Toward Intelligent IoT Endpoint Detection and Response Using Digital Twins via Firmware Emulation

Shin-Ming Cheng, Yi-Ching Lui, Nien-Jen Tsai, and Bing-Kai Hong

Abstract

The properties of short time-to-market, heterogeneity, constrained resources, and unfriendly interfaces for IoT endpoint devices render system-based security mechanisms in traditional desktops, such as antivirus, inapplicable. Moreover, popular network-based security solutions, such as IDS, might not completely detect and mitigate the rising fileless IoT attacks. This article leverages recent innovation, firmware emulation, to enable a digital twin (DT) of a targeted actual IoT endpoint device and to realize an intelligent IoT endpoint detection and response (EDR) platform. Inbound traffic to the actual IoT endpoint device is mirrored to the DT in the platform, and the system-level monitoring module integrated into the softwarized DT provides deep IoT endpoint detection in ways that are not possible on physical IoT endpoint devices. Machine learning algorithms are proposed to identify malicious behavior from system calls and network packets collected from system-level and network-level monitors, and suspicious packets containing harmful commands are further determined. The EDR consequently updates the IDS rules so that traffic to the actual IoT endpoint device with the same malicious patterns is recognized and blocked, thereby achieving endpoint response. In the experiment, we enable emulation of IoT endpoint devices with ARM, MIPS, and X86 architectures and realize Mirai malware and remote code execution (RCE) attacks to validate the proposed EDR platform. With a 99.94% accuracy rate in attack determination, we believe that the proposed solution is feasible for the protection of IoT endpoint devices behind the edge. Such outcomes identify secure functionalities that DT using firmware emulation could offer in the IoT paradigm, thereby opening the door to innovative mechanisms to combat IoT attacks.

Introduction

With sensing, computing, and communication capabilities, the Internet of Things (IoT) bridges the physical world and cyberspace, providing various kinds of applications to humans. The IoT framework comprises endpoint sensors and actuators (known as IoT endpoint devices), infrastructure edge nodes for data relaying (known as edge nodes), and IoT application servers. The fact that any modification to IoT devices in the cyber world will affect users' safety and privacy makes IoT a valuable target for adversaries. Moreover, IoT applications are typically designed for specific purposes, and instead of security requirements, aspects such as cost, performance, or power are the main considerations during the design process. For example, the short time-to-market increases the possibility of hard-coded passwords, thereby making IoT endpoints vulnerable to malware infection and fileless attacks [1]. Consequently, the security issue in IoT has been an ever-increasing concern.

Without sufficient computing power and a convenient access interface, traditional well-developed *host-based* protection mechanisms (e.g., antivirus) cannot be directly shifted into the IoT paradigm. *Network-based* solutions located at edge nodes such as a firewall or intrusion detection system (IDS) are feasible, where flow and packets to/from IoT endpoint devices are monitored and malicious ones are identified and blocked [2]. Various kinds of features, such as traffic volume, IP address, and port number, flow semantics, or payload and data, are extracted for the estimation of malicious traffic or malfunctioning IoT devices [3]. The powerful machine learning (ML) algorithms are applied as inference techniques from measured features to detect unknown malicious traffic automatically [2].

Although ML-based network detectors remarkably improved the defense performance of IoT endpoint devices, the fundamental issue that only network traffic can be utilized for analysis still restricts the degree of protection. E-Spion [4] first leverages system-level information such as CPU utilization or system call during the execution of the target process to determine the abnormal behavior. However, E-Spion suggests that the physical hardware of an IoT device should be connected to the IDS, which introduces a hardware dependence issue and is not scalable. By virtually rehosting firmware into an emulated IoT system, the operations of firmware are virtualized and decoupled from the original IoT endpoint hardware. The barriers of constrained resources and inaccessibility in IoT endpoint devices are tackled accordingly [5]. The emulated IoT endpoint being enabled with powerful computation capability could operate exactly the same as the original IoT endpoint hardware. Acting as a virtual replica, the

The authors are with National Taiwan University of Science and Technology, Taiwan; Shin-Ming Cheng is also with the Research Center for Information Technology Innovation, Academia Sinica, Taiwan. Nien-Jen Tsai is the corresponding author.

Digital Object Identifier: 10.1109/IOTM.001.2400070

emulated IoT endpoint could reflect the current status in high fidelity and is considered as a Digital Twin (DT) of the original IoT endpoint hardware [6].

By integrating system-level monitors in virtualized DT, the system behavior during operation such as system call or instruction executed can be well captured [7], which complements network-based solutions. Rather than directly enabling dynamic analysis and assessment in the emulated IoT system (e.g., fuzzing and concolic execution), this article achieves system-level detection in emulated IoT devices so that the original IoT endpoint is protected. In particular, the system-level monitors in the emulated IoT system apply an ML algorithm to identify the malicious action and determine the corresponding network traffic or packets. Then the edge could easily block suspicious traffic incurring the malicious activity so that IoT endpoint devices behind the edge are protected. We believe such IoT endpoint detection and response (EDR) is the first feasible solution without the support of IoT hardware and thus is much more scalable and efficient. We believe that introducing an emulated IoT system that acts as DT paves the way to designing practical defenses and future research that uses this as a foundation are expected to be developed as new security solutions for IoT endpoint devices.

The contributions of our study are as follows,

- **Digital Twin Framework.** Designed and implemented a digital twin framework that uses firmware emulation to create virtualized replicas of IoT devices for security analysis.
- **Enhanced System-Level Monitoring.** Advanced system-level monitoring capabilities with Strace, Mshell, and SystemTap leverage digital twins to detect and respond to potential security threats in IoT devices.
- **Firmware Vulnerabilities Identification.** Identified three previously known vulnerabilities in IoT firmware through testing and analysis using the digital twin framework.

The remainder of this article is organized as follows. We survey the existing research in network-based edge detectors and DT. The core technology applied to enable CDT, firmware rehosting, is extensively described. The novel intelligent IoT EDR based on the emulated IoT system is proposed and the experimental results are discussed. Finally, we conclude this work.

Background and Related Work

IoT Attacks

Recently, IoT botnets and malware received lots of attention due to the Mirai's source-code release and damage on global websites from its variants. Moreover, hackers exploit existing or unknown vulnerabilities on the victim devices to achieve fileless attacks without transporting a malicious binary [1]. The current IoT attacks consist of following stages [8]:

Stage 0: Scanning. By investigating reactions of crafted requests, the adversary could locate and identify vulnerable victims for the following attacks.

Stage 1: Exploitation. It is typically achieved by brute force password guessing or exploiting RCE vulnerabilities to gain access to the victim.

Stage 2: Downloading. The malicious binary is delivered from a loader to the victim.

Stage 3a: Execution. The victim is compromised via the execution of payload.

Stage 3b: Compromise. The adversary completely takes over the full control of the victim and could manipulate the victim persistently.

Stage 4: Communication. The victim communicates with the Command and Control (C2) server or the adversary and receives instructions from them.

Stage 5: Attack Action. The compromised victims acting as bots or relays launch stealth attacks such as distributed denial-of-service (DDoS) or lateral movement.

Before launching a malware attack, an adversary first collects publicly available IoT devices using a search engine such as shodan, which is often referred to as reconnaissance (see Step 0). The infection is typically achieved by brute force password guessing (see Step 1). In the downloading stage, the adversary typically leverages `wget`, `curl`, or `echo` commands to download malware to the victim (see Step 2), and then execute malware to infect the IoT device to form a botnet (see Step 3a). Unlike a fileless attack, an adversary will use the infected victims to connect to a C2 server (see Step 4), so that the hacker can control a large number of infected zombies via C2 server to launch DDoS attacks against specific services in a short period of time (see Step 5).

Regarding fileless attacks, the adversary deliberately hides their actions using known RCE vulnerabilities and leaves no files behind (see Step 1), thereby increasing the difficulty for later forensics [1]. Subsequently, the adversary can implant backdoors into the chosen victim devices to execute advanced persistent threats (see Step 3b). Such attacks are highly suitable for conducting subsequent attacks such as privilege escalation, data theft, information exposure, or network compromise.

IoT Network-Based Detector

With powerful computing capability, detectors located could monitor the communication traffic from/to the targeted IoT endpoints in real-time [2]. Such network-based IDS extracts features from the packet header, payload, or network flow and further recognizes specific activities from the measured features [3]. Traditionally, activities are compared with malicious ones in a signature database predefined by security experts. If there is a match, then the activity is determined as suspicious. For example, Heimdall [9] leverages whitelist and blacklist queries from VirusTotal as the basis for determination.

To detect unknown malicious traffic automatically, ML-based solutions with relatively high accuracy and a low false alarm rate have been proposed [2]. FlowGuard [10], for instance, inspects all packets passing through and identifies DDoS traffic, which is then blocked to protect IoT endpoint devices behind the edge. The Long Short-Term Memory (LSTM) ML technique is applied because temporally correlated DDoS traffic can be precisely captured. Passban IDS [2] learns the system's normal behavior during the training phase and then detects anomalies in incoming network traffic, particularly DDoS attacks. Additionally, to identify malicious traffic other than attack actions (i.e., stage 5), flow semantics are analyzed by investigating several consecutive interactions.

Traditionally, activities are compared with malicious ones in a signature database predefined by security experts. If there is a match, then the activity is determined as suspicious.

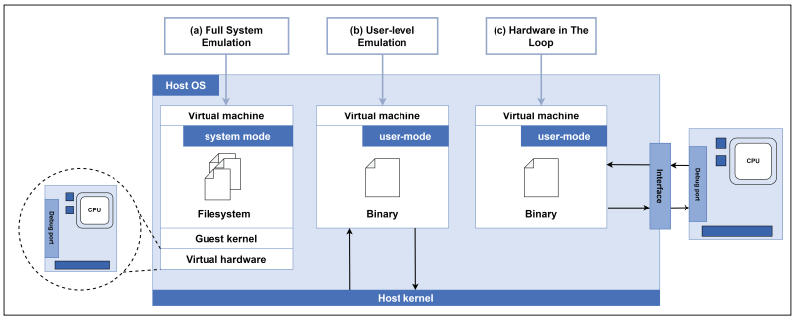

FIGURE 1. Firmware Emulation Techniques.

Digital Twin

Originated from intelligent manufacturing, a virtual twin that digitally projects a physical entity receives data from the physical counterpart and replicates its behavior [6]. With the aid of output from the virtual representation, real-time monitoring and controlling, fault diagnostics and early prediction, or dynamic optimization of the asset are enabled.

DT has become increasingly prevalent in the realm of IoT, serving as virtualized counterparts to sensors, actuators [11], and other IoT devices. They find application across various domains including healthcare [12], where they simulate digital patients, as well as in intelligent vehicles and IoT device management [13]. For instance, through the execution of representative functionalities and close integration with physical sensors, software replicas can closely mimic real-world behavior. Consequently, DTs serve as specialized logical entities tailored to specific IoT applications [11]. Leveraging in-body, on-body, and environmental sensors along with affordable devices, it becomes feasible to create digital representations of patients linked to targeted individuals. By harnessing data collected from these sensors, it becomes possible to discern the activities experienced by the individual, thereby facilitating improved care outside traditional healthcare settings and enabling the practice of "precision medicine" [12].

In order to handle the enormous amount of data measured from the physical object, various kinds of ML algorithms are proposed in the virtual twin [14]. In particular, a predictive model is responsible for predicting information using ML algorithms or neural networks so that further decisions are made and trade-offs are analyzed [6].

Firmware Rehosting for IoT Cybersecurity

By isolating execution from co-located physical hardware, emulation is now becoming a popular tool for software development, security analysis, and logic debugging [5]. The firmware of the targeted IoT endpoint device is extracted and rehosted within an emulation environment using three main approaches below.

User Program Emulation: As shown in Fig. 1b, the single binary of a particular service is executed as a process using QEMU user mode without emulating the entire firmware, kernel, and peripherals. A CPU emulator with a virtual stack and memory segment interprets and executes each instruction decoded from the binary. While process-level emulation proves efficient and apt for intricate security tests on a single binary like fuzzing, it does have drawbacks. Operations related to peripherals can lead to unexpected failures.

Hardware in The Loop (HITL): As illustrated in Fig. 1c, the CPU emulator in HITL can receive actual hardware responses when executing relevant instructions, as they are forwarded to the real IoT hardware via the debug port. However, hardware dependence inevitably reduces scalability and parallelism.

Full System Emulation: We can alternatively fully emulate the kernel, filesystem, and common peripherals using QEMU system mode, as depicted in Fig. 1a. The advantages of independence and high fidelity come with the cost of laborious and error-prone manual configurations, given the heterogeneous architectures and peripherals in IoT devices. Addressing the demand from a vast and rapidly increasing number of IoT devices, researchers have been focusing on automated emulation solutions.

DT Framework in Intelligent IoT EDR

Figure 2 depicts the proposed intelligent IoT EDR as a DT framework, aimed at monitoring, identifying, and thwarting malicious behavior at the system level. The DT is structured into data and model components. In Fig. 2a, the data component primarily comprises emulated IoT devices facilitated by rehosting technology, a topic thoroughly discussed later. To comprehend the behavior of these emulated devices, we've integrated a system-level monitoring module, detailed later. Network traffic bound for the actual IoT device is mirrored to the virtual DT's data component for thorough system and network level scrutiny. The data extracted from this analysis is then fed into the model component for behavioral analysis.

In Fig. 2b, the malicious behavior detector focuses on scrutinizing system calls for abnormal commands and labeling irregular log files. Additionally, the command extractor depicted in Fig. 2c identifies commands within the abnormal logs, translates them into IDS rules, and deploys these rules to the EDR. Subsequent traffic exhibiting similar attack patterns will be promptly identified and blocked based on these established rules.

Rehosted Firmware in DT

An emulated firmware is considered a DT of the actual IoT endpoint device since it operates exactly the same as the actual device. With sufficient computing power, we opt for full system emulation due to its high fidelity. To build the virtualized DT, we first extract the firmware of the actual IoT device and then rehost it using a well-known emulation tool, Firmadyne [15]. The firmware datasets we used are the same as those in the Firmadyne dataset package. We extracted the firmware using Binwalk, which involved unpacking compressed archives, extracting file systems, and isolating embedded files. This process allows for smooth acquisition of the filesystem and necessary modifications. Once we confirm the firmware architecture, we replace the original kernel to support essential system tools. Subsequently, the system is simulated using QEMU system-level emulation with appropriate configurations. Finally, the network interface undergoes intensive configuration, resulting in the successful establishment of the DT.

Ensuring compatibility is crucial when integrating system-level monitoring modules into emulated firmware. This involves testing and adjusting the kernel accordingly. Additionally, the diverse range of IoT

devices complicates automated firmware emulation. It's essential to thoroughly examine the various versions of libraries utilized in the system. Sometimes we need to compile the relevant tools with a suitable cross-compiler to ensure compatibility.

We present the first firmware emulation research within the context of DT, as compared to other studies. To assess fidelity in firmware rehosting, the study [5] examines various methods and their respective levels of fidelity. While our emulation achieves only module-level fidelity, we enhance the automation of our system, thereby improving scalability for further analysis. For instance, rather than prioritizing fidelity improvements, we might slightly modify the booting and kernel-related configuration to match the virtual environment. This is advantageous because some analysis tools can only run on specific versions of the kernel.

System-Level Monitoring

In contrast to network-based IDS solutions, which can only monitor network traffic, the proposed DT framework incorporates a system-level monitoring module. This enables the comprehensive capture of system-level behaviors outlined in Steps 1 and 3 in and earlier section. We specifically use the following three approaches for system-level monitoring: Strace, Mshell, and SystemTap. Strace and Mshell operate in user-space, while SystemTap operates in kernel-space. Detailed explanations are provided below.

Strace: As depicted in Fig. 3a, this is a typical debugging tool utilized for tracking system calls in progress. It operates at the user level and is implemented through the underlying `ptrace` probe. This tool allows us to observe the behavior of a specific process by hooking into the IoT device. All system calls performed by the process are recorded for subsequent analysis.

Middle Shell (Mshell): The shell serves as a user interface for interacting with the underlying system, enabling users to execute commands. Our modification involves transforming the default shell into an Mshell, capable of logging all commands intended for execution by a process. These logged commands are subsequently forwarded to the original shell for execution. Notably, unlike system calls, only executed commands are recorded in this solution.

SystemTap: Since the Strace and Mshell solutions are tailored for specific processes, a comprehensive system-level monitoring approach is necessary when the targeted process is unknown. As shown in Fig. 3c, SystemTap is integrated into the hosted OS to analyze the behavior of all running processes, including the kernel, by recording system calls. This solution is notably less conspicuous to user processes and is better suited for examining modern malware or fileless attacks equipped with sophisticated anti-detection technology.

Intelligent IoT Endpoint Detection and Response (EDR)

With the advantages of high fidelity and scalability, the system-level monitors integrated with the emulated IoT system can capture the runtime behavior of IoT endpoint devices behind the edge, thereby opening the door to enable the detection of malicious behavior. The presence of re-hosted firmware in emulated IoT devices also offers a new potential avenue to implement proactive protection for IoT endpoint devices behind the edge.

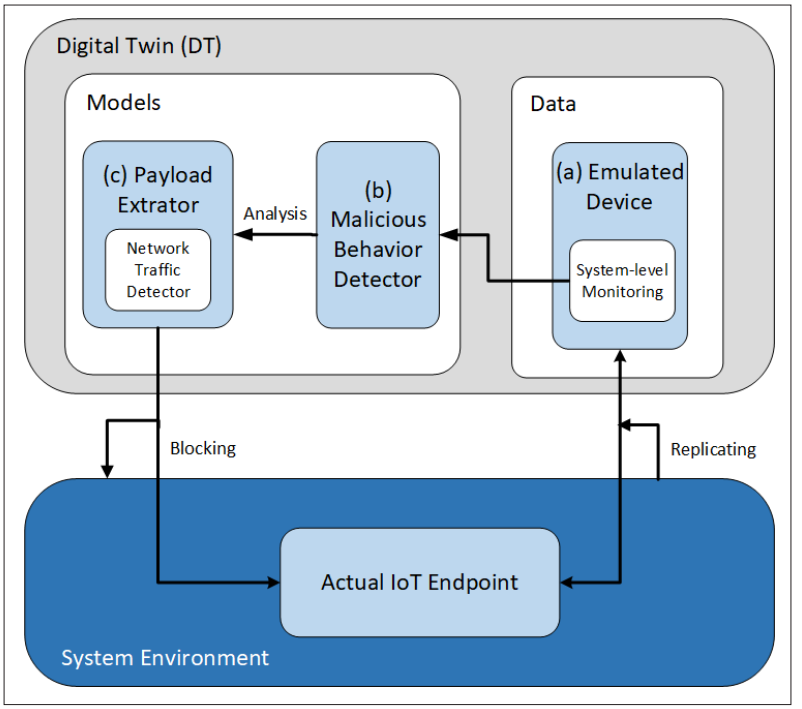

FIGURE 2. Framework for DTs.

Network Architecture and Procedures

Figure 4 describes the network architecture of the proposed intelligent IoT EDR. Different from the abstract DT framework depicted in Fig. 2, this figure concentrates more on the operation procedures and real traffic flow of the EDR . In particular, how the system-level malicious behavior of DT is detected and the corresponding attack is mitigated.

Step 0. Using the full system emulation techniques mentioned earlier, the images of emulated firmware are constructed and stored offline. Once an IoT endpoint device attaches to the EDR platform, its format is analyzed. The corresponding DT is efficiently launched by loading the images into the EDR.

Steps 1. and 2. The inbound traffic to the protected IoT devices mirrors the actual device and the DT. Since the DT is emulated from real firmware, it can be used for responding to the inbound traffic. However, in some cases DT may not send an appropriate response about peripheral devices, actual device assists DT to respond. The integrated system-level monitors intercept all the commands and system calls executed in the DT. In this case, even without malicious binary, the fileless attack can be identified. EDR could leverage the ML algorithm to determine if the downloaded binary or runtime behavior is malicious or not. The details of such endpoint detection will be described next.

Step 3. When malicious actions are detected, the EDR leverages the payload extractor mentioned in Fig. 2c to identify packets containing the malicious payload. Additionally, both the actual device and the DT are rebooted to synchronize their states. Subse-

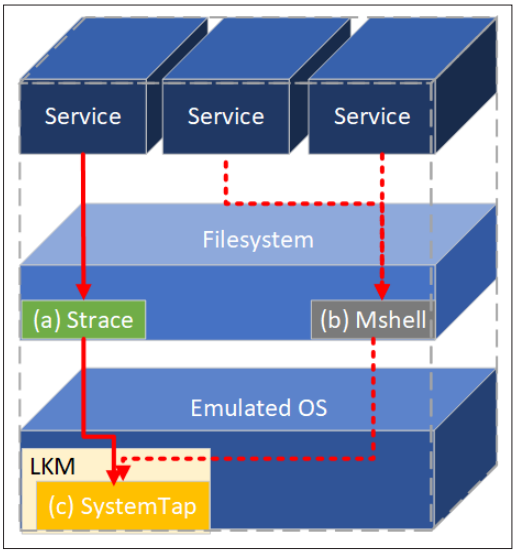

FIGURE 3. System-level monitoring.

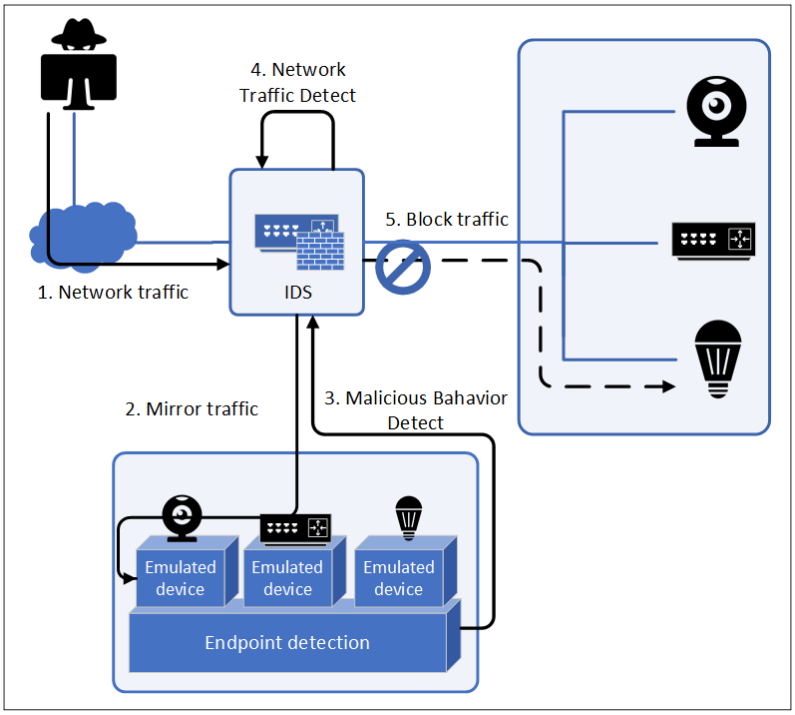

FIGURE 4. Network architecture of intelligent IoT EDR using full system firmware re-hosting.

quently, a YARA rule corresponding to the identified packets is constructed and relayed to the IDS within the EDR, where we look for networking-related command text strings.

Step 4. The IDS inside EDR will update the rule database according to the information received from the DT.

Step 5. The malicious traffic directed to the actual IoT endpoint device is immediately blocked. As a result, endpoint detection and protection are enabled via emulated DT's aid.

Endpoint Detection

Our endpoint detection acquires system-level behavior with the assistance of the emulated DT. The system-level data is then converted into a particular format to serve as input for the proposed detector. This detector consists of three phases:

1. Raw data collection
2. Feature extraction and pre-processing
3. Verification and analysis

We implement an ML-based detector that reasons about the semantics of system-level log sequences for identifying malicious behavior.

Raw Data Collection: The system-level monitor collects the system call sequence as the raw data. The data is labeled according to the type of target process and binary, e.g., malware or benign ware. Following is an example of raw data.

```
734 1629037867.467243 open("/dev/FTWDT101
watchdog", O_RDWR unfinished …
729 1629037867.346495 write(1, "Yowai: Raping
you sorry 0", 24) = 24 0.001728
732 1629037867.664298 read(0, unfinished ...
728 1629485115.479671 close(3) = 0 0.000215
728 1629485115.510892 ioctl(0, TCGETS, 0x7ec-
3da5c) = -1 ENOTTY (Inappropriate ioctl for
device) 0.000199
```

Feature Extraction and PrePprocessing: The system-level monitor collects the system call sequence as the raw data, consisting of system call name, system call parameters, and return value. We simply extract the name of the system call as features for the following processing. The parameters of the system calls are not considered to prevent confusion to the machine learning model. Then we concatenate names of system calls into a chronological sequence. The previous example after feature extraction becomes

```
open write read close ioctl
```

Verification and Analysis: We apply TF-IDF to convert system call sequences into vectors. If the entire dataset contains 167 different system call names, the vector dimension is (1,167). For example, if the dataset only contains the following two system call name sequences:

```
1: read read write write open close ioctl
2: open read write open read write
```

By using TFIDF, the dataset is converted as

```
   close ioctl open read write
1: 0.390548~0.390548~0.277878~0.555756~0.555756
2: 0.000000~0.000000~0.577350~0.577350~0.577350
```

The pre-processed features are then inputted into the ML model for detection, where Support Vector Machine (SVM) and Random Forest (RF) algorithms are applied.

After analyzing the system-level behavior, the packets containing malicious behavior will be captured. To enhance accuracy, we built another ML-based detector that considers the relationship of network traffic for identifying malicious behavior. As shown in Table 1, most IoT endpoint devices test the connection status by constructing DNS queries to a few domains, such as google.com. The cumulative count of DNS queries is beneficial. Additionally, since IoT endpoint devices do not actively establish connections to other devices, the number of unique IP addresses plays an important role in the features. For model selection, we chose the Support Vector Machine (SVM) and Random

Forest (RF) algorithms to train our dataset.

Validation and Performance Evaluation

Experimental Setup

The intelligent EDR in the experimental environment is implemented using an Intel NUC with an Intel Core i3-8109U processor, 32GB RAM, and a 256GB SSD. Ubuntu 18.04 is chosen as the operating system, and Security Onion is selected as the IDS engine with data visualization tools. To avoid direct modification of incoming network packets, Security Onion uses a mirror port to replicate network traffic, so external network interface cards are needed on the NUC to achieve this function.

Regarding the actual IoT endpoint devices for protection, commercial digital video recorders (DVRs), IP cameras, and routers are deployed with ARM, MIPS, and X86 architectures. Depending on the IoT device, we obtain and extract its firmware and activate a virtual DT for the IoT device using firmware emulation techniques, ensuring it possesses the same characteristics as the actual device. At the same time, we compile the corresponding kernel module using a cross-compiler for different architectures, integrate it into the DT, and collect system calls using the system-level monitoring module.

Attack Implementation

Regarding malware attacks, we implemented Mirai and its variants using multiple publicly available proofs of concept (PoCs) and Metasploit modules.[1] Additionally, we implemented four PoCs for fileless attacks targeting endpoint devices, including CVE-2020-10514, CVE-2019-10999, and CVE-2020-10987. Two notorious attacks were selected: buffer overflow and command injection. A buffer overflow occurs when large data is sent, exceeding the buffer size, which can cause system malfunctions or allow attackers to take control. Command Injection is a common type of web injection attack where administrators fail to filter sensitive characters in a website's input form, enabling attackers to send payloads to execute arbitrary commands.

Dataset

By applying the developed malware and fileless attacks mentioned in the previous subsection within our experimental environment, we generated trace results through fuzzing, some of which can be labeled as malware. To avoid a high false alarm rate, we first collect and analyze historical data to understand what typical behavior looks like for commands and log entries. Here are two examples:

1. `` `dd if=/dev/zero of=/dev/sda bs=512 count=1` ``: This command has the potential to cause disk wipe or data corruption. We need to check if this command is executed during routine maintenance or if it is unexpected, which could indicate data corruption.
2. `` `Aug 8 23:45:12 server sshd[1234]: Failed password for invalid user admin from 192.168.1.100 port 22 ssh2` ``: This log entry could indicate a brute force attack or unauthorized access attempt. We compare this against historical failed login attempts to determine if it is part of a larger brute force attack. Moreover, system binaries such as `init`, `/sbin/syslogd`, or `~/bin/sh` were executed to generate datasets labeled as benign. The dataset encompasses three architectures, with the number of traces for ARM, MIPS, and X86 being 457,373; 579,339; and 152,458, respectively.

Type	Feature
Data Size	TCP Upload Bytes, TCP Download Bytes, UCP Upload Bytes, UCP Download Bytes
Packet Count	TCP Upload Packets, TCP Download Packets, UCP Upload Packets, UCP Download Packets
Counter	Total Number of DNS Domain Name, Total Number of Unique IP
Other	Direction of Connection

TABLE 1. Extract feature from network traffic.

Evaluation Matrices

In the evaluation phase, we adopted the common evaluation metrics, namely, accuracy, recall, precision, false-positive rate, and F1-measure, to assess the performance of our proposed method. These metrics are defined based on the following intermediate measures.

- True positive (TP): samples correctly classified as positive.
- False positive (FP): samples incorrectly classified as positive.
- True negative (TN): samples correctly classified as negative.
- False negative (FN): samples incorrectly classified as positive.

Accuracy refers to the proportion of correct judgments of true and false. Precision refers to how much is true when the judgment is true. Recall is the probability of the samples in the positive class being classified correctly:

$$Recall = \frac{TP}{TP + FN}. \quad (1)$$

F1-measure is the weighted average of precision and recall:

$$F1\_measure = \frac{2 * (Recall * Precision)}{Recall + Precision}. \quad (2)$$

Experimental Results

The experiment was conducted on a dataset comprising all samples from our dataset. We utilized 10-fold cross-validation to validate our experimental results, ensuring the robustness of the experiment. The dataset was split into a training dataset, containing seventy percent of the overall dataset, and a testing dataset, containing thirty percent of the overall dataset. The results were averaged over ten independent runs, with the training and test sets determined by 10-fold cross-validation. Table 2 presents the experimental results of the two models. It is evident that Random Forest outperforms the other model, especially in terms of recall (i.e., 100%). Specifically, in the system-level detector, Random Forest achieved an accuracy of 99.22%, an F1-score of 99.29%, precision of 99.05%, and recall of 99.54%, outperforming SVM, which achieved lower accuracy and precision. Similarly, in the network traffic detector, Random Forest maintained a high level of performance, with nearly perfect accuracy

System-level Detector				
Classifier	**Accuracy**	**F1-score**	**Precision**	**Recall**
Random Forest	99.22%	99.29%	99.05%	99.54%
SVM	94.72%	96.78%	96.01%	97.55%
Network Traffic Detector				
Classifier	**Accuracy**	**F1-score**	**Precision**	**Recall**
Random Forest	99.94%	99.8%	99.95%	99.95%
SVM	99.35%	98.47%	99.43%	99.43%

TABLE 2. Performance comparison of Random Forest and SVM in system-level detector and network traffic detector.

(99.94%), F1-score (99.8%), precision (99.95%), and recall (99.95%). These results highlight the superior robustness and reliability of Random Forest in both detection scenarios, making it a more effective choice for IoT EDR systems. In particular, Random Forest's ability to maintain a strong balance between precision and recall suggests that it is highly capable of minimizing false positives and negatives, which is critical in maintaining security without overwhelming administrators with false alerts.

In the experiment, we exploit a command injection vulnerability in a commercial DVR device and then launch the telnet service. After observing the system call \texttt{write} interacting with the web service via a system-level monitor, we identify the packets containing the malicious payload and convert the payload into an IDS rule. For example, due to the command injection vulnerability in the DVR web service, the IDS rule is:

```
alert tcp any any -> any any (msg: "Command
injection"; content: "GET /goform/setUsbUn-
load/.js?deviceName=A;.*"; pcre:"/[a-zA-Z0-9]
{2}/"; sid:101;)
```

Conclusion

In order to enable a feasible detection and response solution for resource-constrained IoT endpoint devices, this article leverages a powerful edge to establish a DT of the actual IoT endpoint devices through firmware emulation. Integrating a system-level monitoring module with a software-defined DT could investigate the precise operational behavior of an IoT device, thereby resolving the drawbacks of typical network-based IDS solutions where only packets can be observed. We propose an ML-based detector in EDR where system calls are leveraged to reason the operational semantics, allowing harmful behavior to be detected. With the aid of IDS, malicious traffic to the actual IoT device can be blocked, thereby achieving endpoint response. Experimental results demonstrate that the EDR successfully captures malware and fileless attacks targeting commercial IoT endpoints deployed behind the edge with high accuracy, reaching 99.94%. As the first IoT DT facilitating the detection and protection of IoT endpoint devices, this article demonstrates the potential of firmware emulation in securing the IoT paradigm. Inspired by this article, many DT applications using firmware emulation are expected to be proposed for the security enhancement of IoT endpoint devices.

[1] https://docs.metasploit.com/docs/modules.html.

Acknowledgments

This work was partly supported by the National Science and Technology Council (NSTC), Taiwan, under Grant 113-2634-F-011-002-MBK.

References

[1] F. Dang *et al.*, "Understanding Fileless Attacks on Linux-Based IoT Devices with HoneyCloud," *Proc. ACM MobiSys 2019*, June 2019, pp. 482–93.

[2] M. Eskandari *et al.*, "Passban IDS: An Intelligent Anomaly Based Intrusion Detection System for IoT Edge Devices," *IEEE Internet Things J.*, vol. 7, no. 8, Aug. 2020, pp. 6882–97.

[3] P. M. S. Sánchez *et al.*, "A Survey on Device Behavior Fingerprinting: Data Sources, Techniques, Application Scenarios, and Datasets," *IEEE Commun. Surveys Tuts.*, vol. 23, 2nd Quarter 2021, pp. 1048–77.

[4] A. Mudgerikar, P. Sharma, and E. Bertino, "Edge-Based Intrusion Detection for IoT Devices," *ACM Trans. Manag. Info. Systems*, vol. 11, no. 4, Oct. 2020.

[5] C. Wright *et al.*, "Challenges in Firmware Re-Hosting, Emulation, and Analysis," *ACM Computing Surveys*, vol. 54, no. 1, Apr. 2021, pp. 1–36.

[6] R. Eramo *et al.*, "Conceptualizing Digital Twins," *IEEE Softw.*, 2021, accepted for publication.

[7] H. Alasmary *et al.*, "SHELLCORE: Automating Malicious IoT Software Detection Using Shell Commands Representation," *IEEE Internet Things J.*, 2021, accepted for publication.

[8] J. Khoury, M. Safaei Pour, and E. Bou-Harb, "A Near Real-Time Scheme for Collecting and Analyzing IoT Malware Artifacts at Scale," *Proc. 17th Int'l. Conf. Availability, Reliability and Security*, 2022, pp. 1–11.

[9] J. Habibi *et al.*, "Heimdall: Mitigating the Internet of Insecure Things," *IEEE Internet Things J.*, vol. 4, no. 4, Aug. 2017, pp. 968–78.

[10] Y. Jia *et al.*, "FlowGuard: An Intelligent Edge Defense Mechanism Against IoT DDoS Attacks," *IEEE Internet Things J.*, vol. 7, no. 10, Oct. 2020, p. 9552–62.

[11] R. Minerva, G. M. Lee, and N. Crespi, "Digital Twin in the IoT Context: A Survey on Technical Features, Scenarios, and Architectural Models," *Proc. IEEE*, vol. 108, no. 10, Oct. 2020, pp. 1785–1824.

[12] H. Elayan, M. Aloqaily, and M. Guizani, "Digital Twin for Intelligent Context-Aware IoT Healthcare Systems," *IEEE Internet Things J.*, vol. 8, no. 23, Dec. 2021, pp. 16,749–57.

[13] G. Mylonas *et al.*, "Digital Twins from Smart Manufacturing to Smart Cities: A Survey," *IEEE Access*, vol. 9, Oct. 2021, pp. 143,222–1,432,492.

[14] M. M. Rathore and S. A. Shah, "The Role of AI, Machine Learning, and Big Data in Digital Twinning: A Systematic Literature Review, Challenges, and Opportunities," *IEEE Access*, vol. 9, Feb. 2021, pp. 32,030–352.

[15] D. D. Chen *et al.*, "Towards Automated Dynamic Analysis for Linux-Based Embedded Firmware," in Proc. NDSS 2016, Feb. 2016.

Biographies

Shin-Ming Cheng (smcheng@mail.ntust.edu.tw) received the B.S. and Ph.D. degrees in computer science and information engineering from the National Taiwan University, Taipei, Taiwan, in 2000 and 2007, respectively. Since 2012, he has been on the faculty of the Department of Computer Science and Information Engineering, National Taiwan University of Science and Technology, Taipei, where he is currently a Professor. Since 2022, he has served as the Deputy Director-General in Administration for Cyber Security, Ministry of Digital Affairs. His current interests are mobile network security, IoT system security, malware analysis and AI robustness. He has received IEEE Trustcom 2020 Best Paper Awards.

Yi-Ching Lui (m10815109@mail.ntust.edu.tw) received Master degree in computer science and information engineering from National Taiwan University of Science and Tchnology, Taipei, Taiwan, in 2022.

Nien-Jen Tsai (m11115009@mail.ntust.edu.tw) is an open-source enthusiast who enjoys enhancing computational speed and security. She has contributed to projects such as LLVM, libFuzzer, QEMU, and OpenSSL. Her current research focuses specifically on system and network security.

Bing-Kai Hong (d10815003@mail.ntust.edu.tw) received his B.S. degree in computer science and information engineering from the National Taiwan University of Science and Technology, Taipei, Taiwan, in 2018. He is currently a Ph.D. candidate of the Department of Computer Science and Information Engineering, National Taiwan University of Science and Technology, Taipei. He visited EURECOM and NICT Cybersecurity Lab in 2018 and 2019, respectively. His research interests are secure system integration and development using virtualization technologies in mobile networks and IoT systems. He has received a 4-year scholarship of the Ministry of Science and Technology, CISC 2020 and TANET 2021 best paper awards.