Hindawi
Security and Communication Networks
Volume 2022, Article ID 1443978, 17 pages
https://doi.org/10.1155/2022/1443978

Research Article

IoT-DeepSense: Behavioral Security Detection of IoT Devices Based on Firmware Virtualization and Deep Learning

Jin Wang,[1] **Chang Liu**,[1] **Jiangpei Xu**,[1] **Juan Wang**,[2] **Shirong Hao**,[2] **Wenzhe Yi**,[2] **and Jing Zhong**[2]

[1]*Electric Power Research Institute of State Grid Hubei Electric Power Company, Wuhan 430072, Hubei, China*
[2]*School of Cyber Science and Engineering, Wuhan University, Wuhan, Hubei, China*

Correspondence should be addressed to Juan Wang; jwang@whu.edu.cn

Received 15 October 2021; Revised 30 January 2022; Accepted 4 February 2022; Published 18 March 2022

Academic Editor: Weizhi Meng

Recently, IoT devices have become the targets of large-scale cyberattacks, and their security issues have been increasingly serious. However, due to the limited memory and battery power of IoT devices, it is hardly possible to install traditional security software, such as antivirus software for security defense. Meanwhile, network-based traffic detection is difficult to obtain the internal behavior states and conduct in-depth security analysis because more and more IoT devices use encrypted traffic. Therefore, how to obtain complex security behaviors and states inside IoT devices and perform security detection and defense is an issue that needs to be solved urgently. Aiming at this issue, we propose IoT-DeepSense, a behavioral security detection system of IoT devices based on firmware virtualization and deep learning. IoT-DeepSense constructs the real operating environment of the IoT device system to capture the fine-grained system behaviors and then leverages an LSTM-based IoT system behavior abnormality detection approach to effectively extract the hidden features of the system's behavior sequence and enforce the security detection of the abnormal behavior of the IoT devices. The design and implementation of IoT-DeepSense are carried out on an independent Internet of things behavior detection server, without modifying the limited resources of IoT devices, and have strong scalability. The evaluation results show that IoT-DeepSense achieves a high behavioral detection rate of 92%, with negligible impact on the performance of IoT devices.

1. Introduction

In recent years, the Internet of things (IoT) has developed rapidly and has been continuously expanded and applied in the fields of smart homes, smart cities, industrial systems, and smart medical products [1]. Statistics show that the number of IoT devices will reach about 50 billion [2] by 2020. These interconnected IoT devices also bring a lot of smart applications, services, and data, resulting in more emerging data-centric businesses [3].

With the rapid growth of the Internet of things applications and devices, attacks against the Internet of Things are becoming more and more serious. For example, remote attackers attack and damage patients' implantable medical devices [4], which may not only cause huge economic losses to individuals but also endanger the safety of patients. Also, with the widespread use of IoT devices in other key areas, attackers may jeopardize public network security. For example, in 2016, a distributed denial-of-service (DDoS) [5] attack against domain name system provider Dyn Company resulted in the inaccessibility of multiple websites such as GitHub and Twitter. This attack is performed through a botnet composed of a large number of IoT devices, including IP cameras, gateways, and even baby monitors. Furthermore, in April 2021, the Mozi botnet that targets IoT devices had controlled approximately 438,000 nodes for DDoS attacks, data exfiltration, and remote command execution. Each compromised node in the botnet is instructed to find a new victim IoT device for infection [6]. Therefore, it is foreseeable that the botnets will continue to expand rapidly and cause more serious damage.

However, most enterprises and users lack privacy and security awareness, and only focus on the realization of the core functions of the product, while ignoring potential

security issues. Unless the user initiates a firmware update, IoT device vendors usually do not update and patch their device security vulnerabilities. At the same time, due to limited power consumption and resources, IoT devices are usually unable to install and run traditional security software, such as antivirus software and IDS, resulting in vulnerabilities in IoT devices (e.g., default passwords and unpatched errors) that cannot be eliminated in a long time [7]. Therefore, under the situation where the number of IoT devices is increasing rapidly and security problems are constantly emerging, it is of great significance how to conduct security detection on the behavior of the IoT devices to realize the IoT security defense.

At present, the research on the behavior security of IoT devices mainly focuses on behavior detection schemes based on network traffic. Network traffic-based detection can only detect the security issues of IoT devices from the network layer. It is difficult to obtain the internal behavior status of the system and conduct an in-depth security analysis. Besides, more and more IoT devices currently use encrypted traffic, which also makes traffic-based security detection more difficult.

The firmware virtualization technology simulates the operating environment of the embedded system based on the firmware image, realizes large-scale and automated dynamic analysis of the embedded firmware binary file, and then mines firmware vulnerabilities to implement the embedded firmware security analysis. The firmware virtualization technology can simulate the real operating environment of the firmware, which helps to obtain the operating data and security status of the internal system layer of the device and thus realize the behavior detection and defense at the system level of the IoT device. Therefore, we propose a method to perform system behavior detection on IoT devices using firmware virtualization and combining it with deep learning. Given the difficulty in obtaining complex security behaviors and states inside IoT devices, the system operating environment of IoT devices is simulated based on firmware virtualization technology and then the complex security behaviors and states inside IoT devices are collected. Aiming at the problem that the internal behavior information structure of IoT devices is complex and difficult to detect, a deep learning-based IoT behavior security detection scheme is proposed. In response to the demand for risk mitigation of abnormal behaviors of the Internet of things, the attack stage and security risks are analyzed and determined, and the risk mitigation policies are presented.

In particular, our main contributions can be summarized as follows:

(i) We propose a fine-grained dynamic system behavior capture mechanism to collect the complex security behaviors and states inside the IoT devices in real time.

(ii) We propose an LSTM-based IoT system behavior abnormality detection model. The abnormal behavior detection model can effectively extract the hidden features of the system's behavior sequence and well express the internal dependencies, to successfully implement the security detection of the abnormal behavior of the IoT devices.

(iii) We design and implement IoT-DeepSense, an IoT device behavior security detection system that combines firmware virtualization and deep learning. The evaluation results show that the system can achieve a high behavioral detection rate of 92%, with negligible impact on the performance of IoT devices.

The rest of the study is organized as follows. Section 2 introduces the background, and Section 3 describes our system overview and the design details. The implementation and evaluation of our system are described in Section 4. We present related work in Section 5. Finally, we discuss limitations and future work in Section 6 and conclude this study in Section 7.

2. Background

In this section, we describe the background of firmware virtualization and recurrent neural networks.

2.1. Firmware Virtualization. Recently, IoT device risk vulnerabilities are increasing day by day, often with very serious security consequences. However, due to the limited resources of IoT devices, it is a problem to install security monitoring software on IoT devices to obtain the security states of IoT physical devices and defend them. Network function virtualization (NFV) can detect and defend the security states of IoT devices by constructing virtual security functions. However, network security functions such as virtual IDS and virtual firewall only obtain limited and simple device security status by analyzing network traffic. The firmware virtualization technology can build the operating environment of the IoT device system based on the IoT firmware virtualization, to obtain the complex and comprehensive internal behaviors and states of the IoT device.

Since most IoT devices are not X86/X64 architectures, they cannot be emulated using VMware-like software like standard desktop or server operating systems. However, mainstream virtualization software (e.g., QEMU) can already support firmware virtualization of IoT devices with MIPS or ARM architecture, so that IoT device firmware (including operating systems and applications) can run in a virtualized environment.

QEMU [8] is a fast processor emulator based on dynamic binary conversion. Unlike traditional simulators that interpret the target program by instruction, QEMU converts multiple basic blocks at once. More importantly, QEMU caches the converted blocks and uses block linking to link them together. This keeps the executive program in the code cache most of the time, thereby minimizing the overhead of conversion.

For firmware emulation, in addition to instruction conversion, address space conversion is also very important. QEMU has two execution modes: system mode and user mode. The implementation of address space conversion is different in different execution modes. In system mode,

QEMU uses a software memory management unit (MMU) to handle memory access. The software MMU maps the client's memory address (GVA) to the host memory address (HVA). This mapping process is transparent to the guest operating system, so the guest operating system can set the GVA to client physical address (GPA) mapping and handle page faults through the page table interface. QEMU adds GVA-to-GPA conversion logic for each memory access. To speed up the conversion, QEMU uses an address translation cache (TLB) to cache the conversion result. Moreover, to avoid invalid code cache and block linking whenever the address translation changes, all converted blocks are indexed using GPA, and block linking is only performed when two basic blocks are within the same physical page. The mapping from GPA to HVA is done using linear mapping (HVA = GPA + offset). Contrary to system model simulation, in user mode simulation, the host virtual address (HVA) is equal to the client virtual address (GVA) plus a constant offset. Therefore, the conversion in user mode simulation is much faster than the conversion in system mode simulation.

There are already several frameworks that support the use of QEMU to implement an IoT virtualized environment. Avatar achieves this goal by building a hybrid execution environment that includes a processor emulator (QEMU) and actual hardware. Avatar uses an emulator to execute and analyze instructions while passing I/O operations to physical hardware, but Avatar must obtain the physical hardware of the device under test and must manually identify and interact with the debug port on the device interface, reducing the practicability of this architecture. FIRMADYNE added hardware support for IoT firmware in the system mode QEMU. FIRMADYNE supports the popular ARM and MIPS architectures among IoT manufacturers. To obtain hardware support, FIRMADYNE implements a complete simulation system by modifying the kernel and drivers to handle the abnormality of the Internet of things due to a lack of actual hardware. Due to FIRMADYNE's full-system emulation, this study chooses to use FIRMADYNE to virtualize IoT devices with firmware to obtain the real operating environment of IoT devices.

2.2. RNN. Artificial neural network (ANN) [9] was inspired by the biological learning system and loosely modeled its basic functions. The biological learning system is a complex network of interconnected neurons [10]. The most common type of standard neural network is a feedforward neural network. Here, the collection of neurons is organized hierarchically: an input layer, an output layer, and at least one intermediate hidden layer. Feedforward neural networks are limited to static classification tasks. Therefore, they are limited to providing a static mapping between input and output.

To model the time prediction task, we need a dynamic classifier. We need to extend the feedforward neural network to dynamic classification. To obtain this property, we need feedback on the signal of the previous time steps back to the network. These networks with recursive connections for processing sequence data are called recurrent neural networks (RNNs) [11]. The typical structure of RNN includes three layers, namely the input layer, the output layer, and the hidden layer. Because the gradient disappears or the gradient explodes, the RNN is limited to learning about ten-time steps. When we deal with short-term dependencies, recurrent neural networks can work well.

Long short-term memory recurrent neural network (LSTM-RNN) [12] solves the problem of gradient disappearance and gradient explosion during long sequence training. The LSTM network is biologically reasonable to a certain extent and can learn more than 1,000-time steps, depending on the complexity of the network being constructed. In short, LSTM can perform better than ordinary RNN in longer sequences. LSTM has achieved considerable success and has been widely used in many problems, such as natural language processing and speech recognition.

The basic LSTM neural network is composed of an input layer, a hidden LSTM layer, and an output layer. However, this architecture can be extended to deeper LSTMs, where multiple hidden LSTM layers are stacked on top of each other [13]. This is done by taking the output of each LSTM cell and using them as the input of the cell at the corresponding location in the next LSTM layer.

The LSTM [14] network uses a memory unit to replace the basic unit in the hidden layer of the RNN. There are three gates in the memory unit of the LSTM network, including the input gate, the output gate, and the forget gate. Since IoT device behavior information is collected and recorded in chronological order and has temporal characteristics, LSTM can be applied to IoT device behavior analysis scenarios.

3. System Design

3.1. System Overview. Due to the limited computing and storage resources of IoT devices, it is difficult to install monitoring software on real IoT devices to obtain the internal behavior state of the device, thereby performing security state detection and defense. To solve this problem, we have proposed IoT-DeepSense, a behavioral security detection architecture and system for IoT devices based on firmware virtualization and deep learning. Using the firmware virtualization technology, we simulate the real operating environment for each real Internet of things device and then collect the complex security behaviors inside the Internet of Things devices and generate device behavior logs. Through abnormal detection and analysis of the device behavior logs, we can realize the behavioral security detection and defense of Internet of Things devices.

The overall architecture of IoT-DeepSense is shown in Figure 1. The architecture includes a device behavior collection module, an abnormal behavior detection module, and an abnormal behavior analysis and risk mitigation module. The device behavior collection module simulates the real operating environment of the IoT device based on the firmware virtualization technology, then collects the fine-grained complex security behaviors inside the IoT device, and generates device behavior logs. The abnormal behavior detection module performs abnormal behavior

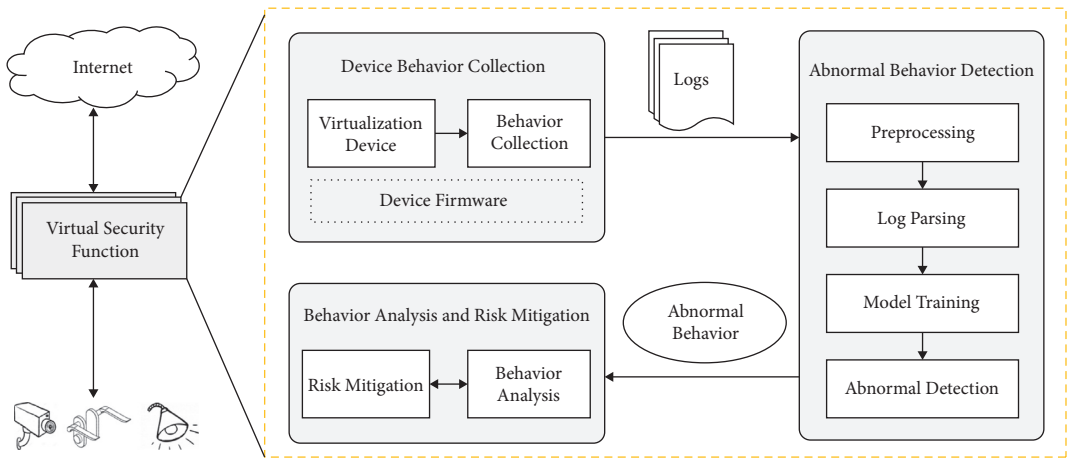

Figure 1: System architecture of IoT-DeepSense.

detection on the device behavior log based on deep learning technology and sends the detected abnormal behavior to the behavior analysis and risk mitigation module. The behavior analysis and risk mitigation module analyze the abnormal behavior and propose risk mitigation policies.

3.1.1. Device Behavior Collection Module. This module contains two sub-modules, the virtualization device module and the behavior collection module. (1) Virtualization device module: the virtualization device module uses HTTP or FTP to download the firmware image of the IoT device from the IoT device supplier support website and then uses the firmware virtualization technology to generate the IoT virtual device based on the firmware image, thereby simulating the real operating environment of IoT devices. (2) Behavior collection module: the behavior collection module obtains the complex security behavior and states inside the IoT virtual device in real time through the device log behavior collection script, including a timestamp, process number, process name, system call name, and system call input value and return value (if any), environment variables, and other information. Then, the module records the system behavior in the device behavior log file and sends the device behavior log to the abnormal behavior detection module.

3.1.2. Abnormal Behavior Detection Module. This module performs abnormal behavior detection on device behavior logs based on deep learning. This module includes preprocessing module, log parsing module, model training module, and abnormal detection module. (1) Preprocessing module: the preprocessing module is responsible for data cleaning and the deduplication operations of device behavior logs. (2) Log parsing module: the log parsing module is responsible for converting unstructured device behavior logs into structured device behavior logs, then further extracting meaningful information, and finally generating system behavior event sequences. (3) Model training module: the model training models the system behavior event sequence based on the long- and short-term memory (LSTM) model. (4) Abnormal detection module: for the newly collected unknown system behavior, it will be converted into a sequence of system behavior events through a preprocessing and log parsing module and then predicted using a deep learning model.

3.1.3. Behavior Analysis and Risk Mitigation Module. The module analyzes abnormal behavior to obtain fine-grained abnormal behavior information, such as abnormal file operations and abnormal executable files. The risk mitigation module implements risk mitigation policies including device isolation, traffic filtering, and user notification, which effectively mitigate the security risks brought by abnormal behaviors.

3.2. Dynamic Behavior Collection. Based on firmware virtualization, we have designed a scheme for capturing the dynamic behavior of IoT devices as shown in Figure 2. The purpose of the solution is to collect the system behavior generated during the operation of the IoT system, and hence, we virtualize an IoT virtual device for each IoT real device and simulate the real operating environment of the IoT device, and then, we track and record the security behavior and states generated in the simulated IoT operating environment. The collected behavior log file contains the system behavior in chronological order: each line includes a timestamp, process number, process name, system call name, system call input value and return value (if any), and other information such as environment variables. The chronologically recorded system behavior information allows us to construct various feature vectors to characterize the behavior of IoT devices with different accuracy.

3.2.1. Virtualization Device Based on Firmware. The system virtualizes an IoT virtual device for each IoT real device and then simulates the real operating environment of the IoT device. We collect complex system behaviors and states inside IoT devices in the real operating environment of the Internet of things and generate system behavior logs from the system behaviors and states. The virtualization device is

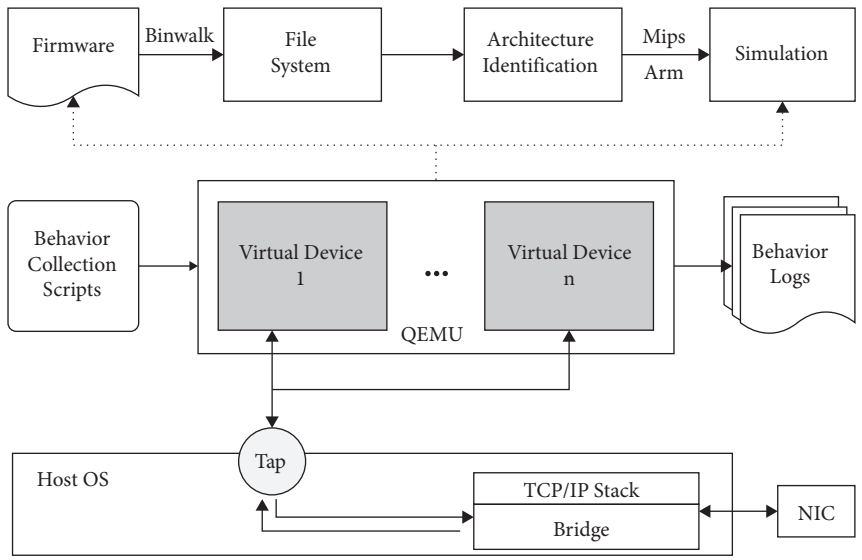

Figure 2: Dynamic behavior capture of IoT devices.

based on FIRMADYNE. FIRMADYNE is an automated dynamic analysis system for Linux-based IoT firmware. In QEMU system mode, hardware support for IoT firmware is added to implement full-system simulation to provide support for popular ARM and MIPS architectures in IoT device firmware. The process of virtualized equipment is mainly divided into four steps.

(1) Download the firmware image. A Web crawler is used to download the firmware image of the target device from the IoT provider website. For dynamic websites that are difficult to automatically crawl, the vendor's FTP website is used. For the collected files, we use Binwalk to check the format of non-firmware files for file filtering. Binwalk is a well-known firmware decompression tool that can extract various data from binary through pattern matching.

(2) Extract the file system. A custom file extraction program is built based on the API of the firmware extraction tool Binwalk, which is used to extract the kernel and root file system. Then, the extracted file system and kernel are compressed into TAR packages and stored for standardization and normalization operations.

(3) Device architecture identification. After extracting the root file system from the firmware image, the system recognizes the architecture and endianness of the target device. Currently, firmware virtualization supports emulating ARM little-endian platforms, MIPS big-endian platforms, and little-endian platforms. For each firmware image, the QEMU system emulator uses the modified kernel that matches its corresponding architecture to guide the file system extracted in the second step.

(4) Simulation. In the simulation phase, the modified kernel tracks and records system calls to the file system, network, and other related kernel subsystems to infer the device system and network configuration. After collecting the information, this information is fed back to our simulation framework to develop a more accurate QEMU configuration for the system. Finally, the system configures the matching network environment to communicate with the simulation firmware.

When configuring the IoT virtual device network, the network TAP device on the host is first instantiated, which is associated with one of the analog network interfaces (such as eth0) in the firmware. For firmware mirroring that uses VLANs, we assign the corresponding VLAN ID to the TAP interface to successfully communicate with the simulated network service. Next, the TAP interface is configured with an IP address that is on the same subnet as the IP address assigned by the firmware to the emulated interface.

This network configuration limits the network functions of IoT virtual devices and prevents them from accessing external networks. Although it can successfully implement automatic dynamic analysis and vulnerability detection of embedded firmware, it cannot make the IoT virtual device perform monitoring tasks. To solve this problem, we add bridges and routes to the IoT virtual device network configuration, so that the IoT virtual device can access the external network. First, a virtual bridge is instantiated on the host. An interface of the virtual bridge is connected to the host's network card (ens33). The host network card serves as one end of the bridge and connects to the external network. The other interface of the bridge is connected to the TAP device. The TAP device serves as the interface at the other end of the bridge and connects to the network interface of the virtual device of the Internet of things, thereby realizing the connection of the two networks at both ends of the bridge. The IP address of the IoT virtual device is modified to correspond to the IP of the real IoT device, and a default route is added.

3.2.2. System Behavior Collection. Virtual IoT devices based on firmware virtualization simulation are all based on Linux.

The system call is an interface provided by the Linux kernel. In IoT devices, the function of system calls mainly includes network communication, creation of new processes, I/O operations, and file operations. Malicious IoT devices and trusted IoT devices have different system behaviors, such as malicious devices requesting more permission to frequently access sensitive resources or frequent I/O operations, resulting in different system call frequencies. In addition, because the system behavior of an IoT device usually involves multiple system calls in sequence, a single system call cannot be considered independent. The system call sequence reveals the dynamic behavior of the application, and different system call sequences reflect different behaviors. Therefore, the process information and the frequency, sequence, and parameters of system calls during the operation of the IoT virtual device can all characterize its dynamic behavior. The following introduces the four types of dynamic characteristics of process information and system call frequency, sequence, and parameters in detail.

(1) Process Information. When the Internet of things device is running, the process information inside the system will reflect the system-related behavior information. For example, when the IoT device becomes the Bot host controlled by the Mirai botnet, the IoT device will have an additional malicious process during its operation (e.g., mirai.mips, where mips indicates that the IoT device is based on a MIPS architecture). The process communicates with the C&C server and operates according to malicious commands issued by the server. Therefore, whether the IoT device is attacked by malware can be detected through the process information. This shows that process information can be used as an important feature of behavior detection.

(2) System Call Frequency. The system call frequency represents the number of occurrences of each type of system call when the Internet of things device runs within a specific time. The frequency of system calls that occur during the execution of the Internet of things system carries information about related behaviors. When an IoT device is maliciously attacked, the device may use some specific system calls more frequently than normal. For example, under DDoS attacks, system calls related to I/O operations made by IoT devices may be much more frequent than normal. This indicates that the increased frequency of system calls may be a sign of malicious behavior.

(3) System Call Sequence. The system call sequence describes the local time relationship between system calls within a limited time range. System calls are fine-grained operating system information. The system call sequence reflects the execution path of the Internet of things device during operation. Different system call sequences reflect different behaviors. Because the functions of IoT devices are relatively fixed and single, the system call sequence has a certain regularity. For example, Table 1 shows the system call sequence information generated by the execution flow of motion [15]. Motion is a highly configurable program for monitoring video signals from multiple types of cameras. The main loop consists of a series of blocks. Each cycle begins with the camera capturing image frames. When motion is detected, the current motion frame is saved to the file system. Then, the application also independently saves snapshots at regular intervals (e.g., every 5 seconds). The two modules, save motion frame and save snapshot, use the same program flow to save the image to a file, thereby generating the same system call sequence as shown in the table.

After that, some predefined functions (e.g., waiting for the camera client to connect) may trigger external programs. This main loop repeats at the specified frame rate (e.g., 3 frames per second). Depending on the function, some blocks may not be executed in every cycle. When the system is maliciously attacked, it will destroy the original regular system call sequence. Therefore, the system call sequence can be used to extract features for malicious behavior detection.

(4) System Call Parameters. Some abnormal behaviors of IoT devices are not directly reflected in the execution path of the device, but in the form of abnormal parameters. For example, the attacker forged a small piece of code to make the system call sequence the same as the normal system call sequence [16], but by modifying the file path parameters, the device information was leaked. As shown in Table 2, the current motion frame is leaked to the desired location in the file system, while the generated system call sequence still looks legitimate. We only modify the location of the saved file (FilePath = "/path/to/Ideal location for attackers"), which makes the motion frame data leaked. Since the code blocks we added to use the same program routines as the "Save Motion Frame" and "Save Snapshot" code blocks, the system call sequence patterns generated by these three code blocks are the same. At the same time, the cross-block conversion does not generate a new pattern. If only one legal block is executed (save motion frames), the resulting sequence is still legal, because the inserted block (leaked motion frame) looks like another unexecuted block (save snapshot). The only way the system call sequence can detect such malicious execution is to learn the time relationship between two legal blocks through a pattern long enough. However, since the image size may vary greatly, this detection method is highly unlikely. Therefore, we need to obtain the specific location of the read-write file and the file name and other information from the system call parameters, to accurately detect abnormal behavior of the Internet of things.

Besides the above four types of dynamic characteristics including process information and system call frequency, sequence, and parameters, other hidden behavior features of IoT systems need to be mined to achieve more accurate IoT behavior security detection. The method for obtaining fine-grained system behavior information executed by the IoT virtual device in firmware is as follows.

When IoT virtual devices use QEMU for software simulation, to realize full software simulation, a custom pre-built kernel for ARM and MIPS architecture is used to replace the kernel extracted from the firmware image. During the custom pre-built kernel compilation process, an analysis module was added to the custom Linux kernel

Table 1: System call sequence.

Time	Function	System call sequence
T1	Get time	gettimeofday–gettimeofday
T2	Get frame	(ioctl)–rt sigprocmask–ioctl–ioctl–rt sigprocmask
T3	Get sport frame	open–fstat64–mmap2–write– ... –write–close–munmap–clone–write
T4	Save snapshot	open–fstat64–mmap2–write– ... –write–close–munmap–clone–write unlink–symlink
T5	Wait for connection	Select (accept–ioctl–write)–(write–munmap–close)–(mmap2–gettimeofday)
T6	Frame rate control	gettimeofday–(nanosleep)

Table 2: Image leakage under the same system call sequence.

Time	Function	System call sequence
T1	Save sport frame	open–fstat64–mmap2–write– ... –write–close–munmap–clone–write
T2	Leak sport frame	open–fstat64–mmap2–write– ... –write–close–munmap–clone–write
T3	Save snapshot	open–fstat64–mmap2–write– ... –write–close–munmap–clone–write

module, which uses the kernel dynamic probe (kprobes) framework to track multiple system calls, thereby helping to debug and simulate the original IoT system environment. Operations such as assigning MAC addresses, creating bridges, restarting the system, and executing programs are all monitored by the firmware virtualization framework to properly configure the simulated network environment.

Since the modified kernel can intercept system calls to the file system, network, and other related kernel subsystems, we can obtain the system call information of all processes executed in the firmware by setting the relevant mask of the kernel system call parameters.

By setting the corresponding bits for these system calls, each time the system is called, the system records the information about the system calls in the system behavior log. We can set the corresponding bit in the firmadyne.syscall parameter at startup to transfer information so that the IoT virtual device outputs the system behavior during the system operation and records it in the system behavior log file. The IoT dynamic behavior capture solution saves the collected IoT system behavior records into behavior log files, which contain chronological system behavior. Each line entry is a system behavior, including a timestamp, process number, process name, system call name, system call input, return value (if any), environment variables, and other information. Recording system behavior information in chronological order enables us to construct various complex features, characterize various behaviors of IoT devices with different precisions, and provide a basis for implementing abnormal behavior detection.

3.3. Abnormal Behavior Detection. The fine-grained behavior logs of IoT devices record the system behavior of IoT devices when they are running. Therefore, behavior logs are one of the most valuable data sources for anomaly detection. In addition, due to the complexity and a huge number of behavior logs, manual detection of abnormal behavior becomes infeasible. The keyword matching method based on explicit keywords (e.g., "Error") and the regular expression method based on structural features can only detect a single abnormal behavior log, and it is difficult to detect most behavior log anomalies. These anomalies can only be inferred based on their behavior log sequence, which contains multiple behavior logs that violate conventional rules. Therefore, we need an automatic anomaly detection method based on the system behavior log sequence.

As shown in Figure 3, we propose a deep learning-based abnormal behavior detection scheme for real-time abnormal detection of the behavior logs of the IoT system in this study. The system behavior collected during the normal operation of the Internet of things system has a certain periodicity and stability. The solution is based on the system behavior logs collected by the dynamic behavior capture module during the normal operation of the IoT device, and the deep learning method is used to learn the effective behavior characteristics of the IoT system from a large number of fine-grained system behaviors, thereby realizing the abnormal behavior of the IoT real-time detection without any modification to the existing infrastructure.

The abnormal behavior detection scheme based on deep learning mainly includes two stages, the training stage and the detection stage. During the training phase, the system preprocesses the IoT behavior log files to obtain the normal system behavior execution flow. Then, through the log parsing module, the unstructured behavior log is parsed into a sequence of system behavior events. The system behavior event sequence is used as the input of the abnormal behavior detection model, which learns the complex features of the system behavior event sequence and then constructs the abnormal behavior detection model. In the real-time detection phase, the system generates an IoT system behavior log at that moment and then generates a new system behavior sequence through preprocessing. Through log analysis, the system behavior sequence is parsed into a system behavior event sequence. Finally, the trained abnormal behavior detection model is used to detect whether the system behavior is normal. If it is detected as abnormal behavior, the system behavior will be sent to the abnormal behavior analysis module for further analysis.

3.3.1. Preprocessing and Log Parsing. To collect the data set, we simulated the normal usage of IoT devices. Our data

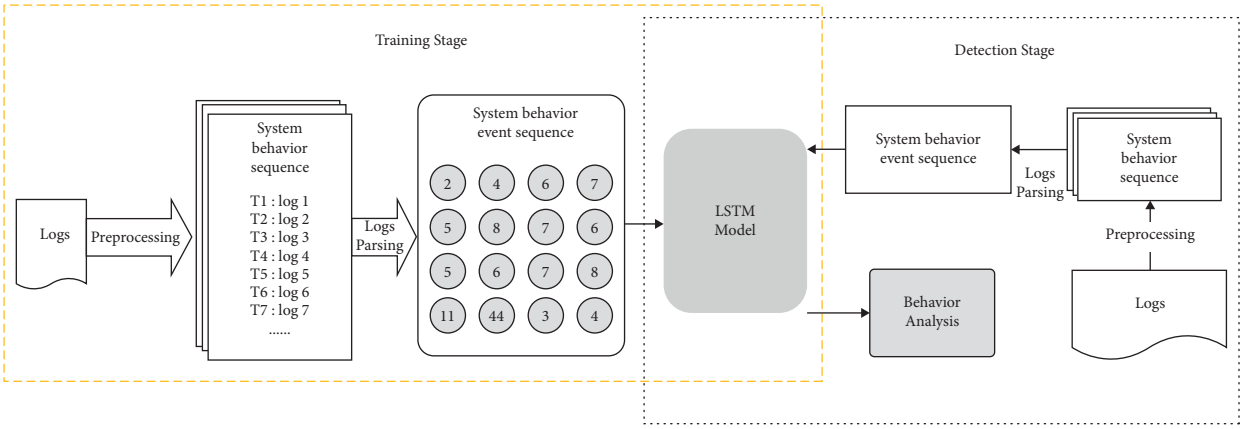

Figure 3: Anomaly detection scheme based on deep learning.

collection method is as follows: firstly, the IoT device is started and allowed to perform any initial configuration or firmware update. Secondly, when the IoT device is in a stable state, we interact with the device through smart applications. We also provide some idle time for the device to communicate without user intervention. According to the activity of the device, we captured tens of thousands of system behaviors from each device and recorded them in the system behavior log file.

For the characteristics of the system behavior log collected based on firmware virtualization in this study, we preprepared it from the following aspects to improve the efficiency of log analysis.

(1) System behavior cleaning. In the system behavior log, many system behaviors only contain constant strings, and there are no internal parameters. The repeated occurrence of these system behaviors will result in a large number of duplicate messages in the system behavior log. Many system behavior log messages will appear repeatedly at the same time. We delete these types of system behaviors to reduce data redundancy, thereby greatly improving the efficiency of subsequent log parsing.

(2) Delete information related to firmware simulation. During the simulation process of the IoT virtual device, system behavior information related to the device simulation process will appear in the behavior log. This information can reflect the system behavior of the device during the simulation, but it cannot reflect the system behavior of the device during operation. We delete the system behavior information related to the equipment simulation process to reduce the impact on the analysis of system operation behavior.

In general, the behavior log records the fine-grained system behavior of the IoT device, and the current security status information of the IoT device can be obtained by analyzing the behavior log. System behavior log files are unstructured text, which increases the difficulty of analyzing system behavior. To realize automated system behavior analysis, the system behavior log must be analyzed first, to parse the original system behavior log data into a system behavior event sequence. Therefore, we first structure the unstructured system behavior log into several parts (e.g., date, time, and content), then extract meaningful information from these parts, and finally generate a sequence of system behavior events.

To achieve the goal of automated log analysis, academia and industry have proposed many data-driven methods, including frequent pattern mining (LogCluster [17]), iterative partitioning (IPLoM [18]), layering clustering (LKE [19]), longest common subsequence calculation (Spell [20]), and parse tree (Drain [21]). These log analysis methods can automatically generate common event templates based on log data. Log parsing should make full use of the inherent structure and characteristics of log messages to obtain good parsing accuracy, instead of directly applying standard algorithms such as clustering and frequent pattern mining [22]. Among several parsing methods, Drain uses a fixed-depth tree structure to represent log messages and effectively extract event templates. This method uses the characteristics of logs and performs well in many log parsing situations.

Due to the complex structure of the behavior log and rich event templates, the above methods still cannot accurately parse the behavior log, and considering the accuracy, robustness, and efficiency of several log analysis methods, we further improved Drain to implement the system behavior log analysis. The following describes the analysis steps in detail.

During log parsing, an unstructured system behavior log file containing free text log messages is used as input. The unstructured system behavior logs we collected consist of a constant part and a variable part. The constant section displays the event template of the log message and remains unchanged for each log event. The variable section shows the parameters of the system during dynamic operation, which may change between different events. For the characteristics of the behavior log format, a regular expression function is written to split the log message. We structure the behavior log message into headers with Time, Syscall, PID, and Content.

When the log message is parsed, a structured system behavior log and a system behavior event template with

summary event counts are output. The content of structured system behavior in the structured system behavior log file is shown in Table 3. System behavior is structured to include event ID, PID information, system calls, event templates, event template-related event behavior list, and other system behavior content. The content of the system behavior event template file includes the event ID, the content of the event template, and the number of occurrences of the event template in the entire system behavior log file.

To distinguish each system behavior, we generate an identifier EventID for each system behavior. We use UTF8 to encode the important content of the structured system behavior (PID name, system call, event template, and event parameter list) and then use MD5 to encrypt the encoded content into a string of 8-bit length, and finally, for each encrypted character strings are mapped to one-to-one decimal numbers as EventID of each system behavior.

The system processes the structured system behavior log file to output the sequence of corresponding system behavior events for each PID. As shown in Table 4, the sequence of system behavior events executed by the process with PID 623 is 4, 5, 6, 6, 7, 8, 9, 10, 11, 12, 13, 14, and 15. Only three system behavior event sequences are illustrated in the table, and all system behaviors in the system behavior log are similarly operated. Finally, the system behavior event sequence is processed, PID-related information is deleted, and the system behavior event sequence is generated as shown in Table 5. The system behavior event sequence represents a collection of system behavior events executed by an IoT system program, and the system uses it as an input of an anomaly detection model.

Table 3: Example of structured system behavior.

EventID	22
Syscall	Sys socket
PID	372
PIDName	snmpd
EventTemplate	Family
ParameterList	["2","1","0"]

Table 4: PID and system behavior event sequence.

PID	Event sequence
623	4,5,6,6,7,8,9,10,11,12,13,14,15
647	3,2,5,6,2,2,2,4
649	1,2,3,2,6,6,6,8,9,6,2,4,5,6,3

Table 5: System behavior event sequence.

4 5 6 6 7 8 9 10 11 12 13 14 15
3 2 5 6 2 2 2 4
1 2 3 2 6 6 6 8 9 6 2 4 5 6 3

3.3.2. Model Training. The log parsing process generates a collection of system behaviors representing the execution of IoT system programs. Once the system behavior log entries are parsed into a sequence of system behavior events, the sequence of system behavior events will reflect a specific execution order execution path. Each data in this data set are mapped to each system behavior one by one.

The collection $S = \{s_1, s_2, s_3, s_4, \ldots, s_n\}$ is used to represent the system behavior sequence data set, and s_i represents the system behavior at position i in the current system behavior sequence. Obviously, s_i is one of the n possible system behaviors in the set S and is strongly dependent on the latest system behavior collection that appeared before s_i. Therefore, we model anomaly detection in the system behavior sequence as a multi-class behavior classification problem, where each different system behavior defines a class. We trained the anomaly detection model as a multi-class classifier in the latest context. The input is the historical record of recent system behavior, and the output is the probability distribution of n system behaviors in the system behavior set S, indicating the probability that the next system behavior in the sequence is $s_i \in S(i = 1, 2, \ldots, n)$.

Suppose t is the sequence ID of the next system behavior to be displayed. The input used for classification is the window w of the h most recent system behaviors, i.e., $w = \{s_{t-h}, s_{t-h+1}, \ldots, s_{t-2}, s_{t-1}\}$, where each s_i in S is the system behavior contained in the system behavior set S. Due to the repeatability of the system behavior, the same system behavior may appear several times in the window w.

The training phase relies on the system behavior generated by the normal execution of the IoT device system. For each system behavior log sequence of length h in the training data, the system update uses $s_i \in S(i = 1,2,\ldots, n)$ as the probability distribution of the next system behavior. For example, suppose that a small amount of system behavior logs generated by normal execution are parsed into a series of system behavior events: $\{s_5, s_8, s_{11}, s_2, s_{26}, s_6, s_3\}$. Assuming that the window size h is 4, the input sequence and output label pairs of the training model will be $\{s_5, s_8, s_{11}, s_2 \longrightarrow s_{26}\}$, $\{s_8, s_{11}, s_2, s_{26} \longrightarrow s_6\}$, and $\{s_{11}, s_2, s_{26}, s_6 \longrightarrow s_3\}$.

The output of the training phase is the conditional probability distribution $P(s_t = s_i \mid < s_{t-h+1}, \ldots, s_{t-2}, s_{t-1} >)$ (for each system behavior, $s_i \in S(i = 1,2,\ldots,n)$). The detection stage uses this model to predict and compare the predicted system behavior with the actual system behavior.

Inspired by recent research, we found that the abnormal behavior detection of the Internet of things in this study is naturally applicable to the LSTM-RNN model. The system behaviors generated by IoT devices can be organized in chronological order. Our anomaly detection model based on LSTM is shown in Figure 4, where the purpose and parameter settings of each layer are as follows.

(1) Input layer: after preprocessing and log analysis, the normal system behavior will be converted into a system behavior event sequence. We will use the parsed normal system behavior event sequence as the input layer data. Then, we group h consecutive system behaviors to form a system behavior window. Using the system behavior window, we can model the temporal relationship of the behavior of adjacent systems.

(2) LSTM layer: the input layer inputs system behavior to the LSTM layer. In each step, system behavior is

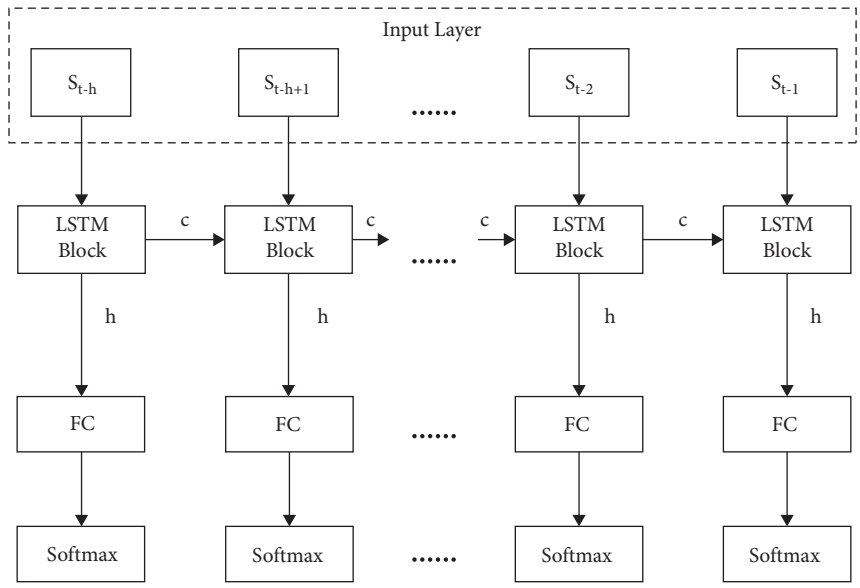

Figure 4: Behavior detection model based on LSTM.

assigned to the LSTM unit. In a single-layer LSTM, the output of the LSTM cell includes the cell state and the hidden state, and the hidden state and the cell state of the LSTM cell are transmitted to the next LSTM cell. Taking 2-layer LSTM as an example, the hidden state of each LSTM cell is also transferred to the stacked lower LSTM cell as its input.

(3) FC layer: we have placed a hidden full-connection (FC) layer between the LSTM layer and the softmax layer, whose size is equal to the number of system behavior categories.

(4) Softmax layer: we output the fully connected layer to the softmax layer for standardization. The output of the softmax layer is a probability distribution for each system behavior, and the probability distribution indicates how likely each system behavior becomes the next system behavior.

The loss function in training is categorical cross-entropy. The optimizer is Adam. In addition, the size of the epoch is 50. After training 50 epochs, the neural network converges.

3.3.3. Abnormal Behavior Detection. Our LSTM-based abnormal behavior detection model can learn the comprehensive and complex associations and patterns contained in a series of system behaviors generated by the execution path of the IoT virtual device system. In the process of abnormal behavior detection, we assume that the IoT virtual device is trusted, the data and operating environment in the IoT virtual device are ensured to be trusted and safe, and the system behavior collected based on the IoT virtual device is also trusted, and the attacker cannot carry out the attack and change the information of the system. For example, Intel SGX [23] technology can be used to provide data storage, data transmission, and run-time security protection for IoT virtual devices.

In summary, our system can detect attacks that cause abnormal behavior in the system's behavior sequence, which can lead to abnormal behavior in the system behavior log. For example, a password blasting attack may perform Telnet or SSH login operations for password verification multiple times, which is reflected in the system behavior log. The category system behavior will increase dramatically; for example, malware will download malware on the Internet of things device and gain control of IoT devices, which is reflected in the malware process in the device system. The malware process will appear to delete files, change network configuration, and other malicious operations, resulting in abnormal system behavior sequences. For example, some attacks will leave traces in the system behavior log. The attack may cause the IoT system to stop working, so the corresponding system behavior sequence ends early or abnormal system behavior occurs.

To realize the real-time abnormality detection of the system behavior of the IoT device, the system will collect the system behavior regularly (in seconds). Then, preprocessing, log analysis, and other processes are carried out, and finally, the system behavior set at this stage is obtained. Behavior $S_i \in S(i = 1,2,\ldots, n)$ is normal or abnormal, $w = \{s_{t-h}, s_{t-h+1},\ldots, s_{t-2}, s_{t-1}\}$is sent to the LSTM-based abnormal behavior detection model as its input, and finally the model outputs the probability distribution of each system behavior: $P(s_t = s_i|w) = \{s_1{:}p_1, s_2{:}p_2,\ldots, s_n{:}p_n\}$.

This probability distribution describes the probability that each system behavior from the system behavior set S becomes the next system behavior when based on normal IoT system behavior. In fact, S_i may be multiple system behaviors. For example, if the camera and the server are in communication, the system behavior S_i of the camera may be "send video information to the server" or "receive configuration information of the server." In different behavioral contexts, both are normal system behaviors.

The abnormal behavior detection model based on LSTM can learn information about the behavior of multiple systems that can be the next system behavior in the model training phase. In the detection phase, we sort them according to the probability of the possible system behavior Si, and if the value of system behavior is among the first p candidate values, they are regarded as normal values. Otherwise, the system behavior is marked as abnormal execution, and then, the abnormal behavior is provided to the abnormal behavior analysis and risk mitigation module for subsequent analysis and other steps.

3.4. Behavior Analysis and Risk Mitigation. IoT attacks are generally divided into multiple stages, and attacks at different stages will cause different device abnormal behavior. When the behavior analysis module obtains the abnormal behavior, it first obtains fine-grained abnormal behavior content from the structured system behavior log file according to the behavior event ID. According to the content of abnormal behavior, different risk mitigation policies are performed.

We propose a general risk mitigation method that is very flexible in different situations. The key idea is that we design a virtual security function for each real IoT device. This virtual security function performs device monitoring tasks, performs abnormal behavior detection, and manages the communication between the IoT real device and the Internet. This module is also responsible for generating and enforcing restricted network access for connected devices. Based on the analysis results of abnormal behavior, it blocks malicious access in real time to reduce the risk of abnormal behavior of the Internet of things. At the same time, the virtual security function also has an isolation function, which controls the access of IoT devices to other IoT devices in the intranet, to prevent one device from being compromised and endangering other devices in the same network segment. This can potentially prevent vulnerable IoT devices from being maliciously triggered by accidental access traffic while ensuring that the system can promptly warn administrators of risks. We propose the following risk mitigation policy, which aims to maintain as many IoT device functions as possible while minimizing security risks.

3.4.1. Network Isolation. The goal of network isolation is to prevent IoT devices with abnormal system behavior from communicating with other devices. To this end, the virtual security function divides the user's network into two virtual networks: an untrusted network and a trusted network. When the IoT device is running, the behavior security detection system will detect the behavior of the IoT device system in real time. The required network isolation level is determined based on the abnormal detection results. IoT devices with normal system behavior have been placed in a trusted network. IoT devices that have detected abnormal behavior are placed in an untrusted network and are strictly isolated from other devices.

According to the different risks caused by abnormal device behavior, we have designed three different isolation levels for IoT devices. (1) Strict isolation level: the IoT device is not allowed to communicate with other IoT devices, and the device does not have Internet access rights. (2) Limited isolation level: this allows devices to communicate with other devices in untrusted network coverage and a limited set of remote targets on the Internet (e.g., cloud services from vendors). (3) Trusted isolation level: this allows the device to communicate with other devices in the trusted network coverage and unrestricted Internet access. Network isolation at the device level granularity ensures that when threatened, any vulnerable device cannot infect other devices in the trusted network. The virtual security function can intercept the traffic in the network and ensure that it is filtered according to the required isolation level.

3.4.2. Flow Filtering. The goal of traffic filtering is to prevent attackers from communicating with vulnerable devices or exposing data. Traffic filtering is performed by virtual security functions and can target specific protocols or endpoints to minimize the impact of the functions of the affected devices.

3.4.3. User Notification. In some cases, network isolation and traffic filtering are insufficient to provide adequate protection. For example, if the IoT device has been attacked by malware, the malware prevents the device from automatically restarting the device to delete the malware in memory. In this case, the effective measure to protect the security of IoT devices is to manually restart or delete the devices at risk. Therefore, the purpose of the user notification is to help the user identify the problematic device, then warn the user that there is a device that cannot overcome the security vulnerability, and ensure that the user restarts or deletes it.

4. Implementation and Evaluation

4.1. Implementation. We evaluate the performance of IoT-DeepSense using the physical machine with an Intel i7-6700 CPU. The machine has 8 GB of memory, 500 G of hard disk, and runs Ubuntu 16.04.

We build a firmware virtualization environment based on FIRMADYNE. We have successfully conducted device simulations and tested multiple firmware versions of NETHEAR WNAP320, WNDAP350, WNDAP360, and WNAP210 series devices. The behavior detection model of IoT-DeepSense is implemented based on PyTorch.

4.2. Data Set. We simulate a normal and several attacked IoT environment, capture the system behaviors separately, and then use the collected system behaviors as the data set of the following experiment. The types of attacks we simulate mainly include IoT malware (Mirai and BrickerBot), DDoS (TFN), and password cracking (Hydra). Mirai is a classic DDoS attack, and BrickerBot is a permanent denial-of-service attack. TFN is an open-source DDoS tool that can perform various DDoS attacks, such as ICMP flooding, SYN

Table 6: System behaviors in our data set.

Type	Normal	Mirai	BrickerBot	Hydra	TFN	Total
Number of system behaviors	42600	4830	4800	2722	1756	56708
Data size	2.9 M	350 K	347 K	200 K	128 K	3.9 M

flooding, UDP flooding, and Smurf attacks. Hydra is an open-source tool for brute force password cracking that provides options for attacking login names of various protocols. Table 6 shows the type and the corresponding number of system behaviors in our data set.

4.3. Functional Evaluation

4.3.1. Behavior Collection. To test the effectiveness of the IoT behavior capture solution, we successfully performed device simulation based on multiple firmware versions of NETGEAR WNAP320, WNDAP350, WNDAP360, and WNAP210 series devices. According to the activity of the device, we collect the system behavior generated by each IoT virtual device running normally for 10 days. Then, we conduct an IoT attack on each device and collect the abnormal system behavior generated by each IoT virtual device under different types of attacks. The system behavior of each IoT virtual device is recorded in the corresponding system behavior log file.

Table 6 shows the system behavior information of the IoT virtual device captured by the behavior capture scheme. We successfully captured the system behavior of the device in a normal scenario with a data size of 2.9 M and a number of 42,600. In addition, we successfully captured different abnormal system behavior in different attack scenarios. It is found that the information contained in the fine-grained system behavior logs is very rich. For example, in the Mirai attack scenario, the system behavior log details that BusyBox executes a file transfer instruction to download the malware (mirai.mips) to the IoT device, then set file permissions on the malware, and finally run the malware. After the malware is successfully executed on the IoT device, the mirai.mips process is generated, so the mirai.mips process and its corresponding process operations appear in the system behavior log. In the scenario of the BrickerBot attack, the system behavior logs recorded in detail the BusyBox process executing a series of commands to destroy the device.

It can be seen from this that our system behavior capture scheme successfully captures the system's fine-grained behavior, and these fine-grained system behaviors can reflect the state of the system. The system behavior collected by the system behavior capture scheme is used as the data set of the abnormal behavior detection scheme. The data set contains the normal system behavior of each device of about 40,000 and the abnormal system behavior of about 15,000. The training set of the model includes all normal system behavior (approximately 24,000), and the validation set and test set contain normal system behavior (approximately 8,000) and abnormal system behavior (approximately 75,00). In addition to some basic system behavior characteristics (e.g., the number of system behaviors, number of system calls, PID, and parameters), there are some hidden characteristics (e.g., the timing characteristics of system behavior) in the system behavior data set. To learn this rich feature information, we model the system behavior based on deep learning to perform abnormal detection.

4.3.2. Abnormal Behavior Detection. Abnormal behavior detection is a binary classification problem. We label the classification results as true positive (TP), true negative (TN), false positive (FP), and false negative (FN). TP is the abnormal system behavior accurately determined by the abnormal behavior detection model. TN is accurately determined normal system behavior. If the method determines a certain system behavior as abnormal, but the system behavior is normal, we mark the result as FP. If this method determines a certain system behavior as normal, but the system behavior is abnormal, we mark the result as FN.

In addition to the number of FP and FN, the ability of the classification method is usually evaluated by standard indicators such as precision, recall, and F1 score.

Precision represents the percentage of true abnormal behavior among all detected abnormal behaviors. The calculation method is as follows:

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}. \tag{1}$$

Recall represents the percentage of abnormal behavior in the detected system behavior:

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}. \tag{2}$$

The $F1$ score value is the harmonic average of precision and recall:

$$F1 = \frac{2 * \text{precision} * \text{recall}}{\text{precision} + \text{recall}}. \tag{3}$$

(1) Effect of the Value of h. We first test the window value h of different sizes, that is, the effect of system behavior sequence length on the classification results. We record the classification performance of IoT behavior anomaly detection classification models under different window sizes. The test results are shown in Figure 5(a). When the window size is 6, the anomaly detection classification model has a good classification effect compared with the models with other window sizes, with a high recall rate of 93.2%, a precision of 90.9%, and an $F1$ score of 92.03%. When the length of the system behavior sequence is too short, the sequence cannot cover all the characteristics of malicious behavior and normal behavior. When the system behavior sequence is very long, the deep learning network cannot store more information.

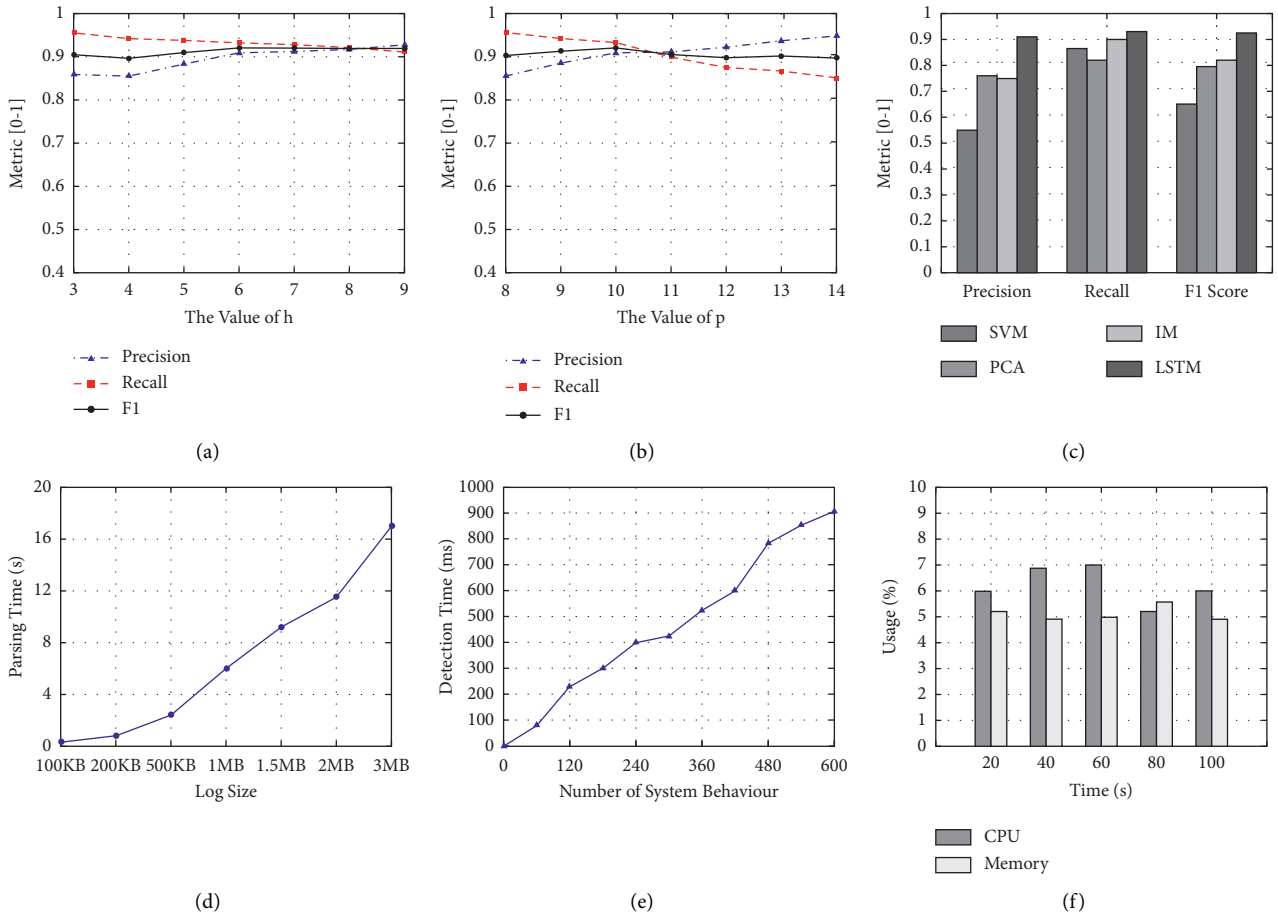

Figure 5: Performance test result. (a) Effect of the value of h. (b) Effect of the value of p. (c) Comparison with other algorithms. (d) Time cost of log parsing. (e) Time cost of behavior detection. (f) The consumption of CPU and memory.

(2) Effect of the Value of p. In the process of abnormal behavior detection, the next system behavior after the predicted window w is sorted according to the probability from large to small. If the system behavior is among the first p system behaviors, it is regarded as a normal value. Otherwise, the system behavior is regarded as abnormal behavior.

It is worth noting that many factors need to be considered when setting the candidate value p. If the candidate value p is set too large, some abnormal behavior information will be ignored, which may bring hidden dangers to the security of future IoT devices. If the candidate p is set too small, the normal system behavior may be judged as abnormal behavior information, which will interfere with abnormal detection. To test the influence of the candidate value p, we recorded the model anomaly detection results in the case of different candidate values p. The result is shown in Figure 5(b). When $p = 10$, the overall model has a great classification effect with a precision of 91.0%, F1 score of 90.5%, and recall rate of 90.1%, so we choose the value 10 as the size of the candidate value p.

(3) Comparison with Other Algorithms. Through the above analysis of the influence of the window size h and the influence of the candidate value p, we determined that we chose $h = 6$ and $p = 10$ as the parameters of our abnormal behavior detection model. In addition, we also determined that when the hidden layer is 3 layers, compared with other hidden layers, LSTM has the best performance. We compare LSTM with some representative log-based anomaly detection works, using support vector machine (SVM), principal component analysis (PCA), and invariant mining (IM). All these algorithms can be used to implement log-based anomaly detection and perform well. Figure 5(c) shows the comparison results of these four algorithms in terms of precision, recall, and $F1$ score. The results show that when the LSTM algorithm performs system behavior classification, the F1 value reaches 92%, which has a good classification effect.

This is because LSTM considers the sequence information of the system behavior sequence, and the other algorithms cannot capture this important information. The abnormal system behavior data set may contain a small amount of normal system behavior, which will affect the overall detection effect of the model, but in general, the abnormal behavior detection model can play a better behavior detection effect.

4.4. Performance

4.4.1. Behavior Log Parsing Performance. To measure the processing efficiency of behavior log parsing, we recorded

the running time required to complete the entire parsing process. We parse system behavior log files of different sizes and record the parsing time. The result is shown in Figure 5(d), the system behavior resolution time increases as the size of the system behavior log increases. There are about 1,500 system behavior records in the average 100 kB system behavior log. The average 100 kB file, that is, 1,500 system behavior parsing, takes 0.5 seconds. It can be seen that the parsing time of the system behavior log is basically negligible.

In the training phase of the abnormal behavior detection model, a large amount of normal system behavior needs to be collected for training, so a large amount of normal system behavior needs to be analyzed. In our experiments, we collected system behavior logs generated by each device running normally for 1 hour for training. The system behavior log size is about 3 MB, the system behavior is about 40,000, and it takes about 17 s of parsing time. The time consumed by this process is completed before the model training. When the model training is successful, real-time detection of system behavior will not require such a large number of system behavior logs to be parsed. Therefore, the performance of parsing behavior logs is basically negligible.

4.4.2. Behavior Detection Performance. After the system behavior log is prepared and parsed, we perform model training on the system behavior event sequence, the window size is 6, the predetermined value p is 10, and the number of hidden layers is 3 to train the model. The time consumption of model training is about 3,101 s. After the deep learning-based abnormal behavior detection model is trained, the system performs real-time detection on the IoT system behavior based on the model. The efficiency of detection is an important indicator of the detection effect, which reflects the time spent in detecting abnormal system behavior. Since the Internet of things is a system that requires timeliness, we do not want the abnormal behavior detection system to cause excessive time consumption in the detection process, thereby affecting the experience of using IoT devices.

To test the time consumption of the system in the process of detecting abnormal behavior, we recorded the time it took to detect different amounts of system behavior. Figure 5(e) shows the time used for abnormal behavior detection under different system behaviors. The results show that the time of abnormal behavior detection increases with the increase in the number of system behaviors, but the average detection time per 100 system behaviors is about 150 ms, which is within the acceptable range.

4.4.3. CPU and Memory Resource Consumption. We build virtual security functions for each device on the IoT behavior detection server. This virtual security function needs to perform behavior capture, anomaly detection, risk mitigation, and other functions during operation, so it needs to occupy a certain amount of CPU and memory resources for calculation. Figure 5(f) shows the amount of CPU and memory resources used during the abnormal behavior detection process of the virtual security function running stably for 100 seconds. The results in the figure show that the virtual security function takes up an average of 6.2% of CPU resources and an average of 5.1% of memory resources, which are within the acceptable range.

5. Related Work

5.1. IoT Behavior Security. Although the Internet of things has great potential, it also faces many security challenges [24]. Unfortunately, the security and privacy risks of IoT devices have not received enough attention. Due to a large number of IoT devices and a lack of defense capabilities, they are very attractive targets [25]. For example, IoT devices are used as robots to launch DDoS or spam, smart meters are attacked to reduce utility bills, and handheld scanners are hacked to enter logistic companies.

Most previous research on IoT security and privacy has focused on the use of firewalls [26], intrusion detection [27], access control strategies [28–30], and software patches [31] to protect the IoT infrastructure from attacks. These measures cannot guarantee the behavior security of IoT devices. For example, firewall rules can hardly guarantee that the door is locked when the user is not at home. In addition, the analysis of IoT devices and environments focuses on ensuring the security of IoT applications by analyzing source code. For example, some systems [32] infer the context of the application through run-time prompts to enforce permissions based on that context or require the user to authorize through the interface [33], while other systems apply static models to check for violations. Unfortunately, the current dynamic method is not enough to identify violations, while the static method [34] lacks accuracy and only implements limited strategies.

At present, there has been a lot of research on the behavior security of the Internet of things. HoMonit [35] is used for detecting abnormal behavior on the smart home platform, using side-channel information leakage in the wireless communication channel (packet size and packet interval) to infer the type of communication events between the smart device and the hub and then compare the inferred sequence of events with the expected program logic to identify inappropriate behavior without any modification to the existing infrastructure. IoTMon [36] is an IoT device physical interaction control system that can discover any possible physical interaction and generate all potential interaction chains across applications in the IoT environment. IoTMon also evaluates and mitigates the security risks of each discovered interaction chain between applications based on its physical influence. FlowFence [37] uses the taint tracking technology to track the information flow of sensitive data in the Internet of things applications, to ensure that the authorized users can use the data safely and legally to prevent the leakage of sensitive information. MCshield [38] is a DDoS defense framework. This framework deploys multiple smart filters at the edge of the attack source/target network to filter malicious traffic by learning DDoS behavior.

5.2. Firmware Virtualization. Avatar [39] is a framework that supports dynamic security analysis of embedded system

firmware. The framework performs dynamic analysis by running firmware on actual hardware. Although this method is very accurate, it has significant problems. Firstly, Avatar must obtain the physical hardware of the device under test, which puts a huge financial burden on developers. Secondly, Avatar needs to manually identify and interact with the debug port on the device interface, which reduces the scalability of this technology, especially for consumer devices that may not support hardware debugging. How to run the firmware in a virtualized environment and analyze the security of IoT devices is an important task. To solve this problem, FIRMADYNE [40] proposed a complete system simulation that relies on software, so as to realize the large-scale and automated dynamic analysis of embedded firmware binaries. FIRMADYNE solves the inherent challenges of dynamic analysis of embedded systems, such as the presence of hardware-specific peripherals, the use of non-volatile memory, and the creation of dynamically generated files. Costin [41] provides an extensible automated firmware dynamic analysis framework, using pure software simulation to find vulnerabilities in embedded devices. The framework can test the firmware Web application security issues through simulation. FIRM-AFL [42] is the first high-throughput grey box fuzzer for IoT firmware. It proposes a new technology for enhanced process simulation to solve the performance bottleneck caused by QEMU system mode simulation. P2IM [43] presents an abstract model for the I/O behaviors of the processor-peripheral interfaces to enable peripheral-oblivious emulation of MCU devices. uEMU [44] builds a general model for each peripheral to learn how to correctly emulate firmware execution at individual peripheral access points. It takes the image as input and symbolically executes it by representing unknown peripheral registers as symbols. During symbolic execution, it infers the rules to respond to unknown peripheral accesses.

However, the existing research work on the behavior of IoT devices focuses on the security detection of the network behavior and application behavior of IoT devices, and there is no comprehensive consideration of the behavior of IoT devices at the system level. Due to limited resources, it is difficult for IoT devices to obtain monitoring system-level behavior by installing monitoring software. Firmware virtualization technology can solve this problem. By simulating the real operating environment of the firmware, the operating data and security status of the system layer inside the device can be obtained, to perform system-wide security status detection and defense and provide a new idea for solving the problem of behavior security of the Internet of things devices. However, the current work in this area is mainly focused on the dynamic analysis and vulnerability mining of IoT devices, failing to consider the behavioral security detection of IoT devices. Therefore, we propose a behavior security detection system for IoT devices based on firmware virtualization and deep learning in this study.

5.3. Log-Based Anomaly Detection. Anomaly detection plays an important role in the management of modern large distributed systems. Logs that record system run-time information are widely used for anomaly detection. Lang et al. [45] used the SVM algorithm, a commonly used supervised classification method, to build a failure prediction model based on event logs. However, supervised methods require lots of manual efforts to construct data and labels, so unsupervised methods are more practical. Xu et al. [46] proposed an unsupervised method for detecting anomalous sequences of events using PCA, but PCA is data-sensitive so the detection accuracy of PCA will vary on different data sets. Lou et al. [47] converted log sequences into event count vectors and then used IM to mine invariants within vectors. The mined invariants would reflect the normal workflow of the detected system. Then, they used these invariants to detect anomalies in system logs. If the invariant relationship of the log session did not hold, it would be judged as an abnormal session.

6. Discussion and Future Work

This study designs and implements an IoT device behavior safety detection system based on firmware virtualization and deep learning. It implements IoT device behavior security detection based on fine-grained real-time dynamic system behavior, analyzes abnormal behavior, and proposes corresponding risk mitigation strategies. However, there are limitations in the design of this article, and further research and improvement are needed in future work.

(1) Due to the wide variety of IoT devices, firmware virtualization technology only supports firmware images that emulate fixed architectures (ARM and MIPS) and fixed systems (Linux). Even if the firmware image meets the appealing architecture and system conditions, there may be problems with incomplete image files and image encryption, resulting in a considerable number of IoT devices that cannot successfully simulate virtual IoT devices based on the device firmware and thus cannot obtain the IoT device's fine-grained dynamic behavior. Hence, firmware virtualization in a more in-depth way should be studied to realize a more general simulation method of the real operating environment of IoT devices, to better simulate IoT virtual devices.

(2) There are many types of IoT attack methods, and some attack types can achieve security detection and defense through security tools such as intrusion detection systems (e.g., Snort) and firewalls. System behavior can be used as a supplement to network behavior, thereby improving the framework for the secure detection of IoT device behavior. We expect to be combined with software-defined security in the future to generate multiple virtual security functions for each type of IoT device or each IoT device. The virtual security function can be used to detect and defend against a variety of IoT attacks against the network layer, or it can be based on firmware virtualization to simulate a real system environment to detect and defend against IoT system-level attacks.

(3) For the training process of abnormal behavior model based on deep learning, the training data set comes from the system behavior collected when the IoT device is initially connected to the network to maintain stable operation. However, in the actual environment, the system behavior of IoT devices may change due to reasons such as firmware updates. This change leads to the incompleteness of the previously trained model. In future work, we will further improve model accuracy and system performance.

7. Conclusion

In this study, we design and implement IoT-DeepSense, an IoT device behavior security detection system based on firmware virtualization and deep learning. Based on firmware virtualization technology, we build the real operating environment of the IoT system to capture the fine-grained system behaviors, then conduct security detection of the IoT device behaviors based on deep learning, analyze the abnormal behavior, and mitigate the risk according to the detection results. The implementation of the entire system is carried out on a separate IoT behavior detection server, which does not require modification of IoT devices with limited resources and is highly scalable. The evaluation results show that our abnormal behavior detection model has a good effect with an F1 score of 92%.

Data Availability

To test the effectiveness of the IoT behavior capture solution, we successfully performed device simulation based on multiple firmware versions of NETGEAR WNAP320, WNDAP350, WNDAP360, and WNAP210 series devices. According to the activity of the device, we collect the system behavior generated by each IoT virtual device running normally for 10 days. Then, we conduct an IoT attack on each device and collect the abnormal system behavior generated by each IoT virtual device under different types of attacks. The system behavior of each IoT virtual device is recorded in the corresponding system behavior log file.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grant 61872430 and in part by the Hubei Key Research and Development Program under Grant 2020BAA003 and the National Basic Research Program of China (973 Program) under Grant 2014CB340600.

References

[1] I. Lee and K. Lee, "The internet of things (iot): applications, investments, and challenges for enterprises," *Business Horizons*, vol. 58, no. 4, pp. 431–440, 2015.

[2] G. Davis, "2020:Life with 50 billion connected devices," in *Proceedings of the 2018 IEEE International Conference on Consumer Electronics (ICCE)*, p. 1, Las Vegas, NV, USA, 2018.

[3] M. M. Hossain, M. Fotouhi, and R. Hasan, "Towards an analysis of security issues, challenges, and open problems in the internet of things," in *Proceedings of the 2015 IEEE World Congress on Services*, pp. 21–28, IEEE, New York, NY, USA, July 2015.

[4] B. Edge, "Hacking the human heart," 2016, http://bigthink.com/future-crimes/hacking-the-human-heart.

[5] Wikipedia, "Dyn cyberattack," 2016, https://securityintelligence.com/posts/internetof-threats-iot-botnets-network-attacks/.

[6] D. McMillen, "Internet of threats: iot botnets drive surge in network attacks," 2021, https://www.euronews.com/2016/10/22/what-we-know-about-the-dyn-cyber-attack.

[7] W. Zhou, Y. Jia, A. Peng, Y. Zhang, and P. Liu, "The effect of iot new features on security and privacy: new threats, existing solutions, and challenges yet to be solved," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 1606–1616, 2018.

[8] F. Bellard, "Qemu, a fast and portable dynamic translator," *USENIX Annual Technical Conference, FREENIX Track*, vol. 41, p. 46, 2005.

[9] M. van Gerven and S. Bohte, "Editorial: artificial neural networks as models of neural information processing," *Frontiers in Computational Neuroscience*, vol. 11, p. 114, 2017.

[10] R. C. O'Reilly and M. J. Frank, "Making working memory work: a computational model of learning in the prefrontal cortex and basal ganglia," *Neural Computation*, vol. 18, no. 2, pp. 283–328, 2006.

[11] T. Mikolov, M. Karafiat, L. Burget, J. Cernocky, and S. Khudanpur, "Recurrent neural network based language model," in *Proceedings of the Eleventh Annual Conference of the International Speech Communication Association*, Chiba, Japan, September 2010.

[12] J. A. Pérez-Ortiz, F. A. Gers, D. Eck, and J. Schmidhuber, "Kalman filters improve lstm network performance in problems unsolvable by traditional recurrent nets," *Neural Networks*, vol. 16, no. 2, pp. 241–250, 2003.

[13] A. Graves, A. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Proceedings of the 2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 6645–6649, IEEE, USA, May 2013.

[14] K. Greff, R. k. Srivastava, J. Koutnik, B. R. Steunebrink, and J. Schmidhuber, "LSTM: a search space odyssey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 10, pp. 2222–2232, 2016.

[15] Motion, "Motion," 2019, https://motion-project.github.io/.

[16] M. Yoon, S. Mohan, J. Choi, M. Christodorescu, and L. Sha, "Learning execution contexts from system call distribution for anomaly detection in smart embedded system," in *Proceedings of the Second International Conference on Internet-of-Things Design and Implementation*, pp. 191–196, Pittsburgh, USA, April 2017.

[17] R. Vaarandi and M. Pihelgas, "Logcluster-a data clustering and pattern mining algorithm for event logs," in *Proceedings of the 2015 11th International Conference on Network and Service Management (CNSM)*, pp. 1–7, IEEE, Barcelona, Spain, November 2015.

[18] A. A. Makanju, A. N. Zincir-Heywood, and E. E. Milios, "Clustering event logs using iterative partitioning," in *Proceedings of the 15th ACM SIGKDD International Conference*

on Knowledge Discovery and Data Mining, pp. 1255–1264, Paris, France, June 2009.

[19] Q. Fu, J. Lou, Y. Wang, and J. Li, "Execution anomaly detection in distributed systems through unstructured log analysis," in *Proceedings of the 2009 Ninth IEEE International Conference on Data Mining*, pp. 149–158, IEEE, Washington, DC, USA, 2009.

[20] M. Du and F. Li, "Spell: streaming parsing of system event logs," in *Proceedings of the 2016 IEEE 16th International Conference on Data Mining (ICDM)*, pp. 859–864, IEEE, Barcelona, Spain, December 2016.

[21] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: an online log parsing approach with fixed depth tree," in *Proceedings of the 2017 IEEE International Conference on Web Services (ICWS)*, pp. 33–40, IEEE, Honolulu, HI, USA, June 2017.

[22] J. Zhu, S. He, J. Liu et al., "Tools and benchmarks for automated log parsing," in *Proceedings of the 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pp. 121–130, IEEE, Montreal, QC, Canada, May 2019.

[23] V. Costan and S. Devadas, "Intel sgx explained," *IACR Cryptology ePrint Archive*, vol. 2016, no. 86, 118 pages, 2016.

[24] T. Yu, V. Sekar, S. Seshan, Y. Agarwal, and C. Xu, "Handling a trillion (unfixable) flaws on a billion devices: rethinking network security for the internet-of-things," in *Proceedings of the 14th ACM Workshop on Hot Topics in Networks*, pp. 1–7, Philadelphia, PA, USA, November 2015.

[25] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang, and W. Zhao, "A survey on internet of things: architecture, enabling technologies, security and privacy, and applications," *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1125–1142, 2017.

[26] S. Kubler, K. Främling, and A. Buda, "A standardized approach to deal with firewall and mobility policies in the iot," *Pervasive and Mobile Computing*, vol. 20, pp. 100–114, 2015.

[27] B. B. Zarpelão, R. S. Miani, C. T. Kawakani, and S. C. de Alvarenga, "A survey of intrusion detection in internet of things," *Journal of Network and Computer Applications*, vol. 84, pp. 25–37, 2017.

[28] W. He, M. Golla, R. Padhi et al., "Rethinking access control and authentication for the home internet of things (iot)," in *Proceedings of the 27th {USENIX} Security Symposium ({USENIX} Security 18)*, pp. 255–272, Baltimore, MD, USA, August 2018.

[29] S. Qiu, D. Wang, G. Xu, and S. Kumari, "Practical and provably secure three-factor authentication protocol based on extended chaotic-maps for mobile lightweight devices," *IEEE Transactions on Dependable and Secure Computing*, vol. 2020, Article ID 3022797, 1 page, 2020.

[30] C. Wang, D. Wang, G. Xu, and D. He, "Efficient privacy-preserving user authentication scheme with forward secrecy for industry 4.0," *Science China Information Sciences*, vol. 65, no. 1, pp. 1–15, 2022.

[31] O. Leiba, Y. Yitzchak, R. Bitton, A. Nadler, and A. Shabtai, "Incentivized delivery network of iot software updates based on trustless proof-of-distribution," in *Proceedings of the 2018 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pp. 29–39, IEEE, London, UK, April 2018.

[32] Y. J. Jia, Q. A. Chen, S. Wang et al., "Contexlot: towards providing contextual integrity to appified iot platforms," *NDSS*, vol. 2, no. 2, p. 2, 2017.

[33] Y. Tian, N. Zhang, Y.-H. Lin et al., "Smartauth: user-centered authorization for the internet of things," in *Proceedings of the 26th {USENIX} Security Symposium ({USENIX} Security 17)*, pp. 361–378, Baltimore, MD, USA, 2017.

[34] Z. B. Celik, P. McDaniel, and G. Tan, "Soteria: automated iot safety and security analysis," in *Proceedings of the 2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*, pp. 147–158, 2018.

[35] W. Zhang, Y. Meng, Y. Liu, X. Zhang, Y. Zhang, and H. Zhu, "Homonit: monitoring smart home apps from encrypted traffic," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1074–1088, Toronto, ON, Canada, October 2018.

[36] W. Ding and H. Hu, "On the safety of iot device physical interaction control," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pp. 832–846, Toronto, ON, Canada, October 2018.

[37] E. Fernandes, J. Paupore, A. Rahmati, D. Simionato, M. Conti, and A. Prakash, "Flowfence: practical data protection for emerging iot application frameworks," in *Proceedings of the 25th {USENIX} Security Symposium ({USENIX} Security 16)*, pp. 531–548, Austin, TX, USA, August 2016.

[38] N. Dao, T. V. Phan, J. Kim, T. Bauschert, S. Cho, and D. Do, "Securing heterogeneous iot with intelligent ddos attack behavior learning," *IEEE Systems Journal*, pp. 1–10, 2017.

[39] J. Zaddach, L. Bruno, A. Francillon, and D. Balzarotti, "Avatar: a framework to support dynamic security analysis of embedded systems' firmwares," *NDSS*, vol. 14, pp. 1–16, 2014.

[40] D. D. Chen, M. Woo, D. Brumley, and M. Egele, "Towards automated dynamic analysis for linux-based embedded firmware," *NDSS*, vol. 16, pp. 1–16, 2016.

[41] A. Costin, A. Zarras, and A. Francillon, "Automated dynamic firmware analysis at scale: a case study on embedded web interfaces," in *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, pp. 437–448, Xi'an, China, June 2016.

[42] Y. Zheng, A. Davanian, H. Yin, C. Song, H. Zhu, and L. Sun, "Firm-afl: high-throughput greybox fuzzing of iot firmware via augmented process emulation," in *Proceedings of the 28th {USENIX} Security Symposium ({USENIX} Security 19)*, pp. 1099–1114, Santa Clara, CA, USA, August 2019.

[43] B. Feng, A. Mera, and L. Lu, "P2im: scalable and hardware-independent firmware testing via automatic peripheral interface modeling," in *Proceedings of the 29th USENIX Security Symposium (USENIX Security 20)*, pp. 1237–1254, USENIX Association, Santa Clara, CA, USA, September 2020.

[44] W. Zhou, L. Guan, P. Liu, and Y. Zhang, "Automatic firmware emulation through invalidity-guided knowledge inference," in *Proceedings of the 30th USENIX Security Symposium (USENIX Security 21)*, USENIX Association, Santa Clara, CA, USA, November 2021.

[45] Y. Liang, Y. Zhang, H. Xiong, and R. Sahoo, "Failure prediction in ibm bluegene/l event logs," in *Proceedings of the Seventh IEEE International Conference on Data Mining (ICDM 2007)*, pp. 583–588, IEEE, Omaha, NE, USA, 2007.

[46] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*, pp. 117–132, Big Sky, MT, USA, October 2009.

[47] J. Lou, Q. Fu, S. Yang, Y. Xu, and J. Li, "Ming invariants from console logs for system problem detection," in *Proceedings of the USENIX Annual Technical Conference*, pp. 1–14, Boston, MA, USA, June 2010.