INSO - Industrial Software
Institute of Computer Aided Automation | Faculty of Informatics | TU Wien

Exposé of Master Thesis

Automated Scalable Emulation System of Linux-based Firmware with Focus on Security Analysis

Advisor: Thomas Grechenig

Thomas Weber
1025654
066938
Computer Engineering

September 24, 2021

. . . .
Thomas Grechenig

Exposé of Master Thesis

1 Problem Description

The problem that is aimed to be solved is the design of an automated scalable firmware emulation system, which uncovers unknown security issues, also known as zero day vulnerabilities, in firmware of embedded systems. Embedded systems are not always as easy to test as ordinary computers because of less interfaces for human interaction like a screen or a mouse. Additionally, their firmware is often based on proprietary SDKs (Software Development Kits) without accessible source code for any used applications. Moreover, the complexity of testing such devices is increased as different CPU architectures like ARM, MIPS, PowerPC and SPARC are used in embedded systems for power saving and performance reasons. These instruction sets, which differ to traditional x86/x64 architectures, combined with restricted debugging and interaction capabilities of a typical embedded system pose a challenge for security researchers. As IoT-devices are omnipresent in many people's daily routines today and bot-nets in the past are built upon cameras, video recorders or routers, one cannot deny that security is an often uncared issue. To fix these security issues, it must be known where they reside. This is not easy to achieve since there are thousands of different kinds of devices with various functionalities in all imaginable areas of life in use nowadays.
Many vendors started to produce all kinds of devices with a link to the world-wide-web to sell more pieces, even when the link appears to be not really useful at first sight. Although such vendors often provide packed/compiled firmware updates on their web-pages, a full source-code review to evaluate and ensure security for such embedded devices is not feasible. It is however still possible to achieve a partial and sometimes very high coverage by emulating and testing firmware of such devices. Such emulations for firmware based on different CPU architectures are theoretically completely possible, but would imply too much effort in reality. Sometimes there are problems with specific and proprietary hardware configurations of a platform, which is the reason why the firmware of some devices is partially emulatable only. For security researchers, a remote debugger on the current audited embedded system is often a fond wish. This problem mostly applies to network based services but also to local services on the embedded systems.

2 Expected Results

This thesis focuses on building a firmware emulation system which is able to find security problems in embedded devices without possessing the hardware. It aims to answer the following research questions:
Is it possible to emulate firmware to satisfy at least the most services at runtime?
What monitoring possibilities regarding system calls and program crashes can be achieved?
How can debugging of a single service be achieved within the emulated firmware?
What is the percentage of the successful emulatability-rate of such firmware?
How many of the emulated firmware samples can be emulated automatically and how many are only manually emulateable?

3 Methodological Approach

The methodological approach for writing the thesis and assembling proof-of-concept code is structured in different work packages that are described in the following.

3.1 Literature Review

First of all, a complete literature review of all relevant publications concerning this issue must be gathered. This helps to collect information about the state of the art, the concept of other systems with similar purpose and the general do-ability. A proper set of requirements for a design of an automated firmware emulation system can also be deduced. This serves to build a solid base for the theoretical background.

3.2 Thesis Phases

The following phases of writing the thesis will be needed to answer the described problem:

- Theory and Introduction
- Architectural Design
- Security Analysis

3.3 Theory and Introduction

The theory part covers a short overview about the state of the art and concepts of related firmware emulation systems.
The introduction covers basics about design and prototyping of embedded systems. Foundations and terms for the whole system which will be built are explained in this section.

3.4 Architectural Design

In this chapter, an architectural design is introduced that combines different techniques of other publications with a new system construction concept.
For the first step to design an automated firmware emulation system, a sufficient set of samples must be collected from different vendors. This helps to construct the system as generic as possible and to deal with a wide-spread spectrum of various firmware types. To achieve this, self-written and open-source crawlers can be used. Other firmware images, which are exclusively stored in devices can only be extracted there. The extraction process will be discussed on a theoretical level followed by a practical example.
The extraction of files from the firmware is the next logical step and is the first process, which must be fully automated. This will be done with self-written software in combination with open-source tools. After the extraction, the nested directories must be pre-processed to get a functional file system. By a simple analysis of a file system a proper emulation environment can be created with cross-compiler and embedded system frameworks. These tools are also used for software/firmware development of real embedded systems. The automated creation of such an environment consists of the preparation of a kernel, the initial emulation, and the full emulation.

3.5 Security Analysis

The security analysis step is primarily chosen in order to test the stability of the emulated firmware images with regards to security features. Port scans, fuzzers, and other automated tools are used to scan the emulated firmware after a successful system startup. This aggressive test for weaknesses in the actual implementation of the firmware also stresses the emulation system, which constitutes a good stability test.

Prototype-level emulations of firmware samples from different vendors are done in this step. By monitoring them, a statement about flexibility of the framework regarding different vendor techniques can be made. The emulated fimware samples will be checked for security aspects by using different methods.
One automated method is a credential scan, which can be done with Nessus. Login credentials are provided for a security scanner, which connects to the target system and searches for local and remote vulnerabilities. Another method, which is very useful for manual testing, is a direct shell access in combination with a remote debugger. This enables engineers and security researchers to dissect a firmware during run-time.

4 State of the Art

Past projects with similar approaches like Chen et al. [2], Kim et al. [6] and Costin et al. [4] are built around QEMU from Bellard [1] and binwalk. There are also specialized projects to look for or implement backdoors or any form of authentication bypass in firmware for example Shoshitaishvili et al. [8] and Zaddach et al. [10], using machine learning like Costin et al. [5] and re-hosting firmware like Clements et al. [3].
The current technique is simple and effective. First, the firmware is extracted with binwalk, based on magic numbers and filesystem-structures. Sometimes the API of binwalk is used to create custom extraction output, but in general the result is the same. Second, the extracted files and/or the filesystem are analyzed by custom scripts to determine the endianess and the CPU-architecture. Until the stage where the analysis is finished, the most approaches are designed in a similar way. After that, there are differences. One approach is to reconstruct the filesystem and build a custom kernel and libraries to emulate specific hardware, like Zaddach et al. showed with his work that covers hardware in the loop [9]. The other approach just tries to run the firmware in a changeroot environment with a generic kernel [6][2][4].
The first approach can be applied to few firmware samples but is not feasible for any larger-scaled analysis. Due to the fact that a custom kernel with hardware in the loop must be built it is mandatory to do many manual tasks and own the target hardware. This is a big timely and budgetary effort.
The second approach will not be able to emulate (nearly) all firmware samples. They use prepared Linux images which do not provide any flexibility within the kernel configuration. The architectures are also limited to the images which are provided by the linux distribution.
In all these approaches the project buildroot [7] is not integrated. It is usually used to kick-start the development of embedded systems but it can also be utilized for automated emulation purposes in combination with orchestration.
This thesis aims to combine and extend both approaches on a buildroot basis in order to provide a system for manual and automated analysis.

5 Relevance to the Curriculum

This thesis implements a CPU-architecture independent emulation environment which can be automatically adjusted to the targeted executables and their root-filesystem. To achieve this, statistics and heuristics are used. The curriculum *Computer Engineering* touches the topics of this study in some aspects. As reverse-engineering and security testing of embedded systems are in the foreground the Digital Circuits and Systems trail from the curriculum are important and necessary respectively to the overall picture. To see the relation to the curriculum, the most linked courses are listed in the following:

- 183.645 Advanced Security for Systems Engineering
- 182.700 HW/SW Codesign
- 182.701 HW/SW Codesign - lab course
- 182.707 Advanced Digital Design
- 182.717 Networked Embedded Systems
- 389.159 Network Security

6 Table of Contents

Planned structure of thesis:

References

[1] Fabrice Bellard. "QEMU, a fast and portable dynamic translator." In: *USENIX Annual Technical Conference, FREENIX Track*. 2005, pp. 41–46.

[2] Daming D Chen et al. "Towards Automated Dynamic Analysis for Linux-based Embedded Firmware." In: *NDSS*. 2016. DOI: 10.14722/ndss.2016.23415.

[3] Abraham A Clements et al. "HALucinator: Firmware re-hosting through abstraction layer emulation". In: *29th USENIX Security Symposium (USENIX Security 20)*. 2020, pp. 1201–1218.

[4] Andrei Costin, Apostolis Zarras, and Aurélien Francillon. "Automated dynamic firmware analysis at scale: a case study on embedded web interfaces". In: *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*. ACM. 2016, pp. 437–448. DOI: 10.1145/2897845.2897900.

[5] Andrei Costin, Apostolis Zarras, and Aurélien Francillon. "Towards Automated Classification of Firmware Images and Identification of Embedded Devices". In: *IFIP International Conference on ICT Systems Security and Privacy Protection*. Springer. 2017, pp. 233–247. DOI: 10.1007/978-3-319-58469-0_16.

[6] Mingeun Kim et al. "FirmAE: Towards Large-Scale Emulation of IoT Firmware for Dynamic Analysis". In: *Annual Computer Security Applications Conference*. 2020, pp. 733–745. DOI: 10.1145/3427228.3427294.

[7] Thomas Petazzoni and Free Electrons. "Buildroot: a nice, simple and efficient embedded Linux build system". In: *Embedded Linux System Conference*. Vol. 2012. 2012.

[8] Yan Shoshitaishvili et al. "Firmalice-Automatic Detection of Authentication Bypass Vulnerabilities in Binary Firmware." In: *NDSS*. 2015. DOI: 10.14722/ndss.2015.23294.

[9] Jonas Zaddach et al. "AVATAR: A Framework to Support Dynamic Security Analysis of Embedded Systems' Firmwares." In: *NDSS*. 2014. DOI: 10.14722/ndss.2014.23229.

[10] Jonas Zaddach et al. "Implementation and implications of a stealth hard-drive backdoor". In: *Proceedings of the 29th annual computer security applications conference*. ACM. 2013, pp. 279–288. DOI: 10.1145/2523649.2523661.