %d",iVar4);
    cmem_detachSharedBuff(ppcVar2);
    return 0x2649;
```

```
  }
  ppcVar10 = (char **)((int)ppcVar2 + ((uint)pcVar3 >> 1));
  local_20[0] = (char *)dm_decryptFile((uint)pcVar5,ppcVar2,(
      uint)pcVar3 >> 1,(int)ppcVar10);
  if (local_20[0] == (char *)0x0) {
    return 0x232a;
  }
LAB_00090894:
  uVar6 = dm_restoreCfg(ppcVar10,local_20[0],0);
  cmem_detachSharedBuff(ppcVar2);
  return uVar6;
}
```

4.9 Firmware update process

After logging in to the web interface, the user is able to update the firmware of the device by selecting and uploading a `.bin` file. The client side logic for this is controlled in `/web/main/softup.htm`. A form is used that posts a file parameter `filename` to the `/cgi/softup` endpoint, as shown below:

```
<form action="/cgi/softup" enctype="multipart/form-data" method=
    "post">
<p><b class="item_T" id="t_file">Firmware File Path</b><input
    type="file" name="filename" id="filename" /></p>
</form>
```

A button labelled `Upgrade` is defined, which invokes the javascript function `doSubmit` when clicked, both shown below:

```
<p class="tail"><input type="button" class="button_L_T" id="
    t_upgrade" value="Upgrade" onclick="return doSubmit()" /></p>
```

```
<script language="javascript" type="text/javascript">
function doSubmit()
{
        if($.id("filename").value == "")
        {
                $.alert(ERR_FIRM_FILE_NONE);
                return false;
        }

        var regex = /^.+\.bin$/;
        if (!regex.test($.id("filename").value)){
                //alert("Invalid file type.");
                $.alert(CMM_CFG_FILE_FORMAT_ERR);
                return false;
        }
        var formObj = $.d.forms[0];
        try
```

```
        {
                formObj.target = "up_frame";
                formObj.action = "/cgi/softup";
                formObj.submit();
        }catch(e){}

        $.guage(["<span class='T_T_uploading'>"+s_str.uploading+
            "</span>", "<span class='T_T_wait_upload'>"+s_str.
            wait_upload+"</span>"], 10, 300, function(){
                $.guage(["<span class='T_T_upgrading'>"+s_str.
                    upgrading+"</span>", "<span class='T_
                    T_wait_upgrade'>"+s_str.wait_upgrade+"</span>
                    "], 100, 1800, function(){$.deleteCookie("
                    Authorization");if(INCLUDE_LOGIN_GDPR_ENCRYPT
                    ){try{$.newencryptorManager.cleanStorage()}
                    catch(e){} }window.parent.$.refresh();});
                $.cgi("/cgi/softburn", null, function(ret){
                        if (ret && ret != ERR_NETWORK && ret !=
                            ERR_EXIT && ret != ERR_NONE_FILE) $.
                            errBack(ret, "softup.htm");
                }, false, true);
        });
}
</script>
```

From this we conclude the only client side check being done is that a post must include the filename parameter, which must end with `.bin`.

To better understand the firmware update process, and see if there are any security mechanism in place, we investigate what occurs in the backend when a post request to `/cgi/softup` is received. These requests are presumably processed by `/usr/bin/httpd`, and thus we load that binary into Ghidra. After analysis has completed, we look for string references to `/cgi/softup`, and find one in `http_init_main`:

```
undefined4 http_init_main(void) {
  int iVar1;
  int *piVar2;
  undefined1 *puVar3;

  http_inetd_init();
  http_parser_init();
  http_auth_init();
  http_alias_init();
  http_session_init();
  http_menu_init();
  http_cgi_init();
  http_cgi_gdpr_init();
  http_tool_init();
  for (piVar2 = &g_http_alias_conf_admin; *piVar2 != 0; piVar2 =
       piVar2 + 1) {
    http_alias_addEntryByArg(0,*piVar2,0,0,g_http_author_admin);
```

```
}
puVar3 = g_http_alias_conf_default;
while( true ) {
  if (*(int *)puVar3 == 0) break;
  http_alias_addEntryByArg(0,*(int *)puVar3,0,0,
      g_http_author_default);
  puVar3 = (undefined1 *)((int)puVar3 + 4);
}
http_alias_addEntryByArg(2,"/cgi/conf.bin",0,http_rpm_backup,
    g_http_author_default);
http_alias_addEntryByArg(2,"/cgi/confencode",0,
    http_rpm_confencode,g_http_author_default);
http_alias_addEntryByArg(2,"/cgi/confup",0,http_rpm_restore,
    g_http_author_default);
http_alias_addEntryByArg(2,"/cgi/bnr",0,http_rpm_conferr,
    g_http_author_default);
http_alias_addEntryByArg(2,"/cgi/softup",0,http_rpm_update,
    g_http_author_default);
http_alias_addEntryByArg(2,"/cgi/softburn",0,http_rpm_softerr,
    g_http_author_default);
http_alias_addEntryByArg(2,"/cgi/log",0,http_rpm_log_main,
    g_http_author_default);
http_alias_addEntryByArg(2,"/cgi/info",0,http_rpm_info,
    g_http_author_default);
http_alias_addEntryByArg(2,"/cgi/lanMac",0,http_rpm_lanMac,
    g_http_author_default);
http_alias_addEntryByArg(2,"/cgi/auth",0,http_rpm_auth_main,
    g_http_author_default);
http_alias_addEntryByArg(2,"/cgi/pvc",0,http_rpm_autoPvc,
    g_http_author_default);
http_alias_addEntryByArg(2,"/cgi/ansi",0,http_rpm_ansi,
    g_http_author_default);
http_alias_addEntryByArg(2,"/cgi/logout",0,http_rpm_logout,
    g_http_author_default);
http_alias_addEntryByArg(2,"/cgi/route",0,http_rpm_routeTbl,
    g_http_author_default);
http_alias_addEntryByArg(2,"/cgi/updateWlThroughput",0,
    http_rpm_updateWlThroughput,g_http_author_default);
http_alias_addEntryByArg(2,"/cgi/https",0,
    http_rpm_downloadCACert,g_http_author_default);
http_alias_addEntryByArg(2,"/cgi/getParm",0,http_rpm_getParm,
    g_http_author_default);
http_alias_addEntryByArg(2,"/cgi/login",0,http_rpm_login,
    g_http_author_default);
http_file_init();
iVar1 = http_inetd_setMsgCtl();
if (iVar1 != 0) {
                /* WARNING: Subroutine does not return */
  exit(-1);
}
signal(0xd,(__sighandler_t)0x1);
dm_shmInit(0);
FUN_00403ec4();
http_inetd_main();
```

```
  return 0;
}
```

Here we see that `http_rpm_update` is registered as a handler for the `/cgi/softup` endpoint. This handler function is shown below:

```
int http_rpm_update(int *param_1) {
  bool bVar1;
  int iVar2;
  undefined4 uVar3;
  char *pcVar4;
  size_t sVar5;
  undefined *puVar6;
  uint uVar7;
  int *piVar8;
  undefined4 local_a38;
  undefined auStack2612 [256];
  char acStack2356 [256];
  char acStack2100 [2052];
  int local_30;

  local_a38 = 0;
  DAT_00446db0 = 0;
  if (*(int *)(*param_1 + 0x1038) == 2) {
    DAT_00446db0 = 0x1162b;
    iVar2 = 0x193;
  } else {
    uVar7 = param_1[6];
    iVar2 = cmem_getUpdateBufSize();
    if (iVar2 + 0x40bb8U < uVar7) {
      DAT_00446db0 = 0x1162b;
      iVar2 = 0x19d;
    } else {
      if ((DAT_00446db4 == (undefined *)0x0) &&
         (DAT_00446db4 = (undefined *)
             cmem_updateFirmwareBufAlloc(),
         DAT_00446db4 == (undefined *)0x0)) {
        DAT_00446db0 = 0x232a;
        return 500;
      }
      puVar6 = DAT_00446db4;
      uVar3 = cmem_getUpdateBufSize();
      bVar1 = false;
      while ((param_1[6] != 0 &&
             (iVar2 = http_parser_illMultiObj(param_1,
                 acStack2356,0,puVar6,uVar3,&local_a38),
             -1 < iVar2))) {
        local_30 = iVar2;
        iVar2 = strcmp(acStack2356,"filename");
        if (iVar2 == 0) {
          DAT_00446db8 = local_30;
          bVar1 = true;
          puVar6 = auStack2612;
```

```
        uVar3 = 0x100;
      }
    }
    if (bVar1) {
      param_1[0xe] = (int)FUN_0040b7b0;
      puVar6 = &DAT_00419e68;
    } else {
      DAT_00446db0 = 0x1162b;
      iVar2 = cmem_updateFirmwareBufFree(DAT_00446db4);
      if (iVar2 < 0) {
        cdbg_printf(8,"http_rpm_update",0xa4,"Detach big 
            buffer error\n");
      }
      DAT_00446db4 = (undefined *)0x0;
      puVar6 = &DAT_0041a4c8;
    }
    sprintf(acStack2100,"<html><head></head><body>%s</body></
        html>",puVar6);
    piVar8 = (int *)param_1[0x21];
    param_1[7] = 0;
    param_1[0xf] = (int)g_http_file_pTypeHTML;
    if (((piVar8 == (int *)0x0) || (*piVar8 != 1)) ||
       (pcVar4 = strstr((char *)param_1[3],"/cgi_gdpr"),
           pcVar4 == (char *)0x0)) {
      iVar2 = http_io_output(param_1,acStack2100);
      iVar2 = -(uint)(iVar2 == -1);
    } else {
      pcVar4 = (char *)http_buf_getptr(piVar8[7],0);
      sVar5 = strlen(acStack2100);
      strncpy(pcVar4,acStack2100,sVar5);
      iVar2 = *(int *)(param_1[0x21] + 0x1c);
      sVar5 = strlen(acStack2100);
      *(size_t *)(iVar2 + 0xc) = *(int *)(iVar2 + 0xc) + sVar5
          ;
      sVar5 = strlen(acStack2100);
      http_buf_incrpos(iVar2,sVar5);
      iVar2 = 0;
    }
  }
}
return iVar2;
}
```

This handler mostly deals with allocating a buffer for the firmware update, and processing HTML parameters. Notable is that `FUN_0040b7b0` is used as a function pointer, which is assigned to `param_1]0xe]`, and then passed to `http_io_output`. Decompiling this function yields the code shown below:

```
undefined4 FUN_0040b7b0(void) {
  int iVar1;

  iVar1 = rdp_updateFirmware(DAT_00446db4,DAT_00446db8);
  if (iVar1 == 0) {
```

```
    rdp_action("ACT_REBOOT",0);
    do {
                    /* WARNING: Do nothing block with infinite
                        loop */
    } while( true );
  }
  DAT_00446db0 = iVar1;
  iVar1 = cmem_updateFirmwareBufFree(DAT_00446db4);
  if (iVar1 < 0) {
    cdbg_printf(8,"http_rpm_softburn",0xd0,"Detach big buffer 
        error\n");
  }
  DAT_00446db4 = 0;
  return 0;
}
```

Clearly `rdp_updateFirmware` is a call that starts the updating process, where `DAT_00446db4` is the previously allocated firmware update buffer, and `DAT_00446db8` is the contents of the `.bin` file. The `rdp_updateFirmware` function is not part of `/usr/bin/httpd`, but some quick investigation reveals this function is exported by `/lib/libcmm.so`, which subsequently calls `rsl_sys_updateFirmware` after acquiring a lock, and is shown below:

```
undefined4 rsl_sys_updateFirmware(int param_1,uint param_2) {
  int iVar1;
  uint uVar2;
  int iVar3;
  uint uVar4;
  uint uVar5;
  uint uVar6;
  code *pcVar7;
  uint local_38;
  undefined auStack52 [24];

  local_38 = 0;
  iVar1 = oal_sys_getTagOffset();
  if ((param_2 < 0x30000) || (uVar2 = cmem_getUpdateFirmwareSize
      (), uVar2 < param_2)) {
    cdbg_printf(8,"rsl_sys_updateFirmware",0xc7f,"The file\'s 
        length is bad: %d\n",param_2);
    return 0x1196;
  }
  uVar2 = cmem_getUpdateFirmwareSize();
  if (uVar2 < param_2) {
    return 0x1196;
  }
  iVar1 = param_1 + iVar1;
  memcpy(auStack52,(void *)(iVar1 + 0x40),0x10);
  memcpy((void *)(iVar1 + 0x40),auStack52,0x10);
  uVar2 = *(uint *)(iVar1 + 0xa4);
  uVar2 = uVar2 << 0x18 | uVar2 >> 0x18 | (uVar2 & 0xff0000) >>
      8 | (uVar2 & 0xff00) << 8;
```

```
cdbg_printf(8,"rsl_sys_updateFirmware",0xcc3,"=======0x%08x
    =======",uVar2);
uVar4 = *(uint *)(iVar1 + 0x34);
uVar5 = rsl_dev_getProductId();
if ((uVar4 << 0x18 | uVar4 >> 0x18 | (uVar4 & 0xff0000) >> 8 |
     (uVar4 & 0xff00) << 8) == uVar5) {
  uVar4 = *(uint *)(iVar1 + 0x38);
  uVar5 = rsl_dev_getProductVer();
  if ((uVar4 << 0x18 | uVar4 >> 0x18 | (uVar4 & 0xff0000) >> 8
       | (uVar4 & 0xff00) << 8) == uVar5)
  {
    uVar2 = oal_sys_getAddHverFlash();
    uVar5 = uVar2 << 0x18 | uVar2 >> 0x18 | (uVar2 & 0xff0000)
         >> 8 | (uVar2 & 0xff00) << 8;
    printf("flash hver is %d\n",uVar5);
    uVar4 = *(uint *)(iVar1 + 0x3c);
    printf("image hver is %d\n",
           uVar4 << 0x18 | uVar4 >> 0x18 | (uVar4 & 0xff0000)
             >> 8 | (uVar4 & 0xff00) << 8);
    uVar4 = *(uint *)(iVar1 + 0x3c);
    if ((uVar4 << 0x18 | uVar4 >> 0x18 | (uVar4 & 0xff0000) >>
         8 | (uVar4 & 0xff00) << 8) < uVar5)
    {
      cdbg_printf(8,"rsl_sys_updateFirmware",0xcfd,
                  "Firmware Additional HardwareVersion check 
                     failed\n");
      return 0x119a;
    }
    *(uint *)((int)&__DT_REL[0x3c6].r_info + param_1) = uVar2;
    *(uint *)(iVar1 + 0x3c) = uVar2;
    system("echo 0 > /proc/tplink/led_sys");
    uVar4 = *(uint *)(iVar1 + 0x8c);
    uVar2 = *(uint *)(iVar1 + 0x90);
    rsl_createSwSignature
              (uVar4 << 0x18 | uVar4 >> 0x18 | (uVar4 & 0
                  xff0000) >> 8 | (uVar4 & 0xff00) << 8,
               uVar2 << 0x18 | uVar2 >> 0x18 | (uVar2 & 0
                   xff0000) >> 8 | (uVar2 & 0xff00) << 8,
               &local_38);
    uVar4 = *(uint *)(iVar1 + 0x8c);
    uVar2 = *(uint *)(iVar1 + 0x90);
    uVar6 = uVar4 << 0x18 | uVar4 >> 0x18 | (uVar4 & 0xff0000)
         >> 8 | (uVar4 & 0xff00) << 8;
    uVar4 = uVar2 << 0x18 | uVar2 >> 0x18 | (uVar2 & 0xff0000)
         >> 8 | (uVar2 & 0xff00) << 8;
    uVar2 = local_38;
    cdbg_printf(8,"rsl_sys_updateFirmware",0xd0f,
                "NEW: swRevision-0x%x, platformVer-0x%x, 
                   swSignature-0x%x\n",uVar6,uVar4,local_38);
    uVar5 = rsl_getCurrSwSignature();
    if (uVar5 != local_38) {
      rsl_sys_restoreDefaultCfg();
    }
    uVar5 = *(uint *)(iVar1 + 0x8c);
```

```
      iVar3 = rsl_checkSwVerRollBack
                          (uVar5 << 0x18 | uVar5 >> 0x18 | uVar5
                             >> 8 & 0xff00 | (uVar5 & 0xff00) << 8
                          );
      if (iVar3 != 0) {
        cdbg_printf(8,"rsl_sys_updateFirmware",0xd2c,"
            rsl_checkSwVerRollBack ret err.",uVar6,uVar4,
                    uVar2);
      }
      if ((*(int *)(iVar1 + 0x84) == 0) && (*(int *)(iVar1 + 0
          x88) == 0)) {
        pcVar7 = oal_sys_writeAppFlash;
      } else {
        pcVar7 = oal_sys_writeAppBootFlash;
        param_1 = param_1 + 0x200;
        param_2 = param_2 - 0x200;
      }
      iVar1 = (*pcVar7)(param_1,param_2);
      if (iVar1 == 0) {
        return 0;
      }
      cdbg_printf(8,"rsl_sys_updateFirmware",0xd45,"Update
         firmware error!\n");
      return 1;
    }
  }
  cdbg_printf(8,"rsl_sys_updateFirmware",0xce9,"Firmware version
      check failed\n",uVar2);
  return 0x1197;
}
```

We see various checks that validate the length of the firmware file, as well as some inspection of firmware header fields, such as product id, product version, hardware version, and a software signature field[10]. Also note the two `memcpy` calls at the beginning, which copies out the MD5 checksum header field identified earlier into `auStack52`, but then immediately copies it back without modifications or further references. One possible explanation for this odd behaviour is that the original code base contains verification functionality that has been conditionally disabled during compilation. Finally, either `oal_sys_writeAppFlash` or `oal_sys_writeAppBootFlash` is called, based on the firmware header, to flash the new firmware image to the device. Both these functions call `FUN_00097d0c` to perform the actual flashing, shown below:

```
undefined4 FUN_00097d0c(undefined4 param_1,uint param_2,
    undefined4 param_3) {
  int __fd;
  int iVar1;
```

[10]This is not a cryptographic signature or checksum, but rather an identifier used to determine if the default configuration should be restored

```
  undefined4 uVar2;
  int iVar3;
  undefined4 local_40;
  undefined4 local_3c;
  uint local_38;
  undefined4 local_34;
  undefined4 local_30;
  int local_2c;

  iVar3 = 3;
  do {
    local_40 = 0;
    local_3c = 0;
    local_38 = 0;
    local_34 = 0;
    local_30 = 0;
    local_2c = 0;
    if (param_2 < 0x800001) {
      __fd = open("/dev/flash0",0);
      if (__fd < 0) {
        cdbg_printf(8,"__writeFlash",0x141,"Open flash pseudo 
            device failed\n");
        uVar2 = 0xfffffffe;
      } else {
        local_40 = param_3;
        local_3c = param_1;
        local_38 = param_2;
        iVar1 = ioctl(__fd,2,&local_40);
        if (iVar1 < 0) {
          cdbg_printf(8,"__writeFlash",0x14b,"FLASH_API: ioctl 
              error\n");
          uVar2 = 0xfffffffd;
          close(__fd);
        } else {
          close(__fd);
          if (local_2c != 10) {
            return 0;
          }
          cdbg_printf(8,"__writeFlash",0x154,"OUT_OF_SCOPE");
          uVar2 = 0xfffffffc;
        }
      }
    } else {
      cdbg_printf(8,"__writeFlash",0x13a,"write_flash:Too many 
          bytes - %d > %d bytes\n",param_2,0x800000);
      uVar2 = 0xffffffff;
    }
    cdbg_printf(8,"writeFlash",0x164,"write flash error %d",
        uVar2);
    iVar3 = iVar3 + -1;
    usleep((__useconds_t)"etFwId");
  } while (iVar3 != 0);
  return uVar2;
}
```

Here we see that the `/dev/flash0` device is opened, and written to using an `ioctl` call. The difference between writeAppFlash and writeAppBootFlash is that the former only starts writing at offset 0x20000, the start of the kernel mtd section –thus skipping the boot section–, whereas the latter starts writing at offset 0.

Note that while the firmware header does suggest the possibility for signing [38], no such code was found in the update process. Interestingly a `rdp_verifyFirmware` function does exist, but apart from the name change it behaves identically to `rdp_updateFirmware`, in that it only acquires a lock, and then calls `rsl_sys_updateFirmware`. No other functions or strings were found that indicate any such verification functionality exists, but is unused, or accessed via other code paths.

Now that we have fully traced the firmware update process we conclude that there are only some minor sanity checks. There are no security checks that validate the integrity of firmware updates, nor is there any requirement that firmware updates must be cryptographically signed by the vendor, thus providing our answer to **RQ 10**. This also means it is possible to flash our own firmware, not signed by the vendor, thus providing a positive answer to **RQ 3**.

4.10 Dynamic analysis

For the final part of our investigation we perform some dynamic analysis of the device through the root shell obtained over the UART interface. We begin by printing a list of running processes:

```
# ps
  PID USER       VSZ STAT COMMAND
    1 admin     1068 S    init
    2 admin        0 SW   [kthreadd]
    3 admin        0 SW   [ksoftirqd/0]
    4 admin        0 SW   [kworker/0:0]
    5 admin        0 SW   [kworker/u:0]
    6 admin        0 SW<  [khelper]
    7 admin        0 SW   [sync_supers]
    8 admin        0 SW   [bdi-default]
    9 admin        0 SW<  [kblockd]
   10 admin        0 SW   [kswapd0]
   11 admin        0 SW<  [crypto]
   19 admin        0 SW   [mtdblock0]
   20 admin        0 SW   [mtdblock1]
   21 admin        0 SW   [mtdblock2]
   22 admin        0 SW   [mtdblock3]
   23 admin        0 SW   [mtdblock4]
   24 admin        0 SW   [mtdblock5]
   25 admin        0 SW   [mtdblock6]
```

```
  26 admin           0 SW   [kworker/u:1]
  33 admin           0 SW   [kworker/0:1]
  81 admin        3464 S    cos
  82 admin        1068 S    /bin/sh
 171 admin        2648 S    igmpd -n
 174 admin        2664 S    mldProxy -n
 175 admin        3464 S    cos
 176 admin        3464 S    cos
 177 admin        3464 S    cos
 180 admin        2612 S    ntpc
 183 admin        2620 S    dyndns /var/tmp/dconf/dyndns.conf
 186 admin        2620 S    noipdns /var/tmp/dconf/noipdns.conf
 189 admin        2620 S    cmxdns /var/tmp/dconf/cmxdns.conf
 233 admin           0 SW   [RtmpCmdQTask]
 234 admin           0 SW   [RtmpWscTask]
 235 admin           0 SW   [RtmpMlmeTask]
 248 admin        1244 S    wlNetlinkTool
 251 admin        1244 S    wlNetlinkTool
 252 admin        1244 S    wlNetlinkTool
 254 admin        1064 S    wscd -i ra0 -m 1 -w /var/tmp/wsc_upnp/
 282 admin        5052 S <  httpd
 293 admin        2612 S    dnsProxy
 296 admin        1072 S <  dhcpd /var/tmp/dconf/udhcpd.conf
 301 admin        3068 S    snmpd -f /var/tmp/dconf/snmpd.conf
 304 admin        3228 S    tmpd
 307 admin        3088 S    tdpd
 316 admin         988 S    dhcpc
 330 admin        2612 S    diagTool
 364 admin        2636 S    pwdog
 366 admin        3136 S    tddp
 378 admin        2636 S    pwdog
 381 admin        2636 S    pwdog
 397 admin        2608 S    afcd
 412 admin        1136 S    dropbear -p 22 -r /var/tmp/dropbear/
     dropbear_rsa_hos
1364 admin        1060 R    ps
```

We can see that all processes run as admin, even potentially sensitive application such as the web server. The principle of least privilege is not being applied, opting instead to let everything run with all permissions. This means any exploitation of the web server immediately means admin privilege on the device; there is no need for subsequent privilege escalation vulnerabilities.

Using Checksec [56] on `httpd` reveals that many exploit mitigation measures such as RELRO, stack canaries, NX, PIE, and RPATH are not used. This is reflected by reading from `/proc/<httpd-pid>/maps`:

```
00400000-00422000 r-xp 00000000 1f:02 665          /usr/bin/httpd
00431000-00432000 rw-p 00021000 1f:02 665          /usr/bin/httpd
00432000-00456000 rwxp 00000000 00:00 0            [heap]
```

```
2b21a000-2b220000 r-xp 00000000 1f:02 53          /lib/ld-uClibc
    -0.9.33.2.so
2b220000-2b222000 rw-p 00000000 00:00 0
2b22f000-2b230000 r--p 00005000 1f:02 53          /lib/ld-uClibc
    -0.9.33.2.so
2b230000-2b231000 rw-p 00006000 1f:02 53          /lib/ld-uClibc
    -0.9.33.2.so
2b231000-2b23d000 r-xp 00000000 1f:02 99          /lib/libcutil.
    so
2b23d000-2b24c000 ---p 00000000 00:00 0
2b24c000-2b24d000 rw-p 0000b000 1f:02 99          /lib/libcutil.
    so
2b24d000-2b254000 r-xp 00000000 1f:02 51          /lib/libos.so
2b254000-2b263000 ---p 00000000 00:00 0
2b263000-2b264000 rw-p 00006000 1f:02 51          /lib/libos.so
2b264000-2b333000 r-xp 00000000 1f:02 57          /lib/libcmm.so
2b333000-2b342000 ---p 00000000 00:00 0
2b342000-2b348000 rw-p 000ce000 1f:02 57          /lib/libcmm.so
2b348000-2b361000 rw-p 00000000 00:00 0
2b361000-2b365000 r-xp 00000000 1f:02 46          /lib/libxml.so
2b365000-2b374000 ---p 00000000 00:00 0
2b374000-2b375000 rw-p 00003000 1f:02 46          /lib/libxml.so
2b375000-2b381000 r-xp 00000000 1f:02 97          /lib/libpthread
    -0.9.33.2.so
2b381000-2b390000 ---p 00000000 00:00 0
2b390000-2b391000 r--p 0000b000 1f:02 97          /lib/libpthread
    -0.9.33.2.so
2b391000-2b396000 rw-p 0000c000 1f:02 97          /lib/libpthread
    -0.9.33.2.so
2b396000-2b398000 rw-p 00000000 00:00 0
2b398000-2b399000 r-xp 00000000 1f:02 94          /lib/librt
    -0.9.33.2.so
2b399000-2b3a8000 ---p 00000000 00:00 0
2b3a8000-2b3a9000 rw-p 00000000 1f:02 94          /lib/librt
    -0.9.33.2.so
2b3a9000-2b3fd000 r-xp 00000000 1f:02 55          /lib/libgdpr.so
2b3fd000-2b40d000 ---p 00000000 00:00 0
2b40d000-2b418000 rw-p 00054000 1f:02 55          /lib/libgdpr.so
2b418000-2b53f000 r-xp 00000000 1f:02 47          /lib/libcrypto.
    so.0.9.8
2b53f000-2b54e000 ---p 00000000 00:00 0
2b54e000-2b564000 rw-p 00126000 1f:02 47          /lib/libcrypto.
    so.0.9.8
2b564000-2b566000 rw-p 00000000 00:00 0
2b566000-2b5ac000 r-xp 00000000 1f:02 60          /lib/libssl.so
    .0.9.8
2b5ac000-2b5bb000 ---p 00000000 00:00 0
2b5bb000-2b5bf000 rw-p 00045000 1f:02 60          /lib/libssl.so
    .0.9.8
2b5bf000-2b61f000 r-xp 00000000 1f:02 48          /lib/libuClibc
    -0.9.33.2.so
2b61f000-2b62e000 ---p 00000000 00:00 0
2b62e000-2b62f000 r--p 0005f000 1f:02 48          /lib/libuClibc
    -0.9.33.2.so
```

```
2b62f000-2b630000 rw-p 00060000 1f:02 48         /lib/libuClibc
    -0.9.33.2.so
2b630000-2b635000 rw-p 00000000 00:00 0
2b635000-2b637000 r-xp 00000000 1f:02 88         /lib/libdl
    -0.9.33.2.so
2b637000-2b646000 ---p 00000000 00:00 0
2b646000-2b647000 r--p 00001000 1f:02 88         /lib/libdl
    -0.9.33.2.so
2b647000-2b648000 rw-p 00002000 1f:02 88         /lib/libdl
    -0.9.33.2.so
58800000-58864000 rw-s 00000000 00:04 32769      /SYSV000004d2 (
    deleted)
7ffb5000-7ffd6000 rwxp 00000000 00:00 0          [stack]
7fff7000-7fff8000 r-xp 00000000 00:00 0          [vdso]
```

We see that both the stack and heap have read, write, and execute permission bits set, whereas modern security practise would recommend marking executable sections non-writable. Reading from `/proc/sys/kernel/randomize_va_space` yields 1, rather than 2, meaning ASLR is only partially enabled.

4.11 Findings

In this section we summarize our findings of the conducted research. We begin with providing summarized answers to our research questions, and then provide an overview of vulnerabilities found, along with a short discussion on their perceived impact.

4.11.1 Summary to research questions

RQ 1) A UART interface is exposed, that outputs debug information, and provides an unauthenticated root shell.

RQ 2) Firmware can be extracted from the flash chip, and a very similar version is also available on the vendor website.

RQ 3) It is possible to flash unsigned firmware onto the device through the web interface, and firmware updates provided by the vendor are not signed.

RQ 4) The device, in default configuration, runs DNS, UPnP, DHCP, SSH, and HTTP services.

RQ 5) We have identified which files have been modified, some of which are due to security updates, but detailed analysis is considered out of scope.

RQ 6) Several prior vulnerabilities have been fixed, but some still remain unpatched.

RQ 7) See Appendix A.5.

RQ 8) We are able to extract default credentials, as well as hard coded (fallback) crypographic keys and parameters.

RQ 9) We have investigated how several credentials, and other pieces of sensitive information, are generated, but were unable to confirm how WiFi WPS pins are generated.

RQ 10) The firmware update process contains no real security checks, outside basic sanity checks to prevent flashing firmware intended for different hardware, based on fields in the firmware header.

4.11.2 Overview of vulnerabilities

See Table 4.1 for an overview of vulnerabilities found. Note that some entries represent multiple closely related vulnerabilities, making the absolute number of vulnerabilities higher than indicated by the **VULNxx** notation.

- **VULN01** has high impact, because it bypasses any security mechanisms available if an attacker has physical access to the device.
- **VULN02** has moderate impact, as it is susceptable to bruteforce attacks, but a stronger password can be configured to mitigate the issue.
- **VULN03** and **VULN04** have low impact, because these credentials are already widely known, and a user is asked to configure a new password at first login.
- **VULN05**, **VULN06**, **VULN08**, **VULN09**, **VULN10**, and **VULN11** have high impact, because these vulnerabilities can lead to Denial-Of-Service attacks, or even Remote-Code-Execution.
- **VULN07** has moderate impact, as an attacker already needs admin credentials to restore the config, but could be used to trick users into restoring malicious configs that set up backdoors.
- **VULN12** has low impact, as it is unclear if this leads to any exploitable vulnerabilities.
- **VULN13**, **VULN14**, and **VULN18** have moderate impact, as obtaining config files normally requires admin credentials, but unknowning users might expose their config files, and by extention their plaintext credentials.
- **VULN15**, **VULN16**, and **VULN17** have low impact, as the contents of default configs are not that sensitive, and exported configs are compressed before encryption, slightly reducing the risk of ECB mode.

Vulnerability	Description	Impact
VULN01	Unauthenticated UART root shell	High
VULN02	WiFi protected by 8 digit WPS pins	Moderate
VULN03	Unsalted MD5 hash in `/etc/passwd.bak`	Low
VULN04	Weak default credentials (`admin:1234`)	Low
VULN05	Multiple outdated dependencies with known CVEs	High
VULN06	Multiple known vulnerabilities regarding GDPR web system	High
VULN07	Command injection vulnerability when restoring config file with UPnP enabled	Moderate
VULN08	Known buffer overflow in `libcmm.so:dm_fillObjByStr`	High
VULN09	Buffer overflow in `libcmm.so:dm_fillObjByStr (value part)`	High
VULN10	Buffer overflow in `libcmm.so:dm_fillObjByStr (key part)`	High
VULN11	Buffer overflow in `httpd:FUN_0040b640 (testarg)`	High
VULN12	Improper HTML encoding in `lib.js:htmlEncodeStr`	Low
VULN13	Plaintext user password in (exported) config	Moderate
VULN14	Plaintext WiFi password in (exported) config	Moderate
VULN15	Weak crypto (DES/ECB) in default configs	Low
VULN16	Weak crypto (DES/ECB) in exported configs	Low
VULN17	Hardcoded crypto key for default configs	Low
VULN18	Hardcoded crypto key for exported configs	Moderate
VULN19	Hardcoded crypto key/iv for in memory config	Low
VULN20	Hardcoded WiFi WPS pin fallback (`12345670`)	Low
VULN21	Unsalted MD5 hash in `dropbearpwd`	Moderate
VULN22	No option to validate signed firmware updates	Low
VULN23	All processes run as admin	High
VULN24	Exploit mitigation (RELO/stack canaries/NX/PIE) not used in binaries	High
VULN25	Stack and heap both have write and execute permission bits set	High
VULN26	ASLR only partially enabled (`randomize_va_space 1`)	Low

Table 4.1: Overview of vulnerabilites

- **VULN19** has low impact, as an attacker capable of capturing the config memory block is likely also able to extract any per-device crypto key, unless a hardware enforced security is used to prevent easy key extraction.
- **VULN20** has low impact, as it is unlikely an attacker can trigger this pin fallback easily.
- **VULN21** has moderate impact, as it can allow an attacker to recover the user password via Rainbow tables, unless a strong password is used, but requires admin access on the device.
- **VULN22** has low impact, as being able to flash custom firmware (e.g. OpenWRT/DD-WRT) is desirable, but not having the option to see if firmware is signed by the vendor to prevent flashing backdoored firmware is a missed opportunity.
- **VULN23**, **VULN24**, and **VULN25** have high impact, as it greatly simplifies exploitation of vulnerabilities such as Remote-Code-Execution.
- **VULN26** has low impact, because compared to mode 2, only the layout of data segments is not randomized.

Chapter 5

Related Work

There has been a substantial amount of research into the state of IoT security, embedded security, router security, and even TP-Link products specifically, conducted by members of the academic community [17, 16, 18, 13, 7, 10, 1, 15], researchers via blog posts [21, 19, 20, 22], and even consumer organizations [5]. A number of technical books have been released focusing on practical skills regarding security research of IoT devices [8, 2].

Researchers have also identified the need for automated vulnerability assessment, and have proposed methods to achieve this [13, 16, 18]. This is still an active topic of research, and while some tools are available that can automate parts of common tasks, such as extracting and performing rudimentary analysis of firmware [50, 49, 48, 51], this is still an ongoing area of research.

Weak IoT security is not a new phenomenon, and the associated risks were long known [7]. The rise of botnets based on the Mirai malware family, and later derivatives, have lead to record breaking DDoS attacks [7, 10]. The resulting outages of important digital infrastructure has shown the significant real world impact, and thus the dire need to address the state of security. Researchers have stated that the responsibility for these DDoS attacks is often passed on to the end users of devices, but argue that the vendors instead should assume responsibility [10].

While some research has been proposed to frustrate the reverse engineering of either hardware and software [4, 3], this is not an answer to the problem at hand. By raising the bar in terms of skills, equipment, time –and thus costs– needed to reverse engineer it is possible less motivated actors are deterred. Evolution in the ransomware threat-scape has shown that major ransomware gangs have undergone significant professionalization, and function much like authentic businesses [6]. This has lead to a Ransomware as a Service (RaaS) model where actors with less technical capabilities buy the services of more capable actors [12]. This illustrates how a small, but well motivated and funded, group can still have substantial impact in a larger ecosystem. Ultimately vulnerabilities should be fixed, rather than made harder to find.

There is a clear need for effective, standardized, and automated analysis of new IoT devices, including routers such as the TP-Link TR-WR802N. As this is not yet a reality, but a topic for future research, it is vital security researchers continue to perform semi-automated, or even manual, analysis of these devices in the meantime, such as done in this research. This is further exemplified by the fact that vulnerable devices already pose an active threat today.

Chapter 6

Future work

This research has provided a broad overview regarding known vulnerabilities and the general state of security. Future research directions include closer inspection of proprietary closed source binaries. Given prior command injections and buffer overflow vulnerabilities, it is not unlikely more vulnerabilities of this kind still linger. Dynamic analysis techniques such as fuzzing and emulation, amongst other methods described in literature [13], could provide to be powerful tools to aid in this process.

Chapter 7

Conclusions

This research has shown that there does seem to be some improvement in the security of TP-Link products, as attempts are made to address previously discovered vulnerabilities. Well establish modern security practices such as layered defenses, principle of least privilege, and exploit mitigation techniques are not, or only partially, utilized. As a result, most vulnerabilities give attackers the ability to completely take over the device, rather than needing to chain multiple exploits to bypass layered security.

Bibliography

[1] Glenn Barrie, Andrew Whyte, and Joyce Bell. Iot security: Challenges and solutions for mining. In *Proceedings of the Second International Conference on Internet of Things, Data and Cloud Computing*, ICC '17, New York, NY, USA, 2017. Association for Computing Machinery.

[2] Fotios Chantiz, Ioannis Stais, Paulino Calderon, Evangelos Deirmentzoglou, and Beau Woods. *Practical IoT Hacking: The Definitive Guide to Attacking the Internet of Things*. No Starch Press, 2020.

[3] Shuai Chen, Junlin Chen, and Lei Wang. A chip-level anti-reverse engineering technique. In *Special Issue on Frontiers of Hardware and Algorithms for On-chip Learning, Special Issue on Silicon Photonics and Regular Papers.*, volume 14, New York, NY, USA, 2018. Association for Computing Machinery.

[4] Jean-Luc Danger, Sylvain Guilley, and Florian Praden. Hardware-enforced protection against software reverse-engineering based on an instruction set encoding. In *Proceedings of ACM SIGPLAN on Program Protection and Reverse Engineering Workshop 2014*, PPREW'14, New York, NY, USA, 2014. Association for Computing Machinery.

[5] The American Consumer Institute Center for Citizen Research. Securing iot devices: How safe is your wi-fi router? `https://www.theamericanconsumer.org/wp-content/uploads/2018/09/FINAL-Wi-Fi-Router-Vulnerabilities.pdf`, 2018.

[6] Samuel Greengard. The worsening state of ransomware. *Commun. ACM*, 64(4):15–17, mar 2021.

[7] Harm Griffioen and Christian Doerr. Examining mirai's battle over the internet of things. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, page 743–756, New York, NY, USA, 2020. Association for Computing Machinery.

[8] Aditya Gupta. *The IoT Hacker's Handbook: A Practical Guide to Hacking the Internet of Things*. Apress, 2019.

[9] Sheharbano Khattak, Naurin Rasheed Ramay, Kamran Riaz Khan, Affan A. Syed, and Syed Ali Khayam. A taxonomy of botnet behavior, detection, and defense. *IEEE Communications Surveys Tutorials*, 16(2):898–924, 2014.

[10] Constantinos Kolias, Georgios Kambourakis, Angelos Stavrou, and Jeffrey Voas. Ddos in the iot: Mirai and other botnets. *Computer*, 50(7):80–84, 2017.

[11] Yonglei Liu, Zhigang Jin, and Ying Wang. Survey on security scheme and attacking methods of wpa/wpa2. In *2010 6th International Conference on Wireless Communications Networking and Mobile Computing (WiCOM)*, pages 1–4, 2010.

[12] Routa Moussaileb, Benjamin Bouget, Aurélien Palisse, Hélène Le Bouder, Nora Cuppens, and Jean-Louis Lanet. Ransomware's early mitigation mechanisms. In *Proceedings of the 13th International Conference on Availability, Reliability and Security*, ARES 2018, New York, NY, USA, 2018. Association for Computing Machinery.

[13] Abdullah Qasem, Paria Shirani, Mourad Debbabi, Lingyu Wang, Bernard Lebel, and Basile L. Agba. Automatic vulnerability detection in embedded devices and firmware: Survey and layered taxonomies. *ACM Comput. Surv.*, 54(2), mar 2021.

[14] Tamara Radivilova and Hassan Ali Hassan. Test for penetration in wi-fi network: Attacks on wpa2-psk and wpa2-enterprise. In *2017 International Conference on Information and Telecommunication Technologies and Radio Electronics (UkrMiCo)*, pages 1–4, 2017.

[15] Franziska Schwarz, Klaus Schwarz, Daniel Fuchs, Reiner Creutzburg, and David Akopian. Firmware vulnerability analysis of widely used low-budget tp-link routers. *Electronic Imaging*, 2021(3):135–1–135–11, 2021.

[16] Chin-Wei Tien, Tsung-Ta Tsai, Ing-Yi Chen, and Sy-Yen Kuo. Ufo - hidden backdoor discovery and security verification in iot device firmware. In *2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pages 18–23, 2018.

[17] John Viega and Hugh Thompson. The state of embedded-device security (spoiler alert: It's bad). *IEEE Security Privacy*, 10(5):68–70, 2012.

[18] Jueqi Wang, Hongshuai Li, Junyou Ye, and Jianchu Xiao. Research on intelligent reverse analysis technology of firmware of internet of things. In *2021 IEEE International Conference on Power, Intelligent Computing and Systems (ICPICS)*, pages 164–169, 2021.

[19] Reverse engineering my router's firmware with binwalk. https://embeddedbits.org/reverse-engineering-router-firmware-with-binwalk/.

[20] 'amazon's choice' best-selling tp-link router ships with vulnerable firmware. https://cybernews.com/security/amazon-tp-link-router-ships-with-vulnerable-firmware/.

[21] Buffer overflow vulnerability in tp-link routers can allow remote attackers to take control. https://securityintelligence.com/buffer-overflow-vulnerability-in-tp-link-routers-can-allow-remote-attackers-to-

[22] Unauthenticated root shell on tp-link tl-wr902ac router. https://pwn2learn.dusuel.fr/blog/unauthenticated-root-shell-on-tp-link-tl-wr902ac-router/.

[23] Using ghidra to extract a router configuration encryption key. https://hackaday.com/2021/07/15/using-ghidra-to-extract-a-router-configuration-encryption-key/.

[24] Tp-link's attempt at gdpr compliance. https://hex.fish/2021/05/10/tp-link-gdpr/.

[25] Hacking the tl-wpa4220, part 2: The command injections. https://the-hyperbolic.com/posts/hacking-the-tlwpa4220-part-2/.

[26] Hacking the tl-wpa4220, part 4: The buffer overflow. https://the-hyperbolic.com/posts/hacking-the-tlwpa4220-part-4/.

[27] Buffer 0verflow in tp-link devices. https://github.com/liyansong2018/CVE/tree/main/2021/CVE-2021-29302.

[28] Tp-link router products. https://www.tp-link.com/en/search/?q=router&t=product&p=1.

[29] Tp-link tl-wr802n. https://www.tp-link.com/en/home-networking/wifi-router/tl-wr802n/.

[30] Mediatek mt7628k/n/a. https://www.mediatek.com/products/homeNetworking/mt7628k-n-a.

[31] Download for tl-wr802n v4. https://www.tp-link.com/en/support/download/tl-wr802n/#Firmware.

[32] Download for tl-wr802n v4. https://www.tp-link.com/en/support/download/tl-wr802n/#GPL-Code.

[33] Cryptojs. https://github.com/brix/crypto-js.

[34] Cyberchef. https://gchq.github.io/CyberChef/.

[35] Des3.js source. https://www.yisu.com/zixun/179627.html.

[36] Tl-wr702n default password. https://twitter.com/LargeCardinal/status/682591420969029632.

[37] Cves for busybox. https://nvd.nist.gov/vuln/search/results?adv_search=true&isCpeNameSearch=true&query=cpe%3A2.3%3Aa%3Abusybox%3Abusybox%3A1.19.2%3A*%3A*%3A*%3A*%3A*%3A*%3A*.

[38] Firmware format analysis for tp-link firmwares with the version 3 header (0x03000000). https://github.com/xdarklight/mktplinkfw3.

[39] Tom wu's pure javascript implementation of arbitrary-precision integer arithmetic. https://github.com/creationix/jsbn.

[40] Cves for uclibc. https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&isCpeNameSearch=true&seach_type=all&query=cpe:2.3:a:uclibc:uclibc:0.9.33.2:*:*:*:*:*:*:*.

[41] pppd cve-2020-8597. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-8597.

[42] Command line utility to convert tp-link router backup config files. https://github.com/sta-c0000/tpconf_bin_xml.

[43] Cves for dropbear. https://nvd.nist.gov/vuln/search/results?adv_search=true&isCpeNameSearch=true&query=cpe%3A2.3%3Aa%3Adropbear_ssh_project%3Adropbear_ssh%3A2012.55%3A*%3A*%3A*%3A*%3A*%3A*%3A*.

[44] Cves for openssl. https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&isCpeNameSearch=true&seach_type=all&query=cpe:2.3:a:openssl:openssl:0.9.8zh:*:*:*:*:*:*:*.

[45] Cves for iptables. https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&isCpeNameSearch=true&seach_type=all&query=cpe:2.3:a:netfilter:iptables:1.4.17:*:*:*:*:*:*:*.

[46] Cves for u-boot. https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&isCpeNameSearch=true&seach_type=all&query=cpe:2.3:a:denx:u-boot:1.1.3:*:*:*:*:*:*:*.

[47] Cves for linux. https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&isCpeNameSearch=true&seach_type=all&query=cpe:2.3:o:linux:linux_kernel:2.6.36:-:*:*:*:*:*:*.

[48] Firmware modification kit. https://bitsum.com/firmware_mod_kit.htm.

[49] Firmware analysis toolkit. https://github.com/attify/firmware-analysis-toolkit.

[50] Binwalk. https://github.com/ReFirmLabs/binwalk.

[51] Firmwalker. https://github.com/craigz28/firmwalker.

[52] Bus pirate. http://dangerousprototypes.com/docs/Bus_Pirate.

[53] Flashrom. https://www.flashrom.org/Flashrom.

[54] Ghidra. https://ghidra-sre.org/.

[55] Burp suite. https://portswigger.net/burp.

[56] Checksec. https://github.com/slimm609/checksec.sh.

[57] Bindiff. https://www.zynamics.com/bindiff/manual/.

Appendix A

Appendix

A.1 TL-WR802N Boot log

```
DDR Calibration DQS reg = 00008688


U-Boot 1.1.3 (Jun 23 2020 - 17:33:43)

Board: Ralink APSoC DRAM:  64 MB
relocate_code Pointer at: 83fb8000
flash manufacture id: 20, device id 70 17
Warning: un-recognized chip ID, please update bootloader!
============================================
Ralink UBoot Version: 4.3.0.0
--------------------------------------------
ASIC 7628_MP (Port5<->None)
DRAM component: 512 Mbits DDR, width 16
DRAM bus: 16 bit
Total memory: 64 MBytes
Flash component: SPI Flash
Date:Jun 23 2020  Time:17:33:43
============================================
```

```
icache: sets:512, ways:4, linesz:32 ,total:65536

dcache: sets:256, ways:4, linesz:32 ,total:32768

 ##### The CPU freq = 580 MHZ ####

 estimate memory size =64 Mbytes

RESET MT7628 PHY!!!!!!

continue to starting system.

0

disable switch phyport...


3: System Boot system code via Flash.(0xbc020000)

do_bootm:argc=2, addr=0xbc020000

## Booting image at bc020000 ...

   Uncompressing Kernel Image ... OK

No initrd

## Transferring control to Linux (at address 8000c150) ...

## Giving linux memsize in MB, 64

Starting kernel ...



LINUX started...

 THIS IS ASIC
Linux version 2.6.36 (jenkins@mobile-System) (gcc version 4.6.3
   (Buildroot 2012.11.1) ) #1 Tue Jun 23 17:35:59 CST 2020

 The CPU feqenuce set to 575 MHz

 MIPS CPU sleep mode enabled.
CPU revision is: 00019655 (MIPS 24Kc)
Software DMA cache coherency
Determined physical RAM map:
 memory: 04000000 @ 00000000 (usable)
Initrd not found or empty - disabling initrd
Zone PFN ranges:
  Normal   0x00000000 -> 0x00004000
```

```
Movable zone start PFN for each node
early_node_map[1] active PFN ranges
    0: 0x00000000 -> 0x00004000
Built 1 zonelists in Zone order, mobility grouping on.  Total
    pages: 16256
Kernel command line: console=ttyS1,115200 root=/dev/mtdblock2
    rootfstype=squashfs init=/sbin/init
PID hash table entries: 256 (order: -2, 1024 bytes)
Dentry cache hash table entries: 8192 (order: 3, 32768 bytes)
Inode-cache hash table entries: 4096 (order: 2, 16384 bytes)
Primary instruction cache 64kB, VIPT, , 4-waylinesize 32 bytes.
Primary data cache 32kB, 4-way, PIPT, no aliases, linesize 32
    bytes
Writing ErrCtl register=00024c64
Readback ErrCtl register=00024c64
Memory: 61424k/65536k available (2414k kernel code, 4112k
    reserved, 635k data, 160k init, 0k highmem)
NR_IRQS:128
console [ttyS1] enabled
Calibrating delay loop... 386.04 BogoMIPS (lpj=772096)
pid_max: default: 4096 minimum: 301
Mount-cache hash table entries: 512
NET: Registered protocol family 16
bio: create slab <bio-0> at 0
Switching to clocksource Ralink Systick timer
NET: Registered protocol family 2
IP route cache hash table entries: 1024 (order: 0, 4096 bytes)
TCP established hash table entries: 2048 (order: 2, 16384 bytes)
TCP bind hash table entries: 2048 (order: 1, 8192 bytes)
TCP: Hash tables configured (established 2048 bind 2048)
TCP reno registered
NET: Registered protocol family 1
squashfs: version 4.0 (2009/01/31) Phillip Lougher
fuse init (API version 7.15)
msgmni has been set to 119
io scheduler noop registered
io scheduler deadline registered (default)
Ralink gpio driver initialized
i2cdrv_major = 218
Serial: 8250/16550 driver, 2 ports, IRQ sharing enabled
serial8250: ttyS0 at MMIO 0x10000d00 (irq = 21) is a 16550A
serial8250: ttyS1 at MMIO 0x10000c00 (irq = 20) is a 16550A
brd: module loaded
flash manufacture id: 20, device id 70 17
Warning: un-recognized chip ID, please update SPI driver!
N25Q064A13ESE40F(20 ba171000) (8192 Kbytes)
mtd .name = raspi, .size = 0x00800000 (8M) .erasesize = 0
    x00010000 (64K) .numeraseregions = 0
Creating 7 MTD partitions on "raspi":
0x000000000000-0x000000020000 : "boot"
0x000000020000-0x000000160000 : "kernel"
0x000000160000-0x0000007c0000 : "rootfs"
mtd: partition "rootfs" set to be root filesystem
0x0000007c0000-0x0000007d0000 : "config"
```

```
0x0000007d0000-0x0000007e0000 : "romfile"
0x0000007e0000-0x0000007f0000 : "rom"
0x0000007f0000-0x000000800000 : "radio"
Register flash device:flash0
PPP generic driver version 2.4.2
PPP MPPE Compression module registered
NET: Registered protocol family 24
Mirror/redirect action on
u32 classifier
    Actions configured
Netfilter messages via NETLINK v0.30.
nf_conntrack version 0.5.0 (959 buckets, 3836 max)
ip_tables: (C) 2000-2006 Netfilter Core Team, Type=Linux
TCP cubic registered
NET: Registered protocol family 10
ip6_tables: (C) 2000-2006 Netfilter Core Team
IPv6 over IPv4 tunneling driver
NET: Registered protocol family 17
Ebtables v2.0 registered
802.1Q VLAN Support v1.8 Ben Greear <greearb@candelatech.com>
All bugs added by David S. Miller <davem@redhat.com>
VFS: Mounted root (squashfs filesystem) readonly on device 31:2.
Freeing unused kernel memory: 160k freed

starting pid 34, tty '': '/etc/init.d/rcS'
cp: can't stat '/etc/SingleSKU_FCC.dat': No such file or
    directory
rdm_major = 253
spiflash_ioctl_read, Read from 0x007df100 length 0x6, ret 0,
    retlen 0x6
Read MAC from flash(  7df100) 60-ffffffa4-ffffffb7-05-ffffffb6-3
    e
GMAC1_MAC_ADRH -- : 0x000060a4
GMAC1_MAC_ADRL -- : 0xb705b63e
Ralink APSoC Ethernet Driver Initilization. v3.1  256 rx/tx
    descriptors allocated, mtu = 1500!
NAPI enable, Tx Ring = 256, Rx Ring = 256
spiflash_ioctl_read, Read from 0x007df100 length 0x6, ret 0,
    retlen 0x6
Read MAC from flash(  7df100) 60-ffffffa4-ffffffb7-05-ffffffb6-3
    e
GMAC1_MAC_ADRH -- : 0x000060a4
GMAC1_MAC_ADRL -- : 0xb705b63e
PROC INIT OK!
add domain:tplinkwifi.net
add domain:tplinkap.net
add domain:tplinkrepeater.net
add domain:tplinklogin.net
tp_domain init ok
L2TP core driver, V2.0
PPPoL2TP kernel driver, V2.0
Set: phy[0].reg[0] = 3900
Set: phy[1].reg[0] = 3900
Set: phy[2].reg[0] = 3900
```

```
Set: phy[3].reg[0] = 3900
Set: phy[4].reg[0] = 3900
Set: phy[0].reg[0] = 3300
Set: phy[1].reg[0] = 3300
Set: phy[2].reg[0] = 3300
Set: phy[3].reg[0] = 3300
Set: phy[4].reg[0] = 3300
resetMiiPortV over.
Set: phy[0].reg[4] = 01e1
Set: phy[0].reg[0] = 3300
Set: phy[1].reg[4] = 01e1
Set: phy[1].reg[0] = 3300
Set: phy[2].reg[4] = 01e1
Set: phy[2].reg[0] = 3300
Set: phy[3].reg[4] = 01e1
Set: phy[3].reg[0] = 3300
Set: phy[4].reg[4] = 01e1
Set: phy[4].reg[0] = 3300
turn off flow control over.

starting pid 82, tty '/dev/ttyS1': '/bin/sh'
~ # [ util_execSystem ] 141: ipt_init cmd is "/var/tmp/dconf/rc
    .router"

[ dm_readFile ] 2061: can not open xml file /var/tmp/pc/
    reduced_data_model.xml!, about to open file /etc/
    reduced_data_model.xml
spiflash_ioctl_read, Read from 0x007c0000 length 0x10000, ret 0,
     retlen 0x10000
spiflash_ioctl_read, Read from 0x007c0000 length 0x10, ret 0,
    retlen 0x10
[ dm_loadCfg ] 2364: software version is not match, in config,
    version = 0
[ dm_readFile ] 2061: can not open xml file /var/tmp/pc/
    default_config.xml!, about to open file /etc/default_config.
    xml
[ parseConfigNode ] 525: Meet unrecognized object node "
    PhDDNSCfg", skip the node
[ parseConfigNode ] 530: Meet unrecognized parameter node "
    PhDDNSCfg", skip the node
[ parseConfigNode ] 525: Meet unrecognized object node "ACL",
    skip the node
[ parseConfigNode ] 530: Meet unrecognized parameter node "ACL
    ", skip the node
[ parseConfigNode ] 530: Meet unrecognized parameter node "
    X_TP_TimeZoneSetByUser", skip the node
[ parseConfigNode ] 530: Meet unrecognized parameter node "
    MACAddressControlEnabled", skip the node
[ parseConfigNode ] 530: Meet unrecognized parameter node "
    X_TP_MACAddressControlRule", skip the node
[ parseConfigNode ] 530: Meet unrecognized parameter node "Vlan
    ", skip the node
[ parseConfigNode ] 530: Meet unrecognized parameter node "
    MACAddressControlEnabled", skip the node
```

```
[ parseConfigNode ] 530:  Meet unrecognized parameter node "
    X_TP_MACAddressControlRule", skip the node
[ parseConfigNode ] 530:  Meet unrecognized parameter node "Vlan
    ", skip the node
[ parseConfigNode ] 530:  Meet unrecognized parameter node "
    MACAddressControlEnabled", skip the node
[ parseConfigNode ] 530:  Meet unrecognized parameter node "
    X_TP_MACAddressControlRule", skip the node
[ parseConfigNode ] 530:  Meet unrecognized parameter node "Vlan
    ", skip the node
[ parseConfigNode ] 525:  Meet unrecognized object node "
    X_TP_QuickSave", skip the node
[ parseConfigNode ] 525:  Meet unrecognized object node "
    X_TP_QuickSave", skip the node
[ parseConfigNode ] 525:  Meet unrecognized object node "
    X_TP_WANUSB3gLinkConfig", skip the node
[ parseConfigNode ] 525:  Meet unrecognized object node "
    QueueManagement", skip the node
[ parseConfigNode ] 525:  Meet unrecognized object node "
    X_TP_IPTV", skip the node
[ parseConfigNode ] 525:  Meet unrecognized object node "
    VoiceService", skip the node
[ parseConfigNode ] 530: spiflash_ioctl_read , Read from 0
    x007df100 length 0x6, ret 0, retlen 0x6
 Meet unrecognizspiflash_ioctl_read , Read from 0x007df200 length
      0x4, ret 0, retlen 0x4
ed parameter nodspiflash_ioctl_read , Read from 0x007df300 length
     0x4, ret 0, retlen 0x4
e "VoiceService"spiflash_ioctl_read , Read from 0x007df400 length
     0x10, ret 0, retlen 0x10
, skip the node
spiflash_ioctl_read , Read from 0x007df500 length 0x29, ret 0,
    retlen 0x29

[ parseConfigNospiflash_ioctl_read , Read from 0x007df600 length
    0x21, ret 0, retlen 0x21
de ] 525: Meet spiflash_ioctl_read , Read from 0x007df700 length
     0x10, ret 0, retlen 0x10
unrecognized objspiflash_ioctl_read , Read from 0x007df700 length
     0x10, ret 0, retlen 0x10
ect node "Storagspiflash_ioctl_read , Read from 0x00020000 length
     0x1d0, ret 0, retlen 0x1d0
eService", skip spiflash_ioctl_read , Read from 0x007df100 length
     0x6, ret 0, retlen 0x6
the node
[ parseConfigNode ] 525:  Meet unrecognized object node "
    X_TP_SpeedDialCfg", skip the node
[ parseConfigNode ] 525:  Meet unrecognized object node "
    X_TP_MultiIspDialPlan", skip the node
[ parseConfigNode ] 525:  Meet unrecognized object node "
    X_TP_CallLogCfg", skip the node
[ parseConfigNode ] 530:  Meet unrecognized parameter node "
    X_TP_Band", skip the node
[ parseConfigNode ] 530:  Meet unrecognized parameter node "
```

```
    X_TP_Band", skip the node
[ parseConfigNode ] 530:  Meet unrecognized parameter node "
    X_TP_Band", skip the node
[ parseConfigNode ] 530:  Meet unrecognized parameter node "
    WEPKeyIndex", skip the node
[ parseConfigNode ] 530:  Meet unrecognized parameter node "
    WEPKeyIndex", skip the node
[ parseConfigNode ] 530:  Meet unrecognized parameter node "
    WEPKeyIndex", skip the node
[ parseConfigNode ] 530:  Meet unrecognized parameter node "
    WEPKeyIndex", skip the node
[ parseConfigNode ] 530:  Meet unrecognized parameter node "
    X_TP_Band", skip the node
[ parseConfigNode ] 530:  Meet unrecognized parameter node "
    WEPKeyIndex", skip the node
[ parseConfigNode ] 530:  Meet unrecognized parameter node "
    WEPKeyIndex", skip the node
[ parseConfigNode ] 530:  Meet unrecognized parameter node "
    WEPKeyIndex", skip the node
[ parseConfigNode ] 530:  Meet unrecognized parameter node "
    WEPKeyIndex", skip the node
===>Enter Router mode
[ oal_sys_readMacFlash ] 1934:   7df100 set flash mac : 60:A4:B7
    :05:B6:3E.
[ oal_sys_readMacFlash ] 1934:   7df100 set flash mac : 60:A4:B7
    :05:B6:3E.
sendto: No such file or directory
pid 81 send 2001 error
[ util_execSystem ] 141:  oal_startDynDns cmd is "dyndns /var/
    tmp/dconf/dyndns.conf"

Get SNTP new config
[ util_execSystem ] 141:  oal_startNoipDns cmd is "noipdns /var/
    tmp/dconf/noipdns.conf"

[ util_execSystem ] 141:  oal_startCmxDns cmd is "cmxdns /var/
    tmp/dconf/cmxdns.conf"

ioctl: No such device
[ util_execSystem ] 141:  oal_br_addBridge cmd is "brctl addbr
    br0;brctl setfd br0 0;brctl stp br0 off"

[ util_execSystem ] 141:  oal_ipt_addLanRules cmd is "iptables -
    t filter -A INPUT -i br+ -j ACCEPT
"

[ util_execSystem ] 141:  oal_intf_setIntf cmd is "ifconfig br0
    192.168.0.1 netmask 255.255.255.0 up"

[ util_execSystem ] 141:  oal_util_setProcLanAddr cmd is "echo "
    br0 16820416," > /proc/net/conRaeth v3.1 (
    ntract_LocalAddrNAPI
"
```

```
[ util_exec ,SkbRecycleSystem ] 141:  o)
al_intf_enableIn
phy_tx_ring = 0x030d2000 , tx_ring = 0xa30d2000
tf cmd is "ifcon
phy_rx_ring0 = 0x030d3000 , rx_ring0 = 0xa30d3000
fig eth0 up"

[fe_sw_init:5357]rt305x_esw_init.
disable switch phyport...
GMAC1_MAC_ADRH -- : 0x000060a4
GMAC1_MAC_ADRL -- : 0xb705b63e
RT305x_ESW: Link Status Changed
[ rsl_getUnusedVlan ] 1079:  GET UNUSED VLAN TAG 1 : [3]
[ rsl_getUnusedVlan ] 1079:  GET UNUSED VLAN TAG 2 : [4]
[ rsl_getUnusedVlan ] 1079:  GET UNUSED VLAN TAG 3 : [5]
[ rsl_getUnusedVlan ] 1079:  GET UNUSED VLAN TAG 4 : [6]
[ util_execSystem ] 141:  oal_addVlanTagIntf cmd is "vconfig add
     eth0 3"

[ util_execSystem ] 141:  oal_intf_enableIntf cmd is "ifconfig
    eth0.3 up"

set if eth0.3 to *not wan dev
[ util_execSystem ] 141:  oal_addVlanTagIntf cmd is "vconfig add
     eth0 4"

[ util_execSystem ] 141:  oal_intf_enableIntf cmd is "ifconfig
    eth0.4 up"

set if eth0.4 to *not wan dev
[ util_execSystem ] 141:  oal_addVlanTagIntf cmd is "vconfig add
     eth0 5"

[ util_execSystem ] 141:  oal_intf_enableIntf cmd is "ifconfig
    eth0.5 up"

set if eth0.5 to *not wan dev
[ util_execSystem ] 141:  oal_addVlanTagIntf cmd is "vconfig add
     eth0 6"

[ util_execSystem ] device eth0.3 entered promiscuous mode
141:  oal_intf_edevice eth0 entered promiscuous mode
nableIntf cmd isbr0: port 1(eth0.3) entering forwarding state
 "ifconfig eth0.br0: port 1(eth0.3) entering forwarding state
6 up"

set if eth0.6 to *not wan dev
[ util_execSystem ] 141:  oal_addVlanTagdevice eth0.4 entered
    promiscuous mode
Intf cmd is "vcobr0: port 2(eth0.4) entering forwarding state
nfig add eth0 2"br0: port 2(eth0.4) entering forwarding state


[ util_execSystem ] 141:  oal_intf_enableIntf cmd is "ifconfig
```

```
   eth0.2 up"

device eth0.5 entered promiscuous mode

set if eth0.2 tbr0: port 3(eth0.5) entering forwarding state
o wan dev
[ vlabr0: port 3(eth0.5) entering forwarding state
n_addLanPortsIntoBridge ] 606:  add lan Port 255 from br0
[ util_execSystem ] 1device eth0.6 entered promiscuous mode
41:  oal_br_addIbr0: port 4(eth0.6) entering forwarding state
ntfIntoBridge cmbr0: port 4(eth0.6) entering forwarding state
d is "brctl addif br0 eth0.3"

[ util_execSystem ] 141:  oal_br_addIntfIntoBridge cmd is "brctl
    addif br0 eth0.4"

[ util_execSystem ] 141:  oal_br_addIntfIntoBridge cmd is "brctl
    addif br0 eth0.5"

[ util_execSystem ] 141:  oal_br_addIntfIntoBridge cmd is "brctl
    addif br0 eth0.6"

[ util_execSystem ] 141:  rsl_initIPv6CfgObj cmd is "echo 1 > /
   proc/sys/net/ipv6/conf/all/disable_ipv6"

[ util_execSystem ] 141:  oal_eth_setIGMPSnoopParam cmd is "for
   i in /sys/devices/virtual/net/*/bridge/multicast_snooping;do
   echo 1 > $i ; done"

[ util_execSystem ] 141:  oal_wlan_ra_setCountryRegion cmd is "
   cp /etc/SingleSKU_CE.dat /var/Wireless/RT2860AP/SingleSKU.dat
   "

[ util_execSystem ] 141:  oal_wlan_ra_setCountryRegion cmd is "
   iwpriv ra0 set CountryRegion=1"

ra0       no private ioctls.

[ util_execSystem ] 141:  oal_wlan_ra_loadDriver cmd is "insmod
   /lib/modules/kmdir/kernel/drivers/net/wireless/mt_wifi_ap/
   mt_wifi.ko"

ADDRCONF(NETDEV_CHANGE): eth0.4: link becomes ready
ADDRCONF(NETDEV_CHANGE): eth0.5: link becomes ready
ADDRCONF(NETDEV_CHANGE): eth0.6: link becomes ready
ADDRCONF(NETDEV_CHANGE): eth0.2: link becomes ready


=== pAd = c085f000, size = 1509912 ===

<--- RTMPAllocTxRxRingMemory, Status=0, ErrorValue=0x
<--- RTMPAllocAdapterBlock, Status=0
RtmpChipOpsHook(492): Not support for HIF_MT yet!
mt7628_init()--->
```

```
mt7628_init(FW(8a00), HW(8a01), CHIPID(7628))
e2.bin mt7628_init(1156)::(2), pChipCap->fw_len(64848)
mt_bcn_buf_init(218): Not support for HIF_MT yet!
<--mt7628_init()
[ util_execSystem ] 141: oal_wlan_ra_initWlan cmd is "ifconfig
    ra0 up"

TX_BCN DESC a32be000 size = 320
RX[0] DESC a32c0000 size = 2048
RX[1] DESC a32c1000 size = 2048
RT_CfgSetApcliMacAddress : invalid mac setting
cfg_mode=9
cfg_mode=9
wmode_band_equal(): Band Equal!
AndesSendCmdMsg: Could not send in band command due to diable
    fRTMP_ADAPTER_MCU_SEND_IN_BAND_CMD
APSDCapable[0]=0
APSDCapable[1]=0
APSDCapable[2]=0
APSDCapable[3]=0
APSDCapable[4]=0
APSDCapable[5]=0
APSDCapable[6]=0
APSDCapable[7]=0
APSDCapable[8]=0
APSDCapable[9]=0
APSDCapable[10]=0
APSDCapable[11]=0
APSDCapable[12]=0
APSDCapable[13]=0
APSDCapable[14]=0
APSDCapable[15]=0
default ApCliAPSDCapable[0]=0
Key1Str is Invalid key length(0) or Type(0)
Key1Str is Invalid key length(0) or Type(0)
Key2Str is Invalid key length(0) or Type(0)
Key2Str is Invalid key length(0) or Type(0)
Key3Str is Invalid key length(0) or Type(0)
Key3Str is Invalid key length(0) or Type(0)
Key4Str is Invalid key length(0) or Type(0)
Key4Str is Invalid key length(0) or Type(0)
WscKeyASCII=8
WscKeyASCII=8
[RTMPReadParametersHook:297]wifi read profile faild.
load fw image from fw_header_image
AndesMTLoadFwMethod1(2263)::pChipCap->fw_len(64848)
FW Version:1
FW Build Date:20180704090333
CmdAddressLenReq:(ret = 0)
CmdFwStartReq: override = 1, address = 1048576
CmdStartDLRsp: WiFI FW Download Success
MtAsicDMASchedulerInit(): DMA Scheduler Mode=0(LMAC)
efuse_probe: efuse = 10000012
RtmpChipOpsEepromHook::e2p_type=0, inf_Type=4
```

```
RtmpEepromGetDefault::e2p_dafault=2
RtmpChipOpsEepromHook: E2P type(2), E2pAccessMode = 2, E2P
    default = 2
NVM is FLASH mode
1. Phy Mode = 14
exec!
spiflash_ioctl_read, Read from 0x007f0000 length 0x400, ret 0,
    retlen 0x400
eeFlashId = 0x7628!
FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

Country Region from e2p = ffff
tssi_1_target_pwr_g_band = 34
2. Phy Mode = 14
3. Phy Mode = 14
NICInitPwrPinCfg(11): Not support for HIF_MT yet!
NICInitializeAsic(651): Not support rtmp_mac_sys_reset () for
    HIF_MT yet!
mt_mac_init()-->
MtAsicInitMac()-->
```

```
mt7628_init_mac_cr()-->
MtAsicSetMacMaxLen(1277): Set the Max RxPktLen=450!
<--mt_mac_init()
        WTBL Segment 1 info:
                MemBaseAddr/FID:0x28000/0
                EntrySize/Cnt:32/128
        WTBL Segment 2 info:
                MemBaseAddr/FID:0x40000/0
                EntrySize/Cnt:64/128
        WTBL Segment 3 info:
                MemBaseAddr/FID:0x42000/64
                EntrySize/Cnt:64/128
        WTBL Segment 4 info:
                MemBaseAddr/FID:0x44000/128
                EntrySize/Cnt:32/128
AntCfgInit(2952): Not support for HIF_MT yet!
MCS Set = ff ff 00 00 01
MtAsicSetChBusyStat(861): Not support for HIF_MT yet!
FW LOG: !!!! Pass, dont need recal (total fail[0])

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101
total fail[0])

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101
total fail[0])

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101
total fail[0])

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101
total fail[0])

FW LOG: !!!! Pass, dont need recal (total fail[0])

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101
total fail[0])

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101
total fail[0])

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101
total fail[0])

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101
total fail[0])

FW LOG: !!!! Pass, dont need recal (total fail[0])

FW LOG: RxDCOC Set DC Valid(8)(2)

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101
total fail[0])

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101
```

```
total fail[0])

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101
total fail[0])

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101
total fail[0])

CmdSlotTimeSet:(ret = 0)
FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101
```

```
FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

 [PMF] ap_pmf_init:: apidx=0, MFPC=0, MFPR=0, SHA256=0
 [PMF] RTMPMakeRsnIeCap: RSNIE Capability MFPC=0, MFPR=0
 [PMF] ap_pmf_init:: apidx=1, MFPC=0, MFPR=0, SHA256=0
 MtAsicSetRalinkBurstMode(3156): Not support for HIF_MT yet!
 MtAsicSetPiggyBack(796): Not support for HIF_MT yet!
FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

FW LOG: rlmRF_AUX_TZ u2cfgopt=0x101

 reload DPD from flash , 0x9F = [c400] doReload bit7[0]
 CmdLoadDPDDataFromFlash: Channel = 10, DoReload = 0
 MtAsicSetTxPreamble(3135): Not support for HIF_MT yet!
 MtAsicAddSharedKeyEntry(1344): Not support for HIF_MT yet!
 The 4-BSSID mode is enabled, the BSSID byte5 MUST be the
     multiple of 4
```

```
MtAsicSetPreTbtt(): bss_idx=0, PreTBTT timeout = 0xf0
ap_ftkd> Initialize FT KDP Module...
Main bssid = 60:a4:b7:05:b6:3e
<==== rt28xx_init, Status=0
@@@ ed_monitor_exit : ====>
@@@ ed_monitor_exit : <====
mt7628_set_ed_cca: TURN OFF EDCCA  mac 0x10618 = 0xd7083f0f,
    EDCCA_Status=0
WiFi Startup Cost (ra0): 3.600s
[ util_execSystem ] 141:  oal_wlan_ra_initWlan cmd is "echo 1 >
    /proc/tplink/led_wlan_24G"

[ util_execSystem ] 141:  oal_wlanSet_ed_chk_proc()::ed_chk=0
_ra_initWlan cmdmt7628_set_ed_cca: TURN OFF EDCCA  mac 0x10618 =
     0xd7083f0f, EDCCA_Status=0
 is "iwpriv ra0 set ed_chk=0"

[ util_execSystem ] 141:  oal_wlan_ra_setStaNum cmd is "iwpriv
    ra0 set MaxStaNum=32"

[ util_execSystem ] 141:  oal_br_addIntfIntoBridge cmd device
    ra0 entered promiscuous mode
is "brctl addif br0: port 5(ra0) entering forwarding state
br0 ra0"

br0: port 5(ra0) entering forwarding state
[ util_execSystem ] 141:  oal_br_addIntfIntoBridge cmd is "
    brctldevice apcli0 entered promiscuous mode
 addif br0 apcli0"

[ util_execSystem ] 141:  oal_br_addIntfIntoBridge cmd is "brctl
     addif br0 apcli0"

brctl: bridge br0: Device or resource busy
[ util_edevice ra1 entered promiscuous mode
xecSystem ] 141:  oal_br_addIntfIntoBridge cmd is "brctl addif
    br0 ra1"

[ utispiflash_ioctl_read, Read from 0x007f0000 length 0x2, ret
    0, retlen 0x2
l_execSystem ] 141:  oal_wlan_ra_initEnd cmd is "wlNetlinkTool
    &"

[ util_execSystem ] 141:  oal_wlan_ra_initEnd cmd is "killall -q
     wscd"

[ util_execSystem ] 141:  oal_wlan_ra_initEnd cmd is "wscd -i
    ra0 -m 1 -w /var/tmp/wsc_upnp/ &"

[ util_execSystem ] 141:  rsl_initLanWlanObj cmd is "echo 0 > /
    proc/tplink/wl_mode"

WLAN-Start wlNetlinkTool
[ oal_wlan_ra_loadDriver ] 2107:  no 5G chip.
```

```
[ rsl_initLanWlanObj ] 9431:  perror:1
Waiting for Wireless Events from interfaces...
swWlanChkAhbErr: netlink to do
wscd: SSDP UDP PORT = 1900
sendto: No such file or directory
pid 81 send 2030 error
sendto: No such file or directory
pid 81 send 2004 error
[ util_execSystem ] 141:  oal_startDhcps cmd is "dhcpd /var/tmp/
    dconf/udhcpd.conf"

iptables: Bad rule (does a matching rule exist in that chain?).
[ util_execSystem ] 141:  oal_lan6_startDhcp6s cmd is "dhcp6s -c
     /var/tmp/dconf/dhcp6s_br0.conf -P /var/run/dhcp6s_br0.pid
    br0 &"

[ util_execSystem ] 141:  oal_lan6_startRadvd cmd is "radvd -C /
    var/tmp/dconf/radvd_br0.conf -p /var/run/radvd_br0.pid &"

[ util_execSystem ] 141:  oal_startSnmp cmd is "snmpd -f /var/
    tmp/dconf/snmpd.conf"

mldProxy# file: src/mld_ifinfo.c;line: 102; error = No such file
     or directory
mldProxy# Err: get LLA failed
radvd starting
[Jan 01 00:00:08] radvd: no linklocal address configured for br0
[Jan 01 00:00:08] radvd: error parsing or activating the config
    file: /var/tmp/dconf/radvd_br0.conf
[ rsl_initEwanObj ] 298: Initialize EWAN, enable(1)!
[ rsl_setEwanObj ] 208: Get Ethernet's stack!
[ rsl_setEwanObj ] 262: enable ethernet interface now!
[ oal_ewan_enable ] 469: pEwan->ifName(eth0.2)
[ util_execSystem ] 141:  oal_br_delIntfFromBridge cmd is "brctl
     delif br0 eth0.2"

brctl: bridge br0: Invalid argument
[ rsl_setEwanObj ] 268: EWAN.ifname(eth0.2)!
[ wan_conn_wanIpConn_getConnectionInfo ] 906: GET MAC(60:A4:B7
    :05:B6:3F) successfully!
[ util_execSystem ] 141:  oal_intf_setIfMac cmd is "ifconfig
    eth0.2 down"

[ util_execSystem ] 141:  oal_intf_setIfMac cmd is "ifconfig
    eth0.2 hw ether 60:A4:B7:05:B6:3F up"

[ util_execSystem ] 141:  oal_intf_enableIntf cmd is "ifconfig
    eth0.2 up"

[ rsl_initWanPppConnObj ] 398: into rsl_initWanPppConnObj!
[ rsl_initWanPppConnObj ] 515: rsl_initWanPppConnObj successed!
[ rsl_initWanPppConnObj ] 398: into rsl_initWanPppConnObj!
```

```
[ rsl_initWanPppConnObj ] 515: rsl_initWanPppConnObj successed!
[ rsl_initAppObj ] 1065: ==> start dhcp client

[ util_execSystem ] 141: oal_ipt_fwDdos cmd is "iptables -D
    FORWARD -j FIREWALL_DDOS
"

iptables: No chain/target/match by that name.
[ util_execSystem ] 141: oal_ipt_forbidLanPing cmd is "iptables
     -t filter -D INPUT -i br+ -p icmp --icmp-type echo-request -
    j DROP
iptables -t filter -D FORWARD -i br+ -p icmp --icmp-type echo-
    request -j DROP
"

iptables: Bad rule (does a matching rule exist in that chain?).
iptables: Bad rule (does a matching rule exist in that chain?).
[ util_execSystem ] 141: oal_ddos_delPingRule cmd is "iptables
    -t filter -D INPUT ! -i br+ -p icmp --icmp-type echo-request
    -j ACCEPT
"

iptables: Bad rule (does a matching rule exist in that chain?).
[ util_execSystem ] 141: oal_ipt_setDDoSRules cmd is "iptables
    -F FIREWALL_DDOS"

[ util_execSystem ] 141: ddos_clearAll cmd is "rm -f /var/tmp/
    dosHost"

[ util_execSystem ] 141: oal_initFirewallObj cmd is "ebtables -
    N FIREWALL"

[ util_execSystem ] 141: oal_initIp6FirewallObj cmd is "
    ip6tables -F"

[ util_execSystem ] 141: oal_initIp6FirewallObj cmd is "
    ip6tables -X"

[ util_execSystem ] 141: oal_initIp6FirewallObj cmd is "
    ip6tables -P INPUT ACCEPT"

[ util_execSystem ] 141: oal_initIp6FirewallObj cmd is "
    ip6tables -P FORWARD DROP"

[ util_execSystem ] 141: oal_initIp6FirewallObj cmd is "
    ip6tables -P OUTPUT ACCEPT"

[ util_execSystem ] 141: oal_initIp6FirewallObj cmd is "
    ip6tables -N FIREWALL"

[ util_execSystem ] 141: oal_initIp6FirewallObj cmd is "
    ip6tables -N FWRULE"

[ util_execSystem ] 141: oal_initIp6FirewallObj cmd is "
```

```
    ip6tables -N SETMSS"

[ util_execSystem ] 141:  oal_initIp6FirewallObj cmd is "
    ip6tables -A INPUT -i lo -p ALL -j ACCEPT -m comment
                                    --comment "loop back""

[ util_execSystem ] 141:  oal_initIp6FirewallObj cmd is "
    ip6tables -A INPUT  -m conntrack --ctstate RELATED,
    ESTABLISHED -j ACCEPT"

[ util_execSystem ] 141:  oal_initIp6FirewallObj cmd is "
    ip6tables -A INPUT -i br+ -p tcp --dport 23 -j ACCEPT"

[ util_execSystem ] 141:  oal_initIp6FirewallObj cmd is "
    ip6tables -A INPUT -p tcp --dport 23 -j DROP"

[ util_execSystem ] 141:  oal_initIp6FirewallObj cmd is "
    ip6tables -A INPUT -i br+ -p tcp --dport 22 -j ACCEPT"

[ util_execSystem ] 141:  oal_initIp6FirewallObj cmd is "
    ip6tables -A INPUT -p tcp --dport 22 -j DROP"

[ util_execSystem ] 141:  oal_initIp6FirewallObj cmd is "
    ip6tables -A INPUT -i br+ -p icmpv6 --icmpv6-type echo-
    request -j ACCEPT"

[ util_execSystem ] 141:  oal_initIp6FirewallObj cmd is "
    ip6tables -A INPUT -p icmpv6 --icmpv6-type echo-request -j
    DROP"

[ util_execSystem ] 141:  oal_initIp6FirewallObj cmd is "
    ip6tables -A FORWARD -i br+ -m conntrack --ctstate RELATED,
    ESTABLISHED -j ACCEPT"

[ util_execSystem ] 141:  oal_initIp6FirewallObj cmd is "
    ip6tables -A FORWARD -o br+ -m conntrack --ctstate RELATED,
    ESTABLISHED -j ACCEPT"

[ util_execSystem ] 141:  oal_initIp6FirewallObj cmd is "
    ip6tables -A FORWARD -j FIREWALL"

[ util_execSystem ] 141:  oal_initIp6FirewallObj cmd is "
    ip6tables -I FORWARD 1 -j SETMSS"

[ util_execSystem ] 141:  oal_fw6_setFwEnabeld cmd is "ip6tables
     -D FIREWALL -j ACCEPT"

ip6tables: Bad rule (does a matching rule exist in that chain?).
[ util_execSystem ] 141:  oal_fw6_setFwEnabeld cmd is "ip6tables
     -F FIREWALL"

[ util_execSystem ] 141:  oal_fw6_setFwEnabeld cmd is "ip6tables
     -A FIREWALL -j ACCEPT"
```

```
[ rsl_initWanL2tpConnObj ] 245: L2TP Connection(ewan_l2tp) is
   not enable.

[ rsl_initWanL2tpConnObj ] 245: L2TP Connection() is not enable.

[ rsl_initWanPptpConnObj ] 239: PPTP Connection(ewan_pptp) is
   not enable.

[ rsl_initWanPptpConnObj ] 239: PPTP Connection() is not enable.

[ util_execSystem ] 141:  setupModules cmd is "insmod /lib/
   modules/kmdir/kernel/net/netfilter/nf_conntrack_ftp.ko"

[ util_execSystem ] 141:  setupModules cmd is "insmod /lib/
   modules/kmdir/kernel/net/ipv4/netfilter/nf_nat_ftp.ko"

[ util_execSystem ] 141:  oal_openAlg cmd is "iptables -D
   FORWARD_VPN_PASSTHROUGH -p udp --dport 500 -j DROP"

iptables: Bad rule (does a matching rule exist in that chain?).
[ util_execSystem ] 141:  setupModules cmd is "insmod /lib/
   modules/kmdir/kernel/net/ipv4/netfilter/nf_nat_proto_gre.ko"

[ util_execSystem ] 141:  setupModules cmd is "insmod /lib/
   modules/kmdir/kernel/net/ipv4/netfilter/nf_nat_pptp.ko"

[ util_execSystem ] 141:  oal_openAlg cmd is "iptables -D
   FORWARD_VPN_PASSTHROUGH -p tcp --dport 1723 -j DROP"

iptables: Bad rule (does a matching rule exist in that chain?).
[ util_execSystem ] 141:  oal_openAlg cmd is "iptables -D
   FORWARD_VPN_PASSTHROUGH -p udp --dport 1701 -j DROP"

iptables: Bad rule (does a matching rule exist in that chain?).
[ util_execSystem ] 141:  setupModules cmd is "insmod /lib/
   modules/kmdir/kernel/net/netfilter/nf_conntrack_tftp.ko"

[ util_execSystem ] 141:  setupModules cmd is "insmod /lib/
   modules/kmdir/kernel/net/ipv4/netfilter/nf_nat_tftp.ko"

[ util_execSystem ] 141:  setupModules cmd is "insmod /lib/
   modules/kmdir/kernel/net/netfilter/nf_conntrack_h323.ko"

[ util_execSystem ] 141:  setupModules cmd is "insmod /lib/
   modules/kmdir/kernel/net/ipv4/netfilter/nf_nat_h323.ko"

[ util_execSystem ] 141:  setupModules cmd is "insmod /lib/
   modules/kmdir/kernel/net/netfilter/nf_conntrack_sip.ko"

[ util_execSystem ] 141:  setupModules cmd is "insmod /lib/
   modules/kmdir/kernel/net/ipv4/netfilter/nf_nat_sip.ko"

[ util_execSystem ] 141:  setupModules cmd is "insmod /lib/
   modules/kmdir/kernel/net/netfilter/nf_conntrack_rtsp.ko"
```

```
[ util_execSystem ] 141:  setupModules cmd is "insmod /lib/
    modules/kmdir/kernel/net/ipv4/netfilter/nf_nat_rtsp.ko"

nf_nat_rtsp v0.6.21 loading
enable switch phyport...
Set: phy[0].reg[0] = 3900
Set: phy[1].reg[0] = 3900
Set: phy[2].reg[0] = 3900
Set: phy[3].reg[0] = 3900
Set: phy[4].reg[0] = 3900
[cmd_dutInit():1094] init shm
[tddp_taskEntry():151] tddp task start
Set: phy[0].reg[0] = 3300
Set: phy[1].reg[0] = 3300
Set: phy[2].reg[0] = 3300
Set: phy[3].reg[0] = 3300
Set: phy[4].reg[0] = 3300
resetMiiPortV over.
Set: phy[0].reg[4] = 01e1
Set: phy[0].reg[0] = 3300
Set: phy[1].reg[4] = 01e1
Set: phy[1].reg[0] = 3300
Set: phy[2].reg[4] = 01e1
Set: phy[2].reg[0] = 3300
Set: phy[3].reg[4] = 01e1
Set: phy[3].reg[0] = 3300
Set: phy[4].reg[4] = 01e1
Set: phy[4].reg[0] = 3300
turn off flow control over.
[ util_execSystem ] 141:  prepareDropbear cmd is "dropbearkey -t
     rsa -f /var/tmp/dropbear/dropbear_rsa_host_key"

Will output 1024 bit rsa secret key to '/var/tmp/dropbear/
    dropbear_rsa_host_key'
Generating key, this may take a while...
[ util_execSystem ] 141:  prepareDropbear cmd is "dropbearkey -t
     dss -f /var/tmp/dropbear/dropbear_dss_host_key"

Will output 1024 bit dss secret key to '/var/tmp/dropbear/
    dropbear_dss_host_key'
Generating key, this may take a while...
start ntp_request
[ oal_sys_getOldTZInfo ] 592:  Open TZ file error!
[ util_execSystem ] 141:  oal_sys_unsetTZ cmd is "echo "" > /etc
    /TZ"

[ util_execSystem ] 141:  prepareDropbear cmd is "dropbear -p 22
     -r /var/tmp/dropbear/dropbear_rsa_host_key -d /var/tmp/
    dropbear/dropbear_dss_host_key -A /var/tmp/dropbear/
    dropbearpwd"

[ ntp_start ] 504:  ntp connect failed, return.
```

```
[ util_execSystem ] 141:  oal_sys_unsetTZ cmd is "echo "" > /etc
    /TZ"

Get SNTP start config
start ntp_request
[ util_execSystem ] 141:  oal_sys_unsetTZ cmd is "echo "" > /etc
    /TZ"

[ util_execSystem ] 141:  oal_sys_unsetTZ cmd is "echo "" > /etc
    /TZ"

[ ntp_start ] 504:  ntp connect failed, return.

[ util_execSystem ] 141:  oal_sys_unsetTZ cmd is "echo "" > /etc
    /TZ"

Get SNTP start config
start ntp_request
[ util_execSystem ] 141:  oal_sys_unsetTZ cmd is "echo "" > /etc
    /TZ"

[ util_execSystem ] 141:  oal_sys_unsetTZ cmd is "echo "" > /etc
    /TZ"
```

A.2 iwlist scan results

```
Cell 05 - Address: 60:A4:B7:05:B4:88
                    Channel:2
                    Frequency:2.417 GHz (Channel 2)
                    Quality=69/70  Signal level=-41 dBm
                    Encryption key:on
                    ESSID:"TP-Link_B488"
                    Bit Rates:1 Mb/s; 2 Mb/s; 5.5 Mb/s; 11 Mb/s;
                         9 Mb/s
                               18 Mb/s; 36 Mb/s; 54 Mb/s
                    Bit Rates:6 Mb/s; 12 Mb/s; 24 Mb/s; 48 Mb/s
                    Mode:Master
                    Extra:tsf=000000006e22fb98
                    Extra: Last beacon: 7036ms ago
                    IE: Unknown: 000C54502D4C696E6B5F42343838
                    IE: Unknown: 010882848B961224486C
                    IE: Unknown: 030102
                    IE: Unknown: 2A0104
                    IE: Unknown: 32040C183060
                    IE: Unknown: 2
                        D1A6E1017FFFF0000010000000000000000000000000000000000

                    IE: Unknown: 3
                        D160205000000000000000000000000000000000000000

                    IE: IEEE 802.11i/WPA2 Version 1
                        Group Cipher : CCMP
                        Pairwise Ciphers (1) : CCMP
```

```
    Authentication Suites (1) : PSK
IE: Unknown: 7F090000000000000000000
IE: Unknown: 0B05000000127A
IE: Unknown:
    DD180050F2020101000003A4000027A4000042435E0062322F00

IE: Unknown: 4
    A0E14000A002C01C800140005001900
IE: Unknown:
    DD8D0050F204104A0001101044000102103B000103104700103883309230921

IE: Unknown: DD07000C4300000000
```

A.3 TCP portscan results

```
# Nmap 7.80 scan initiated Mon Oct 11 14:25:48 2021 as: nmap -sV
    -sC -oA scan -p1-65535 192.168.0.1
Nmap scan report for _gateway (192.168.0.1)
Host is up (0.0059s latency).
Not shown: 65531 closed ports
PORT      STATE SERVICE VERSION
22/tcp    open  ssh     Dropbear sshd 2012.55 (protocol 2.0)
| ssh-hostkey:
|   1024 0b:b6:8a:8d:d1:d0:98:0d:90:a4:a0:66:5d:95:81:06 (DSA)
|_  1040 b3:94:8a:f8:d3:7f:6d:d4:18:46:a5:89:fe:59:04:05 (RSA)
53/tcp    open  domain  (unknown banner: UNKNOWN)
| dns-nsid:
|   NSID: rec-inc-pv013-1 (7265632d696e632d70763031332d31)
|   id.server: rec-inc-pv013-1
|_  bind.version: UNKNOWN
| fingerprint-strings:
|   DNSVersionBindReqTCP:
|     version
|     bind
|_    UNKNOWN
80/tcp    open  http
| fingerprint-strings:
|   GetRequest:
|     HTTP/1.1 200 OK
|     Content-Type: text/html; charset=utf-8
|     Content-Length: 24304
|     Set-Cookie: JSESSIONID=deleted; Expires=Thu, 01 Jan 1970
    00:00:01 GMT; Path=/; HttpOnly
|     Connection: keep-alive
|     <!DOCTYPE html>
|     <html xmlns="http://www.w3.org/1999/xhtml">
|     <head>
|     <META http-equiv=Content-Type content="text/html; charset=
    utf-8" />
|     <META http-equiv=Pragma content=no-cache>
|     <META http-equiv=Expires content=0>
|     <!--
|     <link rel="stylesheet" href="../css/login.css" type="text/
```

```
    css" />
|       <link rel="stylesheet" href="../img/login/login.css" type
    ="text/css" />
|       <link rel="Shortcut Icon" href="../img/login/favicon.ico"
    type="image/jpeg" />
|       <style type="text/css">
|       body{
|       font-family:Arial, sans-serief;
|       background-color:#FFFFFF;
|       margin:0px;
|       padding:0px;
|       div.loginBox
|       display: block;
|       position:relative;
|       margin-top:10%;
|       text-align:center;
|     HTTPOptions, RTSPRequest:
|       HTTP/1.1 405 Method Not Allowed
|       Content-Type: text/html; charset=utf-8
|       Content-Length: 124
|       Set-Cookie: JSESSIONID=deleted; Expires=Thu, 01 Jan 1970
    00:00:01 GMT; Path=/; HttpOnly
|       Connection: close
|_      <html><head><title>405 Method Not Allowed</title></head><
    body><center><h1>405 Method Not Allowed</h1></center></body
    ></html>
|_http-title: Site doesn't have a title (text/html; charset=utf
    -8).
1900/tcp open  upnp     Portable SDK for UPnP devices 1.6.19 (
    Linux 2.6.36; UPnP 1.0)
2 services unrecognized despite returning data. If you know the
    service/version, please submit the following fingerprints at
    https://nmap.org/cgi-bin/submit.cgi?new-service :
==============NEXT SERVICE FINGERPRINT (SUBMIT INDIVIDUALLY)
    ==============
SF-Port53-TCP:V=7.80%I=7%D=10/11%Time=61642D64%P=x86_64-pc-linux
    -gnu%r(DNS
SF:VersionBindReqTCP,34,"\x002\0\x06\x85\x80\0\x01\0\x01
    \0\0\0\0\x07versio
SF:n\x04bind\0\0\x10\0\x03\xc0\x0c\0\x10\0\x03\0\0\0\0\0\x08\
    x07UNKNOWN");
==============NEXT SERVICE FINGERPRINT (SUBMIT INDIVIDUALLY)
    ==============
SF-Port80-TCP:V=7.80%I=7%D=10/11%Time=61642D5F%P=x86_64-pc-linux
    -gnu%r(Get
SF:Request,5FB3,"HTTP/1\.1\x20200\x20OK\r\nContent-Type:\x20text
    /html;\x20
SF:charset=utf-8\r\nContent-Length:\x2024304\r\nSet-Cookie:\
    x20JSESSIONID=
SF:deleted;\x20Expires=Thu,\x2001\x20Jan\x201970\x2000:00:01\
    x20GMT;\x20Pa
SF:th=/;\x20HttpOnly\r\nConnection:\x20keep-alive\r\n\r\n\xef\
    xbb\xbf<!DOC
SF:TYPE\x20html>\x20\r\n<html\x20xmlns=\"http://www\.w3\.org
```

```
    /1999/xhtml\">
SF:\r\n\r\n<head>\r\n<META\x20http-equiv=Content-Type\x20content
    =\"text/ht
SF:ml;\x20charset=utf-8\"\x20/>\r\n<META\x20http-equiv=Pragma\
    x20content=n
SF:o-cache>\r\n<META\x20http-equiv=Expires\x20content=0>\r\n\r\n
    <!--\x20\r
SF:\n<link\x20rel=\"stylesheet\"\x20href=\"\.\./css/login\.css
    \"\x20type=\
SF:"text/css\"\x20/>\r\n<link\x20rel=\"stylesheet\"\x20href
    =\"\.\./img/log
SF:in/login\.css\"\x20type=\"text/css\"\x20/>\r\n-->\r\n<link\
    x20rel=\"Sho
SF:rtcut\x20Icon\"\x20href=\"\.\./img/login/favicon\.ico\"\
    x20type=\"image
SF:/jpeg\"\x20/>\r\n<style\x20type=\"text/css\">\r\nbody{\r\n\
    x20\x20\x20\
SF:x20font-family:Arial,\x20sans-serief;\r\n\x20\x20\x20\
    x20background-col
SF:or:#FFFFFF;\r\n\x20\x20\x20\x20margin:0px;\r\n\x20\x20\x20\
    x20padding:0
SF:px;\r\n}\r\ndiv\.loginBox\r\n{\r\n\x20\x20\x20\x20display:\
    x20block;\r\
SF:n\x20\x20\x20\x20position:relative;\r\n\x20\x20\x20\x20margin
    -top:10%;\
SF:r\n\x20\x20\x20\x20text-align:center;\r\n}")%r(HTTPOptions
    ,148,"HTTP/1\
SF:.1\x20405\x20Method\x20Not\x20Allowed\r\nContent-Type:\
    x20text/html;\x2
SF:0charset=utf-8\r\nContent-Length:\x20124\r\nSet-Cookie:\
    x20JSESSIONID=d
SF:eleted;\x20Expires=Thu,\x2001\x20Jan\x201970\x2000:00:01\
    x20GMT;\x20Pat
SF:h=/;\x20HttpOnly\r\nConnection:\x20close\r\n\r\n<html><head><
    title>405\
SF:x20Method\x20Not\x20Allowed</title></head><body><center><h1
    >405\x20Meth
SF:od\x20Not\x20Allowed</h1></center></body></html>")%r(
    RTSPRequest,148,"H
SF:TTP/1\.1\x20405\x20Method\x20Not\x20Allowed\r\nContent-Type:\
    x20text/ht
SF:ml;\x20charset=utf-8\r\nContent-Length:\x20124\r\nSet-Cookie
    :\x20JSESSI
SF:ONID=deleted;\x20Expires=Thu,\x2001\x20Jan\x201970\x2000
    :00:01\x20GMT;\
SF:x20Path=/;\x20HttpOnly\r\nConnection:\x20close\r\n\r\n<html><
    head><titl
SF:e>405\x20Method\x20Not\x20Allowed</title></head><body><center
    ><h1>405\x
SF:20Method\x20Not\x20Allowed</h1></center></body></html>");
MAC Address: 60:A4:B7:05:B4:88 (Unknown)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel, cpe:/o:
    linux:linux_kernel:2.6.36
```

```
Service detection performed. Please report any incorrect results
     at https://nmap.org/submit/ .
# Nmap done at Mon Oct 11 14:27:21 2021 -- 1 IP address (1 host
    up) scanned in 92.50 seconds
```

A.4 UDP portscan results

```
# Nmap 7.80 scan initiated Mon Oct 11 15:18:59 2021 as: nmap -sV
     -sC -oA scan-udp -sU 192.168.0.1
Nmap scan report for _gateway (192.168.0.1)
Host is up (0.0040s latency).
Not shown: 997 closed ports
PORT     STATE         SERVICE VERSION
53/udp   open          domain  (unknown banner: UNKNOWN)
| dns-nsid:
|   NSID: rec-inc-pv013-1 (7265632d696e632d70763031332d31)
|   id.server: rec-inc-pv013-1
|_  bind.version: UNKNOWN
|_dns-recursion: Recursion appears to be enabled
| fingerprint-strings:
|   DNSVersionBindReq:
|     version
|     bind
|     UNKNOWN
|   NBTStat:
|_    CKAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
67/udp   open|filtered dhcps
1900/udp open          upnp?
| upnp-info:
| 192.168.0.1
|     Server: Linux/2.6.36, UPnP/1.0, Portable SDK for UPnP
    devices/1.6.19
|_    Location: http://192.168.0.1:1900/gatedesc.xml
1 service unrecognized despite returning data. If you know the
    service/version, please submit the following fingerprint at
    https://nmap.org/cgi-bin/submit.cgi?new-service :
SF-Port53-UDP:V=7.80%I=7%D=10/11%Time=61643E08%P=x86_64-pc-linux
     -gnu%r(DNS
SF:VersionBindReq,32,"\0\x06\x85\x80\0\x01\0\x01\0\0\0\0\
     x07version\x04bin
SF:d\0\0\x10\0\x03\xc0\x0c\0\x10\0\x03\0\0\0\0\0\x08\x07UNKNOWN
     ")%r(NBTSta
SF:t,32,"\x80\xf0\x80\x90\0\x01\0\0\0\0\0\0\
    x20CKAAAAAAAAAAAAAAAAAAAAAAAAAA
SF:AAAAA\0\0!\0\x01");
MAC Address: 60:A4:B7:05:B4:88 (Unknown)

Service detection performed. Please report any incorrect results
     at https://nmap.org/submit/ .
# Nmap done at Mon Oct 11 15:39:13 2021 -- 1 IP address (1 host
    up) scanned in 1214.43 seconds
```

A.5 Third party dependencies

Table A.1 shows the list of open source dependencies, along with their version number and release date.

Program	Version	Release date
busybox	1.19.2	2011-09-06
Ralink U-Boot	1.1.3 / 4.3.0.0	2005-08-14 / unknown
linux	2.6.36	2010-10-20
uClibc	0.9.33.2	2012-05-15
wireless_tools	29	2007-09-18
dropbear	2012.55	2012-02-22
ebtables	2.0.10-4	2011-12-15
iproute2	2.6.39	2011-06-30
openssl	0.9.8zh	2015-12-03
traceroute	2.0.3	2007-01-09
IGD	unknown	2013-01-07?
wsc_upnp	0.2.2	unknown
iptables	1.4.17	2012-12-25
bpalogin	2.0.2	2003
wide-dhcpv6	20080615	2008-06-15
pppd	2.4.5	2009-11-17
radvd	1.5	2009-09-10
xl2tpd	1.1.12	2007-10-20

Table A.1: Open source dependencies

A.6 List of closed source binaries

- `/lib/libcmm.so`
- `/lib/libcutil.so`
- `/lib/libgdpr.so`
- `/lib/libxml.so`
- `/usr/bin/afcd`
- `/usr/bin/cli`

- /usr/bin/mxdns
- /usr/bin/cos
- /usr/bin/dhcpc
- /usr/bin/dhcpd
- /usr/bin/diagTool
- /usr/bin/dnsProxy
- /usr/bin/dyndns
- /usr/bin/httpd
- /usr/bin/igmpd
- /usr/bin/ipping
- /usr/bin/mldProxy
- /usr/bin/noipdns
- /usr/bin/ntpc
- /usr/bin/pwdog
- /usr/bin/reg
- /usr/bin/snmpd
- /usr/bin/tddp
- /usr/bin/tdpd
- /usr/bin/tmpd
- /usr/bin/wanType

A.7 Config decompression tool

```
#include <stdint.h>
#include <string.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>

typedef unsigned char byte;
typedef uint32_t uint;

typedef struct state{
```

```
    byte *data;
    uint pad;
    uint bits;
    uint bits_left;
} state_s;

// FUN_00019540
bool next_bit(state_s *state) {
  if (state->bits_left == 0x0) {
    state->bits = ((uint)state->data[1] * 0x100 + (uint)state->
        data[0]);
    state->data += 2;
    state->bits_left = 16;
  }
  state->bits_left -= 1;
  state->bits <<= 1;
  return (state->bits & 0x10000) != 0;
}

// FUN_0001959c
int next_block(state_s *state) {
  int result = 1;
  do {
    result = next_bit(state) + result * 2;
  } while (next_bit(state));
  return result;
}

uint cen_uncompressBuff(byte *src,byte *dest,uint len) {
  int block_offset;
  int block_size;
  byte *block_ptr;
  uint count;
  uint src_len;
  state_s state;

  state.bits = 0;
  state.bits_left = 0;

  memcpy(&src_len,src,4);
  if (src_len <= len) {
    count = 0;
    if (src_len != 0) {
      state.data = src + 5;
      *dest++ = src[4];
      count = 1;
      while (count < src_len) {
        if (!next_bit(&state)) {
          count += 1;
          *dest++ = *state.data++;
        } else {
          block_size = next_block(&state) + 2;
          block_offset = (next_block(&state) - 2) << 8;
          block_ptr = dest - (*state.data++ + 1 + block_offset);
```

```
          for(int i = 0; i < block_size; ++i) {
            *dest++ = *block_ptr++;
          }
          count += block_size;
        }
      }
    }
    if (count == src_len) {
      return count;
    }
    printf("Length is not match depackLen = %d\tsrcDataLen = %d"
        ,count, src_len);
    return 0;
  }
  printf("Invalid file or file is too long!\n");
  return 0;
}

#define OUTPUT_LEN 1024*1024
byte output[OUTPUT_LEN];

int main(int argc, char *argv[]) {
    FILE *fp = fopen(argv[1], "rb");
    fseek(fp, 0, SEEK_END);
    long size = ftell(fp);
    fseek(fp, 0, SEEK_SET);

    char *buffer = malloc(size);
    fread(buffer, size, 1, fp);
    fclose(fp);

    printf("Read %ld bytes\n", size);

    uint decomp_length = cen_uncompressBuff(buffer + 16, output,
         OUTPUT_LEN);

    printf("Decompressed %d bytes\n", decomp_length);
    free(buffer);

    fp = fopen(argv[2], "wb");
    fwrite(output, decomp_length, 1, fp);
    fclose(fp);

    return 0;
}
```

A.8 Firmware filesystem compare script

```
from hashlib import sha256
from os import walk
from os.path import join, islink
```

```python
# must be in increasing order of release
VERSIONS = ['190218', '190428', '200623']


def get_file_list(path):
    all_files = []

    for (root, _dirs, files) in walk(path):
        for file in files:
            full_path = join(root, file)
            if not islink(full_path):
                all_files.append(full_path[len(path):])

    return all_files


def get_file_lists(versions):
    file_lists = dict()

    for version in versions:
        file_list = get_file_list(version + '-squashfs-root')
        file_list.sort()
        file_lists[version] = file_list

    return file_lists


def get_file_sha256(path):
    with open(path, 'rb') as f:
        return sha256(f.read()).hexdigest()


def compare_versions(old_version, new_version, file_lists):
    old_files = file_lists[old_version]
    new_files = file_lists[new_version]
    old_index = 0
    new_index = 0
    files_added = []
    files_removed = []
    files_kept = []
    files_modified = []

    print(f'Comparing version {old_version} to {new_version}')
    print(f'- Old version has {len(old_files)} files')
    print(f'- New version has {len(new_files)} files')

    while old_index < len(old_files) and new_index < len(
        new_files):
        old_file = old_files[old_index]
        new_file = new_files[new_index]

        if old_file == new_file:
            old_path = join(old_version + '-squashfs-root',
                old_file[1:])
```

```
            new_path = join(new_version + '-squashfs-root',
                new_file[1:])
            old_hash = get_file_sha256(old_path)
            new_hash = get_file_sha256(new_path)

            if old_hash == new_hash:
                files_kept.append(old_file)
            else:
                files_modified.append(old_file)
            old_index += 1
            new_index += 1
        elif old_file > new_file:
            files_added.append(new_file)
            new_index += 1
        else:
            files_removed.append(old_file)
            old_index += 1

    print(f'- Files added: {len(files_added)}')
    for file in files_added:
        print(f'  - {file}')
    print(f'- Files removed: {len(files_removed)}')
    for file in files_removed:
        print(f'  - {file}')
    print(f'- Files modified: {len(files_modified)}')
    for file in files_modified:
        print(f'  - {file}')
    print(f'- Files kept: {len(files_kept)}\n')


def compare(versions):
    file_lists = get_file_lists(versions)

    for i in range(len(versions) - 1):
        compare_versions(versions[i], versions[i+1], file_lists)


if len(VERSIONS) > 1:
    compare(VERSIONS)
else:
    print('ERROR: need at least two versions to compare')
```