Research on Automated Vulnerability Mining of Embedded System Firmware

Xiaoyi Li[1], Lijun Qiao[2], Yanbin Sun[1(✉)], and Quanlong Guan[3]

[1] Cyberspace Institute of Advanced Technology, Guangzhou University, Guangzhou, China
sunyanbin@gzhu.edu.cn
[2] The People's Armed Police Sergeant School, Beijing, China
[3] Jinan University, Guangzhou, China

Abstract. The development of the Internet of Things (IoT) makes people pay more and more attention to the security of embedded systems. The most important of it is the security issues brought by firmware. The threat posed by firmware vulnerabilities is fatal. Researching firmware vulnerability mining technology is a way to effectively protect embedded systems. However, it's not easy to move the software's vulnerability mining methods to firmware. The existing firmware vulnerability mining work can effectively solve some problems, but it still has some shortcomings. In this paper, we first summarize the main challenges of firmware research. Then we analyze the work related to firmware vulnerability mining. After that, we classify and analyze the existing firmware vulnerability mining work from two aspects of method and technology. At the same time, we have made some performance comparisons on the exiting work. Finally, we give some suggestions on the future direction of the firmware vulnerability mining work.

Keywords: Firmware · Embedded system · Vulnerability mining

1 Introduction

Nowadays, with the development of IoT, more and more embedded devices are connected to the Internet. In the trend of the IoT, embedded devices are almost everywhere. They penetrate into every aspect of our lives and play a decisive role in the future of IoT security. Today, the security of embedded systems is getting more and more attention, and the core of embedded system security is firmware.

The term firmware is defined as a layer of bonded microcode between the CPU instruction set and the actual hardware in a Datamation article written by Opler A [1] in 1967. However, this definition is gradually expanded to the level of computer data with the development of computer hardware devices. The firmware is given a new meaning in the IEEE Standard Glossary of Software Engineering Terminology, Std 610.12-1990 [2]: The combination of a hardware device and computer instructions and data that reside as read-only software on that device.

X. Sun et al. (Eds.): ICAIS 2020, CCIS 1254, pp. 105–117, 2020.
https://doi.org/10.1007/978-981-15-8101-4_11

After that, the rise of the IoT has allowed the definition of firmware to be developed again. According to Zhang [3], firmware refers to a binary program stored in a non-volatile memory (e.g., FLASH, ROM) of an embedded device, which is non-volatile and solidified. Zaddach J et al. [4] generalize the firmware to all code sets (machine code and virtual machine code) running on the hardware processor. These definitions introduce the features of the firmware from different aspects, but their core content is actually the same: firmware is the set of software that makes an embedded system functional.

Embedded devices are mostly controlled by firmware, which is usually provided by the device vendor and has strong specificity and privacy. Unfortunately, these device vendors typically do not consider security when designing firmware. This makes millions of homes and small businesses face known and unknown threats on the network at all times. With these vulnerabilities, an attacker can easily control and destroy a device. Tor hidden services is also a new security issue. Q. Tan et al. [5] present practical Eclipse attacks on Tor HSs that allow an adversary with an extremely low cost to block arbitrary Tor HSs. A firmware vulnerability is fatal for some equipment related to social infrastructure, which seriously threatens people's lives and property. Therefore, we must accurately identify the vulnerability in the firmware. Although the existing firmware vulnerability mining research has achieved results in some aspects, there are still some limitations.

In summary this paper makes the following contributions:

- We point out the challenges brought by the firmware.
- We review and compare the work related to firmware vulnerability mining.
- We classify the related work from two aspects of method and technology.
- We analyze the technical route of the work including the simulator.
- We summarize the future work and development direction of firmware vulnerability mining.

2 Challenges

The reason why firmware has many security issues and is difficult to detect is that firmware is essentially different from traditional software. These differences hinder the security of the firmware, so that the original mature software security policies and detection methods can not be applied to the firmware. At the same time, these differences are also challenges that we must overcome in the security research process. We summarize some important firmware challenges as follows.

2.1 Complex Format

The format of the firmware is a very complicated problem. Unlike traditional software, the firmware does not have a standard file format. In today's embedded device market, the firmware formats used by different vendors are different. Although the exact format of the firmware is difficult to determine, Zaddach J et al. classify the firmware into the following three categories based on the components and functions of the firmware:

- Full-blown (full-OS/kernel + bootloader + libs + apps).
- Integrated (apps + OS-as-a-lib).
- Partial updates (apps or libs or resources or support).

In addition, these objects can be arbitrarily grouped and packaged into various archives or file system images. The combination of different firmware categories and different packaging methods allows the firmware format to be changed at will, which greatly increases the complexity of the firmware format.

2.2 Update Difficulty

For firmware vendors, it is important to consider how to obtain more profit, and the security of firmware is something that will be considered after that. On the other hand, the diversity of platforms and applications increases the difficulty of compiling and maintaining. In addition, since some firmware involves important infrastructure, special inspection standards are required for supervision.

For users, the process of updating the firmware is not as easy as the software update. Updating the firmware sometimes requires the user to have some knowledge of the hardware device, and even need to learn the knowledge of the specific update software. To make matters worse, if the update process fails, it may cause device crash. And updating the firmware does not bring new features.

2.3 Diverse Architecture

Different from the hardware architecture of traditional computers, embedded devices have many choice. These architectures have their own unique features in various fields, which are difficult to replace for each vendor. And it is difficult to achieve uniformity in one aspect.

In terms of processor architecture, embedded devices are more diverse than traditional computers. ARM and MIPS processors are the most widely distributed. The choice of architecture for simple devices is diverse, such as PowerPC, ARC and Motorola 6800/68000 with smaller memory.

The use of the operating system is equally diverse. Complex devices usually use a mature operating system (e.g., Linux, Windows NT), and Linux is currently the most popular operating system. And simple devices use a proprietary real-time operating system (e.g., VxWorks). It even includes some questions about Internet of Vehicles. Z. Tian et al. [6] propose to consider dynamical and diversity attacking strategies in the simulation of reputation management scheme evaluation.

3 Review of Firmware Vulnerability Mining

In recent years, people gradually realize the significance and value of firmware vulnerability research, and urgently want to apply software vulnerability mining methods to the firmware. However, research on firmware vulnerability mining is not as smooth as imagined. We study the existing firmware vulnerability mining methods. Hou et al. [7]

and Xie et al. [8] have some good classification of existing methods. Based on these efforts, we reclassify existing research into the following categories from the method level. And we compare them in Table 1.

3.1 Static Analysis

Static analysis is a way to directly analyze firmware content to discover bugs in a program without having to execute the program on the actual device or simulator.

Costin et al. [9] propose a static analysis framework for the process of firmware collection, filtering, unpacking, and large-scale analysis. The framework determines whether it contains a private encryption key or a string of known errors by studying the firmware information extracted from each firmware sample. This work is tested in approximately 32,356 firmware images collected. It proves that 693 firmware images are affected by at least one vulnerability. Of these, 38 are newly discovered vulnerabilities. However, this approach faces the classic trade-off of static analysis. That is, the analysis of the firmware is too broad or too specific.

3.2 Symbolic Execution

Instead of specific variables, symbolic values are used to simulate each path to produce the possibility of each execution. Solving the mathematical expression of a result gives the path to the result.

FIE [10] is a major concern for memory security issues and is used to automatically analyze firmware vulnerability detection systems that are widely used in MSP430 microcontrollers. It compiles the source code of the analysis firmware into LLVM bytecode for analysis as input to the symbol execution. It is based on the KLEE [11] symbolic execution engine. FIE significantly improves code coverage with state pruning and memory blur. It can be used to discover two types of vulnerabilities. However, FIE is limited to analyzing small firmware written in C and must obtain firmware source code. And for the vulnerability reported by the system, the user must manually verify.

Firmalice [12] is a symbolic analysis system for analyzing binary code in complex firmware on different hardware platforms. It is based on the angr [13] symbol execution engine. Firmalice generates a program dependency graph for the firmware from the static analysis module and uses this graph to create an authentication slice from the entry point to the privileged program point. It attempts to find the path to the privileged program point and performs a certificate bypass check on the successfully arrived symbol state. After testing, Firmalice can effectively detect a large number of complex backdoors without relying on the implementation details of the firmware itself. However, Firmalice requires manual operation when providing security policies for devices, so it cannot be used for large-scale analysis.

Avatar [14] is a dynamic analysis framework based on embedded device firmware. It acts as a software agent between the embedded device and the simulator based on the S2E [15]. The firmware instructions are executed in the simulator, and I/O operations are introduced into the embedded device. The state is passed between the simulator and the device while the firmware is running. And the state remains the same when it is passed. The experimental results show that Avatar can play a good supporting role

for reverse engineering of firmware, vulnerability detection and hardcoded backdoor discovery. However, Avatar is much slower on the emulator than on the actual device. And Avatar relies on hardware devices.

3.3 Fuzzing

Fuzzing is to construct random and unintended malformed data as the input of the program, and monitor the abnormalities that may occur during the execution of the program.

Firmadyne [16] is a framework for dynamically analyzing vulnerabilities in Linux-based embedded firmware. It is an automated, scalable dynamic analysis technology. It collects firmware on the vendor's website. A binwalk script effectively implements the extraction of file systems and optional kernels. During the simulation phase, Firmadyne performed an initial simulation on the QEMU [17] simulator. This learning process, it will continually modify the network configuration for QEMU. Fimadyne provides three analysis channels to analyze firmware. Firmadyne can accurately simulate the firmware of network devices and has good versatility. However, Firmadyne uses a general-purpose kernel in the simulation process. This approach prevents it from analyzing vulnerabilities in kernel or kernel modules.

Dai et al. [18] propose a method based on using dynamic fuzzing and static taint tracing on the simulator to locate and exploit firmware vulnerabilities. This method uses risk weights to design a set of fuzzing cases, which improves code coverage and dynamic analysis capabilities. It constructs a taint propagation path graph by identifying and tracing tainted data sources for binary data. The taint path graph is then dynamically executed on the simulator and fuzzed to detect vulnerabilities in the firmware. However, this method also has certain limitations. The simulator cannot accurately simulate some firmware images that require hardware support.

3.4 Comprehensive Analysis

Comprehensive analysis refers to the use of several different methods for firmware vulnerability mining. And some tools are effectively integrated to provide more accurate analysis results.

Avatar2 [19] is a dynamic multi-target orchestration framework designed to support interoperability between any number of different dynamic binary analysis frameworks, debuggers, simulators, and physical devices. Avatar2 is a completely redesigned system compared to Avatar. It includes the Avatar2 kernel, targets, protocols, and endpoints. In the end, Avatar2 integrated five targets: GDB, OpenOCD [20], QEMU, PANDA [21], angr. These targets provide a large number of analytical combinations. However, the GDB stubs are highly dependent on the architecture of the analysis target and are difficult to abstract in a generic way.

3.5 Others

In addition to the above literature, the following literature also contributes to the firmware vulnerability mining and can be classified by the above methods, which is not described in detail here.

Bojinov et al. [22] propose a vulnerability scanning work for the embedded web interface of IoT devices. They scan a total of 21 devices. More than 40 new vulnerabilities are discovered and a new type of web vulnerability called XCS is discovered.

FEMU [23] proposes a simulation framework that mixes firmware and hardware. It implements consistent SoC verification by using the ported QEMU at the BIOS level. But this method requires the support of hardware devices.

Hu et al. [24] study embedded firmware without file system. They discuss the problems of library function identification and backdoor detection. This method successfully identify the main contents of a real firmware and detect multiple backdoors.

Shang et al. [25] design a vulnerability analysis system for industrial embedded devices, including multiple analysis modules. It provides a theoretical approach to the development of vulnerability analysis systems for industrial control systems.

Li et al. [26] aim at the recurrence of the same vulnerability in the development process of embedded device firmware. Finally, they perform vulnerability detection on similar firmware based on the analysis result.

Genius [27] is a vulnerability search system based on digital feature search technology. It learns advanced features from control flow graphs. Genius tests in 8126 firmware and averages the search in less than a second.

Table 1. Performance comparison of firmware vulnerability mining

Tool/method	Architecture	Vulnerability type	Support	Large-scale analysis	Simulator	Vulnerability mining
Costin et al.	–	Backdoor	Homology	√	×	×
FIE	MSP430	Memory security	Source code	√	×	√
Firmalice	–	Backdoor	–	×	×	√
Avatar	–	–	Device	×	√	√
Firmadyne	Linux (ARM, MIPS)	–	–	√	√	×
Dai et al.	ARM, MIPS	–	–	×	√	√
Avatar2	–	–	–	×	√	√

4 Technical Route

The above firmware vulnerability mining methods can be divided into two categories from the technical level: No-simulator and Simulator.

No-simulator usually rely on firmware source code (e.g., FIE). However, the firmware source code is usually not provided by the vendor. And it becomes the biggest bottleneck

of this type of method. Methods that do not use firmware source code are also present (e.g., Firmalice), but such methods have poor analytical capabilities.

Simulator pays more attention to the execution of the firmware in the embedded device. It has no excessive restrictions on the input, and has relatively better analysis and expansion capabilities. It is suitable for firmware analysis without actual device or device without a debug interface. The technical route is clear. We design a framework for vulnerability mining with a simulator (see Fig. 1).

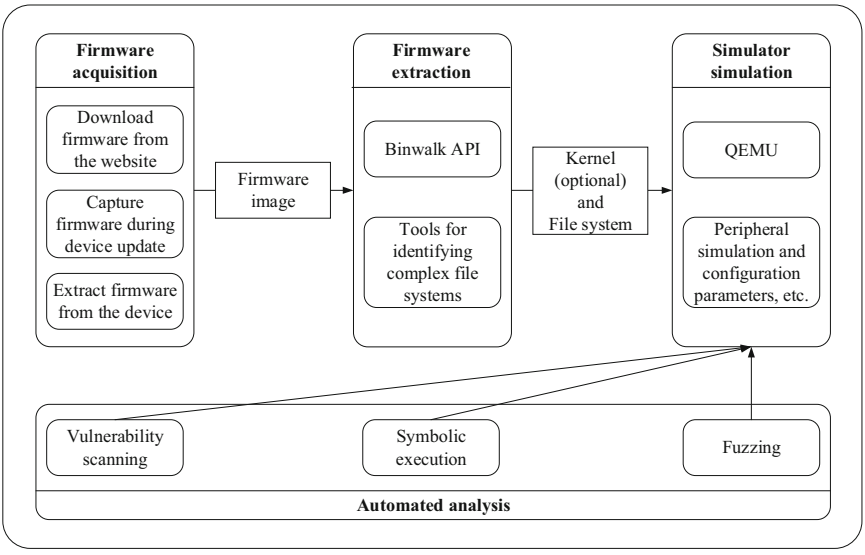

Fig. 1. The framework for vulnerability mining with a simulator

4.1 Firmware Acquisition

In the firmware acquisition phase, our main task is to accurately obtain the firmware to be analyzed and provide sufficient firmware information resources for the analysis. The technology of this process is relatively fixed.

The first method is the most ideal and the most common method. For some larger vendors, the firmware is usually posted on the official website. At the same time, the firmware released on the official website usually has a lot of firmware information. These firmware information plays a role in the later analysis. Some vendors' official websites do not provide firmware, but there are dedicated FTP download sites.

The second method is complicated. It requires us to find the firmware from the official website update package when the firmware is updated. We can capture it during the firmware transfer. This requires us to have some understanding of the firmware and the corresponding firmware update process.

The worst case is that we have to extract the firmware from the embedded device. At this time we have to understand part of the structure of the device, find the interface to

extract the firmware, and use the professional tools and methods to extract the firmware. Among them edge computing is a new security issue. Z. Tian et al. [28] propose a method named CloudSEC meaning real time lateral movement detection for the edge-cloud environment.

4.2 Firmware Extraction

In the firmware extraction phase, we need to extract the parts we need from the complete, and identify the hardware architecture information that the firmware depends on. Binwalk [29] is the most common tool in the process. We only need to use the API of the binwalk to write analysis scripts according to the content we need. Besides binwalk, firmware-mod-kit [30], FRAK [31], and Binary Analysis Toolkit (BAT) [32] are also optional firmware extraction tools.

The content we want to extract is mainly the file system in the firmware. Normal compressed files and file systems can be recognized and extracted by binwalk, but some complex or packaged files modified by the vendor require special unpacking tools. They may need some digital technology. Z. Tian et al. [33] propose a secure digital evidence framework using blockchain (Block-DEF).

Another part worthy of our attention is the kernel in the firmware, which is the part that must be used during the simulation. However, it is optional during the firmware extraction process. This is because the kernel in the firmware usually only performs some of the device-related functions. So we can use a pre-prepared kernel.

4.3 Simulator Simulation

The main work of the simulator simulation phase is to apply the extracted parts from the firmware to the simulator and finally run the firmware. Traditional device emulation is roughly divided into three levels: complete hardware emulation, hardware over-approximate, and firmware adaptation.

We can see that the performance of the simulator is the key to the simulator simulation phase. What needs to be considered is how to simulate more accurately. Using the improved QEMU is a broad idea. QEMU is a machine simulator and virtualization platform. It has the ability to simulate a complete system and user-mode.

Although the performance of QEMU is good enough, its shortcomings are obvious. First of all, the architecture it supports is not enough, which makes it difficult to imitate dedicated complex devices. Secondly, in addition to the kernel and file system, QEMU needs to input the most basic parameters for system configuration. These parameters are not recognized by QEMU. In addition, QEMU is also helpless with the simulation of peripheral devices and memory mapping.

4.4 Automated Analysis

After the firmware has been run on the simulated device, our task is to analyze the virtual device using dynamic analysis. This process is usually targeted at a specific category or categories of vulnerabilities. It is limited by the analytical capabilities of the analysis tool. Here are some common firmware vulnerability mining methods.

At its simplest, you can scan the device for vulnerabilities. This method is based on a vulnerability database. Nmap [34] is an analysis tool that provides information about exploits that may exist on a device. However, this type of method cannot discover new vulnerabilities.

Symbolic execution (as described in Sect. 3.2) is a very efficient method of analysis that can reduce the difficulty of analysis and theoretically reveal all vulnerabilities. The commonly used symbolic execution engines are angr and KLEE, both of which are very powerful enough to enable in-depth vulnerability mining. An analytical framework based on the symbolic execution engine is also a good choice.

Fuzzing (as described in Sect. 3.3) is considered to be the most effective method of vulnerability analysis. However, in the current firmware vulnerability detection tools, there are few tools that use fuzzing. This is mainly limited by the difficulty of fuzzing and device interaction. AFL [35], LibFuzzer [36], Honggfuzz [37], etc. are simple and powerful fuzzing tools. These tools also provide more options for exploiting embedded devices.

In addition, analytical methods combining symbolic execution with fuzzing are also evolving, and Driller [38] is a typical example of this approach. It adds the dynamic symbolic execution engine angr to the AFL. Simulation of network devices may involve network routing decisions. Z. Tian et al. [39] present a prefix level route decision prediction model.

5 Future Work

The existing firmware vulnerability mining methods have formed a complete technical route. However, there are still many shortcomings in these methods. We can continue to find more versatile methods for the key issues in each step along this technical route. And we can even extend the new technology route. We believe that the future direction of firmware vulnerability mining should aim to achieve a large-scale automated firmware vulnerability mining tool with in-depth analysis capabilities. This paper proposes the following new ideas and suggestions.

5.1 Introducing Machine Learning Module

Since the firmware does not have a standard format (as described in Sect. 2.1), the firmware extraction phase relies too heavily on unpacking tools like binwalk. Existing methods are discussed in a single form in addition to binwalk, while binwalk often has false positives when dealing with complex formats, and scripts written using the API of the binwalk are less versatile. If there are multiple file systems in the firmware, then binwalk alone can't complete our needs. Some firmware may require trusted communication services. Vcash [40] is a reputation framework for identifying denial of traffic service.

We can consider adding a separate machine learning module. The module first needs to acquire a large number of different categories of firmware. After that, it is the work that machine learning needs to accomplish. We formatted a large number of different

categories of firmware to select the main features of each part of the firmware, and vectorized these features to finally train the optimal firmware model. With this firmware model, we can identify the firmware and split the firmware more accurately and conveniently during the firmware extraction phase. It is no longer limited by the type and packaging of firmware.

5.2 Improve In-depth Analysis

Existing methods can take many approaches during the automated analysis phase (as described in Sect. 4.4). However, they generally have problems with insufficient analytical capabilities. Vulnerability scanning can only verify existing vulnerabilities; symbolic execution is generally detected for certain types of vulnerabilities; the use of fuzzing is almost always on the surface. This limits the ability of large-scale vulnerability mining to a certain extent.

We consider improving the software's fuzzing tools to adapt it to the firmware. The focus is on the interaction of the fuzzing process with the emulation device and how to run the scripts we need on the firmware of the different architectures. At the same time, symbolic execution needs to overcome the limitations of single vulnerability detection, which requires further study of the symbolic execution engine. This will greatly improve the analysis capabilities of firmware vulnerability mining and achieve large-scale in-depth analysis.

5.3 Integrate Existing Tools

A number of tools have been developed for different phases of analysis of different architectures that overlap in functionality but have their own analytical capabilities. We consider ways to integrate multiple tools. The work of Avatar2 is similar to ours, but the shortcomings of Avatar2 are obvious (as described in Sect. 3.4).

Our idea is not limited to the integration of tools at a certain stage, but the integration of stage tools with the overall tools. For example, by integrating Firmayne and Angr, firmware vulnerability exploitation of general network devices with good performance can be achieved; Costin's system can greatly improve the analysis capabilities of Firmalice as input from Firmalice. It should be noted that in the process of integrating tools, the synchronization of the running state of the devices and the memory data between different frameworks is crucial. Integrating existing frameworks is designed to enable large-scale, general-purpose analysis.

6 Conclusion

In this paper, we summarize the important firmware challenges. We divide the existing firmware vulnerability mining work into four categories: static analysis, symbolic execution, fuzzing, and comprehensive analysis. The contributions and deficiencies of the typical work in these four categories are analyzed in detail and compared. Then we divide them into No-simulator and Simulator technically. We notice the lack of No-simulator and detail analysis of the technical route of the Simulator. We divide this technical route

into four stages: firmware acquisition, firmware extraction, simulator simulation, and automated analysis. At the same time, the techniques commonly used and the problems that should be paid attention to in each stage are summarized. Finally, we propose the future direction of the firmware vulnerability mining work for the above analysis. This includes three suggestions: introducing machine learning modules, improving in-depth analysis capabilities, and integrating existing tools. Our work is aimed at implementing a large-scale automated firmware vulnerability mining tool with in-depth analysis capabilities.

Acknowledgments. This work is funded by the National Key Research and Development Plan (Grant No. 2018YFB0803504), the National Natural Science Foundation of China (No. 61702223, 61702220, 61871140, 61602210, 61877029, U1636215), the Science and Technology Planning Project of Guangdong (2017A040405029), the Science and Technology Planning Project of Guangzhou (201902010041), the Fundamental Research Funds for the Central Universities (21617408, 21619404).

References

1. Opler, A.: Fourth generation software. Datamation **13**(1), 22–24 (1967)
2. IEEE Standards Coordinating Committee.: IEEE standard glossary of software engineering terminology (IEEE Std 610.12–1990). Los Alamitos. CA: IEEE Computer Society, 169 (1990)
3. Zhang, P.: Research on embedded operating system recognition technology for firmware. Inform. Eng. Univ (2012)
4. Zaddach, J., Costin, A.: Embedded devices security and firmware reverse engineering. Black-Hat USA (2013)
5. Tan, Q., Gao, Y., Shi, J., Wang, X., Fang, B., Tian, Z.: Toward a comprehensive insight to the eclipse attacks of tor hidden services. IEEE Internet Things J. **6**(2), 1584–1593 (2019)
6. Tian, Z., Gao, X., Su, S., Qiu, J., Du, X., Guizani, M.: Evaluating reputation management schemes of internet of vehicles based on evolutionary game theory. IEEE Trans. Veh. Technol. IEEE (2019)
7. Hou, J., Li, T., Chang, C.: Research for vulnerability detection of embedded system firmware. Procedia Comput. Sci. **107**, 814–818 (2017)
8. Xie, W., Jiang, Y., Tang, Y., et al.: Vulnerability detection in IoT firmware: a survey. In: 2017 IEEE 23rd International Conference on Parallel and Distributed Systems (ICPADS), pp. 769–772. IEEE (2017)
9. Costin, A., Zaddach, J., Francillon, A., et al.: A large-scale analysis of the security of embedded firmwares. In: 23rd {USENIX} Security Symposium ({USENIX} Security 14), pp. 95–110 (2014)
10. Davidson, D., Moench, B., Ristenpart, T., et al.: {FIE} on firmware: finding vulnerabilities in embedded systems using symbolic execution. In: 22nd {USENIX} Security Symposium ({USENIX} Security 13), pp. 463–478 (2013)
11. Michel, S., Triantafillou, P., Weikum, G.: Klee: a framework for distributed top-k query algorithms. In: Proceedings of 31st International Conference on Very Large Data Bases, pp. 637–648. VLDB Endowment (2005)
12. Shoshitaishvili, Y., Wang, R., Hauser, C., et al.: Firmalice-automatic detection of authentication bypass vulnerabilities in binary firmware. In: NDSS (2015)

13. Shoshitaishvili, Y., Wang, R., Salls, C., et al.: Sok: (state of) the art of war: offensive techniques in binary analysis. In: 2016 IEEE Symposium on Security and Privacy (SP), pp. 138–157. IEEE (2016)
14. Zaddach, J., Bruno, L., Francillon, A., et al.: Avatar: a framework to support dynamic security analysis of embedded systems' firmwares. In: NDSS, pp. 1–16 (2014)
15. Chipounov, V., Kuznetsov, V., Candea, G.: S2E: a platform for in-vivo multi-path analysis of software systems. In: ACM SIGARCH Computer Architecture News, vol. 39, no. 1, pp. 265–278. ACM (2011)
16. Chen, D.D., Woo, M., Brumley, D., et al.: Towards automated dynamic analysis for Linux-based embedded firmware. In: NDSS, pp. 1–16 (2016)
17. Bellard, F.: QEMU, a fast and portable dynamic translator. In: USENIX Annual Technical Conference, FREENIX Track, vol. 41, p. 46 (2005)
18. Dai, Z.: Research on the localization of firmware vulnerability based on stain tracking. J. Shandong Univ. (Nat. Sci.) **51**, 41–46 (2016)
19. Muench, M., Nisi, D., Francillon, A., et al.: Avatar 2: a multi-target orchestration platform. In: Workshop on Binary Analysis Research (colocated with NDSS Symposium) (February 2018), BAR vol. 18 (2018)
20. Högl, H., Rath, D.: Open on-chip debugger–openocd. Fakultat fur Informatik, Technical report (2006)
21. Dolan-Gavitt, B., Hodosh, J., Hulin, P., et al.: Repeatable reverse engineering with PANDA. In: Proceedings of the 5th Program Protection and Reverse Engineering Workshop, p. 4. ACM (2015)
22. Bojinov, H., Bursztein, E., Lovett, E., et al.: Embedded management interfaces: emerging massive insecurity. BlackHat USA **1**(8), 14 (2009)
23. Li, H., Tong, D., Huang, K., et al.: FEMU: a firmware-based emulation framework for SoC verification. In: Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis, pp. 257–266. ACM (2010)
24. Hu, C., Xue, Y., Zhao, L., et al.: Backdoor detection in embedded system firmware without file system. J. Commun. **34**(8), 140–145 (2013)
25. Wenli, S.: Study on the vulnerability analysis method for industrial embedded devices. Autom. Instrum. **36**(10), 63–67 (2015)
26. Lee, D.: Firmware vulnerability detection in embedded device based on homology analysis. Comput. Eng. (2016)
27. Feng, Q., Zhou, R., Xu, C., et al.: Scalable graph-based bug search for firmware images. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 480–491. ACM (2016)
28. Tian, Z., et al.: Real time lateral movement detection based on evidence reasoning network for edge computing environment. IEEE Trans. Industr. Inform. **15**(7), 4285–4294 (2019)
29. Heffner, C.: Binwalk: firmware analysis tool (2010). https://code.google.com/p/binwalk/. Accessed 03 Mar 2013
30. Heffner, C., Collake, J.: Firmware mod kit-modify firmware images without recompiling (2015)
31. Cui, A.: FRAK: firmware reverse analysis konsole. In: Proceedings of Black Hat USA, pp. 1–33 (2012)
32. Hemel, A., Coughlan, S.: BAT–Binary Analysis Toolkit. Accessed Jan 2017
33. Tian, Z., Li, M., Qiu, M., Sun, Y., Su, S.: Block-DEF: a secure digital evidence system using blockchain. Inform. Sci. **491**, 151–165 (2019)
34. Lyon, G.: Nmap–free security scanner for network exploration & security audits (2009)
35. Zalewski, M.: American fuzzy lop (2017). http://lcamtuf.coredump.cx/afl
36. Serebryany, K.: LibFuzzer a library for coverage-guided fuzz testing. LLVM project (2015)

37. Swiecki, R.: Honggfuzz: a general-purpose, easy-to-use fuzzer with interesting analysis options. https://github.com/google/honggfuzz. Accessed 21 June 2017
38. Stephens, N., Grosen, J., Salls, C., et al.: Driller: augmenting fuzzing through selective symbolic execution. NDSS **16**(2016), 1–16 (2016)
39. Tian, Z., Su, S., Shi, W., Du, X., Guizani, M., Yu, X.: A data-driven model for future internet route decision modeling. Future Gener. Comput. Syst. **95**, 212–220 (2019)
40. Tian, Z., Su, S., Yu, X., et al.: Vcash: a novel reputation framework for identifying denial of traffic service in internet of connected vehicles. IEEE Internet Things J. **7**(5), 3901–3909 (2019)