Overview of IoT Fuzzing Techniques

Tuan-Dat Tran
(3012345)

University of Duisburg-Essen
tuan-dat.tran@stud.uni-due.de

***Abstract*—Due to the rising popularity of IoT devices and embedded systems and their usage in, not only in the business sector, but also at home, the view has been shifting on the security of those devices. To address this issue, there have been many approaches in detecting, analyzing and mitigation of security flaws in IoT devices. One of the ideas to detect vulnerabilities in an automated manner is IoT Fuzzing. Contrary to regular fuzzing it comes with its own constraints and techniques to optimize performance and coverage of attack surfaces.**

In this paper we are comparing techniques used by IoT fuzzers to circumvent the adversities presented by IoT devices like app-based approaches by IoTFuzzer and Snipuzz or emulation approaches used by Firm-Afl.

Due to the wide range of different IoT fuzzing tools we are dividing the comperison of the techniques based on the type of IoT fuzzing tool. We also outline the evolution of IoT fuzzing techniques to visualize the progress made in the field. This overview can then be used to choose the optimal usage of a specific IoT fuzzing device in a given use case or combine different techniques used in different fuzzing tools to create a novel approach and find new security flaws through an combined usage of IoT fuzzing techniques.

I. Introduction

Internet of Things (IoT) devices and embedded systems are becoming more and more prevalent, and with billions of devices being connected to the internet they are an integral part of everyday life[10]. Despite IoT devices being so widespread they are riddled with security vulnerabilities, which makes them an easy target for attackers, since many of those vulnerabilities are considered "low hanging fruits". This led to over 70 unique attack incidents[12] between 2010 and 2016, while the number of IoT devices and embedded systems in use is steadily rising and with it the amount of vulnerabilities in the wild.

While implementation flaws and app over-privilege are just some of the many security problems an IoT device can have, detection and mitigation of these security flaws has proven itself to be challenging[13]. One approach to discover those flaws is called fuzz-testing, or fuzzing. Mitigation of found security flaws can often be hard due to the nature of embedded devices being heavily customized and often not adhering to one specific standard. Therefore, the fixing of security flaws is often left to the manufacturer of the device, since they possess the necessary toolchains, source code and pipelines to provide security patches to their devices.

Fuzzing is a method to test software for flaws by automatically generating and sending malformed data to the software. There are many ways to generate and send data to the software. An example for a specific type of input generation is mutation based fuzzing, which is utilized by IoT Fuzzer[7]. Mutation based fuzzing takes a valid input and changes specific parts of it to trigger an unexpected state in the software and therefore crashing it. Crashing or bringing the software into an unexpected state is the general goal of fuzzing, since behavior like this indicates the presence of a bug.

Due to fuzzing being an automated process, fuzzing became a common tool for software testing in software development. Conventional fuzzing of software can be easily done concurrently, since software can, in most cases, be easily executed concurrently[13]. This increases the throughput of the fuzzer and thus the amount of test cases the software is tested against. This is one of the issues, which IoT fuzzers have to deal with, since the fuzzing IoT devices usually include fuzzing the physical device itself if there is no emulation solution available, while emulation enables another class of issues and complexity to the fuzzing process. An example for an arising problem due to emulation is the acquisition of the firmware. The process of firmware acquisition is different for every device, since it is dependant on the willingness of the manufacturer to publicly release the firmware. If the manufacturer does not release the firmeware for his device the firmware needs to be extracted directly from the device, which can vary in difficulty depending on the device[13].

Alternativly to fuzzing there are other ways to test software for vulnerabilities like static and dynamic firmware analysis. Static firmware analysis is the analysis of firmware without executing it by using tools like binwalk[1] to unpack the firmware and reverse engineering it with an reverse engineering tool like IDA[11][6]. For dynamic analysis the firmware is executed to be investigated. This can be done in a multitude of ways, for example running the firmware on the original device or emulating the device to have the firmware run in the emulated environment. The running firmwares behavior is then analyzed[5]. The advantage of static analysis is the possibility to automate and scaling the processes of analyzing the firmware[4], since the testing does not depend on a physical device. On the other hand, static analysis also yields a high amount of false positives and may not find completly new vulnerabilities, with the usage of its heuristics[17]. Another challenge during static analysis is the handling of packed or obfuscated code. This can be overcome with dynamic analysis[16] by emulating the physical device, which increases scalability and eliminates the need to acquire the physical device to test it[5].

Since IoT devices offer a large surface area regarding communication e.g. network protocols, their companion app or their web interface[3][2][15]. For this reason fuzzers, which were not originally designed to fuzz IoT devices can still be utilized for IoT fuzzing, like in the case of boofuzz, which was developed with the intent to fuzz network protocols[2]. IoT fuzzers can also make use of techniques used by dynamic analysis, since both approaches require execution of the firmware. This makes emulation a feasable way of testing IoT devices to increase scalability[8]. In this work we will focus mainly on fuzzers, which were primarily developed for IoT fuzzing, but since techniques used by non-IoT focused fuzzers are also used by fuzzers, that focus on IoT devices, non-IoT focused fuzzers will be considered in the overview.

Even though IoT fuzzers are used for finding security vulnerabilities in devices and fixing those errors or learning from them and mitigating them is the next logical step we will not discuss mitigation techniques in this paper, since this is outside of our scope. We will also not dive deep into specific techniques and how they work in detail or are implemented.

By creating an overview of different IoT fuzzing techniques we hope to archive a comprehensive list of IoT fuzzing tools and their properties to help developers and researchers to find the right tool for their job and weight in the positive on negative aspects of existing approaches to improve upon them.

II. Background

A. *IoT devices and embedded systems*

The terms IoT devices and embedded systems describe a great amount of devices. Embedded systems are devices, interact with their surroundings via sensors and regulators and are built to serve a specific purpose[13]. IoT devices on the other hand are broadly described as devices, which extend regular devices with an internet connection and enable them to communicate over it[14]. The term embedded devices can describe many devices such as cameras or industrial control systems (ICS), which makes it hard to make general statements about embedded devices. The same is the case for IoT devices, which includes the definition of internet capable embedded systems. Ongoing, when we describe IoT devices, the description also fits embedded systems if not explicitly mentioned.

IoT devices, due to being built for specific purposes, don't need as much processing power as a general computer does. This leads to them having an hardware platform specifically tailored to their use case. For this reason the amount of different hardware and software architectures used in IoT devices is diverse and unlistable. The works of Hahm et al.[9] proposes a classification into low-end and high-end IoT devices and deviding those two classificatoins into three subcategories for low-end devices. Those classes represent the complexity and computing capability of those devices with "Class 0" having the least resources and "Class 2" devices having the most resources. In the works of Muench et al.[13] a similar classification is done. They are classified in "Type-0" to "Type-III" systems. T0 (Type-0) systems represent multi-purpose systems, which don't fall under the classification of embedded systems or IoT devices. T1 (Type-1) devices are devices, which use a general purpose operating system like Linux. The OS (operating system) is often modified to be more lightweight and offer a lightweight user environment like busybox. T2 (Type-2) devices run on customized operating systems, which are tailored to the devices use case. In order to save space and computational power typical OS functions like a Memory Management Unit may be omited. T3 (Type-3) devices run on a single control loop. On these devices the firmware and the software, which runs the devices functionalities, are a single instance. This leads to a so called "monolithic firmware", consisting of the application and system code compiled together.

We will later use these classes, and especially those proposed by Muench et al.[13] to classify IoT devices when comparing IoT fuzzing techniques, since different types of IoT devices require different approaches to fuzzing.

Due to the heterogenic nature of IoT devices in terms of e.g. OS, instruction sets or memory layouts, analysis of the firmware proves difficult[4]. Reasons for this are the different requirements a manufacturer has for his device like the energy efficiency, real-time capabilty or memory footprint[9].

Like mentioned earlier IoT devices, and especially home-based ones, use a multiple ways to connect to the internet. Either directly through WiFi or via a intermediary device like a smartphone and connecting to it with Bluetooth[15]. Another way is having an IoT hub, which acts as proxy between other IoT devices and either another intermediary via Bluetooth or directly WiFi. This leads to many ways an IoT network can be structured, depending on the kind and number of IoT devices (Figure 1).

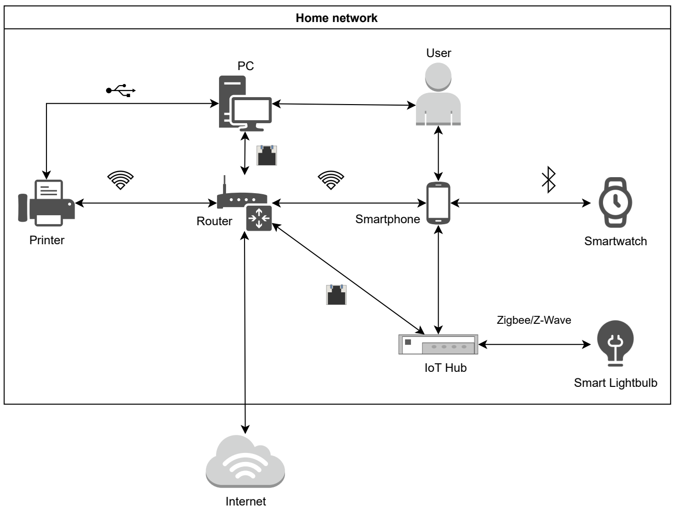

Figure 1. Example of IoT home network.

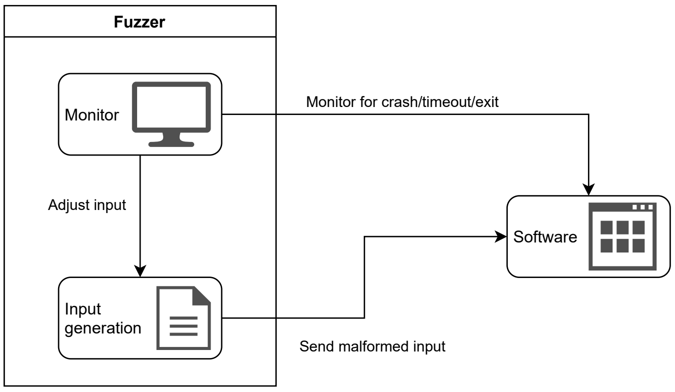

Figure 2. Generalization of fuzzing process.

B. *Fuzzing*

III. IoT Fuzzing

Mutation based fuzzing Taint Emulation Message

A. *Adversaries*

B. *Whitebox Fuzzing*

C. *Blackbox Fuzzing*

D. *Greybox Fuzzing*

IV. Overview of IoT Tools and Techniques

In this section we are going to look the overview of different IoT Fuzzing devicesTable I. As we can see the fuzzers requireing firmware, i.e. whitebox fuzzers, are mostly designed to fuzz IoT devices, built on top of linux devices. This is due to.

A. *Techniques*

B. *Classification*

V. Conclusion

In this paper we created an overview of the different IoT fuzzing techniques used by state of the art IoT fuzzing tools and compared their approaches in regards of input generation, execution speed, crash detection heuristics and their device scopes based on the classification in the work of Muench et al.[13]. The comparison was done seperatly based on whether they were black-/white- or greybox fuzzers.

References

[1] *Binwalk*. https://github.com/ReFirmLabs/binwalk.

[2] *boofuzz*. https://github.com/jtpereyda/boofuzz.

[3] Jiongyi Chen et al. "IoTFuzzer: Discovering Memory Corruptions in IoT Through App-based Fuzzing". In: *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018.* The Internet Society, 2018. URL: http://wp.internetsociety.org/ndss/wp-content/uploads/sites/25/2018/02/ndss2018%5C_01A-1%5C_Chen%5C_paper.pdf.

[4] Andrei Costin, Jonas Zaddach, Aurélien Francillon, and Davide Balzarotti. "A Large-Scale Analysis of the Security of Embedded Firmwares". In: *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014.* Ed. by Kevin Fu and Jaeyeon Jung. USENIX Association, 2014, pp. 95–110. URL: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/costin.

[5] 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, AsiaCCS 2016, Xi'an, China, May 30 - June 3, 2016.* Ed. by Xiaofeng Chen, XiaoFeng Wang, and Xinyi Huang. ACM, 2016, pp. 437–448. DOI: 10.1145/2897845.2897900. URL: https://doi.org/10.1145/2897845.2897900.

[6] Yaniv David, Nimrod Partush, and Eran Yahav. "FirmUp: Precise Static Detection of Common Vulnerabilities in Firmware". In: *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2018, Williamsburg, VA, USA, March 24-28, 2018.* Ed. by Xipeng Shen, James Tuck, Ricardo Bianchini, and Vivek Sarkar. ACM, 2018, pp. 392–404. DOI: 10.1145/3173162.3177157. URL: https://doi.org/10.1145/3173162.3177157.

[7] The OWASP Foundation. *Fuzzing | OWASP*. 2021. URL: https://web.archive.org/web/20210414111843/https://owasp.org/www-community/Fuzzing (visited on 04/14/2021).

[8] Zhijie Gui, Hui Shu, Fei Kang, and Xiaobing Xiong. "FIRMCORN: Vulnerability-Oriented Fuzzing of IoT Firmware via Optimized Virtual Execution". In: *IEEE Access* 8 (2020), pp. 29826–29841. DOI: 10.1109/ACCESS.2020.2973043. URL: https://doi.org/10.1109/ACCESS.2020.2973043.

[9] Oliver Hahm, Emmanuel Baccelli, Hauke Petersen, and Nicolas Tsiftes. "Operating Systems for Low-End Devices in the Internet of Things: A Survey". In: *IEEE Internet Things J.* 3.5 (2016), pp. 720–734. DOI: 10.1109/JIOT.2015.2505901. URL: https://doi.org/10.1109/JIOT.2015.2505901.

[10] Mark Hung. "Leading the IoT Gartner Insight on How to Lead in a Cnnected World". In: *Gartner Research* 1 (2017), pp. 1–5.

[11] *IDA Pro*. https://hex-rays.com/ida-pro/.

[12] David McMillen. "Security attacks on industrial control systems". In: *Technical Report*. IBM, 2015.

[13] Marius Muench, Jan Stijohann, Frank Kargl, Aurélien Francillon, and Davide Balzarotti. "What You Corrupt Is Not What You Crash: Challenges in Fuzzing Embedded Devices". In: *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018.* The Internet Society, 2018. URL: http://wp.internetsociety.org/ndss/wp-content/uploads/sites/25/2018/02/ndss2018%5C_01A-4%5C_Muench%5C_paper.pdf.

[14] Brien Posey. *IoT devices*. 2021. URL: https://web.archive.org/web/20210520072243/https://internetofthingsagenda.techtarget.com/definition/IoT-device (visited on 05/20/2021).

[15] Dong Wang, Xiaosong Zhang, Ting Chen, and Jingwei Li. "Discovering Vulnerabilities in COTS IoT

Tool	Year	Fuzzing approach	Techniques	Scope
SIoTFuzzer	2021	Blackbox		
IoTFuzzer	2018	Blackbox		
Firm-AFL[17]	2019	Greybox		
Snipuzz	2021	Blackbox		
Firmcorn	2020			
FirmFuzz	2019			
Diane	2021	Blackbox		
HFuzz	2019			
IoTHunter	2019			
WMIFuzzer	2019	Blackbox		

Table I
An overview of different IoT fuzzing tools.

Devices through Blackbox Fuzzing Web Management Interface". In: *Secur. Commun. Networks* 2019 (2019), 5076324:1–5076324:19. doi: 10.1155/2019/5076324. url: https://doi.org/10.1155/2019/5076324.

[16] Jonas Zaddach, Luca Bruno, Aurélien Francillon, and Davide Balzarotti. "AVATAR: A Framework to Support Dynamic Security Analysis of Embedded Systems' Firmwares". In: *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014*. The Internet Society, 2014. url: https://www.ndss-symposium.org/ndss2014/avatar-framework-support-dynamic-security-analysis-embedded-systems-firmwares.

[17] Yaowen Zheng, Ali Davanian, Heng Yin, Chengyu Song, Hongsong Zhu, and Limin Sun. "FIRM-AFL: High-Throughput Greybox Fuzzing of IoT Firmware via Augmented Process Emulation". In: *28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-16, 2019*. Ed. by Nadia Heninger and Patrick Traynor. USENIX Association, 2019, pp. 1099–1114. url: https://www.usenix.org/conference/usenixsecurity19/presentation/zheng.