Poster: Combining Fuzzing with Concolic Execution for IoT Firmware Testing

Jihyeon Yu
yjhy8783@sju.ac.kr
Sejong University, Seoul, Republic of Korea

Juhwan Kim
juhwan@sju.ac.kr
Sejong University, Seoul, Republic of Korea

Yeohoon Yun
yeohoon1991@sju.ac.kr
Sejong University, Seoul, Republic of Korea

Joobeom Yun*
jbyun@sejong.ac.kr
Sejong University, Seoul, Republic of Korea

ABSTRACT

The supply of IoT devices is increasing year by year. Even in industries that demand sophistication, such as unmanned driving, construction, and robotics industry, IoT devices are being utilized. However, the security of IoT devices is lagging behind this development due to their diverse types and challenging firmware execution environments. The existing methods, such as direct device connectivity or partial emulation, are used to solve this. However, full system emulation is better suited for the large-scale analysis, because it can test many firmwares without requiring devices. Therefore, recent studies have integrated emulation and software testing techniques such as fuzzing, but they are still unsuitable for testing various firmware and inefficient. In this poster, we propose FirmColic, which combines fuzzing with concolic execution to mitigate these limitations. FirmColic is a type of augmented process emulation, which improves the effectiveness of fuzzing using keyword extraction based on concolic execution. Also, we apply five arbitration techniques in an augmented process emulation environment for the high success rates of the emulation. We prove that FirmColic has faster detection, more crash detection, and a higher code coverage than the previous studies.

CCS CONCEPTS

• **Security and privacy → Vulnerability management**;

KEYWORDS

Keyword extraction;Fuzzing;Firmware;IoT devices

ACM Reference Format:
Jihyeon Yu, Juhwan Kim, Yeohoon Yun, and Joobeom Yun. 2023. Poster: Combining Fuzzing with Concolic Execution for IoT Firmware Testing . In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS '23), November 26–30, 2023, Copenhagen, Denmark.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3576915.3624373

*Corresponding author

CCS '23, November 26–30, 2023, Copenhagen, Denmark

ACM ISBN 979-8-4007-0050-7/23/11.
https://doi.org/10.1145/3576915.3624373

1 INTRODUCTION

Over the years, we have encountered various Internet of Things (IoT) devices in almost every aspect of our lives. IoT devices are improving not only quantitatively but also qualitatively so that they can be applied to sophisticated industries, such as construction and the robotics industry. However, due to the fierce competition among manufacturers for these technologies, the security of IoT devices has not improved much. These diverse targets and lax security conditions are optimal situations, which make them easy targets for attackers. Recent studies have aimed to automatically test the firmware on a wide range of IoT devices, and one of these studies applies testing techniques that have previously been successful in software to the firmware. A typical example is Firm-AFL [8], which is a system that combines a QEMU [1]-based automated emulation framework called Firmadyne [2] with a coverage-guided greybox fuzzing tool called AFL(American Fuzzy Lop). Firm-AFL complements the shortcomings of the emulation instability and throughput. Also, it connects AFL with the firmware in the emulation environment, which enables the testing of the fuzzing techniques for the firmware web application.

However, they have very low success rates of emulation, because most of the fuzzers for the IoT firmware use QEMU, which is compatible with a small number of firmware. In addition, web application fuzzing has a fixed input format, so the existing fuzzers are sometimes not efficient with generating input, and keyword extraction through a manual analysis is inevitable to compensate for this. In order to overcome these limitations, we propose FirmColic, which combines concolic execution with a fuzzer for the IoT firmware testing. The concolic execution technique can solve complex path constraints, and it is used to automatically solve the path constraints as the keywords for the firmware's web application binary in our system. Also, in order to improve the previous low success rates of emulation, we apply five arbitration techniques from FirmAE [3], which is one of the recently released emulation studies, to our system. Finally, we propose a Ghidra [4] handler in order to enable the program analysis of the architectures that are not supported in the concolic execution.

2 IOT FIRMWARE FUZZING

IoT devices accomplish preset tasks under limited conditions and resources. IoT devices have software, hardware, and firmware that enable their interaction and communication. Due to its role in controlling the IoT devices, the firmware faces the threat of remote

attacks. Despite these risks, the source code of the firmware is not distributed, which poses threats to the product after its release, because there is no in-depth security check on the firmware. Therefore, we need to analyze the firmware in an emulation environment due to the absence of the source code and the diversity of firmware. In recent studies, the emulation techniques can be classified into four main categories: user-mode emulation, system-mode emulation, unicorn engine, and augmented process emulation. Each emulation technique has its limitations. For example, the unicorn engine used in RSFuzzer [7] lacks the support for interrupts, the user-mode emulation does not emulate the hardware application programming interface (API), and the system-mode emulation can incur process overhead. To overcome these limitations, augmented process emulation introduced in Firm-AFL is usually performed in the user-mode emulation, and then changes the system-mode emulation when it encounters un-processable system calls. However, augmented process emulation is also unstable due to the low emulation success rates and the possible problems with fuzzing input processing because it is based on less-handled QEMU.

In this poster, we present a system that applies augmented process emulation and aims at a relatively stable emulation environment using five arbitration techniques from FirmAE to improve the emulation success rate.

3 FIRMCOLIC OVERVIEW

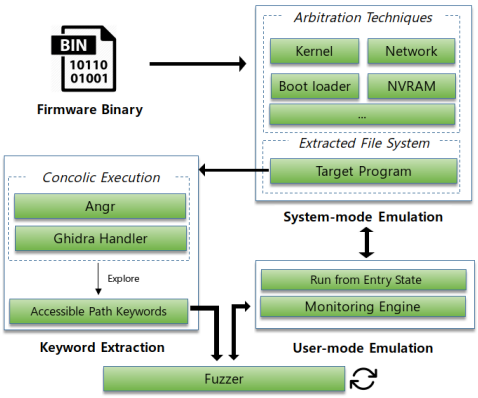

Figure 1: Overview of FirmColic

We propose FirmColic, a system for high emulation success rates and fast coverage growth using keyword extraction. The overview of FirmColic is shown in Figure 1. FirmColic takes firmware binary file as an input. Then, as mentioned in Section 2, it applies two emulation techniques for the augmented process emulation, which include the user-mode and the system-mode emulation. In addition, it performs keyword extraction for the target program based on concolic execution. Finally, the extracted keywords are passed to the fuzzer at the beginning of the fuzzing, and the fuzzer generates test cases that are based on the passed keywords and attempts to fuzz them to the target program in the user-mode emulation. We describe each step in detail as follows.

System-mode Emulation Existing full-system emulation systems aim to fully match the actual physical environment. However, the diversity of firmware types from various vendors makes it hard to achieve a high emulation success rate in large-scale analysis. This problem can only be solved by fully emulating all physical devices, which is practically impossible. For dynamic analysis testing of many firmwares, it is more efficient to adjust the emulation environment through some manipulation than to exactly emulate the behavior of the physical device. Mingeun et al. [3] state that there are five reasons for low emulation success, which include the boot, network, NVRAM, kernel, and other issues. To address this, we apply optimization heuristic techniques called arbitration techniques to each problem, which increase the emulation success rate of the existing study Firmadyne by 16.28% to 79.36%. For our dynamic and static analysis systems, we implemented some interventions via arbitration techniques in the system-mode emulation and automated the interworking with the AFL.

User-mode Emulation By quickly translating system calls, user-mode emulation enables processes compiled for one central processing unit (CPU) to run on another CPU at a high speed. This emulation is faster than system-mode emulation, so it communicates directly with the fuzzer. User-mode emulation, which does not emulate peripherals for faster speed, passes the system calls that it cannot process to system-mode emulation. It also maintains the state so that the fuzzer can run from the entry state of the target program for each input. Finally, the monitoring engine checks the program code coverage and the crash caused by the input during the fuzzer running.

Keyword Extraction The fuzzer can use keywords in the fuzzing process as important references when generating the test cases. This is especially significant in web API fuzzing, where the input format has some degree of fixedness. In the previous studies, such as Firm-AFL, the keywords were collected by manual analysis, which were then passed to the fuzzer. However, this method is influenced by the ability of the analyst and it requires a lot of effort. Sometimes, the collected keywords may be meaningless for the increased code coverage and the crash detection. In order to alleviate the existing problem, our system adopted concolic execution for the keyword extraction. The concolic execution is usually utilized to solve hard constraints to pass through the program paths. In other words, we can gather keywords that can access and pass through the program paths. We use Angr [6] as a concolic execution tool, but Angr has limited support for MIPS and ARM, which are popular as a firmware architecture. FUZZWARE [5], a system that applies symbolic execution using Angr to IoT firmware fuzzing, also supports only ARM

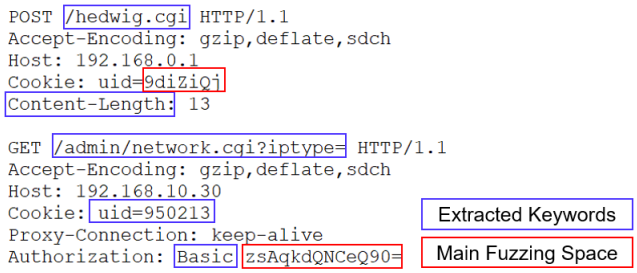

Figure 2: An example of a test case based on extracted keywords

Table 1: Time-to-Crash (TC), Total-of-Paths (TP), and Unique Crashes (UC), FirmColic versus Firm-AFL

	$DIR825_{httpd}$	$DIR850_{hnap}$	$WR940N_{httpd}$	$DIR817LW_{hnap}$	$TVIP110WN_{video.cgi}$	$EX6150_{httpd}$	$DIR860_{httpd}$
Firm-AFL #TC	1h 8m	1h 11m	-	48m	10h 14m	x	x
#TP	251	216	684	204	756	x	x
#UC	23	147	6	58	116	x	x
FirmColic #TC	48m	38m	6h 28m	32m	8h 58m	24m	1h 58m
#TP	436	742	1468	588	1856	1355	722
#UC	72	202	10	77	366	84	16
Architecture	mipseb	mipseb	mipseb	mipseb	mipseb	mipsel	armel
TP (increase)	174 %	344 %	215 %	288 %	246 %	-	-
UC (increase)	313 %	137 %	167 %	133 %	316 %	-	-

Cortex-M. We integrated Ghidra with Angr to overcome the architectural limitations that hinder concolic execution. It is possible to analyze the registers and memory mapping of the target programs that Angr cannot analyze because Ghidra already supports most of the MIPS and ARM architectures, and it is easy to extend it for unsupported architectures. When concolic execution is performed for all the program paths, the collected keywords are sent to the fuzzer.

Fuzzer When the emulation environment is activated and keyword extraction step ends, the fuzzer generates the input values based on the extracted keywords. An example of a test case that is based on the extracted keywords is shown in Figure 2. The generated and the mutated test cases with partially inserted extracted keywords are repeatedly sent to a user-mode emulator, which is the environment where the target program runs. The created test case is placed in the queue and mutated if it increases the code coverage of the target program. Our system prototype uses AFL, but replacing it with another fuzzer for the mutation efficiency and throughput can also be used for a future study.

4 EVALUATION

We evaluated the performance of FirmColic by comparing it with a previous system, Firm-AFL. We fuzzed all the target programs for 24 hours and measured the results as the average of 3 trials. The desktop computer that we used for the evaluation had an Intel i7 processor, 32GB of RAM, Ubuntu 18.04 LTS, and AFL version 2.52b.

To evaluate our system, we compared Time-to-Crash (TC), Total-of-Paths (TP), and Unique Crashes (UC) with Firm-AFL. Our experimental results are shown in Table 1. The experimental targets were router and IP camera firmwares from popular vendors, such as Dlink, TP-Link, and Netgear. In the Time-to-Crash experiment, we only counted the first time when a 1-day or N-day vulnerability was found, and ignored false-positive crash detections. In the case of Total-of-Paths, we could not measure the code coverage because we targeted binary files inside the firmware file system, not the source files. Therefore, we used Total-of-Paths as a proxy for coverage measurement. In Table 1, we can see that most of FirmColic's TCs are similar to or much shorter than Firm-AFL's TCs. Moreover, it shows that TP and UC detected by FirmColic can be up to 3 times more than Firm-AFL. Finally, Firm-AFL failed to emulate two of the target firmwares, but FirmColic succeeded in emulation and analysis through optimization arbitration techniques, which indicates that it provides better emulation environments than the previous system.

In summary, FirmColic outperforms Firm-AFL in fuzzing performance in terms of TC, TP, and UC, and it is compatible with firmwares that were not emulated by Firm-AFL. The benefit of the emulation success is expected to increase as the number of targets grows.

5 CONCLUSION AND FUTURE WORKS

In this poster, we propose FirmColic, a framework that combines fuzzing and concolic execution for IoT firmware testing. Our system applies five arbitration techniques to improve the emulation success rates. Moreover, we also enhance the fuzzing efficiency by extracting keywords from the target using concolic execution. We demonstrate that FirmColic detects crashes faster and covers more paths than a previous system, Firm-AFL. However, the concolic execution step can be time-consuming, which results in higher overhead. Also, some of the extracted keywords may be redundant. To overcome these limitations, we plan to research how to extract only the meaningful keywords, instead of all the keywords.

REFERENCES

[1] Fabrice Bellard. 2005. QEMU, a fast and portable dynamic translator.. In *USENIX annual technical conference, FREENIX Track*, Vol. 41. Califor-nia, USA, 46.

[2] Daming D Chen, Maverick Woo, David Brumley, and Manuel Egele. 2016. Towards Automated Dynamic Analysis for Linux-based Embedded Firmware.. In *NDSS*, Vol. 1. 1–1.

[3] Mingeun Kim, Dongkwan Kim, Eunsoo Kim, Suryeon Kim, Yeongjin Jang, and Yongdae Kim. 2020. FirmAE: Towards Large-Scale Emulation of IoT Firmware for Dynamic Analysis. In *Annual Computer Security Applications Conference*. 733–745.

[4] Roman Rohleder. 2019. Hands-on ghidra-a tutorial about the software reverse engineering framework. In *Proceedings of the 3rd ACM Workshop on Software Protection*. 77–78.

[5] Tobias Scharnowski, Nils Bars, Moritz Schloegel, Eric Gustafson, Marius Muench, Giovanni Vigna, Christopher Kruegel, Thorsten Holz, and Ali Abbasi. 2022. Fuzzware: Using Precise {MMIO} Modeling for Effective Firmware Fuzzing. In *31st USENIX Security Symposium (USENIX Security 22)*. 1239–1256.

[6] Yan Shoshitaishvili, Ruoyu Wang, Christopher Salls, Nick Stephens, Mario Polino, Andrew Dutcher, John Grosen, Siji Feng, Christophe Hauser, Christopher Kruegel, et al. 2016. Sok:(state of) the art of war: Offensive techniques in binary analysis. In *2016 IEEE symposium on security and privacy (SP)*. IEEE, 138–157.

[7] Jiawei Yin, Menghao Li, Yuekang Li, Yong Yu, Boru Lin, Yanyan Zou, Yang Liu, Wei Huo, and Jingling Xue. 2023. RSFuzzer: Discovering Deep SMI Handler Vulnerabilities in UEFI Firmware with Hybrid Fuzzing. In *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2155–2169.

[8] Yaowen Zheng, Ali Davanian, Heng Yin, Chengyu Song, Hongsong Zhu, and Limin Sun. 2019. FIRM-AFL: high-throughput greybox fuzzing of iot firmware via augmented process emulation. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*. 1099–1114.