Knowledge Transfer from LLMs to Provenance Analysis: Semantic-Augmented APT Detection

Fei Zuo*, Junghwan Rhee*, Yung Ryn Choe§, Chung Hwan Kim†
*University of Central Oklahoma, §Sandia National Laboratories, †University of Texas at Dallas
{fzuo, jrhee2}@uco.edu, yrchoe@sandia.gov, chungkim@utdallas.edu

***Abstract*—Advanced Persistent Threats (APTs) have caused significant losses across a wide range of sectors, including the theft of sensitive data and harm to system integrity. As attack techniques grow increasingly sophisticated and stealthy, the arms race between cyber defenders and attackers continues to intensify. The revolutionary impact of Large Language Models (LLMs) has opened up numerous opportunities in various fields, including cybersecurity. An intriguing question arises: can the extensive knowledge embedded in LLMs be harnessed for provenance analysis and play a positive role in identifying previously unknown malicious events? To seek a deeper understanding of this issue, we propose a new strategy for taking advantage of LLMs in provenance-based APT detection. In our design, the state-of-the-art LLM offers additional details in provenance data interpretation, leveraging their knowledge of system calls, software identity, and high-level understanding of application execution context. The advanced contextualized embedding capability is further utilized to capture the rich semantics of event descriptions. We conducted an empirical analysis of how LLMs interpret system events and obtained multiple insightful findings. We also verified through comparison that the contextualized embeddings generated by LLMs for provenance data outperform existing counterparts. Finally, our evaluation using real-world data shows that supervised threat detection achieves a precision of 99.0%, and semi-supervised anomaly detection attains a precision of 96.9%. The findings suggest that LLMs play a positive role in provenance-based APT detection and demonstrate promising potential for future applications in this area.**

***Index Terms*—Intrusion Detection, APT Detection, Provenance Analysis, GPT, LLM.**

I. Introduction

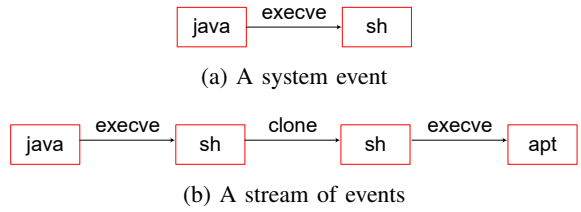
In recent years, advanced persistent threats (APTs) targeting key sectors such as government, finance, and business have been on the rise. These cyberattacks employ increasingly complex techniques, are prolonged in duration, and are difficult to detect, resulting in significant economic losses. As early as 2017, APT actors breached the network of the credit reporting agency Equifax, ultimately resulting in losses exceeding 425 million US dollars [1]. According to CrowdStrike's report [2], "*cloud environment intrusions increased by 75% from 2022 to 2023*". Meanwhile, the number of victims named on dedicated leak sites increased by 76%, indicating that data-theft extortion remains rampant.

Among the emerging technologies for robust APT detection, system provenance analysis is being considered as a promising mechanism, thus attracting widespread attention. As cyber threats become more complex and frequent, traditional approaches to threat detection and response are proving inadequate. Therefore, we also notice that leveraging the progress in AI to assist and automate system provenance analysis has become more of a need than an option. Industry statistics show that in practical incident response applications, fully deployed AI-driven systems "*were able to identify and contain a breach 28 days faster than those that didn't, saving USD 3.05 million in costs*" for the organizations [3]. The recent survey [4] indicates that "*70% of cybersecurity professionals believe AI is highly effective in detecting previously undetectable threats*". Additionally, 73% of cybersecurity teams want to shift their focus to an AI-empowered preventive strategy.

System provenance data are essentially a set of events with temporal order and causal relationships. Modern auditing tools such as `Sysdig` [5] can track event streams with timestamps, and for each event they provide a rich set of properties. In the presence of large volumes of provenance data, manual analysis suffers from two major limitations: it is time-consuming and requires domain expertise from analysts. However, considering the success of large language models (LLMs) in a wide range of security applications, we recognize that LLM-assisted system provenance analysis may offer several notable benefits.

First, the extensive knowledge and strong reasoning capabilities of LLMs can be leveraged to enrich the semantics of system event descriptions, thereby helping security analysts better understand system behavior. For instance, we observe that LLMs are capable of interpreting the functionality of individual system calls (e.g., `execve` or `clone`) and explaining the purpose of critical system files (e.g., `/etc/passwd`). We will provide more specific examples and detailed explanations in Section IV-B. Moreover, provenance data is property-rich and semantically diverse unstructured sequential data. LLMs are well-suited to understanding textual and unstructured data, giving them inherent advantages in analyzing such data without extensive manual feature engineering.

Though we have witnessed a number of meaningful attempts to integrate LLMs into various cybersecurity applications [6]–[8], the potential of LLMs to enhance provenance-based APT detection remains under-explored. Therefore, in this work, we aim to explore how LLMs can assist in analyzing complex system events and develop a pipeline that leverages their ability to detect APT attacks in practice. More concretely, for system events collected by `Sysdig`, we adopt a heuristics-based system event representation method according to their specific types. We then fully leverage the extensive knowledge of LLMs to augment the semantics of these events' descriptions. We empirically studied the performance of LLMs in in-



(a) A system event

(b) A stream of events

Fig. 1: System provenance data

terpreting diverse system events and obtained several insightful findings. In the later APT detection phase, we comprehensively consider supervised and semi-supervised anomaly detection models. Our evaluation was conducted based on a publicly accessible dataset, with samples derived from real-world attacks in cloud environments. The experiment results show that our proposed technique is able to distinguish malicious events from benign ones, maintaining a high level of accuracy even the attack was previously unseen. We have made our research resources publicly available to facilitate future related studies[1].

II. Preliminaries

A system event is the fundamental unit in provenance data used to track and record system-level activities. As Figure 1(a) shows, it consists of two interrelated system entities and the operation between them. An entity refers to a component (e.g., a process or a file) within a system responsible for producing, modifying, or processing information and resources. The operation exerted by one system entity on another is described using a system call. Interdependent events can be organized into a stream based on their temporal order or causal relationships, thereby representing a series of system operations, as illustrated in Figure 1(b). The order of events affects semantics, and events are connected by directed edges that indicate the flow of data or control.

When event streams are treated as a basic analysis object for detecting APT attacks, an apparent drawback is the need to confront the dependency explosion problem. In particular, APT attacks are characterized by their persistence, as the malicious activities of threat actors often last for a prolonged period of time. As a result, when the time window expands, the number of event streams (also referred to as paths) grows rapidly. For instance, the prior work [9] proposes a rareness-based path selection method to filter paths that are unlikely to be malicious in advance.

Alternatively, system events can also be regarded as informative features for describing system behaviors from a fine-grained view. Especially, modern auditing tools are capable of providing rich properties for an individual system event. For example, additional information such as network type (e.g., IPv4), client IP, server IP, and server port can be supplemented for network events. Therefore, it should be noted that **we focus on system events** in this work. This brings two obvious advantages. ❶ The dependency explosion problem can be avoided. ❷ It mitigates the impact of noise. In contrast, a path may contain substantial noise, since attackers tend to conceal their malicious activities within a multitude of legitimate operations [10].

[1]We share our resources with the research community in the following link: https://osf.io/wfyp7/?view_only=d130fdf059274f21acb85d000d1bf67e

III. Methodology

The primary focus of this work is to explore the potential of LLMs in enhancing provenance analysis. We are particularly interested in how the extensive knowledge of LLMs can be transferred to the field of APT detection and play an active role. Therefore, we do not intend to develop sophisticated APT detection techniques. Instead, we follow the standard pipeline of AI-based applications to implement a simple and straightforward prototype. Furthermore, at each stage of the APT detection process, we fully leverage the capabilities provided by LLMs in order to explore their potential. Later, our evaluation comprehensively examines the performance of the LLM across all stages of the pipeline.

Our pipeline begins by preparing the input for the LLM. This step is the only stage in the entire pipeline that requires human intervention. However, it is a one-time effort. Specifically, we intuitively define a set of features to describe system events based on their types, using heuristic rules. Next, the LLM's extensive knowledge is leveraged to perform event interpretation, where augmented semantics for system events are generated with proper prompt engineering. Subsequently, we take advantage of the LLM's contextualized embedding capability to generate embeddings for such events. Finally, the resulting representations are used to train a machine learning model for threat detection.

Particularly, we employ OpenAI API to access and integrate the capabilities of GPT-4o into our applications. GPT-4o is a state-of-the-art multimodal LLM released by OpenAI in May 2024, ensuring its responses are informed by the most recent and relevant information available. Through the API, we can automatically interact with GPT-4o to perform various tasks, such as generating text in batches.

A. System behavior interpretation

As mentioned earlier, we use system events to represent the behavior of a system. Specifically, we categorize system events into three types: process events, file events, and network events. In general, for all event types, some common properties, such as system call name, user name, user shell, and process name, are retained. Beyond these, based on heuristics, we selectively preserve additional properties for different event types to serve as supplementary features. For process events, system calls related to process creation (e.g., `fork`, `clone`, `execve`) and termination (e.g., `kill`) are involved. In particular, the program image inside a process is replaced when a new process is created. Therefore, system call arguments that indicate the specific content being executed need to be preserved. This information is especially important for interpreter processes such as `Python` and `Bash`, since the executed scripts are often more meaningful than the interpreter itself. For file events, the file descriptor name needs to be

preserved, particularly for system files with special purposes. For example, in Linux, reading the `/etc/passwd` file is primarily intended to obtain user account information. For network events, attributes such as network type, client IP, server IP, and server port are preserved. Finally, the details of a system event are encapsulated into a JSON object, where each key-value pair describes a certain property.

Listing 1 shows an example of a system event which is made by the `dhclient` process receiving data from a network socket through the `recvfrom` system call. Since this is a network event, the fields such as network type, client IP, server IP, and server port are preserved, but the `fd_filename` field is not used as shown by its value `<NA>`. This system call is executed by the `root` user whose shell is `/bin/bash`.

```
{
    'proc_name': 'dhclient',
    'type': 'recvfrom',
    'fd_filename': '<NA>',
    'user_name': 'root',
    'user_shell': '/bin/bash',
    'fd_type': 'ipv4',
    'fd_cip': 'IP2',
    'fd_sip': 'IP1',
    'fd_sport': '67'
}
```

Listing 1: A JSON example for a network event.

Moreover, we observe that some property values of system events may contain randomly generated data or temporary files. Such noise does not contribute to the meaningful interpretation of system events. Therefore, we employ the normalization algorithm proposed in [11] to mitigate such noise.

The JSON objects obtained through the aforementioned processing are fed to the LLM service. Through prompt engineering, we instruct the LLM to interpret the system events and produce the final summary. The textual explanations generated in this manner are then further used as input for subsequent embedding generation. In Section IV-B, we provide several concrete examples illustrating the responses produced by GPT-4o of OpenAI. More importantly, experts with extensive system-level knowledge manually evaluated GPT-4o's ability to understand system provenance data, thereby validating the potential of leveraging LLMs to assist provenance-based APT detection.

B. Contextualized Embeddings

Provenance data is recorded in an unstructured form, such as file names or executable paths, which are textual in nature. To facilitate subsequent machine learning tasks, embedding methods are required to learn the numerical representations for the unstructured data. Multiple embedding techniques have been widely used in various cybersecurity tasks [12], [13]. Currently, mainstream embedding methods can be categorized into two main types: static embeddings and contextualized embeddings. The earlier work [9], [11] adopted static embedding methods to generate numerical representations for the system events. Since 2024, OpenAI has launched two powerful embedding models, namely `text-embedding-3-small` and `text-embedding-3-large` [14]. In particular, they use the state-of-the-art contextualized embeddings method. Contextualized embeddings are text representations whose vectors change depending on the surrounding context in a corpus. Compared with static embeddings such as `Doc2Vec`, they capture different meanings of the same token in different contexts, leading to richer and more accurate semantic representations.

It is worth highlighting again that the focus of this work is to explore the capability of LLMs in enhancing provenance-based APT detection. Hence, throughout the pipeline, we leverage the capabilities of LLMs wherever possible. When interacting with GPT-4o via the API, we choose `text-embedding-3-small` as the text embedding approach. Compared to its more powerful counterpart `text-embedding-3-large`, the selected version is smaller but still highly efficient. More specifically, we chose 1,536 as the embedding dimension for the output of the `text-embedding-3-small` model. According to the evaluation results released by OpenAI [14], its performance is comparable to that of `text-embedding-3-large`. Next, following the previous work [9], we employ Principal Component Analysis (PCA) with an RBF (Radial Basis Function) kernel [15] to compress the vectors to 256 dimensions. Although dimensionality reduction inevitably leads to some loss of information, it accelerates subsequent AI model training and facilitates a fair comparison with prior methods during the evaluation stage.

C. Malicious event identification

We consider two categories of classic models as our threat detectors: supervised learning methods and semi-supervised anomaly detection methods. For the supervised learning, two representative algorithms are included. The first one is multilayer perceptron (`MLP`), which is a neural network model. The second detector is gradient-boosted decision trees (`GBDT`), which is a representative ensemble learning model. For the semi-supervised learning, we train an anomaly detector using `XGBOD` [16] algorithm. In practice, obtaining a fully labeled dataset is inherently challenging and usually expensive. However, semi-supervised learning uses a combination of labeled and unlabeled data to construct AI models. Specifically, semi-supervised outlier detection can be performed even when the training data consists only of observations describing normal behavior. We can then predict whether unknown events are caused by an attack by evaluating their deviation from known events. After all, in reality, collecting normal system events is often much easier than acquiring malicious system events.

IV. Evaluation

We are particularly interested in exploring the interpretability of descriptive tests from GPT-4o regarding system events, as well as the ability of OpenAI's contextualized embedding model to capture the semantics of system events, and their effectiveness in APT detection. Specifically, our evaluation is intended to seek a deeper understanding of the following research questions (**RQs**):

- **RQ1:** Whether the extensive knowledge and semantic understanding capabilities of LLMs can be transferred to produce meaningful interpretations of system events in assisting provenance analysis? (**§IV-B**)
- **RQ2:** Does the contextualized embedding capability of LLMs provide a significant advantage in anomaly detection compared to previous methods? (**§IV-C** and **§IV-D**))
- **RQ3:** How is the generalization capability of our proposed APT detection technique when confronted with a previously unseen attack? (**§IV-E**)

A. Dataset

We perform empirical investigations using the publicly accessible `ProvSec` [17] dataset. `ProvSec` is a benchmark dataset designed for cybersecurity system provenance analysis. This dataset contains fine-grained data, including full details of system calls (with parameters) and other fields that are useful for describing events. `ProvSec` includes different vulnerable cases, each with paired benign and attack executions that are carefully labeled, providing a realistic and comprehensive resource for advancing security-oriented AI and detailed forensic studies. The training set contains 7,605 samples, denoted as $\mathcal{D}_{Train}$, while the test set comprises 1,902 samples (accounting for 20% of the total), denoted as $\mathcal{D}_{Test}$. The adversary and benign events are roughly balanced in both the training and test sets.

B. Explanations Generated by LLM

We randomly sampled 500 interpretations of system events generated by the LLM and manually examined their correctness. All participants possessed expertise in system-level provenance. No significant errors were observed in the evaluations. During our exploration of LLM explanations of system call events, we identified that there are multiple notable details that are beneficial for understanding the events, especially for cybersecurity purposes. We list several concrete examples as demonstrations.

1) **Knowledge of System Calls:** First, in LLM's explanation, a simple system call name is expanded to a full sentence of what it indicates by explaining what is the system call for. (blue annotation)
2) **Knowledge of Software Identity:** Second, for commonly used software, LLM is aware of what software it is. It helps the understanding of its existence or expected behavior if the user lacks knowledge. (violet annotation)
3) **High-Level Knowledge of Application Execution Context:** Based on the prior two types of knowledge, LLM may suggest what is the current behavior doing in a higher-level description that is easier to understand to human. (yellow annotation)
4) **Comment on Possible Suspiciousness:** Lastly, LLM sometimes comments on possible usages of the current behavior and potential suspicious cases that is worth for the attention for possible security threats. This is a useful extra knowledge extracted from the LLM. (pink annotation)

In the following examples, we explain the LLM's knowledge with the four types of details aforementioned.

Example 1 shows the explanatory text generated by the LLM for the system event "`dhclient recvfrom root /bin/bash <NA> IP2 IP1 67`". It is worth noting that although we adopt the JSON format to describe a system event when interacting with LLMs, herein, for the sake of convenience and brevity, we use a textual format to represent a system event. Unless otherwise specified, all subsequent examples in this paper will follow this approach.

Example 1

This event is a network activity event involving the process `dhclient`, which is a DHCP client typically used to obtain IP addresses and network configuration from a DHCP server. The event type is `recvfrom`, indicating that the process received data from a network socket. The data was received over an IP v4 connection from the source IP address `IP1`, which is likely the DHCP server, using the source port 67, the standard port for DHCP server communications. The destination IP address is `IP2`, which appears to be a broadcast address. The process is running under the user `root` with the shell `/bin/bash`. This event suggests that the DHCP client is receiving network configuration information from the DHCP server.

Manual Evaluation: The first example is about a `dhclient` process performing a `recvfrom` system call event. LLM first explains the identity of this software correctly as a DHCP client annotated with violet color. It adds the explanation of what this system call indicates (blue annotation). Then the LLM explains a high-level application context of this event as the DHCP client receiving network configuration information from the DHCP server (yellow annotation).

Given the system event "`screen openat ubuntu /bin/bash LC_TIME`", Example 2 shows the corresponding explanatory text generated by the LLM.

Example 2

This event is an `openat` system call where the process named `screen` is attempting to open a file or directory named `LC_TIME`. The action is performed by a user named `ubuntu` who is operating within a `Bash` shell environment. This type of event typically indicates a process accessing or modifying locale settings related to time, possibly for configuring or retrieving time-related information.

Manual Evaluation: The second example is about the `openat` system call event made by the `screen` software. LLM explains that this system call is regarding opening a file or directory correctly (blue annotation). Then it adds a high-level description that it is typically about accessing or modifying locale settings related to time, which is inferred by

the accessed file, LC_TIME. (yellow annotation)

The last system event used as a sample is "echo mprotect www-data /usr/sbin/nologin <NA>". The corresponding explanatory text generated by the LLM is shown in Example 3.

Example 3

This event is of type mprotect and involves the process echo being executed by the user www-data, who has a restricted shell set to /usr/sbin/nologin. The mprotect system call is typically used to change the memory protection of a region of memory, which can be indicative of a process attempting to modify its memory permissions. The absence of a specific file descriptor suggests that this operation is not directly associated with a file. This event could be part of a legitimate operation or potentially suspicious activity, depending on the context, as www-data is a common user for web services, and the restricted shell indicates limited direct user interaction.

Manual Evaluation: The last example is about the echo process executing the mprotect system call. LLM first explains the meaning of this system call changing the memory protection of a region of memory. We found a more useful description is about LLM's comment on this event's usage. LLM mentions this event could be part of a legitimate operation or potentially suspicious given a web-service user with a restricted shell. Indeed, this echo process was caused by a security exploit from a malicious behavior dataset and LLM correctly identified its potential risk.

C. Comparison

We compare the representation learning adopted in our approach with the embedding method used in the previous paper [9], and examine their impact on the final threat detection. More concretely, the PV-DM model of Doc2Vec was employed in [9]. When using PV-DM to generate embeddings for system events, we consider two different scenarios. The first scenario assumes that we have exhaustively collected as many words as possible for the vocabulary, so there is no out-of-vocabulary (OOV) issue. Surely, this is difficult to achieve in practice. Therefore, the second, more realistic scenario is that we only learn an embedding model from $\mathcal{D}_{Train}$. As such, during the testing phase, there may be words that have never been encountered before.

As mentioned earlier, for the threat detector, we consider two different supervised learning methods here: MLP and GBDT. For MLP, we conduct training for 50 rounds and choose the model that achieves the highest accuracy score. For GBDT, we use grid search to find the parameters that yield the best accuracy score. As shown in Figure 2, regardless of which machine learning model was used as the detector, our proposed method achieves significantly better performance compared to the baselines. When using MLP and GBDT as detectors, the accuracy reaches 99.1% and 98.3%, respectively. Moreover, when the OOV problem is present, the performance of detectors trained on feature vectors generated by Doc2Vec further deteriorates. It is worth noting that the size of our testing set is relatively small, containing only 1,902 samples. In real-world scenarios, due to the diversity of unstructured data such as file names or executable paths, the OOV problem is likely to be even more severe. The advantages of large language models would become even more apparent.

TABLE I: APT Detection Performance

Model	Accuracy	Precision	Recall	F_1-Score
MLP	99.1%	99.0%	99.0%	99.0%
GBDT	98.3%	97.2%	99.1%	98.1%
XGBOD	96.1%	96.9%	94.7%	95.8%

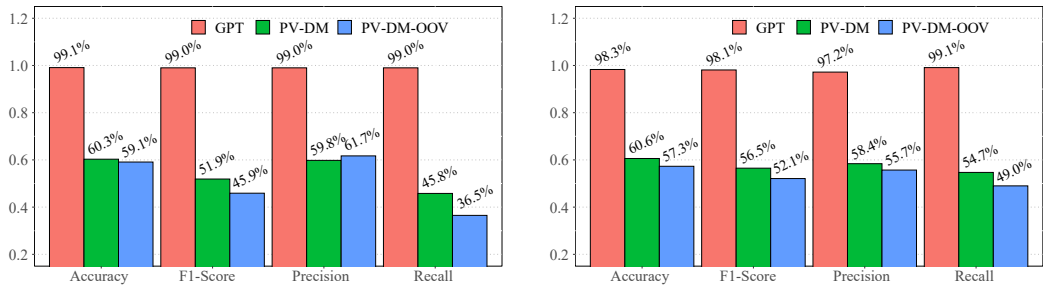
D. Semi-supervised Detection Performance

In the previous section, we have witnessed the classification performance of two supervised models, i.e., MLP and GBDT. In the supervised learning setting, our method effectively distinguishes adversary events from normal events. Now, we consider a more challenging but also more practical scenario—verifying whether our proposed method can still be effective under a semi-supervised learning setting. This is aligned with real-world application scenarios, where fully labeled data is hard to obtain, and AI applications are built based on a combination of labeled and unlabeled data.

In detail, we adopt XGBOD [16] to build a threat detection model, which is a semi-supervised outlier detection algorithm. Specifically, our anomaly detection was implemented based on PyOD [18], an open-source Python toolbox for performing scalable outlier identification. We compared the performance of this model with the two previous supervised learning methods, as shown in Table I. Not surprisingly, the performance of semi-supervised learning is slightly worse than that of supervised learning. However, the XGBOD-based APT detection model still achieved high precision of 96.9%. Thus, it can be concluded that the extensive knowledge of LLMs regarding system events can be effectively transferred to the application of APT detection.

E. Case Study: Detecting Unseen Attacks

We perform a case study in wild to inspect whether the proposed technique can be generalized over previously unseen attacks and make accurate predictions. In detail, we consider the unseen attack through exploiting CVE-2021-44228. This vulnerability was discovered in the Log4j logging library, which has severe and widespread impacts. Malicious actors can use the Log4j flaw to run almost any code they want on vulnerable systems. In detail, we randomly extract 500 adversary events from the Log4j attack to construct a new test dataset. We also include 500 benign events in this new testing set because the ROC (receiver operating characteristic) curve is a plot of the true positive rate (TPR) against the false positive rate (FPR) at varying threshold settings. It should be noted that the training set contains no system events from the



(a) MLP-based detector

(b) GBDT-based detector

Fig. 2: Comparison of the final threat detection performance.

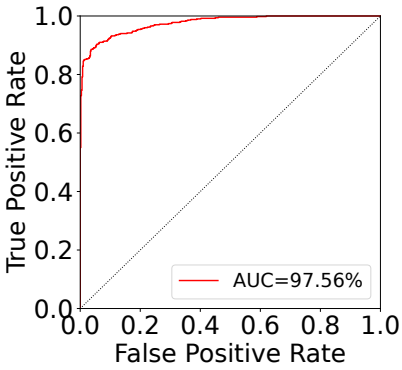


Fig. 3: The ROC Curve.

`Log4j` attack to make the evaluation results convincing. Still, we adopt the semi-supervised algorithm `XGBOD` [16] to detect adversary events launched by the `Log4j` attack. We plot a ROC curve to graphically illustrate the performance of this APT detector. The ROC obtained on the testing set is shown in Figure 3, with an AUC value of 97.56%. The proposed technique thus demonstrates strong adaptability especially when faced with attacks that were never encountered before.

V. Related Work

Using provenance in APT detection and investigation has been explored by a large body of work [10], [19]. Multiple attack detection approaches were proposed based on anomaly scores. For example, the threat detection system `PIDAS` [20] computes the anomaly score of a certain length of a path in a provenance graph and compares it with a predefined threshold. A subsequent work, `Pagoda` [21], considers the anomaly score of both a single provenance path and the whole graph. Moreover, `NoDoze` [22] uses a network diffusion algorithm that propagates anomaly scores across dependency graphs to calculate anomaly scores. Tag propagation is another widely adopted strategy. `SLEUTH` [23] introduced two types of tags, namely, trustworthiness tags and confidentiality tags, in the attack detection system. In a nutshell, an alarm is triggered when a node with low trustworthiness accesses a node with high confidentiality. The later work [24] improved the original tag-based system and thus reduced false alarms significantly in the detection of APT-style attacks. However, tag propagation-based approaches suffer from the "dependency explosion" problem. To address this issue, `Holmes` [25] prioritizes broad detection using relatively simple signatures to catch a wide range of malicious activity. It then uses alert filtering to reduce false alarms. But `Holmes` involves empirical parameters, which may lead to unstable detection results.

VI. Conclusion

This paper explores the potential of LLMs for provenance-based APT detection. We found the state-of-the-art LLM offers multiple enhancements over provenance event details. Specifically, our analysis summarizes LLM can offer knowledge on system calls, knowledge on software identity, high-level knowledge on application execution context, and comments on possible suspiciousness beyond a brief description of a system call. Such new details empower the semantic details thus improving the performance of APT detection. Our experiment further shows that the LLM's contextualized embedding capability effectivity improves the detection performance over the representative method adopted by previous work in supervised learning-based detectors. Not only that, even in the semi-supervised learning scenario, our anomaly detection can still achieve a precision as high as 96.9%. The findings suggest that LLMs contribute effectively to provenance-based APT detection and hold promising potential for future applications.

Acknowledgments

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525. This article describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the article do not necessarily represent the views of the U.S. Department of Energy or the United States Government. This work was supported through contract CR-100043-23-51577 with the U.S. Department of Energy.

REFERENCES

[1] Federal Trade Commission, "Equifax data breach settlement," https://www.ftc.gov/enforcement/refunds/equifax-data-breach-settlement, 2024, accessed: 2024-12.

[2] CrowdStrike, "Global threat report," https://go.crowdstrike.com/global-threat-report-2024.html, 2024, accessed: 2024-12.

[3] IBM Security, "Cost of a data breach report," https://www.ibm.com/reports/data-breach, 2022, accessed: 2023-05.

[4] J. Fox, "Top 40 AI cybersecurity statistics," https://www.cobalt.io/blog/top-40-ai-cybersecurity-statistics, 2024, accessed: 2024-12.

[5] Sysdig Inc., "Sysdig monitor," https://docs.sysdig.com/en/docs/sysdig-monitor/, 2024, accessed: 2024-12.

[6] P. Yan, S. Tan, M. Wang, and J. Huang, "Prompt engineering-assisted malware dynamic analysis using GPT-4," *IEEE Transactions on Dependable and Secure Computing*, vol. 22, no. 6, pp. 7712–7728, 2025.

[7] X. Qu, F. Zuo, X. Li, and J. Rhee, "Context matters: Investigating its impact on ChatGPT's bug fixing performance," in *IEEE/ACIS 22nd International Conference on Software Engineering Research, Management and Applications (SERA)*, 2024.

[8] C. Zhang, H. Liu, J. Zeng, K. Yang, Y. Li, and H. Li, "Prompt-enhanced software vulnerability detection using ChatGPT," in *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings*, 2024, pp. 276–277.

[9] Q. Wang, W. U. Hassan, D. Li, K. Jee, X. Yu, K. Zou, J. Rhee, Z. Chen, W. Cheng, C. A. Gunter *et al.*, "You are what you do: Hunting stealthy malware via data provenance analysis." in *Network and Distributed System Security Symposium (NDSS)*, 2020.

[10] F. Zuo, J. Rhee, S. Lu, Y. Song, and X. Zhang, "An empirical study on the multi-stage nature of apt attacks in cloud computing," in *IEEE 22nd International Conference on Mobile Ad-Hoc and Smart Systems (MASS)*, 2025, pp. 585–591.

[11] F. Zuo, J. Rhee, Y. R. Choe, C. Fu, and X. Qu, "Few-shot learning-based cyber incident detection with augmented context intelligence," in *IEEE 49th Annual Computers, Software, and Applications Conference (COMPSAC)*, 2025, pp. 961–969.

[12] F. Zuo, X. Zhang, Y. Song, J. Rhee, and J. Fu, "Commit message can help: Security patch detection in open source software via Transformer," in *IEEE/ACIS 21st International Conference on Software Engineering Research, Management and Applications (SERA)*, 2023, pp. 345–351.

[13] F. Zuo, C. Tompkins, Q. Zeng, L. Luo, Y. R. Choe, and J. Rhee, "BinSimDB: Benchmark dataset construction for fine-grained binary code similarity analysis," in *International Conference on Security and Privacy in Communication Systems*, 2024, pp. 203–225.

[14] OpenAI, "New embedding models and API updates," https://openai.com/index/new-embedding-models-and-api-updates/, 2024, accessed: 2024-2.

[15] F. Zuo and B. Bu, *Machine Learning: Principles and Practice (Python Edition)*. Tsinghua University Press, 2021.

[16] Y. Zhao and M. K. Hryniewicki, "XGBOD: Improving supervised outlier detection with unsupervised representation learning," in *2018 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2018, pp. 1–8.

[17] M. Shrestha, Y. Kim, J. Oh, J. Rhee, Y. R. Choe, F. Zuo, M. Park, and G. Qian, "Provsec: Open cybersecurity system provenance analysis benchmark dataset with labels," *International Journal of Networked and Distributed Computing*, vol. 11, no. 2, pp. 112–123, 2023.

[18] Y. Zhao, Z. Nasrullah, and Z. Li, "PyOD: A python toolbox for scalable outlier detection," *Journal of machine learning research*, vol. 20, no. 96, pp. 1–7, 2019.

[19] Z. Li, Q. A. Chen, R. Yang, Y. Chen, and W. Ruan, "Threat detection and investigation with system-level provenance graphs: A survey," *Computers & Security*, vol. 106, p. 102282, 2021.

[20] Y. Xie, D. Feng, Z. Tan, and J. Zhou, "Unifying intrusion detection and forensic analysis via provenance awareness," *Future Generation Computer Systems*, vol. 61, pp. 26–36, 2016.

[21] Y. Xie, D. Feng, Y. Hu, Y. Li, S. Sample, and D. Long, "Pagoda: A hybrid approach to enable efficient real-time provenance based intrusion detection in big data environments," *IEEE Transactions on Dependable and Secure Computing*, vol. 17, no. 6, pp. 1283–1296, 2018.

[22] W. U. Hassan, S. Guo, D. Li, Z. Chen, K. Jee, Z. Li, and A. Bates, "NoDoze: Combatting threat alert fatigue with automated provenance triage," in *Network and Distributed Systems Security Symposium*, 2019.

[23] M. N. Hossain, S. M. Milajerdi, J. Wang, B. Eshete, R. Gjomemo, R. Sekar, S. Stoller, and V. Venkatakrishnan, "SLEUTH: Real-time attack scenario reconstruction from COTS audit data," in *26th USENIX Security Symposium*, 2017, pp. 487–504.

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

[25] S. M. Milajerdi, R. Gjomemo, B. Eshete, R. Sekar, and V. Venkatakrishnan, "Holmes: real-time apt detection through correlation of suspicious information flows," in *IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 1137–1152.