Hindawi
Security and Communication Networks
Volume 2021, Article ID 6291276, 13 pages
https://doi.org/10.1155/2021/6291276

Research Article

Using Graph Representation in Host-Based Intrusion Detection

Zhichao Hu, Likun Liu, Haining Yu, and Xiangzhan Yu

School of Cyberspace Science, Harbin Institute of Technology, Harbin, China

Correspondence should be addressed to Xiangzhan Yu; yxz@hit.edu.cn

Received 29 September 2021; Accepted 19 November 2021; Published 7 December 2021

Academic Editor: Gu Zhaoquan

Cybersecurity has become an important part of our daily lives. As an important part, there are many researches on intrusion detection based on host system call in recent years. Compared to sentences, a sequence of system calls has unique characteristics. It contains implicit pattern relationships that are less sensitive to the order of occurrence and that have less impact on the classification results when the frequency of system calls varies slightly. There are also various properties such as resource consumption, execution time, predefined rules, and empirical weights of system calls. Commonly used word embedding methods, such as Bow, TI-IDF, N-Gram, and Word2Vec, do not fully exploit such relationships in sequences as well as conveniently support attribute expansion. To solve these problems, we introduce Graph Representation based Intrusion Detection (GRID), an intrusion detection framework based on graph representation learning. It captures the potential relationships between system calls to learn better features, and it is applicable to a wide range of back-end classifiers. GRID utilizes a new sequence embedding method Graph Random State Embedding (GRSE) that uses graph structures to model a finite number of sequence items and represent the structural association relationships between them. A more efficient representation of sequence embeddings is generated by random walks, word embeddings, and graph pooling. Moreover, it can be easily extended to sequences with attributes. Our experimental results on the AFDA-LD dataset show that GRID has an average improvement of 2% using the GRSE embedding method comparing to others.

1. Introduction

As the Internet continues to grow, cybersecurity, which protects us from cybercrime and threatening activities, has become an important part of our daily lives [1, 2]. Whether it is a computing device for personal use, an embedded device in the Internet of Things, or a server in a large network, the security principles are the same. Host-based intrusion detection systems (HIDS) are one of the most effective means of defeating attackers who bypass the network perimeter. HIDS are a subset of intrusion detection systems (IDS) that monitor activity on individual hosts [2, 3]. In contrast, Network IDS monitors communications between hosts and attempts to detect the presence of malicious activity in that network traffic [4]. Over the past few decades, a great deal of research has been spent to improve the performance of HIDSs.

Compared to NIDS, HIDS has the advantage of granularity and the ability to detect internal attacks [5]. System call based HIDS refers to the analysis of collected Linux system call traces, and this approach is effective on ordinary hosts. However, HIDS has recently suffered from big data challenges due to the rapid growth of computer technology and data centers. Linux syscall traces generated by data centers are a kind of big data, which are massive and complex. As such, they pose new challenges to traditional data processing methods [6].

Machine learning and data mining algorithms have been widely applied to intrusion detection. Researchers and engineers have made efforts to improve the algorithms in order to cope with increasingly sophisticated intrusion methods as well as massive amounts of data [7, 8]. For these methods, both traditional machine learning methods and deep learning methods are inseparable from feature engineering [1], although most deep learning methods, such as CNN based methods, will provide automatic feature engineering [9]. In general, good feature engineering can reduce the training cost of subsequent tasks and get better results in prediction [10].

In system call based intrusion detection systems, we are dealing with a large number of sequences consisting of system calls [6, 8]. The tasks can be classified into misuse detection (with labels) and anomaly detection (without labels) depending on whether the type of attack is known or not [11]. Sequences composed of system calls are essentially sequences of words, i.e., sentences. However, system call sequences have some unique features compared to sentences. For example, there is an implicit pattern relationship between system calls, and the order change between patterns has less impact. The number of occurrences of system calls often has little impact on the classification results. Commonly used word embedding methods, such as Bow, TI-IDF, N-Gram, and Word2Vec, do not fully exploit such relationships in sequences. System calls also have various properties, such as resource consumption, execution time, predefined rules, and empirical weights of system calls, and none of these methods can simply support such extensions [12–15].

In this work, we propose a new sequence embedding method called Graph Random State Embedding (GRSE), which overcomes these limitations. Our contributions can be summarized as follows:

(i) We introduce Graph Representation based Intrusion Detection (GRID), an intrusion detection framework using graph representation. It can capture potential relationships between system calls to learn better features, and it can also use multiple classifiers.

(ii) We propose GRSE, a new sequence embedding method that uses graph structures to model finite sequence terms, and represent structural association relationships between them. A more efficient representation of sequence embedding is generated by random walk, word embedding, and graph pooling. Moreover, it can be easily extended to sequences with attributes.

The rest of the paper is organized as follows. Section 2 describes the most relevant related work in this area. Section 3 provides an overview of the problem statement and describes the proposed approach. The experiment with results is described in Section 4. Section 5 provides the conclusion and future work.

2. Related Works

2.1. Host-Based Intrusion Detection. There are many researches on intrusion detection based on host system call in recent years. Forrest et al. [16] discussed the application and results of system call monitoring and data modeling in anomaly intrusion detection. This work was formed ten years ago, but since then, many new technologies and applications have been produced. Ou et al. [17] and Liu et al. [4] explored the combined use of misuse detection and anomaly detection techniques and effectively improved the efficiency and accuracy of intrusion detection. Datasets have always been an extremely important component in machine learning, Creech and Hu [3] published datasets ADFA-LD and ADFA-WD for host detection, which are widely used. Ahmed et al. [7] wrote a survey on network anomaly detection technology and data set problems. Sarraf and Swetha [8] make use of ADFA intrusion dataset to learn long-term sequences of system-call executed during an attack on a Linux based web server. The model can effectively predict and classify sequences of system-calls most likely to occur during a known or unknown (zero-day) attacks. Zhao et al. [18] classify multiple classes of webshell based on the implementation of webshell and then propose a heuristic detection method based on fuzzy matching and recurrent neural network. Hammad et al. [19] focus on feature selection and develop an optimal subset of features with Correlation-based Feature Selection (CFS). These researches give inspiration for subsequent research in terms of feature engineering and representation learning.

2.2. Graph Embedding. Graphs are common data structures to represent information with connections. If we want to make predictions on those graphs, we need a way to transform them into d-dimensional vectors of real numbers. So, we use graph embeddings, a low dimension representation that helps generalize better the input data.

Graph embedding is part of representation learning. The main purpose is to represent the nodes or graphs in a graph into a low-dimensional, real-valued, dense vector form, so that the obtained vectors can do further inference to better achieve downstream tasks. Graph embedding includes vertex embedding/graph embedding, and the main methods of embedding are matrix decomposition, random walk, and deep learning [20].

(1) Matrix decomposition: the matrices associated with graphs include adjacency matrix, degree matrix, Laplace matrix, and node transfer probability matrix. Usually, these are sparse matrices, so a matrix decomposition based approach can yield low-dimensional vectors [21].

(2) Random walk: the embedding can be done by randomly wandering the graph to get some sequences, treating the sequences as sentences, and then using the word vector approach. Among these methods, (1) consider the network structures: DeepWalk, GraRep, struct2vec, LINE, node2vec, and GraphSAGE [22, 23]. where DeepWalk [24] bridges the gap between network embedding and word embedding by treating nodes as words and generating short random walks as sentences. Then, neural language models such as Skip-gram can be applied to these random walks to obtain network embeddings. The advantage is that, firstly, it can generate random walks on demand. node2vec defines a bias random walk policy generation sequence based on DeepWalk, taking into account both BFS and DFS walks, still trained with skip gram. (2) Consider the structure and other information of Trans-Net, CENE, and CANE [25].

(3) Deep learning: typical methods in this category are GCN, SDNE, and Graph2Vec [26]. In addition,

GraphGAN and ANE introduced the idea of GAN into the graph domain and also achieved good results [27].

When embedding the whole graph instead of nodes, a common method is graph pooling. Based on the node embedding results, more abstract and higher-level features are continuously generated to finally complete the embedding of the whole graph. This kind of pooling on graph structure data layer by layer is also called hierarchical pooling [28]. Currently, hierarchical pooling is divided into two major categories: 1. graph collapse shaped pooling represented by DiffPool [29], and 2. TopK pooling represented by SAGPool [30].

(1) Graph collapse pooling refers to pooling where multiple nodes are combined into a single cluster, and this cluster is treated as a separate node in the next layer, and the eigenvalue of the new node is often the weighted sum of the nodes contained in that cluster. With each graph collapse pooling, the number of nodes of the graph becomes smaller and smaller. In general, the last pooling layer causes the graph to collapse to just one node, i.e., the graph representation.

(2) TopK pooling means that, after each aggregation of node features, the nodes are given an importance score based on their feature values and adjacencies by some algorithm, after which the Top K important nodes are selected to be retained, while the remaining nodes are removed. Unlike graph collapse pooling, Top-K pooling is not possible to gradually reduce to only one node left. A common practice is to find the average value of all feature nodes and then the maximum value of all feature nodes for each layer of Top-K pooling, then concatenate these two one-dimensional vectors into a one-dimensional vector, sum the vectors obtained from all layers, and combine the representations of all layers to obtain the final graph vector.

3. GRID: Proposed Method

In this section, we describe the components of our proposed method GRID. As shown in Figure 1, GRID initially builds a full system call graph from input sequences (system calls). It then computes the embedding vectors of each nodes with Random State Walk and Node Embedding methods with GRSE denoted as GR. The vectors and their labels are then piped to the classifier as train data. Further, for intrusion detection, the method will first extract subgraph for the input sequence and then convert the subgraph to vector. Finally, the classifier preforms prediction on the embedding vector.

Graph Random State Embedding (GRSE) is the embedding method used by GRID including graph transform, random state walk, subgraph extracting, and graph pooling. Below, we discuss the working of GRID and GRSE in detail.

3.1. Problem Statement. Denote S as the set of all system calls. Sampling a process from the moment t over a continuous period, the produced system calls are a sequence:

$$s^t = \left\{s_1^t, s_2^t, s_3^t, \ldots, s_n^t\right\}, s_k^t \in S, \forall 1 \le k \le n, \tag{1}$$

where n is the length of system call sequence. The number of system calls in different sampling periods is not fixed, so n is not a constant value. A system call may occur multiple times in the same sequence at intervals or consecutively.

Given a dataset $D = \{(s^1, y_1), (s^2, y_2), \ldots\}$, the task of intrusion detection is to learn a mapping $f: \mathcal{X} \longrightarrow \mathcal{Y}$, where $\mathcal{X}$ is the set of input system call sequences, and $\mathcal{Y}$ is the set of labels associated with each system call sequence.

Graph Random State Embedding (GRSE) is the proposed embedding method chosen by GRID to transform sequence to vector. It trains on the dataset $D' = \{s^1, s^2, \ldots\}$, where D' is the set of input system call sequences without labels. The trained embedding model then output embedding vectors for each input sequence.

3.2. Transform System Calls to Graph. By taking each item in the sequence as a node and the sequential relationship between items as an edge, we can transform the sequence of system calls into a directed graph. If two system calls are adjacent in the system call sequence, they are considered to have a directed edge between the corresponding nodes in the graph. As in equation (1), there is a directed edge between s_1^t and s_2^t, starting at s_1^t and ending at s_2^t.

Based on such an approach, a graph can be constructed based on the sampled system call data of a process to represent the relationship between system calls. The sequence of system calls generated by a process over a period of time can be considered as an access path on the graph, which is of length n.

Let the generated system call relationship graph be a directed graph $G = (V, E)$, where V is the set of nodes in the graph, and E is the set of edges in the graph; then, we have $V \equiv S$. Let A be the adjacency matrix of the graph, which is a $|V|$ order square matrix, and A_{ij} denotes the edge weights of nodes i to j. During a random traversal of the graph, the probability of moving from node i to node j is equivalent to the weight of that directed edge. An example of an adjacency matrix is shown as follows:

$$A = \begin{pmatrix} 7 & 1 & \cdots & 1 & 1 \\ 1 & 2 & \cdots & 2 & 15 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & 1 & \cdots & 0 & 0 \\ 5 & 0 & \cdots & 1 & 1 \end{pmatrix}. \tag{2}$$

Unlike general graphs where diagonal entries of the adjacency matrix are all zero, the diagonal elements of A may not be zero. This is because the same system call may occur consecutively in a sequence, and when it is greater than zero, it means that the call occurs consecutively, thus generating an edge that points to the node itself.

To generate the full graph, we should follow these steps:

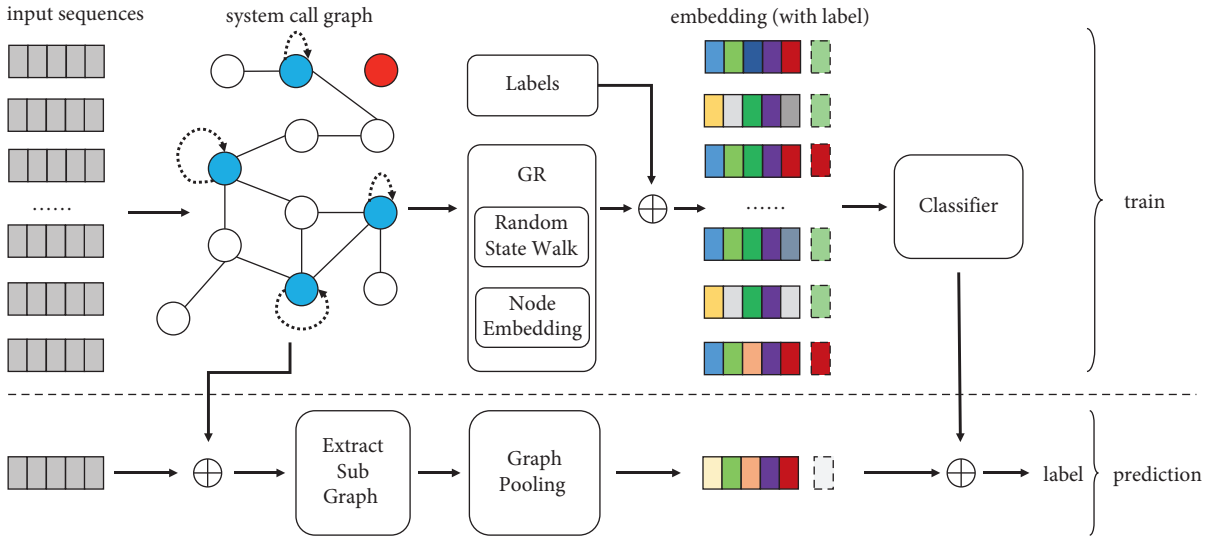

Figure 1: Schematic diagram of GRID.

(1) Create a graph with $|S|$ nodes, i.e., full graph, which uses all system calls as nodes.

(2) Beginning with the second item of each sequence, the current item forms a directed edge with the previous one. The node is allowed to generate an edge with itself.

(3) After processing all sequences, calculate the adjacency matrix of the graph according to the frequency of edge occurrences.

An example of generating a graph from a sequence is shown in Figure 2.

As can be seen from the figure, the right side is the full graph under construction. The red nodes have both in-degree and out-degree 0, the white nodes are normal nodes, and the blue nodes contain a self-edge side. The input sequence is $\{3, 3, 7, 6, 6\}$. The sequence term combinations $(3, 3)$ and $(6, 6)$ will update two edges pointing to their own nodes 3 and 6, and the other two sequence term combinations $(3, 7)$ and $(7, 6)$ update two directed edges.

Algorithm 1 is a pseudocode for generating graph from system call sequences.

Algorithm 2 describes how to calculate the weighted adjacency for a graph with input sequences; it also produces node index map V_A which maps node name to index in adjacency matrix. The function weight is used to transform frequency-based matrix to weighted matrix. Simply, we can use frequency as weight directly.

So far, we can get the full graph G, adjacency matrix A and node index map V_A.

3.3. GRSE with Random State Walk. Based on the constructed full graph, GRSE first generates the embedding vector for each node and then merges the node vectors to generate the embedding representation of the graph. The flow diagram of graph representation is shown in Figure 3.

GRSE generates a vectorized representation of graph nodes based on the Word2Vec method and works as follows.

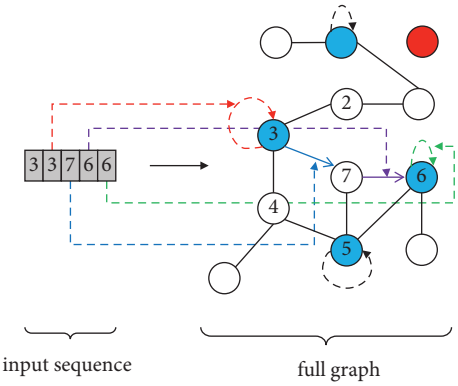

Figure 2: Example: transform system calls to graph.

(1) First, a random walk is performed in the graph network to generate graph node paths, emulating the process of text generation (in the scenario of this paper, i.e., the process executing a system call) to obtain multiple graph node access sequences.

(2) Then, based on the random walk sequence generated in the previous step, GRSE utilizes the Word2Vec model to learn the vector representation of the nodes to obtain the embedding vector of all nodes.

The system call graph has the following characteristics:

(i) There are nodes in the graph, which have edges towards themselves and therefore need to include the node itself as a candidate when accessing subsequent nodes.

(ii) When accessing other nodes from a node, the selection weight of subsequent nodes is unequal.

(iii) There are nodes in the graph that have an out-degree of 0. If the current node is of this type, then it needs special consideration when selecting subsequent nodes.

Require: System call set S, system call sequences D'
Ensure: graph $G(V, E)$
(1) Initialization: Create an empty graph $G = (V, E)$
(2) **for** each $s \in S$ **do**:
(3) $V \leftarrow s$
(4) **end for**
(5) **for** each $s^t \in D'$ **do**:
(6) **for** $i = 1$ to $|s^t| - 1$ **do**:
(7) **if** $s^t_{i-1} \neq s^t_i$ and $(s^t_{i-1}, s^t_i) \notin E$ **then**
(8) $E \leftarrow (s^t_{i-1}, s^t_i)$
(9) **end if**
(10) **end for**
(11) **end for**

Algorithm 1: GenerateGraph (S, D').

Require: graph $G = (V, E)$, system call sequences D', pattern weight function weight
Ensure: adjacency A, node index map V_A
(1) Initialization: Create an empty map V_A, a zero matrix $A^{|V|\times|V|}$
(2) index $\leftarrow 0$
(3) **for** each $v \in V$ **do**:
(4) $V_A[v] = i$
(5) **end for**
(6) **for** each $s^t \in D'$ **do**:
(7) **for** $k = 1$ to $|s^t| - 1$ **do**:
(8) $i \leftarrow V_A[s^t_{k-1}]$
(9) $j \leftarrow V_A[s^t_k]$
(10) $A[i][j] \leftarrow A[i][j] + 1$
(11) **end for**
(12) **end for**
(13) $A \leftarrow$ weight (A)

Algorithm 2: CalcAdjacency (G, D', weight).

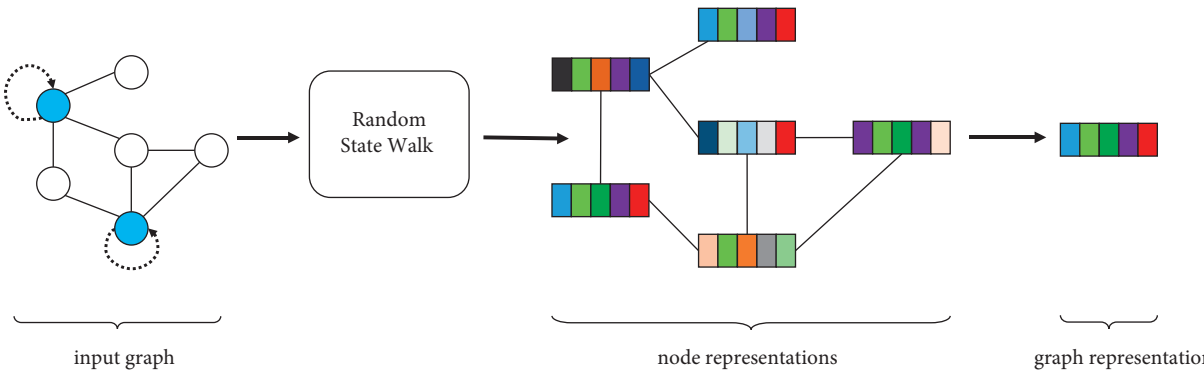

Figure 3: Flow diagram of graph representation.

To address these traits, this paper designs the RandomStateWalk sequence generation method. The method is based on the node transfer probability for random walk. The node transfer probability matrix $P^{|V|\times|V|}$ is calculated from the adjacency matrix A of the graph by the method shown in

$$P[ij] = \frac{A[ij]}{\sum_{k=0}^{|V|} A[ik]}. \quad (3)$$

The algorithm is described as shown in Algorithm 3.

Algorithm 4 describes the random state walk process starting from a node.

Require: graph $G(V, E)$, walks per node γ, walk length t, node index map V_A, node transition probability matrix P
Ensure: graph walks $\mathscr{W}$
(1) $\mathscr{W} \leftarrow \varnothing$
(2) **for** $v \in V$ **do**
(3) **for** $i = 1$ to γ **do**:
(4) $\mathscr{W} \leftarrow \mathscr{W} \cup$ WalkForNode (G, v, t, V_A, P)
(5) **end for**
(6) **end for**

Algorithm 3: RandomStateWalk (G, γ, t, V_A, P).

Require: graph $G(V, E)$, start node v, walk length t, node index map V_A, node transition probability matrix P
Ensure: walk path w_v from node v
(1) $w_v \leftarrow \varnothing$
(2) $v_s \leftarrow v$
(3) **while** $|w_v| < t$ **do**:
(4) $w_v \leftarrow$ app end (w_v, v_s)
(5) next $\leftarrow$ Get Next (v_s)
(6) **if** |next| == 0 **then**
(7) $v_s \leftarrow$ Random Choose Node (V)
(8) **else**
(9) $v_s \leftarrow$ Probability Choose Nod (next, P, V_A)
(10) **end if**
(11) **end while**

Algorithm 4: WalkForNode (G, v, t, V_A, P).

Specially, if the subsequent node of current is not found when walking, i.e., the current node has an out-degree of 0, the Random Choose Node function reselects a random one from all nodes (including the current node) as the next node to continue the sequence generation operation. When the set of subsequent nodes of the node is not empty, the Probability Choose Node function randomly selects one as the subsequent node according to the node transfer probability P.

An example of a random state walk is shown in Figure 4. In this case, the blue nodes (node 3, node 5) are nodes that can access themselves, the red nodes (node 1, node 2) are nodes with in-degree 0, and the arrows pointing to the edges indicate the directedness of the edges. Each node is chosen as the starting point to walk multiple times, and the length of each walking path is set to 5.

After the random state walk is completed, a set of paths with length 5 will be obtained. From the figure, it can be seen that when walking to node 1 or node 2, there is no subsequent node to choose, path $\{1, 1, 6, 7, 5\}$ selects node 1 and node 6 as subsequent nodes at random sequentially, and path $\{5, 4, 3, 2, 5\}$ selects 5 as subsequent node.

By random walk of the graph, we obtain the set of graph walks $\mathscr{W}$. Then, training is performed based on Word2Vec; it outputs the embedding representation of all nodes. Let the embedding space be K-dimensional, and then the vector representation of all nodes in K-dimensional space can be denoted as

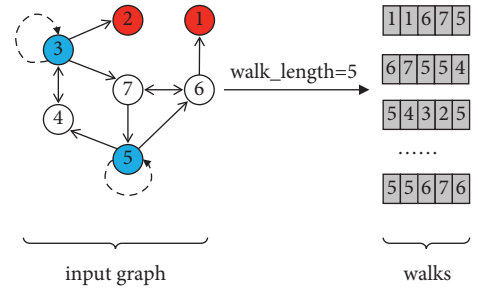

Figure 4: Example of random state walk (walk_length = 5).

$$R^K = \left\{R_0^K, R_1^K, \ldots, R_{n-1}^K\right\}, n = |V|. \quad (4)$$

3.4. Extract Subgraph. Since the input data are sequences rather than graphs, GRSE needs to transform the sequences into graphs. In the prediction stage, the full graph has been created, and the transformation operation can be completed by extracting the subgraph corresponding to this input sequence from the full graph.

For a given sequence s^t, let the subgraph corresponding to the sequence be G_{s^t}, and its adjacency matrix is $A_{s^t}^{|V|\times|V|}$. Algorithm 5 describes the details of extraction.

Compared to the full graph G, the subgraph retains all the node information, so $V_{s^t} \equiv V$. But the subgraph filters the edge information based on the input sequence and

Require: full graph $G = (V, E)$, system call sequence s^t, node index map V_A
Ensure: subgraph G_{s^t}, adjacency A_{s^t}
(1) $V_{s^t} \leftarrow V$
(2) $E_{s^t} \leftarrow \varnothing$
(3) Create a zero matrix $A^{|V|\times|V|}$
(4) **for** $k = 1$ to $|s^t| - 1$ **do**:
(5) **if** $s^t_{k-1} \neq s^t_k$ and $(s^t_{k-1}, s^t_k) \notin E$ **then**
(6) $E \leftarrow (s^t_{k-1}, s^t_k)$
(7) **end if**
(8) **end for**
(9) $G_{s^t} \leftarrow (V_{s^t}, E_{s^t})$
(10) $D'_{s^t} \leftarrow \{s^t\}$
(11) $A_{s^t} \leftarrow$ Calc Adjacency $(G_{s^t}, D'_{s^t}, \text{weight})$

Algorithm 5: ExtractSubGraph (G, s^t, V_A).

reconstructs the adjacency matrix. The adjacency matrix of the subgraph is of the same order as A and shares the node index map V_A.

3.5. Graph Embedding with Pooling. Graph embedding takes the set of node vectors R^K and the adjacency matrix A of the graph as input to generate a vector g that represents the entire graph structure data. A simple way is to aggregate all nodes at once to generate a graph vector, but this approach does not preserve the structural information of the graph well, so the classification accuracy is not high. A more common approach is to use a pooling operation like the CNN algorithm, which continuously generates more abstract, higher-level features. This kind of pooling on the graph structure data layer by layer is also known as hierarchical pooling.

In this paper, we use a concise version of graph collapse pooling method, which belongs to hierarchical Pooling. A diagram of the graph pooling process is shown in Figure 5.

Multiple nodes are combined into a cluster during pooling, and this cluster is treated as a separate node in the next layer, and the feature value of the new node is a weighted sum of the nodes contained in this cluster. A pair of nodes with group color (e.g., green) will generate a new node with this group color in next layer (e.g., green), and we will keep the single node to the next layer directly for clustering. With each graph collapse pooling, the number of nodes in the graph becomes smaller and smaller. Eventually, the last pooling layer of the pooling algorithm collapses the graph to just one node, a one-dimensional vector.

Since the vector is Euclidean data, it can be fed into a traditional machine learning classifier or neural network algorithm for classification. In this way, the Graph embedding operation is completed, and g is obtained.

The weight vector $W^{|V|}$ required in the fusion calculation of the nodes within the cluster is calculated from the adjacency matrix A of the graph, as shown in

$$W[i] = \frac{\sum_{k=0}^{|V|} A[ik]}{\sum_{m=0}^{|V|}\sum_{k=0}^{|V|} A[mk]}. \quad (5)$$

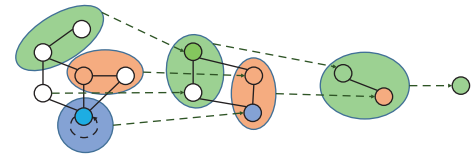

Figure 5: Graph pooling for subgraph.

The pooling process used in this paper is described as shown in Algorithm 6.

The function groups the nodes in the graph and returns the set of all groupings, with two nodes in each group. The following grouping strategy is used.

(1) The nodes in the graph are processed in order from highest to lowest weight, and a neighboring node with the highest weight is selected for combination and added to the result set.

(2) If the result set is empty, the two nodes with the highest weights are selected for combination, and the combined result is added to the result set.

By group fusion in this way, it is ensured that all nodes are eventually fused into one node. The vector of this node is the result of the graph embedding.

3.6. GRID: Graph Representation for Intrusion Detection. After we finished training the model, we obtained the system call graph G and the GRSE-based and embedding models. As a summarize, there are 3 key steps in prediction of GRID:

(1) Extract subgraph G_{s^t} from the full graph for input sequence (system calls);

(2) Convert subgraph G_{s^t} to embedding vector g with Algorithm 6;

(3) Use trained classifier to predict.

4. Experimental Results

4.1. Datasets. We use ADFA-LD dataset to perform train and prediction for intrusion detection. The ADFA-LD was created by researchers at the Australian Defence Force

Require: sub graph $G_{s^t} = (V_{s^t}, E_{s^t})$, node index map V_A, adjacency A_{s^t}
Ensure: vector g
(1) Initialization: create zero vector g which has K elements;
(2) $gi \leftarrow -1$
(3) **while** true **do**
(4) pairs = find_pair_nodes(G_{s^t})
(5) **for** $(i, j) \in$ pairs **do**:
(6) $R^K[i] \leftarrow W[i] * R^K[i] + W[j] * R^K[j]$
(7) $V_{s^t} \leftarrow V_{s^t} - \{V_j\}$
(8) update G_{s^t}, R^K, A_{s^t}
(9) $gi \leftarrow i$
(10) **end for**
(11) **if** |pairs| == 1 **then**
(12) break;
(13) **end if**
(14) **end while**
(15) $g \leftarrow R^K[gi]$

Algorithm 6: WeightedGraphPooling $(G_{s^t}, R^K, V_A, A_{s^t})$.

Academy as a public dataset that represents the structure and methodology of the modern attacks [3]. The dataset contains records created from the evaluation of system-call-based HIDS. Ubuntu Linux version 11.04 was used as the host operating system to build ADFA-LD.

It comprises three dissimilar data categories: Training, Validation, and Attack, each group of data containing raw system call traces. Each training dataset was gathered from the host for normal activities, with user behaviors ranging from web browsing to LATEX document preparation. ADFA-LD incorporates system call traces of different types of attacks. Table 1 shows the number of traces for each category of AFDA-LD.

Here are details of each attack class in the ADFA-LD dataset:

(1) Hydra-FTP: Password brute force (FTP by Hydra).
(2) Hydra-SSH: Password brute force (SSH Hydra).
(3) Adduser: Add new super user (Client-side poisoned executable).
(4) Java-Meterpreter: the uploads of Java executable Meterpreter payloads for the remote compromise of a target host.
(5) Meterpreter: the uploads Linux executable Meterpreter payloads for the remote compromise of a target host.
(6) Web shell: privilege escalation using C100 web shell

The syscall in the sample set is essentially a word sequence. From the word model perspective, we analyze whether there is a common pattern in the data of different categories of labels in the sample set, and we analyze their short-term patterns. The statistical word frequency histogram is shown in Figure 6.

From the figure, we can see that the pattern "168 168" has the highest frequency, "168 265" and "265 168" are close, and the frequency difference between different patterns of words is large, so the parameter walk_length of random state walk should be small.

The syscall sequence length reflects the total number of syscalls called from the start of the process to the final completion of the attack/attacked, and the probability density of trace length for different category class datasets is shown in Figure 7.

From the figure, we can see that the normal and attack sequences behave roughly the same in the trace length distribution, mainly in the [100, 750] interval, indicating that a sequence may contain a very large number of short-term patterns, and thus the number of random wanderings can be increased.

4.2. Training and Evaluation Setup. In this paper, we choose the following embedding methods for comparison.

(1) Bag-of-words: The bag-of-words model is one of the most popular representation methods for object categorization. The key idea is to quantize each extracted key point into one of visual words and then represent each image by a histogram of the visual words [12].
(2) TF-IDF: The goal of using tf-idf instead of the raw frequencies of occurrence of a token in a given document is to scale down the impact of tokens that occur very frequently in a given corpus and that are hence empirically less informative than features that occur in a small fraction of the training corpus.
(3) N-Gram: An n-gram model is a type of probabilistic language model for predicting the next item in such a sequence in the form of a $(n-1)$–order Markov model.
(4) Word2Vec: The semantic information of words is characterized by learning the text in terms of word vectors, i.e., by an embedding space that makes semantically similar words close together in that space. Two main models in Word2Vec model are Skip-Gram and CBOW.

Table 1: ADFA-LD dataset.

Dataset	Attack type	Traces amount	Label
Training	Training	833	Normal
Validation	Validation	4373	Normal
Attack	Hydra-FTP	162	Attack
Attack	Hydra-SSH	148	Attack
Attack	Adduser	91	Attack
Attack	Java-meterpreter	125	Attack
Attack	Meterpreter	75	Attack
Attack	Web shell	118	Attack

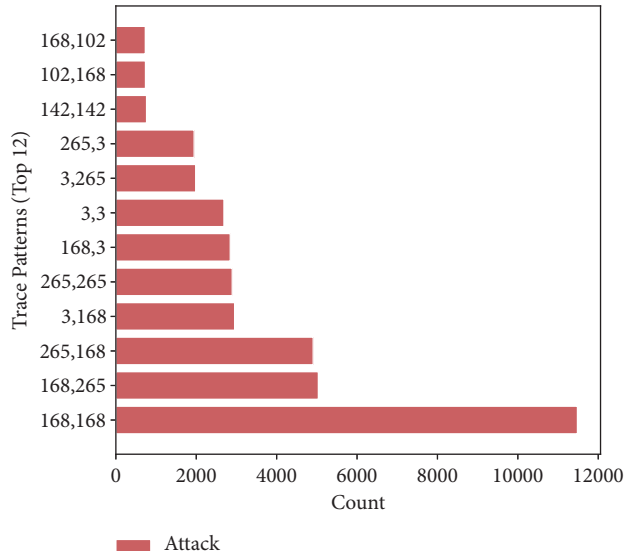

Figure 6: Histogram of short trace patterns.

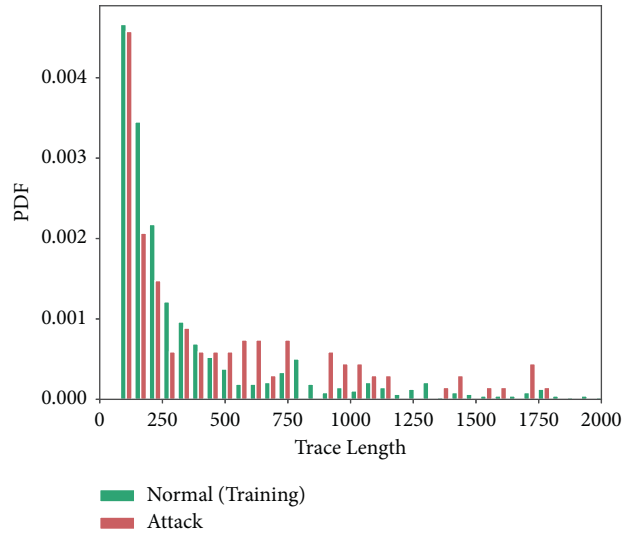

Figure 7: PDF of trace length.

(5) GRSE: It is a sequence embedding model based on graph representation learning. It models sequence terms by graph structure and mines sequence structure and numerical features using graph embedding.

For each embedding method, the back-end classifier and the training parameters are the same, listed as follows:

(i) There are two back-end classifiers, KNN and MLP, selected to perform classification.

(ii) The number of neighbors in KNN is set to 4, i.e., $n = 4$.

(iii) To prevent overfitting, the classification model is trained using k-fold cross-validation, setting $k = 10$.

4.3. Results and Analysis. For GRSE, the system call full graph $G = (V, E)$ generated from the training set has $|V| = 174$ and $|E| = 1583$.

We use AUC, RECALL, and ACCURACY as the evaluation metrics for classification. The comparison results are shown in Table 2 and Figure 8.

From the results, it can be seen that GRSE achieves better results than other embedding methods in all evaluation metrics with the highest AUC (0.9207 with KNN and 0.95555 with MLP) and the highest ACCURACY (0.9419 with KNN and 0.9213 with MLP) using a uniform backend classifier and training parameters. And, it achieved the highest RECALL (0.8983) tied with the bag-of-words model with KNN. The RECALL of GRSE with MLP is near to 1.0 and improves 3% than Bag-of-Words and N-Gram (best RECALL 1.0).

From the perspective of different classifiers, MLP achieves better AUC and RECALL than KNN despite the loss in ACCURACY. It can be seen that all embedding methods conform to such a variation trend, which is mainly due to the difference between classifiers. The GRID anomaly detection method includes two parts, sequence embedding and classification recognition. The classifier can be selected according to the usage scenario, and compared to the general sequence embedding methods, GRSE has a certain degree of improvement on the classification effect when different classifiers are used as the backend. This is due to the fact that the extracted full graph of system calls can better describe the operating state of the system when the set of sequences is limited. Random walk based on state transfer probability generates more valid sequences, which can increase the sampling of attack or normal system calls and accomplish data augmentation. On the other hand, the graph structure also describes the association relationship between different system calls in the sequence, so that the embedding approach that integrates the information of the calls themselves as well

Table 2: Classification result of different embedding methods.

Embedding methods	Classifier	AUC	Recall	Accuracy
Bag-of-words	KNN	0.9172	**0.8983**	0.9352
TF-IDF	KNN	0.8917	0.8644	0.9172
N-gram	KNN	0.9036	0.8729	0.9327
Word2Vec	KNN	0.9079	0.8814	0.9329
GRSE	KNN	**0.9207**	**0.8983**	**0.9419**
Bag-of-words	MLP	0.9483	**1.0**	0.8993
TF-IDF	MLP	0.9495	0.9915	0.9097
N-gram	MLP	0.9445	**1.0**	0.8919
Word2Vec	MLP	0.9455	0.9830	0.9100
GRSE	MLP	**0.9555**	0.9915	**0.9213**

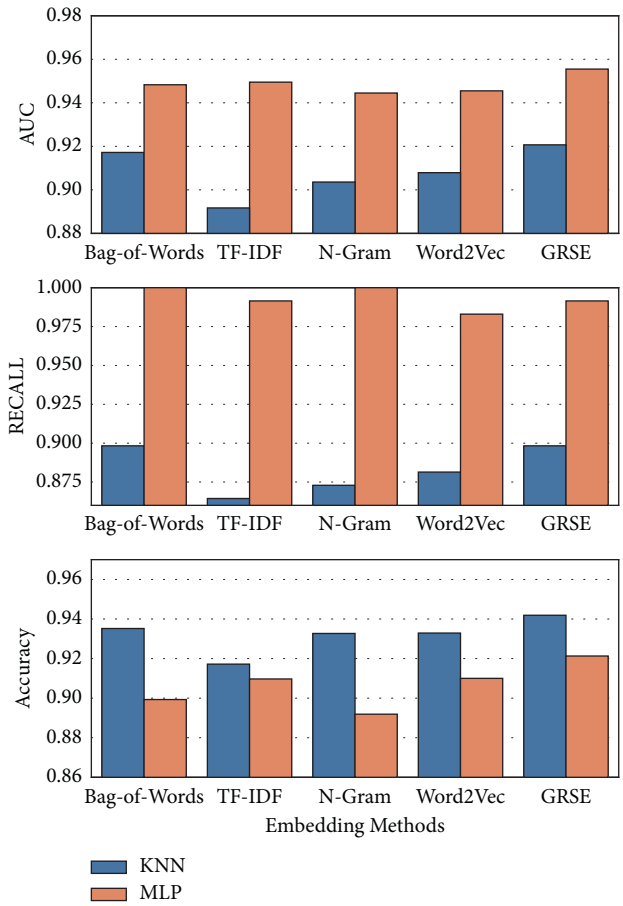

Figure 8: Comparison of different embedding methods.

as the relationship between the calls provides better embedding effect.

Word2Vec is used in GRSE for the learned embedding of word vectors. Comparing GRSE and Word2Vec, the GRSE improves the RECALL (1.69% with KNN and 0.75% with MLP), the AUC (1.28% with KNN and 1% with MLP), and ACCURACY (0.9% with KNN and 1.13% with MLP). These indicate that feature mining through graph structure is effective.

In GRSE, Walk Length is a key superparameter that has a correlation with the effective pattern length in the sequence. Take KNN classifier as example; the AUC of the classification results with the change of the walk length at different number of walks is shown in Figure 9 and ACCURACY is shown in Figure 10. As can be seen from the figure, shorter walks achieve higher performance than longer walks. This is due to the system call sequence characteristics, where short patterns predominate, and this is consistent with the previous analysis of the dataset.

The sequence embedding method GRSE proposed in this paper can be used not only in intrusion detection scenarios based on system call data; it is applicable to a kind of sequence embedding scenarios, which are characterized as follows.

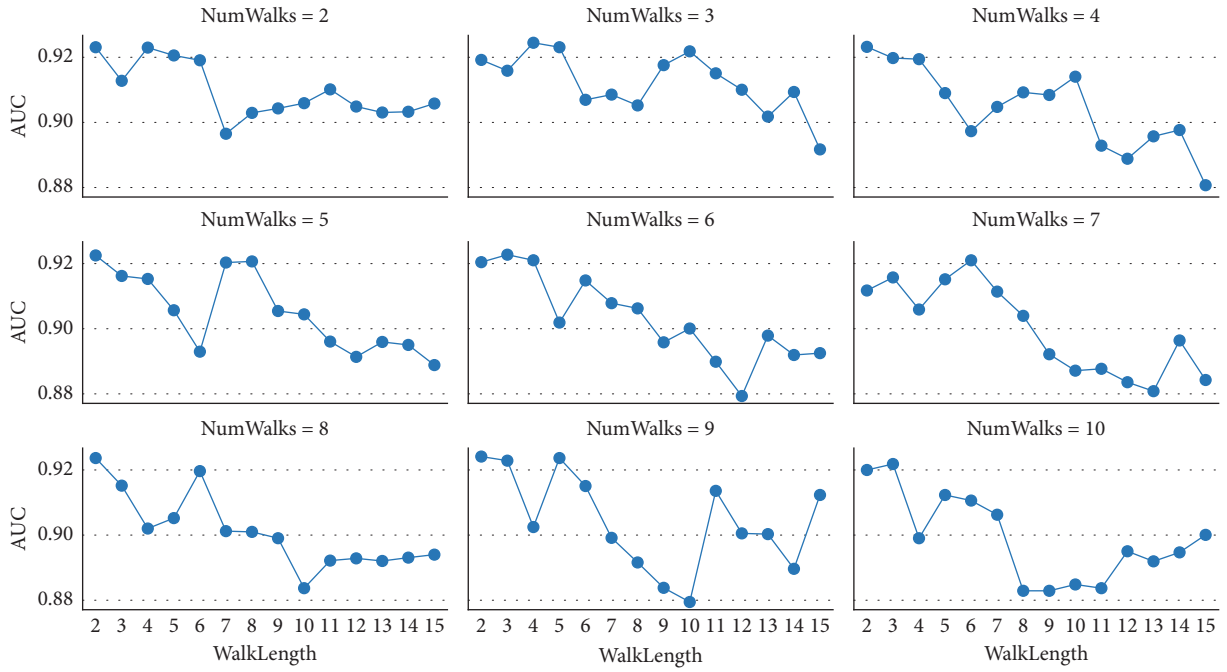

Figure 9: Param analysis of GRSE with different walk length and num walks (AUC).

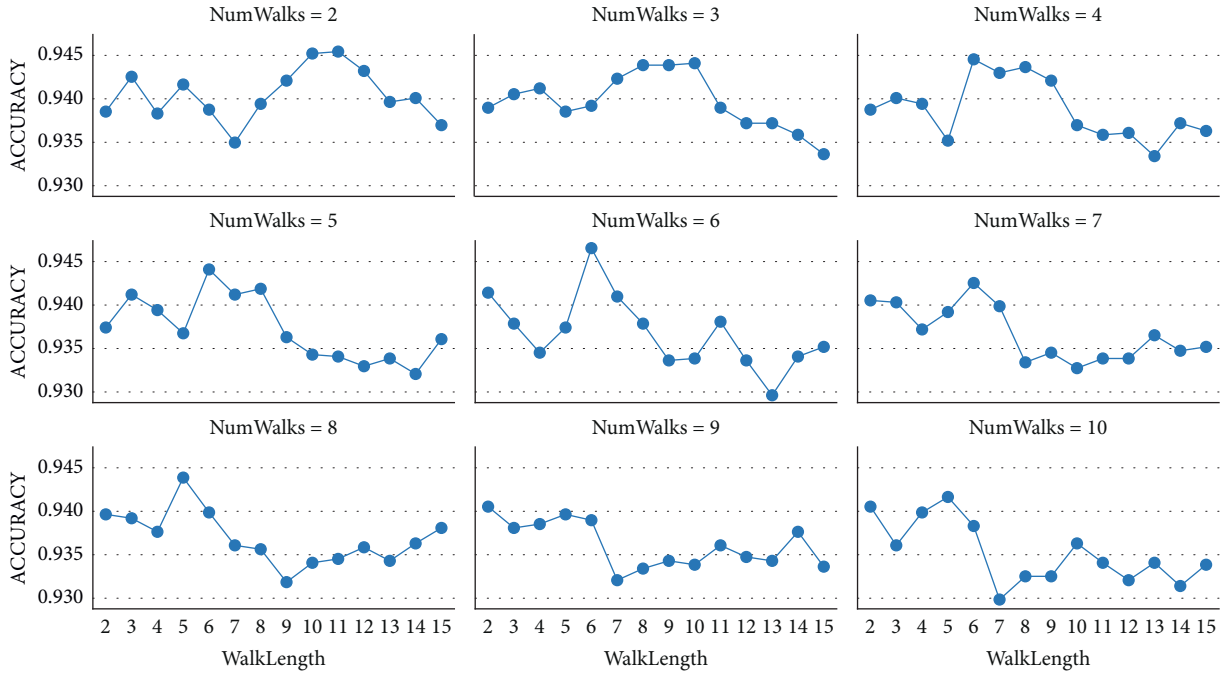

Figure 10: Param analysis of GRSE with different walk length and num walks (accuracy).

(i) The range of values that the sequence terms can take is limited, so that it will not form a superlarge scale graph, and the demand for computational resources is not high, which is conducive to model training as well as practical use online.

(ii) There are implicit associative relationships between sequence items, similar to phrases, and insensitive to the order between frequent patterns, such that sample features can be better generalized by random walks on the graph.

(iii) Each item in the sequence has a variety of potential attributes that can be used, such as resource consumption, execution time, predefined rules, and experience weights of system calls in the intrusion detection scenario of this paper. The general word embedding methods are not conducive to the

expansion of such attributes, and the adoption of attributed graph can be well extended without affecting the upstream and downstream tasks.

(iv) The length of subsequences with correlations fluctuates widely, and it is difficult to mine all combinations by sampling only. Meanwhile, the positive example samples have large uncertainty. The graph structure is suitable for describing such data and can replace neural networks such as TextCNN to a certain extent to complete the mining of the features of the association structure.

5. Conclusions

In this paper, we study the host-based intrusion detection problem and introduce GRID, an intrusion detection framework based on graph representation learning. It captures the potential relationships between system calls to learn better features, and it is applicable to a wide range of back-end classifiers. We also propose GRSE, a new sequence embedding method that uses graph structures to model a finite number of sequence items and represent the structural association relationships between them. A more efficient representation of sequence embeddings is generated by random walks, word embeddings, and graph pooling. Moreover, it can be easily extended to sequences with attributes. Through experiments on the AFDA-LD dataset, we demonstrate that GRID achieves better performance using the GRSE embedding method.

As a future work, it is necessary to generate graph and extract subgraphs for sequences with multiple attributes or predefined rules. Also, we will explore how to improve the performance of GRSE on datasets with larger full graph.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work is supported by the National Key R & D Program of China (No. 2018YFB1800702), the National Natural Science Foundation of China (No. 62172123), the National Natural Science Foundation of China (No. 61732022), and Natural Science Foundation of Heilongjiang Province of China (No. YQ2021F007).

References

[1] E. Rodríguez, B. Otero, N. Gutiérrez, and R. Canal, "A survey of deep learning techniques for cybersecurity in mobile networks," *IEEE Communications Surveys Tutorials*, vol. 23, no. 3, pp. 1920–1955, 2021.

[2] A. K. Kassem, S. Abo Arkoub, B. Daya, and P. Chauvet, "A survey of methods for the construction of an intrusion detection system," in *Artificial Intelligence and Applied Mathematics in Engineering Problems*, D. J. Hemanth and U. Kose, Eds., Springer International Publishing, Cham, pp. 211–225, 2020.

[3] G. Creech and J. Hu, "A semantic approach to host-based intrusion detection systems using contiguousand discontiguous system call patterns," *IEEE Transactions on Computers*, vol. 63, no. 4, pp. 807–819, 2013.

[4] J. Liu, K. Xiao, L. Luo, Y. Li, and L. Chen, "An intrusion detection system integrating network-level intrusion detection and host-level intrusion detection," in *Proceedings of the 2020 IEEE 20th International Conference on Software Quality, Reliability and Security (QRS)*, pp. 122–129, IEEE, Macau, China, 11 December 2020.

[5] A. Laszka, W. Abbas, S. S. Sastry, Y. Vorobeychik, and X. Koutsoukos, "Optimal thresholds for intrusion detection systems," in *Proceedings of the Symposium and Bootcamp on the Science of Security, ser. HotSos '16*, pp. 72–81, Association for Computing Machinery, New York, NY, USA, 19 April 2016.

[6] M. Liu, Z. Xue, X. Xu, C. Zhong, and J. Chen, "Host-based intrusion detection system with system calls," *ACM Computing Surveys*, vol. 51, no. 5, pp. 1–36, 2019.

[7] M. Ahmed, A. Naser Mahmood, and J. Hu, "A survey of network anomaly detection techniques," *Journal of Network and Computer Applications*, vol. 60, pp. 19–31, 2016, https://www.sciencedirect.com/science/article/pii/S1084804515002891.

[8] G. Sarraf and M. S. Swetha, "Intrusion prediction and detection with deep sequence modeling," in *Security in Computing and Communications*, S. M. Thampi, G. Martinez Perez, R. Ko, and D. B. Rawat, Eds., Springer Singapore, Singapore, pp. 11–25, 2020.

[9] Y. Li, Z. Hao, and H. Lei, "Survey of convolutional neural network," *Journal of Computer Applications*, vol. 36, no. 9, pp. 2508–2515, 2016.

[10] M. Usman, M. A. Jan, X. He, and J. Chen, "A survey on representation learning efforts in cybersecurity domain," *ACM Computing Surveys*, vol. 52, no. 6, pp. 1–28, Oct. 2020.

[11] M. Almansor and K. B. Gan, "Intrusion detection systems: principles and perspectives," *Journal of Multidisciplinary Engineering Science Studies*, vol. 4, no. 11, pp. 2458–2925, 2018.

[12] Y. Zhang, R. Jin, and Z.-H. Zhou, "Understanding bag-of-words model: a statistical framework," *International Journal of Machine Learning and Cybernetics*, vol. 1, no. 1-4, pp. 43–52, 2010.

[13] J. Ramos, "Using tf-idf to determine word relevance in document queries," *Proceedings of the first instructional conference on machine learning*, vol. 242, no. 1, pp. 29–48, 2003.

[14] W. Cheng, C. Greaves, and M. Warren, "From n-gram to skipgram to concgram," *International Journal of Corpus Linguistics*, vol. 11, no. 4, pp. 411–433, 2006.

[15] K. W. Church, "Word2vec," *Natural Language Engineering*, vol. 23, no. 1, pp. 155–162, 2017.

[16] S. Forrest, S. Hofmeyr, and A. Somayaji, "The evolution of system-call monitoring," in *Proceedings of the 2008 annual computer security applications conference (acsac)*, pp. 418–430, IEEE, Anaheim, CA, USA, 8 December 2008.

[17] Y.-j. Ou, Y. Lin, Y. Zhang, and Y.-j. Ou, "The design and implementation of host-based intrusion detection system," in

Proceedings of the 2010 Third International Symposium on Intelligent Information Technology and Security Informatics, ser. IITSI '10. USA, pp. 595–598, IEEE Computer Society, Jian, China, 2 April 2010.

[18] Z. Zhao, Q. Liu, T. Song, Z. Wang, and X. Wu, "Wsld: detecting unknown webshell using fuzzy matching and deep learning," in *Information and Communications Security*, J. Zhou, X. Luo, Q. Shen, and Z. Xu, Eds., Springer International Publishing, Cham, pp. 725–745, 2020.

[19] M. Hammad, W. El-medany, and Y. Ismail, "Intrusion detection system using feature selection with clustering and classification machine learning algorithms on the unsw-nb15 dataset," in *Proceedings of the 2020 International Conference on Innovation and Intelligence for Informatics, Computing and Technologies (3ICT)*, pp. 1–6, IEEE, Sakheer, Bahrain, 20 December 2020.

[20] Z. Zhang, P. Cui, and W. Zhu, "Deep learning on graphs: a survey," 2020, https://arxiv.org/abs/2103.00742.

[21] D. Hakkani-Tur and G. Riccardi, "A general algorithm for word graph matrix decomposition," in *Proceedings of the 2003 IEEE International Conference on Acoustics, Speech, and Signal Processing, 2003. Proceedings.(ICASSP'03)*, vol. 1, p. I, IEEE, Hong Kong, China, 6 April 2003.

[22] A. Grover and J. Leskovec, "node2vec: scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 855–864, ACM, San Francisco California USA, 13 August 2016.

[23] F. Fouss, A. Pirotte, J.-M. Renders, and M. Saerens, "Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation," *IEEE Transactions on Knowledge and Data Engineering*, vol. 19, no. 3, pp. 355–369, 2007.

[24] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: online learning of social representations," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 701–710, ACM, New York USA, 24 August 2014.

[25] C. Tu, H. Liu, Z. Liu, and M. Sun, "Cane: context-aware network embedding for relation modeling," *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, vol. 1, pp. 1722–1731, 2017.

[26] A. Narayanan, M. Chandramohan, R. Venkatesan, L. Chen, Y. Liu, and S. Jaiswal, "graph2vec: learning distributed representations of graphs," 2017.

[27] H. Wang, J. Wang, J. Wang et al., "Graphgan: graph representation learning with generative adversarial nets," *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, no. 1, 2018.

[28] J. Lee, I. Lee, and J. Kang, "Self-attention graph pooling," 2019, https://icml.cc/media/Slides/icml/2019/halla(11-11-00)-11-12-05-4517-self-attention_.pdf.

[29] R. Ying, J. You, C. Morris, X. Ren, W. L. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," 2019, https://arxiv.org/abs/1806.08804.

[30] J. Lee, I. Lee, and J. Kang, "Self-attention graph pooling," in *Proceedings of the International Conference on Machine Learning*, pp. 3734–3743, PMLR, Long Beach, California, USA, 9 June 2019.