Log2vec: A Heterogeneous Graph Embedding Based Approach for Detecting Cyber Threats within Enterprise

Fucheng Liu
Institute of Information Engineering, Chinese Academy of Sciences
Beijing, China
School of Cyber Security, University of Chinese Academy of Sciences
Beijing, China
liufucheng@iie.ac.cn

Yu Wen*
Institute of Information Engineering, Chinese Academy of Sciences
Beijing, China
wenyu@iie.ac.cn

Dongxue Zhang
Institute of Information Engineering, Chinese Academy of Sciences
Beijing, China
zhangdongxue@iie.ac.cn

Xihe Jiang
Institute of Information Engineering, Chinese Academy of Sciences
Beijing, China
School of Cyber Security, University of Chinese Academy of Sciences
Beijing, China
jiangxihe@iie.ac.cn

Xinyu Xing
The Pennsylvania State University
Pennsylvania, USA
JD Security Research Center
California, USA
xxing@ist.psu.edu

Dan Meng
Institute of Information Engineering, Chinese Academy of Sciences
Beijing, China
mengdan@iie.ac.cn

ABSTRACT

Conventional attacks of insider employees and emerging *APT* are both major threats for the organizational information system. Existing detections mainly concentrate on users' behavior and usually analyze logs recording their operations in an information system. In general, most of these methods consider *sequential relationship among log entries* and model users' sequential behavior. However, they ignore other relationships, inevitably leading to an unsatisfactory performance on various attack scenarios.

We propose log2vec, a heterogeneous graph embedding based modularized method. First, it involves a heuristic approach that converts log entries into a heterogeneous graph in the light of diverse relationships among them. Next, it utilizes an improved graph embedding appropriate to the above heterogeneous graph, which can automatically represent each log entry into a low-dimension vector. The third component of log2vec is a practical detection algorithm capable of separating malicious and benign log entries into different clusters and identifying malicious ones. We implement a prototype of log2vec. Our evaluation demonstrates that log2vec remarkably outperforms state-of-the-art approaches, such as deep learning and hidden markov model (HMM). Besides, log2vec shows its capability to detect malicious events in various attack scenarios.

*Corresponding Author.

CCS '19, November 11–15, 2019, London, United Kingdom

ACM ISBN 978-1-4503-6747-9/19/11…$15.00
https://doi.org/10.1145/3319535.3363224

CCS CONCEPTS

• **Computing methodologies** → **Anomaly detection**.

KEYWORDS

Insider Threat Detection, Advanced Persistent Threats, Graph Construction, Heterogeneous Graph Embedding

ACM Reference Format:
Fucheng Liu, Yu Wen, Dongxue Zhang, Xihe Jiang, Xinyu Xing, and Dan Meng. 2019. Log2vec: A Heterogeneous Graph Embedding Based Approach for Detecting Cyber Threats within Enterprise. In *2019 ACM SIGSAC Conference on Computer and Communications Security (CCS '19), November 11–15, 2019, London, United Kingdom.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3319535.3363224

1 INTRODUCTION

Modern information systems have been important and irreplaceable components for today's enterprises and organizations. However, these systems are constantly under risk of attacks from insider employees, who have authorized access to them and intentionally use this access to influence their confidentiality, integrity or availability [7]. Meanwhile, another emerging attack, *advanced persistent threat (APT)* also threatens these systems. Specifically, *APT* actors initially compromise accounts and hosts in a target system and from these hosts, they would stealthily and persistently compromise multiple hosts through intranet and steal confidential information [4]. These two types of attacks have been deemed primary and costly threats for modern enterprises [29, 51, 56, 58].

There exist differences in detecting these two kinds of attacks. A common scenario of insider threats is that a malicious employee obtains credentials of another legitimate user through key logger, and utilizes this new identity to steal confidential information (that is, masquerade attack). The existing approaches in general convert user's various operations (also log entries) into sequences, which

can hold information, e.g. ***sequential relationship among log entries***, and then use sequence processing technique, e.g. deep learning, to learn from past events and predict the next event [12, 47]. Essentially, these log-entry-level approaches model user's normal behavior and flag deviations from it as anomalies.

Unfortunately, this kind of methods ignores other relationships. For instance, comparison of user's daily behavior is a common means for conventional insider threat detection [41, 57]. This detection is based on the premise that a user's daily behavior is relatively regular during a period (***logical relationships among days***). The aforementioned prediction approaches overlook this relationship and would drop in their performance. Besides, they require normal log entries or even a large number of labeled data for model training. However, in a real-world scenario, there exist rare attack actions, limiting their capability of correct prediction.

In *APT*, the perpetrator usually exploits the intranet (compromises multiple hosts) to escalate his privilege and steal confidential intelligence. Many approaches focus on analyzing user's logon operations to detect anomalous ones [4, 50]. Nevertheless, this type of methods is generally capable of analyzing the particular relationship, ***interactive relationship among hosts***, and cannot detect the previously mentioned insider threats, involving numerous other operations, such as file operations and website browses. Moreover, the suspicious hosts produced by these methods, inevitably involve numerous benign operations, thereby leading to many efforts for subsequent artificial error correction. Besides, certain deep learning methods may detect malicious logons by analyzing users' logon sequences in the log-entry-level granularity [12, 47]. However, they also demand a large amount of labeled data for training.

In summary, we face three problems: 1) how to simultaneously detect the two attack scenarios aforementioned, specifically considering all the three mentioned relationships (in bold and italic) for a detection system; 2) how to conduct a fine-grained detection in the *APT* scenario, specifically deeply mining and analyzing relationships among log entries within a host; 3) how to perform the detection without attack samples for training models.

Our Designs. We introduce log2vec, a novel approach to detect the two typical attacks. Log2vec comprises three components as shown in Figure 1: (1) graph construction. Log2vec constructs a heterogeneous graph to integrate multiple relationships among log entries; (2) graph embedding (also graph representation learning). It's a powerful graph processing approach to learn each operation's representation (vector) base on their relationships in such a graph. Vectorizing user's operations enables a direct comparison of their similarities to find out anomalies; (3) detection algorithm, to effectively group malicious operations into individual clusters and figure them out.

Log2vec introduces specialized designs to tackle the aforementioned problems. First, log2vec's first component constructs a heterogeneous graph. This data structure is constructed based on the three previous relationships, which are the main ones that the existing methods have used in solving the two attack scenarios [4, 12, 38, 41, 47, 50, 57] (for problem-1). Second, we divide a log entry into five attributes. According to these attributes, we deeply consider relationships among logs within a host and devise fine rules to correlate them. This design enables normal and anomalous log entries to own different topologies in such a graph,

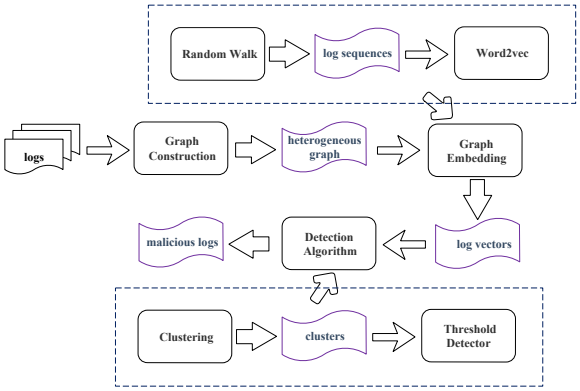


Figure 1: Log2vec schematic overview. Log2vec includes graph construction, graph embedding and detection algorithm (rounded rectangle). Graph embedding is composed of random walk and word2vec (above-dotted rectangle). Detection algorithm consists of clustering and threshold detector (below-dotted rectangle).

which can be captured and detected by log2vec's latter components (for problem-2). Third, log2vec's graph embedding and detection algorithm represent and group log entries into different clusters without attack samples, applicable to the data imbalanced scenario (for problem-3). Additionally, graph embedding itself can automatically learn representation (vector) for each operation instead of manually extracting domain-specific features, thereby independent of expert's knowledge. Our improved version can further differentially extract and represent multiple relationships among operations from the above heterogeneous graph.

In summary, we make the following contributions:

- We propose log2vec, a novel approach for cyber threat detection, capable of effectively detecting user's malicious behavior through capturing and representing rich relationships among user's operations. Log2vec performs a log-entry-level detection without attack samples for training, under various attack scenarios, such as the aforementioned typical attacks of insider employees and *APT*.
- We propose a novel method that can translate a user's log entries into a heterogeneous graph, holding information that reflects the user's typical behavior and exposes malicious events. It is a generic rule-based approach for graph construction. (Section 3)
- We propose an improved graph embedding, which can differentially extract and represent multiple relationships among operations from the aforementioned heterogeneous graph. (Section 4)
- Our evaluation illustrates that log2vec significantly outperforms state-of-the-art techniques, such as TIRESIAS, DeepLog and HMM, in multiple attack scenarios. Meanwhile, log2vec shows that its clustering algorithm is experimentally superior to traditional clustering one, *k*-means in insider threat detection. (Section 6)

- We further explore the effect of log2vec's parameters on detection performance. When we employ log2vec, a method of representation learning, to analyze user's behavior for cyber threat detection, we find that this type of detection requires different combinations of parameters for each user to achieve the best performance, because multiple users and attack scenarios produce various user's behavior and attack patterns. Following our experiment, we classify log2vec's parameters into two categories. One is sensitive to user's behavior and attack scenarios and another is not. Furthermore, we provide our suggestions on how to set these parameters. (Section 6.2 and Section 6.4)

2 OVERVIEW

We first present an example to illustrate our problem. Then, we detail log2vec and introduce the adversarial model.

2.1 Motivating Example

A log file that we aim to detect is depicted in Figure 2a. It records a user's operations, such as login operations, removable device usage and network operations [46], in an enterprise. Figure 2b denotes attributes of such a log entry, such as subject (e.g. user id), operation type (e.g. visit or send), object (e.g. website or email), time and host (e.g. server id). In fact, attributes and relationships among log entries reveal a user's behavior. For instance, the first logon time and last logoff time usually indicate the user's working hours. A system administrator may frequently log in to servers and subsequently perform operations regarding system maintenance (e.g. open and write a configuration file shown in day1).

Figure 2c displays a sequence approach [12, 47], coding each log entry in Figure 2a and concatenates them ordered by time into sequences. This type of methods utilizes deep learning, e.g. Long Short-Term Memory (LSTM), to learn from past events and predict the next events [12, 47]. Apparently, they mainly capture ***causal and sequential relationships among log entries***.

Unfortunately, they ignore other relationships. For instance, there are a large number of device connect and file copy operations in day3, far more than before (implying a data breach). This discrepancy can be detected by directly comparing user's daily behavior [41, 57]. The comparison is based on the premise that a user's daily behavior is relatively regular and similar during a period (***logical relationships among days***). Although deep learning, e.g. LSTM, can remember long-term dependencies (multiple days) over sequences [12], it does not explicitly compare user's daily behavior and cannot achieve a satisfactory performance (see Section 6.2). Similarly, they cannot hold another relationship, ***interactive relationship among hosts*** in Figure 2d, and cannot work well on the *APT* detection. Additionally, some of them require a large number of labeled data for training. However, in our detection scenario, there exist rare attack actions (see Section 6.1, **Dataset**).

Figure 2d demonstrates a graph indicating a user's behavior across hosts, from logons in Figure 2a (red font). We can analyze these ***interactive relationships among hosts*** to find out anomalous logons. For instance, an administrator may regularly log in to a group of hosts for system maintenance while an *APT* perpetrator

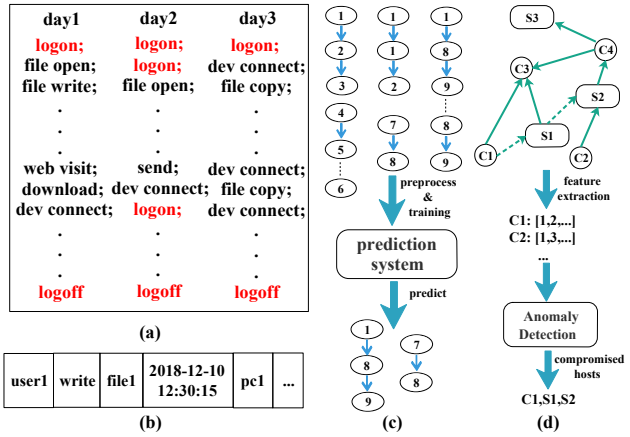


Figure 2: Sequence and graph approaches for cyber threat detection. (a) a log file recording user's operations (b) attributes of a log entry (c) sequence approach: learn from past events and predict the next ones; (d) graph approach: extract Computer's/ Server's features and detect suspicious ones

can only visit hosts that he can. This difference can be caught by features of logon traces [4]. For instance, the number of hosts involved in a benign trace (1 or 3, solid lines) is generally different from that of *APT* (2, dashed lines). After analyzing these features, compromised hosts can be identified. However, these hosts comprise numerous benign operations and manually extracted domain-specific features apparently cannot be applied to the attack in Figure 2c.

In summary, the existing methods have the following deficiencies: fail to 1) cover various cyber threats, such as the aforementioned insider threats and *APT*; 2) exactly detect malicious operations (log-entry level), specifically deeply mining and analyzing relationships among log entries within a day/host; 3) perform the detection without attack samples for training.

2.2 Architecture

Log2vec is composed of three key components: graph construction, graph embedding and detection algorithm as shown in Figure 1.

Graph construction. Log2vec's first component is a rule-based heuristic approach to map relationships among log entries that reflect users' typical behavior and expose malicious operations, into a graph. According to the existing methods [4, 38, 41, 50, 57], log2vec mainly takes three relationships into account: (1) causal and sequential relationships within a day; (2) logical relationships among days (3) logical relationships among objects. (Section 3.2)

To deeply mining and analyzing relationships among log entries within a day/host, we divide a log entry into five primary attributes (subject, object, operation type, time and host), named as meta-attributes (see Section 3.1). When designing rules regarding the three relationships, we consider different combinations of these meta-attributes to correlate fewer log entries and map finer logs' relationships into the graph. For instance, an administrator logs in to his own computer, then remotely logs in to a server and opens a file to view the state of the system in day2 (Figure 2a). This sequence reflects a sequential relationship among log entries. We use a rule,

log entries of ***the same user*** *are connected* ***in chronological order*** (rule A), to map this relationship into a graph. We consider two meta-attributes, subject and time.

In another example, we consider another meta-attribute, operation type, and use a rule, *log entries of* ***the same user*** *and* ***the same operation type*** *are connected* ***in chronological order*** (rule B), to concatenate device connection operations within a day. The number of log entries involved in daily sequences only regarding device connections is obviously smaller than that of ones comprising all operations (Figure 2a). After generating three sequences of device connections corresponding to the 3 days, we employ other rules to correlate them according to their similarities. As the sequence in day3 involves numerous connection operations, far more than others, it has lower link weights with others. This difference can be captured by graph embedding. Likewise, log2vec converts the relationship in Figure 2d into the graph (see Section 3.2.3).

Through different combinations of log's attributes, we devise various behavioral sequences involving fewer log entries and map multiple finer relationships among log entries within a day and a host, into the graph. After graph embedding and detection algorithm, log2vec produces small clusters to reveal anomalous operations. In practice, the numbers of log entries involved in suspicious clusters are very small and even equal to 1. Therefore, log2vec more finely mines user's behavior in the above two attack scenarios.

According to each rule in log2vec, we translate log entries into sequences or subgraphs, all of which constitute a heterogeneous graph. Each rule, corresponding to an edge type, is derived from a specific relationship, as above examples. As different relationships play different roles in various scenarios, we use multiple edge types, instead of weights, to discriminate them.

Graph embedding. Graph embedding, also graph representation learning, is a machine learning approach, capable of converting nodes (log entries) in the heterogeneous graph into low-dimension vectors [11, 14, 39, 54]. Specifically, graph embedding involves random walk and word2vec in Figure 1. The former extracts context of each node and feeds it into the latter that calculates its vector.

Random walk is a popular graph traverse algorithm. Assuming a walker resides at a node in a graph, he decides the next node to visit according to weight and type of each edge. The path, a sequence of nodes, generated by him is regarded as the context of these nodes (see Section 4.1). For instance, when a walker resides at a node belonging to the sequence of device connections in day1 or day2 (Figure 2a), generated by graph construction, he would seldom choose the node (device connection) in the sequence of day3 because of low link weight. Likewise, when residing at a node in sequence of day3, he would rarely reach the sequence of day1 or day2. Therefore, log2vec extracts paths involving nodes of day1 and day2 or individually day3.

The word2vec model is used to calculate vector of each node with its paths (context). To be specific, these paths are taken as sentences in natural language and processed by this model to learn vector of each word (node) (see Section 4.2). This method preserves the proximity between a node and its context [11, 14, 39, 54], meaning that a node (log entry) and its neighbors (log entries having close relationships with it) share similar embeddings (vectors). In the above example, log entries (device connections) in day1 and day2 have been in the same path and thereby share similar vectors while those in day3 possess different ones.

Further, log2vec improves the existing graph embedding [11, 14, 39, 54] (see Section 4.1.2, Section 4.1.3 and comparison with baselines [11, 14] in Section 6.2). The novel version is capable of determining importance of each relationship among log entries based on attack scenarios and processing them differentially. In the above example, log2vec determines that rule B is more important than rule A and extracts more paths corresponding to the edge type defined by rule B in random walk. After word2vec, vectors of device connections in day3 tend to be similar due to many extracted paths regarding rule B while they are all dissimilar to that of the logon operation in day3 due to few ones regarding rule A. Therefore, log2vec is biased to extract and represent logs' relationships. This example also shows how log2vec groups logons (benign) and device connections (malicious) within a day into different clusters, instead of viewing them as a daily sequence.

Detection Algorithm. Log2vec adopts a clustering algorithm to analyze the above vectors and group benign operations (log entries) and malicious ones into different clusters (Section 5.1). In the above example, the clustering algorithm groups device operations in the sequences of day1 and day2 into one cluster and those in day3's sequence into another one. After the clustering, we set a threshold to identify malicious clusters. That is, clusters, whose sizes are smaller than the threshold, are viewed as malicious (Section 5.2).

In summary, there are close relationships among benign operations (user's typical behavior), the same as malicious ones. However, there are fewer or even no correlations between them. Following this observation, log2vec differentially extracts and represents these relationships, to separate them into different clusters. Furthermore, the number of malicious operations is small [2, 4] and thereby smaller clusters tend to be malicious.

2.3 Adversarial Model

Adversarial model includes the following three attack scenarios commonly in enterprises and governments.

The first scenario is that an insider employee misuses his authority to perform malicious operations, such as accessing databases or application servers, and then sabotaging system or stealing intellectual property for personal benefits.

Second, a malicious insider obtains credentials of other legitimate user through peeping or key logger, and utilizes this new identity to seek confidential information or create chaos in the company. These two scenarios belong to typical attacks of insider employees.

The third attack is that an *APT* actor compromises a host in the system and from this host, he persistently compromises multiple hosts for escalating his privilege and stealing confidential files.

3 GRAPH CONSTRUCTION

In Section 3.1, we formally define a log entry. Section 3.2 details rules to construct heterogeneous graphs.

3.1 Definition

A log entry in Figure 2b is defined as a tuple involving five meta-attributes $< sub, obj, A, T, H >$, where *sub* is a collection of users;

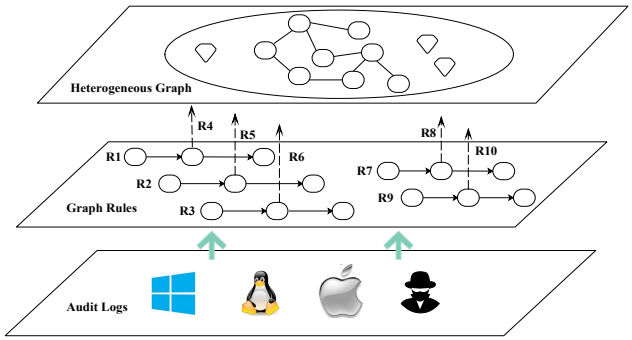


Figure 3: Graph construction. Log2vec utilizes ten rules to construct a heterogeneous graph. R1 indicates rule1. R1~R3 correspond to causal and sequential relationships within a day. They construct user's daily behavioral sequences. R4~R6 respectively bridge these daily sequences based on logical relationships among days. R7 and R9 respectively construct user's logon and web browsing behavioral sequences. R8 and R10 bridge them following logical relationships among objects. Diamonds in the heterogeneous graph are suspicious log entries.

obj is a collection of objects, such as file, removable storage device and website; A is a collection of operation types, such as file operation and browser usage; T is a collection of time and H is a collection of hosts, such as computer or server.

In addition, *sub*, *obj*, A and H have their own sets of attributes. For instance, a user writes a file in a server. The attributes of *sub* involve role (e.g. system administrator) and organizational unit that he is affiliated to (e.g. department of field service). The attributes of *obj* may include file type and size. The attribute of H is function of server (e.g. file server). For a logon operation, the attributes of A include authentication type, such as Kerberos or NT LAN Manager (NTLM), and logon type (e.g. interactive or network). Note that a logon operation generally comprises a source host and a destination one. In particular, log2vec regards the latter one as *obj* and the former one as H. In other words, it treats a logon operation in the following way, a user (*sub*) logs in to (A) a destination host (*obj*) in a source one (H), just as a user writes a file in a server.

3.2 Rules of Graph Construction

Relationships involve three categories: (1) causal and sequential relationships within a day; (2) logical relationships among days; (3) logical relationships among objects. For each category, we propose a few rules based on different combinations of meta-attributes.

As shown in Figure 3, we first present rule1~rule3 to concatenate log entries within a day into sequences, corresponding to relationship (1). These three rules model user's behavior from different aspects, e.g. day, host and operation type. Through this design, rare operations, which the user acts in an unfamiliar host, or belong to the operation type seldom acted, are isolated in the graph. We then propose rule4~rule6 to separately bridge these daily sequences based on relationship (2). The anomalous behavioral sequences would be separated from benign ones. These six rules map the relationships in Figure 2c, into a graph. They mainly correlate user's operations of various types on multiple hosts across days.

Last, four rules (rule7~rule10), corresponding to relationship (3), are presented to consider how a user logs into/compromises a host and sends confidential data to the outside. Specifically, they construct user's patterns of logon and web browsing operation. Rules regarding logon operation, rule7 and rule8, consider how these hosts are interactively accessed within intranet, such as an instance in Figure 2d. Rules regarding web operation, rule9 and rule10, focus on user's browser usage via the Internet. The intranet and Internet are the main intrusion sources, e.g. logging into a host or driven-by download. Besides, Internet is also the main way whereby confidential files are leaked. Note that another channel of leaking data, device connection and file copy, has been considered by the first six rules as the example in Section 2.2.

Log2vec aims at analyzing each user's typical behavior and thereby it constructs a heterogeneous graph for each user (*sub*). The following rules are proposed as edge types in such a graph.

3.2.1 Constructing graph within a day. In enterprise, users own various roles and conduct diverse operations. Therefore, multiple rules are designed towards detecting their behavior. First, log2vec considers meta-attributes, *sub* and T, and constructs a user's daily behavioral sequence, a common way to detect user's daily behavior [38, 41]. Meanwhile, this design is also used to reduce false positives. For instance, administrators usually log into multiple hosts per day for crash recovery. These operations should be correlated into one sequence. Specifically, an administrator often logs into different hosts and acts only a few operations on each one per day. Due to irregular system problems, hosts that he logs into for recovery are varied. If we divide his operations according to hosts, a large number of benign operations from multiple hosts would respectively locate in different parts of the graph and be mistakenly identified as anomalies. Accordingly, we propose the rule:

Rule1 (edge1): *log entries of the same day are connected in chronological order with the highest weight (value 1).*

Second, log2vec constructs a user's daily behavior on each host. This design is based on the observation that other users, different from administrators, generally conduct actions on specified several hosts or only his own. Thereby, the unfamiliar hosts that the user seldom or never logs into before are indicative of anomalous logons [27, 50]. It deserves serious consideration on how to distinguish hosts (H) in the graph. We present the following rule:

Rule2 (edge2): *log entries of the same host and the same day are connected in chronological order with the highest weight (value 1).*

Some malicious log entries are related to multiple operation types, A. For instance, a user searches for confidential files (file operation) on others' hosts and sends them to his home email (email operation). These consecutive operations can be correlated by Rule2. However, there exist attack actions regarding only one type. For instance, a user visits a website and uploads confidential files. It is appropriate to map them into the graph without other operation types as noise, beneficial to the following precise detection. Meanwhile, they occur on a host. We thereby construct a user's daily behavior regarding each operation type on each host:

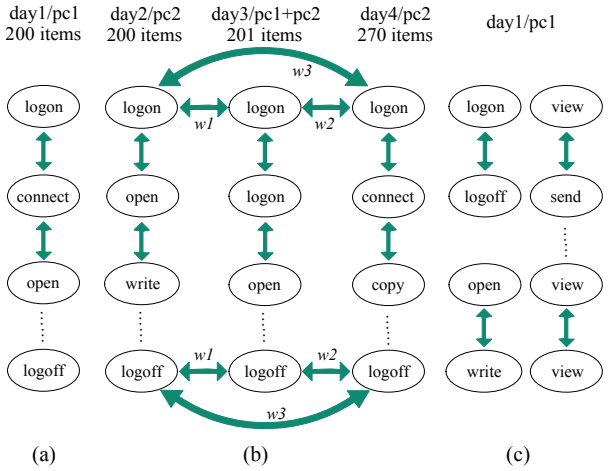


Figure 4: Examples of constructing graph within a day and among days. (a) user's daily log sequence involving log entries of the same host (pc1) (b) user's daily log sequences are connected. Log sequences of day2 and day3 have similar numbers of log entries (200, 201), thus their weight $w1$ higher than $w2$ and $w3$. The sequence in (a) can be respectively connected to ones in (b) according to rule4. However, when considering rule5, the sequence of day3 (b) should be divided into two subsequences, day3/pc1 and day3/pc2, and only the former can be connected to the one in (a) due to the same host, pc1 (rule5). (c) user's daily log sequence involving log entries of the same operation type and host.

Rule3 (edge3): *log entries of the same operation type, the same host and the same day are connected in chronological order with the highest weight (value 1).*

Through these rules, log2vec constructs various daily behavioral sequences. They are finer (some involving only a few log entries) and comprise more stronger logical relationships than the conventional ones [38, 41].

3.2.2 Constructing graph among days. An employee's daily behavior is in general regular during a period. The existing methods mainly focus on numbers of daily operations [13, 27, 41, 46, 57]. Specifically, an employee usually has an average workload every day. An exception far exceeding this average is likely to involve malicious operations, as the example depicted in Section 2.2.

A logical relationship among daily log sequences reflects this regularity of user's behavior. Specifically, the numbers of log entries involved in benign daily sequences are close to the average and thus close to each other, while those of malicious ones are far different from them. Put another way, these benign daily sequences share similar behavioral patterns, different from the malicious ones.

To compare user's daily behavioral sequences derived from rule1~rule3, we present rule4~rule6, comprising meta-attributes separately corresponding to those of rule1~rule3. Through these rules, log2vec respectively isolates anomalous behavioral sequences, from various scenarios mentioned in Section 3.2.1, in the graph.

Rule4 (edge4): *daily log sequences are connected and their weights are positively related to similarity of the numbers of log entries that they contain.*

This rule involves *sub* and T. Figure 4b shows an instance. The connection mode is that the first/last log entry of each sequence is connected to those of the others. As previously mentioned, the three rules (rule4~rule6) are proposed to closely link daily log sequences, reflecting user's regular behavior (close to the average of log entries), and separate malicious ones (far exceeding the average). Therefore, this connection satisfies our demand. How to partition anomalous log sequences depends on edge weight as shown in Figure 4b. The similar numbers of log entries involved in two sequences cause a high weight between them. Appendix A details edge weight computation.

Further, we propose a rule involving H:

Rule5 (edge5): *daily log sequences of the same host are connected and weights are positively related to similarity of the numbers of log entries that they contain.*

Last, we present a rule involving *sub*, H, A and T:

Rule6 (edge6): *daily log sequences of the same operation type and the same host are connected and weights are positively related to similarity of the numbers of log entries that they contain.*

3.2.3 Constructing graph based on objects. Object is a relatively complicated meta-attribute because it indicates different types of entities such as domain names, files, emails. In particular, objects of logon operation are destination hosts as previously mentioned. We design rules to preserve relationships among objects in the graph based on attack of intranet and penetration from the Internet.

In intranet, an *APT* actor compromises a computer. If he cannot gain information or permissions that he wants. He would employ this account to seek for other users' credentials [4, 50]. For instance, the attacker utilizes the stolen account to log in to a remote server (*obj*) from a computer and injects a backdoor for monitoring information stream and stealing credentials. However, the authentication protocol exposes the threat. This server may be a database used for information query and a service principal name (SPN), specifying the service that this server provides, has been registered in active directory (AD). Therefore, the victim accesses this server for querying information through Kerberos protocol. However, other services (e.g. installing software involving a backdoor) have not been registered in AD. NTLM, an alternative protocol, would replace Kerberos. Hence, we partition normal logon and anomalous one into different subgraphs, through differentiating authentication protocols and hosts for each user:

Rule7 (edge7): *log entries of accessing the same destination host from the same source host with the same authentication protocol are connected in chronological order with the highest weight (value 1).*

Rule8a (edge8): *log sequences of the same destination host and source one with different authentication protocols are connected and weights are positively related to the similarities of the numbers of log entries that they contain.*

Rule8b (edge8): *log sequences of different destination hosts or source ones with the same authentication protocol are connected and weights are positively related to the similarities of the numbers of log entries that they contain.*

Appendix A introduces weight computations of Rule8a and Rule8b.

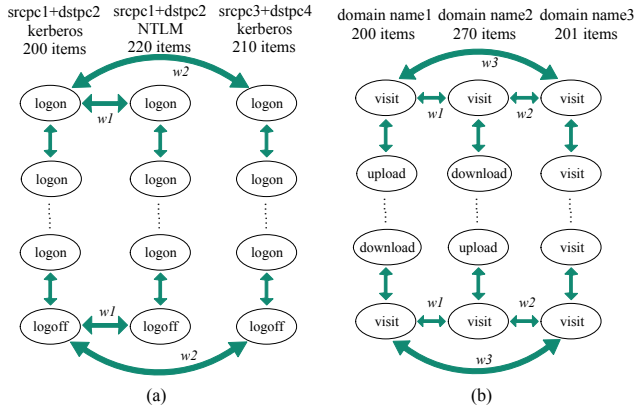


Figure 5: Examples of constructing graph among objects (hosts and domain names). (a) logon operations of the same source host, destination one and authentication protocol (e.g. Kerberos, NTLM) are connected into log sequences. The first sequence and the second one are connected due to the same destination and source host (rule8a). The first one and the third one are connected due to the same authentication protocol (rule8b). (b) log entries of accessing the same domain name are connected into log sequences. Log2vec correlates sequences of different domain names.

Internet is a major channel whereby an employee leaks intelligence to the outside (e.g. upload data). Meanwhile, malware infecting hosts in intranet is mostly from the Internet. For instance, a victim accidentally downloads a malicious tool from an unfamiliar website or clicks a link in an email and installs malware. Afterward, the program executes *APT*s. Therefore, websites (or domain names) are crucial objects [20, 24]. We propose rules for each user:

Rule9 (edge9): *log entries of the same host and accessing the same domain name are connected in chronological order with the highest weight (value 1).*

The goal of an attacker is to entice users to download the malicious code and execute it [17]. Therefore, websites providing software to download, instead of news to visit, are relatively more dangerous. Access mode (visit, download, upload) is an important element and we present a rule:

Rule10 (edge10): *log sequences of the same host and different domain names are connected and weights are positively related to the similarities of accessing modes and numbers of log entries that they contain.*

Figure 5b depicts an instance. In log2vec, the access mode is superior to the number. Thereby, weight $w1$ should be higher than $w2$ and $w3$ because of no uploads and downloads in sequence of domain name3 (implying a website just providing news), which is a difference in accessing mode. Appendix B details how to calculate edge weight.

4 GRAPH EMBEDDING

In Section 4.1, we present an improved random walk capable of extracting context of each node (also log entry) from the above heterogeneous graph and Section 4.2 introduces a specific word2vec model, to represent each node.

4.1 Random Walk with Different Sets of Edge Type

Random walk generates paths traversed by walkers based on edge type and weight. The transition probability is shown in the following paragraph. Log2vec's improvements are controlling the number of neighbor nodes (*neigh*) and extracting context in varying proportions of sets of edge type (*ps*), which are designed towards tackling problems of imbalanced dataset and varied attack scenarios.

4.1.1 Transition probability. We specify a set of edge type, S_n (e.g. {edge1, edge4}) and the walker only considers edges, pertaining to this set, to traverse. The transition probability at node v is defined:

$$P(t|v) = \begin{cases} \frac{w_{(t,v)}}{W_{N(v)}}, & (t,v) \in E, \psi(t,v) \in S_n, R_{w_{(t,v)}} \leq neigh \\ 0, & (t,v) \in E, \psi(t,v) \in S_n, R_{w_{(t,v)}} > neigh \\ 0, & (t,v) \in E, \psi(t,v) \notin S_n \\ 0, & (t,v) \notin E \end{cases} \quad (1)$$

where $N(v)$ denotes specific neighbor nodes of v and $W_{N(v)}$ is the sum of weights between v and $N(v)$. In fact, $N(v)$ does not involve v's all neighbors and only comprises ones that satisfy the three subsequent conditions. Specifically, t is an arbitrary neighbor node to v. If t belongs to $N(v)$, it must meet the three requirements. First, there is at least an edge between t and v. Second, there exists at least an edge type between t and v, $\psi(t,v)$, pertaining to S_n. Note that there may be more than one edge type between t and v but each random walk actually needs only one of them, which is introduced in the next paragraph, thus $\psi(t,v)$ referring to just one type (e.g. edge1 or edge4) in this formula. In the third condition, $w_{(t,v)}$ indicates weight between t and v, corresponding to the edge type $\psi(t,v) \in S_n$. $R_{w_{(*,v)}}$ means the descending weight rank order of a certain node connecting to v with the specific edge type $\psi(*,v) \in S_n$. $w_{(t,v)}$ is taken into account iff its ranking $R_{w_{(t,v)}}$ is not greater than the number of neighbor nodes (*neigh*) that we set in Section 4.1.2.

With regard to our task, we concretely define five sets of edge type, {edge1, edge4}, {edge2, edge5}, {edge3, edge6}, {edge7, edge8} and {edge9, edge10}, respectively corresponding to ten rules in Section 3.2. Only one set is considered in each random walk. This definition allows us to differentially extract each node's context. It also guarantees that each two nodes share at most one edge type in function 1. Specifically, in each set, there exist a type focusing on intra group (e.g. edge1, edge2, edge3, edge7 and edge9) and another paying attention to inter group (e.g. edge4, edge5, edge6, edge8 and edge10). For instance, we specify S_n as {edge1, edge4}. Edge1 concatenates nodes (log entries) within a day while edge4 correlates log sequences of different days. As a result, there is not such two nodes, whose edge types simultaneously belong to these two types.

4.1.2 The number of neighbor nodes. Each set involves two edge types, separately focusing on intra group (the same day/object) and inter group, as previously mentioned. Edge type of intra group (e.g. edge1 and edge9) constructs sequences of log entries and each node in such a sequence has two neighbor nodes. Log2vec allows the walker not to revisit the last node that he has visited and thus he actually has only one node to visit in the sequence. Edge type of inter group (e.g. edge4 and edge10) correlates multiple log sequences.

Therefore the first/last node of a sequence has many neighbor nodes. Log2vec, in essence, tunes the numbers of neighbor nodes (*neighs*) of these nodes.

There exist two reasons why log2vec tunes *neigh*. The first one is that insider threat typically embraces only a few malicious log entries. They cannot be isolated in the vector space if random walk considers all neighbor nodes. For instance, we suppose that there exist two malicious daily sequences containing similar numbers of log entries. Meanwhile, these two sequences also connect to other benign ones comprising far different numbers of log entries from them. Although the edge weight between two malicious sequences ($w_{(t,v)}$) is very high due to the similar numbers of log entries, there are such many benign ones connecting to them (increasing $W_{N(v)}$) that transition probability between these two malicious ones ($P(t|v)$) sharply decreases. Therefore, context of these malicious sequences inevitably involves a large number of benign ones and their vectors cannot be isolated.

Second, the fact that in an enterprise more than one employee may perform attacks [21], results in different numbers of malicious log sequences. The optimal *neigh* is thereby varied based on various users. It is noticeable that *neigh* is a hyperparameter and set experimentally. However, we in principle tend to set a lower value, such as 1, to ensure that the most similar sequence connects to the given one.

4.1.3 The proportion of sets of edge type. Log2vec can determine importance of each set of edge type based on different scenarios and differentially extract contextual information. Generally, varied attack scenarios leave different traces to be detected. That is, only one or two meta-attributes of a log entry become anomalous in each kind of insider attacks. For instance, an employee stealthily connects a storage device to a computer and copies a large number of confidential files. However, he seldom performs such operations before. Log2vec should catch this trace and extract more contextual information using {edge3, edge6} (operation type, *A*). Another malicious user uploads confidential files to an unfamiliar website. Log2vec should capture information using {edge9, edge10} (website, *obj*).

More specifically, log2vec initially gives the proportion, 1:1:1:1:1, corresponding to {edge1, edge4}, {edge2, edge5}, {edge3, edge6}, {edge7, edge8} and {edge9, edge10}. Next, log2vec selects {edge3, edge6}, {edge7, edge8} and {edge9, edge10} to adjust because they cover meta-attributes concerning anomalies. Specifically, {edge1, edge4}, {edge2, edge5} and {edge3, edge6} belong to the same type, focusing on user's daily behavior. Compared with others, {edge3, edge6} involves more meta-attributes, operation type (*A*) and host (*h*), and can produce finer behavioral sequences. {edge7, edge8} and {edge9, edge10} respectively concentrate on destination host (*obj*) in intranet and website (*obj*) in Internet. Last, log2vec compares user's behavior during current period (e.g. two months) and the last one. Meanwhile, it also makes a comparison of his behavior with that of his colleagues, who process similar tasks. For instance, if an employee performs a large number of logons, different from his previous behavior and his colleagues', log2vec should increase the value of {edge7, edge8} in the proportion (*ps*). In practice, the value is two times of sum of other sets and thereby *ps* is 1:1:1:8:1. More details are given in Appendix C.

In addition, we set the number of random walk, r. Each value of the proportion is multiplied by r. We take r=10 as an example. According to the above proportion (1:1:1:8:1), log2vec conducts random walk 80 times and 10 times when we respectively set S_n as {edge7, edge8} and other sets. The pseudocode of this subsection is detailed in Appendix D.

4.2 Word2vec Model

Log2vec employs a word2vec model, skip-gram [32, 33], to map log entries into low-dimensional vectors through context (paths) generated from the heterogeneous graph. It is an objective function that maximizes the probability of its neighbors conditioned on a node. More precisely, it aims to maximize the logarithmic function:

$$\sum_{v=1}^{V} logPr(w_{v-c}, \cdots, w_{v+c}|w_v) \tag{2}$$

where c is the window of training context. We assume that probability of observing its neighbor node is independent of observing any other neighbor node given a log entry and then $Pr(w_{v-c}, \cdots, w_{v+c}|w_v)$ can be computed as

$$\prod_{-c\le j\le c, j\neq 0} Pr(w_{v+j}|w_v) \tag{3}$$

Further, $Pr(w_{v+j}|w_v)$ is defined using the softmax function:

$$\frac{\exp(V_{w_v}^{\top} V'_{w_{v+j}})}{\sum_{i=1}^{V} \exp(V_{w_v}^{\top} V'_{w_i})} \tag{4}$$

where V_{w_i} and V'_{w_i} are respectively the input and output vectors of log entry w_i. $|V|$ in general is very large and the algorithm of Hierarchical softmax, using a binary tree to factorize expensive partition function of the skip-gram model [33, 60], is adopted here to efficiently approximate the function 4. After this model, vectors of log entries are obtained.

5 DETECTION ALGORITHM

Section 5.1 details a clustering algorithm. Section 5.2 introduces a threshold detector. Section 5.3 demonstrates how to select parameters in the detection algorithm.

5.1 Clustering Algorithm

This subsection is to group benign and malicious log entries into different clusters. Nevertheless, the conventional idea of updating cluster centers, such as *k*-means, is not fit for insider threat detection because it heavily relies on the initialization of cluster centers and *k*, leading to unsatisfactory performance.

We adopt an alternative method, adept at log entries' pair-wise similarity comparison. Formally, we suppose clusters $(C_1, C_2, ..., C_n)$ have been obtained and they must satisfy the following conditions:

$\forall x_1 \in C_1, \exists x_2 \in C_1, sim(x_1, x_2) \geq \delta_1$;

$\forall y \in V \backslash C_1, sim(x_1, y) < \delta_1$;

where V is a set including all log entries and δ_1 is a threshold. Note that C_1 is chosen arbitrarily and *sim* is employed to measure similarity between log entries using cosine distance (two log entries become similar when their *sim* is $0 \rightarrow 1$). Its pseudocode is given in Appendix E.

5.2 Threshold Detector

After clustering, log2vec ranks clusters according to the number of log entries they contain. Smaller clusters tend to be suspicious. Table 1 and Table 2 demonstrate two examples of the clustering algorithm's output. In Table 1, the smallest cluster is larger than a threshold, δ_2 (e.g. 80) and thereby identified as legitimate operations. Table 2 shows several small clusters (< 80), indicative of insider threat. Therefore, log entries in these clusters are detected as malicious.

5.3 Parameters Selection

There exist two thresholds, δ_1 and δ_2, to be selected. The conventional methods, e.g. silhouette score [43], cannot work well in our scenarios, because they measure the whole clustering effect, paying equivalent attention to each cluster. We instead propose a more appropriate approach, focusing on smaller clusters:

1) δ_2 is the number of logs involved in the largest suspicious cluster, produced by log2vec. Owing to diverse attack scenarios, it is varied. We set $\delta_2 = pro * total$, where $total$ is the number of logs to be detected and pro is the proportion of malicious log entries to the total. We consider one situation that all malicious log entries are likely to be grouped into one cluster. Therefore, δ_2 should be equal to its number, $pro * total$, and log2vec then detects this cluster as malicious. Analysts can set pro according to the existing work and specific scenarios [17, 34, 46, 57]. In our experiment, we set pro to 1% in all datasets and hence δ_2 is $1\% * total$.

2) In another common situation, malicious log entries are distributed in various clusters. In general, a perpetrator performs several different operations for an attack, e.g. logging into a host (logon) and stealing a file (file copy). According to user's operation type (e.g. logon and file operation), we set the number of malicious clusters ($nmal$), to $nbp * nt$, where nt is the number of operation type. nbp reflects the number of suspicious behavioral patterns on each operation type. In log2vec, different behavioral patterns are grouped into various clusters. $nbp * nt$ thereby constitutes the number of malicious clusters, $nmal$. In practice, we select nbp according to the attribute information regarding each operation type. For instance, an email operation could involve attributes, such as send address, receive address, carbon copy, blind Carbon Copy, size and attachments. Although a malicious email generally possesses one or more anomalous attributes, we set nbp to the number of these attributes for simplicity.

3) When δ_1 is $0 \rightarrow 1$, the number of clusters tend to be larger because two log entries in the same cluster must share a higher similarity. We utilize the number of logs in the largest suspicious cluster, δ_2 and the number of malicious clusters, $nmal$ to set δ_1. When δ_1 is $0 \rightarrow 1$, a cluster splits into many smaller ones. We choose the largest δ_1 to simultaneously cover the following two situations, allowing the number of log entries in clusters lower than δ_2 and the number of these clusters less than or equal to $nmal$.

We take Table 2 as an example, when δ_1 is a value (e.g. 0.64), we view clusters, the number of log entries in which is less than or equal to δ_2 (e.g. 80), as malicious. Meanwhile, the number of these suspicious clusters is less than or equal to $nmal$ (e.g. 6). Then δ_1 is added by an interval (e.g. 0.01) and a new cluster containing 9 log entries is produced. Hence, the number of suspicious clusters is greater than $nmal$, 6, beyond our setting. In other words, when setting δ_1 to 0.65, the suspicious clusters involve benign ones. We thereby choose 0.64 as δ_1.

Until now, log2vec has output malicious logs. Analysts can conduct manual forensics. Certainly, they can also utilize advanced methods regarding forensics [17, 22, 55].

Table 1: log2vec's output for a benign user. The smallest cluster is larger than the threshold (e.g. 80), implying no anomalies.

cluster id	#log
1	162
3	177
2	177
0	7172

Table 2: log2vec's output for a malicious user. Several small clusters < 80, indicative of insider threat.

cluster id	#log
6	1
5	9
4	9
1	31
2	34
3	51
0	4744

6 EXPERIMENT

This section evaluates log2vec in different attack scenarios. Section 6.1 introduces experimental setup. We compare baselines in Section 6.2. Log2vec's detection result is demonstrated in Section 6.3. The parameter sensitivity is assessed in Section 6.4. Section 6.5 and Section 6.6 respectively display the effectiveness of log2vec's sets of edge type and clustering algorithm over k-means.

6.1 Experimental Setup

We conduct this experiment on nine servers and each possesses two Intel Xeon E5-2630 v3 CPUs (32 processors in total) running at 2.4GHz, 128G memory. We utilize spark-2.2.0 to implement components of graph construction and random walk and python3.6 to implement models of skip-gram and detection algorithm.

Dataset. We use a synthetic dataset, CERT Insider Threat Test Dataset [10] and a real-world dataset, Los Alamos National Lab's (LANL's) comprehensive cyber-security events dataset [21].

The CERT dataset is a comprehensive dataset with relatively complete user behavioral records and attack scenarios. We use r6.2, the newest version of this dataset. We utilize five files separately recording logon operation, removable storage device usage, file operation, network operation and email traffic, and another file concerning all users' roles and their affiliations. This dataset comprises 135,117,169 operations of 4,000 users during 516 days. There are five attack scenarios where six users have 470 malicious operations, obviously an extremely imbalanced problem that is common in insider threat detection. Five scenarios include various insider threats, used to examine whether log2vec can determine importance of each set of edge type according to different scenarios, and differentially extract and represent them.

The LANL dataset comprises over one billion log entries collected over 58 days for 12425 users and 17684 computers, within LANL's corporate, internal computer network. It involves 749 malicious logons with 98 compromised accounts, a typical *APT*. We utilize two files separately regarding authentication and process to detect

malicious operations. This dataset examines whether log2vec can detect the attack scenario regarding *APT* mentioned in Section 2.3.

Both datasets are to prove log2vec's powerful detection on users' malicious operations and capability of covering various attacks.

Parameter settings. In CERT dataset, we detect the six malicious users' behavior during two months and take the last ones as the last period used in random walk. The specific time is shown in Table 5 in Appendix F. During the chosen two months, these users acted 37,257 operations involving 428 malicious ones. Also, twelve benign users, performing 71,334 operations, are randomly chosen to examine FPRs. The organizational units in this enterprise are business_unit, functional_unit, department, team and supervisor in descending order of size and the first four units and role (e.g. Salesman, ITAdmin) are used to define a colleague used in random walk. Due to no log entries regarding logons with authentications in this dataset, we mainly utilize rule1~rule6, rule9 and rule10 in Section 3 to construct graphs. In random walk, {edge3, edge6} and {edge9, edge10} are taken into account to determine the proportion of sets of edge type. According to policies in Appendix C, due to explicit averages, changes of connect and network operation in quantity separately correspond to these two sets. The specific proportions for six malicious users, calculated by log2vec, are shown in column *ps* in Table 6 in Appendix H.

The LANL dataset incorporates 98 malicious users and we randomly choose 50 of them to examine log2vec's effectiveness. This detected dataset involves 701,643 log entries, which contains 495 malicious operations, 66.1% of the total. In addition, we randomly choose 90 benign users, who performed 983,421 operations, to examine FPRs. As this dataset is mainly used to detecting *APT* and there is no information regarding websites, log2vec utilizes rule1~rule8 in Section 3 to construct graphs and fixes proportions of sets of edge type for all users to 1:1:1:6 in random walk.

Other parameters for the two datasets are set as follows. In random walk, we set walk length $l = 60$, number of walks per node $r = 10$ and the number of neighbor nodes $neigh = 1$. In skip-gram, vector's dimension d is 100 and window c is 10. In detection algorithm, we set the threshold $\delta_2 = pro * total$, where $pro = 1\%$, $total$ is the number of log entries to be detected for each user. The number of malicious clusters, $nmal$ is $nbp * nt$. The number of operation type $nt = 5$ in CERT dataset and 2 in LANL dataset. We select nbp according to the largest number of attributes regarding operation type. In CERT dataset, email operation possesses the largest number, 6. In LANL dataset, logon operation owns 8 attributes. According to δ_2 and $nmal$, we obtain the similarity threshold δ_1.

Baseline. We utilize eleven baseline methods for CERT dataset. TIRESIAS and DeepLog are advanced log-entry-level approaches on anomaly detection [12, 47]. Hidden markov models [41] (markov-s and markov-c) and deep learning models [57] (DNN and LSTM) are state-of-the-art on this dataset. STREAMSPOT is an advanced approach to detect malicious information flows [31]. We employ node2vec [14] and metapath2vec [11] to show the effect of log2vec's improvement in random walk, on processing heterogeneous graph. Log2vec-euclidean and log2vec-cosine are used to show that log2vec's clustering is superior to k-means in solving this problem. In addition, we introduce a new version of log2vec, log2vec++, whose parameters are flexibly set according to different users and attacks. We use an ensemble detection method [4] and TIRESIAS for LANL dataset to display the effectiveness of log2vec on *APT*. Their specific parameters are detailed in Appendix G.

Table 3: Detection result of different approaches

Method (dataset)	AUC	Method (dataset)	AUC
log2vec (CERT)	**0.86**	log2vec++ (CERT)	**0.93**
TIRESIAS (CERT)	0.39	DeepLog (CERT)	0.10
DNN (CERT)	0.73	LSTM (CERT)	0.71
markov-s (CERT)	0.79	markov-c (CERT)	0.80
metapath2vec (CERT)	0.61	node2vec (CERT)	0.54
log2vec-Euclidean (CERT)	0.36	log2vec-cosine (CERT)	0.35
STREAMSPOT (CERT)	0.70	TIRESIAS (LANL)	0.85
log2vec (LANL)	**0.91**	ensemble method (LANL)	0.89

6.2 Comparison with baselines

We employ AUC (area under the ROC curve) to compare different approaches' performances. In Table 3, log2vec outperforms baselines. TIRESIAS and DeepLog are the state-of-the-art log-entry-level approaches to anomaly detection. In CERT dataset, they both only employ ***causal and sequential relationship within a day***, without the other two relationships, ***logical relationships among days*** and ***logical relationships among objects***. Therefore, they cannot achieve satisfactory performances (0.39, 0.10). Lack of enough malicious samples also causes this result. For instance, TIRESIAS requires pre-labeled security events for training, but our scenarios, CERT and LANL datasets are imbalanced. As shown in Table 4, some users perform 22, 18 or even 4 malicious actions.

Deep learning methods (DNN and LSTM) are different from TIRESIAS and DeepLog. Specifically, TIRESIAS takes sequences of log entries ordered by time as input, while the LSTM in baselines uses sequences of statistics features extracted from log entries per day. They produce suspicious days but we compute their AUCs according to benign and malicious log entries per day. Although DNN and LSTM are inferior to log2vec in log-entry level, they achieve better performances than TIRESIAS and DeepLog, because they consider more relationships, e.g. ***logical relationships among days***. Hidden markov models (markov-s and markov-c) are designed to identify suspicious days during which malicious events occur. STREAMSPOT aims to detect malicious information flow graphs. Instead, Table 3 shows their AUCs in log-entry level (more details in Appendix G). These methods as well cannot beat log2vec.

Metapath2vec and node2vec are advanced graph embedding models. Node2vec is designed to process a homogeneous graph and thereby achieves an inferior performance. Metapath2vec can process a heterogeneous graph. In fact, the main difference between metapath2vec and log2vec's graph embedding is that log2vec retains the capability of adjusting the proportion of sets of edge type. If the proportion is default (1:1:1:1:1), these two methods are the same. Apparently, insider threats demand different proportions and hence log2vec achieves a superior performance. Note that log2vec-Euclidean and log2vec-cosine are discussed in Section 6.6

In LANL dataset, there exist enough data for TIRESIAS's training (3.6 million log entries). The reason why it can obtain a presentable

result (0.85), is that we design a specified event in the dataset. Specifically, TIRESIAS views events, which appear less than 10 times, as the specified one. Therefore, TIRESIAS successfully classifies this event as anomalous, due to enough samples. However, the specified event involves numerous benign events. If TIRESIAS learns patterns of these various rare events, whose average number of occurrence is 2.1, instead of the specified one, it does not work (AUC is 0.05). This is because the dataset in our scenarios is extremely imbalanced and lacks enough malicious samples for training. Moreover, TIRESIAS neglects other relationships, especially ***logical relationships among objects***. The ensemble method identifies characteristic patterns of *APT* as statistical features and analyzes them to detect compromised hosts, while log2vec performs a more fine-grained detection, identifying malicious log entries.

Log2vec's potential. Log2vec++ remarkably outperforms log2vec. The main difference between them is that two parameters of log2vec++, *neigh*++ and δ_1++ are flexibly set as shown in Table 6 in Appendix H, while log2vec's *neigh* is 1 and δ_1 is set according to Section 5.3. Two reasons contribute to the difference. First, insider threat and *APT* datasets are generally extremely imbalanced. We need to exactly set combinations of parameters for each user to figure out a very few malicious operations. Second, multiple users and attacks produce various user's behavior and attack patterns. Log2vec should precisely extract and represent information indicating these patterns and behavior from log entries. Therefore, it demands different combinations of parameters for each user.

Table 4: Log2vec's detection results on six malicious users (e.g. ACM2278) in CERT and 50 attackers in LANL dataset. The metrics is the number of detected malicious log entries/ the number of total malicious log entries.

User Id	ACM2278	CMP2946	HIS1706	PLJ1771
# detected log entries/ # total	22/22	155/242	4/8	11/18
User Id	CDE1846	MBG3183	LANL dataset	
# detected log entries/ # total	134/134	4/4	419/495	

6.3 Detection of Malicious Users

Table 4 shows log2vec's detection result. Although log2vec does not detect all malicious operations, it achieves the best performances on detecting insider attacks (CERT) and *APT* (LANL) compared with baselines (see Section 6.2). Log2vec shows its generic to multiple distinct attacks: 1) stealing data by removable drive or data upload (ACM2278, CMP2946, MBG3183), 2) logging into other users for confidential files (CDE1846). These two types of attacks belong to the first scenario mentioned in Section 2.3. 3) Masquerade attack. Attacker steals credentials of other legitimate users through key logger and sends out alarming emails, causing panic (HIS1706, PLJ1771). This is the second scenario. 4) The *APT* perpetrator compromises a host and from this host persistently compromises multiple ones (LANL dataset), belonging to the third scenario.

6.4 Parameter Sensitivity

We study how the six parameters in graph embedding influence log2vec's performance on detecting user CMP2946 in CERT dataset. Specifically, we employ the metrics, AUC as a function of these six parameters. When testing one parameter, the other five are fixed to their default values. Meanwhile, δ_1 in the detection algorithm is set to 0.65, at which log2vec achieves the best performance according to Section 5.3. δ_2 is adjusted to compute AUC.

Figure 6a depicts a first-increasing and then-steady line, implying that 10 walks per node (r) are enough to extract the context of each node (log entry). Similarly, after l and d reach around 60 and 100 respectively, log2vec's performances do not significantly increase in Figure 6b and Figure 6c. In Figure 6d, we observe that AUC rises to the peak at c = 10. We classify these four parameters into the same category. They are all insensitive to detected users and attacks. More specifically, when r = 10, l = 60, d = 100 and c = 10, log2vec's performance is close to the promising result for all suspicious users and attacks. This finding is different from other experiments in social networks [3, 14]. This is because the number of malicious operations in insider threat detection or *APT* detection is significantly small. Therefore, the four values are enough to extract each log entry's context and learn its representation. Meanwhile, remaining undetected malicious log entries are mainly accounted for by other log2vec's parameters.

In Figure 6e, when *ps* is 1:1:1:5, log2vec achieves the best performance. In Figure 6f, value 4, 7, 8 and 9 produce promising performances. In Figure 7a, we jointly tune *ps* and *neigh*. Log2vec apparently achieves the best performances when *ps* is 1:1:1:6, no matter how *neigh* is adjusted. *ps* thereby is a more crucial parameter than *neigh*. In contrast to the aforementioned four parameters, *ps* and *neigh* are sensitive to varied users and attacks. In Table 6 in Appendix H, log2vec and log2vec++ flexibly adjust *ps* and outperform other approaches. Moreover, log2vec++ also adjusts *neigh* and δ_1, and thereby beats log2vec. Besides, δ_2 is as well sensitive to scenarios according to its definition. Therefore, we classify these four parameters into the same category, sensitive to detected users and attacks. Section 4.1.3 introduces how to set *ps*. We usually set *neigh* to value 1. The way of setting δ_1 and δ_2 is given in Section 5.3.

6.5 Effectiveness of Log2vec's Sets of Edge Type

As shown in Figure 3, five out of the ten rules are used to construct log sequences, e.g. rule1, and the other five are to respectively bridge them, e.g. rule4. These two categories of rules are complementary to each other. If we cut down one of the former rules, e.g. rule1, its corresponding rule in the latter, e.g. rule4, cannot be applied to the graph construction due to lack of daily log sequences. If we just employ the former, e.g. rule3 without rule6, log2vec only constructs user's daily log sequences and cannot detect anomalous ones like the instance in Section 2.2. Therefore, we evaluate log2vec's sets of edge type, defined by these rules in pairs.

We conduct the evaluation on the six attackers in CERT dataset and six chosen randomly ones in LANL dataset. Specifically, we cut down a set of edge type and use the remaining ones for detection. Figure 7b demonstrates that performances all drop without any set of edge type, especially {edge3, edge6} and {edge7, edge8}, the most important ones for varied scenarios as mentioned in Section 4.1.3.

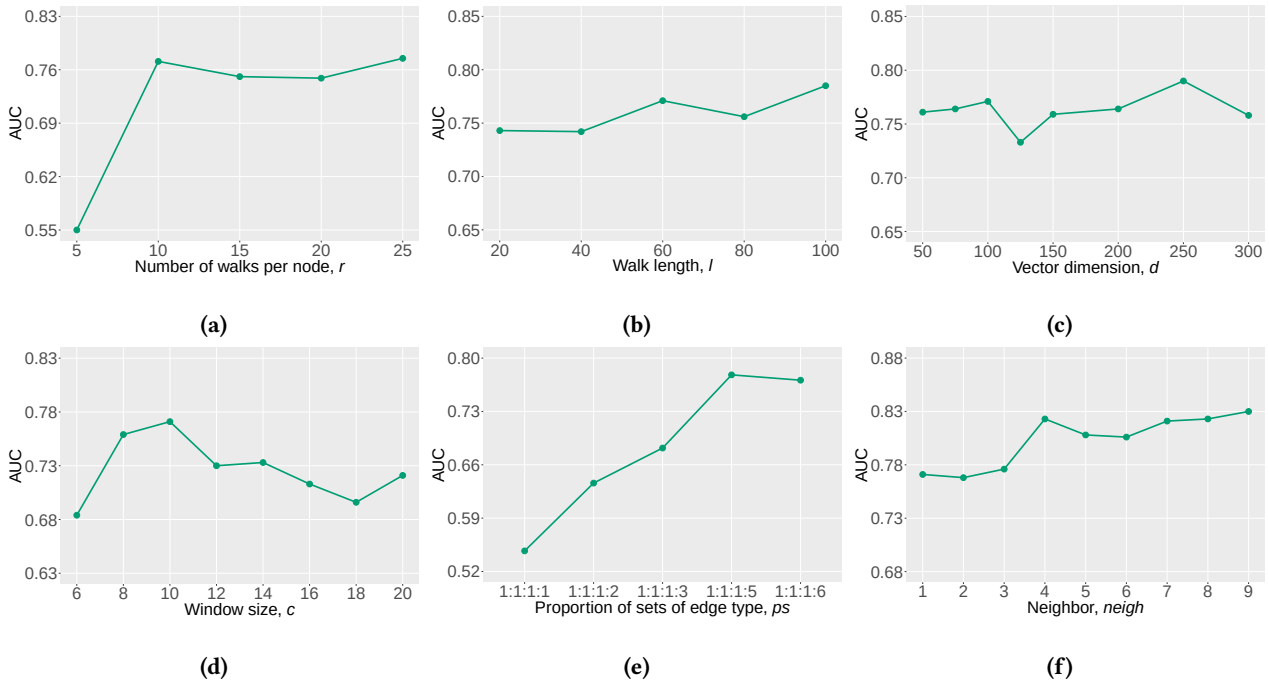


Figure 6: Performance of log2vec on different parameters.

Log2vec's performance without {edge9, edge10} drops remarkably on users, who launch attacks through the Internet, regarding Rule9 and Rule10. Log2vec's without {edge1, edge4} or {edge2, edge5} performance decreases mainly when the original proportion of the sets of edge type is 1:1:1:1, meaning these two sets play a nearly equal role to the others in detecting these specific attackers.

6.6 Effectiveness of Log2vec's Clustering

We take user CDE1846 in CERT dataset, as an example to demonstrate that log2vec's clustering algorithm is more fit for the attack scenarios in this paper, compared with k-means. Figure 7c presents their Receiver Operating Characteristic curves (or ROC curves) smoothed by the method of bionormal [15]. The main difference between them is that they separately employ k-means with cosine distance, Euclidean one and clustering algorithm in Section 5.1. In the figure, AUCs regarding k-means are all below 0.5. This is because these methods use cluster centers to group log entries, leading to heavily depending on their initialization. However, the number of malicious operations in insider threat dataset is extremely small. Accordingly, these operations are hardly chosen as centers during initialization and tend to be grouped into their neighboring clusters. Another reason is that k is difficult to set. If k is too small, malicious log entries are more likely to be gathered with benign ones.

7 DISCUSSION

7.1 Evasion

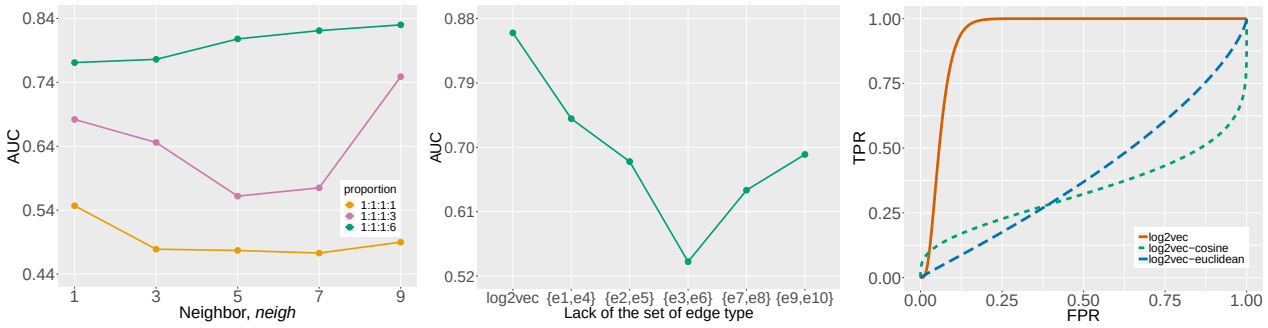
Log2vec figures out malicious clusters in the light of their sizes. If the attacker plans to evade detection, he may conduct enough malicious operations to expand the corresponding cluster as large as benign ones. However, this process would more likely draw attention to his behavior. For instance, the existing security information and event management (SIEM) would detect these operations.

7.2 Limitations

Graph rules. Log2vec is designed towards detecting three significant scenarios regarding insider threat and *APT* as introduced in the adversarial model. The three relationships used in log2vec are derived from the existing method [4, 38, 41, 50, 57]. These relationships involve user's behavioral pattern, attack of intranet and penetration from the Internet. Certainly, we can consider new relationships and design new rules. For instance, new features of *APT* are still exploited and we can put them into the graph. Similarly, there exist other attack scenarios [34]. Therefore, we need to incorporate new relationships into the graph for covering them. For instance, email networks can be converted into graph rules for detecting spearphishing. These plans are left to future work.

False positive rate. FPRs produced by log2vec are respectively 0.08 and 0.1 in both datasets, fairly high in a production environment and would increase their efforts during analysts' analysis. To tackle this problem, we can employ the existing methods regarding combating threat alert fatigue [6, 16, 25] to assist analysts in reducing false positives. This direction is also our future work.

Log2vec's parameters. As mentioned in log2vec's potential (see Section 6.2), although log2vec outperforms baselines, this version of graph embedding and detection algorithm still can be improved, like log2vec++, if we can tune parameters flexibly for each user. This work can be conducted in two directions. The first one is to research on choosing parameters automatically. Another is to



(a) (b) (c)

Figure 7: (a) Log2vec's performance when jointly tuning *neigh* and *ps*. (b) Log2vec's performance without certain sets of edge type, e.g {e1, e4} meaning lack of {edge1, edge4}. (c) Performance of log2vec on user CDE1846 in CERT dataset.

improve the existing version of graph embedding and detection algorithm, e.g. reducing parameters.

Graph neural network. Graph neural network (GNN) is a prevailing method and has outperformed graph embedding in such various domains, as citation network, knowledge graphs and social events forecasting [9, 23, 36, 37]. GNN has a more complicated structure for learning graph's topology, compared with graph embedding's word2vec model, e.g. skip-gram. However, a majority of its variants require labeled data for training. Insider threat detection and *APT* detection are data imbalanced problems and there exist only a few anomalous samples. Therefore, it needs numerous efforts to employ GNN's powerful function in this field.

8 RELATED WORK

Insider Threat Detection and *APT* Detection. The issues of insider threat detection and *APT* detection have been extensively researched on various scenarios [5, 18, 48, 49].

A typical method is extracting features of user's behavior and processing them by machine learning models to find malicious events [13, 27, 57, 61]. Features from social data summarized in [13], include email communication pattern and online activity data such as web browsing patterns, and file and machine access patterns. In company, features also incorporate login, printer and connect [46].

Another typical approach is transforming user's behavior into sequences or graphs and analyzing them [12, 35, 38, 40, 41, 47]. The methods of this kind construct normal patterns of user behavior and then compare them with new behavior or predict new one to identify anomalies, while log2vec transforms relationships among log entries into a graph and use graph embedding to mine anomalies based on various relationships among log entries.

Provenance tracking systems are proposed to monitor and analyze activities of a system [17, 22, 26, 30, 55]. There exist two differences between log2vec and them. First, a majority of them aim for attack forensics instead of cyber threat detection. Second, these systems utilize causal graphs to track operations and interactions of processes (e.g. IPC syscalls operating on pipes and semaphores) in the system-call level. Log2vec analyzes logs recording user's behavior in information system (e.g. connect a removable storage device and browse a website). It mainly captures and represents multiple relationships among logs that reflect user's typical behavior.

HOLMES is an advanced approach to detect *APT* [34]. However, it constructs high-level scenario graphs according to information flows, only reflecting causal and sequential relationship among log entries. Thereby, its detection results produced by these scored graphs according to the severity levels of their components, cannot achieve a satisfactory performance in our scenarios.

Graph Embedding. Graph analysis has been widely researched and applied to diverse domains [8, 19, 53]. Among these methods, graph embedding has risen great attention [44].

Recently, deep learning is used in graph embedding [28, 45] probably due to its outstanding supervised learning ability. Xu et al. utilize this kind of graph embedding to compute the similarity of two binary functions from different platforms [59].

Word2vec-based graph embedding is another kind receiving high concerns [14, 39, 42, 54]. In cybersecurity, Backes et al. use this method to perform a social relation inference attack [3]. Inspired by graph embedding and metapath [52], Dong et al. present a new method processing heterogeneous graph that the above methods cannot [11]. Log2vec improves this method and makes it applicable to the heterogeneous graph in this paper.

9 CONCLUSIONS

We propose log2vec, a heterogeneous graph embedding based cyber threat detection. To the best of our knowledge, log2vec involves the first sophisticated and effective heterogeneous graph construction in this field. To analyze this graph, an improvement of graph embedding is designed whose output is processed by a practical detection algorithm. We implement a prototype of log2vec. The evaluation demonstrates that log2vec outperforms other state-of-the-art methods in log-level granularity. It can detect cyber threats effectively in varied scenarios.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers and Prof. Daphne Yao for their constructive comments. We also thank Dr. Fangxiao Ning's technical help. This work was supported by the Strategic Priority Research Program of CAS, Grant No.XDC02010300.

REFERENCES

[1] David Arthur and Sergei Vassilvitskii. 2007. K-means++: The Advantages of Careful Seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '07)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1027–1035. http://dl.acm.org/citation.cfm?id=1283383.1283494

[2] A. Azaria, A. Richardson, S. Kraus, and V. S. Subrahmanian. 2014. Behavioral Analysis of Insider Threat: A Survey and Bootstrapped Prediction in Imbalanced Data. *IEEE Transactions on Computational Social Systems* 1, 2 (June 2014), 135–155. https://doi.org/10.1109/TCSS.2014.2377811

[3] Michael Backes, Mathias Humbert, Jun Pang, and Yang Zhang. 2017. Walk2Friends: Inferring Social Links from Mobility Profiles. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17)*. ACM, New York, NY, USA, 1943–1957. https://doi.org/10.1145/3133956.3133972

[4] A. Bohara, M. A. Noureddine, A. Fawaz, and W. H. Sanders. 2017. An Unsupervised Multi-Detector Approach for Identifying Malicious Lateral Movement. In *2017 IEEE 36th Symposium on Reliable Distributed Systems (SRDS)*. 224–233. https://doi.org/10.1109/SRDS.2017.31

[5] Y. Chen, C. M. Poskitt, and J. Sun. 2018. Learning from Mutants: Using Code Mutation to Learn and Monitor Invariants of a Cyber-Physical System. In *2018 IEEE Symposium on Security and Privacy (SP)*. 648–660. https://doi.org/10.1109/SP.2018.00016

[6] Tobias Chyssler, Stefan Burschka, Michael Semling, Tomas Lingvall, and Kalle Burbeck. 2004. Alarm Reduction and Correlation in Intrusion Detection Systems.. In *DIMVA*. 9–24.

[7] Matthew L. Collins, Michael C. Theis, Randall F. Trzeciak, Jeremy R. Strozer, Jason W. Clark, Daniel L. Costa, Tracy Cassidy, Michael J. Albrethsen, and Andrew Preston Moore. 2012. Common Sense Guide to Mitigating Insider Threats, Fifth Edition. *Common Sense Guide to Mitigating Insider Threats Edition* (2012).

[8] Hanjun Dai, Bo Dai, and Le Song. 2016. Discriminative Embeddings of Latent Variable Models for Structured Data. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48 (ICML'16)*. JMLR.org, 2702–2711.

[9] Songgaojun Deng, Huzefa Rangwala, and Yue Ning. 2019. Learning Dynamic Context Graphs for Predicting Social Events. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '19)*. ACM, New York, NY, USA, 1007–1016. https://doi.org/10.1145/3292500.3330919

[10] The CERT Division. 2018. Insider Threat Tools. https://www.cert.org/insider-threat/tools/.

[11] Yuxiao Dong, Nitesh V. Chawla, and Ananthram Swami. 2017. Metapath2Vec: Scalable Representation Learning for Heterogeneous Networks. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '17)*. ACM, New York, NY, USA, 135–144. https://doi.org/10.1145/3097983.3098036

[12] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. 2017. DeepLog: Anomaly Detection and Diagnosis from System Logs Through Deep Learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17)*. ACM, New York, NY, USA, 1285–1298. https://doi.org/10.1145/3133956.3134015

[13] Gaurang Gavai, Kumar Sricharan, Dave Gunning, Rob Rolleston, John Hanley, and Mudita Singhal. 2015. Detecting Insider Threat from Enterprise Social and Online Activity Data. In *Proceedings of the 7th ACM CCS International Workshop on Managing Insider Security Threats (MIST '15)*. ACM, New York, NY, USA, 13–20. https://doi.org/10.1145/2808783.2808784

[14] Aditya Grover and Jure Leskovec. 2016. Node2Vec: Scalable Feature Learning for Networks. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*. ACM, New York, NY, USA, 855–864. https://doi.org/10.1145/2939672.2939754

[15] James A. Hanley. 1988. The Robustness of the "Binormal" Assumptions Used in Fitting ROC Curves. *Medical Decision Making* 8, 3 (1988), 197–203. https://doi.org/10.1177/0272989X8800800308 arXiv:https://doi.org/10.1177/0272989X8800800308 PMID: 3398748.

[16] Wajih Ul Hassan, Shengjian Guo, Ding Li, Zhengzhang Chen, Kangkook Jee, Zhichun Li, and Adam Bates. 2019. NoDoze: Combatting Threat Alert Fatigue with Automated Provenance Triage. *Network and Distributed Systems Security Symposium* (Feb 2019). http://par.nsf.gov/biblio/10085663

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

[18] P. Hu, H. Li, H. Fu, D. Cansever, and P. Mohapatra. 2015. Dynamic defense strategy against advanced persistent threat with insiders. In *2015 IEEE Conference on Computer Communications (INFOCOM)*. 747–755. https://doi.org/10.1109/INFOCOM.2015.7218444

[19] Zhipeng Huang, Yudian Zheng, Reynold Cheng, Yizhou Sun, Nikos Mamoulis, and Xiang Li. 2016. Meta Structure: Computing Relevance in Large Heterogeneous Information Networks. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*. ACM, New York, NY, USA, 1595–1604. https://doi.org/10.1145/2939672.2939815

[20] Luca Invernizzi, Stanislav Miskovic, Ruben Torres, Christopher Kruegel, Sabyasachi Saha, Giovanni Vigna, Sung-Ju Lee, and Marco Mellia. 2014. Nazca: Detecting Malware Distribution in Large-Scale Networks.. In *NDSS*, Vol. 14. 23–26.

[21] Alexander D. Kent. 2016. *Cyber security data sources for dynamic network research*. 37–65 pages.

[22] Samuel T. King and Peter M. Chen. 2003. Backtracking Intrusions. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles (SOSP '03)*. ACM, New York, NY, USA, 223–236. https://doi.org/10.1145/945445.945467

[23] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[24] Bum Jun Kwon, Jayanta Mondal, Jiyong Jang, Leyla Bilge, and Tudor Dumitraş. 2015. The Dropper Effect: Insights into Malware Distribution with Downloader Graph Analytics. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security (CCS '15)*. ACM, New York, NY, USA, 1118–1129. https://doi.org/10.1145/2810103.2813724

[25] Aron Laszka, Jian Lou, and Yevgeniy Vorobeychik. 2016. Multi-defender strategic filtering against spear-phishing attacks. In *Thirtieth AAAI Conference on Artificial Intelligence*.

[26] Kyu Hyung Lee, Xiangyu Zhang, and Dongyan Xu. 2013. High Accuracy Attack Provenance via Binary-based Execution Partition.. In *NDSS*. The Internet Society. http://dblp.uni-trier.de/db/conf/ndss/ndss2013.html#LeeZX13

[27] P. A. Legg, O. Buckley, M. Goldsmith, and S. Creese. 2015. Caught in the act of an insider attack: detection and assessment of insider threat. In *2015 IEEE International Symposium on Technologies for Homeland Security (HST)*. 1–6. https://doi.org/10.1109/THS.2015.7446229

[28] Zemin Liu, Vincent W. Zheng, Zhou Zhao, Fanwei Zhu, Kevin Chen-Chuan Chang, Minghui Wu, and Jing Ying. 2017. Semantic Proximity Search on Heterogeneous Graph by Proximity Embedding.

[29] Ponemon Institute LLC. 2016. Cost of Cyber Crime 2016: Reducing the Risk of Business Innovation. https://saas.hpe.com/en-us/marketing/cyber-crime-risk-to-business-Innovation. Accessed Augest 22, 2017.

[30] Shiqing Ma, Juan Zhai, Fei Wang, Kyu Hyung Lee, Xiangyu Zhang, and Dongyan Xu. 2017. MPI: Multiple perspective attack investigation with semantics aware execution partitioning. In *USENIX Security*.

[31] Emaad Manzoor, Sadegh M. Milajerdi, and Leman Akoglu. 2016. Fast Memory-efficient Anomaly Detection in Streaming Heterogeneous Graphs. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*. ACM, New York, NY, USA, 1035–1044. https://doi.org/10.1145/2939672.2939783

[32] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. *CoRR* abs/1301.3781 (2013). arXiv:1301.3781

[33] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 3111–3119.

[34] S. Momeni Milajerdi, R. Gjomemo, B. Eshete, R. Sekar, and V. Venkatakrishnan. 2019. HOLMES: Real-Time APT Detection through Correlation of Suspicious Information Flows. In *2019 2019 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, Los Alamitos, CA, USA, 447–462. https://doi.org/10.1109/SP.2019.00026

[35] Pablo Moriano, Jared Pendleton, Steven Rich, and L Jean Camp. 2017. Insider Threat Event Detection in User-System Interactions. In *Proceedings of the 2017 International Workshop on Managing Insider Security Threats (MIST '17)*. ACM, New York, NY, USA, 1–12. https://doi.org/10.1145/3139923.3139928

[36] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. 2016. Learning convolutional neural networks for graphs. In *International conference on machine learning*. 2014–2023.

[37] Namyong Park, Andrey Kan, Xin Luna Dong, Tong Zhao, and Christos Faloutsos. 2019. Estimating Node Importance in Knowledge Graphs Using Graph Neural Networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '19)*. ACM, New York, NY, USA, 596–606. https://doi.org/10.1145/3292500.3330855

[38] P. Parveen, N. McDaniel, V. S. Hariharan, B. Thuraisingham, and L. Khan. 2012. Unsupervised Ensemble Based Learning for Insider Threat Detection. In *2012 International Conference on Privacy, Security, Risk and Trust and 2012 International Confernece on Social Computing*. 718–727. https://doi.org/10.1109/SocialCom-PASSAT.2012.106

[39] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. DeepWalk: Online Learning of Social Representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '14)*. ACM, New York, NY, USA, 701–710. https://doi.org/10.1145/2623330.2623732

[40] William Eberle PhD, Jeffrey Graves, and Lawrence Holder PhD. 2010. Insider Threat Detection Using a Graph-Based Approach. *Journal of Applied Security Research* 6, 1 (2010), 32–81. https://doi.org/10.1080/19361610.2011.529413 arXiv:https://doi.org/10.1080/19361610.2011.529413

[41] Tabish Rashid, Ioannis Agrafiotis, and Jason R.C. Nurse. 2016. A New Take on Detecting Insider Threats: Exploring the Use of Hidden Markov Models. In *Proceedings of the 8th ACM CCS International Workshop on Managing Insider Security Threats (MIST '16)*. ACM, New York, NY, USA, 47–56. https://doi.org/10.1145/2995959.2995964

[42] Leonardo F.R. Ribeiro, Pedro H.P. Saverese, and Daniel R. Figueiredo. 2017. Struc2Vec: Learning Node Representations from Structural Identity. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '17)*. ACM, New York, NY, USA, 385–394. https://doi.org/10.1145/3097983.3098061

[43] Peter J. Rousseeuw. 1987. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *J. Comput. Appl. Math.* 20 (1987), 53 – 65. https://doi.org/10.1016/0377-0427(87)90125-7

[44] Sam T. Roweis and Lawrence K. Saul. 2000. Nonlinear Dimensionality Reduction by Locally Linear Embedding. *Science* 290, 5500 (2000), 2323–2326. https://doi.org/10.1126/science.290.5500.2323

[45] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. 2009. The Graph Neural Network Model. *IEEE Transactions on Neural Networks* 20, 1 (Jan 2009), 61–80. https://doi.org/10.1109/TNN.2008.2005605

[46] Ted E. Senator, Henry G. Goldberg, Alex Memory, William T. Young, Brad Rees, Robert Pierce, Daniel Huang, Matthew Reardon, David A. Bader, Edmond Chow, Irfan Essa, Joshua Jones, Vinay Bettadapura, Duen Horng Chau, Oded Green, Oguz Kaya, Anita Zakrzewska, Erica Briscoe, Rudolph IV L. Mappus, Robert McColl, Lora Weiss, Thomas G. Dietterich, Alan Fern, Weng-Keen Wong, Shubhomoy Das, Andrew Emmott, Jed Irvine, Jay-Yoon Lee, Danai Koutra, Christos Faloutsos, Daniel Corkill, Lisa Friedland, Amanda Gentzel, and David Jensen. 2013. Detecting Insider Threats in a Real Corporate Database of Computer Usage Activity. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '13)*. ACM, New York, NY, USA, 1393–1401. https://doi.org/10.1145/2487575.2488213

[47] Yun Shen, Enrico Mariconti, Pierre Antoine Vervier, and Gianluca Stringhini. 2018. Tiresias: Predicting Security Events Through Deep Learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18)*. ACM, New York, NY, USA, 592–605. https://doi.org/10.1145/3243734.3243811

[48] Xiaokui Shu, Danfeng Yao, and Naren Ramakrishnan. 2015. Unearthing Stealthy Program Attacks Buried in Extremely Long Execution Paths. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security (CCS '15)*. ACM, New York, NY, USA, 401–413. https://doi.org/10.1145/2810103.2813654

[49] Xiaokui Shu, Danfeng (Daphne) Yao, Naren Ramakrishnan, and Trent Jaeger. 2017. Long-Span Program Behavior Modeling and Attack Detection. *ACM Trans. Priv. Secur.* 20, 4, Article 12 (Sept. 2017), 28 pages. https://doi.org/10.1145/3105761

[50] Hossein Siadati and Nasir Memon. 2017. Detecting Structurally Anomalous Logins Within Enterprise Networks. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17)*. ACM, New York, NY, USA, 1273–1284. https://doi.org/10.1145/3133956.3134003

[51] SolarWinds. 2015. SolarWinds Survey Investigates Insider Threats to Federal Cybersecurity. http://www.solarwinds.com/company/newsroom/press_releases/threats_to_federal_cybersecurity.aspx. Accessed August 22, 2017.

[52] Yizhou Sun, Jiawei Han, Xifeng Yan, Philip S. Yu, and Tianyi Wu. 2011. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. *Proceedings of the VLDB Endowment* 4, 11 (8 2011), 992–1003.

[53] Yizhou Sun, Brandon Norick, Jiawei Han, Xifeng Yan, Philip S. Yu, and Xiao Yu. 2012. Integrating Meta-path Selection with User-guided Object Clustering in Heterogeneous Information Networks. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '12)*. ACM, New York, NY, USA, 1348–1356. https://doi.org/10.1145/2339530.2339738

[54] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. LINE: Large-scale Information Network Embedding. In *Proceedings of the 24th International Conference on World Wide Web (WWW '15)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, 1067–1077. https://doi.org/10.1145/2736277.2741093

[55] Yutao Tang, Ding Li, Zhichun Li, Mu Zhang, Kangkook Jee, Xusheng Xiao, Zhenyu Wu, Junghwan Rhee, Fengyuan Xu, and Qun Li. 2018. NodeMerge: Template Based Efficient Data Reduction For Big-Data Causality Analysis. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18)*. ACM, New York, NY, USA, 1324–1337. https://doi.org/10.1145/3243734.3243763

[56] TrendMicro. 2012. APT myths and challenges. http://blog.trendmicro.com/trendlabs-security-intelligence/infographic-apt-myths-and-challenges/. Accessed November 21, 2018.

[57] Aaron Tuor, Samuel Kaplan, Brian Hutchinson, Nicole Nichols, and Sean Robinson. 2017. Deep Learning for Unsupervised Insider Threat Detection in Structured Cybersecurity Data Streams. *CoRR* abs/1710.00811 (2017). arXiv:1710.00811

[58] N. E. Weiss and R. S. Miller. 2015. The Target and other financial data breaches: Frequently asked questions. https://fas.org/sgp/crs/misc/R43496.pdf. Accessed November 21, 2018.

[59] Xiaojun Xu, Chang Liu, Qian Feng, Heng Yin, Le Song, and Dawn Song. 2017. 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 (CCS '17)*. ACM, New York, NY, USA, 363–376. https://doi.org/10.1145/3133956.3134018

[60] Pinar Yanardag and S.V.N. Vishwanathan. 2015. Deep Graph Kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '15)*. ACM, New York, NY, USA, 1365–1374. https://doi.org/10.1145/2783258.2783417

[61] Ting-Fang Yen, Alina Oprea, Kaan Onarlioglu, Todd Leetham, William Robertson, Ari Juels, and Engin Kirda. 2013. Beehive: Large-scale Log Analysis for Detecting Suspicious Activity in Enterprise Networks. In *Proceedings of the 29th Annual Computer Security Applications Conference (ACSAC '13)*. ACM, New York, NY, USA, 199–208. https://doi.org/10.1145/2523649.2523670

A EDGE WEIGHT COMPUTATION AMONG DAILY LOG SEQUENCES

Edge weight w is directly proportional to similarity of the numbers of logs that two daily log sequences contain. Specifically, if the numbers of them are close, their weight w is high (smaller than 1) and even equals to 1 when identical. We employ the function:

$$d(a,b) = \frac{max(a,b)}{min(a,b)} - 1 \tag{5}$$

where a and b indicate the numbers of log entries that two sequences contain. Apparently, when a and b are close, then $d(a,b)$ is approaching zero and $d(a,b)$ equals to 0 only if $a = b$. Specially, when a or b is zero, let $d(a,b)$ equals to 10000 that is large enough to achieve our purpose and let $d(a,b)$ equals to 0 when a and b are both zero, which are both needed in Section 3.2.3. After getting $d(a,b)$, we introduce the following edge weight function:

$$w(u,v) = e^{-d(a,b)} \tag{6}$$

where u and v denote two daily log sequences.

It can be seen that this function is well suited to our requirement. For instance, if a and b are close, $d(a,b)$ approaches zero and w tends to be maximum, that is, 1. In particular, when $d(a,b)$ is too large resulting from the huge difference between a and b, w tends to be zero. In fact, w has been zero in our prototypical system when $d(a,b)$ is 10000. In summary, weight w belongs to $[0,1]$.

B EDGE WEIGHT COMPUTATION AMONG LOG SEQUENCES OF DIFFERENT DOMAIN NAMES

Weights of sequences of the same host and different domain names depend on the similarities of accessing modes and numbers of log entries. Specifically, we employ function 5 to separately calculate each $d(a,b)$ for the numbers of visit, upload, download and sum of them between two sequences. We then add these four results together as function 7 and feed it into function 6.

$$d(a,b)=d(a,b)_{visit}+d(a,b)_{upload}+d(a,b)_{download}+d(a,b)_{total} \tag{7}$$

We take $w3$ in Figure 5b as an example. As a result of the absence of upload and download operations in sequence of domain name3, minimums in function 5 are both zero, and hence $d(a,b)_{upload}$ and $d(a,b)_{download}$ are both 10000, which we have defined in Appendix A, thus producing the output that $w3$ is zero. Specially, $d(a,b)_{upload}$ is zero if neither domain name1 nor domain name3

has upload operations, not affecting $d(a, b)$. Access mode therefore can be taken into consideration through such functions.

C SETTING PROPORTION OF SETS OF EDGE TYPE

With respect to the way how the proportion of sets (*ps*) is determined, we consider a user's behavior and his colleagues' during current period and the last one. First, we define a month or several months as a time interval and take the same number of months before this period as the last one. Second, the organizational units in an enterprise (e.g. functional_unit, department) and role (e.g. Salesman, ITAdmin) are used to define a colleague in log2vec.

As presented in Section 4.1.3, log2vec selects sets of {edge3, edge6}, {edge7, edge8} and {edge9, edge10} to adjust. The reference to adjust the proportion is changes of operation types in quantity. Specifically, with respect to {edge9, edge10}, we focus on change of network operation in quantity, corresponding to detecting penetration of malware from Internet and data breach through it, while {edge7, edge8} relies on that of logon operation with authentication protocol, corresponding to detecting *APT*. {edge3, edge6} is based on changes of device connect operation and file operation in quantity, and which one is taken into account depends on whether this operation type explicitly has an average of daily operations in this period. This design corresponds to detecting data breach through removable storage device and system sabotage (e.g. install, modify and delete software) on user's own host or others'.

We set the proportion 1:1:1:1:1 corresponding to sets of {edge1, edge4}, {edge2, edge5}, {edge3, edge6}, {edge7, edge8} and {edge9, edge10} separately. Moreover, we set the number of random walk, r. Each value of the proportion is multiplied by r and the result is the number of walks of the corresponding set.

We propose three policies to determine the proportion of sets of edge type (*ps*). The first one is to observe whether daily averages of connect/file, logon and network operation of the detected user change between two periods, namely current period and the last one. The result is 1:1:1:1:1 if none of them change, meaning no anomalies regarding these operation types explicitly occur. If one of them (e.g. network operation) changes, the corresponding value in the proportion is multiplied by two times of sum of other sets (e.g. 1:1:1:1:8) so that random walk extracts more corresponding context (e.g. context regarding network operation).

Second, if two of them change, e.g. connect operation and network operation, function 8 and function 9 are presented to compare changes of these two operation types of the user and his colleagues in quantity. In other word, change of his colleagues' operations is used as a reference to determine the change degrees of the user's operations.

$$r = \frac{max(a, b) - min(a, b)}{min(a, b)} \tag{8}$$

In the above function, for instance, a and b respectively indicate averages of connect operation of the user in this period and the last one and we can obtain $r^{user}_{connect}$. In fact, we can obtain four results ($r^{user}_{connect}, r^{colleague}_{connect}, r^{user}_{network}, r^{colleague}_{network}$) separately corresponding to connect operation/network operation of the user and his colleagues. This function obviously denotes the rate of change of the same operation type in quantity between two periods for the user or his colleagues. It does not yet involve relationship between the user and his colleagues that is solved in the function:

$$out = \frac{r_{max}}{r_{min}} \tag{9}$$

where out (e.g. $out_{connect}$) is a comparison of the same operation type between the user and his colleagues (e.g. $r^{user}_{connect}, r^{colleague}_{connect}$). Put another way, it compares the rates of change of the same operation type during two periods between the user and his colleagues.

Last, $out_{connect}$ and $out_{network}$ are compared and the higher one makes its corresponding set's value multiplied by two times of sum of other sets in the proportion (e.g. 1:1:8:1:1 if $out_{connect}$ is high). If identical, the proportion remains invariant. In particular, if operation type of colleagues is always zero, it is not taken into account because there is no useful information and it cannot be calculated in function 8. A constant (e.g. 0.1) is used to replace its result in function 8 that is attained by taking into account users' behavior in other departments.

If all the three operation types change in quantity, namely connect operation (or file operation), logon operation and network operation, log2vec calculates $out_{connect}$ (or out_{file}), out_{logon} and $out_{network}$. Afterward, they are compared and the highest one makes its corresponding set's value multiplied by two times of sum of other sets in the proportion.

Algorithm 1 Random walk with different sets of edge types

Input: The heterogeneous graph $G = (V, E, T)$, all sets of edge type S_e and sets of the corresponding numbers of walk R, walk length l, the number of neighbor nodes *neigh*

Output: Random walk traces

```
1:  Initialize walks to Empty
2:  for s ∈ S_e and r ∈ R do
3:      for i = 1 to r do
4:          for v ∈ V do
5:              initialize walk to [v]
6:              cur_v = v
7:              for j = 1 to l do
8:                  next_v = GetNeighbors(cur_v, G, s, neigh)
9:                  append next_v to walk
10:                 cur_v = next_v
11:             end for
12:             append walk to walks
13:         end for
14:     end for
15: end for
16: return walks
```

D PSEUDOCODE OF RANDOM WALK

We give the following pseudocode of random walk. For each set of edge type s, the algorithm generates its corresponding number of random walk r (line2) and each round contains l steps (line7). Notice that r and l are hyperparameters. When a walker resides at cur_v, he chooses next_v to traverse according to function 4, where S_n indicates s here (line8). After putting next_v into *walk* used to record trace in this round (line9), the walker has stands at next_v

(line10) and chooses the next node again. *walks*, sets of *walk* is the sequences of log entries/nodes fed into the later word2vec model.

E PSEUDOCODE OF CLUSTERING

Algorithm 2 depicts the pseudocode of clustering algorithm. $set1$ contains all logs (line2) and for each log entry, line4 ∼ line13 aim to search its neighbors. Specifically, $set2$ is used to record logs whose neighbors have been found and $set3$ includes logs that are close to each other. $set4$ is their difference set to preserve logs whose neighbors have not been searched. The algorithm searches log entries' neighbors until $set4$ is empty (line7 ∼ line13). After line15 each time, we obtain a cluster. Ultimately, the algorithm outputs clusters that meet the conditions in Section 5.1. This algorithm ensures that each log entry, whose similarity with the given one is larger than δ_1 (cosine distance, two log entries become similar when δ_1 is $0 \rightarrow 1$), must be grouped into the same cluster.

Algorithm 2 Clustering Algorithm

Input: Triple [log1, log2, similarity(sim)], all log entries V, a threshold δ_1
Output: Clusters

1: Initialize output to Empty
2: $set1 = V$
3: **for** $v \in set1$ **do**
4: $\quad set2 = [v]$
5: $\quad set3 = GetNeighbors(v, [log1, log2, sim], \delta_1)$
6: $\quad set4 = set3 \backslash set2$
7: $\quad$ **while** $set4 \neq \phi$ **do**
8: $\quad\quad$ **for** $t \in set4$ **do**
9: $\quad\quad\quad set3 = set3 \bigcup GetNeighbors(t, [log1, log2, sim], \delta_1)$
10: $\quad\quad$ **end for**
11: $\quad\quad set2 = set2 \bigcup set4$
12: $\quad\quad set4 = set3 \backslash set2$
13: $\quad$ **end while**
14: $\quad set1 = set1 \backslash set2$
15: $\quad output = output.append(set2)$
16: **end for**
17: **return** *output*

F DETECTION PERIOD OF CERT DATASET

Table 5 shows detection period of log2vec and training and test period of HMM, DeepLog for six suspicious users in CERT dataset. The period of log2vec is also the one of metapath2vec, node2vec, log2vec-Euclidean, log2vec-cosine and log2vec++.

G PARAMETERS OF BASELINE METHODS

- TIRESIAS. In our scenarios, we treat user's operations as security events, its original input. In CERT dataset, we divide user's operations into three sequences according to operation type, e.g. file, web and email operations, because these log entries possess enough information for model's learning. The input information is host id, timestamp, operation type and specific event information (e.g. filename, url). We train and test the sequences involving malicious log entries for each suspicious user. The training set is all log entries before the first malicious ones. The test set is the period during which the attack occurs. The training method is gradient descent. We tune the number of unrolling w to 5, 10 and 20, to get the best performance. In LANL dataset, TIRESIAS trains and tests its logon operations. The dataset is logs recording suspicious users' behavior in January. Specifically, the training and validation periods are the first twelve days in January, the proportion is 8:2. The test dataset is from 13th January to 31st January. The input information is timestamp, event id, source user, destination user, source computer, destination computer, authentication type, logon type and authentication orientation. We set the number of unrolling w to 20. For these two datasets, we respectively set the training batch size to 128 and 512, the number of memory array k to 4, and the number of hidden LSTM Memory Array units to 128. The output is a probability distribution over the n candidate events. If the next real value is in the first g predicted events (sorted by probability), TIRESIAS views this operation as normal. The predicted label is then compared with the real one. We adjust g to compute AUCs and their mean AUC.

Table 5: Detection period of log2vec, DeepLog and HMM for six suspicious users in CERT dataset and the corresponding numbers of log entries, malicious log entries and days

User Id	Log2vec	DeepLog/HMM	
	Detection Period (#log:#malicious)	Test Period (#day)	Training Period(#day)
ACM2278	2010.7-8 (7782:22)	2010.7-8 (44)	2010.6 (22)
CMP2946	2011.2-3(8584:242)	2011.2-3 (43)	2011.1 (21)
HIS1706	2010.7-8(6856:8)	2010.7-8 (43)	2010.6 (22)
PLJ1771	2010.7-8(4161:18)	2010.7-8 (31)	2010.6 (22)
CDE1846	2011.2.21-4.25 (4879:134)	2011.2.21-4.25(45)	2011.1.21-2.20(21)
MBG3183	2010.9-10(4995:4)	2010.9-10 (42)	2010.8 (22)

- DeepLog. We utilize DeepLog's log key anomaly detection model. In CERT dataset, we analyze user's behavior. Therefore, we set operation type as log key, e.g. logon, device connect and upload. For each suspicious user, the input is recent log keys, and the output is a probability distribution over the n log keys. The training and test set are shown in Table 5 in Appendix F. The training method is gradient descent. The window size h is 10, the number of layers L is 2 and the number of memory units in one LSTM block α is 64. If the real value is in the first g predicted log key candidates, DeepLog treats this operation as normal. The predicted label is compared with the real one. We adjust g to compute AUC.
- Hidden markov model. We transform daily log entries into a sequence. The two feature sets are both used: the simple set (7 features), markov-s and the comprehensive one (16 features), markov-c. The training period and test one are shown in Table 5 in Appendix F. We set $\epsilon = 0.01$ and $\lambda = 0.05$. This method outputs probability of daily log sequence. We set a threshold. If the sequence probability is below the threshold, we classify it as anomalous. For each malicious sequence, we view all logs involved in it as malicious. Through tuning this threshold, we compute users' mean AUC.

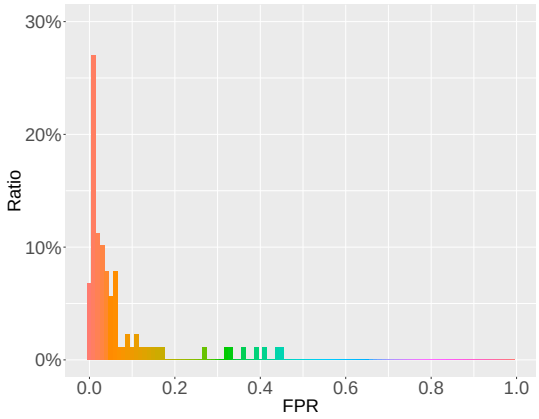


Figure 8: Distribution of false positive rates on 90 benign users in LANL dataset.

- Deep learning model. We use Deep Neural Network (DNN) and Long Short-Term Memory (LSTM) Recurrent Neural Network. For DNN and LSTM, we separately fix the number of hidden layers to 3 and 3, the hidden layer dimension to 20 and 3, the batch size to 256 and 21. The training method is Adam algorithm. We use the first 418 days as training dataset and day 419~516 as testing dataset, as the work does [57]. The method outputs anomaly score for each daily sequence. We set a threshold. If the anomaly score is above the threshold, we classify it as anomalous. For each malicious sequence, we view all log entries involved in it as malicious. We tune this threshold to compute AUC.
- STREAMSPOT. This method detects malicious information flows [31]. It defines an information flow (graph) as a unit of functionality (e.g.checking email). In CERT dataset, one log entry is nearly corresponding to a graph in STREAMSPOT. To compare this method, we define user's operations of the same type during an interval (one hour) as a graph. STREAMSPOT scores each information flow graph. We set a threshold. If the score is greater than the threshold, the corresponding graph is classified as malicious, and all log entries involved in it are viewed as malicious. We tune this threshold to compute six users' mean AUC.
- Metapath2vec. We use the same constructing graph and detection algorithm as log2vec's because metapath2vec is a method of graph embedding. Metapath2vec demands the specific metapath. We mainly define four general metapaths, {edge1, edge4}, {edge2, edge5}, {edge3, edge6} and {edge9, edge10}. For instance, only if the next edge belongs to edge1 or edge4, it can be considered to traverse in the round of {edge1, edge4}. The proportion of them is 1:1:1:1 because of no comparison of user's and his colleagues' behavior changes in this model. We set $r = 23$, $l = 60$ to make them close to the parameters in log2vec.
- Node2vec. This graph embedding also uses the same graph construction and detection algorithm as log2vec's. This model just processes homogeneous graph so we view the eight edge types as the same one and sum all their weights between every two nodes. Due to traversing more neighbor nodes in our task, we set $p = 4$ and $q = 0.5$. The number of neighbor nodes ($neigh$) in log2vec is designed for edge type of inter group (e.g. edge5, edge10). When a walker encounters the edge of inter group in log2vec, he also has a edge of intra group (e.g. edge1, edge3) to choose. Therefore we set the number of neighbor nodes in this model as the number in log2vec +1. We set $r = 90$ and $l = 60$.
- Log2vec-Euclidean. It shares the same graph construction and graph embedding as log2vec's. It uses k-means with Euclidean distance to detect anomalies. For each user, it utilizes K-means++ initialization [1] and performs k-means 3 times from 70-150 clusters. We determine the optimal k according to the maximum average silhouette score [43]. Log2vec-Euclidean produces clusters. We set a threshold. If the number of log entries involved in a cluster is smaller than the threshold, we will view this cluster as malicious. For each malicious cluster, we treat all logs involved in it as malicious. Through tuning this threshold, we compute six users' mean AUC in CERT dataset.
- Log2vec-cosine. It uses k-means with cosine distance. For each user, it performs k-means 10 times from 70-150 clusters. The remaining procedure is the same as log2vec-Euclidean.
- Log2vec++. Log2vec++'s $neigh$++ and δ_1++ are flexibly set as shown in Table 6, while log2vec's $neigh$ is 1 and δ_1 is set according to Section 5.3.
- Ensemble method. We utilize network flow logs of the first day in LANL dataset. We set both p_c and p_{cs} to 0.2. p_b/p_m is 2. Other parameters are set to the same ones in the work of Atul Bohara et al. [4]. This method produces anomalous hosts. We tune the threshold θ_avg to compute AUC in the host-level granularity, as the work does [4].

H PARAMETERS SETTING

Table 6 depicts log2vec++'s and log2vec's AUCs and three parameters for each user in CERT dataset.

Table 6: Log2vec++'s and log2vec's AUCs and three parameters of them for each user in CERT dataset. Log2vec++ utilizes $neigh$++ and δ_1++. Log2vec employs δ_1 and $neigh$ is 1. They share ps according to policies in Appendix C.

User Id	log2vec++	log2vec	$neigh$++	δ_1++	δ_1	ps
ACM2278	1	0.99	1	0.67	0.7	1:1:6:1
CMP2946	0.83	0.77	9	0.61	0.65	1:1:1:6
HIS1706	0.97	0.87	1	0.63	0.66	1:1:1:1
PLJ1771	0.83	0.63	2	0.57	0.69	1:1:6:1
CDE1846	0.92	0.92	1	0.66	0.66	1:1:1:1
MBG3183	1	0.99	4	0.62	0.64	1:1:1:6

I FALSE ALARMS OF BENIGN USERS

In LANL dataset, 90 benign users are examined by log2vec. In Figure 8, the mean and median of FPRs are 0.08 and 0.03 respectively. The maximum of FPR is 0.45 and most of FPRs are less than 0.13.

In CERT dataset, log2vec tests 12 benign users. The mean and median of FPRs are both 0.1. The maximum of FPR is 0.13.