TPE-Det: A Tamper-Proof External Detector via Hardware Traces Analysis Against IoT Malware

Ziming Zhao, *Member, IEEE*, Zhaoxuan Li, *Member, IEEE*, Tingting Li, and Fan Zhang, *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. A series of solutions based on system calls, system logs, or hardware performance counters achieve promising results. However, such internal monitors are easily tampered with, especially against adaptive adversaries. In addition, existing system log records typically exhibit substantial volume, resulting in data explosion problems. In this article, we present TPE-Det, a side-channel-based external monitor to cope with these issues. Specifically, TPE-Det leverages the serial peripheral interface bus to extract the on-chip traces and designs a recovery pipeline for operating logs. The advantages of this external monitor are adversary-unperceived and tamper-proof. The restored logs mainly include file operation commands, which are lightweight compared to complete records. Meanwhile, we deploy a series of machine learning models with respect to statistical, sequence, and graph features to identify malware. Empirical evaluation shows that our proposal has tamper-proof capability, high-detection accuracy, and low-time/space overhead compared to state-of-the-art methods.**

***Index Terms*—Internet of Things (IoT) security, malware detection, serial peripheral interface (SPI) bus, tamper-proof external monitor.**

I. Introduction

MALWARE continues to be prevalent in Internet infrastructure nowadays and has caused profound attention from the security community [1]. Although malware is not a new threat, the boom of Internet of Things (IoT) [2], [3] 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, these IoT devices are often highly vulnerable to attack and might be used to create large, powerful botnets [4]. Most famously, Mirai was used to launch vast volumetric DDoS [5], [6], [7], [8], [9], e.g., Dyn, a DNS provider, suffered a 1.2 Tbps attack.

In recent years, researchers have proposed a series of methods to detect malware, but this arms race does not end there as adversaries deploy some counter-reconnaissance strategies. For example, some typical methods [10], [11] collect system logs or system calls [12] for malware detection. However, these solutions are adversary-perceived, i.e., attacker can be aware of the existence of the detection program through scanning processes or checking some dynamic fields [13]. Then, the attacker could try to inject fake state data into the system [14], tamper with system logs [15], or directly kill the monitor process [16]. This ultimately leads to detection failure, also for some hardware performance counters (HPCs) [17], [18]-based solutions.

In summary, existing solutions have the following limitations.

1) *Adversary-Perceived:* Given that most detection programs are deployed inside the system, adversary perception is possible through system state inspection or process scanning [13]. As shown in Fig. 1, such adversary perception capabilities may be the first step in subsequent counter-reconnaissance activities.
2) *Expose Risks for Tampering:* After attackers perceive the existence of an internal monitor, they may perform a series of tampering operations to hide or delete their attack traces. Since the log records used for detection are retained on the victim device, they can easily be maliciously tampered with [14] and [16]. In Fig. 1, on the one hand, the attacker may directly kill the internal monitoring program.On the other hand, they may also tamper with the recorded logs. Either process killing or wrong logs will directly cause the internal monitor to fail [15].
3) *Huge Logging Records:* Moreover, an additional problem is that the existing logging methods are redundant and bulky [19], [20]. Specifically, previous solutions, such as using syscall or syslog, usually consume >100-MB space overhead for recording behavior of ∼10K malware. This is not practical in real-world scenarios because one of the characteristics of IoT devices is limited available resources [21].

Considering the above problems of internal monitors [22], we intend to develop an external monitor to advance this

Manuscript received 12 August 2024; accepted 12 August 2024. This work was supported in part by National Key Research and Development Program of China under Grant 2023YFB3106800; in part by the National Natural Science Foundation of China under Grant 62227805, Grant 62072398, and Grant 62172405; in part by the Natural Science Foundation of Jiangsu Province under Grant BK20220075; in part by the Fok Ying-Tung Education Foundation for Young Teachers in the Higher Education Institutions of China under Grant 20193218210004; and in part by the Key Research and Development Program of Zhejiang Province under Grant 2023C01039. This article was presented at the International Conference on Compilers, Architectures, and Synthesis for Embedded Systems (CASES) 2024 and appeared as part of the ESWEEK-TCAD Special Issue. This article was recommended by Associate Editor S. Dailey. *(Corresponding author: Fan Zhang.)*

Ziming Zhao, Tingting Li, and Fan Zhang are with the College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China (e-mail: zhaoziming@zju.edu.cn; litt2020@zju.edu.cn; fanzhang@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, University of Chinese Academy of Sciences, Beijing 100049, China (e-mail: lizhaoxuan@iie.ac.cn).

Digital Object Identifier 10.1109/TCAD.2024.3444712

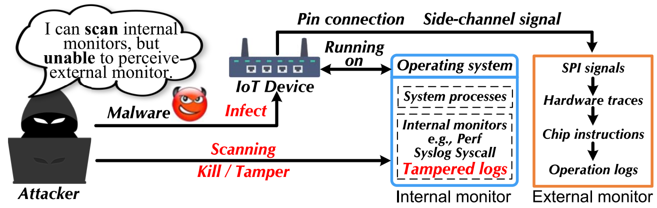


Fig. 1. Illustration of internal and external monitors.

landscape. In this article, we present TPE-Det, a Tamper-Proof External monitor for malware Detection. We leverage the hardware design knowledge of bus-connected system-on-a-chip (SoC) and design TPE-Det based on the serial peripheral interface (SPI) bus to monitor and analyze the on-chip instructions. Then, we develop a suite of pipelines to recover system file operation logs to realize information collection in a side-channel manner ("orange box" in Fig. 1). Finally, typical machine learning (ML) models and tailor-made deep neural networks (DNNs) are deployed to identify malware.

In a nutshell, we make the following contributions.

1) We investigate the limitations of existing malware detection schemes against adaptive adversaries. A series of issues are revealed, including that internal detectors, could be perceived by adversaries, monitoring processes being killed by attackers, and the exposed risk of logs being tampered with.
2) We propose a novel external monitor, named TPE-Det, that leverages the SPI bus to capture on-chip traces and design the operating logs recovery pipeline in a side-channel manner. Since TPE-Det is external to the device, it is tamper-proof and the adversary cannot perceive the existence of TPE-Det. An extra benefit is that the recorded logging of TPE-Det is concise.
3) We deploy a series of ML models and tailored DNNs with respect to statistical, sequence, and graph features to perform malware identification.
4) Empirical evaluations on our physical testbed demonstrate that TPE-Det clearly outperforms state-of-the-art (SOTA) methods, especially when against adaptive adversaries. We also conduct a series of experiments for concept drift, overhead evaluation, and providing more deep insights.

II. Related Work

Existing IoT malware detection approaches can be roughly categorized into hardware-based (host-based) and network-based ones, we outline the related work here.

A. Hardware/Host-Based Methods

Hardware/host-based methods [23] aim to extract the processor fingerprint from different programs running on the IoT devices. Some previous arts [17], [18], [24], [25] propose to detect malware based on HPCs runtime information. The hardware part design of TPE-Det 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*, etc.) and the count results are usually numeric variables. For another, more importantly, the process (e.g., *perf* tool [19]) for statistics HPC can be killed by adaptive adversaries like that the attacker kills internal monitors (mentioned in Section I). In addition, the number of available HPCs is limited on today's microprocessors. And HPCs rely on the operating environment, especially existing work [26], [27] reveals that measured HPC values collected for the same program running inside a virtual machine (VM) substantially differ from those collected on a bare-metal system. Costin et al. [28] conducted a large-scale firmware analysis for embedded devices. PREEMPT [26] repurposes the embedded trace buffer (ETB) to collect signal values through the joint test action group (JTAG) debug interface and connect the host machine via universal asynchronous receiver-transmitter (UART). Different from them, TPE-Det introduces a new direction that utilizes side-channel SPI signals to recover logs.

Another typical scheme captures system calls, e.g., HRAT [12] constructs function call graph to profile malware's behavior. Moreover, system logs are usually used to analyze whether there is malicious activity, e.g., Deeplog [11] deploys recurrent neural network (RNN) and FedTrans [29] leverages Transformer to model log records. IoTGuard [30] implements a code instrument to collect the app's information at runtime by adding extra logic. Meanwhile, some technologies, such as data compaction [31] and alternative tag propagation semantics [32], are presented to combat dependence explosion in long-term monitoring. However, these methods could be adversary-perceived and the logs can be tampered with. Therefore, TPE-Det explores a new perspective and utilizes SPI signal to restore operation commands in a side-channel manner, realizing tamper-proof capability.

B. Network-Based Methods

Network-based methods tend to profile the special traffic pattern when the IoT devices are remoted intrusion [33], [34]. To profile the traffic pattern, some works [35], [36], [37] design supervised learning methods based on statistical features, e.g., random forests (RFs). And some other arts utilize Markov [38] or RNNs [39], [40], [41] to portray the sequential features (e.g., packet length sequence) of attacks. For the IoT devices, Gu et al. [42] presented IoTGaze to discover the threats by sniffing event interaction in wireless traffic. Wang et al. [43] performed a cross-analysis for mobile companion apps to evaluate IoT devices' security.

Although traffic capture programs, such as *tshark*, can also be killed or packets could be tampered with, monitoring network traffic is a promising approach, that is orthogonal to host-based detection and they also can be combined [4]. A representative art, Hawkware [10] combines traffic analysis and system calls to detect malware. Overall, we select four ETB-based models (from PREEMPT), 12 HPC-based ML/ensemble models, HRAT, Deeplog, FedTrans, and Hawkware as the baselines in evaluations.

III. Motivation

In this section, we clarify the motivation for the design of TPE-Det. First, we explain that even malware that infects random access memory (RAM) will perform a series of file operations on read only memory (ROM). Then, we introduce the benefits of external monitors in terms of adversary-unperceived and tamper-proof. Finally, we form the core idea of designing an external monitor to track file operations using SPI, and explain how it will be beneficial in some real-world scenarios.

1) *Why Malware Infections Involve File Operations?* 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 [4]. This persistent malware usually implants malware viruses into the electrically erasable programmable ROM (EEPROM) to achieve persistence on IoT devices. In addition, even typical malware that infects RAM will perform a series of file operations on 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 [44]. More details can be found in Section VI-E. These file operations on ROM can be captured by SPI, so we intend to leverage SPI signals to analyze on-chip traces.
2) *What Are the Benefits of Using an External Monitor?* Using internal monitor to obtain details of malware is a viable scheme, such as causality analysis for system logs [45], [46], [47] or adopt system hooking for investigation [30]. However, these proposed techniques have some limitations. On the one hand, the internal monitor could be adversary-perceived, e.g., the attacker may detect the existence of a running monitor by checking some dynamic fields [13]. On the other hand, internal monitors are prone to be subverted, e.g., the attacker could tamper with operation log files [14], [15] even directly kill the monitoring process [16], [48]. Existing research [22], [49] suggests that if leveraging hardware design knowledge to develop an external monitor [50], such as a bus-connected SoC, it will tend to be tamper-proof. PREEMPT [26] is a representative external monitor, and we also compare it to illustrate the advantages of TPE-Det in detection performance (in Section VI).
3) *How Does TPE-Det Facilitate Real-World Security Scenarios?* We elaborate here that TPE-Det could promote typical security scenarios involving investigation forensics and honeypots. For investigation forensics [45], [46], [47], correct logs are necessary to support practitioners in building threat intelligence, correlation analysis, etc. The attacker could tamper with logs or kill processes to cause the failure of internal monitors [15], [16] (as stated above), which will directly affect the forensic results (we develop the adaptive adversary experiments in Section VI-D). For IoT honeypots [51], [52], serve as critical tools in the cybersecurity landscape, they are used to attract attackers and capture/collect malicious behaviors. However, an adaptive attacker [13], [53] will actively terminate its attack behavior when it senses the presence of a monitoring process, this will result in the honeypot being unable to collect corresponding threat intelligence and attack information. Given that TPE-Det is adversary-unperceived and tamper-proof, it could facilitate investigation forensics and honeypot scenarios.

IV. Assumptions and Threat Model

A. Adversary Model

We consider strong/adaptive adversaries that adopt various attack strategies. They could exploit remote transmission to implant malware/viruses or leave them into the IoT devices' built-in chips to achieve potential persistent attacks, including malware, that infects RAM or residing in ROM, e.g., Mirai [5] and Hajime [6]. These malicious activities/executions will involve a series of file operations [5], [6], e.g., file creation, writing, permission modification, and self-induced deletion. More importantly, *adaptive adversaries* indicate that a series of counter-reconnaissance technologies will be deployed. Specifically, the attacker will scan the processes running on the victim device, and once the presence of internal monitors is discovered, they will directly kill the program or tamper with the recorded logs.

B. Assumptions

We explain some assumptions here. Given that TPE-Det 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 such an external monitor is promising, especially in facilitating real-world security scenarios, e.g., investigation forensics and honeypots (stated in Section III). We also perform more security discussions in Section VII-A. Although our physical testbed has a specific architecture, our approach is not limited to architecture and the detector has cross-architecture capabilities (Section VI-E). Furthermore, we intend to enable hardware trace monitoring before infection to avoid missing some malicious behavior. This assumption is similar to typical anti-virus software, in which practitioners usually run anti-virus programs in advance to detect potential malware [54].

V. Design of TPE-Det

In Fig. 2, we depict the high-level architecture of TPE-Det. Consider a running IoT device, which the SPI bus can be used to dump content for Flash chips. We connect the SPI bus in parallel with a logic analyzer that parses the digital signals to the Flash traces. These traces will be recovered as system operating logs, which could be fed to ML models for classification.

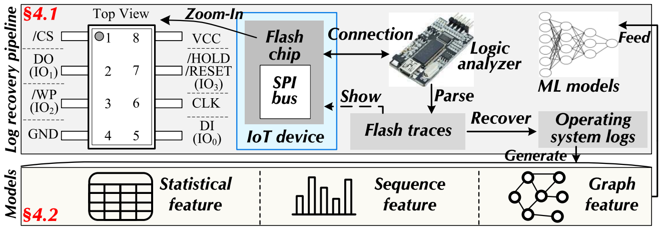


Fig. 2. Overview of TPE-Det.

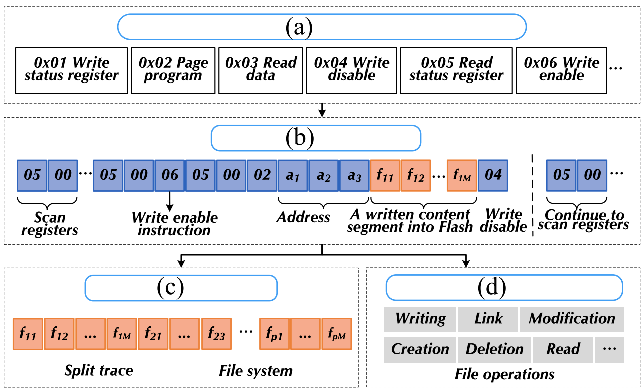


Fig. 3. Log recovery pipeline of external monitor. (a) SPI signal digitization (instruction). (b) Flash trace. (c) Structure file system. (d) Operation log recovery.

A. External Monitor and Log Recovery

We clarify here the proposed design pipeline to externally recover the operation log in the file system, as shown in Fig. 3. The so-called "externally recover logs" refers to the external monitor (e.g., leveraging SPI signals), not the internal monitor, so as to make the adversary imperceptible and tamper-proof.

Collect the Digital Signals: To acquire digital signals emanating from the Flash chip integrated within an IoT device, a logic analyzer is interfaced in parallel with the SPI bus's pins. This logic analyzer, serving the role of a physical probe, is affixed to the SPI pins and executes 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. The specific sampling frequency setting is explained in Section VI-A. Upon collection, the resultant sampling data are archived within an upper computer system.

Extract the Flash Traces: The collected digital signals mainly consist of the chip instructions[1] and the content written into Flash (i.e., the nonvolatile data). The chip instructions are related to the chip type and the physical state, so the specific instruction hex can be confirmed according to the corresponding chip manual. For instance, the W25Q128BV [55] chip[2] has some instructions, and their hexadecimal as follows.

1) $0x01$: Write Status Register Instruction.
2) $0x02$: Page Program Instruction.
3) $0x03$: Read Data Instruction.
4) $0x04$: Write Disable Instruction.
5) $0x05$: Read Status Register Instruction.
6) $0x06$: Write Enable Instruction.

These chip instructions are instrumental in analyzing the SPI trace, enabling us to discern the executed operations, such as "Read" and "Write." Furthermore, they allow us to ascertain the specific nonvolatile data content and its corresponding address. As illustrated in Fig. 3(b), we can extract the Flash traces, e.g., $< a_1, a_2, a_3 >$ represents the 24-bit address and $< f_{11}, f_{12}, \ldots, f_{1M} >$ refers to content segments written.

Structure the File System: In this step, TPE-Det will meticulously structure the file system to assemble segments. The primary objective of this process is to enhance the comprehension of the nonvolatile data's content and its precise storage location within the device. This insight is invaluable for practitioners, as it provides a deeper understanding of the attack mechanisms, which is essential for forensic investigations. Note that this process does not influence the malware detection phase, given the subsequent models rely solely on the features derived from operation commands, as elaborated in Section V-B.

In reality, the structuring process is inherently related to the file system architecture. When employing various file systems, the necessary adaptation is confined to modifying the mapping procedure that correlates storage addresses with file directories, as well as the decompression algorithm (customarily deployed to economize on space overhead) for the nonvolatile data. We implement a prototype for the journaling flash file system (JFFS2) [58] log structure[3] used in our testbed. Specifically, within the JFFS2 framework, there are two data entities that are intimately associated with file operations, i.e., *jffs2_raw_dirent* and *jffs2_raw_inode*.

We can discern these two data entities by examining the parameter *magic* + *nodetype*, and ascertain the entity length utilizing the parameter *totlen*. The entity *jffs2_raw_dirent* is responsible for delineating the file's location, specifically within its parent directory, while *jffs2_raw_inode* is tasked with retaining the file's management information. Notably, the latter contains the actual written content within the Flash memory in its *data* parameter and leverages the *mode* parameter to document file types and modes. 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 [60]. Based on these two entities, we mount corresponding nodes to structure the tree-shaped file system. Considering that the content may be compressed to conserve space, we apply the corresponding decompression algorithm (stored in the *compr* of *jffs2_raw_inode*) to retrieve the complete data. For example, the compression algorithms of JFFS2 and their hexadecimal are {ZERO: 0x01, RTIME: 0x02, RUBINMIPS: 0x03, COPY: 0x04, DYNRUBIN: 0x05, ZLIB: 0x06, LZO: 0x07}. By employing the above process,

[1]They refer to the SPI instruction set and are fully controlled through the SPI bus, e.g., the W25Q128FV contains 45 basic instructions [55].

[2]Particularly, our design does not depend on a specific chip and only needs to satisfy the above-mentioned Nyquist–Shannon sampling theorem. According to product reports of the chip manufacturer, the maximum frequency of common SPI Flash chips is 80–133 MHz [56]. Modern logic analyzers support a sampling rate of 500 MS/s [57].

[3]JFFS2 is widely used in IoT devices due to its power-disconnected reliability and space-efficient properties [58], such as [59] mentioned that 333 firmware was collected from Axis Communications (a network camera device manufacturer), and about 85% of them use the JFFS2 file system.

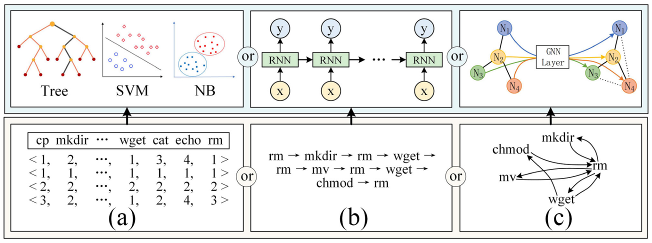


Fig. 4. Models regarding three types of features. (a) Statistical feature. (b) Sequence feature. (d) Graph feature.

we are able to accurately map file addresses and meticulously parse the contents of nonvolatile data.

Recover the Operating System Logs: Upon structuring the file system, we can glean a wealth of information, including file types, names, permissions, contents, access timestamps, and so on. This available comprehensive information on nodes can subsequently be harnessed to reconstruct the behavior logs of the operating system. For example, *emerging instances of entities → file creation*; *changing the entities' contents → file writing*; and *mounting the entities to the garbage collection node → file deletion*. The logs that are recovered encompass a range of file operations, such as creation, deletion, reading, writing (including content), permission modifications, timestamps of the last modification, soft links, and more. Readers might be concerned about the potential impact on the log recovery process if adversaries were to tamper with system logs or eliminate binary scripts (as mentioned in Section IV). However, as long as the SPI signal collection is conducted proactively, TPE-Det can recover the operation log even if the malicious script is deleted, attacker's tampering will also be recorded. This robust capability stems from the design of TPE-Det as an external monitoring tool. All file operations, such as reading, permission modifications, deletions, and so forth, are meticulously recorded and archived on an upper computer for log recovery purposes.

B. ML Models for Detection

As shown in Fig. 2, the system operation logs will be fed into ML models[4] for malware identification. In Fig. 4, we deploy TPE-Det with a series of models that regard the statistical feature, sequence feature, and graph feature.

1) The statistical feature refers to counting commands in a fixed order to form a vector. For example, given a statistical feature vector $\{C_1{:}i_1, C_2{:}i_2, \ldots, C_n{:}i_n,\}$, it means that i_1 C_1 commands, i_2 C_2 commands, $\cdots$, i_n C_n commands are involved. We use a series of ML models to analyze the statistical feature, including decision tree (DT), RF, support vector machine (SVM), XGBoost (XGB), and Naive Bayes (NB). All these models are set with default parameters of Python *scikit-learn* library.

[4]System operation logs can also be combined with lightweight rule-matching methods for detection and analysis. Given the wide application of ML technology in IoT security [61], [62], [63], we tend to use a series of ML models for analysis here so that we can perform a fair comparison with existing works that use ML to analyze logs from HPC, *Syslog*, *Syscall*, etc.

2) The sequence feature refers to the command calling sequence of malware. Fig. 4(b) displays the partial command execution sequence of malware whose MD5 is *7044865a1cfd07535400d7e041786940*. It performs a series of operations, including *rm*, *mkdir*, *rm*, *wget*, etc. We use long short-term memory (LSTM) [64], a typical RNN model, to identify these sequence features to discover behavioral patterns commonly used by attackers. In the tth time step operation of the LSTM unit, the forget gate f_t can be calculated as

$$f_t = \sigma\big(W_f \cdot \big[h_{t-1}, X_t\big] + b_f\big) \tag{1}$$

where h_{t-1} is the current hidden state and X_t denotes the tth input, i.e., the one-hot encoded vector of tth operation. The memory gate v_t can be computed as follows:

$$v_t = \sigma\big(W_v \cdot \big[h_{t-1}, X_t\big] + b_v\big). \tag{2}$$

And the temporary memory cell $\tilde{C}_t$ is computed by

$$\tilde{C}_t = \tanh\big(W_c \cdot \big[h_{t-1}, X_t\big] + b_c\big). \tag{3}$$

Then, the next cell state can be updated as follows:

$$C_t = f_t \cdot C_{t-1} + v_t \cdot \tilde{C}_t. \tag{4}$$

Finally, the output gate o_t can be obtained as

$$o_t = \sigma\big(W_o \cdot \big[h_{t-1}, X_t\big] + b_o\big). \tag{5}$$

And the next hidden state can be calculated as follows:

$$h_t = o_t \cdot \tanh(C_t). \tag{6}$$

Finally, the hidden state of the last step is used to obtain the final classification results $\hat{Y}$ with a fully connected layer f_c (W_c) and a Softmax function. Notably, the above parameter matrices W and b are all learnable parameters

$$\hat{Y} = \arg\max(\mathrm{Softmax}(W_c \cdot h_t)). \tag{7}$$

3) The graph feature is formed in a similar way to the sequence feature, the difference is that the same commands correspond to one node in the graph structure. As Fig. 4(c) shows, it uses a directed graph record, i.e., $(N_1 \rightarrow N_2) \neq (N_2 \rightarrow N_1)$ since their order is different. Intuitively, graph-structure records focus on call dependencies and are relatively robust even in the presence of obfuscation operations. Then, we analyze the operation command graph based on the traditional graph neural network (GNN) model [65], whose node features iteratively with

$$h_v^{(k)} = \phi\Big(h_v^{(k-1)}, f\Big(\Big\{h_u^{(k-1)} : u \in \mathcal{N}(v)\Big\}\Big)\Big) \tag{8}$$

where $\mathcal{N}(v)$ represents a set of nodes adjacent and reachable to the node v. Meanwhile, the function ϕ is injective and f operates on the set of neighbor nodes' feature vectors, which are called multisets. Also, the initial features $h_v^{(0)}$ of the graph node refer to its one-hot encoded vector.

In practice, we use multilayer perceptrons (MLPs) to model and learn the composition of two functions $f^{(k+1)} \circ \varphi^{(k)}$, thanks to the universal approximation theorem [66]. In the first iteration, we do not need MLPs before summation if input features are one-hot encodings as their summation alone is injective. Then, the graph model updates node representations as

$$h_v^{(k)} = \mathrm{MLP}^{(k)}\left((1+\epsilon^{(k)}) \cdot h_v^{(k-1)} + \sum_{u \in \mathcal{N}(v)} h_u^{(k-1)}\right) \tag{9}$$

where ϵ is a learnable parameter or a fixed scalar.

Furthermore, the graph-level readout is performed based on information from all depths/iterations of the model, due to a sufficient number of iterations is the key to achieving good discriminative power. To this end, we achieve this by an architecture similar to Jumping Knowledge Networks [67], where we make graph representations concatenated across all iterations/layers of the model as follows:

$$h_G = \mathrm{CONCAT}\left(\sum_{v \in G} h_v^{(k)} | k = 0, 1, \ldots, K\right) \tag{10}$$

$$\hat{Y} = \arg\max(\mathrm{Softmax}(W_c \cdot h_G)). \tag{11}$$

Finally, similar to the sequence model, the final classification results $\hat{Y}$ can also be obtained with a fully connected layer and a Softmax function, as described in (11). In this way, this model possesses the superior discriminative/representational power based on the neighbor aggregation and graph readout functions, so as to identify the malicious operations better.

VI. Evaluation

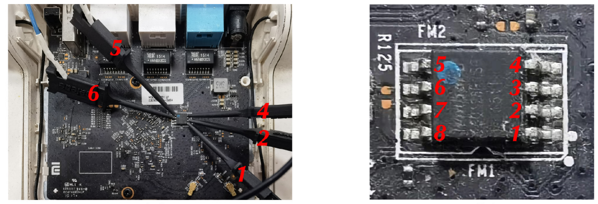
In this section, we comprehensively evaluate TPE-Det in terms of detection performance and overhead. Moreover, we perform a series of experiments, including time/space concept drift, deep insights, and adaptive adversary evaluation. Our code is available online.[5]

A. Experiment Setup

Testbed: Our testbed is established with wireless router (installed the W25Q128BV Flash [55], SPI bus, and MT7620 SoC chip), a logic analyzer (the Saleae Logic Pro 8 [57], supports a maximum sampling rate of 500 MS/s), an upper computer (installed an i7-9700 CPU, and 64 GB memory). Among them, the router is the victim IoT device, and the upper computer runs TPE-Det. The wireless router runs OpenWrt [68] which the file system is JFFS2. For the logic analyzer, we set the sampling rate is 500 MS/s, which means that it could achieve 3–6 times oversampling for common SPI Flash chips [56] (i.e., satisfies Nyquist–Shannon sampling theorem). One end of the logic analyzer is probed on pins 1, 2, 4, 5, and 6 of the Flash chip (Fig. 5), while the other end is connected to the upper computer.

Parameter Settings: All ML classifiers use the default parameters of the Python *scikit-learn* library. For the LSTM, we set the time step as 128, the hidden layer as 400, the number of layers as 2, and the learning rate as 1e-3. For the GIN, we set the number of hidden units as 64, the dropout as 0.5, the GNN layers as 5, they are all consistent with the original paper [65].

[5]Online repository: https://github.com/Secbrain/TPE-Det/.



(a) (b)

Fig. 5. Physical IoT device of TPE-Det testbed. (a) Connect pins to extract SPI signals. (b) Flash chip.

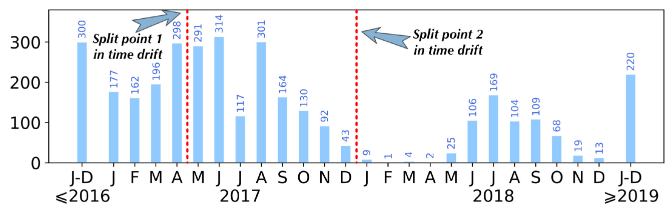


Fig. 6. Malware details of the dataset.

Datasets: The dataset used for evaluation is ScriptDataset [16], including 3439 malicious Linux shell scripts and 9337 benign firmware scripts. We depict the distribution of malware over time in Fig. 6, spanning 2012 to 2020. Among them, most malware samples are mainly concentrated in 2017 and 2018. For the dataset split, we default to adapt {*train*:*test* = 8:2}. We also divide malware as {*train*:*test* = 5:5} and {*train*:*test* = 2:8} to develop space bias experiments and as two split points (red dotted lines) in Fig. 6 to conduct time bias experiments [69]. Each group of experiments will be performed 5 times with different random seeds.

B. Detection Effect Evaluation

We first evaluate the malware detection effect of TPE-Det and SOTA methods, including four ETB-based models, 12 HPC-based models, and four syscall/log-based schemes. For four metrics (i.e., accuracy, precision, recall, and F1 score), the average detection results and standard deviation are summarized in Table I. We observe that our detectors based on recovery command achieve dominant results, especially GNN, XGB, RF, and LSTM (realize 95.87%–98.05% F1), which clearly outperforms 20 baselines. For the baseline, the syscall/syslog-based methods are generally better than ETB-based and HPC-based schemes. Nonetheless, TPE-Det still outperforms syscall/syslog-based methods by 3.91%–13.82% F1 score. For ETB-based baselines (belonging to external monitor), TPE-Det realizes >10.12% F1 score higher than PREEMPT [26]. Regarding HPC-based baselines, we can see that model ensembles indeed bring improvement, e.g., 7.99%–30.99% F1 improvement. However, MLP-Boost (the best in HPC-based) achieves 85.84% F1 (12.21% lower than our GNN), this can be attributed to the amount of HPC information is limited.

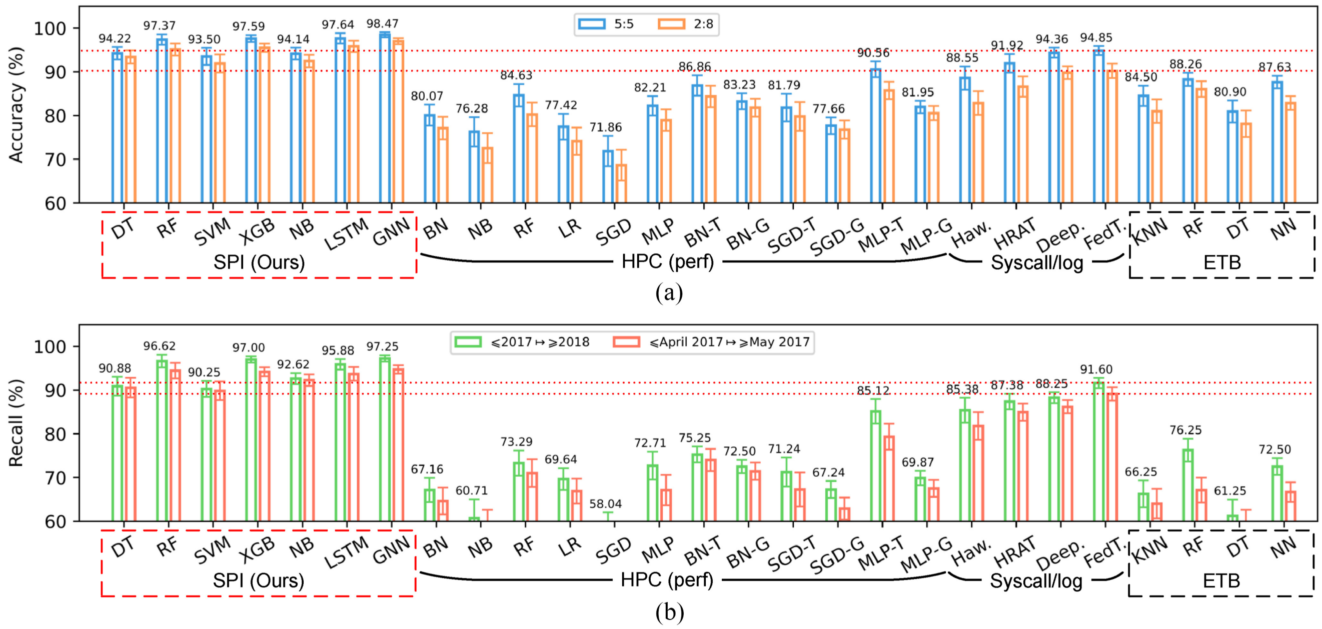


Fig. 7. Concept drift experiments in terms of space and time biases. (a) Space bias. (b) Time bias.

TABLE II
DETECTION EFFECT (%) OF CONCEPT DRIFT EXPERIMENTS IN TERMS OF SPACE BIAS AND TIME BIAS

Drift Exper.		Space bias								Time bias							
		{5 : 5}				{2 : 8}				≤2017 ↦ ≥2018				≤April 2017 ↦ ≥May 2017			
Model		Acc	Pre	Rec	F1	Acc	Pre	Rec	F1	Acc	Pre	Rec	F1	Acc	Pre	Rec	F1
SPI (ours)	DT	94.22	88.79	89.88	89.33	**93.39**	**86.49**	**89.39**	**87.92**	**93.51**	**87.91**	90.88	**89.37**	91.91	94.44	**90.53**	92.44
	RF	**97.37**	**94.44**	**95.87**	**95.15**	**95.09**	**90.18**	**91.75**	**90.95**	**96.44**	**91.91**	**96.62**	**94.21**	**94.49**	95.42	**94.44**	**94.93**
	SVM	93.50	87.60	88.37	87.98	**91.89**	**84.67**	**85.31**	**84.99**	92.65	85.95	90.25	88.05	91.16	93.74	**89.82**	91.74
	XGB	**97.59**	**95.21**	**95.87**	**95.54**	**95.50**	**91.10**	**92.29**	**91.69**	**96.85**	**92.82**	**97.00**	**94.87**	**94.90**	**96.41**	**94.18**	**95.28**
	NB	94.14	88.72	89.65	89.18	**92.46**	**85.64**	**86.48**	**86.06**	**94.41**	**89.17**	**92.62**	**90.86**	**93.10**	94.97	**92.27**	**93.60**
	LSTM	**97.64**	**95.01**	**96.28**	**95.64**	**95.82**	**91.83**	**92.73**	**92.28**	**95.88**	**90.88**	**95.88**	**93.31**	**94.61**	**96.34**	**93.69**	**95.00**
	GNN	**98.47**	**96.88**	**97.44**	**97.16**	**96.99**	**93.80**	**95.09**	**94.44**	**97.98**	**96.05**	**97.25**	**96.65**	**95.94**	**97.80**	**94.71**	**96.23**
ETB	KNN	84.50	69.09	76.73	72.71	80.98	63.94	67.28	65.57	78.63	63.86	66.25	65.03	68.91	75.39	64.03	69.23
	RF	88.26	77.08	80.22	78.62	86.04	72.64	77.21	74.85	84.43	73.05	76.25	74.62	73.52	81.19	67.11	73.49
	DT	80.90	63.14	69.75	66.28	78.08	58.96	61.11	60.01	74.50	56.98	61.25	59.04	62.83	68.75	58.67	63.31
	NN	87.63	75.96	79.06	77.48	82.83	66.90	71.76	69.23	82.56	70.30	72.50	71.38	71.82	78.53	66.67	72.12
HPC (perf)	BN	80.07	62.58	64.51	63.53	77.11	57.12	60.09	58.57	79.30	65.01	67.16	66.07	76.54	89.53	64.62	75.06
	NB	76.28	56.06	54.92	55.48	72.55	48.99	47.84	48.41	75.07	58.06	60.71	59.36	71.53	85.05	58.13	69.06
	RF	84.63	71.12	72.19	71.65	80.24	63.01	64.45	63.72	82.94	70.86	73.29	72.05	81.69	94.05	70.98	80.90
	LR	77.42	58.16	57.42	57.79	74.09	51.96	49.65	50.78	80.35	66.47	69.64	68.02	75.13	84.36	66.89	74.62
	SGD	71.86	47.62	45.38	46.47	68.65	42.00	43.22	42.60	74.24	56.93	58.04	57.48	69.18	83.48	54.36	65.84
	MLP	82.21	66.24	69.17	67.67	78.93	60.64	61.94	61.28	82.41	69.87	72.71	71.26	77.63	89.30	67.11	76.63
	BN-Boost	86.86	74.64	77.55	76.06	84.34	70.26	72.56	71.39	84.36	73.33	75.25	74.28	83.00	93.54	74.00	82.63
	BN-Bag	83.23	69.38	67.48	68.42	81.78	66.25	65.87	66.06	82.71	70.65	72.5	71.56	81.25	92.57	71.42	80.63
	SGD-Boost	81.79	66.87	64.11	65.46	79.78	62.90	60.71	61.78	81.59	68.59	71.24	69.89	77.22	88.27	67.24	76.34
	SGD-Bag	77.66	58.88	56.31	57.57	76.78	57.04	55.65	56.34	79.30	64.98	67.24	66.09	75.59	89.27	62.89	73.79
	MLP-Boost	90.56	81.53	83.94	82.72	85.72	73.32	73.83	73.57	89.88	81.85	85.12	83.45	85.81	93.75	79.33	85.94
	MLP-Bag	81.95	66.87	65.27	66.06	80.54	64.20	62.63	63.40	81.21	68.25	69.87	69.05	79.04	92.00	67.51	77.88
Syscall/log	Hawkware	88.55	77.72	80.57	79.12	82.85	67.83	68.99	68.41	89.16	79.88	85.38	82.54	87.44	94.50	81.78	87.68
	HRAT	91.92	84.43	85.81	85.11	86.59	74.49	76.34	75.40	90.96	83.31	87.38	85.30	89.46	95.26	84.93	89.80
	Deeplog	94.36	88.94	90.29	89.61	89.78	80.32	82.19	81.24	91.30	83.65	88.25	85.89	90.38	95.80	86.18	90.73
	FedTrans	94.85	89.49	91.62	90.54	90.19	80.03	84.66	82.28	92.61	84.94	91.60	88.14	92.08	96.12	89.11	92.48

C. Concept Drift Experiments

Moreover, we conduct concept drift experiments in terms of space bias and time bias [69]. 1) *Space bias* refers to unrealistic assumptions about the ratio of benign/malware, we divide the malware as {*train*:*test* = 5:5} and {*train*:*test* = 2:8} in Fig. 7(a) and the left part of Table II and 2) *Time bias* refers to malware could behave differently over time, we set two split points (Fig. 6) of malware samples. As shown in Fig. 7, against spatial/temporal biases, TPE-Det still maintains good performance compared to baselines. Particularly, both ETB-based and HPC-based schemes are not robust enough when against concept drift. Specifically, for accuracy results in space bias experiments in Fig. 7(a), we observe that only four models (RF, XGB, LSTM, and GNN) are superior to the best baseline (i.e., FedTrans) when {*train*:*test* = 5:5}, while all seven models perform better than the baselines when {*train*:*test* = 2:8}. Similar results can also be observed in Fig. 7(b), our seven SPI-based models all realize better recall than the baselines when the setting refers to ≤April 2017 ↦ ≥May 2017.

We report more detailed results in Table II in terms of the accuracy, precision, recall, and F1 score. From the left

TABLE I
DETECTION EFFECT OF EACH MODEL AND BASELINES

	Model	Acc (%)	Pre (%)	Rec (%)	F1 (%)
SPI (ours)	DT	95.77±1.24	90.38±2.95	94.32±2.07	92.31±2.53
	RF	**97.73±1.08**	**93.86±1.81**	**97.96±1.35**	**95.87±1.56**
	SVM	94.28±1.88	87.52±2.14	91.85±1.68	89.63±1.83
	XGB	**97.81±0.57**	93.52±1.51	**98.69±0.58**	**96.03±0.78**
	NB	96.44±1.20	90.71±1.61	**96.65±0.94**	93.59±1.37
	LSTM	**97.65±1.05**	**93.85±1.74**	**97.67±0.79**	**95.72±0.87**
	GNN	**98.94±0.47**	**97.14±1.09**	**98.98±0.42**	**98.05±0.74**
ETB	KNN	89.82±1.97	80.20±2.70	82.53±1.75	81.35±2.40
	RF	93.42±1.31	86.81±2.43	89.08±2.05	87.93±2.21
	DT	88.06±2.07	76.83±3.56	79.62±2.88	78.20±3.12
	NN	92.48±1.27	85.61±1.73	86.61±0.73	86.11±1.09
HPC (perf)	BN	83.79±2.28	69.75±2.37	70.16±2.66	69.96±2.46
	NB	78.23±3.22	59.91±3.18	57.64±4.05	58.75±3.71
	RF	87.90±2.48	76.69±3.77	79.04±2.78	77.85±2.96
	LR	79.48±2.83	61.24±2.91	64.63±2.34	62.89±2.54
	SGD	75.96±3.25	55.42±3.07	54.29±3.92	54.85±3.56
	MLP	85.47±2.08	72.51±3.48	74.09±2.87	73.29±3.09
	BN-Boost	88.76±2.17	78.82±2.23	79.62±1.58	79.22±1.88
	BN-Bag	85.83±1.62	74.07±1.38	72.78±1.20	73.42±1.27
	SGD-Boost	82.97±2.93	68.53±2.38	67.83±2.75	68.18±2.57
	SGD-Bag	80.70±1.72	64.31±1.57	63.46±1.73	63.88±1.68
	MLP-Boost	92.29±1.59	84.80±2.38	86.90±2.01	85.84±2.24
	MLP-Bag	84.10±1.20	71.00±1.38	69.14±1.54	70.06±1.47
Syscall/log	Hawkware	91.23±2.49	81.58±3.38	87.05±2.67	84.23±2.86
	HRAT	93.27±1.94	86.32±2.26	89.08±1.57	87.68±1.75
	Deeplog	95.93±1.21	91.58±1.35	93.45±0.95	92.51±1.18
	FedTrans	96.83±1.02	93.53±1.79	94.76±1.15	94.14±1.55

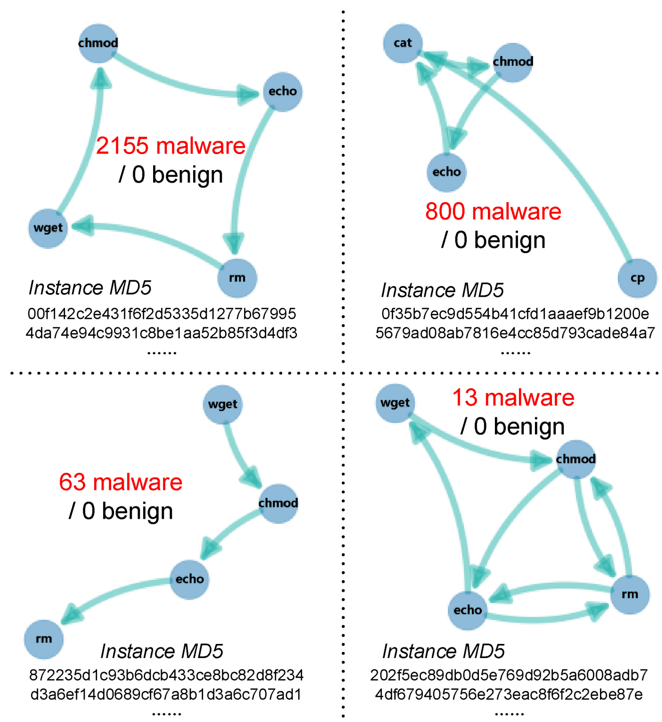


Fig. 8. Deep insights into TPE-Det.

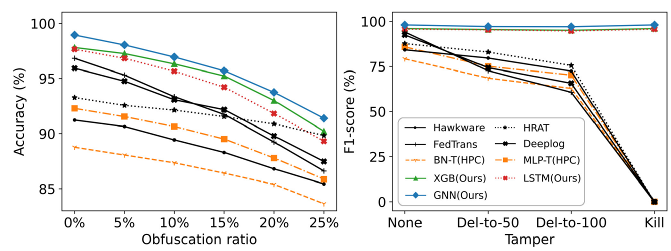


Fig. 9. Evaluation with obfuscation and tampering.

part of Table II, we find that all SPI-based models achieve better performance (involving four metrics) than baselines when {*train*:*test* = 2:8}. This presents that our scheme can indeed extract effective and stable features via leveraging SPI to recover execution behaviors, even if only a small number of samples are used for training. From the right part of Table II, we see some different phenomena. Except for the recall, for the other three indicators, not all seven SPI-based models are higher than the baseline. This may be attributed to the fact that the attack behavior of malware has a more certain pattern (so TPE-Det can achieve a higher-detection rate even if under the time bias setting), while the execution activities of benign programs are more diverse. We then analyze some malware instances to obtain more insights.

As shown in Fig. 8, we visualize command graph patterns for some malware. It is clear that there are 2,155 malware that have the structure of *wget* → *chmod* → *echo* → *rm*, while none benign present this pattern. In addition, Fig. 8 also displays that the other three command graph patterns are shared by 800, 63, and 13 malware, respectively. This can echo back the results in Table II, i.e., GNN performs the best performance, because many malware have isomorphic command graph structures. Overall, our SPI-based recovered operation commands are representative features for malware detection, even against space/time concept drift.

D. Adaptive Adversary Evaluation

We develop the adaptive adversary evaluation next.[6] For one thing, we consider adaptive adversaries who could perform operation obfuscation. For another, strong attackers could directly kill the monitor process or tamper with recorded logs.

1) Operation Obfuscation: The attacker may add additional operations into the malicious script for obfuscation [70], [71]. We set the ratio as {5%, 10%, 15%, 20%, 25%} to obfuscation randomly, the accruacy results are summarized in the left part of Fig. 9. We observe that HRAT is less affected by confusion since it takes feature interferences/modifications into account during model training [12]. The results of FedTrans and Deeplog are comparable. As the confusion ratio increases, the accuracy of most models tends to decrease slowly. When the obfuscation ratio is 25%, the top-5 performance models, in order, are GNN (ours) > XBG (ours) > HRAT > LSTM (ours) > Deeplog.

2) Log Tampering: To evaluate the tamper-proof capabilities, we consider the attackers could directly kill the internal monitor process or tamper with operation log files [14], [15], [16]. Specifically, we only save the first 50

[6] Although PREEMPT is an external monitor, it is not clear how to recover file operations based on ETB trace [26], so PREEMPT does not support operation log recovery and cannot be evaluated the tamper-proof capability against the adaptive adversary.

TABLE III
COMMAND RECOVERY AGAINST ADAPTIVE ADVERSARIES

Tamper	Method	*cat*	*chmod*	*cp*	*echo*
Kill	TPE-Det	**97.73%**	**99.18%**	**98.12%**	**97.27%**
	Syslog	0%	0%	0%	0%
Del-to-50	TPE-Det	**97.25%**	**99.18%**	**97.98%**	**96.86%**
	Syslog	69.21%	63.22%	72.63%	73.76%

Tamper	Method	*grep*	*ln*	*mkdir*	*mv*
Kill	TPE-Det	**97.82%**	**99.06%**	**99.13%**	**99.21%**
	Syslog	0%	0%	0%	0%
Del-to-50	TPE-Det	**97.43%**	**99.06%**	**99.13%**	**99.21%**
	Syslog	68.61%	76.21%	77.06%	52.13%

Tamper	Method	*rm*	*rmdir*	*touch*	*wget*
Kill	TPE-Det	**99.75%**	**99.64%**	**99.53%**	**99.67%**
	Syslog	0%	0%	0%	0%
Del-to-50	TPE-Det	**99.75%**	**99.64%**	**99.53%**	**99.62%**
	Syslog	45.62%	69.23%	76.93%	75.36%

(or 100) lines of generated logs after each malware execution to simulate adversary tampering, e.g., *cat /dev/null > /var/*log*/syslog*, execute the malware script, and *cat /var/*log*/syslog | head -n* 50 > *save.txt*. The log recovery results of TPE-Det and *Syslog* against the kill processes and tampering with logs are summarized in Table III. We find that *Syslog* is greatly affected by tampering and fails directly when killed (corresponding to 0% command recovery in *Syslog*). However, TPE-Det is almost unaffected whether by tampering or killing the process, i.e., achieve 97.25%–99.75% command recovery. Compare the results between "Kill" and "Del-to-50," we find that the command recovery results of TPE-Det for *chmod*, *ln*, *mkdir*, *mv*, *rm*, *rmdir*, and *touch* are consistent. The recovery ratio of other commands changes because the log deletion operation may involve those commands. For the right part of Fig. 9, it is clear that all methods based on HPC (*perf*), *Syscall*, and *Syslog* are severely affected when tampered with and will fail when killed. In comparison, TPE-Det always maintains superior detection, which echoes our original intention of such a tamper-proof design.

E. Deep Insights Into TPE-Det

In this section, we provide here more deep insights into TPE-Det with respect to cross-architecture analysis, operation recovery case, and unknown detection.

1) Cross-Architecture Analysis: A variety of architectures is typically used on IoT devices, such as x86, ARM, MIPS, etc., thus cross-architecture analysis is practical in the real world [72]. We analyze all malware instances in the dataset and find that their command structures are completely isomorphic when different architectures are involved, as the top part of Fig. 10 shows. Specifically, Fig. 10(a) corresponds to the malware with MD5 is f6ff16d9b855beae3fcfb7d272c34582. There are a series of executable files for different architectures involve MIPS, SH4, x86, ARM, and so on. Nonetheless, their command patterns are the same, refer to *wget* → *chmod* → *program execution* → *rm*. This means that our proposal

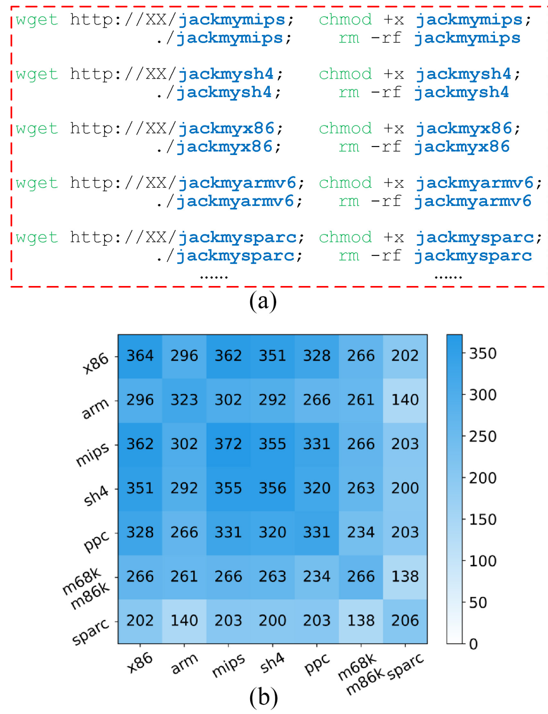


Fig. 10. Cross-architecture malware analysis. a) Arch-homogeneous malware. (b) Cross-arch heatmap.

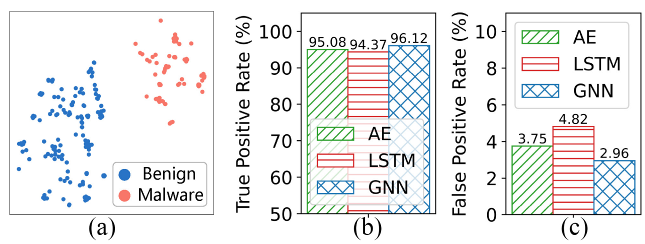


Fig. 11. Visualization and unknown detection evaluation. (a) t-SNE visualization. (b) TPR results. (c) FPR results.

supports cross-architecture malware detection. Furthermore, we count the frequency of various architectures appearing simultaneously to draw the heatmap in Fig. 10(b). We observe that *x*86, *mips*, *sh*4, and *ppc* are the most common.

2) Recovery Case: We describe the log recovery case for Hajime [6] and Mirai [5], more details are stored in the online repository..

3) Unknown Detection: We also explore unknown detection effects, statistical features are modeled with autoencoder (AE), both LSTM and GNN are also reconstructed into encoder + decoder architectures. Then, training with benign only and performing anomaly detection based on reconstruction loss. Fig. 11 shows the feasibility of unknown detection, and the t-SNE visualization indicates that benign and malware are distinguishable under our feature space.

F. Overhead Evaluations

We measure the time and space overhead here. In Fig. 12, our ML classifiers introduce minimal time overhead, our DNN models are the same level as the HPC-based model

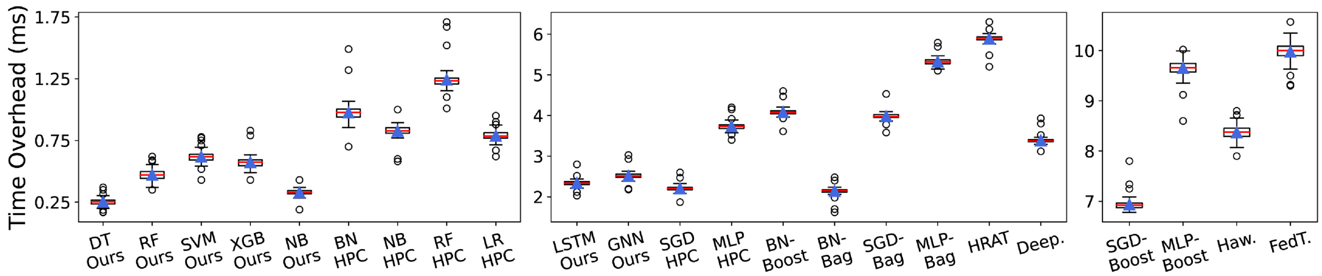


Fig. 12. Time overhead of TPE-Det and baselines.

ensemble. The most time-consuming is FedTrans because it is based on the Transformer model. For the space overhead, our models only induce 70.31–327.17 KB, the most space-consuming is still FedTrans since it contains massive parameters. Noteworthy, among various log records in Table IV, ours is the smallest, only needing 4.31 MB for all benign/malware samples combined, while *Perf*, *Syscall*, and *Syslog* all require more than 100 MB. For log recovery time, it takes ∼0.02 s to analyze 5 MB SPI signals, which is acceptable due to usually there are not so many traces generated.

G. Extended Experiments

We also perform extended experiments with different chips and datasets. Specifically, we use another chip that installs the W25X64 Flash [73] and SPI bus. Meanwhile, we consider the BadThings [74] dataset, which contains binary malware executables. The evaluation results of the GNN model in TPE-Det are reported in Fig. 14. We observe that for different chips, there is almost no impact on the performance of TPE-Det, and the slight differences in classification metrics may be due to factors, such as the changed frequency, during the signal acquisition process. For the BadThings dataset, the performance of TPE-Det decreases slightly (e.g., ∼1.2% accuracy drop), probably because the BadThings corpora [74] contains more malware samples. Overall, TPE-Det can detect malware accurately and has good scalability, such as applying to different chips and datasets.

TABLE IV
SPACE OVERHEAD OF LOGS AND MODELS

Log space overhead		Ours	perf	Syscall	Syslog
9,337 Benign		**1.45MB**	213.34MB	532.15MB	71.23MB
3,439 Malware		**2.86MB**	348.92MB	987.37MB	136.88MB
Model (Ours)	**Size**	**Model**	**Size**	**Model**	**Size**
DT	70.31KB	BN (HPC)	567.78KB	BN-Boost	2.33MB
RF	98.52KB	NB (HPC)	441.82K	BN-Bag	2.24MB
SVM	132.17KB	RF (HPC)	721.73KB	SGD-Boost	4.95MB
XGB	154.55KB	LR (HPC)	357.29KB	SGD-Bag	3.69MB
NB	84.96KB	SGD (HPC)	695.78KB	MLP-Boost	8.84MB
LSTM	327.17KB	MLP (HPC)	2.83MB	MLP-Bag	7.51MB
GNN	251.34KB	Hawkware	2.75MB	HRAT	1.39MB
–	–	FedTrans	160.33MB	Deeplog	532.18KB

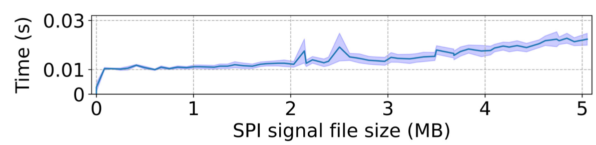


Fig. 13. Log recovery time of the hardware trace.

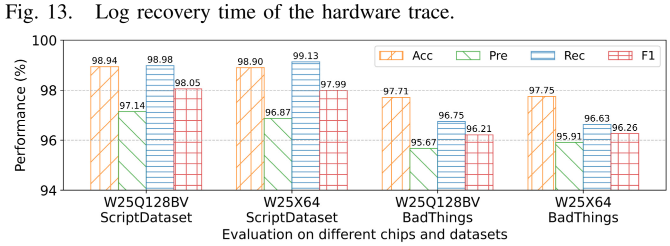


Fig. 14. Evaluate TPE-Det (GNN) on different chips and datasets.

VII. DISCUSSION, LIMITATIONS, AND FUTURE WORK

A. Security Discussion

As mentioned in the introduction, TPE-Det is dedicated to performing the analysis of the side-channel information external to the system, which can address the challenges of tampering risks and huge logging records. In this way, attackers are difficult to perceive the monitoring process or manipulate the logs, so that the detection results of TPE-Det can be guaranteed. Actually, physical access does increase the possible attack surface [75]. While we would like to clarify that considering security scenarios involving investigation forensics and honeypots (as stated in Section III), such physical access is acceptable in the real world [26], [76]. In view of scenarios, such as forensics and honeypots, practitioners are allowed to have many permissions on the equipment, including physical access support [45], [46], [52], to obtain as much attack intelligence and information as possible [47], [51]. Also, the Logic analyzer in TPE-Det prototype connects the flash chip physically through pins so that the IoT device system will not be directly affected. It can also be considered for devising integrated customized solutions for production in the future.

B. Practicality and Expandability

As experimented in Section VI-B, it is undeniable that TPE-Det may yield some misclassifications, even if TPE-Det proposes three designed models for the statistical, sequence, and graph features. When customers have a low tolerance for false negatives, we can improve the positive probability of the classification or ensemble multiple models (e.g., voting result is negative only if no positive). Meanwhile, TPE-Det can leverage the traceability forensic analysis [21] to verify and determine the detection reports based on the recovered hardware-part operation. In addition, the high-level detection logic in TPE-Det can be extended according to specific business scenarios. For example, we could focus on choosing the appropriate model of graph features for security applications that centered on continuous and various file operations. In addition to forensics and IoT honeypots, other scenarios that require recording correct device operation logs against adaptive adversaries can also benefit from TPE-Det.

C. Limitations and Future Work

Although the evaluations mainly use the ScriptDataset dataset, other datasets, such as BinaryString and OpenWrtLogs, are still applicable given that previous work has extensively confirmed that malware does exhibit certain command patterns. We admit that physical access may not be convenient sometimes, but as a novel external monitor, TPE-Det can bring new perspectives. Meanwhile, a feasible direction is to devise integrated customized solutions for industrial production to advance this side-channel-manner analysis.

VIII. CONCLUSION

In this article, we propose TPE-Det, a tamper-proof and lightweight external malware detector. In particular, TPE-Det leverages the SPI bus to monitor and extract the on-chip traces and we design a suite of operation log recovery pipeline in a side-channel manner. We implement TPE-Det and evaluate it extensively on our physical testbed. By comparing the SOTA methods involving HPC, ETB, *Syscall*, and *Syslog*, we demonstrate that TPE-Det can achieve remarkable detection results even against adaptive adversaries. Meanwhile, TPE-Det introduces negligible CPU and memory utilization. Furthermore, we develop a series of experiments in terms of concept drift, deep insights, cross-architecture analysis, unknown detection, and overhead evaluation to present the effectiveness, stability, scalability, lightness, and practicality of TPE-Det.

REFERENCES

[1] E. Cozzi, M. Graziano, Y. Fratantonio, and D. Balzarotti, “Understanding Linux malware,” in *Proc. IEEE Symp. Security Privacy*, 2018, pp. 161–175.

[2] X. Zhang, M. Hu, J. Xia, T. Wei, M. Chen, and S. Hu, “Efficient federated learning for cloud-based AIoT applications,” *IEEE Trans. Comput. Aided Design Integr. Circuits Syst.*, vol. 40, no. 11, pp. 2211–2223, Nov. 2021.

[3] M. Hu, E. Cao, H. Huang, M. Zhang, X. Chen, and M. Chen, “AIoTML: A unified modeling language for AIoT-based cyber-physical systems,” *IEEE Trans. Comput. Aided Design Integr. Circuits Syst.*, vol. 42, no. 11, pp. 3545–3558, Nov. 2023.

[4] O. Alrawi, C. Lever, K. Valakuzhy, K. Snow, F. Monrose, and M. Antonakakis, “The circle of life: A large-scale study of the IoT malware lifecycle,” in *Proc. USENIX Security Symp.*, 2021, pp. 3505–3522.

[5] M. Antonakakis et al., “Understanding the Mirai botnet,” in *Proc. 26th USENIX Security Symp.*, 2017, pp. 1093–1110.

[6] S. Herwig, K. Harvey, G. Hughey, R. Roberts, and D. Levin, “Measurement and analysis of Hajime, a peer-to-peer IoT botnet,” in *Proc. NDSS*, 2019, pp. 1–15.

[7] X. Ling et al., “DDoSMiner: An automated framework for DDoS attack characterization and vulnerability mining,” in *Proc. 22nd ACNS*, 2024, pp. 283–309.

[8] Z. Zhao, Z. Li, F. Zhang, T. Li, and J. Yin, “Poster: Combine topology and traffic to calibrate P2P botnet identification in large-scale network,” in *Proc. ACM SIGCOMM Posters Demos*, 2024, pp. 16–18.

[9] Z. Zhao, Z. Liu, H. Chen, F. Zhang, Z. Song, and Z. Li, “Effective DDoS mitigation via ML-driven in-network traffic shaping,” *IEEE Trans. Dependable Secure Comput.*, vol. 21, no. 4, pp. 4271–4289, Jul./Aug. 2024.

[10] S. Ahn, H. Yi, Y. Lee, W. R. Ha, G. Kim, and Y. Paek, “Hawkware: Network intrusion detection based on behavior analysis with ANNs on an IoT device,” in *Proc. 57th ACM/IEEE DAC*, 2020, pp. 1–6.

[11] M. Du, F. Li, G. Zheng, and V. Srikumar, “DeepLog: Anomaly detection and diagnosis from system logs through deep learning,” in *Proc. ACM SIGSAC CCS*, 2017, pp. 1285–1298.

[12] K. Zhao et al., “Structural attack against graph based android malware detection,” in *Proc. ACM SIGSAC CCS*, 2021, pp. 3218–3235.

[13] W. Cui, V. Paxson, N. Weaver, and R. H. Katz, “Protocol-independent adaptive replay of application dialog,” in *Proc. NDSS*, 2006, pp. 1–15.

[14] L. Babun, A. K. Sikder, A. Acar, and A. S. Uluagac, “The truth shall set thee free: Enabling practical forensic capabilities in smart environments,” in *Proc. NDSS*, 2022, pp. 1–17.

[15] R. Paccagnella, K. Liao, D. Tian, and A. Bates, “Logging to the danger zone: Race condition attacks and defenses on system audit frameworks,” in *Proc. ACM SIGSAC CCS*, 2020, pp. 1551–1574.

[16] H. Li et al., “Understanding and detecting remote infection on Linux-based IoT devices,” in *Proc. ACM ASIA CCS*, 2022, pp. 873–887.

[17] N. Patel, A. Sasan, and H. Homayoun, “Analyzing hardware based malware detectors,” in *Proc. ACM 54th DAC*, 2017, pp. 1–6.

[18] H. Sayadi, N. Patel, P. D. S. Manoj, A. Sasan, S. Rafatirad, and H. Homayoun, “Ensemble learning for effective run-time hardware-based malware detection: A comprehensive analysis and classification,” in *Proc. 55th ACM/ESDA/IEEE DAC*, 2018, pp. 1–6.

[19] “Perf tutorial.” Perf. 2023. [Online]. Available: https://perf.wiki.kernel.org/index.php/Tutorial

[20] E. Allman. “Syslog.” 2023. [Online]. Available: https://en.wikipedia.org/wiki/Syslog

[21] Z. Zhao et al., “CMD: Co-analyzed IoT malware detection and forensics via network and hardware domains,” *IEEE Trans. Mobile Comput.*, vol. 23, no. 5, pp. 5589–5603, May 2024.

[22] M. Botacin, P. L. D. Geus, and 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. 1–34, 2018.

[23] Q. Wang et al., “You are what you do: Hunting stealthy malware via data provenance analysis,” in *Proc. NDSS*, 2020, pp. 1–17.

[24] J. Demme et al., “On the feasibility of online malware detection with performance counters,” in *Proc. ACM ISCA*, 2013, pp. 559–570.

[25] M. Botacin and A. Grégio, “Why we need a theory of maliciousness: Hardware performance counters in security,” in *Proc. 25th ISC*, 2022, pp. 381–389.

[26] K. Basu, R. Elnaggar, K. Chakrabarty, and R. Karri, “PREEMPT: PReempting malware by examining embedded processor traces,” in *Proc. 56th ACM/IEEE DAC*, 2019, pp. 1–6.

[27] B. Zhou, A. Gupta, R. Jahanshahi, M. Egele, and A. Joshi, “Hardware performance counters can detect malware: Myth or fact?” in *Proc. ASIA CCS*, 2018, pp. 457–468.

[28] A. Costin, J. Zaddach, A. Francillon, and D. Balzarotti, “A large-scale analysis of the security of embedded firmwares,” in *Proc. 23rd USENIX Security Symp.*, 2014, pp. 95–110.

[29] G. De 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 *Proc. NDSS*, 2022, pp. 1–16.

[30] Z. B. Celik, G. Tan, and P. D. McDaniel, “IOTGUARD: Dynamic enforcement of security and safety policy in commodity IoT,” in *Proc. NDSS*, 2019, pp. 1–15.

[31] M. N. Hossain et al., “Dependence-preserving data compaction for scalable forensic analysis,” in *Proc. 27th USENIX Security Symp.*, 2018, pp. 1723–1740.

[32] M. N. Hossain, S. Sheikhi, and R. Sekar, “Combating dependence explosion in forensic analysis using alternative tag propagation semantics,” in *Proc. IEEE Symp. Security Privacy (SP)*, 2020, pp. 1139–1155.

[33] L. Li et al., “An automated alert cross-verification system with graph neural networks for IDS events,” in *Proc. 27th CSCWD*, 2024, pp. 2240–2245.

[34] Z. Li et al., “metaNet: Interpretable unknown mobile malware identification with a novel meta-features mining algorithm,” *Comput. Netw.*, vol. 250, Aug. 2024, Art. no. 110563.

[35] Z. Zhao, Z. Li, Z. Song, W. Li, and F. Zhang, “Trident: A universal framework for fine-grained and class-incremental unknown traffic detection,” in *Proc. ACM Web Conf.*, 2024, pp. 1608–1619.

[36] A. Saha, N. Ganguly, S. Chakraborty, and A. De, "Learning network traffic dynamics using temporal point process," in *Proc. IEEE INFOCOM*, 2019, pp. 1927–1935.

[37] Z. Zhao et al., "DDoS family: A novel perspective for massive types of DDoS attacks," *Comput. Security*, vol. 138, Mar. 2024, Art. no. 103663.

[38] M. Shen, M. Wei, L. Zhu, and M. Wang, "Classification of encrypted traffic with second-order Markov chains and application attribute bigrams," *IEEE Trans. Inf. Forensics Security*, vol. 12, pp. 1830–1843, 2017.

[39] Z. Zhao et al., "ERNN: Error-resilient RNN for encrypted traffic detection towards network-induced phenomena," *IEEE Trans. Dependable Secure Comput.*, early access, Feb. 3, 2023, doi: 10.1109/TDSC.2023.3242134.

[40] Z. Song et al., "I2RNN: An incremental and interpretable recurrent neural network for encrypted traffic classification," *IEEE Trans. Dependable Secure Comput.*, early access, Feb. 28, 2023, doi: 10.1109/TDSC.2023.3245411.

[41] Z. Zhao, Z. Li, Z. Song, and F. Zhang, "Work-in-progress: Towards real-time IDS via RNN and programmable switches co-designed approach," in *Proc. IEEE RTSS*, 2023, pp. 431–434.

[42] T. Gu, Z. Fang, A. Abhishek, H. Fu, P. Hu, and P. Mohapatra, "IoTGaze: IoT security enforcement via wireless context analysis," in *Proc. IEEE INFOCOM*, 2020, pp. 884–893.

[43] X. Wang, Y. Sun, S. Nanda, and X. F. Wang, "Looking from the mirror: Evaluating IoT device security through mobile companion apps," in *Proc. 28th USENIX Security Symp.*, 2019, pp. 1151–1167.

[44] "Mirai-source-code." Accessed: Oct. 2, 2023. [Online]. Available: https://github.com/jgamblin/Mirai-Source-Code

[45] P. Fei, Z. Li, Z. Wang, X. Yu, D. Li, and K. Jee, "SEAL: Storage-efficient causality analysis on enterprise logs with query-friendly compression," in *Proc. 30th USENIX Security Symp.*, 2021, pp. 2987–3004.

[46] L. Yu et al., "ALchemist: Fusing application and audit logs for precise attack provenance without instrumentation," in *Proc. NDSS*, 2021, pp. 1–18.

[47] Q. Wang, W. U. Hassan, A. Bates, and C. Gunter, "Fear and logging in the Internet of Things," in *Proc. NDSS*, 2018, pp. 1–16.

[48] J. Haseeb, M. Mansoori, and I. Welch, "A measurement study of IoT-based attacks using IoT kill chain," in *Proc. IEEE 19th Int. Conf. Trust, Security Privacy Comput. Commun. (TrustCom)*, 2020, pp. 557–567.

[49] S. Ul Haq, Y. Singh, A. Sharma, R. Gupta, and D. Gupta, "A survey on IoT & embedded device firmware security: Architecture, extraction techniques, and vulnerability analysis frameworks," *Discov. Internet Things*, vol. 3, no. 1, p. 17, 2023.

[50] M. S. Awal and M. T. Rahman, "Disassembling software instruction types through impedance side-channel analysis," in *Proc. IEEE HOST*, 2023, pp. 227–237.

[51] H. Griffioen and C. Doerr, "Examining Mirai's battle over the Internet of Things," in *Proc. ACM SIGSAC CCS*, 2020, pp. 743–756.

[52] Z. Yin, Y. Xu, C. Zhou, and Y. Jiang, "Empirical study of system resources abused by IoT attackers," in *Proc. 37th IEEE/ACM ASE*, 2023, pp. 1–13.

[53] S. Valizadeh and M. Van Dijk, "MalPro: A learning-based malware propagation and containment modeling," in *Proc. ACM SIGSAC CCSW*, 2019, pp. 45–56.

[54] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "DREBIN: Effective and explainable detection of android malware in your pocket," in *Proc. NDSS*, 2014, pp. 1–15.

[55] "W25Q128FV: Winbond." 2013. [Online]. Available: https://www.pjrc.com/teensy/W25Q128FV.pdf

[56] (Winbond Corp., Taichung, Taiwan). *Code Storage Flash Memory*. (2023). [Online]. Available: https://www.winbond.com/hq/product/code-storage-flash-memory/?__locale=en

[57] (Saleae, San Francisco, CA, USA). *Saleae Logic Analyzer*. (2023). [Online]. Available: https://www.saleae.com/

[58] D. Woodhouse, "JFFS: The journalling flash file system," in *Proc. Ottawa Linux Symp.*, 2001, pp. 1–12.

[59] K. Liu et al., "On manually reverse engineering communication protocols of Linux-based IoT systems," *IEEE Internet Things J.*, vol. 8, no. 8, pp. 6815–6827, Apr. 2021.

[60] (Free Softw. Found. Inc., Boston, MA, USA). *The GNU C Library Reference Manual*. (2023). [Online]. Available: https://www.gnu.org/software/libc/manual/pdf/libc.pdf

[61] F. Alwahedi, A. Aldhaheri, M. A. Ferrag, A. Battah, and N. Tihanyi, "Machine learning techniques for IoT security: Current research and future vision with generative AI and large language models," *Internet Things Cyber-Phys. Syst.*, vol. 4, pp. 167–185, Jan. 2024.

[62] H. El-Sofany, S. A. El-Seoud, O. H. Karam, and B. Bouallegue, "Using machine learning algorithms to enhance IoT system security," *Sci. Rep.*, vol. 14, no. 1, 2024, Art. no. 12077.

[63] L. Xiao, X. Wan, X. Lu, Y. Zhang, and D. Wu, "IoT security techniques based on machine learning: How do IoT devices use AI to enhance security?" *IEEE Signal Process. Mag.*, vol. 35, no. 5, pp. 41–49, Sep. 2018.

[64] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," 2014, *arXiv:1412.3555*.

[65] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *Proc. ICLR*, 2019, pp. 1–17.

[66] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Netw.*, vol. 4, no. 2, pp. 251–257, 1991.

[67] K. Xu, C. Li, Y. Tian, T. Sonobe, K. Kawarabayashi, and S. Jegelka, "Representation learning on graphs with jumping knowledge networks," in *Proc. 35th ICML*, 2018, pp. 5449–5458.

[68] "OpenWrt Porject." OpenWrt. 2021. [Online]. Available: https://openwrt.org/

[69] F. Pendlebury, F. Pierazzi, R. Jordaney, J. Kinder, and L. Cavallaro, "TESSERACT: Eliminating experimental bias in malware classification across space and time," in *Proc. 28th USENIX Security Symp.*, 2019, pp. 729–746.

[70] M. I. Sharif, A. Lanzi, J. T. Giffin, and W. Lee, "Impeding malware analysis using conditional code obfuscation," in *Proc. NDSS*, 2008, pp. 1–13.

[71] J. Garcia, M. Hammad, and S. Malek, "Lightweight, obfuscation-resilient detection and family identification of android malware," in *Proc. 40th ICSE*, 2018, p. 497.

[72] D. Vasan, M. Alazab, S. Venkatraman, J. Akram, and Z. Qin, "MTHAEL: Cross-architecture IoT malware detection based on neural network advanced ensemble learning," *IEEE Trans. Comput.*, vol. 69, no. 11, pp. 1654–1667, Nov. 2020.

[73] "W25X64: Winbond." 2013. [Online]. Available: https://www.alldatasheet.com/datasheet-pdf/pdf/207472/WINBOND/W25X64.html

[74] "A labeled dataset with malicious and benign IoT network traffic." Accessed: Jan. 5, 2024. [Online]. Available: https://badthings.info/

[75] S. Etigowni, D. Tian, G. Hernandez, S. Zonouz, and K. Butler, "CPAC: Securing critical infrastructure with cyber-physical access control," in *Proc. 32nd Annu. Conf. Comput. Security Appl.*, 2016, pp. 139–152.

[76] A. Kumar and T. J. Lim, "EDIMA: Early detection of IoT malware network activity using machine learning techniques," in *Proc. IEEE 5th WF-IoT*, 2019, pp. 289–294.