Embedded Security Systems - Report 2

Bartosz Drzazga, Mateusz Jachniak

August 31, 2021

Contents

1 List 3

1.1 Exercise

You will pick one of the chosen device with embedded system implemented (for instance router, camera, electronic baby sitter, etc.) and try to reverse engineer it. You have to 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. Analyze 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.

Your task is to overcome each security means by using different analysis methods and prepare scripts/toolkits facilitating attacks. You should be able to establish connection between target board and your computer via UART or other interface in order to perform the communication and take control over the device.
Useful commands and tools: busybox, dd, grep, sed, regular expressions, python, bash scripts, pearl, binwalk, strings, u-boot, command line for u-boot, ·

1.2 Information and data gathering

For this task we selected a Linksys WAG120N router. The device is identified by FCC ID Q87-WAG120N and this is our starting point for information gathering. The most useful knowledge for our task we can get from FCC web page are photos of interior with labels of important components and closeup photos of used chips. Later on it turned out that it is far easier to read chip names from those photos rather than from the chips with bare eye. From the photos we know locations of PIFA and PCB antennas, chip names and probable location of UART interface. We include two photos from FCC web page, one with PIFA antenna, and the other with probable UART interface.

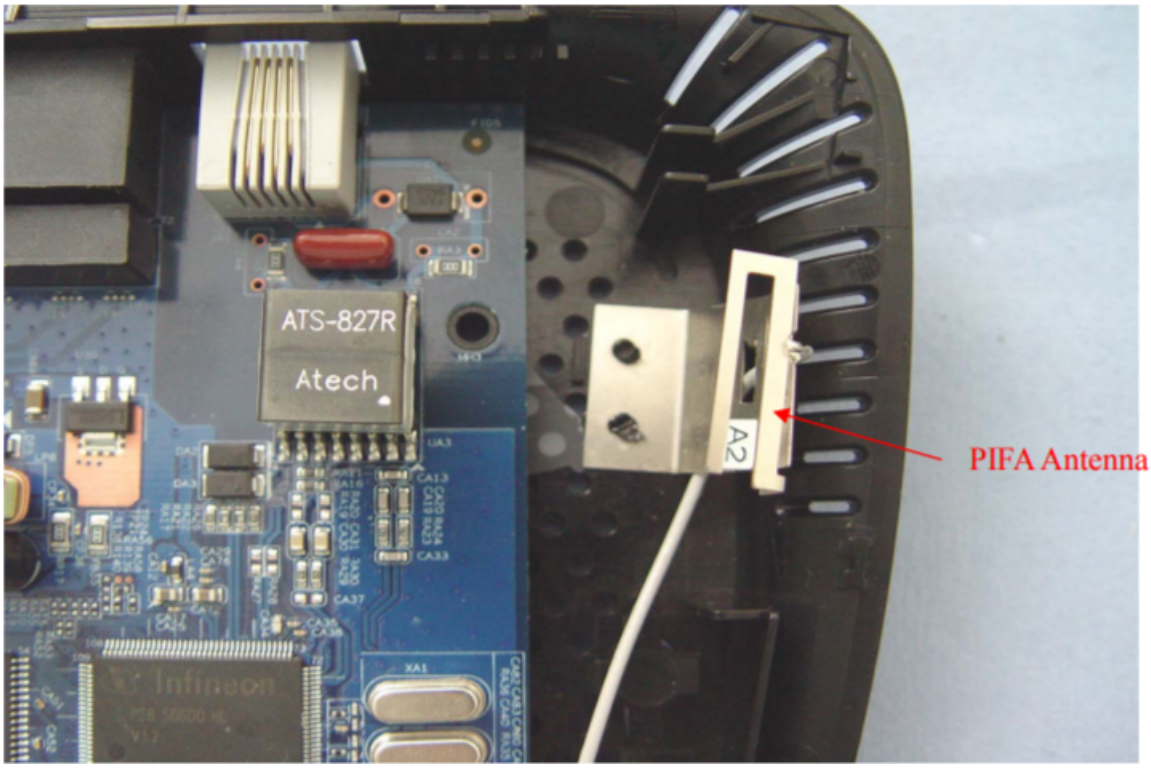

Figure 1: PIFA antenna in Linksys WAG120N, source: FCC

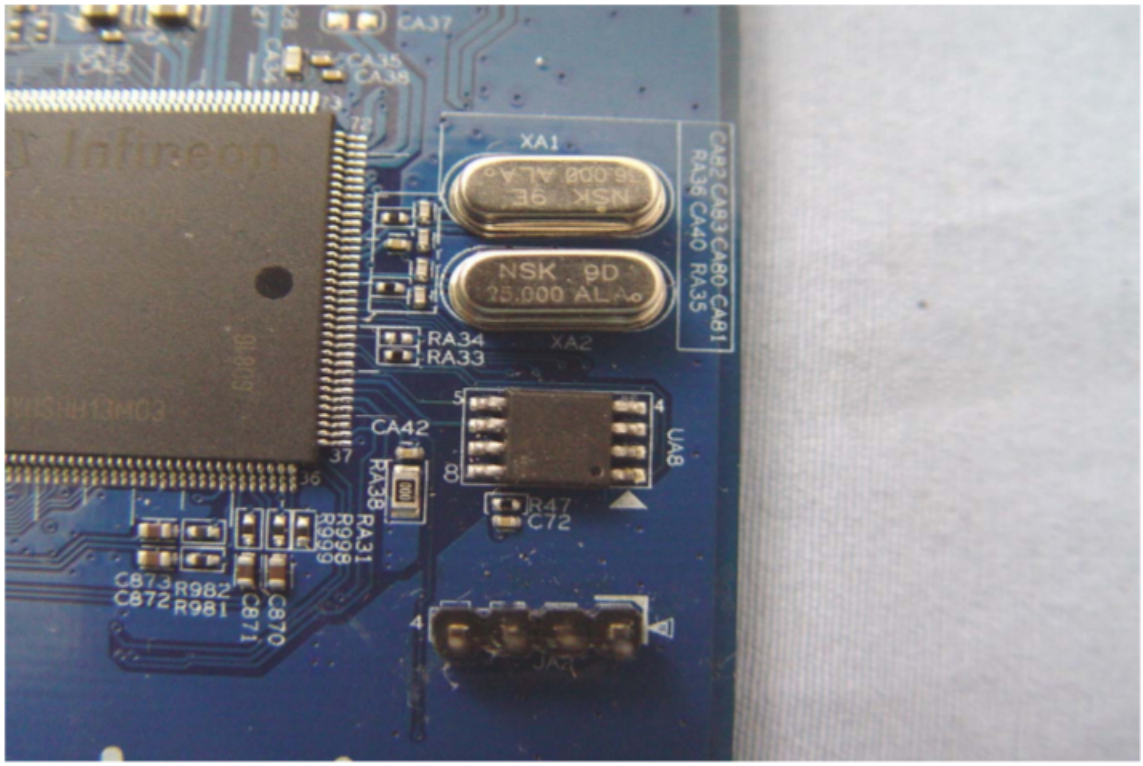

Figure 2: Suspected UART interface of Linksys WAG120N, source: FCC

We also looked up the device on the Internet, one of the most complete information source was TechInfoDepot website, where we found a table with a summary of the device:

Platform	
Brand • Model • Rev	**Linksys** WAG120N
FCC ID	Q87-WAG120N
IC ID	3839A-WAG120N
Type	wireless router, dsl modem
CPU1	**Infineon** PSB 50600
CPU1 Speed	**266 Mhz**
CPU2	**Ralink** RT3050
CPU2 Speed	**320 Mhz**
Flash1 Chip	Brand? Model?
Flash1 Size	**4 MiB**
Flash2 Size	**2 MiB**
RAM1 Size	**32 MiB**
RAM1 Chip	**EtronTech** EM63A165TS-6G
RAM2 Size	**8** MB
RAM2 Chip	**EtronTech** EM638165TS-6G
ETH chip1	**Infineon** PSB 50600
Switch	**Infineon** PSB 50600
Ethernet Port Count	4-10/100-LAN
Ethernet interface OUI	none specified
Wireless interface OUI	none specified
Expansion IF types	none specified
Connector type	barrel
Flags	ADSL2+
Other	
Default IP address	192.168.1.1
Default login user	admin
Default login password	admin
Manuf/OEM/ODM	**SerComm**
Retail	
FCC approval date	31 August 2009
Country of manuf	China
Radio 1	
Chip1	**Ralink** RT3050
Wireless interface OUI	none specified
Antenna Connector Type	none specified
MIMO status	**1x1:1**
Wireless Standard	**IEEE 802.11b/g/n**

Figure 3: Technical details of Linksys WAG120N, source: TechInfoDepot

We can gather more information by connecting the router to the computer with the help of the UART interface (we cover establishment of connection in the next chapter).

Over UART we can observe the boot sequence log which displays a lot of useful information. Below we present only some of the messages:

```
ROM VER: 1.2.0
```

```
CFG 04
EEPROM Data OK



U-Boot 1.1.5-2.0 (Jul 22 2009 - 14:05:20)
```

```
## Booting image at 00030000 ...
   Image Name:   MIPS Linux-2.4.31-Amazon_SE-3.6.
   Created:      2009-07-31  20:22:18 UTC
   Image Type:   MIPS Linux Kernel Image (lzma compressed)
   Data Size:    577500 Bytes = 564 kB
```

We can also observe IP and MAC addresses and software used:

- Linux version 2.4.31-Amazon_SE-3.6.10.4.patch.3-R0416V36_BSP_SPI_FLASH_A4 (root@localhost) (gcc version 3.3.6) #6 Sat Aug 1 05:22:10 CHOT 2009
- Squashfs 2.2 (released 2005/07/03) (C) 2002-2004, 2005 Phillip Lougher
- BusyBox v1.00 (2009.07.31-19:39+0000) multi-call binary

1.3 UART detection, determine voltage levels and pins order, establish the connection

In order to get inside of the device we unscrew 4 Phillips screws. One of the screws is protected with a warranty sticker so it is impossible to unscrew that one screw without voiding the warranty and without leaving traces.

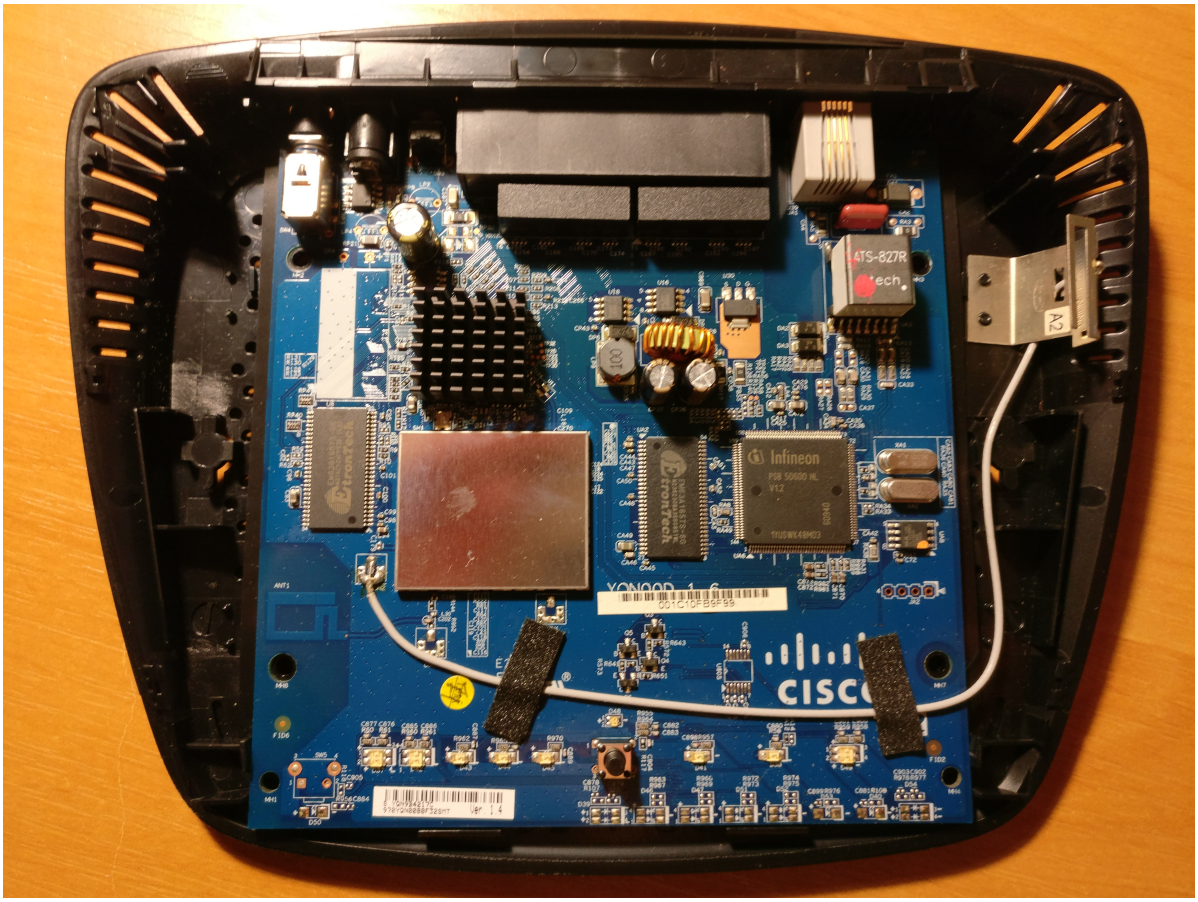

Figure 4: PCB of the device

Immediately, we can locate the same components as in the photos from FCC, including the candidate for UART connector. The only difference is that the final product is manufactured without 2mm femal connectors. That should not be a problem for us, we can just stick male connectors into the holes in PCB.

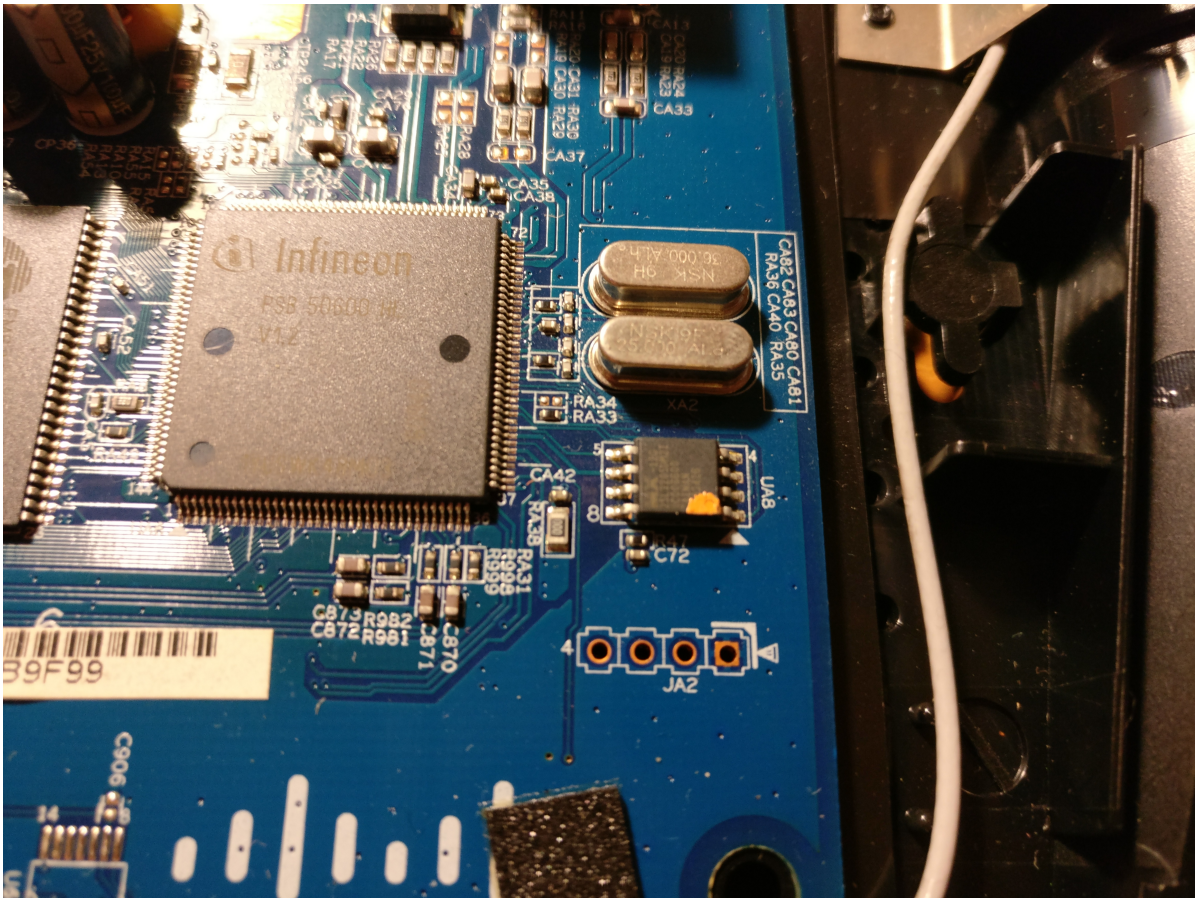

Figure 5: UART interface on the PCB

The problem is that the respective pins are unlabeled. We will try to identify them by measurements. We know the location of PIFA antenna and we will use its ground plane as ground in the process of identification.

We start be measuring electrical resistance between each of 4 pins and the ground. The only pin with 0 Ω is pin 1, thus we are sure this is GND.

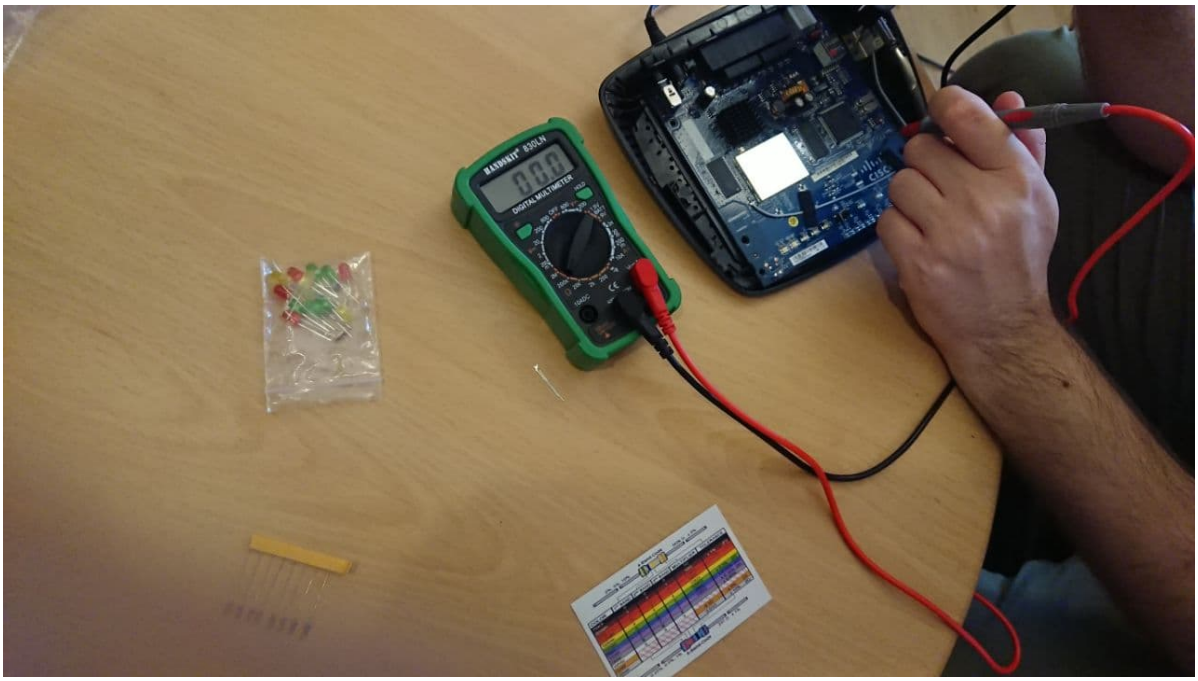

Figure 6: Electrical resistance between pin 1 and ground

Then, we try to find VCC, RX and TX. We do this by measuring voltage on the rest of 3 pins. The 4th pin is at 0 V so this has to be RX. On pin 3 we measured rather stable $3.2 - 3.3V$ and on pin 2 we measured a lot of fluctuations in voltage level. We conclude that pin 2 is TX, pin 3 is VCC and pin 4 is RX.

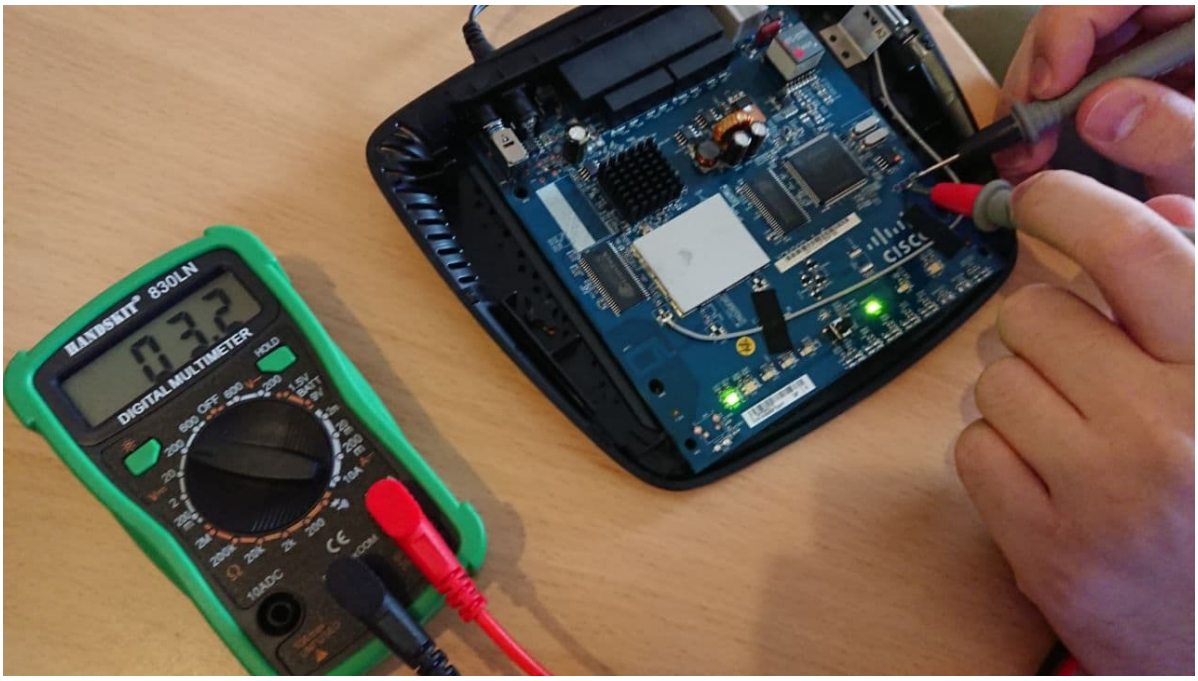

Figure 7: Voltage of VCC

We have identified all 4 pins of UART, that is:

Pin number	Pin role
1	GND
2	TX
3	VCC (3.3 V)
4	RX

and now we are able to connect a PC to the router via a USB-UART converter.
The connection is made in a following way:

- GND to GND
- TX to RX
- RX to TX
- VCC to VCC (but may be disconnected if another source of power is used)

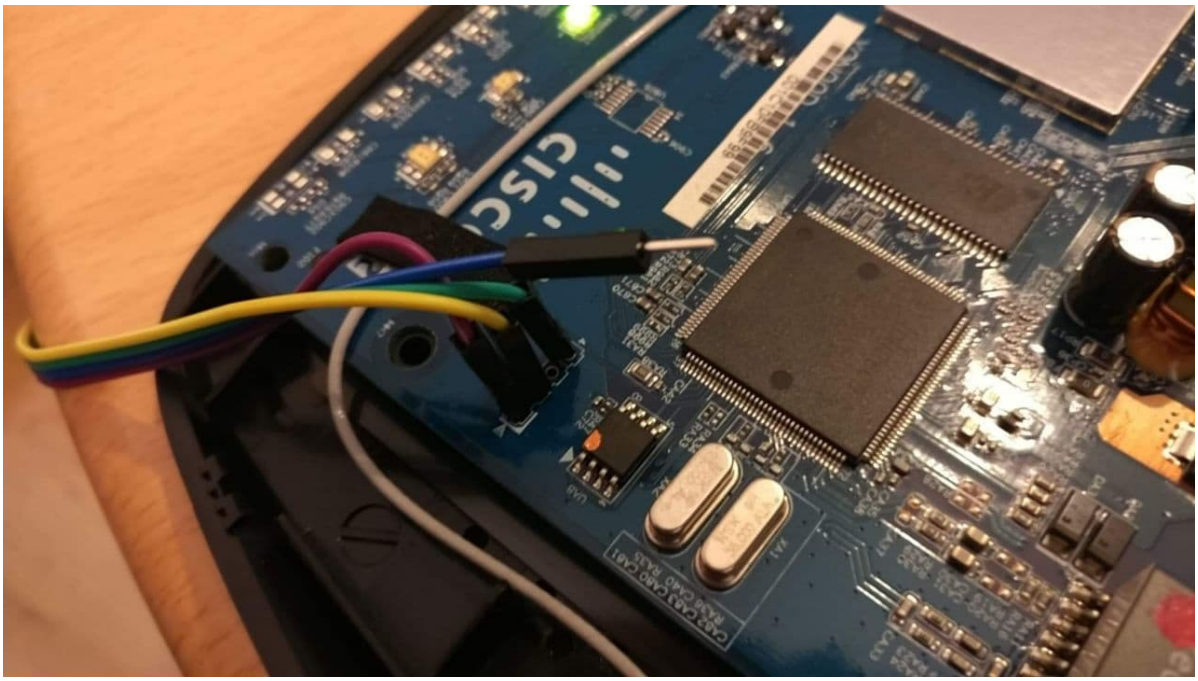

Figure 8: USB-UART connector plugged in router's UART connector

To connect to serial console we use screen. First, we need to locate the converter in PC's host system. The device is available at
`/dev/serial/by-id/usb-Prolific_Technology_Inc._USB-Serial_Controller-if00-port0`
We also need to pass a correct baud rate to screen. We decided to guess and our first try 115200 was correct, we were able to establish connection with command:
`screen /dev/serial/by-id/usb-Prolific_Technology_Inc._USB-Serial_Controller-if00-port0 115200`

1.4 Get firmware from the board or download it from manufacturer

It is possible to extract the firmware from the device over UART. During boot sequence, the bootloader, which is U-Boot in our case, will allow for direct interaction and print out:

```
Type "run flash_nfs" to mount root filesystem over NFS

Hit any key to stop autoboot:  3 ... 2 ... 1 ... 0
## Booting image at 00030000 ...
   Image Name:   MIPS Linux-2.4.31-Amazon_SE-3.6.
   Created:      2009-07-31  20:22:18 UTC
   Image Type:   MIPS Linux Kernel Image (lzma compressed)
   Data Size:    577500 Bytes = 564 kB
   Load Address: 80002000
   Entry Point:  801b2040
   Verifying Checksum ... OK
   Uncompressing Kernel Image ... OK

Starting kernel ...
```

In the above snippet from the boot sequence log we can see we have about 3 seconds to stop autoboot procedure. If we hit any key during that time we would be dumped into U-Boot prompt. This feature is useful as it allows e.g. booting the kernel with custom parameters. For example to bypass the installed init system and drop to a root shell you may use:

```
=> setenv bootargs ${bootargs} init=/bin/bash
=> boot
```

U-Boot also allows to read/write flash/memory, and will dump the ASCII hex to the terminal window. It is possible to parse the hexdump and convert it into actual binary values.

The other approach is just downloading the image from the Internet. The firmware image used in the WAG120N hardware version 1.0 has been downloaded from Linksys Official Support site. Firmware version 1.00.19 (ETSI) Annex A, released on 17/10/2014. Size: 4.0 MB.

1.5 Analyze firmware, extract it, get access to the essential resources of the device

The first thing we have done with a firmware image was running a command strings on it:

```
$ strings -n 10 WAG120N-EU-ANNEXA-ETSI-1.00.20-code.bin

U-Boot 1.1.5-2.0 (Jul 22 2009 - 14:05:28)
Can not read from source buffer
Not enough buffer for decompression
u-boot image
+Rf$I5?wdK
lG{tw*k.xh)
xz|* qnO X}bf
Tc15<lHk3't"
entry download..
%02x:%02x:%02x:%02x:%02x:%02x
getPIDfromFlash
read ok..!!
erase all,keep data from 0x%x to 0x%x
normal erase
erase complete
enter download.
Download mode
...
```

Taking a look at the strings output, we see references to the U-Boot bootloader and the Linux kernel. It suggests that this device does in fact run Linux, and U-Boot.
Next step that we took was running a hexdump command:

```
$ hexdump -C WAG120N-EU-ANNEXA-ETSI-1.00.20-code.bin

00000000  aa 55 ff ff 03 02 01 00  0c 00 05 04 bf 80 00 60  |.U.............`|
00000010  00 00 00 07 bf 80 00 10  00 00 00 00 bf 80 00 20  |..............  |
00000020  00 00 00 00 bf 80 02 00  00 00 00 02 bf 80 02 10  |................|
00000030  00 00 00 00 bf 80 02 30  00 00 00 02 bf 80 02 20  |.......0.......  |
00000040  00 00 00 30 bf 80 02 40  00 00 14 c9 bf 80 02 80  |...0...@........|
00000050  00 03 63 25 bf 80 02 90  00 00 08 1d bf 80 02 a0  |..c%............|
00000060  00 00 00 00 bf 80 02 10  00 00 00 01 bf 10 30 10  |..............0.|
00000070  00 00 00 20 a0 01 00 01  00 00 39 1f 10 00 01 23  |...  ......9....#|
00000080  00 00 00 00 10 00 01 21  00 00 00 00 68 8c 68 8c  |.......!....h.h.|
00000090  00 00 00 00 10 00 01 77  00 00 00 00 10 00 01 75  |.......w.......u|
000000a0  00 00 00 00 10 00 01 73  00 00 00 00 10 00 01 71  |.......s.......q|
000000b0  00 00 00 00 10 00 01 6f  00 00 00 00 10 00 01 6d  |.......o.......m|
```

```
000000c0  00 00 00 00 10 00 01 6b  00 00 00 00 10 00 01 69  |.......k.......i|
000000d0  00 00 00 00 10 00 01 67  00 00 00 00 10 00 01 65  |.......g.......e|
000000e0  00 00 00 00 10 00 01 63  00 00 00 00 10 00 01 61  |.......c.......a|
000000f0  00 00 00 00 10 00 01 5f  00 00 00 00 10 00 01 5d  |........_.......]|
00000100  00 00 00 00 10 00 01 5b  00 00 00 00 10 00 01 59  |.......[.......Y|
00000110  00 00 00 00 10 00 01 57  00 00 00 00 10 00 01 55  |.......W.......U|
00000120  00 00 00 00 10 00 01 53  00 00 00 00 10 00 01 51  |.......S.......Q|
00000130  00 00 00 00 10 00 01 4f  00 00 00 00 10 00 01 4d  |.......O.......M|
00000140  00 00 00 00 10 00 01 4b  00 00 00 00 10 00 01 49  |.......K.......I|
00000150  00 00 00 00 10 00 01 47  00 00 00 00 10 00 01 45  |.......G.......E|
00000160  00 00 00 00 10 00 01 43  00 00 00 00 10 00 01 41  |.......C.......A|

...
```

We can see hexdump does not immediately reveal anything interesting.
Our next step was running binwalk on it:

```
$ binwalk WAG120N-EU-ANNEXA-ETSI-1.00.20-code.bin

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

9660          0x25BC          U-Boot version string, "U-Boot 1.1.5-2.0 (Jul 22 2009 - 14:05:28)"
9708          0x25EC          CRC32 polynomial table, big endian
11012         0x2B04          uImage header, header size: 64 bytes, header CRC: 0xF5170888, created: 2009-07-22 06:05:29, image size: 47540 bytes, Data Address: 0x80400000, Entry Point: 0x80400000, data CRC: 0x84EF8694, OS: Linux, CPU: MIPS, image type: Firmware Image, compression type: lzma, image name: "u-boot image"
11076         0x2B44          LZMA compressed data, properties: 0x5D, dictionary size: 8388608 bytes, uncompressed size: 147212 bytes
65434         0xFF9A          Sercomm firmware signature, version control: 1, download control: 256, hardware ID: "YQN", hardware version: 0x0, firmware version: 0x0, starting code segment: 0x100, code size: 0x7300
72028         0x1195C         Sercomm firmware signature, version control: 29184, download control: 24933, hardware ID: "d ok..!!", hardware version: 0x7266, firmware version: 0x2578, starting code segment: 0xA78, code size: 0x0
196608        0x30000         uImage header, header size: 64 bytes, header CRC: 0x24D867E3, created: 2014-09-10 08:48:23, image size: 577048 bytes, Data Address: 0x80002000, Entry Point: 0x801B2040, data CRC: 0xAA92C548, OS:
```

```
    Linux, CPU: MIPS, image type: OS Kernel Image, compression type: lzma,
    image name: "MIPS Linux-2.4.31-Amazon_SE-3.6.]"
196672          0x30040         LZMA compressed data, properties: 0x5D,
    dictionary size: 8388608 bytes, uncompressed size: 1986560 bytes
851968          0xD0000         Squashfs filesystem, big endian, lzma
    compression, version 2.1, size: 2994562 bytes, 748 inodes, blocksize: 65536
     bytes, created: 2014-09-10 08:49:01
```

Now we have our first clue. Binwalk has found two uImage headers (which is the header format used by U-Boot), each of which is immediately followed by an LZMA compressed file.

Binwalk breaks out most of the information contained in these uImage headers, including their descriptions e.g.: 'u-boot image'. It also shows the reported compression type of 'lzma'. Since each uImage header is followed by LZMA compressed data, this information appears to be legitimate.

The LZMA files can be extracted with dd and then decompressed with the lzma utility. Any trailing garbage should be ignored by lzma during decompression, but we encountered some problems in that step.

```
$ dd if=WAG120N-EU-ANNEXA-ETSI-1.00.20-code.bin bs=1 skip=11076 of=uboot.lzma
4183228+0 records in
4183228+0 records out
4183228 bytes (4,2 MB, 4,0 MiB) copied, 9,4641 s, 442 kB/s
$ lzma -d uboot.lzma
lzma: uboot.lzma: Compressed data is corrupt
$ dd if=WAG120N-EU-ANNEXA-ETSI-1.00.20-code.bin bs=1 skip=196672 of=kernel.
    lzma
3997632+0 records in
3997632+0 records out
3997632 bytes (4,0 MB, 3,8 MiB) copied, 9,06466 s, 441 kB/s
$ lzma -d kernel.lzma
lzma: kernel.lzma: Compressed data is corrupt
```

For some reason lzma utility was claiming the data is corrupted. All of Internet resources we found state, that any trailing data will be ignored. It seems that our version:

```
$ lzma -V
xz (XZ Utils) 5.2.5
liblzma 5.2.5
```

behaves differently. Version 5.2.5 was released on 2020-03-18 while the previous versions are 5.2.4 from 2018 and 5.2.3 from 2016. We suspect there were some changes that none of the online guides describe, but we did not search much as we found a workaround for this problem. We were able to extract all of components of the firmware with binwalk using flags `e` (extract known files) and `M` (scan recursively):

```
$ binwalk -eM WAG120N-EU-ANNEXA-ETSI-1.00.20-code.bin
```

```
Scan Time:     2020-11-11 14:44:40
Target File:   /home/dell/Desktop/studies/ESS/lab3/WAG120N-EU-ANNEXA-ETSI
    -1.00.20-code.bin
MD5 Checksum:  15640dd3b6557068f095f30ca9431725
Signatures:    391

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

9660          0x25BC          U-Boot version string, "U-Boot 1.1.5-2.0 (Jul 22
    2009 - 14:05:28)"
9708          0x25EC          CRC32 polynomial table, big endian
11012         0x2B04          uImage header, header size: 64 bytes, header CRC
    : 0xF5170888, created: 2009-07-22 06:05:29, image size: 47540 bytes, Data
    Address: 0x80400000, Entry Point: 0x80400000, data CRC: 0x84EF8694, OS:
    Linux, CPU: MIPS, image type: Firmware Image, compression type: lzma, image
     name: "u-boot image"
11076         0x2B44          LZMA compressed data, properties: 0x5D,
    dictionary size: 8388608 bytes, uncompressed size: 147212 bytes
65434         0xFF9A          Sercomm firmware signature, version control: 1,
    download control: 256, hardware ID: "YQN", hardware version: 0x0, firmware
    version: 0x0, starting code segment: 0x100, code size: 0x7300
72028         0x1195C         Sercomm firmware signature, version control:
    29184, download control: 24933, hardware ID: "d ok..!!", hardware version:
    0x7266, firmware version: 0x2578, starting code segment: 0xA78, code size:
    0x0
196608        0x30000         uImage header, header size: 64 bytes, header CRC
    : 0x24D867E3, created: 2014-09-10 08:48:23, image size: 577048 bytes, Data
    Address: 0x80002000, Entry Point: 0x801B2040, data CRC: 0xAA92C548, OS:
    Linux, CPU: MIPS, image type: OS Kernel Image, compression type: lzma,
    image name: "MIPS Linux-2.4.31-Amazon_SE-3.6.]"
196672        0x30040         LZMA compressed data, properties: 0x5D,
    dictionary size: 8388608 bytes, uncompressed size: 1986560 bytes
851968        0xD0000         Squashfs filesystem, big endian, lzma
    compression, version 2.1, size: 2994562 bytes, 748 inodes, blocksize: 65536
     bytes, created: 2014-09-10 08:49:01


Scan Time:     2020-11-11 14:44:41
Target File:   /home/dell/Desktop/studies/ESS/lab3/_WAG120N-EU-ANNEXA-ETSI
    -1.00.20-code.bin.extracted/2B44
```

```
MD5 Checksum:   5fb265e57908c9aaf6fbfa4783c42aab
Signatures:     391

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

103364          0x193C4         Certificate in DER format (x509 v3), header
    length: 4, sequence length: 64
122416          0x1DE30         U-Boot version string, "U-Boot 1.1.5-2.0 (Jul 22
    2009 - 14:05:20)"
137328          0x21870         CRC32 polynomial table, big endian


Scan Time:      2020-11-11 14:44:41
Target File:    /home/dell/Desktop/studies/ESS/lab3/_WAG120N-EU-ANNEXA-ETSI
    -1.00.20-code.bin.extracted/30040
MD5 Checksum:   02d4f03de57da69b9a5a9dc2d7794fec
Signatures:     391

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

1188560         0x1222D0        Certificate in DER format (x509 v3), header
    length: 4, sequence length: 130
1596464         0x185C30        Linux kernel version 2.4.31
1616456         0x18AA48        Unix path: /usr/lib/libc.so.1
1985035         0x1E4A0B        LZMA compressed data, properties: 0x6C,
    dictionary size: 0 bytes, uncompressed size: 1953765504 bytes
```

From the output we can see that binwalk successfully extracted both lzma archives. The fist one is U-Boot image and the other one is kernel image. The kernel and the bootloader images are essential resources of the router. But binwalk also found the file system (Squashfs filesystem, big endian, lzma compression, version 2.1, size: 2994562 bytes, 748 inodes, blocksize: 65536 bytes, created: 2014-09-10 08:49:01), so it was our next target. In fact the previous binwalk command also extracted the filesystem, but we decided to extract the filesystem manually.

To get it, we tried a SquashFS and unsquashfs-2.1 firmware kits:

```
$ dd if=WAG120N-EU-ANNEXA-ETSI-1.00.20-code.bin bs=1 skip=851968 of=wag120.
    squashfs
3342336+0 records in
3342336+0 records out
3342336 bytes (3,3 MB, 3,2 MiB) copied, 7,78881 s, 429 kB/s
```

```
$ file wag120.squashfs
wag120.squashfs: data
```

Here, we can see the file command was unable to correctly identify the filesystem. Each of SquashFS images starts with 'sqsh' magic string (which indicates the beginning of a SquashFS image), so what we may be looking for here is a non-standard SquashFS signature inside the firmware file. We open the file in a plain text editor and notice that the file starts with `sqlz` string. The standard SquashFS image starts with `sqsh`, but we have already seen that the firmware developers have used LZMA compression elsewhere in this image. Also, most firmware that uses SquashFS tends to use LZMA compression instead of the standard zlib compression. So this signature could be a modified SquashFS signature that is a concatination of 'sq' (SQuashfs) and 'lz' (LZma).

After simple correction of the signature the file utility is able to correctly identify the filesystem:

```
$ file wag120.squashfs
wag120.squashfs: Squashfs filesystem, big endian, version 45.61375,
    -4630544833276940353 bytes, 751 inodes, blocksize: -272646673 bytes,
    created: Tue Dec  8 21:10:05 2071
$ sasquatch wag120.squashfs
SquashFS version [11520.49135] / inode count [-285081600] suggests a SquashFS
    image of a different endianess
Reading a different endian SQUASHFS filesystem on wag120.squashfs
Filesystem on wag120.squashfs is (45:61375), which is a later filesystem
    version than I support!
$ unsquashfs -f -d wag120 wag120.squashfs
Reading a different endian SQUASHFS filesystem on wag120.squashfs
Filesystem on wag120.squashfs is (45:61375), which is a later filesystem
    version than I support!
```

although the standard tools failed to extract it.

We suspect, that there is a problem with data inside squashfs file (version: 45.61375, negative size of bytes and blocks). When we were searching how to overcome this issue, we found the solution presented before with binwalk and recursive extraction.

Since we have got two squashfs files (one generated and modified manually by us and the other generated by binwalk), we decided to check what is the difference between them. As we suspected, there was a little amount of garbage in the end of first generated squashfs file, so this could be the source of the problem. However, we tried to "unsasquash" the file extracted by binwalk and got the same errors. We suspect differences in endianness (big-endian or little-endian). It shows how powerful binwalk is. We were able to extract all files and get the filesystem from the downloaded image with only one command. Binwalk was able to properly handle non-standard SquashFS Magic: 'sqlz' and other difficulties, so in the end, we got the essential resources of the device: kernel image. boot loader image and whole file system of the router.

1.6 Retrieve root password or get access to the root shell on the device

The first thing we did when we managed to establish UART connection was checking who is the user we are logged in and its privileges.

```
# sudo
-/bin/sh: sudo: not found
# echo $USER
root
```

It looks like every UART session uses root user and the task is basically done as we have the highest privileges possible.

We decide to look around and check what we can do. First, we search for other users. Inspection of `/etc/passwd` and `/etc/shadow` files confirms there are no other users than root and nobody pseudo user. List of running processes shows that everything is running as root. It means we can do anything, the only limitation is that the filesysem is mounted in readonly mode. The user is still able to control all processes and create files in some directories:

```
# cd tmp/
# touch whoami
# ls -al | grep whoami
-rw-r--r--    1 root     root             0 Jan  1 00:01 whoami
```

The user is able to read any file, including e.g. file with a password to GUI panel:

```
# cat /tmp/htpasswd
admin:admin
```

2 Conclusion

It turns out Linksys WAG120N is a poorly protected device. When conducting firmware/software reverse engineering we encountered some problems (mostly because tools version and settings mismatch) but in the end we were able to extract everything with just one command. The device basically has no anti-tampering measures as simple warranty sticker and removed UART connector are not an obstacle even for an inexperienced person. However, it might be very hard to get access to the interior without leaving a trace.

References

[1] FCC ID https://fccid.io/Q87-WAG120N

[2] TechInfoDepot http://en.techinfodepot.shoutwiki.com/wiki/Linksys_WAG120N

[3] Linux man page for screen https://linux.die.net/man/1/screen

[4] U-Boot Reference Manual https://www.digi.com/resources/documentation/digidocs/PDFs/90000852.pdf

[5] Linux man page for binwalk http://manpages.org/binwalk

[6] Sources and changelog of lzma https://packages.msys2.org/package/liblzma?repo=msys&variant=x86_64