future internet

MDPI

Article

A Survey of Security Vulnerability Analysis, Discovery, Detection, and Mitigation on IoT Devices

Miao Yu [1], Jianwei Zhuge [1,2,*], Ming Cao [3], Zhiwei Shi [3] and Lin Jiang [4]

1 Institute of Network Science and Cyberspace, Tsinghua University, Beijing 100091, China; yum18@mails.tsinghua.edu.cn
2 Beijing National Research Center for Information Science and Technology, Beijing 100000, China
3 China Information Technology Security Evaluation Center, Beijing 100085, China; cao_grace@yeah.net (M.C.); szw80286@163.com (Z.S.)
4 China Luoyang Electronic Equipment Test Center, Luoyang 471000, China; JL13sky@163.com
* Correspondence: zhugejw@cernet.edu.cn

Received: 24 December 2019; Accepted: 28 January 2020; Published: 6 February 2020

Abstract: With the prosperity of the Internet of Things (IoT) industry environment, the variety and quantity of IoT devices have grown rapidly. IoT devices have been widely used in smart homes, smart wear, smart manufacturing, smart cars, smart medical care, and many other life-related fields. With it, security vulnerabilities of IoT devices are emerging endlessly. The proliferation of security vulnerabilities will bring severe risks to users' privacy and property. This paper first describes the research background, including IoT architecture, device components, and attack surfaces. We review state-of-the-art research on IoT device vulnerability discovery, detection, mitigation, and other related works. Then, we point out the current challenges and opportunities by evaluation. Finally, we forecast and discuss the research directions on vulnerability analysis techniques of IoT devices.

Keywords: internet of things (IoT); vulnerability discovery; vulnerability detection; vulnerability mitigation

1. Introduction

Internet of Things (IoT) is becoming the most popular and practical online platform. It connects various sensors and controllers to the Internet and helps to achieve seamless communication between people and things. It tends to be the crucial future of the Internet. Especially in recent years, with the prosperity of the IoT industry, the variety and quantity of devices have grown rapidly. Globally, the total number of current active IoT devices has reached 7 billion [1]. They have been widely used in smart homes, smart wear, smart manufacturing, smart car, smart medical care, and many other life-related fields. We believe that it will greatly improve the quality of our lives.

At the same time, security vulnerabilities of IoT devices often occur, and they are very difficult to be eliminated. HP's report showed that 70% of IoT products contain security vulnerabilities, and, on average, there are 25 vulnerabilities per device [2]. The attacker engaged in various illegal activities by maliciously exploiting vulnerabilities and controlling devices. The most well-known case is in 2016; the Mirai virus controlled hundreds of thousands of IoT devices and built botnets by manipulating the controlled devices. It launched Tbps-level denial-of-service (DoS) attacks on targets, including the DNS service provider Dyn causing severe problems such as partial Internet paralysis in the United States [3]. In conclusion, with the universal usage of IoT devices, the proliferation of security vulnerabilities will bring severe risks to the security and privacy of the users and even the safety of human lives and property.

Facing frequent attacking risks, IoT security research has become increasingly popular. After the concept of "Internet of Things" was first proposed by American Auto-ID in 1999 [4], the security

researchers have also contributed to IoT by working on standards of security architecture and communications [5,6]. Subsequently, there was a lot of discussion about IoT security issues [7–12]. Zhang et al. [13] and Mahmoud et al. [14] pointed out the challenges and research directions. Therefore, researchers began to use traditional security research methods in the field of IoT Security [15]. With the development of artificial intelligence (AI), the survey of the machine and deep learning methods for IoT security has also emerged [16]. Alrawi et al. [17] systematically summarized the IoT vulnerabilities from device, mobile application, cloud endpoint, and communication in smart homes. For the summary of vulnerability analysis, Xie et al. [18] summed up techniques of detecting IoT vulnerability. Recently Zheng et al. [19] published a survey of IoT vulnerability discovery techniques. In the two papers above, the boundaries between vulnerability discovery and vulnerability detection technologies are blurred. In this paper, the technology of vulnerability discovery is to mine unknown vulnerabilities, and the technology of vulnerability detection is to detect the existence of known vulnerabilities. Through the above investigations, we find that the current study focuses on IoT security issues and lack analysis techniques. Secondly, in this kind of vulnerability analysis, they mainly focus on vulnerability discovery and detection and lack attention to the techniques of vulnerability mitigation. There is a problem that the technical summary of IoT security is not comprehensive enough.

In order to overcome the above problem, we want to make some contributions in three aspects:

- First, we shift our focus from IoT architecture to IoT devices. Second, the classification of IoT device security technologies has been refined. In addition, we summarize the current research, which is considered from the basic framework of vulnerability analysis, discovering the unknown vulnerability, detecting known vulnerability, and mitigating vulnerability.
- We evaluate the current research of vulnerability analysis on IoT devices. In addition, we analyze in depth the reasons that hinder the development of security technologies and point out the challenges and opportunities.
- We review the technological development context and point out future research directions for related researchers.

This paper is organized as below: Section 2 describes the IoT security background. It introduces the IoT architecture, the device components, and the attack surfaces. Section 3 reviews current research works related to IoT device security, including vulnerability analysis, discovery, detection, and mitigation. Section 4 summarizes the challenges and opportunities based on the evaluation of vulnerability analysis technology. Section 5 points out the hot-spot directions of future research. Finally, Section 6 gives the conclusions.

2. Background

2.1. IoT Architecture

With the rapid development of the Internet, more and more household and industrial devices are connected to the Internet, which offers us diversified lives. IoT architecture is mainly developed in two directions: consumer-level and industry-level.

On the consumer-level, we have several device types like industrial manufacturing, smart home, smart medical, and smart cars if they are divided by application scenarios. Among them, the development of smart home is relatively mature. The Internet giants—Samsung, Google, Apple, and XiaoMi have a large share of the market. In addition, the IoT platforms like SmartThings [20], Google Weave [21], Apple HomeKit [22], HomeAssistant [23], and XiaoMi IoT [24] are released. By investigating these platforms, we find that most IoT adopts the "Device <->Cloud<->User"'s architecture, as Figure 1 shows. The smart devices are generally deployed at homes. They communicate with cloud servers, and directly or indirectly access the network through WiFi [25], ZigBee [26], Blue-tooth [27], or other protocols. They upload the data that are collected by the sensor and receive the control command, which is issued to the actuator. The IoT architecture not only relies on the cloud

from the vendor, but also on the cloud from a third party. It supports mutually and offers diverse services for various functions. Users can connect to the cloud to view the status attribute and download data by their mobile phone or PC. For some simple scenarios such as wearable devices, the "Device <-> User" architecture is more practical.

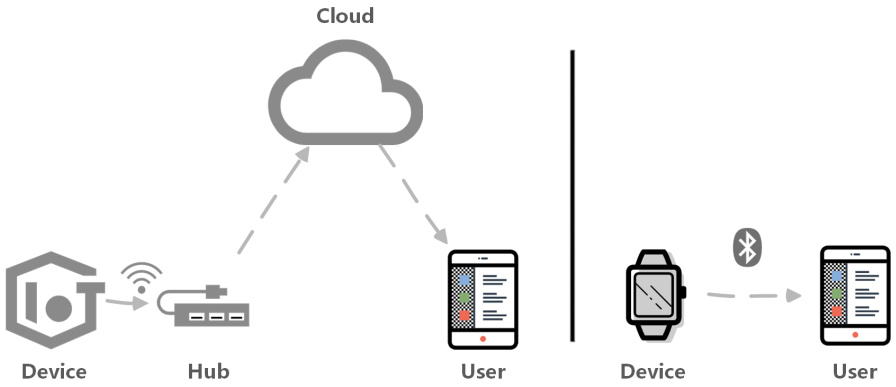

Figure 1. Internet of Things (IoT) architecture on the consumer-level.

On the industry-level, the IoT architecture continues the Information Technology (IT) approach to centrally manage to interact between users and devices by servers in Figure 2. The difference is that the apparatus must first communicate with the Programmable Logic Controller (PLC) through operational technology (OT). Thus, the devices in the industry are equivalent to PLC and "sensors + actuators." Security research focuses on PLC. The "Device <-> User" architecture also exists in industrial scenarios. Administrators use configuration software to control devices. Although user-oriented industrial terminals such as smart meters have also tried the cloud model, there is no large-scale promotion due to security considerations.

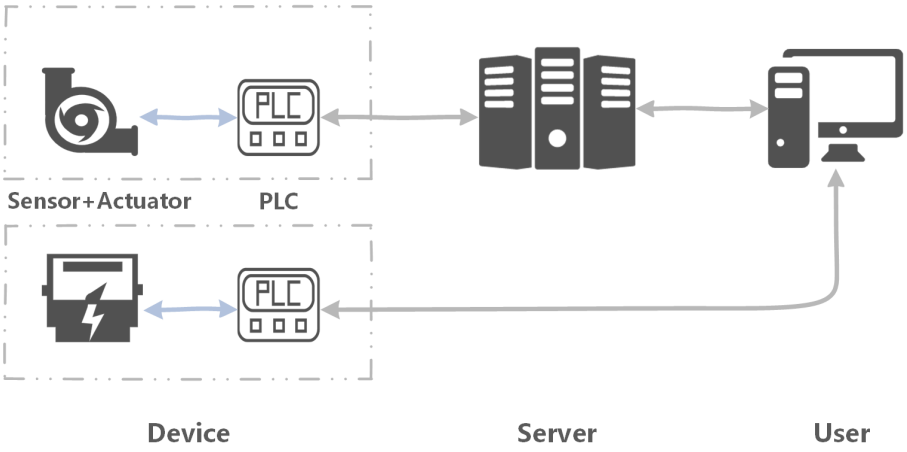

Figure 2. IoT architecture on the industry-level.

2.2. Device Composition

Whether a big and complex machine for car manufacturing or a small and smart bracelet for wearing, they contain relatively fixed components such as chips, flash, firmware, and so on. Its composition mainly includes hardware and software parts.

(1) Hardware parts:

- **Logic chip.** For complex devices, it has an operating system so that it needs multiple logic chips or CPU. Simple embedded devices may only use a single microprocessor to run programs.
- **Memory.** Provides the storage space for system and program running, ranging from a few KB to GB.

- **Flash storage.** The location where the IoT device firmware is stored. Part of the device's bootloader is also stored in the flash.
- **Network module.** The difference between IoT devices and traditional embedded devices is that they connect to the Internet. They generally adopt wireless technology to connect to the Internet with the hub, such as access points (APs).
- **Serial debug interface.** The IoT device often requires means for communicating with the external world for debugging. The serial debug interface could be to send and receive commands to and from the vendor developers. One of the most commonly used interfaces is the universal asynchronous receiver/transmitter (UART).

(2) Software parts:

- **BootLoader.** It is a small program. Before the IoT device system runs, it initializes the hardware device and loads the firmware to the boot device. Thus, it brings the system's software and hardware environment to a suitable state to prepare the correct environment.
- **Firmware.** The firmware includes the operating system, file system, and service programs. Security research on IoT devices generally starts with firmware analysis.

2.3. Attack Surface

IoT devices not only have attack surfaces in the field of traditional software security but also introduce new attack surfaces due to their special structure and requirements. According to the IoT architecture and device composition, attack surfaces can be divided into three layers in Figure 3.

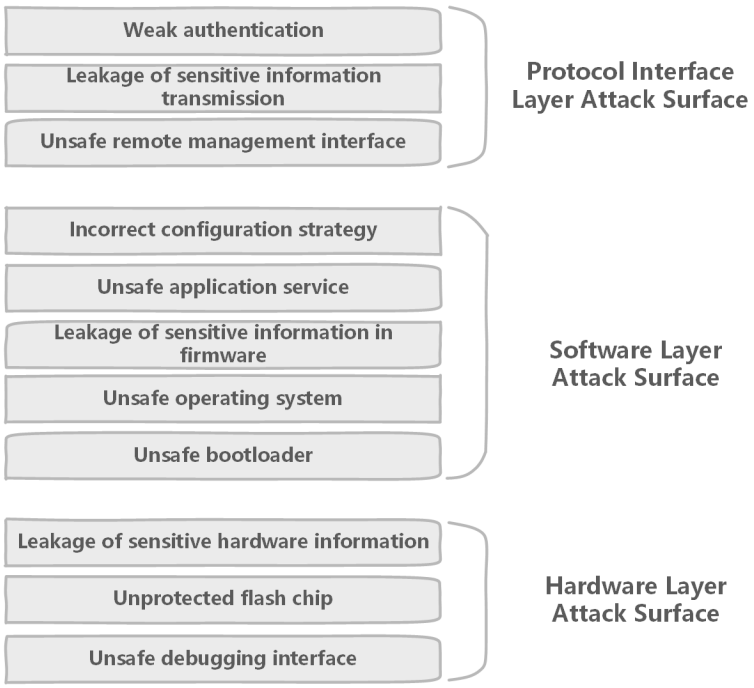

Figure 3. Attack surface of IoT device.

2.3.1. Attack Surface on the Hardware Layer

Attack surface on the hardware layer is different from the traditional security field. It mainly includes three aspects: unsafe debugging interface, unprotected flash chip, and leakage of sensitive hardware information.

1. **Unsafe debugging interface.** When the IoT device is manufactured, the debug interface such as UART is left on the circuit board to facilitate the repairing. If it is no authentication or weak

authentication, attackers can obtain high authority shell to modify or replace the firmware by the interface. The unsafe debugging interface is the first item on an IoT security check-list.

2. **Unprotected flash chip.** Because the flash chip is often used to store firmware, it has become the focus of attention. If the chip is not read-write protected, security researchers can read the firmware for analysis or write modified firmware to bypass authentication of interface access.
3. **Leakage of sensitive hardware information.** The hardware circuit layout is not well sealed. Leakage of hardware information such as sounds and power consumption causes a side-channel attack [28–31], which attackers can acquire important information such as encryption keys.

2.3.2. Attack Surface on the Software Layer

Attack Surface on the Software Layer corresponds to the software part of the bootloader and firmware in the device composition. It mainly includes the following five aspects: unsafe bootloader, unsafe operating system, leakage of sensitive information in firmware, unsafe application service, and incorrect configuration strategy:

1. **Unsafe bootloader.** It is often easy to ignore the point of attack because the bootloader is a piece of code that is loaded from the chip after the device running. Its function is to initialize the device and load the firmware. Thus, it has a high risk when problems arise. For example, *checkm8* [32], the Boot ROM exploit, has widely been proclaimed as the most important single exploit ever released for iPhone, iPad, Apple TV, and Apple Watch devices.
2. **Unsafe operating system.** Due to the short development cycle and lightweight requirements of the IoT device, the kernel of the operating system is tailored, and the version is usually not up-to-date, which causes various buffer overflow problems such as privilege escalation. In addition, devices use various sensors and communication modules including a large number of drivers in the kernel. For example, the Marvell WiFi chip driver was found multiple vulnerabilities such as CVE-2019-14901, CVE-2019-14897, and CVE-2019-14896 [33]. They cause stack-based or heap-based buffer overflow in the kernel. This is also an important part of the attack surface.
3. **Leakage of sensitive information in firmware.** Local storage of IoT devices generally uses a lightweight storage solution. Developers often ignore security and use plain text or simply encrypting data, which can easily lead to the leakage of sensitive information.
4. **Unsafe application service.** Application services development lacks security standards. Simple and unsafe application code is compiled and used directly to speed up product development. Therefore, it is easy to introduce unknown vulnerabilities. IoT security researchers have discovered a large number of application vulnerabilities developed by manufacturers, including backdoors that are unknown for some reason.
5. **Incorrect configuration strategy.** Services such as ssh, telnet are enabled for easy management of IoT products. There will be configuration problems. Weak authentication policies are configured by default, which allows attackers to easily obtain the shell of device. For example, Telestar Digital GmbH IoT radio devices could be exploited by remote attackers to hijack devices by telnet servers without authentication [34]. The vulnerabilities have been tracked as CVE-2019-13473 [35] and CVE-2019-13474 [36].

2.3.3. Attack Surface on the Protocol Interface Layer

The attack surface on the protocol interface layer represents communication and application programming interface (API). It involves the device directly controlled by the user side, the device indirectly controlled through the cloud, and the above two types of communication process information protection issues. Security on the protocol level is not involved. For example, the abuse of IoT communication protocols and the AR-DDoS [37] attack is performed by the IoT communication protocols Constrained Application Protocol (CoAP) [38], SSDP [39], and SNMP. Its target is not the flaws of IoT devices. However, it is also an important research direction of IoT security. Attack surface

on the protocol interface layer mainly includes the following three aspects: the unsafe interface of remote management, leakage of sensitive information transmission, and weak authentication.

1. **Unsafe interface of remote management.** For portable management, IoT devices use remote management interfaces such as HTTP services, which bring multiple vulnerabilities such as SQL injection, Cross-site Scripting (XSS), and remote execution vulnerability.
2. **Leakage of sensitive information transmission.** The IoT communication protocol will use weak encryption algorithms or even no encryption, which will lead to the leakage of sensitive information. For example *Passwords in the Air* [40], the WiFi password is transmitted in plain text when the IoT device is connected to the network.
3. **Weak authentication.** Due to security requirements, the management of IoT devices requires authentication binding. However, a new attack surface has emerged. Attackers can bypass authentication, duplicate bind, and obtain other user's information. The *Phantom Device Attack* [41] found four specific attack methods on this attack surface.

3. Vulnerability Analysis, Discovery, Detection, and Mitigation

At this stage, there is no precise classification of IoT security. In addition, the core of security research is vulnerability. Therefore, we focus on the device's vulnerability. Around its life cycle, the research process is divided into three stages: discovery, detection, and mitigation. Because of the particularity of IoT security, it is impossible to have standard interfaces to support analysis. Thus, research on the basic analytical framework of IoT is also valuable to research content. To comprehensively review IoT security technologies, we summarize by the following four aspects: (1) Research on the basic framework of vulnerability analysis, which performs firmware simulation to help analyze IoT security issues. (2) Research on vulnerability discovers the technology, which studies methods to discover unknown vulnerabilities in IoT devices. (3) Research on vulnerability detection, which studies methods to detect known vulnerabilities based on the features and signatures of existing vulnerabilities. (4) Research on vulnerability mitigation, which studies methods to automatically fix vulnerability or access control methods to limit malicious behavior. In addition, this section mainly summarizes the IoT vulnerability analysis technologies, which require a series of pre-conditions, such as firmware extraction [42]. Thus, we mark the technical requirements, but do not sum them up.

3.1. Research on the Basic Framework of Vulnerability Analysis

To address the growing concerns about the security of IoT systems, it is vital to perform an accurate analysis of firmware binaries, even when the source code or the hardware documentation is not available [43]. However, vulnerability analysis in the IoT security field is obstructed by the lack of dedicated the basic framework. For example, the dynamic analysis relies on the ability to execute software in a controlled environment, often an instrumented emulator [43]. Thus, the basic framework mainly provides the features of dynamic debugging by semi-simulation and full simulation methods. It can perform complex dynamic analyses to support IoT security research.

Technical requirements: The ability to fetch firmware of the IoT device.

For the lack of specialized analysis tools for firmware, especially dynamic analysis tools, *Avatar* [43] proposed a framework to analyze firmware combining both simulated execution mode on simulators and the actual execution mode on real devices. When the firmware is running in the simulation mode, *Avatar* forwards the operation to the actual device in the case of input/output (I/O) access. The real device returns the results to the simulator after dealing with operation so that the simulator can continue the execution. It effectively solves the problem of specific peripheral components without source code and documentation. Then, *Prospect* [44] and *Surrogate* [45] also proposed similar dynamic analysis frameworks. Four years later, the author's team of *Avatar* re-developed *Avatar2* [46], which allows security researchers to inter-operate between different dynamic analysis frameworks, debuggers, simulators, and real devices. In addition, the authors

also show how to use *Avatar2* to record the execution flow of the device. Chen et al. [47] proposed *Firmadyne*, focusing on Linux-based devices. The first use software for system-wide simulation, then adopt dynamic analysis methods such as scanning and probing to discover vulnerabilities. The simulation features of the above frameworks are based on QEMU [48]. For sensor operations that are not easy to simulate, semi-simulation frameworks [43–46] are to guide I/O operations to physical hardware by software agent methods when executing firmware instructions in Table 1.

Table 1. This table is a summary of the basic framework of vulnerability analysis. √ = Yes. Semi-simulation = The framework needs to rely on the real-world device to receive forwarded I/O access.

Ref.	Architecture Support			Simulation Type
	ARM	MIPS	x86	
Avatar [43]	√			Semi-simulation
Prospect [44]		√		Semi-simulation
Surrogate [45]	√			Semi-simulation
Avatar2 [46]	√			Semi-simulation
Firmadyne [47]	√	√		Full simulation

3.2. Research on Vulnerability Discovery

With the increase in the number of vulnerabilities in IoT devices and the rise of attack trends, security researchers pay more and more attention to the vulnerability mining of devices. This section describes the technology of vulnerability discovery, including dynamic analysis and static analysis. By studying traditional program security analysis, we find the dynamic analysis that involves fuzzing [49] and taint checking [50], while the static analysis involves symbolic execution [51], taint analysis, and data-flow analysis [50].

3.2.1. Dynamic Analysis Method

The dynamic analysis method needs tools of simulation firmware for dynamic debugging or performs on-chip debugging on a physical device to obtain feedback information. It mainly adopts fuzz testing to find the trigger point of the vulnerability.

Technical requirements: The ability to dynamically debug on an IoT device.

In card security research, Alimi et al. [52] used a universal algorithm to generate test samples and fuzz mobile phone cards or bank cards. For some modern smart cards containing web servers, Kamel et al. [53] have found some bugs based on the generated method of the HTTP protocol to fuzz these web servers. In terms of car safety, Koscher [54] and Lee [55] can change the state of the car by mutating the packets sent to the Controller Area Network (CAN) bus [56] to a fuzz smart system of the car. Due to the difficulty of extracting firmware from the IoT device, *IoTFuzzer* [57] captures crash information by the user side to avoid this problem. Firstly, it inserts a stub to the interaction protocol code of the mobile application. Secondly, the authors of *IoTFuzzer* mutate data that are captured from the stub and sent to the device. Finally, they judge the effect of fuzzing by heartbeat packets and response. Because devices are difficult to debug directly, researchers have begun to combine simulation technology to find the vulnerability. Costin et al. [58] implement the fully automated framework that applies dynamic firmware analysis techniques to achieve automated vulnerability discovery of Web interfaces within embedded firmware images. Recently, targets of Srivastava et al. [59] are no longer limited to web interfaces. They present FirmFuzz [59], an automated device-independent emulation and dynamic analysis framework for Linux-based firmware images (camera and router). Zheng et al. [60] proposed Firm-AFL, the first high-throughput greybox fuzzer for IoT firmware. In addition, they extended AFL [61], which is the currently popular fuzzer to the field of IoT. For research of fuzzing, Muench et al. [62] analyzed the universality of traditional anomaly state detection methods for the IoT device, and they implemented a system based on *Avatar* [43] and *Panda* [63]. In addition,

they compare the throughput of a blackbox fuzzer under different configurations, including native execution (directly sending inputs to the hardware), partial emulation (redirecting only hardware requests to the hardware), and full emulation [60]. This is a performance evaluation of vulnerability analysis techniques. In conclusion, the types of vulnerabilities discovered by the aforementioned dynamic analysis techniques are diverse in Table 2. They are mainly memory issues such as buffer overflow (OB) and null pointer dereference (NPD). There will also be some web server vulnerabilities such as XSS, SQL injection because of the study of the web interface.

3.2.2. Static Analysis Method

The static analysis method can discover vulnerabilities in IoT devices without executing firmware. The process provides an understanding of the program code to find bugs. Thus, it is generally more scalable.

Technical requirements: The ability to fetch firmware of IoT devices.

The analytical static analysis process is as follows: (1) Extract the firmware. (2) Reverse binary program in firmware. (3) Find the security problem by manual audit. In academia, researchers mainly explore automated static analysis methods to find vulnerabilities. In Table 2, we summarize the static analysis, including research targets, subdivided technology, and types of finding vulnerabilities. Costin et al. [64] first analyze the firmware of embedded devices on a large scale and automatically. They automatically decompress and process firmware and use fuzzy hashes to match weak keys in firmware. *FIE* [65] based on *KLEE* [66] constructs the symbolic execution engine of embedded devices. It formulates memory specification, interrupts specification and chip specification to find out the problem of violating custom security specification in firmware. *Firmalice* [67] is also based on the symbolic execution method, which finds the authentication bypass vulnerability through the input determinism of the backdoor. For the taint analysis method, *SainT* [68] and *DTaint* [69] adopt static methods to discover vulnerability on the basis of APP or binary code of devices, respectively.

Table 2. This table is a summary of IoT device vulnerability discovery technology. BO = Buffer Overflow. NPD = Null Pointer Dereference. CI = Command Injection. CSRF = Cross-site Request Forgery

Category	Ref.	Target	Technology	Types of Finding Vulnerabilities
Dynamic Analysis	Alimi [52]	Smart Card	Fuzzing	Logic Vulnerability
	Kamel [53]	Smart Card	Fuzzing	Logic Vulnerability
	Kosche [54]	Smart Car	Fuzzing	Weak Access Control
	Lee [55]	Smart Car	Fuzzing	Weak Access Control
	IoTFuzzer [57]	Smart Home	Fuzzing	BO, NPD
	Costin [58]	Router	Fuzzing	CI, XSS, CSRF, SQL Injection
	FirmFuzz [59]	Smart Home	Fuzzing	BO, NPD, CI, XSS
	Firm-AFL [60]	Smart Home	Fuzzing	BO, NPD
Static Analysis	Costin2014 [64]	Binary code	Fuzzy hash	Weak Authentication, Backdoor
	FIE [65]	Binary code	Symbolic execution	BO
	Firmalice [67]	Binary code	Symbolic execution	Backdoor
	SainT [68]	APP	Static Taint analysis	Data Leakage
	DTaint [69]	Binary code	Static Taint analysis	BO, CI

3.3. Research on Vulnerability Detection

The dynamic and static analysis techniques from the previous section can also be applied to detect known vulnerabilities. In large-scale detection scenarios, the dynamic analysis relies on architecture-specific tools to execute. In addition, the static analysis method detects known vulnerabilities by way of mining zero-day, increasing performance, and time consumption. At present, researchers mainly adopt the following two methods: network scanning and code similarity detection.

3.3.1. Network Scanning Method

The network scanning method detects known vulnerability by sending probe packets with payload to services of online IoT devices. It is more versatile in the security field. With the development of IoT security, special topics for IoT devices of network scanning emerge.

Technical requirements: The ability to know vulnerability information such as Proof of Concept (PoC).

Cui et al. [70] scan existing embedded devices on the Internet to discover a list of devices with weak password and other types of vulnerabilities. After 2013, search engines such as Shodan [71], Censys [72], and Zoomeye [73] emerge, which identify and detect weak passwords, backdoor, and known vulnerability. However, it is difficult to find most security vulnerabilities only by external scanning. In addition, there are ethical issues with unauthorized analysis of devices on the Internet. Thus, the vulnerability scanning is typically performed in laboratories and intranets. The advantage of this method is that the detection from the service layer does not need to consider the structure of the device. It is fast, effective, and suitable for large-scale testing. The current commercial vulnerability detection systems are mainly based on this method.

3.3.2. Similarity Detection Method

Security researchers have introduced software code similarity detection methods to detect known vulnerabilities due to a large number of unpatched known vulnerabilities in IoT devices. At this stage, the research on similar detection is mainly aimed at the traditional software security field and gradually supports IoT devices by across-architectures. There is no research paper specifically for IoT firmware similarity detection. In Figure 4, the basic idea of similarity detection method is to extract the original features from the code such as strings, an instruction sequence, basic block, syntax tree, function call graph, and so on. Then, these features are measured similarly by the algorithm. Finally, it is determined whether there is a vulnerability in the corresponding code fragment. This section is mainly divided into the following two points: similarity detection on source code and similarity detection on binary code.

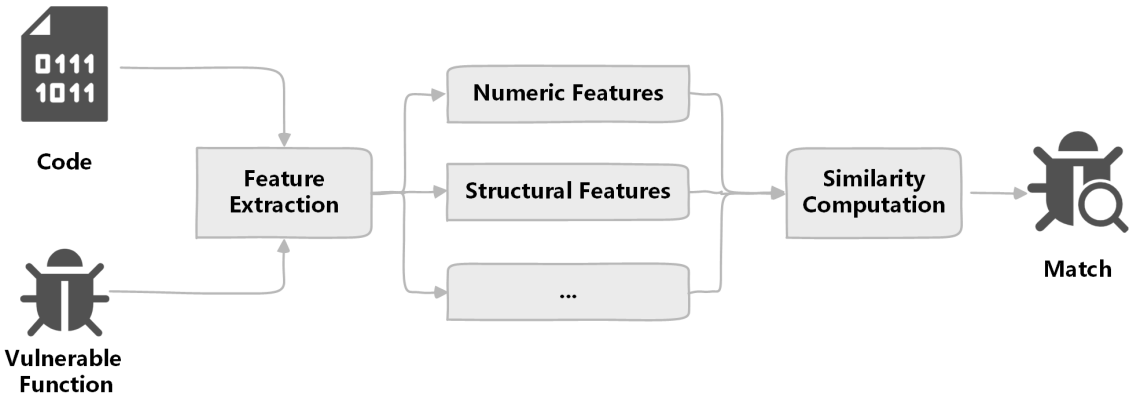

Figure 4. Similarity detection architecture.

Technical requirements: The ability to fetch firmware of IoT devices.

(1) Similarity Detection on Source Code

In terms of detecting known vulnerabilities based on source code, *CP-Miner* [74] adopts a token-based method that uses a lexer to generate a token sequence and search for repeated token sequences to measure similarity. *ReDeBug* [75] proposed a scalable method that can combine patch code to determine features of the vulnerability code before repaired, and it provides detecting the unpatched code clones. However, the above code-based approach does not apply to IoT. In most cases, security researchers can not obtain the source code of firmware.

(2) Similarity Detection on Binary Code

In terms of detecting known vulnerabilities based on binary code, researchers mainly faced the problem that it is difficult to detect code similarity due to different compiler code generation algorithms, different compiler optimization options, and different instruction sets. *N-Grams* [76] and *N-Perms* [77] are early means for vulnerability search [78]. Karim et al. [79] use binary sequences or code in memory to match algorithms without any understanding of code semantics. Thus, these kinds of methods are difficult to deal with the opcode reordering problem caused by different compilation options. To improve the matching accuracy, the *Tracelet-based* [80] approach reconstructs the code into an execution sequence and uses a solver to handle its constraints and data constraints. Thus, it solves the problem of operation code disorder. Furthermore, *TEDEM* [81] adopts symbols to simplify binary programs and judge the similarity of code by tree editing distance as the basic blocks. It can even find vulnerabilities on different operating systems.

Due to some common syntactic features, features of the basic block are difficult to express similarities between binary files. Researchers began to consider adopting the program of Control Flow Graph (CFG) [82] to describe the behavior of the program. Therefore, the similarity comparison can be performed by the graph. *BinDiff* [83] and *Binslayer* [84] check the similarity between the two binaries based on the similarity measure of CFG isomorphism, but not specifically designed for vulnerability detection. It is difficult to find cross-platform vulnerability code fragments by comparing two completely different binaries of CFG. Egele et al. [85] proposed *Blanket Execution* and point out that the research on establishing semantic similarity of binaries based on static analysis is easily affected by compilation chain and compilation optimization level. Therefore, they suggested extracting the dynamic run-time features of the program to counter the changes of CFG caused by the above reasons. *BinHunt* [86] and *iBinHunt* [87] use symbolic execution and theorem proving techniques to examine the semantic equivalence between basic blocks and find out which semantics are different.

However, the firmware of IoT devices is highly heterogeneous, including multiple architectures such as MIPS, ARM, PPC, x86, and so on. Their opcodes, register names, and memory addressing methods are different. Thus, the above methods are difficult to be applied to cross-architecture code vulnerability detection on a large scale. Until the last two or three years, researchers began to study the issues of cross-architecture similarity detection based on binary code [88–90]. *Multi-MH* [88] is the first proposed binary-based method of similarity detection for cross-architecture. Above all, the binary code is converted into intermediate code. Then, this method uses specific input to test the program and captures the semantics of the base block based on the behavior of I/O. Finally, it adopts captured CFG to detect vulnerabilities. However, its performance overhead is too expensive in the face of a large set of functions. *DiscovRE* [89] checks whether the CFG of a set of function pairs is similar through the graph matching algorithm and accelerates CFG matching by pre-filtering. However, its pre-filtering process is not reliable and leads to too many under-reporting of vulnerabilities. *BinGo* [90] captures complete functional semantics by introducing selective inline related library functions and user-defined functions for cross-platform code search. However, it is not designed specifically for IoT devices. The *Genius* [91] uses a traditional method of machine learning to learn high-level feature representations from CFG. In addition, it encodes graph embedding [92] as a high-dimensional numerical feature vector. Then, the graph matching algorithm is used to measure the similarity between the objective function and a set of function binaries, which can effectively improve the performance and scalability. Xu et al. [93] first adopt the deep learning method for cross-platform similarity detection on binary code, which is graph embedding technology based on the neural network model. In cross-version code similarity detection, *αDiff* [94] has taken an important step. It extracts three semantic features, including function, inter-function, and inter-module features, to detect based on the Deep Neural Networks (DNN) model. Gao et al. also proposed *VulSeeker* [95] and *VulSeeker-Pro* [96], those vulnerability search methods combined with a deep learning model to improve the accuracy of vulnerability detection. These two methods were verified to be more accurate than existing methods such as *Gemini* [93].

3.4. Research on Vulnerability Mitigation

Based on vulnerability discovery and detection, mitigation is also a research issue of concern to the industry. According to public literature of research, the main research hot-spots are automated patch generation and access control. The former research aims to fix vulnerabilities, while later research can limit malicious behavior.

3.4.1. Automated Patch Generation

The technology of automated patch generation in this section is not specifically targeted at the IoT field but an extension of the traditional security field. The vulnerability repair work is usually done at the source level by the vendor development team. After obtaining the external vulnerability report, they eliminate the vulnerability by reproducing the vulnerability trigger condition and analyzing the vulnerability mechanism. Automatic patch generation holds out the promise of automatically correcting software defects without the need for developers to diagnose, understand, and correct these defects manually [97].

Technical requirements: The ability to fetch and update IoT device firmware.

The researchers in the field of software engineering proposed to automatically generate the patch by learning the correct code in the C language [97,98], Java language [99], and other source code levels, which achieved the initial feasible effect. Another idea is to change the form of the program without changing its function. *GenProg* [100] uses an extended form of genetic programming to evolve a program variant that retains required functionality but is not susceptible to a given defect. However, it can generate nonsensical patches due to the randomness of mutation operations. Thus, Kim et al. [101] proposed the Pattern-based Automatic program Repair (PAR) to solve the above problem. In terms of the android platform, Zhang et al. [102] proposed *AdaptKpatch*, which is an adaptive kernel hotfix framework and LuaKpatch which inserts a type-safe dynamic language engine into the kernel to execute patches. These solutions solve the problem that the patch chain of the Android platform is too long, the fragmentation and the ecological layout are not matched, and the subdivision repair is not timely. However, they do not consider solving the problem of the automatic generation of hotfixes in the cross-CPU architecture. They still need to be manually written based on the field of knowledge and experience. The Cyber Grand Challenge (CGC) [103] of DARPA drives researchers to work on automated defense methods at the binary code level. However, these methods mainly adopted generalized defense mechanisms such as binary code hardening [104,105], boundary checking, and pointer patching [106].

3.4.2. Access Control Method

The access control method is to manage the IoT device's permission for the user side or the platform to restrict or stop the malicious behavior of attackers.

Technical requirements: Scalability for the user side or the cloud side.

Fernandes et al.'s [107] first in-depth study of IoT security focused on a platform such as SmartThings. They found that great majority applications are overprivileged due to the capabilities being too coarse-grained, and devices used to communicate asynchronously with applications via events, which do not sufficiently protect events that carry sensitive information such as lock codes [107]. Many devices in the smart home are excessive permissions and ambiguous permissions management resulting in attacks on IoT devices and disclosure of privacy. Researchers at the University of Michigan have come up with a series of solutions to solve these problems. In 2016, they proposed *Flowfence* [108], a system based on data flow to protect privacy leakage. The application is divided into two components: (1) A set of Quarantined Modules that operate on sensitive data in sandboxes, and (2) Code that does not operate on sensitive data but orchestrates execution by chaining Quarantined Modules together via taint-tracked opaque handles—references to data that can only be dereferenced inside sandboxes. Then, in 2017, they implemented *ContexIoT* [109] based on context information, which can help users

implement effective access control to prevent attackers from performing dangerous operations by identifying sensitive operation context identification and ensuring context integrity in runtime. Finally, *Tyche* [110] was proposed in 2018, a risk-based permission model for smart homes to solve the problem of excessive access permissions by building an Access Control Capabilities Lists (ACCLs) based on the source code level. *Smartauth* [111] and *FACT* [112] are also based on the ACCLs. However, they build the ACCLs in different ways. *Smartauth* builds by documents that are identified by natural language processing (NLP) technology and APP source code, while *FACT* builds during the phase of device development.

4. Discussion

In the previous section, we have investigated in-depth research on the technologies of IoT vulnerability analysis at the present stage. In this section, firstly, we evaluate the vulnerability analysis technology. Secondly, we point out the challenges of current research by evaluation. Finally, we propose technological opportunities to deal with these challenges.

4.1. Evaluation

In Table 3, we evaluate from five aspects, including attack surface, technical requirement, architecture support, operating system support, and combining with AI. During vulnerability analysis, researchers need the support of technologies such as simulation, debugging interface, network traffic, Firmware, and APP. These technical requirements define the methodology and purpose of the study. For example, the *IoTFuzzer* [57] uses the analysis method of peripheral systems by transferring the target to the APP. The advantage of this method is its better generality to avoid the complexity of the architecture. Its disadvantage is that the coarse-grained crash information hinders the further analysis of the vulnerability. The assessment of these aspects makes it easy to analyze the technical challenges and future development trends.

Table 3. This table is evaluation of vulnerability analysis techniques. √ = Yes. S = Attack surface on software layer. P = Attack surface on protocol interface layer.

Component	Ref.	Attack Surface	Technical Requirement						Architecture Support				OS Support				With AI
			Simulation	Debugging Interface	Netwrok Traffic	Firmware (Binary)	Source Code	Cloud (APP)	ARM	MIPS	X86	Other	Linux	Vxworks	Windows	Android	
Basic Framework	Avatar, 2014 [43]	S	√	√		√			√				√				
	Prospect, 2014 [44]	S	√			√				√			√				
	Surrogate, 2015 [45]	S	√	√		√			√				√				
	Avatar2, 2018 [46]	S	√	√		√			√				√				
	Firmadyne, 2016 [47]	S, P1	√			√			√	√			√				
Dynamic Analysis	Alimi, 2014 [52]	P1			√							√					
	Kamel, 2013 [53]	P1			√							√					
	Koscher, 2010 [54]	S			√							√					
	Lee, 2015 [55]	S			√							√					
	IoTFuzzer, 2018 [57]	S, P1						√	√	√	√	√	√	√	√	√	
Static Analysis	Costin, 2016 [58]	P1	√			√			√	√			√				
	FirmFuzz, 2019 [59]	S, P1	√			√			√	√			√				
	FIRM-AFL, 2019 [60]	S, P1	√			√			√	√			√				
	Costin, 2014 [64]	S, P1				√			√	√			√	√			
	FIE, 2013 [65]	S				√						√					
	Firmalice, 2015 [67]	S				√			√				√	√			
	SainT, 2018 [68]	S, P1						√	√	√	√	√	√	√	√	√	
	DTaint, 2018 [69]	S				√			√	√			√				
Network Scanning	Cui, 2010 [70]	S, P			√				√	√	√	√	√	√	√	√	
	Shodan [71]	S, P			√				√	√	√	√	√	√	√	√	
	Censys, 2015 [72]	S, P			√				√	√	√	√	√	√	√	√	
	Zoomeye [73]	S, P			√				√	√	√	√	√	√	√	√	

Table 3. *Cont.*

Component	Ref.	Attack Surface	Technical Requirement						Architecture Support				OS Support				With AI
			Simulation	Debugging Interface	Netwrok Traffic	Firmware (Binary)	Source Code	Cloud (APP)	ARM	MIPS	X86	Other	Linux	Vxworks	Windows	Android	
Similarit Detection	CP-Miner, 2004 [74]	S					√				√		√				
	ReDeBug, 2012 [75]	S					√				√		√				
	Rendezvous, 2013 [78]	S				√					√		√				
	Karim, 2005 [79]	S				√					√				√		
	Tracelet-based, 2014 [80]	S				√					√		√				
	TEDEM, 2014 [81]	S				√					√		√		√		
	BinDiff, 2005 [83]	S				√			√	√	√		√		√		
	Binslayer, 2013 [84]	S				√					√		√				
	Egele, 2014 [85]	S				√					√		√				
	BinHunt, 2008 [86]	S				√					√				√		
	iBinHunt, 2012 [87]	S				√					√		√				
	Multi-MH, 2015 [88]	S				√			√	√	√		√				
	DiscovRE, 2016 [89]	S				√			√	√	√		√		√		√
	BinGo, 2016 [90]	S				√			√		√		√		√		√
	Genius, 2016 [91]	S				√			√	√	√		√				√
	Xu, 2017 [93]	S				√			√	√	√		√				√
	αDiff, 2018 [94]	S				√			√	√	√		√				√
	VulSeeker, 2018 [95]	S				√			√	√	√		√				√
	VulSeeker-Pro, 2018 [95]	S				√			√	√	√		√				√
Automated Patch Generation	Long, 2015 [97]	S					√						√				
	Long, 2016 [98]	S					√						√				
	Long, 2017 [99]	S					√						√				
	GenProg, 2011 [100]	S					√						√				
	Kim, 2013 [101]	S					√						√				
	AdaptKpatch, 2016 [102]	S				√			√							√	
	Shoshitaishvili, 2017 [105]	S				√							√		√		
	Shoshitaishvili, 2018 [104]	S				√							√		√		
	Xandra, 2018 [106]	S				√							√		√		
Access Control	Flowfence, 2016 [108]	P						√	√	√	√	√	√	√	√	√	
	ContexIoT, 2017 [109]	P						√	√	√	√	√	√	√	√	√	
	Tyche, 2018 [108,110]	P						√	√	√	√	√	√	√	√	√	
	Flowfence, 2016 [108]	P						√	√	√	√	√	√	√	√	√	
	SmartAuth, 2017 [111]	P						√	√	√	√	√	√	√	√	√	√
	FACT, 2017 [112]	P						√	√	√	√	√	√	√	√	√	

4.2. Challenges

The above evaluation reveals the challenges of current research of vulnerability analysis on IoT devices. As shown in Table 4, the impact on various technical fields is different. For IoT device vulnerability analysis technology, the challenges are as follows:

Table 4. This table summarizes impact scope of challenges and opportunities. The scope of influence includes four kinds such as basic framework of vulnerability analysis (T1), technology of vulnerability discovery (T2), technology of vulnerability detection (T3), technology of vulnerability Mitigation (T4). √ = Challenge or opportunity affects this field of technology.

Category	Name	T1	T2	T3	T4
Challenges	Complexity and heterogeneity of device	√	√	√	√
	Limitations of device resources	√	√	√	
	Closed-source measures	√	√	√	√
Opportunities	Application of AI technology		√	√	√
	Dependency of third-party and open source code		√	√	√
	Development of peripheral systems		√	√	

(1) Complexity and Heterogeneity of Device

This issue has always been the biggest challenge of IoT device vulnerability analysis technology. The IoT device is more heterogeneous than PC and mobile. It uses many CPU architectures such as ARM, MIPS, x86, and different types of operating system platforms such as Linux, Windows, and Android. It usually customizes firmware and memory usage. This makes it difficult to directly apply the industry's automated detection and discovery of vulnerabilities to the IoT field. The complexity of IoT devices aggravates the difficulty of static and dynamic analysis techniques. We find that arm-based Linux devices such as routers are selected as research targets mostly at this stage. The research on similarity detection expands cross-architecture scenarios [88–91,93–96]; others do not challenge this issue.

(2) Limitations of device resources

IoT devices generally run a reduced operating system or even run a single program on a microcontroller due to the lightweight requirements of products. The above reasons create the characteristics of limited devices resources. For the program of IoT device security testing, it is not very easy to deploy related analysis modules to the target to implement monitoring analysis on the periphery of the running program. Security researchers can not use traditional security analysis methods and tools. They need to restructure the analysis platform. In addition, dynamic analysis performance is reduced because the computing power of the device hardware is limited. In recent years, researchers have built simulation systems to address this challenge in the field of basic framework [43–47] and vulnerability discovery [59,60,64]. However, it has not been solved well, and this is still a long-term challenge.

(3) Closed-Source Measures

For general software, we can mine or detect source code or binary programs. For IoT device manufacturers, these can not be applied due to their closed source strategy. Source code analysis is no longer applicable to IoT vulnerability analysis, such as similarity detection on source code in Section 3.3.2. They even encrypt the firmware and strengthen the authentication of the serial debugging interface and think it is safer. For example, for the latest firmware of Dlink DIR-882(867, 878), 360 clear robots are all encrypted. Thus, vulnerability analysis based on source code, firmware, and the debugging interface is becoming increasingly difficult. Through previous evaluations, we find that vulnerability discovery and detection technology have avoided relying on these requirements, such as debugging interface [59,60], and firmware [57] in the past two years. However, there are new problems such as incomplete information.

4.3. Opportunities

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

(1) Application of AI Technology

In recent years, two technological waves of AI and IoT have emerged and integrated, promoting society into the era of AIoT (AI + IoT). The development of AI technology has also brought new solutions and methods to the security of IoT. At present, there are related studies using AI technology for access control [111] and similarity detection [89–91,93–96]. With the development of IoT and AI, new vulnerability discovery, detection, and mitigation technologies inevitably appear. When AI technology is applied to IoT devices, it is also a new opportunity of AI adversarial attack and defense. For example, the attacker pollutes the training set of the smart speaker and induces it to reply to a question with some negative information (abusive words). Security researchers prevent these problems by modifying AI algorithms.

(2) The Dependency of Third-Party and Open-Source Code

IoT firmware development relies heavily on third-party and open-source code. The manufacturers usually take new features, high performance, and low power consumption as the main targets of their products and shorten the development cycle as much as possible to enhance market competitiveness. Therefore, they adopt the agile development model. Many IoT manufacturers directly reuse open source code, refer to public code implementation, cross-compile PC platform code and rely on third-party libraries. Cui et al. [113] found that 80.4% of printer firmware contained multiple known vulnerabilities at the time of release, and many of the latest released firmware updates still contained third-party library vulnerabilities that were announced eight years ago. Although this has exposed a large number of security issues, it has led to unique vulnerability discovery technologies. It is possible to mine homology vulnerabilities through the similarity of different levels of information. The similarity detection will also advance the application in the IoT field.

(3) Development of Peripheral Systems

IoT devices are becoming more interactive. It tends to improve to promote the development of IoT peripheral systems. IoT devices usually interact with terminals (mobile and PC), cloud endpoint, and other systems because of the characteristics of IoT. It not only adds new attack surfaces but also helps the development of peripheral analysis technology to solve the problem of difficult firmware acquisition and analysis. For example, the current research of *IoTFuzzer* [57] and access control framework [108–112] all have automated analysis and protection by peripheral systems.

5. Research Directions

In the previous sections, we have introduced challenges and opportunities. We find that IoT vulnerability discovery, detection, and mitigation technologies continue the trajectory of traditional security research but also have their different research directions.

- **AI-based vulnerability discovery and detection technology**. Whether function or security, IoT and AI technologies are rapidly converging. The current AI technology is successfully used in vulnerability detection. As research progresses, it will expand to other vulnerability analysis techniques. For example, Generative Adversarial Networks (GANs) [114] have been applied in abnormal detection of IoT system behavior [115]. In the future, GANs may have a potential application in IoT vulnerability discovery because they may learn different attack scenarios to generate samples similar to a zero-day attack and provide algorithms with a set of samples beyond the existing attacks [16].
- **Large-scale vulnerability analysis techniques**. Complexity and heterogeneity of IoT devices hinder automation and large-scale analysis research in Section 4.2. However, this demand has been urgent in the IoT security industry. Security researchers need a cross-platform approach to overcome this problem, which is a long-term research direction.

- **Automated vulnerability exploiting**. To exploit the vulnerability in IoT devices and protect the device from intrusion, we need to generate PoC in an automated way. It helps to understand the hazards and causes of vulnerabilities better. With the development of the IoT field, the automation attack and defense will also become a hotspot.
- **Vulnerability analysis based on a peripheral system**. Through the above challenges, we found that it is aggravatingly difficult to analyze devices by static and dynamic methods directly. IoT devices are becoming more interactive. Not only will there be more and more vulnerabilities in combination with peripheral systems, but also study on peripheral system analysis methods will increase.
- **Automatic generation patch of multi-platform on binary code**. For some IoT vendors' closed-source and security inaction, device firmware can not be patched in time. To this end, we need an automated repair method for cross-platform binary code vulnerabilities. The automated patch generation on the binary code level requires fully understanding the formation mechanism and the elimination condition. There will be thousands of security vulnerability templates if we rely entirely on expert domain knowledge. Thus, it is difficult to achieve a scaled and feasible solution. At the same time, the variety of operating systems and hardware architecture brings technical challenges. It is a long-term goal of the whole security field to solve the problem of the automatic generation of multi-platform binary code patches.

6. Conclusions

With the rapid development of the IoT, users' security and privacy protection bring significant impact and challenge. Although the research on the security of IoT devices has gradually risen, it is still in the start-up stage in the information security field. Thus, a comprehensive summary of current research is needed to guide the development of IoT security. This paper analyzes IoT architecture and attack surfaces from the consumer-level and industry-level. It reveals the background of current research. We first refined the classification from four aspects: analysis tool, discovering, detecting, and mitigating vulnerability. Based on the above aspects, we review the technologies of vulnerability analysis. Moreover, we summarize targets, features, and research directions. Then, we evaluate vulnerability analysis techniques and find that the current research faces challenges, including complexity and heterogeneity of devices, limitations of device resources, and closed-source measures for a long time. Opportunities also accompany by challenges. The technologies about AI and peripheral system analysis will appear widely in the field of IoT security. In the future, there will be more and more technologies combined with new fields to implement automated vulnerability analysis on large-scale and cross-architecture.

Author Contributions: Conceptualization, J.Z.; Funding acquisition, J.Z. and Z. S.; Investigation, M.Y.; Project administration, J.Z. and Z.S.; Writing—original draft, M.Y.; Writing—review and editing, J.Z. and Z.S.; M.C. and L.J. All authors have read and agreed to the published version of the manuscript.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Lueth, K.L. State of the IoT 2018: Number of IoT Devices Now at 7B—Market Accelerating. Available online: https://iot-analytics.com/state-of-the-iot-update-q1-q2-2018-number-of-iot-devices-now-7b/ (accessed on 6 December 2019).
2. Rawlinson, K. Internet of Things Research Study. Available online: https://www8.hp.com/us/en/hp-news/press-release.html?id=1744676 (accessed on 6 December 2019).
3. Wikipedia. Mirai(malware). Available online: https://en.wikipedia.org/wiki/Mirai_(malware) (accessed on 6 December 2019).
4. Trevor, H. Internet of Things (IoT) History. Available online: https://www.postscapes.com/iot-history/ (accessed on 6 December 2019).

5. Gan, G.; Lu, Z.; Jiang, J. Internet of things security analysis. In Proceedings of the International Conference on Internet Technology and Applications, Wuhan, China, 16–18 August 2011.
6. Suo, H.; Wan, J.; Zou, C.; Liu, J. Security in the internet of things: A review. In Proceeding of the International Conference on Computer Science and Electronics Engineering, Hangzhou, China, 23–25 March 2012.
7. Zhao, K., Ge; L. A survey on the internet of things security. In Proceedings of the 2013 Ninth International Conference on Computational Intelligence and Security, Leshan, China, 14–15 December 2013.
8. Pescatore, J.; Shpantzer, G. *Securing the Internet of Things Survey*; SANS Institute: Bethesda, MD, USA, 2014; pp. 1–22.
9. Balte, A.; Kashid, A.; Patil, B. Security issues in Internet of things (IoT): A survey. *Int. J. Adv. Res. Comput. Sci. Softw. Eng.* **2018**, *5*, 450–455.
10. Ngu, A.H.; Gutierrez, M.; Metsis, V.; Nepal, S.; Sheng, Q.Z. IoT middleware: A survey on issues and enabling technologies. *IEEE Int. Things J.* **2016**, *4*, 1–20, doi:10.1109/JIOT.2016.2615180.
11. Yang, Y.; Wu, L.; Yin, G.; Li, L.; Zhao, H. A survey on security and privacy issues in Internet-of-Things. *IEEE Int. Things J.* **2017**, *4*, 1250–1258, doi:10.1109/JIOT.2017.2694844.
12. Alaba, F.A.; Othman, M.; Hashem, I.A.T.; Alotaibi, F. Internet of Things security: A survey. *J. Net. Comput. Appl.* **2017**, *88*, 10–28, doi:10.1016/j.jnca.2017.04.002.
13. Zhang, Z.K.; Cho, M.C.Y.; Wang, C.W.; Hsu, C.W.; Chen, C.K.; Shieh, S. IoT security: ongoing challenges and research opportunities. In Proceedings of the 7th IEEE International Conference on Service-Oriented Computing and Applications, Matsue, Japan, 17–19 November 2014.
14. Mahmoud, R.; Yousuf, T.; Aloul, F.; Zualkernan, I. Internet of things (IoT) security: Current status, challenges and prospective measures. In Proceedings of the 10th International Conference for Internet Technology and Secured Transactions (ICITST), London, UK, 14–16 December 2015.
15. Fernandes, E.; Rahmati, A.; Eykholt, K.; Prakash, A. Internet of things security research: A rehash of old ideas or new intellectual challenges. *IEEE Secur. Priv.* **2017**, *15*, 79–84, doi:10.1109/MSP.2017.3151346.
16. Al-Garadi, M.A.; Mohamed, A.; Al-Ali, A.; Du, X.; Guizani, M. A survey of machine and deep learning methods for internet of things (IoT) security. *arXiv* **2018**, arXiv:1807.11023. Available online: https://arxiv.org/abs/1807.11023 (accessed on 6 December 2019).
17. Alrawi, O.; Lever, C.; Antonakakis, M.; Monrose, F. Sok: Security evaluation of home-based iot deployments. In Proceedings of the IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 19–23 May 2019.
18. Xie, W.; Jiang, Y.; Tang, Y.; Ding, N.; Gao, Y. Vulnerability detection in iot firmware: A survey. In Proceedings of the IEEE 23rd International Conference on Parallel and Distributed Systems (ICPADS), Shenzhen, China, 15–17 December 2017.
19. Zheng, Y.; Wen, H.; Cheng, K.; Song, Z.W.; Zhu, H.S.; Sun, L.M. A Survey of IoT Device Vulnerability Mining Techniques. *J. Cyber Secur.* **2019**, *4*, 61–75, doi:10.19363/J.cnki.cn10-1380/tn.2019.09.06.
20. Samsung. Samsung SmartThings. Available online: https://www.smartthings.com/ (accessed on 6 December 2019).
21. Google. Google Weave Project. Available online: https://developers.google.com/weave/ (accessed on 6 December 2019).
22. Apple Inc. Apple HomeKit. Available online: http://www.apple.com/ios/home/ (accessed on 6 December 2019).
23. Home, A. Home Assistant. Available online: https://www.home-assistant.io (accessed on 6 December 2019).
24. Mi Inc. IoT Developer Platform. Available online: https://iot.mi.com/ (accessed on 6 December 2019).
25. WiFi, A. WiFi. Available online: https://www.wi-fi.org/ (accessed on 6 December 2019).
26. Zigbee, A. Zigbee. Available online: https://zigbee.org/ (accessed on 6 December 2019).
27. Bluetooth Technology Website. Available online: https://www.bluetooth.com/ (accessed on 6 December 2019).
28. Liu, X.; Zhou, Z.; Diao, W.; Li, Z.; hang, K. When good becomes evil: Keystroke inference with smartwatch. In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, 12–16 October 2015.
29. Das, A.; Borisov, N.; Caesar M. Do you hear what i hear?: Fingerprinting smart devices through embedded acoustic components. In Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, 3–7 November 2014.

30. Vasyltsov, I.; Lee, S. Entropy extraction from bio-signals in healthcare IoT. In Proceedings of the 1st ACM Workshop on IoT Privacy, Trust, and Security, Singapore, 14 April 2015.
31. McCann, D.; Eder, K.; Oswald, E. Characterising and comparing the energy consumption of side channel attack countermeasures and lightweight cryptography on embedded device. In Proceedings of the International Workshop on Secure Internet of Things (SIoT), Vienna, Austria, 21–25 September 2015.
32. Stokes, P., SentinelOne. Checkm8: 5 Things You Should Know about the New Ios Boot Rom Exploit. Available online: https://www.sentinelone.com/blog/checkm8-5-things-you-should-know-new-ios-boot-rom-exploit/ (accessed on 6 December 2019).
33. MITRE Corp. Marvell WiFi. Available online: https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=+Marvell+WiFi (accessed on 6 December 2019).
34. Paganini, P. Million of Telestar Digital GmbH IoT Radio Devices Can Be Remotely Hacked. Available online: https://securityaffairs.co/wordpress/91069/hacking/telestar-iot-radio-devices-hack.html (accessed on 6 December 2019).
35. MITRE Corp. CVE-2019-13473. Available online: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-13473 (accessed on 6 December 2019).
36. MITRE Corp. CVE-2019-13474. Available online: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-13474 (accessed on 6 December 2019).
37. Costa Gondim, J.J.; de Oliveira Albuquerque, R.; Clayton Alves Nascimento, A.; García Villalba, L.J.; Kim, T.H. A methodological approach for assessing amplified reflection distributed denial of service on the internet of things. *Sensors* **2016**, *16*, 1855, doi:10.3390/s16111855.
38. Wikipedia. Constrained Application Protocol. Available online: https://en.wikipedia.org/wiki/Constrained_Application_Protocol (accessed on 6 December 2019).
39. UPnP Corp. UPnP Device Architecture 1.0. Available online: http://www.upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.0-20080424.pdf (accessed on 6 December 2019).
40. Li, C.; Cai, Q.; Li, J.; Liu, H.; Zhang, Y.; Gu, D.; Yu, Y. Passwords in the Air: Harvesting Wi-Fi Credentials from SmartCfg Provisioning. In Proceedings of the 11th ACM Conference on Security & Privacy in Wireless and Mobile Networks, Stockholm, Sweden, 18–20 June 2018.
41. Zhou, W.; Jia, Y.; Yao, Y.; Zhu, L.; Guan, L.; Mao, Y.; Zhang, Y. Phantom Device Attack: Uncovering the Security Implications of the Interactions among Devices, IoT Cloud, and Mobile Apps. *arXiv* **2018**, arXiv:1811.03241.
42. Vasile, S.; Oswald, D.; Chothia, T. Breaking All the Things—A Systematic Survey of Firmware Extraction Techniques for IoT Devices. In Proceedings of the International Conference on Smart Card Research and Advanced Applications, Montpellier, France, 12–14 November 2018.
43. Zaddach, J.; Bruno, L.; Francillon, A.; Balzarotti, D. AVATAR: A Framework to Support Dynamic Security Analysis of Embedded Systems' Firmwares. In Proceedings of the Network and Distributed System Security (NDSS) Symposium, San Diego, CA, USA, 23–26 February 2014.
44. Kammerstetter, M.; Platzer, C.; Kastner, W. Prospect: peripheral proxying supported embedded code testing. In Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security, Kyoto, Japan, 3–6 June 2014.
45. Koscher, K.; Kohno, T.; Molnar, D. SURROGATES: Enabling Near-Real-Time Dynamic Analyses of Embedded Systems. In Proceedings of the 9th USENIX Workshop on Offensive Technologies (WOOT 15), Washington, DC, USA, 10–11 August 2015.
46. Muench, M.; Nisi, D.; Francillon, A.; Balzarotti, D. Avatar 2: A Multi-target Orchestration Platform. In Proceedings of the Workshop on Binary Analysis Research (colocated with NDSS Symposium), San Diego, CA, USA, 18 February 2018.
47. Chen, D.D.; Woo, M.; Brumley, D.; Egele, M. Towards Automated Dynamic Analysis for Linux-based Embedded Firmware. In Proceedings of the Network and Distributed System Security (NDSS) Symposium, San Diego, CA, USA, 21–24 February 2016.
48. Bellard, F. QEMU, a fast and portable dynamic translator. In Proceedings of the USENIX Annual Technical Conference, Anaheim, CA, USA, 10–15 April 2005.
49. Wikipedia. Fuzzing. Available online: https://en.wikipedia.org/wiki/Fuzzing (accessed on 6 December 2019).

50. Wikipedia. Taint Checking. Available online: https://en.wikipedia.org/wiki/Taint_checking (accessed on 6 December 2019).
51. King, J.C. Symbolic execution and program testing. *Commun. ACM* **1976**, *19*; 385–394.
52. Alimi, V.; Vernois, S.; Rosenberger, C. Analysis of embedded applications by evolutionary fuzzing. In Proceedings the 2014 International Conference on High Performance Computing & Simulation (HPCS), Bologna, Italy, 21–25 July 2014.
53. Kamel, N.; Lanet, J.L. Analysis of HTTP protocol implementation in smart card embedded web server. *Int. J. Inf. Netw. Security (IJINS)* **2013**, *2*, 417.
54. Koscher, K.; Czeskis, A.; Roesner, F.; Patel, S.; Kohno, T.; Checkoway, S.; McCoy, D.; Kantor, B.; Anderson, D; Shacham, H.; et al. Experimental security analysis of a modern automobile. In Proceedings of the IEEE Symposium on Security and Privacy (SP), Berkeley, CA, USA, 16–19 May 2010.
55. Lee, H.; Choi, K.; Chung, K.; Kim, J.; Yim, K. Fuzzing can packets into automobiles. In Proceedings of the 29th International Conference on Advanced Information Networking and Applications, Gwangiu, Korea, 24–27 March 2015.
56. Wikipedia. CAN bus. Available online: https://en.wikipedia.org/wiki/CAN_bus (accessed on 6 December 2019).
57. Chen, J.; Diao, W.; Zhao, Q.; Zuo, C.; Lin, Z.; Wang, X.; Lau, W.C.; Sun, M.; Yang, R.; Zhang, K. Iotfuzzer: Discovering Memory Corruptions in Iot through App-Based Fuzzing. In Proceedings of the Network and Distributed System Security (NDSS) Symposium, San Diego, CA, USA, 18–21 February 2018.
58. Costin, A.; Zarras, A.; Francillon, A. 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, Xi'an, China, 30 May–3 June 2016.
59. Srivastava, P.; Peng, H.; Li, J.; Okhravi, H.; Shrobe, H.; Payer, M. FirmFuzz: Automated IoT Firmware Introspection and Analysis. In Proceedings of the 2nd International ACM Workshop on Security and Privacy for the Internet-of-Things, London, UK, 15 November 2019.
60. Zheng, Y.; Davanian, A.; Yin, H.; Song, C.; Zhu, H.; Sun, L. FIRM-AFL: high-throughput greybox fuzzing of iot firmware via augmented process emulation. In Proceedings of the 28th USENIX Security Symposium (USENIX Security 19), Santa Clara, CA, USA, 14–16 August 2019.
61. Zalewski, M. American Fuzzy Lop. Available online: http://lcamtuf.coredump.cx/afl (accessed on 6 December 2019).
62. Muench, M.; Stijohann, J.; Kargl, F.; Francillon, A.; Balzarotti, D. What You Corrupt Is Not What You Crash: Challenges in Fuzzing Embedded Devices. In Proceedings of the Network and Distributed System Security (NDSS) Symposium, San Diego, CA, USA, 18–21 February 2018.
63. Dolan-Gavitt, B.; Hodosh, J.; Hulin, P.; Leek, T.; Whelan, R. Repeatable reverse engineering with PANDA. In Proceedings of the 5th Program Protection and Reverse Engineering Workshop, Los Angeles, CA, USA, 15 December 2015.
64. Costin, A.; Zaddach, J.; Francillon, A.; Balzarotti, D. A large-scale analysis of the security of embedded firmwares. In Proceedings of the 23rd USENIX Security Symposium (USENIX Security 14), San Diego, CA, USA, 20–22 August 2014.
65. Davidson, D.; Moench, B.; Ristenpart, T.; Jha, S. FIE on Firmware: Finding Vulnerabilities in Embedded Systems Using Symbolic Execution. In Proceedings of the 22nd USENIX Security Symposium (USENIX Security 13), Washington, DC, USA, 14–16 August 2013.
66. Celik, Z.B.; Babun, L.; Sikder, A.K.; Aksu, H.; Tan, G.; McDaniel, P.; Uluagac, A.S. KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs. In Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation(OSDI 2008), San Diego, CA, USA, 8–10 December 2008.
67. Shoshitaishvili, Y.; Wang, R.; Hauser, C.; Kruegel, C.; Vigna, G. Firmalice-Automatic Detection of Authentication Bypass Vulnerabilities in Binary Firmware. In Proceedings of the Network and Distributed System Security (NDSS) Symposium, San Diego, CA, USA, 8–11 February 2015.
68. Celik, Z.B.; Babun, L.; Sikder, A.K.; Aksu, H.; Tan, G.; McDaniel, P.; Uluagac, A.S. Sensitive information tracking in commodity IoT. In Proceedings of the 27th USENIX Security Symposium (USENIX Security 18), Baltimore, MD, USA, 15–17 August 2018.

69. Cheng, K.; Li, Q.; Wang, L.; Chen, Q.; Zheng, Y.; Sun, L.; Liang, Z. DTaint: detecting the taint-style vulnerability in embedded device firmware. In Proceedings of the 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Luxembourg, 25–28 June 2018.
70. Cui, A.; Stolfo, S.J. A quantitative analysis of the insecurity of embedded network devices: results of a wide-area scan. In Proceedings of the 26th Annual Computer Security Applications Conference, Austin, TX, USA, 6–10 December 2010.
71. Al-Alami, H;, Ali, H.; Hussein, A.B. Vulnerability scanning of IoT devices in Jordan using Shodan. In Proceedings of the 2nd International Conference on the Applications of Information Technology in Developing Renewable Energy Processes & Systems (IT-DREPS), Amman, Jordan, 6–7 December 2017.
72. Durumeric, Z.; Adrian, D.; Mirian, A.; Bailey, M.; Halderman, J.A. A search engine backed by Internet-wide scanning. In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, 12–16 October 2015.
73. Knownsec, Inc. Zoomeye. Available online: https://www.zoomeye.org/ (accessed on 6 December 2019).
74. Li, Z.; Lu, S.; Myagmar, S.; Zhou, Y. CP-Miner: A Tool for Finding Copy-paste and Related Bugs in Operating System Code. In Proceedings of the 6th Symposium on Operating System Design and Implementation (OSDI 2004), San Francisco, CA, USA, 6–8 December 2004.
75. Jang, J.; Agrawal, A.; Brumley, D. ReDeBug: finding unpatched code clones in entire os distributions. In Proceedings of the IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 20–23 May 2012.
76. Wikipedia. N-gram. Available online: https://en.wikipedia.org/wiki/N-gram (accessed on 6 December 2019).
77. Myles, G.; Christian, C. K-gram based software birthmarks. In Proceedings of the 2005 ACM Symposium on Applied Computing, Santa Fe, NM, USA, 13–17 March 2005.
78. Khoo, W.M.; Mycroft, A.; Anderson R. Rendezvous: A search engine for binary code. In Proceedings of the 10th Working Conference on Mining Software Repositories, San Francisco, CA, USA, 18–19 May 2013.
79. Karim, M.E.; Walenstein, A.; Lakhotia, A.; Parida, L. Malware phylogeny generation using permutations of code. *J. Comput. Virol.* **2005**, *1*, 13–23, doi:10.1007/s11416-005-0002-9.
80. David, Y.; Yahav, E. Tracelet-based code search in executables. *Acm Sigplan Notices* **2014**, *49*, 349–360, doi:10.1145/2666356.2594343.
81. Pewny, J.; Schuster, F.; Bernhard, L.; Holz, T.; Rossow, C. Leveraging semantic signatures for bug search in binary programs. In Proceedings of the 30th Annual Computer Security Applications Conference, New Orleans, LA, USA, 8–12 December 2014.
82. Allen, F.E. Control flow analysis. *ACM Sigplan Notices* **1970**, 55, 7, doi:10.1145/390013.808479.
83. Dullien, T.; Rolles, R. Graph-based comparison of executable objects. In Proceedings of the SSTIC'05, Rennes, France, 1–3 July 2005.
84. Bourquin, M.; King, A.; Robbins, E. Binslayer: accurate comparison of binary executables. In Proceedings of the 2nd ACM SIGPLAN Program Protection and Reverse Engineering Workshop, Rome, Italy, 26 January 2013.
85. Egele, M.; Woo, M.; Chapman, P.; Brumley, D. Blanket execution: Dynamic similarity testing for program binaries and components. In Proceedings of the 23rd USENIX Security Symposium (USENIX Security 14), San Diego, CA, USA, 20–22 August 2014.
86. Gao, D.; Reiter, M.K.;Song, D. Binhunt: Automatically finding semantic differences in binary programs. In Proceedings of the International Conference on Information and Communications Security, Birmingham, UK, 20–22 October 2008.
87. Ming, J.; Pan, M.; Gao, D. iBinHunt: Binary hunting with inter-procedural control flow. In Proceedings of the International Conference on Information Security and Cryptology, Seoul, Korea, 28–30 November 2012.
88. Pewny, J.; Garmany, B.; Gawlik, R.; Rossow, C.; Holz, T. Cross-architecture bug search in binary executables. In Proceedings of the IEEE Symposium on Security and Privacy (SP), San Jose, CA, USA, 17–21 May 2015.
89. Eschweiler, S.; Yakdan, K.; Gerhards-Padilla, E. discovRE: Efficient Cross-Architecture Identification of Bugs in Binary Code. In Proceedings of the Network and Distributed System Security (NDSS) Symposium, San Diego, CA, USA, 21–24 February 2016.

90. Chandramohan, M.; Xue, Y.; Xu, Z.; Liu, Y.; Cho, C.Y.; Tan, H.B.K. Bingo: Cross-architecture cross-os binary search. In Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, Seattle, WA, USA, 13–18 November 2016.
91. Feng, Q.; Zhou, R.; Xu, C.; Cheng, Y.; Testa, B.; Yin, H. Scalable graph-based bug search for firmware images. In Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, 24–28 October 2016.
92. Yan, S.; Xu, D.; Zhang, B.; Zhang, H.J.; Yang, Q.; Lin, S. Graph embedding and extensions: A general framework for dimensionality reduction. *IEEE Transact. Pattern Anal. Mach. Intell.* **2007**, *29*, 40–51, doi:10.1109/TPAMI.2007.250598.
93. Xu, X.; Liu, C.; Feng, Q.; Yin, H.; Song, L.; Song, D. Neural network-based graph embedding for cross-platform binary code similarity detection. In Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, Dallas, TX, USA, 30 October–3 November 2017.
94. Liu, B.; Huo, W.; Zhang, C.; Li, W.; Li, F.; Piao, A.; Zou, W. αDiff: cross-version binary code similarity detection with DNN. In Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, Montpellier, France, 3–7 September 2018.
95. Gao, J.; Yang, X.; Fu, Y.; Jiang, Y.; Sun, J. Vulseeker: a semantic learning based vulnerability seeker for cross-platform binary. In Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, Montpellier, France, 3–7 September 2018.
96. Gao, J.; Yang, X.; Fu, Y.; Jiang, Y.; Shi, H.; Sun, J. Vulseeker-pro: enhanced semantic learning based binary vulnerability seeker with emulation. In Proceedings of the 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Tallinn, Estonia, 26–30 August 2019.
97. Long, F.; Rinard, M. Prophet: Automatic Patch Generation via Learning from Successful Patches. https://core.ac.uk/download/pdf/78062945.pdf (accessed on 6 December 2019).
98. Long, F.; Rinard, M. Automatic patch generation by learning correct code. In Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, St. Petersburg, FL, USA, 20–22 January 2016.
99. Long, F.; Amidon, P.; Rinard, M. Automatic inference of code transforms for patch generation. In Proceedings of the 11th Joint Meeting on Foundations of Software Engineering, Paderborn, Germany, 4–8 September 2017.
100. Le Goues, C.; Nguyen, T.; Forrest, S.; Weimer, W. Genprog: A generic method for automatic software repair. *IEEE Trans. Soft. Eng.* **2011**, *38*, 54–72, doi:10.1109/TSE.2011.104.
101. Kim, D.; Nam, J.; Song, J.; Kim, S. Automatic patch generation learned from human-written patches. In Proceedings of the International Conference on Software Engineering, San Francisco, CA, USA, 18–26 May 2013.
102. Zhang, Y.; Chen, Y.; Bao, C.; Xia, L.; Zhen, L.; Lu, Y.; Wei, T. Adaptive kernel live patching: An open collaborative effort to ameliorate android n-day root exploits. In Proceedings of Black Hat USA, Las Vegas, NA, USA, 30 July–4 August 2016.
103. DARPA. Cyber Grand Challenge. Available online: https://www.darpa.mil/program/cyber-grand-challenge (accessed on 6 December 2019).
104. Shoshitaishvili, Y.; Bianchi, A.; Borgolte, K.; Cama, A.; Corbetta, J.; Disperati, F.; Dutcher, A.; Grosen, J.; Grosen, P.;Machiry, A.; etc. Mechanical phish: Resilient autonomous hacking. *IEEE Secur. Priv.* **2018**, *16*, 12–22, doi:10.1109/MSP.2018.1870858.
105. Shoshitaishvili, Y.; Weissbacher, M.; Dresel, L.; Salls, C.; Wang, R.; Kruegel, C.; Vigna, G. Rise of the hacrs: Augmenting autonomous cyber reasoning systems with human assistance. In Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, Dallas, TX, USA, 30 October–3 November 2017.
106. Nguyen-Tuong, A.; Melski, D.; Davidson, J.W.; Co, M.; Hawkins, W.; Hiser, J.D.;Morris, D.; Nguyen, D.; Rizzi, E. Xandra: An Autonomous Cyber Battle System for the Cyber Grand Challenge. *IEEE Secur. Priv.* **2018**, *16*, 42–51, doi:10.1109/MSP.2018.1870876.
107. Fernandes, E.; Jung, J.; Prakash, A. Security analysis of emerging smart home applications. In Proceedings of the IEEE Symposium on Security and Privacy (SP), San Jose, CA, USA, 22–26 May 2016.

108. Fernandes, E.; Paupore, J.; Rahmati, A.; Simionato, D.; Conti, M.; Prakash, A. Flowfence: Practical data protection for emerging iot application frameworks. In Proceedings of the 25th USENIX Security Symposium (USENIX Security 16), Austin, TX, USA, 10–12 August 2016.
109. Jia, Y.J.; Chen, Q.A.; Wang, S.; Rahmati, A.; Fernandes, E.; Mao, Z.M.; Prakash, A. ContexIoT: Towards Providing Contextual Integrity to Appified IoT Platforms. In Proceedings of the Network and Distributed System Security (NDSS) Symposium, San Diego, CA, USA, 26 February–1 March 2017.
110. Rahmati, A.; Fernandes, E.; Eykholt, K.; Prakash, A. Tyche: A risk-based permission model for smart homes. In Proceedings of the IEEE Cybersecurity Development (SecDev), Cambridge, MA, USA, 30 September–2 October 2018.
111. Tian, Y.; Zhang, N.; Lin, Y.H.; Wang, X.; Ur, B.; Guo, X.; Tague, P. Smartauth: User-centered authorization for the internet of things. In proceedings of the 26th USENIX Security Symposium (USENIX Security 17), Vancouver, BC, Canada, 16–18 August 2017.
112. Lee, S.; Choi, J.; Kim, J.; Cho, B.; Lee, S.; Kim, H.; Kim, J. FACT: Functionality-centric access control system for IoT programming frameworks. In Proceedings of the 22nd ACM on Symposium on Access Control Models and Technologies, Indianapolis, IN, USA, 21–23 June 2017.
113. Cui, A.; Costello, M.; Stolfo, S. When firmware modifications attack: A case study of embedded exploitation. In Proceedings of the Network and Distributed System Security (NDSS) Symposium, San Diego, CA, USA, 24–27 February 2013.
114. Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; Bengio, Y. Generative adversarial nets. In Proceedings of the Advances in Neural Information Processing Systems 27 (NIPS 2014), Montreal, QC, Canada, 8–13 December 2014.
115. Hiromoto, R.E.; Haney, M.; Vakanski, A. A secure architecture for IoT with supply chain risk management. In Proceedings of the 9th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), Bucharest, Romania, 21–23 September 2017.