CMD: Co-analyzed IoT Malware Detection and Forensics via Network and Hardware Domains

Ziming Zhao, Zhaoxuan Li, Jiongchi Yu, Fan Zhang✉, *Member, IEEE*, Xiaofei Xie, Haitao Xu, Binbin Chen, *Member, IEEE*

Abstract—With the widespread use of Internet of Things (IoT) devices, malware detection has become a hot spot for both academic and industrial communities. Existing approaches can be roughly categorized into network-side and host-side. However, existing network-side methods are difficult to capture contextual semantics from cross-source traffic, and previous host-side methods could be adversary-perceived and expose risks for tampering. More importantly, a single perspective cannot comprehensively track the multi-stage lifecycle of IoT malware. In this paper, we present CMD, a co-analyzed IoT malware detection and forensics system by combining hardware and network domains. For the network part, CMD proposes a tailored capsule neural network to capture the contextual semantics from cross-source traffic. For the hardware part, CMD designs an entire file operation recovery process in a side-channel manner by leveraging the Serial Peripheral Interface (SPI) signals from on-chip traces. These traffic provenance and operating logs information could benefit the anti-virus countermeasures for security practitioners. By practical evaluation, we demonstrate that CMD realizes outstanding detection effects (*e.g.*, ∼99.88% F1-score) compared with seven state-of-the-art methods, and recovers 96.88%∼99.75% operation commands even if against adaptive adversaries (that could kill processes or tamper with operation log files). A by-product benefit of such an external monitor is CMD introduces zero latency on the IoT device, and incurs negligible IoT CPU utilization. Also, since SPI focuses on file operations, the proposed hardware trace forensics does not have the data explosion problem like previous work, *e.g.*, recovered logs of CMD only take up limited extra space overhead (*e.g.*, ∼0.2 MB per malware). Furthermore, we provide the model interpretability for the capsule network and develop a case study (Hajime) of the operation logs recovery.

Index Terms—IoT malware detection, forensic analysis, SPI bus, multi-stage lifecycle

1 Introduction

Malware continues to be prevalent in Internet infrastructure nowadays and has drawn the attention of many security professionals [1]–[8]. Although malware is not a new threat, the boom in the Internet of Things (IoT) broadens and amplifies its attack surface. In other words, the recent surge in embedded device adoption and the IoT revolution is rapidly changing the malware landscape. Unfortunately, compared to desktop and mobile, IoT devices are often highly vulnerable due to limited resources, improperly configured, unpatched, and headless nature (*e.g.*, lack a graphical user interface), thereby they might be used to create large, powerful botnets [9]. Most famously, Mirai was used to launch vast volumetric DDoS [10], *e.g.*, Dyn (a DNS provider) suffered a 1.2 Tbps attack. Recently, the new variant Hajime has come into view of the security community as it infected no less than millions of devices based on the P2P protocol [11]. Although Hajime has not launched any DDoS attacks to date, it could become a severe threat if its full potential is realized.

To detect malware, researchers present a series of solutions with respect to the network-side or host-side. On the one hand, the former intends to profile the network traffic fingerprint when the IoT devices are accessed [12]–[14]. On the other hand, the latter aims to detect malware based on the runtime information, such as the operation log [15], system call [16], and performance counters [17]–[19]. Although these methods have achieved good results, there are some limitations when security practitioners put most existing proposals into practice. We summarize them as the following challenges.

① *Lack of lifecycle tracking for IoT malware.* Previous research reveals the threat lifecycle of IoT malware, which involves the infection vectors, payload properties, persistence methods, capabilities, C&C infrastructure, etc [9]. Either the network-side or host-side approaches can only provide

Manuscript received February 22, 2023; revised July 30, 2023. This work was supported in part by National Natural Science Foundation of China (62227805, 62072398, 62172405), by SUTD-ZJU IDEA Grant for visiting professors (SUTD-ZJUVP201901), by National Key R&D Program of China (2020AAA0107700), by Alibaba-Zhejiang University Joint Institute of Frontier Technologies, by Zhejiang Key R&D Plan (2021C01116), by Leading Innovative and Entrepreneur Team Introduction Program of Zhejiang (2018R01005), by Research Institute of Cyberspace Governance in Zhejiang University, by National Key Laboratory of Science and Technology on Information System Security (6142111210301), by State Key Laboratory of Mathematical Engineering and Advanced Computing, and by Key Laboratory of Cyberspace Situation Awareness of Henan Province (HNTS2022001). (Corresponding author: Fan Zhang)

- *Ziming Zhao, Fan Zhang, and Haitao Xu are with the College of Computer Science and Technology, Zhejiang University, Hangzhou, 310027, China. is also with ZJU-Hangzhou Global Scientific and Technological Innovation Center, 311200, with the Key Laboratory of Blockchain and Cyberspace Governance of Zhejiang Province, 310027, with Jiaxing Research Institute, Zhejiang University, 314000, and with Zhengzhou Xinda Institute of Advanced Technology, Zhengzhou, 450001, China. He is a visiting professor at the Singapore University of Technology & Design (SUTD). E-mail: {zhaoziming, fanzhang, haitaoxu}@zju.edu.cn.*
- *Zhaoxuan Li is with the State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, 100093, China, and also with the School of Cyber Security, UCAS, Beijing, 100049, China. E-mail: lizhaoxuan@iie.ac.cn.*
- *Jiongchi Yu and Xiaofei Xie are with the School of Computing and Information Systems, Singapore Management University, Singapore 188065. E-mail: jcyu.2022@phdcs.smu.edu.sg, xfxie@smu.edu.sg.*
- *Binbin Chen is with Advanced Digital Sciences Center, Singapore, Singapore, 138632, and with Singapore University of Technology and Design, Singapore, Singapore, B96049. E-mail: binbin_chen@sutd.edu.sg.*

insights into the partial life cycle of the IoT malware. For instance, the network-side methods are not enabled to precisely locate the malicious script path and grasp the concrete behavior, *e.g.,* binary file execution. While the host-side methods may not know the provenance of malicious binary and how it was planted, such as scanning and infection stages. We will introduce the detail of the IoT malware lifecycle and our motivations in § 2. Incomplete information may hinder effective anti-virus countermeasures, or lead to the semantic gap [20], [30] between the model inference results and operations of the network administrator. Furthermore, there are still some challenges in network-side or host-side unilaterally.

② *Hard to capture contextual semantics from cross-source traffic.* The previous network-side methods generally propose to discover malicious interactions at the session level or source level. They provide a local characterization of traffic patterns but are not very applicable to IoT-centralized network topologies. For example, Hajime [11] utilizes the peer-to-peer (P2P) distributed hash table (DHT) to disseminate software updates to its bots. This means that it is difficult to capture these contextual semantics from the session-level or source-level traffic. Moreover, producing a dataset with session-level or source-level labels is inappropriate and labor-intensive in this scenario, *e.g.,* it is not easy to define the P2P node lookup traffic as benign or malicious.

③ *Host-side analysis could be adversary-perceived and expose risks for tampering.* Using host-side analysis to obtain details of malware is a viable scheme, such as causality analysis for system logs [21]–[25]. Some investigation approaches will adopt system hooking or adding extra logic [26] as Anti-Viruses core processes usually run with administrator privileges [27]. However, these proposed techniques have some limitations. On the one hand, these above processes could be adversary-perceived, *e.g.,* the attacker may detect the existence of a running monitor by checking some dynamic fields [28]. On the other hand, internal monitors (such as the ones using existing hardware features) are prone to be subverted, *e.g.,* the attacker could try to inject fake state data into the system [29]. If leveraging hardware design knowledge to develop an external monitor, such as a bus-connected System-on-a-Chip (SoC), it will tend to be tamper-proof [30].

In this paper, we advance a comprehensive solution to bridge the network-side and host-side. To this end, we present CMD, a co-analyzed system by combining the network with hardware domains to track the IoT malware lifecycle (coping with the challenge ①). Among them, the network-side focus on detecting malicious behaviors while the host-side emphasizes fine-grained forensic analysis. Moreover, we design a tailored capsule neural network [31] to extract cross-source contextual semantics and identify the attack traffic (coping with the challenge ②). For the host side, we propose an entire file operation recovery process by leveraging the Serial Peripheral Interface (SPI) signals from on-chip traces. This side-channel-manner forensic analysis, as an external monitor, will not be perceived by the adversary and has the advantage of tamper-proof properties (coping with the challenge ③).

In a nutshell, we make the following contributions:

- We carefully examine the problems in the current IoT malware landscape and propose a novel system, named CMD, to achieve efficient detection and fine-grained forensics.
- For the network part, CMD presents a capsule neural network to identify the malicious behavior to adapt to the cross-source traffic characterization.
- For the hardware part, CMD leverages the SPI bus to monitor and extract the on-chip traces. And we design the inlined operating logs recovery method to benefit the anti-virus countermeasures. Moreover, this side-channel-manner hardware forensics process will not be perceived by adversaries and it is tamper-proof.
- By practical evaluation, we demonstrate that CMD realizes the predominant traffic detection effect (*e.g.,* ∼99.88% F1-score), compared with seven state-of-the-art (SOTA) traffic classification models. Also, CMD recovers 96.88%∼99.75% operation commands, even if against adaptive adversaries (that could kill processes or tamper with operation log files). Particularly, the recovered logs only take up limited extra space overhead (*e.g.,* ∼0.2 MB per malware). Results in the integration test show that CMD introduces negligible CPU utilization in the IoT device. In addition, we conduct a case study of the Hajime attack for log recovery, and interpretability experiments for the capsule network to provide deep insights.

2 Background and Motivation

In this section, we give a brief overview of Mirai's operation to provide a background on IoT malware. We then discuss the motivation behind our proposed system.

Background. Different from desktop and mobile malware, IoT malware can be a bigger threat given its limited resources, improperly configured, unpatched, and headless nature (*e.g.,* lack a graphical user interface) [9]. In addition, IoT malware has already incorporated advanced polymorphic and anti-analysis tactics such as self-induced malicious script deletion, process killing, or system log tampering [61]–[64].

The summary of Mirai's operation [10] is shown in Figure 1, which contains multiple phases. First, the compromised device performs scanning to find new vulnerable devices, and a brute-force login will be conducted as soon as identifying the potential victim. Then, it will send the victim device IP and associated credentials to a hardcoded report server after successful login. Subsequently, a separate loader program asynchronously infected the device by determining the underlying system environment. Next, this victim device downloads and executes the architecture-specific malware to complete the infection. Finally, these bots will listen for attack commands to launch DDoS attacks, *etc.*

Motivation. As mentioned above, the IoT malware involves multi phases [10] in its lifecycle [9], including ① scanning and brute-force, ② report server, ③ loader program, ④ downloading and executing, ⑤ command & control (C2) and attack. For early detection, the traffic analysis techniques are more concerned with phases ②∼④ before the attack is launched. However, the identification results only provide users with which flow is malicious without fine-grained investigation such as detailed script location

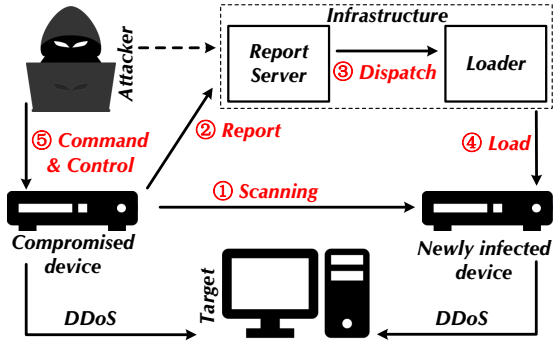


Fig. 1. The infection process of Mirai.

in the devices. The host-side technologies enable runtime information analysis based on the system API call (or logs) yet lack the mastery of the infection provenance. If we could bridge the network-side and host-side, more attack details will be collected to facilitate the anti-virus countermeasures in the entire IoT ecosystem. This is the main motivation to combine the network and hardware domains.

Furthermore, our design principles for network and hardware parts are also derived from the latest trends in IoT malware nowadays. (i) For the network part, recent IoT malware tends to be decentralized manner based on P2P protocol so that their dissemination process could be hidden in multiple sessions and even multiple sources. Therefore, we develop the tailored feature map production to capture cross-source contextual semantics. This process naturally correlates adjacent packets from the arrival sequence, which is different from the typical traffic identification solutions such as session-level or flow-level. Then, based on the generated feature map, we leverage the capsule neural network to extract those key behaviors manifested by malicious events.

(ii) For the hardware part, the existing forensic analysis could be adversary-perceived and expose risks for tampering. The adversary perception refers to the attacker who could realize the existence of a running monitor by checking patterns of some dynamic fields [28]. Subsequently, they may perform a series of tampering to disrupt normal operations or evade detection [29]. That is to say, internal monitors (such as the ones using existing hardware features) are prone to be subverted, while external monitors, such as a bus-connected System-on-a-Chip (SoC), tend to be tamper-proof [30].

An observation is that the evolution of IoT malware tends to use many persistence methods, such as installing themselves as either a service, a startup script, a system module, or a backdoor [9]. This persistent malware usually implants malware viruses into the Electrically Erasable Programmable Read-only Memory (EEPROM) instead of in-memory to achieve persistence on IoT devices. In addition, even typical malware that infects Random Access Memory (RAM) will perform a series of file operations on Read Only Memory (ROM). For example, Mirai can read the executable binary into RAM for malicious activities, while it also involves some operations on ROM such as using cat to analyze architecture (*e.g.*, e_machine field); using $wget$, $tftp$, or $echo$ to transfer the payload [32]. These file operations on ROM can be captured by Serial Peripheral Interface (SPI), so we intend to leverage SPI signals to analyze on-chip traces.

3 Design Space and Threat Model

Problem Space. The vulnerabilities of IoT devices are mainly in terms of two aspects: (i) The attackers access the IoT devices and invade them by remote exploitation or default credentials, such as weak passwords. Subsequently, the malicious loader program or malware binary could be implanted through the network. (ii) The malware programs in compromised devices execute the pre-designed commands to complete the infection. Among them, the former usually produces some corresponding network traffic fingerprints. And the latter mainly involves a series of file operations [10], [11], *e.g.*, file creation, writing, permission modification, and self-induced deletion. Therefore, the main goal of CMD is to detect the aforementioned malicious behaviors and forensic analysis to report operation details.

Adversary Model. We consider strong adversaries that adopt dynamic attack strategies with abundant IoT malware resources. On the one hand, the attackers can access IoT devices and implement malicious activities, such as brute-force password cracking and ARP sniffing. On the other hand, they could implant malware loader and binary into the IoT devices' built-in chips to achieve infection. We consider centralized IoT malware (*e.g.*, Mirai [10]), and also decentralized ones (*e.g.*, Hajime [11]).

For defenders, the hardware access of the IoT device and its traffic are available. However, a single perspective may lack a complete picture of the attack (*i.e.*, the lifecycle of malware). Specifically, if only the network-side traffic is used for analysis, it may not be possible to grasp the infection location and the specific malicious behaviors performed by the malware on the device. While only the host-side forensics lacks knowledge of the attack source, such as who performs the brute force and where the executable binary is from. Therefore, we tend to combine the network and hardware domains to advance malware detection and forensics. Meanwhile, we do not assume additional collaborations from other Internet entities, such as IP blacklists provided by security vendors.

Assumptions. We explain some assumptions here. Given the fact that the hardware-part design of CMD needs to collect SPI signals for the protected device, so physical access is required. We admit that physical access may not be convenient sometimes, but as a novel external monitor, CMD can bring new directions and perspectives for forensic analysis. Meanwhile, a feasible direction is to devise integrated customized solutions for industrial production to advance this side-channel-manner analysis. Furthermore, we intend to enable both network traffic detection and hardware trace analysis before infection to avoid missing some malicious behavior (the discussion of data explosion is in § 4.1 and the evaluation of log data overhead is in § 5.5). In fact, the assumptions about the active state of the CMD are similar to typical anti-virus software, which practitioners are usually running anti-virus programs in advance to detect potential malware [33]–[35].

4 Design Details of CMD

In this section, we first introduce the high-level detection logic of CMD. Then we elaborate on the design details in terms of the hardware part and network part.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3311012

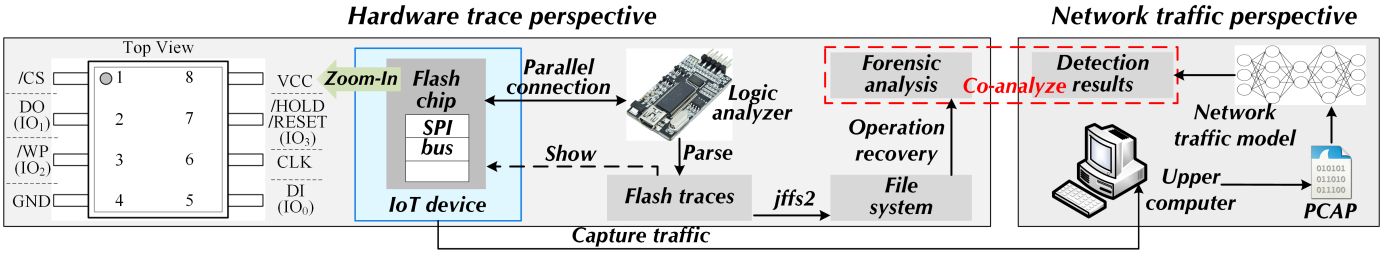


Fig. 2. The overview of CMD. It combines both the hardware and network perspectives.

4.1 Overview

In Figure 2, we depict the architecture overview of CMD, including the hardware trace perspective and network traffic perspective. (i) Hardware perspective: Consider a running IoT device, in which the Serial Peripheral Interface (SPI) bus can be used to dump content for the Flash chip. We connect the SPI bus in parallel with a logic analyzer that parses the digital signals to the Flash traces. The Flash traces could be used to structure the file system (*e.g.*, JFFS2 [41]) and recover the operating system logs, thereby realizing the host-side analysis. (ii) Network perspective: An upper computer is employed to capture traffic through the device. As a network-side detection, it can quickly detect malicious behavior in communication so as to cooperate with the host-side analysis to tackle the multi-stage IoT malware.

Specifically, we provide an illustrative explanation of detection logic in Figure 3. We feed the network traffic to a tailored capsule network to identify attacks. The IoT device will run normally if the traffic classification result is "benign". Otherwise, the infected sources will be reported to the anti-virus software when the model reports malicious behaviors. Meanwhile, the hardware-part forensic analysis is designed to recover file operation logs, which could provide fine-grained intrusion information (*e.g.*, location and behavior) to benefit the anti-virus countermeasures. Overall, CMD identifies the malicious behaviors of IoT malware lifecycle in a co-analyzed manner from network traffic and hardware traces.

Notably, running hardware trace analysis does not need to wait for the network traffic alarm, instead, both network traffic detection and hardware trace forensics are active before infection (as stated in assumptions of § 3), so as to avoid missing some malicious behavior. Readers may concern about whether the long-term collection will cause the forensic data explosion, which is mentioned in previous research [23], [36], [37]. We clarify here that the hardware part of CMD is designed to focus on recovering file operation commands (such as *wget*, *cat*, and other commands commonly used by malware) and non-volatile data content and file storage locations. Compared to previous work that analyzes causality analysis graphs of system kernel calls, these logs generated by CMD do not take up much space, *e.g.*, the forensic analysis results only occupy ∼0.2 MB space overhead for each malware in § 5.5, which is acceptable.

4.2 Design of Network Part

We introduce here the network-part design used to detect malicious traffic. Our intention is to capture the contextual

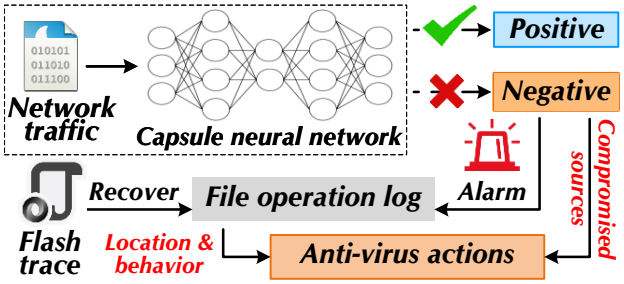


Fig. 3. The high-level detection logic of CMD.

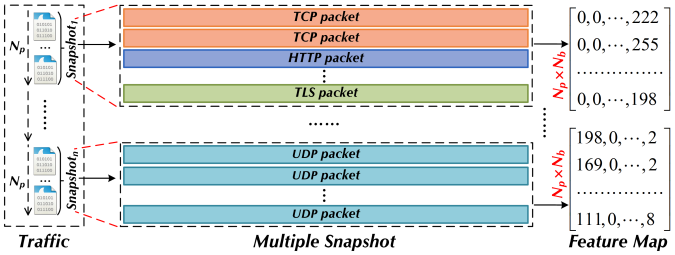


Fig. 4. The cross-source traffic feature map production.

semantics of traffic packets to discover attack behaviors at various stages of the IoT malware.

Traffic Capture and Preprocessing. The protected IoT device with CMD captures the passing traffic (based on *libpcap* library) and mirrors them to the upper computer. Note that the traffic will be temporarily stored in the volatile memory regions (*e.g.*, $\backslash tmp$ in Linux) that will not produce interfering SPI signals in subsequent hardware forensics. On the upper computer, it runs *tshark* API to capture the raw packets. Considering the malicious behaviors could be from the different sessions even various IPs (*e.g.*, P2P broadcast), we manufacture traffic into multiple snapshots in order of arrival timestamp to form a series of feature maps. As Figure 4 shows, the PCAP file will be split according to the number of packets N_p. The last snapshot will be padded with zero if it is not enough N_p packets. We truncate the first N_b bytes in each packet, so that a series of $N_p \times N_b$ (*e.g.*, 512×512) feature matrixes could be generated.

Model Architecture. To characterize the contextual semantics of packets in malicious traffic, we leverage the capsule component to design our model. In Figure 5, the architecture mainly contains five parts, namely input, convolution, primary capsule, label capsule, and fully connected. Specifically, the 512×512 input is concatenated with three

convolutional layers (with the ReLU activation), whose feature maps are $[252\times252\times4]$, $[61\times61\times64]$, $[18\times18\times256]$ respectively. The $C_h\times C_d\times C_c$ output (*i.e.*, $[18\times18\times256]$) of convolution expresses the activities of local feature detectors, and will be fed to the primary capsule layer.

The primary capsule layer includes 8 parallel convolution units whose kernel is 9×9, the stride is 2, and the out channel is set $P_c=32$. It generates $P_h\times P_d\times P_c$ capsule outputs (each output is an 8D vector) and its receptive fields overlap with the location of the capsule center. Particularly, each capsule of the $P_h\times P_d$ grid in the same channel shares the weights.

The label capsule layer maintains one 16D capsule for each classification label. Between the primary and label capsule layers, it is equipped with a routing mechanism (clarifying subsequently) iteratively to build appropriate connection weight. The output of the label capsule layer can be used to classify result prediction, as well as be fed to the fully connected layer for reconstruction (more details in the loss function section).

Dynamic Routing between Capsule Layers. The r-iteration (*i.e.*, $\tau\in\{1,\cdots,r\}$) routing mechanism is shown in Algorithm 1, the output u_i of a primary capsule i will be multiplied by a weight matrix W_{ij} to get the "prediction vectors" $\hat{u}_{ij}$ for a label capsule j, *i.e.*, $\hat{u}_{ij}=W_{ij}\times u_i$. For label j, $s_j^\tau=\sum_i c_{ij}^\tau\hat{u}_{ij}$ refers to the weighted sum over all $\hat{u}_{ij}$ at the τ-th iteration, where the c_{ij}^τ are coupling coefficients between capsule. Particularly, it is calculated as "routing softmax" $c_{ij}^\tau=\mathrm{Softmax}(b_{ij}^\tau)=\frac{\exp(b_{ij}^\tau)}{\sum_k\exp(b_{kj}^\tau)}$, and b_{ij}^τ denotes the log prior probabilities for primary capsule i should be coupled to label capsule j. Therefore, $\sum_k c_{kj}^\tau=1$ is satisfied. Furthermore, the magnitude (*i.e.*, 2-norm) of the label capsules' output vector is associated with the probability of the represented category. Specifically, it will be mapped by a non-linear "Squash" function described in Eq. (1) to ensure that short vectors get shrunk to almost zero, and long vectors get shrunk to a length slightly below 1.

$$a_j^\tau=\frac{\|s_j^\tau\|^2}{1+\|s_j^\tau\|^2}\frac{s_j^\tau}{\|s_j^\tau\|},\|s_j^\tau\|=\sqrt{\sum\nolimits_{k\in[1,L_p]}s_j^\tau[k]^2}\quad(1)$$

Among them, a_j^τ is the squashed vector of capsule j at τ-th iteration. During iteration, the probability b_{ij}^τ will be updated based on the dot product of the current result a_j^τ, thus making the label capsule focus on the prediction vectors $\hat{u}_{ij}$ that coincide with the direction of its output vector.

$$b_{ij}^\tau=b_{ij}^{\tau-1}+a_j^{\tau-1}\cdot\hat{u}_{ij},\ b_{ij}^1=0,\tau\in[1,r]\quad(2)$$

All initial probabilities (b_{ij}^1) are set to zero, so that each primary capsule's output is initially sent to all label capsules with equal probability. After r iterations, each label capsule will output a $1\times L_p$ vector v_j (*i.e.*, $v_j=a_j^r$), which carries potential information from traffic features such as semantics. The predicted label C_{pre} is represented as the capsule with the maximum-magnitude outputs.

$$C_{\mathrm{pre}}=\arg\max_j\mathrm{Softmax}(\|v_j\|)=\arg\max_j\frac{\exp(\|v_j\|)}{\sum_k\exp(\|v_k\|)}\quad(3)$$

Loss Function. In Figure 5, the loss function includes two parts: (i) the reconstruction loss L_r of the fully connected layer and (ii) the sum of the predicted loss L_k for each label capsule k.

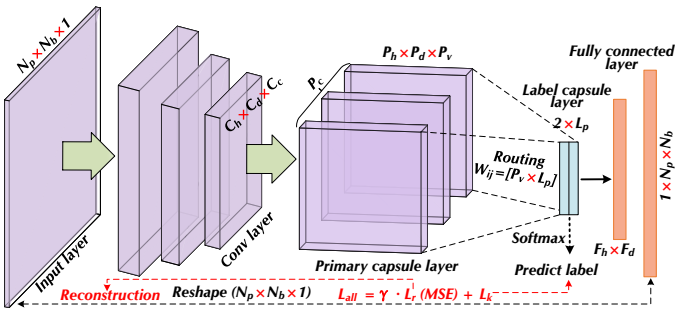


Fig. 5. The architecture of traffic detection model based on capsule network.

Algorithm 1 Function Routing(u_i, r)

Require: Output u_i of primary capsule i, number of iterations τ
Ensure: The predict label C_{pre} of instance X
1: $\hat{u}_{ij}\leftarrow W_{ij}u_i$, $b_{ij}^1\leftarrow0$ for all primary and label capsule i, j
2: **for all** label capsule j **do**
3: **for all** $\tau\in\{1,\cdots,r\}$ **do**
4: $c_{ij}^\tau\leftarrow$ Softmax(b_{ij}^τ) for all capsule i
5: $a_j^\tau\leftarrow$ Squash(s_j^τ) = Squash($\sum_i c_{ij}^\tau\hat{u}_{ij}$) (*c.f.*, Eq. (1))
6: $b_{ij}^{\tau+1}\leftarrow b_{ij}^\tau+a_j^\tau\cdot\hat{u}_{ij}$ for all capsule i
7: **end for**
8: **end for**
9: **return** $C_{\mathrm{pre}}\leftarrow$ Label($\|v_j\|$) = Label($\|a_j^r\|$) (*c.f.*, Eq. (3))

$$L_{\mathrm{all}}=\sum\nolimits_k L_k+\gamma L_r, L_r=\|X,\tilde{X}\|_2=\sqrt{\sum\nolimits_i(X_i-\tilde{X}_i)^2}\quad(4)$$

where $\gamma=5e\text{-}4$ is used to balance the numerical difference between the two losses. The reconstruction loss refers to the ℓ_2 distance between the output $\tilde{X}$ from the fully connected layer and the flattening vector of the input sample X. For the predicted loss, we intend to give a significant magnitude for the output vector of the label capsule if and only if the prediction is the ground-truth label k. Therefore, we set the separate margin loss L_k for each label capsule k:

$$L_k=T_k\cdot\max(0,\mu_0-\|v_k\|)^2+\lambda(1-T_k)\cdot\max(0,\|v_k\|-\mu_1)^2\quad(5)$$

where $T_k=1$ if the inference result is right. The thresholds $\mu_0=0.9$ and $\mu_1=0.1$ refer to acceptable differences to guide the model to more stable and rapid loss convergence. Meanwhile, the $\lambda=0.5$ down-weighting of the loss for misidentified categories stops the initial learning from shrinking the activity vector lengths of all label capsules.

Details in Model Training and Testing. In the model training phase, we slice natively the dataset according to the timestamp to form a series of feature maps for input. Those feature maps that contain malicious behavior are considered positive instances. Some hyperparameter settings reference the $batch\_size=10$, the $epoch$ is set to 50, and the learning rate is 1e-3. During testing, the trained model outputs either "benign" or "malicious" for the test samples. If the inference result is malicious, then the timestamp and IP in the corresponding feature map are recorded, and the information can facilitate anti-virus countermeasures, as discussed in § 4.1.

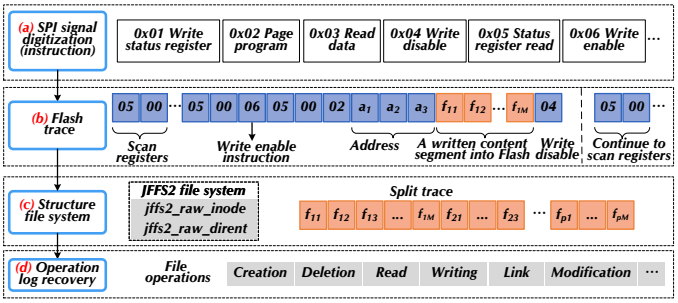


Fig. 6. Processing pipeline of the hardware part.

4.3 Design of Hardware Part

The hardware part is designed to externally recover the operation log in the file system. The so-called "externally recover logs" refers to the external monitor (*e.g.,* leveraging SPI signals), not the internal monitor, which echoes back to the challenge ③ in § 1 (so as to make the adversary imperceptible and tamper-proof). We clarify here the hardware part processing in Figure 6.

Collect the Digital Signals. To acquire digital signals from the Flash chip of the IoT device, we connect the logic analyzer with the pins of the SPI bus in parallel. The logic analyzer lies as a physical probe that mounts on the SPI pins and performs periodic sampling with the frequency F_l. Note that $F_l > 2 \times F_o^{max}$ should be satisfied according to the Nyquist-Shannon sampling theorem, where F_o^{max} denotes the maximum frequency of the original signals. Particularly, according to product reports of the chip manufacturer, the maximum frequency of common SPI Flash chips is 80∼133 MHz [38]. Modern logic analyzers support a sampling rate of 500 MS/s [39], which means that it could achieve 3∼6 times oversampling (satisfies Nyquist-Shannon sampling theorem). We set $F_l = 500$ in experiments, and these sampling data will be saved in the upper computer.

Extract the Flash Traces. The collected digital signals mainly consist of the chip instructions [1] and the content written into Flash (*i.e.,* the non-volatile data). The chip instructions are related to the chip type and the physical state. For instance, the W25Q128FV [40] chip we used has read status register instructions shown in Figure 7. The instruction is entered by driving $/CS$ low and shifting the instruction code "05h" for Status Register-1, "35h" for Status Register-2, or "15h" for Status Register-3 into the DI pin on the rising edge of CLK. The status register bits are then shifted out on the DO pin at the falling edge of CLK with the most significant bit (MSB) first, as shown in Figure 7.

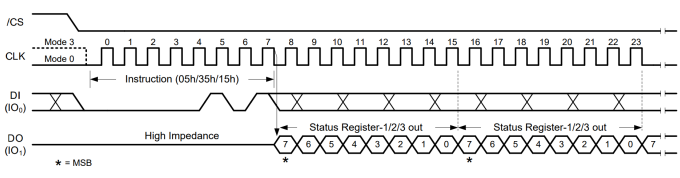


Fig. 7. The write status register-1/2/3 instruction.

1. The chip instructions here refer to the SPI instruction set and are fully controlled through the SPI bus, *e.g.,* the W25Q128FV contains 45 basic instructions [40].

In general, we focus on the operations involving reading, writing, etc., and some corresponding instruction hexadecimal of W25Q128FV are as follows:

- $0x01$: Write Status Register Instruction
- $0x02$: Page Program Instruction
- $0x03$: Read Data Instruction
- $0x04$: Write Disable Instruction
- $0x05$: Read Status Register Instruction
- $0x06$: Write Enable Instruction

These chip instructions can help us analyze the SPI trace, and we can determine the performed operations (such as "Read" and "Write"), as well as the written non-volatile data content and corresponding address. As shown in Figure 6(b), we can extract the Flash traces, *e.g.,* $< a_1, a_2, a_3 >$ represents the 24-bit address and the following $< f_{11}, f_{12}, \cdots, f_{1M} >$ refers to the written content segment into Flash. Particularly, these hexadecimal SPI instructions are related to the chip product number, so the specific instruction hex can be confirmed according to the corresponding chip manual.

Structure the File System. Considering the write operations could be in arbitrary positions due to the Linux blocks implementation in the kernel, we will structure the file system to assemble segments in this step. Take the Journalling Flash File System (JFFS2) [41] log structure as an instance [2], which is an append-only file system. JFFS2 system has mainly two data entities closely related to file operations, named $jffs2\_raw\_dirent$ and $jffs2\_raw\_inode$ as Figure 6(c).

We can distinguish these two entities according to parameter $magic+nodetype$, and determine the entity length based on parameter $totlen$. Among them, $jffs2\_raw\_dirent$ can describe the file location (parent directory), and $jffs2\_raw\_inode$ stores the file management information. Particularly, $jffs2\_raw\_inode$ carries the written content of Flash in its parameter $data$ and uses $mode$ to record the file types & mode. For example, {S_IXOTH: 01} denotes the execute or search permission bit for other users, and {S_IWOTH: 02} denotes the write permission bit for other users [43]. Based on these two entities, we mount corresponding nodes to structure the tree-shaped file system. Note that the content might be compressed to reduce space overhead as shown in Figure 8, we execute the corresponding decompression algorithm (stored in the $compr$ of $jffs2\_raw\_inode$) to get the complete data. For example, {LZO: 0x07}[3] is the default compression algorithm in our testbed that is very common and popular given its quite competitive compression ratio and extremely fast decompression [44].

2. We choose JFFS2 because it is designed from the outset for embedded devices, and provides a robust file system to allow reliable use of Flash devices as data storage [41]. For example, JFFS2 allows recovery when the system has failed abnormally, without the file system itself being left in an unusable state, even if power is disconnected at the moment the Flash device is in the middle of being written to. Meanwhile, JFFS2 is also widely used in IoT devices. Such as [42] mentioned that 333 firmware was collected from Axis Communications (a network camera device manufacturer), and about 85% of them use the JFFS2 file system. In addition, if other file systems are used, it only needs to change the mapping process of the storage address to the file directory and the decompression algorithm of the non-volatile data.

3. LZO is a portable lossless data compression library written in ANSI C. It offers pretty fast compression and extremely fast decompression. We can download the LZO executable [44] and call the interface to perform data decompression.

```
#define JFFS2_COMPR_NONE       0x00
#define JFFS2_COMPR_ZERO       0x01
#define JFFS2_COMPR_RTIME      0x02
#define JFFS2_COMPR_RUBINMIPS  0x03
#define JFFS2_COMPR_COPY       0x04
#define JFFS2_COMPR_DYNRUBIN   0x05
#define JFFS2_COMPR_ZLIB       0x06
#define JFFS2_COMPR_LZO        0x07
```

Fig. 8. The compression algorithm and corresponding hexadecimal.

Recover the Operation Logs. After structuring the file system, we will obtain information such as the file type, name, permission, content, access time, etc. Then these available nodes' information can be used to recover the operating system behavior logs. For example, *emerging instances of entities → file creation*; *changing the entities' contents → file writing*; *mounting the entities to the garbage collection node → file deletion*. Finally, these recovered logs include file creation, deletion, read, writing (including content), permission modification, last modification time, soft link, etc.

Readers could concern that whether the log recovery process will be affected if adversaries delete the binary scripts (*i.e.,* the self-induced malicious script deletion mentioned in § 2). As long as SPI signal collection is run in advance, CMD can recover the operation log even if the malicious script is deleted. This is because the CMD's hardware part is designed as an external monitor, and all file operations (*e.g.,* reading, writing, permission modification, deletion, *etc.*) will be recorded and saved in the upper computer (for log recovery).

5 Evaluation

In this section, we comprehensively evaluate CMD in terms of the detection performance and operation recovery forensics. For the network part, we compare SOTA methods, as well as provide deep insights of interpretability from the aspect of traffic behavior. For the hardware part, we evaluate the command recovery effect and depict the file operation portrait. Moreover, we conduct the integration test for CMD and the overhead evaluation. Finally, a typical case study for Hajime will be introduced. Our code is available online[4]. Overall, the experiments are designed to answer the following research questions:

RQ1. How effective is network traffic detection of CMD compared to SOTA methods (§ 5.2)?

RQ2. Against adaptive adversaries, does hardware trace forensics of CMD work (§ 5.3)?

RQ3. Considering the multi-stage attack, what is the integration test effect of CMD (§ 5.4)?

RQ4. How much is the overhead of CMD on the IoT device and the upper computer respectively (§ 5.5)?

RQ5. Can CMD analyze a typical malware example to provide more insights (§ 5.6)?

5.1 Experiment Setup

Testbed. Our testbed is established with a wireless router (installed the W25Q128FV Flash chip [40], SPI bus, and MT7620 SoC chip that is MIPS-EL architecture), a logic analyzer (the Saleae Logic Pro 8 [39], supports a maximum sampling rate of 500 MS/s), an upper computer (installed an i7-9700 CPU, a GTX 1070 GPU, and 64 GB memory), and three client hosts. Among them, the router is the victim IoT device, and the upper computer runs CMD. One end of the logic analyzer is probed on pins 1, 2, 4, 5, and 6 of the Flash chip (Figure 9), while the other end is connected to the upper computer. In addition, the three hosts act as benign users and attackers alternately with a period. Particularly, the wireless router runs OpenWrt [45] which the file system is JFFS2.

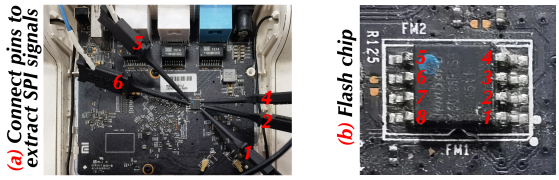


Fig. 9. The physical IoT device of CMD testbed.

Datasets. The datasets used for evaluation are summarized in Table 1, including network traffic from IoT-23 [46] and binary executables from BadThings [9]. (i) Network traffic: the IoT-23 captures malicious traffic when running common malware such as Mirai, Hajime, Gafgyt, etc. It contains the traffic of scanning, brute-force login, file downloading, C2, DDoS, and so on. Together with benign flows generated by three different IoT devices: a Philips HUE smart LED lamp, an Amazon Echo home intelligent personal assistant, and a Somfy smart Doorlock. All these IoT devices are physical hardware and allow to capture real-world communication behavior on an underlying network with unrestricted connections. In addition, we also enrich benign background traffic by bringing legitimate instances from the B-Profile system[5] of IDS2017 [47].

(ii) Binary executables: the BadThings possesses more than 166K malware samples collected across a series of popular architectures, *e.g.,* ARM, PPC, SH4, *etc.* Among them, 15,906 binary executables of MIPS-EL architecture are used for evaluation since our testbed is MIPS-EL architecture. These samples exist in the form of executable files (*e.g.,* ELF) that we can run to perform the specific infection process. The command quantities involved in each malware binary are obtained by reverse engineering [9].

TABLE 1
The dataset of IoT malware used in our evaluation.

IoT-23	
Benign	Soomfy Doorlock, Phillips HUE, Amazon Echo
Malware	Mirai, Torii, Trojan, Gafgyt, Kenjiro, Okiru, Hakai, IRCBot, Hajime, Muhstik, Hide and Seek
BadThings	
IoT malware corpora that contain binary executables	

Baselines. Some state-of-the-art (SOTA) models are briefly introduced as follows: (i) Mousika [48] proposes the binary decision tree that is translated from the deep neural network with knowledge distillation. (ii) FlowLens [49]

4. See code repository on https://github.com/CMD-IoT/CMD/

5. Including multiple protocols (TCP, UDP, ICMP, etc.) and multiple applications (FTP, HTTP, DNS, NetBIOS, Mail, SNMP, SSH, etc.).

devises a compact representation of packet length and adjusts the granularity of the flow's frequency distribution intervals to classification with machine learning (ML) models. (iii) Whisper [50] utilizes sequential information based on the frequency domain features to detect malicious traffic. (iv) Kitsune [14] discovers abnormal behavior by using AutoEncoder to examination on each packet. (v) FS-Net [51] uses the bi-GRU model to learn sequence features and classify common applications. (vi) ET-BERT [52] handles the raw packets in hexadecimal and deploys a pre-trained transformer to represent and learn the contextualized datagram-level information. (vii) FlowPic [53] processes packet length and timestamp fields and converts them into pictures and uses Convolutional Neural Networks (CNNs) to identify traffic.

Additionally, we also explore the advantages of CMD's hardware trace forensics by comparing to the internal monitor. Specifically, we choose the representative internal monitor $Syslog$ [54] which is a commonly used management tool for system logs. Meanwhile, many existing forensic schemes leverage $Syslog$ to collect system logs [55]–[58].

5.2 Network Traffic Detection Effect (RQ1)

Compare with SOTA. We first compare the detection effect of CMD with a series of SOTA models from ML and deep neural network (DNN). In the experiments, the dataset division ratio refers to $train$: $test$ = 7:3, and the per-group division & experiment will be randomly performed 10 times. The results are summarized in Table 2, including accuracy Acc, precision Pre, recall Rec, F1-score $F1$, and the Area Under Curve (AUC). The overall detection effect, in terms of the Acc and $F1$, is DNN-based > ML-based methods. For ML-based approaches, Mousika shows more accuracy loss since it mainly focuses on per-packet characterization. And Whisper, as a frequency domain analysis-based model, performs better results than FlowLens, *e.g.*, Acc of the former is 97.74% and the latter is 97.04%. Meanwhile, Whisper and Kitsune are evenly matched in accuracy and F1-score, *i.e.*, a ∼0.07% difference. In five DNN-based models, the detection effect CMD outperforms others in terms of various metrics. For example, the F1 scores of CMD, Kitsune, FS-Net, ET-BERT, and FlowPic are 99.88%, 97.79%, 98.74%, 99.42%, and 97.82% respectively.

Moreover, we plot the Receiver Operating Characteristic (ROC) curve in Figure 10. It is clear that CMD has fewer false positives while achieving a higher true positive rate. For the Area Under Curve (AUC) metric in Table 2, the models refer to CMD > ET-BERT > FS-Net > Kitsune > FlowPic > Whisper > FlowLens > Mousika. ET-BERT is a competitive baseline given it is based on a pre-trained large model, while it needs more inference time as stated in § 5.5. Particularly, FlowPic also converts traffic to pictures and leverages the CNN model for classification, but there is still some performance gap between FlowPic and the capsule network of CMD. This can be attributed to the rich representation of extracted feature maps and the dynamic routing of capsule networks to capture contextual semantics. We next provide some deep insights into network traffic semantic identification.

Deep Insights for Interpretability. To further understand how CMD identifies these attack behaviors based on features, we use the Integrated Gradients method [59] from the Captum [60] to generate the attribution matrix to provide the model interpretability. Figure 11 shows four behavior instances involving Gafgyt, Hajime, and Mirai three widespread malware. We provide more interpretability instances of identification with attribution maps in our online repository (the aforementioned URL).

In subfigure (a), we find that CMD captures a series of snippets for ARP sniffing in Gafgyt's traffic, *e.g.*, around 187-th, 227-th, 311-th, 351-th packets. From subfigure (b), it is clear that exists a very obvious traffic slice for the UDP data field in the range 277-th∼298-th. Specifically, the IP 192.168.100.111 sends multiple DHT broadcast packets by Hajime and gets a response from 210.103.70.224. As shown in Figure 12, the 288-th packet is a request for peer host acquisition "get_peer", the message refers to {"**id**": "..E.....^6..¿*.+..^", "**info_hash**": ".*G.bo)CN...+Z.?.x#."}. And the 295-th packet responses the message with {"**id**": ".*.o.....KL..$].'.", "**nodes**": ".*M.:)P...L0O." (omitted, 208 bytes in total), "**token**": "[..-_I.1F.....k."}, it indicates to return the closest node.

Figure 11(c) displays the brute-force login process of Gafgyt, the encrypted packets of SSH and TCP appear alternately thus the corresponding attribution map presents intermittently salient areas, and finally, the 173-th SSH packet achieves successful login. For Figure 11(d), the attribution matrix presents sporadic significant pixels corresponding to the TCP payload positions that record the transferred bytes during download. Particularly, the 131-th packet means the HTTP GET request, and subsequently, the multiple response segments contain the related binary content. Noteworthy that we find that Mirai will disguise itself as a system file, such as $ntpd$, $sshd$, $slav.x86$, $slav.mips$, and $slav.arm7$ in Figure 13.

Overall, the capsule network design realizes extracting

TABLE 2
The traffic detection effect (%) of CMD and seven SOTA models.

Type	Model	Acc	Pre	Rec	$F1$	AUC
ML	Mousika	96.31	96.89	95.60	96.24	96.63
	FlowLens	97.04	97.62	96.31	96.96	97.37
	Whisper	97.74	97.89	97.56	97.72	97.43
DNN	Kitsune	97.81	97.90	97.69	97.79	97.95
	FS-Net	98.76	98.89	98.60	98.74	98.83
	ET-BERT	99.41	99.13	99.68	99.42	99.52
	FlowPic	97.83	97.92	97.73	97.82	97.86
	CMD	**99.87**	**99.84**	**99.91**	**99.88**	**99.92**

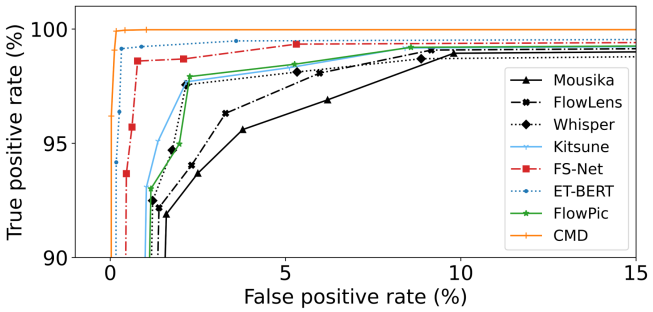


Fig. 10. The ROC results of network traffic detection.

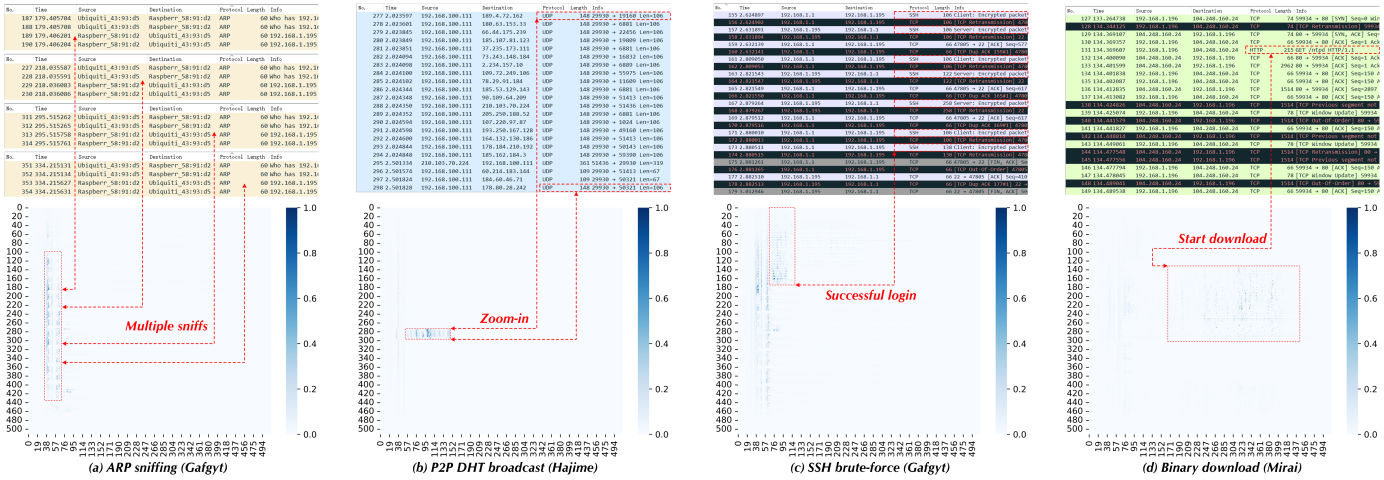


Fig. 11. Attribution maps are based on the network-part model in CMD. The captured contextual semantics from traffic involved four typical behaviors.

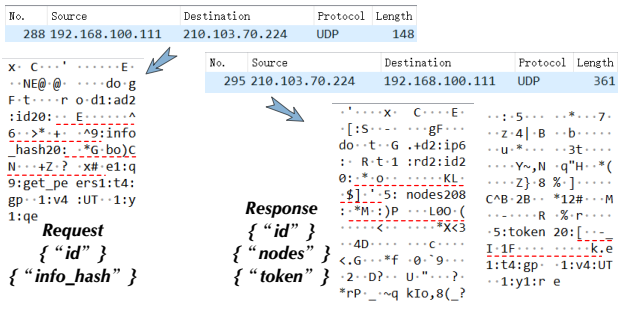


Fig. 12. The DHT message of P2P broadcasts in Hajime.

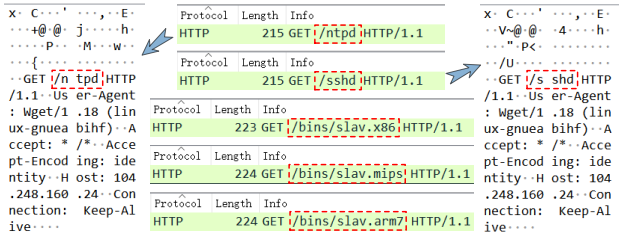


Fig. 13. The HTTP message for binary downloading in Mirai.

the contextual semantics from the traffic feature map even in cross-source scenarios (*e.g.*, decentralized DHT). The above experimental observations cater to our design intentions for the network part of CMD.

5.3 Hardware Trace Forensic Evaluation (RQ2)

Malware Command Recovery. In the aspect of hardware traces, we record the recovered file operations including specific commands and corresponding quantities. We use ∼15K malware of MIPS-EL architecture to conduct and capture the SPI traces to recover operation logs. Meanwhile, *Syslog* [54] is deployed as an internal monitor to record operations since many previous forensic schemes are based on it [55]–[58]. To evaluate the tamper-proof capabilities of CMD and *Syslog*, we consider adaptive attackers in two types of settings. For one thing, the attacker directly kills the internal monitor process. This is very possible, given that the adversary could have advanced privileges on compromised IoT devices and then typical malware does perform a series of process kill operations [61], *e.g.*, Hajime could involve *killall /sbin/telnetd*, *pkill -9 cron*, *etc.* operations [62]. For another, the attacker tampers with operation log files which are mentioned in many existing works [63], [64]. Specifically, we only save the first 50 lines of generated logs after each malware execution to simulate adversary tampering, *e.g.*, *cat /dev/null > /var/log/syslog*, execute the malware binary, and *cat /var/log/syslog | head -n 50 > save.txt*.

The results are shown in Table 3. For common seven commands involved in malware (*i.e.*, *wget*, *chmod*, *echo*, *cp*, *cat*, *mv*, *rm*), CMD all realizes 100% recovery by the hardware-part design. In other words, if the malware calls the corresponding command, CMD will perceive in the forensic analysis. About the number of each command, the recovery results by CMD vary from ∼97% to ∼99%. When the adversary kills the internal monitor process (the top part of Table 3), *Syslog* cannot record any operations while CMD is not affected. Specifically, CMD's recovery results of *wget*, *chmod*, and *rm* are relatively high, achieve 99.69%, 99.15%, and 99.75% respectively. Although the effect of *cp* and *cat* is slightly inferior, the results are still more than 97%. When the adversary tampers with the operation log (the bottom part of Table 3), *Syslog* can record part of logs, *e.g.*, the recovery of the per-command quantity is 45%∼80%. Actually, recovery result by *Syslog* is related to the adversary's tampering strategy, for example, if the attacker deletes more log entries, the recovery rate of *Syslog* will be lower. For CMD, log tampering slightly changes the recovery result, mainly including *wget*, *echo*, *cp*, and *cat* given that the adversary's tampering process may involve these commands. Nevertheless, the impact is less than 0.5%. Overall, CMD is tamper-proof compared to typical internal monitors when an adversary may kill processes or modify logs, this echoes our original design intention stated in § 2.

File Operation Portrait. Moreover, we portray the proportions and relationships of these commands in Figure 14. It is a Sankey diagram in which the depth sequence references *create* → *write* → *read* → *chmod* → *link* → *move* → *delete*. First, it is clear that a large fraction (*i.e.*, ∼96.7%) of

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3311012

TABLE 3
The command recovery results against adaptive adversaries. The entry "Recovery" calculates whether each type of command is recovered in the corresponding malware, and the entry "Quantity" represents the proportion of the recovered command quantity.

Kill the internal monitor process								
Operation		*wget*	*chmod*	*echo*	*cp*	*cat*	*mv*	*rm*
CMD	Recovery	100%	100%	100%	100%	100%	100%	100%
	Quantity	99.69%	99.15%	97.32%	98.07%	97.62%	98.34%	99.75%
Syslog	Recovery	0%	0%	0%	0%	0%	0%	0%
	Quantity	0%	0%	0%	0%	0%	0%	0%

Tampering with the log file								
Operation		*wget*	*chmod*	*echo*	*cp*	*cat*	*mv*	*rm*
CMD	Recovery	100%	100%	100%	100%	100%	100%	100%
	Quantity	99.63%	99.15%	96.88%	97.95%	97.13%	98.34%	99.75%
Syslog	Recovery	91.52%	71.30%	83.27%	88.65%	90.72%	68.90%	74.29%
	Quantity	76.85%	62.43%	75.67%	71.82%	69.80%	51.44%	46.07%

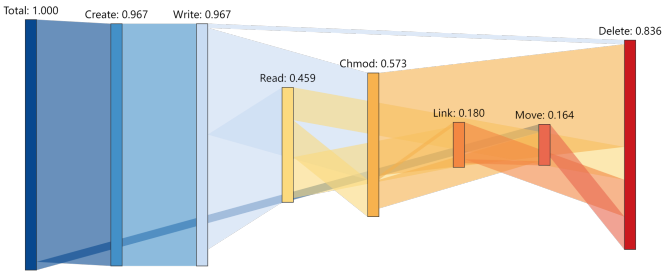


Fig. 14. Sankey diagram for file operations.

malware will create and write files. After writing the file, ∼42.6% malware will use *chmod* to modify permissions (*i.e.,* a part of 57.3%, and another part is after the read), some malware (45.9%) will call read file to further production or obfuscation, and a fraction of the rest could perform the delete operation (1.6%) or nothing (6.6%). Noteworthy that a large percentage of malware will execute file deletion to hide and destroy traces. This is one of the reasons why IoT malware is tricky for security practitioners. We provide more operation details by using the case of Hajime in § 5.6.

5.4 Integration Test for Multi-Stage Attacks (RQ3)

In this section, we conduct the integration test with the multi-stage process that mimics real-world IoT attacks. It considers 6 stages (4 malicious + 2 benign) in which the adversary deploys dynamic malware intrusion strategies. Specifically, in **Stage 1**, the users perform a series of normal routine operations (including disconnecting the wireless, connecting into the wireless, operations of automated browsers by Selenium [65]). In **Stage 2**, the administrator modifies the configuration file (*e.g.,* DNS) of the IoT device with authentication. In **Stage 3**, the device suffers some scanning and sniffing (*e.g.,* PortSpider [66]). In **Stage 4**, the attacker adopts brute-force login to password cracking (*e.g.,* Hydra [67]). In **Stage 5**, the adversary implants (via FTP transmission) the malware scripts or binaries into the device after successful login. In **Stage 6**, those malicious files will be executed on the IoT device.

Figure 15 depicts the accuracy curve with the dynamic attacks. Overall, CMD achieves ∼99.9% Acc that outperforms the performance of only using network traffic detection given the hardware trace forensics could facilitate results fine-tuning. For instance, when the network part model incorrectly reports a legitimate HTTP download as malicious,

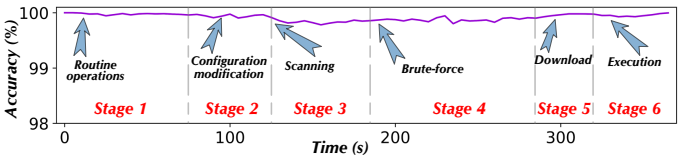


Fig. 15. The dynamic performance of CMD on multi-stage IoT attacks.

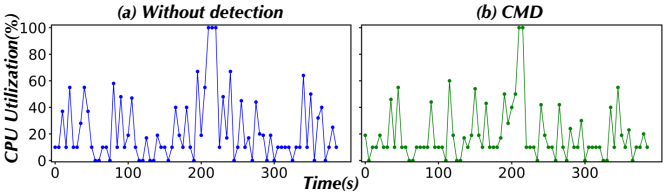


Fig. 16. The CPU utilization of the IoT device.

the hardware part analysis further validates the downloaded sample and conducts forensics. If the forensics result is benign[6], this false positive will be corrected and thus improve the integration accuracy.

In stage 1, there is almost no false positive. While in stage 2 it emerges some slight fluctuation (∼0.1%) due to some DNS configuration file modifications. For stages 3 and 4, CMD could stably detect >99.7% sniffing, scanning, and brute-force behaviors. However, the identification effect shows some improvement in stages 5 and 6 since the corresponding file operations are verified by SPI traces. In general, our co-analyzed design from network and hardware domains could cope with the multi-stage malware lifecycle and bridge the detection and forensics.

5.5 Overhead Evaluations (RQ4)

To further analyze the overhead introduced by CMD, we measure here the CPU utilization and time/space overhead.

CPU Utilization of IoT Device. Figure 16(a) shows the scene (including benign and malicious behaviors) without detection whose CPU utilization is universal in 10%∼70%. Particularly, the CPU utilization reaches about 100% at $t = 210$ due to suffering brute-force password cracking. Figure 16(b) refers to that uses CMD to detect malware and forensic analysis, and its CPU curve is very similar to Figure 16(a) that indicates CMD introduces negligible CPU utilization.

The Overhead of Network Traffic Detection. For the network traffic detection part of CMD and seven baseline models (§ 5.2), we measure model inference time. Note that they are all running on the upper computer instead of IoT devices. As illustrated in Figure 17, the inference time of the ML-based model is indeed less than DNN-based. Three ML-based methods take about $<1ms$. Among the DNN-based models, CMD and FlowPic require the least inference time ($\sim$2.48ms), while ET-BERT has the most time overhead, about 6.83ms. Considering that it usually has a long interval between the infection phase and attack phase of IoT malware, the ms-level detection latency is acceptable. Such as Mirai infection is used to create a large-scale botnet to aggregate

6. For example, if the downloaded sample does not contain any *chmod*, *cat*, *cd* commands, *etc.*, it is most likely not malware [61].

the ability for bandwidth overflow, while the attack launch depends on the specific purpose. Therefore after detection and forensics, we could perform anti-virus countermeasures when the compromised device listens to command & control commands. In addition, for space overhead, ET-BERT is also the largest, requiring >500 MB, while the capsule network of CMD is about 30 MB, which is acceptable since the model is stored in the upper computer instead of the IoT device.

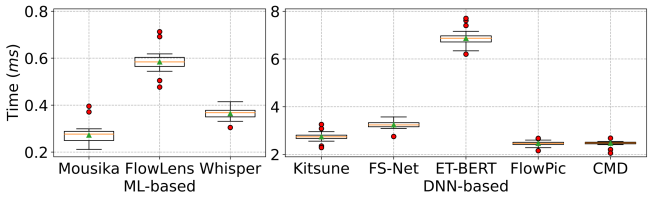


Fig. 17. Inference time for network traffic detection.

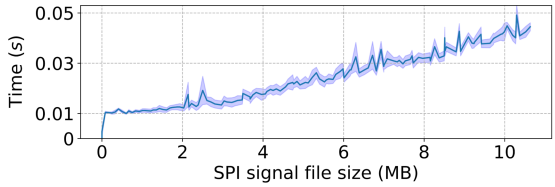


Fig. 18. Log recovery time of the hardware trace forensics in CMD.

The Overhead of Hardware Trace Forensics. For the overhead of the hardware forensics part, we plot the time *vs.* SPI signal file size in Figure 18. The log recovery time is approximately linear with the file size, and the slight fluctuation is because the compression ratio of the algorithm may be different for diverse file contents. Specifically, it takes about $0.04s$ to analyze 10 MB SPI signals, which is acceptable due to usually there are not so many traces generated (see subsequent explanation). We also measure the space utilization for recovered logs (given previous work mentions the data explosion problem [23], [36], [37]), and statistical results for § 5.4 are shown in Table 4. In stage 1, we find that no SPI trace is generated. While appearing about additional 30.45 KB of space utilization when the IoT configuration files are modified in stage 2. This means that routine operations that do not involve file operations hardly incur extra space overhead. For scanning and brute-force login, there is also no operation log required to analyze in stages 3 and 4. When downloading and executing the binary, it will produce corresponding SPI signals. The last two columns in Table 4 record the occupied space by 10 malware on average. It needs ∼793.82 KB and ∼1801.96 KB of space in stages 5 and 6. In previous works for forensic analysis [21], [37] whose extra space is generally more than dozens of MB, such a small space overhead for recovered logs in CMD (∼0.2 MB per malware) is acceptable.

TABLE 4
The space overhead of the recovery log in each stage of § 5.4, the last two columns record occupied space on average for every 10 malware.

Stage	S_1	S_2	S_3	S_4	S_5	S_6
Size (KB)	0	30.45	0	0	793.82	1801.96

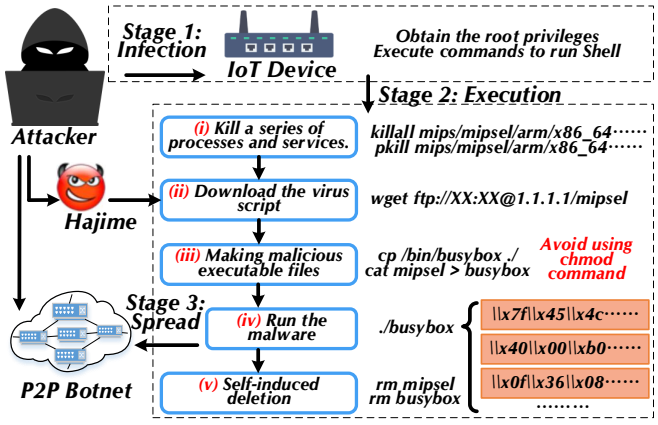


Fig. 19. Log recovery case study of Hajime attack.

5.6 Case Study of the Operation Logs Recovery (RQ5)

Motivated by the enormous potential threat of Hajime [11] that is spread based on P2P protocol, we elaborate here on a case study for its file operation recovery. As shown in Figure 19, the attack process of Hajime can be divided into three stages: infection, execution, and spread. (i) In the infection stage, the attacker will implant the loader program after successful login. The brute-force login process and script transmission can be detected by the network-part model of CMD (given the assumption of enable CMD before infection in § 3). And the content of the loader program can be recovered by hardware-part forensics when it is written in the chip.

(ii) In the execution stage, it will run the pre-designed commands consisting of closing the port, downloading the binary, etc. Among them, CMD can forensics to the file creation and writing ($wget$), file creation and permission setting (cp), file writing (cat), script content ($busybox$), and file deletion (rm). Particularly, when making the malicious executable file, it adopts $cp + cat$ to avoid using $chmod$ command (existing the exposed risk). Nonetheless, these operations will be tracked by CMD given they trigger corresponding SPI signal generation. (iii) In the spread stage, a series of scanning and sniffing behaviors are conducted that can be detected in CMD based on network traffic. Overall, almost all stages of IoT malware can be detected by CMD, which benefits security practitioners to adopt early countermeasures before the attack is launched.

6 Discussion

Robustness and Extensibility. We admit that it exists some imperfections in the ML model such as misclassification, yet CMD would mitigate these problems since its co-analyzed design. When customers have a low tolerance for false negatives, we can improve the positive probability of the classification or perform a model ensemble (*e.g.*, voting result is negative only if no positive), thereby more instances will be verified by the hardware-part operation recovery. The high-level detection logic in CMD can be extended according to specific business scenarios. For example, we could increase the priority of hardware-part forensic analysis for security applications that centered on file operations, *i.e.*, alarm immediately as soon as appear suspect SPI signal.

Limitations and Future Works. Our work has a few limitations. (i) First, the SPI signal collection of CMD needs the instrument for the side-channel-manner analysis, and a feasible direction is to devise integrated customized solutions for industrial production. (ii) Then, as part of future work, we would explore which traffic could be analyzed in parallel to improve efficiency in multi-device scenarios.

(iii) In addition, hardware trace collection and operation log recovery may also be combined with existing solutions based on the causality analysis graph [21]–[25]. For example, a variant of CMD may be to design the hardware part as an independent detection module that can identify malicious operation logs. In this way, the detection results of the network part and the hardware part can be mutually verified.

(iv) Finally, the hardware part of CMD focuses on the on-chip file operations given the existing persistence technologies in the IoT malware landscape nowadays. For those malware that infect memory, on the one hand, as long as the infection process involves file operations on ROM, corresponding SPI signals can be captured by CMD. And for malware that infects RAM, file operations in ROM are common during the intrusion process. For example, Mirai [32] needs to use cat to analyze the architecture, and use $wget$, $echo$, *etc.* to transfer the payload (mentioned in § 2). On the other hand, combining some memory detection approaches [68]–[70] could further enhance IoT protection.

7 Related Work

We explain several SOTA baseline models in § 5.1 and here introduce briefly some other related work.

Network Traffic Detection. To profile the traffic pattern, some works [53], [71]–[74] design supervised learning methods based on statistical features, *e.g.,* random forests. And some other arts utilize Markov [75], [76] or recurrent neural networks [77]–[79] to portray the sequential features (*e.g.,* packet length sequence) of attacks. For the IoT devices, Gu et al. [12] present IoTGaze to discover the threats by sniffing event interaction in wireless traffic. Wang et al. [80] perform a cross-analysis for mobile companion apps to evaluate IoT devices' security. Wan et al. [13] introduce IoTArgos which characterizes the communications data based on TCP/IP and IoT protocol stacks. CMD is more concerned with capturing contextual semantics from the cross-source traffic feature map.

Hardware-Based Solutions. The hardware-based methods aim to extract the processor information from different programs running on the devices. Some previous arts [17]–[19], [81] propose to detect malware based on Hardware Performance Counters (HPC) runtime information. The hardware part design of CMD is different from the HPC-based method in the following two aspects. For one thing, the previous methods use HPC to detect malware, which is the classification problem. Our hardware-part design focuses on recovering operation logs for forensic analysis. Log recovery using HPC could be difficult because the available events are very limited (*e.g., cpu-cycles*, *cpu-clock*, *L1-dcache-loads*, *LLC-prefetch-misses*, *etc.*) and the count results are usually numeric variables. For another, more importantly, the process (*e.g.,* $perf$ tool [82]) for statistics HPC can be killed by adaptive adversaries like that the attacker kills internal monitors (mentioned in § 5.3). In addition, the number of available HPCs is limited on today's microprocessors. And HPCs rely on the OS kernel, *e.g.,* the measured results inside a VM differ from the hardware. Recently, PREEMPT [83] re-purposes the Embedded Trace Buffer (ETB) to identify malware. Costin et al. [84] conduct a large-scale firmware analysis for embedded devices. Different from these methods, CMD introduces a new direction that utilizes side-channel SPI signals to recover logs.

Forensic Analysis. For forensic analysis and attack investigation, a series of causality analysis methods [21]–[25] are proposed based on the system events or log auditing. Meanwhile, some technologies such as data compaction [36] and alternative tag propagation semantics [37] are presented to combat dependence explosion in long-term monitoring. Moreover, Wang et al. [15] design a selection algorithm to identify malicious parts based on the OS-level provenance. IoTGuard [26] implements a code instrument to collect the app's information at runtime by adding extra logic. Pagani et al. [85] suggest using automated algorithms to evaluate and design memory forensics techniques. The forensics of CMD is specially designed for the IoT malware, as an external monitor, that cannot be perceived by adversaries and has the advantage of being tamper-proof.

8 Conclusion

In this paper, we propose CMD, a novel IoT malware detection and forensics system combining the network and hardware domains. For the network part, CMD presents a tailored capsule neural network to capture the cross-source contextual semantics from the traffic feature map. For the hardware part, CMD leverages the SPI bus to monitor the on-chip traces to recover file operations in a side-channel manner, such an external monitor is tamper-proof against adaptive adversaries. CMD realizes almost the whole-lifecycle detection that benefits security practitioners to adopt anti-virus countermeasures before the attack is launched. Our experiments demonstrate that CMD could achieve remarkable detection results, take up limited space overhead for recovered logs, and only introduce negligible CPU utilization in the IoT device. We also provide deep insights for model interpretability and malware operation portrait, as well as a case study for Hajime. Our research bridges the hardware and network perspectives which advances the multi-stage tracking of IoT malware lifecycle.

Acknowledgments

We are grateful to the reviewers for their very constructive feedback and insightful comments.

References

[1] E. Cozzi *et al.*, "Understanding linux malware," in *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2018, pp. 161–175.
[2] D. Kumar *et al.*, "All things considered: An analysis of iot devices on home networks," in *USENIX Security Symposium*. USENIX, 2019, pp. 1169–1185.
[3] B. Vignau, R. Khoury, S. Hallé, and A. Hamou-Lhadj, "The evolution of iot malwares, from 2008 to 2019: Survey, taxonomy, process simulator and perspectives," *J. Syst. Archit.*, vol. 116, p. 102143, 2021.

[4] D. D. Chen *et al.*, "Towards automated dynamic analysis for linux-based embedded firmware," in *NDSS*. The Internet Society, 2016, pp. 1:1.1–8.1.

[5] C. Fu, Q. Zeng, and X. Du, "Hawatcher: Semantics-aware anomaly detection for appified smart homes," in *USENIX Security Symposium*. USENIX, 2021, pp. 4223–4240.

[6] T. Abera *et al.*, "Invited - things, trouble, trust: on building trust in iot systems," in *DAC*. ACM, 2016, pp. 121:1–121:6.

[7] A. Sadeghi *et al.*, "Security and privacy challenges in industrial internet of things," in *DAC*. ACM, 2015, pp. 54:1–54:6.

[8] S. Ahn *et al.*, "Hawkware: Network intrusion detection based on behavior analysis with anns on an iot device," in *DAC*. ACM, 2020, pp. 1–6.

[9] O. Alrawi, C. Lever *et al.*, "The circle of life: A large-scale study of the iot malware lifecycle," in *USENIX Security Symposium*. USENIX Association, 2021, pp. 3505–3522.

[10] M. Antonakakis *et al.*, "Understanding the mirai botnet," in *2017 26th USENIX Security Symposium*. USENIX Association, 2017, pp. 1093–1110.

[11] S. Herwig *et al.*, "Measurement and analysis of hajime, a peer-to-peer iot botnet," in *NDSS*. The Internet Society, 2019.

[12] T. Gu *et al.*, "Iotgaze: Iot security enforcement via wireless context analysis," in *INFOCOM*. IEEE, 2020, pp. 884–893.

[13] Y. Wan, K. Xu, G. Xue, and F. Wang, "Iotargos: A multi-layer security monitoring system for internet-of-things in smart homes," in *INFOCOM*. IEEE, 2020, pp. 874–883.

[14] Y. Mirsky *et al.*, "Kitsune: An ensemble of autoencoders for online network intrusion detection," in *NDSS*. The Internet Society, 2018.

[15] Q. Wang, W. U. Hassan *et al.*, "You are what you do: Hunting stealthy malware via data provenance analysis," in *NDSS*. The Internet Society, 2020.

[16] M. Graziano *et al.*, "Needles in a haystack: Mining information from public dynamic analysis sandboxes for malware intelligence," in *USENIX Security Symposium*. USENIX Association, 2015, pp. 1057–1072.

[17] N. Patel *et al.*, "Analyzing hardware based malware detectors," in *DAC*. ACM, 2017, pp. 25:1–25:6.

[18] H. Sayadi *et al.*, "Ensemble learning for effective run-time hardware-based malware detection: a comprehensive analysis and classification," in *DAC*. ACM, 2018, pp. 1:1–1:6.

[19] J. Demme, M. Maycock, J. Schmitz, A. Tang, A. Waksman, S. Sethumadhavan, and S. J. Stolfo, "On the feasibility of online malware detection with performance counters," in *ISCA*. ACM, 2013, pp. 559–570.

[20] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2010, pp. 305–316.

[21] R. Yang, S. Ma, H. Xu, X. Zhang, and Y. Chen, "Uiscope: Accurate, instrumentation-free, and visible attack investigation for GUI applications," in *NDSS*. The Internet Society, 2020.

[22] P. Fang, P. Gao *et al.*, "Back-propagating system dependency impact for attack investigation," in *USENIX Security Symposium*. USENIX Association, 2022.

[23] P. Fei, Z. Li *et al.*, "SEAL: storage-efficient causality analysis on enterprise logs with query-friendly compression," in *USENIX Security Symposium*. USENIX Association, 2021, pp. 2987–3004.

[24] L. Yu, S. Ma *et al.*, "Alchemist: Fusing application and audit logs for precise attack provenance without instrumentation," in *NDSS*. The Internet Society, 2021.

[25] Q. Wang, W. U. Hassan, A. Bates, and C. A. Gunter, "Fear and logging in the internet of things," in *NDSS*. The Internet Society, 2018.

[26] Z. B. Celik, G. Tan, and P. D. McDaniel, "Iotguard: Dynamic enforcement of security and safety policy in commodity iot," in *NDSS*. The Internet Society, 2019.

[27] M. Botacin, F. D. Domingues, F. Ceschin, R. Machnicki, M. A. Z. Alves, P. L. de Geus, and A. Grégio, "Antiviruses under the microscope: A hands-on perspective," *Comput. Secur.*, vol. 112, p. 102500, 2022.

[28] W. Cui, V. Paxson, N. Weaver, and R. H. Katz, "Protocol-independent adaptive replay of application dialog," in *NDSS*. The Internet Society, 2006.

[29] L. Babun, A. K. Sikder *et al.*, "The truth shall set thee free: Enabling practical forensic capabilities in smart environments," in *NDSS*. The Internet Society, 2022.

[30] M. Botacin, P. L. de Geus, and A. R. A. Grégio, "Who watches the watchmen: A security-focused review on current state-of-the-art techniques, tools, and methods for systems and binary analysis on modern platforms," *ACM Comput. Surv.*, vol. 51, no. 4, pp. 69:1–69:34, 2018.

[31] S. Sabour, N. Frosst, and G. E. Hinton, "Dynamic routing between capsules," in *NIPS*, 2017, pp. 3856–3866.

[32] "Mirai-Source-Code," https://github.com/jgamblin/Mirai-Source-Code.

[33] Y. Miretskiy, A. Das, C. P. Wright, and E. Zadok, "Avfs: An on-access anti-virus file system," in *USENIX Security Symposium*. USENIX, 2004, pp. 73–88.

[34] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck, "DREBIN: effective and explainable detection of android malware in your pocket," in *NDSS*. The Internet Society, 2014.

[35] C. Wu, Y. Zhou, K. Patel, Z. Liang, and X. Jiang, "Airbag: Boosting smartphone resistance to malware infection," in *NDSS*. The Internet Society, 2014.

[36] M. N. Hossain, J. Wang *et al.*, "Dependence-preserving data compaction for scalable forensic analysis," in *USENIX Security Symposium*. USENIX Association, 2018, pp. 1723–1740.

[37] M. N. Hossain, S. Sheikhi, and R. Sekar, "Combating dependence explosion in forensic analysis using alternative tag propagation semantics," in *IEEE Symposium on Security and Privacy*. IEEE, 2020.

[38] Winbond.

[39] "Saleae logic analyzer," https://www.saleae.com/.

[40] "W25Q128FV Datasheet," https://www.pjrc.com/teensy/W25Q128FV.pdf.

[41] W. David, "Jffs: The journalling flash file system," in *Ottawa linux symposium*, vol. 2001. Citeseer, 2001.

[42] K. Liu, M. Yang, Z. Ling, H. Yan, Y. Zhang, X. Fu, and W. Zhao, "On manually reverse engineering communication protocols of linux-based iot systems," *IEEE Internet Things J.*, vol. 8, no. 8, pp. 6815–6827, 2021.

[43] "The GNU C Library Reference Manual," https://www.gnu.org/software/libc/manual/pdf/libc.pdf.

[44] "LZO," http://www.oberhumer.com/opensource/lzo/, 2017.

[45] "OpenWrt Porject," https://openwrt.org/, 2021.

[46] "A labeled dataset with malicious and benign IoT network traffic," https://www.stratosphereips.org/datasets-iot23.

[47] C. I. for Cybersecurity, "Intrusion DetectionEvaluation Dataset (CICIDS2017)." [EB/OL], 2018, https://www.unb.ca/cic/datasets/ids-2017.html Accessed November 27, 2020.

[48] G. Xie, Q. Li *et al.*, "Mousika: Enable General In-Network Intelligence in Programmable Switches by Knowledge Distillation," in *INFOCOM*. IEEE, 2022, pp. 1938–1947.

[49] D. Barradas, N. Santos *et al.*, "Flowlens: Enabling efficient flow classification for ml-based network security applications," in *NDSS*. The Internet Society, 2021.

[50] C. Fu, Q. Li, M. Shen, and K. Xu, "Realtime Robust Malicious Traffic Detection via Frequency Domain Analysis," in *CCS*. ACM, 2021, pp. 3431–3446.

[51] C. Liu, L. He, G. Xiong, Z. Cao, and Z. Li, "Fs-net: A flow sequence network for encrypted traffic classification," in *INFOCOM*. IEEE, 2019, pp. 1171–1179.

[52] X. Lin, G. Xiong, G. Gou, Z. Li, J. Shi, and J. Yu, "ET-BERT: A contextualized datagram representation with pre-training transformers for encrypted traffic classification," in *WWW*. ACM, 2022, pp. 633–642.

[53] T. Shapira and Y. Shavitt, "Flowpic: Encrypted internet traffic classification is as easy as image recognition," in *INFOCOM Workshops*. IEEE, 2019, pp. 680–687.

[54] "Syslog," https://en.wikipedia.org/wiki/Syslog, 2023.

[55] S. A. Crosby and D. S. Wallach, "Efficient data structures for tamper-evident logging," in *USENIX Security Symposium*. USENIX Association, 2009, pp. 317–334.

[56] J. Oberheide, E. Cooke, and F. Jahanian, "Cloudav: N-version antivirus in the network cloud," in *USENIX Security Symposium*. USENIX Association, 2008, pp. 91–106.

[57] G. D. L. T. Parra, L. Selvera, J. Khoury, H. Irizarry, E. Bou-Harb, and P. Rad, "Interpretable federated transformer log learning for cloud threat forensics," in *NDSS*. The Internet Society, 2022.

[58] J. Zeng, Z. L. Chua, Y. Chen, K. Ji, Z. Liang, and J. Mao, "WATSON: abstracting behaviors from audit logs via aggregation of contextual semantics," in *NDSS*. The Internet Society, 2021.

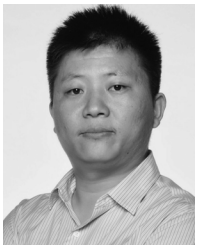
[59] M. Sundararajan, A. Taly, and Q. Yan, "Axiomatic attribution for deep networks," in *ICML*, ser. Proceedings of Machine Learning Research, vol. 70. PMLR, 2017, pp. 3319–3328.
[60] "Captum: Model Interpretability for PyTorch," https://captum.ai/.
[61] H. Li, Q. Huang, F. Ding, H. Hu, L. Cheng, G. Gu, and Z. Zhao, "Understanding and detecting remote infection on linux-based iot devices," in *AsiaCCS*. ACM, 2022, pp. 873–887.
[62] J. Haseeb, M. Mansoori, and I. Welch, "A measurement study of iot-based attacks using iot kill chain," in *TrustCom*. IEEE, 2020, pp. 557–567.
[63] R. Paccagnella, K. Liao, D. Tian, and A. Bates, "Logging to the danger zone: Race condition attacks and defenses on system audit frameworks," in *CCS*. ACM, 2020, pp. 1551–1574.
[64] H. N. Noura, O. Salman, A. Chehab, and R. Couturier, "Distlog: A distributed logging scheme for iot forensics," *Ad Hoc Networks*, vol. 98, 2020.
[65] "Selenium automates browsers." https://www.selenium.dev/, 2023.
[66] "A lightning fast multithreaded network scanner framework with modules." https://github.com/xdavidhu/portSpider, 2023.
[67] "Hydra," https://github.com/vanhauser-thc/thc-hydra, 2023.
[68] F. Pagani and D. Balzarotti, "Back to the whiteboard: a principled approach for the assessment and design of memory forensic techniques," in *USENIX Security Symposium*. USENIX Association, 2019, pp. 1751–1768.
[69] M. Botacin, A. Grégio, and M. A. Z. Alves, "Near-memory & in-memory detection of fileless malware," in *MEMSYS*. ACM, 2020, pp. 23–38.
[70] R. Bhatia, B. Saltaformaggio, S. J. Yang, A. I. Ali-Gombe, X. Zhang, D. Xu, and G. G. R. III, "Tipped off by your memory allocator: Device-wide user activity sequencing from android memory images," in *NDSS*. The Internet Society, 2018.
[71] A. Saha, N. Ganguly, S. Chakraborty, and A. De, "Learning network traffic dynamics using temporal point process," in *INFOCOM*. IEEE, 2019, pp. 1927–1935.
[72] F. Xu, Y. Li, H. Wang, P. Zhang, and D. Jin, "Understanding Mobile Traffic Patterns of Large Scale Cellular Towers in Urban Environment," *IEEE/ACM Trans. Netw.*, vol. 25, no. 2, pp. 1147–1161, 2017.
[73] F. Liu, J. Guo, X. Huang, and J. C. S. Lui, "eBA: Efficient Bandwidth Guarantee Under Traffic Variability in Datacenters," *IEEE/ACM Trans. Netw.*, vol. 25, no. 1, pp. 506–519, 2017.
[74] C. Xu, J. Shen, and X. Du, "A Method of Few-Shot Network Intrusion Detection Based on Meta-Learning Framework," *IEEE Trans. Inf. Forensics Secur.*, vol. 15, pp. 3540–3552, 2020.
[75] M. Korczynski and A. Duda, "Markov chain fingerprinting to classify encrypted traffic," in *INFOCOM*. IEEE, 2014, pp. 781–789.
[76] M. Shen, M. Wei *et al.*, "Classification of Encrypted Traffic With Second-Order Markov Chains and Application Attribute Bigrams," *IEEE Trans. Inf. Forensics Secur.*, vol. 12, no. 8, pp. 1830–1843, 2017.
[77] F. Ciucu, F. Poloczek, and J. B. Schmitt, "Sharp per-flow delay bounds for bursty arrivals: The case of fifo, sp, and EDF scheduling," in *INFOCOM*. IEEE, 2014, pp. 1896–1904.
[78] Z. Zhao, Z. Li, J. Jiang, F. Yu, F. Zhang, C. Xu, X. Zhao, R. Zhang, and S. Guo, "Ernn: Error-resilient rnn for encrypted traffic detection towards network-induced phenomena," *IEEE Transactions on Dependable and Secure Computing*, 2023.
[79] Z. Song, Z. Zhao, F. Zhang, G. Xiong, G. Cheng, X. Zhao, and S. Guo, "I2rnn: An incremental and interpretable recurrent neural network for encrypted traffic classification," *IEEE Transactions on Dependable and Secure Computing*, 2023.
[80] X. Wang *et al.*, "Looking from the mirror: Evaluating iot device security through mobile companion apps," in *USENIX*, 2019.
[81] M. Botacin and A. Grégio, "Why we need a theory of maliciousness: Hardware performance counters in security," in *ISC*, ser. Lecture Notes in Computer Science, vol. 13640. Springer, 2022, pp. 381–389.
[82] "Perf Tutorial," https://perf.wiki.kernel.org/index.php/Tutorial, 2023.
[83] K. Basu *et al.*, "PREEMPT: preempting malware by examining embedded processor traces," in *DAC*. ACM, 2019, p. 166.
[84] A. Costin, J. Zaddach, A. Francillon, and D. Balzarotti, "A large-scale analysis of the security of embedded firmwares," in *USENIX Security Symposium*. USENIX Association, 2014, pp. 95–110.
[85] F. Pagani and D. Balzarotti, "Back to the whiteboard: a principled approach for the assessment and design of memory forensic techniques," in *USENIX Security Symposium*. USENIX Association, 2019.

Ziming Zhao is a Ph.D. student in Zhejiang University, Hangzhou, China. He has published more than 5 papers in international journals and conference proceedings, including TIFS, TDSC, and AAAI. His research interests include machine learning, traffic identification, and privacy-preserving.

Zhaoxuan Li (Student Member, IEEE) is a Ph.D. student in State Key Laboratory of Information Security (SKLOIS), Institute of Information Engineering (IIE), Chinese Academy of Sciences (CAS), Beijing, China. He has published more than 10 papers in international journals and conference proceedings, including TIFS, TDSC, COMNETS, ESE, and ICWS. His research interests include traffic identification, blockchain security, formal methods, and privacy-preserving.

Jiongchi Yu received his bachelor's degree from Zhejiang University, China. He is pursuing the PhD degree at the School of Computing and Information Systems, Singapore Management University (SMU), Singapore. His research focuses on traditional software testing and security issues of cloud native infrastructures.

Fan Zhang (Member, IEEE) received his Ph.D. degree from the Department of Computer Science and Engineering, University of Connecticut, CT, USA, in 2011. He is currently a Full Professor with the College of Computer Science and Technology, Zhejiang University, Hangzhou, China, and also with the Alibaba–Zhejiang University Joint Institute of Frontier Technologies, Hangzhou. His research interests include system security, hardware security, network security, cryptography, and computer architecture.



Xiaofei Xie received the B.E., M.E., and Ph.D. degrees from Tianjin University. He is currently an Assistant Professor with Singapore Management University, Singapore. His research mainly focuses on program analysis, traditional software testing, and quality assurance analysis of artificial intelligence. He was a recipient of the two ACM SIGSOFT Distinguished Paper Awards in FSE'16 and ASE'19.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2023.3311012

Haitao Xu received the Ph.D. degree in computer science from the College of William and Mary, VA, USA, in December 2015. He is currently an Assistant Professor with the School of Cyber Science and Technology, Zhejiang University. His research interests include intersection of cyber security, privacy, and data analytics.

Binbin Chen (Member, IEEE) received the B.Sc. degree in computer science from Peking University and the Ph.D. degree in computer science from the National University of Singapore. Since July 2019, he has been an Associate Professor in the Information Systems Technology and Design (ISTD) pillar, Singapore University of Technology and Design (SUTD). He currently also holds a joint appointment as Principal Research Scientist at Advanced Digital Sciences Center, which is a University of Illinois research center located in Singapore. His current research interests include wireless networks, cyber-physical systems, and cyber security for critical infrastructures.