THREATRACE: Detecting and Tracing Host-based Threats in Node Level Through Provenance Graph Learning

Su Wang, Zhiliang Wang, *Member, IEEE,* Tao Zhou, Xia Yin, *Senior Member, IEEE,* Dongqi Han, Han Zhang, Hongbin Sun, Xingang Shi, *Member, IEEE,* Jiahai Yang, *Senior Member, IEEE*

***Abstract*—Host-based threats such as Program Attack, Malware Implantation, and Advanced Persistent Threats (APT), are commonly adopted by modern attackers. Recent studies propose leveraging the rich contextual information in data provenance to detect threats in a host. Data provenance is a directed acyclic graph constructed from system audit data. Nodes in a provenance graph represent system entities (e.g., $processes$ and $files$) and edges represent system calls in the direction of information flow. However, previous studies, which extract features of the provenance graph, are not sensitive to the small quantity of threat-related entities and thus result in low performance when hunting stealthy threats.**

We present THREATRACE, an anomaly-based detector that detects host-based threats at system entity level without prior knowledge of attack patterns. We tailor GraphSAGE, an inductive graph neural network, to learn every benign entity's role in a provenance graph. THREATRACE is a real-time system, which is scalable of monitoring a long-term running host and capable of detecting host-based intrusion in their early phase. We evaluate THREATRACE on five public datasets. The results show that THREATRACE outperforms seven state-of-the-art host intrusion detection systems.

***Index Terms*—Host-based intrusion detection, graph neural network, data provenance, multimodel framework.**

I. Introduction

NOWADAYS, attackers tend to perform intrusion activities in important hosts of those big enterprises and governments [1]. They usually exploit zero-day vulnerabilities to launch an intrusion campaign in a target host stealthily and persistently, which makes it hard to detect.

Recent studies [2], [3], [4], [5], [6], [7], [8], [9] propose leveraging the rich contextual information on data provenance to perform host-based intrusion detection. Compared to the raw system audit data, data provenance contains richer contextual information, which is useful for separating the long-term behavior of threats and benign activities [7], [10].

Some anomaly-based methods [6], [7] use graph kernel algorithms to dynamically model the graph and detect abnormal graphs by clustering approaches. However, the provenance graph of a system under stealthy intrusion campaigns may be similar to those of benign systems. Therefore, leveraging graph kernel to extract the features of the graph is not sensitive to the small quantity of anomalous nodes in the graph. The graph-kernel-based methods also lack the capability to locate the position of anomalous nodes, which is essential to trace abnormal behavior and fix the system. [8] focuses on hunting malware by detecting anomalous paths in a provenance graph. However, some complex threats (e.g., APT) split their campaigns into several parts instead of appearing in a complete path, which makes it difficult to be detected at path level. Misuse-based methods [2], [3], [9] typically define some attack patterns and use them to match anomalous behavior. Considering that modern attackers frequently exploit zero-day vulnerabilities, misuse-based methods are difficult to detect unknown threats.

We present THREATRACE, an anomaly-based detector that is capable of efficiently detecting stealthy and persistent host-based threats at node level. THREATRACE takes data provenance graphs as source input and tailors an inductive graph neural network framework (GraphSAGE) [11] to learn the rich contextual information in data provenance graphs. GraphSAGE is a Graph Neural Network (GNN). GNN is a group of neural network models designed for various graph-related work and has succeeded in many domains (e.g., Computer Vision [12], Natural Language Processing [13], Chemistry and Biology [14], etc.). We utilize its ability to learn structural information about a node's role in a provenance graph [11] for node-level threat detection. Note that THREATRACE aggregates information about the node neighborhood on the provenance graph, which is similar to node embedding methods. The main difference is that the core of THREATRACE is a deep learning method, which aims at addressing an end-to-end anomalous nodes detection task while node embedding methods aim at extracting nodes' low-dimensional vector representations [15].

Using GraphSAGE for threat detection is challenging: 1) How to train the model without prior knowledge of the zero-day attack patterns? 2) How to tackle the problem of data imbalance [16]?

We tailor a novel GraphSAGE-based framework that tackles those challenges. Different from the previous graph level detection methods, THREATRACE learns every benign node's role in the data provenance graphs to capture stealthy abnormal behavior without prior knowledge of attack patterns. THREATRACE is a semi-supervised anomaly detection method, which requires the training data to be benign. We design a multi-model framework to learn different kinds of benign nodes, which tackles the problem of data imbalance and effectively improves the detection performance. THREATRACE is a real-time system which can be deployed in a long-term running system with acceptable computation and memory overhead. It is capable of detecting host intrusions in their early phase and tracing the location of anomalous behavior.

We evaluate THREATRACE in five public datasets. The results demonstrate that THREATRACE can effectively detect host-based threats, which have a small proportion in the whole-system provenance graph, with fast processing speed and acceptable resource overhead.

Our paper makes contributions summarized as follows:

- **Novel node level threat detection**. To the best of our knowledge, THREATRACE is the first work to formalize the host-based threat detection problem as an anomalous nodes detection and tracing problem in a provenance graph. We propose a novel GraphSAGE-based multi-model framework to detect stealthy threats at node level.
- **High detection performance and novel capability**. We evaluate THREATRACE's detection performance in five public datasets and compare it with three state-of-the-art host-based threat detection approaches. THREATRACE outperforms them in these datasets. We further evaluate THREATRACE's threat tracing ability in a host. Results show that THREATRACE can successfully detect and trace the anomalous elements.
- **Complete system and open source**. We implement an open-source host-based threat detection system [1].

This paper is organized as follows. Related work is introduced in §II. Background and motivation of our work are introduced in §III. We introduce the threat model in §IV. An overall description of THREATRACE is presented in §V. We introduce the experiments in §VI and discuss some issues and limitations in §VII. We conclude this paper in §VIII.

II. Related Work

We study the problems of provenance-based threat detection and anomaly tracing. Thus, we discuss related work in these areas.

Provenance-based threat detection. Data provenance is attractive in the host threat detection community. The related studies can be classified as misuse-based and anomaly-based.

Misuse-based methods detect abnormal behavior based on learning patterns of known attacks. Holmes [3] focuses on the alert generation, correlation, and scenario reconstruction of host-based threats. It matches a prior definition of exploits in a provenance graph based on expert knowledge of existing TTPs (Tactics, Techniques, and Procedures). Poirot [2] detects threats based on correlating a collection of indicators found by other systems and constructs the attack graphs relying on expert knowledge of existing cyber threat reports. It detects threats according to the graph matching of the provenance graphs and attack graphs. It is hard for them to detect unknown threats, which are not included in the TTPs and threat reports.

Anomaly-based approaches learn models of benign behavior and detect anomalies based on the deviations from the models. StreamSpot [6] proposes detecting intrusion by analyzing information flow graphs. It extracts features of the graph to learn benign models and uses a clustering approach to detect abnormal graphs. Unicorn [7] further proposes using a WL-kernel-based method to extract the features of the whole graph and learning evolving models to detect abnormal graphs. Graph kernel methods are used to measure the similarity of different graphs. Unicorn has better performance than StreamSpot due to the contextualized graph analysis and evolving models. However, because of the restriction of graph kernel methods, they have difficulty in detecting stealthy threats. IPG [17] embeds the provenance graph of each host and reports suspicious hosts based on an autoencoder method. As a graph level approach, IPG has the same limitations as StreamSpot and Unicorn. SHADEWATCHER [18] maps the concepts of system entity interactions to the concepts of user-item interactions in recommendation, detects cyber threats by predicting the preferences of a system entity on its interactive entities. Graph neural networks are used to improve detection effectiveness. ProvDetector [8] is proposed to detect malware by exploring the provenance graph. ProvDetector embeds paths in a provenance graph and uses Local Outlier Factor method [19] to detect malware. Besides malware, host-based threats are more diverse. Therefore, it is not enough to mine threats from the paths of the provenance graph. Pagoda [20] focuses on detecting host-based intrusion in consideration of both detection accuracy and detection time. Pagoda requires a rule database, which is different from THREATRACE.

Besides provenance, there are other data sources used for host-based anomaly detection. Log2vec [16] proposes to detect host threats based on anomalous logs detection. It uses logs to construct a heterogeneous graph, extracts log vectors based on graph embedding, and detects malicious logs based on clustering. It is different from provenance-based methods because nodes in a provenance graph are entities of a system instead of logs. DeepLog [21] uses Long Short-Term Memory (LSTM) to model normal log patterns and detect anomalous log based on deviation. LogRobust [22] and LogGAN [23] are two other anomalous logs detectors based on LSTM. LogRobust proposes an attention-based Bi-LSTM model to capture the contextual information in the log sequences. It is capable of identifying and handling unstable log events. LogGAN is the first approach to apply adversarial learning for anomalous log detection. The generative adversarial network (GAN) component relieves the effect caused by the imbalanced quantity of benign and anomalous instances. [24] detects stealthy program attacks by modeling normal program traces. It uses clustering methods to detect anomaly. [25] models each state as the invocation of a system call from a particular call site by a finite-state automaton (FSA). However, neither program traces nor system calls are suitable for stealthy threat detections due to the lack of contextual information. [26] takes a graph of authenticating entities as input to detect the lateral movement of APT. Nodes in the authentication graph represent either machines or users. However, this approach cannot detect intrusion campaigns within a single host.

Anomaly tracing. After deploying an intrusion detection system in a host under monitoring, it is important to trace abnormal behavior instead of only raising alarms. THREATRACE uses the node classification framework to directly trace anomalies after detecting them. Anomaly-based state-of-the-art detectors [6], [7] raise alarms when the provenance graph of the system is detected as abnormal. However, they cannot trace the

[1] https://github.com/threaTrace-detector/threaTrace/.

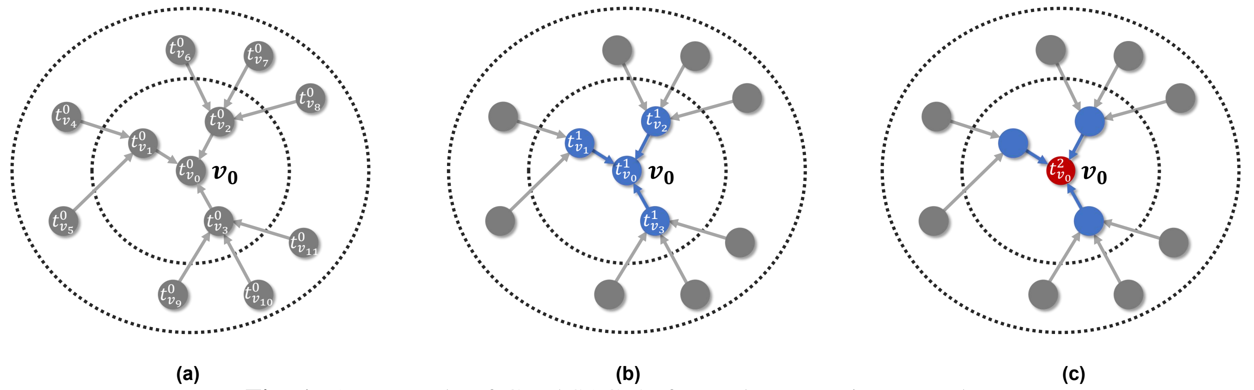

Fig. 1: An example of GraphSAGE's forward propagation procedure.

position of anomalies. Detectors that are capable of tracing the location of anomalies are usually misuse-based such as Holmes [3] and Poirot [2]. They need prior knowledge of abnormal graph patterns. Nodoze [10] is another approach to identify anomalous paths in provenance graphs. Unlike the detectors mentioned above, Nodoze is a secondary triage tool which needs alerts from other detectors as input. Alerts from the current anomaly-based threats detector are not suitable for Nodoze because they are graph-level and do not have detailed information. RapSheet [9] is another secondary triage system which uses alerts from other TTPs-based EDR (Endpoint Detection and Response) as input and filters false positives in constructed TPGs (Tactical Provenance Graphs). RapSheet is faced with the same problem as those misuse-based detection methods. SLEUTH [27] performs tag and policy-based attack detection and tag-based root-cause and impact analysis to construct a scenario graph. PrioTracker [28] quantifies the rareness of each event to distinguish abnormal operations from normal system events. PrioTracker proposes a forward tracking technique for timely analysis of attack causality. MORSE [29] researches the dependence explosion problem in the retracing of attacker's steps. The attack detection part of MORSE's framework is the same as the SLEUTH [27] system. The attack detection part of SLEUTH [27] and MORSE [29] are both misuse-based. ATLAS [30] proposes a sequence-based learning approach for detecting threats and constructing the attack story. Different from the misuse-based methods [2], [3], [29], [27], ATLAS uses attack training data to learn the co-occurrence of attack steps through temporal-ordered sequences. The methods introduced above propose to trace the intrusion and construct the attack story. THREATRACE can trace the anomaly without the knowledge database or attack training data as input. However, it cannot construct the attack story. We plan to research the gap between anomaly-based methods and attack story construction in future work.

III. Background & Motivation

A. Data provenance

In recent years, data provenance is proposed to be a better data source for host-based threat detection. It is a directed acyclic graph constructed from system audit data, which represents the relationship between subjects (e.g., $processes$) and objects (e.g., $files$) in a system. Data provenance contains rich contextual information for threat detection.

B. GraphSAGE

GraphSAGE [11] is a general and inductive GNN framework for efficiently generating node embeddings with node features. Unlike the transductive methods, which embed nodes from a single fixed graph, the inductive GraphSAGE learns embedding functions and operates on evolving graphs. Timeliness is an essential factor for preventing intrusion. Therefore, as an inductive GNN approach, GraphSAGE is suitable for detecting threats via analyzing streaming provenance graphs.

The core of GraphSAGE is the forward propagation (FP) algorithm. We define a provenance graph G as a 5-tuple $(V, E, \mathcal{X}_v, \mathcal{X}_e, \mathcal{T}_e)$. V is the set of nodes in the graph and E is the set of directed edges. $\mathcal{X}_v : V \rightarrow \sum$ is a function that maps each node to its type from an alphabet $\sum$. $\mathcal{X}_e : E \rightarrow \sum$ is a function that maps each edge to its type. $\mathcal{T}_e$ is the timestamp of every edge $e \in E$. Note that we use the same notations in the rest of the paper. The FP algorithm takes graph information $G = (V, E, \mathcal{X}_v, \mathcal{X}_e, \mathcal{T}_e)$, features assigning function $\mathcal{F}$, hop number K, neighborhood function $\mathcal{N} : V \rightarrow 2^V$, and parameters of GraphSAGE model [11] (a set of weight matrices $\mathbf{W}^k, \forall k \in \{1, ..., K\}$; aggregator functions $AGGREGATE_k, \forall k \in \{1, ..., K\}$; non-linear activation function σ; the batch mode parameter $Batch\ size$) as input, and outputs the vector representations z_v for $\forall v \in V$. The parameters (e.g. aggregator functions, which are *mean aggregator* in our detector) of GraphSAGE are used to aggregate and process information from a node's ancestors. z_v is a N_n-dimensional vector where N_n is the number of different node types. The index of the biggest element in z_v means the predicted class.

Fig. 1 illustrates an example of the FP algorithm. In this example, hop number K is set as 2. v_0 is the node that we want to get the vector representation z_{v_0} through the FP algorithm. In Fig. 1 (a), the vector of each node $v_i \in V$ is initiated with its features $t^0_{v_i}$. In the first propagation procedure from Fig. 1 (a) to (b), v_0 and its first hop neighbor nodes v_i $(i \in \{1, 2, 3\})$ aggregate information of their neighbor nodes through $t^1_{\mathcal{N}(v_i)} \leftarrow AGGREGATE_1(\{t^0_u, \forall u \in \mathcal{N}(v_i)\})$ and $t^1_{v_i} \leftarrow \sigma(\mathbf{W}^1 \cdot \text{CONCAT}(t^0_{v_i}, t^1_{\mathcal{N}(v_i)}))$ $(i \in \{0, 1, 2, 3\})$. $\mathcal{N}(v_i)$ denotes the set of neighbor nodes of v_i. In the second propagation procedure from Fig. 1 (b) to (c), v_0 aggregates information from its neighbor nodes v_i $(i \in \{1, 2, 3\})$ through $t^2_{\mathcal{N}(v_0)} \leftarrow AGGREGATE_2(\{t^1_u, \forall u \in \mathcal{N}(v_0)\})$ and $t^2_{v_0} \leftarrow \sigma(\mathbf{W}^2 \cdot \text{CONCAT}(t^1_{v_0}, t^2_{\mathcal{N}(v_0)}))$. Finally, the vector representation z_{v_0} of v_0 is assigned as $t^k_{v_0}/\|t^k_{v_0}\|_2$.

z_v has different further usages in different phases. In the executing phase, the index of the biggest element in z_v is regarded as the predicted class. In the training phase, a loss value is calculated between z_v and the true class vector of v through a loss function. Then, the loss value is used to tune the weight matrices $\mathbf{W}^k$ through the typical backward propagation algorithm of deep learning models. In this paper, we use cross-entropy loss as the loss function.

GraphSAGE has been proved to be capable of learning structural information about a node's role in a graph [11]. Due to space constraints, we refer the readers with the interest of the theoretical analysis to the paper [11].

C. Motivation Example

We present an example (Fig. 2) to illustrate the limitation of state-of-the-art threat detection work and the intuition of our approach. The example is generated from the DARPA TC #3 THEIA experiment [31]. The dark nodes are related to intrusion campaigns. The attack on the graph lasts for two days. The attacker exploits `Firefox 54.0.1` backdoor to implant the file named `/home/admin/profile` in the victim host. It then runs as a process with root privilege to connect with the attacker operator console `b.b.b.b:80`. A file named `/var/log/mail` is implanted and elevated as a new process with root privileges. Finally, the attacker carries out a port scan for `c.c.c.c`.

Limitations of state-of-the-art threat detection work. There are some challenges to detect the threat presented in the example, which leads to limitations of the state-of-the-art threat detection work.

P1 : *Rules.* There are some misuse-based [2], [3], [9] methods for host-based intrusion detection. Rules are important for misuse-based methods, and TTP rules based on MITRE Att&CK are commonly used. However, it is hard to select a ruleset. Some MITRE ATT&CK behaviors are not always malicious [9]. If the rules are macroscopic, it will generate many false alarms. On the contrary, micro rules are hard to detect zero-day attacks.

P2 : *Stealthy attacker.* Anomaly-based methods Unicorn [7] and StreamSpot [6] use graph kernel algorithms to dynamically model the graph and detect abnormal graphs by clustering approaches. However, the intrusion may be stealthy, which means that a system's provenance graph under intrusion campaigns may be similar to those of benign systems. In this example, there are millions of benign nodes and less than 30,000 anomalous nodes. The proportion of anomalous nodes is less than 1%, which thus results in a high similarity of the attack graph and benign graph. Therefore, graph-kernel-based methods [6], [7] are insensitive to the small quantity of anomalous nodes. ProvDetector [8] proposes to select several rarest paths in a graph and detect anomalous paths through their embedding. ProvDetector achieves outstanding performance in malware detection. However, there are more complicate intrusions besides malware. In this example, there are thousands of edges generated by node ①, which cause challenges for path-level detection methods.

P3 : *Anomaly tracing.* Anomaly-based [6], [7] methods, which use the features of graphs for modeling, lack the capability of tracing the location of anomalous behavior. In this example, these methods can only raise alarms for the graph instead of the specific attack entities (e.g., `/home/admin/profile`).

P4 : *Expanding provenance.* Methods [3], [6] that store the provenance graph in memory lack the scalability on long-term running systems.

Intuition of our approach.

The central insight behind our approach is that even for a stealthy intrusion campaign, which tries to hide its behavior, the nodes corresponding to its malicious activities still have different behavior from benign nodes. In Fig. 2, the anomalous process node ① `/var/log/mail` has thousands of `connect` edges to remote IP nodes, which is different from a benign process node ② `/usr/bin/fluxbox`. This phenomenon inspires our work to formalize the host intrusion detection problem as an anomalous nodes detection problem. We tailor a GraphSAGE-based framework to complete a nodes detection task. The detection results of the attack in Fig. 2 are presented as a case study in §VI-C. The motivation example is also used to illustrate the design of our approach. We will go back to the example several times in §V.

IV. Threat Model

In this paper, we focus on detecting and tracing anomalous entities in a host caused by intrusion campaigns. We assume the adversary has the following characteristics:

- **Stealthy.** Instead of simply performing an attack, the attacker consciously hides their malicious activities, trying to mix their behavior with a large amount of benign background data, which makes the victim system perform like a benign mode.
- **Persistent.** The intrusion campaigns tend to last for a long time.
- **Frequent usage of zero-day exploits.** The attacker tends to use zero-day exploits to attack a system. Therefore, we assume that we do not have any attack patterns for training.
- **Has attack patterns in the provenance graph.** In order to complete a malicious activity which is different from benign activities, the attacker's behavior should leave some attack patterns in the provenance. The attack patterns would make the local structure of an attacker's node different from those of benign nodes with the same label. For example, in Fig. 2, the local structure of the attacker's $process$ node ① is significantly different from the benign $process$ node ②. As another example, a $process$, which never interacts with $files$ before and suddenly starts reading and writing on a $file$, may be an anomalous $process$ controlled by the attacker. As a provenance-based approach, THREATRACE is restricted by the granularity of the provenance graph. THREATRACE cannot detect threats which have no different patterns in the provenance graph compared to the benign entities learned in training phase. We further discuss this limitation in §VII.

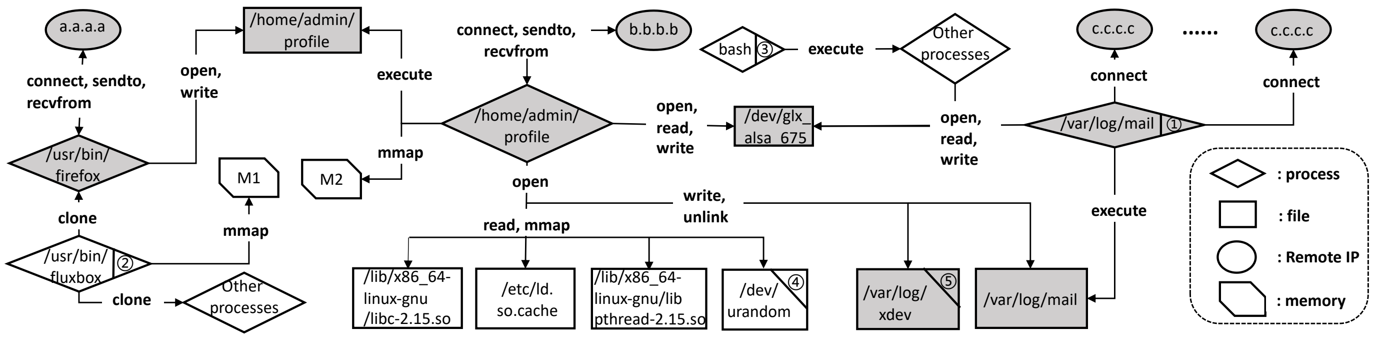

Fig. 2: A toy example of a provenance graph, which contains host-based intrusion behaviour.

We define an *entity* of a running system as benign when there are no malicious activities related to it. An *entity* is defined as abnormal when its behavior is different from a benign entity, which is usually caused by some intrusion campaigns. The purpose of THREATRACE is to detect the anomalous entities by monitoring the host and analyzing the nodes on the provenance graph.

We also assume the correctness of the following contents: 1) the provenance collection system; 2) the ability of GraphSAGE to learn structural information about a node's role in a graph, which has been proved in [11].

V. DESIGN

In this section, we present the design of THREATRACE. The main components are shown in Fig. 3 and presented as follows. We detail them further in the denoted subsections.

- **(§V-A) Data Provenance Generator.** This component collects audit data of a host in a streaming mode and transforms them into a provenance graph for following analysis.
- **(§V-B) Data Storage.** This component allocates data to the disk and memory. We store the whole graph in the disk to store the history information and maintain a subgraph with limited size in memory for training and detecting. This storage strategy guarantees the scalability (**P4**) and dynamic detection capability of THREATRACE.
- **(§V-C) Model.** This is the core component of THREATRACE. We use graph data originated and allocated by the previous two components as input and output anomalous nodes.
  It is challenging to make ideal abnormal nodes detection:
  C1: THREATRACE is a semi-supervised anomaly detection method, which does not have prior attack knowledge in the training phase. Therefore, we cannot train the model in traditional binary classification mode.
  C2: Host-based threat detection has the challenge of data imbalance [16]. Anomalous nodes may have a small proportion in the provenance graph during execution. Thus, it is more likely to raise false positives which may disturb the judgment and cause "threat fatigue problem" [32].
  C3: The difference between an anomalous node and benign nodes may not be obvious enough (e.g., nodes ④ and ⑤ in Fig. 2), which thus results in a false negative.
  In order to tackle these challenges, we tailor a GraphSAGE-based multi-model framework to learn the different classes of benign nodes in a provenance graph without abnormal data (**P1**, **C1**). Then we detect abnormal nodes based on the deviation from the predicted node type and its actual type. In this way, THREATRACE detects abnormal nodes that have a small proportion in a provenance graph under a stealthy intrusion campaign (**P2**) and directly locates them (**P3**). We propose a probability-based method, which reduces false positives and false negatives (**C2**, **C3**).
- **(§V-D) Alert and Trace.** We obtain the abnormal nodes from the previous component and determine whether to raise alerts and trace anomalies in this component. We set a waiting time threshold and a tolerant threshold to reduce false positives and determine whether to raise alerts or not. Because THREATRACE detects threats at node level, we directly trace the position of abnormal behavior in the local neighbor of the abnormal nodes.

A. Data Provenance Generator

Like many other provenance-based threat detection approaches [2], [3], [6], [7], we use an external tool Camflow [33] to construct a single, whole-system provenance graph with time order. Camflow provides strong security and completeness guarantees to information flow capture.

B. Data Storage

This module is designed to allocate graph data from the previous module to disk and memory for later usage. In a streaming mode, we append incoming nodes and edges to the whole graph which is stored in the disk. We also maintain a subgraph in memory consisting of $active$ nodes, $related$ nodes, and edges between them. We define $active$ nodes as the entities in the graph that are used to be trained or detected. Nodes that can reach an $active$ node in 2 hops are named as $related$ nodes, which contain history information for training or detecting. Fig. 1(b) can be an example of the subgraph maintaining in memory. Nodes in blue denote the $active$ nodes, and the gray nodes denote the $related$ nodes. The subgraph maintaining strategies are different in training and executing phase, which are presented in §V-C.

C. Model

This is the core component of THREATRACE. We design the framework based on the characteristics of host-based threats.

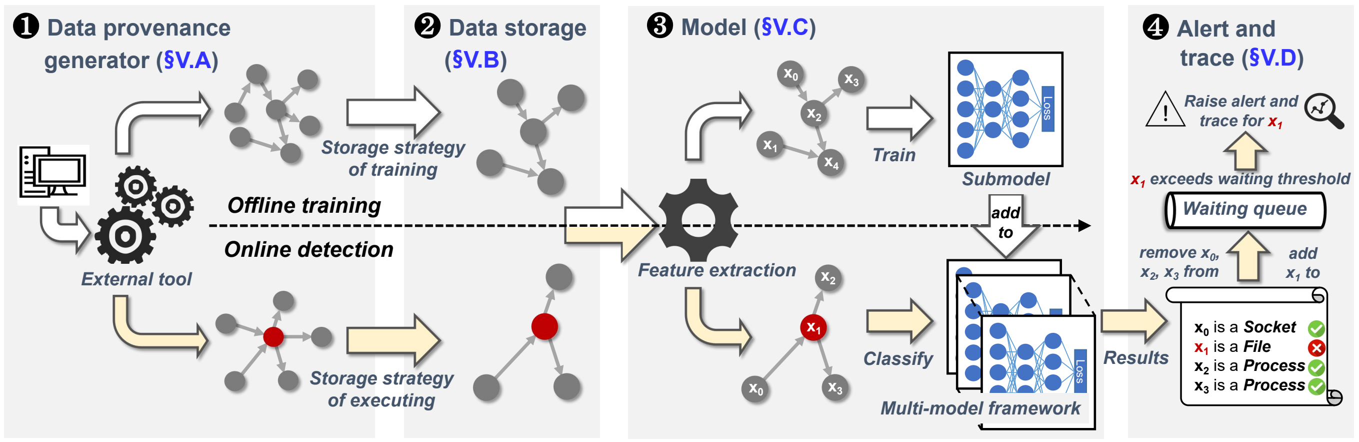

Fig. 3: The overview of components in THREATRACE. THREATRACE ❶ uses external tools to generate a streaming provenance graph (§V-A), ❷ stores the graph in disk and memory through different storage strategies (§V-B), ❸ trains the multi-model framework and uses the framework to detect the graph (§V-C), ❹ raises and traces alerts based on the detection results (§V-D).

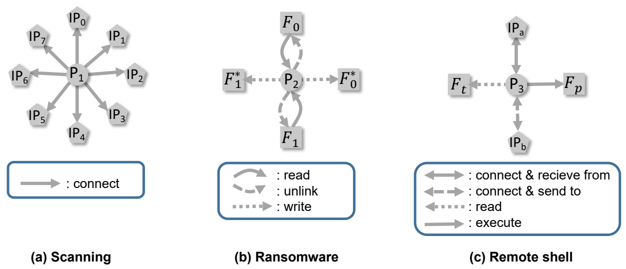

Fig. 4: Examples of three anomalous nodes with different kinds. (a) A scanning process P_1 which connects to different socket nodes IP_i; (b) A ransomware process node P_2 which reads files and generates the encrypted copies; (c) A remote shell process P_3 which recieves instructions from IP_a and sends the target file F_t to IP_b.

Feature Extraction. It is challenging to detect abnormal nodes without learning attack patterns (**C1**). The traditional anomaly detection task usually uses benign and abnormal data to train a binary classification model in supervised mode. However, because we assume we do not have prior knowledge about attack patterns during the training phase, we cannot train our model with binary labeled data. GraphSAGE has an unsupervised node classification mode, but it is unfit for threat detection because it assumes that adjacent nodes have a similar class. In a provenance graph, the adjacent nodes may have different classes (e.g., two adjacent nodes may be a *process* and a *file*).

Since GraphSAGE's unsupervised mode is unsuitable for threat detection, we tailor a feature extraction and label assignment approach, which allows THREATRACE to train the model in a supervised mode without abnormal data and learn every benign node's local structure information. We set a node's label as its node type and extract its features as the quantity distribution of different edge types related to it. The GraphSAGE model is trained with labeled data in a supervised mode, which learns different roles of benign nodes. The technical intuition behind the feature extraction approach is that the behavior (or purpose) of a node is reflected in the edges related to it. Take scanning process, ransomware process, and remote shell process as examples (Fig. 4). In Fig. 4 (a), the behavior of an anomalous scanning *process* P_1 is directly represented by plenty of *connect* edges out of P_1. For the ransomware process P_2 in Fig. 4 (b), it first reads some *files* and generates the encrypted copies. Then, it deletes (unlink) the original *files*. For the remote shell process P_3 (Fig. 4 (c)), it receives instructions from IP_a, executes F_p to escalate the privilege, and sends F_t to another remote address IP_b. These three kinds of anomalous process nodes have different local structures compared to benign process nodes (e.g. node ② in Fig. 2). Therefore, the model uses the extracted feature to learn the hidden distribution of different nodes. If a node is misclassified during the executing phase, it means that its underlying distribution is different from the benign nodes which are learned in the training phase. That is, it may be an abnormal node with a malicious task different from the benign nodes.

The specific extraction process is as follows. We first count the number of different node types (e.g., $file$, $process$, $socket$) and edge types (e.g., $read$, $write$, $fork$) as N_n and N_e before training. We further set a node type and edge type mapping function as $\mathcal{M}_v : \sum \rightarrow \mathbb{N}$ and $\mathcal{M}_e : \sum \rightarrow \mathbb{N}$ to map node type and edge type to an integer range from 0 to $N_n - 1$ and 0 to $N_e - 1$. We learn a function $\mathcal{L} : V \rightarrow \{0, ..., N_n - 1\}$ to assign label for $\forall v \in V$ as $\mathcal{L}(v) = \mathcal{M}_v(\mathcal{X}_v(v))$. We learn a function $\mathcal{F} : V \rightarrow \mathbb{N}^{2*N_e}$ to assign features for $\forall v \in V$ as $\mathcal{F}(v) = [a_0, a_1, ..., a_{N_e-1}, a_{N_e}, a_{N_e+1}, ..., a_{N_e*2-1}]$, where:

$$a_i = \begin{cases} \#\{e \in In(v) : i = \mathcal{M}_e(\mathcal{X}_e(e))\}, & i \in \{0, ..., N_e - 1\} \\ \#\{e \in Out(v) : i = \mathcal{M}_e(\mathcal{X}_e(e)) + N_e\}, & i \in \{N_e, ..., N_e * 2 - 1\} \end{cases} \quad (1)$$

Through our feature extraction method, the features of benign node ② and anomalous node ① in Fig. 2 are [0,0,0,0,0,0,0,2,1,0,0,0,0,0] and [0,0,0,0,0,0,0,0,1,1,1,1,25301], which are significantly different in the last dimension.

Training Method. In the training phase, we split the whole training graph into several subgraphs. Each subgraph is constructed by a number (we set it as 150,000 in this paper) of randomly selected *active* nodes, their *related* nodes, and edges between them. The sets of *active* nodes in different subgraphs are disjointed. Then, the model is trained by each subgraph G in order. Instead of using the whole graph for training, we only need to store one subgraph with a limited size in memory, which can guarantee the scalability.

Our model is based on GraphSAGE, which uses a forward propagation (FP) algorithm to aggregate information from a node's ancestors. The FP algorithm is introduced in §III-B. The choice of $Batch\ size$ (BS) affects the runtime overhead and we evaluate it in §VI-E. Hop number K (§III-B) denotes the size of ancestors which each node aggregates information from. We set K as 2 in this paper to balance the representing ability and runtime overhead. This is also the reason of defining $related$ nodes as nodes that can reach $active$ nodes in two hops (§V-B). Note that we have $K-1$ hidden layers in our model because the aggregation happens between two layers: the first aggregation happens from the input layer to the hidden layer, and the second aggregation happens from the hidden layer to the output layer. The FP algorithm (§III-B) repeats several times. After each iteration, we calculate the cross-entropy loss of z_v and tune the weight matrices $\mathbf{W}^k, \forall k \in \{1, ..., K\}$. Through the FP and tuning procedure, the model learns the representation of different classes of benign nodes by exploring their features and local structures.

Under a stealthy intrusion campaign, the proportion of anomalous nodes in the provenance graph may be small. Therefore, it is more likely to raise false positives that may disturb the judgment (**C2**). Motivated by the characteristics of intrusion detection scenes, we tailor a multi-model framework to reduce false positives. The characteristic of host-based intrusion detection task provides the intuition of the multi-model framework. In the intrusion detection scene: 1) The numbers of node types are unbalanced (e.g., there may be thousands of $process$ nodes and few $Remote\ IP$ nodes when there is no frequent network connection in the host). It could be very hard to mine the underlying representation with a single model because of the unbalanced node types. 2) Nodes with the same type may have different missions (e.g., the `fluxbox` process ② and `bash` process ③ in Fig. 2), which makes it hard to classify those nodes into one class with a single model. Therefore, it is unrealistic to train only one model to distinguish all classes of benign nodes.

We consider a node v has a dominant label $\mathcal{L}(v) = \mathcal{M}_v(\mathcal{X}_v(v))$ and a hidden label $\hat{\mathcal{L}}(v)$. The hidden label represents its specific function, which is just a concept for distinguishing nodes with the same dominant label and we do not know its value. We train multiple submodels to learn the representation of those nodes with small proportions and those nodes with the same dominant label but different hidden labels. The training method for the multi-model framework is shown in Alg. 1. We maintain a list X, which stores the nodes that have not been correctly classified. It is initialized with all $active$ nodes in the training subgraph. After training a submodel (line 6-13), we validate nodes of X in this model (line 15-21) and delete the correctly classified nodes from X. The nodes left in X will be used to train a new submodel. We repeat this procedure until there are no nodes in X, which means that the underlying representation of each node in the subgraph has been learned by a submodel.

Algorithm 1: The training method of the multi-model framework

```
Input: Subgraph $G = (V, E, \mathcal{X}_v, \mathcal{X}_e, \mathcal{T}_e)$; Features
       mapping function $\mathcal{F} : V \to \mathbb{N}^{2*N_e}$; Label
       mapping function $\mathcal{L} : V \to \{0, ..., N_n\}$; Hop
       number $K$; Times that FP algorithm iterates
       $epoch$
Output: Number of submodels $cnt$; Submodels
        $\{\mathbb{M}_0, \mathbb{M}_1, ..., \mathbb{M}_{cnt-1}\}$
1  Remove the correctly classified $active$ nodes from $V$;
2  $X \leftarrow active$ nodes in $V$;
3  $cnt \leftarrow 0$;
4  while $X$ not empty do
5      $\hat{V} \leftarrow$ nodes in $K$-hop neighborhood of $\forall v \in X$;
6      for $k = 1...epoch$ do
7          $\hat{G} \leftarrow (\hat{V}, E, \mathcal{X}_v, \mathcal{X}_e, \mathcal{T}_e)$ ;
8          $z \leftarrow$ FP$(\hat{G}, \mathcal{F}, K, AGGREGATE_k, \mathbf{W}^k, \sigma)$;
9          //$z$ are the representations of $\forall v \in \hat{V}$
10         $\hat{z} \leftarrow z_{\forall v \in X}$;
11         $loss \leftarrow$ cross_entropy$(\hat{z}, \mathcal{L}(X))$;
12         $\mathbf{W}^k \leftarrow$ backward$(loss)$;
13     end
14     $M \leftarrow$ softmax$(\hat{z})$;
15     for $v \in X$ do
16         $C_v \leftarrow$ index of the biggest element in $M_v$;
17         $\hat{C}_v \leftarrow$ index of the second biggest element in $M_v$;
18         if $C_v = \mathcal{L}(v)$ and $M_{vC_v}/M_{v\hat{C}_v} > R_t$ then
19             remove $v$ from $X$ ;
20         end
21     end
22     $\mathbb{M}_{cnt} \leftarrow$ current trained submodel;
23     $cnt \leftarrow cnt + 1$
24 end
```

We design a probability-based method to further reduce the risk of generating false positives (line 18-20). Using a trained submodel to validate nodes of X, we receive a $|X| * N_n$ probabilistic matric M where:

$$M_{ij} = P(\mathcal{L}(X_i) = j) \tag{2}$$

The technical intuition of the probability-based method is to calculate the model's "confidence" of predicting a node type. For every node $X_i \in X$, we mark C_i and $\hat{C}_i$ as the index of the biggest element and second biggest element in M_i. The naive method considers that node X_i has been correctly classified when $C_i = \mathcal{L}(X_i)$. However, consider this situation: M_{iC_i} is not significantly bigger than $M_{i\hat{C}_i}$, which means that this submodel does not have a high degree of confidence in correctly classifying the node X_i. For example, consider that the model has $(0.5, 0.4, 0.05, 0.05)$ probabilities to classify A into four different classes. For another node B, the model has $(0.5, 0.2, 0.15, 0.15)$ probabilities to classify it into these four classes. Although the model has the same probability to classify A and B into the first class, the degree of confidence of classifying B is higher than A. We set a threshold R and consider X_i is correctly classified by this submodel when:

$$C_i = \mathcal{L}(X_i) \text{ and } \frac{M_{iC_i}}{M_{i\hat{C}_i}} > R_t \quad (3)$$

During the training phase, THREATRACE produces more submodels with a higher R, which thus results in fewer false positives during executing. By designing the multi-model framework and the probability-based method, THREATRACE achieves a lower false positive rate (**C2**). The evaluation is presented in §VI-D.

Note that because we split the whole training graph into several subgraphs, we need to carry out Alg. 1 several times, which may result in a large number of submodels. Therefore, we apply a heuristic method (line 1 of Alg. 1). We validate the subgraph G in the existing submodels and remove the correctly classified $active$ nodes from V. It is based on the insight that the multi-model framework has already learned the representation of those removed nodes. As we introduce before, THREATRACE trains several submodels to learn the hidden distribution of different benign nodes. For those nodes that have been learned by previous submodels, we do not need to repeatedly train them again. By applying the pre-filtering strategy, we reduce the number of initialized $active$ nodes in X, which further reduces the number of submodels. More macroscopically, we apply a similar heuristic method for graphs in the training set. We use a graph for training only when it has nodes which the current multi-model framework cannot correctly classify. It is an incremental work, which prevents producing "excess" submodels. The intuition behind the heuristics is the same as the method in line 1 of Alg. 1. For graphs which have been completely learned by current submodels, it is unnecessary to train them again.

By designing the multi-model framework and the probability-based method, THREATRACE achieves a lower false positive rate (**C2**). The evaluation is presented in §VI-D.

Executing Method. In the executing phase, we maintain a subgraph $\hat{G}$ in memory. $\hat{G}$ is empty originally and keeps appending with newly-arrived nodes and edges. The incoming edges' destination nodes and their 2-hop descendants are defined as $active$ nodes. The descendants and $related$ nodes are obtained from the whole graph in disk. We set a parameter called $Subgraph\ Size$ (SS). When the number of newly-arrived edges reaches SS, we use $\hat{G}$ for detection and start constructing a new subgraph in memory. Note that SS is not the number of edges in $\hat{G}$ because we additionally add some edges from the graph in disk. Through the streaming executing mode, we can dynamically detect the currently active entities of the system and guarantee long-term scalability.

We design a multi-model framework and a probability-based training method to reduce false positives. However, there is another challenge of false negatives (**C3**). To tackle this challenge, we also propose a probability-based method in the executing phase. For an abnormal node v, which is under detected, we detect it iteratively in submodels and get the probability list M_v of each submodel. We mark C_v and $\hat{C}_v$ as the class with the biggest and second biggest probability node v to be. Because v is an abnormal node with different behavior from benign nodes, it may not be classified to any class with high probability in every submodel. However, there exists a risk that it may still be correctly classified to its dominant label $\mathcal{L}(v)$ in one of the submodels, which is not desirable as we do not want to correctly classify the abnormal nodes. Thus, during the executing phase, we also use the threshold R and consider node v is correctly classified only when:

$$C_v = \mathcal{L}(v) \text{ and } \frac{M_{vC_v}}{M_{v\hat{C}_v}} > R_t \quad (4)$$

It means that a submodel classifies node v to the correct class with high assurance when equation 4 is satisfied. If v does not satisfy equation 4 in every submodel, we consider that it is an abnormal node. For anomalous nodes during executing, a higher R means the detector has a stricter standard of considering them as benign, which reduces false negatives (**C3**). For benign nodes, because more submodels for classifying benign nodes are trained with a higher R, the increase of false positives is less than the decrease of false negatives. Note that R should not be set as high as possible because the model is difficult to converge when R is too high. We further discuss the selection of R detailedly in §VI-D. Besides, although we detect nodes in $\hat{V}$ iteratively in every submodel, the size of $\hat{V}$ is reducing. Therefore, the number of submodels does not have a significant impact on processing speed. We evaluate the processing speed of THREATRACE in §VI-E.

The final prediction of the multi-model framework. A node is detected as benign if it is correctly classified in at least one submodel. Otherwise, it will be detected as anomalous.

D. Alert and Trace

This component receives abnormal nodes from the previous component and stores them in a queue Q. We set a time threshold T and maintain Q dynamically instead of directly raising alerts to the monitored system. The reason is that THREATRACE detects abnormal nodes in a streaming mode to detect threats in an early stage while learning nodes' roles by the whole graph in the training stage in order to learn the rich contextual information. Therefore, a benign node may be detected as abnormal before it reaches its final stage. We first store abnormal nodes in Q and set a waiting time threshold T for it to reach its final stage. If it has not changed to benign within time T, we pop it from Q and consider it as an abnormal node. A node may be detected several times under the streaming mode. Therefore, an anomalous node may be detected as benign at first and when it starts performing the malicious activity, it will be detected again as an $active$ node. When the number of abnormal nodes exceeds a tolerant threshold $\hat{T}$, which denotes the maximum number of false positives that may be generated, THREATRACE raises an alert for the system and traces anomaly in the 2-hop ancestors and descendants of abnormal nodes. We discuss the selection of T and $\hat{T}$ in §VI-D.

VI. EVALUATION

We evaluate THREATRACE in five public datasets, which are frequently used by state-of-the-art host-based threats detectors. We focus on the following aspects:

Q1. The threat detection performance compared to state-of-the-art detectors. (§VI-A, VI-B, VI-C)

Q2. The influence of parameters. (§VI-D)
Q3. The ability to trace the abnormal behavior. (§VI-C)
Q4. The ability to detect abnormal behavior in the early stage of an intrusion. (§VI-C)
Q5. The runtime and system resource overhead. (§VI-E)
Q6. The robustness against adaptive attacks. (§VI-F)
Q7. The influence of hop number K. (§VI-G)

Here we introduce the dataset, experimental setup, and implementation of comparison work. We compare THREATRACE to three anomaly-based host threats detectors (Unicorn, ProvDetector, and StreamSpot) and four anomalous log detectors (Log2vec, DeepLog, LogRobust and LogGAN). We use five public datasets: StreamSpot, Unicorn SC-1 and SC-2, DARPA TC #3 and #5 datasets. The overview of datasets is shown in TABLE II, III, and IV. We count the quantity of edges based on the quantity of logs. The quantity of nodes is obtained by counting the number of non-duplicate entities in all logs. In StreamSpot and Unicorn datasets, the normal and abnormal graphs are originally divided. We cannot divide nodes because the groundtruth of nodes is not provided. For DARPA TC datasets, we label the abnormal data by the name of attack events in the groundtruth files. For StreamSpot, Unicorn, and LogGAN, we directly use the open-source projects [34], [35], [36]. The results of StreamSpot and Unicorn are almost the same as the original papers. ProvDetector is not open source. Therefore, we reimplement it based on the original paper. We cannot compare the performance with ProvDetector's original paper because we do not have the private dataset of the paper for evaluation. For DeepLog and LogRobust, we use github projects implemented by other researchers [37], [38]. For Log2vec, we obtain the project from the authors. We cannot compare the performance with the log-level detectors' original papers [16], [21], [22], [23] because their datasets cannot be transformed to provenance format. We do not compare THREATRACE to other state-of-the-art misuse-based detectors because they require a priori expert knowledge to construct a model [2], [3], [9], [30]. These approaches are introduced in §II. Note that THREATRACE is a node-level detector while the comparison methods detect anomalies at varying granularity such as graph-level, path-level, and log-level. We cannot find other state-of-the-art anomaly-based methods that detect anomalies in provenance graphs at node level. Although the comparison methods' detection granularities are different from THREATRACE, the detection target of these approaches are similar. Therefore, we choose them as comparison methods and try our best to fairly compare their performance. The comparison principle is introduced as follows.

For graph-level experiments (§VI-A, VI-B), THREATRACE raises an alert for a graph when the quantity of the anomalous nodes exceeds the tolerant threshold $\hat{T}$. StreamSpot and Unicorn can directly raise an alert for a graph as they are graph-level detectors. As a path-level detector, provDetector raises an alert when there are more than k_p anomalous paths in a graph. We enumerate the parameter k_p and set k_p as 2 to make ProvDetector achieve the better results. We do not compare with log-level detectors because the datasets of graph-level experiments do not have log data.

For node-level experiments (§VI-C), we do not compare with StreamSpot, Unicorn and provDetector because their detection granularity is coarser than THREATRACE. THREATRACE raises an alert for a node when it is detected as anomalous while the log-level detectors raise an alert for a node when a log associated with this node is detected as anomalous. For both graph-level and node-level experiments, we calculate the precision, recall, accuracy, F-score and false positives rate (FPR) based on the comparison of detection results and the ground truth. For detection and runtime experiments, we tune the hyperparameters of the comparison methods and choose the best results.

We show that THREATRACE can achieve better detection performance (**Q1**) in these datasets. In §VI-D, we evaluate how the specially designed parameters influence the threat detection performance (**Q2**). In §VI-C, we use DARPA TC datasets to further evaluate THREATRACE's capability to trace anomaly (**Q3**) and detect persistent intrusions in the early stage (**Q4**). We evaluate THREATRACE's runtime performances in terms of executing speed and system resource overhead (**Q5**) in §VI-E. We evaluate the robustness against adaptive attacks (**Q6**) in §VI-F. In §VI-G, we evaluate how the hop number K influences THREATRACE's detection and runtime performance (**Q7**). TABLE V, VI, IV show an overview of the datasets.

The parameters of THREATRACE in §VI-A, VI-B, VI-C, VI-F are shown in TABLE I. We set these parameters based on the detection and runtime performance in experiments of each subsection. Each submodel has the same GraphSAGE hyperparameters. Specifically, we set most of them as default values. We set the number of hidden layer's neurons as 32, which is approximately half of the features' number in Unicorn SC-2 dataset. The submodels have one hidden layer because

TABLE I: Parameters of experiments.

Parameter	Description	Section	Values
BS	Batch mode parameter of GraphSAGE [11]	V-C	$5,000$
SS	Maximu number of *active* nodes	V-C	$200,000$
R	Probability rate threshold	V-C	1.5
T	Waiting time threshold	V-D	168
$\hat{T}$	Tolerant threshold	V-D	2

TABLE II: Overview of StreamSpot dataset.

Scene	# of graph	Average # of nodes	Average # of edges
Benign	500	8,315	173,857
Attack	100	8,891	28,423

TABLE III: Overview of Unicorn SC datasets.

dataset	Scene	# of graph	Average # of nodes	Average # of edges
SC-1	Benign	125	265,424	975,226
	Attack	25	257,156	957,968
SC-2	Benign	125	238,338	911,153
	Attack	25	243,658	949,887

TABLE IV: Overview of DARPA TC #3 and #5 datasets

Dataset	Scene	# of benign nodes	# of abnormal nodes	# of edges
#3	THEIA	3,505,326	25,362	102,929,710
	Trace	2,416,007	67,383	6,978,024
	CADETS	706,966	12,852	8,663,569
	fivedirections	569,848	425	9,852,465
#5	THEIA	8,384,495	162,714	126,824,068
	Trace	1,825,351	29,312	3,165,213
	CADETS	482,326	20,524	11,248,647
	fivedirections	921,201	1,204	6,274,387

each node aggregates information from its 2-hop neighbors. We test the influence of parameters in §VI-D.

A. StreamSpot Dataset

We compare THREATRACE to two state-of-the-art detectors: StreamSpot and Unicorn.

Dataset. The StreamSpot dataset (TABLE II) is StreamSpot's own dataset which is publicly available [39]. It contains 6*100 information flow graphs derived from five benign scenes and one attack scene. Each scene runs 100 times to generate 100 graphs using the Linux SystemTap logging system [40]. The benign scenes involve different benign activities: checking Gmail, browsing CNN.com, downloading files, watching YouTube and playing a video game. The attack scene involves a drive-by download attack. The victim host visits a malicious URL, which exploits a Flash vulnerability and gets the root access to the victim. We use the same validation strategy as Unicorn to randomly split the dataset into a training set with 75*5 benign graphs and a testing set with 25*5 benign graphs and 25 attack graphs. We do not compare with ProvDetector because edges in StreamSpot dataset do not have the timestamp attribute, which is necessary for ProvDetector. We repeat this procedure and report the mean evaluation results.

Result. The results are shown in TABLE V. We analyze two factors of the results.

(1) **Why both THREATRACE and Unicorn achieve great performance in this dataset?** StreamSpot dataset is relatively weaker than other datasets, which can be reflected in two aspects. Firstly, StreamSpot dataset's average node quantity is only 3.5% of Unicorn SC dataset's and 0.9% of DARPA TC dataset's. Therefore, the abnormal patterns in StreamSpot dataset are easier to find because there are less benign background nodes. Secondly, anomalous graphs are significantly different from benign graphs. The most obvious evidence is also the size of graphs. The average edge quantity in benign graph is 173,857 while in anomalous graph the number is only 28,423. For comparison, the average edge quantities in benign graph and anomalous graph of Unicorn SC dataset are 911,153 and 949,887, which do not have significant difference.

(2) **Why Unicorn cannot achieve a 100% precision and recall?** We analyze deep into this dataset and find that the 100 graphs of the attack scene can be generally split into two classes with 95 and 5 graphs. The 5 graphs in the second group are significantly different from the other 95 graphs. They are more similar to the graphs of the Youtube scene, which means that threats in the second group are more stealthy. Drive-by download attack happened when the victim browses a website, which is similar to the youtube scene. It is hard for a graph-kernel-based method to detect those 5 graphs.

TABLE V: Comparison to StreamSpot and Unicorn on the StreamSpot dataset.

Detector	Precision	Recall	Accuracy	F-Score	TP	TN	FP	FN	FPR
StreamSpot	0.73	0.91	0.93	0.81	22.8	116.7	8.3	2.2	0.066
Unicorn	0.95	0.97	0.99	0.96	24.32	123.64	1.36	0.68	0.011
THREATRACE	0.98	0.99	0.99	0.99	24.8	124.5	0.5	0.2	0.004

B. Unicorn SC Datasets

The Unicorn SC datasets consist of two datasets: SC-1 and SC-2 [41]. They are Unicorn's own dataset [41], which are more complex than the StreamSpot dataset. We use them to compare THREATRACE's detection performance with Unicorn and ProvDetector. We cannot use StreamSpot for comparison because it cannot deal with a large number of edges [7].

Dataset. The Unicorn SC datasets (TABLE III) are generated in a controlled lab environment following the typical cyber kill chain model. Each graph, which is captured by CamFlow (v0.5.0), contains the whole-system provenance of a host running for three days. There are background benign activities in both benign and attack graphs. We follow the same 5-fold cross-validation as Unicorn: we use 4 groups (each group contains 25 graphs) of benign graphs to train, and the 5th group of benign graphs and 25 attack graphs for validation. We apply a streaming mode to replay the validation graphs and detect them dynamically. We repeat this procedure and report the mean evaluation results.

Result. As shown in TABLE VI and VII, THREATRACE achieves better performance than Unicorn using its own datasets. The reason that Unicorn has low performance is introduced in Unicorn's paper [7]: the attacker has prior knowledge about the system and thus acts more stealthily than attackers of other datasets. We also find that THREATRACE raises alerts for less than 10 nodes in most attack graphs, which demonstrates the stealthy characteristic of this dataset. The comparison results in this dataset further demonstrate our motivation that the graph-kernel-based method is hard to detect stealthy threats. Besides, THREATRACE also raises fewer false positives than Unicorn and ProvDetector, which is important to alleviate the "threat fatigue problem". ProvDetector does not perform well in this dataset. This could be explained by the detection granularity. ProvDetector is a path-level detector, which is capable of hunting anomalous paths in a provenance graph. However, Unicorn SC datasets are generated following the typical cyber kill chain model, which divides the attack activities into several steps. For example, the attackers in Unicorn SC-1 exploit a vulnerability (CVE-2016-4971) to install a remote access trojan into the victim from an FTP server. As the trojan is installed, the remote access trojan establishes a command and control (C&C) channel with another host of the attackers. In this scene, the anomalous entities cannot be aggragated into a single path as the attackers have at least two remote hosts to conduct the intrusion. Therefore, anomalous entities in each single path may not be anomalous enough for raising an alert. Unfortunately, Unicorn SC datasets do

TABLE VI: Comparison to Unicorn and ProvDetector on the Unicorn SC-1 dataset.

Detector	Precision	Recall	Accuracy	F-Score	TP	TN	FP	FN	FPR
Unicorn	0.86	0.95	0.90	0.90	23.75	21.13	3.87	1.25	0.155
ProvDetector	0.71	0.65	0.69	0.68	16.2	18.5	6.5	8.8	0.260
THREATRACE	0.93	0.98	0.95	0.95	24.5	23.15	1.85	0.5	0.074

TABLE VII: Comparison to Unicorn and ProvDetector on the Unicorn SC-2 dataset.

Detector	Precision	Recall	Accuracy	F-Score	TP	TN	FP	FN	FPR
Unicorn	0.75	0.80	0.77	0.78	20	18.33	6.67	5	0.267
ProvDetector	0.67	0.6	0.65	0.63	15	17.5	7.5	10	0.300
THREATRACE	0.91	0.96	0.93	0.93	24	22.5	2.5	1	0.100

not provide the ground truth of nodes. We cannot analyse deeply why ProvDetector does not perform well. In DARPA TC datasets (§VI-C), we analyze the results deeply at node level and path level to study the performance of ProvDetector.

C. DARPA TC Datasets

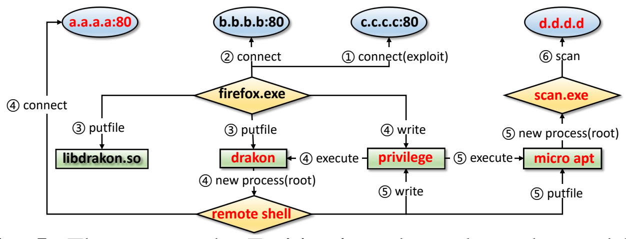

Fig. 5: The case study. Entities in red are those detected by THREATRACE.

We use DARPA TC datasets to evaluate THREATRACE's ability to detect and trace long-term running intrusions. Datasets in the last two subsections do not have the ground truth of nodes and thus, we cannot use them to evaluate THREATRACE's ability of anomaly tracing.

Dataset. DARPA TC datasets (TABLE IV) are generated in the red-team vs. blue-team engagement of the DARPA Transparent Computing program. Datasets of the third and fifth engagements are publicly available [31]. Therefore, we use DARPA TC #3 and #5 datasets for evaluation in this section.

The third engagement lasted for two weeks and the fifth engagement lasted for 9 days. During the engagements, the red team performed attacks and benign background activity to hosts with different systems. Three kinds of attackers are involved in the red-team including Nation State, Common Threat and Metasploit. The Nation State attackers' goal is to steal proprietary and personal information from the targeted company. They use the Nginx backdoor, the Firefox backdoor, the browser extension, Drakon APT and micro APT to accomplish the goal. The Common Threat attackers' goal is to steal personal identifiable information for financial gain by deceiving the targeted users into providing access to the target network. They accomplish the goal using phishing e-mails, powershell scripts, a malicious Excel spreadsheet macro, a Pine backdoor and a malware executable. The provenance collecting tools captured provenance data of the whole system from start to end. The provenance data and ground truth are publicly available [31]. Therefore, we use the ground truth to label the abnormal nodes and evaluate THREATRACE's performance of detecting them, which shows the ability of tracking anomaly. The original dataset of each scene contains several files. We choose files that contain anomalous behavior as testing set. We remove part of the dataset because some graphs are generated in exceptional accidents such as outages and shutdown of hosts. Files that have no threats are used to train the models and files with threats are used for evaluation. Because we check a node's 2-hop ancestors and descendants to track its position (§V-D) when it is detected as an anomaly, we define metrics as follows:

- **True Positive.** The anomalous nodes which are detected as abnormal or the anomalous nodes one of whose 2-hop ancestors and descendants has been detected as abnormal. Those nodes are defined as true positives because we can hunt them during alert tracing.
- **False Positive.** The benign nodes which are detected as abnormal and do not have anomalous nodes in their 2-hop ancestors and descendants. Those nodes are defined as false positives because we cannot hunt any anomalous nodes during alert tracing of them.
- **True Negative.** Other benign nodes.
- **False Negative.** Other anomalous nodes.

We assume the correctness of the manual inspection of nodes' 2-hop ancestors and descendants. The workforce cost of tracing anomaly is acceptable since the average number of nodes' 2-hop ancestors and descendants is 1.8 in this dataset. Because a node in a provenance graph denotes an entity in a system, it is not difficult to inspect its behavior manually.

Result. The results in TABLE VIII show THREATRACE and comparison methods's performance of detecting abnormal nodes, which means the ability of anomaly tracing. The results of THREATRACE are acceptable considering that the dataset is highly unbalanced. We use the motivation example (Fig. 2) as the case study to further describe THREATRACE's anomaly tracing ability. It is simplified as an attack graph in Fig. 5. The attack lasts for two days and the victim is a `Ubuntu 12.04 x64` host. The attacker exploits `Firefox 54.0.1` backdoor using a malicious website `c.c.c.c:80`. When the victim connects to this website (①), a drakon implant begins running in memory and results in a connection to the attacker operator console `b.b.b.b:80` (②). The drakon implant is not shown in the figure because there is not enough information about it in the ground truth file. The attacker further writes `libdarkon.so` and `drakon` executable binary to the victim host's disk (③). Then, the attacker uses a `privilege` escalated execution capability to execute the `drakon`, which

TABLE VIII: Results of node level detection experiments in DARPA TC #3 and #5 datasets.

Dataset/ Scene	Detector	Precision	Recall	F-Score	FPR	TP	TN	FP	FN
#3/ THEIA	THREATRACE	0.87	0.99	0.93	0.001	25,297	3,501,561	3,765	65
	Log2vec	0.62	0.66	0.64	0.003	16,752	3,495,271	10,055	8,610
	DeepLog	0.16	0.14	0.15	0.005	3,458	3,486,805	18,521	21,904
	LogRobust	0.42	0.35	0.39	0.003	8,952	3,493,184	12,142	16,410
	LogGAN	0.36	0.31	0.33	0.004	7,928	3,491,125	14,201	17,434
#3/ Trace	THREATRACE	0.72	0.99	0.83	0.011	67,382	2,389,233	26,774	1
	Log2vec	0.54	0.78	0.64	0.018	52,741	2,371,785	44,222	14,642
	DeepLog	0.41	0.68	0.51	0.027	45,989	2,349,733	66,274	21,394
	LogRobust	0.51	0.75	0.61	0.020	50,559	2,367,713	48,294	16,824
	LogGAN	0.50	0.73	0.59	0.021	49,133	2,365,885	50,122	18,250
#3/ CADETS	THREATRACE	0.90	0.99	0.95	0.002	12,848	705,605	1,361	4
	Log2vec	0.49	0.85	0.62	0.016	10,871	695,741	11,225	1,981
	DeepLog	0.23	0.74	0.35	0.044	9,482	675,842	31,124	3,370
	LogRobust	0.41	0.83	0.55	0.022	10,727	691,692	15,274	2,125
	LogGAN	0.35	0.80	0.49	0.027	10,263	688,104	18,862	2,589
#3/ fivedirections	THREATRACE	0.67	0.92	0.78	0.001	389	569,660	188	36
	Log2vec	0.22	0.76	0.34	0.002	324	568,712	1,136	101
	DeepLog	0.03	0.47	0.05	0.012	198	562,817	7,031	227
	LogRobust	0.12	0.64	0.21	0.003	270	567,923	1,925	155
	LogGAN	0.10	0.71	0.17	0.005	301	567,094	2,754	124
#5/ THEIA	THREATRACE	0.70	0.92	0.80	0.008	150,286	8,321,358	63,137	12,428
	Log2vec	0.57	0.88	0.69	0.013	143,871	8,276,443	108,052	18,843
	DeepLog	0.33	0.54	0.41	0.021	88,621	8,204,391	180,104	74,093
	LogRobust	0.56	0.88	0.68	0.013	142,643	8,272,753	111,742	20,071
	LogGAN	0.54	0.81	0.65	0.013	131,852	8,273,570	110,925	30,862
#5/ Trace	THREATRACE	0.81	0.85	0.83	0.003	25,024	1,819,337	6,014	4,288
	Log2vec	0.50	0.58	0.53	0.009	16,938	1,808,126	17,225	12,374
	DeepLog	0.31	0.38	0.34	0.014	11,274	1,800,134	25,217	18,038
	LogRobust	0.40	0.47	0.43	0.011	13,788	1,804,452	20,899	15,524
	LogGAN	0.33	0.39	0.36	0.013	11,458	1,802,030	23,321	17,854
#5/ CADETS	THREATRACE	0.63	0.86	0.73	0.021	17,685	472,045	10,281	2,839
	Log2vec	0.48	0.73	0.58	0.033	14,996	466,401	15,925	5,528
	DeepLog	0.19	0.21	0.20	0.038	4,250	464,079	18,247	16,274
	LogRobust	0.47	0.73	0.57	0.034	14,904	465,688	16,638	5,620
	LogGAN	0.42	0.59	0.49	0.035	12,050	465,344	16,982	8,474
#5/ fivedirections	THREATRACE	0.64	0.75	0.69	0.001	908	920,700	501	296
	Log2vec	0.28	0.63	0.38	0.002	756	919,219	1,982	448
	DeepLog	0.07	0.29	0.11	0.005	350	916,349	4,852	854
	LogRobust	0.21	0.58	0.31	0.003	696	918,624	2,577	508
	LogGAN	0.16	0.43	0.23	0.003	512	918,519	2,682	692

results in a process `remote shell` with root privilege (④). The new `remote shell` process connects to a new operator console `a.a.a.a:80` with root access (④) and the attacker stops their campaign temporarily. Two days later, the attacker writes a `micro apt` to disk and elevates it as a new process `scan.exe` with root privileges (⑤). Finally, the attacker carries out a port scan of `d.d.d.d` (⑥).

In this case, THREATRACE raises alerts for the `darkon` executable binary, `remote shell`, `privilege` escalated execution, malicious ip `a.a.a.a:80`, `micro apt` and its process `scan.exe`, the scanned ip object `d.d.d.d`. THREATRACE achieves this result in the situation that the anomalous nodes only take a small part (less than 1%) in the whole provenance graph due to the deliberate prolonging of the attack duration, which shows THREATRACE's ability to detect and trace anomaly in a long-term running system. Besides, THREATRACE successfully detects some early steps of the attack such as `darkon` and `privilege` which shows THREATRACE's ability to detect and prevent intrusion in the early phases. We also note that THREATRACE fails to detect some components of the attack like `libdarkon.so` and malicious ip addresses `b.b.b.b:80` and `c.c.c.c:80`. We find that these components do not show significant influence during the attack period: they act more like auxiliary roles. This partially explains THREATRACE's failure on detecting those attack parts. Fortunately, there is usually no need to detect every part of an intrusion campaign. In this case, THREATRACE detects some important components such as the `darkon` executable binary, which is enough to detect and stop the intrusion in its early phase.

We search deep into the dataset to find the reason for the unsatisfactory detection results of log-level detectors. We discover that the logs of anomalous entities in DARPA TC datasets are not significantly different from benign logs. For example, an associated log of `micro apt` in Fig. 5 is "`the remote shell put a file named micro apt into the host`", which is a benign file downloading log in a host. The anomalous behavior of `micro apt` is hidden in the neighbor information among the graph. DeepLog, LogRobust, and LogGAN do not detect anomalies in a graph. Therefore, it is hard for them to catch the anomalous contextual information among a graph. Log2vec detects anomalous logs in a graph using nodes embedding and clusting methods. It aggregates information from a node's neighbor and thus obtains better results than other log-level detectors. However, Log2vec categorizes the logs of `micro apt` to the benign clusters. A possible explanation is that the node of Log2vec is a log instead of a system entity. Therefore, the information of an entity is scattered in a graph because logs of the entity are displayed as several nodes in Log2vec's graph and thus results in inadequate information aggregation. That is also why provenance graph is proposed to be a better data source by recent studies [2], [3], [4], [5], [6], [7], [8], [9].

D. Influence of Different Parameters

THREATRACE shows an improvement in detection performance compared to the state-of-the-art detector in the Unicorn SC-2 dataset. We further use the Unicorn SC-2 dataset to study how the parameters, which are specially designed for host-based threat detection, influence the detection performance. The parameters we test are SS, BS, R, T, $\hat{T}$. The values in Tabel I make up the baseline, which gains encouraging detection and runtime performance in Unicorn SC-2 dataset. When we test one of them, we keep the others the same as the baseline.

- $Subgraph\ Size$ (SS). We set SS for the Data Storage module (§V-B). As shown in Fig. 6(a), it has a weak impact on detection performance. THREATRACE detects the provenance graph in a streaming mode. The subgraph of current system execution is updated in memory and transferred to the detection module when its size reaches SS. Thus, SS affects the arrival interval of subgraphs, which may influence the time for benign nodes to reach their final stage. Besides, the randomly divided dataset may have an impact on the results. Compared to detection performance, SS has a much greater impact on runtime performance (§VI-E).
- $Batch\ Size$ (BS). BS is the parameter we set for the Model module (§V-C). As shown in Fig. 6(b), BS has almost no influence on detection performance, which is probably caused by the randomly splitted dataset. We evaluate its influence on runtime performance in §VI-E.
- R. This is the rate threshold we set for the training phase. As shown in Fig. 6(c), the detection performance improves when R increases from 1.0 to 1.5. R has no significant influence on detection performance from 1.5 to 2.0. However, when it exceeds 2.0, the performance tends to decrease.
- T and $\hat{T}$. These two parameters influence the Alert and Trace module (§V-D). The results are shown in Fig. 6(d)(e), which can be explained by the function of T and $\hat{T}$. T is the waiting time threshold we set for benign nodes to reach their final stage. In real-time streaming detection mode, a benign node may be detected as an anomaly before it reaches its final stage. Thus, a bigger T means we "give more evolutionary time" to the benign nodes and results in higher $Precision$, lower $Recall$ and FPR. Note that although it decreases the number of false positives, it also decreases THREATRACE's ability to detect intrusion as early as possible: it needs more time to wait for an anomalous node before raising alert for it. The average time of raising an alert for an anomalous graph is 434.45 seconds when T = 150, while it turns to 874.32 seconds when T is increased to 600. $\hat{T}$ is the tolerant threshold for THREATRACE to raise alerts for the monitored system. Bigger $\hat{T}$ means less false positives and more false negatives, which thus results in higher $Precision$, lower $Recall$ and FPR.

The evaluation of different parameters shows how THREATRACE's special design for host-based threat detection influences the detection performance. The results of each parameter can be explained by the design principle. We can leverage the function of different parameters to optimize THREATRACE's performance, which is discussed in §VII.

E. Runtime Overhead

We study the processing speed, CPU utilization, and memory usage during THREATRACE's execution in Unicorn SC-

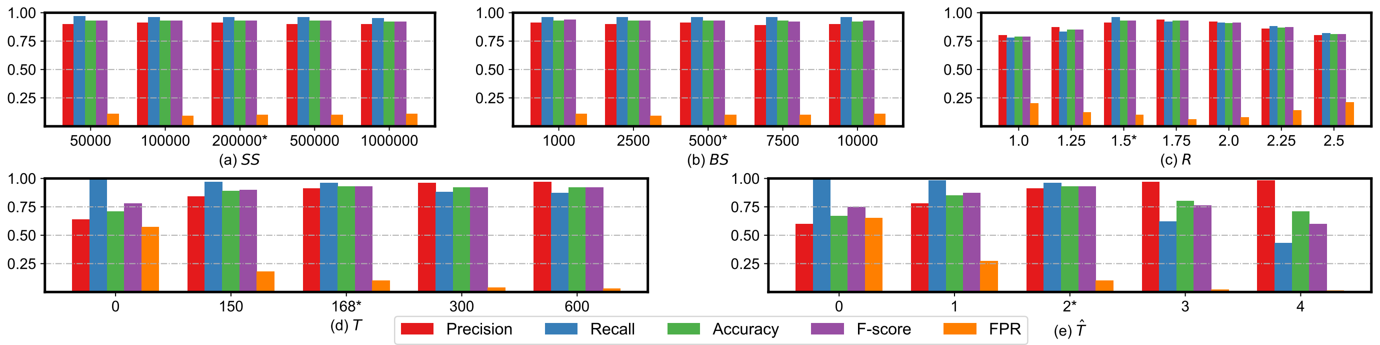

Fig. 6: Detection performance with varying parameters.

2 dataset. This section focuses on THREATRACE's runtime performance in the execution phase. The experiments in this subsection are done in an Ubuntu 16.04.6 LTS machine with 16 vCPUs and 64GiB of memory. We reimplement Unicorn and ProvDetector and evaluate them in the same machine. We repeat the experiments and report the average results.

Processing speed. We evaluate the processing speed of the entire detector, which is composed of four components. Fig. 7(a)(b) shows the processing speed of THREATRACE in different parameters. We test THREATRACE's processing speed in different *Subgraph Size* (SS) and *Batch Size* (BS). The red line is the maximum edges generated speed (M) in the three datasets we previously discussed. The green and blue lines are the maximum processing speed of Unicorn and ProvDetector. The results show that even in the worse case, THREATRACE's processing speed is still faster than M. In the best case, THREATRACE's is about five and three times faster than Unicorn and ProvDetector. We test one parameter and set the others as the baseline in TABLE I.

- *Batch Size* (BS). BS is the parameter we set for the Model module (§V-C). As shown in Fig. 7(a), with the increase of BS, THREATRACE's processing speed has an increasing trend and tends to be flat.
- *Subgraph Size* (SS). SS is the parameter we set for the Data Storage module (§V-B). As shown in Fig. 7(b), runtime performance improves as we increase SS.

CPU utilization & Memory usage. Fig. 7(c)(d)(e)(f) demonstrate THREATRACE's CPU utilization and memory usage with different SS and BS. The green and blue lines are the baseline results of Unicorn and ProvDetector. The results show that with the increase of BS, CPU utilization tends to be bigger and reaches its maximum value when $BS = 10$. After that, it becomes smaller. In terms of SS, CPU utilization has a decreasing trend when SS increases. There is no significant impact of varying BS and SS on memory usage. THREATRACE's CPU utilization and memory usage are bigger than Unicorn and ProvDetector. This is primarily because THREATRACE is a deep-learning-based method which needs more computation resources than traditional methods. It is a limitation of THREATRACE and we plan to research the problem of decreasing computation resources in future work.

We do not present THREATRACE's runtime overhead with other parameters because they only influence detection performance. At the same time, BS and SS have little influence on detection performance. These independent and insensitive characteristics provide the convenience of tuning THREATRACE.

In order to study the impact of SS on memory usage more adequately, we explore the change of node quantity in memory over time with two different SS. The experiments are conducted on Unicorn SC-2 dataset. The results are shown in Fig. 8. From the results we can discover that the node quantity in memory is basically fixed. The number is bigger than SS because SS controls the quantity of *active* nodes while there are *related* nodes in memory as well.

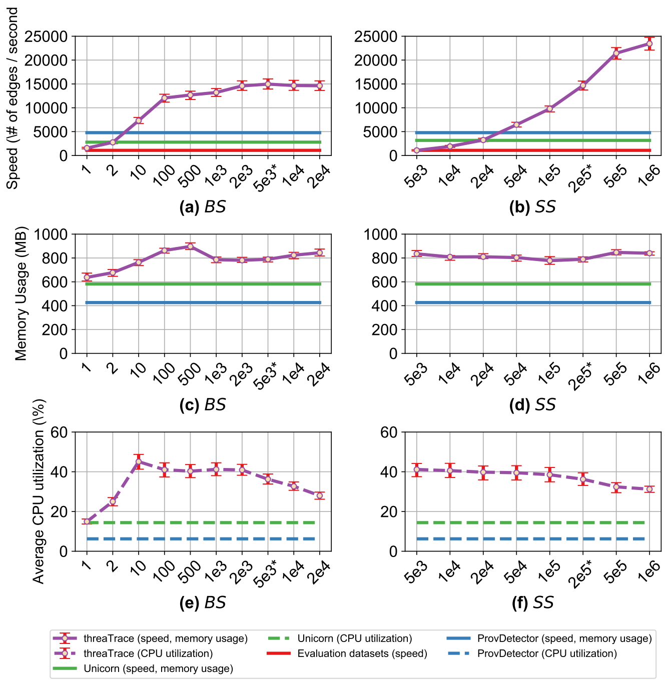

Fig. 7: Runtime performance with varying parameters. The evaluation of each parameter is done with the remaining parameters constant.

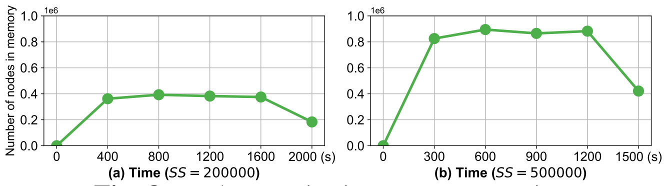

Fig. 8: Node quantity in memory over time

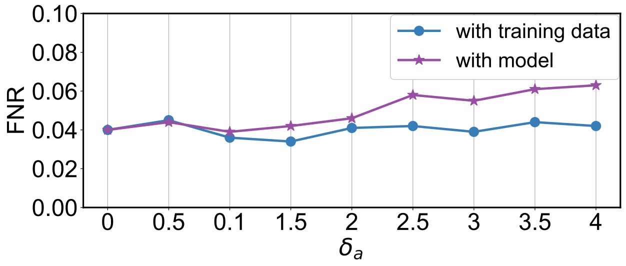

Fig. 9: Results of the evasion attack experiments.

F. Adaptive attacks

Attack Methodology. THREATRACE is a Graph-based detection system. Therefore, we borrow the idea of existing adversarial attacks against Graph-based detectors [42], [43], [44] to develop adaptive attacks on THREATRACE, called optimization-based evasion attacks. The purpose of this attack is to make abnormal nodes evade the detection of THREATRACE in the execution phase. Specifically, because THREATRACE judges the node as abnormal when it is misclassified, the purpose of the attacker is to make the abnormal node be classified into the correct class to avoid detection. The attacker needs to find a perturbation on the abnormal node's features, which is constructed with edges between the node and its neighbors. Note that we suppose the attacker has compromised the system by implanting some abnormal nodes (such as *Malware*, *Remote Shell*, anomalous *Dynamic Link Library*) into it. Therefore, the attacker can control the related edges of the abnormal nodes. The perturbation should be as small as possible to keep the original function of the abnormal node and reduce cost. In a word, suppose x is the feature of an abnormal node which can be detected by THREATRACE originally, the optimization-based evasion attacks' goal is to change the feature to $\hat{x}$ to evade detection with the constraint $\frac{\|\hat{x}-x\|_2}{\|x\|_2} < \delta_a$, where δ_a limits the perturbation. In order to construct a node's adversarial features $\hat{x}$, we assume that the attacker knows THREATRACE's feature extraction method mentioned in §V. For other adversarial's background knowledge of THREATRACE, we study two kinds of attackers based on the background knowledge.

(1) Attackers with training data. The attackers have the training data. The evasion attack with training data can be performed in two steps. The first step is to find a benign node x_b in the training data, which is most similar to the anomalous node x and has the same class as x. Formally,

$$argmin_{x_b} \|x_b - x\|_2 \quad s.t. \quad class(x_b) = class(x) \tag{5}$$

The second step is to solve the optimization problem:

$$argmin_{\hat{x}} \|\hat{x} - x_b\|_2 \quad s.t. \quad \frac{\|\hat{x}-x\|_2}{\|x\|_2} < \delta_a, \hat{x} \in \mathbb{N} \tag{6}$$

(2) Attackers with THREATRACE*'s model.* The attackers knows THREATRACE's model, including the parameters after training and hyperparameters. Therefore, the attackers can solve the optimization problem directly based on the loss of the model. Formally,

$$argmin_{\hat{x}}(loss(\hat{x})) \quad s.t. \quad \frac{\|\hat{x}-x\|_2}{\|x\|_2} < \delta_a, \hat{x} \in \mathbb{N} \tag{7}$$

$Loss(\hat{x})$ is the loss function of THREATRACE, indicating whether a sample is classified into the correct category. For an anomalous sample, the attacker needs to make THREATRACE correctly classify it to evade detection. Therefore, $loss(\hat{x})$ should be as small as better. This problem can be easily solved by a gradient-based method. THREATRACE consists of several submodels. For simplicity, we choose the first submodel for the target model to evade.

For these two kinds of attackers, once $\hat{x}$ is successfully solved, the attacker controls the abnormal node to interact with other nodes in the system according to the new distribution of edges, so that its feature will change to $\hat{x}$.

Robustness Evaluation and Analysis. We use Unicorn SC-2 dataset for evaluation. We first choose the abnormal nodes detected by THREATRACE as the base samples and then apply the optimization-based evasion approach to compute $\hat{x}$ for each abnormal node. After that, we change the related edges of the adversarial samples to change their features extracted. The results are shown in Fig. 9. For attackers with training data, the imitation of training data has almost no effect on detection performance. For attackers with THREATRACE's model, the evasion effection increases with the increasing of δ_a. However, compared to the original result, the FNR is acceptable (raise from 0.04 to 0.07), which demonstrates THREATRACE's good robustness against optimization-based evasion attacks.

G. Influence of hop number K

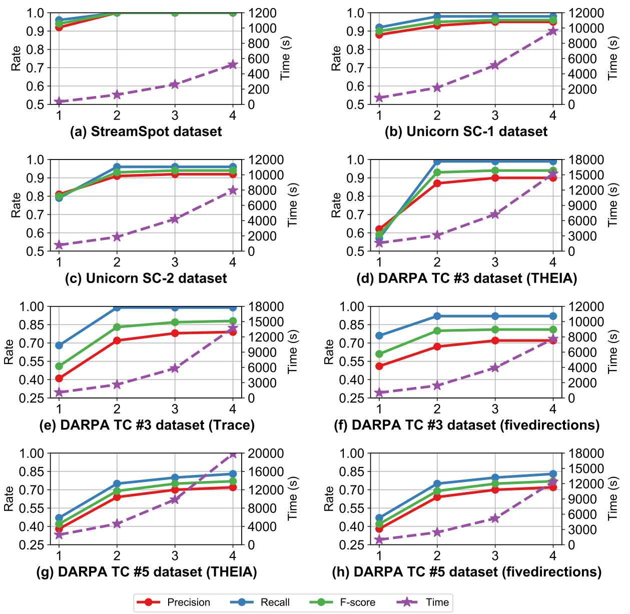

Fig. 10: Results of hop number experiments.

We evaluate the detection and runtime performance of THREATRACE with different hop number K to study how K affects the performance. We design experiments in five datasets: StreamSpot, Unicorn SC-1, Unicorn SC-2, DARPA TC #3 (THEIA, Trace, fivedirections), and DARPA TC #5 (THEIA, fivedirections). K is set from 1 to 4. The results are illustrated in Fig. 10. With the increase of K, the detection performance increases while the processing time also increases significantly. The results can be explained by the FP algorithm (§III-B) of GraphSAGE. A bigger hop number means the model aggregates more information from a node's neighbors, which is beneficial to learn node representation and promote the detection performance. However, aggregating more information also denotes that the model needs more time to execute the whole workflow, which thus results in more processing time. Another interesting discovery is that the detection performance has a significant improvement when K increases from 1 to 2 while the improvement is not significant when K increases from 2 to 4. This phenomenon indicates that a node's two hops neighbors provide the vast majority of information that the model aggregates. In this paper, we also set K as 2 in experiments of other subsections to obtain a good trade off between detection and runtime performance.

VII. Discussion & Limitation

We discuss some issues, limitations, and future work of THREATRACE in this section.

Closed-world assumption. Host-based threat detection is essentially a classification problem. It is based on a closed world assumption[45], [46], which assumes that every benign behavior has been involved in the training set and the rest are abnormal. However, in real-life problems, it is hard to guarantee that all benign cases are covered. As the same as other anomaly-based detectors, THREATRACE faces this problem too. We cannot guarantee that we learn every benign node in training phase. However, system administrators can update models periodically using updated benign data provenance of the system. It is an incremental work with no need to retrain the previous models. Therefore, it does not need much time (§V-C). As a use case, recently we deploy THREATRACE in a host of an Internet company. We update the models per week initially. With the increase of time, the update frequency reduces to one time each month because for a single user, the quantity of benign behaviors has an upper bound. Each time we update the models, we put the recent provenance data into the current models and obtain new benign data which cannot be correctly classified. Then we use the new benign data to train new submodels. The new submodels are added to the multi-model framework. The workflow takes approximately one hour. As an important problem, how to update deep learning models effectively is one of our future work.

Adversarial attacks. The robustness goal of this paper is to be robust against the evasion attack that we study in §VI-F. There are other adaptive attacks that may fail THREATRACE, such as poisoning attack [47] and graph backdoor [48]. Future work can study the robustness against more attacks. Some pioneering methods for enhancing GNN's robustness can be found in [49], [50]. They are not suitable for THREATRACE because they focus on robustness in graph convolutional networks, which are not inductive models (§III-B).

Evaluation. In host-based threat detection domain, public datasets are usually generated in a manually constructed environment [39], [41] and thus may not represent typical workloads in practice. Data generated from red-team vs. blue-team engagement [31] is partly close to the actual situation. It is important to generate a public dataset for the community which can reflect the typical workloads in the real situation and is in consideration of privacy and confidentiality. We plan to research this problem in future work.

System overhead. It is necessary to maintain a stable system overhead during deployment. We demonstrate that THREATRACE has a fast processing speed and acceptable system overhead in §VI-E. However, if the host's performance is relatively low in practice, we need to adjust BS and SS to balance detection timeliness and system overhead. In the worst case, we may have to perform offline detection.

Threat fatigue problem [32]. It is important to avoid "threat fatigue problem" by reducing false positives. Although we try to reduce false positives by multi-model framework and probability-based method, THREATRACE still raises many false positives in DARPA TC datasets. It may be related to the large number of benign nodes (Table IV). THREATRACE has limitations of controlling the number of false positives when there are plenty of nodes in the graph, and we seek means to alleviate this phenomenon. In practice, an enterprise or government usually has a list of reliable softwares. We suggest the administrator maintain a whitelist to record system entities that are related to those reliable softwares. Besides, if a false positive entity is inspected as benign and believed to be reliable in the future, it can also be added to the Whitelist. Whitelist is a typical method in recent work [3], [9] for reducing false positives. Apart from the whitelist, the administrator can use false positives to train more submodels and thus avoid raising the same false positives in the future.

Unsupervised. THREATRACE needs benign data for training. Therefore, THREATRACE is a semi-supervised approach. In practice, the administrator can collect provenance data when the system is running without threats (e.g., only trusted software and websites can be used and visited).

GNN models. We use GraphSAGE as our base model. Besides GraphSAGE, there are other inductive GNN structures which can be used for threat detection, such as Graph Attention Networks [51]. Due to time and space constraints, we plan to research other GNN structures in the future.

VIII. Conclusion

We present THREATRACE, a real-time host-based node level threat detection system that takes whole-system provenance graphs as input to detect and trace stealthy intrusion behavior. THREATRACE uses a multi-model framework to learn different roles of benign nodes in a system based on their features and the causality relationships between their neighbor nodes, and detect stealthy intrusion behavior based on detecting abnormal nodes while locating them at the same time. The evaluation results show that THREATRACE successfully detects and locates stealthy threats with high detection performance, processing speed, and acceptable system resource overhead.

References

[1] J. Navarro, A. Deruyver, and P. Parrend, "A systematic survey on multi-step attack detection," *Computers & Security*, vol. 76, pp. 214–249, 2018.

[2] S. M. Milajerdi, B. Eshete, R. Gjomemo, and V. Venkatakrishnan, "Poirot: Aligning attack behavior with kernel audit records for cyber threat hunting," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 1795–1812.

[3] S. M. Milajerdi, R. Gjomemo, B. Eshete, R. Sekar, and V. Venkatakrishnan, "Holmes: real-time apt detection through correlation of suspicious information flows," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 1137–1152.

[4] M. Barre, A. Gehani, and V. Yegneswaran, "Mining data provenance to detect advanced persistent threats," in *11th International Workshop on Theory and Practice of Provenance (TaPP 2019)*, 2019.

[5] G. Berrada and J. Cheney, "Aggregating unsupervised provenance anomaly detectors," in *11th International Workshop on Theory and Practice of Provenance (TaPP 2019)*, 2019.

[6] E. Manzoor, S. M. Milajerdi, and L. Akoglu, "Fast memory-efficient anomaly detection in streaming heterogeneous graphs," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 1035–1044.

[7] X. Han, T. Pasquier, A. Bates, J. Mickens, and M. Seltzer, "Unicorn: Runtime provenance-based detector for advanced persistent threats," in *NDSS*, 2020.

[8] Q. Wang, W. U. Hassan, D. Li, K. Jee, X. Yu, K. Zou, J. Rhee, Z. Chen, W. Cheng, C. A. Gunter, and H. Chen, "You are what you do: Hunting stealthy malware via data provenance analysis," in *27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26*, 2020.

[9] W. U. Hassan, A. Bates, and D. Marino, "Tactical provenance analysis for endpoint detection and response systems," in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020.

[10] 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 *NDSS*, 2019.

[11] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in neural information processing systems*, 2017, pp. 1024–1034.

[12] J. Gao, T. Zhang, and C. Xu, "I know the relationships: Zero-shot action recognition via two-stream graph convolutional networks and knowledge graphs," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 8303–8311.

[13] A. Rahimi, T. Cohn, and T. Baldwin, "Semi-supervised user geolocation via graph convolutional networks," *arXiv preprint arXiv:1804.08049*, 2018.

[14] H. Dai, C. Li, C. Coley, B. Dai, and L. Song, "Retrosynthesis prediction with conditional graph logic network," in *Advances in Neural Information Processing Systems*, 2019, pp. 8870–8880.

[15] S. Singh, P. K. Sharma, S. Y. Moon, D. Moon, and J. H. Park, "A comprehensive study on apt attacks and countermeasures for future networks and communications: challenges and solutions," *The Journal of Supercomputing*, vol. 75, no. 8, pp. 4543–4574, 2019.

[16] F. Liu, Y. Wen, D. Zhang, X. Jiang, X. Xing, and D. Meng, "Log2vec: A heterogeneous graph embedding based approach for detecting cyber threats within enterprise," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 1777–1794.

[17] Z. Li, X. Cheng, L. Sun, J. Zhang, and B. Chen, "A hierarchical approach for advanced persistent threat detection with attention-based graph neural networks," *Security and Communication Networks*, vol. 2021, 2021.

[18] 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.

[19] "Novelty detection with local outlier factor," Website, 2019, https://scikit-learn.org/stable/modules/outlierdetection.html#novelty-detection-with-local-outlier-factor.

[20] Y. Xie, D. Feng, Y. Hu, Y. Li, S. Sample, and D. Long, "Pagoda: A hybrid approach to enable efficient real-time provenance based intrusion detection in big data environments," *IEEE Transactions on Dependable and Secure Computing*, 2018.

[21] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, 2017, pp. 1285–1298.

[22] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li *et al.*, "Robust log-based anomaly detection on unstable log data," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 807–817.

[23] B. Xia, J. Yin, J. Xu, and Y. Li, "Loggan: A sequence-based generative adversarial network for anomaly detection based on system logs," in *International Conference on Science of Cyber Security*. Springer, 2019, pp. 61–76.

[24] X. Shu, D. Yao, and N. Ramakrishnan, "Unearthing stealthy program attacks buried in extremely long execution paths," in *Acm Sigsac Conference on Computer & Communications Security*, 2015.

[25] R. Sekar, M. Bendre, D. Dhurjati, and P. Bollineni, "A fast automaton-based method for detecting anomalous program behaviors," in *Proceedings 2001 IEEE Symposium on Security and Privacy. S&P 2001*. IEEE, 2000, pp. 144–155.

[26] B. Bowman, C. Laprade, Y. Ji, and H. H. Huang, "Detecting lateral movement in enterprise computer networks with unsupervised graph ai," in *23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)*, 2020, pp. 257–268.

[27] 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.

[28] Y. Liu, M. Zhang, D. Li, K. Jee, Z. Li, Z. Wu, J. Rhee, and P. Mittal, "Towards a timely causality analysis for enterprise security," in *NDSS*, 2018.

[29] M. N. Hossain, S. Sheikhi, and R. Sekar, "Combating dependence explosion in forensic analysis using alternative tag propagation semantics," in *IEEE S&P*, 2020.

[30] 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 *30th USENIX Security Symposium (USENIX Security 21)*, 2021.

[31] A. D. Keromytis, "Transparent computing engagement 3 data release," Website, 2018, https://github.com/darpa-i2o/Transparent-Computing/blob/master/README-E3.md.

[32] "How many alerts is too many to handle?" Website, 2020, https://www2.fireeye.com/StopTheNoise-IDC-Numbers-Game-Special-Report.html.

[33] 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.

[34] X. M. Han, "Unicorn," Website, 2021, https://github.com/crimson-unicorn.

[35] M. Emaad, "sbustreamspot-core," Website, 2020, https://github.com/sbustreamspot/sbustreamspot-core.

[36] B. Xia, "Isf," Website, 2020, https://github.com/coderxor/ISF.

[37] "Deeplog," Website, 2020, https://github.com/wuyifan18/DeepLog.

[38] "logdeep," Website, 2020, https://github.com/donglee-afar/logdeep.

[39] "Streamspot data," Website, 2016, https://github.com/sbustreamspot/sbustreamspot-data.

[40] B. Jacob, P. Larson, B. Leitao, and S. Da Silva, "Systemtap: instrumenting the linux kernel for analyzing performance and functional problems," *IBM Redbook*, vol. 116, 2008.

[41] X. M. Han, "shellshock-apt," Website, 2019, https://github.com/margoseltzer/shellshock-apt.

[42] D. Zügner, A. Akbarnejad, and S. Günnemann, "Adversarial attacks on neural networks for graph data," in *the 24th ACM SIGKDD International Conference*, 2018.

[43] B. Wang and N. Z. Gong, "Attacking graph-based classification via manipulating the graph structure," *ACM*, 2019.

[44] L. Sun, D. Y, C. Yang, J. Wang, and B. Li, "Adversarial attack and defense on graph data: A survey," 2020.

[45] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *2010 IEEE symposium on security and privacy*. IEEE, 2010, pp. 305–316.

[46] I. H. Witten and E. Frank, "Data mining: practical machine learning tools and techniques with java implementations," *Acm Sigmod Record*, vol. 31, no. 1, pp. 76–77, 2002.

[47] X. Tang, Y. Li, Y. Sun, H. Yao, P. Mitra, and S. Wang, "Transferring robustness for graph neural network against poisoning attacks," in *Proceedings of the 13th International Conference on Web Search and Data Mining*, 2020, pp. 600–608.

[48] Z. Xi, R. Pang, S. Ji, and T. Wang, "Graph backdoor," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021.

[49] D. Zhu, Z. Zhang, P. Cui, and W. Zhu, "Robust graph convolutional networks against adversarial attacks," in *the 25th ACM SIGKDD International Conference*, 2019.

[50] M. Jin, H. Chang, W. Zhu, and S. Sojoudi, "Power up! robust graph convolutional network against evasion attacks based on graph powering," in *arXiv preprint arXiv:1905.10029*, 2019.

[51] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.