DeepHunter: A Graph Neural Network Based Approach for Robust Cyber Threat Hunting

Renzheng Wei[1,2], Lijun Cai[1](✉), Lixin Zhao[1], Aimin Yu[1], and Dan Meng[1]

[1] Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
{weirenzheng,cailijun,yuaimin,mengdan}@iie.ac.cn

Abstract. Cyber Threat hunting is a proactive search for known attack behaviors in the organizational information system. It is an important component to mitigate advanced persistent threats (APTs). However, the attack behaviors recorded in provenance data may not be completely consistent with the known attack behaviors. In this paper, we propose DeepHunter, a graph neural network (GNN) based graph pattern matching approach that can match provenance data against known attack behaviors in a robust way. Specifically, we design a graph neural network architecture with two novel networks: *attribute embedding networks* that could incorporate Indicators of Compromise (IOCs) information, and *graph embedding networks* that could capture the relationships between IOCs. To evaluate DeepHunter, we choose five real and synthetic APT attack scenarios. Results show that DeepHunter can hunt all attack behaviors, and the accuracy and robustness of DeepHunter outperform the state-of-the-art method, Poirot.

Keywords: Cyber threat hunting · Robustness · Provenance analysis · Graph neural network · Graph pattern matching

1 Introduction

Threat hunting is a proactive search for intruders who are lurking undetected in the organizational information system. A typical task for a threat hunter is to match system events against known adversarial behavior gained from CTI (Cyber Threat Intelligence). Threat hunting is increasingly becoming an important component to mitigate the Advanced Persistent Threats (APTs), as large enterprises or organizations seek to stay ahead of the latest cyber threats and rapidly respond to any potential attacks.

Existing threat hunting tools (e.g., Endpoint Detection and Response tools, namely EDR) rely on matching low-level Indicators of Compromise (IOCs) or TTP rules (i.e., adversarial Tactics, Techniques, and Procedures). However, simple rules matching methods are prone to high volumes of false alarms, which

J. Garcia-Alfaro et al. (Eds.): SecureComm 2021, LNICST 398, pp. 3–24, 2021.
https://doi.org/10.1007/978-3-030-90019-9_1

leads to the "threat alert fatigue" problem. To overcome this problem, recent works [13,16,31] start to focus on the relationship between IOCs or the correlation among threat alerts. One approach [16] to hunt the ransomware takes advantage of the sequential relationship among IOCs, but the mined sequential patterns typically can not capture long-term attack behaviors.

Recent research suggests that the *provenance graph* can incorporate the long-term historical context and facilitate threat investigation. Based on the provenance graph, many works [13,31] have made advancements to improve the performance of threat hunting. For example, RapSheet [13] leverages dependency relations in the provenance graph to correlate the threat alerts generated by EDR tools, then drops the alerts that do not conform to the APT "kill chain". Poirot [31] improves the accuracy of threat hunting by designing a graph pattern matching algorithm to search the provenance graph for the *query graph* that represents the known attack behavior.

Although the provenance graph can greatly facilitate threat hunting tasks, there still exist several limitations in the existing approaches:

- **Expert knowledge needed.** Existing threat hunting tools or methods need analysts with expert knowledge on known attacks and target systems (e.g., Windows, Linux, macOS, etc.). For example, one needs to estimate the number of entry points of APT attacks when setting Poirot's threshold.
- **Efficiency.** The size of the provenance graph is very large because of the presence of long-term attacks. So the provenance graph-based approaches (i.e., graph matching/searching algorithms) must be efficient.
- **Lack of robustness (most important).** In practice, real attack activities recorded in provenance data are not completely consistent with the known attack behaviors due to auditing/monitoring systems, attack mutations, and random noise. For example, one or more attack steps in CTIs might disappear in the provenance graph. This sort of inconsistency weakens the ability of provenance graph-based methods [13,31,32] to correlate threat alerts. Even worse, the attack provenance graphs might be disconnected, which will bring errors into path-based approaches, i.e., Poirot [31]. We will detail this scenario in Sect. 3.3.

In recent years, graph neural networks (GNNs) have shown great success in handling graph data. Inspired by that, our idea is to view the threat hunting task as a graph pattern matching problem and leverage the powerful GNN model to estimate the matching score between the provenance graph and the given query graph. The graph neural networks have several advantages on the graph pattern matching problem: (1) The graph neural networks naturally excel at efficiency, since modern GPUs can largely accelerate matrix computations by parallel processing. (2) No additional expert knowledge about attacks and target systems is needed, as the graph neural network is trained in an end-to-end manner. What we need is to learn a GNN-based graph pattern matching model that could extract robust graph patterns that are resistant to the inconsistency mentioned earlier. Basically, if both the node attributes (i.e., IOC information) and the graph structures (i.e., dependency relations between IOCs) in the query graph are largely matched in the provenance graphs, the model should output a high matching score and raise alarms.

Unfortunately, there is no off-the-shelf GNN-based architecture that can be simply applied to solve our problem due to two reasons. First, indicators are the entity with multiple attributes (i.e., file names, IP addresses, ports, process names, etc.). Different attributes may have different importance to the graph pattern matching task. Second, the two input graphs for graph pattern matching have different characteristics: The query graph is small and noise-free; The provenance graph is bigger and contains redundant nodes, as the provenance graph represents low-level system events.

To solve these problems in threat hunting, we propose two novel graph neural network structures: the *attribute embedding network* and the *graph embedding networks*. The attribute embedding network encodes attributes into vectors. In particular, we add the attention mechanism to the attribute embedding network. So it could assign higher weights to those attributes that are important to the graph matching task. The graph embedding networks are used for representing graph structures. To better represent distinct input graphs, we employ two different graph embedding networks to encode them, respectively. Specifically, we design one graph embedding network to represent the provenance graph and adopt GCN [22] to represent the query graph. At last, we utilize a powerful relation learning network (i.e., NTN [45]), instead of the traditional Siamese network, to learn a metric for computing the matching score. With this new design practice, we could build the GNN model for graph pattern matching, which is robust against different degrees of inconsistency between the query graph and the provenance graph in threat hunting.

We implemented our proposed technique as **DeepHunter**, a GNN-based graph pattern matching model for threat hunting. To evaluate the accuracy and robustness of DeepHunter, we choose 5 APT attack scenarios with different degrees of inconsistency. Particularly, one of these scenarios (Q5) contains disconnected attack provenance graphs. Experimental results show that DeepHunter can identify all of the attack behaviors in 5 APT scenarios, and it is resistant to various degrees of inconsistency and the disconnected attack provenance graphs. The robustness of DeepHunter outperforms the state-of-the-art APT threat hunting method, Poirot. Moreover, DeepHunter could find attacks that Poirot can not identify under the specific complex attack scenario, Q5+ETW. We also compare DeepHunter with other graph matching approaches, including a non-learning approach and GNN-based approaches. Results show that the performance of DeepHunter is superior to these methods.

In summary, this paper makes the following contributions:

- We propose DeepHunter, which is a GNN-based graph pattern matching approach for cyber threat hunting. DeepHunter can tolerate the inconsistency between the real attack behaviors recorded in provenance data and the known attack behaviors to some extent.
- We design a graph neural network architecture with two novel networks: *attribute embedding networks* and *graph embedding networks*. These two

networks could capture complex graph patterns, including IOC information and the relationships between IOCs.

- We choose 5 APT attack scenarios with different degrees of inconsistency between the provenance graph and the query graph, including 3 real-life APT scenarios and 2 synthetic APT scenarios, to evaluate our approach.
- Our evaluation illustrates that DeepHunter outperforms the state-of-the-art APT threat hunting approach (i.e., Poirot) in accuracy and robustness. Meanwhile, DeepHunter, as a graph pattern matching model, is superior to other graph matching methods (i.e., non-learning-based and GNN-based) in the threat hunting task.

2 Related Work

2.1 Threat Hunting Approaches

In this work, we mainly focus on the threat hunting methods. Poirot [31] is a related work to DeepHunter. We will introduce and compare it with DeepHunter in the evaluation (Sect. 7.2). RapSheet [13] is an approach that could improve EDR's threat hunting ability using the provenance graph analysis. But RapSheet [13] requires complete paths remained in the provenance graph to correlate alerts. Obviously, the disconnected attack provenance graphs will undermine the performance of RapSheet.

For APT detection and investigation, both Holmes [32] and NoDoze [14] correlate alerts using the provenance graphs. To hunt stealthy malware, ProvDetector [49] proposes a graph representation learning approach to model process' normal behavior in provenance graphs. However, these methods assume an accurate normal behavior database for reducing false alarms. We know that the normal behavior model may create a risk of the poisoning attack due to concept drift as benign usage changes. Additionally, all of these methods are path-based approaches. So their robustness could be influenced by the disconnected provenance graphs.

Some methods use IOCs or threat alerts as a clue to identify attack behaviors (i.e., zero-day attack [47] and C&C [35]). However, these methods overlook the relationship between indicators or alerts. So it could bring high false positives.

2.2 Provenance Graph Analysis

Provenance graph analysis is widely applied to the APT attack detection [51], forensic analysis [18], and attack scenario reconstruction [17,38], etc. Recent works [15,32] seek to bridge the semantic gap between low-level system events and high-level behaviors. Many recent works (i.e., Morse [18], BEEP [24], MPI [28], and OmegaLog [15], etc.) are proposed to address the dependency explosion problem in provenance graphs. StreamSpot [29] views the provenance graph as a temporal graph with typed nodes and edges, then proposes a graph sketching algorithm for anomaly detection.

2.3 Graph Matching Approaches

Graph pattern matching and graph similarity computation have been studied for many real applications [9,25]. The graph pattern match problem is NP-complete. NeMa [21] is a neighborhood-based heuristic algorithm, which uses optimization techniques to improve the efficiency. Other works [31,39,48] are path-based approaches, which could not resistant to disconnected attack provenance graphs. Recently, many graph neural networks [4,5,26,50] have been proposed for graph pattern matching. However, these approaches do not take into account the size difference between the query graph and the provenance graph. Additionally, the graph characteristics present new challenges to existing graph neural networks, such as being typed, directed. We compare three of them in the evaluation.

3 Background and Motivation

In this section, we first introduce the background knowledge of the provenance graph and the query graph (Sect. 3.1). Then, we briefly illustrate several common motivating situations where the threat hunting approach calls for high robustness (Sect. 3.2). Finally, taking an APT attack scenario with disconnected attack graphs as an example, we illustrate how this scenario affects existing methods and explain why DeepHunter can resist this situation (Sect. 3.3).

3.1 Background

Provenance Graph. Provenance graph is generally a directed acyclic graph (DAG) [11], where the nodes represent system entities, and the edges represent the dependency relation between these entities. There are two types of nodes in provenance graphs: subjects, which represent processes, and objects, which represent other system entities such as files, Windows registry, and network sockets, etc. Subject node's attributes include process name, command line arguments. Object node's attributes include file names, IP addresses, ports, etc. Table 1 shows the nodes and edges we consider in this work. Provenance graph can represent dependencies between system events.

Table 1. A summarization of nodes and edges in provenance graphs and query graphs.

Subject type	Object type	Attributes	Relations
Process	Process	Name, Augments	Fork, Start
	File	File name	Read, ImageLoad, Write
	Socket	Src/dst IP, Src/dst port	Recv, Send
	Registry	Key name	Write

Query Graph. The query graph G_q in our work can be constructed by manually or automatically [19,27,53] extracting IOCs together with the relationships among them from CTIs (including human-written reports or other threat intelligence feeds with structured standard formats (e.g., STIX [34], OpenIOC [8] and MISP [33])). Both nodes and edges of the query graph are the same as the provenance graph, as shown in Table 1. We set the node's or edge's attributes to null if CTIs do not include the corresponding information.

3.2 Motivating Situations

To motivate our work, we introduce several common situations that could lead to inconsistency or disconnected provenance graphs, which will weaken the existing methods' threat hunting ability. Firstly, provenance systems (e.g., Spade [10]) that are developed for recording system events in the application layer may overlook certain attack activities. For instance, Spade does not trace system activities until user space is started. Hence, if the attack occurred before the tracing of Spade, an incomplete attack provenance graph would be generated. Fortunately, the whole-system provenance trackers (like Hi-Fi [40], LPM [6], and CamFlow [36]) can overcome this problem. Indeed these whole-system provenance trackers begin recording system activities in the early boot phase as the $INIT$ process starts.

Secondly, even if the whole-system provenance system is applied, some targeted attacks could also lead to inconsistency or disconnected attack provenance graphs. Taking the microarchitectural side-channel attack as an example, there is no connection between the attacker process and the victim process in the provenance graph. Coordinated attacks could generate disconnected attack provenance graphs as well. If attackers control multiple entry points of a compromised system and coordinate to achieve an operational goal, each entry point may correspond to an isolated attack graph.

Finally, attack mutations (or inaccurate CTIs) are another reason that incurs the inconsistency or disconnected attack provenance graphs. In the next section, we present an APT scenario with attack mutations to illustrate its influence to threat investigation.

3.3 An APT Attack Scenario

Recently, cryptocurrency mining malware is one of the most prevalent threats in the wild. Figure 1 describes a typical cryptominer's progression, including the EternalBlue exploitation stage, the persistence stage, and the cryptocurrency mining stage. At its exploitation stage, *wininit* is responsible for configuring and reconnaissance scans. *svchost* exploits the EternalBlue vulnerability for propagation. At the persistence stage (persistence I), *spoolsv.exe* creates an executable binary and adds its path to the "run keys" in the Windows registry. At the last stage, the cryptominer process *minner.exe* is started.

Now, let's consider what happens if the attacker changes the persistence techniques. For example, the attacker adopts an alternative persistence technique II

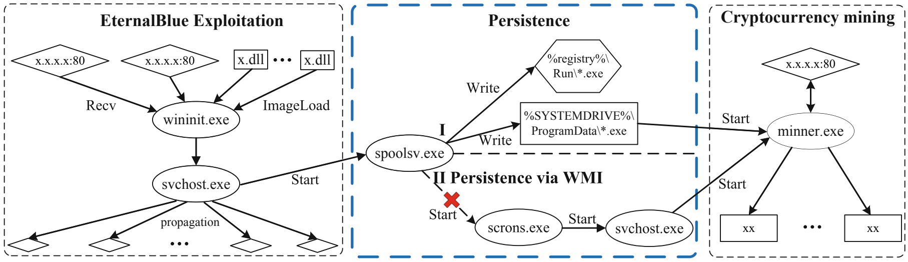

Fig. 1. Query graph of synthetic APT attack scenarios illustrated in Sect. 3.3. Ovals, diamonds, rectangles, and hexagons represent processes, sockets, files, and registry entries, respectively. Many nodes and relations are not shown in the figure for clarity.

(persistence via WMI in Fig. 1). WMI is a preinstalled system tool and it can achieve fileless attacks [12,30]. We further assume that the running provenance system (not a whole-system provenance system) can not capture the dependency between the *spoolsv.exe* process and the *scrons.exe* process (which is the host process of WMI script). Hence, the connection between the EternalBlue exploitation stage and the cryptocurrency mining stage is broken in attack provenance graphs.

Note that the query graph used by analysts is the EternalBlue exploitation stage, the cryptocurrency mining stage, and the upper part of the persistence stage (persistence technique I) in Fig. 1. So the behavior recorded in the attack provenance graph is inconsistent with the given query graph. Besides, the attack provenance graph is disconnected. This situation makes both threat hunting and forensic investigation more difficult. In Sect. 7.2, we will show that this attack mutation can seriously impair the existing threat hunting approach (i.e., Poirot). Additionally, existing provenance graph-based threat correlation methods, like [13,14,32], will definitely lose the correlation between the alerts of the exploitation stage and the alerts of the cryptocurrency mining stage. And the path-based anomaly scores (e.g., rareness score [14,49] and threat score [13]) may also be affected by disconnected attack provenance graphs.

In contrast, DeepHunter is robust against this attack mutation. Intuitively, although there exist inconsistencies and disconnected attack graphs in this scenario, most node attributes and the main graph structures are preserved. DeepHunter can learn robust graph patterns from training data which are resistant to inconsistencies. We detail the design of DeepHunter in Sect. 5.

4 Design Overview and Challenges

4.1 Graph Pattern Matching for Cyber Threat Hunting

We aim to determine if a provenance graph and a given query graph represent the same attack behaviors for a threat hunting task. In this work, we formulate the threat hunting task as a graph pattern matching problem.

Given a query graph G_q, the output is a matching score s of (G_q, G_p^i), where $G_p^i \in S = \{G_p^1, G_p^2, \ldots, G_p^N\}$, S is the set of provenance graphs. Our goal is to learn a graph matching model $\mathcal{M}$, where $\mathcal{M}(G_p, G_q) = 1$ indicates that the provenance graph G_p and the query graph G_q represent the identical behavior; otherwise, $\mathcal{M}(G_p, G_q) = -1$ indicates that they are different. The graph pattern matching model $\mathcal{M}$ must meet three requirements: ① No expert knowledge needed; ② High efficiency (Graph pattern matching is NP-complete in the general case.); ③ High robustness.

As aforementioned, using a graph neural network to extract graph patterns and further compute matching scores is particularly appealing, since it can learn a graph matching model without expert knowledge. Also, once the graph matching model is learned, the matching score can be efficiently computed, and thus we no longer rely on any expensive graph pattern matching algorithms.

4.2 Challenges and Solutions

It has been demonstrated that graph neural networks can learn complex graph patterns for downstream tasks, such as binary code similarity detection [52] and memory forensic analysis [46]. In this work, we need the graph neural network to extract graph patterns for matching two graphs. In particular, the graph patterns should be composed of node attributes and graph structures. We show the challenges of designing graph neural networks and our corresponding solutions as follows.

Challenge 1. How to represent node attribute information effectively? There are multiple node types in graphs, each node has many attributes, and different attributes may have different importance to the graph pattern matching task. Previous work [29] considers the provenance graph as a heterogeneous graph and searches the heterogeneous graph following the meta-paths. However, constructing the meta-path needs expert knowledge on the target systems.

We propose the attribute embedding network (detailed in Sect. 5.1) to represent the node's attributes. We treat the node type (e.g., process, file, socket, etc.) as one of a node's attributes and employ the attention mechanism to automatically learn which attributes contribute most to the graph matching task.

Challenge 2. How to represent graph structures effectively? Previous graph pattern matching models [4,25,50] utilize the same neural network structure to represent both input graphs. But the characteristics of two input graphs for threat hunting are distinct, as mentioned in Sect. 1.

We adopt two different graph neural networks: One is GCN for the query graph, and the other is specially designed to represent the provenance graph structure. We introduce them as the graph embedding networks, as detailed in Sect. 5.2.

5 DeepHunter's Graph Pattern Matching Model

5.1 Attribute Embedding Network for Encoding Node's Attributes

The goal of the attribute embedding network is to obtain the input feature h_u^0 for each node u, which incorporates $u's$ attributes information. Specifically, we first generate an embedding v_i for each attribute i of the node u (as depicted on the left of Fig. 2), and then compute $u's$ input feature h_u^0 by aggregating $u's$ attribute embeddings (as depicted on the right of Fig. 2).

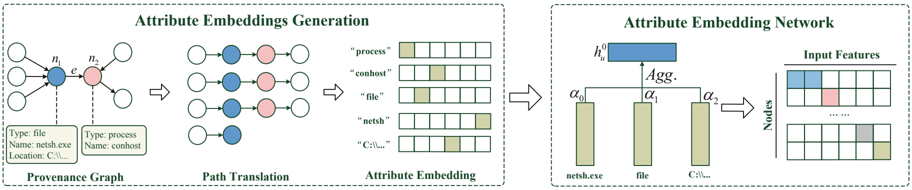

Fig. 2. The process of generating input features.

To obtain the attribute embedding v_i, inspired by the path embedding method of ProvDetector [49], we view a path in the provenance graph as a sentence and then adopt an unsupervised NLP model (word2vec [23]). Specifically, we first translate paths in the provenance graphs into sentences which consist of attributes. For example, the colored nodes n_1, n_2, and the edge e between them in the provenance graph of Fig. 2 can be translated into a sentence as follows: Process *conhost* reads file *netsh.exe* in $C : \backslash\backslash Windows \backslash\backslash System32$. Then we feed the sentences into a word2vec model to learn the vector representation v_i for each attribute i.

We represent a node $u's$ input feature as the aggregation of its attribute embeddings v_i. Common aggregation functions include *sum* and *average*. However, for the graph pattern matching task, the importance of each node attribute may be different. Hence, we use the attention mechanism to learn the weight for each attribute of a node. Specifically, we compute node $u's$ input feature h_u^0 by

$$h_u^0 = \sum\nolimits_{i \in A_u} \alpha_i v_i, \tag{1}$$

where A_u is the attribute set of the node u, α_i is the weight of the $i-th$ attribute, v_i is the embedding of attribute i.

5.2 Graph Embedding Networks for Encoding Graph Structures

Graph embedding networks aim to represent graph structures of both the query graph and the provenance graph. There are two stages in the graph embedding networks: the node-level embedding stage and the graph-level embedding stage. We use two different graph embedding networks to encode the provenance graph G_p and the query graph G_q respectively.

Stage 1: Node-Level Embedding. It is the node embedding method that leads to the difference between the query graph embedding network and the provenance graph embedding network. We adopt the existing graph convolutional network (GCN) [22] to embed the nodes in the query graph. As a matter of fact, the query graph is usually small and noise-free, which can be handled by GCN. To embed the nodes in the provenance graphs, we design a provenance graph embedding network (shown in Fig. 3) which is capable of dealing with the redundant nodes while remaining the key information for matching the query graph. The provenance graph node embedding network is detailed as follows.

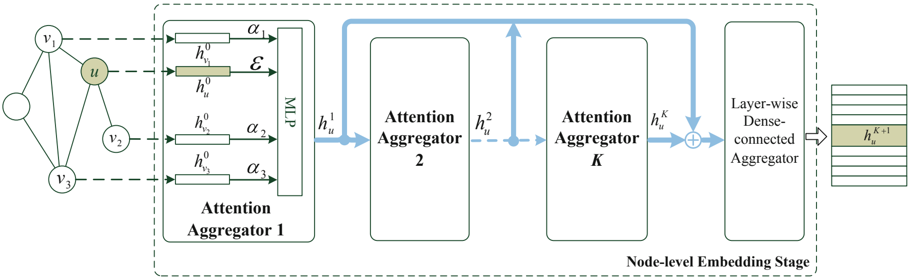

Fig. 3. Provenance graph node-level embedding network structure.

Firstly, we design a GNN layer, called *attention aggregator*, which could aggregate information from neighbors for the target node. Note that, for a node u in the provenance graph, different neighbors may have different importance for graph matching when incorporating their node features into node u. In particular, the redundant neighbors should be assigned a lower importance value to reduce their impact on $u's$ hidden representation h_u, while the nodes that can match the corresponding ones in the query graph should be assigned a higher importance value. For this purpose, we adopt another attention mechanism that can learn weights for neighbors of node u. Formally, the hidden representation h_u outputted by the layer k can be computed via a neural aggregation function that is achieved by

$$h_u^k = MLP(\epsilon^{(k)} h_u^{k-1} + \sum\nolimits_{v \in N_u} \alpha_v h_v^{k-1}), \tag{2}$$

where h_u^k is the output of layer k of the provenance graph embedding network, and it is the hidden representation of node u; α_v is the attentional weight of node v ($v \in N_u$, where N_u is the set of node $u's$ neighbors).

To aggregate the information into h_u from distant nodes, we then add more layers defined by Eq. 2. The number of layers, K, means that the GNN can aggregate information from u's $K - hops$ neighbors. However, simply adding layers may squash exponentially-growing information (including noise) into fixed-size vectors. To address this issue, we adopt the Layer-wise Dense-connected Aggregator which is proposed by [50]. This strategy is formulated as follows:

$$h_u^{K+1} = MLP([h_u^0; h_u^1; \dots h_u^K]), \tag{3}$$

where $[\cdot;\cdot]$ is the feature concatenation operation.

Stage 2: Graph-Level Embedding. Now we obtain the node embedding h for each node in both the query graph and the provenance graphs. How to generate a low-dimensional embedding vector for a graph using node embeddings? In this work, we adapt the Global Context-Aware Attention strategy proposed in SimGNN [4] to obtain the graph-level embedding h_G. Intuitively, nodes that are similar to the global context will be assigned larger weights, which allows the corresponding node embeddings to contribute more to the graph-level embedding. Different from SimGNN, we normalize the weights into 1, because we do not want the graph size to affect the calculation of the matching score. Hence, we replace the sigmoid function in SimGNN with a softmax function $\sigma(\boldsymbol{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$. This graph-level embedding is formally represented by the following equation:

$$\begin{aligned} h_G &= \sum\nolimits_{u=1}^{N} \sigma(h_u c) h_u \\ &= \sum\nolimits_{u=1}^{N} \sigma(h_u \tanh((\frac{1}{N} \sum\nolimits_{m=1}^{N} h_m) W)) h_u, \end{aligned} \tag{4}$$

where N is the number of nodes in a graph, $\tanh(\cdot)$ is a activation function, W is the trainable parameters.

5.3 GNN-Based Architecture for Graph Pattern Matching

Based on the attribute embedding network and the graph embedding network, DeepHunter's graph pattern matching model could learn robust graph patterns. The framework of DeepHunter's graph pattern matching model is shown in Fig. 4. It consists of two branches. The upper branch of Fig. 4 deals with CTI information, and the lower one is for provenance data. At the beginning of each branch, the query graph and the provenance graph are constructed. Then both of them are fed into our GNN-based models.

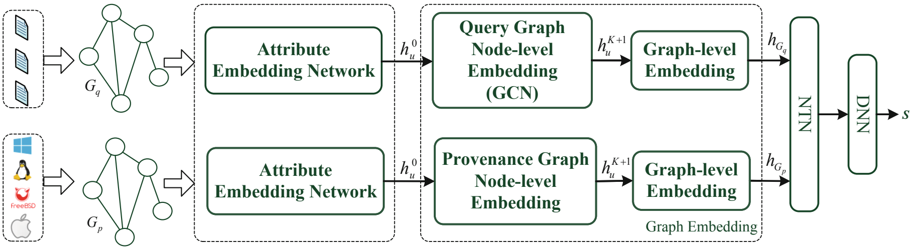

Fig. 4. The framework of DeepHunter's graph pattern matching model.

Given the output of two branches, h_{G_q} and h_{G_p}, many existing graph matching models adopt the Siamese architecture [7] to learn the relation between them. However, the Siamese architecture that directly computes the inner product of h_{G_q} and h_{G_p} is too simple to model the complex relation. Instead, we employ Neural Tensor Network (NTN), which is a powerful relation learning network, to replace the inner product operation. We compare NTN and the traditional Siamese architecture in Sect. 7.3.

After the NTN layer, we connect multi-layer dense neural networks (DNNs) and output the graph matching score s. At last, to compute the loss, we compare s against the ground-truth label using the following mean squared error loss function:

$$\mathcal{L} = \sum_{(G_{p_i}, G_{q_i}) \in \mathcal{D}} (\hat{s} - s(h_{G_{p_i}}, h_{G_{q_i}}))^2, \tag{5}$$

where $D = \{(G_{p_1}, G_{q_1}), (G_{p_2}, G_{q_2}), ...\}$ is the training dataset.

We train the proposed model in an end-to-end way. We leverage stochastic gradient descent to estimate parameters. After a number of training epochs, the loss value will be small and stable, the accuracy of validation data will be high, which demonstrates that the model is trained well.

6 Implementation

6.1 Provenance Graph Reduction

In practice, due to hosting long-term system logs is prohibitively expensive, analysts attempt to reduce the provenance graph and yet preserve the quality of threat hunting [13]. In this work, we prune the provenance graph as follows.

First, we leverage the MITRE ATT&CK TTPs and the IOCs to generate suspicious events. Specifically, DeepHunter uses the EDR tool (i.e., BLUESPAWN [44]), which provides matching rules to detect MITRE ATT&CK TTPs. Besides, DeepHunter also matches the IOCs (extracted from threat intelligence, such as APT reports) using regular expressions. The events identified by both the EDR tool and the IOC matching are regarded as suspicious events.

We then propose the provenance graph reduction algorithm (Algorithm 1), which could prune the provenance graphs based on the suspicious events. Inspired by Poirot [31], we select *seed nodes* from the nodes that match the IOCs. For example, suppose IOC α has x matched nodes in provenance graphs, IOC β has y matched nodes, and IOC γ has z matched nodes. If $z = min\{x, y, z\}$, then these z nodes are *seed nodes*. We start from a *seed node* and execute $adaptiveBFS$ searching on the provenance graphs. A suspicious subgraph generated by the graph reduction algorithm could cover all IOCs' alerts.

The $adaptiveBFS$ is an adapted Breadth-First Search (BFS) algorithm. Specifically, during BFS on the provenance graph, only the nodes related to suspicious events and the process node could be visited.

Algorithm 1: Provenance Graph Reduction Algorithm

```
Input: Provenance Graphs: $G_p$, Indicators Set: $I$, Matched Nodes Set: $P$;
Output: Suspicious Subgraphs: $SuspGraphs$
Function ExpandSearch($SeedNodes$, $Susp$):
    foreach $node \in SeedNodes$ do
        $start\_node \leftarrow node$; $subgraph \leftarrow adaptiveBFS(start\_node, P)$;
          $Susp \leftarrow ComposeGraph(Susp, subgraph)$;
        if $Susp$ contains all indicators in $I$ then
            Add $Susp$ to $SuspGraphs$
        else
            $remain\_nodes \leftarrow$ seed nodes from $P$ that are not matched with any
              indicators in $I$; $ExpandSearch(remain\_nodes, Susp)$;
        end
    end
    return $SuspGraphs$
```

Obviously, the suspicious subgraphs generated by our provenance graph reduction algorithm contain lots of false positives (the threat alert fatigue problem). Therefore, it is still necessary for analysts to use our graph pattern matching model (Sect. 5) to calculate the matching score.

6.2 Training Data Generation

Training DeepHunter requires a large number of positive samples (G_{p_i}, G_{q_i}) $(\mathcal{M}(G_{p_i}, G_{q_i}) = 1)$ and negative samples (G_{p_i}, G_{q_i}) $(\mathcal{M}(G_{p_i}, G_{q_i}) = -1)$. The query graph can be considered as a summarization of its corresponding provenance graph. Therefore, we use the graph summarization techniques to generate the matched query graph G_q for each provenance graph G_p. We also add random noise to improve the robustness. We detail the training data generation method as follows.

Firstly, we extract a subgraph as G_{p_i} from the provenance graphs. Specifically, we start from a process node and use DFS on the provenance graph. We limit the length of the paths, which is less than 4. Then we refine the G_{p_i} using two graph summarization rules as follows:

- Merge process nodes that have the same process name;
- Remove duplicate paths. If two paths are duplicates (i.e., two sequences of node name are equal), only one is reserved.

Then we add noise to G_{p_i} by:

- randomly dropping edges or object nodes on G_{p_i};
- randomly removing one or more attributes of a node.

After the above two steps, we generate a G_{q_i} for the G_{p_i}. So (G_{q_i}, G_{p_i}) is a positive sample for training.

Table 2. APT attack scenarios description and the source of query graphs.

Scenario	Short description	Query graph source
Q1+CADETS	A Nginx server was exploited and a malicious file was downloaded and executed. The attacker tried to inject into sshd process, but failed	DARPA TC 3 reports
Q2+TRACE	The Firefox process was exploited and established a connection to the attacker's operator console. The attacker downloaded and executed a malicious file	DARPA TC 3 reports
Q3+TRACE	A Firefox extension (a password manager) was exploited. A malicious file was downloaded and executed to connect out to the C&C server	DARPA TC 3 reports
Q4+ETW	Detailed in Sect. 3.3	Fig. 1 with persistence I
Q5+ETW	The attack mutation of scenario Q4	Fig. 1 with persistence II

At last, we construct the negative sample (G_{q_j}, G_{p_i}) by randomly combining G_{p_i} and G_{q_j}, where $i \neq j$. By doing this, we simulate the situation where most of the node attribute information and the main graph structure of G_{p_i} is preserved in G_{q_i}.

7 Evaluation

7.1 Attack Scenarios and Experimental Setup

To evaluate the efficacy of DeepHunter, we utilize provenance data which contain 5 APT attack scenarios, including 3 real-life APTs(DARPA TC engagement 3) and 2 synthetic APTs. For each of the attack scenarios, the corresponding query graph is also provided. To simulate real-world threat hunting, the query graphs we used in the evaluation are either generated by the third-party or constructed based on the public APT reports. The description of APT scenarios and the source of corresponding query graphs are shown in Table 2.

Inconsistency Scores. Before evaluating robustness, we define three inconsistency scores to quantify the degree of the inconsistency between the query graph and the corresponding provenance graph. Specifically, we compute graph edit distance (GED) and the number of *missing nodes* and *missing paths*. GED measures the cost that transforms G_q into G_p. We adopt a graph matching toolkit [20,43] to calculate GED and normalize [41] the GED scores for different graph sizes. The *missing node* of the query graph is the node that we cannot find its alignments in provenance graphs. The *missing path* means that for an edge from node i to j in the query graph, there is no path from the nodes aligned to i to the nodes aligned to j in provenance graphs. Table 3 shows the inconsistency scores of the scenarios in Table 2. We can see from Table 3 that the chosen scenarios contain different degrees of inconsistency.

Table 3. Inconsistency scores of different scenarios. The values in parentheses on the second and third columns are the number of missing nodes and paths, respectively.

Scenario	Missing nodes (%)	Missing paths (%)	GED
Q1+CADETS	0	0	0.192
Q2+TRACE	0	6.25%(1)	0.303
Q3+TRACE	0	16%(4)	0.504
Q4+ETW	3.8%(1)	4%(1)	0.454
Q5+ETW	21.4%(6)	20%(7)	0.557

Experimental Setup. The provenance data from DARPA are collected by two provenance systems: CADETS [2] and TRACE [3]. Besides, we synthesized attacks in scenarios Q4+ETW and Q5+ETW on Windows 7 32 bit systems. The provenance data of Q4+ETW and Q5+ETW, including benign system activities and attack behaviors, were collected by our provenance system which is based on Windows ETW [1].

We employed the gensim [42] Python library to obtain the attribute embeddings v_i (detailed in Sect. 5.1). We implemented the proposed graph neural network model using PyTorch [37]. We trained whole neural network-based models using 2 Nvidia Tesla P4 GPU. Other experiments (e.g., provenance graph construction, graph reduction, etc.) are conducted on a server with two Intel Xeon E5-2630 v3 CPUs and 128 GB memory running CentOS system.

Datasets. We generated a dataset for each provenance system and named the dataset after the provenance system. We used the graph reduction method illustrated in Sect. 6.1 to prune the provenance graph. The generated subgraphs (i.e., test graphs) were manually labeled based on the corresponding reports' timestamp. We also generated training graph pairs using the method detailed in Sect. 6.2. The characteristics of our datasets are shown in Table 4.

Table 4. The characteristics of graph datasets used in our evaluation.

Dataset	Raw graph size	# of test graphs		# of training graphs
		Benign	Attack	
CADETS	904 MB	10	1	150,000
TRACE	22.5 GB	9	6	150,000
ETW	40 GB	105	10	300,000

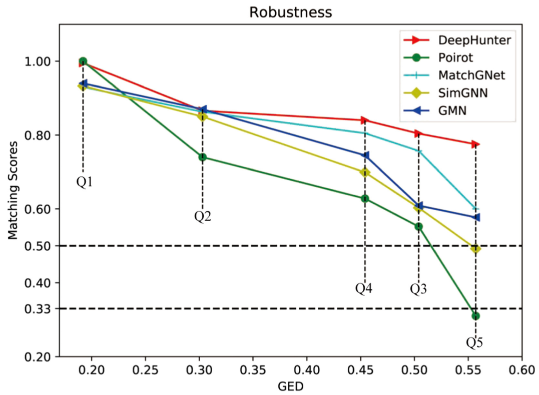

Fig. 5. Axis x: graph edit distance between G_q and G_p of each scenario in Table 2; Axis y: matching scores between G_q and G_p of each scenario. Q1–Q5 represent attack scenarios

7.2 Robustness

We evaluate the impact of inconsistency on DeepHunter and the state-of-the-art Poirot. We also analyze why Poirot fails in scenario Q5+ETW which contains disconnected attack provenance graphs.

State-of-the-Art Poirot. Poirot is a heuristic graph pattern matching algorithm that can compute the graph alignment score between the query graph and the provenance graph. Poirot searches for aligned nodes in the provenance graphs according to the $information\ flows$ in the query graph. During the search, Poirot omits the paths that are impossible to be adopted by attackers.

Experimental Results. We compare DeepHunter with Poirot using all scenarios in Table 2. The matching scores calculated by Poirot, DeepHunter, and other GNN-based graph matching models are shown in Fig. 5. We can see that all matching scores calculated by DeepHunter are greater than the threshold (which is 0.5). This result shows that the accuracy of DeepHunter can be guaranteed in scenarios where there exist various degrees of inconsistency.

Moreover, as the degree of the inconsistency increases, all matching scores decrease. But the curve of DeepHunter is more stable. On the contrary, the curve of Poirot drops faster than the GNN-based graph matching models. Even worse, the matching score calculated by Poirot is less than its threshold in the most inconsistent scenario Q5+ETW , which means that Poirot fails to identify this attack.

Additionally, we detail the false positive results of DeepHunter and Poirot in Table 5. The results can demonstrate that the high robustness of DeepHunter is not built upon false positives.

Why Does Poirot Fail? When searching on the disconnected attack provenance graphs in Q5+ETW, the paths which start from nodes belonging to the

Table 5. False positive results of DeepHunter and Poirot.

Dataset	CADETS	TRACE	ETW
# of test graphs	11	15	115
# of FPs (DeepHunter)	0	0	1
# of FPs (Poirot)	0	1	2

EternalBlue exploitation stage to nodes belonging to the cryptocurrency mining stage can not be found. So the *graph alignment score* computed using the Eq. (2) in Poirot [31] becomes smaller (The *influence scores* of the missing paths are all equal to 0. And the denominator of Eq. (2) in Poirot [31], $|F(G_q)|$, which is the number of *flows* in the query graph, remains unchanged). As a result, this type of inconsistency in Q5+ETW leads to the invalidation of Poirot. On the contrary, DeepHunter does not rely on complete connectivity remained in the provenance graph. As long as most node attribute information and the main graph structure information between the query graph and the traceability graph are matched, DeepHunter can recognize that the two graphs represent the same attack behavior. Therefore, DeepHunter has a more robust cyber threat hunting ability.

7.3 Comparison with Other Graph Matching Models

We compare DeepHunter with a non-learning graph matching approach and other GNN-based graph matching models. Note that these GNN-based models are not specifically designed for threat hunting. We evaluate all the graph matching approaches using the AUC value, since it is a strict metric. If a small mistake is made, the error would be obvious.

DeepHunter vs. Non-learning Approach. We compare DeepHunter with the Weisfeiler Lehman (WL) kernel, a non-learning method for calculating the graph similarity. We set the number of iteration of the WL kernel from 1 to 10 and put the best results in Table 6. The result of the WL kernel is not desirable because it is designed for graph isomorphism testing. In contrast, the graph matching in a threat hunting task is more similar to determining whether a query graph can be regarded as an abstraction of the provenance graph.

DeepHunter vs. GNN-based Graph Matching Models. We compare DeepHunter's graph matching model with other GNN-based graph matching networks: MatchGNet [50], SimGNN [4] and GMN [26]. MatchGNet proposed a Hierarchical Attentional Graph Neural Encoder (HAGNE) which could embed the provenance graph. Given the graph-level embeddings, MatchGNet employs the Siamese network to learn the similarity metric. We believe that the Siamese network is not enough to learn the complex relationship between the two graphs.

Hence, we substitute the Siamese network of MatchGNet with the NTN layer. We call the modified model MatchGNet-NTN. By comparing DeepHunter and MatchGNet-NTN, we can evaluate the effectiveness of our graph embedding networks. As can be seen in Table 6, the performance of DeepHunter outperforms MatchGNet and MatchGNet-NTN.

We also evaluate the effectiveness of the attribute embedding network. Instead of the attribute embedding network, we directly use the one-hot encoding of attributes as the node's input feature. We call this model DeepHunter-wo-AEN. Table 6 shows that DeepHunter-wo-AEN is inferior to DeepHunter, which demonstrates the attribute embedding network is necessary for our graph matching task.

At last, we evaluate the other two graph matching networks: SimGNN and GMN. Like DeepHunter, SimGNN also leverages GNN to represent input graphs and then utilizes NTN to learn the similarity between two graph-level embeddings. But the graph neural networks in SimGNN are not specifically designed for representing the provenance graphs. Besides, SimGNN believes that if there is a difference in the size of the two input graphs, then the two graphs are not similar. GMN takes into account the node correlation across graphs to model the relation. The results of SimGNN and GMN are shown in Table 6. We can see that the performance of DeepHunter is superior to both SimGNN and GMN.

Table 6. AUC values of graph matching models on three datasets.

Dataset	CADETS	TRACE	ETW
DeepHunter	1	**0.951**	**0.916**
MatchGNet [50]	1	0.880	0.805
MatchGNet-NTN	1	0.932	0.844
MatchGNet-wo-AEN	1	0.891	0.820
SimGNN [4]	1	0.906	0.805
GMN [25]	1	0.846	0.830
WL kernel	1	0.492	0.301

8 Conclusions

We propose DeepHunter, a GNN-based graph pattern matching approach for cyber threat hunting. More importantly, DeepHunter is robust against the inconsistency between real attack behaviors recorded by provenance data and known attack behaviors to some extent. Our extensive evaluations show that DeepHunter can tolerate various scenarios with different inconsistency scores, including disconnected attack provenance graphs. In our synthetic APT attack scenario, DeepHunter is superior to the state-of-the-art APT threat hunting approach Poirot. Our research showcased a successful application of the graph neural network on the threat hunting task.

Acknowledgment. This work is supported by the Strategic Priority Research Program of Chinese Academy of Sciences, Grant No. XDC02040200.

References

1. Event tracing. https://docs.microsoft.com/en-us/windows/win32/etw/event-tracing-portal
2. Causal, adaptive, distributed, and efficient tracing system (cadets) (2018). https://www.cl.cam.ac.uk/research/security/cadets/. Accessed 21 Sept 2020
3. Trace: Preventing advanced persistent threat cyberattacks (2018). https://archive.sri.com/work/projects/trace-preventing-advanced-persisten-threat-cyberattacks. Accessed 21 Sept 2020
4. Bai, Y., Ding, H., Bian, S., Chen, T., Sun, Y., Wang, W.: SimGNN: a neural network approach to fast graph similarity computation. In: Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining, pp. 384–392 (2019)
5. Bai, Y., Ding, H., Gu, K., Sun, Y., Wang, W.: Learning-based efficient graph similarity computation via multi-scale convolutional set matching. In: AAAI, pp. 3219–3226 (2020)
6. Bates, A., Tian, D.J., Butler, K.R., Moyer, T.: Trustworthy whole-system provenance for the linux kernel. In: 24th {USENIX} Security Symposium ({USENIX} Security 15), pp. 319–334 (2015)
7. Bromley, J., Guyon, I., LeCun, Y., Säckinger, E., Shah, R.: Signature verification using a "siamese" time delay neural network. In: Advances in Neural Information Processing Systems, pp. 737–744 (1994)
8. FireEye (2018). https://openioc.org. openIOC
9. Fyrbiak, M., Wallat, S., Reinhard, S., Bissantz, N., Paar, C.: Graph similarity and its applications to hardware security. IEEE Trans. Comput. **69**(4), 505–519 (2019)
10. Gehani, A., Tariq, D.: SPADE: support for provenance auditing in distributed environments. In: Narasimhan, P., Triantafillou, P. (eds.) Middleware 2012. LNCS, vol. 7662, pp. 101–120. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-35170-9_6
11. Gibson, T., Schuchardt, K., Stephan, E.G.: Application of named graphs towards custom provenance views. In: Workshop on the Theory and Practice of Provenance (2009)
12. Graeber, M.: Abusing Windows Management Instrumentation (WMI) to Build a Persistent, Asyncronous, and Fileless Backdoor. Black Hat, Las Vegas (2015)
13. Hassan, W.U., Bates, A., Marino, D.: Tactical provenance analysis for endpoint detection and response systems. In: Proceedings of the IEEE Symposium on Security and Privacy (2020)
14. Hassan, W.U., et al.: NODOZE: combatting threat alert fatigue with automated provenance triage. In: NDSS (2019)
15. Hassan, W.U., Noureddine, M.A., Datta, P., Bates, A.: OmegaLog: high-fidelity attack investigation via transparent multi-layer log analysis. In: Proceedings NDSS (2020)
16. Homayoun, S., Dehghantanha, A., Ahmadzadeh, M., Hashemi, S., Khayami, R.: Know abnormal, find evil: frequent pattern mining for ransomware threat hunting and intelligence. IEEE Trans. Emerg. Top. Comput. **8**, 341–351 (2017)

17. Hossain, M.N., et al.: {SLEUTH}: real-time attack scenario reconstruction from {COTS} audit data. In: 26th {USENIX} Security Symposium ({USENIX} Security 17), pp. 487–504 (2017)
18. Hossain, M.N., Sheikhi, S., Sekar, R.: Combating dependence explosion in forensic analysis using alternative tag propagation semantics. In: 2020 IEEE Symposium on Security and Privacy (SP). IEEE (2020)
19. Husari, G., Al-Shaer, E., Ahmed, M., Chu, B., Niu, X.: TTPDrill: automatic and accurate extraction of threat actions from unstructured text of CTI sources. In: Proceedings of the 33rd Annual Computer Security Applications Conference, pp. 103–115 (2017)
20. Kaspar, R.: https://github.com/dzambon/graph-matching-toolkit (2018). mig-logcleaner-resurrected
21. Khan, A., Wu, Y., Aggarwal, C.C., Yan, X.: NeMa: fast graph search with label similarity. Proc. VLDB Endowment **6**(3), 181–192 (2013)
22. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907 (2016)
23. Le, Q., Mikolov, T.: Distributed representations of sentences and documents. In: International Conference on Machine Learning, pp. 1188–1196 (2014)
24. Lee, K.H., Zhang, X., Xu, D.: High accuracy attack provenance via binary-based execution partition. In: NDSS (2013)
25. Li, Y., Gu, C., Dullien, T., Vinyals, O., Kohli, P.: Graph matching networks for learning the similarity of graph structured objects. In: Chaudhuri, K., Salakhutdinov, R. (eds.) Proceedings of the 36th International Conference on Machine Learning. Proceedings of Machine Learning Research, Long Beach, California, USA,09–15 Jun 2019, vol. 97, pp. 3835–3845. PMLR (2019). http://proceedings.mlr.press/v97/li19d.html
26. Li, Y., Gu, C., Dullien, T., Vinyals, O., Kohli, P.: Graph matching networks for learning the similarity of graph structured objects. arXiv preprint arXiv:1904.12787 (2019)
27. Liao, X., Yuan, K., Wang, X., Li, Z., Xing, L., Beyah, R.: Acing the IOC game: toward automatic discovery and analysis of open-source cyber threat intelligence. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 755–766 (2016)
28. Ma, S., Zhai, J., Wang, F., Lee, K.H., Zhang, X., Xu, D.: {MPI}: Multiple perspective attack investigation with semantic aware execution partitioning. In: 26th {USENIX} Security Symposium ({USENIX} Security 17), pp. 1111–1128 (2017)
29. Manzoor, E., Milajerdi, S.M., Akoglu, L.: Fast memory-efficient anomaly detection in streaming heterogeneous graphs. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1035–1044 (2016)
30. Micro, T.: cryptocurrency Miner Uses WMI and EternalBlue To Spread Filelessly (2017). https://blog.trendmicro.com/trendlabs-security-intelligence/cryptocurrency-miner-uses-wmi-eternalblue-spread-filelessly/. Accessed 4 May 2020
31. Milajerdi, S.M., Eshete, B., Gjomemo, R., Venkatakrishnan, V.: POIROT: aligning attack behavior with kernel audit records for cyber threat hunting. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, pp. 1795–1812 (2019)

32. Milajerdi, S.M., Gjomemo, R., Eshete, B., Sekar, R., Venkatakrishnan, V.: HOLMES: real-time apt detection through correlation of suspicious information flows. In: 2019 IEEE Symposium on Security and Privacy (SP), pp. 1137–1152. IEEE (2019)
33. MISP: Open Source Threat Intelligence Platform & Open Standards For Threat Information Sharing (2019). https://www.misp-project.org/
34. Mitre: Structured Threat Information eXpression (STIX) (2018). https://stixproject.github.io
35. Oprea, A., Li, Z., Yen, T.F., Chin, S.H., Alrwais, S.: Detection of early-stage enterprise infection by mining large-scale log data. In: 2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, pp. 45–56. IEEE (2015)
36. Pasquier, T., et al.: Practical whole-system provenance capture. In: Proceedings of the 2017 Symposium on Cloud Computing, pp. 405–418 (2017)
37. Paszke, A., et al.: Pytorch: an imperative style, high-performance deep learning library. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems, vol. 32, pp. 8024–8035. Curran Associates, Inc. (2019). http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf
38. Pei, K., et al.: HERCULE: attack story reconstruction via community discovery on correlated log graph. In: Proceedings of the 32nd Annual Conference on Computer Security Applications, pp. 583–595 (2016)
39. Pienta, R., Tamersoy, A., Tong, H., Chau, D.H.: MAGE: matching approximate patterns in richly-attributed graphs. In: 2014 IEEE International Conference on Big Data (Big Data), pp. 585–590. IEEE (2014)
40. Pohly, D.J., McLaughlin, S., McDaniel, P., Butler, K.: Hi-fi: collecting high-fidelity whole-system provenance. In: Proceedings of the 28th Annual Computer Security Applications Conference, pp. 259–268 (2012)
41. Qureshi, R.J., Ramel, J.-Y., Cardot, H.: Graph based shapes representation and recognition. In: Escolano, F., Vento, M. (eds.) GbRPR 2007. LNCS, vol. 4538, pp. 49–60. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-72903-7_5
42. Řehůřek, R., Sojka, P.: Software framework for topic modelling with large corpora. In: Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks, Valletta, Malta, pp. 45–50. ELRA, May 2010. http://is.muni.cz/publication/884893/en
43. Riesen, K., Emmenegger, S., Bunke, H.: A novel software toolkit for graph edit distance computation. In: Kropatsch, W.G., Artner, N.M., Haxhimusa, Y., Jiang, X. (eds.) GbRPR 2013. LNCS, vol. 7877, pp. 142–151. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38221-5_15
44. Smith, J. (2021). https://libraetd.lib.virginia.edu/public_view/5138jf509. Accessed 4 Mar 2021
45. Socher, R., Chen, D., Manning, C.D., Ng, A.: Reasoning with neural tensor networks for knowledge base completion. In: Advances in Neural Information Processing Systems, pp. 926–934 (2013)
46. Song, W., Yin, H., Liu, C., Song, D.: DeepMem: learning graph neural network models for fast and robust memory forensic analysis. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pp. 606–618 (2018)
47. Sun, X., Dai, J., Liu, P., Singhal, A., Yen, J.: Using bayesian networks for probabilistic identification of zero-day attack paths. IEEE Trans. Inf. Forensics Secur. **13**(10), 2506–2521 (2018)

48. Tong, H., Faloutsos, C., Gallagher, B., Eliassi-Rad, T.: Fast best-effort pattern matching in large attributed graphs. In: Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 737–746 (2007)
49. Wang, Q., et al.: You are what you do: Hunting stealthy malware via data provenance analysis. In: Proceedings of the Symposium on Network and Distributed System Security (NDSS) (2020)
50. Wang, S., et al.: Heterogeneous graph matching networks for unknown malware detection. In: Proceedings of the 28th International Joint Conference on Artificial Intelligence, pp. 3762–3770. AAAI Press (2019)
51. Xiong, C., et al.: CONAN: a practical real-time APT detection system with high accuracy and efficiency. IEEE Trans. Depend. Secur. Comput. (2020)
52. Xu, X., Liu, C., Feng, Q., Yin, H., Song, L., Song, D.: Neural network-based graph embedding for cross-platform binary code similarity detection. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pp. 363–376 (2017)
53. Zhu, Z., Dumitras, T.: ChainSmith: automatically learning the semantics of malicious campaigns by mining threat intelligence reports. In: 2018 IEEE European Symposium on Security and Privacy (EuroS&P), pp. 458–472. IEEE (2018)