A comprehensive survey of vulnerability detection method towards Linux-based IoT devices

Xixing Li
lxx.scholar@gmail.com
Information Engineering University
Zhengzhou, Henan, China

Qiang Wei
prof_weiqiang@163.com
Information Engineering University
Zhengzhou, Henan, China

Zehui Wu
Information Engineering University
Zhengzhou, Henan, China

Wei Guo
Information Engineering University
Zhengzhou, Henan, China

ABSTRACT

The IoT devices have introduced vulnerabilities and new attack vectors, making many devices a prime target for cybercriminals, while enriching people's daily lives and industries. Vulnerability detection can effectively address this growing threat. However, due to variability of software and hardware, non-disclosure of source code and documentation, and limited resources of IoT devices, security analysis has never been an easy task. Although researchers have developed many new methods to overcome various challenges in the past decade, key challenges still hinder the practical application of firmware vulnerability mining. Therefore, this paper aims to systematically summarize existing work and analyze the challenges of this field and its solutions. Result: By summarizing the state-of-the-art approaches for static, dynamic, and hybrid analysis of IoT firmware and network service programs, we identify their advantages, disadvantages, and limitations. We found that network service programs are the main attack surface for 0-day vulnerability. Meanwhile, in the short term, static analysis and dynamic analysis are still mainstream techniques for vulnerability detection. Moreover, we point out that unique running workflow and environments are the biggest challenges for vulnerability detection. This survey serves as a reference for researchers and practitioners interested in IoT device security analysis and helps identify promising research directions for the future.

CCS CONCEPTS

• **Security and privacy** → **Vulnerability scanners**; **Embedded systems security**; • **Computer systems organization** → **Firmware**.

ACM Reference Format:
Xixing Li, Qiang Wei, Zehui Wu, and Wei Guo. 2023. A comprehensive survey of vulnerability detection method towards Linux-based IoT devices. In *2023 2nd International Conference on Networks, Communications and Information Technology (CNCIT 2023), June 16–18, 2023, Qinghai, China.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3605801.3605808

CNCIT 2023, June 16–18, 2023, Qinghai, China

ACM ISBN 979-8-4007-0062-0/23/06…$15.00
https://doi.org/10.1145/3605801.3605808

1 INTRODUCTION

The rapid development and widespread application of IoT devices have brought many conveniences to daily life. However, the security of IoT devices has also received increased attention as various security vulnerabilities have been discovered in these devices. Research has found that on average, each IoT device contains 25 vulnerabilities[43], with 70% of IoT devices being reported to have security vulnerabilities. Given the large number of these devices and their exposure to the Internet, IoT has become a breeding ground for various cyber attacks, posing a serious threat to the entire network security ecosystem.

Vulnerability discovery has been a research hotspot in recent years[1, 18, 26], and both the academic and industrial communities have proposed many effective methods and tools for vulnerability discovery. However, most of them are aimed at general software on desktop systems (Windows, Linux, Mac). In terms of vulnerability discovery for IoT devices, due to the huge differences in software and hardware among IoT devices of different manufacturers, as well as the non-disclosure of source code and documentation for IoT devices, the current methods and tools for IoT device vulnerability discovery still have many limitations, making it difficult to achieve efficient, automated, and batch vulnerability discovery.

This survey aims to summarize the core methods of vulnerability discovery for IoT devices and comb through the challenges faced in this field, as well as indicate the future development direction.

2 RESEARCH METHOD

To comprehensively summarize this field, the systematic literature reviews[2] method was adopted in this paper. We first presented the research questions to be studied, then determined the scope of literature selection, and finally presented the research results through analysis after summarizing the literature.

2.1 Research Question

We propose three research questions:

- RQ1: What are the main attack surfaces on IoT devices and what types of vulnerabilities exist?
- RQ2: How do existing literature explore vulnerabilities in embedded systems?

- RQ3: What is the difference between vulnerability detection on IoT devices and general software?

2.2 Literature Selection

This literature review selected papers from four renowned digital databases: (1) ACM Digital Library, (2) SpringerLink, (3) IEEE Xplore, and (4) DBLP. The search keywords including "IoT security", "vulnerability detection", "static analysis", "dynamic analysis", and so on. These keywords are related to the research question we proposed. We have collected a lot of papers. But IoT domain includes cloud platforms, smart terminals, IoT devices, etc. We only focus on detecting the vulnerability in IoT devices. Especially the most widely used Linux-based devices.

3 TAXONOMY OF APPLICATION DOMAINS

In recent years, there have been many studies on the security of IoT devices. Due to different definitions of vulnerabilities by different researchers, the boundaries of research are not similar. In order to standardize the discussion, this article limits the scope of vulnerabilities: 1. Assuming that attackers can only access and attack devices through the network. 2. Treating backdoors as a subclass of vulnerabilities for discussion. 3. The scope of IoT devices is limited to the most widely used Linux-based devices. As shown in the Table 1, existing vulnerability detection methods can be classified into three categories according to their purposes: known vulnerabilities, 0-day vulnerabilities, and backdoors.

Known vulnerabilities refer to vulnerabilities that have been publicly disclosed, but the vulnerability scope in the original disclosure does not involve the IoT device under analysis. The purpose of researchers is to confirm whether the device under analysis is affected by the vulnerability. The methods mainly used are static analysis and artificial intelligence analysis theory. An undisclosed vulnerability (also known as a 0-day vulnerability) refers to a vulnerability that has not been publicly disclosed. The purpose of researchers is to design various methods to discover these vulnerabilities for the first time. The methods mainly used are static analysis, dynamic analysis, and fuzz testing. Backdoors refer to defects in the access control layer such as hard-coded credentials and hidden interfaces in the device under analysis. The methods mainly used are static analysis.

4 STATIC ANALYSIS METHOD

Static analysis methods can discover vulnerabilities in IoT devices without executing the program. This process discovers errors by understanding the program code, so it is more scalable. However, due to not executing the code, there are issues of missed vulnerabilities and false positives. According to the concept of static vulnerability mining, as shown in the Table 2, it is mainly divided into pattern matching-based methods, code similarity comparison-based methods, and taint analysis-based methods.

Pattern matching-based method refers to identifying inherent patterns in vulnerabilities and converting them into rules for automated vulnerability mining. In 2014, Costin[10] et al. first proposed a large-scale automated firmware analysis method for embedded devices. This method automatically unpacks and processes the firmware, uses fuzzy hashing to match weak keys in the firmware, and finds the similarity between different firmware images across four different dimensions through association analysis. In 2015, Firmalice[29] generated a program dependency graph of the firmware using static program analysis, obtained an authentication slice from the entry point to the privileged program point, and used symbolic execution to determine if the constraints in the path were deterministic, which can be identified as a backdoor. In 2017, Stringer[31] proposed a method based on static data comparison analysis to detect undisclosed features and hard-coded authentication backdoors in COTS device firmware. In the same year, HumIDIFy[32] proposed a semi-automated method to detect hidden features in binary code of COTS embedded device firmware. This method uses a semi-supervised learning classifier to identify the binary code and features in the firmware, compares its expected functionality with a profile, and identifies unexpected functionality.

Code similarity comparison-based method means extracting features from vulnerable code, encoding them in a specific format, and then calculating the similarity between the encoded features to determine the presence of similar vulnerabilities in the firmware under analysis. In 2016, discovRE[14] used a control flow graph comparison method for similarity recognition and could detect known vulnerabilities in different architectures. Given a vulnerable binary function, similar functions can be found in binary programs of different compilers, optimization levels, operating systems, and CPU architectures. First, numerical features are extracted from the vulnerable function and filtered using machine learning methods. Then, combined with the structural features, similar functions are further screened out. In 2018, FirmUp[12] proposed to use normalized fragment representation programs to solve the problem of cross-compilers, cross-optimization options, and cross-architectures. Its main method is to split the function into basic block levels, then slice the basic block into smaller fragments, and normalize the register names and address offsets in this fragment. At the function unit, a table is made, and the number of identical code fragments in two tables is used as the basis for similarity comparison. VulSeeker[19] proposed a vulnerability search method based on semantic learning. For a given target function and vulnerability function, VulSeeker first constructs a labeled semantic flow graph and extracts their numerical vectors as the basic block features. Then, by feeding the numerical vectors of basic blocks into a customized DNN model with semantic-awareness, the embedding vectors of the entire binary function are generated. Finally, the cosine distance is used to measure the similarity between two binary functions. In 2016, Genius[16] proposed a graph search-based vulnerability code clone detection tool. Since directly comparing control flow graphs requires a higher overhead, the author proposed to convert control flow graphs into numerical feature vectors. Through vector comparison, code similarity detection between different IoT devices is accomplished. First, the control flow graph of the function is extracted using IDA, and then a password book is generated using clustering algorithms. Based on the password book, the characteristics are encoded to obtain the final feature database, and similar vulnerabilities are found through indexing the database. In 2017, Xu[35] and colleagues proposed Gemini based on Genius, which also utilizes graph matching algorithms for similarity detection. Gemini employs deep neural networks to directly vectorize graphs,

Table 1: This table is a summary of the vulnerability detection methods classified by the purpose

Detection Purpose	Vulnerability Detection Methods	Basic Technique
Known Vulnerability	FirmUp[12],Costin[10],DiscovRE[14], Genius[16], Gemini[35], Vulseeker[19], IoTseeker[20],IHB[7]	Static Analysis, machine-learning-based Analysis
Undisclosed Vulnerability	IoTFuzzer[5], Snipuzz[17], Diane[27], FirmAFL[41], SRFuzzer[39],ARMAFL[15], FirmCorn[22], RPFuzzer[34], FirmFuzz[30], Fw-Fuzz[21], SATC[6] KARONTE[28], DTaint[8], EmTaint[9], FIE[13], WMIFuzzer[33], Zheng[42], Yao[36]	Static Analysis, Dynamic Analysis, Fuzzing
Backdoor	Firmalice[29], HumIDIFy[32], Stringer[31],Costin[10], Costin2[11]	Static Analysis

Table 2: This table is a summary of the static analysis methods

	Literature	Target	Technology	Purpose	Types of Vulnerabilities	Large Scale
pattern matching-based method	Costin[10]	Linux-based devices	correlation analysis	backdoor	week authorization, default certificate	✓
	Stringer[31]	COTS devices	static data comparison	backdoor	undocumented functionality, hardcoded backdoor	✓
	HumIDIFy[32]	COTS devices	semi-supervised learning	backdoor	hidden functionality, backdoor	✓
	Firmalice[29]	COTS devices	symbolic execution, program slicing	backdoor	authentication bypass	✗
	CryptoREX[38]	COTS devices	IR-based analysis	0day vulnerability	Crypto misuse	✓
taint analysis-based method	KARONTE[28]	COTS devices	multi-binary interaction, taint analysis	0day vulnerability	taint style vulnerabilities (BOF/ CI)	✓
	SainT[3]	IoT app	taint analysis	0day vulnerability	sensitive information leak	✓
	EmTaint[9]	COTS devices	taint analysis, structured symbolic expression	0day vulnerability	taint style vulnerabilities (BOF/ CI)	✓
	DTaint[8]	COTS devices	taint analysis	0day vulnerability	taint style vulnerabilities (BOF/ CI)	✓
	IoTFuzzer[5]	IoT app	taint analysis blackbox fuzzing	0day vulnerability	extract protocol param	✗
code similarity comparison-based method	discovRE[14]	binary program	control flow graph comparison	known vulnerability	detecting the vulnerabilities similar to original input	✓
	FirmUp[12]	Firmware	IR-based analysis	known vulnerability		✓
	VulSeeker[19]	Binary program	semantic feature machine-learning-based Analysis	known vulnerability		✓
	Genius[16]	Binary program	CFG feature machine-learning-based Analysis	known vulnerability		✓
	Gemini[35]	Binary program	ACFG feature machine-learning-based Analysis	known vulnerability		✓

transforming graph comparison into vector distance comparison, thus further enhancing detection efficiency.

Taint analysis-based method is to discover taint-style vulnerabilities by analyzing the data dependency relationships between program variables to determine whether data can propagate from a taint source to a taint sink. DTaint[8] converted firmware into intermediate representation in 2018. The conversion process identified pointer aliasing and indirect calls for each function and generated a data flow graph from process kernels and processes. Using this data flow graph, sinks were identified, and a depth-first search algorithm was used to backtrack from sinks to sources to detect the existence of taint vulnerabilities. Along these paths, taint data constraints were checked to identify vulnerabilities. Similarly, SainT[3] detected privacy-related vulnerabilities in IoT apps in 2018 by tracing the information flow between sources and external sinks. Information flow tracing was used to identify privacy-sensitive data that may be exposed to outside attackers. In 2018, IoTFuzzer[5] employed fuzz testing technology alongside taint analysis techniques to uncover vulnerabilities. Specifically, taint analysis was conducted on apps associated with devices to detect frequently used hard-coded strings, user inputs, and system APIs in command messages of IoT applications. Any functions that employed these protocol fields as parameters were logged to enable input mutation. Karonte[28] introduced a static analysis approach in 2020 to analyze embedded device firmware, which modeled and tracked interactions between multiple binary programs. Taint information was propagated across binary programs to detect insecure interactions and identify vulnerabilities. In 2021, SATC[6] used a shared variable naming pattern between frontend files and backend functions to precisely locate input-related code in backend binary files. Static taint analysis was then applied to mine vulnerabilities with taint-style based on keywords. In 2022, EmTaint[9] proposed a novel approach combining sanitization rule checks and static pollution analysis based on structured symbolic expressions to effectively detect indirect calls and mine pollution-style vulnerabilities.

5 DYNAMIC ANALYSIS METHOD

Dynamic analysis is a method to analyze devices in operation to discover vulnerabilities. The actual execution environment can be divided into real devices and emulation platforms.

The Table 3 demonstrates that Firmadyne[4] and FirmAE[24] do not directly involve in vulnerability mining, but they mitigate the simulation issues present in Linux-based devices that solely rely on firmware, ultimately laying the groundwork for model-based dynamic analysis. Dynamic analysis commonly utilizes Fuzzing, a well-established vulnerability mining method utilized in desktop operating systems.

In 2013, RPFuzzer[34] developed a framework to uncover router protocol vulnerabilities and introduced a two-stage test case generator (TFTCG) utilizing a math model to enhance prior test case generation methods. Should an exception arise, RPFuzzer's debugger, constructed on a modified Dynamips platform, records register values. During SNMP protocol testing, the exploitation of RPFuzzer found eight vulnerabilities, five of which were zero-day vulnerabilities. In 2018, IoTFuzzer[5] automated the parsing of program logic in official mobile applications for IoT devices, generating test messages compliant to the protocol of the tested devices, and evaluated the effectiveness of fuzz testing through automated heartbeat packets and their respective information. Subsequently in 2019, FirmFuzz[30] used a headless web browser controlled by the Fuzzer as a proxy server to capture and mutate input data. In addition, FIRM-AFL[41] became the pioneer in the high-throughput gray-box fuzzer application in IoT firmware in 2019. It took full advantage of the high throughput characteristics of QEMU's user mode, as well as the high compatibility characteristics of QEMU's system mode, allowing for high-throughput grey-box fuzzy testing with program execution coverage in IoT firmware. The experiment results indicated that FIRM-AFL was 8.2 times faster than the full-system gray-box testing program TriforceAFL. In 2019, SRFuzzer[39] parsed the collected network and performed targeted mutations to conduct fuzz testing. It expanded its capability for survival detection, as well as incorporated response-based monitors (to detect front-end triggered XSS vulnerabilities), routing-based monitors (to detect command injection vulnerabilities), and signal-based monitors (to detect memory corruption vulnerabilities by monitoring SIGSEGV and SIGABRT through ptrace). Then in 2021, Snipuzz[17] presented a novel automated black-box fuzz testing approach to overcome the challenge of applying syntax-based fuzz testing on various non-standard communication protocol formats widely used in IoT devices. Snipuzz acted as a client for device communication, inferred message fragments based on responses, and performed mutations. Each message fragment represents the contiguous bytes that reflect the approximate code coverage feedback. Using this message fragment-based mutation strategy, Snipuzz substantially reduced the search space. Evaluating 20 genuine IoT devices, the experiment identified five zero-day vulnerabilities, of which three were exclusively detected by SNIPUZZ. In 2021, Diane[27] has detected the existence of fuzzing triggers in IoT companion applications. These triggers execute after input validation but prior to data encoding conversion, enabling the creation of inputs that comply with protocol standards but lack constraints for conducting fuzz testing. Diane has created a tool that employs a combination of static and dynamic analyses to identify triggers within the Android companion app and then repurposes them to automatically conduct device fuzz testing.

6 RESULT

6.1 Network service programs are the main attack surface for 0-day vulnerability

The fundamental purpose of the Internet of Things is to interconnect all physical devices via the internet[37]. As a result, the majority of IoT devices feature one or more network service programs that provide external users with various services[28]; these programs inevitably attract malicious attacks from hackers. Although the basic components of IoT devices are typically developed with open-source components and are comparatively rigorously tested, network service programs are the manufacturer's core business and are generally developed in-house with less extensive testing. Consequently, these programs are often vulnerable to 0-day vulnerabilities, and are the most vulnerable attack surface due to their insufficient code robustness. Notably, once vulnerabilities are revealed, the impact can be catastrophic because they can be remotely

Table 3: This table is a summary of the dynamic analysis methods

Literature	Target	Technology	Dependence			Types of Vulnerabilities	Architecture
			Real Device	Firmware	App		
Firmadyne[4]	Linux-based devices	full system emulation		✓		emulation purpose	ARM, MIPS
FirmAE[24]	Linux-based devices	arbitration emulation		✓		emulation purpose	ARM, MIPS
FIRMAFL[41]	Linux-based devices	greybox fuzzing		✓		BOF/NPD	ARM, MIPS
IoTFuzzer[5]	IP-Cam/NAS/router	blackbox fuzzing	✓		✓	BOF/NPD	ARM/MIPS
RPFuzzer[34]	cisco router	greybox fuzzing		✓		DoS	MIPS
SRFuzzer[39]	router	blackbox fuzzing	✓			BOF/NPD/XSS	ARM/MIPS
Snipuzz[17]	IP-Cam/NAS/router	blackbox fuzzing message snippet inference	✓			NPD/DoS	ARM/MIPS
Diane[27]	IP-Cam/smart home	blackbox fuzzing app static analysis	✓		✓	BOF/DoS	ARM/MIPS
FirmFuzz[30]	IP-Cam/router	taint analysis, structured symbolic expression		✓		BOF/NPD/CI/XSS	ARM
Fw-Fuzz[21]	router	greybox fuzzing	✓			BOF/CI	MIPS/ARM/PPC
WMIFuzzer[33]	router	blackbox fuzzing	✓			BOF/NPD	ARM/MIPS
FirmCorn[22]	router	greybox fuzzing		✓		BOF/NPD	ARM/MIPS

targeted, rendering the network service programs among the most fragile attack surfaces in IoT devices.

According to a summary of current research, it has been found that during the vulnerability mining process, both dynamic analysis techniques (FirmAFL[41], Snipuzz[17], SRFuzzer[39], WMIFuzzer[33]) and static analysis techniques (SATC, KARONTE) focus primarily on network service programs as the main object of analysis.

In recent years, researchers[6, 25] have focused on optimizing general vulnerability mining methods for network service programs in devices. SATC[6] discovered that there is a phenomenon of shared keywords between the front-end and back-end of network service programs and utilized this to improve static analysis. WMIFuzzer reduces the blind spots of fuzzing through behavior analysis of web management interfaces in IoT device. However, we believe that current academic research on the characteristics of network service programs in devices lacks sufficient depth. Further exploration in this area holds great potential for enhancing device vulnerability mining capabilities in the future. It has been discovered that in Linux-based devices, third-party components are a significant source of attack beyond network service programs. Due to the high hardware and software coupling in IoT devices, various third-party components[40] (such as the linux operating system, busybox, libssl, and libjson) exist on the system. Although these components are not directly exposed to the network, they tend to have lower versions and more known vulnerabilities, making them vulnerable to hackers' quick discovery and indirect exploitation. Currently, many studies focus on rapidly identifying these known vulnerabilities. Furthermore, manufacturers are paying increasing attention to the harm caused by third-party components and actively maintaining updates in devices. Thus, the threat posed by known vulnerabilities in third-party components is likely to decrease in the future.

6.2 Static analysis and dynamic analysis are still mainstream analysis techniques in the short term

Static and dynamic analysis methods each have their advantages and disadvantages, suitable for specific analysis scenarios, and cannot replace each other. Static analysis requires less dependence on analysts as they only need to obtain the device firmware and offers a broad scope of analysis. However, it suffers from misreporting and omission of vulnerabilities. On the other hand, dynamic analysis has more reliable vulnerability detection but lacks comprehensiveness due to being limited to execution paths. IoT devices are highly coupled with hardware and software, making it challenging to run dynamic analysis independently. Currently, dynamic analysis faces challenges in deploying plug-in tools that can run on both real and simulated platforms and support multiple analysis requirements[18]. Furthermore, AI-based analysis[23] methods are emerging as a promising direction to aid in both static and dynamic analysis; however, the establishment of a large enough dataset is a significant challenge.

6.3 Unique running workflow and environments are the biggest challenges for vulnerability discover

The unique running workflow and environments of IoT devices distinguishes them from regular software. IoT devices rely on multiple programs cooperating with each other to offer various external services over a network, which contrasts with widely studied general

programs in common security analysis methods. General programs often receive inputs through files or standard I/O and provide a single function. Furthermore, incorporating mobile apps in IoT device operation drives researchers to innovate vulnerability mining theories. Modeling this unconventional operational mode poses considerable challenges that require optimization of vulnerability mining methods. Regarding the system environment, general programs are hardware independent and, on the desktop system, the degree of openness and computational resources are abundant, allowing convenient deployment of various dynamic analysis methods. However, IoT devices have limited computational resources and lack secondary development interfaces, causing difficulties in directly applying many matured tools in desktop systems.

REFERENCES

[1] Omar Alrawi, Chaz Lever, Manos Antonakakis, and Fabian Monrose. 2019. SoK: Security evaluation of home-based IoT deployments. In *2019 IEEE symposium on security and privacy, SP 2019, san francisco, CA, USA, may 19-23, 2019*. IEEE, 1362–1380. https://doi.org/10.1109/SP.2019.00013 tex.bibsource: dblp computer science bibliography, https://dblp.org tex.biburl: https://dblp.org/rec/conf/sp/AlrawiLAM19.bib tex.timestamp: Wed, 16 Oct 2019 14:14:51 +0200.

[2] David Budgen and Pearl Brereton. 2006. Performing systematic literature reviews in software engineering. In *Proceedings of the 28th international conference on Software engineering (ICSE '06)*. Association for Computing Machinery, New York, NY, USA, 1051–1052. https://doi.org/10.1145/1134285.1134500

[3] Z. Berkay Celik, Leonardo Babun, Amit Kumar Sikder, Hidayet Aksu, Gang Tan, Patrick D. McDaniel, and A. Selcuk Uluagac. 2018. Sensitive information tracking in commodity IoT. In *27th USENIX security symposium, USENIX security 2018, baltimore, MD, USA, august 15-17, 2018 (USENIX'18)*, William Enck and Adrienne Porter Felt (Eds.). USENIX Association, 1687–1704. https://www.usenix.org/system/files/conference/usenixsecurity18/sec18-celik.pdf tex.bibsource: dblp computer science bibliography, https://dblp.org tex.biburl: https://dblp.org/rec/conf/uss/CelikBSATMU18.bib tex.timestamp: Mon, 01 Feb 2021 08:43:20 +0100.

[4] Daming D Chen, Maverick Woo, David Brumley, and Manuel Egele. 2016. Towards Automated Dynamic Analysis for Linux-based Embedded Firmware. In *23rd Annual Network and Distributed System Security Symposium, NDSS 2016, San Diego, California, USA, February 21-24, 2016 (NDSS'16)*. The Internet Society. https://doi.org/10.14722/ndss.2016.23415

[5] Jiongyi Chen, Wenrui Diao, Qingchuan Zhao, Chaoshun Zuo, Zhiqiang Lin, XiaoFeng Wang, Wing Cheong Lau, Menghan Sun, Ronghai Yang, and Kehuan Zhang. 2018. IoTFuzzer: Discovering Memory Corruptions in IoT Through App-based Fuzzing. In *Proceedings 2018 Network and Distributed System Security Symposium*. https://doi.org/10.14722/ndss.2018.23159 Issue: February.

[6] Libo Chen, Yanhao Wang, Quanpu Cai, Yunfan Zhan, Hong Hu, Jiaqi Linghu, Qinsheng Hou, Chao Zhang, Haixin Duan, and Zhi Xue. 2021. Sharing More and Checking Less: Leveraging Common Input Keywords to Detect Bugs in Embedded Systems *(Security 21)*. 303–319. https://www.usenix.org/conference/usenixsecurity21/presentation/chen-libo

[7] Yu Chen, Hong Li, Weiwei Zhao, Lin Zhang, Zhongjin Liu, and Zhiqiang Shi. 2017. IHB: A scalable and efficient scheme to identify homologous binaries in IoT firmwares. In *2017 IEEE 36th International Performance Computing and Communications Conference (IPCCC)*. 1–8. https://doi.org/10.1109/PCCC.2017.8280478 ISSN: 2374-9628.

[8] Kai Cheng, Qiang Li, Lei Wang, Qian Chen, Yaowen Zheng, Limin Sun, and Zhenkai Liang. 2018. DTaint: Detecting the Taint-Style vulnerability in embedded device firmware. In *Proceedings - 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2018 (DSN'18)*. IEEE, 430–441. https://doi.org/10.1109/DSN.2018.00052 ISSN: 2158-3927.

[9] Kai Cheng, Tao Liu, Le Guan, Peng Liu, Hong Li, Hongsong Zhu, and Limin Sun. 2022. Finding Taint-Style Vulnerabilities in Linux-based Embedded Firmware with SSE-based Alias Analysis. *ArXiv* (2022).

[10] Andrei Costin, Jonas Zaddach, Aurélien Francillon, and Davide Balzarotti. 2014. 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 (Security 14)*, Kevin Fu and Jaeyeon Jung (Eds.). USENIX Association, 95–110. https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/costin tex.bibsource: dblp computer science bibliography, https://dblp.org tex.biburl: https://dblp.org/rec/conf/uss/CostinZFB14.bib tex.timestamp: Mon, 01 Feb 2021 08:43:17 +0100.

[11] Andrei Costin, Apostolis Zarras, and Aurélien Francillon. 2016. 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 (ASIA CCS '16)*. Association for Computing Machinery, New York, NY, USA, 437–448. https://doi.org/10.1145/2897845.2897900

[12] Yaniv David, Nimrod Partush, and Eran Yahav. 2018. 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 '18, Vol. 53)*. Association for Computing Machinery, New York, NY, USA, 392–404. https://doi.org/10.1145/3173162.3177157 ISSN: 15232867 Issue: 2.

[13] Drew Davidson, Benjamin Moench, Thomas Ristenpart, and Somesh Jha. 2013. FIE on firmware: Finding vulnerabilities in embedded systems using symbolic execution. In *Proceedings of the 22th USENIX security symposium, washington, DC, USA, august 14-16, 2013*, Samuel T. King (Ed.). USENIX Association, 463–478. https://www.usenix.org/conference/usenixsecurity13/technical-sessions/paper/davidson tex.bibsource: dblp computer science bibliography, https://dblp.org tex.biburl: https://dblp.org/rec/conf/uss/DavidsonMRJ13.bib tex.timestamp: Mon, 01 Feb 2021 08:43:12 +0100.

[14] Sebastian Eschweiler, Khaled Yakdan, and Elmar Gerhards-Padilla. 2016. discovRE: Efficient Cross-Architecture Identification of Bugs in Binary Code. In *23rd Annual Network and Distributed System Security Symposium, NDSS 2016, San Diego, California, USA, February 21-24, 2016*. The Internet Society. https://doi.org/10.14722/ndss.2016.23185

[15] Rong Fan, Jianfeng Pan, and Shaomang Huang. 2020. ARM-AFL: Coverage-Guided Fuzzing Framework for ARM-Based IoT Devices. In *Applied Cryptography and Network Security Workshops - ACNS 2020 Satellite Workshops, AIBlock, AIHWS, AIoTS, Cloud S&P, SCI, SecMT, and SiMLA, Rome, Italy, October 19-22, 2020, Proceedings (Lecture Notes in Computer Science, Vol. 12418)*, Jianying Zhou, Mauro Conti, Chuadhry Mujeeb Ahmed, Man Ho Au, Lejla Batina, Zhou Li, Jingqiang Lin, Eleonora Losiouk, Bo Luo, Suryadipta Majumdar, Weizhi Meng, Martín Ochoa, Stjepan Picek, Georgios Portokalidis, Cong Wang, and Kehuan Zhang (Eds.). Springer, 239–254. https://doi.org/10.1007/978-3-030-61638-0_14

[16] Qian Feng, Rundong Zhou, Chengcheng Xu, Yao Cheng, Brian Testa, and Heng Yin. 2016. Scalable Graph-based Bug Search for Firmware Images. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi (Eds.). ACM, 480–491. https://doi.org/10.1145/2976749.2978370

[17] Xiaotao Feng, Ruoxi Sun, Xiaogang Zhu, Minhui Xue, Sheng Wen, Dongxi Liu, Surya Nepal, and Yang Xiang. 2021. Snipuzz: Black-box Fuzzing of IoT Firmware via Message Snippet Inference. In *CCS'21: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS'21, Vol. 1)*. Association for Computing Machinery. https://doi.org/10.1145/3460120.3484543 Publication Title: Proceedings of ACM Conference on Computer and Communications Security (Anonymous Submission to ACM CCS 2021) _eprint: 2105.05445.

[18] Xiaotao Feng, Xiaogang Zhu, Qing-Long Han, Wei Zhou, Sheng Wen, and Yang Xiang. 2023. Detecting Vulnerability on IoT Device Firmware: A Survey. *IEEE/CAA Journal of Automatica Sinica* 10, 1 (Jan. 2023), 25–41. https://doi.org/10.1109/JAS.2022.105860 Conference Name: IEEE/CAA Journal of Automatica Sinica.

[19] Jian Gao, Xin Yang, Ying Fu, Yu Jiang, and Jiaguang Sun. 2018. VulSeeker: a semantic learning based vulnerability seeker for cross-platform binary. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. Association for Computing Machinery, New York, NY, USA, 896–899. https://doi.org/10.1145/3238147.3240480

[20] Jian Gao, Xin Yang, Yu Jiang, Houbing Song, Kim-Kwang Raymond Choo, and Jiaguang Sun. 2021. Semantic Learning Based Cross-Platform Binary Vulnerability Search For IoT Devices. *IEEE Transactions on Industrial Informatics* 17, 2 (Feb. 2021), 971–979. https://doi.org/10.1109/TII.2019.2947432 Conference Name: IEEE Transactions on Industrial Informatics.

[21] Zicong Gao, Weiyu Dong, Rui Chang, and Yisen Wang. 2020. Fw-fuzz: A code coverage-guided fuzzing framework for network protocols on firmware: NA. *Concurrency and Computation: Practice and Experience* 34 (April 2020). https://doi.org/10.1002/cpe.5756

[22] Zhijie Gui, Hui Shu, Fei Kang, and Xiaobing Xiong. 2020. FIRMCORN: Vulnerability-oriented fuzzing of IoT firmware via optimized virtual execution. *IEEE Access* 8 (2020), 29826–29841. https://doi.org/10.1109/ACCESS.2020.2973043 tex.bibsource: dblp computer science bibliography, https://dblp.org tex.biburl: https://dblp.org/rec/journals/access/GuiSKX20.bib tex.timestamp: Tue, 03 Mar 2020 09:38:04 +0100.

[23] Irfan Ul Haq and Juan Caballero. 2021. A Survey of Binary Code Similarity. *Comput. Surveys* 54, 3 (April 2021), 51:1–51:38. https://doi.org/10.1145/3446371

[24] 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 (ACSAC '20)*. Association for Computing Machinery, New York, NY, USA, 733–745. https://doi.org/10.1145/3427228.3427294

[25] Marius Muench, Jan Stijohann, Frank Kargl, Aurelien Francillon, and Davide Balzarotti. 2018. What You Corrupt Is Not What You Crash: Challenges in Fuzzing Embedded Devices. In *Proceedings 2018 Network and Distributed System Security Symposium (NDSS'18)*. https://doi.org/10.14722/ndss.2018.23166

[26] Ibrahim Nadir, Haroon Mahmood, and Ghalib Asadullah. 2022. A taxonomy of IoT firmware security and principal firmware analysis techniques. *International Journal of Critical Infrastructure Protection* 38, C (Sept. 2022). https://doi.org/10.1016/j.ijcip.2022.100552

[27] Nilo Redini, Andrea Continella, Dipanjan Das, Giulio De Pasquale, Noah Spahn, Aravind Machiry, Antonio Bianchi, Christopher Kruegel, Giovanni Vigna, Giulio De Pasquale, Noah Spahn, Aravind Machiry, Antonio Bianchi, Christopher Kruegel, and Giovanni Vigna. 2021. Diane: Identifying Fuzzing Triggers in Apps to Generate Under-constrained Inputs for IoT Devices. In *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*, Vol. 2021-May. IEEE, 484–500. https://doi.org/10.1109/SP40001.2021.00066 ISSN: 10816011.

[28] Nilo Redini, Aravind Machiry, Ruoyu Wang, Chad Spensky, Andrea Continella, Yan Shoshitaishvili, Christopher Kruegel, and Giovanni Vigna. 2020. Karonte: Detecting Insecure Multi-binary Interactions in Embedded Firmware. In *In Proceedings of the IEEE Symposium on Security & Privacy (S&P)*. 1544–1561. https://doi.org/10.1109/sp40000.2020.00036

[29] Yan Shoshitaishvili, Ruoyu Wang, Christophe Hauser, Christopher Kruegel, and Giovanni Vigna. 2015. Firmalice - Automatic Detection of Authentication Bypass Vulnerabilities in Binary Firmware. In *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015*. The Internet Society. https://doi.org/10.14722/ndss.2015.23294

[30] Prashast Srivastava, Hui Peng, Jiahao Li, Hamed Okhravi, Howard Shrobe, and Mathias Payer. 2019. FirmFuzz: automated IoT firmware introspection and analysis. In *Proceedings of the 2nd International ACM Workshop on Security and Privacy for the Internet-of-Things*. 15–21.

[31] Sam L. Thomas, Tom Chothia, and Flavio D. Garcia. 2017. Stringer: Measuring the Importance of Static Data Comparisons to Detect Backdoors and Undocumented Functionality. In *Computer Security - ESORICS 2017 - 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 10493)*, Simon N. Foley, Dieter Gollmann, and Einar Snekkenes (Eds.). Springer, 513–531. https://doi.org/10.1007/978-3-319-66399-9_28

[32] Sam L. Thomas, Flavio D. Garcia, and Tom Chothia. 2017. HumIDIFy: A Tool for Hidden Functionality Detection in Firmware. In *Detection of Intrusions and Malware, and Vulnerability Assessment - 14th International Conference, DIMVA 2017, Bonn, Germany, July 6-7, 2017, Proceedings (Lecture Notes in Computer Science, Vol. 10327)*, Michalis Polychronakis and Michael Meier (Eds.). Springer, 279–300. https://doi.org/10.1007/978-3-319-60876-1_13

[33] Dong Wang, Xiaosong Zhang, Ting Chen, and Jingwei Li. 2019. Discovering Vulnerabilities in COTS IoT Devices through Blackbox Fuzzing Web Management Interface. In *Innovative Mobile and Internet Services in Ubiquitous Computing*, Prosanta Gope (Ed.), Vol. 2019. Hindawi, 5076324. https://doi.org/10.1155/2019/5076324 ISSN: 1939-0114.

[34] Zhiqiang Wang, Yuqing Zhang, and Qixu Liu. 2013. RPFuzzer: A Framework for Discovering Router Protocols Vulnerabilities Based on Fuzzing. *KSII Trans. Internet Inf. Syst.* 7, 8 (2013), 1989–2009. https://doi.org/10.3837/tiis.2013.08.014

[35] Xiaojun Xu, Chang Liu, Qian Feng, Heng Yin, Le Song, and Dawn Song. 2017. Neural network-based graph embedding for cross-platform binary code similarity detection. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security, CCS 2017, dallas, TX, USA, october 30 - november 03, 2017 (CCS'17)*, Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM, 363–376. https://doi.org/10.1145/3133956.3134018 tex.bibsource: dblp computer science bibliography, https://dblp.org tex.biburl: https://dblp.org/rec/conf/ccs/XuLFYSS17.bib tex.timestamp: Tue, 10 Nov 2020 19:59:50 +0100.

[36] Min Yao, Baojiang Cui, and Chen Chen. 2020. Research on IoT Device Vulnerability Mining Technology Based on Static Preprocessing and Coloring Analysis. In *Innovative Mobile and Internet Services in Ubiquitous Computing (Advances in Intelligent Systems and Computing)*, Leonard Barolli, Aneta Poniszewska-Maranda, and Hyunhee Park (Eds.). Springer International Publishing, Cham, 254–263. https://doi.org/10.1007/978-3-030-50399-4_25

[37] Chi Zhang, Yu Wang, and Linzhang Wang. 2020. Firmware Fuzzing: The State of the Art. In *12th Asia-Pacific Symposium on Internetware (Internetware'20)*. Association for Computing Machinery, New York, NY, USA, 110–115. https://doi.org/10.1145/3457913.3457934

[38] Li Zhang, Jiongyi Chen, Wenrui Diao, Shanqing Guo, Jian Weng, and Kehuan Zhang. 2019. CryptoREX: Large-scale Analysis of Cryptographic Misuse in IoT Devices. 151–164. https://www.usenix.org/conference/raid2019/presentation/zhang-li

[39] Yu Zhang, Wei Huo, Kunpeng Jian, Ji Shi, Haoliang Lu, Longquan Liu, Chen Wang, Dandan Sun, Chao Zhang, and Baoxu Liu. 2019. SRFuzzer: an automatic fuzzing framework for physical SOHO router devices to discover multi-type vulnerabilities. In *Proceedings of the 35th Annual Computer Security Applications Conference, {ACSAC} 2019, San Juan, PR, USA, December 09-13, 2019 (ACSAC'19)*, David Balenson (Ed.). ACM, 544–556. https://doi.org/10.1145/3359789.3359826

[40] Binbin Zhao, Shouling Ji, Jiacheng Xu, Yuan Tian, Qiuyang Wei, Qinying Wang, Chenyang Lyu, Xuhong Zhang, Changting Lin, Jingzheng Wu, and Raheem Beyah. 2022. A large-scale empirical analysis of the vulnerabilities introduced by third-party components in IoT firmware. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2022)*. Association for Computing Machinery, New York, NY, USA, 442–454. https://doi.org/10.1145/3533767.3534366

[41] 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*. 1099–1114.

[42] Yaowen Zheng, Yuekang Li, Cen Zhang, Hongsong Zhu, Yang Liu, and Limin Sun. 2022. Efficient greybox fuzzing of applications in Linux-based IoT devices via enhanced user-mode emulation. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2022)*. Association for Computing Machinery, New York, NY, USA, 417–428. https://doi.org/10.1145/3533767.3534414

[43] Yaowen Zheng, Zhanwei Song, Yuyan Sun, Kai Cheng, Hongsong Zhu, and Limin Sun. 2019. An Efficient Greybox Fuzzing Scheme for Linux-based IoT Programs Through Binary Static Analysis. In *2019 IEEE 38th International Performance Computing and Communications Conference (IPCCC)*. 1–8. https://doi.org/10.1109/IPCCC47392.2019.8958740 ISSN: 2374-9628.