An Interpretation Method for Provenance-based Intrusion Detection Based on GNNExplainer

Ziyang Yu[1,2], Wentao Li[1,2], Xiu Ma[1,2] Xinbo Han[1,2], Ning Li[1], Qiujian Lv[1], and Weiqing Huang[1]

[1] Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
yuziyang@iie.ac.cn, lining01@iie.ac.cn

Abstract. With the growing menace of Advanced Persistent Threats (APTs) in recent years, provenance graphs have become the focus of studies exploring various techniques to analyze APTs using Graph Neural Networks (GNNs) for automated host intrusion detection. However, a significant barrier to the practical adoption of GNN-based host intrusion detection systems (IDS) is their lack of interpretability. To address this issue, security operators need an approach that enhances their understanding of how IDS make decisions.

To overcome these limitations, we introduce an interpreter by adapting and extending GNNExplainer, a model and task-agnostic tool, to work with large-scale heterogeneous graphs. This adaptation enables precise identification of graph components—nodes, edges, and features, that are critical to the detection outcomes of PIDS. We demonstrate the utility of our approach through a detailed analysis of two primary PIDS models: threaTrace and Kairos, focusing on node classification and link prediction tasks, respectively. Our contributions include successfully expanding GNNExplainer to handle the complexity and scale of provenance graphs, identifying key features influencing PIDS model predictions, and comprehensively evaluating the interpreter's performance using fidelity and stability metrics. The results affirm our method's superiority over conventional interpretability approaches like GraphLime and LIME, significantly advancing the interpretability of GNN-based PIDS and thereby facilitating a more effective response to security threats.

Keywords: Security Applications · GNN Interpretability · Provenance Graphs · Host Intrusion Detection

1 Introduction

In recent years, the rise of Advanced Persistent Threats (APTs) has significantly challenged cybersecurity efforts[1]. The need for effective and prompt detection methods to counteract APTs is crucial. Intrusion detection plays a crucial role in identifying deviations that may signal threats[2]. Utilizing provenance graphs has emerged as an innovative approach in enhancing intrusion detection capabilities.

Provenance graphs, encapsulating a system's operational history through structured audit logs, are critical in intrusion detection[3–8]. Originating from system-level logs, they detail interactions among kernel objects, offering a comprehensive system narrative. Despite their detail, they present challenges in modeling complex relationships and features[1, 9, 10]. To overcome these challenges, Graph Neural Networks (GNNs) are employed, leveraging their capacity to integrate neighboring information and effectively capture both structural and feature data.[5–7, 11–13]. Research splits into two approaches: one focuses on identifying anomalous system objects via node classification[6, 5], and the other on detecting abnormal interactions through link prediction[7, 11].

Despite their success in lab settings, GNN models' lack of transparency and interpretability hinders their real-world application in security[14, 15]. The "black-box" nature of these models challenges the adoption of GNN-based Provenance Intrusion Detection Systems (PIDSes) due to difficulty in trusting system decisions without clear justifications. These systems often yield binary outcomes (abnormal or normal) without explaining the influence of specific edges or nodes, increasing the alarm analysis workload. Thus, there's a critical need for interpreters that can clarify the impact of particular nodes or edges, significantly easing the experts' analysis efforts.

Existing methods for interpreting GNN models help clarify the importance of nodes, edges, and features in making predictions. Techniques like SA[16] and CAM[17] use gradients or feature activations to assess input significance. GraphLime[19] focuses on the importance of node features for classification, while LRP[21] links model parameters with input-output features. However, works above are limited by GNNs' structure, tasks or the type and scale of the input graphs, which makes the cost of switching and deploying new models high. The complexity and scale of provenance graphs add to the challenge, making it harder for security operators to verify the accuracy of anomalies detected by PIDS. There's a need for a general interpretability method capable of handling large, heterogeneous graphs like those in PIDS, and that's independent of model or task. Such a method would significantly aid security operators in understanding and reacting to PIDS alerts.

In reaction to these challenges, this paper presents an interpreter aimed at making the decision-making of GNN-based PIDSes transparent, utilizing GNNExplainer[23], a versatile GNN interpretation tool that's model and task-agnostic. Given the primary tasks of PIDSes are node classification and link prediction, we focus on threaTrace and Kairos due to their significant contributions to these areas[6, 7]. This study's main contributions are threefold:

1. We expanded GNNExplainer, originally focused on small-scale homogeneous graphs, to work with large-scale heterogeneous graphs, ensuring stable interpretative performance.

2. We used the interpreter to identify the features that most significantly affect the predictions of PIDS models.

3. We evaluated the interpreter's effectiveness using fidelity and stability metrics, showing improved interpretability over baseline methods like GraphLime and LIME.

2 Related Work

2.1 Provenance-based Intrusion Detection

Methods that identify anomalies in host behaviors through provenance data typically analyze kernel audit logs to establish normal patterns, treating deviations as potential threats[8, 3]. Within this framework, GNNs are utilized for two main tasks: node classification and link prediction[24].

StreamSpot employs hash computations on subgraphs to generate a fixed-size feature vector[32]. SIGL employs a graph autoencoder with GNNs to compress provenance graph features into a vector, analyzing software installations[8]. Unicorn divides the provenance graph into temporal snapshots, using graph kernel methods for feature extraction[3].

Significant advancements include threaTrace and Kairos[6, 7]. threaTrace uses GNNs for node classification within provenance graphs to detect anomalies[6]. Kairos focuses on link prediction, using GNNs to predict connections and identify security risks[7]. These efforts demonstrate the versatility of GNNs in improving detection in provenance-based systems.

2.2 Explanations for Graph Neural Networks

Interpretability methods in graph models provide insights into the importance of various graph components, crucial for improving predictive accuracy. These methods aim to explain graph input features and are categorized into four main types based on their approach:

Gradients/Features-Based Methods: These methods use gradients or hidden feature activations to estimate the importance of inputs. For example, SA uses squared gradient values as importance metrics[16], while CAM identifies key nodes by mapping features from the last layer back to the input space, requiring specific GNN layers[17]. However, their applicability is limited by GNN structure and task-specific constraints.

Surrogate Methods: This approach uses a simpler model to approximate the complex model's predictions, helping explain the significance of node features in tasks like classification. GraphLime extends the LIME algorithm to graphs[19], and PGM-Explainer creates a probabilistic model for GNN explanations[20].

Decomposition Methods: These methods break down predictions to reflect the importance of input features, often delving into model parameters. LRP provides interpretations based on model parameters but focuses only on node importance[21], not addressing broader graph structures.

Perturbation-based Methods: By generating masks for input graphs, these methods highlight critical features across different tasks (e.g., nodes, edges).

PGExplainer learns edge masks for explanations[22], while GNNExplainer optimizes masks for edges and node features, aiding in interpreting GNN predictions in feature-rich graphs[23].

Most of these interpretability approaches, while offering valuable insights, have limitations, such as being model or task-specific, not fully applicable to link prediction, or only addressing node importance. Adapting and deploying new models remains a challenge due to these constraints.

3 Method

3.1 The Overall Process of of Interpretation

In this subsection, we outline the comprehensive process for interpreting GNN-based PIDS. As illustrated in Fig. 1, the interpretation process encompasses several sequential steps:

Train a GNN-based PIDS model: As a crucial input for the interpreter, we train the selected Provenance Intrusion Detection System (PIDS) model using the relevant dataset to obtain the desired model for interpretation. This trained model serves as the foundation upon which the interpreter will analyze and elucidate the decision-making process, highlighting the significance of specific nodes or edges in detecting anomalies within the system.

Anomaly detection with the trained PIDS: Considering the vast scale of provenance graphs and the fact that security operators typically focus on the positive (anomalous) results generated by PIDS, it's essential to use these positive outcomes as another input for the interpreter. This approach allows the interpreter to concentrate on analyzing and providing explanations for instances where the PIDS has identified potential security threats, thereby enabling a more focused and efficient analysis conducive to the practical needs of security operations. Based on the positive results (anomalous nodes or edges) from the PIDS, we can extract the IDs of these positive nodes and edges and input them into the interpreter for explanation.

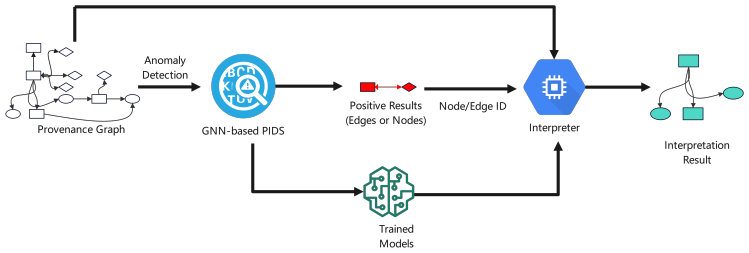

Fig. 1. The pipeline of interpretation.

Parameter Configuration: To activate and operationalize the interpreter effectively, it's crucial to configure its initial parameters meticulously. These parameters encompass the depth of neighborhood exploration to understand the extent of the graph's traversal, the selection of the top-K features to focus on the

most influential attributes, the model mode to tailor the interpreter's operation according to specific requirements, the task level to define the granularity of the analysis, and the return type to determine the format of the interpretation results. Setting these parameters accurately is essential for tuning the interpreter's performance to provide meaningful and actionable insights.

Interpret the positive result: Once the model and positive results are established and the interpreter is properly configured, use the interpreter to elucidate the positive outcomes. This process will identify the features, edges, and nodes that have the most significant impact on producing these positive results.

Using expert knowledge to analyze the results: Based on the interpreter's final output, security operators can focus on analyzing the reasons behind anomalies by concentrating on the few features that have the most significant impact on the results. This targeted approach enables them to efficiently determine whether an alert is a false positive or to expedite the response to genuine alerts by leveraging expert knowledge. This process not only streamlines the analysis and decision-making processes but also enhances the overall effectiveness and accuracy of the security operations. By doing so, the interpreter not only sheds light on the critical components within the provenance graph that contribute to anomalies but also enhances the understanding of the underlying mechanisms driving the PIDS's detections, thereby aiding in the refinement of security measures and the development of more robust intrusion detection strategies.

3.2 Background On GNN-baesd Provenance Intrusion Detection Systems

Figure. 2 illustrates the typical workflow of a GNN-based Provenance Intrusion Detection System (PIDS). The process begins with generating a provenance graph from audit logs, which is then transformed into a feature vector using various graph representation techniques. Firstly, we train GNN using provenance graph vectors with no abnormal behavior to obtain a well-trained model. Then the graph vectors feed into the trained GNN model, undergoing encoding and decoding processes to produce predictions. The system calculates the reconstruction error Δ based on neighborhood structures to aid in anomaly detection. The outcome is the determination of detected behavior as either anomalous or normal. GNNs primarily address two tasks within this context: node classification and link prediction[24]. Node classification assigns labels to graph nodes, helping identify anomalies or malicious actions. Link prediction, conversely, aims to reveal potential relationships or suspicious connections, which is crucial for spotting possible attack vectors. Leveraging GNNs for these tasks has significantly advanced the capabilities of security monitoring and threat identification in provenance-based intrusion detection systems.

After thorough training and anomaly detection in provenance graphs, we obtain crucial inputs the interpreter, which are shown in Fig. 1: the trained model and the positive nodes or edges that need to be interpreted.

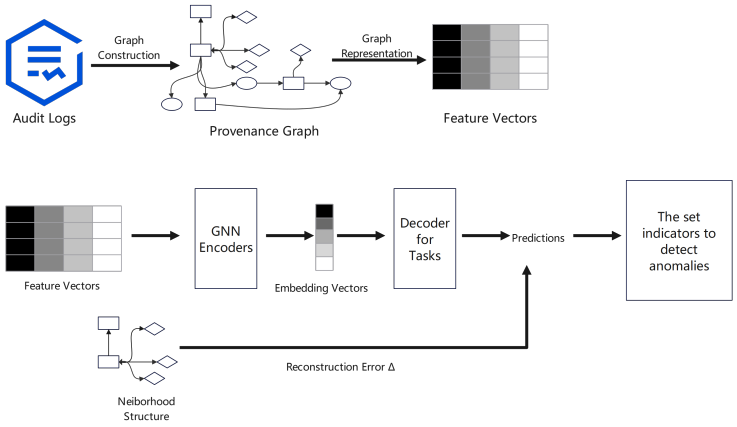

Fig. 2. A typical framework for a GNN-based PIDS.

3.3 The Details of the Interpreter

To explain the PIDS, we selected GNNExplainer as the interpreter because it presents a novel method for explaining the predictions of any GNN across all graph-based machine learning tasks without necessitating modifications to the underlying GNN architecture or re-training[23].

GNNExplainer learns soft masks for both edges and features to explain predictions. Soft masks are initialized randomly and treated as trainable variables. GNNExplainer then combines the original graph with the masks through element-wise multiplications. It optimizes the masks by maximizing the mutual information between the predictions of the original graph and the predictions obtained from the new graph.

The Architecture of GNNExplainer Firstly, given the input graph, masks are obtained to indicate important input features. Different types of masks are generated based on different interpretation tasks, such as node masks, edge masks, and node feature masks. Then, the generated masks are combined with the input graph to obtain a new graph containing essential input information. Finally, the new graph is input into the pre-trained GNN model to obtain new predictions, which are compared with the original predictions to evaluate the masks and update the mask generation algorithm. Intuitively, the important input features captured by the masks should convey critical semantic meanings, resulting in new predictions that are essentially similar to the original predictions.

There are three types of masks: soft masks, discrete masks, and approximated discrete masks. In Fig. 3, node feature masks are soft masks, edge masks are discrete masks, and node masks are approximated discrete masks. Soft masks contain continuous values in the range $[0, 1]$, and the mask generation algorithm can directly backpropagate updates.

The Formulation of GNNExplainer This subsubsection introduces the formulation of the "Mask Generation" part in Fig. 3. For a node v detected as positive by PIDS, our goal is to find the important subgraphs $G_S \subseteq G_P$, where

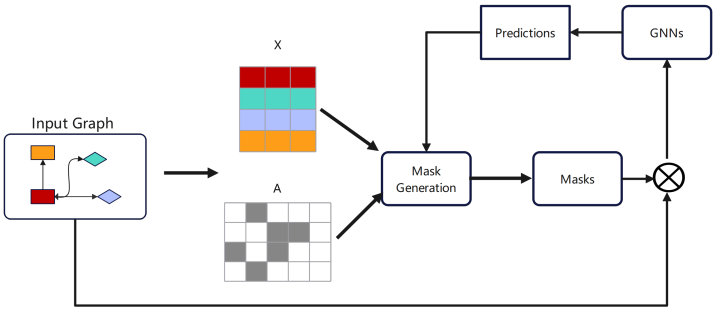

Fig. 3. The architecture of the interpreter. The interpreter employs different mask generation algorithms to obtain different types of masks. Note that the mask can correspond to nodes, edges, or node features. In this example, we show a soft mask for node features, a discrete mask for edges, and an approximated discrete mask for nodes. Then the mask is combined with the input graph to capture important input information. Finally, the trained GNNs evaluate whether the new prediction is similar to the original prediction and can provide guidance for improving the mask generation algorithms.

G_P represents a set of nodes of the input provenance graph P and the important feature subset $X_s = x_j \mid v_j \in G_S$ that are crucial for the positive result $\hat{y}$. By utilizing mutual information (MI), GNNExplainer can be represented as below.

$$\max_{G_s} MI(Y, (G_S, X_S)) = H(Y) - H(Y|G = G_S, X = X_S) \tag{1}$$

For instance, when $v_j \in G_P(v_i), v_j \neq v_i$, if removing node v from $G_P(v_i)$ significantly decreases the abnormal score for anomaly detectiont, then we can consider v_j as an important node for v_i. Similarly, if $(v_j, v_k) \in G_C(v_i), v_j, v_k \neq v_i$, and removing the relationship (v_j, v_k) from $G_P(v_i)$ significantly reduces the abnormal score, then we consider this relationship important for v_i.

Returning to the previous equation 1, once a GNN-based PIDS model is trained, the entropy $H(Y)$ becomes deterministic. Thus, the equation above becomes minimizing the conditional entropy $H(Y|G = G_S, X = X_S)$ as below.

$$H(Y|G = G_S, X = X_S) = -\mathbb{E}Y|G_S, X_S[logP\Phi(Y|G = G_S, X = X_S)] \tag{2}$$

Then, we can optimize equation 2 as follows:

$$\min_{\mathcal{G}} \mathbb{E}G_S \sim GH(Y|G = G_S, X = X_S) \tag{3}$$

Then, Jensen's inequality allows us to derive equation 4 from equation 3.

$$\min \mathcal{G}H(Y|G = \mathbb{E}_{\mathcal{G}}[G_S], X = X_S) \tag{4}$$

Clearly, GNN-based PIDS models do not satisfy the assumption of convex functions. However, it has been found in the literature that by combining the objective function described above with regularization terms, the learned local optimal solutions already exhibit good explanatory power[23].

For the expectation $\mathbb{E}_{\mathcal{G}}[GS]$, it is implemented through a mask, denoted as $A_c \odot \sigma(M), where M \in \mathbb{R}^{n \times n}$, meaning that what the actual interpreter needs to learn is the mask M.

Furthermore, in general, we are interested in understanding "why a particular sample is predicted as a certain class" rather than understanding the global model. Therefore, we further modify the objective function to:

$$\min_{M} -\sum_{p=1}^{P} 1[y = p] \log P_{\Phi}(Y = y | G = A_p \odot \sigma(M), X = X_p) \tag{5}$$

Lastly, we compute the element-wise multiplication of $\sigma(M)$ and A_p and remove low values in M through thresholding to arrive at the explanation G_S for the GNN-based PIDS model's prediction $\hat{y}$ at positive node v.

4 Evaluations

4.1 Environment Setup and Dataset

Here, we briefly introduce the implementation and experimental setup. All experiments are performed on a server running Ubuntu 20.04 with 3.20GHz 32-core Intel Xeon Silver 4215R CPU, 128 GB of memory and Nvidia GeForce RTX 3090.

PIDSes & Datasets. We use two GNN-based provenance intrusion detector systems. ThreaTrace pinpoints only anomalous nodes that might be involved in the attack[6]. Kairos[7], on the other hand, focuses on the anomaly edges in the attack chains. These two works respectively employ node classification and link prediction as tasks for the decoder.

Table 1. Summary of the experimental datasets.

Dataset	# of Nodes	# of Edges(in millions)	# of Attack Edges	% of Attack Edges
DARPA-E3-CADETS	178,965	10.1	1248	0.012%
DARPA-E3-THEIA	690,105	32.4	3119	0.010%

We obtain our experimental datasets from DARPA[25]. They are the few open-source datasets widely used in evaluating provenance-based systems[3, 4, 6, 7]. Table 1 summarizes the statistics of the graphs in those datasets.

Implementation of Our Interpreter and Baselines We implemented our interpreter in Python. Firstly, We use scikit-learn[26] and PyG[27], a PyTorch-based graph neural networks library, to reproduce the anomaly detection work of ThreatTrace and Kairos. After we gained the trained model and data, we use PyG to adapt the GNNExplainer and the baseline interprters. For the interpreters, we fix the interpretation results into K-dimension and set neighborhood scale $\sigma_n = 3$ by default and evaluate its effect below. For baselines, we choose LIME[31] , a classic linear explaination method in deep learning, and GraphLime[19], a gradients-based method on graph neural networks which is derived from LIME.

4.2 Fidelity-Conciseness Evaluation

Assessing the quality of interpretation results is essential, relying on two methods: comparison with Ground Truth (GT) and fidelity evaluation. GT comparison measures the match between interpretation results and a known correct interpretation, while fidelity evaluates the consistency of the model's predictions between the interpreted results and the original graph. Fidelity focuses on whether essential predictive elements are maintained in the subgraph. Due to the lack of interpretation GTs for Darpa Datasets, fidelity is used to evaluate the quality of interpretation results.

To assess the fidelity of our interpretations, we adopt an indicator inspired by [28], termed the Label Flipping Rate (LFR). LFR measures the proportion of data labeled as 'abnormal' that is reclassified as 'normal' following the application of interpretation results. This metric assumes that interpreters of higher fidelity will exhibit higher LFR values, as they more accurately identify the crucial features responsible for the data's abnormal classification. It is important to note, however, that an increase in LFR, driven by including more features in the interpretation, may inadvertently compromise the conciseness of the interpretation.

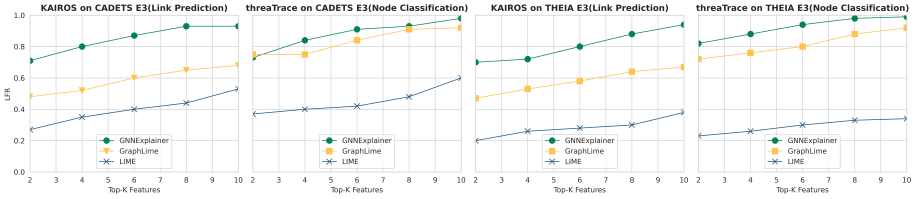

Fig. 4. Fidelity evaluation of interpreters(*higher is better*).

Our evaluation, depicted in Figure 4, explores the fidelity-conciseness trade-off. The x-axis ranks the top-K important features, illustrating that, with constant interpreter, interpreted model, and dataset, the Label Flipping Rate (LFR) escalates in tandem with K — the number of top-k significant features determined by the interpreter. Despite this trend, GNNExplainer consistently outperforms both GraphLime and LIME in fidelity assessments for any given interpreted model and dataset configuration. This superiority underscores GNNExplainer's efficacy in not only identifying pivotal features with greater precision but also in maintaining a balance between fidelity and conciseness of interpretations.

4.3 Stability Evaluation

Stability is essential for interpreters, ensuring consistent explanations for similar inputs. It verifies that minor changes to the input don't significantly alter the interpretation, crucial for the reliability of the insights provided. Thus, stability is a key metric for the effectiveness and trustworthiness of interpretation methods.

To evaluate the stability of our interpretation methods, we focus on the consistency of interpretations for identical samples across different executions. Drawing inspiration from [29, 30], we concentrate on the indexes of important feature dimensions, disregarding their specific values. For any two sets of interpretation vectors, v_1 and v_2, representing important dimension indexes obtained from consecutive runs under identical settings, we employ the *Jaccard Similarity* (**JS**) to quantify the similarity between these sets. JS is calculated as the size of the intersection divided by the size of the union of the two sets: $|set(v_1) \cap set(v_2)|/|set(v_1) \cup set(v_2)|$. By measuring and averaging the similarity of interpretation results for each anomaly across multiple runs, we gain insights into the interpreters' stability.

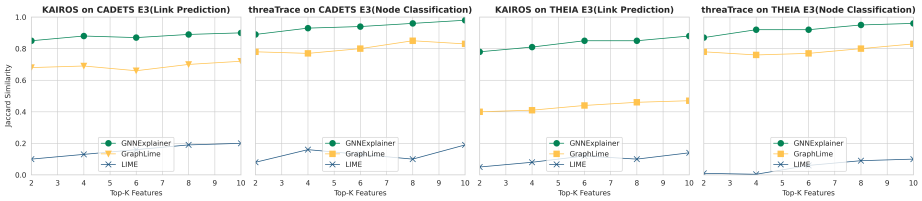

Fig. 5. Stability evaluation of interpreters(*higher is better*).

The findings, illustrated in Figure 5, reveal that GNNExplainer consistently exhibits superior stability across different models and their respective datasets when compared to GraphLime, and markedly surpasses LIME. LIME's lower performance in this context underscores its limitations for graph data, highlighting the importance of employing interpretation methods like GNNExplainer that are specifically designed for such data structures.

5 Conclusions

This paper introduces a novel interpretation methodology for Provenance Intrusion Detection Systems (PIDS) leveraging the GNNExplainer framework. We innovatively adapt and extend GNNExplainer to accommodate the unique complexities of ultra-large-scale heterogeneous provenance graphs. This adaptation enables the precise identification of the features most significantly influencing the model's decisions. Our evaluation focuses on the interpretation of two seminal PIDS models within the DARPA dataset, employing both fidelity and stability as metrics for assessment. Through comparative analysis with GraphLime and LIME, our findings affirm the effectiveness and superiority of our method.

Our method, while effective, has limitations and areas for improvement. Stability and fidelity in link prediction remain lower compared to node classification, indicating potential for future enhancements. Additionally, the impact of increasing neighborhood search depth on interpretability has not been explored. From a security perspective, relying solely on interpreter outputs is insufficient; manual analysis of these results still imposes significant workload on security operators. Future work should investigate integrating expert knowledge to automate fur-

ther analysis of interpretability outcomes. We aim to address these shortcomings in our subsequent research.

References

1. Wang, R., Nie, K., Wang, T., Yang, Y., & Long, B. (2020, January). Deep learning for anomaly detection. In Proceedings of the 13th international conference on web search and data mining (pp. 894-896).
2. Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. ACM computing surveys (CSUR), 41(3), 1-58.
3. Han, X., Pasquier, T., Bates, A., Mickens, J., Seltzer, M. (2020, February). UNICORN: Runtime Provenance-Based Detector for Advanced Persistent Threats. In Network and Distributed Systems Security (NDSS) Symposium 2020 (pp. 1-18). Internet Society.
4. Milajerdi, S. M., Gjomemo, R., Eshete, B., Sekar, R., Venkatakrishnan, V. N. (2019, May). Holmes: real-time apt detection through correlation of suspicious information flows. In 2019 IEEE Symposium on Security and Privacy (SP) (pp. 1137-1152). IEEE.
5. Zengy, J., Wang, X., Liu, J., Chen, Y., Liang, Z., Chua, T. S., Chua, Z. L. (2022, May). Shadewatcher: Recommendation-guided cyber threat analysis using system audit records. In 2022 IEEE Symposium on Security and Privacy (SP) (pp. 489-506). IEEE.
6. Wang, S., Wang, Z., Zhou, T., Sun, H., Yin, X., Han, D., ... Yang, J. (2022). Threatrace: Detecting and tracing host-based threats in node level through provenance graph learning. IEEE Transactions on Information Forensics and Security, 17, 3972-3987.
7. Cheng, Z., Lv, Q., Liang, J., Wang, Y., Sun, D., Pasquier, T., Han, X. (2023, October). KAIROS: Practical Intrusion Detection and Investigation using Whole-system Provenance. In 2024 IEEE Symposium on Security and Privacy (SP) (pp. 5-5). IEEE Computer Society.
8. Han, X., Yu, X., Pasquier, T., Li, D., Rhee, J., Mickens, J., ... & Chen, H. (2021). SIGL: Securing Software Installations Through Deep Graph Learning. In 30th USENIX Security Symposium (USENIX Security 21) (pp. 2345-2362).
9. Zhang, Z., Cui, P., Zhu, W. (2020). Deep learning on graphs: A survey. IEEE Transactions on Knowledge and Data Engineering, 34(1), 249-270.
10. Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., ... Sun, M. (2020). Graph neural networks: A review of methods and applications. AI open, 1, 57-81.
11. Zhang, M., Chen, Y. (2018). Link prediction based on graph neural networks. Advances in neural information processing systems, 31.
12. Ying, Z., You, J., Morris, C., Ren, X., Hamilton, W., Leskovec, J. (2018). Hierarchical graph representation learning with differentiable pooling. Advances in neural information processing systems, 31.
13. Hamilton, W., Ying, Z.,& Leskovec, J. (2017). Inductive representation learning on large graphs. Advances in neural information processing systems, 30.
14. Adadi, A., & Berrada, M. (2018). Peeking inside the black-box: a survey on explainable artificial intelligence (XAI). IEEE access, 6, 52138-52160.
15. Guidotti, R., Monreale, A., Ruggieri, S., Turini, F., Giannotti, F., & Pedreschi, D. (2018). A survey of methods for explaining black box models. ACM computing surveys (CSUR), 51(5), 1-42.

16. Baldassarre, F., & Azizpour, H. (2019). Explainability Techniques for Graph Convolutional Networks. In International Conference on Machine Learning (ICML) Workshops, 2019 Workshop on Learning and Reasoning with Graph-Structured Representations.
17. Pope, P. E., Kolouri, S., Rostami, M., Martin, C. E., & Hoffmann, H. (2019). Explainability methods for graph convolutional neural networks. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 10772-10781).
18. Yuan, H., Yu, H., Gui, S., & Ji, S. (2022). Explainability in graph neural networks: A taxonomic survey. IEEE transactions on pattern analysis and machine intelligence, 45(5), 5782-5799.
19. Huang, Q., Yamada, M., Tian, Y., Singh, D., & Chang, Y. (2022). Graphlime: Local interpretable model explanations for graph neural networks. IEEE Transactions on Knowledge and Data Engineering.
20. Vu, M., & Thai, M. T. (2020). Pgm-explainer: Probabilistic graphical model explanations for graph neural networks. Advances in neural information processing systems, 33, 12225-12235.
21. Baldassarre, F., & Azizpour, H. (2019). Explainability techniques for graph convolutional networks. arXiv preprint arXiv:1905.13686.
22. Luo, D., Cheng, W., Xu, D., Yu, W., Zong, B., Chen, H., & Zhang, X. (2020). Parameterized explainer for graph neural network. Advances in neural information processing systems, 33, 19620-19631.
23. Ying, Z., Bourgeois, D., You, J., Zitnik, M., & Leskovec, J. (2019). Gnnexplainer: Generating explanations for graph neural networks. Advances in neural information processing systems, 32.
24. Zipperle, M., Gottwalt, F., Chang, E., & Dillon, T. (2022). Provenance-based intrusion detection systems: A survey. ACM Computing Surveys, 55(7), 1-36.
25. A. D. Keromytis, "Transparent Computing Engagement 3 Data Release", https://github:com/darpa-i2o/Transparent-Computing/blob/master/README-E3:md, last accessed 2018.
26. "scikit-learn: machine learning in Python," https: //scikit-learn:org/, 2021.
27. Fey, M., & Lenssen, J. E., "Fast graph representation learning with PyTorch Geometric," in ICLR Workshop on Representation Learning on Graphs and Manifolds, 2019.
28. Guo, W., Mu, D., Xu, J., Su, P., Wang, G., & Xing, X. (2018, October). Lemna: Explaining deep learning based security applications. In proceedings of the 2018 ACM SIGSAC conference on computer and communications security (pp. 364-379).
29. Fan, M., Wei, W., Xie, X., Liu, Y., Guan, X., & Liu, T. (2020). Can we trust your explanations? Sanity checks for interpreters in Android malware analysis. IEEE Transactions on Information Forensics and Security, 16, 838-853.
30. Warnecke, A., Arp, D., Wressnegger, C., & Rieck, K. (2020, September). Evaluating explanation methods for deep learning in security. In 2020 IEEE european symposium on security and privacy (EuroS&P) (pp. 158-174). IEEE.
31. Ribeiro, M. T., Singh, S., & Guestrin, C. (2016, August). " Why should i trust you?" Explaining the predictions of any classifier. In Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining (pp. 1135-1144).
32. Manzoor E, Momeni S, Venkatakrishnan V, et al. Fast memory-efficient anomaly detection in streaming heterogeneous graphs [J]. International Conference on Knowledge Discovery and Data Mining (KDD'16), 2016.