ProvG-Searcher: A Graph Representation Learning Approach for Efficient Provenance Graph Search

Enes Altinisik
ealtinisik@hbku.edu.qa
Qatar Computing Research Institute
HBKU, Qatar

Fatih Deniz
fdeniz@hbku.edu.qa
Qatar Computing Research Institute
HBKU, Qatar

Hüsrev Taha Sencar
hsencar@hbku.edu.qa
Qatar Computing Research Institute
HBKU, Qatar

ABSTRACT

We present ProvG-Searcher, a novel approach for detecting known APT behaviors within system security logs. Our approach leverages provenance graphs, a comprehensive graph representation of event logs, to capture and depict data provenance relations by mapping system entities as nodes and their interactions as edges. We formulate the task of searching provenance graphs as a subgraph matching problem and employ a graph representation learning method. The central component of our search methodology involves embedding of subgraphs in a vector space where subgraph relationships can be directly evaluated. We achieve this through the use of order embeddings that simplify subgraph matching to straightforward comparisons between a query and precomputed subgraph representations. To address challenges posed by the size and complexity of provenance graphs, we propose a graph partitioning scheme and a behavior-preserving graph reduction method. Overall, our technique offers significant computational efficiency, allowing most of the search computation to be performed offline while incorporating a lightweight comparison step during query execution. Experimental results on standard datasets demonstrate that ProvG-Searcher achieves superior performance, with an accuracy exceeding 99% in detecting query behaviors and a false positive rate of approximately 0.02%, outperforming other approaches.

CCS CONCEPTS

• **Security and privacy → Intrusion detection systems**; *Systems security*; *Intrusion/anomaly detection and malware mitigation.*

KEYWORDS

threat hunting, APT behaviors, security logs, provenance graph, graph reduction, subgraph matching, graph neural networks, order embeddings, graph entailment

ACM Reference Format:
Enes Altinisik, Fatih Deniz, and Hüsrev Taha Sencar. 2023. ProvG-Searcher: A Graph Representation Learning Approach for Efficient Provenance Graph Search. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS '23), November 26–30, 2023, Copenhagen, Denmark*. ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3576915.3623187

CCS '23, November 26–30, 2023, Copenhagen, Denmark

ACM ISBN 979-8-4007-0050-7/23/11…$15.00
https://doi.org/10.1145/3576915.3623187

1 INTRODUCTION

Causality analysis and provenance graphs have emerged as crucial tools for understanding and mitigating risks associated with cyber attacks targeting computer systems. A provenance graph is a holistic representation of kernel audit logs, describing interactions between system entities [53, 54]. By interconnecting isolated system events, provenance graphs offer two indispensable capabilities for security analysis. First, they facilitate identifying causal relationships and tracing data lineage, thereby providing an enriched context for discerning the nature of system events. Second, they enable the application of advanced graph algorithms in audit log analysis. Consequently, provenance graph analysis has been extensively employed in the detection of anomalous system activities [15, 20, 39, 41, 49, 69, 71, 83]; root-cause analysis and forensic tracking [16, 24, 28, 32, 43]; attack story generation [1, 26, 27, 56]; and supporting alert validation and investigation [22, 23, 52, 75].

Another domain that can greatly benefit from using provenance graphs is the efficient search for known attack behaviors within vast repositories of historical system logs. This is an under-studied problem with crucial implications for the practice of threat hunting. Due to the increasing complexity and volume of cyber attacks, relying solely on detection tools available within organizations is no longer feasible for in-house security teams. Consequently, threat hunters must continuously scan descriptions of new threat behaviors provided by threat intelligence sources and operate under the assumption that the same attackers may also bypass their organization's security controls.

Consider a scenario where a threat hunter discovers news of a new attack targeting an organization operating in the same industry as theirs. In this case, the appropriate course of action is to hypothesize that the attackers may have already infiltrated their systems and to search for traces of an ongoing intrusion in their system logs. In the context of provenance analysis, this necessitates converting an externally observed threat behavior into a query that can be searched within system-level provenance graphs [51, 59, 72, 85]. This problem setting can indeed be viewed as an instance of the graph entailment (subgraph matching) problem. To better illustrate our use case, we present an example scenario in Fig. 1. The scenario involves a threat intelligence report [65], which describes how an adversary compromises a `nginx` web server by downloading a malicious payload and executing to gain root privileges. A threat hunter leverages this information to create a query graph $\mathcal{G}_Q$, where each type of system entity is represented with a distinct shape and color. The objective is to search the extensive system-level provenance graph $\mathcal{G}$ and identify nodes that match the query graph in terms

An extended version of this work, offering additional results and analyses, is available in [14].

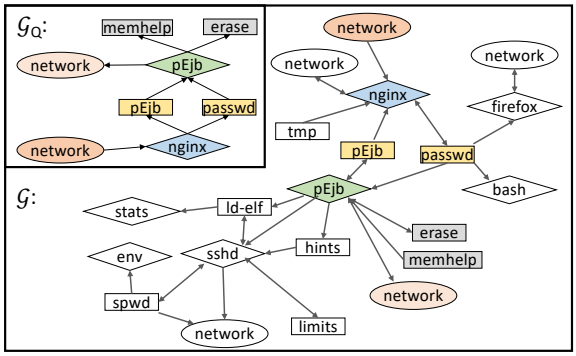

Figure 1: Illustration of the search for threat behaviors in the system-level provenance graph as a subgraph matching problem. $\mathcal{G}_Q$ represents the query graph generated from the described interactions in a threat intelligence text, and $\mathcal{G}$ displays part of the system-level provenance graph centered around the process that executed the malicious code. Matching nodes are colored with the same colors, while unrelated nodes are uncolored or omitted for clarity.

of both color and connection pattern. In this example, the larger subgraph shows where such an alignment is possible, and $\mathcal{G}_Q$ can be confirmed as a subgraph of $\mathcal{G}$, enabling the threat hunter to infer that the described threat behavior is present within that system.

In this work, we propose a solution to this problem and develop a system named ProvG-Searcher, which can efficiently and accurately identify matching subgraphs within a large provenance graph corresponding to a given query graph.The subgraph matching problem involves determining whether a query graph is isomorphic to a subgraph of a larger graph both structurally and in its key features. This is a fundamental problem in graph analysis with significant practical implications across a broad range of application domains. However, exact subgraph matching is known to be NP-complete. Thus, exact methods based on combinatorial approaches are unable to solve large-scale instances. To perform subgraph matching within a reasonable time frame, several approximate solutions have been proposed. Earlier inexact subgraph matching methods typically relied on heuristics to discern better alignment of nodes [40, 42] or translated the discrete optimization problem into a continuous one [46, 63, 64]. Recently, graph neural networks (GNNs) have achieved significant success in graph representation learning [8]. This led to the development of several learning-based methods for approximate subgraph matching with superior performance [5, 35, 38, 44, 58]. At the core of these methods is the learning of an embedding function that maps each graph into an embedding vector encapsulating its key features. The subgraph relation between two graph embeddings is then evaluated in this continuous vector space.

In the realm of provenance graph analysis, approximate subgraph matching methods encounter distinct challenges absent in other domains where they have been successfully implemented [12, 73, 77]. (More details on the characteristics of graph datasets used in this work are presented in [14].) Provenance graphs are characterized by a large number of nodes and edges, as well as a high average node degree, due to the diverse activities inherent within a typical computing system. This results in a considerable computational burden when searching behaviors and learning graph relationships. Moreover, the coarse-grained nature of logs hinders precise tracking of information and control flows among system entities, leading to erroneous connections between nodes. These factors render search methods based on node alignment between graphs largely impractical. Applying learning-based methods, based on GNNs, to large graphs introduces further complications. GNNs carry out computation through a series of message-passing iterations, during which nodes gather data from neighboring nodes to update their own information. The updated information of all nodes is then pooled together to create a graph-level representation. In this context, increasing the model depth beyond a few layers (i.e., the number of iterations) to more effectively capture relationships results in an exponential expansion of a GNN's receptive field, which consequently leads to diminished expressivity due to oversmoothing [45, 80].

To improve the efficiency of provenance graph analysis, several methods have been proposed for simplifying provenance graphs, including entity pruning [1, 22, 37, 79], removal of redundant semantics [15, 28, 50, 79, 84], behavior abstraction [21, 78, 82] and dependence explosion mitigation [25, 27, 36, 47]. These data reduction methods primarily focus on identifying anomalous interactions and preserving forensic tractability. Nevertheless, they often do not meet the objective of preserving sufficient integrity to support the search for more general graph patterns.

Previous Hypothesis-Driven Threat Hunting Techniques and Limitations: Enhancing threat hunting capabilities necessitates tackling two distinct challenges: effective search and query generation. Milajerdi et al. [51] focused on the search aspect by proposing a non-learning-based method, called Poirot, to facilitate the search of provenance graphs using known APT attack behaviors as query graphs. To reduce search complexity, they adopted a heuristic that assumes an attacker's limited ability to exploit multiple vulnerabilities, allowing them to exclude graph paths without attack nodes. A significant drawback of this approach is that the entire search computation must be performed at the query time. In situations where the nodes of the query graph are commonly present in the target provenance graph, the search must consider numerous potential alignments, which may result in covering a large portion of the graph with each query.

In a similar vein, Wei et al. proposed DeepHunter in [72], a learning-based subgraph matching technique that leverages Neural Tensor Networks (NTN) to model the subgraph relationship between two graph-level embeddings and to calculate a matching score. Their method identifies subgraphs surrounding indicators of compromise (IoCs) related to the query, effectively using the query as a filter to reduce the provenance graph. A match is determined by exhaustively comparing embedding similarities through NTN's pairwise comparisons between the query embedding and

https://en.wikipedia.org/wiki/Subgraph_isomorphism_problem

In the cybersecurity context, graph representation learning has already demonstrated notable advancements and widespread application, particularly in the domain of vulnerability detection [7, 13].

sampled subgraph embeddings. The efficiency of this method declines due to its dependency on the query, particularly as the size of the provenance graph increases.

The other aspect of hypothesis-driven threat hunting involves creating queries that correspond to specific behaviors to be investigated. In contrast to the methods used in studies [51, 72], which relied on manually generated query graphs from incident reports, Zong et al. [85] tackled this as a query discovery problem. They developed a discriminative subgraph pattern mining technique to generate query templates automatically. Alternatively, Satvat et al. [59] employed a natural language processing pipeline to identify relationships between system entities in threat intelligence reports, ultimately extracting query graphs using this information. We focus on the first aspect of the problem, assuming that queries are already available.

Approach Overview and Summary of Contributions: The design of ProvG-Searcher addresses the challenge of efficiently processing numerous queries within extensive provenance graph repositories. As a departure from earlier proposed hypothesis-driven threat hunting methods, our technique leverages a graph representation learning approach that enable the bulk of search computation to be conducted offline, accompanied by a lightweight comparison step performed during query time. To effectively learn the diverse range of relations in a provenance graph and mitigate the complexities introduced by its size, we conduct a process-centric partitioning of provenance graphs. Additionally, we implement a behavior-preserving graph reduction method, which incorporates graph versioning to integrate timing information. The central component of our search methodology is the embedding of subgraphs in a vector space, where subgraph relationships can be directly evaluated. To achieve this, we employ order embeddings, which allow learning hierarchical entity representations while maintaining the hierarchical structure through the coordinate-wise ordering of these representations in the embedding space. Consequently, ProvG-Searcher reduces subgraph matching to simple comparisons between query and precomputed subgraph embeddings. This approach eliminates the necessity for exhaustive computation of all pairwise interactions between query and target subgraph embeddings. Overall, our study offers significant contributions in the following areas:

- A graph simplification strategy that preserves the diverse range of behaviors present in a provenance graph, thereby fostering effective learning and search capabilities.
- The use of order embeddings to facilitate the efficient evaluation of subgraph relationships in the embedding space.
- A versatile search methodology that is not exclusively biased towards attack behavior but can be generalized to identify any type of behavior.
- The ability to efficiently search vast quantities of historical log data, owing to the compactness of subgraph representations and the simplification of provenance graphs.
- A substantial decrease in false-matching rates and improved accuracy compared to other hypothesis-driven threat hunting approaches.

The source code and trained models associated with our study are accessible online: https://github.com/EnesAltinisik/ProvG-Searcher.

2 CHALLENGES IN LEARNING TO SEARCH PROVENANCE GRAPHS

The strength of the provenance graph lies in what it reveals about contextual relationships between system events. By mapping the recorded interactions in individual audit log events onto a graph that shows the chronological interplay between processes and other system resources, such as files, network sockets, memory, and registry objects, in the form of system calls, they allow for the use of graph analysis techniques. In this regard, a provenance graph is a heterogeneous, typed, directional, and dynamic graph that provides a coarse-grained insight into system state changes. This rich representation, however, also poses a number of key challenges for graph-learning methods.

Challenge #1: Size of Graphs. Many system events are recurrent, in mundane nature, and affect a large number of system objects such as system and software updates, backup, and data synchronization jobs. As a result, audit logs collected from a typical machine may easily yield provenance graphs with a very large number of nodes and edges over shorter durations of time [74]. The expressive power of a GNN is determined by its capacity, which is generally expressed in terms of the width and depth of a neural network, i.e., the embedding size and the number of layers of a GNN, respectively. Loukas [45] studied the difficulty of well-known subgraph analysis tasks and determined that even verifying whether a subgraph meets a given property requires the product of a GNN's depth and width higher than a (low-order) polynomial of the size of a graph. This lower bound implies that for large query and provenance graphs learning subgraph relations will indeed be difficult. Further, for large graphs increasing the model depth, i.e., the number of layers in a GNN, often translates to exponential expansion in the number of neighboring nodes. This implies that the scope (i.e., receptive field) of a GNN has to be restricted through methods such as subgraph-based sampling [10, 81] when learning subgraph relationships.

Challenge #2: High-Degree Nodes. The average degree of a node in the provenance graph can be quite high. For example, the IP node of a DNS server may have incoming and outgoing node degrees that easily exceed tens of thousands, while a /dev/null device might have incoming node degrees reaching several thousands. This is partly due to the dependence explosion problem, where long-running processes interacting with many other system entities appear as highly connected nodes on the graph. The computation in GNNs is performed through several message-passing iterations in which nodes aggregate information from adjacent neighbors to update their information. For high-degree nodes, this aggregation step is likely to suppress useful characteristics. It will result in degraded expressivity due to the well-known over-smoothing behavior where node embeddings become uninformative after several rounds of message-passing [30]. This is especially a concern for system behaviors that occur infrequently. This indeed requires adopting a representation that exhibits system behaviors in a balanced manner.

Challenge #3: Preserving Time Order of Events. Edges in a provenance graph represent a time-ordered sequence of events. Disregarding the timing information essentially introduces spurious information flows among system entities. The computation in

GNNs includes the creation of computation graphs rooted at each node describing the structure for message-passing and aggregation. The timing information can be incorporated during graph creation by enforcing a time order going from root to leaves. In practice, however, a recursive neighborhood aggregation scheme is used to avoid the computational overhead of repeated creation of these computation graphs. This scheme cannot preserve the causality relation between edges as all nodes concurrently aggregate information from their neighbors. In addition, it needs to be considered that a queried behavior may not explicitly include the timing information between depicted events. Therefore, timing information must be utilized so that subgraph relationships do not explicitly depend on it.

Challenge #4: Semantic Gap with Query. Another challenge is the potential mismatch in the degree of expressiveness between the queried behavior and its observed version on the provenance graph. An essential source for threat behavior includes incident response reports where an analyst gathers evidence related to steps of malicious activity on a system [59]. In this regard, the query graph may not necessarily convey the system-level interactions with all its details. For example, a query graph may show a browser or application process as part of an attack vector. However, on the audit logs, this process may correspond to a cloned version of the main process or another process spawned by a launcher process to handle the task, which may not be a part of the query. Additionally, missing or unrecorded events or the inclusion of additional interactions can cause the queried behavior to only partially match its version on the provenance graph. Thus, the learned representation must offer a degree of robustness to such variations.

Challenge #5: Setting a Learning Objective. The learning objective serves as a guiding principle for the model's training process. Establishing a clear and well-defined learning objective is essential for the model to effectively learn subgraph relations in query graphs. This entails generating training samples that accurately represent the characteristics of the query graphs that are expected to be encountered in practical usage. To promote model generalization, it is essential to draw training and test samples from the same data distribution. This consistency allows the model to effectively learn the underlying patterns and relations in the data, enhancing its ability to recognize subgraph relations in new and unseen data. In the context of threat hunting, query graphs are anticipated to exhibit a strong relationship with observed attack behaviors. Consequently, it is vital to adapt the model's training process to reflect these behaviors.

3 SYSTEM DESIGN AND METHODOLOGY

Our approach to hypothesis-driven threat hunting utilizes provenance graphs and frames the task as a subgraph entailment problem. Provenance graphs depict audit logs as labeled, typed, and directed graphs, where nodes represent system entities and directed edges indicate transformation or flow of information due to distinct system calls. Timestamps assigned to each node and edge capture the graph's evolving nature. Our technique aims to effectively identify system behaviors of interest by representing queries as graphs and searching for them within the larger context of the provenance graph.

3.1 Problem Formulation

A graph $\mathcal{G}$ is defined as a set of nodes $\mathcal{V} = \{v_1, ..., v_N\}$ and a set of edges $\mathcal{E} = \{e_1, ..., e_M\}$ where each node and each edge are associated with a type. Given a target and a query graph, the solution to the graph entailment problem involves detecting every query instance in the target. Since exact subgraph matching on graphs with the scale of provenance graphs is not feasible, we employ an approximate matching method to make subgraph predictions. Our method involves a sequence of steps that reduce the size of a graph while ensuring that the system behavior is preserved at a higher level of abstraction.

Following this reduction, the primary problem we want to solve is formulated as follows. Let $\mathcal{G}$ represent a reduced provenance graph. We decompose $\mathcal{G}$ into a set of overlapping subgraphs by extracting the k-hop ego-graph $\mathcal{G}_p$ of each process node $p \in \mathcal{V}_P$. Given the set of ego-graphs $\mathcal{P} = \{\mathcal{G}_p | \mathcal{G}_p \subseteq \mathcal{G}\}$, our goal is to learn an embedding function $\eta : \mathcal{G}_p \rightarrow \mathbb{R}^d$ that maps each ego-graph to a d-dimensional representation $\mathbf{z} \in \mathbb{R}^d$, capturing the key structural information of a graph for use in conjunction with a suitable subgraph prediction function φ. Hence, the encoder must incorporate an inductive bias to effectively represent the subgraph relationship while learning a mapping in which the subgraph prediction function $\varphi(\mathbf{z}_p, \mathbf{z}_q)$ serves as a vector-based measure to confirm the existence or absence of this relation. It must further be noted that since provenance graphs are typically very large, $\varphi(\mathbf{z}_p, \mathbf{z}_q)$ needs to be evaluated over all $\mathbf{z}_p$ values for a given $\mathbf{z}_q$. Therefore, effective computation of $\varphi(\mathbf{z}_p, \mathbf{z}_q)$ is very critical.

In our method, we employ a subgraph embedding function that effectively addresses both issues. For this, we utilize the notion of order embedding, which aims to encode the ordering properties among entities into target representation space [2, 9, 67, 68]. Order embeddings specifically model hierarchical relations with anti-symmetry and partial-order transitivity, which are inherent to subgraph relationships. To develop the embedding function η, we utilize an inductive graph neural network [18] and apply the order embedding technique introduced by Vendrov et al. [67]. This approach enables us to learn a geometrically structured embedding space that effectively represents the relationships between subgraphs. At the query execution state, the encoder η is applied independently to the query graph $\mathcal{G}_Q$. This is done by identifying ego-graphs $\mathcal{Q} = \{\mathcal{G}_q | \mathcal{G}_q \subseteq \mathcal{G}_Q\}$ corresponding to all anchor nodes $q \in \mathcal{V}_Q$ and computing the embeddings $\mathbf{z}_q = \eta(\mathcal{G}_q)$ for all ego-graphs in $\mathcal{Q}$.

Then, the subgraph prediction function is evaluated by considering the newly computed embeddings $\mathbf{z}_q$ from the query and the precomputed subgraph embeddings $\mathbf{z}_p$. This involves identifying (p, q) node pairs that satisfy the subgraph relation $\varphi(\mathbf{z}_p, \mathbf{z}_q)$. To determine whether one graph is a subgraph of another, one can simply check that all neighbors of $q \in V_Q$ satisy the subgraph relationship. However, such a comparison enforces an exact match of the query, which cannot handle cases where discrepancies exist between the query and the logs. To address this issue and achieve greater generality, we propose the use of a soft-decision metric,

An ego-graph of depth k, centered around a node p, is an induced subgraph that includes p and all nodes within a distance k from it.

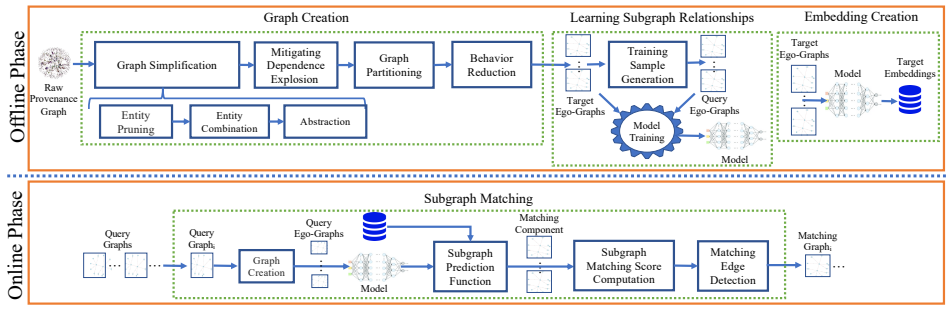

Figure 2: Overview of ProvG-Searcher and its key components.

defined as follows:

$$\mathcal{G}_Q \subseteq \mathcal{G}_P \text{ iff } g(\mathcal{G}_p^*, \mathcal{G}_Q) \geq \tau \text{ where}$$
$$\mathcal{G}_p^q = \{\cup \mathcal{G}_p | \forall q \in V_Q,\ \varphi(z_p, z_q) \text{ is satisfied}\} \quad (1)$$

Here, $\mathcal{G}_p^*$ represents a graph obtained by combining all ego-graphs $\mathcal{G}_p$ that satisfy the subgraph relationship with the query ego-graphs of $\mathcal{G}_Q$, and $g(.,.)$ is a scoring function that computes the intersection of $\mathcal{G}$ and the query graph $\mathcal{G}_Q$.

3.2 System Overview

Our system, displayed in Fig. 2, consists of an offline embedding stage and an online prediction stage. During the offline phase, audit logs are collected and used to create a provenance graph. Several graph simplification steps are taken to optimize graph learning and better align with queried behavior (Sec. 4.1.1). These involve removing non-essential nodes and edges related to pipe and memory accesses, merging cloned processes with their parents, and consolidating network communication events over time intervals. Each node that describes a unique system entity is then assigned to an abstract class that describes a higher-level categorization, such as a system file, a user application process, etc. After simplifying the graph, the next step is to version the nodes, which incorporates timing information into the graph to prevent spurious information flows between nodes (Sec. 4.1.2). In the final step, the graph is partitioned into overlapping subgraphs by extracting the k-hop ego-graph of each process node. Here, k also signifies the number of GNN layers used to obtain a subgraph representation. We then apply another level of reduction to the abstracted ego-graphs by removing repeated behaviors using an iterative label propagation (Sec. 4.1.4). This leaves unique traits in each subgraph to learn as part of a subgraph relation.

We then employ a k-layer GNN to learn a representation of the subgraph relation by training it on positive and negative pairs of graphs to learn an inductive embedding function that will be used in conjunction with a subgraph prediction function. For this, we utilize order embeddings which provide a natural way to model transitive relational data such as entailing graphs. These embeddings essentially obey a structural constraint whereby $\mathcal{G}_q$ is deemed a subgraph of $\mathcal{G}_p$ if and only if all the coordinate values of $\mathbf{z}_p$ are higher than $\mathbf{z}_q$'s (Sec 4.2). During the prediction stage, the query graph undergoes the same processing steps as the provenance graph and is partitioned into subgraphs. Afterward, the order relations between the query ego-graph embeddings and the precomputed ego-graph embeddings in the provenance graph are computed to determine whether the subgraph relation exists (Sec. 4.3).

3.3 Threat Model

We assume that adversaries cannot tamper with the system or the kernel auditing facility responsible for collecting security logs, ensuring the integrity of the constructed provenance graphs. However, adversaries who possess knowledge of our system might make slight adjustments to their behavior in an attempt to better conceal their traces. To obscure their actual behavior from threat hunters, an adversary may choose to refrain from performing specific steps or replace some of the attack steps with alternative ones. Hence, it is crucial for our system to exhibit robustness against the insertion and deletion of nodes and edges in the query graph representing the attack behavior. Although limiting the impact of such modifications poses a significant challenge, it is widely recognized that adversaries face difficulties in frequently altering their tactics, techniques, and procedures. Moreover, engaging in arbitrary activities during an attack would increase the risk of detection by other system-wide attack detection tools. As a result, we proceed with the assumption that the fundamental characteristics of the attack behavior will remain unchanged and that similarity will be largely preserved.

4 SYSTEM COMPONENTS

4.1 Graph Creation

ProvG-Searcher processes raw system logs through multiple reduction steps before constructing a streamlined provenance graph that represents various interactions between subjects (e.g., processes) and objects (e.g., processes, files, network sockets). The graph creation module produces a partitioned version of this graph to facilitate effective learning of subgraph relationships.

4.1.1 Graph Simplification (Challenges #1 and #4). We start the process by adopting the approach from previous research [70, 72], where we retain only process, file, network, and registry objects. We maintain all read, write, and modify attribute events for processes, files, sockets, and registries. Additionally, we preserve clone, fork, or execute events for processes, while removing open and close events to avoid redundancy, as they precede or follow read or write events.

We expand upon the generic graph simplification approaches in two main aspects. The first aspect pertains to the handling of threads during the creation of provenance graphs. Applications often use threads to enhance performance and scalability, but query graphs might not exhibit this behavior [62]. To ensure consistency across both graphs, we merge threads into their parent process as illustrated in Fig. 3. Furthermore, to capture changes in the behavior of remote servers over time, we adopt the method described in [28], treating each remote IP and port combination as a distinct source within 10-minute time windows.

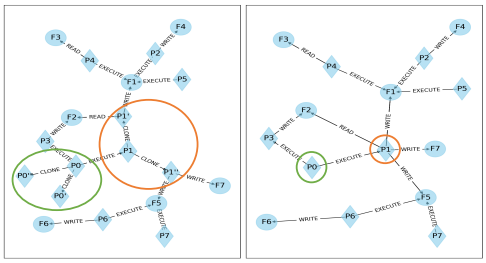

Figure 3: An example provenance graph consisting of processes (◇) and files (○). The node labels represent the names of the files and processes. The nodes that represent threads of processes P_0 and P_1 are surrounded by circles. The simplified graph after combining threads is shown on the right side.

The second aspect focuses on achieving a consistent representation of system entities and their interactions in both the query and provenance graphs. A discrepancy in these representations could hinder the model's generalization capability. This is particularly concerning when the query lacks the granularity and detail typically found in system logs, potentially leading to mismatches or misinterpretations when comparing the query and provenance graphs. For example, a browser process in the query may correspond to one of several processes, such as Firefox, Chrome, or Safari, in the provenance graph. Similarly, two files with the same name may be associated with different functions in the context of different applications. To avoid such ambiguities, we use a system directory-based abstraction for all system entities, with the exception of network objects, in order to provide a consistent description for each entity. Specifically, we assign category labels to each entity based on its root directory in the file system, indicating a higher-level function for each entity. In contrast, network objects are abstracted based on their source IP, destination IP, and destination port. Each IP address is categorized as *public, private*, or *local* based on its usage, while ports are categorized as *reserved* if they are less than 1024 and as *user* otherwise. Overall, this led to the use of more than 70 abstraction categories, which are shared in detail in [14].

Abstracting system entities not only helps reveal recurring patterns in a graph but also allows for further reduction in graph complexity. When object nodes—such as files, network sockets, and registry entries—within the same abstraction category are connected to a single subject (process) node through a shared event type, we merge these nodes into one node with the same object abstraction. To maintain causality relationships during deduplication, we keep the timestamp of the first event if the flow starts from a process to an object (e.g., write or attribute modification) and the timestamp of the last event if the flow originates from an object and leads to a process. Since these nodes are connected to only one process, this procedure preserves the causality relationships between nodes.

4.1.2 Mitigating Dependence Explosion (Challenges #2 and #3). The dependence explosion, caused by high in-degree and out-degree nodes in a provenance graph, significantly impedes the learning of subgraph relationships. This is because tracing through such nodes leads to an exponential increase in the possible node interactions that must be considered. We implement two strategies to address this problem while extracting ego-graphs from the provenance graph. First, we leverage the available event timestamp information to impose a timing constraint on the flows. To satisfy this requirement, we employ the graph versioning approach proposed in [28], which effectively encodes time dependencies into the provenance graph by creating a new version of a node whenever the corresponding system entity receives new information. This method ensures that all paths in the extracted ego-graphs have edges with monotonically increasing timestamp values, thereby preserving the causal order of events. Additionally, it allows the elimination of repeated events between two versioned nodes. It's worth noting that incorporating node versioning in provenance graphs does not necessitate the inclusion of edge timestamps in the query graph.

The second strategy we propose aims to address the challenges of oversmoothing in GNNs by alleviating the impact of dependence explosion. For this, we designate specific nodes as sink nodes. Notably, interactions with high-degree nodes, such as DNS server IP addresses or cache files, do not provide discriminative information that aids in learning subgraph relationships. Moreover, any system entity interacting with these high-degree nodes will appear to receive information from numerous other system entities. This contributes to the oversmoothing phenomenon, as it results in an expanded receptive field for a GNN. By treating these nodes as sink nodes, we effectively prevent non-informative information flows, leading to more accurate and meaningful learning of subgraph relationships. It is important to note that non-process nodes with zero in-degree or out-degree, such as log files written to by all processes without reads, or configuration files that are only read, are also considered sink nodes. This is because there is no flow of information between the neighbors of these nodes, making their role in understanding subgraph relationships less significant.

4.1.3 Graph Partitioning (Challenge #1). A pattern within a graph can always be detected in a sufficiently large local neighborhood surrounding a specific node. An ego-graph with depth k, centered around node v, is an induced subgraph that includes v and all nodes within a distance k from it. In fact, any pattern with a radius smaller than k can be found within an ego-graph of depth k, where the value k can be determined based on the characteristics of query graphs and the graph representation learning algorithm. To take advantage of this, the reduced provenance graph, containing versioned and sink nodes, is partitioned into subgraphs by extracting the ego-graph of each process node which are crucial for understanding the system behavior [62]. Given that processes are the only active system entities, adopting a process-centric view of the

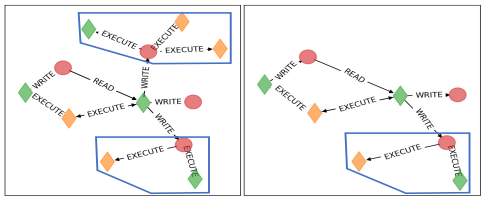

Figure 4: The left figure displays the ego-graph of a process P_1 after applying simplification and dependence explosion steps to the initial graph shown in Fig. 3. The right figure presents the resulting graph after applying behavior-preserving reduction. Algorithm 1 is utilized to identify the recurring behavior, as indicated within the blue boxes.

provenance graph indirectly encompasses all relationships involving other entities. This approach also contributes to reducing the computational complexity of both offline and online stages. Even though the graph partitioning step is performed once, it is essential for this task to be as efficient as possible. To extract ego-graphs, we use a dynamic programming algorithm presented in [14].

4.1.4 Behavior-Preserving Graph Reduction (Challenges #1 and #4). At this stage, resulting ego-graphs may still contain redundant information. For instance, consider an ego-graph showing a user process that has written to hundreds of `var` directory files, possibly in different contexts. A given query involving this user process, however, is likely to be relevant to only one of these contexts. Therefore, from a search perspective, a user process writing to a `var` directory is more informative than tracking the number of written files. Moreover, as repeated events can dominate the information aggregation step, GNNs may primarily learn those repetitive behaviors while neglecting less frequent ones. To avoid the suppression of observed system behaviors, it is necessary to identify and eliminate recurring patterns within each ego-graph.

We propose a behavior-preserving graph reduction method that captures the local graph structure surrounding each node at varying depths using iterative label propagation. First, we collapse the node versions back onto the original nodes. (While this loses the benefits of versioning, maintaining time-dependency during ego-graph creation ensured that only relevant system entities are included in each ego-graph, eliminating all spurious interactions with unrelated system entities.) We compute a hash value for each node by aggregating edges with their neighboring nodes' hash values. For each node, the abstraction category of the node is assigned as its 0-hop hash, and the hashes of neighboring nodes are accumulated from incoming and outgoing edges using the following equation:

$$nh[n][forw][l] = hash\left(\sum_{e,v \in In(n)} (e + nh[v][forw][l-1])\right),$$

$$nh[n][back][l] = hash\left(\sum_{e,v \in Out(n)} (e + nh[v][back][l-1])\right).$$

A similar approach was taken by Watson [82] to cluster similar system behaviors on the provenance graph. However, unlike ProvG-Searcher, Watson adopts a sequence modeling approach by decomposing subgraphs into paths and utilizing TransE knowledge embeddings.

Algorithm 1 Behavior-Preserving Graph Reduction Method

Require: $\mathcal{G}_p$: ego-graph of anchor node v_p, nh: hashes of node n
Ensure: $\mathcal{G}_p$: reduced ego-graph of anchor node v_p

```
forward ← V_p
for l = 0, .., k do
    unique ← dict()
    for all e, v ∈ In_e(forward) do
        unique[e + nh[v]['forw'][k − l]].append(v)
    end for
    for all hash ∈ unique do
        forward.append(random_select(unique[hash])
    end for
end for
backward ← V_p
for l = 0, .., k do
    unique ← dict()
    for all e, v ∈ Out_e(backward) do
        unique[e + nh[v]['back'][k − l]].append(v)
    end for
    for all hash ∈ unique do
        backward.append(random_select(unique[hash])
    end for
end for
unique_nodes ← forward|backward
G_p ← G_p.subgraph(unique_nodes)
```

Here, $nh[n][forw][l]$ and $nh[n][back][l]$ represent the hash values of node n at l-hop distance in the forward and backward directions, respectively. The function $hash()$ is the SHA-256 function that takes a string as an input and returns a cryptographic hash value. We use the function $In(n)_e$ to retrieve all incoming edges with their source nodes for a given input node n, while $Out(n)_e$ returns all outgoing edges and their target nodes. The set function $\sum$ returns the unique strings in its input in sorted order.

We perform behavior-preserving reduction starting from the anchor node of an ego-graph. At each depth l, we determine all unique $(k-l)$ hash values of the neighbor nodes and select one node for each unique hash value. These selected nodes form the set of unique nodes for that depth. We repeat this process at all depths up to k and obtain a set of unique nodes for the entire ego-graph. Using these unique nodes, we create a reduced ego-graph that preserves the behavior of the original one. The detailed steps of our behavior-preserving reduction are provided in Algorithm 1. The *random_select* function takes a list of nodes as input and returns one random node from the input list, and the *subgraph* function is used to create a reduced graph with the input nodes. Figure 4 depicts the resulting ego-graph obtained after applying the behavior-preserving reduction step.

4.2 Learning Subgraph Relationships

4.2.1 Subgraph Prediction Function (Challenge #5): Subgraph relationship essentially imposes a hierarchy over graphs. Therefore, a vector representation for subgraphs should take into account the structure of this hierarchy to effectively evaluate the relationship

between two graphs. In [67], Vendrov et al. introduced order embeddings to model the transitivity and antisymmetry of partially ordered data, which naturally applies to the representation of entailing graphs. Order embeddings ensure the preservation of partial ordering between elements by maintaining the order relations of coordinates in the embedded space such that for two graphs $\mathcal{G}_p$ and $\mathcal{G}_q$ and their embeddings $\mathbf{z}_p, \mathbf{z}_q \in \mathbb{R}^d$

$$\mathcal{G}_q \subseteq \mathcal{G}_p \text{ if and only if } \forall_{i=1}^{d} \mathbf{z}_{p_i} \geq \mathbf{z}_{q_i}. \tag{2}$$

That is, $\mathcal{G}_q$ is a subgraph of $\mathcal{G}_p$ if and only if all the coordinate values of $\mathbf{z}_p$ are higher than $\mathbf{z}_q$'s. To impose this constraint on the learned relation, [67] proposed an order violation penalty function

$$E(\mathbf{z}_q, \mathbf{z}_p) = \| \max\{0, \mathbf{z}_q - \mathbf{z}_p\} \|^2 \tag{3}$$

to measure the extent to which two embeddings violate their order, i.e., $E(\mathbf{z}_q, \mathbf{z}_p) \neq 0$ if Eq. (2) is not satisfied. Consequently, we optimize our GNN to minimize the order violation penalty to learn an approximate order embedding function using the following max-margin loss

$$\mathcal{L}(\mathbf{z}_q, \mathbf{z}_p) = \sum_{(\mathbf{z}_q, \mathbf{z}_p) \in S^+} E(\mathbf{z}_q, \mathbf{z}_p) + \sum_{(\mathbf{z}_{q'}, \mathbf{z}_{p'}) \in S^-} \max\{0, \alpha - E(\mathbf{z}_{q'}, \mathbf{z}_{p'})\} \tag{4}$$

where S^+ denotes a set of positive graph pairs that satisfy the subgraph relation, and S^- is the set of negative pairs for which this relation is not satisfied. This loss crucially encourages positive samples to have zero penalty and negative samples to have a penalty greater than a margin α, thereby ensuring that two embeddings have a violation penalty of α if the subgraph relation does not hold. Thus, the subgraph prediction function introduced in Eq. (1) becomes a proxy for the order violation penalty, i.e., $\varphi(z_p, z_q) = E(\mathbf{z}_q, \mathbf{z}_p)$. In our evaluation, as an alternative, we also utilized a neural network model to learn the intrinsic relationship between embeddings $\mathbf{z}_q$ and $\mathbf{z}_p$ of entailing graphs as a representation for $\varphi(\mathbf{z}_p, \mathbf{z}_q)$.

4.2.2 Training Sample Generation (Challenge #5). Training ProvG-Searcher requires positive and negative pairs of query and target graphs. These pairs can be represented as $(\mathcal{G}_q^+, \mathcal{G}_p)$ and $(\mathcal{G}_q^-, \mathcal{G}_p,)$ where $\mathcal{G}_q^+$ is a subgraph of $\mathcal{G}_p$ and $\mathcal{G}_q^-$ is not. During training, the model first computes embeddings for all graphs in a batch of positive and negative pairs, then evaluates the resulting loss as defined in Eq. (4). We backpropagate this loss to update the network weights and minimize its value.

To ensure generalization, $\mathcal{G}_p$ is chosen as an ego-graph of a node v_p within a reduced provenance graph $\mathcal{G}_\mathbf{P}$. There are two crucial factors to consider when creating a paired query graph. The first is the size of the query graphs. In line with previous research [51, 59, 72, 85], we opted to limit the size of reduced query graphs to 10-15 edges considering 3-hop ego-graphs. We note that in unreduced query graphs, this may correspond to 40-50 edges as discussed in the findings of Table 1 in Sec. 5.1. The second factor is the strategy employed to generate $\mathcal{G}_q^+$ and $\mathcal{G}_q^-$. The most straightforward approach to create $\mathcal{G}_q^+$ involves subsampling a set of nodes or edges from $\mathcal{G}_p$ and extracting the corresponding node or edge-induced graph. However, a random selection scheme could expose the model to repetitive behaviors and lead to overfitting common graph patterns. As for $\mathcal{G}_q^-$, choosing a graph at random may not only generate easy negative samples but also inadvertently yield an actual subgraph of $\mathcal{G}_p$, particularly when $\mathcal{G}_p$ is large.

To circumvent these pitfalls, we propose a new graph sampling method based on path frequency. First, possible flows for each ego-graph, $\mathcal{G}_p \in \mathcal{G}_\mathbf{P}$, are determined via forward and backward depth-first search around the anchor node v_p, where a flow represents a path between two nodes of $\mathcal{G}_p$ that passes through v_p. To generate positive graph pairs, we count the unique flows for each ego-graph $\mathcal{G}_p$ belonging to the same process path. Then, for each $\mathcal{G}_p$, we randomly select a flow from all its flows based on their inversely weighted frequency in all ego-graphs of the same path. Once the flow is selected, we expand it by randomly choosing some incoming and outgoing edges of the nodes in the selected flow until the desired number of edges is reached.

Creating a negative example is more challenging as one needs to avoid introducing both superficial and unlikely behaviors to $\mathcal{G}_q^-$. One can indeed create a hard negative example $\mathcal{G}_q$ by synthetically adding edges and nodes to a target graph $\mathcal{G}_p$ to violate the subgraph relationship. However, this may result in implausible behaviors. Alternatively, one can identify an arbitrary flow from the list of known unique flows, which is not contained within the target $\mathcal{G}_p$, and use the corresponding process's ego-graph to expand this flow which may result in a very easy example for the model. Instead, we follow a three-step procedure to create a negative sample: First, we pick a flow from an ego-graph with the same anchor process as the target graph and expand from it. (E.g., to create a negative example for a Firefox process, we prefer to choose an ego-graph of another Firefox process and subsample it.) However, this may not always be possible if there are not many instances of the same process. Because in such a case, the same behavior may potentially be used to generate many negative examples, thereby biasing the model. To avoid this, as a second step, we utilize the behavior of another process with the same abstraction, i.e., using a Chrome process instead of a Firefox. Where this is not possible, as a last resort, we pick a random flow and expand from it.

To ensure that the generated $(\mathcal{G}_q^-, \mathcal{G}_p)$ pairs violate the subgraph relationship, we apply an independent validation step. For this, we first check if any node or edge abstraction is present in $\mathcal{G}_q$ but absent in $\mathcal{G}_p$.If all categories of system entities are indeed found within $\mathcal{G}_p$, we proceed to analyze all 1- and 2-hop flows in the query graphs, taking both edge types and node abstractions into account. Should at least one distinct flow fail to meet the subgraph relationship criteria, the pair is deemed a negative sample. Sample positive and negative query graphs can be seen in [14].

4.2.3 GNN Architecture and Features. Graph neural networks (GNNs) are expressed as message-passing networks that rely on three key functions, namely `MSG`, `AGG`, and `UPDATE`. These functions work together to transfer information between the different components of the network and update their embeddings. Typically, these functions operate at the node level, exchanging messages between a node v_i and its immediate neighboring nodes $\mathcal{N}_{v_i}$. In layer l, a message between two nodes (v_i, v_j) depends on the previous layer's

Subgraph relation is a partial order relation as it has the following three properties Given three graphs $\mathcal{G}_a$, $\mathcal{G}_b$, $\mathcal{G}_c$, subgraph relationship satisfies the following three properties: (i) $\mathcal{G}_a \subseteq \mathcal{G}_a$ (reflexivity) (ii) if $\mathcal{G}_a \subseteq \mathcal{G}_b$ and $\mathcal{G}_b \subseteq \mathcal{G}_c$, then $\mathcal{G}_a \subseteq \mathcal{G}_c$ (transitivity); and (iii) if $\mathcal{G}_a \subseteq \mathcal{G}_b$ and $\mathcal{G}_b \subseteq \mathcal{G}_a$, then $\mathcal{G}_a = \mathcal{G}_b$ (antisymmetry).

hidden representations h_i^{l-1} and h_j^{l-1}, i.e, $m_{ij}^l = \texttt{MSG}(h_i^{l-1}, h_j^{l-1})$. Then, AGG combines the messages from $\mathcal{N}_{v_i}$ with h_i^{l-1} to produce v_i's representation for layer l in UPDATE. Various adaptations of this core message passing framework with alternative MSG, AGG, and UPDATE implementations have been proposed [18, 29, 33, 66]. To leverage the comprehensive representation of provenance graphs, we deploy a multi-relational GNN that can also incorporate information by taking into account both edge type and edge direction relations [61].

The expressive power of GNNs is known to increase when node and edge features become more distinct [19]. To take advantage of this, it is essential to assign suitable node and edge features. We employ two separate one-hot encoding representations for each object type and abstraction category, and the node features for both the provenance and query graphs are determined in the same way.

4.3 Subgraph Matching Score Computation

Our technique relies on two measures to achieve robustness against inexact queries, in cases where the query may not precisely match the system events being searched for. The first measure is utilized when assessing the subgraph relationship between two ego-graphs, as defined in Eq. (5), by permitting a certain degree of order violation, i.e., $\varphi(z_p, z_q) = E(\mathbf{z}_q, \mathbf{z}_p) \leq \tau_{ovp}$. The second measure allows for partial matching of the query graph within the provenance graph, which is achieved by using a graph intersection-based scoring function. The graph $\mathcal{G}^*$, as described in Eq. (1), is the union of all possible matches $\mathcal{G}_p$ to $\mathcal{G}_Q$ and may contain several disconnected parts. The scoring function intersects the query graph with each connected component (CC) of $\mathcal{G}^*$ and utilizes the ratio of edges in the intersected graphs to the total number of edges in $\mathcal{G}_Q|$ to compute the final matching score, as defined below:

$$g(\mathcal{G}^*, \mathcal{G}_Q) = max\left(\left\{\frac{|\mathcal{G}_1^* \cap \mathcal{G}_Q|}{|\mathcal{G}_Q|}, \ldots, \frac{|\mathcal{G}_n^* \cap \mathcal{G}_Q|}{|\mathcal{G}_Q|}\right\}, \tau\right) \quad (5)$$

$$\text{where } CC(\mathcal{G}^*) = \{\mathcal{G}_1^*, \ldots, \mathcal{G}_n^*\} \text{ and}$$

$$max(S, \tau) = \{max(S) \text{ if } max(S) > \tau, 0 \text{ otherwise}\}.$$

The connected component that yields the highest score above the threshold τ, together with its intersected edges, is identified as the matching subgraph corresponding to the query. The intersected edge-induced graph extracted from this connected component is returned as a response to the query.

5 RESULTS

We evaluate our approach on four DARPA TC [11] datasets–Theia, Trace, Cadets, and FiveDirections–which feature eight distinct attack scenarios as described in [65]. The Theia dataset was collected from hosts operating on Ubuntu 12.04, the Trace dataset was collected from hosts operating on Ubuntu 14.04, the Cadets dataset was obtained from a FreeBSD 11.0 host, and the FiveDirections dataset was collected from a Windows 7 machine. The attack scenarios used to evaluate our approach include an Nginx server backdoor, a Firefox backdoor, a backdoor with one of Firefox's extensions (password manager), and a phishing email with a malicious Excel document.

In this section, we start by evaluating the efficiency of our graph reduction strategies. Subsequently, we assess the capacity of order embeddings to represent subgraph relationships using DARPA TC datasets. Next, we examine our technique's ability to search for and identify subgraphs with two types of queries: those derived from converting DARPA TC attack logs into query graphs and those representing generic system activities. Finally, we compare ProvG-Searcher to other hypothesis-driven threat hunting methods in terms of both effectiveness and performance.

Table 1: Reduction in Ego-Graph Size During Graph Creation Process (GS: Graph Simplification, DEM: Dependence Explosion Mitigation, BR: Behavior-Preserving Reduction)

Dataset	Initial		GS (Sec. 4.1.1)		DEM (Sec. 4.1.2)		BR (Sec. 4.1.4)	
	N	E	N	E	N	E	N	E
Theia	206k	13M	22k	6.9M	159	336	19	38
Trace	29k	490k	400	1461	329	1303	16	28
Cadets	5k	160k	1.9k	76k	43	75	12	15
FiveDirection	25k	8.7M	2.1k	4.2M	350	1034	71	527

5.1 Reduction in Graph Size

We demonstrate the effectiveness of our graph creation method in terms of reduction in the graph size. Table 1 summarizes the results obtained for each dataset, where we compute the average count of nodes and edges in 3-hop ego-graphs of process nodes. The first column of the table presents the average number of nodes and edges in each ego-graph after the graph simplification steps, up until the entity abstraction step described in Sec. 4.1.1 is applied. (Since DeepHunter also applies these steps [72], we consider this as our starting point). We then process the provenance graph by applying all remaining graph reduction steps. Our results show a substantial reduction in the ego-graph size across all datasets. For instance, on the Theia dataset, the ego-graphs initially contain 206K nodes and 13M edges, while the final ego-graphs contain only 19 nodes and 38 edges, on average. The variation across datasets can be attributed to the nature of graphs where the node degrees are much smaller in the Trace and Cadets dataset.

These results demonstrate the effectiveness of our approach in reducing the size of ego-graphs while still preserving the diverse behaviors exhibited by processes. In fact, we observe that several reduced ego-graphs $\mathcal{G}_p$ are duplicated, containing identical nodes and edges. To ensure all graph relationships are learned on an equal footing, regardless of their prevalence, we retain only one sample from each set of repeated ego-graphs. This reduces the total number of ego graphs from 15k, 235k, 195k, and 17k to 1k, 11k, 3k, and 3k for Theia, Trace, Cadets, and Five Directions datasets, respectively. It's essential to clarify that several other approaches have been proposed to simplify provenance graphs for various computational purposes. (See Section 6.2 for a brief review.) However, the effectiveness of a graph simplification method should be evaluated based on its intended primary task. For instance, a graph simplification technique designed to support forensic tractability or anomaly detection may not be suitable for our specific use case, and vice versa. In the context of searching for threat behaviors, it is worth mentioning that Poirot [51] does not employ graph reduction, whereas DeepHunter [72] suggests preserving parts of the graph around EDR alerts.

5.2 GNN Architecture and Parameter Selection

In the design phase of our model, we evaluate the performance of the subgraph prediction function (Sec. 4.2) on the Theia dataset to determine the most suitable GNN architecture and optimal model parameters. We train the models using 80% of the 3-hop ego-graphs and reserve 20% for testing. During the training phase, we ensure the test samples are not seen. We conduct the training on 400 batches with a batch size of 1024, which includes an equal number of positive and negative target-query pairs. After the training, we evaluate the models on 10 batches.

To identify the most effective GNN architecture for our system, we assess the performance of several well-known graph neural network architectures, such as GCN[33], GIN [76], and GraphSage [18]. Additionally, we experiment with the Multi-Relation GNN architecture [60, 61], where each edge type and direction are represented separately. Following a thorough evaluation, we determine that the multi-relational GraphSage model, which integrates GraphSage with the Multi-Relation GNN, delivers the best performance among the tested architectures.

We also analyze the impact of the number of layers and aggregation method used to obtain subgraph embeddings on the model's performance. Although the performance differences are not substantial, using three layers yields the best results. In addition, we explore a variety of pooling methods, such as add pooling, mean pooling, graph multiset pooling [4], and utilizing only the anchor node's embedding. Our findings indicate that add pooling, which aggregates the embeddings of all nodes in the graph, surpasses the other pooling techniques. Detailed findings from these tests are available in [14]. We conduct further experiments to identify optimal values for batch size, scheduling scheme, weight decay parameter, and embedding size. The results reveal that, apart from the embedding size, the choice of other parameters does not significantly affect the performance. We observe that improvements become marginal when the embedding size exceeds 256 dimensions. Consequently, we choose this value for our experiments.

5.3 Power of Order Embeddings

We initially assess the effectiveness of order embeddings in identifying subgraph relationships between two graphs. We train a separate model for each dataset to learn the subgraph prediction function, i.e., $\varphi(z_p, z_q)$. We employ a 3-layer multi-relational GraphSage GNN architecture with add pooling and an embedding size of 256. The ability of order embeddings to detect subgraph relationships among 3-hop graphs is illustrated by the ROC curves shown in Fig. 5(a). The results indicate that our method exhibits strong performance, with AUC scores ranging from 96.6 to 98.3 across the datasets, effectively distinguishing positive queries from negative ones in the provenance graph.

To examine the robustness properties of order embeddings, we use the same set of test samples from the Theia dataset while considering two types of changes in attacker behaviors. In the first scenario, we examine the inclusion of additional activities by the attacker on the system, relative to the query graph. This behavior can be emulated by reducing the number of nodes and edges in the query graph. In the second scenario, we assume the attacker performs fewer actions on the system compared to the query, which is simulated by removing nodes and edges from the target graph. The results indicate that, in scenario #1 where nodes and edges are eliminated from the query graphs order embeddings achieve AUC scores of 93.3 and 90.2 when removing 15% of the query nodes and 45% of the query edges, respectively. A similar behavior is observed in scenario #2, where nodes and edges are removed from the target graph. Our technique achieves AUC scores of 92.9 and 93.0 when eliminating up to 15% of the query nodes and 45% of the query edges from the target graphs, respectively. Further details on both scenarios can be found in [14]. Based on these findings, we can conclude that our technique is capable of handling imprecise queries.

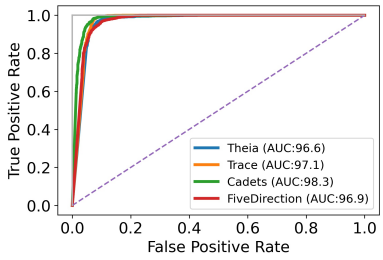

Figure 5: ROC curves for validating subgraph relationship between 3-hop graphs using our model on all the datasets

5.4 Accuracy in Subgraph Entailment Decision

We now test the performance of ProvG-Searcher in accurately determining whether a query graph is entailed within a provenance graph by computing the subgraph matching scores, Eq. (5), between the query and the target graphs. For this, we use the models generated earlier to evaluate the subgraph relations among 3-hop graphs. We test their performance on two sets: (i) attack queries underlying DARPA-TC datasets [65], and (ii) a new test set comprising 5-hop ego-graphs involving generic behaviors extracted from the test portion of the Theia dataset.

5.4.1 DARPA-TC Attack Queries. The DARPA-TC dataset consists of eight attack scenarios, each consisting of up to three processes. We began by evaluating the subgraph prediction function, which involves extracting and searching process-centric ego-graphs from the query graph within the corresponding provenance graphs. Table 2 displays the number of matching ego-graphs compared to the total number associated with each process in the provenance graph. For instance, process P1 has 846 instances within the provenance graph in the TRACE dataset's attack query. Our subgraph prediction function identifies only two, or 2/846, as matching candidates. Notably, no missed matches are observed in any of the test scenarios. Upon analyzing the false matches, we find that all returned ego-graphs are connected, and on average, 78.9% of all query nodes appear in those ego-graphs. A comparison between the number of query nodes in correctly-matching and incorrectly-matching ego-graphs reveals that the former contain, on average, 57% more query nodes. This indicates that our subgraph matching function effectively localizes the query within the provenance graph. We proceed to calculate the overall graph-matching scores for each scenario by

Table 2: Number of Matches Identified Per Query Ego-Graph

Dataset	Query	P1	P2	P3
Theia	Q1	2/846	1/1	1/1
Trace	Q2	1/21023	1/1	-
	Q3	1/239	1/2	-
Cadets	Q4	3/15	1/1	-
	Q5	1/15	1/1	-
	Q6	2/15	3/4	1/2
Five Direction	Q7	6/724	-	-
	Q8	9/724	3/10	1/1

first merging all the returned ego-graphs into a single graph, $\mathcal{G}^*$. Following that, we compute the corresponding scores $g(\mathcal{G}^*, \mathcal{G}_Q)$, as described in Sec. 4.3. The resulting score values consistently exceeded 0.9, indicating a high degree of matching accuracy in all scenarios. To assess the false matching rate, we also tested each trained model by querying them with attack queries from other datasets. The results showed that none of these queries yielded a match, with the highest observed matching score being around 0.2.

5.4.2 Generic Queries. We also evaluate the performance of ProvG-Searcher on a set of queries that include generic system behaviors. A generic behavior refers to any system activity that does not necessarily involve attack events. These behaviors can be obtained by randomly subsampling ego-graphs of process nodes from the system-level provenance graph. However, this approach would result in a sampling that mainly comprises the most prevalent system behaviors, leading to repeated queries with the same behaviors. To address this, we adopt the same approach used during model training to learn diverse system behaviors (Sec. 4.2.2). For this purpose, we generate ten batches of test samples, each consisting of 5-hop ego-graphs for both positive and negative sample pairs (5K each). Here, a positive sample includes an ego-graph with a matching subgraph, while a negative sample includes an ego-graph with a non-matching but similar graph.

We validate the subgraph relationship by calculating the order violation penalty, $\varphi(z_p, z_q)$, between the ego-graphs of the provenance graph and those in the query. During the evaluation, we set the threshold τ_{ovp} to the value that produces the highest accuracy in our experiments. The optimum value for τ_{ovp} varies between 0.025 and 0.045 depending on the dataset. Next, we combine the matching ego-graphs to create $\mathcal{G}^*$ and calculate the final matching score, $g(\mathcal{G}^*, \mathcal{G}_Q)$. The overall performance in determining whether the query graph is contained within the target graph results in an AUC value of 99.8. We also conducted an assessment of ProvG-Searcher 's robustness to imprecise queries. Our findings, showcasing its robustness against node and edge removal, as well as its ability to match mutated malware behavior, are discussed in detail in [14].

5.5 Comparison with Previous Approaches

This section presents a comparative evaluation of our technique against other hypothesis-driven threat hunting approaches, specifically Poirot and DeepHunter [72]. As the source codes for these systems are not publicly accessible, we have implemented them to the best of our ability for the purpose of this comparison. To expand upon the tests for detecting attack queries (Sec. 5.4.1) and conduct a more comprehensive evaluation with a larger sample size, we conducted additional tests involving generic system behaviors, similar to Sec. 5.4.2. For this comparison, we generated 5K positive and 5K negative query-target pairs from each dataset and evaluated all techniques on the same data.

Poirot [51]: Poirot is a non-learning-based search method that falls under the category of methods requiring the entire search computation to be executed at query time. However, this constraint makes Poirot less suitable for handling large datasets or supporting highly responsive applications. During operation, Poirot searches for all query nodes in the provenance graph until it finds the first acceptable alignment that exceeds a predefined threshold or exhaustively examines all possible paths. As the graph size or number of queries increases, this extensive search also leads to a tendency to generate false alignments. When evaluating the alignment of query nodes with those on a graph path, the Poirot algorithm calculates an attacker influence score by considering the number of compromised ancestor processes present on the path. However, Poirot's definition of an ancestor process is not clearly established. To address this ambiguity, we conduct experiments using various definitions of ancestor processes, including immediate ancestors, top-most ancestors, and top-most ancestors identified through clone, fork, or execute events. We report the best-obtained values.

We first assess the search complexity of Poirot on Theia and Trace datasets. As these datasets contain only a single instance of some processes in the queried attack behavior (i.e., mail and profile processes), they are quite favorable for Poirot. The execution time of Poirot on these queries has been reported to vary from several minutes [51] to over an hour [17], which can be attributed to the level of optimization in the code. Upon analysis, we determined that the influence score computation requires traversing a large portion of the graph. For instance, the query on the Theia dataset for the Firefox backdoor vulnerability traverses around 40% of all edges and 13% of all nodes. This factor can significantly hinder Poirot's performance, particularly with larger and more complex datasets.

The performance of Poirot in searching generic system behaviors entailed in the test samples associated with the three datasets are presented in Table 3. These results show that ProvG-Searcher and Poirot perform very similarly in correctly matching query system behaviors entailed in positive samples to corresponding target graphs, with ProvG-Searcherslightly outperforming Poirot by around 1% in all cases. However, as shown in the last column of Table 3, Poirot produces a considerably higher false-positive rate on Theia (3%) and Trace (5%) datasets, compared to ProvG-Searcher, which falsely matches only around 0.01% of the neqative samples. On the Cadets dataset, Poirot also yields a low false-positive rate (0.69%), which can be attributed to the relatively simpler nature of the graphs in the Cadets dataset (Table 1). It is important to note that both ProvG-Searcher and Poirot significantly outperformed other techniques used for testing.

IsoRankN [40]. IsoRankN is a network alignment method designed to identify equivalent nodes or edges in multiple networks. It achieves this by optimizing a global objective function that measures network similarity through node alignment. The method utilizes a spectral clustering algorithm to break down the networks into smaller subgraphs and aligns them based on their topological

properties. However, the results indicate that this method is not highly effective for the subgraph matching task.

DeepHunter [72]: The subgraph matching technique most similar to ours is DeepHunter, which utilizes an NTN to learn the subgraph relationship between graph embeddings. DeepHunter relies on EDR alerts to reduce the provenance graph by identifying query seed nodes close to IoCs and to minimize falsely matching behaviors. However, as we don't have access to graphs created around the IoCs, we ran it on graphs created using our method. To compare with DeepHunter, we integrated its search component with our graph creation module and fine-tuned its parameters for each dataset. We trained the model until its performance plateaued and conducted the tests accordingly. The corresponding results are presented in Table 3. These results clearly illustrate that ProvG-Searcher surpasses DeepHunter by approximately 15-25% in terms of accuracy.

SimGNN [5]: SimGNN uses a more advanced architecture than DeepHunter that also incorporates node-alignment into its decision. For our tests, we optimized its parameters for each dataset. Our results show that, despite this additional modeling step, SimGNN performs similar to DeepHunter, with SimGNN performing slightly worse, consistent with the findings reported in [5] . DeepHunter slightly outperforming it outperforming in on Theia dataset by consistent with the findings reported by DeepHunter.

Runtime Analysis: Our technique's offline phase incurs a fixed one-time cost for each dataset, involving the creation of training samples and model training. In our evaluation, we determined that the collective steps of the offline phase take roughly 400, 287, and 250 minutes for Theia, Trace, and Cadets datasets, respectively. As part of the offline stage, the trained model is also employed to precompute subgraph embeddings for all ego-graphs within the provenance graph. This process averages around 11 minutes for each dataset. The online phase involves generating embeddings for query graphs and performing searches for matching subgraphs using precomputed embeddings. Across all datasets, our technique completes processing 10,000 samples in approximately 48 seconds. In contrast, Poirot's execution times for the same set of queries are considerably longer, requiring 1166, 392, and 327 minutes for Theia, Trace, and Cadets datasets, respectively.

5.6 Ablation Study

To assess the impact of various subcomponents of ProvG-Searcher on its performance, we conducted an ablation study. In this regard, the graph creation component has three processing steps that are indispensable for the operation of our technique. These include the graph simplification (GS), dependence explosion mitigation (DEM), and the graph partitioning (GP) steps. The GS and DEM steps play a crucial role in reducing the size of the graphs, making them suitable for graph representation learning approaches. On the other hand, the GP step is responsible for generating the required ego-graphs, facilitating the subsequent operations of our system. As shown in Table 1, the size of ego-graphs is unmanageably large for all datasets, except for the Trace dataset, before the application of the DEM step. Consequently, the ablation study involving the DEM step, in addition to the behavior-preserving reduction (BR) step, is limited to the Trace dataset. In our study, we also evaluate the impact of the subgraph matching (SM) step, which allows ProvG-Searcher to process large query graphs by partitioning them into k-hop ego-graphs.

Table 3: Performance Comparison on Different Datasets

	Method	Metrics					
		Acc.	F1	AUC	Prec.	Recall	FPR
Theia	IsoRankN	63.20	62.85	63.06	63.46	62.26	35.84
	SimGNN	83.28	84.49	90.39	78.77	91.11	24.56
	DeepHunter	83.67	84.42	90.86	80.69	88.53	21.19
	Poirot	97.38	97.44	97.46	95.16	**99.84**	5.07
	ProvG-Searcher	**99.83**	**99.84**	**99.81**	**99.98**	99.69	**0.02**
Trace	IsoRankN	56.47	36.46	55.07	68.36	24.86	11.61
	SimGNN	75.93	78.63	84.33	70.69	88.57	36.72
	DeepHunter	74.93	77.45	83.51	70.36	86.13	36.28
	Poirot	97.99	98.01	98.40	97.03	**99.02**	3.02
	ProvG-Searcher	**99.34**	**99.33**	**99.34**	**99.98**	98.69	**0.01**
Cadets	IsoRankN	62.17	44.15	56.96	84.35	29.90	5.54
	SimGNN	84.50	85.55	90.27	80.10	91.80	22.80
	DeepHunter	84.11	85.21	90.43	79.69	91.55	23.34
	Poirot	98.18	98.16	99.66	99.30	97.05	0.68
	ProvG-Searcher	**99.78**	**99.76**	**99.80**	**99.96**	**99.61**	**0.03**

As part of our study, we repeated the experiments in Sec. 5.3 and 5.4.2 by partitioning the provenance graph into both 3-hop (lines 1-3) and 5-hop (lines 4-7) ego-graphs when generating subgraph representations. The query graphs, respectively, include 3-hop and 5-hop ego-graphs in both cases. Results corresponding to different combination of subcomponents are presented in Table 4. Specifically, the DEM step alone impacts the accuracy by 1.5% and 5.6% (lines 1-2 and 5-6) depending on the size of ego-graphs. Moreover, the results demonstrate that the BR step leads to a substantial increase in accuracy, around 3% (lines 2-3 and 5-6), considering both 3- and 5-hop ego-graphs. The effectiveness of the SM step is demonstrated in lines 5 and 7. In line 5, where the query and ego-graphs have the same size, the SM function is not required. However, in line 7, the query graphs are partitioned into 3-hop ego-graphs before the match is evaluated using the SM function. Consequently, the SM step enables the utilization of smaller graph representations (3-hop ego-graphs instead of 5-hop ego-graphs), resulting in a more precise query matching. This improvement boosted the accuracy by almost 5.5%, increasing it from 93.33% to 98.83%. When combined together, the BR and SM steps yield an additional improvement of 0.5%, pushing the overall accuracy to an impressive 99.34%.

Table 4: Ablation Study (GS: Graph Simplification, DEM: Dependence Exploision Mitigation, GP: Graph Partitioning BR: Behavior-Preserving Reduction, SM: Subgraph Matching)

	System Components	Metrics			
		Accuracy	Precision	Recall	FPR
1	GS+GP (k=3)	90.01	87.37	93.55	13.52
2	GS+DEM+GP (k=3)	91.52	92.96	90.36	7.23
3	GS+DEM+GP+BR (k=3)	95.09	97.46	93.05	2.66
4	GS+GP (k=5)	87.71	83.95	93.26	17.82
5	GS+DEM+GP (k=5)	93.33	97.21	90.21	3.01
6	GS+DEM+GP+BR (k=5)	96.06	93.51	99.00	6.87
7	GS+DEM+GP+SM (k=5)	98.83	98.67	**99.79**	1.31
8	GS+DEM+GP+BR+SM (k=5)	**99.34**	**99.98**	99.69	**0.02**

6 RELATED WORK

6.1 Threat Hunting and Provenance Analysis

Threat hunting is a proactive defense approach where experts continuously search for traces of an unknown attack. Provenance graphs have been utilized in two main ways to expand existing threat hunting capabilities. The first one aims at leveraging the contextual information revealed by a graph representation to discover irregularities and anomalies that may suggest malicious activity. To this objective, one group of work applied statistics-based techniques to identify unlikely events or event chains [23, 43, 69]. Another group of work proposed clustering and learning-based techniques to distinguish between benign and anomalous patterns in the provenance graph. These works applied several intuitions to graph analysis such as the use of graph sketching techniques [20, 49], graph embedding techniques [39, 71], knowledge graph embedding techniques [82, 83], and sequence-based neural embedding methods [1, 41, 69].

The other use of provenance graphs for threat hunting relates to hypothesis-driven investigations. When knowledge about a new threat is obtained from threat intelligence sources and feeds, offering intelligence on the most current tactics, techniques, and procedures utilized by attackers [3], threat hunters will search within their own environment for those specific attack behaviors. In the context of provenance graphs, this search task can be formulated as a graph pattern matching problem where an observed attacker behavior is expressed as a query graph, and its entailment within a provenance graph has to be determined. Given the NP-completeness of exact graph matching, this formulation of the problem calls for the use of approximate methods. Most inexact methods rely on heuristics to select appropriate seed nodes, and then expand to neighboring nodes according to predetermined rules to match the topology as well as node and edge features [31, 42, 51, 57, 63, 64]. To perform this search rapidly, Poirot examines all paths from the seed node to other nodes and retains only those that are more likely to be under the influence of an attacker for alignment by ensuring that the processes along a selected path share a common ancestor in their process tree. With a similar motivation, DeepHunter generates graph embeddings to evaluate the alignment of query graph with subgraphs extracted from the provenance graph and uses a neural tensor network to model the relation between two graph-level embeddings. To reduce the complexity of the search, the provenance graph is reduced by only considering subgraphs extracted around nodes related to suspicious events identified by alerts of an EDR tool.

6.2 Provenance Graph Simplification

The substantial size of provenance graphs presents a significant challenge for conducting timely analyses, thus impeding their practical usability. To address this issue, one strategy is to utilize the limited memory resource efficiently trough adopting compact representations for referencing nodes and edges [26, 28], identifying the relevant portion of the graph for analysis and caching only that part in main memory [24], or losslessly compressing the graph [16]. While these approaches preserve the utility of data, their effectiveness inevitably diminishes as the complexity of the graphs increases.

Therefore, to simplify provenance graphs, several works have proposed preprocessing methods by removing logs that are associated with dead system entities [37], repeated events [79], and events comprising nodes that are disconnected from the backward and forward tracing graph of a symptomatic, point-of-interest event [1] or (EDR) threat alert event [22]. While lossy reduction techniques cannot provide a foolproof guarantee that every analysis task will yield the desired outcome, it's possible to perform the reduction in a way that doesn't compromise the objective of the forensics analysis. To achieve this objective, three reduction techniques have been proposed. The first, causality-preserving reduction, aims to eliminate events that are redundant for causality reasoning [79]. The second technique, dependency-preserving reduction, reduces causality to reachability and removes events (i.e., edges of the graph) more aggressively, provided that system entities required for backward and forward tracing queries can be correctly identified [28]. The third technique, attack-preserving reduction, has been proposed to preserve attack-relevant causal relations while eliminating those related to benign process activities [15, 50]. In contrast, when searching and matching graph patterns in provenance graphs, it is crucial to capture a wide range of local relations, regardless of their frequency of occurrence. Thus, log reduction techniques that aim to preserve causality or dependency relations along traces in a graph are not ideally suited for learning subgraph relationships.

Another key challenge in analyzing provenance graphs is to mitigate the risk of false dependencies. As the number of nodes and edges in the graph increases, the potential dependencies between system entities grow exponentially, giving rise to the dependency explosion problem. The impact of this explosion is more severe for (long-running) high fanout processes and frequently accessed files. Therefore, many works proposed to address the issue of dependence explosion through execution partitioning of graphs [34, 36, 47, 48], tag propagation for information flow tracking [27], and incorporation of system and application logs [25]. An alternative and less costly approach to determining true dependencies between system entities is to create versioned graphs. Versioning allows encoding the time order of events by creating a version of a process or file when its state changes. Several methods have been proposed to reduce the number of node versions and edges while preserving dependencies [6, 28, 55].

6.3 Graph Neural Networks and Subgraph Matching

Inspired by the success of graph learning methods in several prediction tasks, several graph learning-based methods have been recently proposed to solve approximate subgraph matching task. These methods use graph neural networks to generate node level embeddings that encode the neighborhood structures and features of nodes as well as edges. Then, resulting embeddings are used to model the relation between the data and query graphs. When evaluating the match between two graphs, a common approach is using graph embedding models [5, 12, 44]. This involves learning an inductive function that embeds graphs into a vector space such that similar graphs are mapped closely while dissimilar ones are far apart. At the graph embedding stage, the individual node embeddings are pooled together through one of several schemes as the

final graph-level representation [76]. One challenge these models face is reflecting minute structural differences between graphs in the resulting graph-level embeddings. To address this, the larger data graph is partitioned into subgraphs to better emphasize node-level information, and the evaluation is done between the query graph and those smaller subgraphs comprising the data graph.

An alternative approach to graph embedding is to allow a model to incorporate more complex relation information instead of independently mapping each graph to a vector. This is realized by jointly computing a similarity score between a pair of graphs. With the cross-graph matching approach, the reasoning of the relation between two graphs is made by modeling the node-to-node interactions [35, 38, 58, 73, 77] or by modeling the graph-to-graph interactions [5]. Since the graph-learning model attends a pair of graphs jointly, cross-graph matching methods are potentially stronger than the graph embedding models, and they can be made more resistant to slight variations between graphs. This gain, however, comes at the cost of increased computational complexity. Because the similarity computation has to be done in an online setting (i.e., after the query graph is presented) where the number of pairs of query and target subgraphs that need to be compared depends on the size of the data graph. Therefore, for large-scale graph instances, cross-graph matching methods are not feasible. In contrast, graph embedding models allow operating in a batch setting by precomputing the embeddings for data graphs, thereby limiting the online computation step to computing a similarity measure between a pair of embeddings. In the context of threat hunting, both the size of the provenance graphs and the potential number of queries are considerably large. Consequently, a graph embedding approach emerges as a more suitable solution to perform search in provenance graphs.

7 LIMITATIONS AND CONCLUSIONS

The design of ProvG-Searcher emphasizes learning diverse system behaviors, rather than focusing on the prevalence of those behaviors within the system. This approach can lead to ambiguous representations for queries involving repetitive activities. For instance, a malicious software transferring or encrypting large number of files under a directory may be represented by only a few write events to different file object abstraction categories, making it harder to discern the underlying pattern. A potential solution to address such cases is to train a new model tuned to learn frequently exhibited behaviors. Queries involving such repetitive behaviors can then be searched using this specialized model.

For efficiency and accuracy in matching, our technique performs search on reduced provenance graphs. Part of this reduction involves abstracting system entities by assigning them higher-level category labels. Although this abstraction based on entity's function enhances matching capability, the loss of specificity (such as process and file names, IP addresses) may contribute to false matches. Results from real-life queries describing DARPA TC attack behaviors demonstrate that our technique can accurately match abstracted entities on unreduced provenance graphs. However, for much larger graphs with increasing number of queries, this may lead to additional false matches, such as when two queries represent different behaviors but map to the same abstract behavior representation. Our approach can be adapted to this search mode by maintaining additional indexing information alongside the precomputed and stored ego-graph embeddings. By including information on the anchor process of an ego-graph, we can limit the evaluation of subgraph relationship only to matching entities.

One area that may offer further improvement to our technique involves utilizing deeper GNNs to better leverage their increased expressive power. Our initial tests reveal that using three or four layers yields similar performance. Using higher number of layers requires handling larger graphs during the graph creation phase and reducing batch sizes to accommodate target and query graph pairs within GPU memory during training. Exploring deeper networks will be considered in our future work. Additionally, ensuring the generalizability of our technique to support search across different systems requires an operating system-agnostic representation for system entities and edge types. This aspect will also be considered in our future work.

REFERENCES

[1] Abdulellah Alsaheel, Yuhong Nan, Shiqing Ma, Le Yu, et al. 2021. ATLAS: A Sequence-based Learning Approach for Attack Investigation. In *USENIX Security Symposium*.

[2] Ben Athiwaratkun and Andrew Gordon Wilson. 2018. Hierarchical density order embeddings. *arXiv preprint arXiv:1804.09843* (2018).

[3] MITRE ATT&CK. 2021. MITRE ATT&CK. https://attack.mitre.org. Accessed: February 28, 2023.

[4] Jinheon Baek, Minki Kang, and Sung Ju Hwang. 2021. Accurate learning of graph representations with graph multiset pooling. *arXiv preprint arXiv:2102.11533*.

[5] Yunsheng Bai, Hao Ding, Song Bian, Ting Chen, et al. 2019. Simgnn: A neural network approach to fast graph similarity computation. In *WSDM*.

[6] Adam Bates, Dave Jing Tian, Kevin RB Butler, and Thomas Moyer. 2015. Trustworthy whole-system provenance for the linux kernel. In *USENIX Security Symposium*. 319–334.

[7] Tristan Bilot, Nour El Madhoun, Khaldoun Al Agha, and Anis Zouaoui. 2023. A Survey on Malware Detection with Graph Representation Learning. *arXiv preprint arXiv:2303.16004* (2023).

[8] Fenxiao Chen, Yun-Cheng Wang, Bin Wang, and C-C Jay Kuo. 2020. Graph representation learning: a survey. *APSIPA* (2020), e15.

[9] Meng-Fen Chiang, Ee-Peng Lim, Wang-Chien Lee, Xavier Jayaraj Siddarth Ashok, and Philips Kokoh Prasetyo. 2019. One-class order embedding for dependency relation prediction. In *ACM SIGIR*. 205–214.

[10] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, et al. 2019. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *SIGKDD*. 257–266.

[11] DARPA. 2014. Transparent Computing. http://www.darpa.mil/program/transparent-computing.

[12] Angjela Davitkova, Damjan Gjurovski, and Sebastian Michel. 2021. LMKG: Learned Models for Cardinality Estimation in Knowledge Graphs. *arXiv preprint arXiv:2102.10588* (2021).

[13] Ashita Diwan. 2021. *Representation Learning for Vulnerability Detection on Assembly Code*. McGill University (Canada).

[14] Altinisik Enes, Deniz Fatih, and Sencar Husrev Taha. 2023. ProvG-Searcher: A Graph Representation Learning Approach for Efficient Provenance Graph Search. *arXiv preprint arXiv:2309.03647*.

[15] Pengcheng Fang, Peng Gao, Changlin Liu, Erman Ayday, et al. 2022. {Back-Propagating} System Dependency Impact for Attack Investigation. In *USENIX Security Symposium*. 2461–2478.

[16] Peng Fei, Zhou Li, Zhiying Wang, Xiao Yu, Ding Li, and Kangkook Jee. 2021. SEAL: Storage-efficient Causality Analysis on Enterprise Logs with Query-friendly Compression.. In *USENIX Security Symposium*. 2987–3004.

[17] Peng Gao, Fei Shao, Xiaoyuan Liu, Xusheng Xiao, et al. 2021. Enabling Efficient Cyber Threat Hunting With Cyber Threat Intelligence. In *ICDE*. 193–204.

[18] W. Hamilton, Z. Ying, and J. Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *NIPS*.

[19] William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Representation Learning on Graphs: Methods and Applications. *IEEE Data Eng. Bull.* (2017).

[20] Xueyuan Han, Thomas Pasquier, Adam Bates, James Mickens, and Margo Seltzer. 2020. Unicorn: Runtime provenance-based detector for advanced persistent threats. *arXiv preprint arXiv:2001.01525* (2020).

[21] Wajih Ul Hassan, Lemay Aguse, Nuraini Aguse, Adam Bates, and Thomas Moyer. 2018. Towards scalable cluster auditing through grammatical inference over provenance graphs. In *NDSS*.

[22] Wajih Ul Hassan, Adam Bates, and Daniel Marino. 2020. Tactical provenance analysis for endpoint detection and response systems. In *S&P*. 1172–1189.

[23] Wajih Ul Hassan, Shengjian Guo, Ding Li, Zhengzhang Chen, et al. 2019. Nodoze: Combatting threat alert fatigue with automated provenance triage. In *NDSS*.

[24] Wajih Ul Hassan, Ding Li, Kangkook Jee, Xiao Yu, et al. 2020. This is why we can't cache nice things: Lightning-fast threat hunting using suspicion-based hierarchical storage. In *ACSAC*. 165–178.

[25] Wajih Ul Hassan, Mohammad Ali Noureddine, Pubali Datta, and Adam Bates. 2020. OmegaLog: High-fidelity attack investigation via transparent multi-layer log analysis. In *NDSS*.

[26] Md Nahid Hossain, Sadegh M Milajerdi, Junao Wang, Birhanu Eshete, Rigel Gjomemo, et al. 2017. SLEUTH: Real-time Attack Scenario Reconstruction from COTS Audit Data.. In *USENIX Security Symposium*. 487–504.

[27] Md Nahid Hossain, Sanaz Sheikhi, and R Sekar. 2020. Combating dependence explosion in forensic analysis using alternative tag propagation semantics. In *S&P*. 1139–1155.

[28] Md Nahid Hossain, Junao Wang, R Sekar, and Scott D Stoller. 2018. Dependence-preserving data compaction for scalable forensic analysis. In *USENIX Security Symposium*. 1723–1740.

[29] Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. 2020. Heterogeneous graph transformer. In *Proceedings of The Web Conference 2020*. 2704–2710.

[30] Wenbing Huang, Yu Rong, Tingyang Xu, et al. 2020. Tackling over-smoothing for general graph convolutional networks. *arXiv preprint arXiv:2008.09864* (2020).

[31] Arijit Khan, Yinghui Wu, Charu C Aggarwal, and Xifeng Yan. 2013. Nema: Fast graph search with label similarity. *VLDB Endowment* 6, 181–192.

[32] Samuel T King and Peter M Chen. 2003. Backtracking intrusions. In *SOSP*. 223–236.

[33] T. Kipf and M. Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.

[34] Yonghwi Kwon, Fei Wang, Weihang Wang, et al. 2018. MCI: Modeling-based Causality Inference in Audit Logging for Attack Investigation. In *NDSS*. 4.

[35] Zixun Lan, Limin Yu, Linglong Yuan, et al. 2021. Sub-gmn: The subgraph matching network model. *arXiv preprint arXiv:2104.00186*.

[36] Kyu Hyung Lee, Xiangyu Zhang, and Dongyan Xu. 2013. High Accuracy Attack Provenance via Binary-based Execution Partition. In *NDSS*, Vol. 16.

[37] Kyu Hyung Lee, Xiangyu Zhang, and Dongyan Xu. 2013. LogGC: garbage collecting audit log. In *SIGSAC*. 1005–1016.

[38] Yujia Li, Chenjie Gu, Thomas Dullien, et al. 2019. Graph matching networks for learning the similarity of graph structured objects. In *ICML*. 3835–3845.

[39] Zitong Li, Xiang Cheng, Lixiao Sun, Ji Zhang, and Bing Chen. 2021. A hierarchical approach for advanced persistent threat detection with attention-based graph neural networks. *Security and Communication Networks* 2021 (2021), 1–14.

[40] Chung-Shou Liao, Kanghao Lu, Michael Baym, Rohit Singh, and Bonnie Berger. 2009. IsoRankN: spectral methods for global alignment of multiple protein networks. *Bioinformatics* 12, i253–i258.

[41] Fucheng Liu, Yu Wen, Dongxue Zhang, et al. 2019. Log2vec: A heterogeneous graph embedding based approach for detecting cyber threats within enterprise. In *SIGSAC*. 1777–1794.

[42] Lihui Liu, Boxin Du, Hanghang Tong, et al. 2019. G-finder: Approximate attributed subgraph matching. In *IEEE BigData*. 513–522.

[43] Yushan Liu, Mu Zhang, Ding Li, Kangkook Jee, et al. 2018. Towards a Timely Causality Analysis for Enterprise Security.. In *NDSS*.

[44] Zhaoyu Lou, Jiaxuan You, Chengtao Wen, et al. 2020. Neural subgraph matching. *arXiv preprint arXiv:2007.03092*.

[45] Andreas Loukas. 2019. What graph neural networks cannot learn: depth vs width. *arXiv preprint arXiv:1907.03199* (2019).

[46] Yao Lu, Kaizhu Huang, and Cheng-Lin Liu. 2016. A fast projected fixed-point algorithm for large graph matching. *Pattern Recognition*, 971–982.

[47] Shiqing Ma, Juan Zhai, Fei Wang, Kyu Hyung Lee, et al. 2017. MPI: Multiple Perspective Attack Investigation with Semantic Aware Execution Partitioning. In *USENIX Security Symposium*. 1111–1128.

[48] Shiqing Ma, Xiangyu Zhang, Dongyan Xu, et al. 2016. Protracer: Towards Practical Provenance Tracing by Alternating Between Logging and Tainting. In *NDSS*.

[49] Emaad Manzoor, Sadegh M Milajerdi, and Leman Akoglu. 2016. Fast memory-efficient anomaly detection in streaming heterogeneous graphs. In *SIGKDD*. 1035–1044.

[50] Noor Michael, Jaron Mink, Jason Liu, Sneha Gaur, et al. 2020. On the forensic validity of approximated audit logs. In *ACSAC*. 189–202.

[51] Sadegh M Milajerdi, Birhanu Eshete, Rigel Gjomemo, and VN Venkatakrishnan. 2019. Poirot: Aligning attack behavior with kernel audit records for cyber threat hunting. In *SIGSAC*. 1795–1812.

[52] 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 *S&P*. 1137–1152.

[53] Luc Moreau, Juliana Freire, Joe Futrelle, Robert E McGrath, et al. 2008. The open provenance model: An overview. In *IPAW*. 323–326.

[54] Kiran-Kumar Muniswamy-Reddy and Margo Seltzer. 2010. Provenance as first class cloud data. *SIGOPS* (2010), 11–16.

[55] Thomas Pasquier, Xueyuan Han, Mark Goldstein, Thomas Moyer, et al. 2017. Practical whole-system provenance capture. In *SoCC*. 405–418.

[56] Kexin Pei, Zhongshu Gu, Brendan Saltaformaggio, Shiqing Ma, et al. 2016. Hercule: Attack story reconstruction via community discovery on correlated log graph. In *ACSAC*. 583–595.

[57] Robert Pienta, Acar Tamersoy, Hanghang Tong, and Duen Horng Chau. 2014. Mage: Matching approximate patterns in richly-attributed graphs. In *IEEE BigData*. 585–590.

[58] Indradyumna Roy, Venkata Sai Baba Reddy Velugoti, Soumen Chakrabarti, and Abir De. 2022. Interpretable Neural Subgraph Matching for Graph Retrieval. In *AAAI*, Vol. 36. 8115–8123.

[59] Kiavash Satvat, Rigel Gjomemo, and VN Venkatakrishnan. 2021. Extractor: Extracting attack behavior from threat reports. In *EuroS&P*. 598–615.

[60] Franco Scarselli, Marco Gori, Ah Chung Tsoi, et al. 2008. The graph neural network model. *IEEE transactions on neural networks* 20, 61–80.

[61] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, et al. 2018. Modeling relational data with graph convolutional networks. In *ESWC 2018*. 593–607.

[62] Joerg Thalheim, Pramod Bhatotia, and Christof Fetzer. 2016. Inspector: data provenance using intel processor trace (pt). In *ICDCS*. 25–34.

[63] Yuanyuan Tian, Richard C Mceachin, Carlos Santos, et al. 2007. SAGA: a subgraph matching tool for biological graphs. *Bioinformatics* 23, 232–239.

[64] Hanghang Tong, Christos Faloutsos, Brian Gallagher, and Tina Eliassi-Rad. 2007. Fast best-effort pattern matching in large attributed graphs. In *SIGKDD*. 737–746.

[65] Jacob Torrey. 2020. *Transparent Computing Engagement 3 Data Release*. https://www.darpa.mil/program/transparent-computing

[66] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, et al. 2018. Graph Attention Networks. In *ICLR*.

[67] Ivan Vendrov, Ryan Kiros, Sanja Fidler, and Raquel Urtasun. 2015. Order-embeddings of images and language. *arXiv preprint arXiv:1511.06361*.

[68] Luke Vilnis, Xiang Li, Shikhar Murty, and Andrew McCallum. 2018. Probabilistic embedding of knowledge graphs with box lattice measures. *arXiv preprint arXiv:1805.06627* (2018).

[69] Qi Wang, Wajih Ul Hassan, Ding Li, Kangkook Jee, et al. 2020. You Are What You Do: Hunting Stealthy Malware via Data Provenance Analysis.. In *NDSS*.

[70] Qi Wang, Wajih Ul Hassan, Ding Li, Kangkook Jee, et al. 2020. You Are What You Do: Hunting Stealthy Malware via Data Provenance Analysis.. In *NDSS*.

[71] Su Wang, Zhiliang Wang, Tao Zhou, Hongbin Sun, Xia Yin, et al. 2022. Threatrace: Detecting and tracing host-based threats in node level through provenance graph learning. *TIFS* 17 (2022), 3972–3987.

[72] Renzheng Wei, Lijun Cai, Lixin Zhao, Aimin Yu, and Dan Meng. 2021. Deephunter: A graph neural network based approach for robust cyber threat hunting. In *SecureComm*. Springer, 3–24.

[73] Yuting Wu, Xiao Liu, Yansong Feng, et al. 2019. Relation-aware entity alignment for heterogeneous knowledge graphs. *arXiv preprint arXiv:1908.08210* (2019).

[74] Yulai Xie, Dan Feng, Yuchong Hu, Yan Li, et al. 2018. Pagoda: A hybrid approach to enable efficient real-time provenance based intrusion detection in big data environments. *IEEE TDSC* 17, 1283–1296.

[75] Chunlin Xiong, Tiantian Zhu, Weihao Dong, Linqi Ruan, et al. 2020. CONAN: A practical real-time APT detection system with high accuracy and efficiency. *IEEE TDSC* 19, 1, 551–565.

[76] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826* (2018).

[77] Kun Xu, Liwei Wang, Mo Yu, Yansong Feng, et al. 2019. Cross-lingual knowledge graph alignment via graph matching neural network. *arXiv preprint arXiv:1905.11605* (2019).

[78] Zhiqiang Xu, Pengcheng Fang, Changlin Liu, et al. 2022. Depcomm: Graph summarization on system audit logs for attack investigation. In *S&P*. 540–557.

[79] Zhang Xu, Zhenyu Wu, Zhichun Li, Kangkook Jee, et al. 2016. High fidelity data reduction for big data security dependency analyses. In *SIGSAC*. 504–516.

[80] Hanqing Zeng, Muhan Zhang, Yinglong Xia, Ajitesh Srivastava, et al. 2021. Decoupling the depth and scope of graph neural networks. *NeurIPS* (2021), 19665–19679.

[81] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. 2019. Graphsaint: Graph sampling based inductive learning method. *arXiv preprint arXiv:1907.04931* (2019).

[82] Jun Zeng, Zheng Leong Chua, Yinfang Chen, et al. 2021. WATSON: Abstracting Behaviors from Audit Logs via Aggregation of Contextual Semantics.. In *NDSS*.

[83] Jun Zengy, Xiang Wang, Jiahao Liu, et al. 2022. Shadewatcher: Recommendation-guided cyber threat analysis using system audit records. In *S&P*. 489–506.

[84] Tiantian Zhu, Jinkai Yu, Chunlin Xiong, et al. 2023. APTSHIELD: A Stable, Efficient and Real-time APT Detection System for Linux Hosts. *IEEE TDSC*.

[85] Bo Zong, Xusheng Xiao, Zhichun Li, et al. 2015. Behavior query discovery in system-generated temporal graphs. *arXiv preprint arXiv:1511.05911* (2015).