**Embedded Security Systems**

**Laboratory report for List 3**

**Mikołaj Grzegrzółka 241073, Grzegorz Zaborowski 236447**

**List 3: Reverse engineering of the router (in our case):**

Software and equipment used:

* 3 x TP-LINK TL-WR340G
* USB-UART CP2102 adapter
* Multimeter UNI-T UT33B+
* Electronic soldering station with soldering iron
* Laptop
* PuTTY
* Ghidra
* VxHunter
* THC Hydra

According to the instructions we were supposed to reverse engineer chosen device and perform at least:

1. Information and data gathering,
2. UART detection, determine voltage levels and pins order, establish the connection,
3. Get firmware from the board or download it from manufacturer,
4. Analyse firmware, extract it, get access to the essential resources of the device,
5. Retrieve root password or get access to the root shell on the device.

**Device selection:**

The first step was to choose the right device. Our choice fell on the router as it is the most used device in beginner's guides on how to break into an embedded system. In addition, after a quick look at our junk with old things, we quickly found an old TP Link router, which additionally confirmed our belief that it is a good choice, and it will be easier to break into such an old device than some new models.

**Data gathering:**

First, we started looking for information about hacking our specific device , but we didn't find anything specific. We managed to find only a few information on some Russian internet forums that someone had some problems with device and asked for advice on downloading the firmware and uploading it via the UART port. In the end, it came down to uploading the firmware via the memory programmer, which did not help us with anything. The only thing we managed to get there was a picture with the UART pins described. Then we tried to identify most of the important circuits on the printed circuit board and verify that the picture with the described UART pins is the same as ours. Using FCC ID we found some useful information, such as the components used, photos of the printed circuit board and a few others, but nowhere has the UART interface and pin description been marked.

Board elements (figure 1):

1. DC in – power input 9V 0.6A
2. Reset Button
3. Atheros AR2317-AC1A – a single chip MAC/Baseband/Radio and processor for 2.4 GHz wireless LANs
4. Marvell 88E6060 – a single chip integration of a complete 6-port Fast Ethernet switch with support for a CPU connection
5. EtronTech EM638165TS-6G – a high-speed CMOS synchronous DRAM containing 64 Mbits. It is internally configured as 4 Banks of 1M word x 16 DRAM with a synchronous interface
6. GROUP-TEK HST-2027DR – 10/100BASE-TX transformer module for auto MDI/MDIX applications (two slots)
7. GROUP-TEK HST-1025DR – 10/100BASE-TX transformer module for auto MDI/MDIX applications (one slot)
8. UART interface
9. Macronix 25L1605AM2C – 16Mb flash memory
10. BT B772D Pj99F – NPN transistor

Obraz zawierający tekst, sprzęt elektroniczny, obwód

Opis wygenerowany automatycznie

Figure 1: PCB

According to the information contained in the processor documentation, the PCB also includes the JTAG and SPI interface, but we were not able to identify (and confirm) them.

**Preparation of the device:**

After locating the UART interface, we still had to identify the specific pins (GDN, Vcc, Tx, Rx). To do this, we used a multimeter and made some measurements (table 1 and 2). The measurement takes place between the GDN (on the reverse side of DC in) and the individual pins.

|  |  |  |  |
| --- | --- | --- | --- |
| Pin 1 | Pin 2 | Pin 3 | Pin 4 |
| 5.15 kΩ | 0 Ω | 55 kΩ | 56 kΩ |

Table: 1

|  |  |  |  |
| --- | --- | --- | --- |
| Pin 1 | Pin 2 | Pin 3 | Pin 4 |
| 3.33 V | 0 V | 0 V | 3.33 V |

Table: 2

The above measurements lead us to assume that pin 1 is Vcc and pin 2 is GND. To check and distinguish Tx from Rx, an additional voltage measurement was performed during device boot for pins 3 and 4. For pin 3 we got 0 V and for pin 4 we got 0.4 V-3.33 V. The conclusion from this measurement is that pin 4 is Tx and pin 3 is Rx. In order to confirm the pin identification, we had to connect the UART adapter. Due to the fact that the UART interface was soldered, we had to solder the golden pins ourselves.

This is where the first problems appeared. Due to our poor soldering skills and the fact that the tin used was terribly hard and it was difficult to remove it, after more than an hour of fighting and soldering the gold pins, it turned out that not all the pins were soldered correctly, which was tested by re-measuring with a multimeter (see table 1 and 2). After re-unsoldering and re-soldering, it turned out that we are not even able to fully re-solder one input. The reason was probably the melting of the path from pin 3, which made it impossible for us to work for several days until the newly purchased routers were delivered (we bought two of the same to have one extra). We couldn't do anything with the previous router anymore as it was not possible to repair the path with our equipment and skills (the path is too small).

It was also beneficial to buy two routers instead of one, because in the case of the first router it turned out that we get incorrect measurements for two pins (despite correctly soldered pins). We were not able to tell if these inputs were not working on the router from the beginning or if they were damaged by our soldering of the pins, as we had not done any measurements prior to the soldering of the pins so the next router went to the trash. Learned from the previous incident, we measured the inputs before and after soldering, and this time everything turned out to be working properly. After connecting the UART adapter, it was confirmed that we correctly identified the pins.

**Device boot:**

After correctly connecting to the UART interface, we had to set the correct baud rate. This was done by trial and error, using the table of the most popular baud rate values. In our case it was 38400 bit/s. There was little information during the start-up of the device and they were: the name of the processor and its revision, the channel number and frequency of the WiFi network, the date of creation and the company responsible for the software, and some names of configuration variables along with their size. Additionally, at the very end it turned out to be the boot menu with options: 1) download application program, 2) modify bootrom password, 3) exit the menu, 4) reboot. 5) user commond line. All options turned out to be locked except 'user command line' which didn't allow any actions except listing the processes. There was also no typical login to the system or anything like that. At this point, we realized that the embedded system is probably not Unix-like. We discovered that in addition to the options given in the boot menu (1-5), it is also possible to specify other options (`i`,` r`, `p`,` a`, `s`,` d`, `g`,` k `,` n`, `m`,` Q`, `I`,` A`, `B`,` M`), but even they did not provide us with any meaningful information, contained information such as: CLUSTER POOL TABLE, active Internet connections (including servers), task list, free and aloc memory, netstat, etc. Only interesting one was task list (figure 2, but still we had no idea how to use it).

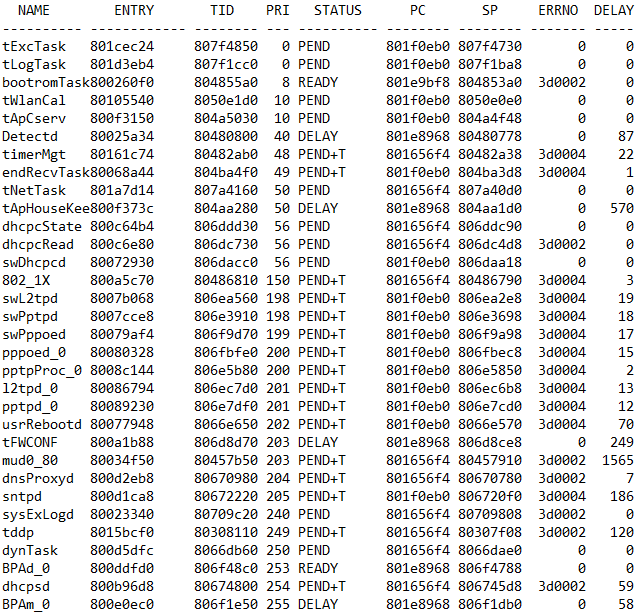


Figure 2: Task list

From the inability to get more information through the UART interface (and firmware download), we moved to the stage of searching and downloading firmware from the Internet.

**What is Binwalk and how does it work?**

It is an open source tool for analysing, reverse engineering and extracting firmware images from devices. It’s main feature is a signature scanning. Binwalk is able to scan a firmware image to search for any different embedded file types and file systems and is also able to extract the files found in the firmware.

**Firmware analysis:**

First, we went to the website of the router manufacturer and downloaded the firmware appropriate for our device and begin to analyse it with Binwalk. First, we check file entropy to if file is encrypted/compressed (figure 3). As we see for the most of bytes entropy is close to zero, which means that that part of file is encrypted or compressed. Also we can use `strings` (linux program) to find some meaningful word, which show us that used operating system is VxWorks 5.5.1 (few first bytes of the file).

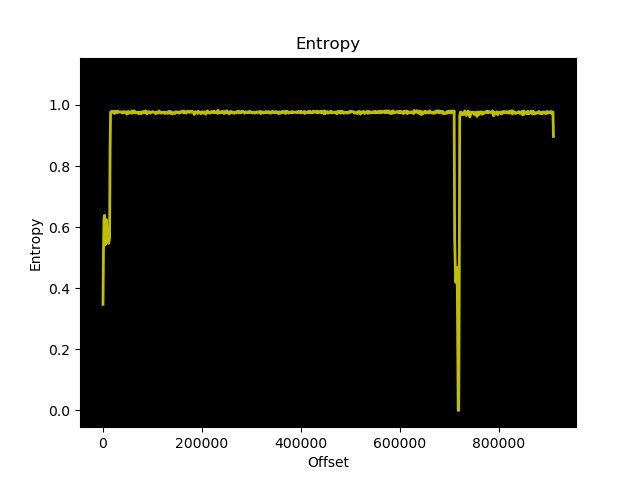


Figure 3: Entropy of firmware file

VxWorks is a proprietary RTOS developed by Wind River Systems. It supports many architectures like Intel, Power-PC, ARM, MIPS, etc. It has its own development suite called Wind River Workbench that includes an editor, compiler toolchain, debugger, and emulator. VxWorks is used by products across a wide range of market areas: aerospace and defence, automotive, industrial such as robots, consumer electronics, medical area and networking. As VXWorks is a proprietary OS, it has many vendor specific attributes, this includes the security model. For example, Stack Overflow Detection for user programs. The firmware structure and file system are also vendor specific.

In addition, we can use the binwalk to try to identify the files contained in the firmware (figure 4). This will find a lot of files that are compressed with the LZMA algorithm.

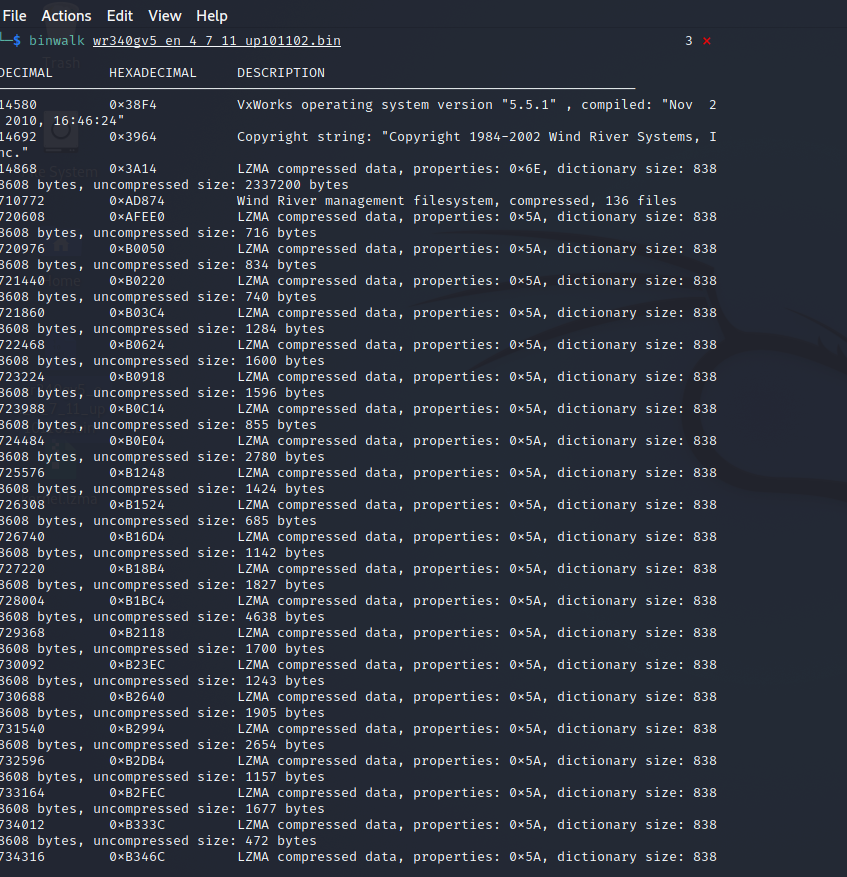


Figure 4: Binwalk found files

After using binwalk to extract and decompress the files, we get 137 files. All files except one (3A14) are web server files such as: html, css, javascript, images (JFIF). Analysing the system file (3A14), we found a lot of server files path: AssignedIpAddrListHelpRpm.htm, &lBackNRestoreHelpRpm.htm, ChangeLoginPwdHelpRpm.htm, DateTimeCfgHelpRpm.htm, +PDdnsAddComexeHelpRpm.htm, DdnsAddHelpRpm.htm, DMZHelpRpm.htm, DomainFilterHelpRpm.htm, DynDdnsHelpRpm.htm, DiagnosticHelpRpm.htm, FireWallHelpRpm.htm, ... . Also some archives and .c files, but nothing what we could understand. Our biggest problem was the lack of knowledge of the system used and a completely different structure than the Unix-like system we are familiar with. During the manual analysis of the system file, we managed to find some interesting lines. The first is:



Apparently, when factory resetting the router, the default username and password combination is wr541/123.

The second is:

Obraz zawierający tekst

Opis wygenerowany automatycznie

Our assumption is that the user `Hello123World` may be some user logged in via the TELNET protocol (we will try to crack the password).

Due to the fact that we did not know what else we could do, we started looking for advice and tools for analysing firmware based on the VxWorks system.

**Ghidra:**

Ghidra is open source software for reverse engineering applications. Ghidra is used to decompile executables so that the recovered code can then be analyzed. This can be used, for example, to analyze malware or the behavior of normal programs.

**VxHunter:**

Firmware analyze ToolSet for VxWorks Based Embedded Device Analyses. The firmware analyze tool are plugins wirtten in Python, mainly used for analyze firmware loading address, fix fuction name with symbol table, etc.

**THC Hydra:**

THC-Hydra is one of the fastest and most reliable programs for checking the security and resistance of systems to brute force password recovery. It can also be used to recover forgotten passwords to any resource that uses a login pair and a password to login to the system. The Hydra supports more than 50 different protocols and call formats as standard. We used this program to launch a password dictionary attack for the user “Hello123World” for the TELNET protocol, but it also failed. We haven't tried any other user/password attacks for TELNET.

**Summary:**

We managed to complete the following tasks: information and data gathering; UART detection, determine voltage levels and pins order, establish the connection; get firmware from the manufacturer site; analyse firmware, extract it, get access to the essential resources of the device (partially). We failed to get root password or to get access to the root shell on the device or modify firmware in malicious way. We encountered a lot of problems while performing these tasks. The first one turned out to be already during the preparation of the device and the identification of the UART interface. We had to use as many as 3 devices, because in the first device we burned a track around one pin, on the second device two pins did not work (we did not check during soldering) and we only managed to do it with the 3rd device. The next problems turned out to be with the boot of device itself, because it turned out that it is not a Unix-like system and there is no login option to the device, and from the boot menu we cannot get any useful information or download the firmware. The biggest difficulty turned out to be the firmware analysis, because it is not a Unix-like system with which we are familiar and its structure may strongly depend on the developer and the device. In addition, the use of programs such as Ghidra and VxHunter also let us down because we were not able to get anything using the standard Ghidra tools, and VxHunter did not want to work for this firmware (possibly some firmware fragments were encrypted). Additionally, we tried to crack the password via TELNET using THC Hydra but it also failed. Finally, we even tried to check whether the device has a popular backdoor a few years ago found in TP Link rooters, which allows the use of the root shell via the website

<http://192.168.0.1/userRpmNatDebugRpm26525557/start_art.html>

but this backdoor is not present in it.

Checking our friends' progress, we found out that everyone else has a Unix-like system, which greatly simplified the analysis of the firmware. We also heard that in some cases, access to the root shell is protected by a blank password. When choosing a device, we thought that having older devices, the security would be weaker, and the newer the device would be more complicated and more secure. It turned out, however, that the new devices are less protected or contain more vulnerabilities than some older devices, which was a big surprise to us.