INTERNATIONAL JOURNAL OF ADVANCE RESEARCH, IDEAS AND INNOVATIONS IN TECHNOLOGY

ISSN: 2454-132X
Impact Factor: 6.078
(Volume 7, Issue 3 - V7I3-1488)
Available online at: https://www.ijariit.com

A Practical approach for Firmware Reverse Engineering

Akash C. Koturwar
koturwar.ka@gmail.com
College of Engineering, Pune, Maharashtra

Dr. V. K. Pachghare
vkp.comp@coep.ac.in
College of Engineering, Pune, Maharashtra

Sharad Hange
sharad.hange@emerson.com
Emerson, Pune, Maharashtra

ABSTRACT

Embedded products have become widely used entities in the network of households and enterprises. The exponential increment in its use has resulted in a lapse in security measures taken while building these products, leaving them vulnerable to security attacks. An important feature is that embedded devices work on what is commonly known as Firmware, a piece of software that helps the embedded devices to work on the shared task. Securing the firmware will be an important step in securing the embedded product setup. To secure the firmware one needs to understand how the firmware works. One way to do so is to reverse engineering. Through reverse engineering, one can attempt to understand the device architecture, functionality, and potential vulnerabilities present in the device. A better understanding of the firmware implementation also helps in adding features if needed. This project aims to apply several types of reverse engineering techniques to increase the efficiency of reverse engineering the firmware and find the vulnerabilities in a firmware-based setup. Also, estimate the level of damage the identified vulnerabilities can cause through rigorous penetration testing.

Keywords— *Security, Firmware, Reverse Engineering*

1. INTRODUCTION

Reverse Engineering is the process of determining the technical objectives of a device, or system by analyzing its performance and functions. It involves breaking a part of a device and analyzing its performance in detail. It is often used to care for old products and to make a new product that uses the same technology without copying anything from the original product. The aim is to determine the design decisions for the final products with little knowledge about the processes involved in the original production.

Reverse engineering can be used in many circumstances like automotive industries, chemical industries, embedded systems, malware analysis, and for the security auditing of the firmware-based product. The goal of this project is to understand the firmware.

Much of security activity on embedded devices has identified vulnerabilities security errors, such as passwords with strong code extracted from firmware images, are not verified firmware upload, multiple unauthorized connections left enabled, and weak password hashing.

To reconstruct the entire target, it is important to understand both hardware and software this research focuses on reverse engineering the functionality used in software, which is typically referred to as targets firmware. Reverse engineering targets firmware gives a clear picture about the device functionality and the vulnerability exposed and extracts secret stored in the firmware which device developers don't want to be seen by everyone.

1.1 Literature Review

Reverse Engineering firmware has been the goal of several previous projects, as the attacks become complex and complex researcher around the globe started to explore the new ways of securing the setup. Reverse engineering yielded the most promising result as they realized that knowing how the setup is built gives the idea about its architecture and functional execution. Usually, Firmware has this things programmed. Some of the online blogs, videos give the common ways to access the firmware of devices like router, security camera[1][2][5].

Related work of Andrei Costin and Jonas Zaddach [7] in their paper titled as embedded device security and firmware reverse engineering they studied common firmware formats and what are the challenges related to each firmware format. This work shows some understanding of compiler, register, assembly language, common processor architecture, and logic structure of disassembler is important to make most out of firmware reverse engineering. They showed how tools like RevNIC, Ghidra, Beanwalk, Hexdump can be used to unpack the firmware. Then as their work proceeds, they showed that the firmware emulation does the firmware reverse engineering a bit easy and they succeeded in finding common vulnerabilities with the existing tools[7][8].

Alyssa Milburn and Niek Timmers [8] in their work for reverse engineering of automotive firmware they showed the firmware emulation with the publicly available tools and with skills, they have tried different ways for extracting the firmware, and with

© 2021, www.IJARIIT.com All Rights Reserved

this experience, they recommend that no matter how complex it is to have full firmware access, attackers always find the way. So, it is better to have the security at the hardware level. In this research they also listed common security configuration that results in compromising security of firmware. With this experience they recommend to increase the complexity of accessing the firmware.

Work by Vitaly Chipounov and George Candea [9] for reverse engineering the binary device drivers with RevNIC showed the new approach and tools for reverse engineering binary drivers. They discussed about the portability of device drivers from like using one device driver for other device.

In papers[7][8][9] different tools and techniques for achieving reverse engineering are discussed the goal of all the paper is to secure the product. Attackers also use reverse engineering to find the vulnerabilities in the product, as security is always evolving process there is always room for the new approach of achieving reverse engineering because the same approach won't be useful for all the product because of their diversity in processor architecture, embedded OS and way of handling the files. To secure the product one needs to find and patch the vulnerability before the attacker does.

1.2 System Requirement and Tools

- Ghidra- Ghidra is a Reverse Engineering framework developed by NSA for NSA's cyber security mission. It helps to analyze malicious code and malware like viruses and can give cyber security experts a better understanding of the potential dangers to their networks and systems
- Binwalk- Designed for identifying files and code embedded inside of firmware image.
- Hexdump- It is used to display the content of binary files in hexadecimal, decimal, octal and in ASCII format. It can be used for data recovery, reverse engineering purposes.

1.3 Concrete System Design

Below figure gives the system design to help achieve goal of firmware reverse engineering

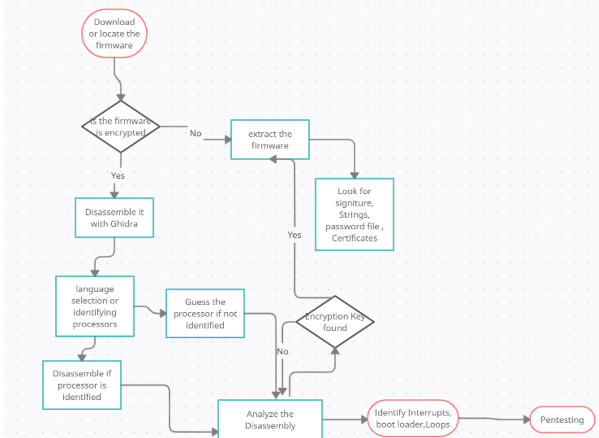


Fig. 1: Concrete System Design

2. IMPLEMENTATION

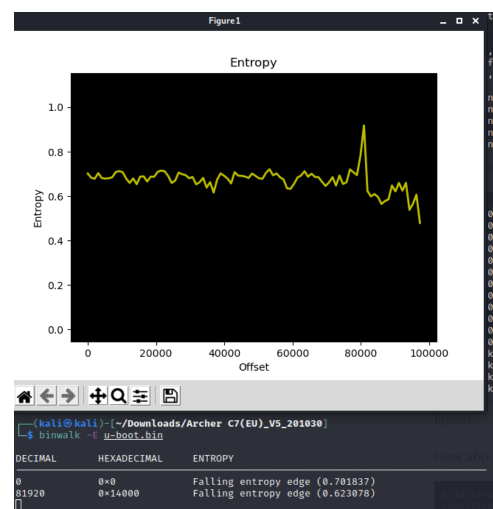
Firmware reverse engineering can give some meaningful results some of its use are mentioned in this chapter one is to get the file system in the firmware and other is to understand the programming behind the firmware which can give some of the important results like the certificates like X509 and the encryption key if the encryption is used. Extracting the file system can also give files like which contains the hardware images of the device to which the firmware is associated and also files like password etc.

Extracting the file system of the firmware

The extension of the firmware is .zip or .PAC in most of the cases but not always. First thing to do after having the file is to understand what type of information the file contains, this can be done by doing "file filename"

```
┌──(kali㉿kali)-[~/Desktop/_Update.PAC-1.extracted]
└─$ file kernal.lzma
kernal.lzma: data
```

File command gives what type of information the file contains. After having this information it is good to see the structure of the file. Now it's better to see if we could unzip the file and what are the files inside the zip, next is to run the file commands on all the files which are extracted after the unzip. In this step we can get to know if the file is encrypted or not if the more information about the file is given after doing binwalk on the file and checking the entropy of the file if the entropy graph is flat line then the file is most likely to be encrypted if not then it is compressed and it can be extracted. Entropy of firmware file when it is not encrypted is as shown in below fig.



Running the binwalk with a signature parameter will give the more details about the firmware if it is not encrypted like the bootloader of the firmware image header address at 23231(decimal) and the compressed boot loader image at 23296(decimal). Based on the uImage header at 78731(decimal we know that the CPU architecture is MIPS and the kernel version of the Linux and based on the image address found at 1166967(decimal)we know that root file system.

```
└─$ binwalk c7v5_us-up-ver1-0-16-P1\[20201030-rel55736\]_2020-10-30_15.32.02.bin

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
21876         0x5574          U-Boot version string, "U-Boot 1.1.4-gbec4c55d-dirty (Oct 20 2020 - 22:43:09)"
21940         0x55B4          CRC32 polynomial table, big endian
23232         0x5AC0          uImage header, header size: 64 bytes, header CRC: 0xAA7490A7, created: 2020-10-20 14:43:11, image size: 41172 bytes, Data
 MIPS, image type: Firmware Image, compression type: lzma, image name: "u-boot image"
23296         0x5B00          LZMA compressed data, properties: 0x5D, dictionary size: 8388608 bytes, uncompressed size: 97476 bytes
65041         0xFE11          XML document, version: "1.0"
78731         0x1338B         uImage header, header size: 64 bytes, header CRC: 0xD2250C62, created: 2020-10-20 14:54:37, image size: 1088171 bytes, Dat
U: MIPS, image type: Multi-File Image, compression type: lzma, image name: "MIPS OpenWrt Linux-3.3.8"
78803         0x133D3         LZMA compressed data, properties: 0x6D, dictionary size: 8388608 bytes, uncompressed size: 3164228 bytes
1166967       0x11CE77        Squashfs filesystem, little endian, version 4.0, compression:xz, size: 14378698 bytes, 2589 inodes, blocksize: 65536 bytes
15545674      0xED354A        gzip compressed data, from Unix, last modified: 2020-10-21 08:31:07
```

Now let's get the boot loader extracted, for this we will use dd command and the parameters used in the boot command are if(to specify input file), of(to specify the output file), skip will tell the dd command to start extracting the data in output file from the location specified, this location is the starting address of the boot loader and there is argument count which will specify the length of data to be extracted in the output file. The data of the image we are extracting is compressed as lzma so we need to extract that data in file with lzma extension.

```
┌──(kali㉿kali)-[~/Downloads/Archer C7(EU)_V5_201030]
└─$ dd if=c7v5_us-up-ver1-0-16-P1\[20201030-rel55736\]_2020-10-30_15.32.02.bin of=u-boot.bin.lzma bs=1 skip=23296 count=41172
41172+0 records in
41172+0 records out
41172 bytes (41 kB, 40 KiB) copied, 0.122436 s, 336 kB/s
```

© 2021, www.IJARIIT.com All Rights Reserved

Now, lets decompress the lzma file we extracted from the firmware

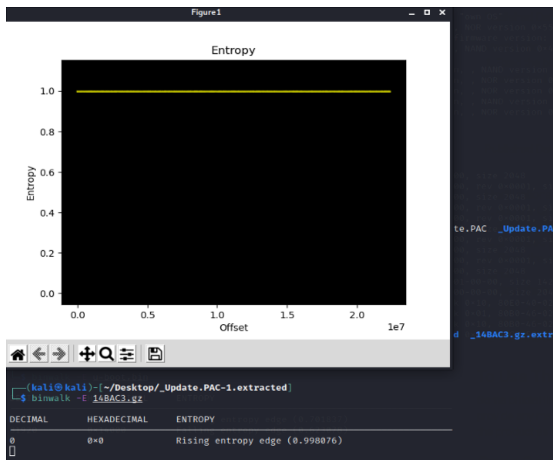
```
┌──(kali㉿kali)-[~/Downloads/Archer C7(EU)_V5_201030]
└─$ unlzma u-boot.bin.lzma
```

After extracting the file from firmware we can look for the string in the extracted and decompressed file strings like 'bootargs' will give more important info about the boot image.

In this way we can manually extract the images and compressed data from firmware and then decompress it. We can also extract the root file system of the firmware, once we have the root file system then we have all the information about the firmware like the password file, encryption keys if any and the certificates. having access to the password file we always have the backdoor, checking the validity of the certificate like if it is valid or expired. the presence of expired certificates can be dangerous.

In case when the firmware file is encrypted then we won't be able to get much information about the file. One way is to use Ghidra and try to find out the encryption key by disassembling the firmware file then there is also a challenging task to choose the language for disassembling the file i.e. processor variant and compiler architecture. When the language selected for disassembling is not the one required then the disassembled functions in Ghidra will give a lot of error and the function name and variable won't help in understanding the disassembly of firmware.

Entropy of firmware file when it is not encrypted is as shown in below fig.



2.1 Dealing with Encrypted Firmware

When the firmware is encrypted reversing it with the above-mentioned methods won't help. It is impossible to give step by step guide for decryption firmware that will fork for all. In this paper, we will look at the common scenarios. The easiest way to decrypt the firmware is tho look for the decryption method inside the firmware. In order for the router to decrypt the firmware, the decryption method should be inside the firmware. This implies that if the firmware version x.3 is encrypted then the decryption method should be inside version x.3 that is an immediate previous version, whenever you encounter encrypted firmware it is always good to go on the vendor website and poke around the previous firmware it is highly possible that you get something.

Common scenarios of encrypted firmware are

- Vx.0 is unencrypted, Vx.1 has a decryption method and Vx.2 is encrypted whose decryption method is in Vx.1
- Vx.0 and all its precious version are encrypted with decryption method1, Vx.1 is unencrypted and has decryption method 2, Vx.2 is encrypted whose decryption method is in Vx.2
- Vx.0 and all its precious version are encrypted with decryption method1, Vx.1 is encrypted and has decryption method 2, Vx.2 is encrypted whose decryption method is in Vx.2

Now let's apply this knowledge to encrypted moxa-nport firmware, here v1.11 is not encrypted and v2.2 is encrypted let's see if we can find something in v1.11.

```
┌──(kali㉿kali)-[~/Desktop]
└─$ binwalk moxa-nport-w2150a-w2250a-series-firmware-v2.2.rom

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
130077        0x1FC1D         MySQL ISAM compressed data file Version 4

┌──(kali㉿kali)-[~/Desktop]
└─$
```

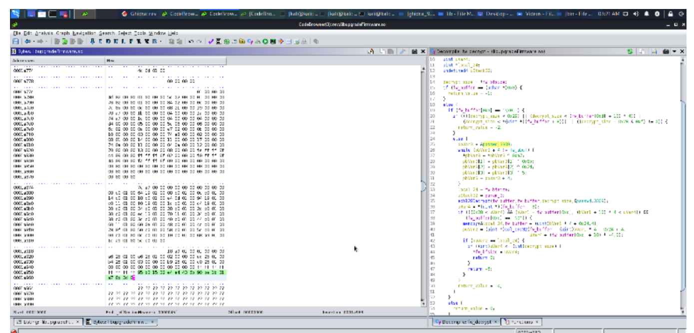
As we can see from the above fig binwalk doesn't give many results for v2.2 that means it's encrypted. Now after exploring the v1.11 root file system you find a file in /lib folder which is libupgradeFirmware.so

```
┌──(kali㉿kali)-[~/Desktop/_moxa-nport-w2150a-w2250a-series-firmware-v1.11.rom.extracted/squashfs-root-0/lib]
└─$ ls
libconfig.so  libpwd.so  libsio.so  libsystem.so  libupgradeFirmware.so
┌──(kali㉿kali)-[~/Desktop/_moxa-nport-w2150a-w2250a-series-firmware-v1.11.rom.extracted/squashfs-root-0/lib]
```

Let's analyze this in Ghidra, a very powerful tool for reverse engineering designed by NSA, it has the functionality to list all the functions related to the loaded file, their dependency on other functions, Ghidra can disassemble the assembly-level code into c code. we ca list all the functions in Ghidra by selecting function list window.

After listing all the function in libupgradeFirmware.so, some function which catches your attention and worth spending time on.Like fwdecrypt function.

In this function we find a variable named 'passwd.3309'
To get the key locate the passwd.3309 variable in the hex window and copy the 128 bit from the starting address of the password variable.



After copying the bytes stored in the 'passwd.3309' variable and doing the operation shown below will give the key(write a program that takes the hex array and performs the following operation on the array).

```
pbVar3 = &passwd.3309;
while (pbVar3 + 4 != fw_ubuf) {
  *pbVar3 = *pbVar3 ^ 0xa7;
  pbVar3[1] = pbVar3[1] ^ 0x8b;
  pbVar3[2] = pbVar3[2] ^ 0x2d;
  pbVar3[3] = pbVar3[3] ^ 5;
  pbVar3 = pbVar3 + 4;
}
```

© 2021, www.IJARIIT.com All Rights Reserved

this key is now supplied to the ecb128decrypt function, this function takes the input buffer convert it into output buffer with supplied key.[11]

```
ecb128Decrypt(fw_buffer,fw_buffer,decrypt_size,&passwd.3309);
uVar4 = *(uint *)(fw_buffer + 8);
```

the encryption starts after the offset (i.e 0x28, 40 in decimal) so we need to create the copy of firmware by removing the offset this we can do by dd command[11].

```
┌──(kali㉿kali)-[~/Desktop]
└─$ dd if=moxa-nport-w2150a-w2250a-series-firmware-v2.2.rom of=firm_off.enc bs=1 skip=40
8874768+0 records in
8874768+0 records out
8874768 bytes (8.9 MB, 8.5 MiB) copied, 31.5185 s, 282 kB/s

┌──(kali㉿kali)-[~/Desktop]
└─$
```

Now that we have offset removed we can apply openssl aes-128-ecb to decrypt the firmware by suppling the key

3. CONCLUSION

In this work study of different Reverse Engineering techniques and tools is done. This will help in building the new approach for Firmware reverse engineering. From [7][8][9] it is clear that the there is a need for more work in Firmware Reverse Engineering. The approach for applying it on industry products is shown in fig 2. Following are the conclusions of the paper

- Though firmware reverse engineering is becoming easier, there are still many challenges related to their unpacking and full-blown analysis.
- Many trivial and non-trivial security vulnerabilities can be found just by using existing tools and frameworks.
- When the firmware is encrypted, look around the different versions of that firmware.
- Once we have the firmware we can use firmware mod kit to pack the firmware by tampering it like adding backdoor.

4. REFERENCES

[1]https://linuxsecurityblog.com/2019/10/03/reverse-engineering-router-firmware/

[2]https://embeddedbits.org/reverse-engineering-router-firmware-with-binwalk/

[3]https://embeddedbits.org/2020-02-20-extracting-firmware-from-devices-using-jtag/

[4]https://payatu.com/blog/munawwar/iot-security---part-7-reverse-engineering-an-iot-firmware

[5]https://hackaday.com/2011/05/30/reverse-engineering-embedded-device-firmware/

[6]https://www.secforce.com/blog/2014/04/reverse-engineer-router-firmware-part-1/

[7]Andrei Costin and Jonas Zaddach Embedded Device Security and Firmware Reverse Engineering, July 2013 At BH13US'13 Las Vegas, USA

[8]Alyssa Milburn and Niek Timmers Effcient Reverse engineering of Automotive Firmware, July 2018

[9] Vitaly Chipounov and George Candea Reverse Engineering Binary Device Drivers with RevNIC, Proc. of the 5th ACM European Conference on Computer Systems , Paris, France, April 2010

[10] Y.David, Nimrod Partush and Eran Yahav FirmUp: Precise Static Detection of Common Vulnerabilities in Firmware, Proc. of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating System, June 2018.

[11] https://docs.huihoo.com/doxygen/openssl/1.0.1c/aesecb8csource.html/aes_ecb_signatur