SAND2025-03552C

arXiv:2503.18316v2 [cs.CR] 25 Mar 2025

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

Fei Zuo
fzuo@uco.edu
University of Central Oklahoma
Edmond, Oklahoma, USA

Junghwan Rhee
jrhee2@uco.edu
University of Central Oklahoma
Edmond, Oklahoma, USA

Yung Ryn Choe
yrchoe@sandia.gov
Sandia National Laboratories
Livermore, California, USA

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 threat 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 comprehensively examine the quality of the resulting embeddings, and it turns out that they offer promising avenues. Subsequently, machine learning models built upon these embeddings demonstrated outstanding performance on real-world data. In our evaluation, supervised threat detection achieves a precision of 99.0%, and semi-supervised anomaly detection attains a precision of 96.9%.

CCS Concepts

• **Security and privacy** → **Intrusion detection systems**; • **Computing methodologies** → *Machine learning*.

Keywords

Intrusion Detection, APT Detection, Provenance Analysis, GPT, Large Language Models

1 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 [3]. According to CrowdStrike's report [4], "*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 [26]. The recent survey [5] 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-powered preventive strategy.

```
The text editor vim performed a read operation to
access the system file /etc/localtime, which is
typically used to retrieve timezone configuration
information.
```

Listing 1: The explanation of a system event (expressed using a triplet "`vim read /etc/localtime`") provided by GPT-4.

Existing AI-powered threat detection techniques have either approached provenance analysis from a graph perspective or from a natural language perspective [2, 29]. In this work, we follow the latter approach, considering the system entities and the interaction between them as various components of a sentence. But unlike previous work, we will take advantage of the recent advancements in large language models (LLMs) to enhance provenance analysis. This approach offers several obvious benefits. First of all, **the extensive knowledge possessed by LLMs can be leveraged to augment the semantics of system event descriptions**. For example, given a simple system event described by the triplet "`vim read /etc/localtime`", the descriptive text generated by GPT-4 through being guided by appropriate prompt engineering is shown in Listing 1. Compared to the original input triplet, GPT-4 accurately supplemented the information that "`vim` is a text editor". Moreover, based on the characteristics of the second system entity, it inferred that the purpose of this file reading behavior was to obtain timezone configuration information. Previous research [21] has shown that understanding such information is highly useful when analyzing malicious behavior by attackers. This is because some adversaries deliberately exclude certain areas from their attacks. For instance, a family of immensely dangerous and destructive crypto-ransomware are used to first checking the infected machine's UI language before launching the attack. If the language is Russian, Ukrainian, or another language from former Soviet Union countries, the attack will not be triggered. However, without the additional information

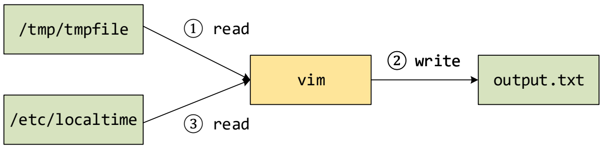


Figure 1: A simplified provenance graph example.

offered by LLMs, similar tasks would typically require significant human effort and expertise. More examples will be provided later in Section 3.3.

It is worth noting that the descriptive texts used to interpret system events are unstructured data. Therefore, to facilitate subsequent learning by artificial intelligence models, we also need to use appropriate embedding models to learn numerical representations of data. This highlights another benefit of introducing LLMs: **their powerful embedding models can accurately capture and retain more semantic information**. In the natural language processing domain, different embedding techniques have been proposed for words [19], sentences [23], and documents [12]. The prior work [29] used `Doc2Vec` [12] to generate embeddings for a series of selected system events. However, when using traditional models like `Word2Vec` [19] or `Doc2Vec` [12], the out-of-vocabulary (OOV) issue will be an unavoidable challenge. For provenance analysis in the wild, it is unrealistic to exhaustively collect words such as file names or executable paths. A usual approach is to ignore words that have never appeared in the vocabulary when generating numerical representations of sentences. However, this leads to information loss. In our proposed method, we offer two targeted solutions. First, for system entities with indicators that have special significance, we make full use of the extensive knowledge of LLMs to interpret them. Second, for entities like system-generated temporary files or hash values, we have designed a heuristic approach to preprocess them. We will show more details in Section 3.

Though we have witnessed a number of meaningful attempts to integrate LLMs into various cybersecurity applications [24, 32, 34], the extent to which LLMs can assist in provenance analysis remains an open question. In this work, we aim to gain a deeper understanding of the potential of LLMs in enhancing provenance analysis. To this end, we developed a proof-of-concept prototype which fully takes advantage of the extensive knowledge of LLMs. Our approach leverages LLMs to augment the semantics of system events' descriptions, and further generate high-quality embeddings. In the later threat 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. 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.

The main contributions of our work are as follows:

- We have explored a systematic strategy for applying the extensive knowledge of LLMs in APT detection through provenance analysis. Our innovative approach leverages LLMs to augment system event descriptions and further utilizes the assistance of LLMs for high-quality representation learning.
- We conducted comprehensive studies on the performance of the advanced contextualized embedding approach in learning the semantics of system event descriptive texts. Subsequently, both the supervised and unsupervised detection methods built based on this achieved high precision.
- Our study utilizes a dataset collected from real-world cyberattacks. Based on this, we showcased the capability of LLMs in improving provenance analysis through extensive experiments. The research resources are available upon request to facilitate future related studies.

The remainder of this paper is organized as follows: First, to help readers better understand this article, we briefly introduce the necessary preliminary knowledge in Section 2. Next, Section 3 describes our system architecture and design in detail. Then, we conduct evaluations and show the results in Section 4. The related work is then reviewed in Section 5. The limitations of this work and potential future directions are discussed in Section 6. Finally, we draw conclusions in Section 7.

2 Background

In this section, we briefly introduce the necessary background on system-level provenance analysis and large language models.

2.1 Provenance Data

Provenance data records complex dependencies across various system events, thus reflecting the interactions between entities within a historical context. System-level provenance is usually presented in the form of graphs, where each node represents a system entity (e.g., a process or a file); edges are system call and timestamp labels related to the nodes. A system call is the programmatic way in which a user app can request a service from an OS kernel.

Figure 1 shows a provenance graph example, which includes four system entities represented by the four nodes in the graph. An entity refers to a component 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, for instance, `read` and `write` in Figure 1. It should be noted that system provenance graphs in reality are very complex and extensive; herein, we only use a simplified example for demonstration purposes. Provenance data is often represented as a graph because graphs can intuitively depict dependencies between events or their chronological order. However, considering that hackers launch multi-stage attacks, extracting attack-relevant events from a vast number of system events over a long time span and linking them is non-trivial. As a result, the generated provenance graph often contains significant noise, posing challenges for attack detection. Therefore, it should be noted that **the proposed approach focuses on system events** in provenance data.

The two interrelated system entities, along with the operation between them, collectively constitute an event. For example, we can observe three system events in Figure 1. In temporal order, they are (1) the text editor `vim` reads a temporary file; (2) the process `vim` saves its content to a file named `output.txt`; and (3) `vim` checks the system's time zone by reading the `/etc/localtime` file. A system event is a basic unit in provenance data for tracking and recording system-level behaviors. Hence, system events extracted from a

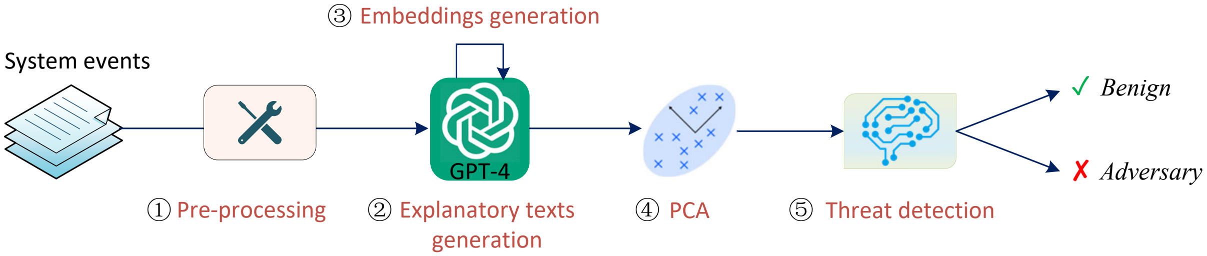


Figure 2: System overview.

provenance data are regarded as informative features for describing a cyber incident in intrusion detection.

2.2 Large Language Models

In recent years, the emergence of large language models (LLMs) is undoubtedly one of the most impactful breakthroughs of AI. A large language model, as the name suggests, is a language model trained on massive amounts of text data and has billions of parameters. As a widely recognized instance of an LLM, the Generative Pre-trained Transformer (GPT) has attracted widespread attention worldwide since its emergence in 2018 [25]. As an implementation of GPT, OpenAI launched ChatGPT[1] to the public in 2022, which demonstrates a powerful ability to generate human-like responses in conversational interactions. Microsoft's Copilot[2] is another representative commercial product of generative AI, which adopts GPT as the core model. Another main competitor in the same field, Gemini[3] was launched by Google in December 2023.

In this work, we chose GPT as our primary LLM because OpenAI has also released a corresponding API for external users to access the advanced LLMs developed by OpenAI. Applying the API offered by OpenAI in applications has three main benefits. First, developers are allowed to interact with a vast knowledge base provided by OpenAI's state-of-the-art AI models in a programmatical manner. Second, it provides a straightforward way to leverage the cutting-edge capabilities in NLP, thus enabling tasks like text generation, interpretation, and summarization. Lastly, other functionalities provided by OpenAI, such as fine-tuning, facilitate users to optimize their application's performance on customized tasks.

2.3 Unstructured Data 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 proposed in the field of natural language processing. In some previous security-related applications [37, 38], the practice of embedding text has played a crucial role. Currently, mainstream embedding methods can be categorized into two main types: static embeddings and contextualized embeddings. When using static embeddings, the same word always has the same representation, even when it appears in different contexts. For example, the word "mouse" can refer to either a computer hardware or an animal, but regardless of the context, it only has one vector representation. In contrast, contextualized embeddings generate dynamic, context-dependent vector representations for text, meaning that the representation changes depending on the surrounding context.

The strength of analogical reasoning ability is an important indicator of whether text embeddings are meaningful. An analogical question typically consists of two pairs of texts, such as ("man", "king") and ("woman", "queen"). To assess how related the two pairs are, the analogy "man is to king as woman is to queen" is formed, and its validity is tested. In the feature space, the analogical question can be represented as $|E_1 - E_2| \approx |E_3 - E_4|$, where E_1, E_2, E_3, and E_4 are the embedding vectors, and $|\cdot|$ represents the norm. This means that the two pairs of texts being examined can form an approximately parallelogram-like relationship in the feature space. Applying this concept to our problem, we can find similar analogies between one pair of system event descriptions and others. If these analogies align with our prior knowledge expected in provenance analysis, we can confirm that the embeddings generated by LLMs are semantically meaningful.

3 Methodology

In this study, 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 created by OpenAI, which has been trained on data up to October 2023, 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.

Figure 2 illustrates the overall pipeline of our system. Given the provenance system data, we first pre-process the raw data according to the specific characteristics of different types of system events. Second, using the LLM service, explanatory texts with augmented semantics for system events are generated with proper prompt engineering. Third, the text embeddings are further produced by taking advantage of the facilities of the LLM. Next, we employ some manifold learning method such as Kernel PCA to adjust the dimensionality of the original embeddings. Lastly, the threat is detected by a well-trained machine-learning model.

3.1 Pre-processing

Input data: We use a system call event whose several properties are extracted such as the system call name, user name, user shell,

---

[1] https://chatgpt.com/
[2] https://copilot.microsoft.com/
[3] https://gemini.google.com/

process name for all events. The file descriptor name is included for file events. For process creation events such as `clone` and `execve`, system call arguments are included because they include specific information about what program is executed. This information is essential for interpreter programs such as Python and Bash because the main program names do not show the script names. For network events, network type (IPv4), client IP, server IP, and server port are used. These fields are provided as a JSON object to the LLM service.

Path normalization: Using the raw data sometimes could create unnecessary noise to the LLM as seen in the randomly generated data or timestamps. Therefore, we applied several data normalization methods to avoid such noise. First, temporary files tend to have less meaningful names due to their nature. We utilized several well-known file extensions for the temporary files such as ".tmp" and ".temp" to generalize their names. Second, the files under certain directories such as the `proc` file system can be noisy because they reflect the internal states of the operating system kernel and their file paths may include the process IDs (PIDs) which are dynamically determined. Therefore, such file names are generalized by excluding the PIDs. Lastly, if file names are like hashes based on known patterns (e.g., MD5 and SHA), their names are normalized using a uniform pattern "hash value".

3.2 Explanatory texts generation

The request messages are structured to include two types of content: one for the "system" role and another for the "user" role. The "system" role content outlines general instructions, detailing the input required from the user and the expected output from GPT-4o. Meanwhile, the "user" role content provides the details of a system call as a query.

System role: For the "system" role, we used a prompt *"here is a json object where the keys are sysdig event fields and the values are the data for the corresponding keys. Please generate only the final summary of the meaning of this event and the event type as a paragraph"*. Our implementation of system call events utilizes the `Sysdig` [10] system which happens to be recognized by the LLM service that we use. This is one example of system call formats and other formats can be utilized as well by instructing the LLM on how to understand them accordingly. This prompt describes that we are providing our data in the JSON format and we would like to have a summary of the meaning of this event. Since the LLM may use verbose expressions, we instruct to produce only the final summary as a paragraph.

User role: The user role uses the details of a system call's properties as the key-value pairs in the JSON format.

```
{
    'proc_name': 'crond',
    'type': 'execve',
    'fd_filename': '<NA>',
    'user_name': 'root',
    'user_shell': '/bin/bash',
    'evt_args': 'sh'
}
```

Listing 2: A JSON example for the user role.

Listing 2 shows an example of a system event which is made by the `crond` process executing the `sh` program listed in the `evt_args` system call parameter list field using the `execve` system call. Since this is a process event, 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`.

Given these input data, the OpenAI `gpt-4o` LLM service responded with the following answer: "*This event is an* `execve` *type, indicating the execution of a new program. The process involved is* `crond`*, which is a daemon used to execute scheduled commands. The event was initiated by the* `root` *user, utilizing the* `/bin/bash` *shell. The specific command executed is* `sh`*, which suggests the initiation of a shell session or script execution. This activity is typical for system maintenance or scheduled tasks managed by the* `cron` *service.*"

This text explanation of a system event is used as an input for the text embedding explained next.

3.3 Notable Details in LLM Explanations

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.

3.4 Embeddings and Dimension Adjustment

As we mentioned earlier, `ProvDetector` [29] uses `Doc2Vec` [12] to embed the descriptive text of system events. `Doc2Vec` [12] is a static embedding method that learns paragraph embeddings via the distributed memory and distributed bag of words models. In contrast, OpenAI has launched two powerful embedding models since January 2024, namely `text-embedding-3-small` and `text-embedding-3-large` [22]. In particular, they use the state-of-the-art contextualized embeddings method, which is considered as an augmentation to static embeddings.

Static embeddings assign a single and fixed representation to each word, regardless of its context. However, contextualized embeddings generate more accurate and flexible representations by considering the context in which words appear, thus providing fine-grained semantics understanding and more nuanced representations. In the NLP domain, contextualized embeddings have led to substantial performance gains on a variety of tasks compared to static embeddings. The embedding models of OpenAI enable us to generate numerical sequences that accurately capture the semantics of natural language, laying a solid foundation for subsequent machine learning tasks.

OpenAI's embedding models can output feature vectors with dimensions as high as 1,536 or 3,072. Generally, higher-dimensional feature vectors can represent more detailed and complex information. However, high-dimensional feature vectors also pose challenges for subsequent machine learning model training. A common approach is to reduce the dimensionality of the feature vectors before the next stage of machine learning model training. Dimensionality reduction inevitably leads to some loss of information, so a trade-off must be made between time cost and information loss. In this work, we use the Kernel Principal Component Analysis (PCA) with an RBF (Radial Basis Function) kernel to compress the high-dimensional feature vectors generated by GPT to 256 dimensions, which is comparable to the setting in the previous study [29].

3.5 Threat detection

It is necessary to emphasize that the primary focus of this work is to explore the potential of LLMs in improving 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 and complex threat detection systems based on architectures like graph neural networks or Transformers. Moreover, we aim to fully leverage the assistance of LLMs to reduce human effort, such as feature engineering. Consequently, 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

Table 1: Attack scenarios involved in `ProvSec` [27]

ID	Software	Vulnerability	Description
#01	Consul	N/A	Consul service APIs misconfiguration, RCE and reverse shell
#02	Nginx	CVE-2017-7529	Web server remote integer overflow vulnerability
#03	Ghostscript	CVE-2018-16509	Python remote shell command execution via ghostscript
#04	PHP	CVE-2018-19518	PHP IMAP remote command execution (RCE) vulnerability
#05	Docker	CVE-2019-5736	Escape from a Docker container: vulnerability on Docker
#06	Tomcat	CVE-2020-1938	Apache Tomcat arbitrary file read / include vulnerability
#07	Redis	CVE-2022-0543	Redis Lua sandbox escape and remote code execution
#08	Django	CVE-2021-35042	Django allows QuerySet.order_by SQL injection vulnerability
#09	Apache	CVE-2021-42013	Path traversal and file disclosure vulnerability in HTTP server
#10	Apache	CVE-2021-41773	Apache web server path traversal and file disclosure vulnerability
#11	Java Log4j	CVE-2021-44228	Log4j vulnerability allows an affected system to be controlled remotely

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` [35] algorithm. This is more aligned with real-world application scenarios, where fully labeled data is hard to obtain, and artificial intelligence applications are built based on a combination of labeled and unlabeled data.

4 Evaluation

Based on the methodology introduced in Section 3, we conduct a set of experimental studies to evaluate the performance of our proposed technique. 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 embedding model to capture the semantics of system events. and their effectiveness in practical threat detection. We would also like to further investigate whether OpenAI's extensive knowledge can be practically helpful for threat detection. Specifically, our evaluation is intended to seek a deeper understanding of the following research questions (**RQs**).

- **RQ1:** Do the embeddings generated by LLMs provide a significant advantage in anomaly detection compared to previous methods? (**§4.3** and **§4.4**)
- **RQ2:** Can the extensive knowledge of LLMs regarding system events be effectively transferred to the application of APT detection? (**§4.5**)
- **RQ3:** How is the generalization capability of our proposed APT detection technique when confronted with a previously unseen attack? (**§4.6**)

4.1 Dataset

We conduct empirical studies based on a publicly available dataset `ProvSec` [27]. This dataset collects provenance analysis data from eleven attack scenarios in the real world. Information about these eleven attacks is summarized in Table 1. When an attack is launched, the corresponding events are recorded and preliminarily regarded as adversarial. Otherwise, if the attack has not been launched, the events extracted from the corresponding provenance data are regarded as benign. A self-evident intuition is that adversary attacks are usually hidden among massive normal events. Therefore, it is unsurprising that there exist some events that may be on both sides. The intersection between the benign events and the preliminary adversary events is not sufficiently iconic to indicate an attack behavior. To this end, those overlapped patterns from the original adversary events should be removed. Please note that, as described in Section 3, we extracted slightly different fields to depict different types of events. As a result, a total of 4,507 unique adversary events remained. Moreover, in reality, the distribution of adversary events and normal events is imbalanced, with the majority of events being benign. To keep rough balance, we randomly extract 5,000 different benign events. In the dataset, 80% of instances are used for training and the remaining 20% for testing, denoted as $\mathcal{D}_{Train}$ and $\mathcal{D}_{Test}$, respectively.

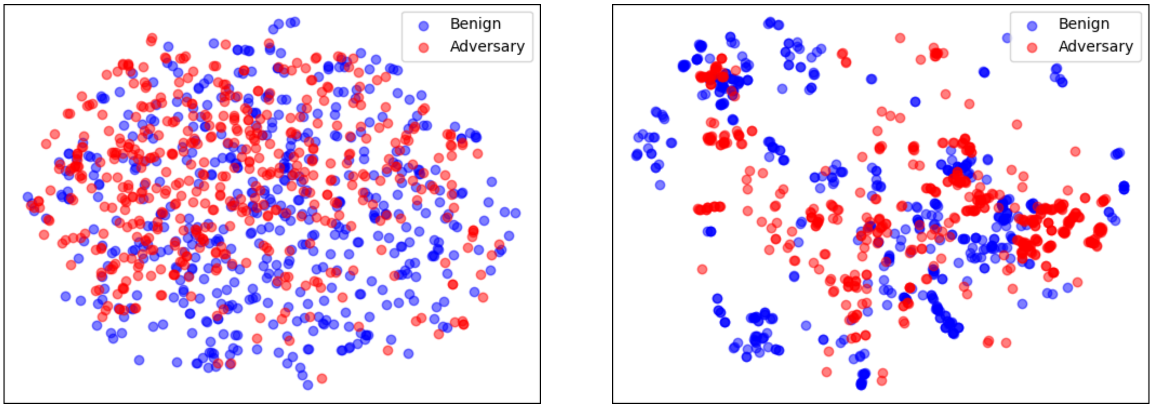
4.2 Environment Settings

When interacting with GPT-4o via the API provided by OpenAI, we select "`gpt-4o`" as the LLM. Furthermore, we choose the `text-embedding-3-small` as our text embedding approach, which is the newest embedding model released by OpenAI in January 2024. Compared to its more powerful counterpart `text-embedding-3-large`, `text-embedding-3-small` 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 [22], its performance is comparable to that of `text-embedding-3-large`.

We implement the prototype system in Python 3. Our system is carried out in the Ubuntu 22.04.1 LTS environment running on a computer equipped with an Intel® Core™ i9 CPU, 64GB RAM. This computer also has a CUDA-based parallel computing platform with a NVIDIA® GeForce RTX™ 3080 GPU.

4.3 Analysis of Embedding Quality

4.3.1 Embedding Visualization. In this section, we first compare `Doc2Vec` [12] with GPT-4o in terms of the interpretability of event embeddings. The former method was used in a prior research `ProvDetector` [29]. To keep balance, for benign and adversary events, we randomly selected 500 samples from each. After that, we project the event embeddings to a two-dimensional space using t-SNE [28]. The result is shown in Figure 3, where blue points represent benign samples while red ones represent adversarial samples.



(a) Embeddings are generated using Doc2Vec

(b) Embeddings are generated using GPT of OpenAI

Figure 3: Visualization based on t-SNE of event embeddings. The blue points and red points represent benign events and adversary events respectively.

It can be seen from Figure 3(a) that the majority of malicious events are mixed with benign ones. This is consistent with the result presented in the existing paper [29]. To address this challenge, previous researchers design a rareness-based selection algorithm to first extract the potentially malicious part from the provenance graph of a process. However, this method is based on heuristics, which consequently relies on human expertise and typically leads to a risk of bias.

In contrast, our method leverages the extensive knowledge of GPT-4o to enrich the semantics of event descriptions. Therefore, we can observe from Figure 3(b) that small cliques formed by similar events, although benign and adversarial samples are not trivially linear separable. This is because we compress the high-dimensional space into a 2-dimensional plane, which leads to information loss. The separating hyper-planes between groups that could originally be distinguished are no longer apparent. However, it also implies that the existing event embeddings can be used as a feasible starting point. Based on this, we can take advantage of the subsequent machine learning models to further amplify the distance between benign and adversary events in a feature space. At the same time, the distance between two benign events, and the distance between two adversarial events can both be reduced. The experiment results presented in Section 4.4 and Section 4.5 validate our hypothesis.

4.3.2 Explicability of Embedding Model. Intuitively, if the numerical representations for events provided by the embedding model are semantically meaningful, the degree of similarity between events can be roughly reflected in the embedding space by the proximity or distance. To delve deeper into the interpretability of GPT-generated embeddings, we showcase two pairs of system events as case studies. The first pair of system events are as following:

“`touch execve root /bin/bash <NA>`”

“`sh execve root /bin/bash <NA> touch`”

Obviously, the processes involved in the two events are completely different. In the latter event, `sh` is a shell command line interpreter that executes a command specified by the event argument, which in this case is `touch`. According to our experience, these two events have the identical functionality. The cosine distance between their numerical embeddings is approximately 0.10884, indicating that the inference of the LLM is consistent with our prior knowledge. We also can observe the similar result when inspecting the following events pair:

“`useradd execve root /bin/bash <NA>`”

“`sh execve root /bin/bash <NA> useradd`”

The cosine distance between their numerical embeddings is approximately 0.10778.

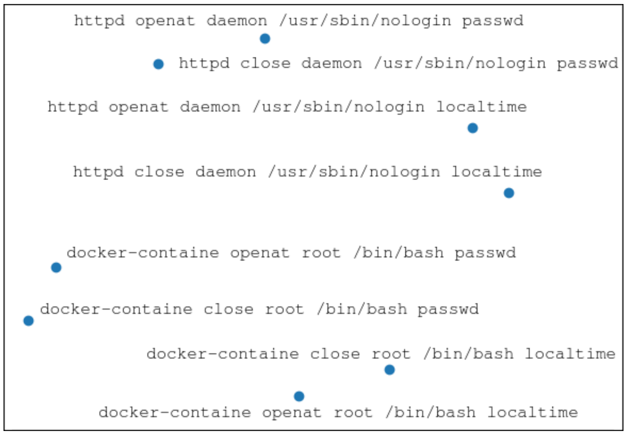
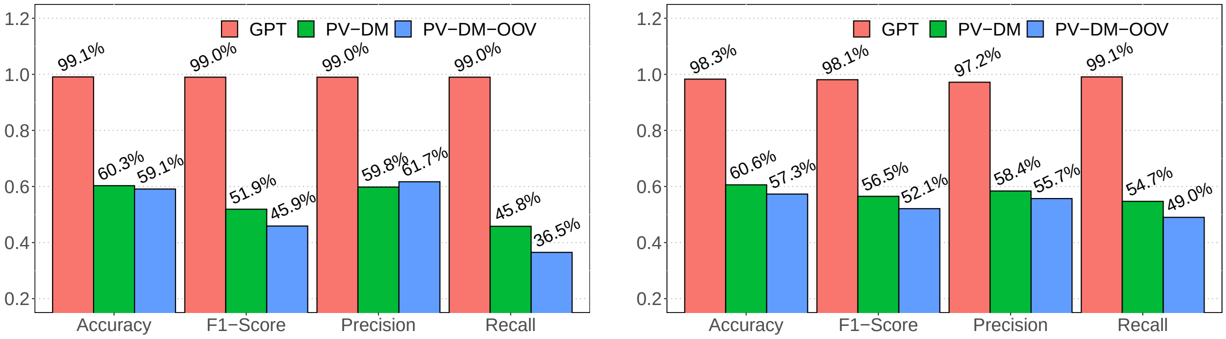


Figure 4: The relative positions of eight system events in the embedding space are visualized using the MDS technique.

Furthermore, we examine the quality of GPT-based system event embeddings in terms of their analogical reasoning performance. More concretely, we plot the relative positions of eight events according to their cosine distance in a lower-dimensional space using the Multidimensional Scaling (MDS) technique. The result is shown in Figure 4, from which it is easy to observe that the distance between the following two events

“`httpd openat daemon /usr/sbin/nologin passwd`”



(a) MLP-based detector

(b) GBDT-based detector

Figure 5: Comparison of different embedding methods for the final threat detection performance.

`“httpd close daemon /usr/sbin/nologin passwd”`
is close to the distance between
`“httpd openat daemon /usr/sbin/nologin localtime”`
`“httpd close daemon /usr/sbin/nologin localtime”`

Similarly, the distance between the following two events
`“docker-containe openat root /bin/bash localtime”`
`“docker-containe close root /bin/bash localtime”`
is close to the distance between
`“httpd openat daemon /usr/sbin/nologin localtime”`
`“httpd close daemon /usr/sbin/nologin localtime”`
This parallelogram relationship in geometry is consistent with our prior knowledge. We limit the presented examples to eight due to space limitation. In our manual investigation, however, we find many such semantic analogies that are automatically learned. Therefore, it can be seen that the embeddings demonstrated good explicability.

4.4 Comparison

In this section, we compare the representation learning adopted in our approach with the embedding method used in the previous paper [29], and examine their impact on the final threat detection. More concretely, the `PV-DM` model of `Doc2Vec` was employed in [29]. 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.

It should be noted that, to ensure fairness, we use Kernel PCA to reduce the dimensionality of the feature vectors generated by GPT, from 1,536 to 256. Similarly, the feature vectors generated by `Doc2Vec` are also kept at the same dimensionality, i.e., 256 dimensions. 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. The experimental results are shown in Figure 5.

Table 2: Threat 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%

As shown in Figure 5, 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.

4.5 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. 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 train artificial intelligence (AI) models. Specifically, semi-supervised outlier detection can be performed even when the training data consists only of observations describing normal behavior. After all, in reality, collecting normal system events is often much easier than acquiring malicious system events. We can then predict whether unknown events are caused by an attack by evaluating their deviation from known events.

In detail, we adopt `XGBOD` [35] to build a threat detection model, which is a semi-supervised outlier detection algorithm. Specifically, our anomaly detection was implemented based on `PyOD` [36], 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 2. Not surprisingly, the performance of semi-supervised learning is slightly worse than that of supervised learning. However, the `XGBOD`-based threat detection model still achieved good detection results, with a 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.

4.6 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 `Log4j` 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.

Following the previous setup, our training set contains 5,000 benign events, and after excluding the `Log4j` attack, the training set includes 3,681 adversary events. After that, we randomly extract 500 adversary events from `Log4j` attack as the testing set. We plot a ROC (receiver operating characteristic) curve to graphically illustrate the performance of this APT detector. We also include 500 benign events in this new testing set because the ROC curve is the plot of the true positive rate (TPR) against the false positive rate (FPR) at varying threshold setting.

Still, we adopt the semi-supervised algorithm `XGBOD` [35] to detect adversary events launched by the `Log4j` attack. The ROC obtained on the testing set is shown in Figure 6, with an AUC value of 97.56%. This indicates that the proposed technique demonstrates strong adaptability especially when faced with attacks that were never encountered before.

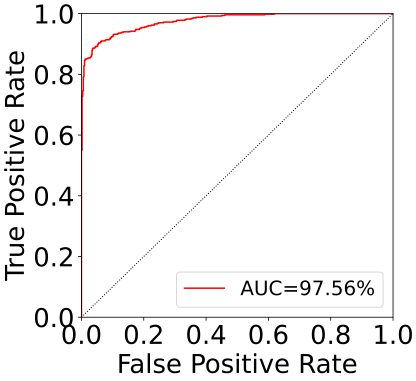


Figure 6: The ROC Curve.

5 Related Work

In this section, we briefly review the existing literature related to our work. Considering the large volume of research in this field, the following review is not intended to be exhaustive.

5.1 Provenance Analysis

Provenance data capture the relationships and interactions between system entities, providing a comprehensive view of information flow and causal relationships within a system. Therefore, provenance analysis plays a significant role in both real-time threat detection and post-incident forensic investigation. Dependence tracking analysis [11] has been used to analyze a large volume of data effectively. Provenance tracking has been done in different data granularity. `BEEP` [13] and `Protracer` [18] use units that are execution partitions of application code which is common in event-handling loops. `MPI` [17] uses user input on data structures to define execution partitions. `PrioTracker` [16] proposed priority-based causality tracking using rareness score and fanout score as indications of unusualness. Bates et al. [1] proposed Linux Provenance Modules (LPM), a kernel-based framework and data loss prevention system for sensitive data. `PalanTir` [33] uses a processor tracing (PT) hardware technique to enable finer-grained instruction level tracking. `Kairos` [2] proposed a graph neural network-based encoder and decoder to learn the temporal evolution of the provenance graph's structural changes. `ProvDetector` [29] is the closest related method to the one proposed in this paper. For a causal path consisting of multiple events, `ProvDetector` uses the `PV-DM` model of `Doc2Vec` [12] to learn an embedding. The shortcomings of this approach are evident: first, the semantics it can capture are shallow. However, our method leverages the extensive knowledge of LLMs to enrich the semantics of event descriptions. Secondly, `ProvDetector` cannot well address the OOV (out-of-vocabulary) challenge. Our approach, on the other hand, does not face this issue.

5.2 Threat Detection

Using provenance in threat detection and investigation has been explored by a large body of work [15]. Multiple attack detection approaches have been proposed based on anomaly scores. For example, the threat detection system `PIDAS` [31] 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` [30], considers the anomaly score of both a single provenance path and the whole graph. Furthermore, `NoDoze` [7] uses a network diffusion algorithm that propagates anomaly scores across dependency graphs to calculate anomaly scores.

Tag propagation is another widely used strategy. `SLEUTH` [8] 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 subsequent work [9] 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` [20] 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 many empirical parameters, which leads to unstable detection results.

5.3 LLMs for Cybersecurity

Existing work largely employs the GPT family of LLMs to address different security challenges. For instance, previous works [24, 34] investigated the capabilities of ChatGPT in detecting vulnerabilities and resolving bugs. In particular, Qu et al. [24] found the debug performance can be remarkably improved after taking advantage of contextual information. Furthermore, Yan et al. [32] took advantage of GPT-4 to generate descriptive texts for each API call. Based on this, they developed a BERT-based dynamic malware analysis technique. Lastly, the knowledge of LLMs was also applied to detect DDoS attacks [6, 14]. To the best of our knowledge, we are the first to propose a technique that leverages the extensive knowledge of LLMs to assist in provenance analysis for APT detection.

6 Discussion and Future Work

Generality of Sysdig: As a widely used security event tracing system, Sysdig event format is well-known to multiple widely used LLMs. For instance, as well as OpenAI's models, open-source LLMs, Meta's LLAMA models, also recognize the sysdig format. The advantage of this event format to LLMs is mainly the clarity of the system event details broken down into clear key-value pairs. Alternatively, for LLMs without the knowledge of Sysdig, the same details of system call event information can be formulated as a prompt.

Usage of OpenAI LLM Models: At the time of our experiment, OpenAI GPT-4o has the best performance as an industry-leading product. Also, the text embedding models of OpenAI are competitive. Therefore, we chose OpenAI's models for our evaluation.

However, we note the proposed method is agnostic to LLM models and is also applicable to other LLM models as well. Other open-source or commercial LLM models such as Meta's LLAMA, Microsoft Copilot, and Google Gemini can be utilized for the same purpose.

Generality of JSON input: We chose the JSON as the input format for LLMs because this is a common format widely utilized by LLMs as an input and output along with other formats such as Markdown. Since JSON is a structure that clearly delivers multiple key-value pairs, we used it for the brevity of the input prompt. Prior state-of-the-art approaches used the sentence format formulating the system call event as a subject, a verb, and an object format. This is just an alternative format that is equally effective for LLMs if we explain the positional role of each token such as the first term is a subject, the second term is a verb, etc. However, this sentence format is likely to make the prompt longer causing a higher cost for the paid LLM models.

Future Work: Our work mainly utilizes the state-of-the-art LLMs of OpenAI for our evaluation because our manual evaluation of the explanation quality is satisfactory. As our future work, we plan to evaluate other LLMs performance for security tasks. Depending on the number of parameters, the training data size, and the inclusion of security context for training, each LLM model's capability could vary. We look forward to investigating the security expertness of different LLM models.

7 Conclusion

In this paper, we explored a novel APT detection method that offers a significantly improved detection performance by utilizing LLMs to augment the semantics of provenance analysis events.

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 NLP-based detection methods.

Our experiment shows that text embedding on the augmented LLM description significantly improves the performance over the representative method adopted by a 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%. This result demonstrates the usefulness of the LLMs for APT detection with regard to the richness of semantic information.

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] Adam Bates, Dave Tian, Kevin R. B. Butler, and Thomas Moyer. 2015. Trustworthy whole-system provenance for the Linux kernel. In *24th USENIX Security Symposium*. 319–334.

[2] Zijun Cheng, Qiujian Lv, Jinyuan Liang, Yan Wang, Degang Sun, Thomas Pasquier, and Xueyuan Han. 2024. KAIROS: Practical intrusion detection and investigation using whole-system provenance. In *IEEE Symposium on Security and Privacy*.

[3] Federal Trade Commission. 2024. Equifax data breach settlement. https://www.ftc.gov/enforcement/refunds/equifax-data-breach-settlement. Accessed: 2024-12.

[4] CrowdStrike. 2024. Global threat report. https://go.crowdstrike.com/global-threat-report-2024.html. Accessed: 2024-12.

[5] Jacob Fox. 2024. Top 40 AI cybersecurity statistics. https://www.cobalt.io/blog/top-40-ai-cybersecurity-statistics. Accessed: 2024-12.

[6] Michael Guastalla, Yiyi Li, Arvin Hekmati, and Bhaskar Krishnamachari. 2023. Application of large language models to DDoS attack detection. In *International Conference on Security and Privacy in Cyber-Physical Systems and Smart Vehicles*. Springer, 83–99.

[7] Wajih Ul Hassan, Shengjian Guo, Ding Li, Zhengzhang Chen, Kangkook Jee, Zhichun Li, and Adam Bates. 2019. NoDoze: Combatting threat alert fatigue with automated provenance triage. In *Network and Distributed Systems Security Symposium*.

[8] Md Nahid Hossain, Sadegh M Milajerdi, Junao Wang, Birhanu Eshete, Rigel Gjomemo, R Sekar, Scott Stoller, and VN Venkatakrishnan. 2017. SLEUTH: Real-time attack scenario reconstruction from COTS audit data. In *26th USENIX Security Symposium*. 487–504.

[9] Md Nahid Hossain, Sanaz Sheikhi, and R Sekar. 2020. Combating dependence explosion in forensic analysis using alternative tag propagation semantics. In *IEEE Symposium on Security and Privacy (SP)*. 1139–1155.

[10] Sysdig Inc. 2024. Sysdig Monitor. https://docs.sysdig.com/en/docs/sysdig-monitor/. Accessed: 2024-12.

[11] Samuel T King and Peter M Chen. 2003. Backtracking intrusions. In *Proceedings of the 19th ACM symposium on Operating systems principles*. 223–236.

[12] Quoc Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *International conference on machine learning*. 1188–1196.

[13] Kyu Hyung Lee, X. Zhang, and Dongyan Xu. 2013. High accuracy attack provenance via binary-based execution partition. In *Network and Distributed System Security Symposium (NDSS)*.

[14] Qingyang Li, Yihang Zhang, Zhidong Jia, Yannan Hu, Lei Zhang, Jianrong Zhang, Yongming Xu, Yong Cui, Zongming Guo, and Xinggong Zhang. 2024. DoLLM: How large language models understanding network flow data to detect carpet bombing DDoS. *arXiv preprint arXiv:2405.07638* (2024).

[15] Zhenyuan Li, Qi Alfred Chen, Runqing Yang, Yan Chen, and Wei Ruan. 2021. Threat detection and investigation with system-level provenance graphs: A survey. *Computers & Security* 106 (2021), 102282.

[16] Yushan Liu, Mu Zhang, Ding Li, Kangkook Jee, Zhichun Li, Zhenyu Wu, Junghwan John Rhee, and Prateek Mittal. 2018. Towards a timely causality analysis for enterprise security. In *Network and Distributed System Security Symposium (NDSS)*.

[17] Shiqing Ma, Juan Zhai, Fei Wang, Kyu Hyung Lee, Xiangyu Zhang, and Dongyan Xu. 2017. MPI: Multiple perspective attack investigation with semantic aware execution partitioning. In *26th USENIX Security Symposium*. 1111–1128.

[18] Shiqing Ma, X. Zhang, and Dongyan Xu. 2016. ProTracer: Towards practical provenance tracing by alternating between logging and tainting. In *Network and Distributed System Security Symposium (NDSS)*.

[19] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems* 26 (2013).

[20] Sadegh M Milajerdi, Rigel Gjomemo, Birhanu Eshete, Ramachandran Sekar, and VN Venkatakrishnan. 2019. Holmes: real-time apt detection through correlation of suspicious information flows. In *IEEE Symposium on Security and Privacy (SP)*. 1137–1152.

[21] Jiang Ming, Dongpeng Xu, Yufei Jiang, and Dinghao Wu. 2017. BinSim: Trace-based semantic binary diffing via system call sliced segment equivalence checking. In *26th USENIX Security Symposium*. 253–270.

[22] OpenAI. 2024. New embedding models and API updates. https://openai.com/index/new-embedding-models-and-api-updates/. Accessed: 2024-2.

[23] Matteo Pagliardini, Prakhar Gupta, and Martin Jaggi. 2018. Unsupervised Learning of Sentence Embeddings Using Compositional n-Gram Features. In *Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*.

[24] Xianshan Qu, Fei Zuo, Xiaopeng Li, and Junghwan Rhee. 2024. 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)*.

[25] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. *Improving language understanding by generative pre-training*. Technical Report. OpenAI.

[26] IBM Security. 2022. Cost of a data breach report. https://www.ibm.com/reports/data-breach. Accessed: 2023-05.

[27] Madhukar Shrestha, Yonghyun Kim, Jeehyun Oh, Junghwan Rhee, Yung Ryn Choe, Fei Zuo, Myungah Park, and Gang Qian. 2023. ProvSec: Open cybersecurity system provenance analysis benchmark dataset with labels. *International Journal of Networked and Distributed Computing* 11, 2 (2023), 112–123.

[28] Laurens Van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of machine learning research* 9, 11 (2008).

[29] Qi Wang, Wajih Ul Hassan, Ding Li, Kangkook Jee, Xiao Yu, Kexuan Zou, Junghwan Rhee, Zhengzhang Chen, Wei Cheng, Carl A Gunter, et al. 2020. You are what you do: Hunting stealthy malware via data provenance analysis.. In *Network and Distributed System Security Symposium (NDSS)*.

[30] Yulai Xie, Dan Feng, Yuchong Hu, Yan Li, Staunton Sample, and Darrell Long. 2018. Pagoda: A hybrid approach to enable efficient real-time provenance based intrusion detection in big data environments. *IEEE Transactions on Dependable and Secure Computing* 17, 6 (2018), 1283–1296.

[31] Yulai Xie, Dan Feng, Zhipeng Tan, and Junzhe Zhou. 2016. Unifying intrusion detection and forensic analysis via provenance awareness. *Future Generation Computer Systems* 61 (2016), 26–36.

[32] Pei Yan, Shunquan Tan, Miaohui Wang, and Jiwu Huang. 2023. Prompt engineering-assisted malware dynamic analysis using GPT-4. *arXiv preprint arXiv:2312.08317* (2023).

[33] Jun Zeng, Chuqi Zhang, and Zhenkai Liang. 2022. PalanTír: Optimizing attack provenance with hardware-enhanced system observability. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. 3135–3149.

[34] Chenyuan Zhang, Hao Liu, Jiutian Zeng, Kejing Yang, Yuhong Li, and Hui Li. 2024. Prompt-enhanced software vulnerability detection using ChatGPT. In *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings*. 276–277.

[35] Yue Zhao and Maciej K Hryniewicki. 2018. XGBOD: Improving supervised outlier detection with unsupervised representation learning. In *2018 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–8.

[36] Yue Zhao, Zain Nasrullah, and Zheng Li. 2019. PyOD: A python toolbox for scalable outlier detection. *Journal of machine learning research* 20, 96 (2019), 1–7.

[37] Fei Zuo and Junghwan Rhee. 2024. Vulnerability discovery based on source code patch commit mining: a systematic literature review. *International Journal of Information Security* 23, 2 (2024), 1513–1526.

[38] Fei Zuo, Xin Zhang, Yuqi Song, Junghwan Rhee, and Jicheng Fu. 2023. Commit message can help: security patch detection in open source software via transformer. In *2023 IEEE/ACIS 21st International Conference on Software Engineering Research, Management and Applications (SERA)*. 345–351.