Incorporating Gradients to Rules: Towards Lightweight, Adaptive Provenance-based Intrusion Detection

Lingzhi Wang*†, Xiangmin Shen*†, Weijian Li*, Zhenyuan Li‡, R. Sekar§, Han Liu*, and Yan Chen*
*Northwestern University, ‡Zhejiang University, §Stony Brook University
{lingzhiwang2025, xiangminshen2019, weijianli}@u.northwestern.edu, {hanliu, ychen}@northwestern.edu
lizhenyuan@zju.edu.cn, sekar@cs.stonybrook.edu

***Abstract*—As cyber attacks grow increasingly sophisticated and stealthy, it becomes more imperative and challenging to detect intrusion from normal behaviors. Through fine-grained causality analysis, provenance-based intrusion detection systems (PIDS) demonstrated a promising capacity to distinguish benign and malicious behaviors, attracting widespread attention from both industry and academia. Among diverse approaches, rule-based PIDS stands out due to its lightweight overhead, real-time capabilities, and explainability. However, existing rule-based systems suffer low detection accuracy, especially the high false alarms, due to the lack of fine-grained rules and environment-specific configurations.**

In this paper, we propose CAPTAIN, a rule-based PIDS capable of automatically adapting to diverse environments. Specifically, we propose three adaptive parameters to adjust the detection configuration with respect to nodes, edges, and alarm generation thresholds. We build a differentiable tag propagation framework and utilize the gradient descent algorithm to optimize these adaptive parameters based on the training data. We evaluate our system using data from DARPA Engagements and simulated environments. The evaluation results demonstrate that CAPTAIN enhances rule-based PIDS with learning capabilities, resulting in improved detection accuracy, reduced detection latency, lower runtime overhead, and more interpretable detection procedures and results compared to the state-of-the-art (SOTA) PIDS.

I. INTRODUCTION

Advanced Persistent Threats (APT) are becoming a growing threat to both government and industrial sectors, causing significant societal impacts [1]. The Equifax breach in 2017 resulted in the theft of vast amounts of personal data, highlighting the severe privacy and security risks [2], [3]. Moreover, attackers continuously innovate to find new ways to penetrate systems and remain undetected for extended periods. In recent years, provenance-based intrusion detection systems (PIDS) have gained attention from both the security industry and academia for their causality analysis capability. However, prevailing alarm fatigue, excessive runtime overhead, long detection latency, and opaque detection processes (based on black-box) are still open research problems in real-world scenarios [4], [5], [6], [7], [8].

Alarm fatigue [9], [10] is a significant issue plaguing the security industry. A recent survey [11] indicates that security analysts are required to handle an overwhelming average of 5,000 alarms daily, with a majority being false alarms. The excessive volume of alarms can lead to serious consequences. For instance, in the 2013 Target data breach [2], the malicious activities were detected and reported by security tools but overlooked by analysts, resulting in delayed response and expanded losses [12]. In practice, false alarms are equally detrimental as missed alarms. Moreover, significant challenges also arise from runtime overhead and detection latency. High runtime overhead compromises the scalability of detection systems, hindering their deployment on a large scale [4]. Moreover, detection latency critically affects performance, as prolonged latency delays the response to threats, thereby impeding analysts' efficiency in managing alarms.

In academia, a recent trend of PIDS [13], [14], [15], [16] leverage embedding techniques like word2vec [16] and graph2vec [14] to encode system entities and events, and neural networks like Graph Neural Networks (GNNs) [16], [15], [13], [17] and Recurrent Convolutional Neural Networks (RCNNs) [14] to analyze information flow and produce detection results. We refer to them as *embedding-based PIDS*. Although these systems have demonstrated notable achievements in detection performance, they face the following challenges:

- **High Computational Resource Cost.** Embedding-based systems need to embed the graph features into vectors, usually with deep learning techniques [14], [16], [13], [15], which require a significant amount of computational resources. Moreover, many systems require caching of embeddings and deep learning models, leading to high memory consumption [4];
- **Long Detection Latency.** Embedding-based systems take the input structured as graphs or paths, requiring time windows [15], [18] or log batches [16], [14] in their design, which leads to the additional detection delay.
- **Uninterpretable Results.** Many systems [19], [13], [20], [16], [15], [18] only flag deviations from normal behaviors without attack semantics. Moreover, we are unable to open the

†The first two authors made equal contribution.

Network and Distributed System Security (NDSS) Symposium 2025
23 - 28 February 2025, San Diego, CA, USA
ISBN 979-8-9894372-8-3
https://dx.doi.org/10.14722/ndss.2025.23822
www.ndss-symposium.org

"black box" of the deep learning models to better understand the detection process.

On the other hand, a group of PIDS [21], [22], [23], [24], [7] operates based on human-defined mechanisms to generate representations for system entities, propagate information flow, and trigger alarms. We refer to them as *rule-based PIDS*. Leveraging these rules, they take actionable leads, allowing themselves to provide semantic-rich alarms and pinpoint specific suspicious events for further investigation. Besides, the rule-based approach operates on simple arithmetic calculations, which require far fewer computational resources than complex matrix calculations by neural networks. Furthermore, some rule-based approaches process streaming data once it arrives, significantly reducing storage requirements and enabling rapid response with minimal delay. In summary, rule-based PIDS shows advantages in real-world detection tasks due to its fine-grained detection, semantic-rich alarms, lightweight overhead, and minimal detection delay.

Despite rule-based PIDS excelling in the aforementioned perspectives, they also face challenges in real-world deployment. These systems often employ simplistic and universal rules to distinguish between graphs (such as discrete trustworthiness levels of node in [22], [21], [23], universal system parameters in [22], [21], [23], [7]). The inflexible rules lead these PIDS to be either too lenient or too strict. In security operations centers (SOCs), analysts have to manually configure the models, which is a time-consuming process [25]. This highlights the need for an automated configuration methodology and scheme. To achieve this, a rule-based PIDS must be capable of adjusting its rules autonomously based on detection feedback from the training set, without human intervention. This requires an automatic feedback mechanism that establishes a direct connection between the rules and detection results, allowing the system to dynamically adjust its configuration based on the detection outcomes.

In this paper, we propose CAPTAIN, a rule-based PIDS with adaptive configuration learning capability. We aim to leverage the advantages of traditional rule-based PIDS while enabling the system to acquire suitable configurations autonomously. Specifically, we introduce three adaptive parameters to the rule-based PIDS, giving it more flexibility during detection. Moreover, we design a learning module to adjust these parameters automatically based on the detection results during training. We calculate and record the gradients of each adaptive parameter, transferring the rule-based system to a differentiable function. With the help of a loss function, the gradient descent algorithm can be utilized to optimize the adaptive parameters based on the benign training data and thus reduce the false alarms in testing.

We evaluate CAPTAIN in diverse detection scenarios, which are drawn from the widely acknowledged public datasets [26] provided by DARPA Engagement [27] and datasets collected from simulated environments in collaboration with a SOC. The evaluation results demonstrate that CAPTAIN can reduce false alarms by over 90% (11.49x) on average compared to traditional rule-based PIDS. Additionally, CAPTAIN achieves much better detection accuracy with less than 10% CPU usage and significantly lower memory usage and latency compared to the SOTA embedding-based PIDS. We then select a few scenarios as cases to study the explainability of the learned configurations.

In summary, this paper makes the following contributions:

- We propose CAPTAIN, a rule-based PIDS that can adjust its rules using benign data to reduce false alarms. Our design endows CAPTAIN with the lightweight nature, low latency, and interpretability of rule-based PIDS while also having fine detection capability and adaptability.
- We parameterize the rule-based PIDS and propose a novel differentiable tag propagation framework that allows us to optimize adaptive parameters using the gradient descent algorithm.
- We systemically evaluate CAPTAIN across various scenarios, including widely used DARPA datasets and simulated datasets from a partner SOC. The evaluation result demonstrates that CAPTAIN can automatically adapt to various environments, offering superior detection accuracy with reduced latency, lower overhead, and more interpretable detection.

We make the code of our system, all datasets, and all experiments publicly available for future analysis and research[1].

II. BACKGROUND AND RELATED WORK

In this section, we begin by introducing PIDS and highlighting mainstream approaches to clarify our design choices. Next, we provide background on the optimization problem, followed by a description of the threat model and the assumptions.

A. Provenance-based Intrusion Detection

Provenance-based intrusion detection has drawn wide attention for its powerful correlation and causal analysis capabilities. In provenance analysis, the system behaviors are modeled as directed acyclic graphs, called provenance graphs [28], [29], in which nodes represent system entities, such as processes, files, sockets, pipes, memory objects, etc., and edges represent interactions between entities, such as reading files and connecting to a remote host.

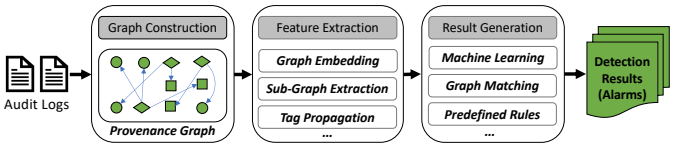


Fig. 1: A brief overview of the workflow and commonly used techniques in mainstream PIDS.

Fig. 1 illustrates an overview of typical PIDS workflow. Based on how to model and aggregate the entity information, the data/control flow, and the system behaviors, we identify two major groups of PIDS: the embedding-based approach and the rule-based approach.

[1]https://github.com/LexusWang/CAPTAIN

Embedding-based PIDS. Embedding-based systems [18], [19], [30], [31], [13], [16], [15], [17] use numerical vectors to represent, propagate, and aggregate the information of system entities, events, data/control flow in the graph. Various techniques in machine learning have been employed to embed the graph features into numerical vectors. Since these techniques typically require the input structured as graphs or paths, embedding-based PIDS usually adopt time windows or log batches in their design, leading to the additional detection delay, which we named as ***buffer time*** in this paper. Concretely, buffer time refers to the time window duration in which all streaming logs are stored to create a graph structure for subsequent processing. Previous studies have implemented a fixed-length window based on time [15] or the number of events [16], while some works [14] utilize the variable length window. Buffer time is crucial in real-world detection scenarios because, in the worst-case scenario, after an attack has been launched, the PIDS would have to wait for the duration of the buffer time before starting the detection procedure.

The methods for embedding graphs differ across various systems. For instance, PROVDETECTOR [32] selects rare paths based on historical event frequency to generate embedding vectors for sequence learning and anomaly detection. ATLAS [31] classifies the extracted sequences from the graph with Long Short-term Memory (LSTM). UNICORN[19] embeds the graph histograms and clusters the embedded graph sketches. WATSON [30] learns the node embedding and event semantics from training traces. More recent embedding-based systems [16], [15], [17] employed graph neural networks in detection. While such systems can achieve better detection accuracy, they fail to provide the rationale for generated alarms. For example, SHADEWATCHER [13] provides the probability of malicious system events without further context. FLASH [16] highlights suspicious entities with causal links but does not provide further attack semantics of those interactions. In practice, more manual efforts are required to review the relevant audit logs thoroughly for further response.

Rule-based PIDS. Rule-based systems [21], [24], [23], [22], [7] leverage human-designed rules to map system entities and events to predefined semantic units (e.g., tags [21], [22], [24], TTP [23], etc.) and propagation these units along the information flow to capture causality. NODOZE[7] assigns an anomaly score to each edge based on the frequency it appears in the benign data and propagates the scores on the graph. POIROT [33] models the detection as a graph-matching problem that aligns manually constructed query graphs on the provenance graph. SLEUTH [21] and MORSE [22] assign tags to system entities and propagate tags among the graph. Rule-based systems have become popular due to their real-time processing capabilities and lightweight properties. However, they usually suffer low detection accuracy due to the lack of fine-grained rules and environment-specific configurations. Most rule-based PIDS [22], [21], [24], [23] implement simple and universal rules, making them inflexible to adapt to different environments. These systems typically rely on a discrete classification to denote system entity properties, such as private/public, and trusted/untrusted, and apply identical rules to all nodes and events. As a result, they lack the nuance required to accurately distinguish between similar graphs, often being either overly lenient or overly strict, which leads to false alarms or missed true alarms. We will elaborate on these challenges in §III.

Table I summarizes the detection granularity of existing PIDS and indicates whether the detection requires buffer time.

In this paper, we build our system CAPTAIN following the rule-based methodology for four reasons: 1) rule-based detection is computationally more efficient [33], [19], [21]; 2) rule-based methods are suitable for real-time detection because they can process the event stream incrementally [29]; 3) rule-based methods offer more explainability to the detection process and results than embedding-based methods [34]; 4) rule-based methods are robust against mimicry attacks [35]. Additionally, CAPTAIN introduces a significant advancement over traditional rule-based PIDS. Unlike systems that rely on simple, universal, and inflexible rules, CAPTAIN offers a mathematically complete framework for automatically fine-tuning detection rules, enabling more precise detection and response to security threats.

TABLE I: Comparison of existing PIDS. Buffer time refers to the waiting time before threat analysis.

	Detection Granularity	Require Buffer Time
CAPTAIN	Edge	No
FLASH	Node	Yes
R-CAID	Node	Yes
KAIROS	Graph[1]	Yes
NODLINK	Node	Yes
PROGRAPHER	Node	Yes
SHADEWATCHER	Edge	Yes
POIROT	Path	Yes
MORSE	Edge	No
UNICORN	Graph	Yes
HOLMES	Edge	No
PROVDETECTOR	Path	Yes
NODOZE	Path	Yes

[1] Although KAIROS triggers alarms on the graph level, it can highlight anomaly nodes and edges in the subgraphs.

B. Configuration as an Optimization Problem

Finding the best configuration for a rule-based PIDS can be formalized as an optimization problem, which, in short, aims to find the best elements θ in a searching space to minimize or maximize an objective function $J(\theta)$. Gradient descent is one of the most common algorithms to solve the optimization problem [36]. It utilizes the gradient of the objective function to the parameters $\nabla_\theta J(\theta)$ to update θ according to $\theta = \theta - l \cdot \nabla_\theta J(\theta)$, where l is the learning rate. Many variants have been proposed based on the gradient descent algorithm. We discuss more algorithms and their applicability to CAPTAIN in § VII-B.

While the optimization problem and its gradient-based solutions play a significant role in quantitative science and

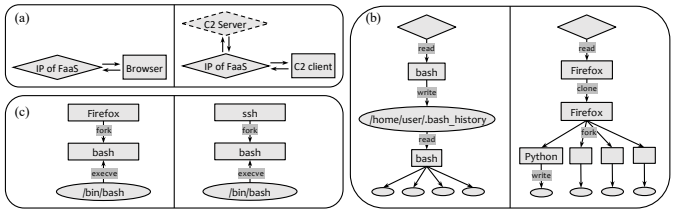


Fig. 2: Three motivating examples from the real-world dataset where the more fine-grained rules are needed in the rule-based PIDS.

engineering [37], it is rarely discussed in the previous rule-based PIDS because of the following challenges. First, no existing work formally defined the adjustable parameters with clear meanings in the context of rule-based PIDS. Second, designing an objective function, calculating, updating, and saving the gradients needed in the gradient descent algorithm are knotty problems for a rule-based PIDS.

C. Threat Model & Assumptions

Similar to many previous works [22], [21], [19], [32], [7], [18], we assume that OS kernels and auditing tools are a part of the trusted computing base (TCB), which means the attackers are not able to tamper system auditing data. Meanwhile, we do not consider hardware manipulation and any other attacks that leave no traces on the provenance graph.

In each detection environment, we presume the availability of audit logs encompassing daily activities and devoid of any malicious activities to serve as the training data. In other words, we adopt the assumption that any alarm generated from this training data is a false alarm.

III. Motivation Examples

In this section, we present motivating examples to elucidate the challenges encountered by traditional, coarse-grained rule-based PIDS, as depicted in Fig. 2. These scenarios contain similar benign and malicious behaviors, which pose challenges to existing rule-based PIDS [22], [21], [23], [7] in adjusting their rules to effectively select an optimal trade-off point between excessive false alarms and missing true alarms.

"Grey" Nodes: With the advancement of cloud computing, connections to Function as a Service (FaaS) platforms like AWS Lambda and Cloudflare Workers have become a typical pattern in provenance graphs. However, recent reports indicate that attackers are exploiting these commercial FaaS services to redirect traffic to Command and Control (C2) servers, as illustrated in Fig. 2a. Existing rule-based PIDS are often too coarse to depict those IPs. For example, SLEUTH [21] only defined three trustworthiness levels: *Benign Authentic*, *Benign*, and *Unknown* for the nodes, and MORSE [22] used a binary initial tag for the trustworthiness. But in this scenario, since both benign and malicious behavior can involve communications with the external FaaS IPs, the coarse-grained rules either lead to excessive false alarms or missing true alarms. A more fine-grained configuration to reflect the "grey scale" of these IPs may help solve the issue, i.e., we can assign a trustworthiness value (a float between 0 and 1) to every node in the provenance graph.

Dependency Explosion: Dependency explosion refers to the scenario where all the subsequent events are treated as they depend on the previous ones, and thus a single suspicious event can lead to millions of system entities considered suspicious [22]. As shown in Fig. 2b, a prolonged and furcate event chain can spread the maliciousness carried by the suspicious entities to entities involved in all subsequent events. Many existing efforts have been made to solve this issue [38], [39], [40], [41], [42]. Unfortunately, their solutions usually require extensive instrumentation of applications/OS [40], [41], [42] and expertise, which limits the usage in the real-world system, especially in end-point host deployment.

To deal with the dependency explosion problem without extensive instrumentation, existing rule-based PIDS adopt methods like event whitelisting [21], cost-based pruning, or dividing suspicious events into different stages in a typical APT lifecycle [23]. MORSE [22] proposed the decay and attenuation mechanisms to reduce the maliciousness carried by the system entities over time to mitigate the dependency explosion. However, the decay and attenuation factor is the same for all nodes and edges. Such identical configuration can lead to the dilemma between excessive false alarms and missing true alarms as attackers may deliberately fork many irrelevant processes or connect to benign entities to evade detection. As shown by the two provenance graphs in Fig. 2b, we want to confine the dependency explosion in the left graph while preserving the maliciousness in the right (the right graph depicts a rare pattern where a `firefox` process forks a `Python` process, indicating the possibility of suspicious script execution from a website.) Ideally, we aim to set a propagation rate for each individual event, allowing for more customized control. This ensures that the maliciousness of unusual events does not fade away too quickly, while it can reset rapidly for more common events.

Customized Alarm Triggering: In rule-based PIDS, alarms are triggered based on predefined conditions that are uniformly applied to all events, regardless of their context. These one-fits-all approaches can lead to suboptimal detection performance. For instance, MORSE sets a series of detection rules and thresholds to trigger alarms and the rules are identical for all processes and events. As the motivating example shows in Fig. 2c, `firefox` and `sshd` are two processes that often have network activities with various IP addresses. While `sshd` may commonly use `bash` for script execution, it is unusual and suspicious for `firefox` to do the same. Despite the similarities in event types and related entities between `firefox` and `sshd`, applying a uniform threshold for all "fork" events could either result in false alarms for `sshd` or miss genuine threats involving `firefox`. To distinguish between such two behaviors, a specific rule is needed: for the common events, we want to set a higher threshold to avoid excessive false alarms, while for rare events, we want to keep the threshold low to avoid missing true alarms. A knottier question then follows: how to set different thresholds about

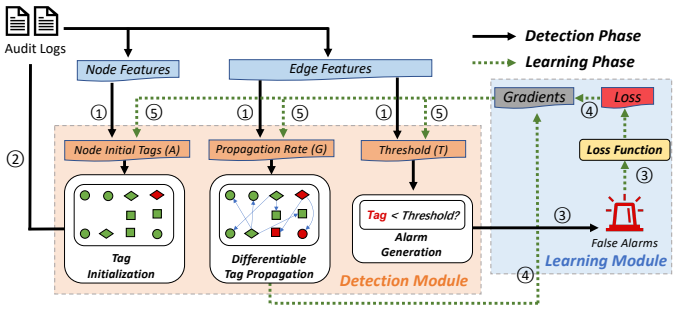


Fig. 3: The overall framework of CAPTAIN. Phases ①-⑤ show the lifecycle of the detection module and the learning module within one training epoch.

the "execute bash" event for each process?

The three motivating examples underscore the challenges that existing rule-based PIDS face in balancing false and true alarms due to their coarse-grained rules. Additionally, these systems lack the flexibility to dynamically adjust rules to accommodate different environments. Theoretically, embedding-based PIDS, leveraging the power of deep learning, can effectively create customized "rules" for every event and entity in the graph by embedding them into the feature space. However, training and deploying deep neural networks require numerous calculation resources and storage space, leading to high runtime overhead. Moreover, these PIDS usually lack the explainability of the detection process and result, making it difficult to pinpoint the malicious behaviors at the event level.

To overcome these limitations, we aim to refine rule-based PIDS using fine-grained rules. By allowing the rule-based detector to learn and adapt from training data, we can enhance detection accuracy without the extensive resource demands and runtime overhead. Our system improves the overall efficacy of rule-based PIDS but also maintains its operational simplicity and clarity.

IV. SYSTEM DESIGN

A. System Overview

As shown in Fig. 3, CAPTAIN consists of two major parts: the detection module and the learning module. The detection module takes the audit logs as input and produces detection results, while the learning module leverages the false alarms during the training phase to fine-tune the detection module.

B. Adaptive Detection Module

The detection module in CAPTAIN comprises three major components: tag initialization, tag propagation, and alarm generation. System entities are assigned initial tags when they appear in the system. As system events happen, tags get updated among system entities, which we refer to as "tag propagation". At the same time, CAPTAIN determines whether to generate alarms based on the pre-defined rules.

Tag Design: Inspired by previous provenance-based methodology based on data flow and control flow [43], [44], [21], [22], CAPTAIN designs two types of tags: data tags and code tags. As shown in Table II, data tags exist on all nodes in the provenance graph. A data tag is represented as a numerical vector `<c, i>`, where `c` denotes the confidentiality score of the data, and `i` denotes the integrity score of the data. In contrast, code tags only exist on process nodes, denoted by `<p>` to indicate the integrity of the code. All scores (`c`, `i`, and `p`) are real numbers ranging from 0 to 1. A low integrity score suggests that the data/code might be compromised and, therefore, is considered untrusted. A low confidentiality score is counterintuitively associated with highly sensitive data (to make the scoring scheme consistent), such as passwords.

TABLE II: Tags in CAPTAIN

Tag Type		Value Range
Data Tag	Confidentiality (c)	0 (Most Confidential) to 1 (Public)
	Integrity (i)	0 (Lowest) to 1 (Highest)
Code Tag	Integrity (p)	0 (Lowest) to 1 (Highest)

The tag system used by CAPTAIN is inspired by previous rule-based PIDS [22], [21]. The focus of this paper is not on the design of the tag system. In essence, any new type of tag can be integrated with CAPTAIN as long as it satisfies the following conditions: 1) The tags are numerical; 2) The tags are updated through arithmetic calculation; 3) Alarms are triggered based on the tag values. For example, we can define an additional "exploitability" tag for all web service nodes and propose the corresponding propagation rules and alarm-triggering rules. We leave tag designing as a practical problem in specific scenarios and use the tags mentioned above to show the improvement purely brought by CAPTAIN's learning module.

We then introduce three adaptive parameters, which are the core of the fine-grained rule-based PIDS:

- **Tag Initialization Parameter** (A) determines the initial tags of system entities.
- **Tag Propagation Rate Parameter** (G) adjusts the propagation effect of system events on tags.
- **Alarm Generation Threshold Parameter** (T) makes adjustments to the alarm generation rules.

We will elucidate how CAPTAIN uses the three parameters to achieve more flexible rule-based detection.

Tag Initialization: Proper tag initialization rules play a crucial role since they determine the initial states of system entities. For instance, if a socket associated with a benign IP address is assigned low integrity, it could lead to many false alarms. However, it is relatively less discussed in previous works [22], [21]. They usually assign the initial tags based on domain knowledge without further adjustments. Although MORSE admits tag initialization policies could be learned from the previous tracing data, it did not discuss it in detail or propose a practical methodology.

We initialize the tags on the nodes when they are initially added to the provenance graph. As illustrated in Fig. 3 step ①, the detection system first checks for new system entities in the event upon receiving the system audit logs. If new entities are identified, they are added to the provenance graph and

assigned initial data tags. Please note that when adding a new process node, we do not specifically initialize its code tag and data tag. A process node's code tag value is inherited from the data tag of the files loaded by the process, and its data tag is propagated from its parents. For non-process nodes, we define the adaptive parameter A at the node level to facilitate fine-grained tag initialization rules. Specifically, for any node $n_i \in N$ in the provenance graph, A stores its initial data tag with respect to its node feature. The node features depend on the information granularity of audit logs, such as file path, process name, command line, IP address, port, etc. For example, if the node feature for a socket is its IP address, then all sockets with the same IP address will be assigned the same initial data tag.

CAPTAIN's learning module refines A through the training process, which will be elaborated upon in §IV-C. The training process starts from the default value A_0. Since we train CAPTAIN on the benign data, we set A_0 conservatively. In this paper, "conservative" means the assumption of the worst-case. Under conservative settings, we aim to capture true attacks as much as possible without considering excessive false alarms. Rule-based PIDS such as MORSE has to use the conservative setting to capture all potential attacks [22], while CAPTAIN only uses the conservative setting as the initial state for training on benign data. For example, we assign all IP addresses an initial integrity of 0 so we do not miss any attackers entering through network communications.

Although the confidentiality scores could be customized in CAPTAIN, we find it hard to learn them from the false alarms on the benign training data. Instead, it often requires specific domain knowledge or personal, subjective judgments to determine what files are sensitive. Therefore, we manually set the initial confidentiality scores and do not adjust them with the learning module.

Tag Propagation: As shown in Fig. 3 step ②, once the tags are initialized, specific system events will trigger the propagation of these tags along the direction of the information flow, resulting in changes to code and data tags. In CAPTAIN, tag propagation will pass and accumulate malicious intentions through tag values. For example, the `(Firefox, read, IP)` event updates the data tag of `Firefox` to the lower value between the data tags of `Firefox` and `IP`. This ensures that potential malicious intentions carried in the information flow are preserved. We designed the propagation rules based on existing work [21], [22] as detailed in Table VIII.

Unfortunately, such a tag propagation mechanism can suffer from the dependence explosion issue [22], leading to excessive alarms. For example, as shown in III, `(bash, write, /home/user/.bash_history)` is a common event that appears in many propagation chains of false alarms, which means this event causes large-scale maliciousness propagation on the provenance graph. However, our investigation shows this is a commonly seen benign activity.

To mitigate such a problem, we introduce another adaptive parameter G, to regulate the propagation rate. Specifically, for any edge $e \in E$ identified with (*src_node_feature, event_type, dest_node_feature*) in the provenance graph, G stores its propagation rate parameter $g_e \in [0, 1]$. Given g_e, the source node src, and the destination node $dest$, we update tag_{dest} as follows during tag propagation.

$$tag_{dest}^{new} = g_e \cdot tag_{rule} + (1 - g_e) \cdot tag_{dest} \tag{1}$$

Where tag_{rule} is the tag value given by the propagation rules defined in Table VIII. Usually, it equals to tag_{src}. If g_e is close to 0, the corresponding tag propagation will lead to small changes in the tag values and vice versa. In this way, G allows CAPTAIN to fine-tune the propagation rules at the edge level.

CAPTAIN refines G through the training process in the learning module, which will be elaborated upon in §IV-C. We also conservatively establish the default values of G (G_0) to ensure all true alarms can be captured before the training starts. Particularly, we set $g_e = 1$ for all edges $e \in E$ to make all propagation fully effective.

Alarm Generation: The alarm generation rules determine whether an alarm should be triggered. As shown in Fig. 3 step ③, whenever an event happens, we will assess whether it satisfies the criteria to trigger alarms. For example, if a process with a data integrity value smaller than 0.5 writes to a normal file, a file corruption alarm will be triggered. As shown in Table IX, we generate alarms based on the event type, the subject and object tags, and the threshold. The threshold could be fine-tuned to control the number of alarms. For example, if event `(sshd, execve, bash)` triggers many false alarms during training, it suggests that we should decrease the alarm-triggering threshold for this event.

We introduce an adaptive parameter T to enable fine-grained adjustment of the alarm threshold. Specifically, for any edge $e \in E$ identified with (*src_node_feature, event_type, dest_node_feature*), T stores its alarm threshold $thr_e \in [0, 1]$. For an edge e, the detection function $f(e)$ is defined as follows.

$$f(e) = tag - thr_e = \begin{cases} malicious, & \text{if } f(e) < 0 \\ benign, & \text{otherwise} \end{cases} \tag{2}$$

where tag is the relevant tag on the node of interest on edge e_i. A low threshold prevents the system from generating the alarms, while a high threshold encourages the system to generate the alarms. In this way, T allows CAPTAIN to adjust the alarm-triggering rules at the edge level.

Like A and G, CAPTAIN's learning module refines T can be refined through the training process (§IV-C). Before the training, we set the default thresholds (T_0) neutrally. In other words, we set $thr_i = 0.5$ for every edge e_i so that we do not encourage or suppress all alarms.

Please note that CAPTAIN's methodology is independent of specific rules or tags, making it broadly applicable to various rule-based PIDS, especially those using taint-analysis methods like SLEUTH [21], HOLMES [23], and CONAN [24].

C. Learning Module

With the three adaptive parameters (A, G, and T) we defined in the previous section, the problem of adaptative configuration for the detector is then converted to the optimization problem

of the multi-variable function aiming to find the optimal values for A, G, and T. In this section, we present our efforts to solve this optimization problem with the gradient descent algorithm, including defining the objective function and constraints, calculating gradients, and searching for the optimal values.

1) Loss Function: The first step of solving the optimization problem is to define the objective function, i.e., the loss function in the context of learning tasks. The learning module aims to find the parameters that can reduce false alarms while maintaining the sensitivity to the true malicious events. Therefore, the loss function comprises two terms: the term of false alarms and the regularizer term.

The false alarm term mainly focuses on penalizing erroneously triggered alarms. Specifically, an event e triggers a false alarm means $f(e)$ should be greater than 0 but it does not. Since $f(e) \in [-1, 1]$, we can use the Mean Squared Error $(y_e - f(e))^2$ as the loss function for all events that trigger a false alarm. y_e is set as 1 for those benign events, and -1 for the malicious events. Because we only use benign data for training, all y_e should be 1. Please also note that we do not calculate the loss for the correctly classified events, i.e., we do not make $f(e)$ close to 1 if $f(e)$ is already greater than 0. This is because the events that do not cause alarms are significantly more than the alarm-triggering events, considering them, therefore, is inefficient and would make the detection system insensible to the malicious events. Thus, the false alarm term in the loss function can be formalized as

$$\mathcal{L}(e) = max(0, (1 - f(e))^2 - 1) \tag{3}$$

Next, we introduce the second component: the regularizer term. As we mentioned, CAPTAIN is trained on benign data to learn the normal behaviors in the detection environment. This is due to the fact that benign behaviors on a system are more consistent than attack activities. Another reason is the malicious training data is much harder to acquire. However, a big challenge is how to guarantee that the detection capability would not be affected if there is no malicious sample in the dataset, i.e. we would not be too lenient to capture the true alarm when reducing false alarms.

To capture all potential malicious events, the adaptive parameters are configured conservatively before training. During training, we only want to make small adjustments to a small portion of the parameters according to the false alarms on training data. For the rest parameters, we would like to keep them as conservative as possible so that the sensitivity to the maliciousness is not compromised. In the area of machine learning, One-class classification (OCC) algorithms are proposed to deal with the situation where only one class of samples is available in the training set [45]. Inspired by [46], [47], we add a regularizer term in the loss function to avoid the parameters becoming too lenient when the malicious training sample is absent. The essential thought is to make the adaptive parameters as close to the default values (A_0, G_0, T_0) as possible. By minimizing the l_2 distance between the adaptive parameters and the default values, we protect the detection capability during training. Therefore, the detection loss can be formalized as

$$Loss = \sum_{e \in E} \mathcal{L}(e) + \alpha||A - A_0||_2 + \gamma||G - G_0||_2 + \tau||T - T_0||_2 \tag{4}$$

where, α, γ, and τ are the regularizer coefficients. In the evaluation section, we discuss the effect of the regularizer coefficients (§VI-D2) and methods used to fine-tune them (§VI-A2).

2) Differentiable Detection Framework: One of the fundamental steps of using gradient descent algorithms is calculating the gradient of each variable. To adjust the adaptive parameters based on the loss function, we need to design a differentiable detection framework, which allows us to keep the gradients of the adaptive parameters with respect to $Loss$.

According to the chain rule, for the parameter a_n, we have

$$\frac{\partial Loss}{\partial a_n} = \sum_{e \in E} \frac{\partial \mathcal{L}(e)}{\partial f} \cdot \frac{\partial f}{\partial a_n} + \alpha \cdot (a_n - a_0) \tag{5}$$

The equations for g_e and thr_e (presented in Appendix §C) hold similarly to Eq. 5. The most challenging part of building an adaptative rule-based PIDS is to compute the gradients, i.e., $\partial f/\partial a_n$, $\partial f/\partial g_e$, and $\partial f/\partial thr_e$. This is because the rule-based detection process is usually modeled as the branch selection based on rules, rather than a differentiable function. To the best of our knowledge, no previous work has formalized or recorded the gradients of parameters in a rule-based PIDS. In CAPTAIN, we model the detection process as a differentiable function and calculate the gradients of each adaptive parameter. When the tags are propagated, the corresponding gradients are updated and recorded. In so doing, we associate the adaptive parameters with the loss, making it possible to perform the gradient descent algorithm to find the optimal parameters that minimize the loss.

As per Eq. 2, $\partial f/\partial thr_e = -1$. This means every time when we want to increase the value of f (the direction of benign), we have to decrease the value of thr_e, and vice versa. This aligns with our intuition because a lower tag value means more maliciousness in our system. Therefore, if we want to be more lenient in the detection, we should set a lower threshold.

The formalization and calculation of $\partial f/\partial a_n$ and $\partial f/\partial g_e$ are more complicated. According to Eq. 2, calculating the gradients of the detection function f is essentially computing the gradients with respect to the variable tag for each node.

We start from the gradients of a_n. According to the propagation policies shown in Table VIII, the updated tag value tag_{rule} is either the lower tag values of involved nodes $min(tag_{src}, tag_{dest})$ or a constant value c. Note the subscripts *src* and *dest* denote the propagation direction. If $tag_{rule} = c$, then

$$\frac{\partial tag_{dest}^{new}}{\partial a_n} = (1 - g_e)\frac{\partial tag_{dest}}{\partial a_n} \tag{6}$$

If $tag_{rule} = tag_{src}$, then

$$\frac{\partial tag_{dest}^{new}}{\partial a_n} = g_e\frac{\partial tag_{src}}{\partial a_n} + (1 - g_e)\frac{\partial tag_{dest}}{\partial a_n} \tag{7}$$

Since a_n is defined as the initial value of the tag on node n, the initial gradient of a_n is

$$\frac{\partial tag_{n'}}{\partial a_n} = \begin{cases} 1, & \text{if } n' = n \\ 0, & \text{if } n' \neq n \end{cases} \tag{8}$$

Then we focus on calculating gradients of g_e, which is more complicated. From Eq. 1, we know that tag^{new}_{dest} is determined by tag_{dest}, tag_{rule}, and g_e. And tag_{dest}, tag_{rule} are related to the propagation rates of previous events. Therefore, tag^{new}_{dest} is influenced by the propagation rates of the current event (denoted by g_e) and all previous events (denoted by $g_{e'}$) happened on src and $dest$ nodes. Consequently, for each propagation, we must update $\partial tag^{new}_{dest}/\partial g_e$ and $\partial tag^{new}_{dest}/\partial g_{e'}$ simultaneously. The case of $g_{e'}$ is similar to that of a_n, we have

$$\frac{\partial tag^{new}_{dest}}{\partial g_{e'}} = g_e \frac{\partial tag_{rule}}{\partial g_{e'}} + (1 - g_e) \frac{\partial tag_{dest}}{\partial g_{e'}} \tag{9}$$

Calculating gradients of g_e involves the Product Rule in calculus, which is

$$\frac{\partial tag^{new}_{dest}}{\partial g_e} = g_e \frac{\partial tag_{rule}}{\partial g_e} + (1 - g_e) \frac{\partial tag_{dest}}{\partial g_e} + tag_{rule} - tag_{dest} \tag{10}$$

The full proof of Eq. 9 and Eq. 10 can be found in the Appendix §C.

As previously stated, g_e are set according to the event features *(src_node_feature,event_type,dest_node_feature)*. Therefore, before the propagation happens,

$$\frac{\partial tag_n}{\partial g_e} = 0, \forall\ e \in E, \forall\ n \in N \tag{11}$$

We use Eq. 11 to initialize the gradient of g_e and update them according to Eq. 9 and 10.

In summary, to employ the gradient descent algorithm, we calculate the gradients of the adaptive parameters with respect to the loss. We first calculate the initial gradient for each node. Afterward, the gradients are updated according to our equations when the tags are propagated. Please note that unlike GNN models such as GCN [48] and GraphSage [49], which only consider the n-hop neighbor of a node, CAPTAIN fine-tunes every edge and node in the graph that can influence the detection results by propagating their gradients.

3) Training and Testing: After calculating the gradients of the adaptive parameters with respect to the loss using the differentiable detection framework, we can now utilize the gradient descent algorithm to optimize the parameters. In machine learning, this process is referred to as "training". The learned parameters are stored as a customized configuration, which is then used to set up CAPTAIN prior to testing.

As shown in Fig. 3, an epoch in training comprises the forward and backward propagation. Before training starts, all adaptive parameters, A, G, and T, are configured as the default settings A_0, G_0, and T_0. Instead of random initialization[50], we use the most conservative setting as the starting point to keep the sensitivity to the maliciousness during training.

When CAPTAIN processes the events, tags are propagated among the graph, updating the gradients according to Eq. 7, 9 and 10, and generate the detection results. Next, we calculate the loss and back-propagate the gradients of loss according to Eq. 5. Finally, parameters are updated in the opposite direction of the gradient as follows:

$$p_{new} = p_{old} - l \cdot \frac{\partial Loss}{\partial p_{old}} \tag{12}$$

where p is the adaptive parameters (which could be a_n, g_e, or thr_e), p_{new} is the new parameters, p_{old} is the old one, $\frac{\partial Loss}{\partial p_{old}}$ is the gradient, and l is the learning rate. We repeat the training process when the maximum epoch is reached or the changes in the result become sufficiently small. Before testing, we configure the parameters according to what we learned in the training stage. Then, CAPTAIN can process audit data and conduct detection as introduced in § IV-A.

V. IMPLEMENTATION

The entire system (including data parsing, tag initializing, tag propagating, alarm generating, and training/testing framework) consists of 5KLoC of Python. We implement the differentiable detection framework by creating two dictionaries for each tag to store the gradients with respect to A and G. As for T, because its gradients are unrelated to the node tags, there is no need to store them within the node. The space used to store those gradient dictionaries during training is analyzed in § VI-C2 and § E-A

VI. EVALUATION

Our evaluation aims to answer the following five research questions: 1) How effectively can CAPTAIN detect the attacks, especially in terms of reducing false alarms? 2) How efficient is CAPTAIN compared with the SOTA PIDS in terms of detection latency and runtime overhead? (§VI-C) 3) How robust is CAPTAIN against adversarial attacks such as mimicry attacks and data poisoning attacks? (§VI-D) 4) How do different components affect the training outcome and the detection performance of CAPTAIN? (§VI-E) 5) Can CAPTAIN acquire explainable knowledge via our learning module? (§VI-F)

A. Experiment Settings

1) Datasets: We evaluate CAPTAIN using public forensic datasets from the DARPA Transparent Computing program and datasets generated within simulated environments in collaboration with an SOC.

DARPA Datasets. The DARPA Transparent Computing program was organized between 2016 and 2019 to perform several red team assessments. In two weeks, the data collecting teams deploy collectors on several target hosts [27]. We use the public-available datasets from Engagement 3 (E3) and Engagement 5 (E5) in our evaluation. [51], [52] provides a detailed description of attacks performed in relevant DARPA datasets.

Simulated Environments. Moreover, we collaborate with an industry SOC to acquire additional datasets within realistically simulated scenarios to avoid the problem of "close-world data" [18]. Specifically, the SOC furnishes detailed host setups from real-world operating environments. Subsequently, we simulate APT attacks [53] employing the Atomic Red Team [54] and online malware repositories [55]. The simulated scenarios encompass five APT attacks across three distinct real-world operational settings, elaborated in detail in §A.

Data Labeling. For each attack scenario, we label entities and events on the kill chains as malicious according to the attack reports. Although previous work [20], [16] provided entity-level data labels, they did not specify the data labeling strategy or context. We also observed that the amount of malicious labels in their dataset is excessively large (e.g., over 12 thousand system entities were marked as malicious within a 30-hour period on the CADETS from DARPA Engagement 3), which is impractical for a real-world SOC. Thus, we decided to use our data labels in the evaluation for a fair and unbiased comparison between CAPTAIN and the baseline systems. We also make our data labels publicly available to facilitate future research.

2) Experiment Setup: We deployed CAPTAIN and performed all experiments on an Ubuntu 22.04.3 Linux Server with an Intel(R) Xeon(R) Platinum 8358 CPU @ 2.60GHz and 1.0 TB memory. We partition each dataset into training and testing sets, adhering to the assumption stated in §II-C that the training set should not contain any malicious activities. Specifically, we find the starting time of the first attack. Then, all data produced before that date becomes the training set, whereas the remaining data becomes the testing set.

To avoid biased results due to overfitting, we performed cross-validation when evaluating the performance of CAPTAIN. However, standard *k*-fold cross-validation is unsuitable for streaming logs because the time series cannot be freely split into *k* groups, as the subsequent tags depend on the previous ones. Therefore, we utilized time series cross-validation, which preserves the chronological order of data. We set a time window and trained the parameters using the data within this window. Afterward, we moved the time window forward until creating *k* different training sets. In our experiment, we set *k*=3 and the length of the time windows to be around 2/3 of the total length of the training set. We also discuss overfitting in § VII-A.

Like many other machine learning systems, CAPTAIN also relies on appropriate hyperparameters, especially three regularizer coefficients α, γ, τ, to train the accurate and robust model (the effect of these hyperparameters are shown in §VI-D2). We provide two methods to fine-tune the hyperparameters, depending on whether a validation set is available. If a validation set exists, we perform grid searching, a generic hyperparameter tuning approach. We train multiple models with different hyperparameter combinations and evaluate their performance on the validation set. The default conservative detector is run on the validation set as the baseline. We select the hyperparameters from the model with the fewest false alarms and equivalent true alarms to the baseline. For instance, the optimal hyperparameters of E3-CADETS is $\alpha = 0.1, \gamma = 0.1, \tau = 0.1$ and the learning rate is 0.01. However, a validation set is usually not available for a PIDS in practice. Even if it exists, it may not encompass all possible attacks. In this case, we propose a heuristic-based method to fine-tune the regularizer coefficients. Recall that the training starts from the conservative adaptive parameters, false alarms in the training set will loosen the conservative settings. The regularizer coefficients are added to the loss function to avoid being too lenient and missing the true alarms. Therefore, the values of the regularizer coefficients α, γ, and τ are determined by the following question: how many false alarms caused by a node/edge can we tolerate in the training set to avoid being overly lenient? Before training starts, we set a number for the allowed false alarms (N), and we can estimate an approximate value of the regularizer coefficients based on N: $\alpha \approx 3N, \gamma \approx 3N, \tau \approx 12N$ (see Appendix §D for the mathematical proof). Please note that these are approximate estimations depending on the extent to which we trust the "benign" training data. A practical way might be: set regularizer coefficients using heuristics; if missing alarms occur compared to the conservative baseline, multiply the coefficients by a factor (e.g., 10) and retrain. Conversely, if no alarms are missed for a long time T, reduce the coefficients by a factor and retrain.

3) Baseline Detectors: We compare CAPTAIN with five SOTA PIDS: FLASH [16], KAIROS [15], SHADEWATCHER [13], NODLINK [18], and MORSE [22] to evaluate their performance from different perspectives. We chose FLASH, KAIROS, and NODLINK because they are the SOTA embedding-based PIDS. In addition, their implementations are open-sourced, allowing us to test them on different datasets. Since some other embedding-based PIDS [20], [19], [23] have already been evaluated in these works and the result shows that FLASH, KAIROS, and NODLINK outperform them, we didn't include them in our evaluation. We chose MORSE since it is the SOTA rule-based PIDS, and we want to evaluate the improvements from our differentiable adaptation framework in CAPTAIN. To get a fair and unbiased comparison, we use the settings from MORSE in the evaluation. According to [15], [16] and our communication with the authors, the detection systems of SHADEWATCHER and PROGRAPHER are not fully open-source due to proprietary license restrictions. Although the data preprocessing module of SHADEWATCHER is open-sourced, we tried our best but could not locate the preprocessing output files used for the following training and testing. We then realized that it was not feasible for us to perfectly replicate their systems for an unbiased comparison. Therefore, we used the code to evaluate the efficiency and latency of data preprocessing and the detection results reported in their paper to evaluate detection accuracy. For FLASH, KAIROS, and NODLINK, we used their open-sourced code as the basis for the evaluation. We reimplemented MORSE according to their paper.

B. Detection Accuracy

1) False Alarm Events Reduction: We first focus on the reduction of false alarm events, which requires the detection granularity at the event level. We compare CAPTAIN with the SOTA event-level PIDS SHADEWATCHER and the classical rule-based PIDS MORSE. Due to the close-source nature of SHADEWATCHER, we used the reported detection results in their paper. The results show that both MORSE and CAPTAIN successfully detected all attacks in the testing dataset. But CAPTAIN reduces the false alarm rate by over 93% (15.66x) compared to SHADEWATCHER and over 95% (20.45x) compared to MORSE. Additionally, SHADEWATCHER holds 80% of events for training and only 10% for testing, while the testing set of CAPTAIN is over seven times larger than the testing set of SHADEWATCHER. This shows that CAPTAIN does not require as much data as SHADEWATCHER for training.

TABLE III: Comparison with the baselines on the TRACE dataset from Engagement 3 in terms of false alarm events

	MORSE	SHADEWATCHER	CAPTAIN
# of Consumed Events	5,188,230	724,236	5,188,230
# of False Alarm Events	22,500	2,405	1,099
False Alarm Rate	0.434%	0.332%	0.0212%

One advantage of the rule-based PIDS like CAPTAIN and MORSE is that it can offer semantic-rich alarms, while SHADEWATCHER only flags deviations from patterns observed during training without explaining the alarms. We then compared the number of false alarms generated by CAPTAIN and MORSE into different categories, shown in Table IV.

Table IV shows that CAPTAIN can reduce false alarms by over 90% (11.49x) on average for all datasets compared with the non-adaptive MORSE. It outperforms MORSE in every alarm category. Those remaining false alarms, especially in C-5, S-2, and S-3, cannot be removed for the following reasons. First, some "false" alarms are not purely benign. They are related to some (potential) attack nodes (e.g., configuration files in `/tmp/atScript/atomic-red-team-Gray_dev1.0/*` in S-2) but cannot be mapped to specific attack steps directly, thus not labeled as "malicious." Second, some nodes and edges do not exist or trigger any alarm in the training set. Third, since our approach aims to be as conservative as possible, some parameters are tuned just enough to eliminate false alarms in the training set. However, these conservative settings may still cause false alarms when applied to the testing set.

2) False Alarm Entity Reduction: While CAPTAIN operates as an event-level detector, we still compare CAPTAIN with the SOTA entity-level PIDS FLASH, KAIROS, and NODLINK(KAIROS and NODLINK give the result at graph-level, but they also support entity-level detection). We transformed the alarm events triggered by MORSE and CAPTAIN into alarm entities using the following method: an entity is considered an alarm entity if it is involved in any alarm events.

It is noteworthy that only a subset of each DARPA dataset is utilized for training and testing in [20], [16], [15]. To simulate the real scenarios in SOC, we utilize the entire dataset for evaluation, dividing the training and testing sets as described in §VI-A2. We retrained the model and performed detection using their code. Additionally, we noticed FLASH and THREATRACE did not count the FPs within the two-hop distance from the labeled attack entities in the ground truth while counting the TPs within the two-hop distance from the detected entities [15]. To ensure a fair comparison without providing excessive leniency, we report the TP and FP results, as well as the one-hop FP result, in which we exclude the FP entities within a one-hop distance from the ground truth. We also did not count the attack entities in the 2-hop neighborhood of the detected entities as TP like [16], [15], [20] did. These strict experiment settings and data labeling methods can explain the difference between our results and the results in their papers, but we believe it is fair and necessary to assess the performance of PIDS in real-world scenarios.

We evaluated the detection accuracy on CADETS, TRACE, and THEIA from Engagement 3 since those datasets were commonly used by the baselines. Table V illustrates the comparison among CAPTAIN and the baselines. On all datasets, CAPTAIN demonstrates superior performance in identifying more TPs while maintaining fewer FPs. FLASH can report a fair amount of TPs but produce an excessive amount of FPs. It leverages GNN to learn k-hop neighborhood structures. While this technique shows promising performance when there is a significant anomaly within k hops, it could degrade when the training set is limited [16]. KAIROS can filter out FPs with the Anomalous Time Window Queue [15]. Although it guaranteed decent detection accuracy at the time-window level; the entity-level accuracy, however, is not satisfactory. NODLINK detects most attack processes. However, missing other relevant entities, such as files and network sockets, requires additional expert efforts to investigate the reported alarms. MORSE and CAPTAIN achieve a lower false positive rate in node-level detection. This is attributed to the semantic-rich alarms that provide additional information for alarm filtering. For instance, a file corruption alarm involves two entities: the process and the corrupted (benign) files. As the corrupted files should not be reported as malicious entities, they can be easily filtered out. This explains why MORSE triggers thousands of false alarms at the event level, but only hundreds of nodes are incorrectly alarmed. For embedding-based PIDS, there is no such semantics for alarm filtering. We include the filtered results in Table V as we believe it highlights an advantage of rule-based PIDS.

CAPTAIN successfully detects all attacks in the dataset. As an event-level detector, CAPTAIN does not report entities that are not directly related to the attack events. For example, if a malware is downloaded and executed, CAPTAIN reports the MalFileCreation alarm and the FileExec alarm, which reveals the malware process, file, and parent process. We do not immediately report the entry network node used by attackers to avoid alarm fatigue, which explains the FNs of CAPTAIN. The investigations on the relevant nodes can be conducted after the first response.

TABLE IV: Comparison with the baseline on different datasets regarding false alarms. T-3, T-5, C-3, and C-5 represent the TRACE and CADETS datasets from Engagement 3 & 5. S-1 to S-3 denote Cloud, Streaming, and Dev datasets from the SOC.

Datasets	FileExe		MemExec		ChPerm		Corrupt		DataLeak		Escalate		Total False Alarms		
	Base	Ours	Base	Ours	Base	Ours	Base	Ours	Base	Ours	Base	Ours	Base	Ours	Reduction
T-3	9	0	49.3K	3.76K	1	1	626	2	955	14	673	0	52.1K	3.78K	**13.78x**
C-3	41	14	N/A	N/A	6	6	13.2K	272	96	50	81	1	13.4K	343	**39.07x**
T-5	181	170	403K	79.8K	0	0	34.4K	19.9K	26.9K	1.20K	18.4K	28	483K	101K	**4.78x**
C-5	1.63K	0	N/A	N/A	N/A	N/A	1.81M	6.64K	7.87K	2	1.42K	0	1.83M	6.64K	**276x**
S-1	3.16K	0	N/A	N/A	26	0	100	73	22	21	N/A	N/A	3.31K	94	**35.21x**
S-2	177	53	N/A	N/A	0	0	12	9	14	7	N/A	N/A	203	69	**2.94x**
S-3	18	0	N/A	N/A	0	0	29	16	23	22	N/A	N/A	60	38	**1.58x**

TABLE V: Comparison with the baselines in terms of node-level detection accuracy

	TP	FP(0-hop)	FP(1-hop)	FN(0-hop)
Engagement3 CADETS				
FLASH	16	4503	4485	10
KAIROS	15	1017	1003	11
NODLINK[1]	3	120	114	2
MORSE	16	51	43	10
CAPTAIN	16	34	26	10
Engagement3 TRACE				
FLASH	5	27202	27178	19
NODLINK[1]	4	170	170	0
MORSE	10	243	234	14
CAPTAIN	10	12	11	14
Engagement3 THEIA				
FLASH	2	53230	53050	13
KAIROS	12	3566	3422	3
NODLINK[1]	4	62	58	0
MORSE	11	220	213	4
CAPTAIN	11	194	187	4

[1]Since NODLINK only provides detected process, we evaluate it on process detection accuracy.

C. Efficiency

In previous sections, we highlighted the simplicity of CAPTAIN's rule-based detection framework compared to other embedding-based PIDS. In this section, we compare CAPTAIN with the SOTA PIDS FLASH, KAIROS, NODLINK, SHADEWATCHER[2], and MORSE to evaluate their efficiency by running their code on our test environment.

1) Detection Latency: The detection latency analysis encompasses three dimensions: **buffer time**, **preprocessing time**, and **detection time**. We have already introduced buffer time in II-A. Preprocessing time refers to the duration taken to convert the raw audit logs into a data structure that the detection systems can process. It involves data parsing, data cleaning, noise reduction, preprocessing, feature extraction, and so on. Detection time refers to the interval between the completion of data processing and the moment the result is produced. It is noteworthy that since the detection granularity is different in different PIDS, the detection time might not reflect the actual latency of each system. For instance, a detector based on the whole graph may take longer to deliver detection results of each graph compared with a detector on the entity/event level. However, this does not necessarily mean the former is slower because a large graph could contain many entities and events. Therefore, we calculate the total time spent on detection across the entire dataset.

We evaluated the latency on TRACE and CADETS from DARPA Engagement 3 since they are covered in the experiments of all baselines. Our evaluation focuses on the testing stage since training can be conducted offline. We modified their code to obtain unbiased results. For example, FLASH divides the data processing into two stages in their code and uses files to store intermediate results. And KAIROS employs the PostgreSQL database and stores the intermediate results in files. These are the steps not required in a real-time streaming pipeline. Consequently, we exclude the I/O time to focus solely on measuring the "pure" preprocessing and detection time.

The result is shown in Table VI. Unlike other PIDS that embeds the graphs and thus require a buffer time ranging from several minutes to hours, CAPTAIN processes audit logs in a streaming fashion, eliminating the need for any buffer time. CAPTAIN is also faster in preprocessing because the logs are not preprocessed for the machine-learning model. Moreover, the tags of CAPTAIN are much simpler than the state vectors used by FLASH and KAIROS. By avoiding the use of text embedding or GNNs to aggregate semantic and contextual graph information, CAPTAIN achieves detection speeds over 10 times faster than the baseline methods. Lastly, compared to the rule-based PIDS MORSE, CAPTAIN only takes a few seconds longer to detect logs spanning 4 to 7 days, once again demonstrating the superiority of rule-based PIDS in terms of latency.

TABLE VI: Comparison of detection latency

	Buffer Time	Preprocessing Time	Detection Time
Engagement3 TRACE			
FLASH	57:49	107:50	64:24
SHADEWATCHER	N/A[1]	100:22	3:40[2]
NODLINK	00:10	135:42	2:48
MORSE	0	58:20	1:29
CAPTAIN	0	58:20	1:31
Engagement3 CADETS			
KAIROS	15:00	15:34	29:46
FLASH	82:52	18:57	7:41
NODLINK	00:10	6:18	6:41
MORSE	0	7:22	1:19
CAPTAIN	0	7:22	1:23

[1]We did not find a clear number in their codes or paper.
[2]SHADEWATCHER extracts the last 10% interactions as the testing set, while the testing set of us is around 2.5 times larger.

[2]Our evaluation was based solely on the open-sourced portion of SHADEWATCHER.

2) Runtime Overhead: Another important metric reflecting the efficiency of a PIDS is the runtime overhead to the system. In this section, we evaluate the runtime overhead of CAPTAIN during detection and conduct a comparative study with other SOTA PIDS. We also analyze the memory consumption of CAPTAIN during the training stage.

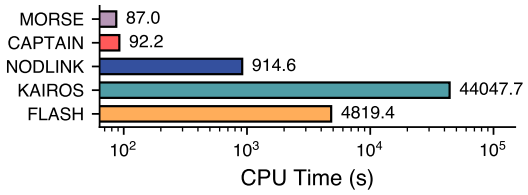
Runtime Overhead in Detection: We use the Python `resource` module to evaluate the resource consumption of detectors on CPU mode. As admitted in [13], GPUs may not be available in most real-life threat detection scenarios. We calculate the total CPU time in user mode. The result is shown in Fig. 4a. Please note that the total CPU time used in Fig. 4a could be more than the detection time in Table VI because of the multi-core CPU usage by deep learning models. Since CAPTAIN only relies on straightforward arithmetic operations on real-number tags, it requires significantly less CPU time than the embedding-based PIDS (around 2% of FLASH and around 0.02% of KAIROS), which need complex matrix and vector operations due to their use of neural networks. Moreover, CAPTAIN's fine-grained detection rules only increase CPU time by a modest 5.6% compared to MORSE. Given the significant improvement on MORSE in detection accuracy (over 90% reduction in false alarms) and the substantial decrease in CPU time compared to other embedding-based PIDS, we believe the slightly higher CPU usage than MORSE is acceptable in real-world detection scenarios.

We use the `psutil` library in Python to monitor the live memory usage during detection. The comparison of memory usage over time is shown in Fig 4b. CAPTAIN finishes detection much faster than the embedding-based PIDS (as already shown in Table VI) and achieves the lowest memory usage throughout the entire detection process. The memory usage and detection time of CAPTAIN and MORSE are similar, shown by the overlapping curves in Fig. 4b. The slight difference in memory usage between CAPTAIN and MORSE is due to the storage of fine-grained rules. In general, CAPTAIN retains the lightweight and fast characteristics of the rule-based PIDS.

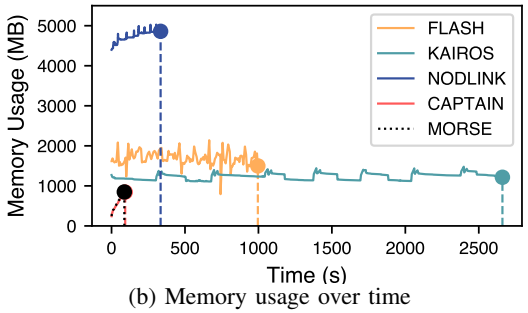
Runtime Overhead in Training: The training of the detectors can be performed offline, where users can allocate ample time and computing resources, including GPUs. Therefore, we did not evaluate the runtime overhead and the time used for training. However, as CAPTAIN records and propagates the gradients for each tag among the big provenance graph, a reasonable concern is that saving those gradients would consume much memory when there is a dependency explosion problem. We evaluated the number of non-zero gradients saved by CAPTAIN during training. The results, shown in Fig. 8, illustrate that most nodes only need to store a small number of non-zero gradients during training. Due to the space limit, the detailed analysis can be found in §E-A.

D. Resilience Against Adversarial Attacks

In this section, we discuss the robustness of CAPTAIN against two main-stream adversarial attacks: adversarial mimicry attack and training set poisoning attack.



(a) Total CPU time used



(b) Memory usage over time

Fig. 4: Comparison of resource consumption when detecting on DARPA Engagement 3 CADETS. In Fig. 4b, the memory usage curve of CAPTAIN and MORSE are overlapped, showing their similar efficiency performance.

1) Adversarial Mimicry Attack: Mimicry attacks on PIDS involve altering the provenance data and incorporating more benign features to "mimic benign behaviors", thereby evading detection. In this section, we evaluated the robustness of CAPTAIN against mimicry attacks using the attack methodology in [35], [16], i.e. inserting benign structures into the attack graphs. Our mimicry attack contains two steps. First, we extract some events of benign system entities from the normal training data. Next, we create "fake" events by substituting the benign entities with the attack entity, simulating the attack entity performing activities similar to those of the benign entities. To verify the effectiveness of the mimicry attack, we use FLASH as the baseline. For evaluation purposes, we use the CADETS from Engagement 3 because of its relatively small scale. The details about our mimicry experiment can be found in the Appendix E-B. The result is shown in Fig. 5.

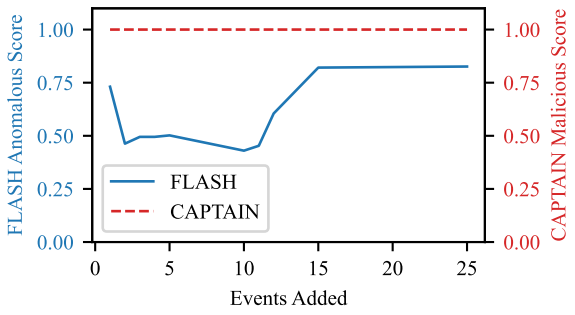


Fig. 5: Adversarial mimicry attack against CAPTAIN and FLASH (use the attack entity `/tmp/test` as an example)

In general, both CAPTAIN and FLASH showed robustness against our mimicry attack attempt to some extent. The attack entity `/tmp/test` is still detected by both systems even after we add some normal activities. However, we see a

significant drop in the anomalous score of FLASH when we introduce a relatively small number of events (fewer than 10), which is also confirmed in their paper [16]. On the other hand, CAPTAIN remains unaffected against the mimicry attack, demonstrating superior robustness compared to the baseline.

The robustness of CAPTAIN results from three main reasons. First of all, the propagation of CAPTAIN is guided by heuristic-based rules, while the propagation and aggregation of FLASH and other embedding-based PIDS are based on the neural network. Second, CAPTAIN is an event-level detector with a finer granularity than entity-level detectors. As mentioned before, CAPTAIN gives the detection result for each streaming event without any buffer time, which means any mimicry insertion after the real attack activities is useless. For those PIDS with the buffer time, any events within the same time window and related to the same entity can affect the final detection result. Third, CAPTAIN is not an anomaly-based detector. In other words, the detection is performed based on the conservative, heuristic rules, while the normal data is only used to reduce the false alarms. Therefore, adding normal features can not compromise the detection capability as it is guaranteed by the conservative detection rules.

2) Training Set Poisoning Attack: Training set poisoning refers to the malicious manipulation of the training set. In the context of cyber security and intrusion detection, it usually involves polluting the benign training set with adversarial events/entities. When the detector is trained on a "benign" dataset that has been polluted, it learns patterns or features introduced by attackers. These patterns and features, once learned, could assist attackers in evading detection in future instances.

We add the regularizer term to the loss function to control the sensitivity of CAPTAIN to the training set. This enhances CAPTAIN's robustness against dataset poisoning attacks. In this section, we present a real case of Pine Backdoor & Phishing Email attack from Engagement 3 TRACE. In this attack, a vulnerable `pine` connected to the email server `128.55.12.73`, downloaded and executed an email attachment. However, IP `128.55.12.73` had multiple activities and caused some "false" alarms in the training set. Please note that this violates our assumption stated in § II-C that the training set must not be compromised by attack entities or events. Nevertheless, this makes it a perfect example to test the robustness against data poisoning attacks.

We analyze the critical parameters for detecting this attack: the initial integrity tag (a) of IP node `128.55.12.73`, to illustrate the effect of the regularization term. Without the regularization term in the loss function, `128.55.12.73` will be assigned a high integrity score (near 1.0) due to its activities in the training set, which lets us miss the phishing email attack starting from it. However, with the help of the regularizer terms ($\alpha = 10$), CAPTAIN is more cautious during training. Like the "grey node" motivating example in §III, CAPTAIN assigns the integrity score as the mediocre 0.488, which removes many false alarms caused by this IP while successfully capturing the phishing email attack during testing (more details can be found in §VI-F). The propagation rate of event `(pine, read, 128.55.12.73)` is also more conservative against the polluted training set due to the regularizer term.

E. Ablation Study

1) Individual Adaptive Parameters: We conduct an ablation study on each adaptive parameter. As each parameter can be learned independently, we have seven separate experiments, each optimizing a subset of the parameters, i.e. $\{A\}$, $\{G\}$, $\{T\}$, $\{A, G\}$, $\{A, T\}$, $\{G, T\}$, and $\{A, T, G\}$.

According to Fig.6, adjusting the threshold (T) alone may not have much effect on the detection results because without changing the initial tags and the propagation rates, the updated tags still tend to be 0 or 1. Fig.6 also shows that tuning all these three parameters can accelerate the training process. Although $\{A, T\}$ and $\{A, T, G\}$ achieve similar good detection results on the testing set, $\{A, T, G\}$ converges using fewer epochs during training. It should be noted that tuning A plays the most major role in mitigating false alarms among these three adaptive parameters. In certain datasets (S-3 and C-3), adjusting A alone sometimes results in fewer false alarms than tuning parameters A, G, and T together because, without the benefit of learning propagation rates and thresholds, we have to assign higher integrity scores to more nodes to counteract false alarms during training. However, reducing false alarms more crudely carries the risk of being excessively lenient.

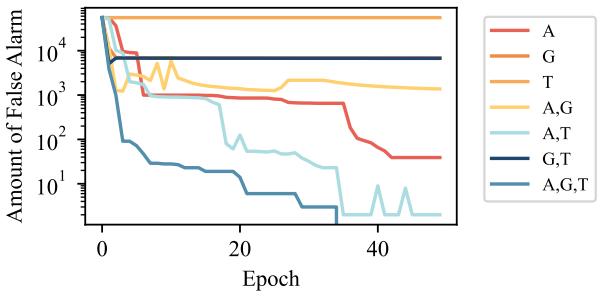


Fig. 6: Ablation study on each parameter.

2) Learning Rate in Optimization: We conducted an experiment on different learning rates on the DARPA Engagement 3 CADETS. The detailed results can be found in § E-C. As illustrated in Fig. 9, a higher learning rate usually leads to a sharper loss decline during the initial training epochs. However, for ongoing training, a lower learning rate may yield a finer-tuned search process and lower loss, necessitating a balance between training speed and training quality.

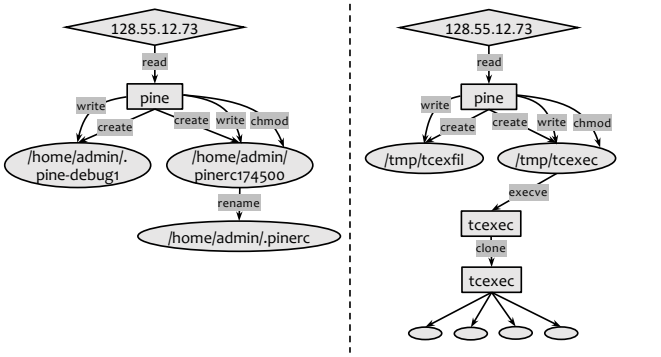
F. Case Study

To illustrate the interpretability of CAPTAIN's detection process, we delve into the training outcomes of the E3-TRACE dataset as a case study. These cases also show the improvement of CAPTAIN compared to existing rule-based PIDS.

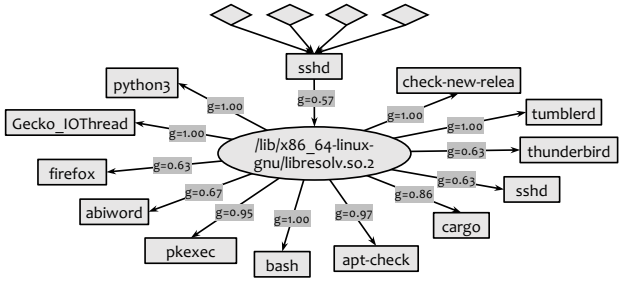
1) Better Detection Capability due to Fine-Grained Rules: The most significant improvement of CAPTAIN is its fine-grained detection capability, enabled by the adaptive parameters. As mentioned in § III, unlike the discrete, human-assigned trustworthiness levels and universal rules used in [22], [21],

[23], the fine-tuned numerical parameters of each entity and event in the graph can better help distinguish nuances between benign and malicious patterns.

Fig. 7a demonstrates how CAPTAIN outperforms the baselines in distinguishing between two similar behaviors using fine-tuned parameters. In both cases, process `pine` reads from `128.55.12.73` (possibly the email server) and creates, writes, and changes the permissions of a new file. It is difficult for previous rule-based PIDS [21], [22] to assign a tag for `128.55.12.73`. Take the SOTA MORSE as an example - if MORSE initialize the node `128.55.12.73` as a benign node with the data integrity score as 1.0, it would miss at least the first stage of the attack. On the other hand, if MORSE assigns the initial tag as "Untrusted" (this is what they did in order to capture all attacks), it would trigger multiple false "Malicious File Creation" and "Change Permissions" alarms on the benign graph. Instead, CAPTAIN assigns a moderate initial tag a_n, 0.488, to this IP and slightly tunes down the threshold thr for the event `(pine, create, /home/admin/pinerc174500)` from 0.5 to 0.478. As $0.478 < 0.488$, the creation of the file will not be alarmed, but in real attacks, the thr for the event `(pine, create, /tmp/tcexec)` is still 0.5, meaning creating a malicious file would still trigger alarms ($0.5 > 0.488$).



(a) The graph on the left side represents normal behavior in the training set, while the graph on the right side depicts an attacker executing a malicious email attachment.



(b) CAPTAIN assigns different propagation rates (g_e) to events to control dependency explosion more precisely.

Fig. 7: Case study from Engagement 3 TRACE. We omitted some irrelevant details to keep the graphs clean and succinct.

CAPTAIN's fine-grained rules also help address the dependency explosion issue caused by `libresolv.so.2`. Unlike MORSE, which sets universal decay/attenuation factors for all processes, CAPTAIN learns distinct propagation rates for different events. As shown in Fig. 7b, CAPTAIN assigns lower propagation rates to events that frequently cause dependency explosions (e.g., being read by `sshd` or `firefox`) while maintaining high propagation rates for other events. Even for the same process, `firefox`, CAPTAIN learns to suppress the dependency propagation when *reading* this file ($g_e = 0.63$) and remains sensitive to maliciousness propagation when *loading* this file ($g_e = 0.99$). Consequently, CAPTAIN avoids being overly lenient for potential attack events when addressing the dependency explosion issue.

The fine-grained parameters of CAPTAIN also outperform many embedding-based PIDS in detecting mimicry attacks. For instance, in Fig. 7a, the attacker can modify the file name of `/tmp/tcexec` with the knowledge that `pine` might interact with a file like `/home/admin/pinerc+numbers`. Since generalization techniques like Word2Vec and Sentence2Vec are widely used in embedding-based PIDS, they can manipulate the file name by appending random digits to `pinerc`, making it similar to `pinerc174500` after embedding. However, to evade CAPTAIN, the attacker must know the exact file name of `pinerc174500`. In addition, deep learning usually relies on the amount of training data. But there are only four events related to file `/home/admin/pinerc174500` during the 7-day training set, making it challenging for the neural networks to learn such patterns.

2) Robustness brought by Differentiable Tag-Propagation Framework: Some previous rule-based PIDS [7] can customize their detection system using benign training data. However, the adjustments are based on the training set rather than the detection results. For example, NODOZE assigns a distinct anomaly score to each event based on its frequency in the training set. However, due to training set poisoning and living-off-the-land attack techniques, the frequent events in the training set cannot be fully trusted. For instance, the events related to `dbus-daemon` are very common in the training set of TRACE in Engagement 3. Typical events such as `dbus-daemon` writes or reads `/var/run/dbus/system_bus_socket` occur over 100,000 times during seven days. Such events and the `dbus-daemon` entity would be assigned with a low anomaly score in frequency-based systems like NODOZE. However, the low anomaly scores allow the attackers to deceive the detectors using living-off-the-land techniques related to the D-Bus process [56]. In contrast, CAPTAIN does not change the parameters related to `dbus-daemon` because, despite the large number of events associated with `dbus-daemon` in the training set, no false alarms are triggered by it. CAPTAIN adjusts the parameters related to an event based on the false alarms caused by it rather than the frequency of it.

VII. DISCUSSION AND FUTURE WORK

A. Overfitting in Parameter Learning

Overfitting is an important issue in machine learning. As CAPTAIN learns the benign patterns in the one-class manner starting from conservative settings, overfitting is, however, a preferred and safer strategy. In one-class learning, we want

to generate a tightest boundary for the target class (benign patterns). Therefore, CAPTAIN avoids generalizing excessively to minimize the risk of evasion, as demonstrated in our example in §VI-F.

B. More Optimization Algorithms and Online Learning

In this paper, we utilize the gradient descent algorithm for optimization due to its simplicity and clearness. We can reach optimal values more quickly and accurately, using algorithms such as *Nesterov Accelerated Gradient (NAG)* [57], *Adadelta* [58], *RMSprop*, *Adagrad* [59], *Adam* [60], *AdaMax*, *Nadam*, and so on. These algorithms require the first-order gradient, offered by the differentiable detection framework in CAPTAIN. We leave the implementation of these algorithms as the extensions of CAPTAIN.

Long-term maintenance is important in real-world scenarios. Although we assess CAPTAIN in an offline setting, CAPTAIN's learning module can compute loss upon receiving the feedback for each event. Subsequently, the learning module can promptly update the adaptive parameters for subsequent detection.

VIII. CONCLUSION

This paper introduces CAPTAIN, a rule-based PIDS capable of automatically adapting to detection environments, enhanced by three adaptive parameters: tag initialization (A), propagation rate (G), and threshold (T). The differentiable detection framework enables the optimization using the gradient descent algorithm. To our knowledge, this is the first effort to incorporate gradient descent methods in optimizing rule-based PIDS. We evaluated CAPTAIN on several datasets. The results demonstrate the superior detection capability, significantly reducing false alarms, detection latency, and runtime overhead, outperforming the SOTA baselines.

ACKNOWLEDGMENT

We would like to thank anonymous reviewers for their constructive feedback. Lingzhi and Xiangmin were supported by the National Science Foundation (NSF) grant 2148177 and funds from the Resilient & Intelligent NextG Systems (RINGS) program. In addition, Zhenyuan was supported by the "Pioneer" and "Leading Goose" R&D Program of Zhejiang (2024C03288). Sekar's work was supported in part by NSF grants 1918667 and 2153056.

REFERENCES

[1] A. Alshamrani, S. Myneni, A. Chowdhary, and D. Huang, "A survey on advanced persistent threats: Techniques, solutions, challenges, and research opportunities," *IEEE Communications Surveys Tutorials*, vol. 21, no. 2, 2019.

[2] H. B. Review, "Missed alarms and 40 million stolen credit card numbers: How target blew it," https://hbr.org/2014/03/could-target-have-prevented-its-security-breach. [Online]. Available: https://hbr.org/2014/03/could-target-have-prevented-its-security-breach

[3] N. Y. Times, "Equifax says cyberattack may have affected 143 million the u.s." https://www.nytimes.com/2017/09/07/business/equifax-cyberattack.html. [Online]. Available: https://www.nytimes.com/2017/09/07/business/equifax-cyberattack.html

[4] F. Dong, S. Li, P. Jiang, D. Li, H. Wang, L. Huang, X. Xiao, J. Chen, X. Luo, Y. Guo *et al.*, "Are we there yet? an industrial viewpoint on provenance-based endpoint detection and response tools," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 2396–2410.

[5] G. P. Spathoulas and S. K. Katsikas, "Using a fuzzy inference system to reduce false positives in intrusion detection," in *2009 16th International Conference on Systems, Signals and Image Processing*, 2009, pp. 1–4.

[6] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, jul 2009. [Online]. Available: https://doi.org/10.1145/1541880.1541882

[7] W. U. Hassan, S. Guo, D. Li, Z. Chen, K. Jee, Z. Li, and A. Bates, "Nodoze: Combatting threat alert fatigue with automated provenance triage," in *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*. The Internet Society, 2019. [Online]. Available: https://www.ndss-symposium.org/ndss-paper/nodoze-combatting-threat-alert-fatigue-with-automated-provenance-triage/

[8] X. Shen, Z. Li, G. Burleigh, L. Wang, and Y. Chen, "Decoding the mitre engenuity att&ck enterprise evaluation: An analysis of edr performance in real-world environments," in *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security*, 2024, pp. 96–111.

[9] C. Dietrich, K. Krombholz, K. Borgolte, and T. Fiebig, "Investigating system operators' perspective on security misconfigurations," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 1272–1289. [Online]. Available: https://doi.org/10.1145/3243734.3243794

[10] S. C. Sundaramurthy, A. G. Bardas, J. Case, X. Ou, M. Wesch, J. McHugh, S. R. Rajagopalan, and L. F. Cranor, "A human capital model for mitigating security analyst burnout," in *Eleventh Symposium On Usable Privacy and Security (SOUPS 2015)*, 2015, pp. 347–359.

[11] B. A. Alahmadi, L. Axon, and I. Martinovic, "99% false positives: A qualitative study of SOC analysts' perspectives on security alarms," in *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022, pp. 2783–2800. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/alahmadi

[12] S. C. Committee, "A "kill chain" analysis of the 2013 target data breach," 2014. [Online]. Available: https://www.commerce.senate.gov/services/files/24d3c229-4f2f-405d-b8db-a3a67f183883

[13] J. Zengy, X. Wang, J. Liu, Y. Chen, Z. Liang, T.-S. Chua, and Z. L. Chua, "Shadewatcher: Recommendation-guided cyber threat analysis using system audit records," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 489–506.

[14] F. Yang, J. Xu, C. Xiong, Z. Li, and K. Zhang, "{PROGRAPHER}: An anomaly detection system based on provenance graph embedding," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 4355–4372.

[15] Z. Cheng, Q. Lv, J. Liang, Y. Wang, D. Sun, T. Pasquier, and X. Han, "Kairos: Practical intrusion detection and investigation using whole-system provenance," *arXiv preprint arXiv:2308.05034*, 2023.

[16] M. U. Rehman, H. Ahmadi, and W. U. Hassan, "Flash: A comprehensive approach to intrusion detection via provenance graph representation learning," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 2024, pp. 139–139.

[17] A. Goyal, G. Wang, and A. Bates, "R-caid: Embedding root cause analysis within provenance-based intrusion detection," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 2024, pp. 257–257.

[18] S. Li, F. Dong, X. Xiao, H. Wang, F. Shao, J. Chen, Y. Guo, X. Chen, and D. Li, "Nodlink: An online system for fine-grained apt attack detection and investigation," *arXiv preprint arXiv:2311.02331*, 2023.

[19] X. Han, T. F. J. Pasquier, A. Bates, J. Mickens, and M. I. Seltzer, "Unicorn: Runtime provenance-based detector for advanced persistent threats," in *27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020*. The Internet Society, 2020. [Online]. Available: https://www.ndss-symposium.org/ndss-paper/unicorn-runtime-provenance-based-detector-for-advanced-persistent-threats/

[20] S. Wang, Z. Wang, T. Zhou, H. Sun, X. Yin, D. Han, H. Zhang, X. Shi, and J. Yang, "Threatrace: Detecting and tracing host-based threats in

node level through provenance graph learning," *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 3972–3987, 2022.
[21] M. N. Hossain, S. M. Milajerdi, J. Wang, B. Eshete, R. Gjomemo, R. Sekar, S. Stoller, and V. Venkatakrishnan, "{SLEUTH}: Real-time attack scenario reconstruction from {COTS} audit data," in *26th USENIX Security Symposium (USENIX Security 17)*, 2017, pp. 487–504.
[22] 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 (SP)*, 2020.
[23] S. M. Milajerdi, R. Gjomemo, B. Eshete, R. Sekar, and V. Venkatakrishnan, "Holmes: Real-time apt detection through correlation of suspicious information flows," in *IEEE Symposium on Security and Privacy (SP)*, 2019.
[24] C. Xiong, T. Zhu, W. Dong, L. Ruan, R. Yang, Y. Cheng, Y. Chen, S. Cheng, and X. Chen, "Conan: A practical real-time apt detection system with high accuracy and efficiency," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 1, pp. 551–565, 2022.
[25] F. B. Kokulu, A. Soneji, T. Bao, Y. Shoshitaishvili, Z. Zhao, A. Doupé, and G.-J. Ahn, "Matched and mismatched socs: A qualitative study on security operations center issues," in *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, 2019, pp. 1955–1970.
[26] K. Bandla, "Apt notes," 2019. [Online]. Available: https://github.com/kbandla/APTnotes
[27] D. T. program, "Transparent Computing Engagement 3 Data Release," 2020. [Online]. Available: https://github.com/darpa-i2o/Transparent-Computing/blob/master/README-E3.md
[28] M. Zipperle, F. Gottwalt, E. Chang, and T. Dillon, "Provenance-based intrusion detection systems: A survey," *ACM Comput. Surv.*, vol. 55, no. 7, dec 2022. [Online]. Available: https://doi.org/10.1145/3539605
[29] Z. Li, Q. A. Chen, R. Yang, Y. Chen, and W. Ruan, "Threat detection and investigation with system-level provenance graphs: a survey," *Computers & Security*, vol. 106, p. 102282, 2021.
[30] 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*, 2021.
[31] A. Alsaheel, Y. Nan, S. Ma, L. Yu, G. Walkup, Z. B. Celik, X. Zhang, and D. Xu, "Atlas: A sequence-based learning approach for attack investigation." in *USENIX Security Symposium*, 2021, pp. 3005–3022.
[32] Q. Wang, W. U. Hassan, D. Li, K. Jee, X. Yu, K. Zou, J. Rhee, Z. Chen, W. Cheng, C. A. Gunter *et al.*, "You are what you do: Hunting stealthy malware via data provenance analysis." in *NDSS*, 2020.
[33] S. M. Milajerdi, B. Eshete, R. Gjomemo, and V. Venkatakrishnan, "Poirot: Aligning attack behavior with kernel audit records for cyber threat hunting," *ACM SIGSAC Conference on Computer and Communications Security*, 2019.
[34] F. Welter, F. Wilkens, and M. Fischer, "Tell me more: Black box explainability for apt detection on system provenance graphs," in *ICC 2023-IEEE International Conference on Communications*. IEEE, 2023, pp. 3817–3823.
[35] A. Goyal, X. Han, G. Wang, and A. Bates, "Sometimes, you aren't what you do: Mimicry attacks against provenance graph host intrusion detection systems," in *30th Network and Distributed System Security Symposium (NDSS 23)*, 2023.
[36] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.
[37] J. R. Martins and A. Ning, *Engineering design optimization*. Cambridge University Press, 2021.
[38] J. Newsome and D. X. Song, "Dynamic taint analysis for automatic detection, analysis, and signaturegeneration of exploits on commodity software." in *NDSS*, vol. 5. Citeseer, 2005, pp. 3–4.
[39] W. Xu, S. Bhatkar, and R. Sekar, "Taint-enhanced policy enforcement: A practical approach to defeat a wide range of attacks." in *USENIX Security Symposium*, 2006, pp. 121–136.
[40] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel, "Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps," *ACM sigplan notices*, vol. 49, no. 6, pp. 259–269, 2014.
[41] S. Ma, X. Zhang, and D. Xu, "Protracer: Towards practical provenance tracing by alternating between logging and tainting," in *23rd Annual Network And Distributed System Security Symposium (NDSS 2016)*. Internet Soc, 2016.
[42] K. H. Lee, X. Zhang, and D. Xu, "High accuracy attack provenance via binary-based execution partition." in *NDSS*, vol. 16, 2013.
[43] J. Park, D. Nguyen, and R. Sandhu, "A provenance-based access control model," in *2012 tenth annual international conference on privacy, security and trust*. IEEE, 2012, pp. 137–144.
[44] T. Pasquier, X. Han, M. Goldstein, T. Moyer, D. Eyers, M. Seltzer, and J. Bacon, "Practical whole-system provenance capture," in *Proceedings of the 2017 Symposium on Cloud Computing*, 2017, pp. 405–418.
[45] S. S. Khan and M. G. Madden, "One-class classification: taxonomy of study and review of techniques," *The Knowledge Engineering Review*, vol. 29, no. 3, pp. 345–374, 2014.
[46] D. M. Tax and R. P. Duin, "Data domain description using support vectors." in *ESANN*, vol. 99, 1999, pp. 251–256.
[47] ——, "Support vector domain description," *Pattern recognition letters*, vol. 20, no. 11-13, pp. 1191–1199, 1999.
[48] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.
[49] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in neural information processing systems*, vol. 30, 2017.
[50] Y. Chen, Y. Chi, J. Fan, and C. Ma, "Gradient descent with random initialization: Fast global convergence for nonconvex phase retrieval," *Mathematical Programming*, vol. 176, pp. 5–37, 2019.
[51] DARPA, "Transparent computing engagement 3 data release," 2018, accessed: 2024-09-01. [Online]. Available: https://drive.google.com/drive/folders/1QlbUFWAGq3Hpl8wVdzOdIoZLFxkII4EK
[52] ——, "Transparent computing engagement 5 data release," 2019, accessed: 2024-09-01. [Online]. Available: https://github.com/darpa-i2o/Transparent-Computing
[53] CISA, "North korean advanced persistent threat focus: Kimsuky," 2020. [Online]. Available: https://www.cisa.gov/uscert/ncas/alerts/aa20-301a
[54] R. Canary, "Explore atomic red team," 2023. [Online]. Available: https://atomicredteam.io/
[55] T. R. Lampert, "CHAOS: a PoC that allow generate payloads and control remote operating systems. Features like persistency and others can be achieved with a program updating and using the Dnsmasq service, which is also a feature of chaos." https://github.com/tiagorlampert/CHAOS, Year the repository was last updated.
[56] HackTricks, "D-bus enumeration & command injection privilege escalation," 2024, accessed: 2024-08-01. [Online]. Available: https://book.hacktricks.xyz/linux-hardening/privilege-escalation/d-bus-enumeration-and-command-injection-privilege-escalation
[57] Y. Nesterov, "A method for unconstrained convex minimization problem with the rate of convergence o (1/k2)," in *Dokl. Akad. Nauk. SSSR*, vol. 269, no. 3, 1983, p. 543.
[58] M. D. Zeiler, "Adadelta: an adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.
[59] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization." *Journal of machine learning research*, vol. 12, no. 7, 2011.
[60] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

APPENDIX A
ATTACK SCENARIOS IN DATASETS

For benign scenarios, the SOC collaborating with us provided three detailed operation environments in the real world: Streaming simulates a data store and streaming server; Dev simulates a development server; Cloud simulates a cloud server. The simulated APT attack chains are generated using the Atomic Red Team [54]. We designed three types of simulated attacks: fully randomized, partially randomized, and deterministic attacks. Each step in a fully randomized attack is randomly selected from the Atomic Red Team library. A partially randomized attack has determined steps at some stages but randomly chooses steps for the rest of the steps. A deterministic attack has defined steps at every attack stage. We generated one deterministic and three partially randomized attacks in our evaluation. We conduct the experiments following the attack campaign pattern in DARPA Engagements to

make it as realistic as possible. Table VII specifies the apps or processes running in each simulated environment.

TABLE VII: Simulated Scenario Summary

Attack	Attack Description
ReverseShell(RS)	Connect to the victim's host, collect system information, and install multiple applications
WebShell(WS)	Connect to the victim's host, collect system information and modify system configurations
AttackChain(AC)	A randomly generated attack chain using Atomic Red Team [54]
Kimsuky(Kim)	A simulated North Korean APT Kimsuky
Chaos	A malicious payload that allows remote control
Benign	**App or Process Involved**
Streaming	Kafka, Mysql, Nginx, Redis, Zookeeper
Developing	Iptables, Zabbix, Gitlab, VSCode, Influxdb, lvextend, redis-server, Qingtengyun, Baota, docker
Cloud	finalshell, postgres, web.py, Apache Struts 2, saltstack, Cloud Workload Protection Platforms

APPENDIX B
CAPTAIN POLICIES

We adopt tag propagation rules, detailed in Table VIII, from MORSE [22] for unbiased comparison. $dtag$ refers to the data tag and $ptag$ refers to the code tag. Please refer to the original paper for formal and detailed descriptions of the hyperparameters such as T_{qb}, T_{qe}, d_b, and d_e. All hyperparameters are set according to the recommendation in [22]. We also adopt the alarm generation rules from MORSE [22], detailed in Table IX. $incl\_exec(p)$ means p is contains the execution permission. $socket(o)$ holds when o refers to a socket.

APPENDIX C
MATHEMATICAL PROOF OF THE OPTIMIZATION

In this section, we prove the equations of gradients of A, G, and T to the loss, i.e. Eq. 7, Eq. 9, and Eq. 10 in §IV-C. According to the loss function Eq.4 and the chain rule in calculus, we have:

$$\begin{aligned}
\frac{\partial Loss}{\partial a_n} &= \sum_{e\in E}\frac{\partial \mathcal{L}(e)}{\partial f}\cdot\frac{\partial f}{\partial a_n} + \alpha\cdot(a_n - a_0)\\
\frac{\partial Loss}{\partial g_e} &= \sum_{e\in E}\frac{\partial \mathcal{L}(e)}{\partial f}\cdot\frac{\partial f}{\partial g_e} + \gamma\cdot(g_e - g_0)\\
\frac{\partial Loss}{\partial thr_e} &= \sum_{e\in E}\frac{\partial \mathcal{L}(e)}{\partial f}\cdot\frac{\partial f}{\partial thr_e} + \tau\cdot(thr_e - thr_0)
\end{aligned} \tag{13}$$

where $\frac{\partial \mathcal{L}}{\partial f}$ is identical for all events according to Eq. 3 and α, γ, and τ are pre-defined hyperparameters. We have

$$\begin{aligned}
f(e) &= tag^{new}_{dest} - thr_e\\
&= g_e\cdot tag_{rule} + (1-g_e)\cdot tag_{dest} - thr_e\\
\frac{\partial f}{\partial thr_e} &= -1\\
\frac{\partial f}{\partial a_n} &= g_e\cdot\frac{\partial tag_{rule}}{\partial a_n} + (1-g_e)\cdot\frac{\partial tag_{dest}}{\partial a_n}
\end{aligned}$$

$f(e)$ is influenced by the propagation rate of the current event (denoted by g_e) and all previous events (denoted by $g_{e'}$). Consequently, every time when tag^{new}_{dest} is updated, we recalculate $\frac{\partial f}{\partial g_e}$ and $\frac{\partial f}{\partial g_{e'}}$.

$$\begin{aligned}
&Since\ \frac{\partial g_e}{\partial g_{e'}} = 0,\\
&\frac{\partial f}{\partial g_{e'}} = g_e\cdot\frac{\partial tag_{rule}}{\partial g_{e'}} + (1-g_e)\cdot\frac{\partial tag_{dest}}{\partial g_{e'}}
\end{aligned} \tag{14}$$

The calculation of $\frac{\partial f}{\partial g_e}$ involves the product of two functions. According to the Product Rule:

$$if\ h(x) = f(x)g(x),\ then\ h'(x) = f'(x)g(x) + f(x)g'(x)$$

we have

$$\frac{\partial f}{\partial g_e} = g_e\frac{\partial tag_{rule}}{\partial g_e} + tag_{rule} + (1-g_e)\frac{\partial tag_{dest}}{\partial g_e} - tag_{dest} \tag{15}$$

APPENDIX D
MATHEMATICAL PROOF OF THE HEURISTIC HYPERPARAMETER SETTING

According to Eq.13, to let $\frac{\partial Loss}{\partial a_n}$, $\frac{\partial Loss}{\partial g_e}$, and $\frac{\partial Loss}{\partial thr_e}$ be zero, we have

$$\begin{aligned}
\alpha &= -\sum_{e\in E}\frac{\partial \mathcal{L}(e)}{\partial f}\cdot\frac{\partial f}{\partial a_n}\cdot(a_n - a_0)^{-1}\\
\gamma &= -\sum_{e\in E}\frac{\partial \mathcal{L}(e)}{\partial f}\cdot\frac{\partial f}{\partial g_e}\cdot(g_e - g_0)^{-1}\\
\tau &= -\sum_{e\in E}\frac{\partial \mathcal{L}(e)}{\partial f}\cdot\frac{\partial f}{\partial thr_e}\cdot(thr_e - thr_0)^{-1}
\end{aligned} \tag{16}$$

Since $\mathcal{L}(e) = max(0, (1-f(e))^2 - 1)$ and $-1 < f(e) < 0$,

$$-\sum_{e\in E}\frac{\partial \mathcal{L}(e)}{\partial f} = \sum_{e\in E}2(1-f(e))$$

$\frac{\partial f}{\partial thr_e} = -1$, $0 < \frac{\partial f}{\partial a_n} < 1$, $-1 < \frac{\partial f}{\partial g_e} < 0$. As mentioned in §IV-B, a_0 is usually 0 while g_0 is 1. We can make the following estimations:

$$\begin{aligned}
f(e) \approx -0.5, \partial f/\partial a_n \approx 0.5, \partial f/\partial g_e \approx -0.5\\
a_n - a_0 \approx 0.5, g_e - g_0 \approx -0.5, thr_e - thr_0 \approx -0.25
\end{aligned} \tag{17}$$

Let $N(e)$ be the number of the allowed false alarms related to event e and $N(n)$ be the number of the allowed false alarms related to node n.

$$\begin{aligned}
\alpha &= \sum_{e\in E}2(1-f(e))\cdot\frac{\partial f}{\partial a_n}\cdot(a_n - a_0)^{-1} \approx 3N(n)\\
\gamma &= \sum_{e\in E}2(1-f(e))\cdot\frac{\partial f}{\partial g_e}\cdot(g_e - g_0)^{-1} \approx 3N(e)\\
\tau &= \sum_{e\in E}2(1-f(e))\cdot\frac{\partial f}{\partial thr_e}\cdot(thr_e - thr_0)^{-1} \approx 12N(e)
\end{aligned} \tag{18}$$

APPENDIX E
ADDITIONAL EXPERIMENT RESULTS

A. Gradients Storage During Training

We maintain two dictionaries for each node n to save the gradients of tag_n, $\nabla^A_n : \{n' : \frac{\partial tag_n}{\partial a'_n}, n' \in N\}$ and $\nabla^G_n : \{e : \frac{\partial tag_n}{\partial g_e}, e \in E\}$. Theoretically, the worst case for saving ∇^A_n is $O(|N|^2)$, and $O(|N||E|)$ for saving ∇^G_n. Please note that we only store non-zero gradients. Every time after the gradients are calculated, we add the non-zero values into ∇^A_n and ∇^G_n and discard the too-small gradients (e.g., $< 10^{-5}$). In practice, most gradients would always be zero for three

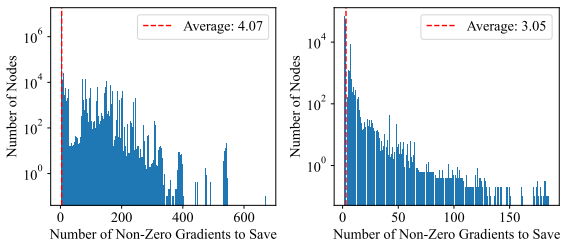
TABLE VIII: Tag propagation policies

Event	Tag to update	New tag value for different subject types		
		benign	suspect	suspect environment
$create(s, x)$	$x.dtag$	$s.dtag$	$s.dtag$	$s.dtag$
$read(s, x)$	$s.dtag$	$\min(s.dtag, x.dtag)$	$\min(s.dtag, x.dtag)$	$\min(s.dtag, x.dtag)$
$write(s, x)$	$x.dtag$	$\min(s.dtag + a_b, x.dtag)$	$\min(s.dtag, x.dtag)$	$\min(s.dtag + a_e, x.dtag)$
$load(s, x)$	$s.ptag$	$\min(s.ptag, x.itag)$		
	$s.dtag$	$\min(s.dtag, x.dtag)$		
$exec(s, x)$	$s.ptag$	$x.itag$	$\min(x.itag, susp\_env)$	$x.itag$
	$s.dtag$	$\langle 1.0, 1.0 \rangle$	$\min(s.dtag, x.dtag)$	$\min(s.dtag, x.dtag)$
$inject(s, s')$	$s'.stag$	$\min(s'.stag, s.itag)$		
	$s'.dtag$	$\min(s.dtag, s'.dtag)$		
$periodically$:	$s.dtag$	$\max(s.dtag, d_b * s.dtag + (1 - d_b) * T_q b)$	no change	$\max(s.dtag, d_e * s.dtag + (1 - d_e) * T_q e)$

TABLE IX: Alarm generation policies

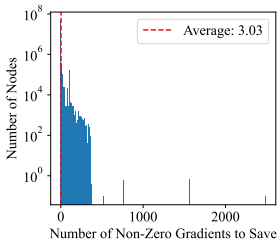
Name	Description	Operation(s)	Data integrity condition	Other conditions
MemExec	Prepare binary code for execution	$mmap(s, p)$, $mprotect(s, p)$	$s.itag < 0.5$	$incl\_exec(p)$
FileExec	Execute file-based malware	$exec(s, o)$, $load(s, o)$	$s.itag < 0.5$	$s.ptag > 0.5$
Inject	Process injection	$inject(s, s')$	$s.itag < 0.5$	$s'.ptag > 0.5$
ChPerm	Prepare malware file for execution	$chmod(s, o, p)$	$s.itag < 0.5$	$incl\_exec(p)$
Corrupt	Corrupt files	$write(s, o)$, $mv(s, o)$, $rm(s, o)$	$s.itag < 0.5$	-
Escalate	Privilege escalation	$any(s)$	$s.itag < 0.5$	changed userid
DataLeak	Confidential data leak	$write(s, o)$	$s.itag < 0.5$	$s.ctag < 0.5$, $socket(o)$
MalFileCreation	Ingress Tool Transfer	$create(s, o)$	$s.itag < 0.5$	$File(o)$

reasons: 1) In most cases, the provenance graph is relatively sparse. For example, a specific process would only read a small set of files on the system; 2) The gradients only get updated when the event changes the tags. Therefore, most benign nodes and edges would not be added to ∇_n^A and ∇_n^G; 3) As g_e is smaller than 1, some gradients would become near-zero after many iterations. Fig. 8 clearly illustrates the long-tailed distribution of the non-zero gradients on the DARPA dataset, which means we only need to store a tiny number of non-zero gradients for most nodes. The average numbers are less than 5 for all those four datasets. Hence, the additional overhead is reasonable during training.

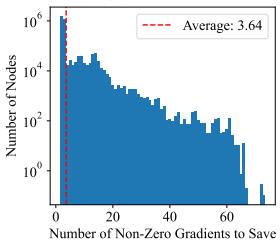


(a) T-3

(b) C-3



(c) T-5



(d) C-5

Fig. 8: The distribution of the numbers of non-zero gradients to A and G that each nodes need to save during tag propagation.

B. Mimicry Attack Experiment

We insert events into the dataset to mimic benign behaviors involving malicious entities. we used the E3 CADETS for this experiment. In the attack, `/tmp/test` file is downloaded and executed, serving as a command and control malware to carry out the rest of the attack. We identified a benign file `/dev/tty` to serve as the target to mimic and replicated interactions with `/dev/tty` on `/tmp/test` to let `/tmp/test` mimic normal behaviors.

C. Ablation Study on Learning Rate

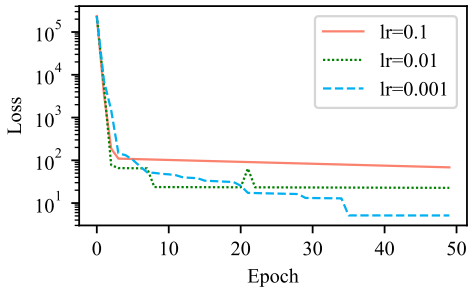


Fig. 9: Training loss using different learning rates.

Fig. 9 shows the training loss with different learning rates on CADETS from DARPA Engagement 3. From the figure, if the training is limited to 20 epochs (due to time and resource issues), a learning rate of 0.01 is advisable; for longer training durations, 0.001 is likely a more preferred choice.