Computers, Materials & Continua

Tech Science Press

DOI: 10.32604/cmc.2023.045739

ARTICLE

Detecting APT-Exploited Processes through Semantic Fusion and Interaction Prediction

Bin Luo[1,2,3], Liangguo Chen[1,2,3], Shuhua Ruan[1,2,3,*] and Yonggang Luo[2,3,*]

[1]School of Cyber Science and Engineering, Sichuan University, Chengdu, 610065, China

[2]Key Laboratory of Data Protection and Intelligent Management (Sichuan University), Ministry of Education, Chengdu, 610065, China

[3]Cyber Science Research Institute, Sichuan University, Chengdu, 610065, China

*Corresponding Authors: Shuhua Ruan. Email: ruanshuhua@scu.edu.cn; Yonggang Luo. Email: iamlyg98@scu.edu.cn

Received: 06 September 2023 Accepted: 06 December 2023 Published: 27 February 2024

ABSTRACT

Considering the stealthiness and persistence of Advanced Persistent Threats (APTs), system audit logs are leveraged in recent studies to construct system entity interaction provenance graphs to unveil threats in a host. Rule-based provenance graph APT detection approaches require elaborate rules and cannot detect unknown attacks, and existing learning-based approaches are limited by the lack of available APT attack samples or generally only perform graph-level anomaly detection, which requires lots of manual efforts to locate attack entities. This paper proposes an APT-exploited process detection approach called ThreatSniffer, which constructs the benign provenance graph from attack-free audit logs, fits normal system entity interactions and then detects APT-exploited processes by predicting the rationality of entity interactions. Firstly, ThreatSniffer understands system entities in terms of their file paths, interaction sequences, and the number distribution of interaction types and uses the multi-head self-attention mechanism to fuse these semantics. Then, based on the insight that APT-exploited processes interact with system entities they should not invoke, ThreatSniffer performs negative sampling on the benign provenance graph to generate non-existent edges, thus characterizing irrational entity interactions without requiring APT attack samples. At last, it employs a heterogeneous graph neural network as the interaction prediction model to aggregate the contextual information of entity interactions, and locate processes exploited by attackers, thereby achieving fine-grained APT detection. Evaluation results demonstrate that anomaly-based detection enables ThreatSniffer to identify all attack activities. Compared to the node-level APT detection method APT-KGL, ThreatSniffer achieves a 6.1% precision improvement because of its comprehensive understanding of entity semantics.

KEYWORDS

Advanced persistent threat; provenance graph; multi-head self-attention; graph neural network

1 Introduction

APTs bring huge losses to governments or enterprises for their stealthiness and persistence, and have been attracting much attention from cybersecurity researchers. In recent years, related studies usually collect system audit logs, take system entities (e.g., processes) as nodes, and system interaction

events (e.g., read, write, connect) as edges to construct a provenance graph, which provides the context for system entity interactions. The provenance graph describes the running history of programs in the system with a graph representation, bringing rich contextual information for detecting and investigating APT activities [1,2].

Existing APT detection approaches based on provenance graphs can be categorized into rule-based detection and learning-based detection. Rule-based detection approaches define rules through prior knowledge about attack activities and match system entity behaviors with predefined rules to achieve APT detection. This type of method can provide explanations for detection results that are beneficial to attack investigation. However, many different techniques could be used for APTs, making rule writing a difficult and burdensome task that requires specialized knowledge of threat models, operating systems, and networks. According to a recent survey, the rules of commercial Security Information Event Management (SIEM) products cover only 16% [3] of the public Tactics, Techniques, and Procedures knowledge (TTPs). In addition, rule-based methods make it difficult to detect unknown attacks while zero-day vulnerabilities are inevitable in APTs. Learning-based approaches train deep learning models in a supervised or semi-supervised way to perform APT detection. Supervised learning suffers from insufficient APT samples. Though semi-supervised learning can train models only based on attack-free logs to detect attacks without the need for APT samples, most existing semi-supervised provenance-based APT detection methods focus on detecting suspicious provenance graphs containing APT attacks. These suspicious graphs often contain thousands of edges and nodes, making it difficult for security engineers to quickly complete attack investigation [2].

This paper proposes a semi-supervised APT-exploited process detection approach called ThreatSniffer, which could achieve fine-grained APT detection based on attack-free audit logs. ThreatSniffer understands system entities from their file paths, interaction sequences, and the number distribution of interaction types. Then, considering that malicious processes exist some unexpected interactions with other system entities, ThreatSniffer understands entity behavior through system entity interaction rationality and then identifies anomaly processes. Evaluation results on the DARPA TC3 Theia dataset show that ThreatSniffer can detect the processes associated with APT attack activities.

The contributions of this paper are summarized as follows:

1. To understand system entities comprehensively, ThreatSniffer embeds entity semantics from three aspects: file paths, interaction sequences, and the number distribution of interaction types. Moreover, ThreatSniffer employs a multi-head self-attention mechanism for semantic fusion.

2. To achieve fine-grained APT detection, ThreatSniffer employs a heterogeneous graph neural network to understand the system entity interaction context in a provenance graph and predict the rationality of interactions to identify anomaly processes.

3. To fit normal system activities, ThreatSniffer adopts a semi-supervised learning strategy for model training. It performs negative sampling on the benign provenance graph to generate non-existent edges to characterize irrational entity interactions.

4. ThreatSniffer is implemented and verified on the DARPA TC3 Theia dataset. The results demonstrate that ThreatSniffer can detect processes exploited by attackers, and achieves higher precision and recall than existing node-level APT detection methods.

The remaining paper is organized as follows: Section 2 introduces existing work related to provenance graph APT detection; Section 3 introduces our motivation and overviews ThreatSniffer, the APT-exploited process detection method proposed in this paper; Section 4 describes the implementation details of ThreatSniffer, including entity semantic embedding and fusion, provenance graph

construction and negative sampling, and the interaction prediction model using graph neural network; Section 5 describes the experimental environment and results; Section 6 summarizes this paper and discusses the direction of future work.

2 Related Work

Rule-based APT detection on provenance graphs. Rule-based detection methods generate predefined rules to describe security threats and then conduct rule matching on provenance graphs to uncover potential attacks. Sleuth [4] designs tags to encode an assessment of the trustworthiness and sensitivity of data as well as processes, and manually customizes policies for carrying out tag propagation and identifying the system entities most likely to be involved in attacks. Sleuth can derive scenario graphs of attack activities. Caused by dependence explosion, it would generate a graph containing numerous benign nodes when facing long-running attacks. Based on Sleuth, Morse [5] introduces tag attenuation and tag decay to mitigate the dependency explosion problem, reducing scenario graph sizes by an order of magnitude. Considering Sleuth's memory consumption issue when handling large amounts of data, Conan [6] uses a finite state machine to describe system entities. It transforms between different states via predefined rules, and alerts when a malicious state combination occurs. Conan utilizes states instead of a provenance graph to record semantics, which ensures constant memory usage over time. When there are a large number of concurrent operations in the system (e.g., a large number of file read and write operations at the same time), Conan can't ensure real-time performance. Holmes [7] customizes detection rules based on TTPs to elevate alerts to the tactics of an attack campaign. It then constructs high-level scenario graphs for intrusion detection. The drawback of Holmes is that it assumes 100% log retention in perpetuity, which is practically prohibitive. Rapsheet [8] introduces skeleton graphs to address the limitation. It creates more TTP matching rules than Holmes. APTSHILED [9] defines suspicious characteristics of system entities and transmission rules using TTPs, and enhances APT detection efficiency by adopting redundant semantics skipping and non-viable node pruning. It outperforms Sleuth, Holmes, and Conan in terms of detection time consumption and memory overhead. **The detection effectiveness of the above rule-based methods relies on the security engineers' understanding of the attack procedure.** To mitigate this dependency, related research utilizes threat intelligence to augment detection rules. Poirot [10] extracts Indicators of Compromise (IOC) and their interrelationships from cyber threat intelligence to construct a query graph of attack behaviors. It then performs APT detection through an inexact graph pattern matching between the provenance and query graph. In practice, attack steps described in threat intelligence are not completely consistent with real attack activities recorded in provenance data. To address this, DeepHunter [11] utilizes graph neural networks for graph pattern matching, offering greater robustness compared to Poirot. ThreatRaptor [12] extracts structured threat behavior from unstructured threat intelligence and describes the threat with TBQL, a domain-specific query language, for querying malicious system activity. **In summary, though rule-based methods can achieve high detection accuracy and explainability. However, they necessitate meticulously crafted, high-quality detection rules grounded in expert insights or threat intelligence, and they cannot handle unknown attacks.**

Learning-based APT detection on provenance graphs. With training datasets with little domain knowledge, learning-based approaches construct detection models for APT detection at various granularities, where semi-supervised models for learning normal system behavior and supervised models for identifying malicious behavior. StreamSpot [13] detects anomalies by dividing the streaming provenance graph into multiple snapshots, extracting local graph features, and clustering the snapshots. StreamSpot handles edges in the provenance graph with a stream fashion and is both time-efficient and memory-efficient. But StreamSpot's graph features are locally constrained. To

mitigate this drawback, Unicorn [14] examines contextualized provenance graphs for APT detection. It can model and summarize the evolving system executions and report abnormal system status. StreamSpot and Unicorn regard suspicious provenance graphs containing attacks as alerts. **Their coarse-grained detection results are not conducive to security engineers' attack investigation, because these suspicious graphs require lots of manual work to find APT-exploited system entities.** Pagoda [15] builds a rule database to characterize benign system behaviors and detect suspicious paths in the provenance graph. Pagoda cannot deal with sequence order transformation and sequence length increase, which are very common when an intrusion process changes its behavior to a variant. To detect these variants, P-Gaussian [16] introduces a Gaussian distribution scheme to characterize and identify intrusion behavior and its variants. However, P-Gaussian still uses a rule database to model benign behaviors. ProvDetecter [17] transfers causal paths into vectors, then a density-based cluster method is deployed to detect the abnormal paths. Considering dependence explosion, based on the assumption that malicious paths are uncommon, ProvDetecter only selects a certain number of rare paths for detection. Attackers may exploit this assumption to evade detection. Atlas [18] utilizes lemmatization and word embedding to abstract the attack and non-attack semantic patterns. It aims to help security engineers recover attack steps while it requires manually providing some known malicious entities as starting points for the paths. **In recent years, graph neural networks have proven to be effective for APT detection [2,19–21], and many researchers have utilized graph neural networks for fine-grained APT detection.** DepComm [22] divides a large provenance graph into process-centric communities and then generates a representative InfoPath for each community as its summary. DepComm cooperates with Holmes for APT detection. Since there are still some less-important events that cannot be compressed by DepComm, it maintains a set of rules to handle these events. Watson [23] utilizes TransE [24] to embed system entity interaction semantics, then combines interaction semantics as the vector representation of behaviors. These vectors are subsequently used for clustering to detect malicious behavior. ShadeWatcher [25] analogizes system entity interactions to user-item interactions in recommender systems. It detects threats by predicting a system entity's preferences for its interacting entities. ShadeWatcher's ability to achieve high-accuracy detection might be challenging when faced with a large provenance graph. **Recent research has focused on utilizing graph neural networks for node-level APT detection.** Deepro [26] achieves fine-grained APT detection by detecting attack-related processes, but its supervised learning method faces the challenge of handling the imbalance between benign and attack samples. In APT-KGL [27], threat intelligence is introduced to augment the APT training samples, and a heterogeneous graph neural network is used to detect malicious processes. Liu et al. [28] utilized an attention-based graph convolutional neural network to infer whether a process is malicious or not. It downsamples and upsamples benign and attack samples respectively to address the sample imbalance problem. **Applying threat intelligence or sampling does not fundamentally address the issue of a lack of attack samples. In addition, supervised models' understanding of attacks is largely constrained by attack samples.** ProGrapher [29] combines whole graph embedding and sequence learning to capture the temporal dynamics between normal snapshots. It detects abnormal snapshots when it deviates from prediction. To achieve fine-grained APT detection, ProGrapher introduces a novel algorithm to pinpoint abnormal entities by computing co-occurrence probability. ThreaTrace [30] adopts node type as node labels and the number distribution of nodes' edge type as node features to perform semi-supervised learning with GraphSage [31] graph neural networks. **It regards the provenance graph as homomorphic and does not effectively take advantage of the rich semantics contained in system audit logs such as file paths, entity interaction sequences, and various types of system interactions.**

With the insight that malicious processes exist some unexpected entity interactions, ThreatSniffer identifies anomaly processes by predicting interaction rationality. Thus ThreatSniffer has a finer detection granularity than StreamSpot and Unicorn. ThreatSniffer performs entity embedding from file paths, interaction sequences, and the number distribution of interaction types, and uses a multi-head self-attention mechanism for semantic fusion. Furthermore, ThreatSniffer utilizes a heterogeneous graph neural network to incorporate the context into entity embeddings. Compared to existing APT detection methods using graph neural networks, ThreatSniffer understands entity semantics more comprehensively, thereby demonstrating better detection performance in Section 5.3. ThreatSniffer selectively samples the non-existent edges from the benign provenance graph as irrational interactions, enabling semi-supervised learning from attack-free audit logs. Consequently, compared to rule-based and supervised learning approaches, ThreatSniffer is more likely to detect zero-day vulnerabilities exploited in APT campaigns and achieves a higher recall.

3 Overview

3.1 Motivation

During the APT lifecycle, attackers typically exploit zero-day vulnerabilities to carry out attacks, stealthily infiltrate the target system, and generate only a few malicious system entities. Audit logs describe the interaction history of system entities. By connecting system entities, system entity interaction provenance graphs can describe system behavior at a fine-grained level. The example in Fig. 1 is the malicious part of a system entity interaction provenance graph. It demonstrates *nginx* being exploited to execute a malicious dropper file. In this attack, the *nginx* is exploited to drop a malicious executable file named *dropper* (③). Then *dropper* is executed via *shell* (④ to ⑥). Subsequently, the attacker communicates with the *dropper* process (⑦ and ⑧), controls it to conduct information gathering (⑩ and ⑪) and modify or read sensitive files (⑬ to ⑯). Sensitive information would be sent to the attacker via a temporary file.

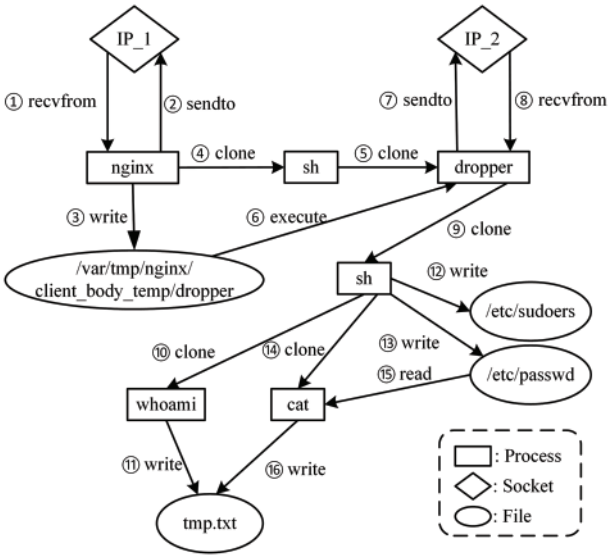

Figure 1: An example of the provenance graph

Malicious behaviors inevitably interact with the underlying operating system, which will be exposed to and captured in system audit logs. Thus no matter how stealthy and slow APT attacks

are, corresponding nodes and interactions can be found in the provenance graph. In recent years, researchers have been leveraging provenance graphs to detect and investigate APT attacks. Conducting attack detection and investigation based on provenance graphs presents the following two main challenges:

Challenge 1: Fine-grained APT detection. An ideal APT detection scheme should be able to pinpoint the system entities exploited during the attack execution. These fine-grained detection outcomes can substantially lighten the workload for security engineers conducting attack investigation. The key to this challenge lies in fully understanding the rich semantics in the provenance graph, which can greatly assist us in determining whether an entity has been exploited by attackers.

Challenge 2: Modeling normal behaviors from the attack-free audit logs. One characteristic of APT attacks is the utilization of zero-day vulnerabilities. Compared to rule-based and supervised learning approaches, anomaly detection has a higher likelihood of detecting zero-day vulnerabilities. The key to this challenge is how to design appropriate deep-learning tasks to distinguish between normal and malicious behavioral activities.

3.2 Approach Intuition

Based on careful observation and analysis of various provenance graphs containing APT activities, two key insights may be helpful for provenance-based APT detection. **The first insight is:** system entities in provenance graphs have different semantics in terms of file paths, interaction sequences, and the number distribution of interaction types. As for file paths, directory names at each level in the file path are crucial for understanding file semantics. System entities with similar file paths usually have similar functions. For example, in Fig. 1, */etc/sudoer* and */etc/passwd* are both system configuration files. For interaction sequences, a program tends to have a fixed behavior pattern. The *shell* process in Fig. 1 is often cloned from the user-level program and then executes system commands through sub-processes. For the number distribution of interaction types, different entity behaviors (considering network accesses and file I/O) lead to different numbers of various interaction types. **The second insight is:** the provenance graph context of a malicious process exists some conflicts. In the given contextual background, malicious processes will interact with the system entities that should not be invoked, making unexpected interactions appeared in the provenance graph. Take Fig. 1 as an example, the *cat* process should not write data to *tmp.txt* after reading the sensitive file */etc/passwd*. Therefore, it is possible to use an interaction prediction model to learn entity interaction context and identify the process nodes that are pertinent to potential attacks by predicting the interaction rationality, enabling fine-grained APT detection.

When there is no malicious activity in the system, the provenance graph constructed from system entities and interactions is defined as benign. When APT activity occurs, it would lead to some unexpected entity interactions. In such instances, the provenance graph constructed from system audit logs is regarded as suspicious. The suspicious provenance graph contains only a small number of system entities and interactions directly associated with the attack behavior. Unlike StreamSpot or Unicorn, which conduct APT detection at the graph-level without reporting specific attack entities, ThreatSniffer constructs benign and suspicious provenance graphs from audit logs, and aims to learn normal system entity interactions from the benign provenance graph and subsequently identify the processes exploited by attackers in the suspicious provenance graph.

3.3 ThreatSniffer Architecture

The architecture of ThreatSniffer, depicted in Fig. 2, is tailored to identify APT-exploited processes from system audit logs during attack investigation. It acquires semantic insights regarding system entities from diverse dimensions. Furthermore, based on the benign provenance graph, it employs an interaction prediction model to align with the system's normal interactions and identify irrational interactions. ThreatSniffer encompasses three key modules: Entity Embedding, Provenance Graph Construction, and Interaction Prediction.

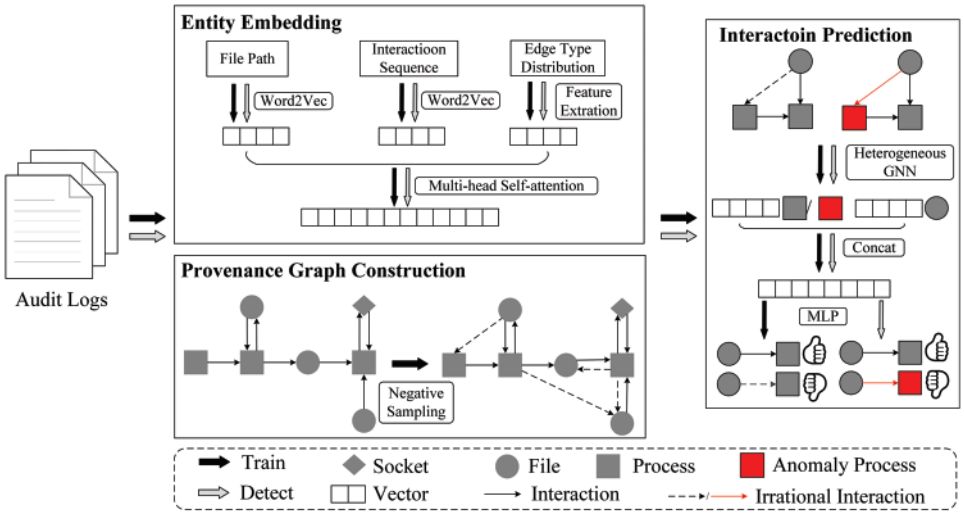

Figure 2: The architecture of ThreatSniffer

Entity Embedding. This module extracts file paths, interaction sequences, and the number distribution of interaction types from system audit logs. It then individually embeds entity semantics from these three dimensions. These embeddings are subsequently fused through a multi-head self-attention mechanism to yield initial node features within the provenance graph.

Provenance Graph Construction. This module obtains system entities and interactions from audit logs and constructs a provenance graph following the direction of information flow. The provenance graph contains rich contextual information about system entities. To train the model on benign data, this module also performs negative sampling on the benign provenance graph to characterize irrational entity interactions.

Interaction Prediction. This module uses a heterogeneous graph neural network to integrate entity embeddings and the provenance graph, and then learns the system's normal interaction behaviors by distinguishing between normal edges and irrational edges generated by negative sampling. When performing detection, this module identifies anomaly processes by predicting the rationality of system entity interactions.

For challenge 1, ThreatSniffer understands system entities from file paths, interaction sequences, and the number distribution of interaction types, and then uses a multi-head self-attention mechanism for semantic fusion. Besides, ThreatSniffer employs a heterogeneous graph neural network for understanding entity context. By fully leveraging the rich semantics of the provenance graph, ThreatSniffer can locate the anomaly processes by predicting the rationality of the system entity interaction.

For challenge 2, ThreatSniffer takes judging the existence of entity interactions in benign provenance as the training task for learning normal behaviors. More specifically, ThreatSniffer considers interactions observed in the benign provenance graph as benign instances while negatively sampling unobserved interactions as malicious.

3.4 Threat Model

The protection of system auditing modules or audit logs is beyond the scope of this paper. Same with the threat models from the previous provenance-based APT detection works [25–30,32,33], this paper assumes that system auditing modules (e.g., Auditd, ETW) fully record system interactions such as file operations, network accesses, etc., from the system kernel, and that the underlying operating system and system auditing modules will not suffer from kernel-level attacks since they are part of the Trusted Computing Base (TCB). Besides, this paper further assumes that system auditing modules employ a secure provenance storage system [34,35]. Attackers cannot undermine the integrity of provenance data by tampering with or deleting system audit logs. At last, ThreatSniffer does not consider hardware Trojan or side-channel attacks that are not visible in system audit logs, because their behavior can not be captured by system auditing modules.

4 Methodology

4.1 Entity Embedding

Graph neural networks pass, aggregate, and update node features on the graph, thereby the complex dependencies of the graph are incorporated into node features for subsequent tasks. Information-rich and discriminative node features are crucial for high-quality graph neural network models. Considering that system entity semantics are reflected in aspects like file paths, interaction sequences, and the number distribution of entity interaction types, ThreatSniffer separately performs entity embedding across these three dimensions and subsequently fuses these semantic representations.

File Path Embedding. Files of the same program are commonly situated in the same directory. Moreover, the folder names of different directories also convey specific meanings. For instance, whether it is a system directory or a program installation directory, the directory name *bin* in the path indicates that the folder contains executable files. Each system entity in the provenance graph, whether it is a file, a process, or a socket, is associated with a system file path. These paths contain important semantic insights about the system entity.

ThreatSniffer extracts file paths of all system entities from audit logs. These file paths are composed of multiple layers of directories. The same directory names tend to have the same semantics. Each file path is considered as a sentence, and each directory name is considered as a word. Based on the perspective that the directory order in a file path remains consistent and instances of polysemy are rare in file paths, ThreatSniffer uses the Skip-Gram-based Word2vec [36] algorithm to generate word embeddings for each directory name of file paths. Given a contextual window C, its goal is to maximize the probability of predicting the context around a given target word t, as shown in Eq. (1) [36].

$$probability = \sum_{t=1}^{V} \sum_{-C \le c \le C, c \ne 0} \log P\left(w_{t+c} \mid w_t\right) \tag{1}$$

C is the contextual window size, while w_t and w_{t+c} are the word embeddings of the target and contextual words. $P\left(w_{t+c} \mid w_t\right)$ is the conditional probability of generating the contextual word w_{t+c}

given the center word t, defined by the Softmax function, as shown in Eq. (2). V is the total number of words in the corpus.

$$P\left(w_{t+c} \mid w_t\right) = \frac{\exp\left(w_{t+c}^{\top} w_t\right)}{\sum_{i=1}^{V} \exp\left(w_i^{\top} w_t\right)} \tag{2}$$

The above file path embedding method assigns a word embedding to each directory name. Embeddings of directories or files with similar context are situated closer in the vector space, which is in line with our intuitive understanding of system entities. For instance, even though the entities */var/tmp/etilqs_MA815rf8hAKkd3W* and */var/tmp/etilqs_sebAB6ur3dkvhCa* have different file paths, they are both temporary system files, so their file names *etilqs_MA815rf8hAKkd3W* and *etilqs_MA815rf8hAKkd3W* have similar word embeddings. Note that ThreatSniffer doesn't treat entity paths as atomic individuals for embedding, it generates embeddings for each directory name in the file path and then obtains the vector embedding using a weighted averaging approach.

Interaction Sequence Embedding. Through a careful analysis of system audit logs, it was observed that some processes exhibit fixed behavioral patterns in the sequence of system entity interactions. For example, during IP address resolution, the system will read */run/resolvconf/resolv.conf* and */etc/hosts* in turn. To embed these semantics in system entity interaction sequences, ThreatSniffer first extracts the interaction sequences from audit logs. It then employs a word embedding model to gain the system entity vectors that contain behavioral pattern semantics from these interaction sequences.

ThreatSniffer handles system audit logs and extracts system entity interaction sequences in chronological order. A significant consideration is that programs often generate numerous repetitive entity interactions during network transfers or file I/O. To downsize these interactions, ThreatSniffer ignores the timestamps of entity interactions and further simplifies interactions into a triplet (*subject*, *object*, *relation*), where *subject* represents the initiator of the interaction (i.e., processes), *object* represents the target of the interaction (i.e., files and sockets), and *relation* represents the type of interaction (e.g., read and write). It only adds interaction to the interaction sequence when it appears for the first time or has not occurred recently. Moreover, some processes collaborate with sub-processes or other processes to accomplish tasks, and some processes have only a small number of system entity interactions (more than 20% of the processes in the Darpa TC3 Theia dataset have fewer than 5 entity interactions), indicating there are also fixed behavioral patterns among different processes. So the second consideration is preserving inter-process collaboration information without partitioning logs by different processes. This prevents both increased processing time and the loss of valuable inter-process collaboration information. Algorithm 1 delineates the procedure for extracting entity interaction sequences using a sliding-window mechanism, with each interaction sequence containing L system entity interactions, allowing for the overlap of neighboring interaction sequences.

Algorithm 1: Interaction sequence extraction

Input: Interactions stream $I = \{i_1, i_2, \ldots\}$, Max consecutive repeat times T, Sequence length limit L, Last sequence remain number N

Output: Interaction sequences $\mathbb{S} = \{S_1, S_2, \ldots\}$

1: $\mathbb{S} \leftarrow \varnothing, S \leftarrow [], lastPositionDict < K, V > \leftarrow < \varnothing, \varnothing >, position \leftarrow 0$

2: **for** $i \in I$ **do**

3: $position \leftarrow position + 1$

(Continued)

Algorithm 1 (continued)

```
    if i ∉ K then
      lastPosition ← position
    else
      lastPosition ← lastPositionDict < i >
    end if
    if position == lastPosition ∨ position > lastPosition + T then
      Append i to S
    end if
    Update < i, position > to lastPositionDict
    if S.length == L then
      Add S to 𝕊
      S ← S[−N :]
    end if
  end for
  return 𝕊
```

Because system entities are rarely polysemy, ThreatSniffer still uses Word2vec [36] as the sequence embedding model. It extracts interaction types and object entities from each triplet in interaction sequences and concatenates them together as a corpus to train the Word2vec model to obtain the vector embeddings of each system entity.

Number Distribution of Entity Interaction Types. The behavior of nodes in the provenance graph is reflected in the entity interactions connected to them. Different node behaviors lead to different number distributions of interaction types. Considering the example of ransomware and remote shell (as shown in Fig. 3), the ransomware process reads files and writes them to encrypted copies, and then erases the original files. The remote shell receives commands from an external IP, executes the corresponding commands, and then sends outcomes to the external IP. In addition, different types of nodes have different interactions, e.g., file read and write exclusively appear between processes and files rather than sockets. This difference in interaction type distribution can intuitively illustrate the diversity of entity types. So the number distribution of interaction types is extractd as part of entity features.

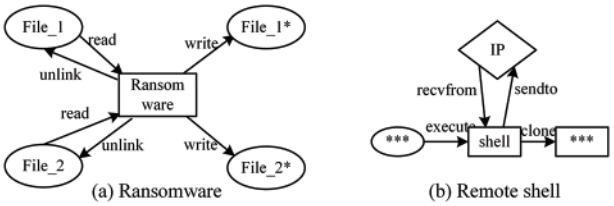

Figure 3: Examples of different processes

ThreatSniffer first counts the number of interaction types, denoted as $M = |\mathcal{X}_e|$. It then establishes a one-to-one mapping $\mathcal{M}_e: \mathcal{X}_e \rightarrow N$ to assign a unique integer from 0 to M for each interaction type. Subsequently, it employs a function $\mathcal{F}: V \rightarrow N^{2*M}$ to obtain entity features $\mathcal{F}(v) = [a_0, a_1, \ldots, a_{M-1}, a_M, a_{M+1}, \ldots, a_{M*2-1}]$, for each entity $\forall v \in V$. Here, a_i is computed as Eq. (3). Entity

features are $M * 2$ dimensions because the source and target nodes have different semantics [30].

$$a_i = \begin{cases} |\{e|e \in \text{In}(v), \mathcal{M}_e(edgeType(e)) = i\}|, i \in 0, \ldots, M-1 \\ |\{e|e \in \text{Out}(v), \mathcal{M}_e(edgeType(e)) + M = i\}|, i \in M, \ldots, 2 * M - 1 \end{cases} \tag{3}$$

Semantic Fusion. After gathering the d dimensional initial features of system entities from the above three key aspects: file paths, interaction sequences, and the number distribution of interaction types, to gain high-quality entity embeddings, ThreatSniffer employs the multi-head self-attention mechanism [37] to fuse semantics as the semantic augmentation layer. This layer captures high-level dependencies among different features. For each batch of input feature matrix X, ThreatSniffer multiplies it separately by parameter matrixes W_q, W_k, and W_v, resulting in $Q = \{q_1, q_2, \ldots, q_n\}$, $K = \{k_1, k_2, \ldots, k_n\}$, and $V = \{v_1, v_2, \ldots, v_n\}$, where n represents the number of attention heads. Then, it calculates the output values of each head using the scaled dot-product model as shown in Eq. (4) [37] and integrates the semantic information from all n subspaces (as in Eq. (5) [37]). W^0 is a learnable matrix. Finally, the outputs are subjected to layer normalization to obtain the ultimate entity embeddings.

$$head_i = \text{softmax}\left(\frac{q_i k_i^{\mathrm{T}}}{\sqrt{d/n}}\right) v_i \tag{4}$$

$$MultiHead(Q, K, V) = \text{concat}(head_1, \cdots, head_h) W^0 \tag{5}$$

4.2 Provenance Graph Construction

ThreatSniffer converts audit logs into a directed provenance graph with multiple types of edges and nodes. Each log entry of audit logs represents a system entity interaction and can be denoted as $(subject, object, relation, timestamp)$, where *subject* and *object* are system entities associated with the interaction, *relation* denotes the type of the interaction, and *timestamp* indicates the time when the entity interaction occurs. Based on the entity interactions recorded in audit logs, a provenance graph $G = (V, E, \mathcal{X}_v, \mathcal{X}_e, \mathcal{T}_e)$ is constructed. Here, V represents nodes in the provenance graph G, corresponding to system entities, and $\mathcal{X}_v = \{process, file, socket\}$ denotes the set of node types. E represents edges, corresponding to all system entity interactions, and $\mathcal{X}_e = \{execute, open, read, write, \ldots, execute\}$ denotes the set of edge types. $\mathcal{T}_e$ involves the chronological order of each edge. Since the provenance graph is required to be directed, the information flow directions of different interaction types are defined in Table 1 and used as the directions of the edges in the provenance graph. The provenance graph often includes redundant system entity interactions generated by network transfer or file I/O. To address this, the pruning approach described in Section 4.1 Interaction Sequence Embedding is adopted to trim the provenance graph. This reduces the complexity of model learning and attack investigation without losing any information about potential attacks.

Table 1: Direction of different interaction types

Subject	Object	Relation	Direction
Process	File	Execute	←
Process	File	Open	←

(Continued)

Table 1 (continued)

Subject	Object	Relation	Direction
Process	File	Read	←
Process	File	Write	→
Process	File	Unlink	→
Process	File	ModiyFileAttributes	→
Process	Socket	Connect	←
Process	Socket	RecvFrom	←
Process	Socket	RecvMsg	←
Process	Socket	ReadSocketParms	←
Process	Socket	SendTo	→
Process	Socket	SendMsg	→
Process	Socket	WriteSocketParms	→
Process	Process	Clone	→

After constructing a benign provenance graph from attack-free audit logs, ThreatSniffer performs negative sampling to learn the system's normal behavioral patterns from this benign graph. Furthermore, interactions observed in the benign provenance graph are considered benign instances, while interactions not observed are extracted as malicious instances. Due to the sparsity of the provenance graph, there is an extreme imbalance between interaction pairs and non-interaction pairs. It is infeasible to treat all unobserved interactions as malicious. Therefore, ThreatSniffer selectively samples the non-existent edges from the benign provenance graph as irrational interactions. The negative sampling procedure is delineated in Algorithm 2. Specifically, similar to the negative sampling methods in mainstream recommender systems [38,39], ThreatSniffer achieves negative sampling by replacing either the *subject* or *object* node with other nodes of the same node type. For each interaction in the benign provenance graph, ThreatSniffer performs negative sampling of $2K$ non-existent edges to create corresponding irrational interactions, where K interactions are generated by replacing *subject* nodes and the other by replacing *object* nodes.

Algorithm 2: Interaction negative sampling

Input: Benign provenance graph $G = (V, E, \mathcal{X}_v, \mathcal{X}_e, \mathcal{T}_e)$, Sample number K, Similarity threshold $\mathcal{T}$
Output: Negatively sampled graph G'

$E' \leftarrow \varnothing$
for $e(subject, relation, object) \in E$ **do**
  $sampleNumber \leftarrow 2 * K$
  $subjectType \leftarrow \mathcal{X}_e\,(subject)$
  $nodes \leftarrow \{node | node \in V, \mathcal{X}_v\,(node) = subjectType\}$
  while $sampleNumber > K$ **do**
    $v' \leftarrow getOneNode(nodes, asSubject)$
    $e \leftarrow (v', relation, object)$
    if $calculateSimilarity(subject, v') < \mathcal{T} \wedge e' \notin E$ **then**
      Add e to E'

(Continued)

Algorithm 2 (continued)

```
            sampleNumber ← sampleNumber − 1
        end if
    end while
    objectType ← 𝒳_e (object)
    nodes ← {node|node ∈ V, 𝒳_v (node) = objectType}
    while sampleNumber > 0 do
        v′ ← getOneNode(nodes, asObject)
        e ← (subject, relation, v′)
        if calculateSimilarity(v′, object) < 𝒯 ∧ e′ ∉ E then
            Add e to E′
            sampleNumber ← sampleNumber − 1
        end if
    end while
end for
G′ ← (V, E, E′, 𝒳_v, 𝒳_e, 𝒯_e)
return G′
```

Since it is impossible to treat all unobserved interactions as malicious instances, it is crucial to include as much information as possible in a small number of negative sampling edges. In lines 7 and 17 of Algorithm 2, ThreatSniffer adopts a degree-based sampling method [40]. This method calculates the probability of a node being selected for replacement based on the node's out-degree or in-degree (when the node is a source node or a destination node) in the provenance graph. Nodes with higher degrees are more likely to be sampled for constructing negative samples. The fundamental idea behind this strategy is that if a widely-used system entity has not interacted with a particular program, there is a high probability that the program will not interact with this entity, thus learning more individualized characteristics about this program.

There are a large number of semantically similar system entities (e.g., */var/tmp/etilqs_MA815rf8hAKkd3W* and */var/tmp/etilqs_sebAB6ur3dkvhCa* both correspond to temporary files of Sqlite). When performing negative sampling for a given system entity interaction, these semantically similar system entities should not be used to generate irrational samples for that interaction. Therefore, ThreatSniffer calculates the similarity score between the replacement node v' and the original node v involved in the interaction, as shown in Eq. (6). $\vec{v'}$ and $\vec{v}$ are initial features of these two entity nodes. Only when the similarity is below a certain threshold will the node be used to construct an irrational sample for that interaction.

$$similarity\left(\vec{v'}, \vec{v}\right) = \frac{\vec{v'} \cdot \vec{v}}{\left|\vec{v'}\right| \times \left|\vec{v}\right|} = \frac{\sum_{i=1}^{n} v'_i \cdot v_i}{\sqrt{\sum_{i=1}^{n} v'^2_i}\sqrt{\sum_{i=1}^{n} v^2_i}} \tag{6}$$

4.3 Interaction Prediction

Graph neural network has a powerful cognitive ability to handle graph data. Recent research has extensively utilized graph neural networks to carry out provenance graph-oriented attack detection. In this interaction prediction model, ThreatSniffer utilizes a heterogeneous graph neural network

for integrating entity embeddings (Section 4.1) and the provenance graph (Section 4.2). The model incorporates contextual information in the provenance graph into entity embeddings and then examines the rationality of system entity interactions to detect stealthy APT-exploited processes.

The provenance graph contains rich contextual information about system runtime. For example, *Firefox (Process)* → */home/admin/clean (File)* → *Clean (Process)* illustrates the steps of downloading and executing a program. Nevertheless, the system entity embedding approach introduced in Section 4.1 cannot adequately capture these causal dependencies. Graph neural network learns these complex dependencies in a provenance graph by aggregating and updating node embeddings along edges on the graph, achieving an effective integration of entity embeddings and provenance graph. Specifically, for a given system entity v, ThreatSniffer adopts the heterogeneous graph neural network as the convolutional layer of interaction prediction model. This layer aggregates embeddings of one-hop neighbors (aka ego network [41]) and updates the vector representation of entity v. This new vector contains the entity's initial embedding and causal dependencies. ThreatSniffer learns rich context in the provenance graph by stacking multiple convolutional layers.

The following Fig. 4 shows the procedure of a convolutional layer, which aggregates and updates entity embedding in a provenance graph. In the rest of the narrative, the notation u and notation v represent the source node and destination node, respectively.

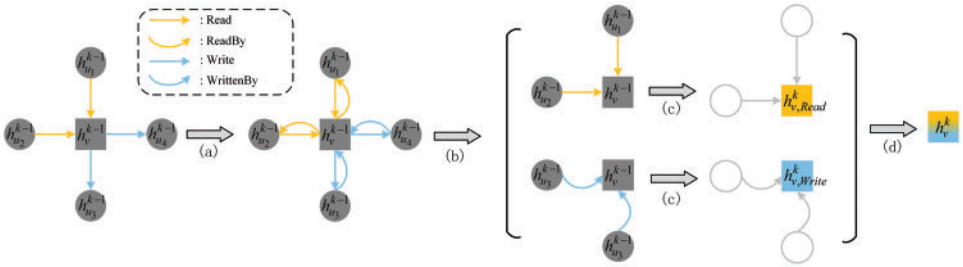

Figure 4: Heterogeneous graph convolutional layer

To learn the direction of edges in the provenance graph, graph neural networks take a node v as the destination node, and aggregate embeddings from source nodes. This loses the contextual information when node v is the source node. Therefore, ThreatSniffer adds corresponding reverse edges for each interaction type in the provenance graph. These reverse edges enable the aggregation to retain the complete context of node v. As step (a) in Fig. 4, ThreatSniffer adds reverse edges such as *ReadBy* and *WrittenBy* for the *Read* and *Write* interactions.

During the convolution of each layer, since there are many different interaction types in the provenance graph, ThreatSniffer employs the heterogeneous graph neural network to handle each interaction type separately, as shown in Eq. (7). In this formula, f_r is the convolutional layer applied for each interaction type r, and MAX is the aggregation function.

$$h_v^{(k+1)} = \underset{r\in\mathcal{R}, v\in dst(r)}{MAX}\left(f_r\left(g_r, h_{ru}^k, h_{rv}^k\right)\right) \tag{7}$$

As step (b) in Fig. 4, taking node v as the destination node, ThreatSniffer extracts subgraphs g_r for each interaction type from the 1-hop neighbors of node v. In step (c), it convolves each interaction type separately using Eq. (8) [31] as f_r to obtain new node embeddings under different interaction type views, e.g., $h_{v,Read}^{k-1}$ and $h_{v,Write}^{k-1}$. Then each new embedding of node v has incorporated its neighbor

information of the corresponding interaction type.

$$h_{v,r}^{k} = \sigma\left(\mathbf{W} \cdot \text{MEAN}\left(\left\{h_{v}^{k-1}\right\} \cup \left\{h_{u}^{k-1}, \forall u \in \mathcal{N}_{r}(v)\right\}\right)\right) \tag{8}$$

$\mathcal{N}_r(v)$ represents neighboring nodes of v with interaction type r. $h_v^k \in \mathbb{R}^d$ denotes the d dimensional vector representation of node v in the No. k propagation layer. This vector aggregates the contextual information of the k-hop neighbors of node v. Similarly, h_v^{k-1} is the vector representation of node v in the No. $k-1$ propagation layer, and h_v^0 is the entity embedding generated by Section 4.1. The convolutional layer integrates the embeddings of each node in $\mathcal{N}_r(v)$ with the embedding of node v itself, takes their average, multiplies by a learnable parameter W, and subsequently employs an activation function σ to update the node embedding. ThreatSniffer chooses *elu* [42] as the activation function here because it does not suffer from neuron death.

In step (d), ThreatSniffer adopts max pooling as the aggregation function MAX to fuse these newly obtained node embeddings. Compared with other aggregation functions, max pooling tends to learn representative features, thereby enhancing the model's expressive capacity.

Once the update for entity embeddings is completed, to predict whether an interaction between two entities is likely to occur, threatSniffer concats their vectors and uses a three-layer Multi-Layer Perceptron (MLP) to compute the interaction rationality score.

4.4 Training and Detection

In short, ThreatSniffer embeds system entities from three aspects and fuses these embeddings using a multi-head self-attention mechanism. Moreover, it constructs provenance graphs from system audit logs and samples non-existent edges as irrational interactions. It then detects anomaly processes with the interaction prediction model. The training and detection procedures are described as follows.

In the training phase, the input is attack-free logs and the negatively sampled benign provenance graph. ThreatSniffer's training goal is to distinguish between original benign interactions and irrational interactions produced through negative sampling on the benign provenance graph. Specifically, ThreatSniffer extracts entity file paths and interaction sequences to gain entity embeddings with Word2vec. After getting file path embeddings, interaction sequence embeddings, and the number distribution of entity interaction types, ThreatSniffer uses a multi-head self-attention mechanism to fuse semantics. Based on the fused entity embeddings and the negatively sampled benign provenance graph, ThreatSniffer aggregates and updates entity embeddings using a heterogeneous graph neural network, and then computes the rationality of generating interaction between two given entities. Subsequently, ThreatSniffer calculates the loss and conducts back-propagation to fit normal system entity interactions without the need for attack samples. Note that the irrational interactions generated via negative sampling are only used for model training, meaning that entity embeddings will not be aggregated or updated along these negatively sampled edges. The multi-head self-attention mechanism and interaction prediction model are optimized by minimizing cross-entropy through back-propagation and gradient descent. Since for each interaction in the provenance graph, ThreatSniffer generates $2K$ non-existent edges as irrational interactions, the training data suffers from sample imbalance. To mitigate this, ThreatSniffer uses the weighted cross-entropy loss, as shown in Eq. (9), to balance the information learned from the benign and irrational samples. The weights for benign and malicious samples are set to $2K$:1. Output of the training phase is four models, i.e., Word2vec for file path, Word2vec for interaction sequence, multi-head self-attention, and heterogeneous graph neural network.

$$L_{wce} = -\frac{1}{N}\sum_{i=1}^{N}[2K \cdot y_i \cdot \log(p_i) + (1 - y_i) \cdot \log(1 - p_i)] \tag{9}$$

In the detection phase, input audit logs may contain APT activities and only a limited number of system entities and interactions are related to the attack. The flowchart of detection is depicted in Fig. 5. Firstly, ThreatSniffer will extract and embed file paths and interaction sequences with trained Word2vec models. Additionally, a suspicious provenance graph will be constructed. The graph makes it easy to count the number distribution of entity interaction types. Secondly, the file path embeddings, interaction sequence embeddings, and number distribution of entity interaction types are fed into the trained multi-head self-attention model for semantic fusion. At last, the new fused entity embeddings and the suspicious provenance graph are integrated by the trained heterogeneous graph neural network to calculate the interaction rationality of two entities involved in each edge, which belongs to the input provenance graph. When the rationality score is smaller than the predefined threshold, the corresponding interaction is considered unexpected. Since at least one of the two entities involved in an interaction is a process, ThreatSniffer takes the process as the initiator of the unexpected interaction, thus the process is regarded as the result of anomaly detection. This result can be provided to security engineers for further investigation.

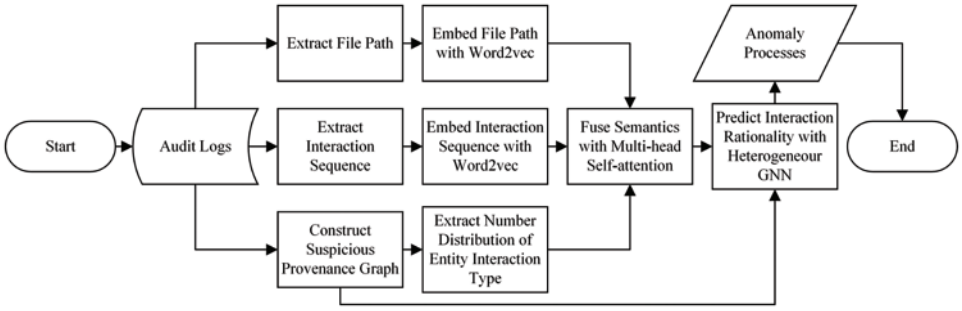

Figure 5: The flowchart of detection phase

5 Evaluation

5.1 Dataset and Experimental Setup

This paper evaluates the effectiveness of ThreatSniffer on the DARPA TC3 Theia dataset [43]. The public APT dataset is a set of Linux system entity interaction logs collected during the third red-team *vs*. blue-team adversarial engagement in April 2018. Red-team attackers used Firefox backdoors, browser extensions, and phishing email attachments to carry out APT campaigns during the engagement. The interaction logs and ground truth are publicly available [44]. The GroundTruth file records the tools and attack steps exploited by attackers. These attack details make it easy to classify benign and attack activities according to the attack details documented in the GroundTruth file. Processes associated with attack activities are marked as *Positive*, and processes occurring during attack activities but unrelated to the attack are marked as *Negative*. The benign provenance graph is constructed based on benign activities and benign interactions are divided into training and validation sets at a 9:1 ratio. The suspicious provenance graph is constructed based on attack activities.

All experiments are conducted on a server with an Intel Xeon Platinum 8255C CPU (12 $*$ 2.50 GHz), 43 GB of physical memory, an Nvidia RTX 3090 (24 GB) GPU, and an operating system

of Ubuntu 20.04. ThreatSniffer is implemented with Python 3.8, Pytorch 1.10 [45], and Deep graph libaray 0.9.1 [46]. Entity embeddings are vectorized in 128 dimensions, where 50 dimensions are file path embeddings, 50 dimensions are interaction sequence embeddings, and the other 28 dimensions are number distributions of interaction types. Self-attention mechanism head count is set to 16. The number of convolutional layers to learn interaction contextual information is 2. The adam optimizer [47] is adopted for model training with a learning rate of 0.001 and a fixed batch size of 2048. ThreatSniffer is trained for 60 epochs and the training is terminated when the loss on the validation set doesn't decrease for 5 consecutive epochs. The dropout [48] technique is adopted to address the over-fitting problem and the dropout rate is set to 0.2.

The metrics Precision, Recall, and F1-score are used to evaluate the effectiveness of ThreatSniffer detecting APT-exploited processes. The precision represents the proportion of processes predicted by ThreatSniffer as anomalies that are truly related to APT campaigns. The recall represents the proportion of all processes related to APT campaigns that are successfully detected by ThreatSniffer. The F1-score calculates the harmonic mean of Precision and Recall, providing a balanced metric.

5.2 Impact of Negative Sampling Number

ThreatSniffer performs negative sampling on the benign provenance graph to characterize irrational entity interactions. The negative sampling number K indicates that each benign interaction is understood through $2K$ non-existent edges. This subsection vary the key parameter K from 5 to 25 to investigate its impact on detection performance. Experimental results are shown in Fig. 6.

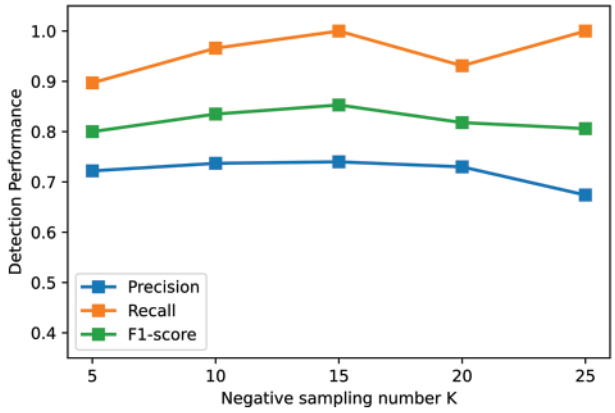

Figure 6: Impact of parameter K

As the negative sampling number K increases, ThreatSniffer gains a better understanding of benign interactions through more irrational edges. Recall increases with the number of negative samples in the initial stages. At $K = 15$, ThreatSniffer can detect all APT-exploited processes. Subsequently, an excessive number of irrational edges leads to overfitting, which results in more false positives and reduces the model's precision. Considering both precision and recall, the negative sampling number of $K = 15$ is chosen for other experiments, where the F1-score is at its peak.

5.3 Comparison Study

Different from existing graph-level APT detection studies such as StreamSpot [13] and Unicorn [14], ThreatSniffer is a node-level detector that detects anomaly processes related to APT campaigns. Unfortunately, there are only a few fine-grained APT detection studies with available source code,

i.e., ShadeWatcher [25], APT-KGL [27], and ThreaTrace [30]. ShadeWatcher is not fully open-source. Its key component is proprietary. So this subsection compares ThreatSniffer with ThreaTrace and APT-KGL to evaluate its detection effectiveness. ThreaTrace designs a GraphSage-based multi-model framework. It takes the node type as the label of entity to learn different kinds of benign nodes in the benign provenance graph. APT-KGL conducts supervised learning on the provenance graph and then detects APT-exploited processes. It defines meta-paths and then applies meta-path-based heterogeneous graph attention network [49] to learn context and embed system entity. Their open-source code [50,51] are used to train models to detect APT-exploited processes. Since ThreatSniffer only reports processes related to APT campaigns while ThreaTrace reports entities, for a fair comparison, the process entities are filtered from ThreaTrace anomaly detection results to compute evaluation metrics. The experimental results are shown in Table 2.

Table 2: Results of the comparison experiment

Method	Precision	Recall	F1-score
ThreaTrace	0.294	0.345	0.317
APT-KGL	0.683	0.966	0.8
ThreatSniffer	0.744	1	0.853

ThreatSniffer shows better detection performance than ThreaTrace. ThreaTrace is almost unable to detect APT-exploited processes[1]. Its basic idea for APT detection is that the predicted node types of anomaly entities will deviate from their actual types. This idea does not align with our intuition about APT campaigns. Processes associated with attack activities inevitably generate unexpected system entity interactions, but these interactions do not cause process nodes to be predicted as other types of nodes, e.g., a malicious process reading and leaking a sensitive file will not cause the node to be recognized as a socket or file entity. In addition, ThreaTrace only uses number distributions of interaction types as entity initial features. Its GraphSage-based multi-model framework does not take into account the semantic differences of various interaction types in the provenance graph. As a result, ThreaTrace is hard to comprehensively understand system behavior.

Compared to APT-KGL, ThreatSniffer shows better detection performance. ThreatSniffer performs anomaly-based detection. It fits normal system activities and treats deviations from normal activities as anomalies. Given that APTs are likely to involve unknown attacks, recent research (e.g., ShadeWatcher [25], ProGrapher [29], Kairos [33]) suggests that anomaly-based detection is relatively suitable for the scenario of APT detection. This enables ThreatSniffer to identify all attack activities and achieves a higher recall. APT-KGL also employs a heterogeneous graph neural network to consider the heterogeneous characteristics of provenance graphs. However, it only relies on the provenance graph to obtain entity embeddings, ignoring the rich semantics of system entities in aspects such as file paths, interaction sequences, etc. ThreatSniffer gets system entity embeddings from multiple aspects and then utilizes a multi-head self-attention mechanism for semantic fusion. Thus ThreatSniffer has a more comprehensive grasp of entity semantics. Entity embeddings with rich semantics enable deep learning models to distinguish benign behaviors from attack activities more accurately [33,52,53]. This is the key factor contributing to ThreatSniffer's higher precision.

[1]ThreaTrace labels the nodes in GroundTruth file and their 2-hop ancestors and descendants as anomalies, even if these nodes are not related to the attack [33]. That is why our reproduction results for ThreaTrace are worse than those in that paper.

5.4 Ablation Study

Feature Ablation. ThreatSniffer understands system entities from three aspects: File Paths (Feat1), Interaction Sequences (Feat2), and Number Distribution of Interaction Types (Feat3). This experiment evaluates their contributions to entity semantics separately. Specifically, the three embedding modules aforementioned in Section 4.1 are individually removed to validate the effectiveness of different features. Results as shown in Table 3 and Fig. 7.

Table 3: Results of feature ablation experiment

Feature	Precision	Recall	F1-score
Feat1+Feat2	0.676	0.793	0.73
Feat1+Feat3	0.477	0.724	0.575
Feat2+Feat3	0.641	0.862	0.735
Feat1+Feat2+Feat3	0.744	1	0.853

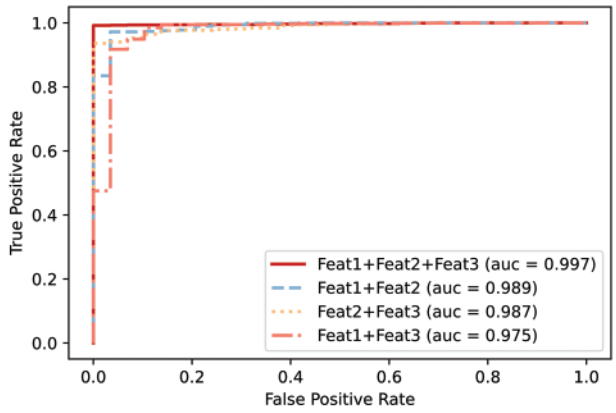

Figure 7: ROC curves of feature ablation experiment

The results indicate that removing any one of the features, i.e., File Paths, Interaction Sequences, and Number Distribution of Interaction Types, leads to varying degrees of decline in detection performance. This suggests that all features contribute positively to the APT-exploited process detection task. The detection performance of ThreatSniffer decreases most when the features in interaction sequences are removed. This is because understanding system entity interaction behavior requires not only knowing the entities involved in interactions but also comprehending the temporal order of interaction sequences. The program behavioral patterns involved in interaction sequences are crucial for understanding interaction behavior. Removing the features in file paths or the number distribution of interaction types has a comparatively minor impact on the model's detection effectiveness. This indicates the semantics in these two aspects only play a secondary role in understanding entity interaction behavior. In some entity attributes such as file paths, APT-exploited entities do not exhibit obvious distinctions from regular system entities, highlighting the stealthy of APT campaigns.

Module Ablation. ThreatSniffer employs a multi-head self-attention mechanism for semantic fusion and conducts negative sampling to extract non-existing edges as irrational interactions. It then adopts a heterogeneous graph neural network to learn the entity interaction context in a provenance

graph and finally uses an MLP to predict the interaction rationality of two given system entities. This experiment removes or replaces each module of ThreatSniffer separately to verify their effectiveness. Each comparison model for module ablation is designed as follows, and the experimental results are shown in Fig. 8.

Model-1: It concats entity initial embeddings together instead of using the multi-head self-attention mechanism for semantic fusion.

Model-2: It uses the single-head self-attention mechanism for semantic fusion.

Model-3: It samples irrational interactions according to a uniform distribution instead of the degree-based sampling.

Model-4: It directly feeds entity embeddings into the three-layer MLP without using the graph neural networks to understand entity interaction context.

Model-5: It uses a dot product to predict the entity interaction rationality instead of MLP.

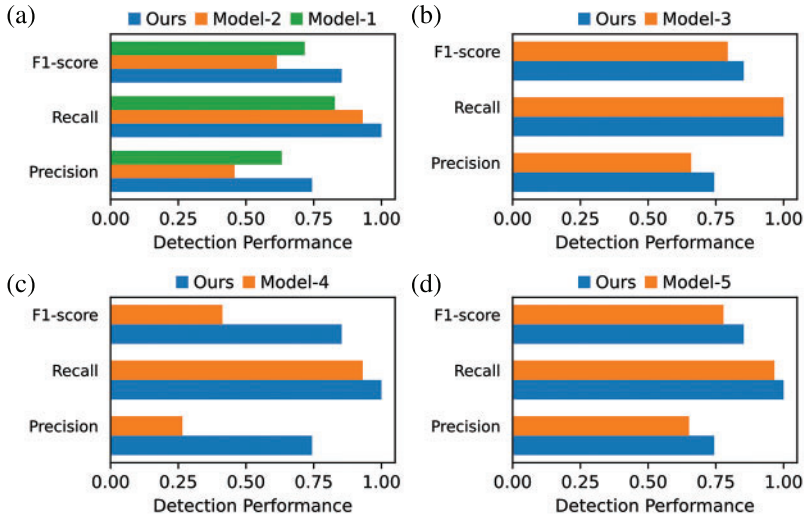

Figure 8: Results of module ablation experiment

To verify the effectiveness of adopting the multi-head self-attention mechanism for semantic fusion, the multi-head self-attention module is replaced with plainly concat (Model-1) or single-head self-attention (Model-2). As shown in Fig. 8a, when the self-attention mechanism is removed or the number of attention heads is reduced, the detection performance decreases, indicating that the multi-head self-attention mechanism can learn the high-level dependencies between different features.

To verify the impact of the degree-based negative sampling component, the degree-based sampling is replaced with random negative sampling (Model-3). The experimental results are shown in Fig. 8b, the model is still able to find all processes associated with APT campaigns, indicating that attack activities do generate some unexpected system entity interactions. These irrational interactions conflict with their contextual background. This characteristic of the attack activities can be captured by sampling non-existent edges as irrational interactions. However, random negative sampling makes lower precision, i.e., Model-3 produces more false positives since degree-based negative sampling can learn more individualized characteristics of entities, enabling ThreatSniffer to have a more accurate understanding of normal system entity interactions.

To validate the necessity of provenance graph contextual information for understanding system entity interactions, the entity embeddings are directly fed into MLP without the graph neural network aggregating the entity interaction context (Model-4). As shown in Fig. 8c, the decline in detection performance after removing the graph neural network is obvious. Because the provenance graph contains the interaction types and causal dependencies between system entities, which intuitively describes the behavior of system entities. The interaction context in provenance graph is vital for understanding entity interaction. This is in line with the experience in the field of NLP, where context enables a better understanding of the current word.

To verify the effectiveness of using MLP to predict entity interaction rationality, MLP is replaced with a dot product (Model-5). The results in Fig. 8d indicate that MLP yields better detection performance, as the MLP possesses stronger expressive capabilities compared with the dot product.

6 Conclusion and Future Work

This paper introduced an APT-exploited process detection approach called ThreatSniffer. It embeds and fuses system entity semantics from three aspects: file paths, interaction sequences, and the number distribution of interaction types, then employs a semi-supervised interaction prediction model for detecting anomaly process. Based on the above design, ThreatSniffer achieves fine-grained APT detection. Evaluation results demonstrate that ThreatSniffer outperforms other node-level APT detection methods. ThreatSniffer can work as a part of SIEM, and point out specific anomaly system entities to speed up attack investigation or threat hunting. Compared to graph-level anomaly detection, fine-grained results and their context are easier to correlate with IOC or other threat intelligence, thus reducing the manual efforts of security analysis.

One limitation of anomaly detection models is false positives. Typically, security engineers need to consult a large amount of reference material to confirm false positives as benign. Recently, large language models (LLM) have demonstrated remarkable advantages in knowledge integration and utilization. LLM might be a new way to assist security engineers in analyzing alerts. Our future work plans to utilize common LLM [54] or security-specific LLM [55] as auxiliary tools for security engineers analyzing the causes of alerts. Another limitation is data poison. ThreatSniffer requires attack-free audit logs as the dataset for training. If attackers poison the training data to include malicious activities, ThreatSniffer will fail to detect such attacks. In practice, audit logs that have been checked by security engineers can be used.

Acknowledgement: The authors would like to thank the associate editor and the anonymous reviewers for their insightful comments that improved this paper.

Funding Statement: This work was supported by the National Natural Science Foundation of China (Nos. U19A2081, 62202320), the Fundamental Research Funds for the Central Universities (Nos. 2022SCU12116, 2023SCU12129, 2023SCU12126), the Science and Engineering Connotation Development Project of Sichuan University (No. 2020SCUNG129) and the Key Laboratory of Data Protection and Intelligent Management (Sichuan University), Ministry of Education.

Author Contributions: The authors confirm contribution to the paper as follows: Study conception and design: B. Luo, L. Chen, Y. Luo. Experiment and interpretation of results: B. Luo. Manuscript preparation: B. Luo, S. Ruan. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: All public dataset sources are as described in the paper.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] W. U. Hassan, S. Guo, D. Li, Z. Chen, K. Jee *et al.,* "NODOZE: Combatting threat alert fatigue with automated provenance triage," in *Proc. of the 26th Network and Distributed Systems Security Symp. (NDSS)*, San Diego, CA, USA, pp. 1–15, 2019.

[2] M. A. Inam, Y. Chen, A. Goyal, J. Liu, J. Mink *et al.,* "SoK: History is a vast early warning system: Auditing the provenance of system intrusions," in *Proc. of the 44th IEEE Symp. on Security and Privacy (SP)*, San Francisco, CA, USA, pp. 2620–2638, 2023.

[3] "SIEM rules ignore bulk of MITRE ATT&CK framework, placing risk burden on users," [Online]. Available: www.scmagazine.com/news/siem-rules-ignore-bulk-of-mitre-attck-framework-placing-risk-burden-on-users (accessed on 15/11/2023).

[4] M. N. Hossain, S. M. Milajerdi, J. Wang, B. Eshete, R. Gjomemo *et al.,* "SLEUTH: Real-time attack scenario reconstruction from cots audit data," in *Proc. of the 26th USENIX Security Symp.*, Vancouver, BC, Canada, pp. 487–504, 2017.

[5] M. N. Hossain, S. Sheikhi and R. Sekar, "Combating dependence explosion in forensic analysis using alternative tag propagation semantics," in *Proc. of the 41st IEEE Symp. on Security and Privacy (SP)*, San Francisco, CA, USA, pp. 1139–1155, 2020.

[6] C. Xiong, T. Zhu, W. Dong, L. Ruan, R. Yang *et al.,* "CONAN: A practical real-time apt detection system with high accuracy and efficiency," *IEEE Transactions on Dependable and Secure Computing (TDSC)*, vol. 19, no. 1, pp. 551–565, 2020.

[7] S. M. Milajerdi, R. Gjomemo, B. Eshete, R. Sekar, V. N. Venkatakrishnan *et al.,* "HOLMES: Real-time APT detection through correlation of suspicious information flows," in *Proc. of the 40th IEEE Symp. on Security and Privacy (SP)*, San Francisco, CA, USA, pp. 1137–1152, 2019.

[8] W. U. Hassan, A. Bates and D. Marino, "Tactical provenance analysis for endpoint detection and response systems," in *Proc. of the 41st IEEE Symp. on Security and Privacy (SP)*, San Francisco, CA, USA, pp. 1172–1189, 2020.

[9] T. Zhu, J. Yu, T. Chen, J. Wang, J. Ying *et al.,* "APTSHIELD: A stable, efficient and real-time APT detection system for linux hosts," in *IEEE Transactions on Dependable and Secure Computing (TDSC)*, vol. 20, no. 6, pp. 5247–5264, 2023.

[10] S. M. Milajerdi, B. Eshete, R. Gjomemo and V. N. Venkatakrishnan, "POIROT: Aligning attack behavior with kernel audit records for cyber threat hunting," in *Proc. of the 26th ACM SIGSAC Conf. on Computer and Communications (CCS)*, London, UK, pp. 1795–1812, 2019.

[11] R. Wei, L. Cai, L. Zhao, A. Yu and D. Meng, "DeepHunter: A graph neural network based approach for robust cyber threat hunting," in *Proc. of the 17th EAI Int. Conf., SecureComm 2021*, pp. 3–24, 2021.

[12] P. Gao, F. Shao, X. Liu, X. Xiao, Z. Qin *et al.,* "Enabling efficient cyber threat hunting with cyber threat intelligence," in *Proc. of the 37th Int. Conf. on Data Engineering (ICDE)*, pp. 193–204, 2021.

[13] E. Manzoor, S. M. Milajerdi and L. Akoglu, "Fast memory-efficient anomaly detection in streaming heterogeneous graphs," in *Proc. of the 22nd ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD)*, San Francisco, CA, USA, pp. 1035–1044, 2016.

[14] X. Han, T. Pasquier, A. Bates, J. Mickens and M. Seltzer, "UNICORN: Runtime provenance-based detector for advanced persistent threats," in *Proc. of the 27th Network and Distributed Systems Security Symp. (NDSS)*, San Diego, CA, USA, pp. 1–19, 2020.

[15] Y. Xie, D. Feng, Y. Hu, Y. Li, S. Sample *et al.,* "Pagoda: A hybrid approach to enable efficient real-time provenance based intrusion detection in big data environments," *IEEE Transactions on Dependable and Secure Computing (TDSC)*, vol. 17, no. 6, pp. 1283–1296, 2018.

[16] Y. Xie, Y. Wu, D. Feng and D. Long, "P-Gaussian: Provenance-based gaussian distribution for detecting intrusion behavior variants using high efficient and real time memory databases," *IEEE Transactions on Dependable and Secure Computing (TDSC)*, vol. 18, no. 6, pp. 2658–2674, 2019.
[17] Q. Wang, W. U. Hassan, D. Li, K. Jee, X. Yu *et al.,* "You are what you do: Hunting stealthy malware via data provenance analysis," in *Proc. of the 27th Network and Distributed Systems Security Symp. (NDSS)*, San Diego, CA, USA, pp. 1–17, 2020.
[18] A. Alsaheel, Y. Nan, S. Ma, L. Yu, G. Walkup *et al.,* "ATLAS: A sequence-based learning approach for attack investigation," in *Proc. of the 30th USENIX Security Symp.*, pp. 3005–3022, 2021.
[19] M. Zipperle, F. Gottwalt, E. Chang and T. Dillon, "Provenance-based intrusion detection systems: A survey," *ACM Computing Surveys*, vol. 55, no. 7, pp. 1–36, 2022.
[20] W. Jiang, "Graph-based deep learning for communication networks: A survey," *Computer Communications*, vol. 185, no. 1, pp. 40–54, 2022.
[21] C. D. Xuan, H. D. Nguyen and M. H. Dao, "APT attack detection based on flow network analysis techniques using deep learning," *Journal of Intelligent & Fuzzy Systems*, vol. 39, no. 3, pp. 4785–4801, 2020.
[22] Z. Xu, P. Fang, C. Liu, X. Xiao, Y. Wen *et al.,* "DEPCOMM: Graph summarization on system audit logs for attack investigation," in *Proc. of the 43rd IEEE Symp. on Security and Privacy (SP)*, San Francisco, CA, USA, pp. 540–557, 2022.
[23] J. Zeng, Z. L. Chua, Y. Chen, K. Ji, Z. Liang *et al.,* "WATSON: Abstracting behaviors from audit logs via aggregation of contextual semantics," in *Proc. of the 28th Network and Distributed Systems Security Symp. (NDSS)*, pp. 1–18, 2021.
[24] A. Bordes, N. Usunier, A. G. Duran, J. Weston and O. Yakhnenko, "Translating embeddings for modeling multirelational data," in *Proc. of the 27th Advances in Neural Information Processing Systems (NIPS)*, Lake Tahoe, Nevada, USA, pp. 2787–2795, 2013.
[25] J. Zeng, X. Wang, J. Liu, Y. Chen, Z. Liang *et al.,* "SHADEWATCHER: Recommendation-guided cyber threat analysis using system audit records," in *Proc. of the 43rd IEEE Symp. on Security and Privacy (SP)*, San Francisco, CA, USA, pp. 489–506, 2022.
[26] N. Yan, Y. Wen, L. Chen, Y. Wu, B. Zhang *et al.,* "DEEPRO: Provenance-based apt campaigns detection via gnn," in *Proc. of the 21st IEEE Int. Conf. on Trust, Security and Privacy in Computing and Communications (TrustCom)*, Wuhan, China, pp. 747–758, 2022.
[27] T. Chen, C. Dong, M. Lv, Q. Song, H. Liu *et al.,* "APT-KGL: An intelligent apt detection system based on threat knowledge and heterogeneous provenance graph learning," in *IEEE Transactions on Dependable and Secure Computing (TDSC)*, pp. 1–15, 2022.
[28] J. Liu, J. Yan, Z. Jiang, X. Wang and J. Jiang, "A graph learning approach with audit records for advanced attack investigation," in *Proc. of the 2022 IEEE Global Communications Conf. (GLOBECOM)*, Rio de Janeiro, Brazil, pp. 897–902, 2022.
[29] F. Yang, J. Xu, C. Xiong, Z. Li and K. Zhang, "PROGRAPHER: An anomaly detection system based on provenance graph embedding," in *Proc. of the 32nd USENIX Security Symp.*, Anaheim, CA, USA, pp. 4355–4372, 2023.
[30] S. Wang, Z. Wang, T. Zhou, H. Sun, X. Yin *et al.,* "THREATRACE: Detecting and tracing host-based threats in node level through provenance graph learning," *IEEE Transactions on Information Forensics and Security (TIFS)*, vol. 17, no. 1, pp. 3972–3987, 2022.
[31] W. L. Hamilton, R. Ying and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. of the 31st Advances in Neural Information Processing Systems (NIPS)*, Long Beach, CA, USA, pp. 1024–1034, 2017.
[32] A. Gehani and D. Tariq, "SPADE: Support for provenance auditing in distributed environments," in *Proc. of the 13th Int. Middleware Conf.*, Montreal, QC, Canada, pp. 101–120, 2012.
[33] Z. Cheng, Q. Lv, J. Liang, Y. Wang, D. Sun *et al.,* "KAIROS: Practical intrusion detection and investigation using whole-system provenance," arXiv preprint arXiv:2308.05034, 2023.
[34] R. Hasan, R. Sion and M. Winsle, "Preventing history forgery with secure provenance," *ACM Transactions on Storage*, vol. 5, no. 4, pp. 1–43, 2009.

[35] A. Bates, D. Tian, K. R. B. Butler and T. Moyer, "Trustworthy whole-system provenance for the linux kernel," in *Proc. of the 24th USENIX Security Symp.*, Washington DC, USA, pp. 319–334, 2015.
[36] T. Mikolov, K. Chen, G. Corrado and J. Dean, "Efficient estimation of word representations in vector space," arXiv preprint arXiv:1301.3781, 2013.
[37] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones *et al.,* "Attention is all you need," in *Proc. of the 31st Advances in Neural Information Processing Systems (NIPS)*, Long Beach, CA, USA, pp. 5998–6008, 2017.
[38] H. Wang, F. Zhang, X. Xie and M. Guo, "DKN: Deep knowledge-aware network for news recommendation," in *Proc. of the 27th World Wide Web Conf. (WWW)*, Lyon, France, pp. 1835–1844, 2018.
[39] Y. Cao, X. Wang, X. He, Z. Hu and T. Chua, "Unifying knowledge graph learning and recommendation: Towards a better understanding of user preferences," in *Proc. of the 28th World Wide Web Conf. (WWW)*, San Francisco, CA, USA, pp. 151–161, 2019.
[40] T. Chen, Y. Sun, Y. Shi and L. Hong, "On sampling strategies for neural network-based collaborative filtering," in *Proc. of the 23rd ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD)*, Halifax, NS, Canada, pp. 767–776, 2017.
[41] J. Qiu, J. Tang, H. Ma, Y. Dong, K. Wang *et al.,* "DeepInf: Social influence prediction with deep learning," in *Proc. of the 24th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD)*, London, UK, pp. 2110–2119, 2018.
[42] D. Clevert, T. Unterthiner and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (ELUs)," in *Proc. of the 4th Int. Conf. on Learning Representations (ICLR)*, San Juan, Puerto Rico, 2015.
[43] "DARPA transparent computing," [Online]. Available: https://github.com/darpa-i2o/Transparent-Computing (accessed on 15/11/2023).
[44] "DARPA engagement 3," [Online]. Available: https://drive.google.com/open?id=1QlbUFWAGq3Hpl8wVdzOdIoZLFxkII4EK (accessed on 15/11/2023).
[45] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury *et al.,* "PyTorch: An imperative style, high-performance deep learning library," in *Proc. of the 33rd Advances in Neural Information Processing Systems (NIPS)*, Vancouver, BC, Canada, pp. 8024–8035, 2019.
[46] M. Wang, D. Zheng, Z. Ye, Q. Gan, M. Li *et al.,* "Deep graph library: A graph-centric, highly-performant package for graph neural networks," arXiv preprint arXiv:1909.01315, 2019.
[47] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," in *Proc. of the 3rd Int. Conf. on Learning Representations (ICLR)*, San Diego, CA, USA, 2014.
[48] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.
[49] X. Wang, H. Ji, C. Shi, B. Wang, P. Cui *et al.,* "Heterogeneous graph attention network," in *Proc. of the 28th World Wide Web Conf. (WWW)*, San Francisco, CA, USA, pp. 2022–2032, 2019.
[50] "THREATRACE," [Online]. Available: https://github.com/threaTrace-detector/threaTrace/ (accessed on 15/11/2023).
[51] "APT-KGL: An intelligent APT detection," [Online]. Available: https://github.com/hwwzrzr/APT-KGL (accessed on 15/11/2023).
[52] L. Zheng, Z. Li, J. Li, Z. Li and J. Gao, "AddGraph: Anomaly detection in dynamic graph using attention-based temporal GCN," in *Proc. of the 28th Int. Joint Conf. on Artificial Intelligence (IJCAI)*, Macao, China, pp. 4419–4425, 2019.
[53] X. Han, X. Yu, T. Pasquier, D. Li, J. Rhee *et al.,* "SIGL: Securing software installations through deep graph learning," in *Proc. of the 30th USENIX Security Symp.*, pp. 2345–2362, 2021.
[54] "GPT-4 technical report," [Online]. Available: https://cdn.openai.com/papers/gpt-4.pdf (accessed on 15/11/2023).
[55] "Microsoft security copilot," [Online]. Available: https://www.microsoft.com/en-us/security/business/ai-machine-learning/microsoft-security-copilot (accessed on 15/11/2023).