Applied Research in Firmware security of embedded devices

Jan Polfers and Svetoslav Stoyanov

November 30th 2020
Module: SEAR
Venlo, Limburg, Netherlands

Abstract

- Complete, but very succinct summary of the paper
- Half a page long
- Short description of research(brief statements of the purpose, methods, results and conclusions)

Contents

List of Figures

1 Introduction

The research we are conducting is about firmware security mainly but we will touch on the topic of embedded devices also because the two work together. In general, most devices with firmware are relatively easy to reverse engineer by hackers who want to find vulnerabilities that they can exploit to attack different parts of the software eco-system. Therefore the goal of this research is to find a way to increase firmware security.

In recent years embedded devices have become very popular in many countries worldwide, they are often used in households, factories, and even in infrastructure objects. Their usage is predicted to increase steadily in the next decades. The firmware used in all embedded devices is basically computer software that is meant to work with specific hardware in order to execute one or a few very concrete functions. The Firmware is not as secure as we would like it to be, there are many ways of hacking (breaching) it so that one (the hacker) can find how exactly it operates and if there are any holes that can be exploited for benefit or to be used to harm either the users or the creators of the firmware. Even though there are many ways of hacking an embedded device, the focus of this research paper is mainly on binary reverse engineering in a firmware context. Reverse engineering is the process by which an artificial object is deconstructed to reveal its designs, architecture and mainly its source code(firmware). Therefore the point of reverse engineering a device is to reach the firmware and gain knowledge of how the device operates.

"A firmware is a program or set of instructions programmed on a hardware device(embedded device) which provides the necessary instructions for how the device communicates with the other computer hardware. For the purpose to be programmed on a hardware, the firmware is usually stored in the flash ROM of a device. While ROM is "read-only" memory, the flash ROM can be erased and rewritten because it is actually a type of flash memory. Firmware can be thought of as "semi-permanent" since it remains the same unless it is updated by a firmware updater." (Christensson, Per, 2020) Anyway an attacker can also modify the firmware and inject it back on the device replacing the old firmware with new corrupted firmware code.

The physical objects containing firmware (embedded devices) and connected to a network are described by the term "Internet Of Things".

Although IEEE IoT Initiative is proceeding to draft a white paper (Roberto Minerva, 27 May 2015) for the formal definition of IoT, there are still no common agreements for the definition of IoT. In this article, we define a "Thing" on IoT that indicates a physical or virtual object which connects to the Internet and has the ability to communicate with human users or other objects. Along with the growth of IoT, new security issues arise while traditional security issues become more severe. The main reasons are the heterogeneity and the large scale of the objects. The impact factors can be further divided into two categories: the diversity of the "Things" and the communication of the "Things". It is divided into two categories given that each of the category encounters different security problems. First, the security problem for the "Things" is created by

vulnerabilities produced by careless program design; this creates opportunities for malware or backdoors installation. Based on the heterogeneity and the scale of the “Things” in IoT, such security problems are more complex compared to the security problems that we have faced now. As for the communication medium of the “Things”, it is expected that the networking environment for IoT will be heterogeneous. Various communication media may face different security challenges. Overlooking these security problems will compromise the availability of the “Things”. As for the contents of the communication, the heterogeneous data structure and protocols also make content protection more complex.

For the sake of this research two methods of reverse engineering embedded device with the goal of extracting its firmware in a form of binary file will be mentioned. The first method is manual and requires several tools and manual file system inspection performed by a person, while the second method is almost entirely automated and requires only one software which does all the work and presents results.

On the other side there are several ways of protecting binaries. First one is ”Binary Code Packing”. ”Binary Code Packing” or Abstract—Packing (or executable compression) is considered as one of the most effective anti-reverse engineering methods in the Microsoft Windows environment. Even though many reversing attacks are widely conducted in the Linux-based embedded system there is no widely used secure binary code packing tools for Linux. (M. Kim et al, 2010) This paper presents two secure packing methods that use AES encryption and the UPX packer to protect the intellectual property (IP) of software from reverse engineering attacks on Linux-based embedded system. We call these methods: secure UPX and AES-encryption packing. Since the original UPX system is designed not for software protection but for code compression, we present two anti-debugging methods in the unpacking module of the secure UPX to detect or abort reverse engineering attacks. Furthermore, since embedded systems are highly resource constrained, minimizing unpacking overhead is important. Therefore, we analyze the performance of the two packing methods from the perspective of: code size, execution time, and power consumption. Our analysis results show that the Secure UPX performs better than AES-encryption packing in terms of the code size, execution time, and power consumption.(M. Kim et al, 2010)

Another method is named ”Defending Embedded Systems with Software Symbiotes”. A large number of embedded devices on the internet, such as routers and VOIP phones, are typically ripe for exploitation. Little to no defensive technology, such as AV scanners or IDS’s, are available to protect these devices. We propose a host-based defence mechanism, which we call Symbiotic Embedded Machines (SEM), that is specifically designed to inject intrusion detection functionality into the firmware of the device.A SEM or simply the Symbiote, may be injected into deployed legacy embedded systems with no disruption to the operation of the device. A Symbiote is a code structure embedded in situ into the firmware of an embedded system. The Symbiote can tightly co-exist with arbitrary host executables in a mutually defensive arrangement, sharing computational resources with its

host while simultaneously protecting the host against exploitation and unauthorized modification. The Symbiote is stealthily embedded in a randomized fashion within an arbitrary body of firmware to protect itself from removal. We demonstrate the operation of a generic whitelist-based rootkit detector Symbiote injected in situ into Cisco IOS with negligible performance penalty and without impacting the routers functionality. We present the performance overhead of a Symbiote on physical Cisco router hardware. A MIPS implementation of the Symbiote was ported to ARM and injected into a Linux 2.4 kernel, allowing the Symbiote to operate within Android and other mobile computing devices. The use of Symbiotes represents a practical and effective protection mechanism for a wide range of devices, especially widely deployed,unprotected, legacy embedded devices.(Cui, Ang and Stolfo, Salvatore, 2011).

The last method is called "Obfuscation" and is used because of continuous growth of the cloning of electronic devices poses a severe threat to our critical infrastructure that uses the Internet, as cloned devices can transmit secret information and cause security concerns. Cloned devices can also be unreliable as they may be manufactured with inferior quality materials, and they may have many defects as they may not be tested properly. It is thus extremely important to protect these electronic devices from cloning. An efficient way to prevent a device being cloned is to prevent the firmware from being copied because, without the proper firmware, the device will not function like the original. In this paper, we present a novel firmware obfuscation method without encrypting the entire memory. The firmware is obfuscated by swapping a subset of instructions. The instructions to be swapped are specifically chosen so that an attacker cannot discover their location. During operation, the hardware reconstructs the original program using a physically unclonable function-generated identifier and a small memory that stores the swapped instructions. An adversary cannot make a program work completely without knowing which instructions have been swapped, as the program will execute in the wrong sequence and produce the incorrect result. Our proposed solution requires only a small overhead to reconstruct the firmware, making it practical for devices with strict resource constraints. This solution also allows remote updates of new obfuscated firmware without any modification and is practical for the rising trend of ubiquitous computing.

2. Methods against reverse engineering binaries 2.1: Binary Code Packing 2.1: Symbiotes Software - Find more methods at -

Research question How does the attacker gain access to the code from the firmware, inside an embedded device?

Scope

Technical background Firmware Binary files (Filesystem)

- Provide the reader with everything they need to know to understand what you are doing and why
- Length: max 3 pages

- Theoretical background (literature review)
- Why the work is important
- Specific research question
- (if applicable) Hypothesis to be tested
- Divide into subsections

2 Methods

- How you performed the experiment / interview / survey or how you set up your comparative analysis
- Length: min. 2 pages
- Methodology (Research strategy, material, planning): which method you used, why, how it was carried out
- NO results or interpretation in this section!

We should primarily focus on experiments and research papers as material. Because these two sources give us a deep understanding of the underlying structure and the causes of the problem. Every paper you read should be logged on one of the following pages. To prove our research we maybe provide interviews.

3 Results

Topics: 1. Reverse engineering in general introduction Half a page or a page about it in general and methods against it
3. Reverse engineering of binaries automated and non automated (Manual and binwalk) This is used as an introduction into firmware reverse engineering.Firmware extraction is a good example.
4. Methods against reverse engineering binaries Explain a list of methods to prevent reverse engineering.
5. Obfuscation methods Old and reliable methods and state of the art
6. Deobfuscation methods Old and reliable methods and state of the art

- Share the data you found
- Length: min. 2 pages
- Describe the results (do NOT add your interpretation/analysis)
- Graphs, figures and tables that show your data belong in this section. Describe the graphs and explain what the reader is seeing

4 Discussion

- Interpret the results from the previous section
- Answer your research question/s and explain if your hypothesis was proven right or not: Refer to starting point (objective research question)
- Length: max. 4 pages
- Evaluate process (Reflection: how did it go)
- Further research (how your research could be improved, what else could be done)