AttackMiner: A Graph Neural Network Based Approach for Attack Detection from Audit Logs

Yuedong Pan[1,2], Lijun Cai[1(✉)], Tao Leng[1,2], Lixin Zhao[1], Jiangang Ma[1], Aimin Yu[1], and Dan Meng[1]

[1] Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
{panyuedong,cailijun,lengtao,zhaolixin,majiangang,yuaimin, mengdan}@iie.ac.cn

[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China

Abstract. In an enterprise environment, intrusion detection systems generate many threat alerts on anomalous events every day, and these alerts may involve certain steps of a long-dormant advanced persistent threat (APT). In this paper, we present AttackMiner, an attack detection framework that combines contextual information from audit logs. Our main observation is that the same attack behavior may occur in various possible contexts, and combining various possible contextual information can provide more effective information for detecting such attacks. We utilize a combination of provenance graph causal analysis and deep learning techniques to build a graph-structure-based model that builds key patterns of attack graphs and benign graphs from audit logs. During detection, the detection system creates provenance graphs using the input audit logs. After being optimized by our customized graph optimization mechanism, it identifies whether an attack has occurred. Our evaluations on the DARPA TC dataset show that AttackMiner can successfully detect attack behaviors with high accuracy and efficiency. Through this effort, we provide security investigators with a new approach of identifying attack activity from audit logs.

Keywords: Host-based intrusion detection · Graph neural network · Attack migration

1 Introduction

Advanced Persistent Threats (APTs), as opposed to regular attacks, are a type of sophisticated attack performed by experienced adversaries employing a variety of offensive strategies and tools [23]. APTs typically involve multiple attack steps over an extended period of time. When a security analyst wants to determine whether an APT has occurred inside the system, he usually needs to conduct a lengthy and complex search and analysis of massive logs, and identify whether there are typical attack behaviors in these logs. This makes these

F. Li et al. (Eds.): SecureComm 2022, LNICST 462, pp. 510–528, 2023.
https://doi.org/10.1007/978-3-031-25538-0_27

attacks difficult to detect. Traditional detectors [7,25] can detect anomalous actions that do not fit into previously learned benign patterns. However, attackers can easily get around them because they treat system calls or network events as temporal sequences [6,7,29] and only consider the sequential relationship between log entries. Existing attack detection and response technologies (e.g., endpoint detection and response tools) rely on low-level indicators of compromise (IOCs) or adversarial tactics, techniques, and procedures (TTPs) criteria [1] being matched. However, simple rule-matching approaches are prone to a huge number of false positives, resulting in "alert fatigue".

Recent research [12,13,25–27,38] suggests that host-based intrusion detection might benefit from abundant contextual knowledge regarding data provenance. Compared to raw system audit data, data provenance offers more contextual information, which helps identify malicious behaviors from benign behaviors [12, 14]. Some anomaly-based graph kernel algorithms [12,25] dynamically model the entire graph and detect anomalous graphs through clustering methods. However, the provenance graph produced by covert intrusion activities carried out in a system may be similar to a benign system. Therefore, it is difficult for the graph kernel algorithm to detect the attack behavior with a few abnormal nodes in the feature graph. To a certain extent, malicious behavior can be detected by detecting anomalous paths in the provenance graph [38]. However, the activity of some complex threats (e.g., APTs) is frequently divided into several parts rather than appearing in a complete path, making path-level detection difficult.

In this paper, we aim to identify the key steps of attack activities from audit logs, helping security analysts identify whether a given audit log contains specific malicious behavior. We propose AttackMiner, a graph neural network-based attack detector capable of detecting attack patterns in various contextual scenarios. AttackMiner takes system audit logs as inputs and uses a graph neural network framework to learn rich contextual information from data sources. AttackMiner mainly includes three stages: (a) processing audit logs and creating provenance graphs based on the log sliding window; (b) constructing attack graphs containing specific attack patterns using attack migration and mutation techniques; and (c) learning to represent attack semantics using a graph neural network model, which aids in accurately determining whether a given audit log contains a specific attack pattern at detection time. Additionally, security analysts can conduct fast attack investigations from detected logs that contain attack behavior, saving significant time compared to investigation activity on massive logs.

Our approach is based on the insight that some complex malicious activities (e.g., APTs) are usually divided into several parts rather than appearing in a complete path. A 14-stage APT knowledge base is provided by the MITRE ATT&CK framework [1] to characterize adversary plans and methodologies for APTs. The Lockheed Martin Cyber Kill Chain [17] is a 7-stage methodology for characterizing APTs. However, the multiple stages of APTs span a long time and generate a large number of logs. This results in significant processing and storage overhead for over-redundant multi-stage models. Therefore, we focus on identifying key actions in the attack activity.

In summary, this paper makes the following contributions:

1. We propose AttackMiner, a GNN-based attack detection framework. AttackMiner combines model learning techniques for natural language and provenance graph processing to help security analysts quickly detect attacks from audit logs.
2. We focus on attack migration, migrating various behaviors representing attacks into various possible log contexts. Through attack migration, attack behaviors with rich backgrounds can be constructed, providing rich data for model learning.
3. We implemented a set of provenance graph optimization methods that greatly reduce the number of redundant edges and nodes. The optimized provenance graph allows AttackMiner to build efficient GNN-based models to accurately detect attack activity.
4. We present a concrete implementation and evaluation of AttackMiner. The experimental results show that AttackMiner can accurately identify various attack behaviors, and the detection effect is better than other advanced attack behavior detectors.

The paper is organized as follows. Related work is introduced in Sect. 2. Motivation and assumptions about our work are introduced in Sect. 3. We introduce the overview of AttackMiner in Sect. 4. In Sect. 5, we provide the formal details of AttackMiner. The evaluation and conclusion are presented in Sect. 6 and Sect. 7, respectively.

2 Related Work

2.1 Log-Based Attack Analysis

Many works [9,10,24,32] use system audit logs to perform attack detection and forensic analysis. Disclosure [5] extracts statistical features from NetFlow logs to detect botnet C&C channels. Opera et al. [28] utilize DNS or web proxy logs to detect early infections in the enterprise. LogLens [6] is a real-time anomaly detection system that deploys an unsupervised learning approach to analyze log sequences. DeepLog [7] converts system logs into natural language sequences using the Long Short-term Memory (LSTM) network that can automatically learn benign patterns and alert anomalies from system behavior. Tiresias [34] utilizes Recurrent Neural Networks (RNNs) in logs to predict specific attack steps. Most of these log analysis methods treat the log as temporal sequences, which only preserves sequential relationships between log entries. AttackMiner takes into account the spatial and interactive relationships between system entities through the message-passing mechanism of graph neural networks.

2.2 Provenance Graph-Based Attack Detection

Provenance graph analysis is widely used in APT attack detection [35], forensic analysis [16], and attack scenario reconstruction [15,28]. Holmes [27] and

RapSheet [13] focus on alert generation, correlation, and scenario reconstruction for host-based threats, but they rely on the knowledge base of adversarial TTPs. Log2vec [23] uses logs to construct compositions, extracts log vectors based on graph embedding, and detects malicious logs based on clustering. It differs from the provenance-based approach because the nodes in the provenance graph are entities of the system rather than logs. StreamSpot [25] is a memory-efficient anomaly detection system that handles provenance graph heterogeneity and streaming challenges, but suffers from shortcomings in handling locally constrained graph features and dynamic cluster maintenance. With only a certain amount of benign data and a small amount of attack data, our method can learn key attack steps via the provenance graph. Models trained with optimized provenance graphs have better detection capabilities.

3 Motivation and Assumptions

Motivation. We define the attack detection problem as a subgraph detection problem. Specifically, we detect whether the provenance graph contains a subgraph representing an attack. If it does, it means that attack events have occurred. Otherwise, no attacks have occurred. The key insight behind our method is that the attack behavior expressed by the attack subgraph may occur at various moments when the system is running. In the context of different audit logs of the system, similar attacks have similar attack patterns, and the attack pattern is learned through deep learning. The representation of this context helps to detect attacks. Specifically, Fig. 1 shows a portion of the provenance graph where the attack occurs, with the shaded area representing a key attack behavior. This attack mainly starts by exploiting the vulnerability of Nginx. The attacker downloads and executes the malicious file $/tmp/vUgefal$, triggering an alert for file execution. The attacker then writes to another file $/var/log/devc$, communicates with the malicious server, and even attempts to inject into the *sshd* process. Likewise, this kind of aggressive behavior can occur in other situations as well. As long as attackers exploit the vulnerability of the software to invade the system, even if the system is running a background task different from that in Fig. 1, the attack behavior in Fig. 1 can be achieved.

That is to say, if the attacker performs some malicious behaviors after invading the system by exploiting a certain software vulnerability this time, the next time he intrudes the system may be through other vulnerabilities and perform similar malicious behaviors.

Assumptions. We assume that the underlying operating system and audit application are part of a trusted computing base (TCB), similar to previous research on provenance tracking [4,30]. The audit logs used to build the provenance graph are hard to tamper with. The source of the attack is outside the enterprise. The attacker uses remote network access to infiltrate the system. The host audit system can normally capture a series of attack behaviors by attackers.

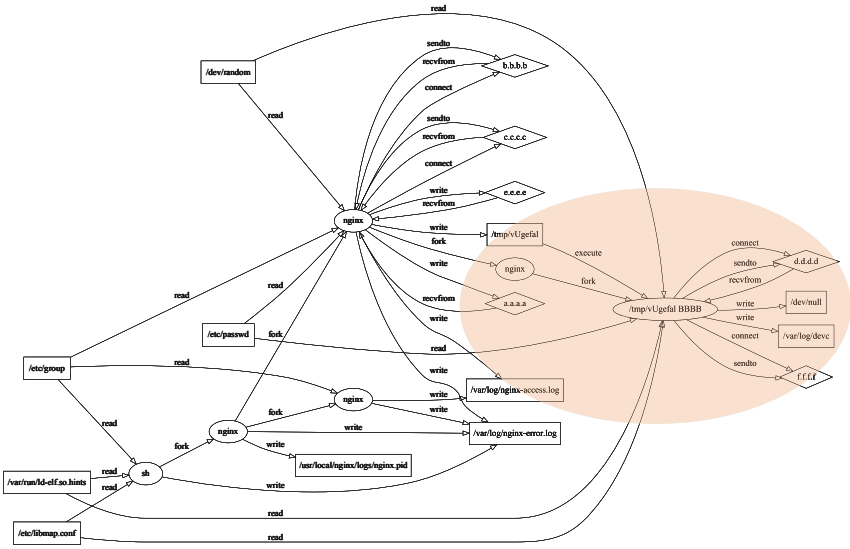


Fig. 1. A portion of the provenance graph where the attack occurs, with the shaded area representing a key attack behavior.

4 Approach Overview

4.1 Overview

AttackMiner is an attack detection tool that models attack and non-attack behaviors. Figure 2 gives an overview of the AttackMiner architecture. It mainly consists of four parts: dynamic log sliding window selection (a), provenance graph construction and optimization (b), deep learning model based on graph neural network (c), and attack detection (d). In the dynamic log window selection stage, we randomly and dynamically adjust the start time of two adjacent windows according to the time span of the current log window, which can improve the representativeness of the data to a certain extent. In the provenance graph construction stage, AttackMiner first constructs benign provenance graphs and then constructs malicious provenance graphs through attack migration technology. We adopt a series of graph optimization methods to reduce the size of the provenance graph while preserving the original semantics as much as possible. In the model training stage, AttackMiner uses a learning model based on a graph neural network to effectively embed the input data. During the attack detection stage, AttackMiner accurately determines whether an attack has taken place on given log files using a novel classifier we developed.

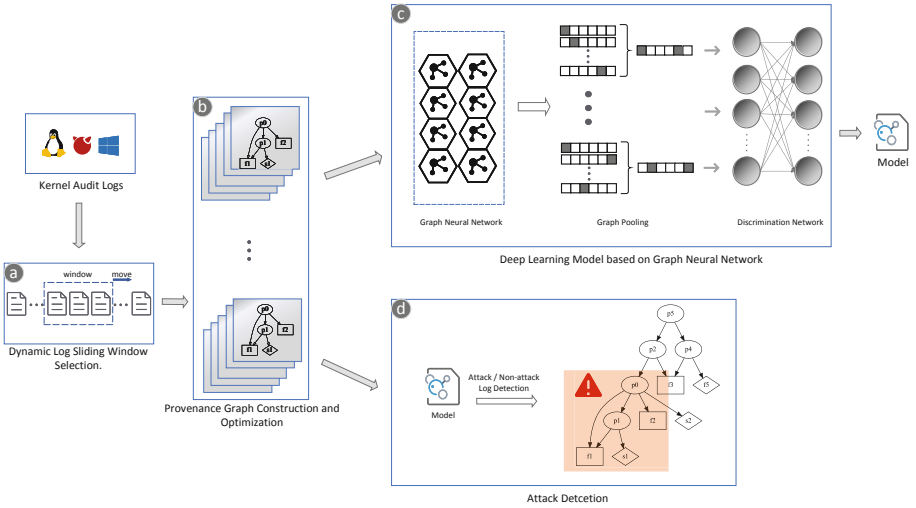


Fig. 2. Overview of AttackMiner architecture.

4.2 Challenges and Solutions

In this work, we first need to consider as much as possible the context in which the attack steps occur, and more importantly, we need to effectively aggregate the information contained in each provenance graph. Below, we present the challenges of designing AttackMiner and our corresponding solutions.

Challenge 1. How to construct provenance graphs to cover both benign and malicious behaviors as much as possible? Previous graph pattern matching models [2,21,39] analyzed small subgraphs but did not involve log window processing. In order to include as many benign and malicious behaviors as possible, we run a dynamic sliding window on the original logs (Sect. 5.1) and combine attack migration and attack mutation techniques (Sect. 5.3) to provide the model with realistic learning materials.

Challenge 2. How to optimize provenance graphs while keeping the original semantics as much as possible? With only a few behaviors in the massive audit log indicating a real attack, searching for traces of an attack is like "finding a needle in a haystack". In order to solve this problem, we adopt log compression methods based on semantic preservation (Sect. 5.2), which simplifies the provenance graph structure and improves the detection efficiency.

Challenge 3. How to effectively represent graph information? Previous work searches for provenance graph information by meta-path [25]. However, representing provenance graph information via meta-paths requires expert knowledge of the target system. Graph neural networks have been shown to learn complex graph patterns for downstream tasks such as memory forensic analysis [35] and binary code similarity detection [40]. In this work, we try to extract graph patterns with graph neural networks (Sect. 5.4).

5 AttackMiner

5.1 Log Window Sliding

Our main goal is not to detect a complete attack activity with a long time span but to focus on a typical steps of an attack, such as when the invaded process communicates with some malicious IP addresses after reading and writing a private file. These actions are usually completed within a certain time window.

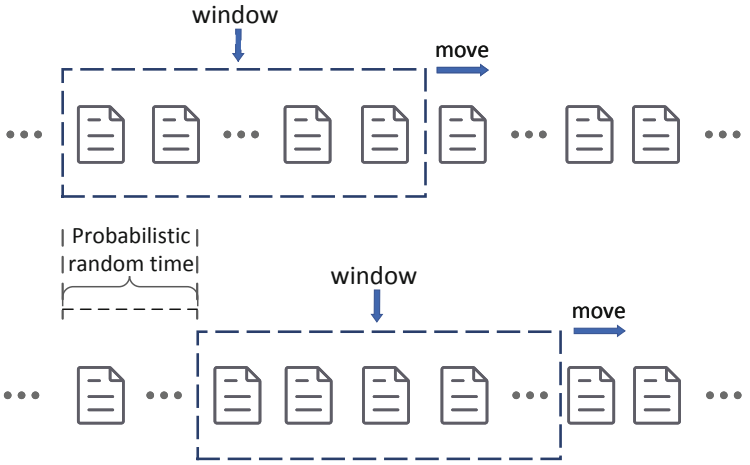


Fig. 3. Illustration of log sliding window in AttackMiner.

The log sliding window is used to select audit logs within a suitable time span for constructing provenance graphs. The two adjacent log windows are not completely separated. This is to cover various situations in the audit log as much as possible. Figure 3 is an illustration of a log sliding window. A window with a certain time span moves on the log in chronological order of events. After each move is completed, AttackMiner constructs the provenance graph with the log events in the current window. In order to cover as many events as possible, the distance that the window moves each time is a random time that varies according to the probability. Additionally, in order to gather useful information, we suggest choosing log windows with an appropriate length L based on the target behavior. According to the findings of Sect. 6.5, AttackMiner still has high accuracy when L is above 60 min.

5.2 Provenance Graph Construction and Optimization

AttackMiner first performs log window sliding on logs generated by benign environments and then converts the selected audit logs into a provenance graph. The provenance graph constructed with logs produced in benign environments

Table 1. A summarization of nodes and edges in provenance graphs

Subject type	Object type	Attributes	Relations
Process	Process	Type, Name	Fork/Clone
	File	Type, File Name	Read, Write, Execute
	Socket	Type, Src/Dest IP	Recv, Send, Connect

is called the benign provenance graph. The entities and events we use when building the provenance graph are shown in Table 1, where subjects and objects correspond to nodes in the graph and relations correspond to edges in the graph.

AttackMiner mainly uses three techniques for graph structure optimization.

First, AttackMiner eliminates isolated nodes in the provenance graph since these nodes do not have enough graph structure information. At the same time, we merge socket nodes with the same IP address connected to the same process. We do not differentiate whether a port is malicious or not in order to obtain a compact provenance graph.

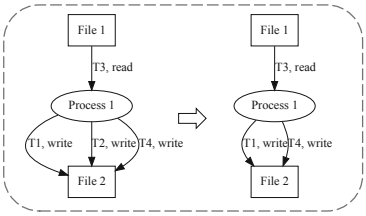


Fig. 4. Event reduction

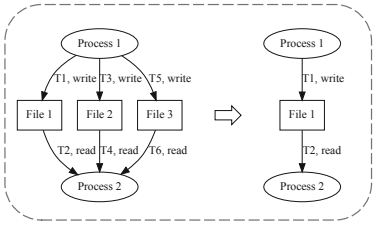


Fig. 5. Node reduction

Second, AttackMiner constructs provenance graphs from audit logs with distinct edges, removing redundant edges while preserving semantics [41]. When a node has an incoming edge, the incoming edge may affect the semantics of the node's outgoing edge. Specifically, as shown in Fig. 4, before the *read* at time $T3$ occurs, the *write* event at $T2$ is repeated with the *write* event at $T1$, so the *write* event at $T2$ can be removed. The *write* at $T4$ occurs after the *read* at $T3$, which affects $Process1$, so it should be retained.

Third, inspired by research [3], AttackMiner merges nodes and their edges whose incoming and outgoing edges are events of the same type. As shown in Fig. 5, file nodes $File1$, $File2$, and $File3$ are merged into one node $File1$ because they share same types of incoming edges (*read*) and outgoing edges (*write*).

5.3 Attack Provenance Graph Construction

We obtain attack information from threat intelligence or audit logs recording malicious behaviors, and use the extracted attack information to construct an attack subgraph G_a. Then we migrate G_a to the benign provenance graph G_b through the attack migration technique, and finally get a malicious provenance graph G_m containing the attack subgraph. Figure 6 depicts an example of attack migration.

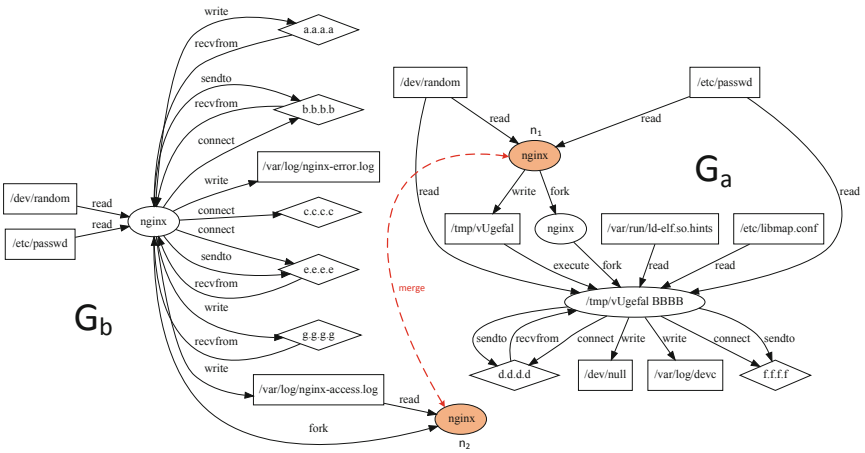


Fig. 6. Illustration of attack migration. (Left) G_b represents part of a benign provenance graph. (Right) G_a represents an attack subgraph that was extracted.

Attack Migration. The purpose of attack migration is to use various log contexts where malicious behaviors may occur as the background and construct attack samples with rich background information. So the trained model has the ability to shield the interference of background noise and pay attention to malicious behavior itself.

As shown in Fig. 6, G_b represents a benign provenance graph. For the sake of illustration, we have selected a portion of the benign provenance graph. The right half of the figure, G_a, is an attack subgraph extracted from attack campaigns. The attack in G_a invades the system through the vulnerability of *nginx*, and the shaded *nginx* node n_1 is the intrusion point for malicious behavior. In the benign provenance graph G_b, we find the target process node with the same type and name as the intrusion point (*nginx* node n_2 marked with shadow in G_b), and then we migrate the events associated with node n_1 to node n_2, so that node n_1 merges with n_2. Algorithm 1 explains our node merging algorithm in detail.

In addition, we randomly mutate the intruding process in the attack subgraph into another process in the benign provenance graph. And we mutate the name of the process with the same name as the intrusion process in the attack subgraph

Algorithm 1: Attack migration algorithm

Input: Benign provenance graph collection $List_{G_b}$; Attack subgraph collection $List_{G_a}$
Output: Malicious provenance graph $List_{G_m}$

$List_{Gm} \leftarrow Null$;
for G_b *in* $List_{G_b}$ **do**
  $\hat{V} \leftarrow$ random subject node in G_b ;
  $attr_{\hat{V}} \leftarrow$ attribute of $\hat{V}$;
  for Ga *in* $List_{Ga}$ **do**
    $V \leftarrow$ intrusion subject node in Ga ;
    `// change the attribute of` V`, merge` $\hat{V}$ `and` V
    $attr_V \leftarrow attr_{\hat{V}}$;
    `// migrate` G_a `to` G_b
    $Gm \leftarrow migrateGraph(Ga, G_b)$;
    $List_{Gm}$.append(G_m)
  end
end
return $List_{Gm}$

to the name after the mutation of the invading process. This process does not fundamentally change the attack pattern and, at the same time, increases the number of attack subgraphs that are not currently triggered but may appear in the future.

5.4 Deep Learning Model

Input Embedding. AttackMiner uses word2vec [20] for word embedding. We first convert complete events in the provenance graph into sentences, such as this sentence converted from an event: Process *nginx* write file *access.log*. Then we feed the sentence into the word2vec model and learn the embedding representation of each word in the sentence. Since each node in the provenance graph contains two attributes, type and name, the input feature h_u of node u can be computed as follows:

$$h_u = \text{Concat}(w^u_{type}, w^u_{name}) \tag{1}$$

where w^u_{type} is the vector representation of the type word of the node u, and w^u_{name} is the vector representation of the name word of the node u.

Learning Model Based on Graph Neural Network. Overly complex models bring certain difficulties to training. We tend to implement our models with simple network structures. We chose the graph attention network (GAT) [37] as the basic component of AttackMiner. The GAT is implemented by stacking simple graph attention layers, which introduce a self-attention mechanism in the propagation process, and the hidden state of each node is calculated by paying attention to its neighbor nodes. In our experiments, we use a multi-head attention mechanism with three heads. More details on GAT can be found in [37].

A Classifier Combining Multiple Receptive Fields. The general approach to graph classification is to add a multilayer perceptron (MLP) layer to the end of the graph network. However, we seek to strengthen the model's robustness while retaining the various traits of vectors. In contrast to the general approach, we try to perform 1D-convolution of node vectors with different receptive fields to obtain high-level features of embedding vectors. Then, we perform graph max-pooling and average-pooling operations on the convolution results, respectively, to obtain two graph embedding vectors. Finally, we concatenate the learned graph embedding vectors and feed them into MLP for multi-classification. The relevant expressions are as follows:

$$y_1 = \text{GraphPooling}(\text{ReLu}(\text{Conv}_1(H_G))) \tag{2}$$

$$y_2 = \text{GraphPooling}(\text{ReLu}(\text{Conv}_2(H_G))) \tag{3}$$

$$\tilde{y} = \text{softmax}(\sum \text{MLP}(\text{Concat}(y_1, y_2))) \tag{4}$$

where H_G denotes the node vectors of graph G following GNN embedding.

6 Evaluation

6.1 Implementation

AttackMiner is implemented in Python 3.8 with around 3100 lines of code (LoC). We use the Gensim [33] library to generate node embeddings for training graphs. The length of the initial node embedding vector is limited to 200 to reduce the size of the model. AttackMiner uses a two-layer GAT model to aggregate graph information. The input and output dimensions of node vectors in the model are 200 and 64, respectively. We optimize the model parameters through the Adam optimizer. We train the model for 20 epochs with the training batch size set to 64. The learning rate changes exponentially. The rate of change is 0.98 and the initial value is 0.01. We use Pytorch [31] as the backend and run our experiments on a server with two Intel Xeon E5-2630 v3 CPUs and 128 GB of memory. The server has two NVIDIA Tesla P4 GPUs and a 64-bit CentOS 7 operating system installed.

6.2 Dataset

We chose to evaluate our system using the DARPA TC [18] dataset, which was generated when the red team played against the blue team in April 2018. In total, we evaluated AttackMiner using four of these attack scenarios. We describe the scenarios and properties in Table 2.

For each attack scenario, we construct the corresponding attack subgraph. We leverage attack migration (detailed in Sect. 5.3) to generate datasets containing benign and malicious provenance graphs. The number of samples for each category of provenance graphs (including benign provenance graphs) is 6000, of which 80% are used as the training set. Before feeding the samples into the learned model, we use the graph optimization method described in Sect. 5.2 to optimize the structure of provenance graphs.

Table 2. Attack scenarios description

Scenario	Short description	$\|V(G)\|$	$\|E(G)\|$
ATK-1	Nginx vulnerability was exploited to attack CADETS FreeBSD. The attacker downloaded a file, elevated it to a new process running as root, and attempted to inject into the sshd process but fails	22	36
ATK-2	Attackers exploited Nginx with malformed HTTP requests to attack CADETS, and downloaded the libdrakon implant file *.so to inject into the sshd process, but it crashed CADETS	15	21
ATK-3	The shell connection connected to the operator console via HTTP. The attacker performed the micro apt implant without root privileges. The micro apt implant was connected to the C2's micro apt listener	36	65
ATK-4	The Nginx vulnerability was exploited by attackers to cause the drakon implant executable to run from disk, resulting in a new drakon process with root privileges and operation with a new connection to the administrator console	28	47

6.3 Effectiveness of Graph Optimization Algorithms

The effectiveness of AttackMiner lies in a set of optimization techniques integrated into its components. As described in Sect. 5.2, to reduce graph complexity, we construct provenance graphs using a set of custom graph optimization techniques. Here, we detail how these techniques contribute to its effectiveness. Figure 7 shows the number of entities and events before and after graph optimization when the log window length L is set to 60 min. Compared to the original graph size, AttackMiner reduces the number of entities in provenance graphs by an average of 80.20% and the number of events by an average of 83.39%. AttackMiner removes redundant or irrelevant events and entities from massive logs that do not provide any semantics to detect different attack patterns. Therefore, the further extracted provenance graph is more representative as the input for model training.

Table 3. Results before and after provenance graph optimization.

Category	Before optimization				After optimization			
	Recall %	Precision %	F1-score %	Accuracy %	Recall %	Precision %	F1-score %	Accuracy %
Benign	92.52	68.70	78.85	85.15	98.46	91.89	95.06	97.72
ATK-1	70.02	82.77	75.87		99.14	99.06	99.01	
ATK-2	92.79	100.00	96.26		98.11	100.00	99.05	
ATK-3	87.38	100.00	93.26		99.12	100.00	99.56	
ATK-4	81.49	82.61	82.05		93.63	98.31	95.91	

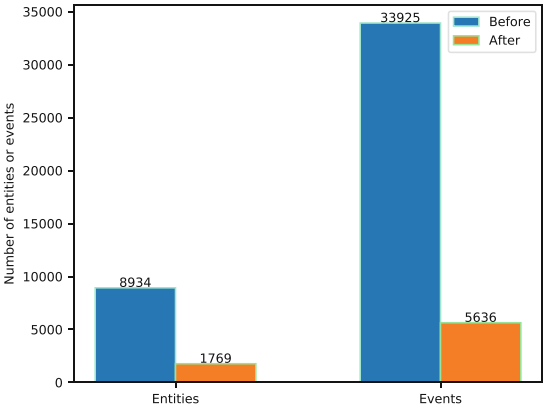


Fig. 7. Changes in average number of entities and average number of events before and after provenance graph optimization.

We present the detection results before and after graph optimization in Table 3 (log window length L is 60 min). As shown in Table 3, the provenance graph optimization improves the overall accuracy by 14.76%, and the F1 scores of the five categories are increased by 20.56%, 30.49%, 2.90%, 6.76%, and 16.89%, respectively. This is because graph optimization removes redundant events from massive logs, making attack patterns more apparent. Overall, the optimized provenance graph serves as a highly representative training set and is used for model training, which significantly improves the generalization ability of the model.

6.4 Comparison Analysis

We implemented a set of cutting-edge approaches that can be used to replace the AttackMiner component and compared their attack detection performance to that of AttackMiner. We chose the time span of the log window to be 60 min. Table 4 and Fig. 8 summarize our results.

Support vector machines (SVM) [36]. To evaluate the effectiveness of graph neural network aggregation, we compare it with SVM. We obtain the embedding vector of each node on the graph through the input embedding method introduced in 5.4, and then perform the graph pooling operation to obtain the embedded representation of the provenance graph. In order to improve the accuracy of the classifier, we implement a multi-classifier by combining multiple binary classifiers and repeatedly adjusting the parameters C, gamma, and the number of iterations of the SVM. In the evaluation experiments, we use a linear kernel function with C=1.0, gamma="auto" and an iteration number of 1000. Although the average accuracy of SVM can reach 86.55%, it cannot detect some attack behaviors well, and its recall on ATK-1 is only 66.90%. The main limitation of SVM is its inability to model the connection relationships between different entities in the provenance graph, which is one of the key features reflecting attack patterns.

Table 4. The results of the comparison analysis.

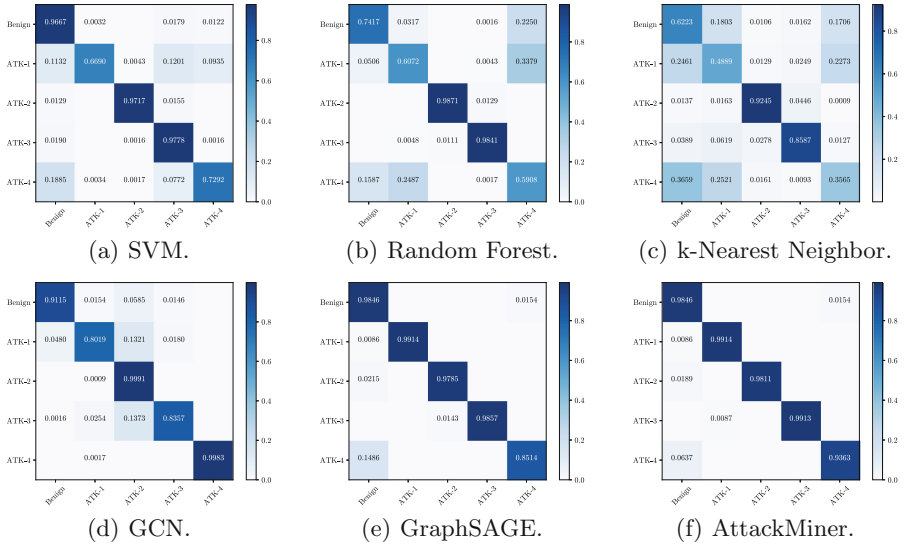
Category	SVM				Random forest			
	Recall %	Precision %	F1-score %	Accuracy %	Recall %	Precision %	F1-score %	Accuracy %
Benign	96.67	75.17	84.58	86.55	74.17	78.77	76.40	78.45
ATK-1	66.90	98.98	79.84		60.72	67.69	64.01	
ATK-2	97.17	99.21	98.18		98.71	98.80	98.75	
ATK-3	97.78	81.97	89.18		98.41	98.10	98.26	
ATK-4	72.92	87.21	79.43		59.08	50.91	54.70	
Category	k-nearest neighbor				GCN			
	Recall %	Precision %	F1-score %	Accuracy %	Recall %	Precision %	F1-score %	Accuracy %
Benign	62.23	49.45	55.11	65.25	91.15	95.08	93.07	90.83
ATK-1	48.89	48.06	48.47		80.19	94.54	86.77	
ATK-2	92.45	92.92	92.69		99.91	74.47	85.34	
ATK-3	85.87	90.62	88.18		83.57	96.43	89.54	
ATK-4	35.65	46.05	40.19		99.83	100.0	99.92	
Category	GraphSAGE				AttackMiner			
	Recall %	Precision %	F1-score %	Accuracy %	Recall %	Precision %	F1-score %	Accuracy %
Benign	98.46	85.23	91.37	95.88	98.46	91.89	95.06	97.72
ATK-1	99.14	100.0	99.57		99.14	99.06	99.01	
ATK-2	97.85	98.45	98.15		98.11	100.0	99.05	
ATK-3	98.57	100.0	99.28		99.12	100.0	99.56	
ATK-4	85.14	98.14	91.18		93.63	98.31	95.91	

Random Forest (RF) [22]. We chose the Gini coefficient as a function to measure segmentation quality. As can be seen from the Table 4, the accuracy of the RF classifier is much lower than that of the SVM. In addition, we also experimented with k-Nearest Neighbor (KNN) [8], which gives less classification performance than RF and SVM, and the classification accuracy is only 65.25%. Figure 8 visually shows the detection distribution of SVM, RF, and KNN. As shown in Fig. 8(a-c), the detection effects of SVM, RF, and KNN are inferior to those of GNN-based methods. This is mainly because GNNs can effectively learn by fusing the attribute information of graphs.

In addition, we compare AttackMiner with GCN [19] and GraphSAGE [11] to evaluate the effectiveness of AttackMiner in detecting migration attacks based on various graph neural networks. GCN is a convolution operation defined on the graph structure. By defining the convolution operation on the graph, the structural information and node information in the graph are captured. GraphSAGE aggregates information by sampling neighbor nodes. As can be seen from Table 4, the accuracies of GCN and GraphSAGE are 90.83% and 95.88%, respectively, which are only 7.05% and 1.88% lower than the AttackMiner prototype, respectively. In total, the attack behavior in the original audit log is typically made more obvious by graph structure optimization and attack migration operations.

6.5 Influence of Log Window Size on Detection Effect

In the process of building the provenance graph, we adopt the log window sliding mechanism. Since log windows of different sizes contain different numbers of



(a) SVM. (b) Random Forest. (c) k-Nearest Neighbor.

(d) GCN. (e) GraphSAGE. (f) AttackMiner.

Fig. 8. Confusion matrices result from comparison analysis.

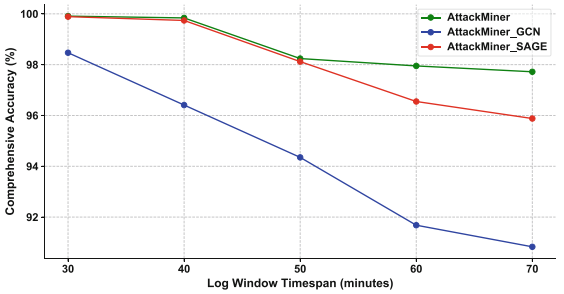


Fig. 9. Accuracy with different log window timespans.

Table 5. Results of different classifiers.

Category	AttackMiner_Tra				AttackMiner			
	Recall %	Precision %	F1-score %	Accuracy %	Recall %	Precision %	F1-score %	Accuracy %
Benign	92.77	93.61	93.19	92.97	98.46	91.89	95.06	97.72
ATK-1	80.36	94.65	86.92		99.14	99.06	99.01	
ATK-2	99.91	82.61	90.44		98.11	100.00	99.05	
ATK-3	93.57	96.24	94.89		99.12	100.00	99.56	
ATK-4	98.13	100.00	99.06		93.63	98.31	95.91	

entities and events, the size of the log window will affect the detection effect to a certain extent. We evaluate the impact on detection results when the log window size is varied from 30 min to 70 min, where the results are evaluated every 10 min. Figure 9 shows the change in detection accuracy for different log window sizes. In Fig. 9, as the log window becomes larger, the detection accuracy decreases slightly, but still maintains a high accuracy (when the log window size becomes 70 min, the AttackMiner detection accuracy is still above 96%). This is because while normal entities and events occupy a larger proportion of log windows with large time spans, AttackMiner learns the characteristics of attack patterns in different contexts through attack migration. At the same time, our graph structure optimization methods make the attack pattern more prominent in the provenance graph, which is also beneficial to the detector detection.

6.6 The Effect of Changes in the Classifier on the Experiment

Traditional classifiers directly input node features into MLP for aggregation classification. Instead of using the traditional method, we designed a new classifier. Our classifier is based on convolutional layers and combines multiple receptive fields. The new classifier can aggregate features from different aspects of the feature vector by combining multiple receptive fields simultaneously. To understand the contribution of our classifier module, we created a variant of AttackMiner called AttackMiner_Tra using the traditional classifier. Table 5 shows the performance metrics for the above settings. Overall, AttackMiner's average precision, recall, and F1-score are all over 4.80% higher than AttackMiner_Tra. We believe that richer attack features can be learned due to the addition of convolutional layers after the graph neural network. The results show that our classifier improves the performance of attack detection.

7 Conclusion

We present AttackMiner, a GNN-based attack detection framework for detecting attack activity from system audit logs. AttackMiner combines provenance graph analysis, natural language processing, and graph neural network techniques. Through optimization and attack migration based on provenance graph structure, AttackMiner can identify high-level patterns of different attacks. We extensively evaluate AttackMiner on real public datasets. AttackMiner successfully detects all the attacks with high accuracy and efficiency. Our research demonstrates the successful application of graph neural networks in attack detection.

Acknowledgment. This work is supported by the Strategic Priority Research Program of Chinese Academy of Sciences, Grant No. XDC02040200.

References

1. Adversarial tactics, techniques and common knowledge. https://attack.mitre.org/wiki/Main Page

2. Trace: Preventing advanced persistent threat cyberattacks(2018). https://archive.sri.com/work/projects/trace-preventing-advanced-persisten-threat-cyberattacks. (Accessed 1 April 2022)
3. Alsaheel, A., et al.: {ATLAS}: A sequence-based learning approach for attack investigation. In: 30th USENIX Security Symposium (USENIX Security 21), pp. 3005–3022 (2021)
4. Bates, A., Tian, D.J., Butler, K.R., Moyer, T.: Trustworthy whole-system provenance for the linux kernel. In: 24th USENIX Security Symposium (USENIX Security 15), pp. 319–334 (2015)
5. Bilge, L., Balzarotti, D., Robertson, W., Kirda, E., Kruegel, C.: Disclosure: detecting botnet command and control servers through large-scale netflow analysis. In: Proceedings of the 28th Annual Computer Security Applications Conference, pp. 129–138 (2012)
6. Debnath, B., et al.: Loglens: A real-time log analysis system. In: 2018 IEEE 38th International Conference On Distributed Computing Systems (ICDCS), pp. 1052–1062. IEEE (2018)
7. Du, M., Li, F., Zheng, G., Srikumar, V.: Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In: Proceedings of the 2017 ACM SIGSAC Conference On Computer And Communications Security, pp. 1285–1298 (2017)
8. Fix, E., Hodges, J.L.: Discriminatory analysis. nonparametric discrimination: Consistency properties. International Statistical Review/Revue Internationale de Statistique **57**(3), 238–247 (1989)
9. Goel, A., Feng, W.C., Maier, D., Walpole, J.: Forensix: A robust, high-performance reconstruction system. In: 25th IEEE International Conference on Distributed Computing Systems Workshops, pp. 155–162. IEEE (2005)
10. Goel, A., Po, K., Farhadi, K., Li, Z., De Lara, E.: The taser intrusion recovery system. In: Proceedings of the Twentieth ACM Symposium On Operating Systems Principles, pp. 163–176 (2005)
11. Hamilton, W., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. In: Advances in Neural Information Processing Systems 30 (2017)
12. Han, X., Pasquier, T., Bates, A., Mickens, J., Seltzer, M.: Unicorn: Runtime provenance-based detector for advanced persistent threats. arXiv preprint arXiv:2001.01525 (2020)
13. Hassan, W.U., Bates, A., Marino, D.: Tactical provenance analysis for endpoint detection and response systems. In: 2020 IEEE Symposium on Security and Privacy (SP), pp. 1172–1189. IEEE (2020)
14. Hassan, W.U., et al.: Nodoze: Combatting threat alert fatigue with automated provenance triage. In: Network and Distributed Systems Security Symposium (2019)
15. Hassan, W.U., Noureddine, M.A., Datta, P., Bates, A.: Omegalog: High-fidelity attack investigation via transparent multi-layer log analysis. In: Network and Distributed System Security Symposium (2020)
16. Homayoun, S., Dehghantanha, A., Ahmadzadeh, M., Hashemi, S., Khayami, R.: Know abnormal, find evil: frequent pattern mining for ransomware threat hunting and intelligence. IEEE Trans. Emerg. Top. Comput. **8**(2), 341–351 (2017)
17. Hutchins, E.M., Cloppert, M.J., Amin, R.M., et al.: Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. Lead. Issues Inf. Warfare Sec. Res. **1**(1), 80 (2011)
18. Keromytis, A.D.: Transparent computing engagement 3 data release (2018). https://github.com/darpa-i2o/Transparent-Computing

19. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907 (2016)
20. Le, Q., Mikolov, T.: Distributed representations of sentences and documents. In: International Conference On Machine Learning, pp. 1188–1196 (2014)
21. Li, Y., Gu, C., Dullien, T., Vinyals, O., Kohli, P.: Graph matching networks for learning the similarity of graph structured objects. In: International Conference on Machine Learning, pp. 3835–3845. PMLR (2019)
22. Liaw, A., Wiener, M., et al.: Classification and regression by randomforest. R news **2**(3), 18–22 (2002)
23. Liu, F., Wen, Y., Zhang, D., Jiang, X., Xing, X., Meng, D.: 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, pp. 1777–1794 (2019)
24. Liu, Y., et al.: Towards a timely causality analysis for enterprise security. In: NDSS (2018)
25. Manzoor, E., Milajerdi, S.M., Akoglu, L.: Fast memory-efficient anomaly detection in streaming heterogeneous graphs. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1035–1044 (2016)
26. Milajerdi, S.M., Eshete, B., Gjomemo, R., Venkatakrishnan, V.: 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, pp. 1795–1812 (2019)
27. Milajerdi, S.M., Gjomemo, R., Eshete, B., Sekar, R., Venkatakrishnan, V.: Holmes: real-time apt detection through correlation of suspicious information flows. In: 2019 IEEE Symposium on Security and Privacy (SP), pp. 1137–1152. IEEE (2019)
28. Oprea, A., Li, Z., Yen, T.F., Chin, S.H., Alrwais, S.: Detection of early-stage enterprise infection by mining large-scale log data. In: 2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, pp. 45–56. IEEE (2015)
29. Parveen, P., McDaniel, N., Hariharan, V.S., Thuraisingham, B., Khan, L.: Unsupervised ensemble based learning for insider threat detection. In: 2012 International Conference on Privacy, Security, Risk and Trust and 2012 International Confernece on Social Computing, pp. 718–727. IEEE (2012)
30. Pasquier, T., et al.: Practical whole-system provenance capture. In: Proceedings of the 2017 Symposium on Cloud Computing, pp. 405–418 (2017)
31. Paszke, A., et al.: Pytorch: An imperative style, high-performance deep learning library. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems, vol. 32, pp. 8024–8035. Curran Associates, Inc. (2019). http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf
32. Pohly, D.J., McLaughlin, S., McDaniel, P., Butler, K.: Hi-fi: collecting high-fidelity whole-system provenance. In: Proceedings of the 28th Annual Computer Security Applications Conference, pp. 259–268 (2012)
33. Rehurek, R., Sojka, P.: Software framework for topic modelling with large corpora. In: Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks. Citeseer (2010)
34. Shen, Y., Mariconti, E., Vervier, P.A., Stringhini, G.: Tiresias: Predicting security events through deep learning. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pp. 592–605 (2018)

35. Song, W., Yin, H., Liu, C., Song, D.: Deepmem: Learning graph neural network models for fast and robust memory forensic analysis. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pp. 606–618 (2018)
36. Suykens, J.A., Vandewalle, J.: Least squares support vector machine classifiers. Neural Process. Lett. **9**(3), 293–300 (1999)
37. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., Bengio, Y.: Graph attention networks. arXiv preprint arXiv:1710.10903 (2017)
38. Wang, Q., et al.: You are what you do: Hunting stealthy malware via data provenance analysis. In: NDSS (2020)
39. Wang, S., et al.: Heterogeneous graph matching networks for unknown malware detection. In: Proceedings of the 28th International Joint Conference on Artificial Intelligence, pp. 3762–3770. AAAI Press (2019)
40. Xu, X., Liu, C., Feng, Q., Yin, H., Song, L., Song, D.: Neural network-based graph embedding for cross-platform binary code similarity detection. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pp. 363–376 (2017)
41. Zhu, T., et al.: General, efficient, and real-time data compaction strategy for apt forensic analysis. IEEE Trans. Inf. Forensics Secur. **16**, 3312–3325 (2021)