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 whole provenance graph, are not sensitive to the small number 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 three public datasets. The results show that THREATRACE outperforms three 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 be detected.

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 algorithm to dynamically model the whole 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 whole graphs is not sensitive to a small number 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 campaign into several parts instead of appearing in a complete path, which makes it difficult to be detected by path-level methods. Misuse-based methods [2], [3], [9] typically define some attack patterns and use them to match anomalous behavior. In consideration of the frequent use of zero-day exploits by modern attackers, misuse-based methods lack the ability 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 as source input and tailors an inductive graph neural network framework (GraphSAGE) [11] to learn the rich contextual information in data provenance. GraphSAGE is a kind of Graph Neural Network (GNN), which is a group of neural network models designed for various graph-related work and has succeeded in many domains (e.g., Computer Vision [12], [13], Natural Language Processing [14], [15], Chemistry and Biology [16], [17], etc.). We utilize its ability to learn structural information about a node's role in a provenance graph [11] for node-level threats detection.

Using GraphSAGE for threats detection is challenging: 1) How to train the model without prior knowledge of attack patterns? 2) How to tackle the data imbalanced problem [18]?

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 a system data provenance graph to capture stealthy abnormal behavior without prior knowledge of attack patterns. 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 that 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 finding the location of anomalous behavior. We evaluate THREATRACE in three public datasets and demonstrate that THREATRACE can effectively detect host-based threats with a small proportion in the whole-system provenance with fast processing speed and acceptable resource overhead.

Our paper makes contributions summarized as follows:

- **Novel node level threats detection**. To the best of our knowledge, THREATRACE is the first work to formalize the host-based threats detection problem as an anomalous nodes

detection and tracing problem in a provenance graph. Based on the problem statement, 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 three public datasets and compare it with three state-of-the-art APTs detection approaches. THREATRACE outperforms them in these datasets. We further evaluate THREATRACE's threats 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 threats 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 introduction 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 threats detection and anomaly tracing. Thus, we discuss related work in these areas.

Provenance-based threats detection. Data provenance is attractive in the host threats detection community recently, which 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] detect 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 not included in the TTPs and cyber threat reports.

Anomaly-based approaches learn models of benign behavior and detect anomalies based on the deviations from the models. StreamSpot [6] proposes to detect intrusion by analyzing information flow graphs. It abstracts features of the graph locally to learn benign models and uses a cluster approach to detect abnormal graphs. Unicorn [7] further proposes a WL-kernel-based method to extract the features of the whole graph, 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 using its contextualized graph analysis and evolving models. However, due to the restriction of graph kernel methods, they have difficulty detecting stealthy threats. IPG [19] 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. Log2vec [18] proposes to detect host threats based on anomaly 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. 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 [20] to detect malware. Besides malware, host-based threats' behavior is more diverse. Therefore, it is not enough to mine threats from the paths of the provenance graph. Pagoda [21] focuses on detecting host-based intrusion in consideration of both detection accuracy and detection time. In Pagoda's framework, a rule database is required, which is different from THREATRACE.

Besides provenance, there are other data sources used for host-based anomaly detection. [22] detects stealthy program attacks by modeling normal program traces and using clustering methods to make anomaly detection. [23] 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 threats detections because of the lack of contextual information. [24] detects the lateral movement of APTs taking a graph of authenticating entities as input. Nodes in the authentication graph represent either machines or users. However, this approach cannot detect intrusion campaigns within a single host.

Anomaly tracing. When 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 when 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 position of anomalies. Detectors that are capable of tracing the location of anomalies are usually misuse-based such as Holmes [3] and Poirot [2], which need prior knowledge about abnormal graph patterns. Nodoze [10] is another approach to identity anomalous path in provenance graphs. Unlike those detectors mentioned above, Nodoze is a secondary triage tool that 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 without detailed information. RapSheet [9] is another secondary triage system that uses alerts from other TTPs-based EDR (Endpoint Detection and Response) and filters false positives in constructed TPGs (Tactical Provenance Graphs). RapSheet faces the same problem as those misuse-based detection methods. SLEUTH [25] performs tag and policy-based attack detection and tag-based root-cause and impact analysis to construct a scenario graph. PrioTracker [26] quantifies the rareness of each event to distinguish abnormal operations from normal system events and proposes a forward tracking technique for timely attack causality analysis. MORSE [27] 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 [25] system. The attack detection part of SLEUTH [25] and MORSE [27] are both misuse-based. ATLAS [28]

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

proposes a sequence-based learning approach for detecting threats and constructing the attack story. Different from the misuse-based methods [2], [3], [27], [25], ATLAS uses attack training data to learn the co-occurrence of attack steps through temporal-ordered sequences. The methods introduced above propose various methods to trace an intrusion and construct the attack story. THREATRACE can trace the anomaly without a knowledge base 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 representing the relationship between subjects (e.g., *processes*) and objects (e.g., *files*) in a system. Data provenance contains rich contextual information for threats detection.

B. GraphSAGE

GraphSAGE [11] is a general inductive GNN framework for efficiently generating node embeddings with node feature information. Unlike the transductive methods, which embed nodes from a single fixed graph, the inductive GraphSAGE learns an embedding function and operates on evolving graphs. Timeliness is an essential factor for preventing intrusion. Therefore, as an inductive GNN approach, GraphSAGE is suitable for performing threats detection via analyzing streaming provenance graphs. 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 (Figure 1) 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 THEIA experiment [29]. The dark nodes are those nodes related to the intrusion campaign. 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 of `c.c.c.c`.

Limitations of state-of-the-art threat detection work. It remains 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. Because many MITRE ATT&CK behaviors are only sometimes 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 algorithm to dynamically model the whole graph and detect abnormal graphs by clustering approaches. However, the intrusion may be stealthy, which means that a system's provenance graph under intrusion campaign may be similar to those of benign systems. In this example, there are millions of benign nodes and less than 30000 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 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 complicated intrusions besides malware. In this example, there are thousands of edges generated by node ①, which brings challenges for path-level detection methods.

P3 : *Anomaly tracing.* Anomaly-based [6], [7] methods modeling the whole graph lack the capability of tracing the location of anomalous behavior, which is vital to trace abnormal behavior and fix the system. In this example, these methods can only raise alarms for the whole 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, which thus results in low practicality.

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 Figure 1, 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 this attack Figure 1 is 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 lots of benign background data, which makes the victim system like a benign mode.

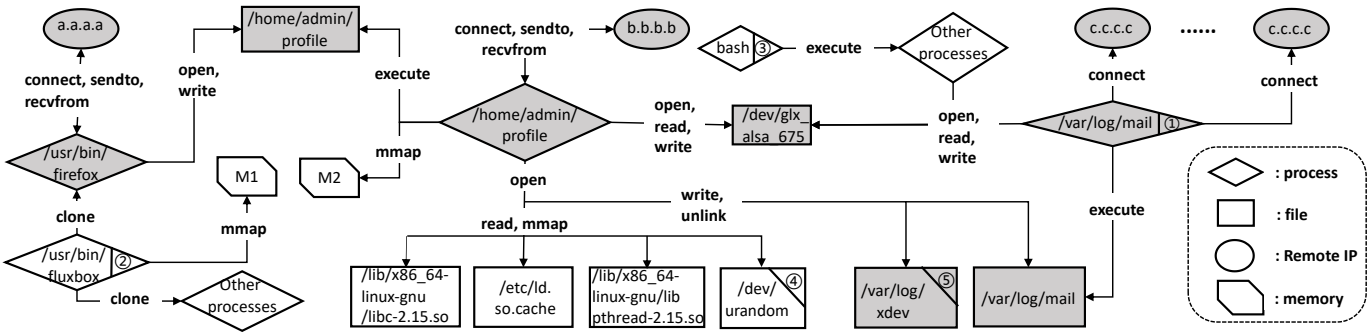

Fig. 1: A toy example of a provenance graph, which contains host-based intrusion behaviour.

- **Persistent.** The attack tends to last for a long time.
- **Frequent use 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 that is different from the benign activity, 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 Figure 1, 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 $process$ that the attacker has controlled. Note that restricted by the granularity of provenance graph, some threats are out of THREATRACE's detection scope. We discussion this limitation in §VII.

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 Figure 2 and presented as follows. We detail them further in the denoted subsections.

- **(§V-A) Data Provenance Generator.** This component collects audit data of a system in a streaming mode and transforms them into a data provenance graph for the following analysis.
- **(§V-B) Data Storage.** This component allocates data to the disk and memory. We store the whole graph in the disk to keep 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: We assume that we do not have prior attack knowledge in the training phase. Thus we cannot train the model in traditional binary classification mode.

  C2: Host-based threat detection has a data imbalanced problem [18]. Anomalous nodes may have a small proportion in the provenance graph during execution. Thus, it is more likely to raise false positives that may disturb the judgment and cause "threat fatigue problem" [30].

  C3: The difference between an anomalous node and benign nodes may not be big enough (e.g., nodes ④ an ⑤ in Figure 1), 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 for the monitored system. 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 threats detection approaches [2], [3], [6], [7], we use an external tool Camflow [31] to construct a single, whole-system provenance graph with time order. Camflow provides strong security and completeness guarantees to information flow capture.

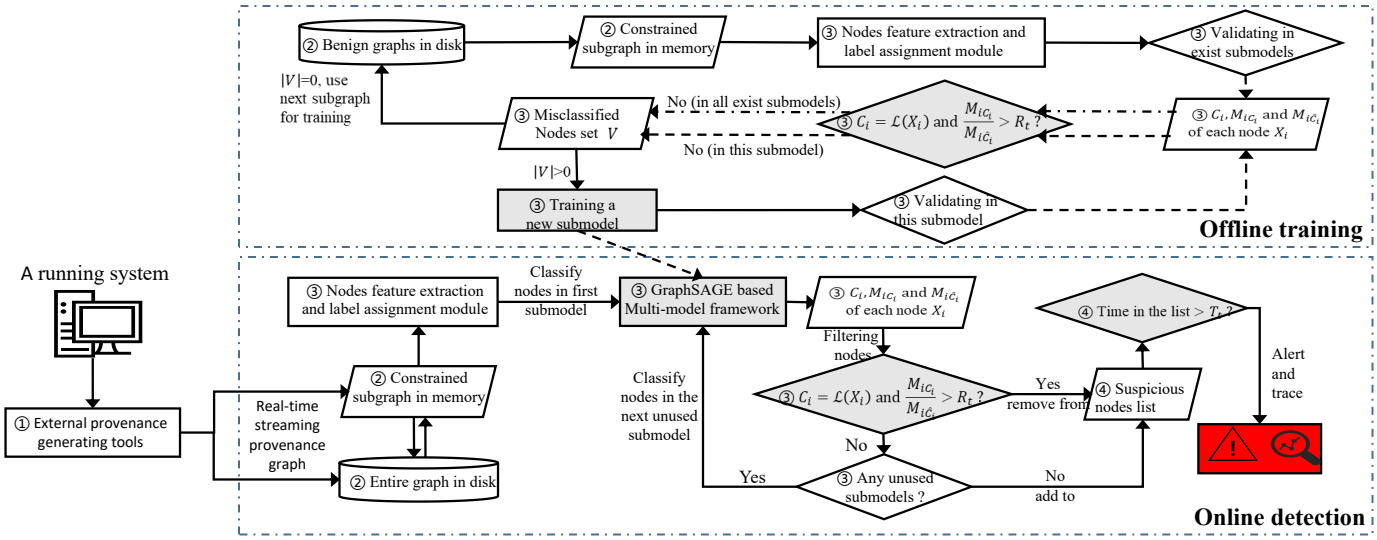

Fig. 2: The overview of components in THREATRACE. ①: Data Provenance Generator; ②: Data Storage; ③: Model; ④: Alert and Trace.

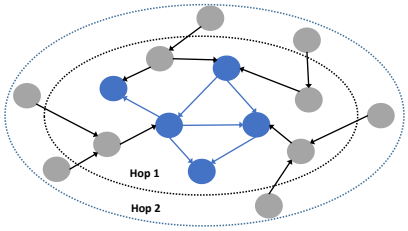

Fig. 3: An example of the subgraph maintaining in memory. Nodes in blue denote the *active* nodes, and the gray nodes denote the *related* nodes.

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. Figure 3 presents an example of the subgraph in memory. 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 and methods based on the characteristics of host-based threats.

Feature Extraction. It is challenging to achieve the goal of detecting 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 threats detection because it assumes that nodes close together have a similar class. In a provenance graph, the close nodes may have different classes (e.g., two close nodes may be a *process* and a *file*).

Since GraphSAGE's unsupervised mode is unsuitable for threats 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 distribution of numbers of different edges' types related to it. The GraphSAGE model is trained with labeled data in a supervised mode, which learns different roles of benign nodes. The feature extraction approach is based on the central insight behind our model that the nodes related to the malicious activities have different behavior from the benign nodes that our model learns during the training phase. Thus, if a node is misclassified during the executing phase, it means that its role is different from what it should be. 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 $\mathcal{M}_v : \sum \to \mathbb{N}$ and $\mathcal{M}_e : \sum \to \mathbb{N}$ 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 $\mathcal{L} : V \to \{0, ..., N_n - 1\}$ to assign label for $\forall v \in V$ as $\mathcal{L}(v) = \mathcal{M}_v(\mathcal{X}_v(v))$. We learn a function $\mathcal{F} : V \to \mathbb{N}^{2*N_e}$ 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 Figure 1 are [0,0,0,0,0,0,0,2,1,0,0,0,0,0] and [0,0,0,0,0,0,0,0,0,0,1,1,1,1,25301], 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 150000 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 scalability.

Our model is based on GraphSAGE, which uses a forward propagation (FP) algorithm [11] to aggregate information from a node's ancestors. A GraphSAGE model uses 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 $AGGREGATE_k, \forall k \in \{1, ..., K\}$; nonlinear 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 and the index of the biggest element in z_v means the predicted class. The choice of $Batch\ size$ (BS) affects the runtime overhead and we evaluate it in §VI-E. K denotes the size of ancestors from that each node aggregates information. 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 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. In an intrusion detection scene: 1) The number of different types of nodes is very unbalanced (e.g., there may be thousands of *process* nodes while only one *stdin* node). This critical unbalanced characteristic causes difficulty for a single model to learn the representation of those nodes with a small proportion. 2) Nodes with the same type may have different missions (e.g., the `fluxbox` process ② and `bash` process ③ in Figure 1), which makes it hard to classify those nodes into one class by one model. Therefore, it is unrealistic to train only one model that can 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 no nodes are left in X, which means that every node in the subgraph has been used to train a submodel that learns its representation.

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

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

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 $\mathcal{F} : V \rightarrow \mathbb{N}^{2*N_e}$; Label mapping function $\mathcal{L} : V \rightarrow \{0, ..., N_n\}$; 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 $z \leftarrow \text{FP}(\hat{G}, \mathcal{F}, K, AGGREGATE_k, \mathbf{W}^k, \sigma)$;
9 //z are the representations of $\forall v \in \hat{V}$
10 $\hat{z} \leftarrow z_{\forall v \in X}$;
11 $loss \leftarrow \text{cross\_entropy}(\hat{z}, \mathcal{L}(X))$;
12 $\mathbf{W}^k \leftarrow \text{backward}(loss)$;
13 **end**
14 $M \leftarrow \text{softmax}(\hat{z})$;
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**

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 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. Thus, we set a threshold R and consider X_i is correctly classified by this submodel when:

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

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 too many submodels. Therefore, we apply a heuristic method (line 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 representations of those removed nodes. 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 that the current multi-model framework cannot correctly classify. It is incremental work, which prevents producing "excess" submodels.

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 it, 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 just because the probability of other classes in this submodel is lower. Once it is correctly classified, the detector raises a false negative. Thus, during the executing phase, we also use the threshold R and consider node v is correctly classified only when:

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

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 it 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 it does not mean R should 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. It is because 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 using three public datasets, which are frequently used by state-of-the-art host-based threats detector evaluation. We focus on the following aspects:

Q1. The threats 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 intusion. (§VI-C)
Q5. The runtime and system resource overhead. (§VI-E)
Q6. The robustness against adaptive attacks. (§VI-F)

Here we introduce the dataset, experimental setup, and implementation of comparison work. In §VI-A, VI-B, we compare THREATRACE to Unicorn, ProvDetector and StreamSpot, three state-of-the-art anomaly-based host threats detectors using their own datasets: StreamSpot and Unicorn SC-2 datasets. For StreamSpot and Unicorn, we implement them with the open-source projects. The results of them are almost the same as the original papers. For ProvDetector, we reimplement the approaches based on the original paper. For ProvDetector, we cannot compare the performance with the original paper because we do not have the private dataset of the paper for evaluation. We do not compare THREATRACE to other related state-of-the-art detectors because: 1) Misuse-based detectors require a priori expert knowledge to construct a model [2], [3], [9], [28]; 2) The source input is not data provenance [18], [22], [24], [28]. These approaches are introduced in §II.

We show that THREATRACE can achieve better detection performance (**Q1**) in both datasets. In §VI-D, we evaluate how the specially designed parameters influence the threats detection performance (**Q2**). In §VI-C, we use the DARPA TC dataset 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. Table V, VI, IV show the overview of the experimental datasets.

The manually set parameters of experiments 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's hyperparameters. Specifically, we set most of those hyperparameters as default values and 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 we set $K = 2$, which denotes ancestors' size from which each node aggregates information. We test the influence of parameters in §VI-D.

TABLE I: Parameters of experiments.

Parameter	Description	Section	Values
BS	Batch mode parameter of GraphSAGE [11]	V-C	5000
SS	Maximun number of $active$ nodes	V-C	200000
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	8315	173857
Attack	100	8891	28423

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 [32]. It contains 6*100 information flow graphs derived from five benign scenes and one attack scene. 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. As shown in Table V, THREATRACE can achieve a perfect detection performance in this dataset. 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 graph 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, which may explain why Unicorn has a high detection performance in this dataset but still cannot detect every abnormal graph. This result demonstrates THREATRACE's ability to detect the stealthy intrusion campaign.

B. Unicorn SC-2 Dataset

This dataset is Unicorn's own dataset [33], which is more complex than the StreamSpot dataset. We use it 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. We follow the same 5-fold cross-validation as Unicorn: 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 use a streaming mode to replay the validation graphs and detect them dynamically. For THREATRACE, we evaluate the detection performance with different neighborhood size, which is setted as 1 and 2 (default). We repeat this procedure and report the mean evaluation results.

Result. As shown in Table VI, THREATRACE achieves a better performance when $K = 2$, which indicates that adequate neighbor information learning is helpful for detection. We further discuss the setting of K in §VII. THREATRACE can achieve better detection performance than Unicorn using its

TABLE III: Overview of Unicorn SC-2 dataset.

Scene	# of graph	Average # of nodes	Average # of edges
Benign	125	238338	911153
Attack	25	243658	949887

TABLE IV: Overview of DARPA TC dataset.

Scene	System	# of benign nodes	# of abnormal nodes	# of edges
THEIA	Ubuntu	3505326	25362	102929710
Trace	Ubuntu	2416007	67383	6978024
CADETS	FreeBSD	706966	12852	8663569
fivedirections	Windows	569848	425	9852465

own dataset. The reason that Unicorn has a 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".

C. DARPA TC Dataset

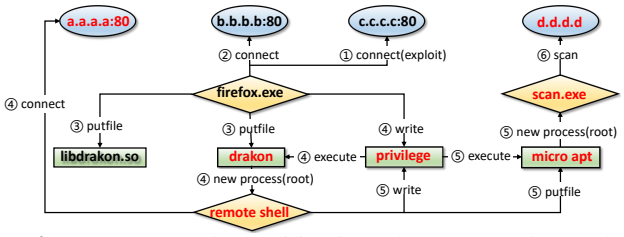

Fig. 4: The case study. Entities in red are those detected by THREATRACE.

We use DARPA TC dataset 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 dataset (Table IV) is generated in the third red-team vs. blue-team engagement of the DARPA Transparent Computing program. The engagement lasted for two weeks and the provenance collecting tools captured provenance data of the whole system from start to end. The provenance data and ground truth are publicly available [29]. 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. We remove part of the dataset because some graphs are generated in exceptional accidents such as outages and shutdown of hosts. We use nodes in benign graphs to train our models and use nodes in graphs containing threats 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 VII show THREATRACE's performance of detecting abnormal nodes, which means the ability of anomaly tracing. The results are acceptable in consideration of the highly imbalanced characteristic of this dataset. We cannot compare THREATRACE with StreamSpot, Unicorn, and ProvDetector at node level because they do not make anomalous nodes detection. The detection results of graph level experiments are shown in Table VIII. The results show that both THREATRACE and the comparison methods have good detection performance. One explanation is that the attackers in the DARPA datasets spend time finding vulnerabilities that leave apparent behavior in the graphs.

Case study. We use the motivation example (Figure 1) as the case study to further describe THREATRACE's anomaly tracing ability. It is simplified as an attack graph in Figure 4. 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 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

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

Detector	Precision	Recall	Accuracy	F-Score	TP	TN	FP	FN	FPR
StreamSpot	0.72	1.0	0.69	0.75	25	108.5	16.5	0	0.11
Unicorn	0.95	0.97	0.99	0.96	24.32	123.64	1.36	0.68	0.011
THREATRACE	1.0	1.0	1.0	1.0	25	125	0	0	0

TABLE VI: 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	0.27
ProvDetector	0.67	0.6	0.65	0.63	15	17.5	7.5	10	0.3
THREATRACE (K = 1)	0.81	0.79	0.8	0.8	19.7	20.5	4.5	5.3	0.18
THREATRACE (K = 2)	0.91	0.96	0.93	0.93	24	22.5	2.5	1	0.1

TABLE VII: Results of node level detection in DARPA TC dataset

Experiment	System	Precision	Recall	F-Score	TP	TN	FP	FN	FPR
THEIA	Ubuntu	0.87	0.99	0.93	25297	3501561	3765	65	0.001
Trace	Ubuntu	0.72	0.99	0.83	67382	2389233	26774	1	0.011
CADETS	FreeBSD	0.90	0.99	0.94	12848	705605	1361	4	0.002
fivedirections	Windows	0.67	0.92	0.80	389	569660	188	36	0.0003

TABLE VIII: Results of graph level detection in DARPA TC dataset

Experiment	Detector	Precision	Recall	F-Score	FP	FN	FPR
THEIA	THREATRACE	1.0	1.0	1.0	0	0	0
	Unicorn	1.0	1.0	1.0	0	0	0
	ProvDetector	0.96	0.92	0.94	0.2	0.4	0.04
CADETS	THREATRACE	1.0	1.0	1.0	0	0	0
	Unicorn	1.0	1.0	1.0	0	0	0
	ProvDetector	0.92	0.94	0.93	0.4	0.3	0.08
fivedirections	THREATRACE	1.0	1.0	1.0	0	0	0
	Unicorn	1.0	1.0	1.0	0	0	0
	ProvDetector	1.0	1.0	1.0	0	0	0
Trace	THREATRACE	1.0	1.0	1.0	0	0	0
	Unicorn	1.0	1.0	1.0	0	0	0
	ProvDetector	1.0	1.0	1.0	0	0	0

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 of 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 the intrusion in the early phases. We also note that THREATRACE fails to detect some components of the attack like `libdarkon.so` and malicous 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.

D. Influence of Different Parameters

THREATRACE shows a detection performance improvement 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 threats 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.

- $Subgraph\ Size\ (SS)$. SS is the parameter we set for the Data Storage module (§V-B). As shown in Figure 5(a), it has little influence on detection performance. THREATRACE detects the provenance graph in a streaming mode. The subgraph of current system execution is updated in the memory and is transferred to the detection module when its size reaches SS. Thus, SS affects the coming interval of subgraphs, which may influence the time we set for benign nodes to reach their final stage. Besides, the randomly divided dataset may have an impact on the results as well. Compared to detection performance, SS has much more influence on runtime performance (§VI-E).
- $Batch\ Size\ (BS)$. BS is the parameter we set for the Model module (§V-C). As shown in Figure 5(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 Figure 5(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 Figure 5(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 threats 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-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. Figure 6(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 orange 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 Figure 6(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 Figure 6(b), runtime performance improves as we increase SS.

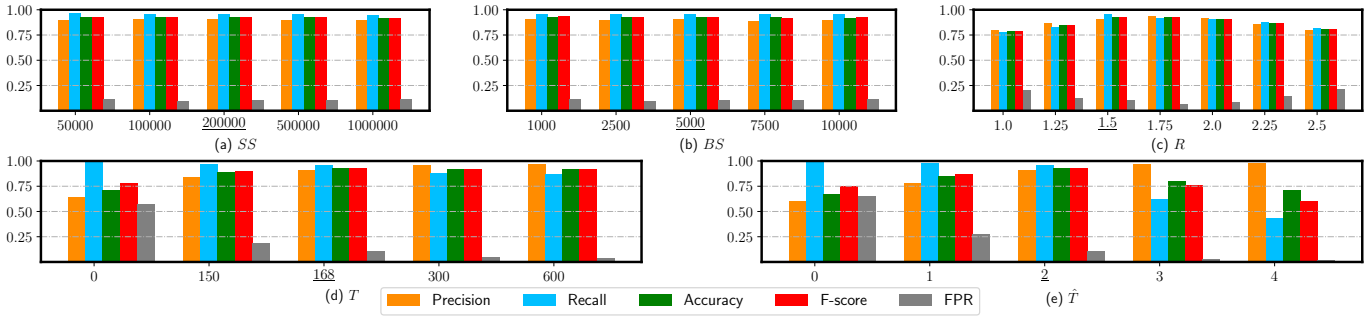

Fig. 5: Detection performance with varying parameters.

CPU utilization & Memory usage. Figure 6(c)(d) demonstrates THREATRACE's CPU utilization and memory usage in different SS and BS. The green and orange 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 that 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.

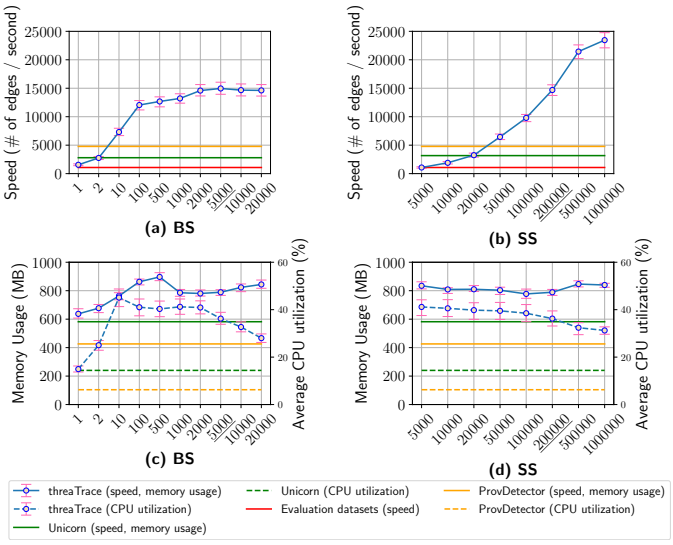

Fig. 6: Runtime performance with varying parameters. The evaluation of each parameter is done with the remaining parameters constant.

F. Adaptive attacks

Attack Methodology. THREATRACE is a Graph-based detection system. Therefore, we borrow the idea of existing adversarial attack against Graph-based detectors [34], [35], [36] to develop an adaptive attack on THREATRACE, called optimization-based evasion attack. 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 feature which is constructed with edges between the node and its neighbor. 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 attack's 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.

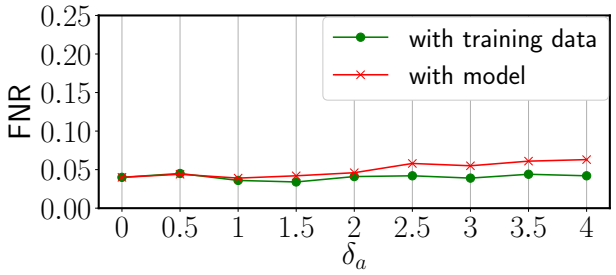

Fig. 7: Result of evasion attack experiments.

(1) Attackers with training data. This kind of attacker has the training data. The optimization-based 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, which is not difficult:

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

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

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

$Loss(\hat{x})$ 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 evasion.

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 Figure 7. 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 attack.

VII. DISCUSSION & LIMITATION

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

Closed-world assumption. Host-based threats detection is essentially a classification problem. It is based on a closed world assumption[37], [38], 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. The conflict between the closed-world assumption and real-life situations makes it hard to design an ideal detector. As the same as other anomaly-based detectors, THREATRACE faces this problem too. We cannot guarantee that we learn every benign node in models especially considering the rapid development of present systems. However, system administrators can update models periodically using updated benign data provenance of the system. It is incremental work with no need to retrain the previous models and thus does not need much time (§V-C).

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 [39] and graph backdoor [40]. Future work can study the robustness against more attacks. Some pioneering methods for enhancing GNN's robustness can be found in [41], [42]. They are not suitable for THREATRACE because they focus on robustness in graph convolutional networks, which are transductive models different from our inductive model (§III-B).

Evaluation. In host-based threats detection domain, public datasets are usually generated in a manually constructed environment [32], [33] and thus may not represent typical workloads in practice. Data generated from red-team vs. blue-team engagement [29] is partly closed to the actual situation. It is important to generate a public dataset for the community which can reflect the typical workloads in 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 [30]. 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 software. 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. 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).

Granularity of data provenance. THREATRACE falls into the category of provenance-based methods. Some threats (e.g., malicious code in a file and thread-based threats) will not demonstrate the attack patterns in the provenance graph because the text information of files and thread activities are too fine-granularityed for data provenance, which thus results in detection failure. It is a common limitation for provenance-based methods. Improving the granularity of data provenance or utilizing another novel data source to detect host-based threats are our future works.

VIII. Conclusion

We present THREATRACE, a real-time host-based node level threats 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*, 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] Z. Wang, T. Chen, J. Ren, W. Yu, H. Cheng, and L. Lin, "Deep reasoning with knowledge graph for social relationship understanding," *arXiv preprint arXiv:1807.00504*, 2018.

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

[15] L. Yao, C. Mao, and Y. Luo, "Graph convolutional networks for text classification," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 7370–7377.

[16] 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.

[17] E. Choi, Z. Xu, Y. Li, M. Dusenberry, and A. Dai, "Learning the graphical structure of electronic health records with graph convolutional transformer," 2020.

[18] 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.

[19] 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.

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

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

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

[28] 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.

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

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

[31] 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.

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

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

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

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

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

[37] 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.

[38] 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.

[39] 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.

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

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

[42] 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.