Journal of Cyber Security
DOI: 10.32604/jcs.2022.026816
Article

Tech Science Press

Research on Known Vulnerability Detection Method Based on Firmware Analysis

Wenjing Wang[1], Tengteng Zhao[1], Xiaolong Li[1,*], Lei Huang[1], Wei Zhang[1] and Hui Guo[2]

[1]Beijing Institute of Control and Electronics Technology, Beijing, 100038, China
[2]State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, 100876, China
*Corresponding Author: Xiaolong Li. Email: lxl-777333@163.com
Received: 03 March 2022; Accepted: 07 April 2022

Abstract: At present, the network security situation is becoming more and more serious. Malicious network attacks such as computer viruses, Trojans and hacker attacks are becoming more and more rampant. National and group network attacks such as network information war and network terrorism have a serious damage to the production and life of the whole society. At the same time, with the rapid development of Internet of Things and the arrival of 5G era, IoT devices as an important part of industrial Internet system, have become an important target of infiltration attacks by hostile forces. This paper describes the challenges facing firmware vulnerability detection at this stage, and introduces four automatic detection and utilization technologies in detail: based on patch comparison, based on control flow, based on data flow and ROP attack against buffer vulnerabilities. On the basis of clarifying its core idea, main steps and experimental results, the limitations of its method are proposed. Finally, combined with four automatic detection methods, this paper summarizes the known vulnerability detection steps based on firmware analysis, and looks forward to the follow-up work.

Keywords: IoT devices; vulnerability mining; automatic detection; static analysis

1 Introduction

In recent years, with the deepening of the integration of informatization and industrialization, industrial control systems have moved from stand-alone to interconnection, from closed to open, and from automation to intelligence. With the significant improvement of productivity, industrial control systems are facing an increasingly severe threats to information security. According to Gartner's report [1], the number of IoT devices will exceed 20 billion in 2020. In 2010, Iran's nuclear facilities were attacked by Stuxnet virus, which delayed the progress of Iran's nuclear program for 18 months to two years, which caused a sensation around the world. In 2012, the important information systems of the oil industry in Iran, Lebanon and other Middle Eastern countries were greatly impacted by

the “flame” virus, forcing Iran to temporarily cut off the Internet connection of the oil sector and related facilities, and hitting the oil exports. Through the statistical analysis of the Internet of Things vulnerability data publicly disclosed by CNVD (As shown in Figs. 1 and 2), it is found that there are about 10,600 Internet of Things vulnerabilities so far, including 4,512 high-risk vulnerabilities, accounting for 42.31%, and 1639 intermediate-risk vulnerabilities, accounting for 42.31%, which is much higher than the proportion of high and intermediate risk vulnerabilities in the traditional industries. Therefore, Internet of Things security has become the focus of governments and enterprises around the world.

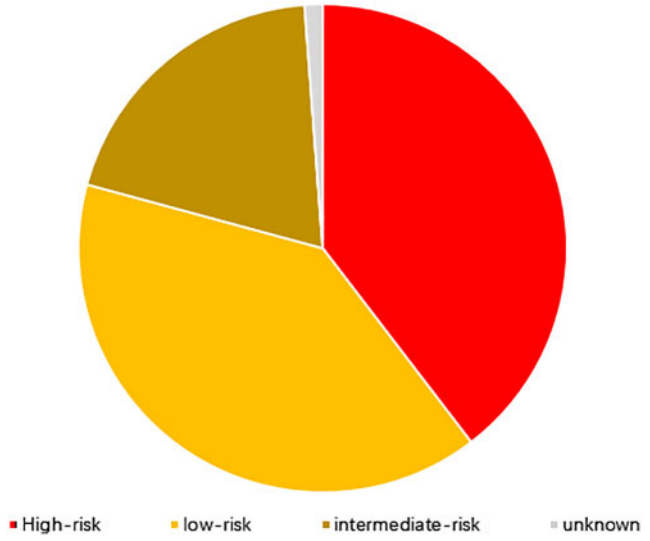

Figure 1: Distribution map of IoT vulnerability hazard levels

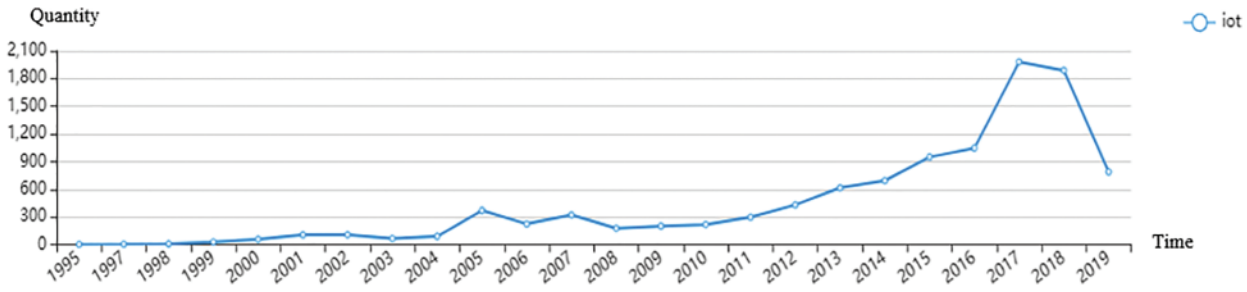

Figure 2: Trend of the number of vulnerabilities in the Internet of Things

According to the statistics of Alibaba mobile security team in 2015, 90% of IoT devices have weak key and buffer overflow vulnerabilities [2,3]. In recent years, a large number of vulnerabilities of IoT devices have been disclosed. For example, at the Black Hat Conference in 2013, Heffners [4] showed the overflow class, password hard coding and command injection vulnerabilities of various webcams, involving DLink, TPLink, Linksys and Trendnet device manufacturers. Attackers can use these vulnerabilities to conduct unauthorized login and hijack the real-time picture of the camera. Since then, various types of IoT devices (from smart homes, such as smart bulbs, thermostats and routers, to electric vehicles and airplanes with wireless networks and entertainment systems) have been disclosed at the hacker conference that have serious vulnerabilities. In addition, real security incidents

caused by security vulnerabilities are also emerging one after another. On October 21, 2016, hackers used a large number of IoT devices infected by Mirai virus to launch DDoS attacks against DNS servers managed by Dyn, affecting the east coast, west coast of the United States and some parts of Europe, and resulting in the inaccessibility of many well-known websites such as Twitter, Github, Amazon, Paypal, BBC, Wall Street Journal and so on. Generally speaking, the attack against the security vulnerabilities of IoT devices will not only cause the leakage of personal privacy, but also cause the loss of personal and property, and even threaten the security of the whole cyberspace. Therefore, vulnerability detection for IoT devices is imminent.

2 Challenges and Opportunities Facing Firmware Vulnerability Detection

2.1 IoT Device Types are Intricate and Have Different Standards

There are many IoT manufacturers, but there is no unified specification. Each manufacturer adopts different file formats, CPU architecture and encryption methods, etc., which makes firmware analysis face great challenges. Therefore, it is difficult to form a perfect standard vulnerability analysis system. On the one hand, the CPU architecture of IoT devices is different from that of general platforms, resulting in differences in program instruction sets. The instruction architecture of general-purpose software is usually X86 or X86_64 [5–7]. The IoT programs usually use embedded architectures such as ARM, MIPS and PowerPC. Therefore, the static analysis scheme directly based on general CPU instruction assembly is no longer applicable. On the other hand, the peripheral I/O hardware of IoT devices is diversified, which increases the difficulty of adapting dynamic analysis technology [8].

2.2 The Firmware is Difficult to Obtain and Decompress

The general dynamic binary analysis technology needs to implement monitoring and analysis on the periphery of the running program. Due to the limitation of the storage resources (storage) of IoT devices, the relevant analysis modules cannot be deployed, resulting in the inapplicability of dynamic analysis technology. At the same time, the computing capability of the hardware CPU is limited, which will reduce the performance of dynamic analysis. In practice, the IoT devices usually do not provide the source code of firmware, and even most of them do not publicly provide the firmware of devices. Therefore, obtaining the firmware is the most difficult work in the research of firmware vulnerability detection. After obtaining the firmware, it is usually impossible to directly analyze the program, because what is obtained is a series of compressed files of program files and data files, which need to be decompressed. However, different manufacturers use different compression methods, and even some manufacturers use obfuscated encryption, which brings great difficulty to firmware analysis.

2.3 Low Success Rate of Firmware Dynamic Simulation

At present, the vulnerability detection based on firmware web interface is a more effective method for firmware vulnerability detection, but this detection method requires real devices or dynamic simulation of firmware. Because some devices are expensive and firmware simulation is more universal, most of the existing research work adopts dynamic simulation of firmware based on QEMU [9,10]. According to relevant material statistics, only 13%–20% of firmware can be fully simulated (that is, it supports all functions of simulated firmware). In addition, the dynamic execution of firmware also causes great difficulties in the detection of firmware vulnerabilities.

2.4 Opportunities for Vulnerability Mining in IoT Devices

The characteristics of IoT devices not only bring challenges to vulnerability mining, but also bring new opportunities.

(1) The Richness of System Interaction: Although it is for vulnerability mining of IoT devices, the IoT devices usually interact with terminal, cloud and other systems, so the device itself has more attack surfaces. For the dynamic binary analysis scheme, we can make full use of the information of the external interactive system to test and analyze the new attack surface.

(2) Massive Reuse of Component Code: During the development process of IoT device programs, in order to save development costs, a large number of open source third-party libraries are used, resulting in a large number of vulnerabilities of third-party components in IoT devices. The previous static analysis technology based on binary comparison is mainly to discover security vulnerabilities through different levels of information (control flow, program block, instruction level), but now we can exploit homologous vulnerabilities through the similarity of different levels of information.

(3) Convergence of Vulnerability Types: General software vulnerability types include memory corruption classes (stack overflow, heap overflow, null pointer application, secondary release, etc.), input verification classes (command injection, etc.), configuration error classes, etc. The location of the vulnerability can be in the kernel, driver and user mode service program. For the firmware of IoT devices containing operating systems, these types of vulnerabilities also exist [11,12]. Therefore, whether static binary or dynamic binary analysis technology, the general vulnerability detection rules are still applicable to the firmware and programs of IoT devices.

3 Automatic Detection and Utilization Technology of Firmware Vulnerabilities

Fuzzy testing [13] is a very effective vulnerability mining method for software and systems, and it is also the most widely used technology for dynamic analysis of IoT devices. By sending random input to the tested object, and by observing its behavior (usually program crash), potential vulnerabilities can be discovered. Software vulnerability is a hot issue at present. Although fuzzy testing technology helps us solve the problem of automatic discovery of program vulnerabilities, parallel fuzzy testing platform can efficiently find a large number of program errors. But both defenders and attackers are more concerned about whether these program vulnerabilities or errors may be exploited. How to quickly analyze and evaluate the exploitability of vulnerabilities is one of the key problems in vulnerability discovery and analysis.

The traditional software vulnerability exploitation is mainly constructed manually. This process requires not only comprehensive system underlying knowledge (including file format, assembly code, internal mechanism of operating system and processor architecture, etc.), but also in-depth and detailed analysis of vulnerability mechanism, so as to construct successful utilization. With software functions becoming more and more complex and vulnerabilities becoming more and more diversified, the traditional utilization methods have been difficult to meet the challenges mentioned above. At present, with the continuous development of program analysis technology, especially after the successful application of stain analysis, symbol execution and other technologies in many fields such as software dynamic analysis and software vulnerability detection, researchers began to try to use these technologies to carry out efficient automatic construction of software vulnerability utilization [14]. Tab. 1 shows the comparison between existing works. Next, this paper will introduce each work in detail.

Table 1: Comparison of automated verification methods for vulnerabilities

Plan	Patch based	Control flow oriented			Data flow oriented	Rop
Method	APEG	AEG	MAYHEM	PolyAEG	FLOWSTITCH	Q
Release time	2008	2011	2012	2013	2015	2011
Core idea	The patch program adds filter conditions to trigger the crash of the original program	Use the program verification technology to find the input that can satisfy the program to enter the unsafe state and can be utilized	Use an index-based memory model to optimize the processing of symbolic memory	All control flow hijacking points are found through dynamic taint analysis to complete the diversified structure of vulnerability samples	Use known memory errors to directly or indirectly tamper with variables at key positions in the original data stream of the program to complete the automated structure of the utilization	Collect the Gadget (driver) in the target program and automatically build the ROP through the Gadget-oriented programming language
Limitations	Unable to handle the case where no filtering judgment is added in the patch	Need to rely on source code for program error search	Only part of the system or library functions can be modeled, and large programs cannot be processed efficiently	There are certain limitations in the face of data execution protection mechanisms	Does not consider satisfying Turing completeness	Did not consider automatically constructing ROP without ret instruction

3.1 Automatic Vulnerability Detection and Utilization Technology Based on Patch Comparison

Early automated patching techniques were used to prevent the spread of worm. With the development of technology, automatic patching technology has slowly penetrated into all aspects of computer software security. Automatic patching technology is divided into two types: based on runtime state and based on detection patch.

The automatic vulnerability repair technology based on runtime state bypasses the vulnerabilities without interrupting the program by comparing the normal execution process or taking some actions. For example, ClearView fixes binary errors by automatically monitoring the normal execution of X86 system registers and memory. When an error occurs, ClearView compares it with the log during normal operation to correct the error. In addition, ClearView can solve the problem of memory write out of bounds and control flow vulnerabilities. The detection based patch technology can use genetic algorithm to generate patches and patch vulnerabilities through constraint solving. For example, GenProg uses genetic algorithms to patch vulnerabilities without requiring software specifications, program comments and other special coding. In order to repair the vulnerability with minimal changes, it uses structural difference algorithm and Delta debugging technology to further reduce the difference between the patched program and the original program. SemFix combines the methods of symbolic execution, constraint solving and program combination. It uses a given test case to constrain the program that needs to be patched into a solution formula, and generates patches through semantic analysis and dynamic symbolic execution. Automated patching technology has made many research achievements in solving computer security problems, but it still can not repair all types of vulnerabilities, and there is no breakthrough for 0 day vulnerabilities. How to better understand the high-level semantics in the program is a difficult problem in the automatic patch technology.

At the IEEES&P conference in 2008, Brumley and others first proposed the automatic vulnerability generation method APEG based on binary patch comparison. The core idea is based on the following assumptions, that is, thhe patch program adds filter conditions that trigger the original program to crash. Therefore, as long as you can find the location where the filter condition is added in the patch program and construct the "violating" input that does not meet the filter condition, it can be considered as a usable input candidate for the original program. According to the specific introduction, the work is mainly divided into three steps: firstly, use binary difference comparison tools (such as BinDiff and EBDS, etc.) to find the location of the patch, that is, the detection point of the patch; Secondly, find out the input data that does not meet the patch detection point as the utilization candidate of the original program; Finally, the monitoring methods such as stain propagation are used to filter all the effective utilization that can cause overflow or control flow hijacking to occur in the original program. According to the experimental results of several patches released by Microsoft, the method has strong reliability and practicability. APEG is the first attempt to automate the construction of vulnerability exploitation. Although the core idea is relatively simple, it has been widely recognized by other researchers because of its strong operability. However, the limitations of APEG are mainly reflected in two aspects: firstly, this method cannot deal with the case that no filter judgment is added to the patch, for example, the patch that increases the buffer length in order to repair the buffer overflow; Secondly, from the actual utilization effect, the constructed utilization type mainly belongs to denial of service, that is, it can only cause the collapse of the original program, but cannot cause direct control flow hijacking.

3.2 Automatic Vulnerability Detection and Utilization Technology Based on Control Flow

3.2.1 Automatic Vulnerability Detection and Utilization Technology Based on Source Code

Source code vulnerability detection aims at the software design and development stage, by extracting the source code model and vulnerability rules, and detecting vulnerabilities in the source code based on static program analysis technology. It has the advantages of high code coverage and low false negatives, but it is highly dependent on known vulnerabilities and high false positives. Source code vulnerability detection methods mainly include vulnerability detection based on intermediate

representation and vulnerability detection based on logical reasoning [15–17]. The vulnerability detection method based on intermediate representation first converts the source code into an intermediate representation that are conducive to vulnerability detection, and then analyzes the intermediate representation to check whether it matches a predefined vulnerability rule, so as to determine whether the source program contains vulnerabilities related to the corresponding vulnerability rule.

The vulnerability detection method based on logical reasoning describes the source code formally, and then uses mathematical reasoning, proof and other methods to verify some properties of the formal description, so as to judge whether the program contains a certain type of vulnerabilities. The vulnerability detection method based on logical reasoning is based on mathematical reasoning, so the analysis is strict and the result is reliable. However, for large-scale programs, it is very difficult to formalize the code. The vulnerability detection method based on intermediate representation does not have the above limitations, and is suitable for analyzing large-scale programs, so it has been more widely used.

At the NDSS meeting in 2011, Avgerinos and others first proposed an effective automatic vulnerability detection and utilization method AEG [18,19]. The core idea of this method is to use the program verification technology to find the input that can satisfy the program to enter the unsafe state and can be used. The unsafe state includes the memory out-of-bounds writing, malicious formatted string, etc., which can be used mainly refers to the EIP of the program is arbitrarily manipulated. The specific process is as follows: firstly, in the preprocessing stage, GNUC compiler is used to build the binary program [20] and LLVM is used to generate the required byte code information; Secondly, in the process of actual analysis, AEG firstly finds out the error location through source code analysis and symbol execution, and generates the corresponding input through path constraints. After that, AEG uses the dynamic analysis method to extract all kinds of information when the program runs, such as the address of the vulnerable buffer on the stack, the return address of the vulnerable function and other environmental data before the vulnerability is triggered. Then, the exploitable samples are finally constructed by integrating vulnerability utilization constraints and dynamic runtime environment information. Through the automatic utilization experiments of 14 groups of real program vulnerabilities, the reliability and effectiveness of this method are proved. AEG integrates the optimized symbol execution and dynamic instruction insertion technology, and realizes the whole process from software vulnerability automatic mining to software vulnerability automatic utilization. In addition, the generated utilization samples directly have the ability of control flow hijacking, which is the first real automatic construction scheme for control flow vulnerability utilization. The limitations of the scheme are as follows: firstly, the scheme needs to rely on the source code for program error search; Secondly, the constructed utilization samples are mainly for stack overflow or string formatting vulnerabilities, and the utilization samples are limited by factors such as compiler and dynamic running environment.

3.2.2 Automatic Vulnerability Detection and Utilization Technology Based on Binary

Binary vulnerability is that the software performs unexpected functions due to the executable files (PE, ELF files, etc.) are not well considered during coding [21]. Because binary vulnerabilities mostly involve the system level, the degree of harm is relatively high. For example, the classic office stack overflow vulnerability (CVE-2012-0158), (CVE-2017-11882) and the patch bypass vulnerability (CVE-2018-0802) of (CVE-2017-11882) are all extremely dangerous 0 and 1 day vulnerabilities. Common binary vulnerabilities include Stack-Overflow, Heap-Overflow, Use-After-Free, Double-Free, and Out-of-bounds.

In order to get rid of the dependence on source code and ensure the universality of system application scenarios, Cha and others proposed mayhem, an automatic generation method for vulnerability utilization based on binary program at the IEEES&P conference in 2012. This method makes comprehensive utilizes the speed advantages of online symbol execution and the low memory consumption characteristics of offline symbol execution, and constructs a memory model based on index, so as to realize a more practical vulnerability mining and utilization automatic generation method. The specific process is as follows: firstly, construct two parallel symbolic execution subsystems, the specific execution subsystem and the symbolic execution subsystem. Secondly, for the specific execution subsystem, the stain propagation technology is introduced to find all JMP instructions or call instructions that can be controlled by the user in the process of program execution, and give them to the symbol execution subsystem as bug candidates. Then, the symbol execution system converts all received tainted instructions into intermediate instructions, and constructs execution path constraints and available constraints. Finally, the symbolic execution system uses the constraint solver to find the utilization samples that meet the path reachable conditions and vulnerability exploitable conditions. In the actual process of symbol execution, in order to ensure the efficiency, the Mayhem system uses an index-based memory model to optimize the loading of symbolic memory, so as to make it a highly usable vulnerability automatic utilization scheme. At present, Mayhem's limitations mainly focus on the following three aspects: first, the system can only model part of the system or library functions, so it cannot deal with large programs efficiently; Secondly, the system cannot deal with multi-threaded interaction issues, such as message passing and shared memory issues; Finally, due to the use of stain propagation method, there are also typical problems such as missed transmission and false transmission.

3.2.3 Automatic Vulnerability Detection and Utilization Technology for Diversified Vulnerabilities

In order to improve China's ability to control software and system vulnerability resources, it is urgent to tackle key problems such as weak intelligence of vulnerability mining and analysis, low accuracy of large flow monitoring, difficult hazard assessment and verification, and lack of scale coordination, and study software and system vulnerability intelligent mining methods and key technologies, software and system vulnerability analysis and availability judgment technologies, the vulnerability analysis and detection technology based on network traffic, vulnerability hazard assessment and verification technology, and vulnerability large-scale collaborative mining and analysis technology are used for automatic detection by using diversified vulnerability characteristics.

Since high-quality and diversified vulnerability utilization samples are of great significance to vulnerability hazard assessment, Wang and others proposed a set of automatic generation method (PolyAEG) of diversity utilization samples for control flow hijacking vulnerabilities at securecomm conference in 2013 [22]. The core idea of this method is to find out all control flow hijacking points of the program through dynamic stain analysis, and to complete the diversity construction of vulnerability utilization samples by constructing different control flow transfer modes. The specific process is as follows: firstly, the program dynamic monitoring is realized and the relevant information of program execution is extracted by expanding the hardware virtualization platform QEMU; Secondly, based on the dynamic acquisition of information, the instruction level stain propagation flow graph iTPG and the global stain state record GTSR are constructed. And on this basis, all possible control flow hijacking points, available springboard instructions and the stain memory area that can store attack codes in the program are obtained. Finally, by constructing different jump instruction chains and attack codes in different tainted memory areas, and solving the path constraints, a diverse set of utilization samples is generated. According to the experimental results of 8 actual vulnerability

samples, the scheme generates up to 4724 utilization samples for a single control flow hijacking vulnerability. PolyAEG implements a complete set of automatic vulnerability utilization diversity structure for control flow hijacking vulnerabilities, which provides effective support for vulnerability hazard assessment. However, the limitations of the scheme are mainly reflected in the following two aspects: one is that he scheme has certain limitations in the face of data execution protection mechanism; The other is that this scheme only considers relying on the existing instructions in the program itself or other class libraries, and does not consider the use of dynamically generated code.

3.2.4 Automatic Vulnerability Detection and Utilization Technology Based on ROP Code

Although there are many innovative systems to ensure software security. But many applications are still vulnerable to hooks and return-oriented programming (ROP) attacks. Although it is impossible to get rid of all vulnerabilities in the application, developers should consider executable space protection during the coding phase. In order to solve the problem of control flow hijacking vulnerability utilization caused by data execution protection and address randomization, Schwartz and others realized a set of ROP code automatic generation method Q [23] for high reliability vulnerability utilization at the USENIX Security conference in 2011. The core idea is to collect the Gadgets in the target program and automatically build the ROP through the Gadget oriented programming language. The specific process is as follows: first, Q is provided with unrandomized fragile programs or other binary libraries, and Q finds the Gadget set with specific functions; Secondly, the programming language QooL provided by Q is used to realize the object code that meets the specific semantic functions, and Q is used to compile the object code into a gadget oriented instruction sequence; Then, the final ROP code is formed by filling the instruction sequence obtained in the previous step with the obtained Gadget set.

Through experiments on 9 real software vulnerabilities, it can be seen that after the data execution protection and address randomization functions are enabled, the stable execution of the exploit code of these vulnerabilities can still be guaranteed through Q. The Q scheme proves that the ROP code can still be effectively and automatically constructed in the system with a small amount of non randomized code, which strengthens the attack effect of control flow hijacking vulnerability exploitation in the real environment. The limitations of the Q scheme itself are mainly reflected in: firstly, the Q scheme does not consider the automatic construction of ROP without "ret" instruction; Secondly, the Q scheme only starts from the practical application effect and does not consider meeting the Turing completeness.

3.3 Automatic Vulnerability Detection and Utilization Technology Based on Data Flow

Variable tracking based on code data flow is the principle of many white box detection tools. It also needs to analyze the code execution process and detect vulnerabilities based on data flow. Instead of directly dividing the source code into strings, it will destroy the semantics of the code and lose the basis of vulnerability detection [24–26].

In the case of large-scale deployment of data execution protection, address randomization, and control flow integrity protection measures, most attackers have shifted from vulnerability exploitation attacks for control flow hijacking to data flow utilization attack. It is against this background that Hu and others first proposed an automatic construction method for data flow utilization, FlowStitch, at the USENIX Security conference in 2015 [27]. The core idea of this method is to use the known memory errors to directly or indirectly tamper with the variables at the key position in the original data flow of the program without changing the program control flow, so as to complete the automatic construction of the utilization. According to the introduction of the article, the specific process is

divided into the following steps: first, take the program containing memory error, the input triggering memory error and special normal input as the three preconditions of the whole automatic utilization system, in which "special normal input" means that before the program error occurs, its execution path must be the same as that triggering memory error; Secondly, the corresponding error execution record and normal execution record are obtained by error input and normal input respectively, and based on this, the influence range of memory error and sensitive data in normal data stream are further extracted respectively; Finally, the sensitive data that may be involved in the impact range of memory errors are determined by comparing the error execution records with the normal execution records. Finally, all sensitive data that may be tampered with are screened and the automatic construction process of data flow oriented utilization is completed. Through the experimental results on 8 real vulnerability samples, it can be seen that the 19 utilization samples automatically constructed by FlowStitch can not only bypass the protection methods such as data execution and fine-grained control flow integrity, but also 10 utilization samples can be successfully executed in the environment of opening address randomization. Flowstich is the first automatic vulnerability exploitation scheme for data flow. Although the utilization samples constructed by FlowStich can not directly run arbitrary malicious code, they still have strong practical value because they can leak the sensitive data on the target host. From the description details of this paper, the limitations of this scheme are mainly reflected in: firstly, the utilization of data flow is based on the premise that there are known memory errors in the program; Secondly, in the process of construction and utilization, it is necessary not only to input the corresponding execution record incorrectly, but also to construct the corresponding normal input and normal execution path.

3.4 Automatic Vulnerability Detection and Utilization Technology for Buffer Vulnerability ROP Attacks

The full name of ROP attack technology is "Return-Oriented Programming", that is, return oriented programming. Its core idea is to achieve its purpose by controlling the program flow and executing the existing executable code of the program. Fig. 3 shows its algorithm flow chart. ROP technology reuses short instruction fragments, which are divided into many sections. Each section ends with a "ret" instruction and becomes the Gadget of ROP. Each Gadget performs only a small part of functions, such as push ebx, ret. The two instructions form a Gadget, and then a ROP chain is formed through several Gadgets. ROP attack technology is to complete its core functions through ROP chain.

3.4.1 Stack Overflow Validation Rules

Fig. 4 shows the schematic diagram of stack overflow verification. According to the different characteristics of function return address and ordinary function return address during stack overflow, malicious samples and normal programs can be distinguished, and the overflow location can be found. The Ret instruction that pops up this overflow position will be listed in the list of suspicious Ret instructions for subsequent ROP attack verification.

3.4.2 ROP Attack Verification Rules

ROP attacks usually achieve the purpose of executing key functions by constructing the values of stacks and register. The key function is to bypass DEP, such as closing DEP or opening a new heap space to execute shellcode to bypass DEP [28,29]. The characteristics of ROPCHAN are as follows: 1. The length of regular Gadget should meet 1–6 instructions. 2. In the actual attacks, the ROP Gadget should be continuous.

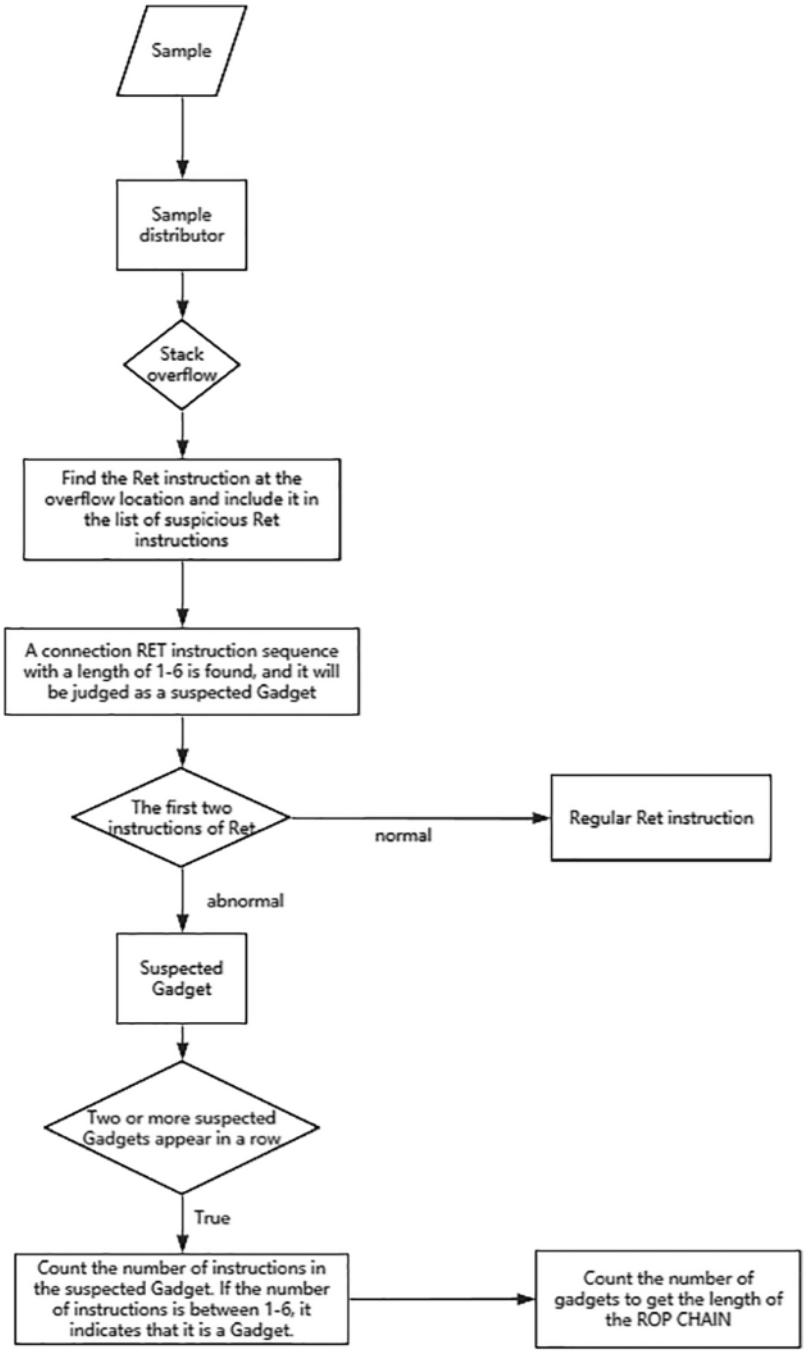

Figure 3: Algorithm flow chart

During the detection process, a continuous RET instruction sequence with a length of 1–6 was found on the basis of stack overflow, and it will be determined as a suspected Gadget. Ret instruction features are divided into: 1. Normal Ret instruction; 2. Ret instruction in Gadget. The conventional Ret instruction is generally generated at the end of the function, indicating that the function is successfully executed and ready to return the address before call. Then, before returning, the system must complete

the operation of recovering the stack frame, restoring the pointer values of EBP and ESP, so that the contents of the function stack can be restored to the stack frame of the upper function. The process of stack frame recovery is shown in Fig. 5.

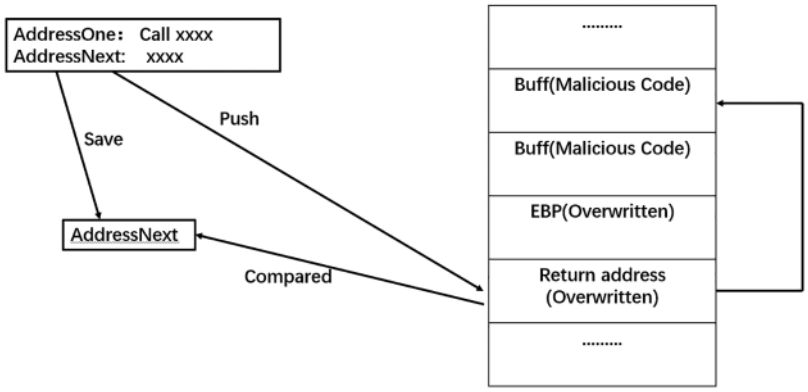

Figure 4: Schematic diagram of stack overflow verification

Step 1: MOV ESP, EBP, that is, ESP points to EBP.

Step 2: POP EBP, put the return address into the EBP before pop-up, that is, the EBP points to the front EBP.

The Ret instruction in the Gadget refers to the instruction before RET in the ROP CHAIN. It is impossible to completely include the above two steps. Because these two instructions restore the stack frame, other ROP CHAIN instructions placed in the stack will be invalidated.

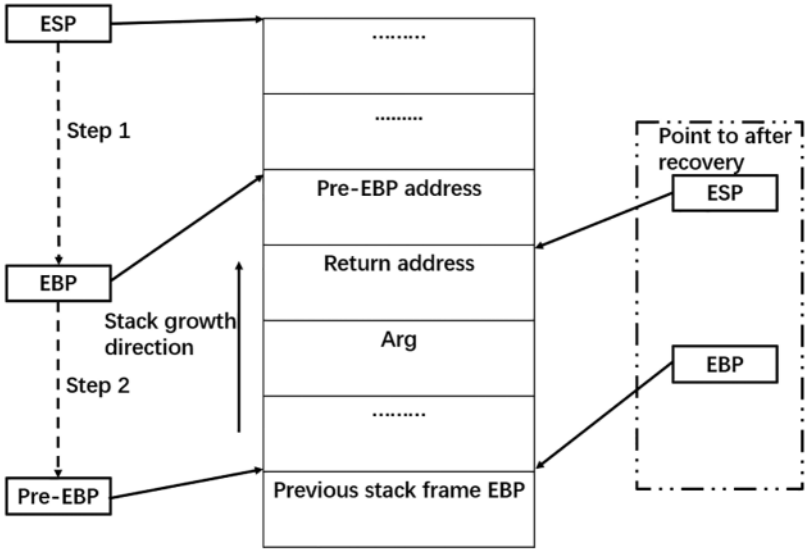

Figure 5: Schematic diagram of stack frame recovery

4 Summary of Known Vulnerability Detection Steps Based on Firmware Analysis

There are a large number of unknown vulnerabilities in Internet of things devices, which brings many potential threats to the devices themselves and cyberspace. Although government departments and security research teams recognize the network security risks brought by the vulnerabilities of IoT devices and the urgency of strengthening the vulnerability detection of IoT devices, there is still a lack of effective technical means for vulnerability detection of IoT devices. Although there are abundant technologies, products and research teams related to vulnerability detection on the market, most of them are for general-purpose systems (Windows, Linux, Mac, Android) and their software.

In terms of IoT device vulnerability detection, due to the huge differences in software and hardware between IoT devices of different manufacturers, the non disclosure of IoT devices source code and documents, it is difficult to build an IoT vulnerability analysis model and establish a unified dynamic simulation environment, and it is difficult to form an efficient, automated and batch IoT device vulnerability detection method. At present, most of the IoT devices vulnerabilities are found through manual analysis by security personnel. At the same time, OWASP has issued firmware security test guidelines, which given some guidance methods for firmware security evaluation. In this chapter, we will refer to OWASP firmware security test guidelines to describe the routine firmware leak detection steps, as shown in Tab. 2.

Table 2: Firmware vulnerability detection steps

1	Information collection	Get details of all relevant technical documentation about the firmware of the target device
2	Get firmware	Use one or more of the recommended methods listed to obtain firmware
3	Firmware feature analysis	Check the characteristics of the target firmware
4	Extract the file system	Get the file system from the target firmware
5	Analysis the file system	Statically analyze vulnerabilities in system configuration files and binaries of extracted files
6	Firmware simulation execution	Simulate firmware files and components
7	Dynamic scanning	Perform dynamic security testing for firmware and application program interfaces

5 Summary and Prospect

The current vulnerability detection technology of IoT devices has made some progress in firmware based Web interface, sensitive information and homology analysis technology. Among them, the known vulnerability detection technology based on firmware Web interface is limited by the execution technology of firmware simulation and the difficulty of starting web service interface. There is still a lot of development space, and a great breakthrough in firmware simulation technology is required. Static analysis technology can effectively solve the analysis of firmware and the analysis of common vulnerabilities in firmware. However, there is still a lack of in-depth thinking and exploration for the efficient analysis of specific vulnerabilities of IoT devices. In addition, there is still a lack of systematic

analysis and research on firmware without operating system and containing specific embedded operating system.

For the homology analysis technology, the current technology has supported the multi-level correlation of large-scale firmware, so as to realize the discovery of homology vulnerabilities. The future development direction should be to effectively extract features and code for specific vulnerability types, so as to achieve accurate and rapid discovery of specific types of homologous vulnerabilities.

Generally speaking, the vulnerability detection technology of IoT devices is still in its infancy. In the future, it is still necessary to start with these three categories of technologies. On the one hand, it proposes general methods and technologies. On the other hand, it will also study corresponding detection technologies for specific types of devices and vulnerabilities.

In order to quickly analyze and determine the exploitability of a large number of software vulnerabilities generated by fuzzy testing technology, researchers have proposed a series of efficient automatic construction schemes for vulnerability utilization, including patch comparison scheme, control flow-oriented scheme and data flow-oriented scheme. The implementation of these schemes can not only help us quickly identify high-risk vulnerabilities from a large number of program vulnerabilities, but also help us reduce the possibility of high-risk vulnerability attacks to a certain extent.

Although the current automatic exploitation of software vulnerabilities has achieved preliminary results, with the increase of software complexity, the deployment of defense methods such as control flow integrity detection and the development and change of software vulnerability types. It has brought challenges to the vulnerability availability evaluation. Therefore, further exploration and research are needed for software vulnerability utilization, and more efficient and reliable automation schemes are proposed.

Funding Statement: The authors received no specific funding for this study.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] Gartner says 8.4 billlion connected "things" will be in use in 2017, up 31 percent from 2016, Gartner. http://www.gartner.com/en/newsroom/. 2019.
[2] https://www.cnblogs.com/alisecurity/p/5261794.html, 2015.
[3] T. F. Tu, X. Y. Liu, L. H. Song and Y. Y. Zhang, "Understanding real-world concurrency bugs in go," in *Proc. of the Twenty-Fourth Int. Conf. on Architectural Support for Programming Languages and Operating Systems*, New York, NY, USA, Association for Computing Machinery, pp. 865–878, 2019.
[4] C. Heffners, "Exploiting network surveillance cameras like a hollywood hacker," 2013.
[5] B. Qin, T. Tu, Z. Liu, T. Yu and L. Song, "Algorithmic profiling for real-world complexity problems," *IEEE Transactions on Software Engineering*, vol. 16, pp. 3067652, 2021.
[6] L. Rao, H. Zhang and T. Tu, "Dynamic outsourced auditing services for cloud storage based on batch-leaves-authenticated merkle hash tree," *IEEE Transactions on Services Computing*, vol. 13, no. 3, pp. 451–463, 2020.
[7] H. Zhang, B. Qin, T. Tu, Z. Guo, F. Gao *et al.,* "An adaptive encryption-as-a-service architecture based on fog computing for real-time substation communications," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 1, pp. 658–668, 2020.

[8] T. F. Tu, L. Rao, H. Zhang and Q. Y. Wen, "Privacy-preserving outsourced auditing scheme for dynamic data storage in cloud," *Security and Communication Networks*, vol. 12, pp. 1–17, 2017.
[9] J. Qin, H. Zhang, J. Guo, S. Wang, Q. Wen *et al.,* "Vulnerability detection on android apps–inspired by case study on vulnerability related with web functions," *IEEE Access*, vol. 8, pp. 106437–106451, 2020.
[10] J. Qin, H. Zhang, S. Wang, Z. Geng and T. Chen, "Active++: An improved android application automatic tester based on active," *IEEE Access*, vol. 7, pp. 31358–31363, 2019.
[11] S. Wang, S. Qin, J. Qin, H. Zhang, T. Tu *et al.,* "KRDroid: Ransomware-oriented detector for mobile devices based on behaviors," *Appl. Sci.*, vol. 11, pp. 6557, 2020.
[12] S. Wang, S. Qin, N. He, T. Tu, J. Hou *et al.,* "KRRecover: An auto-recovery tool for hijacked devices and encrypted files by ransomwares on android," *Symmetry*, vol. 13, pp. 861, 2021.
[13] C. Chen, B. J. Cui, J. X. Ma, R. P. Wu, J. C. Guo *et al.,* "A systematic review of fuzzing techniques," in *Computers & Security*, pp. 118–137, 2018.
[14] W. Xie, Y. Jiang, Y. Tang, N. Ding and Y. Gao, "Vulnerability detection in IoT firmware: A survey," in *2017 IEEE 23rd Int. Conf. on Parallel and Distributed Systems (ICPADS)*, China, pp. 769–772, 2017.
[15] R. Russell, "Automated vulnerability detection in source code using deep representation learning," in *2018 17th IEEE Int. Conf. on Machine Learning and Applications (ICMLA)*, Canada, pp. 757–762, 2018.
[16] H. Wang, "Combining graph-based learning with automated data collection for code vulnerability detection," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 1943–1958, 2021.
[17] T. Ji, Y. Wu, C. Wang, X. Zhang and Z. Wang, "The coming Era of AlphaHacking?: A survey of automatic software vulnerability detection, exploitation and patching techniques," in *2018 IEEE Third Int. Conf. on Data Science in Cyberspace (DSC)*, pp. 53–60, 2018.
[18] Brumley, P. Poosankam, D. Song and J. Zheng, "Automatic patch-based exploit generation is possible: Techniques and implications," in *Proc. of the IEEE Symp. on Security and Privacy (S&P)*, 2008.
[19] T. Avgerinos, K. C. Sang, A. Rebert E. J. Schwartz, M. Woo and D. J. C. O. T. A. Brumley, "Automatic Exploit Generation," vol. 57, no. 2, pp. 74,76–84, 2014.
[20] K. Cha, T. Avgerinos, A. Rebert and D. Brumley, "Unleashing MAYHEM on binary code," in *Proc. of the IEEE Symp. on Security and Privacy (S&P)*, Oakland, 2012.
[21] https://kns.cnki.net/kcms/detail/detail.aspx?dbcode=CMFD&dbname=CMFDTEMP&filename=1021731589.nh&uniplatform=NZKPT&v=sDdoA8%25mmd2BXV4C8q7aykp%25mmd2FCVnhe0x14KmUUKx6X%25mmd2BSUbp1FBUMArUchY8XBkEs94URtJ.
[22] H. Wang, P. R. Su, Q. Li, L. Y. Ying, Y. Yang *et al.,* "Automatic polymorphic exploit generation for software vulnerabilities," in *Proc. of Int. Conf. on Security and Privacy in Communication Networks (SecureComm)*, Suzhou,China, 2013.
[23] J. Schwartz, T. Avgerinos and D. Brumley, "Q: Exploit hardening made easy," in *Proc. of the USENIX Security Symp.*, 2011.
[24] Y. Mao, Y. Li, J. Sun and Y. Chen, "Explainable software vulnerability detection based on attention-based bidirectional recurrent neural networks," in *2020 IEEE Int. Conf. on Big Data (Big Data)*, Beijing, China, pp. 4651–4656, 2020.
[25] M. Yi, X. Xu and L. Xu, "An intelligent communication warning vulnerability detection algorithm based on IoT technology," *IEEE Access*, vol. 7, pp. 164803–164814, 2019.
[26] Y. Tatarinova, "AVIA: Automatic vulnerability impact assessment on the target system," in *Int. Conf. 2018 IEEE Second. on Data Stream Mining & Processing (DSMP)*, pp. 364–368, 2018.
[27] H. Hu, Z. L. Chua, S. Adrian, P. Saxena and Z. K. Liang, "Automatic generation of data-oriented exploits," in *Proc. of the USENIX Security Symp.*, American, 2015.
[28] S. Volckaert, B. Coppens and B. De Sutter, "Cloning your gadgets: Complete ROP attack immunity with multi-variant execution," *IEEE Transactions on Dependable and Secure Computing*, vol. 13, no. 4, pp. 437–450, 2016.
[29] Z. J. Huang, T. Zheng and J. Liu, "A dynamic detective method against ROP attack on ARM platform," in *2012 Second Int. Workshop on Software Engineering for Embedded Systems (SEES)*, China, pp. 51–57, 2012.