PanThreat: Global Resource-Based Anomaly Detection for APTs

Wenhao Yan[†‡], Weiheng Wu[†‡], Bingsheng Bi[†‡], Wei Qiao[†‡], Bo jiang[†‡], Yuling Liu[†‡], Junrong Liu[†‡*]
[†]Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
[‡]School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
{yanwenhao, wuweiheng, bibingsheng, qiaowei, jiangbo, liuyuling, liujunrong}@iie.ac.cn

2025 28th International Conference on Computer Supported Cooperative Work in Design (CSCWD) | 979-8-3315-1305-4/25/$31.00 ©2025 IEEE | DOI: 10.1109/CSCWD64889.2025.11033519

***Abstract*—Advanced Persistent Threats (APTs), due to their stealthiness and complexity, have become a significant security challenge for modern enterprises, often causing severe economic losses. To address these threats, researchers have proposed using provenance graphs to model system entities and their dependencies, aiming to capture the complex scenarios of APT attacks. However, existing Provenance-based Intrusion Detection Systems (PIDS) still suffer from the following challenges: (1) Historical interaction information loss due to the truncation of long-term interaction scenarios; (2) The difficulty in capturing long-distance dependencies leads to the loss of crucial contextual information; (3) Existing methods struggle to balance detection efficiency and granularity.**

We introduce PanThreat, an online detection system that performs fine-grained, real-time analysis of host system logs to identify malicious activities. PanThreat combines attributes encoding through Word2Vec and position encoding using Laplacian feature matrices, while retaining long-term interaction histories and effectively modeling long-range dependencies within provenance graphs. This integrated approach significantly enhances detection accuracy. Additionally, PanThreat leverages the parallel processing capabilities of Graph Transformers to improve detection efficiency. Evaluations on the DARPA E3 dataset and StreamSpot database demonstrate PanThreat's effectiveness in detecting complex APT attacks, outperforming four state-of-the-art methods while maintaining an average processing speed of 58,140 events per second.

***Index Terms*—Threat Detection, Transformer, Host Provenance**

I. Introduction

Advanced Persistent Threats (APTs) pose significant risks to organizations through their covert, enduring, and multi-stage strategies [1]. The key to detecting such attacks lies in the profound understanding and analysis of the direct and indirect interactions among internal entities. Security defense systems must be equipped to deeply analyze these interaction patterns in order to accurately identify and respond to potential threats.

To comprehensively understand the relationships between different entities within a system or network, recent research [2]–[12] have primarily focused on the analysis of provenance graphs constructed from raw system logs. By analyzing the intricate interactions embedded within these provenance graphs, it becomes possible to distinguish between benign activities and potential threats. Although prior studies have shown some effectiveness in detecting APT attacks, we have identified three major challenges in their practical applications, which limit their usability.

- **The Challenge of Historical Information Loss due to Temporal Segmentation of Long-Term Interaction Scenarios**: Existing methods often segment large-scale system logs into time windows to manage memory constraints, creating local provenance graphs for each interval. However, this segmentation only captures interactions within the current window, neglecting historical context from previous intervals. This results in critical information loss, limiting the accuracy of detecting persistent threats.
- **The Challenge of Context Loss due to Difficulty in Capturing Long-Distance Interaction Scenarios**: Numerous approaches employ Graph Neural Networks (GNNs) to model relationships within provenance graphs by aggregating information from neighboring nodes. However, GNNs often struggle to capture dependencies between distant nodes, resulting in the loss of critical contextual information needed to detect complex attack patterns.
- **The Challenge of Detection Delays Due to the Inefficiency of Fine-Grained Detection Methods**: Existing approaches, notably Unicorn [3], commonly identify malicious subgraphs rather than isolating specific malicious nodes, necessitating manual investigation of extensive subgraphs. Although fine-grained detection techniques are emerging [10], limitations in graph processing efficiency and resource availability often restrict these methods to offline analysis, leading to delayed responses to Advanced Persistent Threats.

In this paper, we present PanThreat, an online node-level detection system that efficiently and accurately identifies malicious nodes by learning interaction patterns from benign data. PanThreat captures the predictable interaction behaviors between benign nodes and detects potential anomalies when deviations from these patterns occur. To address the limitations of existing methods, which are restricted to analyzing interactions within specific time windows, We proposed a novel full-cycle provenance graph construction strategy that retains the full history of interactions without incurring additional memory overhead. To address the challenge of missing long-range interaction information, PanThreat enables each node

* Corresponding author

to communicate with all other nodes in the graph, rather than being limited to immediate neighbors. This approach allows PanThreat to effectively model long-distance interactions, thereby enhancing its ability to capture and understand complex long-range dependencies within the graph. To improve the efficiency of detection, we introduce Laplacian eigenvectors (the eigenvectors of the Laplacian matrix of a graph) as positional encodings for nodes within PanThreat. This approach, which applies Laplacian transformations to the adjacency matrix of the provenance graph, allows us to efficiently capture the graph's topological information without relying on multi-layer message passing, thus avoiding the inherent computational bottlenecks of traditional GNNs.

In summary, this paper makes the following contributions:

- We designed and implemented an online detection system, PanThreat, which provides node-level alerts, significantly reducing the manual analysis burden on security analysts and enhancing the system's usability.
- We adopt a full-cycle graph construction strategy to preserve historical interactions while capturing long-range dependencies through information exchange between any nodes in the graph. By leveraging both historical and long-range contextual information, PanThreat enhances detection performance.
- We utilize Laplacian eigenvectors as positional encodings to represent the graph's topological structure, instead of relying on message-passing methods, thereby significantly improving the detection efficiency of PanThreat.
- We evaluated PanThreat on the DARPA E3 and Streamspot datasets, and the results demonstrate that it outperforms the state-of-the-art detection systems in accuracy at both coarse and fine-grained while maintaining high throughput.

II. Related Work

A. Provenance-based Intrusion Detection

Provenance Analysis: Provenance Analysis transforms collected system audit logs into a provenance graph, with system entities (e.g., processes, files, sockets) represented as nodes and system events (e.g., write, open) represented as edges.

Heuristic-based provenance graph detection methods [4], [6], [9], [13], [14]: These methods rely on the knowledge of security and domain experts to define a set of predefined rules, which are then used to match potential threats in provenance graphs. However, due to the limited availability of predefined rules and the unpredictability of rule coverage, these methods face challenges in detecting unknown threats.

Learning-based provenance graph detection methods [2], [3], [5], [7], [8], [10]–[12]: By learning the interaction patterns from benign data to detect anomalies, these approaches operate independently of expert knowledge, making them the preferred choice for most detection systems.

B. Methods of Graph Learning

The topology of provenance graphs, derived from host system logs, reveals complex inter-entity relationships. Leveraging both structural and semantic data can yield deeper insights into system interactions. however, existing message-passing graph learning methods are largely limited to capturing local patterns by propagating representations between connected nodes. For instance, Shadewatcher [10] and ProGrapher [12] employ GNNs to aggregate multi-hop neighbor information for node embedding, while Flash [7] and Threatrace [11] use GraphSAGE to reduce computational complexity via neighbor sampling. Kairos [2] leverages Temporal Graph Networks, employing message passing similar to static GNNs.

Although these methods effectively capture local neighborhood information, they often overlook long-range dependencies, limiting their ability to model interactions between distant nodes within the provenance graph. This constraint hampers a comprehensive understanding of complex, non-local relationships. Moreover, due to the iterative nature of message-passing-based graph methods, computational overhead rises sharply as the graph scales, with each layer's message-passing growing exponentially, significantly increasing overall complexity.

III. Methods

The PanThreat method comprises four key components, as shown in the Fig 1. In the first component, PanThreat constructs a provenance graph from log data to capture the full history of system events and their causal relationships. It uses Word2Vec [15] to learn node attribute features and computes Laplacian eigenvectors [16] from the graph's adjacency matrix to represent its topology as node positional features. In the second component, PanThreat extracts node feature and positional encodings from the provenance graph as input. Using a transformer architecture [17], it learns latent relationships between nodes, producing node embeddings with rich global context. In the third component, PanThreat combines the embeddings of connected nodes to form edge embeddings. A multi-layer perceptron (MLP) generates predicted edge encodings, which are compared to actual encodings to detect anomalous interactions. In the fourth component, PanThreat correlates and integrates these anomalies to reconstruct a comprehensive attack scenario.

A. Provenance Graph Construction and Representation

1) Graph Construction: To effectively correlate the interactions between system entities, Panthreat constructs a provenance graph using the raw audit logs. In the provenance graph, nodes represent system entities, while edges depict interaction behaviors between them. We further transform the attribute information and topology in the provenance graph into vectorized representations to support the subsequent graph comprehension and graph analysis phases.

We extract the relationships between the nodes to construct the edge encoding and employ the Edge Encoding to express interactions between nodes. The Laplacian eigenvectors of the graph's adjacency matrix is computed and utilized as Positional Encoding to represent the spatial positions of nodes within the graph. Additionally, we used a Word2Vec model

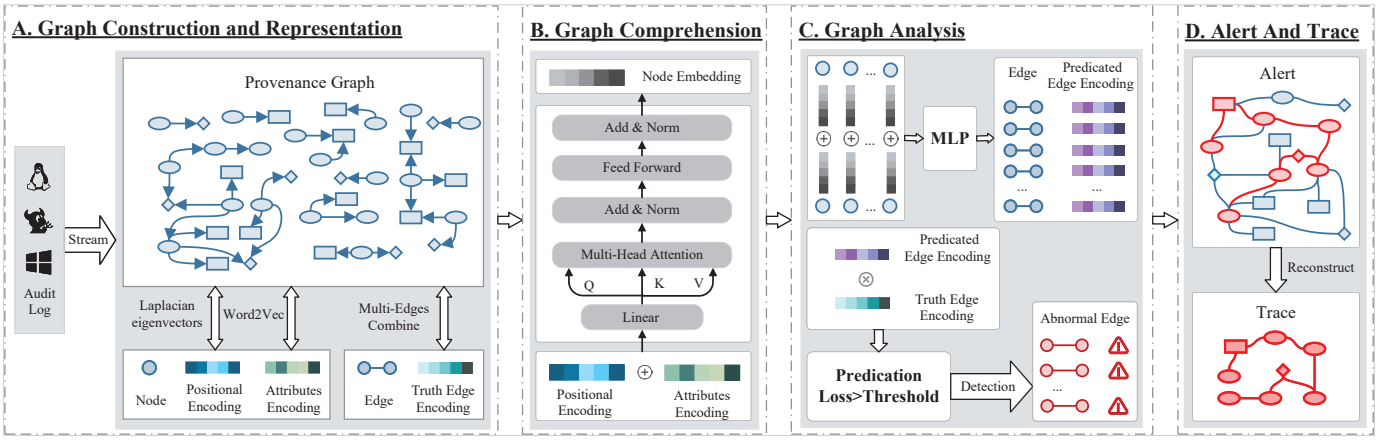

Fig. 1: Overview Of PanThreat

to encode node-related attribute features into node attribute encodings, enabling a more comprehensive capture of each node's semantic features.

2) Edge Encoding: To efficiently represent more information while minimizing resource consumption, we introduce a Multi-Edges Combined technique based on a time decay factor. This approach reduces graph complexity while preserving essential information, including system entity types, attributes, interaction types, and temporal information.

Prior to encoding, we enumerate the number of edge types N_e (e.g., write, read, connect) and map each edge type to an integer index using a mapping function $M_e(e_{i,j})$, where M_e : EdgeType $\rightarrow \{0, 1, \ldots, N_e - 1\}$. A single edge $e_{i,j}$ then represents all interactions between nodes i and j, significantly compressing the graph. As shown in Fig 2, in stage II(b), one

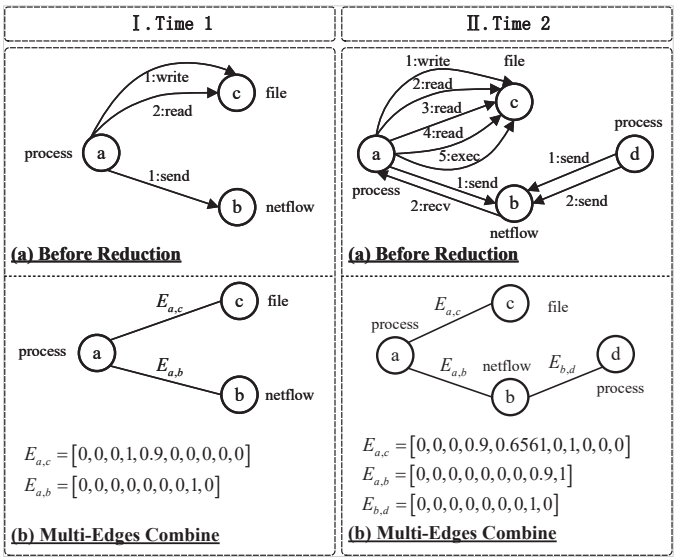

Fig. 2: Example of PanThreat's Edge Encoding

edge between nodes a and c replaces five edges from stage II(a), achieving a five-fold compression. In complex scenarios, millions of redundant edges may be generated, consuming substantial memory resources. By combining multiple edges, we reduce the graph size and alleviate resource constraints.

We consider new behaviors to be more meaningful to the current scenario, while the significance of older behaviors gradually diminishes over time. Therefore, we define a forgetting rate FR to gradually forget outdated behaviors during the update process. Consequently, we construct a sparse edge feature matrix $E_{\text{feat}} \in \mathbb{R}^{N \times N}$, where each non-zero element $E_{\text{feat}}(i, j)$ is an N_e-dimensional vector representing the combined edge features between nodes i and j. Initially, all elements are set to zero, and edges are processed sequentially by timestamp. The update rules for edge $e_{i,j}$ are as follows:

$$\begin{cases} E_{\text{feat}}(i,j) = E_{\text{feat}}(i,j) \times FR, \\ E_{\text{feat}}(i,j)[M_e(e_{i,j})] = 1. \end{cases} \tag{1}$$

As depicted in Fig 2, three additional edges are added between nodes a and c at Time II compared to Time I. Consequently, the edge encoding updates from $[0, 0, 0, 1, 0.9, 0, 0, 0, 0, 0]$ to $[0, 0, 0, 0.9, 0.6561, 0, 1, 0, 0, 0]$.

3) Node Attributes Encoding: To effectively represent interaction scenarios in graphs and fully utilize system logs along with the rich attributes of various system entities, we use encoding techniques to map attributes (e.g., process names, file paths) into vector representations. However, traditional methods such as one-hot encoding and bag-of-words often produce sparse vectors and fail to capture correlations between different attributes.

PanThreat employs a Word2Vec model with contextual information to map these attributes into low-dimensional vectors while preserving deeper correlations among them. As a result, files with similar functionalities (e.g., `/bin/vim`, `/bin/nano`, `/bin/cat`, `/usr/share/vim`) are mapped to closer positions in the vector space, while files with distinct functionalities (e.g., `/bin/vim` and `/bin/python3`) have dissimilar vector representations.

Word2Vec maps semantically similar words to nearby positions in the vector space, capturing their relationships. This ensures that when node attributes are encoded as word vectors, their deeper correlations are preserved. We associate each node's attributes with those of its one-hop neighbors and their interaction types, constructing lists in the format `<src_attribute, edge_type,`

dst_attribute> to serve as Sentences in the Word2Vec corpus. This design increases the likelihood that similar or related attributes will appear in the same context. For instance, both /bin/vim and /bin/nano are commonly involved in writing operations on text files and thus appear more frequently in sentences related to file write actions. As a result, they are mapped to closer positions in the vector space.

4) Node Position Encoding: The Transformer model provides a global receptive field, allowing each piece of information to attend to any other part of the input's context. However, utilizing global attention requires explicitly specifying the positional relationships of the data. In natural language processing, positional encoding helps the Transformer understand the sequential order of words. In graphs, however, nodes are not arranged sequentially, necessitating a positional encoding method that captures both the nodes' relative positions and the graph's topological structure.

Prior research [18] shows that Laplacian eigenvectors offer strong position-awareness, as nodes closer in the graph tend to share similar eigenvector values. Therefore, we compute Laplacian eigenvectors from the graph structure to encode node positions. We precompute the Laplacian eigenvectors for all graphs in the dataset. The eigenvectors are derived from the graph Laplacian matrix factorization:

$$L = I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}, \tag{2}$$

$$Lv_i = \lambda_i v_i,\ i = 1, 2, \ldots, k+1. \tag{3}$$

where A is the adjacency matrix and D is the degree matrix. The k smallest non-trivial eigenvectors, denoted as v_i for each node i, are used as positional encodings for the nodes, with λ_i representing the eigenvalue associated with v_i.

B. Graph Comprehension

PanThreat utilizes the classical Graph Transformer [18], [19] architecture to interpret graph data. To effectively capture latent relationships among all nodes, the Graph Comprehension integrates node attribute encodings with node positional encodings as the node input value $h^{(0)}$. These combined encodings are fed into the Transformer as inputs, enabling a more comprehensive representation of both node attributes and their structural relationships within the graph. Through the Graph Transformer architecture, node embeddings are obtained, which have learned the interaction scenarios within the complete provenance graph.

As shown in Section B of Fig 1, the Transformer architecture consists of multiple stacked layers, each comprising two core components: a self-attention mechanism and a position-wise feed-forward network (FFN). In addition, we apply the layer normalization (LN) before the multi-head self-attention (MHA) and the feed-forward blocks (FFN). Especially, for FFN sub-layer, we set the dimensionality of input, output, and the inner-layer to the same dimension with d. We formally characterize the Panthreat layer as below:

$$h'^{(l+1)} = \text{MHA}(\text{LN}(h^{(l)})) + h^{(l)}, \tag{4}$$

$$h^{(l+1)} = \text{FFN}(\text{LN}(h'^{(l+1)})) + h'^{(l+1)}. \tag{5}$$

Finally, the output $h^{(l+1)}$from the last layer is used as the node embedding. Unlike traditional GNNs, which primarily focus on the local neighborhood of each node, these embeddings capture information from all nodes and learn the full topological structure of the graph.

C. Graph Analysis

Panthreat employs a Multi-Layer Perceptron (MLP) to analyze graphs by predicting the edge type encoding between nodes v_{src} and v_{dst}. The Analysis learns from the hidden representations h output by the Comprehension, modeling the interaction patterns between nodes under the current scenario to predict the edge encoding. The dimensionality of the MLP's output layer corresponds to the number of possible edge types, ensuring it aligns with the dimensionality of the actual edge encoding. The output vector $P(e_{src,dst})$ represents the predicted edge encoding between the source node v_{src} and the destination node v_{dst}:

$$P(e_{src,dst}) = \text{MLP}(h_{src}, h_{dst}) \tag{6}$$

During the training phase, PanThreat optimizes the model by minimizing the prediction loss (PL) between the predicted edge encoding $P(e_{src,dst})$ and the actual edge encoding $L(e_t)$. The loss function is defined as:

$$PL = \text{CrossEntropy}(P(e_{src,dst}), L(e_{src,dst})) \tag{7}$$

During the detection phase, PanThreat evaluates whether node interactions align with normal behavior based on the prediction loss PL. Edges with a low PL indicate that the interaction between nodes conforms to the learned normal behavior of the system. In contrast, edges with a high PL reveal a significant deviation between the actual edge encoding and the predicted normal encoding, identifying these edges as anomalous.

D. Alert And Trace

In large provenance graphs, alerts are often dispersed, requiring extensive forward and backward tracing to investigate causal relationships, which burdens analysts. Furthermore, different stages of APT attacks are scattered across the graph, making their connections hard to discern. PanThreat addresses this by correlating the anomalous edges detected in Section C to generate a concise alert graph. These simplified graphs capture only the nodes and edges directly related to the attack, reducing investigation complexity and helping analysts efficiently trace and understand key attack elements.

IV. Experiments

A. Experiment Settings

1) Implementation: We implemented a PanThreat prototype in Python, utilizing the torch-based version of DGL for the graph learning framework and the Gensim library for the Word2Vec model. We conducted three distinct experiments to comprehensively evaluate PanThreat's performance. All experiments were carried out on a computer running Ubuntu, equipped with an Intel(R) Xeon(R) Silver 4210 CPU, NVIDIA

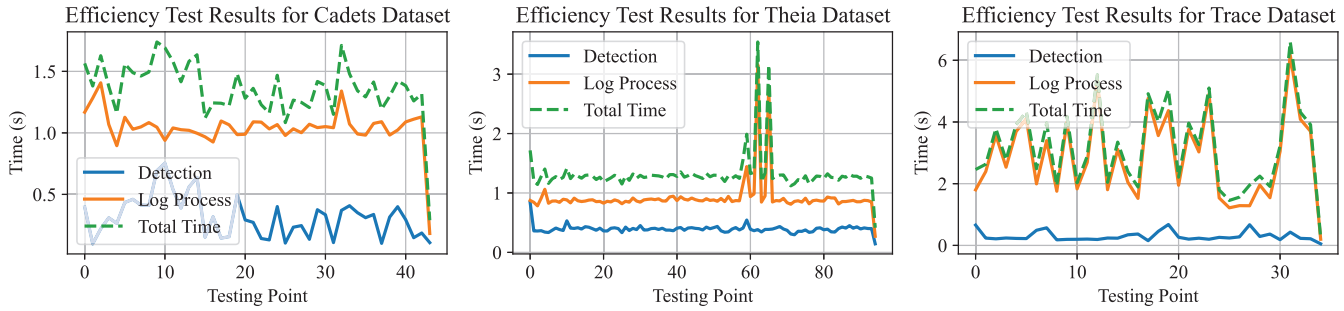

Fig. 3: Efficiency test results on the DARPA dataset

GTX 2080Ti GPU, and 64 GB of RAM. After multiple experiments, we kept all other hyperparameters fixed and adjusted one specific parameter to evaluate PanThreat's performance. By balancing performance and detection effectiveness, we configured the following settings: 6 attention heads and a hidden layer dimension of 48 for the Transformer model, a node positional encoding dimension of 10, an attribute feature encoding dimension of 30, and a forgetting rate (FR) of 0.9.

2) Datasets: We evaluate PanThreat using publicly available datasets from DARPA E3 [20] and StreamSpot [21]. The DARPA E3 dataset, sourced from DARPA's third red-vs-blue team exercise, provides audit logs with both benign and attack behaviors. We use labels from Threatrace [11] for evaluation, enabling node-level detection testing on this dataset. The StreamSpot dataset contains information flow graphs from one attack scenario and five benign scenarios. Since StreamSpot is designed for anomaly detection at the graph level and lacks fine-grained node-level labels for benign or attack classifications, PanThreat's evaluation on this dataset is limited to graph-level anomaly detection. PanThreat determines whether a graph is anomalous by calculating the proportion of anomalous nodes. If this proportion exceeds a predefined threshold, the graph is classified as anomalous.

3) Baselines: To comprehensively evaluate PanThreat, we compare it with two types of provenance graph-based threat detectors: graph-level and node-level detectors. Since the StreamSpot [8] and Unicorn [3] detection methods are limited to graph-level detection, we use these as baselines for graph-level comparisons. For node-level comparisons, we include ThreatTrace [11] and Flash [7] as baselines.

B. Detection Performance

TABLE I: Results of Graph Level Detection Experiment in Steamspot Dataset

Datasets	System	Precision	Recall	Accuracy	F-score
Streamspot	Streamspot	0.73	0.91	0.93	0.81
	Unicorn	0.95	0.97	0.99	0.96
	PanThreat	**0.98**	**0.98**	0.98	**0.98**

We evaluated PanThreat's graph-level performance on the Streamspot dataset, as shown in Table I. PanThreat demonstrates superior precision, recall, and F-score compared to the baseline methods, Streamspot and Unicorn. Unlike Streamspot and Unicorn, which are limited to coarse-grained graph-level detection, PanThreat enables finer-grained analysis of identified malicious subgraphs, allowing for more precise scrutiny rather than issuing broad alerts on densely connected subgraphs within the provenance graph.

We evaluated PanThreat's node-level performance on the DARPA dataset, with Table II presenting a comparison against the baseline methods, ThreatTrace and Flash. PanThreat consistently demonstrates superior precision and F-score across the three datasets in DARPA E3, outperforming both ThreatTrace and Flash. This performance advantage is due to PanThreat's ability to capture extensive long-range dependencies and rich historical context during graph comprehension.

TABLE II: Results of Node Level Detection Experiment in DARPA E3 Dataset

Datasets	System	Precision	Recall	Accuracy	F-score
DARPA E3 Cadets	Threatrace	0.84	0.99	0.99	0.91
	Flash	0.94	0.99	0.99	0.96
	PanThreat	**0.99**	**0.99**	**0.99**	**0.99**
DARPA E3 Theia	Threatrace	0.79	0.99	0.98	0.88
	Flash	0.92	0.99	0.99	0.95
	PanThreat	**0.93**	**0.99**	**0.99**	**0.96**
DARPA E3 Trace	Threatrace	0.82	0.99	0.99	0.90
	Flash	0.95	0.99	0.99	0.97
	PanThreat	**0.97**	**0.99**	**0.99**	**0.98**

C. Detection Efficiency

We evaluated the efficiency of PanThreat using three DARPA datasets. Since PanThreat performs detection on streaming data, we configured it to conduct detection every 100,000 log entries, incorporating historical interaction information to analyze each current batch of 100,000 logs. As shown in Fig 3, PanThreat's processing and detection capabilities are evaluated across these datasets. The variations in log processing times are attributed to differences in audit log formats, which depend on the originating operating systems. PanThreat completes the detection of 100,000 log events in no more than 6.6 seconds, achieving an average processing rate of 58,140 log events per second, with the majority of the time dedicated to log handling.

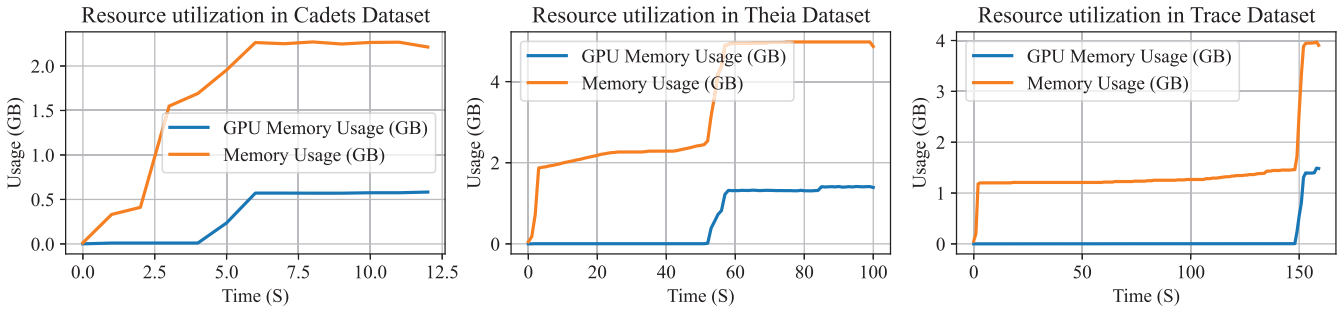

Fig. 4: Efficiency test results on the DARPA dataset

D. Memory Utilization

We evaluated PanThreat's memory utilization across three DARPA datasets during the detection process, considering both system and GPU memory during the detection process. As shown in Fig 4, PanThreat maintains a manageable resource overhead, with system memory usage not exceeding 5GB and GPU memory usage remaining below 1.5GB throughout the operation.

V. Conclusion

We introduce PanThreat, a precise and fine-grained online detection method for identifying Advanced Persistent Threats (APTs). PanThreat utilizes node attribute and positional feature encoders to capture essential graph characteristics, complemented by a Multi-Edges Combined encoder that preserves comprehensive interaction data throughout entire cycles. Additionally, it employs transformer technology to learn long-range dependencies between nodes, enhancing its detection capabilities. Evaluation results demonstrate that PanThreat offers superior detection performance with fast processing speeds and manageable resource utilization.

Acknowledgment

This research is supported by National Key Research and Development Program of China (No.2023YFC2206402), the Strategic Priority Research Program of the Chinese Academy of Sciences (No.XDA0460100), and Youth Innovation Promotion Association CAS (No.2021156).

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] Z. Cheng, Q. Lv, J. Liang, Y. Wang, D. Sun, T. Pasquier, and X. Han, "Kairos: Practical intrusion detection and investigation using whole-system provenance," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2024, pp. 3533–3551.

[3] X. Han, T. Pasquier, A. Bates, J. Mickens, and M. Seltzer, "Unicorn: Runtime provenance-based detector for advanced persistent threats," in *Network and Distributed System Security Symposium*, 2020.

[4] 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.

[5] Z. Jia, Y. Xiong, Y. Nan, Y. Zhang, J. Zhao, and M. Wen, "{MAGIC}: Detecting advanced persistent threats via masked graph representation learning," in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024, pp. 5197–5214.

[6] 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.

[7] M. U. Rehman, H. Ahmadi, and W. U. Hassan, "Flash: A comprehensive approach to intrusion detection via provenance graph representation learning," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 2024, pp. 139–139.

[8] E. Manzoor, S. M. Milajerdi, and L. Akoglu, "Fast memory-efficient anomaly detection in streaming heterogeneous graphs," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016, pp. 1035–1044.

[9] C. Xiong, T. Zhu, W. Dong, L. Ruan, R. Yang, Y. Cheng, Y. Chen, 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*, vol. 19, no. 1, pp. 551–565, 2020.

[10] J. Zengy, X. Wang, J. Liu, Y. Chen, Z. Liang, T.-S. Chua, and Z. L. Chua, "Shadewatcher: Recommendation-guided cyber threat analysis using system audit records," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 489–506.

[11] S. Wang, Z. Wang, T. Zhou, H. Sun, X. Yin, D. Han, H. Zhang, X. Shi, and J. Yang, "Threatrace: Detecting and tracing host-based threats in node level through provenance graph learning," *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 3972–3987, 2022.

[12] F. Yang, J. Xu, C. Xiong, Z. Li, and K. Zhang, "{PROGRAPHER}: An anomaly detection system based on provenance graph embedding," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 4355–4372.

[13] 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.

[14] M. N. Hossain, S. Sheikhi, and R. Sekar, "Combating dependence explosion in forensic analysis using alternative tag propagation semantics," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 1139–1155.

[15] T. Mikolov, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[16] Y. Huang, H. Wang, and P. Li, "What are good positional encodings for directed graphs?" *arXiv preprint arXiv:2407.20912*, 2024.

[17] A. Vaswani, "Attention is all you need," *Advances in Neural Information Processing Systems*, 2017.

[18] V. P. Dwivedi and X. Bresson, "A generalization of transformer networks to graphs," *AAAI Workshop on Deep Learning on Graphs: Methods and Applications*, 2021.

[19] C. Ying, T. Cai, S. Luo, S. Zheng, G. Ke, D. He, Y. Shen, and T.-Y. Liu, "Do transformers really perform badly for graph representation?" *Advances in neural information processing systems*, vol. 34, pp. 28 877–28 888, 2021.

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

[21] V. V. E. Manzoor, S. Momeni and L. Akoglu, "Streamspot code and data," 2016, https://sbustreamspot.github.io/.