Towards Network Anomaly Detection Using Graph Embedding

Qingsai Xiao[1,2], Jian Liu[1,2], Quiyun Wang[1], Zhengwei Jiang[1,2], Xuren Wang[1,3], and Yepeng Yao[1,2]⋆(✉)

[1] Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
{xiaoqingsai, liujian6, wangqiuyun, jiangzhengwei, yaoyepeng}@iie.ac.cn
[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
[3] College of Information Engineering, Capital Normal University, Beijing, China
wangxuren@cnu.edu.cn

Abstract. In the face of endless cyberattacks, many researchers have proposed machine learning-based network anomaly detection technologies. Traditional statistical features of network flows are manually extracted and rely heavily on expert knowledge, while classifiers based on statistical features have a high false-positive rate. The communications between different hosts forms graphs, which contain a large number of latent features. By combining statistical features with these latent features, we can train better machine learning classifiers. Therefore, we propose a novel network anomaly detection method that can use latent features in graphs and reduce the false positive rate of anomaly detection. We convert network traffic into first-order and second-order graph. The first-order graph learns the latent features from the perspective of a single host, and the second-order graph learns the latent features from a global perspective. This feature extraction process does not require manual participation or expert knowledge. We use these features to train machine learning algorithm classifiers for detecting network anomalies. We conducted experiments on two real-world datasets, and the results show that our approach allows for better learning of latent features and improved accuracy of anomaly detection. In addition, our method has the ability to detect unknown attacks.

Keywords: Network Anomaly Detection · Graph Embedding · Feature Engineering · Unknown Attack Discovery

1 Introduction

With the rapid development of the Internet and more network devices are connected to it, modern cyberattacks have emerged in recent years. Attackers aiming to exploit networked systems must take a number of relevant attack steps

⋆ Corresponding author: yaoyepeng@iie.ac.cn

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50423-6_12

in order to achieve their goals. In order to deal with these attack steps, significant attention has been given to identifying correlated anomalous network flow from the network traffic in recent academic research. In addition, for each of the attack steps that compose the entire attack scenario to be interconnected, looking at the network context of individual attack steps is helpful to understand the correlation between anomalous and benign streams and is important in a counterattack against network attacks.

Since modern network anomaly detection systems should not be highly dependent on human expert knowledge, they need to have the ability to autonomously adapt to the evolution of all network behaviors. Therefore, machine learning has become a very common method for detecting network anomalies, and feature engineering has become an important step in machine learning. However, the quality of the features directly determines the effectiveness of the machine learning algorithm. Researchers have proposed a number of ways to extract features from network traffic. Some feature extraction methods have also published as public tools such as CICFlowMeter[7]. Most of these features are statistical features. In that way, the existing feature extraction methods rely heavily on expert knowledge.

The communication between different hosts can be constructed into attributed graphs. These graphs contain huge number of latent features. For example, if the port usage of the two hosts is very similar, the distance of the two hosts in graphs should be close. These latent features cannot be extracted manually, but they are very helpful for machine learning-based algorithms.

In this paper, we propose a network anomaly detection method based on graph embedding. We convert the network traffic into first-order graph and second-order graph. The first-order graph learns the latent features from the perspective of a single host, and the second-order graph learns the latent features from a global perspective. This feature extraction process does not require manual participation or expert knowledge. We use these features to train machine learning algorithm classifiers for detecting network anomalies.

In general, our main contributions are as follows:

- We propose first-order and second-order graph of network traffic, and learn the low-dimensional vector representation of the nodes in the diagram.
- We design a network anomaly detection method based on graph embedding. By combining first-order and second-order low-dimensional vector representations of network traffic with several statistical features, it is possible to improve detection accuracy and detect unknown attacks.
- We built a prototype of the network anomaly detection framework and evaluate the detection accuracy and the ability to discover unknown attacks on real-world datasets.

The rest of the paper is organized as follows. We first introduce the threat model in Section 2. Section 3 is the graph embedding algorithm. Section 4 defines the proposed network anomaly detection framework. Experimental results are presented in Section 5. Finally, we discuss the related work in Section 6 and conclude in Section 7.

2 Threat Model

In order to better understand the method proposed in this paper, we will describe the threat model in this section. The attacker launches an attack on victims through the network. Here, we take remote control Trojan as an example. The attacker sends a control command to the Trojan through the C&C server through the phishing email of the victim, and accepts the stolen data. In many cases, the attacker will use encryption to communicate, but the victim's ability to know the content of the communication is limited. The victim needs to look for abnormal traffic in the network traffic and find attack information.

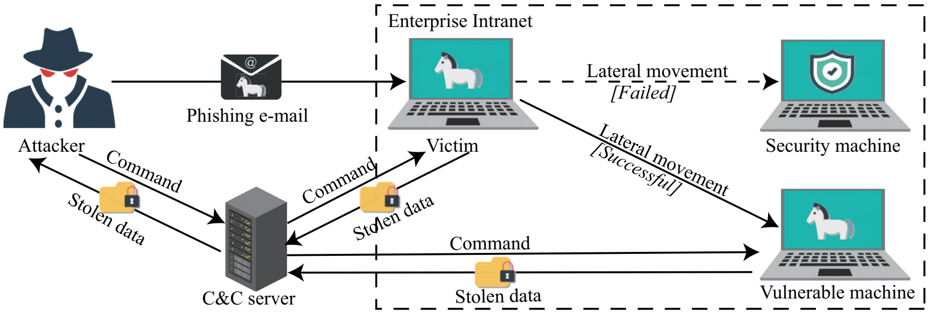


Fig. 1. A use case for the threat model

3 Graph Embedding Algorithm

In this section, we introduce the first-order graph and second-order graph of network traffic, then propose the graph embedding algorithm for these two graphs. At last, we also adopt two optimization methods to reduce the complexity of the proposed algorithm.

3.1 First-Order Graph Embedding

Definition 1. *(**First-Order Graph**) A bipartite graph consisting of IP addresses and ports (e.g. Fig. 2) is denoted as $G_1 = (V_{ip} \bigcup V_{port}, E)$, where V_{ip} is the set of IP addresses and V_{port} is the set of ports. E is the set of edges between IP addresses and ports. $\omega_{i,j}$ is the weight between v_{ip_i} and v_{port_j}. The weight can be the number of packets or the number of bits that flowing out from $PORT_j$ of IP_i. In the following experiments, we consider the weight as the number of packets that flowing out from $PORT_j$ of IP_i.*

The first-order graph learns the latent features from the perspective of a single host. The purpose of first-order graph embedding is that if two hosts have similar port usage distribution, their distance in the vector space should be close.

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50423-6_12

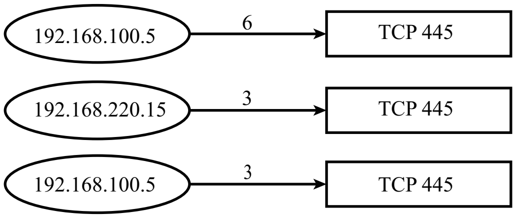


Fig. 2. Example of first-order graph

For $\forall\ v_{ip_i} \in V_{ip}, v_{port_j} \in V_{port}$, let $\overrightarrow{u_{ip_i}}$ and $\overrightarrow{u_{port_j}}$ are embedding vectors for v_{ip_i} and v_{port_j}. We define the following conditional probabilities:

$$p(v_{ip_i}|v_{port_j}) = \frac{\exp\left(\overrightarrow{u_{ip_i}}^T \cdot \overrightarrow{u_{port_j}}\right)}{\sum_{v_{ip_{i'}} \in V_{ip}} \exp\left(\overrightarrow{u_{ip_{i'}}}^T \cdot \overrightarrow{u_{port_j}}\right)} \tag{1}$$

$$p(v_{port_j}|v_{ip_i}) = \frac{\exp\left(\overrightarrow{u_{port_j}}^T \cdot \overrightarrow{u_{ip_i}}\right)}{\sum_{v_{port_{j'}} \in V_{port}} \exp\left(\overrightarrow{u_{port_{j'}}}^T \cdot \overrightarrow{u_{ip_i}}\right)} \tag{2}$$

According to Eq.1 and Eq. 2, we can get the conditional distribution $p(\cdot|v_{port_j})$ and $p(\cdot|v_{ip_i})$.

To understand the latent features of IP addresses and ports, we minimize the distance between conditional distribution and empirical distribution. Therefore, we get the following objective function:

$$\begin{aligned} O_1 = &\sum_j distance\left(\hat{p}(\cdot|v_{port_j}), p(\cdot|v_{port_j})\right) + \\ &\sum_i distance\left(\hat{p}(\cdot|v_{ip_i}), p(\cdot|v_{ip_i})\right) \end{aligned} \tag{3}$$

where $\hat{p}(v_{ip_i}|v_{port_j}) = \frac{w_{i,j}}{\sum_{i'} w_{i',j}}$, $distance(\cdot,\cdot)$ is the distance between two distributions(e.g. Bhattacharyya distance, Jeffries–Matusita distance, KL-divergence distance). In the following experiments, we use the KL-divergence distance.

We use stochastic gradient descent for optimizing Eq. 3, then we are able to get the first-order low-dimensional vector representation $\{\overrightarrow{u_i^{ip}}\}_{i=1,2,3,\ldots,|V_{ip}|}$ and $\{\overrightarrow{u_i^{port}}\}_{i=1,2,3,\ldots,|V_{port}|}$.

3.2 Second-Order Graph Embedding

Definition 2. *(**Second-Order Graph**) A hypergraph consisting of source IP addresses, source ports, destination IP addresses, and destination ports (e.g. Fig. 3) is denoted as $G_2 = (V_{sip} \bigcup V_{sport} \bigcup V_{dip} \bigcup V_{dport}, E)$, where V_{sip} is a set of all source IP addresses; V_{sport} is a set of all source ports; V_{dip} is a set of all destination IP addresses; V_{dport} is a set of all destination ports; E is a set* *of hyperedges consisting of four points: source IP address, source port, destination IP address and destination ports.* $\omega_{i,j,k,l} \in E$ *is the weight of hyperedge* $< v_{sip_i}, v_{sport_j}, v_{dip_k}, v_{dport_l} >$.

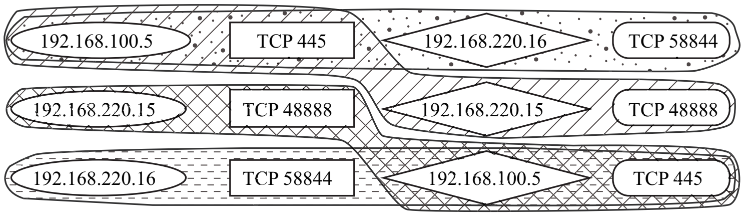


Fig. 3. Example of second-order graph

The second-order graph learns latent features from a global perspective.

For $\forall v \in V_{sip} \bigcup V_{sport} \bigcup V_{dip} \bigcup V_{dport}$, we define C is the context of v. For example, for a hyperedge $< v_i, v_j, v_k, v_l >$, (v_i, v_k, v_l) is the context of v_j. v_i, v_k and v_l are similar. Inspired by[6], we define the following conditional probabilities:

$$p(v|C) = \frac{\exp\left(Sim(v, C)\right)}{\sum_{v' \in V_C} \exp\left(Sim(v', C)\right)} \tag{4}$$

where V_C is a set of nodes which context is C. $Sim(v, C) = \frac{1}{|V_C|} \sum_{v' \in V_C} \overrightarrow{u_v}^T * u_{v'}$ is the similarity between v and C.

Similarity to Eq. 3, we minimize the distance between the conditional distribution and the empirical distribution. Therefore, we get the following objective function:

$$O_2 = \sum_{C \in \mathcal{P}} distance(\hat{p}(\cdot|C), p(\cdot|C)) \tag{5}$$

where $\mathcal{P}$ is a set of C, $\hat{p}(v|C) = \frac{\omega_{v,C}}{\sum_{v' \in V_C} \omega_{v',C}}$, $distance(\cdot, \cdot)$ is the distance between the two distributions. In the following experiments, we use the KL-divergence distance.

We use stochastic gradient descent for optimizing Eq. 5, then we could get the second-order low-dimensional vector representation $\{\overrightarrow{u_i^{sip}}\}_{i=1,2,\ldots,|V_{sip}|}$, $\{\overrightarrow{u_i^{sport}}\}_{i=1,2,\ldots,|V_{sport}|}$, $\{\overrightarrow{u_i^{dip}}\}_{i=1,2,\ldots,|V_{dip}|}$ and $\{\overrightarrow{u_i^{dport}}\}_{i=1,2,\ldots,|V_{dport}|}$.

3.3 Optimization

In order to speed up the calculation speed, we use the following two optimization methods.

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50423-6_12

Fast Sigmoid Algorithm: When optimizing objective function, $sigmoid(x)$ function needs to be calculated many times. In the computer, the $\exp(x)$ function is usually calculated by the series summation ($\exp(x) = \sum_{k=0}^{\infty} \frac{x^k}{k!} = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \cdots$). A large number of multiplications and divisions need to be calculated, which takes a lot of time. So we use a sigmoid table instead of calculating $sigmoid(x)$ function. Firstly, we initialize this sigmoid table by calculating $sigmoid(-\frac{table\_size}{2}), sigmoid(-\frac{table\_size}{2}+1), \cdots, sigmoid(\frac{table\_size}{2})$. When we need to calculate $sigmoid(x)$ in each iteration, we use the point closest to x in the sigmoid table as $sigmoid(x)$. The time complexity of calculating $sigmoid(x)$ is only $O(1)$, and the space complexity is $O(table\_size)$. The size of $table\_size$ is related to the precision of $sigmoid(x)$ function. In the following experiments, $table\_size$ takes 2000.

Alias Method for Sampling: The nodes need to be sampled during each iteration of the training. For nodes that appear frequently, we should have a greater probability of sampling. Let $\Omega = \{\omega_1, \omega_2, ..., \omega_{|V|}\}$ denotes the weight set of nodes, $\omega_{sum} = \sum_{i=0}^{|V|} \omega_i$ is the sum of all weights. A simple way is that the sampling probability of each node is $\{\frac{\omega_1}{\omega_{sum}}, \frac{\omega_2}{\omega_{sum}}, ..., \frac{\omega_{|V|}}{\omega_{sum}}\}$. There is a very naive sampling method that satisfies this sampling requirement: randomly select an integer r from $[1, \omega_{sum}]$, if $r \in [\sum_{i=1}^{j-1} \omega_i, \sum_{i=1}^{j} \omega_i)$, then select the j-th node. We can use prefix sum array and binary search algorithm to search for $\sum_{i=1}^{j-1} \omega_i \leq r < \sum_{i=1}^{j} \omega_i$. The time complexity of this method is $O(\log |V|)$. Another faster method is alias table method[8], which is also the method we used in our experiments. This method generates two auxiliary tables based on Ω at first. Then we can randomly select an integer of $[1, |V|]$ and a decimal of $[0, 1)$ to select a node. This method has a time complexity of $O(1)$ per sample, which saves a lot of time when the number of iterations is large.

4 Network Anomaly Detection Framework

Our framework is composed of five primary modules: network probe, embedding, training, detection and database as shown in Fig. 4.

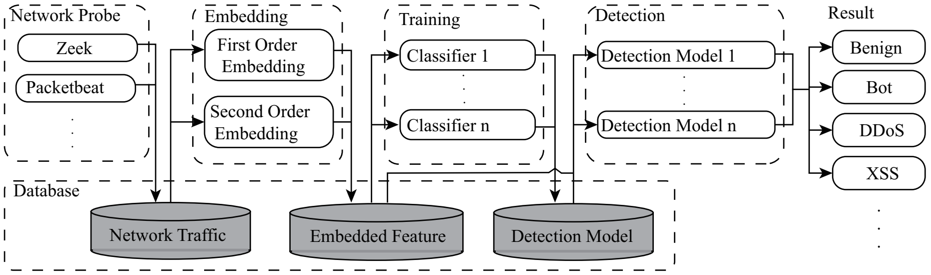


Fig. 4. Overview of the network anomaly detection framework

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50423-6_12

Network Probe. The network probe module captures traffic from network and generates traffic logs. The network probe module can be deployed on devices such as personal computers, firewalls, and servers. As mentioned in Section 3, our algorithm is to calculate the low latitude quantization representation of IP and port using IP, port, protocol, the number of packets and bytes(e.g. Table 1). So the generated log must contain these fields. There are many open source tools (e.g. Zeek[20], Packetbeat[4], and Joy[10]) that generate network traffic logs that meet our requirements. We employ these existing tools to generate and unify network traffic logs.

Table 1. An example of network traffic logs

Source IP Address	Source Port	Destination IP Address	Destination Port	Protocol	Packets	Bytes
192.168.100.5	445	192.168.220.16	58844	TCP	1	108
192.168.100.5	445	192.168.220.15	48888	TCP	1	108
192.168.220.15	48888	192.168.100.5	445	TCP	2	174
192.168.220.16	58844	192.168.100.5	445	TCP	2	174
192.168.100.5	445	192.168.220.15	48888	TCP	1	108
192.168.220.16	58844	192.168.100.5	445	TCP	2	174

Embedding. In the embedding module, we use the method in Section 3 to learn the first-order and second-order low-dimensional vector representation of the nodes in these graphs. For the initial training, we randomly initialize the embedding vector of each node. For each subsequent training, we use the embedding vector of the last training as the initial value of this training, and then use the new data for iterative training. At the end of each training, we store the results in the database for later use.

Training. In the training module, we train multiple machine learning classifiers using features learned in the embedding phase and some statistical features (e.g. flow duration, total forward packets, total backward packets). There are differences in the accuracy of different machine learning algorithms for detecting different types of attacks. In order to improve the accuracy of detection, we calculated the weights of different classifiers on different attack categories on the test set. $acc_{i,j}$ is the accuracy of the i-th classifier on the j-th attack category. $w_{i,j} = \frac{acc_{i,j}}{\sum_i acc_{i,j}}$ is the classification weight of the i-th machine learning classifier on the j-th attack category.

Detection. In the detection module, we use multiple machine learning classifiers to detect new network traffic log. $p_{i,j}$ is the probability that the i-th machine learning classifier considers this network traffic log belongs to the j-th attack category, where $\sum_j p_{i,j} = 1$. The final category of the output of the detection phase is $\arg\max_j \sum_i w_{i,j} * p_{i,j}$

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50423-6_12

Database. Database module is mainly to store network traffic logs, embedding vector and detection models. There are many open source data storage system, including general relational database, distributed storage framework and so on. In our experimental environment, we use MySQL for storage. When the network bandwidth is very large, a lot of network traffic logs will be generated. In this case, we should consider using a distributed storage framework.

5 Evaluation

In this section, we provide three thorough experiments of graph embeddings and network anomaly detection. The first experiment is a clustering of IP addresses. We verified on the CIDDS-001 dataset[13] whether the first-order and second-order low-dimensional vector representation can learn the latent features of IP addresses. The second experiment is network anomaly detection. We evaluate the precision, recall and F1 measure of our method on the CICIDS 2017 dataset[15]. The third experiment is an unknown threat discovery. We use a portion of the attack categories to train our model and then to evaluate if new attack types can be detected.

5.1 Experiment Setup

A proof-of-concept version of our graph embedding algorithms are implemented by C++ with compiler version g++ 4.8.5, which are running on CentOS 7.6 OS. The server is DELL R440 with 48 CPU cores and 128 GB of memory. We also implement machine learning algorithm for anomaly detection via Python3 and Scikit-learn[11].

5.2 IP Address Cluster

For an effective graph embedding algorithm, hosts of the same category in the network should be close in low-dimensional vector space. When running the clustering algorithm, hosts of the same category should be clustered into a cluster. So in this experiment, we use the DBScan clustering algorithm to cluster IP addresses.

We evaluate the effects of the clustering algorithm from three indicators: *accuracy*, *homogeneity*, and *completeness*. *Homogeneity* and *completeness* are independent of the labels.

The baseline method is *IP2Vec*[12]. *IP2Vec* obtains the low-dimensional vector representation of the IP addresses by training a neural network with single hidden layer. We experimented with the same dataset and experimental method as *IP2Vec*.

In the third week of the CIDDS-001 dataset[13], there are a total of 6,439,783 network traffic flows. We use all data during graph embedding, and only cluster the intranet IP addresses (192.168.100.0/24, 192.168.200.0/24, 192.168.210.0/24, 192.18.220.0/24). We divide the IP address into *server* and *client* according to

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50423-6_12

the role that the host plays in the network. We treat *server* and *printer* as *server*; *windows client* and *linux client* as *client*. In the end, there are a total of 7 *servers* and 19 *clients*.

Table 2. Assignment matrix for the CIDDS-001 dataset

Method	Classes	Cluster 1	Cluster 2	Num. Outliers
Our Method	**server**	7	0	0
	client	0	19	0
IP2Vec	**server**	6	0	1
	client	0	19	0

Table 2 shows the result of using DBScan to cluster IP addresses. As shown in the table, our method can distinguish between *server* and *client* precisely. But *IP2Vec* has a point that cannot be divided into any cluster. Table 3 shows the results of two methods on *accuracy*, *homogeneity* and *completeness*. For the *accuracy* and *completeness*, our method outperforms *IP2Vec*.

Table 3. Comparision of our method and *IP2Vec* for clustering the IP Address within the CIDDS-001 dataset

Similarity Measure	Accuracy	Homogeneity	Completeness	Num. Outliers
Our Method	1.0	1.0	1.0	0
IP2Vec	0.9615	1.0	0.8406	1

5.3 Network Anomaly Detection

In this experiment, we used the CICIDS 2017[15] dataset. The CICIDS 2017 dataset contains a total of 2,830,743 network flows. The attack types include DDoS, DoS, Brute Force, XSS, SQL Injection, Infiltration, Port Scan and Botnet. We remove the attack categories of less than 100 flows. For the remaining categories, we used 60% of each category of data as a training set and 40% as a test set.

There are 80 features in CICIDS 2017. In [15], the authors evaluated the importance of each feature on detecting attacks and selected eight most important features: flow duration, total forward packets, total backward packets, total length of forward packets, total length of backward packets, flow bytes per seconds, flow packets per seconds, down/up ratio. Most of the feature extraction tools can extract these eight features. Therefore, in this experiment we use first-order vector representation, second-order vector representation and these eight features training machine learning algorithms.

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50423-6_12

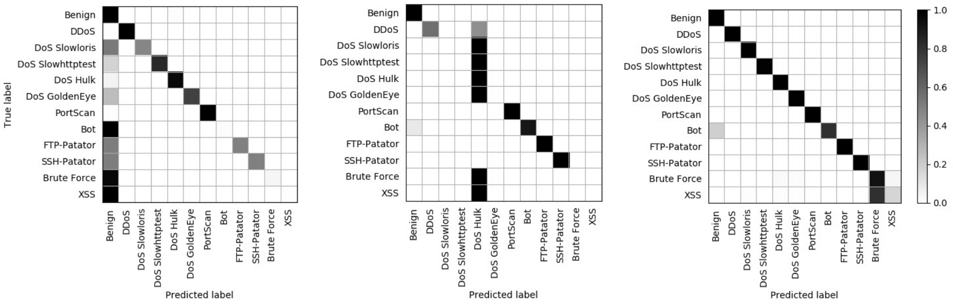
We did three times evaluations in this experiment. For the first time, we only use the raw features. For the second time, we only use embedding features. For the third time, we use both the raw features and the embedding features. We use the random forest algorithm to train and evaluate the results from three indicators: *Precision*, *Recall* and *F1 measure*.

Table 4. Performances of random-forest classifier over raw features, embedding features and all features

Attack	**Raw Features**			**Embedding Features**			**All Features**		
Scenarios	*Precision*	*Recall*	*F1*	*Precision*	*Recall*	*F1*	*Precision*	*Recall*	*F1*
Benign	0.9862	0.9860	0.9861	**0.9999**	**1.0000**	**1.0000**	0.9998	0.9999	0.9999
Bot	1.0000	0.0140	0.0276	**1.0000**	**0.9117**	**0.9538**	0.9502	0.8098	0.8744
DDoS	0.9969	0.9929	0.9949	0.9950	0.5596	0.7163	**0.9993**	**0.9995**	**0.9994**
DoS GoldenEye	0.9178	0.7402	0.8195	0.0000	0.0000	0.0000	**0.9945**	**0.9900**	**0.9922**
DoS Hulk	0.8806	0.9507	0.9143	0.7411	0.9987	0.8509	**0.9979**	**0.9998**	**0.9988**
DoS Slowhttptest	0.9812	0.8291	0.8987	0.0000	0.0000	0.0000	**0.9968**	**0.9998**	**0.9988**
DoS Slowloris	**1.0000**	0.4705	0.6399	0.0000	0.0000	0.0000	0.9857	**0.9885**	**0.9871**
FTP-Patator	**1.0000**	0.4921	0.6596	0.9730	0.9971	0.9849	0.9979	**1.0000**	**0.9990**
PortScan	0.9936	0.9912	0.9924	0.9998	0.9938	0.9968	0.9998	**0.9984**	**0.9991**
SSH-Patator	1.0000	0.4960	0.6631	0.9650	0.9977	0.9811	**1.0000**	**0.9989**	**0.9994**
Brute Force	**1.0000**	0.0381	0.0735	0.0000	0.0000	0.0000	0.7085	**0.9248**	**0.8023**
XSS	**1.0000**	0.0077	0.0152	0.0000	0.0000	0.0000	0.6400	**0.1641**	**0.2612**

Table 4 shows the experimental results. As can be seen from the table, using only raw features to train the classifier, the detection accuracy of the classifier is very high, but the recall is relatively low, especially bot, XSS and brute force. Therefore, the F1 measure of the classifier is low. In addition, the results of a classifier trained using only embedding features are also less effective. Although it has high detection accuracy and recall rate for several attack scenarios(e.g. Benign, Bot and Portscan), it has multiple attack scenarios that cannot be detected (e.g. DoS GoldenEye, DoS Slowhttptest and XSS). If both the raw feature and the embedding feature are used to train the classifier, the performance of the classifier can be greatly improved. It not only has a high detection accuracy rate, but also has a high recall rate.

Fig. 5 shows the confusion matrix of the three times evaluations. As can be seen from the figure, a classifier trained using only raw features classifies a large amount of malicious traffic into benign traffic, which has low attack false positive rate. Although the classifier trained with embedding features cannot accurately detect the attack scenarios, it rarely categorizes malicious traffic as benign traffic, and the attack false negative rate is very low. A classifier trained using both raw features and embedding features has the advantages of both. It can not only identify malicious traffic, but also accurately detect different attack scenarios.



(a) Raw Features (b) Embedding Features (c) All Features

Fig. 5. Confusion matrix of random-forest classifier over different feature sets

5.4 Unknown Threat Discovery

In this experiment, we use 60% of benign traffic and all Bot, PortScan, DoS Hulk, DoS slowloris, FTP-Patator, Brute Force attack traffic as training set, and the remaining 40% of benign traffic and other attack types as test set. The raw random forest algorithm is adopted to train a binary classifier, then to evaluate the ability of our algorithm to detect unknown attack traffic.

Table 5. Results of unknown threat discovery

True Classes	Benign	Anomaly	Total
Benign	906262	173	906435
DDoS	36	127991	128027
DoS Slowhttptest	0	5499	5499
DoS GoldenEye	0	10293	10293
Heartbleed	11	0	11
SSH-Patator	2	5895	5897
Web Attack-XSS	0	652	652
Web Attack-SQL Injection	0	21	21
Infiltration	36	0	36
Total	906347	150524	-

Table 5 shows the prediction result on the test set. 173 out of 906435 benign flows in the test set were predicted as anomaly flows, with an accuracy of 99.98%. 83 of the 150436 attack traffic in the test set were predicted as normal traffic, with an accuracy of 99.94%. Most of the attack traffic can be accurately detected, but Heartbleed and Infiltration is all predicted to be normal traffic. We analyzed the reason may be that the behaviours of these two attack categories are different from the categories of attacks in the training set and were not recognized correctly. Our approach can detect variants of known attacks, but is limited in its ability to detect brand-new attacks.

6 Related Work

The network anomaly detection technology has been divided into three categories according to [5]: statistical-based, knowledge-based and machine learning-based. Deep learning-based network anomaly detection has emerged in recent years due to the development of deep learning.

6.1 Statistical-based Network Anomaly Detection

Caberera et al.[3] proposed a statistical flow modeling method that can detect unknown network attacks. The authors constructed a *Network Activity Models* and *Application Models* and evaluated it on the DARPA'98 dataset. The experimental results showed that the *Network Activity Models* can detect Denial-of-Service and Probing attacks; *Application Models* can distinguish between the normal telnet connections and attacks using telnet connections.

6.2 Knowledge-based Network Anomaly Detection

By integrating specification-based with anomaly-based intrusion detection technology, Sekar et al.[14] proposed a intrusion detection approach by using a state-machine to begin with the specification of network protocol, then expanding the state of the state-machine using existing knowledge to detect the anomaly state.

6.3 Machine Learning-based Network Anomaly Detection

Ariu et al.[2] proposed *HMMPayl*, an intrusion detection system using Hidden Markov Models(HMM). *HMMPayl* represented the HTTP payload as a sequence of bytes, and detected if it is an attack. *HMMPayl* has a detection rate of over 98% on the DARPA'99 dataset[9]. Syarif et al.[16] evaluated the anomaly detection accuracy of five different clustering algorithms: K-Means, improved K-Means, K-Medoids, EM clustering and distance-based outlier detection algorithms. Xu et al.[18] proposed an anomaly detection framework integrating artificial intelligence engine with fog computing, and used semi-supervised machine learning to train a robust detection model.

6.4 Deep Learning-based Network Anomaly Detection

Zhu et al.[21] proposed a intrusion detection approach based on Attention-based Multi-Flow LSTM by using features of multiple flows rather than a single flow as input of LSTM, so that it can capture temporal feature between flows. They also add the attention mechanism to LSTM by achieving a high accuracy on the

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50423-6_12

CICIDS 2017 dataset[15]. Wang et al.[17] proposed a CNN architecture similar to LeNet-5 to train raw binary traffic and classify it. Aldwairi et al.[1] used NetFlow to train the Restricted Boltzmann Machine and achieved a high accuracy on the CICIDS 2017 dataset. Yao et al.[19] proposed a framework for solving network anomaly detection tasks by integrating the graph kernel with deep neural networks, and evaluated different integrating methods.

7 Conclusion

In this paper, we propose a network anomaly detection framework based on graph embedding. We convert the network traffic data into first-order and second-order graph. First-order graphs learn the latent features from the perspective of a single host, and second-order graphs learn the latent features from a global perspective. This feature extraction process does not require human involvement at all. After training the machine learning classifiers using the first-order and second-order vector representation and some statistical features, we use these classifiers to detect new network flows. Three experiments on two real-world datasets showed that our first-order and second-order vector representation can learn latent features and can improve the efficiency of the anomaly detection framework. We also used some of the attack types traffic to train and predict new attack types in network traffic, then evaluated our framework's ability to discover unknown attack types.

Acknowledgement. This research is supported by Key Laboratory of Network Assessment Technology, Chinese Academy of Sciences and Beijing Key Laboratory of Network Security and Protection Technology. We thank the anonymous reviewers for their insightful comments on the paper.

References

1. Aldwairi, T., Perera, D., Novotny, M.A.: An evaluation of the performance of restricted boltzmann machines as a model for anomaly network intrusion detection. Computer Networks **144**, 111–119 (2018)
2. Ariu, D., Tronci, R., Giacinto, G.: HMMPayl: An intrusion detection system based on hidden markov models. computers & security **30**(4), 221–241 (2011)
3. Caberera, J., Ravichandran, B., Mehra, R.K.: Statistical traffic modeling for network intrusion detection. In: Proceedings 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (Cat. No. PR00728). pp. 466–473. IEEE (2000)
4. Elasticsearch: Packetbeat: Network analytics using elasticsearch (2020), https://www.elastic.co/products/beats/packetbeat
5. Garcia-Teodoro, P., Diaz-Verdejo, J., Maciá-Fernández, G., Vázquez, E.: Anomaly-based network intrusion detection: Techniques, systems and challenges. computers & security **28**(1-2), 18–28 (2009)

ICCS Camera Ready Version 2020
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-50423-6_12

6. Gui, H., Liu, J., Tao, F., Jiang, M., Norick, B., Han, J.: Large-scale embedding learning in heterogeneous event data. In: 2016 IEEE 16th International Conference on Data Mining (ICDM). pp. 907–912. IEEE (2016)
7. Lashkari, A.H., Zang, Y., Owhuo, G.: Netflowmeter (2020), http://netflowmeter.ca/
8. Li, A.Q., Ahmed, A., Ravi, S., Smola, A.J.: Reducing the sampling complexity of topic models. In: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 891–900. ACM (2014)
9. Lippmann, R., Haines, J.W., Fried, D.J., Korba, J., Das, K.: The 1999 darpa off-line intrusion detection evaluation. Computer networks **34**(4), 579–595 (2000)
10. McGrew, D., Anderson, B.: Joy (2020), https://github.com/cisco/joy
11. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al.: Scikit-learn: Machine learning in python. Journal of machine learning research **12**(Oct), 2825–2830 (2011)
12. Ring, M., Dallmann, A., Landes, D., Hotho, A.: IP2Vec: Learning similarities between ip addresses. In: 2017 IEEE International Conference on Data Mining Workshops (ICDMW). pp. 657–666. IEEE (2017)
13. Ring, M., Wunderlich, S., Grüdl, D., Landes, D., Hotho, A.: Flow-based benchmark data sets for intrusion detection. In: Proceedings of the 16th European Conference on Cyber Warfare and Security. ACPI. pp. 361–369 (2017)
14. Sekar, R., Gupta, A., Frullo, J., Shanbhag, T., Tiwari, A., Yang, H., Zhou, S.: Specification-based anomaly detection: a new approach for detecting network intrusions. In: Proceedings of the 9th ACM conference on Computer and communications security. pp. 265–274. ACM (2002)
15. Sharafaldin, I., Lashkari, A.H., Ghorbani, A.A.: Toward generating a new intrusion detection dataset and intrusion traffic characterization. In: ICISSP. pp. 108–116 (2018)
16. Syarif, I., Prugel-Bennett, A., Wills, G.: Unsupervised clustering approach for network anomaly detection. In: International conference on networked digital technologies. pp. 135–145. Springer (2012)
17. Wang, W., Zhu, M., Zeng, X., Ye, X., Sheng, Y.: Malware traffic classification using convolutional neural network for representation learning. In: 2017 International Conference on Information Networking (ICOIN). pp. 712–717. IEEE (2017)
18. Xu, S., Qian, Y., Hu, R.Q.: A semi-supervised learning approach for network anomaly detection in fog computing. In: ICC 2019-2019 IEEE International Conference on Communications (ICC). pp. 1–6. IEEE (2019)
19. Yao, Y., Su, L., Zhang, C., Lu, Z., Liu, B.: Marrying graph kernel with deep neural network: A case study for network anomaly detection. In: International Conference on Computational Science. pp. 102–115. Springer (2019)
20. Zeek.org: The zeek network security monitor. (2020), https://www.zeek.org/
21. Zhu, M., Ye, K., Wang, Y., Xu, C.Z.: A deep learning approach for network anomaly detection based on amf-lstm. In: IFIP International Conference on Network and Parallel Computing. pp. 137–141. Springer (2018)