UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Tècnica Superior d'Enginyeria
de Telecomunicació de Barcelona

Leveraging Machine Learning for Logs Reasoning

Master Thesis
submitted to the Faculty of the
Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona
Universitat Politècnica de Catalunya
by

Nina Raganová

In partial fulfillment
of the requirements for the master in
CYBERSECURITY

Supervisor: Felician Paul Almasan Puscas
Co-supervisor: Pere Barlet Ros
Barcelona, September 2024

Contents

List of Figures

List of Tables

Abstract

Logs are a rich source of information providing an insight into the system operation. However, the substantial size of logs makes it difficult to analyze them manually. Therefore, the attention of current research has focused on automatic processing and analysis of this information to discover system breaches. In this thesis, we propose a novel approach to log anomaly detection using Graph Neural Networks by converting the problem to a link prediction task and training the model to recognize the anomalous events in the system execution. We show that using this approach, we are able to discover malicious entities hidden in the system among benign activity.

Acknowledgments

I would like to thank my supervisor Felician Paul Almasan Puscas for all his advice, time that he devoted to the realization of this thesis, and guidance throughout the whole project.

I would also like to thank my co-supervisor Pere Barlet Ros for his ideas and contributions to the discussions about the design of the solutions.

Finally, I would like to thank the research team from Telefónica Innovación Digital for giving me an opportunity to work on this project under their supervision.

1 Introduction

System logs contain valuable information about the system operation, capturing important events happening on the machine. This information provides an insight into the execution of programs on the host, as well as its network communication. Therefore, logs are very useful from the security point of view, because they could contain traces of attacks, which might help with their investigation. In case of an attack, the security analysts would need to go through the log records to identify the potential source and all the steps of the attack. Also, it is often the case that a zero-day vulnerability is found being exploited in the wild and the security analyst needs to find out whether it has affected the company systems. This kind of situations might be difficult to identify in the logs, as sometimes it means that the analyst would need to check large amounts of data that were produced by the system in the time frame of interest.

Machine learning is revolutionizing industries, with applications like Netlix's personalized recommendations, Google's deep learning for spam detection, and Microsoft's approach to phishing detection. These innovations highlight the power of machine learning to learn from data and improve over time.

We plan to leverage machine learning techniques to detect anomalies in logs, aiming to identify potential security threats early by spotting unusual patterns. This approach will enhance our ability to protect against cyber threats.

1.1 Objectives and Motivation

The objective of this master's thesis was to design and implement novel techniques to assist cybersecurity specialists to analyze and interpret system logs. We investigate the use of advanced machine learning techniques to experiment with new approaches to automatic attack detection to facilitate the process of analyzing log data. To achieve our goals, we studied the state-of-the-art systems proposed in various research papers. Having thoroughly analyzed and understood the underlying problem, we searched for real-world datasets containing records of both benign and malicious activity. By analyzing the data, we were able to propose a new approach to logs anomaly detection. To validate our ideas, we developed a novel pipeline for processing system logs information, and designed experiments for its evaluation to find out whether this approach might be effective for this use case.

Provenance describes the totality of system execution and facilitates causal analysis of system activities by reconstructing the chain of events that lead to an attack (backward tracing) as well as the ramifications of the attack (forward tracing). [14] Provenance graphs are a way to visualize the logs and reason over them in a structured way. We represent the collected system logs as a provenance graph, capturing the system execution. Therefore, we apply machine learning methods that take advantage of the graph structure of the problem.

Graph Neural Networks (GNNs) are neural networks designed to learn from graphs. They provide a way to solve node-level, edge-level, or graph-level prediction tasks. To analyze the complex topology of graphs, they aggregate the information from neighboring nodes

and edges for graph reasoning. GNNs have proved to be effective on various real-world problems, such as fraud detection or social network predictions. [22] These results convinced us to experiment with GNNs in terms of intrusion detection by representing the logs as a provenance graph of system operation.

The primary objective of this thesis was to design and implement techniques for analysis and interpretation of system logs, along with provenance graph generation to identify relationships between system components. After the review of related literature, we identified key limitations of previously published work, which inspired us to design a provenance-based intrusion detection system based on GNNs.

We developed the code using programming language Python, as it is standardly used for machine learning problems, and has a lot of suitable libraries available. As we worked with the dataset in the form of a graph, we used the NetworkX [8] library for the data representation, processing, and visualization. We also used multiple machine learning libraries, such as Gensim [2], PyTorch [3], and PyTorch Geometric [4]. The algorithms and machine learning models that were used had been proposed in other articles, and are explained in more detail in Chapter 4. We decided to use Word2Vec [6], TransE [7], and Graph Neural Networks, namely the GraphSAGE [5] model. More details about the implementation are also present in Chapter 4. This master's thesis was developed in collaboration with research team at Telefónica Innovación Digital during a four month internship.

1.2 Project Plan

The plan of the work is depicted in the Gantt diagram in Figure 1. In order to design our own solution, we started with the research phase, which was followed by the design and implementation phase. In the research phase, the main tasks included review of existing literature to get acquainted with the problem, analysis of limitations of the state-of-the-art proposed solutions, and selection of suitable libraries that would facilitate the work with machine learning models and run code with optimal performance. The design and implementation phase consisted of defining and working with two distinct approaches, their design, implementation, and subsequent evaluation. Some additional ideas are proposed for future work in Chapter 6.

We encountered various problems when implementing the machine learning algorithms, mostly with the scalability to such huge amounts of data as are usually present in system logs. Some of them are mentioned in Section 4.3.

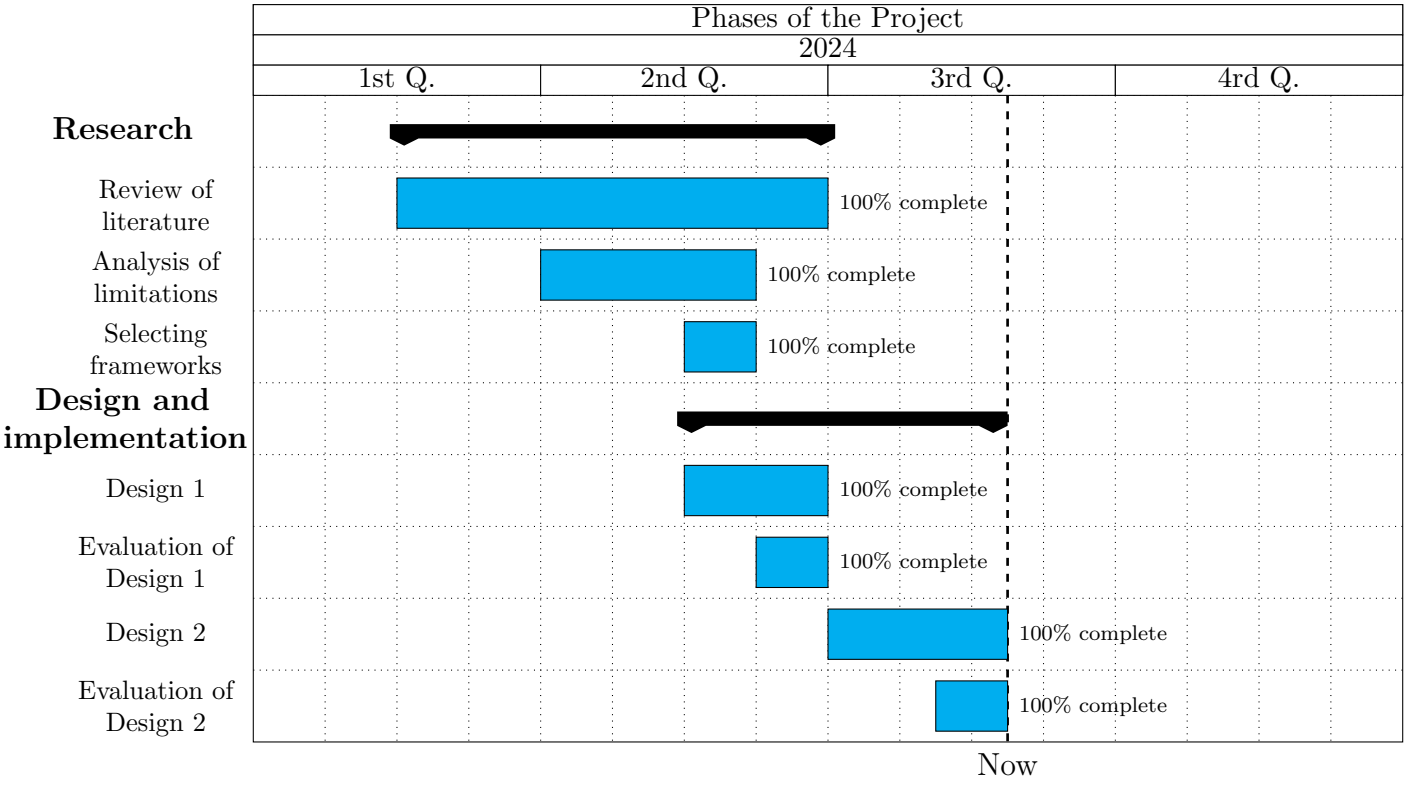

Figure 1: Gantt diagram of the project showing all the phases with their respective tasks.

1.3 Outline of the Thesis

Chapter 2 presents an overview of previously existing work related to the topics mentioned in this thesis, explaining their main ideas and contributions to the field. In section 2.2, we explain the concept of a provenance graph and how it represents the system execution. Section 2.3 deals with the problem of provenance-based intrusion detection, the challenges introduced in the related work, their approach to these issues, and results of the proposed methods.

Chapter 3 shows other related work in the field, i.e. multiple approaches to log compression, the illustration of the problem and the utility of the proposed solutions.

The methodology and implementation details of this thesis are explained in Chapter 4. In this chapter, we introduce the dataset we were working with, and explain the machine learning algorithms we used for the anomaly detection.

The main results and contributions of this thesis can be found in Chapter 5. We describe the entire logs processing pipeline for anomaly detection, show the results of the performed experiments, and analyze the data to show its relevance for future research.

Chapter 6 summarizes the work, points out the contribution of this thesis, and introduces potential ideas for future continuation of the research on logs reasoning. The chapter also

provides a brief recapitulation of the challenges we mention in the thesis, emphasizing the importance of further exploration of this topic.

The source code files containing the implementation of the machine learning models are described in the Appendix A.

2 Background

The notion of the application of machine learning for logs reasoning has been present in many works. In this chapter, we discuss related work that became an inspiration for the development of our own technique for anomaly detection.

2.1 Logs Reasoning

Provenance-based Intrusion Detection Systems (PIDS) utilize data provenance to enhance the detection performance of intrusions and reduce false-alarm rates compared to traditional IDSs. [1] The notion of using information stored in logs for building a provenance-based intrusion detection system has been present in many works, e.g. [9], [10], or [11].

The detection of attacks from the logged information has multiple benefits. This approach is well suited for devices where performance is critical for their operation. A standard real-time detection agent would take up the device's resources and hinder its performance, whereas detecting the attacks based only on the logged information can be carried out by a different device, assuming it has access to the logs. Additionally, this method avoids any potential interoperability issues, as it does not require the access to any other parts of the system than the logs, and does not communicate with any other processes or applications. Moreover, some devices, such as IoT devices, have strong memory constraints when it comes to the software they are able to run. Thus in this case, it is a great advantage that the detection process does not necessarily have to be running on the device itself. Also, this approach is not dependent on a specific type of logs, which means that it can support any operation system or application. Having a single cross-platform solution is easier to develop and maintain, which also contributes to its value.

There are certainly also other real-time techniques for intrusion detection widely used with great performance, for example many EDRs are using behavioral detection, which is based on descriptions of potentially harmful behavior of programs. Nevertheless, logs are undeniably a great source of data about the past events that have happened on the machine. Therefore, they provide a great opportunity to examine the execution with a delay, after several actions have been made and some time has passed. Serving as a memory of the past events of the system, logs are a place to search for past breaches and causes of application crashes or misbehaviors, providing more context. Therefore, building a provenance graph capturing the system operation seems to be an interesting idea for multiple reasons. Not only it can serve as a foundation for a real-time intrusion detection, it could also provide the security analyst with an insight into the former events.

To be able to reason on logs, the first step is to create a provenance graph that would depict the interactions between all the entities present on the system. Therefore, we represent all the entities on the system (e.g. files, processes, or network sockets) as nodes, and the events depicting their interactions as edges between them. This view helps security analysts to directly see the ongoing activity and relationships between the entities. However, as there is a huge amount of activity happening on the system every second, the provenance graphs can easily become extremely large and incomprehensible for human inspection. For an automated intrusion detection using artificial intelligence, it is necessary to convert all

the entities and their interactions to a certain numerical representation. This allows us to feed this data into machine learning models and perform the logs reasoning automatically.

2.2 Provenance Graphs

As the aim of this thesis was to implement and compare techniques for log anomaly detection using machine learning, we searched for a viable option to represent our problem in a convenient way.

A provenance graph is a data structure in the form of a directed graph, where edges capture the history, otherwise called provenance, of the entities, which are represented as the nodes of the graph. Each node and edge has a set of attributes based on its type, giving its description. The direction of an edge defines the causal relationship between its ends, and its type represents a specific operation which was performed. The nodes of the graph correspond to system entities, e.g. files or processes, capturing their state.

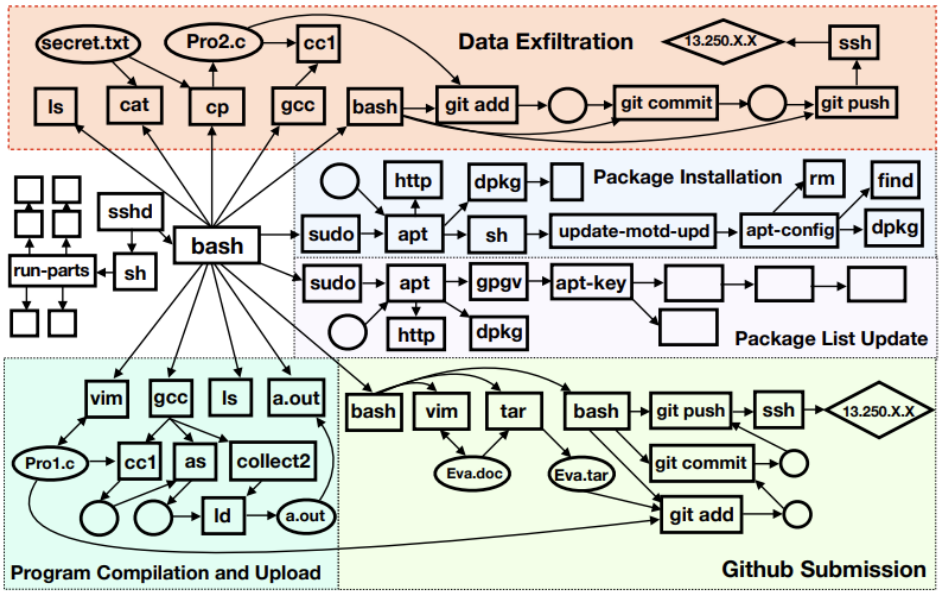

Figure 2: A small example of a provenance graph. Rectangles represent processes, ovals files, diamonds sockets, and edges in the picture system calls. The colored boxes correspond to high-level behaviors. [10]

The provenance graph of the system operation is constructed by parsing system logs. Every record in the log is an event, which is converted to an edge between the subject and the object of the record. By parsing the whole log, we assemble the graph recording all the events that have happened till now. An example of a provenance graph is shown in Figure 2. This data representation allows us to capture the causality of actions executed on the system by tracking their origin. Additionally, a node neighborhood represents the behavior of an entity, which helps us recognize strange behavior patterns. To use a provenance graph as an input for machine learning methods, we need to convert it to a

numerical representation by embedding its nodes and edges. In our specific case, we need to embed the nodes, because some of their attributes are in a textual form, but the edges have only a single attribute, which is the edge type. Therefore, we decided to assign them numbers from 0 to 29, as there are 30 different edge types in our dataset, and embed them as 32-dimensional vectors with the edge type number at the first coordinate and zeros at the other ones. The description of the dataset can be found in more detail in Section 4.1.

2.3 Provenance-based Intrusion Detection Systems

PIDS work on the principle of building provenance graphs from logs to represent the system operation, and consecutive application of machine learning algorithms for anomaly detection. The anomalies in the entity's behavior correspond to actions possibly related to an attack.

Conceptually, the design and domain of these systems can be categorized into several layers [14]. The capture layer deals with the audit log generation and grants the tamperproofness of the log. This matter is not directly addressed in any of the proposed PIDS approaches, as it is always treated as a fundamental assumption that it is the property of the underlying logging mechanism. The aim of the reduction layer is to reduce the size of the logs, while preserving the information needed for detection or later analysis of an attack. The techniques attributed to this layer are usually using graph compression algorithms to get rid of events that are redundant or unimportant for the forensic validity of the log. The infrastructure layer ensures that the log data is structured and stored in a way that makes it easily accessible to the upper layers of the analysis pipeline. This includes query support, being able to retrieve relevant subgraphs without the need to analyze the whole provenance graph to search for the source of the intrusion. The detection layer, which is usually heuristic- or anomaly-based, strives to perform automated analysis of data to recognize potential attack indicators. However, these approaches often lack the explainability of why they raised the alert as the underlying machine learning algorithms do not provide such information. The investigation layer aims to provide alert correlation and triage to effectively inspect the data. Additionally, several works have tried to help the analysts with behavioral diagnosis by deriving the high-level interpretation of low-level audit events. The conclusion of the experiments performed by the authors was that they observed an inverse relation between storage efficiency and anomaly detection performance of the proposed solutions.

One of the recently published papers proposing a PIDS called Flash is based on the assumption that attackers must leave behind identifiable patterns in the system's records to carry out malicious activities distinct from typical behavior [9]. First, they construct a provenance graph of the system behavior from logs and convert nodes to a numerical representation using Word2Vec model to embed the node attributes. After the semantic encoding of the nodes, a GNN is employed for the contextual encoding by using the node embeddings produced by Word2Vec along with the graph connectivity as an input to learn the structural information of each node. The GNN is trained to classify the nodes into categories corresponding to different entity types, such as files, processes, or network flows. The misclassified nodes then correspond to the attack-related entities in the graph.

Their approach was able to achieve F-score over 0.9 on almost all the tested datasets.

A little different perspective was presented by the authors of Watson [10]. Their ambition was to design an automated approach to abstraction of behaviors from low-level audit events based on their context in the logs to reduce analysis workload during an attack investigation. To infer the semantics of the events, they use TransE, a translation-based embedding model, which converts both the nodes and the edges to a vector representation. They proceed to summarize a behavior as a subgraph representing a sequence of events operated on related data and connected by information flows. By performing an adapted DFS algorithm from every entity in the graph, they are able to partition the graph into smaller subgraphs summarizing different behavior instances. To extract the semantics of a behavior instance, they sum the vectors corresponding to the events contained in the subgraph. The final step is the behavior clustering, which is accomplished by calculating the cosine similarity of the subgraphs. The results show that the approach is able to both reduce the analysis workload by presenting only representative instances to the analysts, and preserve the attack behaviors as separate clusters.

We conclude the overview by mentioning Kairos [11], a PIDS availing GNN-based encoder-decoder architecture. Based on the neighborhood structure of the edge and the state of the nodes involved in it, the encoder produces edge embeddings for the decoder, which reconstructs the edges. The difference between the real edge and the reconstructed edge, the reconstruction error, is minimized during the training on the benign edges. For the anomaly detection, the authors decided to construct time window queues by identifying sets of suspicious nodes based on the reconstruction errors. Windows with overlapping suspicious nodes are inserted to the same queue and analyzed together. When a queue exceeds a certain detection threshold, it is labeled as anomalous. Finally, the queue is summarized into a compact representative graph by keeping communities of nodes connected with edges with high reconstruction errors. Employing this approach, they report the accuracy over 96% on all the tested datasets. Moreover, by achieving 100% recall, they correctly identified all the attack time windows in the datasets. The only issue is a few false positives, which are present mostly because of activities previously unseen in the training data present on the system, which are therefore deemed as anomalous.

3 Related Work

According to OWASP Top 10 2021, Security Logging and Monitoring Failures were ranked as the 9-th most critical security concern in web applications. This shows the importance and difficulty to set up and configure logging mechanisms effectively. Collecting and reviewing logs has to be a continuous process, to ensure that any signs of problems do not remain unnoticed. However, storing these records for longer periods of time becomes unsustainable. Some advanced persistent threats (APTs) are known to perform their attacks stealthily to evade detection. Therefore, they sometimes divide the attack into small steps and remain silent for extended periods of time. The complete attack can eventually take months, which can become complicated to detect and investigate. As a result, various research groups have attempted to propose techniques for logs compression with regards to cybersecurity objectives. This approach is striving to narrow the size of the logs, while preserving the records related to attacks.

One of the works aiming to compress the logs to aid with the investigation of the APT attacks is NodeMerge [12]. In this article, the authors propose a template-based data reduction system for online system event storage. Their reduction technique is based on the observation that frequent execution of programs results in large amount of redundant events, as the initialization stage of each process requires to interact with a lot of read-only files. This observation lead the authors to the idea of creating file templates, which are sets of frequently used read-only files. NodeMerge learns these templates automatically and applies them to compress the system logs. The experiments showed that the approach was able to compress the log files 33.7 times on a data analytics server, 15.1 times on a developer's host, while needing 4.2 times less space on other types of hosts. These results show that it is possible to maintain events potentially needed for attack investigation, while saving up more space.

A different approach was chosen by the authors of Winnower [13]. The authors propose grouping log records from replicated tasks in identical nodes of container clusters. This forms repeating patterns that summarize the behavior of individual hosts. The patterns were used to create a representative model using graph grammar learning techniques. Using this technique, the authors claim that their approach reduces necessary storage capacity by 98%, while preserving the important information for attack investigation. However, later research survey [14] states that it does not work in other settings than containers, as a general tool, underperforming the reported results.

The authors of Faust [15] decided to combine multiple techniques together as log reduction modules. The examined methods included both NodeMerge and Winnower, but also other approaches, such as LogGC [16], PCAR [17], or S-DPR [18]. The best results were achieved by combining all the methods, which helped to reduce the log size by 90.7%. The most effective combination of 2 techniques turned out to be the application of NodeMerge along with S-DPR, as the compression in this setting was able to diminish the size of the log by 89.17%. However, while NodeMerge was the only approach to retain 100% of the attack edges present in the dataset, S-DPR was able to preserve only 13.5% of them. These results indicate the complexity of the task, as finding the balance between a high compression ratio and the attack traces retention is usually not straightforward.

4 Methodology and Implementation

Since logs provide contextually rich information about the system operation, they can become targeted by the attackers, which is known as the Indicator Removal technique, described by the MITRE ATT&CK® matrix. This technique is used to evade detection and hinder incident response by erasing artifacts that might include traces of the attack or which were created as a by-product of the attack execution, and may leak information about its realization. Despite it being an important issue, we consider this scenario to be out of the scope of this project, as there are various possible mitigations to prevent such behavior, e.g. copying the log files to a remote data storage. Therefore, a fundamental assumption of our work is that we have a reliable logging agent in place, producing records of system operation as expected during normal execution.

In our proposal, the anomaly detection is converted to a link prediction task. Given a set of existing edges representing a partially observed graph connectivity, the problem is to predict the existence of missing links in the graph. As logs represent only a snapshot of the system execution up to a certain point in time, it is crucial for our anomaly detector to be able to predict interactions, which may happen in the future, to distinguish them from anomalous behavior. In our experiments, we formulate the task as the binary classification of potential links as positive (existing in the graph) or negative (non-existent).

4.1 Dataset

For the experimental part of this thesis, we decided to test our ideas on standardly used DARPA dataset [21]. Defense Advanced Research Projects Agency (DARPA) is a part of the United States Department of Defense responsible for the development of emerging technologies for use by the military. According to DARPA, APTs can remain undetected for years if their individual activities can blend with the background noise inherent in any large, complex environment. Therefore, DARPA started the Transparent Computing (TC) program aiming to make currently opaque computing systems transparent by providing high-fidelity visibility into component interactions during system operation. TC program is divided into five Technical Areas (TAs), but only TA1 and TA5 are interesting for our thesis. TA1 covers Tagging and Tracking of interactions between system components (e.g. file accesses by processes) and allows linking them to each other to build provenance graphs of the environment operation. TA5 performed simulations involving APT infiltrations to collect data for TA1. The resulting dataset consists of various attack scenarios, which were monitored by 5 different logging mechanisms. Our experiments were performed using the logs created by the CADETS provenance capture system. The simulation involved an attack that performed exploitation of a webserver hosted on FreeBSD, establishing a reverse shell, network reconnaissance, and process injection. The edge and node types present in the dataset can be seen in Figure 3. The produced logs are composed of multiple files, some of them containing only benign execution, while some contain also traces of the performed attacks.

```
mapping = {'EVENT_ACCEPT': 0,
           'EVENT_BIND': 1,
           'EVENT_CHANGE_PRINCIPAL': 2,
           'EVENT_CLOSE': 3,
           'EVENT_CONNECT': 4,
           'EVENT_CREATE_OBJECT': 5,
           'EVENT_EXECUTE': 6,
           'EVENT_EXIT': 7,
           'EVENT_FCNTL': 8,
           'EVENT_FORK': 9,
           'EVENT_LINK': 10,
           'EVENT_LOGIN': 11,
           'EVENT_LSEEK': 12,
           'EVENT_MMAP': 13,
           'EVENT_MODIFY_FILE_ATTRIBUTES': 14,
           'EVENT_MODIFY_PROCESS': 15,
           'EVENT_MPROTECT': 16,
           'EVENT_OPEN': 17,
           'EVENT_OTHER': 18,
           'EVENT_READ': 19,
           'EVENT_RECVFROM': 20,
           'EVENT_RECVMSG': 21,
           'EVENT_RENAME': 22,
           'EVENT_SENDTO': 23,
           'EVENT_SENDMSG': 24,
           'EVENT_SIGNAL': 25,
           'EVENT_TRUNCATE': 26,
           'EVENT_UNLINK': 27,
           'EVENT_WRITE': 28,
           'EVENT_PREDICATEOBJECT2': 29
}
```

```
creators = {'EVENT': add_event,
            'FILE': add_file,
            'SUBJECT': add_subject,
            'SRCSINK': add_srcsink,
            'NETFLOW': add_netflow,
            'PIPE': add_pipe,
            'USER': add_user,
            'HOST': add_host}
```

Figure 3: The left side shows the edge types along with the mapping to numbers, while node types are listed on the right.

4.2 Machine Learning Algorithms and Models

The first step of our anomaly detection approach is the conversion of attributes from their textual to numerical representation. Similar to Flash [9], we decided to use the Word2Vec model to obtain the vector form of the nodes' attributes. The model is structured as a two-layer neural network which is trained using a large corpus of text. The words with similar contexts are mapped in the vector space close in terms of cosine similarity.

Our first approach for anomaly detection relies on the creation of node embeddings, which would capture the interactions between them. Node embedding is the process of assigning vectors from a certain space, denoted as the embedding space, that would represent their attributes in a way that the vectors corresponding to nodes with similar attributes would be close in the embedding space. We decided to resort to the application of translation-based methods for their mathematical properties. These methods represent both the nodes and the edges as vectors in the embedding space by trying to satisfy the equation $head + relation = tail$ for all the golden triplets $(head, relation, tail)$, where *head* and *tail* are nodes connected by the action of a certain *relation* type. The training is performed by iterating over all the edges which are present in the graph, therefore called golden triplets, and minimizing the distance between the embedding of $head + relation$ and the embedding of the *tail* node, while the distance for corrupted triplets that were constructed by replacing either the *head* or the *tail* of an existing edge should be much higher.

During our research, we found 3 candidate translation-based embedding algorithms, namely TransE [7], TransH [19], and TransR [20]. TransE is using the *relation* as a translation vector to minimize the error of $head + relation - tail = 0$. This approach requires the least resources and time to run among these algorithms, but it has problems with reflexive, many-to-one, one-to-many, and many-to-many relations. TransH performs the translating on hyperplanes, which is depicted in Figure 4 along with TransE. The main difference is in the projection of the node embeddings into a relation-specific hyperplane, where the *relation* vector is applied. TransR employs an entity space completely different from the relation spaces. For each relation, they create a projection matrix, which projects the entities to the relation-specific space. Both TransH and TransR provide a more advanced approach to the creation of the embeddings and can deal better with the aforementioned relation types. However, we decided to use TransE, because the experimental results [10] show that TransR and TransH cause a significant runtime overhead, with TransR needing over twice as much time for its training as TransE.

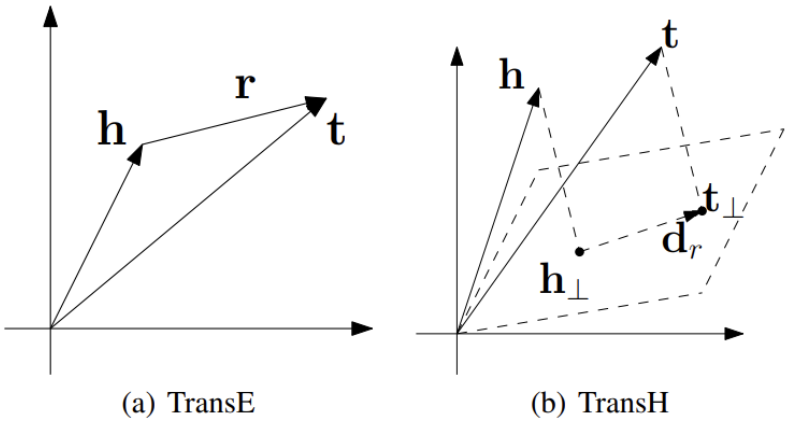

Figure 4: Comparison of the TransE and TransH embedding procedure. TransH employs an additional projection to a hyperplane before the addition of the relation vector. [19]

The second method for anomaly detection that we propose is described in Section 5.2.3. Compared to the first one, we decided to use a more novel approach by taking advantage of the graph structure of the problem and applying GNNs. The difference between a GNN and a standard artificial neural network is in the architecture that is enriched with graph layers. These layers harness the graph connectivity for information passing between the nodes and edges. By aggregating the information from the node's neighborhood, it constructs a hidden state, which is passed to a subsequent readout phase, where it is converted to the output based on the task, e.g. by a small multi-layer perceptron (MLP). We decided to use the GraphSAGE [5] model, which is adapted for working with large graphs by performing sampling of the node's neighbors that contribute to the computation of its hidden state. Figure 5 shows the neighborhood sampling and message aggregation process.

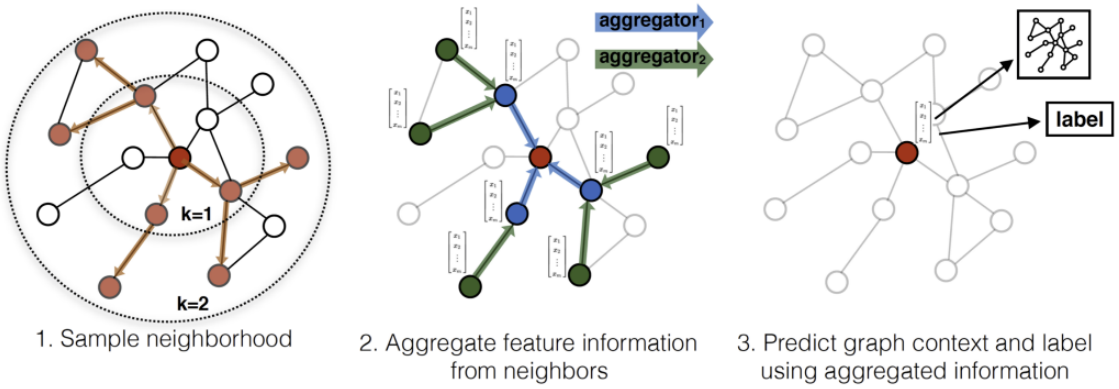

Figure 5: Illustration of the GraphSAGE neighborhood sampling and information aggregation for the node embedding computation. [5]

4.3 Implementation

For the implementation of our anomaly detection pipeline, we decided to use Python as a programming language, as it provides a lot of suitable machine learning libraries. We used the NetworkX [8] library for the provenance graph construction and representation. The attribute embeddings were computed by the Word2Vec model from the Gensim [2] library trained for 20 epochs on the node data.

In our first set of experiments, the graph embeddings were produced by the TransE algorithm trained for 30 epochs using the SGD optimizer with the learning rate set to 0.01. We conducted experiments with and without Word2Vec embedding of the nodes, as TransE is able to start by random initialization of the node embeddings. We tried the batch size of 10, 100, and 1000 for the gradient computation. We also tried a variable embedding length of 20 and 30 dimensions. By running the validation phase after every epoch, we chose the best performing model based on the validation loss.

Our GNN approach was developed using the PyTorch [3] and PyTorch Geometric [4] libraries for the underlying computations. We decided to use the GraphSAGE [5] model with 2 graph layers and dropout probability of 0.5. The nodes and the edges are represented as 32-dimensional vectors that serve as the input for the GNN. The model produces node embeddings, which are forwarded to a 2-layer MLP that uses Scaled Exponential Linear Unit (SELU) activation function at the first layer and the Sigmoid function at the ultimate layer to output the link probability. We also apply the dropout of 0.5 before the final layer. The model performs the backpropagation using the Binary Cross Entropy loss and the Adam optimizer with the fixed learning rate of 0.01. We split the provenance graph into a training and a validation subgraph, comprising 80% and 20% of the original graph, respectively. To sample the input neighborhoods, we create a batch of 128 positive and 128 negative edges with their neighbors. We consider 50 one-hop neighbors and 50 two-hop neighbors of the sampled edges to create the subgraph of the batch. The negative edges are constructed by replacing the destination nodes of the positive edges by random nodes from the graph.

We were initially trying to implement the GNN ourselves, without the use of PyTorch Geometric, to have better control over the training execution. However, we faced a lot of issues, as the training was slow, because of the inefficient implementation of the neighborhood sampling. At first, we were considering all the neighbors of an edge, which turned out to be infeasible in regards to the space complexity. Moreover, we used the NetworkX representation of the graph for the creation of the neighborhoods instead of matrix representation, which provides faster checks of adjacency of edges. Therefore, as the provenance graphs tend to be huge, we had issues with the scalability of our solution. In the end, we decided to get use of the PyTorch Geometric library, which contains built-in objects facilitating the network training.

5 Experiments

In this chapter, we present our proposed solution along with the obtained results. Section 5.1 shows the anomaly detection pipeline and describes the individual components that comprise it. We continue by the comparison of two evaluated approaches for the graph embeddings computation, namely TransE and GNNs, in Section 5.2. We conclude the chapter by showing the results of our anomaly detection process along with the analysis of the data in Section 5.3.

5.1 Anomaly Detection Pipeline

To look for anomalies in the system execution, we defined a processing pipeline shown in Figure 6.

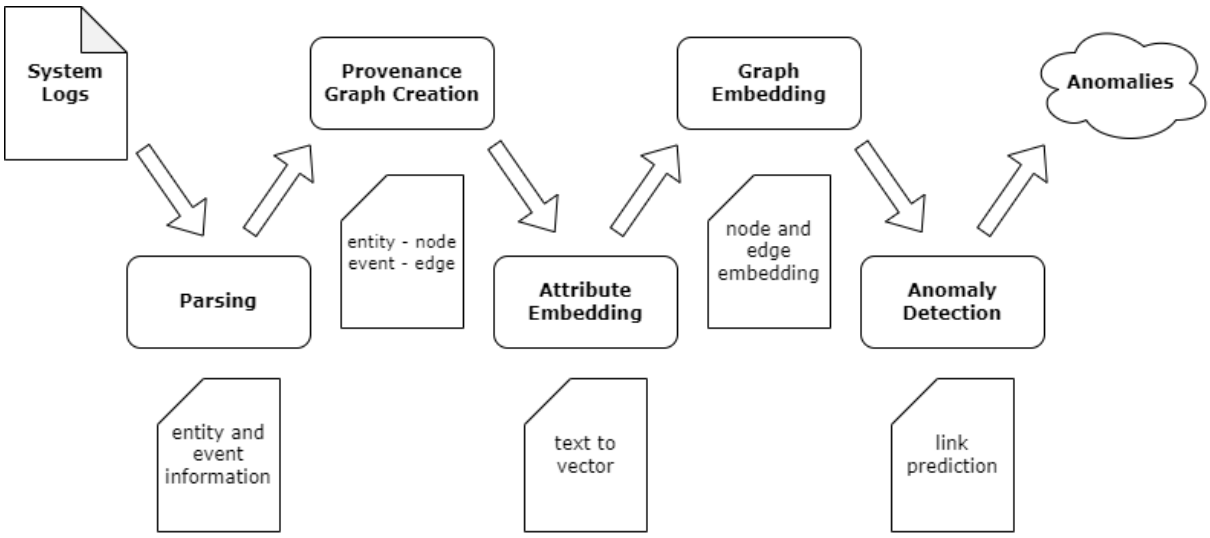

Figure 6: Overview of the anomaly detection pipeline we propose. The system logs are parsed to create a provenance graph, which is then projected to the embedding space for the subsequent anomaly detection.

The input of our anomaly detection process are the system logs created by the system's audit agent. As mentioned before, we assume that the integrity of the logging can be guaranteed and the log files are considered intact. The first step to process them is by parsing the files by extracting the important information about the entities and the performed actions. After this extraction, we build a provenance graph capturing the system operation, which is then used as the input for the anomaly detection. To detect anomalies in the graph, we convert it to a numerical representation by transforming its nodes and edges to vectors capturing their attributes. We first train the Word2Vec model to capture all the node attributes, as some of them are represented in the textual form, by creating a single vector for each node, while the edge vectors are defined representing only the edge type. We have evaluated two different approaches, which are used for the embedding of the graph, for which they utilize the entity interactions. The first tested alternative

is TransE, which computes both the node and edge embeddings based on the connectivity of the graph. The other approach is using a GNN, which produces just the node embeddings, while the edges remain to be represented by their event types. The anomaly detection then corresponds to a link prediction task, as we expect the anomalous behaviors to be represented by unlikely connectivity in the graph. The individual modules are described in more detail below.

5.1.1 Logs Parsing

As described in Section 4.1, the dataset consists of multiple files, some of them capturing only benign execution, while some contain also records related to an attack in progress. These files are saved in JSON format, every line corresponding to an event or entity description. To parse a file, we iterate over the records and extract the important information. As for the events, we save their types and the IDs of the entities that were involved in each event. When parsing the records containing the entity information, we consider different attributes depending on the entity type. There are 7 entity types in the CADETS dataset, namely files, processes, system sensors, netflow objects, pipes, users, and hosts. All entities contain the information about the host they reside on. Apart from that, for files we store only their type (e.g. text file, directory, or PE file). Processes are defined by the process type (process / thread), the parent process from which they were created, and the user that owns the process. For sensors we consider only the sensor type, when parsing the information. For each netflow object we store the local IP address, local port, remote address, and remote port. Pipes are described by the entities at their ends, their sources and destinations. User entities have associated UID, username, and the group IDs of the groups the user belongs to. As the system has only a single host that collects the logs, there is only one host record in the dataset, for which we store its hostname, type (whether it is a desktop or a server), and the IP addresses on its interfaces. Having all these information, we save them to two distinct files, one containing the events, and the other the entities.

5.1.2 Provenance Graph Creation

To create the provenance graph of the system operation, we take the parsed information and encode them into the graph's structure. First, we create the nodes, each storing the parsed information into its attributes. Then we derive the connectivity of the graph based on the events that were recorded in the logs. Each event corresponds to an edge in the graph, connecting the entities that were involved in its execution. The log files contain up to three types of entities in regards to an event. The subject is the entity that performs the action and, therefore, is represented as the source node of the edge. The destination node is identified based on the object of the action, which is specified in the record. However, some actions have two objects, as illustrated by the example in Figure 7. Therefore, in such cases, we create another edge representing the relationship. We connect the objects themselves, to demonstrate the connection between them, and assign the edge a special type otherwise not present in the dataset. Another special case is when the action does not have any objects. An example of such situation is when a user logs in, as such event does not entail any other entity than the user. In these cases, we decided to create a loop in the

graph, i.e. an edge with the same node as its source and destination. As some of the node information are not known at the time of the creation of an entity, certain node attributes are missing in the entity records and stored only in the event records. Therefore, when we encounter an event containing previously unknown node information, we add it to the node attributes to complete them. This way, we append the attribute corresponding to the filepath to the file nodes, and the attribute denoting the executed command on the command line for processes. After following this process, we have a provenance graph capturing all the events that are recorded in the logs.

Figure 7: Example of an event with two objects. A malicious process copies the */etc/shadow* file to the */tmp* directory. Therefore we create an edge between the process *malw.exe* and the */etc/shadow* file, and another one of a special type connecting */etc/shadow* with */tmp/aBcD*.

5.1.3 Attribute Embedding

The next step in the logs processing pipeline is to embed the node attributes. This process ensures that the node attributes that are in the textual form are converted to a numerical representation. We decided to use the Word2Vec model to embed all the node attributes using a positional encoder as proposed in [9]. The positional encoder ensures that the individual attributes maintain the same order for each entity during the embedding, because Word2Vec lacks a built-in concept of sequential ordering of the embedded words. The obtained 32-dimensional vectors reflect all the node attributes and capture the similarity between the nodes. For the embedding computation we use all the available attributes for files, processes, network objects and pipes. We decided to consider only the host and username for users, and name and type for hosts. We do not use any sensor information for their embedding, as these entities are not relevant for the attack, and therefore are assigned a vector of zeros. The embedding of each node is then stored as a node attribute for later use.

5.2 Graph Embedding

To project the graph to an embedding space, we consider all the interactions between the nodes and model the benign behavior of the system. We evaluated two distinct approaches for graph embedding and subsequent anomaly detection. The advantage of TransE consists in its mathematical properties, which are utilizable for detecting anomalies. The other approach uses a GNN for computing node embeddings, and a 2-layer MLP for the link prediction.

5.2.1 TransE

We decided to try to use TransE for the computation of the embeddings, as it uses the relationships between the entities to create their vector representations, which corresponds to the behavior and interactions of the nodes. The desired locations of the nodes are computed using the concept of golden triplets, as explained in Section 4.2. By minimizing the distances from the desired locations of the nodes of the golden triplets, the model should capture the interactions between them. As the model is trained using benign data, the produced embeddings model the normal behavior of the system. In case of an anomaly, the *tail* of the anomalous edge should have higher distance from $head + relation$ than in the benign cases, as it exhibits behavior that collides with the structure of the benign interactions. Therefore, the *tail* of a golden triplet that has a higher distance from $head + relation$ than a certain threshold is considered to be suspicious, based on its relationship with other entities, because it was not embedded well to satisfy the constraints given by its interactions. This signalizes that its behavior was strange, considering all the other entity interactions in the graph that were used to create the embeddings. Therefore, these suspicious nodes need to be analyzed more in order to decide whether they pose a risk to the system.

5.2.2 TransE Link Prediction

To evaluate the performance of TransE for the anomaly detection, we tested the trained model on the data containing also the attack traces. We show the results of our TransE experiments in Table 1 below. We decided to report the results using the Hits@10 metric, which measures the fraction of times when the real link is ranked in the top 10 predicted links. For a golden triplet $(head, relation, tail)$, we replace its *head* by all the entities in the graph and compute the distance of $head + relation$ to the real *tail*. Then we repeat the same process by replacing the *tail* of the triplet and computing the distances of $head + relation$ to the replaced *tail*. By ranking these distances, we check whether the distance of the golden triplet is among the 10 shortest, meaning the link is among the 10 most likely links entailing these entities.

Pretraining	Dimensionality	Batch size	Epochs	Hits@10
No	20	1000	20	24.28
No	30	1000	20	32.42
No	20	1000	7	21.7
No	20	100	17	26.12
No	**20**	**10**	**20**	**36.9**
Yes	30	1000	5	17.84
Yes	30	10	4	12.71

Table 1: The obtained results of experiments using various TransE model parameters.

The results show that the model performs better without the Word2Vec pretraining, just with a random initialization of the node and edge vectors. The best results were obtained

by embedding the nodes and edges as 20-dimensional vectors, using 10 links as the batch size, and training for 20 epochs. However, even the best model was able to predict the link among the top 10 only in 36.9% cases. Even though these results seem to be quite poor, this approach suffers from false negatives, as part of the ranked edges are actually also present in the graph. However, we have computed this metric also independently for the benign and malicious edges, and found out that the malicious ones do not exhibit lower Hits@10 score than the benign ones.

5.2.3 GNNs

To detect anomalies using a GNN, we train it to predict the existence of edges based on their neighborhoods by sampling small subgraphs from the big provenance graph. The input of the GNN are the edge attributes of the sampled edges, the node vectors of the corresponding nodes, and the graph connectivity. Based on this information, the GNN produces node embeddings, which are then fed into a 2-layer MLP, serving as a link predictor. Utilizing the sigmoid function at the last layer of the MLP, we predict the probability of the edge. For the training, we split the benign provenance graph into a training and a validation subgraph. We train the network in a semi-supervised way, by randomly adding negative (non-existent) edges by sampling random destination nodes to the source nodes of positive (existing) edges. The network is trained to perform binary classification, as the negative edges are assigned label 0 and the positive ones label 1. By training the model on the benign data, the model learns the structure of the benign entity neighborhoods. In the testing, the anomalous edges should resemble the negative edges from the training and, therefore, produce lower probabilities than the benign ones. By setting a probability threshold, we analyze only the edges that were misclassified, i.e. real edges that are predicted to be unlikely in the graph.

5.2.4 GNN Link Prediction

The advantage of using a GNN followed by an MLP for the link prediction is that the model directly outputs the link probability. This approach simplifies the anomaly detection, because as anomalous are denoted the links of the graph with low probability of existence. Figure 8 shows the training and validation loss and accuracy over the epochs.

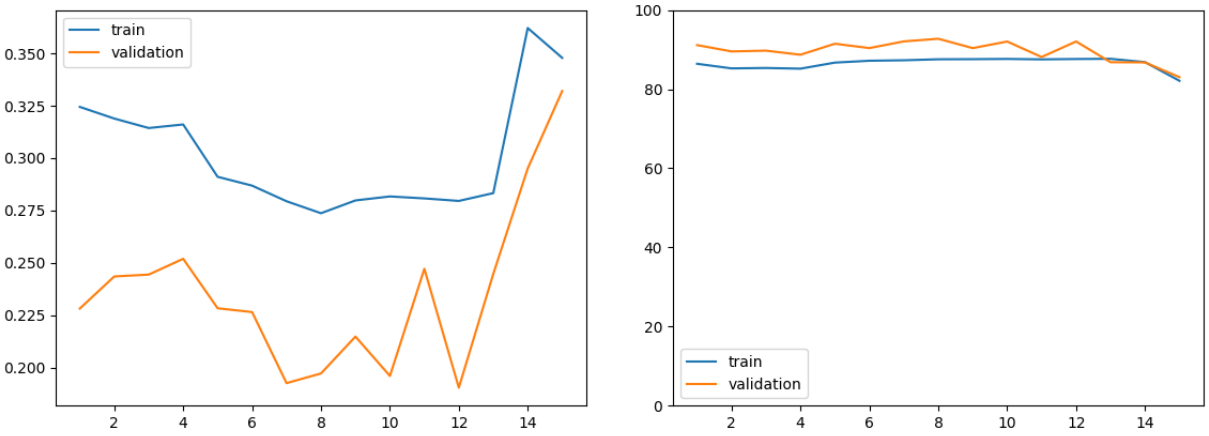

Figure 8: GNN link prediction results over the epochs. The left figure presents the training and validation loss. The figure on the right shows the accuracy of the link prediction as a binary classification task.

The network exhibits signs of learning, therefore, we decided to take the model after the 7th epoch and evaluate it on the test set containing both benign and attack-related edges. In the test setting, we present the network only with the real edges of the graph and expect it to classify all of them as such. Considering 0.5 as the probability threshold, the model was able to predict 96.35% of the edges.

5.3 Anomaly Detection

To perform the anomaly detection, we look for the edges of the provenance graph that were misclassified as negative. As mentioned before, using the probability threshold of 0.5, we were able to predict almost all the edges of the graph. Figure 9 shows the misclassified edges by their corresponding event types. As we can see, most often are misclassified the edges of the `EVENT_EXIT`, `EVENT_BIND`, and `EVENT_PREDICATEOBJECT2` types. We created the last mentioned type to represent the relationship between the entities of an event that contained 2 objects. Therefore, the results show that this representation highly resembles the random destination node sampling of the negative edges. This also aligns with the intuitive conception of this relationship. Using the illustration example presented in Figure 7, the file */tmp/aBcD* presents an unpredictable destination of the *copy* event. Moreover, this event type does not correspond to a real action in the logs, only maintains the connection between the related entities. Therefore, putting the edges of this type apart, we can report an adjusted link prediction accuracy of 98.68%.

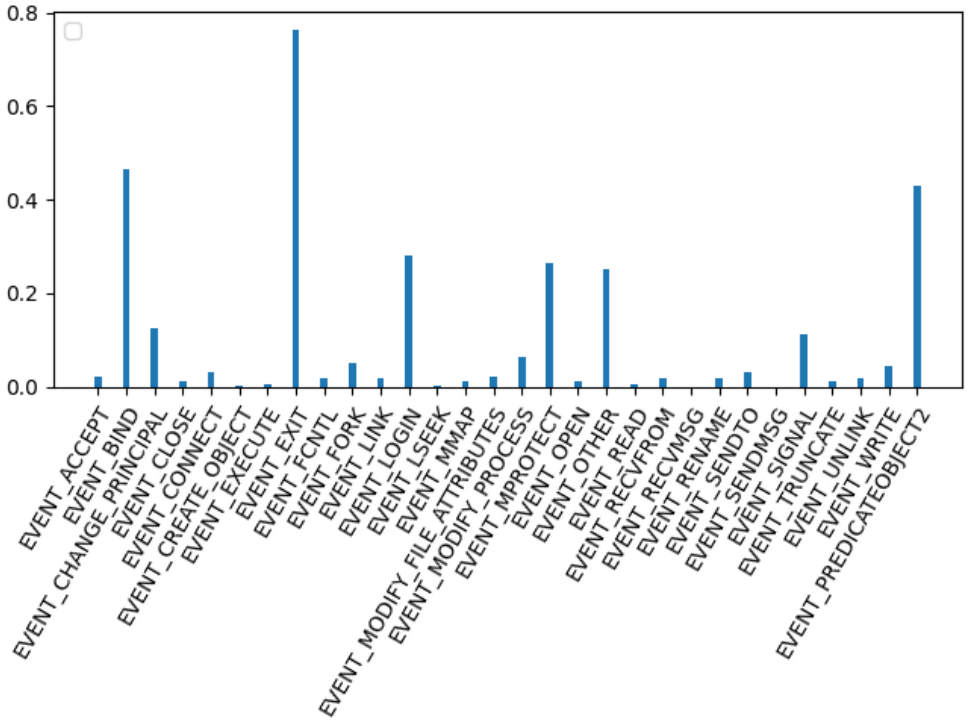

Figure 9: The misclassified edges by the event type. The value corresponds to the ratio of the number of the misclassified edges of the given category to the total number of edges of the category in the dataset.

To analyze the anomalous edges, we wanted to investigate whether they involve the malicious entities. However, as our dataset does not contain labels, but only the description of the attack scenarios and its components, we manually identified 20 entities related to the performed attacks in the logs. We found out that our model was able to detect the interactions of 9 of these entities.

Furthermore, we decided to analyze the effect of the existence probability threshold on the link prediction accuracy and malicious entity discovery. Table 2 shows the relationship between the link prediction accuracy and the probability threshold. We observe a drop to 77.62% by raising the threshold to 0.9, classifying as real only the more likely edges. The accuracy can be adjusted to 80.19% by removing the `EVENT_PREDICATEOBJECT2` edges.

An inverse relationship is observed by comparing the malicious entity discovery for multiple existence probability thresholds. By using the existence probability threshold of 0.9, we are able to detect 17 out of the 20 manually labeled entities. All the 3 undetected entities correspond to unsuccessful steps in the attack execution and have therefore very few interactions with other entities in the graph.

Probability threshold	Link prediction accuracy	# discovered entities
0.5	96.35%	9
0.6	94.84%	10
0.7	89.05%	11
0.8	80.84%	12
0.9	77.62%	17

Table 2: Link prediction accuracy and the number of the discovered malicious entities (out of 20) for various probability thresholds.

Therefore, we conclude that the proposed solution has potential for logs anomaly detection. We observe that different link probability thresholds influence both the link prediction accuracy and the malicious entity discovery. By not predicting over the edges of the `EVENT_PREDICATEOBJECT2` type, we are able to compute the adjusted link prediction accuracy.

6 Summary

Storing logs is important for detecting and responding to active breaches. However, as the log files tend to be large and complex, performing a manual analysis of the system behavior is often infeasible. Therefore, many works have been presented to address this issue by proposing approaches to automatic attack detecion.

We contributed to the research by proposing a novel pipeline for logs processing and subsequent behavior anomaly detection. We take advantage of the graph structure of the problem and represent the system execution as a provenance graph by utilizing the information stored in system logs. We employ a GNN-based model trained to predict missing links in the graph, and assess the likelihood of the links corresponding to the recorded events. This GNN training technique has not been previously applied to this problem. Based on the conducted experiments, we can conclude that this approach is able to discover malicious entities present in the system.

6.1 Future Work

To further enhance the performance of our approach, we would suggest to implement different techniques for sampling of negative edges. For example, by taking into account the types of the entities that are usually connected by a certain type of event and the distribution of the events in the data, we could create more realistic negative edges during the training. This should create more realistic neighborhood structures and lead to better perception of more subtle behavior deviations.

Another proposition would be to replace the MLP predictor by another light-weight binary classifier to determine the best alternative for this use case.

Additionally, we would propose to implement other baselines to further evaluate and compare our model's capabilities.

References

[1] Michael Zipperle, Florian Gottwalt, Elizabeth Chang, and Tharam Dillon. Provenance-based intrusion detection systems: A survey. *ACM Comput. Surv.*, 55(7), dec 2022.

[2] Radim Rehurek and Petr Sojka. Gensim–python framework for vector space modelling. *NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic*, 3(2), 2011.

[3] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[4] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric, 2019.

[5] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs, 2018.

[6] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[7] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013.

[8] Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[9] M. Rehman, H. Ahmadi, and W. Hassan. Flash: A comprehensive approach to intrusion detection via provenance graph representation learning. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 142–142, Los Alamitos, CA, USA, may 2024. IEEE Computer Society.

[10] Jun Zeng, Zheng Leong Chua, Yinfang Chen, Kaihang Ji, Zhenkai Liang, and Jian Mao. Watson: Abstracting behaviors from audit logs via aggregation of contextual semantics. *Proceedings 2021 Network and Distributed System Security Symposium*, 2021.

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

[12] Yutao Tang, Ding Li, Zhichun Li, Mu Zhang, Kangkook Jee, Xusheng Xiao, Zhenyu Wu, Junghwan Rhee, Fengyuan Xu, and Qun Li. Nodemerge: Template based efficient data reduction for big-data causality analysis. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 1324–1337, New York, NY, USA, 2018. Association for Computing Machinery.

[13] Wajih Hassan, Mark Lemay, Nuraini Aguse, Adam Bates, and Thomas Moyer. Towards scalable cluster auditing through grammatical inference over provenance graphs. 01 2018.

[14] Muhammad Adil Inam, Yinfang Chen, Akul Goyal, Jason Liu, Jaron Mink, Noor Michael, Sneha Gaur, Adam Bates, and Wajih Ul Hassan. Sok: History is a vast early warning system: Auditing the provenance of system intrusions. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 2620–2638, 2023.

[15] Muhammad Adil Inam, Akul Goyal, Jason Liu, Jaron Mink, Noor Michael, Sneha Gaur, Adam Bates, and Wajih Ul Hassan. Faust: Striking a bargain between forensic auditing's security and throughput. In *Proceedings of the 38th Annual Computer Security Applications Conference*, ACSAC '22, page 813–826, New York, NY, USA, 2022. Association for Computing Machinery.

[16] Kyu Hyung Lee, Xiangyu Zhang, and Dongyan Xu. Loggc: garbage collecting audit log. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, page 1005–1016, New York, NY, USA, 2013. Association for Computing Machinery.

[17] Zhang Xu, Zhenyu Wu, Zhichun Li, Kangkook Jee, Junghwan Rhee, Xusheng Xiao, Fengyuan Xu, Haining Wang, and Guofei Jiang. High fidelity data reduction for big data security dependency analyses. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 504–516, New York, NY, USA, 2016. Association for Computing Machinery.

[18] Md Nahid Hossain, Junao Wang, R. Sekar, and Scott D. Stoller. Dependence-Preserving data compaction for scalable forensic analysis. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1723–1740, Baltimore, MD, August 2018. USENIX Association.

[19] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph embedding by translating on hyperplanes. *Proceedings of the AAAI Conference on Artificial Intelligence*, 28(1), Jun. 2014.

[20] Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. Learning entity and relation embeddings for knowledge graph completion. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, page 2181–2187. AAAI Press, 2015.

[21] DARPA engagement 3. *https://github.com/darpa-i2o/Transparent-Computing/blob/master/README-E3.md*.

[22] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Yihong Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation. *CoRR*, abs/1902.07243, 2019.

Appendices

Appendix A

We attach the source code containing the implementation of the models.

File *parse_dataset.py* contains the functions for parsing the DARPA CADETS dataset.

File *AA_create_graph.py* includes the provenance graph creation along with the Word2Vec attribute embedding.

Files *data.py*, *metric.py*, *model.py*, and *storage.py* contain utilities for the TransE training.

File *train_test_transE.py* provides the functionality of the TransE training and evaluation.

File *gnn_graphSAGE.py* serves as the GNN training module.

File *test_gnn_graphSAGE.py* contains the evaluation functions of the GNN model.