APT-KGL: An Intelligent APT Detection System Based on Threat Knowledge and Heterogeneous Provenance Graph Learning

Tieming Chen, Chengyu Dong, Mingqi Lv, Qijie Song, Haiwen Liu, Tiantian Zhu, Kang Xu
Ling Chen, and Shouling Ji, Yuan Fan

Abstract—APTs (Advanced Persistent Threats) have caused serious security threats worldwide. Most existing APT detection systems are implemented based on sophisticated forensic analysis rules. However, the design of these rules requires in-depth domain knowledge and the rules lack generalization ability. On the other hand, deep learning technique could automatically create detection model from training samples with little domain knowledge. However, due to the persistence, stealth, and diversity of APT attacks, deep learning technique suffers from a series of problems including difficulties of capturing contextual information, low scalability, dynamic evolving of training samples, and scarcity of training samples. Aiming at these problems, this paper proposes APT-KGL, an intelligent APT detection system based on provenance data and graph neural networks. First, APT-KGL models the system entities and their contextual information in the provenance data by a HPG (Heterogeneous Provenance Graph), and learns a semantic vector representation for each system entity in the HPG in an offline way. Then, APT-KGL performs online APT detection by sampling a small local graph from the HPG and classifying the key system entities as malicious or benign. In addition, to conquer the difficulty of collecting training samples of APT attacks, APT-KGL creates virtual APT training samples from open threat knowledge in a semi-automatic way. We conducted a series of experiments on two provenance datasets with simulated APT attacks. The experiment results show that APT-KGL outperforms other current deep learning based models, and has competitive performance against state-of-the-art rule-based APT detection systems.

Index Terms—Advanced Persistent Threat, Graph Neural Network, Provenance Data, Hosted-Based Security, Data-Driven Security

1 Introduction

APTs (Advanced Persistent Threats) are cyber-attacks launched by experienced attackers on carefully chosen targets, such as government, core infrastructure (e.g., energy, transportation, communication), and important industries (e.g., military, finance, medical). APTs can cause significant security threats such as stealing of sensitive data and damaging to system integrity, e.g., the Stuxnet worm attack on Iranian nuclear facilities, the BlackEnergy malware attack on Ukraine power grid, and the infiltration on Equifax that steals the personal information of 147.7 million Americans.

The most important characteristic of APTs is that an APT attacker would remain undetected for a prolonged period of time and utilize a pipeline of attack techniques to compromise the target hosts [1]. Thus, it is very difficult to detect APTs by utilizing traditional cyber-attack detection methods, which could only detect "single point" attacks [2] [3]. By investigating the experiences of previous researches, provenance data are the best approach for the APT detection task [4] [5] [6]. Provenance data are usually collected by using OS audit logs (e.g., Linux Auditd, Windows ETW). Most existing works model the provenance data as a directed acyclic graph (called a *provenance graph*), where the nodes represent system entities (e.g., processes, files, sockets) and the edges represent system events (e.g., read, write, fork). Fig.1 shows an example of the provenance graph. First, provenance graph can describe how system entities interact with each other and provide rich contextual information for each system entities [7]. Second, provenance graph can connect causally related events even when those events were separated by a long period of time [6].

Most existing APT detection systems based on provenance graphs (e.g., SLEUTH [8], HOLMES [9], CONAN [10]) try to detect APTs by manually designing a variety of rules based on threat knowledge (e.g., kill chain model [11], ATT&CK model [12]). Although the rule based APT detection strategies are efficient and have achieved state-of-the-art performance, they still suffer from the following limitations. First, the design of rules is an extremely difficult task, since it requires expertized domain knowledge of threat models, operation systems, and computer network. Second, rules lack generalization ability, since they are designed under specific conditions (e.g., the attack strategies, the network environment). They easily become obsolete and have to be adjusted or even redesigned, when the conditions change. Third, rules are limited to human experiences and cannot handle complex and latent patterns.

On the other hand, learning based techniques have long

- *T. Chen, C. Dong, M. Lv, Q. Song, H. Liu, and T. Zhu, K.Xu are with the College of Computer Science and Technology, Zhejiang University of Technology, Hangzhou, China.E-mail: tmchen@zjut.edu.cn, 2111912078@zjut.edu.cn, mingqilv@zjut.edu.cn, songqijie@zjut.edu.cn, 2112112262@zjut.edu.cn, ttzhu@zjut.edu.cn, 211912069@zjut.edu.cn.*
- *L. Chen and J. Shou are with the College of Computer Science and Technology, Zhejiang University, Hangzhou, China. E-mail: lingchen@zju.edu.cn, sji@zju.edu.cn.*
- *Y. Fan is with DAS-Security, Hangzhou, China, E-mail: frank.fan@dbappsecurity.com.cn.*
- *Corresponding author: Mingqi Lv.*

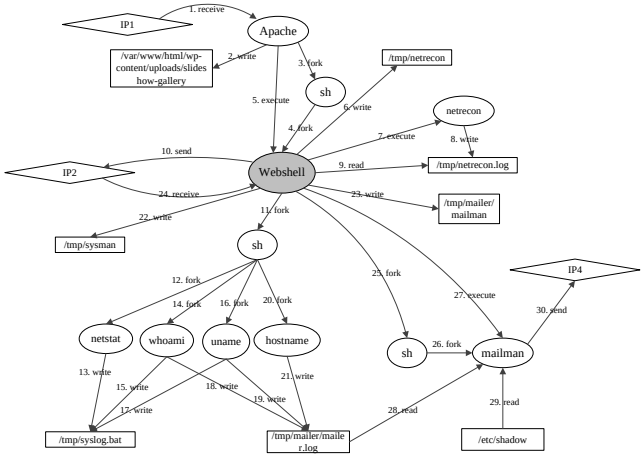

Fig. 1. An example of the provenance graph

been applied for cyber security tasks such as intrusion detection [2] [3], malware detection [13] [14], and fraud detection [15] [16]. As compared with the rule based techniques, learning based techniques have the following advantages. First, learning based techniques (especially deep learning techniques) could extract latent patterns and build detection model from the training dataset with little domain knowledge. Second, the detection model can adapt to new conditions by retraining or fine-tuning in an automatic manner. Since provenance graph is the most commonly used data structure for APT detection, recent studies try to exploit GNN (Graph Neural Network) [17] as the underlying model for the APT detection task. However, due to the complexity of APTs and system provenance, building APT detection model based on GNN is still a challenging task.

First, a provenance graph involves multiple types of nodes (e.g., processes, files, sockets) and multiple types of edges (e.g., read, write, and create). In addition, the nodes and edges represent distinct instances of system entities and system events, so it is difficult for GNNs to learn high-level patterns from these low-level instances.

Second, APT detection is a speed sensitive task. However, most GNNs have high computation and memory cost. In addition, since APTs are usually persistent, the provenance graphs continue to grow over time, making the GNN based methods inevitably suffer from efficiency and memory problems.

Third, APTs are usually stealthy, so it is very difficult to detect them by only looking at individual system entities (e.g., an attack activity might be accomplished by the collaboration of multiple processes and files). However, it is also impractical to consider the whole provenance graph, which is usually too large and dominated by benign system entities.

Fourth, the detection model has to continuously deal with new incoming system entities, while GNNs learn the representations of new system entities by embedding them with the existing nodes and edges in a shared feature space. However, it is infeasible to rerun the embedding procedures whenever new system entities arrive due to the high computation complexity.

Fifth, as compared with benign system activities, real APTs are extremely rare. In additoin, since APTs are highly complex, it would also incur a huge cost to simulate them in a cyber range environment. As a result, it is almost impossible to obtain adequate APT training samples. The low diversity of APT training samples would lead to a low generalizability of the detection models.

Aiming at the above challenges, this paper proposes APT-KGL, an intelligent APT detection system based on heterogeneous provenance graph learning. For the first challenge, APT-KGL adopts a heterogeneous graph [18] to represent the complex system entities and events in the provenance graph, and then uses meta-path based approach [19] to extract higher-level semantic interactions among the system entities. For the second challenge, APT-KGL uses graph embedding technique to learn a low-dimensional vector representation for each node in the heterogeneous graph. Here, the graph embedding can be executed as a pre-training step, and the vector representations are then used to quickly initialize the detection model. For the third and fourth challenges, APT-KGL links new incoming system entities to the existing heterogeneous graph and performs a local graph sampling to incorporate the new incoming system entities and the most related existing nodes into a small and compact local graph. Then, the detection model is built based on this local graph. For the fifth challenge, APT-KGL generates supplementary malicious training samples by mining the open threat knowledge, e.g., CTIs (Cyber Threat Intelligences) and TTPs (Tactics, Techniques, and Procedures). In summary, the main contributions of this paper are as follows.

1) We propose an intelligent APT detection system by leveraging GNNs and threat knowledge. It can automatically learn patterns from the provenance data and use for APT detection.
2) We propose a heterogeneous graph based approach to model provenance data and learn semantic vector representations for various system entities.
3) We propose a local graph sampling based approach to perform APT detection upon new incoming system entities, which do not have pre-learnt vector representations.
4) We design a threat knowledge mining and sample augmentation approach, which can generate supplementary APT training samples from open threat knowledge in a semi-automatic way.
5) We conducted extensive experiments based on real datasets containing a variety of APT activities. The experiment results show that APT-KGL outperforms existing machine learning based APT detection methods and can achieve competitive performance as compared with state-of-the-art rule-based APT detection systems built on indepth domain knowledge.

2 RELATED WORK

As the mainstream of the existing APT detection systems, forensic analysis approaches aim to discover the attack events and attack paths based on predefined rules. For example, PrioTracker [20] enables timely APT attack analysis

by calculating the priority of a system event based on predefined rules. SLEUTH [8] identifies system entities and events that are most likely involved in APTs based on a tag-based approach. The tags are manually designed to encode the trustworthiness of code and sensitivity of data. HOLMES [9] is a hierarchical framework for APT detection. The key component of HOLMES is an intermediate layer that maps low-level audit data to suspicious behaviors based on rules from domain knowledge (e.g., ATT&CK model). CONAN [10] is a state-based framework. Each system entity (i.e., process or file) is represented in an FSA (Finite State Automata) like structure, where the states are inferred through predefined rules. Then, the sequences of states are used for APT detection. Thanks to the carefully designed rules, forensic analysis approaches could be effective, efficient, and easy to deploy. However, designing effective rules heavily relies on in-depth domain knowledge, and the rules easily become obsolete if the conditions change. In addition, the rules are "hard" and usually cannot adapt to new APT attacks.

To address the limitations of forensic analysis approaches, researchers tried to build APT detection model in an automatic way by leveraging learning based techniques (including machine learning and deep learning techniques). For example, Barre et al. [21] build a classifier to detect APTs on top of a set of features (e.g., total quantity of data written, number of system files used) extracted from the provenance data. Berrada et al. [22] extract Boolean-valued features (called contexts) from the provenance graph, and treat APT detection as an anomaly detection task by using unsupervised learning technique. Xiang et al. [23] extract different features from two separated platforms (i.e., PC platforms and mobile platforms), and use several machine learning algorithms to detect APTs. Coulter et al. [24] extract features based on system-driven footprint and apply domain adaptation technique to detect APT malware samples in new domains. UNICORN [6] proposes a WL-kernal based method to extract contextual features from the whole provenance graph and learn evolving models to detect APT attacks. The above studies detect APTs based on traditional machine learning techniques, in which the core step is feature engineering. However, feature engineering also relies heavily on domain knowledge, and thus the learning capability is constrained by the design of features.

Recently, deep learning [25] has shown its superior capability of discovering features from big data in a totally automatic way. Thus, a few studies have tried to use deep learning techniques for APT detection. Since persistence is one of the most fundamental characteristic of APTs, the collected system events usually span over a long period of time and can be represented as a time series. Therefore, RNNs (Recurrent Neural Networks) such as LSTM (Long Short-Term Memory) and GRU (Gated Recurrent Unit) are mostly applied deep learning models [26] [27] [28]. However, RNNs can only capture the sequential correlations among system events, and cannot capture the complex contextual information of nodes in the provenance graphs.

Since provenance graphs have been proven to be the best data sources for the APT detection task, several recent studies have exploited GNNs for APT detection. For example, Wang et al. [29] propose THREATRACE, which uses GraphSAGE to learn the representation of every benign system entity in provenance graphs and detects APT threats by discovering anomaly system entities. Liu et al. [30] propose Log2Vec, an APT detection model based on heterogeneous graph embedding. Specifically, it firstly converts log entries into a heterogeneous graph and represents each log entry as a low-dimensional vector using heterogeneous graph embedding technique. Then, it separates malicious and benign log entries based on the log entry vectors. Li et al. [31] propose a hierarchical GNN to detect APT attacks from both intrahost provenance graph and interhost interactive graph. The core idea is also embedding the nodes into a latent space. However, these methods suffer from the deployment feasibility problem. First, they cannot efficiently handle online data streaming based on the offline graph embedding techniques. Second, they require a large number of training samples while the APT samples are extremely scarce in practice.

To the end, we summarize different categories of APT detection methods in Table 1. APT-KGL is also a GNN based method, but it differs from the existing GNN based methods from the following aspects. First, APT-KGL abstracts low-level provenance graphs by high-level heterogeneous graphs. Second, APT-KGL applies a local graph sampling strategy to process online incoming system entities upon offline graph embedding. Third, APT-KGL generates supplementary malicious training samples by mining the open threat knowledge.

TABLE 1
A SUMMARY OF APT DETECTION METHODS

Category	Publications	Limitations
Rule based methods	[8] [9] [10] [20]	They have low generalizability and cannot detect new APT attacks.
Machine learning based methods	[6] [21] [22] [23] [24]	They rely heavily on handcrafted features and have weak learning capability.
RNN based methods	[26] [27] [28]	They cannot model the complex contextual information of nodes in the provenance graphs.
GNN based methods	[29] [30] [31]	They suffer from the deployment feasibility problem.

3 PRELIMINARY

3.1 Threat Model

According to the threat models (e.g., kill chain model [11], ATT&CK model [12]), the attackers may conduct APTs through several stages and use a variety of techniques during each stage. Among these stages, previous studies have found the following invariant parts of the APTs [10]. First, the attackers must deploy their code to victims. Second, the final targets of APTs remain the same over the years, i.e., stealing sensitive information or causing damage. Third, the attackers will communicate with C&C server. These invariant parts make possible the detection of new APTs, and the goal of APT-KGL is to detect attacks at any of the invariant stages.

The provenance data (e.g., system events such as file operations and network access) can be collected from kernel level information by using OS audit modules (e.g., Auditd,

ETW), which have the advantages of high availability, high credibility, and high stability. In this paper, we focus on APT-KGL's analytic capabilities and assume the correctness of the kernel, the provenance data, and the analysis model.

3.2 System Architecture

The architecture of APT-KGL is shown in Fig. 2, consisting of the following three modules.

1) Provenance graph constructor: It continuously collects kernel level system events by using OS audit logs, and arranges these collected system events as a provenance graph, which represents each system event by an edge of the provenance graph.
2) Heterogeneous graph learner: It transforms the provenance graph into a HPG (Heterogeneous Provenance Graph) by defining the node type set and edge type set. Then, it learns a low-dimensional vector representation for each node (a.k.a. heterogeneous graph embedding), which can preserve the semantic and structural correlations between various types of nodes. Specifically, the heterogeneous graph embedding task is performed based on a meta-path sampling strategy and a hierarchical attention based embedding technique.
3) APT detector: For a new incoming system entity v_n, it links v_n to the existing heterogeneous graph and samples a local small graph by taking v_n as the central node. Then, it treats the APT detection as a node classification task. In order to discover causal and contextual correlations between system entities by exploring more distant graph neighborhoods, it adopts a graph convolutional network for this task.
4) Threat knowledge extractor: It extracts HPGs representing APT attack chains from open threat knowledge in a semi-automatic way, and then augments these HPGs by adding noisy edges and nodes to create supplementary APT training samples.

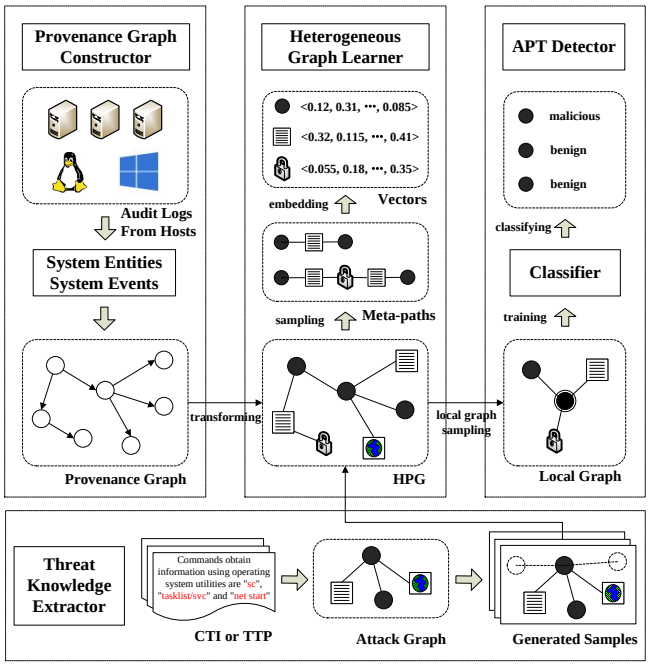

Fig. 2. The architecture of APT-KGL.

4 METHODOLOGY

4.1 Provenance Graph Constructor

After collecting the provenance data logged by OS audit modules, we organize the provenance data into the form of provenance graphs, where the nodes represent system entities (e.g., process, file) and the edges represent system events (e.g., writing to a file, reading from an IP address).

In particular, provenance graphs capture causality relationships between system entities. Causal relationships facilitate the reasoning over system entities that are temporally distant, thus useful in navigating through APT's persistent attack patterns.

4.2 Heterogeneous Graph Learner

4.2.1 Heterogeneous Graph Construction

To apply GNN, we transform the provenance graph into a heterogeneous graph (called a *heterogeneous provenance graph*, abbreviated as *HPG*), defined as follows.

Definition 1. (HPG). A HPG, denoted as $G = (V, E)$, consists of a node set V representing the system entities and an edge set E representing the system events. A HPG is also associated with a node type mapping $\varphi : V \rightarrow A$ and an edge type mapping $\psi : E \rightarrow R$. A and R denote the node type set and the edge type set, where $|A| + |R| > 2$. It indicates that a HPG contains multiple types of system entities and system events. The schema [18] of a HPG G, denoted as $T_G = (A, R)$, is a meta-graph with nodes as node types from A and edges as edge types from R.

Based on the definition, we firstly define the schema of the HPG, i.e., the node type set and the edge type set. For our case, we define seven major node types, i.e., process, file, socket, IPC, memory, network, and attribute. The node type "attribute" is used to describe the features of "process" and "file", and can be further categorized into six subtypes, i.e., common process, sensitive instruction, common file, network data, sensitive data, and uploaded data. Fig. 3(a) shows these node types and their potential relations. Here, "sensitive instruction" represents a process executing a sensitive instruction, "network data" represents a file containing data from the network, "sensitive data" represents a file containing sensitive data, and "uploaded data" represents a file sent to a remote server. In addition, "common process" and "common file" are shared attributes, which are used to prevent the HPG from being separated into many disconnected small graphs. As the example shown in Fig. 3(b), P_1, P_2, P_3, P_4, and P_5 are common processes, F_1 and F_2 are common files. Without "common process" and "common file" attributes (i.e., the dashed arrows in Fig. 3(b)), these system entities would be separated into three disconnected small graphs (as shown by the dashed ovals), and thus the semantic relatedness between disconnected system entities (e.g., P_1 and P_3, F_1 and F_2) could not be learnt jointly. This problem could be extremely serious in practice, since the

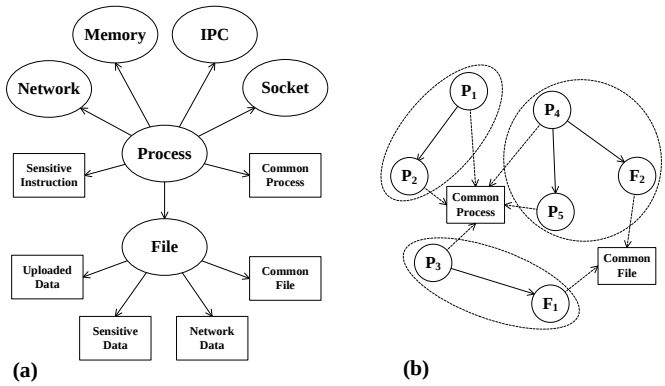

Fig. 3. An illustration of the HPG: (a) the schema of the HPG; (b) an example of the necessity of the "common process" and "common file" attributes.

vast majority of system entities are common and benign.

Then, we define eight edge types, as summarized in Table 2. The edge types represent the relations between system entities. The intrinsic and complex relations between system entities can provide crucial information for detecting APTs. Note that a pair of system entity types could have multiple types of relations. For example, a "process" and a "file" could have the relations such as "read", "write", "create", etc.

Finally, the main difference between the provenance graph and the HPG is as follows. The provenance graph considers specific instances of system entities (e.g., every different file is considered as totally different node). On contrary, the HPG uses generalized concepts to define each node. For example, the semantic relations between different files can be captured by common attribute nodes, which facilitate the learning of generalized patterns.

4.2.2 Heterogeneous Graph Embedding

Due to the high persistence of APTs, the HPG could be extremely large. Hence, we should represent and encode the HPG in a scalable way to make real-time APT detection possible. Based on the existing studies, graph embedding is an efficient and effective way to represent large-scale graphs [32], and thus we adopt heterogeneous graph embedding technique to represent the HPG. We define HPG embedding as follows.

Definition 2. (HPG embedding). Given a HPG $G = (V, E)$, the embedding of G is to learn a function $f : V \rightarrow R^d$ that maps each node $v \in V$ to a d-dimensional vector ($d \ll |V|$). The low-dimensional vector space should be capable of preserving the topological structure and semantic relations in G.

Since HPGs contain multiple types of nodes and edges, conventional graph embedding techniques (e.g., DeepWalk [33], node2vec [34], LINE [35]) cannot be directly applied. To address this problem, meta-path was proposed to guide the random walkers to connect various types of nodes [19]. The random walks on a meta-path can generate a set of neighbors for each node, which can reveal diverse structural and semantic information in a heterogeneous graph. Here, we give the formal definition of meta-path as follows.

Definition 3. (Meta-path). A meta-path is a path defined in the schema of a HPG, and is in the form of $A_1 \rightarrow R_1 \rightarrow A_2 \rightarrow R_2 \rightarrow \ldots \rightarrow R_{L-1} \rightarrow A_L$, where $R = R_1 \circ R_2 \circ R_{L-1}$ defines a composite relations between node type A_1 and A_L. In practice, A_1 and A_L in a meta-path are usually of the same node type, so that the random walk ends on a meta-path could immediately starts on another one.

Based on the schema of the HPG, we define a variety of meta-paths to characterize the relatedness over different types of system entities from different views, which are summarized in Table 3. For example, MP_4 means that two processes are related if they are connected to files with the same attribute, while MP_6 indicates that two processed are related if they perform the same operation on the network.

After defining all the meta-paths, we apply HGAT, a heterogeneous graph embedding technique based on hierarchical attention [36], to learn the d-dimensional vector for each node in the HPG. The advantage of HGAT over other heterogeneous graph embedding techniques (e.g., metapath2vec [19], metagraph2vec [37]) is that it is able to learn the importance of different neighbors and meta-paths for a specific node. This characteristic is especially important for the APT detection task, since different system entities and behaviors usually have different sensitive and dangerous levels.

We apply HGAT for HPG embedding in two steps. In the first step, we generate sequences of system entities with various types from the HPG by using the meta-paths to guide the random walker. Given a HPG $G = (V, E)$ and a set of meta-paths MPS, the random walker works as follows. First, it randomly chooses a meta-path MP_i from MPS (in the form of $A_1 \rightarrow R_1 \rightarrow \ldots \rightarrow A_t \rightarrow R_t \rightarrow A_{t+1} \rightarrow \cdots \rightarrow R_{L-1} \rightarrow A_L$). Second, it travels on the HPG in accordance with MP_i, and the transition probability from node v_j to v_{j+1} at step j is defined in Equation 1, where $\varphi(v_j) = A_t$ and $N_{t+1}(v_j)$ is the set of neighbors of node v_j with entity type A_{t+1}. Finally, it randomly picks another meta-path MP_{i+1} starting with A_L, and repeats the transition process.

$$p(v_{j+1}|v_j, MP_i) = \begin{cases} \frac{1}{|N_{t+1}(v_j)|}, & (v_j, v_{j+1}) \in E, \\ & \varphi(v_{j+1}) = A_{t+1}. \\ 0, & (v_j, v_{j+1}) \in E, \\ & \varphi(v_{j+1}) \neq A_{t+1}. \\ 0, & (v_j, v_{j+1}) \notin E. \end{cases} \quad (1)$$

In the second step, we feed the generated system entity sequences to the HGAT model to learn the system entity embeddings. HGAT is a GNN based on a hierarchical attention structure (i.e., a node-level attention and a semantic-level attention) [36]. Specifically, for a given system entity v_i and a meta-path MP_j, the node-level attention learns the importance of each neighbor of v_i in the generated system entity sequence from MP_j, and aggregates these meaningful neighbors to form a candidate embedding of v_i (denoted as $\mathbf{Z}_i^j$). Then, the semantic-level attention learns the importance of different meta-paths for v_i, and fuses all the candidate embeddings (i.e.,$\mathbf{Z}_i^1, \mathbf{Z}_i^2, ..., \mathbf{Z}_i^{|MPS|}$) to obtain the final embedding of v_i. Note that HGAT should be trained with a downstream task, which is not necessarily the APT detection task. For example, Wang *et al.* [29] adopted node recognition (i.e., recognizing a node as "process", "file", or other types)

TABLE 2
A SUMMARY OF THE SYSTEM EVENT TYPES

Pairs of system entity types	ID	System event type	Description
$Process \rightarrow Process$	R_1	$Process \rightarrow op1 \rightarrow Process$	Here, "op1" can be "fork", "execute", "exit", "clone", or "change".
$Process \rightarrow File$	R_2	$Process \rightarrow op2 \rightarrow File$	Here, "op2" can be "read", "open", "close", "write","loadlib", "create", "unlink", "modify", "truncate", "rename", "mmap", or "update".
$Process \rightarrow Network$	R_3	$Process \rightarrow op3 \rightarrow Network$	Here, "op3" can be "connect", "send", "recv", "read", "close", "accept", or "write".
$Process \rightarrow Memory$	R_4	$Process \rightarrow op4 \rightarrow Memory$	Here, "op4" can be "mprotect" or "mmap".
$Process \rightarrow IPC$	R_5	$Process \rightarrow op5 \rightarrow IPC$	Here, "op5" can be "write", "close", "read", or "mmap".
$Process \rightarrow Socket$	R_6	$Process \rightarrow op6 \rightarrow Socket$	Here, "op6" can be "connect", "send", "recv", "read", "close", "accept" or "write".
$Process \rightarrow Attribute$	R_7	$Process \rightarrow contain \rightarrow Attribute$	A process contains one of the following attributes, i.e., "common process" and "sensitive instruction".
$File \rightarrow Attribute$	R_8	$File \rightarrow contain \rightarrow Attribute$	A file contains one of the following attributes,i.e., "common file", "network data", "sensitive data", and "uploaded data".

TABLE 3
A SUMMARY OF THE META-PATHS

ID	Meta-paths
MP_1	$Process \rightarrow op1 \rightarrow Process$
MP_2	$Process \rightarrow contain \rightarrow Attribute \rightarrow contain^{-1} \rightarrow Process$
MP_3	$Process \rightarrow op2 \rightarrow File \rightarrow op2^{-1} \rightarrow Process$
MP_4	$Process \rightarrow op2 \rightarrow File \rightarrow contain \rightarrow Attribute \rightarrow contain^{-1} \rightarrow File \rightarrow op2^{-1} \rightarrow Process$
MP_5	$Process \rightarrow op5 \rightarrow IPC \rightarrow op5^{-1} \rightarrow Process$
MP_6	$Process \rightarrow op3 \rightarrow Network \rightarrow op3^{-1} \rightarrow Process$
MP_7	$Process \rightarrow op4 \rightarrow Memory \rightarrow op4^{-1} \rightarrow Process$
MP_8	$Process \rightarrow op6 \rightarrow Socket \rightarrow op6^{-1} \rightarrow Process$

as a downstream task to train the embedding model. In this paper, we utilize node classification (i.e., classifying a "process" as benign or malicious) as the downstream task.

4.3 APT Detector

After obtaining all the system entity embeddings, the simplest strategy of detecting APT activities is to train a classifier to classify each of the "process" as benign process or APT attack process, by directly taking the d-dimensional embedding vectors of system entities as features, like most previous studies do [29] [30]. However, this simple strategy has the following problems. First, APTs are usually stealthy, so it is very difficult to detect them by only considering one system entity. Second, the APT detection system has to continuously deal with new incoming system entities, while it is infeasible to frequently rerun the HPG embedding procedure due to its high computation complexity.

To address these problems, we propose a local graph sampling based APT detection strategy. Given a new incoming system entity v_n and the system entities interact with v_n (denoted as NVS), we firstly link each system entity in NVS to the existing HPG (the combined graph is denoted as CG). Note that the entity type "attribute" can guarantee that every system entity will be linked to the existing HPG. Then, we sample a local graph from CG for NVS based on the k-order sub-graph sampling defined as follows.

Definition 4. (k-order sub-graph sampling). For a system entity v_i, the k-order sub-graph sampling of v_i on $G = (V, E)$ (denoted as $SG(v_i, G)$) is defined in Equation 2. Based on the definition, every system entity can form a local graph based on the k-order sub-graph sampling. In addition, we define the k-order sub-graph sampling of a system entity set VS on $G = (V, E)$ as the union of the sampling of all system entities in VS (denoted as $SG(VS, G)$ in Equation 3).

$$SG^{(k)}(v_i, G) = \begin{cases} \{v_j | (v_i, v_j) \in E\} & k = 1 \\ \left\{ SG^{(1)}(v_z, G) | v_z \in SG^{(k-1)}(v_i, G) \right\} & k > 1 \end{cases} \quad (2)$$

$$SG^{(k)}(VS, G) = \mathbf{U}_{v_i \in VS} SG^{(k)}(v_i, G) \quad (3)$$

The local graph could grow too large without constraint. For example, if the local graph contained an internal node v_i with type "common file", it would connect to too many nodes with type "file". To prevent such situation, we put a threshold on the degree of nodes in the local graph. Specifically, we firstly categorized the seven major entity types into two groups, i.e., instantiable entity type (i.e., process and file) and conceptual entity type (i.e., socket, IPC, memory, network, and attribute). Obviously, the number of nodes with conceptual entity type is limited, and thus we only consider the number of nodes with instantiable entity type. For each node in the local graph, the number of its connected nodes with instantiable entity type should be no more than λ. Fig. 4 shows an example of the local graph sampling strategy. The purpose of local graph sampling is twofold. First, rather than utilizing the information of only one system entity, we will aggregate the contextual information from all system entities in the local graph for APT detection. Second, for a new incoming unknown system entity, we will infer its embedding based on the information transferred from the neighbors in the local graph, with no

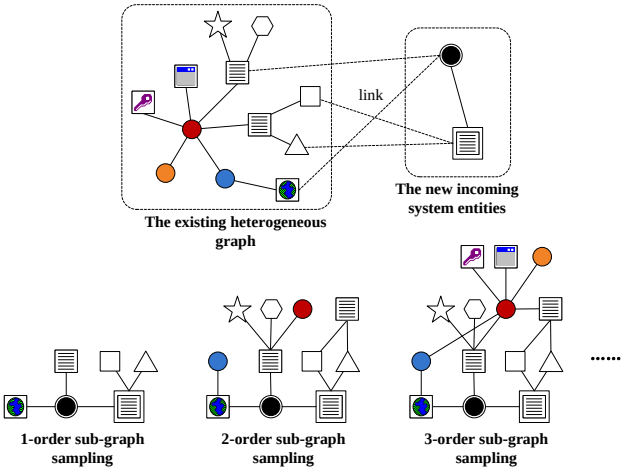

Fig. 4. An example of the local graph sampling strategy.

need to rerun the heterogeneous graph embedding process.

To achieve the above purposes, we perform APT detection as a classification task for each node with type "process" in the local graph. Specifically, we apply R-GCN (Relational Graph Convolutional Network) [38] for the node classification task. R-GCN is an extension of GCN to relational graphs. It accumulates information from neighbors to a central node (denoted by v_i) based on Equation 4, where $h_i^{(l)}$ is the hidden state of node v_i in the l-th R-GCN layer, N_i^r is the set of indices of neighbors of v_i with edge type r, W_0^l and $W_r^{(l)}$ are learnable parameters. Note that more R-GCN layers (i.e., a larger value of l) indicate the model can aggregate contextual information from more remote nodes.

$$h_i^{(l+1)} = \sum_{r \in R} \sum_{j \in N_i^r} \frac{1}{|N_i^r|} W_r^{(l)} h_j^{(l)} + W_0^{(l)} h_i^{(l)} \quad (4)$$

Then, we simply add a softmax activation on the output hidden state of each node of the last layer by minimizing the cross-entropy loss on all labelled nodes to train the node classifier.

4.4 Threat Knowledge Extractor

It is extremely difficult to collect real APT data or simulate APT attacks [24]. Hence, it is usually impossible to obtain adequate training samples of APT activities for the machine learning technique to train a robust detector. On the other hand, the threat knowledge about known APTs are usually abstracted in natural language and shared as CTIs or TTPs.

Following the experiences in [39], we extract threat knowledge from CTIs or TTPs, and model the threat knowledge as *query graphs* in a semi-automatic way. Specifically, we use automated tools (e.g., EXTRACATOR [40], TTPDrill [41]) to extract initial query graphs, which are then manually refined by security experts. Note that the query graphs are essentially provenance graphs, thus we also transform them into HPGs for further processing.

However, the HPGs directly generated from CTIs or TTPs (called *original* HPGs) are not ready as training samples, since they are too "pure", only containing the malicious behaviors in the attack chain of APTs. Actually, in real environment, the malicious behaviors of APTs are always hidden in a large number of benign behaviors. Therefore, we augment the original HPGs by adding noisy nodes and edges using the following strategy.

First, we collect a large number of HPGs by monitoring the normal system running (called *benign HPGs*), and learn the conditional probabilities of each system event type $P_h(op_i|E_j)$ and $P_t(op_i|E_j)$ by analyzing the transitions between pairs of entity types in these benign HPGs. Here, $P_h(op_i|E_j)$ stands for the probability of system event type op_i given that the head node of op_i was with type E_j, while $P_t(op_i|E_j)$ stands for the probability of system event type op_i given that the tail node of op_i was with type E_j.

Then, given an original HPG, we add noisy nodes and edges for each node v_j with type "process" or "file" in the original HPG. Specifically, let the type of v_j be E_j (we call v_j a *root node*), for each system event type op_i, we generate a noisy edge with type op_i and the corresponding noisy tail node v_i with the probability of $P_h(op_i|E_j)$. Likewise, we can also generate a noisy edge with type op_i and the corresponding noisy head node v_i with the probability of $P_t(op_i|E_j)$. Then, let the type of v_i be E_i, we generate more noisy edges and nodes recursively by examining $P_h(op_k|E_i)$ and $P_t(op_k|E_i)$. The final HPG is called *augmented HPG*.

In the augmentation process, we apply several mechanisms to ensure the rationality of the augmented HPGs: 1) To prevent the augmented HPGs from growing too large, we set an upper limit of the number of the generated noisy nodes from a root node. 2) At most one noisy node with each conceptual entity type could be generated from a node with instantiable entity type. 3) To promote the diversity of the augmented HPGs, we apply several random mechanisms. First, the number of noisy nodes with instantiable entity type generated from a node is random. Second, the order of examined system event types when generating noisy edges is random.

Fig. 5 gives an example of generating an augmented HPG from query graph, which stands for a simple attack behavior that executing a malware from network and reading sensitive data from local files.

5 EXPERIMENTS

5.1 Experiment Setup

5.1.1 Dataset

We evaluate APT-KGL using the following two different datasets with multiple types of APT attacks.

Lab Dataset: We simulated APT attacks on real hosts and collected datasets with ground truth for evaluation based on two methods.

In the first method, we perform three types of attacks on a host with Linux system through a red team campaign. The first type is "attack by webshell", which uses webshell scripts to escalate and maintain persistent access to the target system by exploiting web vulnerabilities. The second type is "attack by RAT (Remote Access Trojan)", which implants Trojans to the target system through phishing attacks and then uses the Trojans to steal sensitive information. The third type is "attack by LotL (Living off the Land)", which runs malicious shellcode directly in memory. We utilize a

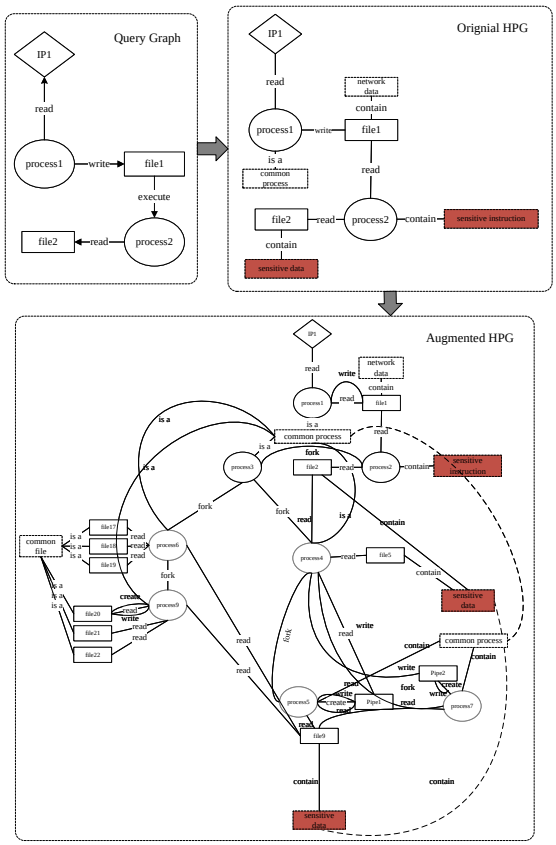

Fig. 5. The architecture of APT-KGL.

tool named SPADE [42] to convert the collected data into a provenance dataset in the form of provenance graphs.

In the second method, we simulate APT attacks in a semi-automatic way by using a tool named Atomic Red Team [43] from Red Canary. First, this tool can generate atomic attacks by referring to ATT&CK, where an atomic attack is a single step of an APT attack following a specific TTP. Second, we manually combine atomic attacks to form complete APT attack samples. Third, we implement these APT attack samples on a host with Windows system and collect provenance data by using a self-developed tool named KELLECT [44]. We give a running example to illustrate how to simulate an APT attack by using Atomic Red Team tool in the Appendix.

Finally, the details of the Lab dataset are shown in the first row of Table 4. The column "Duration" refers to the length of time that the collector was running on the target host, covering both normal activities (e.g., watching online videos, website browsing, document editing, etc.) and APT-related activities. The next several columns provide a breakdown of events into different types of operations. The column "♯Attack" refers to the number of APT attack campaigns. The Lab dataset contains 29864 processes, 10625 files, and 2715960 system events.

DARPA Dataset: This dataset was collected from three hosts installed Windows 10 during an APT attack campaign carried out by a red team as part of the DARPA Transparent Computing program. The details of this datasets are shown in the second row of Table 4.

There are mainly three types of attacks performed during the collection of this dataset, i.e., sensitive information gathering and exfiltration, in-memory attack by exploiting FireFox vulnerabilities, and malicious file downloading and execution. It can be seen that the strategies of attacks in the DARPA dataset are quite different from those in the Lab dataset. However, all these strategies contain one or more invariant stages (e.g., deploying malicious code, stealing sensitive information, communicating with C&C server) of the APTs discussed in Section 3.1.

In addition, while the red team was attacking the target host, benign background activities were also being performed on the host. In general, nearly 99.9% of the system events are related to benign activities. Finally, after the provenance graph construction and compaction, the dataset contains 53084 processes, 219706 files, and 26751468 system events.

In addition to the two datasets, we also created an augmented dataset (called **Knowledge Dataset**) by mining CTIs and TTPs based on the method specified in Section 4.4. In the experiments, we firstly collected CTIs and TTPs that could cover all the types of attacks in Lab Dataset and DARPA Dataset, and then generated augmented HPGs as training samples.

We conducted the experiments on a server with an Intel Xeon E5-2680 v4 CPU (with 14 × 2.4 GHz cores), 128GB of memory, and 4 × GTX 2080Ti GPUs running on Ubuntu 20.04. The source code and part of the datasets are available at https://github.com/hwwzrzr/APT-KGL.

5.1.2 Evaluation Strategy

First, we introduce the strategy of labelling and selection of samples. When initializing an attack in the host, it would create a process. We call this process as the attack origin and other processes spawned by the attack origin (e.g., processes forked by the attack origin) as the attack suspects. According to the previous studies [21], it is ambiguous to label the attack suspects as malicious or benign, since the attack suspects can be arbitrarily extended by the propagation of attack origins. Therefore, in our experiment, we only label the attack origin and the attack suspects performing actual malicious activities as "malicious". On the other hand, since other attack suspects are ambiguous, we tried to exclude them in the training and testing procedures. Specifically, given a process being labeled as "malicious", we exclude the third-order neighbors of it in the provenance graph from being selected as training or testing samples.

Second, we introduce the three evaluation strategies used in our experiments as follows.

1) In-Sample: This evaluation strategy is used to evaluate the APT detection performance in an offline way. Specifically, it builds the HPG on the whole dataset, which is then divided into training set and testing set. To maintain the original proportion of malicious / benign processes in both training and testing sets, we use stratified sampling to choose two-thirds of the malicious processes and two-thirds of the benign processes to generate the training set, and use the rest as testing set. This procedure is repeated three times and the average performance is reported.

TABLE 4
THE DETAILS OF THE DATASETS

Dataset	Duration (hh:mm:ss)	File read	File write	Process/Thread	Network	Others	#Attack
Lab Dataset	15:38:21	45.11%	18.41%	1.94%	14.47%	20.07%	42
DARPA Dataset	24:32:15	45.10%	27.78%	0.91%	9.04%	17.17%	7

2) Out-Sample: This evaluation strategy is used to evaluate the online APT detection performance. Specifically, it adopts stratified sampling to divide the whole dataset into training set (two-thirds of the processes) and testing set (one-third of the processes). The training set is used for HPG construction, HPG embedding, and APT detector training. The testing set is used as new incoming processes to evaluate the APT detector in an online way.
3) No-Sample: This evaluation strategy is used to evaluate the APT detection performance by leveraging the threat knowledge. Specifically, it trains the APT detector on Knowledge Dataset and the benign samples on Lab Dataset or DARPA Dataset, and tests on the corresponding dataset.

Third, we use ACC, Precision, Recall, and Macro-F1 as the performance metrics. Here, ACC refers to accuracy. Precision and Recall are only calculated for detecting malicious samples. Precision, Recall, and Macro-F1 are used with the consideration of the imbalanced malicious / benign samples.

5.1.3 Training Strategy

The dataset is highly class imbalanced, i.e., the majority of training samples are benign processes. Thus, standard training strategy tends to be overwhelmed by the benign samples and ignore the malicious ones. Aiming at this situation, we adopt a cost-sensitive version of the cross entropy loss function. Specifically, we assign a larger weight for malicious samples than that for benign samples during the model training. The ratio of the malicious sample weight to the benign sample weight is $\delta : 1$ ($\delta < 1$).

5.2 Experiment 1: Parameter Tuning

In the first experiment, we investigate the three key parameters in APT-KGL, i.e., the dimension of the node embeddings d(Section 4.1.2), the threshold of the node degree in the local graph λ(Section 3.4), and the order of the subgraph sampling k(Section 3.4). In this experiment, In-Sample evaluation strategy is utilized.

First, the experiment result for tuning parameter d is shown in Fig. 6. Here, we fix $k = 8, \lambda = 10$, and vary d in the range of [8, 128]. The overall performance shows an increasing phase and then a decreasing phase is followed. When d is too small, it is difficult for the node embeddings to encode enough context information. When d is too large, it is more easily for the node embeddings to be affected by noises and lead to overfitting. Besides, ACC is almost unchanged when increasing d. It is because that most of the benign processes can be stably and correctly classified, while they account for a large majority of the samples. Therefore, small fluctuations on the classification performance of malicious processes would not significantly affect the overall ACC. To this end, we set $d = 32$ in the following experiments.

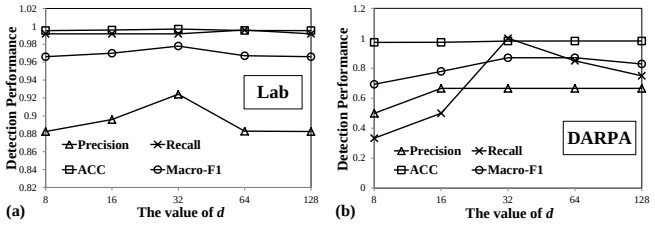

Fig. 6. The effect of parameter d: (a) the experiment results on Lab Dataset; (b) the experiment results on DARPA Dataset.

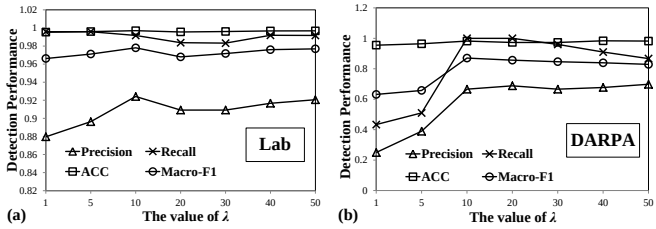

Fig. 7. The effect of parameter λ: (a) the experiment results on Lab Dataset; (b) the experiment results on DARPA Dataset.

Second, the experiment result for tuning parameter λ is shown in Fig. 7. Here, we fix $d = 32, k = 8$, and vary λ in the range of [1, 50]. The curves are relatively stable when increasing $\lambda > 10$, showing that λ has limited influence on the detection performance. It indicates that considering too many system entities of instantiable entity type (i.e., process and file) has limited effect on APT detection. What's more important is the semantics behind the system entities. To this end, we set $\lambda = 10$ to trade-off between detection performance and computation complexity in the following experiments.

Third, the experiment result for tuning parameter k is shown in Fig. 8. Here, we fix $d = 32, \lambda = 10$, and vary k in the range of [0, 8]. There is an improvement phase of the detection performance when increasing k. It demonstrates that the APT detection task should consider multiple system entities and their contexts. Furthermore, the improvement trend in the Lab Dataset is quite different from that in the DARPA Dataset. In the Lab Dataset, there is a dramatic improvement phase when increasing k from 0 to 2, followed by a stable phase. In the DARPA Dataset, there is a constant improvement phase when increasing k from 0 to 6. The reason is that the attacks in the DARPA Dataset are stealthier

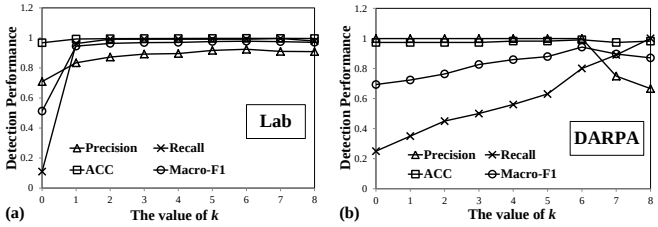

Fig. 8. The effect of parameter k: (a) the experiment results on Lab Dataset; (b) the experiment results on DARPA Dataset.

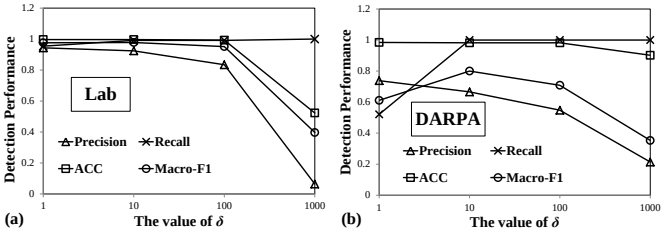

Fig. 9. The effect of parameter δ: (a) the experiment results on Lab Dataset; (b) the experiment results on DARPA Dataset.

and span a longer period of time, and thus it involves more system entities to capture their contexts. To this end, we set $k = 8$ in the following experiments, since the best Recall is achieved at $k = 8$. Recall is more important than other performance metrics, the reason of which will be discussed in Section 5.4.

In the second experiment, we investigate the impact of the malicious sample weight δ. We vary δ in the range of [1, 1000] and the experiment result is shown in Fig. 9. It can be found that setting δ too large or too small would result in poor performance. Setting δ too large would cause overfitting problem (i.e., low ACC and low Precision), while setting too δ small would cause the model to tend to ignore the malicious samples (i.e., low Recall). To this end, we set $\delta = 10$ in the following experiments.

5.3 Experiment 2: Ablation Experiment

In the first experiment, we try to evaluate APT-KGL under different evaluation strategies, i.e., In-Sample, Out-Sample, and No-Sample. The experiment results are shown in Fig. 10. First, Out-Sample has a lower Precision and almost the same Recall as compared with In-Sample, which means that the false alarm rate increases to a certain extent on unknown processes. This result is reasonable that the detection performance would be affected without a full view of the relations between the target process and other system entities. However, Out-Sample has almost the same ACC and a slight lower Macro-F1 as compared with In-Sample. It demonstrates that APT-KGL could still guarantee a satisfied overall detection performance on unknown processes.

Second, No-Sample has the lowest performance and we give a detailed discussion as follows. On one hand, TTPs try to describe all procedures that have any possibility of being attack related, even if the same procedures may also be employed for normal purposes [45]. Therefore, some training samples generated from TTPs could be confused

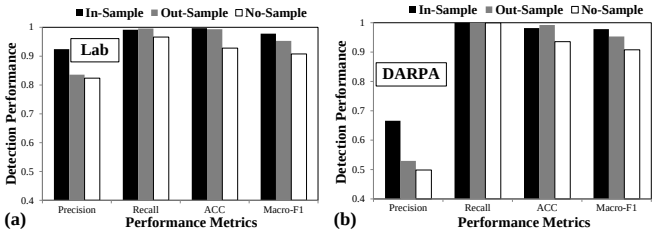

Fig. 10. The evaluation of APT-KGL under different evaluation strategies: (a) the experiment results on Lab Dataset; (b) the experiment results on DARPA Dataset.

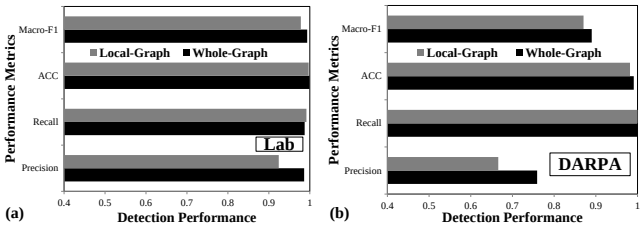

Fig. 11. The evaluation of the local graph sampling component: (a) the experiment results on Lab Dataset; (b) the experiment results on DARPA Dataset.

with normal behaviors, leading to a relatively high false alarm of No-Sample. On the other hand, No-Sample has almost the same Recall as compared with In-Sample. Since we collected CTIs and TTPs to cover all the types of attacks in Lab Dataset and DARPA Dataset, it indicates that these attacks follow the strategies defined in these CTIs and TTPs.

In the second experiment, we try to verify the effectiveness of the local graph sampling component of APT-KGL. To be specific, we evaluate the detection performance of the following two methods. In this experiment, In-Sample evaluation strategy is utilized.

1) Whole-Graph: It treats the whole HPG as the local graph.
2) Local-Graph: It samples the 8-order sub-graph as the local graph.

The detection performance is shown in Fig. 11. From the figure we can see that Whole-Graph outperforms Local-Graph to a slight degree. It implies that some APT attacks are very stealthy, i.e., they do not perform malicious activities immediately after entering the system, but often conceal themselves for a long time and trigger the attacks by an alternative system entity. It would result in a long distance between the malicious process and the manipulated system entity performing the malicious activities in the HPG. However, Whole-Graph would lead to significantly higher computational overhead, which is investigated in Section 5.4.

In the third experiment, we try to verify the effectiveness of the heterogeneous graph embedding component of APT-KGL. Specifically, we test the detection performance of combinations of different meta-paths (i.e., MPC_1 to MPC_6 as follows) on the Lab Dataset. In this experiment, In-Sample evaluation strategy is utilized.

1) MPC_1: It includes the meta-paths that only contain process and file (i.e., MP1 and MP3).
2) MPC_2: It includes the meta-paths that only contain process, file, network, and socket (i.e., MP1, MP3, MP6, and MP8).
3) MPC_3: It includes the meta-paths that only contain process, file, IPC, and memory (i.e., MP1, MP3, MP5, and MP7).
4) MPC_4: It includes the meta-paths that only contain process, file, and file attribute (i.e., MP1, MP3, and MP4).
5) MPC_5: It includes the meta-paths that only contain process, file, and process attribute (i.e., MP1, MP2, and MP3).
6) MPC_6: It includes all meta-paths.

The experiment results are shown in Table 5. First, MPC_6 has the best overall detection performance. It demonstrates that all the meta-paths contribute to the APT detection task, which also implies that APT attacks are highly complex and it is impossible to detect all APT attacks by only considering certain types of system entities and system events. Second, MPC_4 and MPC_5 outperform MPC_1. It shows that process and file attributes can provider richer information to improve the detection performance. Third, MPC_4 outperforms all other methods, except for MPC_6. It means that file attributes are the most important information for APT detection. This result is in consistent with the existing experience that stealing sensitive data from files or performing sensitive operations on files is one of the most important reasons for APT attacks [10]. Fourth, MPC_2 outperforms MPC_3 and MPC_5. It implies that accessing the network is a more important characteristic of APT attacks than manipulating the processes and memory.

TABLE 5
THE EVALUATION OF DIFFERENT META-PATH COMBINA-TIONS

	Precision	Recall	ACC	Macro-F1
MPC_1	0.8505	0.9917	0.9939	0.9566
MPC_2	0.8892	0.9917	0.9956	0.9680
MPC_3	0.8698	0.9959	0.9949	0.9634
MPC_4	0.9036	**1.0000**	0.9964	0.9741
MPC_5	0.8542	0.9959	0.9942	0.9587
MPC_6	**0.9242**	0.9917	**0.9970**	**0.9780**

5.4 Experiment 3: Overhead Experiment

We study the execution speed and memory usage of APT-KGL. Since most computational expensive operations (e.g., the HPG embedding) are executed offline, this section focuses on APT-KGL's runtime overhead (i.e., the APT detector in Section 4.3).

During the online execution phase, APT-KGL detects APT attacks in a local graph, and thus the computation overhead mainly depends on the size of the local graph. Specifically, we monitor the execution speed and memory usage on local graphs with different number of nodes. The results are shown in Table 6, where "Execution Time" indicates the time for performing convolution operation on the local graph and classifying a single node. As the number of nodes increases, the Execution Time becomes slightly longer, while the memory usage exhibits a significant growth. Note that the execution time and memory usage do not increase linearly, since there exists common computation overhead for building and storing the HPGs. The number of nodes in the DARPA dataset is near 300 thousands by collecting provenance data from three hosts for only one day, and thus the experiment results demonstrate that the computation overhead is unacceptable for processing the whole HPGs.

In the two datasets, the average number of nodes in the actual local graphs is approximately 100 by setting $k = 8$. As shown in Table 6, the execution time and memory usage satisfy the requirement of real-time processing.

5.5 Experiment 4: Comparison Experiment

In the first experiment, we evaluate the general detection performance of APT-KGL by comparing it with the following four baselines. All the baselines have considered the class imbalance problem and assign different weights to samples (10:1 for malicious samples and benign samples) during the training process. In this experiment, In-Sample evaluation strategy is utilized.

1) SVM: It refers to the process classification model,which uses the vector space model to create the feature vector (each entry represents the number of nodes of a certain type in the first-order neighbors of the target process node in the HPG) and the SVM as classifier.
2) GAT: It refers to a homogeneous graph neural network [46]. Specifically, it first constructs a homogeneous graph of the same topology with HPG (in Section 4.4.1), but ignores the types of nodes, and then performs node embedding and classification based on the GAT model.
3) HGAT: It refers to a heterogeneous graph neural network [36]. Specifically, it first constructs the HPG, and then performs node embedding and classification based on the HGAT model, without the local graph sampling and the R-GCN steps. It can be viewed as a variant of HPG-APT by only looking at one process.
4) CONAN: It refers to a state-of-the-art APT detection model based on provenance analysis [10]. It manually defines a large number of rules (called atomic suspicious indicators) from four aspects (i.e., code source, behavior, feature and network) to support the APT detection task.

The experiment results are shown in Table 7. First, GAT is completely unable to detect APT attacks, because that GAT only considers the interactions between system entities but ignores all the semantics (e.g., the type of system entities, the context of system events, etc.), which are essential for detecting malicious activities. Second, SVM also has a poor detection performance, because that SVM only considers the types of system entities adjacent to the target process but ignores the downstream activities of these system entities. In practice, the attacker would

TABLE 6
THE COMPUTATION OVERHEAD OF WHOLE-GRAPH

	The number of nodes in the local-graph					Actual local graph
	50 nodes	500 nodes	5000 nodes	50000 nodes	90000 nodes	
Memory Usage	0.821GB	0.981GB	1.439GB	6.361GB	10.263GB	0.845GB
Execution Time	0.1558s	0.1602s	0.1669s	0.6568s	2.1332s	0.1598s

always conceal themselves for a certain period of time before performing the malicious activities. Third, APT-KGL outperforms HGAT. It again shows that it is difficult to detect APT attacks by only looking at one system entity. Fourth, CONAN has a slight advantage over APT-KGL on the Lab Dataset, while APT-KGL outperforms CONAN on the DARPA Dataset. CONAN is a rule-based APT detection system manually designed based on an in-depth analysis on the attack samples. Although the rules could perfectly adapt to the analysed attack samples, they cannot handle the changing patterns and thus exhibt a far lower performance in a different macro environment. On the other hand, APT-KGL is totally data-driven. It can be retrained periodically or when new attack samples are collected, to automatically adapt to the change of macro environment. Thus, APT-KGL has significant advantage over CONAN on generalization ability.

TABLE 7
THE COMPARISON OF DIFFERENT IN-SAMPLE DETECTION METHODS

	Precision	Recall	ACC	Macro-F1
Lab Dataset:				
SVM	**1.0000**	0.0457	0.9640	0.5042
GAT	0.0000	0.0000	0.9669	0.4916
HGAT	0.7094	0.1093	0.9662	0.5125
CONAN	0.9747	**0.9982**	**0.9982**	**0.9916**
APT-KGL	0.9242	0.9917	0.9970	0.9780
DAPRA Dataset:				
SVM	1.0000	0.1253	0.9236	0.5710
GAT	0.0000	0.0000	0.9731	0.4932
HGAT	**1.0000**	0.2491	0.9731	0.6933
CONAN	0.5889	1.0000	0.9758	0.8396
APT-KGL	0.6660	**1.0000**	**0.9819**	**0.8704**

Someone might argue that the detection performance is far lower than those reported in previous studies [15] [16] [44], which generally achieve almost 100% detection precision with nearly zero false alarms. This is because that the evaluation strategies are different. The previous studies evaluate the detection performance in "attack-level". Specifically, if any system entity involved in the attack chain of an APT is detected, the APT can be considered as being correctly detected. The rationale is that the attack chain can be reconstructed by traveling through the provenance graph starting from the detected node. On the other hand, our experiment evaluates the detection performance in "node-level". Since only the attack origin and the attack suspects performing actual malicious activities are labeled as "malicious", a large majority of the false alarms are system entities involved in the attack chain but not being labeled as "malicious". It means that most false alarms of APT-KGL can be tolerated in practice. This is also the reason that we consider Recall as more important than other performance metrics.

In the second experiment, we focus on the detection performance on unknown processes by comparing APT-KGL with the following two baselines. In this experiment, Out-Sample evaluation strategy is utilized.

1) NeighborAvg: It is a variant of APT-KGL. Specifically, a new incoming unknown process is linked to the existing HPG and represented by averaging the embeddings of its neighbors.
2) LabelProp: It is a variant of APT-KGL. Specifically, a new incoming unknown process is first linked to the existing HPG, and then Label Propagation (a semi-supervised learning algorithm) [47] is used to propagate labels of the existing nodes to the unknown process.

The experiment results are shown in Table 8. First, NeighborAvg has a poor detection performance. It is because that NeighborAvg only considers the first-order neighbors and cannot adapt to the stealth of APTs, which would often result in a long distance between the malicious process and the system entity performing the malicious activities in the HPG. Therefore, LabelProp has a far better performance than NeighborAvg, since it considers higher order correlations by propagating labels to a long distance. Second, APT-KGL outperforms LabelProp. It is because that LabelProp only transfers malicious labels without capturing the patterns of malicious activities. There are usually no malicious system entities in a certain range of the malicious process, especially only a local graph is sampled for detection.

TABLE 8
THE COMPARISON OF DIFFERENT OUT-SAMPLE DETECTION METHODS

	Precision	Recall	ACC	Macro-F1
Lab Dataset:				
NeighborAvg	0.9141	0.1416	0.9150	0.5635
LabelProp	**0.9609**	0.5068	0.9677	0.8235
APT-KGL	0.8360	**0.9959**	**0.9934**	**0.9531**
DAPRA Dataset:				
NeighborAvg	0.6672	0.1258	0.8838	0.5740
LabelProp	**0.7492**	0.3312	0.9349	0.6971
APT-KGL	0.5294	**1.0000**	**0.9923**	**0.8315**

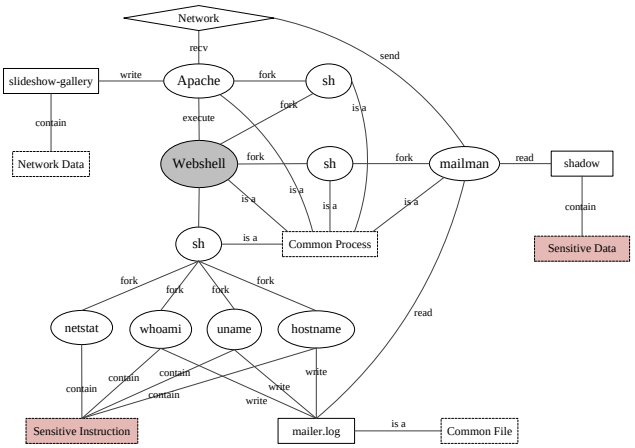

Fig. 12. The HPG of a real attack case.

5.6 Case Study

In this section, we provide a real attack case to illustrate how APT-KGL works. Fig. 1 shows the provenance graph of the attack scenario, where the diamonds represent network, the ovals represent processes, and the rectangles represents files. In this case, the attacker first penetrates the Apache server and leaves a backdoor. When the attacker triggers the backdoor, it creates a process (i.e., "Webshell"). Then, "Webshell" forks several processes to perform sensitive commands (e.g., "netstat" to check the detailed network information) and writes the collected information to a file (i.e., "mailer.log"). Finally, "Webshell" executes a process (i.e., "mailman") to collect the sensitive information from "mailer.log" and another sensitive file (i.e., "shadow"), and send to a remote server.

The HPG transformed from Fig. 1 is partially shown in Fig. 12, where the rectangles with dashed border represent attributes. First, according to [10], the final purpose of APTs is stealing sensitive information or causing damage, and thus they usually share some common features (e.g., performing sensitive instructions, reading sensitive files, etc.). These common features can be learnt by the meta-path based heterogeneous graph embedding. Second, there is usually a certain distance between the original malicious process and the system entities performing the malicious activities (e.g., "Webshell" and "netstat", "Webshell" and "reading from shadow"). Therefore, only looking at one system entity is unable to detect such kind of APT attacks, while APT-KGL can model a chain of system entities based on local graph sampling and convolution operations. For example, take "Webshell" as the process to be classified, if we sample a 3-order sub-graph from it as the local graph, the "Sensitive Instruction" feature can be propagated to it through convolution operations. If we sample a 4-order sub-graph from it as the local graph, the "Sensitive Data" feature can also be propagated to it. These features can potentially help the model to decide "Webshell" is a malicious process.

6 CONCLUSION AND FUTURE WORK

In this paper, we investigate the APT detection problem based on provenance data. We propose APT-KGL, an intelligent APT detection system based on deep learning technique. Specifically, APT-KGL adopts a heterogeneous provenance graph to model all system entities and events, and learns a low-dimensional vector to represent each system entity in a scalable way. Then, APT-KGL reconstructs the attack scene by sampling a small sub-graph from the provenance graph and detects APT attacks on this sub-graph. Based on the above designs, APT-KGL could effectively detect APT attacks with the characteristics of persistence, stealth, and diversity. Through a series of experiments based on real provenance datasets containing APT attacks, we demonstrate that APT-KGL outperforms other current machine learning and deep learning based models, and has competitive performance as compared with state-of-the-art rule-based APT detection systems designed by excessive domain knowledge.

In the future, we will extend our work from the following directions. First, APT-KGL still focuses on detecting the "APT attack point" (i.e., an individual attack behavior). However, a complete APT attack chain is usually comprised of many APT attack behaviors with quite long time intervals. Thus, how to detecting the "APT attack chain" is a problem worthy of study. Second, APT-KGL provides attack detection results, but without the explanation of the results, which is essential for the works such as attack attribution and threat level analysis. Therefore, how to build an explainable APT detection model is also a problem worthy of study.

ACKNOWLEDGMENTS

This work was supported in part by a grant from XYZ.

REFERENCES

[1] A. Alshamrani, S. Myneni, A. Chowdhary, and D. Huang, "A survey on advanced persistent threats: Techniques, solutions, challenges, and research opportunities," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1851–1877, 2019.

[2] R. A. Bridges, T. R. Glass-Vanderlan, M. D. Iannacone, M. S. Vincent, and Q. Chen, "A survey of intrusion detection systems leveraging host data," *ACM Computing Surveys (CSUR)*, vol. 52, no. 6, pp. 1–35, 2019.

[3] A. Singla, E. Bertino, and D. Verma, "Preparing network intrusion detection deep learning models with minimal data using adversarial domain adaptation," in *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, 2020, pp. 127–140.

[4] X. Han, T. Pasquier, and M. Seltzer, "Provenance-based intrusion detection: opportunities and challenges," in *10th USENIX Workshop on the Theory and Practice of Provenance (TaPP 2018)*, 2018.

[5] G. Jenkinson, L. Carata, T. Bytheway, R. Sohan, R. N. Watson, J. Anderson, B. Kidney, A. Strnad, A. Thomas, and G. Neville-Neil, "Applying provenance in {APT} monitoring and analysis: Practical challenges for scalable, efficient and trustworthy distributed provenance," in *9th USENIX Workshop on the Theory and Practice of Provenance (TaPP 2017)*, 2017.

[6] X. Han, T. Pasquier, A. Bates, J. Mickens, and M. Seltzer, "Unicorn: Runtime provenance-based detector for advanced persistent threats," *arXiv preprint arXiv:2001.01525*, 2020.

[7] A. Bates, D. J. Tian, K. R. Butler, and T. Moyer, "Trustworthy {Whole-System} provenance for the linux kernel," in *24th USENIX Security Symposium (USENIX Security 15)*, 2015, pp. 319–334.

[8] 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.

[9] 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.

[10] C. Xiong, T. Zhu, W. Dong, L. Ruan, R. Yang, Y. Chen, Y. Cheng, S. Cheng, and X. Chen, "Conan: A practical real-time apt detection system with high accuracy and efficiency," *IEEE Transactions on Dependable and Secure Computing*, 2020.

[11] T. Yadav and A. Rao, "Technical aspects of cyber kill chain in: International symposium on security in computing and communication, 438–452," 2015.

[12] M. ATT&CK, "Mitre att&ck," *URL: https://attack. mitre. org*, 2021.

[13] Y. Ye, T. Li, D. Adjeroh, and S. S. Iyengar, "A survey on malware detection using data mining techniques," *ACM Computing Surveys (CSUR)*, vol. 50, no. 3, pp. 1–40, 2017.

[14] X. Zhang, Y. Zhang, M. Zhong, D. Ding, Y. Cao, Y. Zhang, M. Zhang, and M. Yang, "Enhancing state-of-the-art classifiers with api semantics to detect evolved android malware," in *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security*, 2020, pp. 757–770.

[15] H. Weng, Z. Li, S. Ji, C. Chu, H. Lu, T. Du, and Q. He, "Online e-commerce fraud: a large-scale detection and analysis," in *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, 2018, pp. 1435–1440.

[16] B. Branco, P. Abreu, A. S. Gomes, M. S. Almeida, J. T. Ascensão, and P. Bizarro, "Interleaved sequence rnns for fraud detection," in *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, 2020, pp. 3101–3109.

[17] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 1, pp. 4–24, 2020.

[18] Y. Sun and J. Han, "Mining heterogeneous information networks: a structural analysis approach," *Acm Sigkdd Explorations Newsletter*, vol. 14, no. 2, pp. 20–28, 2013.

[19] Y. Dong, N. V. Chawla, and A. Swami, "metapath2vec: Scalable representation learning for heterogeneous networks," in *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, 2017, pp. 135–144.

[20] 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.

[21] 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.

[22] G. Berrada, J. Cheney, S. Benabderrahmane, W. Maxwell, H. Mookherjee, A. Theriault, and R. Wright, "A baseline for unsupervised advanced persistent threat detection in system-level provenance," *Future Generation Computer Systems*, vol. 108, pp. 401–413, 2020.

[23] Z. Xiang, D. Guo, and Q. Li, "Detecting mobile advanced persistent threats based on large-scale dns logs," *Computers & Security*, vol. 96, p. 101933, 2020.

[24] R. Coulter, J. Zhang, L. Pan, and Y. Xiang, "Domain adaptation for windows advanced persistent threat detection," *Computers & Security*, vol. 112, p. 102496, 2022.

[25] Y. LeCun, Y. Bengio, G. Hinton *et al.*, "Deep learning. nature, 521 (7553), 436-444," *Google Scholar Google Scholar Cross Ref Cross Ref*, 2015.

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

[27] Y. Shen, E. Mariconti, P. A. Vervier, and G. Stringhini, "Tiresias: Predicting security events through deep learning," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 592–605.

[28] H. N. Eke, A. Petrovski, and H. Ahriz, "The use of machine learning algorithms for detecting advanced persistent threats," in *Proceedings of the 12th International Conference on Security of Information and Networks*, 2019, pp. 1–8.

[29] S. Wang, Z. Wang, T. Zhou, X. Yin, D. Han, H. Zhang, H. Sun, X. Shi, and J. Yang, "threatrace: Detecting and tracing host-based threats in node level through provenance graph learning," *arXiv preprint arXiv:2111.04333*, 2021.

[30] 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.

[31] 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.

[32] J. Wang, P. Huang, H. Zhao, Z. Zhang, B. Zhao, and D. L. Lee, "Billion-scale commodity embedding for e-commerce recommendation in alibaba," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 839–848.

[33] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 701–710.

[34] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016, pp. 855–864.

[35] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *Proceedings of the 24th international conference on world wide web*, 2015, pp. 1067–1077.

[36] X. Wang, H. Ji, C. Shi, B. Wang, Y. Ye, P. Cui, and P. S. Yu, "Heterogeneous graph attention network," in *The world wide web conference*, 2019, pp. 2022–2032.

[37] D. Zhang, J. Yin, X. Zhu, and C. Zhang, "Metagraph2vec: Complex semantic path augmented heterogeneous network embedding," in *Pacific-Asia conference on knowledge discovery and data mining*. Springer, 2018, pp. 196–208.

[38] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. v. d. Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," in *European semantic web conference*. Springer, 2018, pp. 593–607.

[39] 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.

[40] K. Satvat, R. Gjomemo, and V. Venkatakrishnan, "Extractor: Extracting attack behavior from threat reports," in *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2021, pp. 598–615.

[41] G. Husari, E. Al-Shaer, M. Ahmed, B. Chu, and X. Niu, "Ttpdrill: Automatic and accurate extraction of threat actions from unstructured text of cti sources," in *Proceedings of the 33rd annual computer security applications conference*, 2017, pp. 103–115.

[42] A. Gehani and D. Tariq, "Spade: Support for provenance auditing in distributed environments," in *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*. Springer, 2012, pp. 101–120.

[43] "Atomic red team." [Online]. Available: https://github.com/redcanaryco/atomic-red-team

[44] "Kellect." [Online]. Available: https://github.com/acising/kellect/tree/master

[45] 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)*. IEEE, 2020, pp. 1172–1189.

[46] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *stat*, vol. 1050, p. 20, 2017.

[47] X. Zhu and Z. Ghahramani, "Learning from labeled and unlabeled data with label propagation," 2002.

Tieming Chen received the Ph.D. degree in software engineering from Beihang University, China. He is currently a full professor with the college of computer science and technology, Zhejiang University of Technology, China. His research interests include data mining and cyberspace security.

Chengyu Dong received the B.S. degree in network engineering from the Ningbo University of Technology, China, in 2019. His current research interests machine learning and graph neural networks.

MingqiLv received the Ph.D. degree in Computer Science from Zhejiang University, Hangzhou, China, in 2012. He is currently an associated professor with the College of Computer Science and Technology, Zhejiang University of Technology, China. His research interests include spatiotemporal data mining and cyber security.

Qijie Song received the B.E. degree in electronic and engineering from Zhejiang Sci-tech University, Zhejiang, China, in 2016, and the M.S. degress in sofeware engineering from Zhejiang University of technology, Zhejiang, China, 2020. He is currently pursuing Ph.D. degree in computer science in Zhejiang University of Technology. His current research interests include system security.

Haiwen Liu received the B.S. degree in Information System and Information Management from Jiangxi Agricultural University, China, in 2020. His current research interests machine learning and graph neural networks.

Tiantian Zhu received the Ph.D. degree in computer science from Zhejiang University, Hangzhou, China, in 2019. He is currently a lecturer with the college of computer science and technology, Zhejiang University of Technology, China. His research interests include data mining, artificial intelligence, and information security.

Kang Xu received the B.S. degree in network engineering from the Zhejiang University of Technology, China, in 2018. His current research interests machine learning and network security.

Ling Chen received the B.S. and Ph.D. degrees in computer science from Zhejiang University in 1999 and 2004, respectively. He is currently an associate professor with the college of computer science and technology, Zhejiang University, China. His research interests include ubiquitous computing, human computer interaction and pattern recognition.

Shouling Ji is a ZJU 100-Young Professor in the College of Computer Science and Technology at Zhejiang University and a Research Faculty in the School of Electrical and Computer Engineering at Georgia Institute of Technology. He received a Ph.D. in Electrical and Computer Engineering from Georgia Institute of Technology and a Ph.D. in Computer Science from Georgia State University. His current research interests include AI and Security, Data-driven Security and Data Analytics. He is a member of IEEE and ACM.

Yuan Fan recieved the M.S. degree from San Jose State University. He is the founder and CEO of DAS-Security, Hangzhou. He has a deep study in intrusion detection, cloud security, IoT system security, and indus-trial Internet security.