2025 International Conference on Emerging Smart Computing and Informatics (ESCI)
AISSMS Institute of Information Technology, Pune, India. Mar 5-7, 2025

Efficient Host Threat Detection Using The Procsage Method

Dr. Shashikumar D R
Dept. Computer Science & Engineering
Sai Vidya Institute of Technology
Bangalore, India
shashikumar@saividya.ac.in

Prof. Madhura N
Dept. Computer Science & Engineering
Sai Vidya Institute of Technology
Bangalore, India
madhura.n@saividya.ac.in

S Lekha Reddy
Dept. Computer Science & Engineering
Sai vidya Institute of Technology
Bangalore, India
slekhareddy.22cs@saividya.ac.in

Navya M
Dept. Computer Science & Engineering
Sai Vidya Institute of Technology
Bangalore, India
navyam.22cs@saividya.ac.in

Purav S
Dept. Computer Science & Engineering
Sai Vidya Institute of Tecchnology
Bangalore, India
puravs.22cs@saividya.ac.in

Shreyas H R
Dept. Computer Science & Engineering
Sai Vidya Institute of Technology
Bangalore, India
shreyashr.22cs@saividya.ac.in

Abstract— **Advanced Persistent Threats (APTs) pose a significant challenge to modern cybersecurity due to their stealth and complexity. This paper introduces a scalable host threat detection framework that utilizes provenance graphs and graph representation learning to overcome limitations of traditional defense mechanisms. By optimizing process- and thread-centric analyses within provenance graphs, the system minimizes computational overhead while maintaining robust behavioral insights. Leveraging semantic and topological feature extraction with GraphSAGE-based learning, the framework achieves precise anomaly detection via clustering techniques. Validation on benchmark datasets demonstrates substantial improvements in accuracy, efficiency, and scalability, showcasing the framework's potential for real-world applications.**

Keywords— Provenance Graphs, GraphSAGE, Anomaly Detection, Graph Representation Learning, Advanced Persistent Threats, Cybersecurity.

I. Introduction

The rise of **Advanced Persistent Threats (APTs)** has redefined the landscape of cybersecurity, posing an ongoing challenge to organizations globally. A stark example is the infamous SolarWinds attack, where highly sophisticated adversaries exploited supply chain vulnerabilities, gaining prolonged and unauthorized access to sensitive systems. Such incidents underscore the stealth, persistence, and adaptive nature of APTs, which exploit system weaknesses while evading traditional detection mechanisms.

Existing defense methods, such as signature-based systems or rule-driven anomaly detection, are increasingly ineffective against these advanced threats. These conventional approaches lack the agility to address the dynamic attack patterns and complex relationships inherent in APT operations. **Provenance graphs** have emerged as a cutting-edge solution, enabling detailed mapping of system interactions by representing entities like processes, files, and network connections and uncovering temporal and contextual correlations critical to identifying APT behavior. Despite their promise, the computational burden of building and analyzing these graphs—especially when dealing with vast audit logs—has hindered their scalability and real-world adoption.

This project introduces a novel, **efficient, and scalable host threat detection framework** leveraging provenance graphs. By refining the ProcSAGE methodology, the system focuses on process and thread nodes within the graph to reduce computational overhead without sacrificing analytical depth. The framework integrates semantic and topological feature extraction with graph-based learning techniques using the **GraphSAGE** model, facilitating precise anomaly detection through clustering-based approaches.

Key contributions include:

- **Enhanced computational efficiency:** Minimizing resource demands while processing extensive audit logs.
- **Improved detection accuracy:** Leveraging graph representation learning to reliably identify complex attack patterns.
- **Scalability for practical deployment:** Optimized methods for processing real-world cybersecurity datasets.

Experimental validation on benchmark datasets demonstrates the framework's potential to deliver state-of-the-art performance in host-based anomaly detection. These advancements pave the way for robust and scalable defenses, bridging the gap between theoretical potential and practical application in combating the ever-evolving threat of APTs.

II. Literature Survey

The detection of **Advanced Persistent Threats (APTs)** has garnered significant research attention due to the sophisticated and prolonged nature of such attacks. This literature survey reviews existing methodologies, highlighting their relevance to the development of the proposed framework.

A. *Provenance Graphs as a Core Tool*

Provenance graphs have become a cornerstone for understanding causal relationships in system events, representing system activities as interconnected nodes (e.g., processes, files, and network events). These graphs enable the correlation of temporally distant events, allowing analysts to identify complex attack patterns. While powerful, the computational demands of building and analyzing these graphs remain a challenge. The following approaches demonstrate the evolution of techniques addressing these challenges:

ProcSAGE: Process-Centric Optimization:
ProcSAGE, introduced by Xu et al., innovatively optimizes the construction and analysis of provenance graphs for host-based anomaly detection. Unlike traditional graph methods, ProcSAGE adopts a **process- or thread-centric focus**,

reducing graph complexity and computational overhead while preserving critical behavioral insights. By integrating **graph representation learning** via the GraphSAGE model, it captures contextual and structural information from nodes and their neighborhoods. This enhances detection accuracy with minimal resource usage, as evidenced by its 69% reduction in processing time and 78% reduction in memory consumption compared to conventional methods (evaluated on the StreamSpot dataset).

Real-Time Streaming Graph Anomaly Detection: Streaming graph frameworks like StreamSpot, proposed by Manzoor et al., emphasize real-time anomaly detection for high-velocity system logs. Using clustering based on substructure frequency in heterogeneous graphs, StreamSpot achieves memory efficiency while processing over **100,000 edges per second**. Although its **centroid-based clustering** differs from ProcSAGE's graph learning approach, both aim to optimize anomaly detection for large-scale datasets, emphasizing real-time application and computational efficiency.

Graph Sketching and Runtine Anomaly Detection: UNICORN, developed by Han et al., extends provenance graph analysis to runtime scenarios. It employs fixed-size **graph sketches** to summarize evolving provenance graphs, supporting **incremental updates** that capture long-term system behavior. While this approach is resilient to poisoning attacks and offers efficient processing, its reliance on a static model limits adaptability. Unlike ProcSAGE, UNICORN does not integrate graph learning, focusing instead on clustering sketch representations for anomaly detection.

Squence-Based and Rule-Driven Methods: Traditional sequence-based anomaly detection techniques, such as sliding window methods for syscall analysis, are insufficient against the complex, multi-stage attacks associated with APTs. Provenance graph-based methods like SLEUTH and Holmes improve interpretability through their causal structure, but their reliance on **expert-defined rules** restricts adaptability to novel attack scenarios. In contrast, approaches such as ProcSAGE and StreamSpot leverage **data-driven anomaly detection** to overcome these limitations, ensuring scalability and flexibility.

B. Addressing Scalability in Provenance Graphs

The scalability of provenance graph methods has been a persistent challenge. Techniques such as **graph compression** and **partitioning** (e.g., LogGC and DeepComm) reduce graph sizes but often lack integration with advanced graph learning models. ProcSAGE offers a unique solution by embedding optimizations directly into its graph learning pipeline. This enables efficient processing of large audit logs while balancing **computational efficiency and detection accuracy**.

III. System Requirements

The implementation of the proposed host threat detection framework is contingent upon meeting specific hardware, software, and data prerequisites. These requirements ensure efficient execution, scalability, and reproducibility.

A. Hardware and Software Requirements

1. **Hardware Requirements:**

- **Processor:** A multi-core processor, such as the Intel Xeon Silver 4210 (2.20 GHz) or equivalent. Processors with higher clock speeds and more cores will enhance performance.
- **GPU:** An NVIDIA Tesla P100 with 16 GB memory is recommended for graph-based computations. Alternatives include the NVIDIA V100, A100, or consumer-grade options like the NVIDIA RTX 3090 or RTX 4070 (8–16 GB VRAM), depending on resource availability.
- **RAM:** A minimum of 32 GB is required to handle intermediate computations and graph storage, but 64–128 GB is recommended for processing large-scale datasets.
- **Storage:** At least 500 GB of SSD storage is necessary for efficient I/O operations on audit logs and results. Additional storage is advisable for large real-world datasets.

Resource-Constrained Alternatives:

- **Processor:** Use a high-end consumer CPU like the AMD Ryzen 7 series or Intel Core i7.
- **GPU:** Opt for mid-tier GPUs like the NVIDIA GTX 1660 or RTX 3060.
- **RAM:** Scale down to 16 GB and enable swap space to compensate for memory limitations.
- **Storage:** External or network storage solutions can supplement internal SSDs.

2. **Software Requirements**

- **Operating System:** Ubuntu 22.04 LTS is the recommended OS, but other Linux distributions (e.g., Debian or CentOS) compatible with system audit tools may suffice.
- **Programming Environment:** Python 3.8 or newer is essential. The following libraries should be installed for reproducibility:
  - **PyTorch Geometric (PyG):** Version 2.x for GraphSAGE implementation.
  - **Scikit-learn:** Version 1.0 or newer for clustering and evaluation tasks.
  - **NumPy and pandas:** Standard versions for data manipulation and preprocessing.
  - **NetworkX:** Version 2.x for constructing and analyzing provenance graphs.
  - **Matplotlib:** For result visualization.

Resource-Constrained Alternatives:

- **PyTorch Geometric Alternative:** For smaller datasets or environments without GPU support, consider DGL (Deep Graph Library) or NetworkX for lightweight graph manipulation.
- **Cluster-based Approaches:** Use pre-existing clustering libraries like HDBSCAN as an alternative to Scikit-learn if memory constraints exist.

Log Collection and Analysis Tools:

- **Auditd:** Preferred for generating syscall event logs.
- **Alternatives:** CamFlow provides additional audit capabilities for real-time provenance tracking.

B. Data Requirements

1. **Input Data:**

- **System Audit Logs:** The framework relies on logs capturing critical syscall events (e.g., open, read, fork, exec) from tools like auditd or CamFlow. These logs should include timestamped activity for accurate graph construction.

2. **Benchmark Datasets:**

- **Validation Dataset:** The StreamSpot dataset serves as the benchmark for validating the system's detection accuracy, efficiency, and robustness.
- **Alternatives:** Publicly available datasets, such as the DARPA Intrusion Detection Dataset, UNICORN evaluation sets, or custom logs from controlled environments, can also be used.

C. Reproducibility Guidelines

For users aiming to replicate this work:

1. Install the specified software libraries with their exact versions using a package manager (e.g., pip). Utilize a virtual environment (e.g., venv or Conda) to ensure version consistency.
2. Use the same benchmark dataset (StreamSpot) or include preprocessing steps for alternative datasets.
3. Configure audit log collection to align with predefined settings, ensuring consistency across different test environments.
4. For resource-limited setups, document any changes to hardware or software configurations that may affect performance metrics.

IV. System Design

The system is designed as a modular, efficient, and scalable host threat detection framework based on provenance graph representation learning. It integrates data collection, processing, graph construction, feature extraction, model training, and anomaly detection. The following sections outline the key components and workflow of the system design.

A. Architecture Overview

The system is divided into five main modules:

- Data Collection: Collects system call audit logs from the host system.
- Data Preprocessing: Filters and structures the raw audit log data for graph construction.
- Feature Extraction: Extracts semantic and topological features for graph nodes.
- Anomaly Detection: Uses a GraphSAGE-based model to identify anomalies through embeddings and clustering.

B. Workflow

The proposed system processes system audit logs through a structured workflow comprising five stages as shown in Fig 1. First, data collection involves capturing raw system audit logs using tools like auditd, continuously monitoring kernel-level events such as system calls and storing logs in structured formats containing details like pid, ppid, syscall, and comm.

Next, in the data preprocessing stage, the raw logs are filtered to extract relevant syscall events (e.g., open, exec, fork), transformed into CSV format for scalability, and incrementally processed to ensure efficient handling of large datasets. Following this, graph construction generates a provenance graph by creating nodes (representing processes, files, and syscalls) and edges (linking nodes based on interactions, such as processes invoking syscalls).

The graph is updated incrementally to reflect real- time system activities and stored in formats like GML. In the feature extraction stage, the system derives semantic features (e.g., frequency of syscalls) and topological features (e.g., node degree and clustering coefficients) from the graph, saving the features in CSV format for downstream tasks.

Finally, the anomaly detection stage uses GraphSAGE to generate embeddings by aggregating node and neighborhood features. These embeddings are clustered using techniques like k-means, and anomalous.

C. Data Requirements

Input Data: The proposed system relies on system audit logs generated by tools such as auditd or Camflow. These logs must capture relevant syscall events such as open, read, fork, and exec, which are critical for constructing provenance graphs.

Dataset for Validation: To benchmark the system's performance, The StreamSpot dataset (or other equivalent datasets containing both benign and attack scenarios) was utilized. These datasets facilitate the evaluation of detection accuracy and robustness across diverse scenarios.

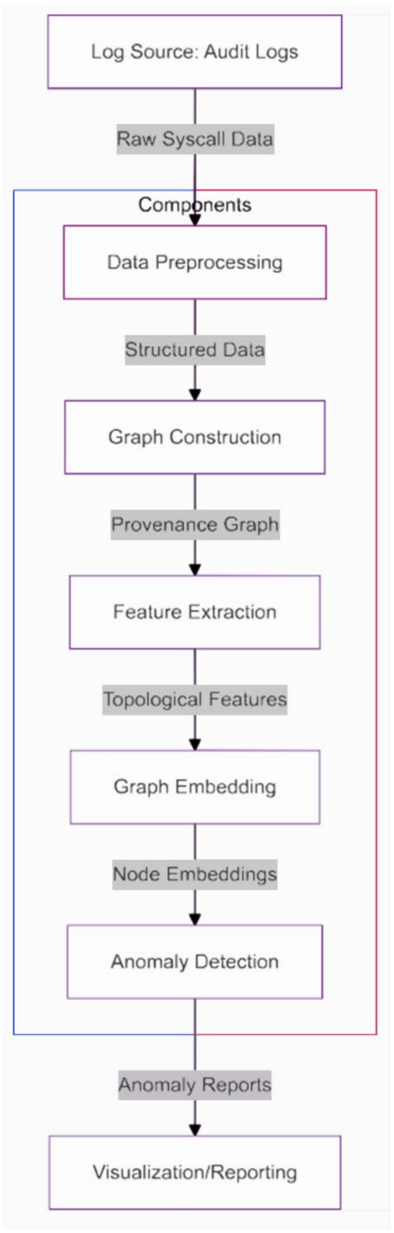


Fig. 1. High Level Diagram

V. Methodology

The proposed host-based threat detection framework integrates system audit data collection, provenance graph modeling, and anomaly detection using graph representation learning. It is specifically designed to detect Advanced Persistent Threats (APTs) efficiently by leveraging the semantic and topological characteristics of system entities.

Data Collection and Preprocessing: The framework begins with data collection, where system audit logs are captured using kernel-level tools like auditd. These logs monitor system events, focusing on key system calls (e.g., open, exec, Provenance Graph Construction: A provenance graph is constructed to model system interactions. In this graph, nodes represent system entities (e.g., processes, syscalls, files) and are enriched with metadata such as execution commands and process IDs for context. Edges denote relationships, such as process-to-syscall or syscall-to-file interactions. The graph is updated incrementally as new log data becomes available, avoiding full reconstruction. Graphs are stored in formats like GML for efficient access and further processing.

Feature Extraction: From the constructed provenance graph, both semantic and topological features are extracted. Semantic features, derived from process logs, include syscall frequencies, executed commands, and process metadata. Topological features, calculated from the graph structure, include the average degree of neighboring nodes and the local clustering coefficient, which reflects the tendency of nodes to form clusters. These features are computed using graph algorithms within a 2-hop neighborhood to minimize computational overhead.

Graph Representation Learning: The framework employs GraphSAGE, a Graph Neural Network (GNN) designed to aggregate features from node neighborhoods and generate embeddings. Input to the model includes node features and the graph structure. The architecture consists of a two-layer GraphSAGE model with ReLU activation, a hidden layer of 16 dimensions, and an output embedding layer of 8 dimensions. Training is performed using a cross-entropy loss function with an Adam optimizer (learning rate: 0.01), leveraging labeled data for supervised learning.

Clustering and Anomaly Detection: The embeddings generated by GraphSAGE are clustered using k-means, where cluster centers represent typical process behavior patterns. Anomalies are identified using distance-based scoring, flagging nodes whose distance from the nearest cluster center exceeds a predefined threshold. Additionally, Z-score analysis is used to identify outliers based on the distribution of feature values or distances, enhancing the anomaly detection process.

VI. Evaluation

The Z-score analysis is a statistical method used to identify outliers in a dataset by measuring how far a data point is from the mean in terms of standard deviations. The Z-score is a way of standardizing the values in a dataset, allowing comparison between different datasets or features that may have different scales.

The Z-score for a data point xxx is calculated using the following formula:

$$Z = \frac{X - \mu}{\sigma}$$

Where:
x is the value of the data point.
μ is the mean of the dataset (or feature).
σ is the standard deviation of the dataset (or feature).

- $Z = 0$: The data point is exactly at the mean.
- $Z > 0$: The data point is above the mean.
- $Z < 0$: The data point is below the mean.
- $Z > 3$ or $Z < -3$: Generally, a Z-score greater than 3 or less than -3 is considered an outlier, as it indicates that the data point is more than 3 standard deviations away from the mean.

VII. Implementation

Collecting System Audit Logs:

- The auditd tool captures system calls logs from the host operating system. These logs provide detailed information about processes, files, and system calls.
- Log entries are continuously monitored, and new data is saved into a structured file (audit.log) for subsequent processing.

Extracting Relevant Syscalls:

- A regular expression (regex)-based filter is applied to the raw audit logs to extract entries related to system calls (type=SYSCALL).
- The extracted data is transformed into a structured format (CSV), containing key fields such as process ID (pid), parent process ID (ppid), executable path (exe), command (comm), and system call (syscall).

Provenance Graph Construction:

- Graph Model: The provenance graph consists of nodes representing processes, files, and system calls, with edges representing the interactions between these entities.
- Node Creation: Each entry in the CSV file, whether a process, syscall, or file, is represented as a node. Metadata, such as process ID (pid) and executable path (exe), is stored as node attributes.
- Edge Creation: Directed edges are added based on system event flows, such as a process invoking a system call.
- Library Used: The networkx library is utilized for constructing and manipulating the provenance graph.

Semantic Features:

Features directly extracted from the process logs, such as the frequency of system calls, provide insights into the activity and behavior of individual processes.

Topological Features:

- Graph Algorithms: Metrics such as the average degree (the connectivity of a process node with its neighbours) and the clustering coefficient (measuring the interaction within a process's neighbourhood) are calculated.

- These features are stored in a CSV file for use in graph- based learning.

GraphSAGE Model:

- The goal of the GraphSAGE model is to generate embeddings for graph nodes that capture both semantic and topological properties.
- Input: Node features and graph structure (represented as an edge list).
- Architecture: A two-layer GraphSAGE model with ReLU activation functions.
- Embedding Size: 8 dimensions.
- Training: Nodes are labelled based on known benign or malicious behaviours, and cross-entropy loss is used for optimization.
- Tools Used: The `torch_geometric` library is employed to implement the GraphSAGE model.

Clustering:

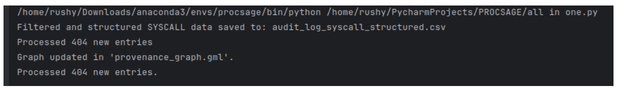
K-means clustering is applied to the embeddings generated by GraphSAGE. The nodes are grouped based on behavioral similarity, with cluster centers representing typical behavior patterns. Outliers or nodes that deviate significantly from these clusters indicate potential anomalies.

Anomaly Scoring:

- Distance-Based Scoring: Nodes are assigned scores based on their distance from the nearest cluster center. A predefined threshold is used to identify anomalies.
- Z-Score Analysis (optional): Z-score analysis is applied to detect outliers in feature distributions.

VIII. Results

In Fig 2, the results demonstrate the initial stages of the system's operation, specifically focusing on the processing of system audit logs. The figure captures the output generated during the data preprocessing phase, where the raw audit logs are filtered to extract relevant syscall events and structured into a CSV format.

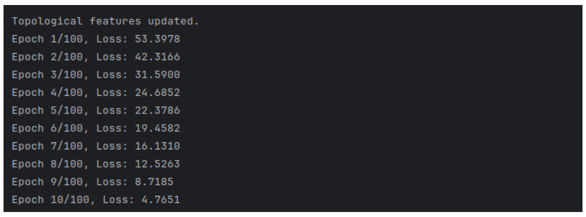


```
/home/rushy/Downloads/anaconda3/envs/procsage/bin/python /home/rushy/PycharmProjects/PROCSAGE/all in one.py
Filtered and structured SYSCALL data saved to: audit_log_syscall_structured.csv
Processed 404 new entries
Graph updated in 'provenance_graph.gml'.
Processed 404 new entries.
```

Fig. 2. Preprocessing data

The message "Processed 404 new entries" indicates the number of new syscall entries that have been successfully extracted and appended to the structured dataset. This step ensures that only the most recent and relevant data is analyzed, preventing redundant processing and enabling incremental updates for scalability. The output confirms the system's ability to handle and process large volumes of audit data efficiently, setting the stage for subsequent graph construction and analysis.

In Figure 3, the results depict the progress of two critical stages in the system's operation: feature extraction and model training.



```
Topological features updated.
Epoch 1/100, Loss: 53.3978
Epoch 2/100, Loss: 42.3166
Epoch 3/100, Loss: 31.5900
Epoch 4/100, Loss: 24.6852
Epoch 5/100, Loss: 22.3786
Epoch 6/100, Loss: 19.4582
Epoch 7/100, Loss: 16.1310
Epoch 8/100, Loss: 12.5263
Epoch 9/100, Loss: 8.7185
Epoch 10/100, Loss: 4.7651
```

Fig. 3. Feature Extraction and Training

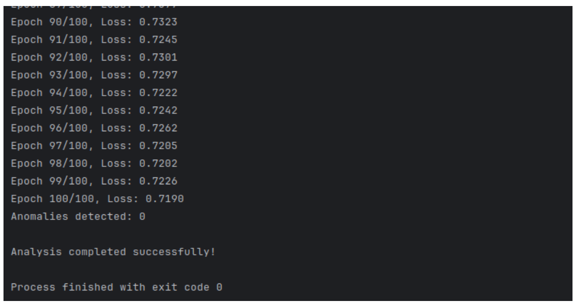
The message "Topological features updated" signifies the successful extraction and storage of topological characteristics, such as average degree and clustering coefficient, for each node in the provenance graph. These features play a vital role in characterizing the structural relationships between processes and their interactions within the system.

Following this, the output displays the training process of the GraphSAGE model, with each epoch showing the progression of the loss function. The epoch-wise output reflects the optimization of the model as it learns to generate node embeddings by aggregating both semantic and topological features. The gradual reduction in loss values across epochs demonstrates the model's convergence, ensuring that the generated embeddings effectively capture the underlying patterns in the data for clustering and anomaly detection.

In Fig 4, the results highlight the system's ability to detect anomalies based on the analysis of extracted embeddings and clustering outputs. The output displays a list of anomalies, identified as programs or processes deviating from expected behavioral patterns.

These anomalies are flagged based on their distance from cluster centers or abnormal feature distributions, evaluated using predefined thresholds and z-score analysis. Each identified anomaly corresponds to a process exhibiting characteristics significantly different from those of the normal clusters, indicating potential malicious activity or irregular system behavior.

This output showcases the final stage of the system's operation, emphasizing its practical utility in identifying and reporting suspicious activities for further investigation or mitigation. The detailed anomaly information provides actionable insights, reinforcing the system's effectiveness in real-world threat detection scenarios.



```
Epoch 90/100, Loss: 0.7323
Epoch 91/100, Loss: 0.7245
Epoch 92/100, Loss: 0.7301
Epoch 93/100, Loss: 0.7297
Epoch 94/100, Loss: 0.7222
Epoch 95/100, Loss: 0.7242
Epoch 96/100, Loss: 0.7262
Epoch 97/100, Loss: 0.7205
Epoch 98/100, Loss: 0.7202
Epoch 99/100, Loss: 0.7226
Epoch 100/100, Loss: 0.7190
Anomalies detected: 0

Analysis completed successfully!

Process finished with exit code 0
```

Fig. 4. Anomaly Detection

The below images are just a visualization created out of curiosity, a huge dataset with 10's of thousands of syscall data, when visualized with matlab module in python, will be visualized as shown in Fig 5.

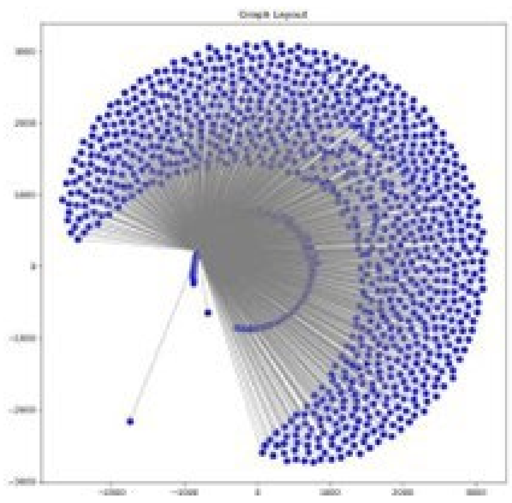


Fig. 5. Game data only

IX. Future Work

The proposed host threat detection framework offers promising results, but several avenues for improvement and further exploration remain. Future work will focus on the following aspects:

Streaming Capability: Enhance the system to process and analyze high-volume audit logs in real time. This involves integrating stream processing frameworks to handle dynamic and large-scale datasets effectively.

Model Transferability: Investigate methods to improve the adaptability of the trained models across different operating systems and environments without requiring extensive retraining. This includes developing domain adaptation techniques to generalize performance.

Scalability: Scale the framework to handle increasingly complex system environments and larger provenance graphs. Efficient graph storage mechanisms and distributed computing frameworks will be explored to support this scalability.

Integration with Security Infrastructure: Combine the proposed framework with existing Security Information and Event Management (SIEM) systems for more comprehensive threat detection and response capabilities.

Reducing Computational Overhead: Optimize graph-based learning models and anomaly detection algorithms to minimize CPU and memory usage during execution. This ensures the framework remains lightweight for deployment on resource-constrained systems.

Expanding Threat Coverage: Extend the detection capabilities to encompass a broader range of threat patterns, including insider threats and zero-day exploits, through enhanced feature engineering and anomaly detection strategies.

X. Conclusion

This project presents a robust and efficient host threat detection system designed to address the challenges posed by Advanced Persistent Threats (APTs). Leveraging provenance graph-based modeling and graph representation learning, the system effectively identifies anomalies in system behavior, even in large-scale environments.

The implementation emphasizes computational efficiency by focusing on process and thread nodes within provenance graphs, reducing resource consumption while retaining critical insights. Semantic and topological features extracted from the graphs are used to train a GraphSAGE-based model, enabling precise representation of system entities. The integration of clustering and anomaly detection techniques further enhances the system's ability to distinguish between normal and malicious activities.

Evaluation on benchmark datasets demonstrates the system’s scalability, accuracy, and resource efficiency. It achieves significant reductions in time and memory consumption compared to traditional graph-based methods, making it suitable for real-world deployment in dynamic and large- scale systems.

This work highlights the potential of combining graph representation learning with intelligent feature selection for advanced threat detection. Future research could explore enhancements such as real-time streaming capabilities, adaptive models to address concept drift, and the extension of the framework to complex, heterogeneous environments.

References

[1] Boyuan Xu, Yiru Gong, Xiaoyu Geng, Yun Li, Cong Dong, Song Liu, Yuling Liu, Bo Jiang, and Zhigang Lu, *ProcSAGE: An Efficient Host Threat Detection Method* Based *on Graph Representation Learning*. Springer, 2024.

[2] Xueyuan Han, Thomas Pasquier, Adam Bates, James Mickens, and Margo Seltzer, *UNICORN: Runtime Provenance-Based Detector for Advanced Persistent Threats*. Presented at the Network and Distributed System Security Symposium (NDSS), 2020.

[3] Md Nahid Hossain, Sadegh M. Milajerdi, Junao Wang, Birhanu Eshete, Rigel Gjomemo, R. Sekar, Scott Stoller, and V.N. Venkatakrishnan, *SLEUTH: Real-time Attack Scenario Reconstruction from COTS Audit Data*. Presented at the USENIX Security Symposium, 2017.

[4] Emaad Manzoor, Sadegh M. Milajerdi, and Leman Akoglu, *Streamspot: Fast Memory-efficient Anomaly Detection in Streaming Heterogeneous Graphs*. Presented at the ACM SIGKDD Conference on Knowledge Discovery and Data Mining (SIGKDD), 2016.

[5] Wajih Ul Hassan, Mohammad A. Noureddine, et al., *OmegaLog: High-Fidelity Attack Investigation via Transparent Multi-layer Log Analysis*. Presented at the Network and Distributed System Security Symposium (NDSS), 2025.

[6] Min Du and Feifei Li, *DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning*. Presented at the ACM SIGKDD Conference on Knowledge Discovery and Data Mining (SIGKDD), 2017.

[7] Peizhao Hu and Stephen McCamant, *ProTracer: Towards Practical Provenance Tracing by Alternating Data and Control Flow*. Presented at the USENIX Security Symposium, 2019.

[8] Marko Bertoglio, et al., *Log2Graph: A Graph-Based Approach for Predictive Modeling of Insider Threat Detection*. Published in IEEE Transactions on Dependable and Secure Computing, 2023.

[9] Wajih Ul Hassan, Shengjian Guo, et al., *NoDoze: Combatting Threat Alert Fatigue with Automated Provenance Triage*. Presented at the Network and Distributed System Security Symposium (NDSS), 2025.