TDAE: Autoencoder-based Automatic Feature Learning Method for the Detection of DNS tunnel

1st Kemeng Wu
Institution of Information Engineering
(Chinese Academy of Science)
School of Cyber Security
(Univ. of Chinese Academy of Science)
Beijing, China
wukemeng@iie.ac.cn

2nd Yongzheng Zhang
Institution of Information Engineering
(Chinese Academy of Science)
School of Cyber Security
(Univ. of Chinese Academy of Science)
Beijing, China
zhangyongzheng@iie.ac.cn

3rd Tao Yin
Institution of Information Engineering
(Chinese Academy of Science)
School of Cyber Security
(Univ. of Chinese Academy of Science)
Beijing, China
yintao@iie.ac.cn

***Abstract*—The DNS protocol is one of the most important network infrastructure protocols. The encrypted information based on this protocol will not be intercepted by the firewall, so the attacker uses this vulnerability to pass private data through the establishment of DNS tunnels and avoids the security inspection. In order to detect the DNS tunnel conveniently and effectively, we present a novel method that uses Autoencoder to learn latent representation of different datasets. Because the feature is not extracted manually, we show how Autoencoder(AE) can automatically learn the concept of semantic similarity among features of normal traffic. We propose a novel method named TDAE which can detect DNS tunnel traffics using Autoencoder algorithms. To verify the validity of our method, we select a labeled dataset and a public and unlabeled dataset as our training set. The experimental results show that the recall rate can exceed 0.9834 on the labeled dataset and 0.9313 on the SINGH-data [1].**

***Index Terms*—Network security, Domain name system, Covert channel, Deep learning, Semi-supervised learning, Detection**

I. Introduction

The DNS protocol is one of the most important protocol in the Internet infrastructure, which has the remarkable ability to map domain names to IP addresses. It also provides a flexible and reliable domain name resolution for the upper network applications. However, the cybercriminals can establish the covert channel to transmit the encrypted data by using special message formats and types. For example, a firewall open port 53 as the default port using UDP, so cybercriminals use the DNS traffic to piggyback their actual communication. DNS tunnels are typically used in man-in-middle attacks, information leaks, and cross the login screen and access the Internet through illegal channels qualification, these malicious behavior have caused serious economic losses to enterprises and individuals. Therefore, the detection of DNS tunnel is of pivotal importance for our increased understanding of the study of web steganography.

There have been several open-source DNS tunneling tools which are widely used by hackers, such as dnscapy [2], iodine [3], dns2tcp [4], ozymandns [5], and dnscat2 [6]. These DNS tunneling tools are similar in coding and function. Moreover, attackers will adjust the DNS tunnel coding according to the existing detection features. These attacks based on the DNS protocol require a principled means which can detect covert channel traffic that is not blocked by the firewall.

Nowadays, with the rapid advancement of computer performance, deep learning algorithms are widely used for DNS tunnel detection. The training methods of deep learning are divided into supervised learning, semi-supervised learning and unsupervised learning. Usually, in anomaly intrusion detection, the performance of supervised learning is much better than unsupervised learning, because the tagged data can help improve the accuracy of the algorithm. However, it is very difficult to mark a large amount of traffic, especially the traffic of DNS tunnels which is hard to capture in the real world. So this also prompt scientists to try to use unsupervised or semi-supervised detection methods to analyze traffic. Additionally, the conventional machine learning methods detect DNS tunnel by extracting features, however, hackers often adjust the structural features of DNS tunnel to avoid detect.

Deep learning algorithms are widely used in network anomaly detection [7] [8] . These methods do not require automatically extract features through deep learning algorithms, which are highly dependent on data that has been marked. We propose an automatic extraction feature method(TDAE) based on Autoencoder that can operate on a large amount of unlabeled data and quickly identify the traffic generated by the DNS tunneling tools. Our detection idea is to use the detection method of anomaly traffic. Since there are a large number of data generated by non-covert tunnels in the real environment, we record the weight of the neural network by training the data of the non-covert tunnels, and then put the rest in the testing stage. If the input data can be restored relatively accurately after decoding, it should be the normal traffic, otherwise, it is determined to be DNS tunnel.

In order to be able to fully evaluate our method, we have evaluated the effectiveness of the proposed scheme using two publicly available datasets. One is the data of different DNS types generated by Alexa [9] which does not contain DNS tunnel traffic and its domain name is whitelisted. The other is the public campus DNS traffic data which is unmarked

This work has been supported by the National Natural Science Foundation of China under No.U1736218.

and contains malicious domain names. In a word, the key contributions of this paper can be summarized as follows:

(1)We use an AE-based semi-supervised feature learning approach for covert tunnel detection, which not contains any prior knowledge of the DNS tunnel;

(2)TDAE can automatically learn the concept of semantic similarity among features of normal traffic;

(3)Since our training phase uses unmarked data and does not manually extract features from the data, it greatly reduces the time required for data preprocessing.

The rest of the paper can be summarized as follows. In the second section, this paper reviews the relevant work background on DNS tunnel detection using machine learning models and non-machine learning approaches. Subsequently, the experimental operation process and AE model are elaborated in detail in the third section. In the fourth section, we present the components of the experiment data and set up important indicators that can judge the performance of different models. Finally, the fifth section summarized the whole paper.

II. Related Work

We categorize the previous work on the DNS tunnel detection for two types: machine learning methods and non-machine learning methods. Because more recent attention has focused on the provision of machine learning, our literature review is mainly related to machine learning methods.

A. *Machine Learning Methods*

In the current research, machine learning detection methods have been widely used in DNS tunnel detection in recent years, most of the detection method is by extracting the DNS tunnel behavioral characteristics. Based on this research trend, in 2009, Jhind first proposed a classifier built with neural networks to classify DNS tunnels by extracting the characteristics of domain names [10]. AILELO attempted to perform DNS tunnel detection using different mainstream machine learning models, such as Principal Component Analysis (PCA), Bayesian classification, k-proximity, neural networks, and support vectors. [11], [12], [13] . The paper [14] showed a detection method over mobile network, which contained the following characteristics: extracts time, source IP, target IP, protocol, length, and ancillary information. This paper also proposed two kinds of machine learning techniques: One Class Support Vector Machine (OCSVM) and K-Means, and finally OCSVM with the Radial Basis Function kernel obtained the higher and best result with F1 score of 0.96. The paper [15] introduced a model that uses random forests, and convert the DNS tunnel to 16 features vectors. The paper [16] developed a better method which obtained 99.96% extremely high accuracy. Nadler used a single day of DNS queries and responses, extract the features and apply an Isolation Forest model [17]. Although these methods have excellent performance, the disadvantages are as follow: Firstly, most studies have no published data sets to verify the accuracy of the experiment. Secondly, the size of these data sets are too small and can not reflect the general meaning, or can not describe a comprehensive evaluation system. Thirdly, most of the experiments use the feature extraction method. If the cybercriminal knows the model features, he will change the encoding mode of the DNS tunnel to avoid detection.

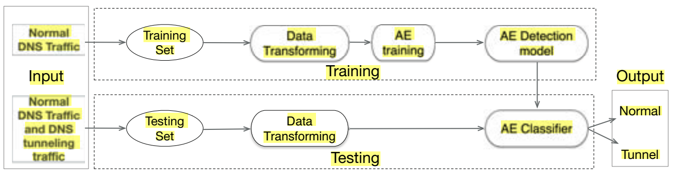

Fig. 1. The main framework of TDAE

B. *Non-machine Learning Methods*

1) Payload Analysis Methods: Payload analysis methods is an effective feature extraction for DNS query and response packets to quickly identify normal traffic and DNS tunnel traffic. These methods can be divided into three types : based on package length, based on entropy and based on distribution. However, as the number of domain names increases, variables such as domain name lengths are no longer reliable. Comparing the larger entropy value of the requested host name and the distribution of specific characters would be more efficient method of determining the DNS tunnel. Jaworski presented a tool to detect the DNS tunnel which was called Splunk, using the entropy of hostname to determine [18]. BORN et al. proposed to use n-gram frequencies to detect DNS tunnels [19]. Bilge calculated the percentage of the number in the domain name [20]. Lockington checked the repeated consonants or numbers [21]. Anirban Das proposed to detect the DNS tunnel by extract 10 character features in the subdomain, and obtained the true positive rate of 91.68% and the false positive rate of 0.4% [22]. However, the hackers can bypass the detection by knowing the relevant features.

2) Traffic Analysis Methods: The detection methods of traffic analysis are as follows: Michael proposed the throughput estimation such as value of DNS traffic per IP address, volume of DNS traffic per domain to detect DNS tunnels [23]. Wendy and Anna presented a flow-based detection which count bytes per flow and the number of flows [24]. However, this type of method has to collect a lot of traffic and can not make real-time detection.

III. Materials and Method

A. *Overview*

We describe a deep learning algorithm that draws on anomaly detection. We first parse the normal DNS request and convert it to a decimal form. Then we put the converted data into AEs for training. In the testing phase, we mix the normal DNS queries with DNS tunnel traffic. According to the normal data training AEs, normal DNS traffic reconstruction can be restored, but the DNS tunnel different from the normal distribution is hard to restore. The framework of this method is shown in Fig.1.

B. Data Transforming

We need to convert the pcap file into a form that the neural network are able to recognize, so that the neural network can easily calculate the weight of each network node. Each pcap file is captured by wireshark and saved in hexadecimal form. A sample packet of a pcap file is:

001c 4200 0018 001c 42ef da91 0800 4500 003b 6aee 4000 4011 4c0d 0ad3 3710 0ad3 3701 c96d 0035 0027 83ef c466 0100 0001 0000 0000 0000 0977 696b 6970 6564 6961 036f 7267 0000 0100 01

This is a DNS query for $Wikipedia.com$. These hexadecimal information contains the content shown in the first stage of Fig.2. First we cut the pcap file according to the length of

Machine(RBM) is the undirected graphical model which has a two-layer neural network [25]. The upper layer of neurons makes up the hidden layer, and the lower layer of neurons makes up the visible layer. RBM network is shown in Fig.2. For the binary states of visible and hidden units, the energy functions are bilinear:

$$E(\boldsymbol{x}, \boldsymbol{h}; \theta) = -\sum_{i \in visible} b_i x_i - \sum_{j \in hidden} a_j h_j - \sum_{i,j} x_i h_j w_{ij} \tag{1}$$

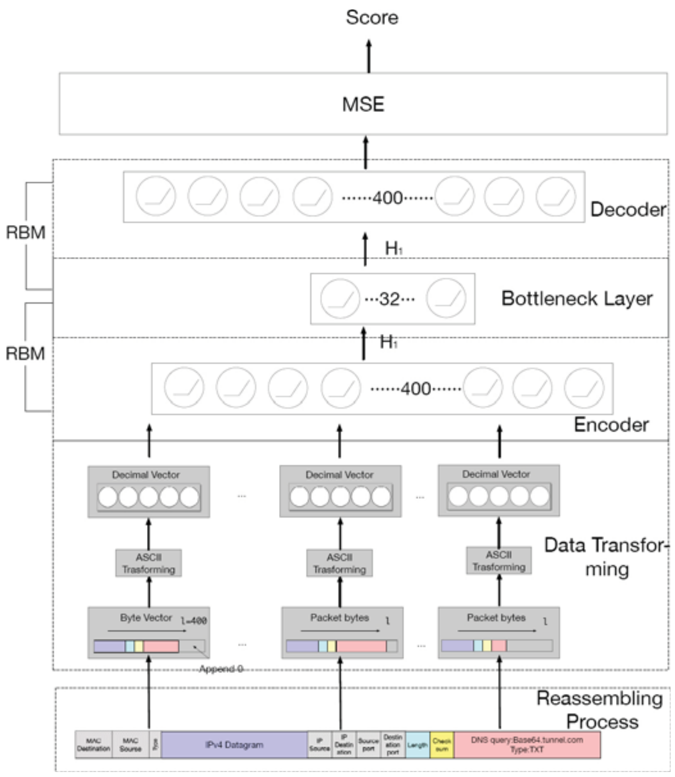

Fig. 2. The Topology of our AE

where w_{ij} represents the weights between visible units x_i, and hidden units h_j, b_i and a_i are their biases. The joint distribution $p(\boldsymbol{x}, \boldsymbol{h}; \theta)$ has the following the energy function:

$$p(\boldsymbol{x}, \boldsymbol{h}; \theta) = \frac{\exp(-E(\boldsymbol{x}, \boldsymbol{h}; \theta))}{Z} \tag{2}$$

Where $Z = \sum_{\boldsymbol{x}, \boldsymbol{h}; \theta} \exp(-E(\boldsymbol{x}, \boldsymbol{h}))$ is a normalization factors which are similar in softmax. The conditional probabilities for Bernoulli-Bernoulli RBM is:

$$\begin{aligned} p\left(h_j = 1 | \boldsymbol{x}; \theta\right) &= \frac{\exp\left(\sum_i w_{ij} x_i + a_j\right)}{1 + \exp\left(\sum_i w_{ij} x_i + a_j\right)} \\ &= f\left(\sum_i w_{ij} x_i + a_j\right) \end{aligned} \tag{3}$$

$$\begin{aligned} p\left(x_i = 1 | \boldsymbol{h}; \theta\right) &= \frac{\exp\left(\sum_j w_{ij} h_j + b_i\right)}{1 + \exp\left(\sum_j w_{ij} h_j + b_i\right)} \\ &= f\left(\sum_j w_{ij} h_j + b_i\right) \end{aligned} \tag{4}$$

Our target of use RBM is to evaluate the parameters of θ, so we need to learn by maximizing RBM's logarithmic release function on the training set (assuming T samples):

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) = \arg\max_{\boldsymbol{\theta}} \sum_{t=1}^{T} \log p\left(\mathbf{x}^{(t)} | \boldsymbol{\theta}\right) \tag{5}$$

Taking the derivative of the negative logarithm probability of the input relative to the weight, we can get:

$$\begin{aligned}\frac{\partial - \log p(x)}{\partial \theta_{ij}} &= \frac{\partial}{\partial \theta}\left(-\log \sum_{h} p(\boldsymbol{x}, \boldsymbol{h})\right) \\ &= \langle x_i, h_j \rangle_{data} - \langle x_i, h_j \rangle_{recon}\end{aligned} \tag{6}$$

The angle brackets are used to represent the mathematical expectation of different distributions p. We summarize a simple learning algorithm:

$$\begin{aligned}\Delta W_{ij} &= \epsilon \left(\langle x_i h_j \rangle_{\text{data}} - \langle x_i h_j \rangle_{\text{recon}}\right) \\ \Delta a_i &= \epsilon \left(\langle h_j \rangle_{\text{data}} - \langle h_j \rangle_{\text{recon}}\right) \\ \Delta b_j &= \epsilon \left(\langle x_i \rangle_{\text{data}} - \langle x_i \rangle_{\text{recon}}\right)\end{aligned} \tag{7}$$

where ϵ is a learning rate, $\langle . \rangle_{\text{recon}}$ represents the distribution of the model definition after one step reconstruction. The possibility of maximizing data or log likelihood proposed by Hinton is equivalent to minimizing the KullbackLeibler(KL) divergence between the data distribution and the equilibrium distribution over the visible variables. To calculate this expectation, the k-step contrast divergence (CD-k) approximation provides surprising results [26]. We used CD-1 and ran a one-step Gibbs sampler, which was good enough:

$$\boldsymbol{x} = \boldsymbol{x^0} \overset{p(\boldsymbol{h}|\boldsymbol{x}^0)}{\longrightarrow} \boldsymbol{h^0} \overset{p(\boldsymbol{x}|\boldsymbol{h}^0)}{\longrightarrow} \boldsymbol{x^1} \overset{p(\boldsymbol{h}|\boldsymbol{x}^1)}{\longrightarrow} \boldsymbol{h^1} \tag{8}$$

Stacking RBM blocks can form the topology of the AE we need, and we train the AE in a greedy layer-wise fashion using individual RBMs. By stacking the individual RBM blocks on top of each other, we can generate a topology with deep AEs.

2) DNS tunnel detection with Autoencoders: Let x express as the initial extracted feature vector, we define the hidden representation $H(x)$ as shown below:

$$H(x) = f_i\left(W_i x + b_1\right) \tag{9}$$

where f_i is relu activation function, $i = 1, 2..., N$. Then we map the potential representation H back to the reconstruction $\hat{x}$ in the output layer:

$$\hat{x} = f_i\left(W_2 H(x) + b_2\right) \tag{10}$$

We use back-propagation algorithm to iterate these tuning of parameters. We build this structured network to train our normal DNS traffic data, choose a single value for $epochs = 50$, and use the mean squared error as the loss function for this semi-supervised fine-tuning phase. During the testing process, we calculate the reconstruction error to determine whether it belongs to the DNS tunnel. We use the a indicators defined in Eq.(9)-Mean Square Error (MSE), which are convenient for evaluating the performance of the DNS tunnel detection:

$$MSE = \frac{1}{m}\sum_{i=1}^{m}\left(x_i - (\hat{x}_i)\right)^2 \tag{11}$$

where m is the dimensionality of the input vectors. If the value of MSE is greater than the threshold, we will mark it as DNS tunnel. The Autoencoder based DNS tunnel detection is described as Algorithm 1:

Algorithm 1 Autoencoder based DNS tunnel detection algorithm

```
Input: Normal dataset $X$; DNS tunnel dataset $x_{(i)}$   $i = 1, \cdots, N$; threshold $\alpha$
Output: mean square error $\frac{1}{m}\sum_{i=1}^{m}(x_i - \hat{x}_i)^2$
1: $\phi, \lambda \leftarrow$ train an autoencoder using the normal dataset $X$;

2: for i=1 to N do
3:     mean square error $\frac{1}{m}\sum_{i=1}^{m}(x_i - \hat{x}_i)^2$;
4:     if reconstruction MSE($i$) $> \alpha$; then
5:         $x^{(i)}$ belong to the DNS tunnel
6:     else
7:         $x^{(i)}$ not belong to the DNS tunnel
8:     end if
9: end for
```

IV. Evaluation

In this section, we evaluate the detection performance of our model. First we introduce the composition of our training set and testing set. Then we introduce the indicators used to evaluate different models. Finally, we give the result we are looking for.

A. Datasets

We use two partial data sets to analyze the AE-based representation. One data set is generated by a large number of normal traffic requests through a software called dns-grind [27]. Then, we download the first 20,000 domain names of the Alexa in the latest version, querying them with the common RR types A, MX, and TXT record type separately. We use this software to create DNS queries for two main reasons: firstly, it can guarantee that the DNS traffic contain only clean traffic, and secondly it can generate many resource record types we want, such as TXT type. This type of record is relatively rare in real-world environments, but DNS tunnels make extensive use of this type of resource record. Another datasets come from the public campus DNS traffic, which consist of more than 4000 active users (in peak load hours) for 10 random days in the month of April, 2016 [1]. We can name these dataset SINGH. Since this data set is large, we randomly extract a part of it as a subset of the data set which satisfies the distribution of the original data set and is not manually marked. In these two parts of the data, we randomly select 80% for our training data, and the rest for testing data.

In the process of generating DNS tunnel traffic and normal DNS traffic, we use wireshark to help collect traffic and filter the DNS queries. We have established the following five DNS tunneling tools: dns2tcp, dnscapy, dnscat2, ozymandns, and iodine. We classify this type of data into a testing set. The sample size of training and testing sets is shown in the following Table I:

The data generated by dns-grind are benign domain names in the whitelist, so these domain names are usually semantic and their lengths of domain names are not very long. According to the description of paper [28], the campus traffic includes not only a large number of benign domain names, but also some domain names generated by Domain Generation Algorithms(DGA) [29]. Although these domain names belong to malicious domain names, they are not generated by DNS tunnels. These domain names belong to non-pronounceable domains. This is very similar to the domain name generated by the DNS tunnel.

TABLE I
THE SAMPLE SIZE OF TRAINING AND TESTING SETS

Dataset	Training Set	Testing Set
Alexa	253,451	63,363
SINGH	569,070	142,268
DNS tunnel	-	94,820

B. Performance Metrics

Accuracy, precision, recall, and F1 scores are the four fundamental evaluation metrics used to evaluate performance of the system. We can classify the correctness of the samples into four categories: True Positive (TP), True negative (TN), False Positive (FP), False Negative (FN). Accuracy, precision, recall, and F1 formulas are as follows:

$$\text{Accuracy} = \frac{\mathrm{TP}+\mathrm{TN}}{\mathrm{TP}+\mathrm{FP}+\mathrm{TN}+\mathrm{FN}} \tag{12}$$

$$\text{Precision} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}} \tag{13}$$

$$\text{Recall} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}} \tag{14}$$

$$\mathrm{F1} = \frac{2^{*}\ \text{Recall} \ * \ \text{Precision}}{\text{Recall} \ + \ \text{Precision}} \tag{15}$$

Accuracy is the ratio of true DNS tunnel to the total observations. Precision is the ratio of correctly predicted DNS tunnel to the total positive observations. Recall is the ratio of correctly predicted DNS tunnel to the predicted positive observations. F1 Score is the weighted average of Precision and Recall.

The PR(Precision-Recall) curve is an extraordinarily critical indicator which allows us to quickly compare with the performance of the classifier in the evaluation of deep algorithms. Its ordinate (the vertical axis) represents precision, and the abscissa (the horizontal axis) represents recall. The PR curve have better performance of the classification when the ratio of positive and negative samples is large.

There are another metrics called the ROC and AUC. ROC (Receiver Operating Characteristic) is the curve which is based on FP rate and TP rate and the area of the ROC curve is AUC (Area Under the Curve). AUC is used to measure the performance of ”two-class problem” deep learning algorithms and the closer the value of AUC is to 1, the better the result is.

C. Results

In this section, we use the terms TDAE, TDDAE, TDSAE, and TDVAE to refer to the DNS tunnel detection model based on sample Autoencoder, Deep Autoencoder [30], Sparse Autoencoder [31], Variational Autoencoder [32].

Figure.3 plots TDAD’s reconstruction MSE for normal traffic and DNS tunnel. The red line is the lowest threshold we select during the testing phase. We can see that most of the points under the red line are normal traffic, while the upper part of it is mostly orange dots, and these points represent DNS tunnels.

We calculate the Reconstruction MSE for each dataset separately, and use scatter plots to represent the detection results under our models in Fig.3. In order to get the best F1 value, the testing phase yields a threshold of reconstruction MSE $\alpha = 0.018$ in Fig.3. The performance evaluation of the experimental results is shown in TableII.

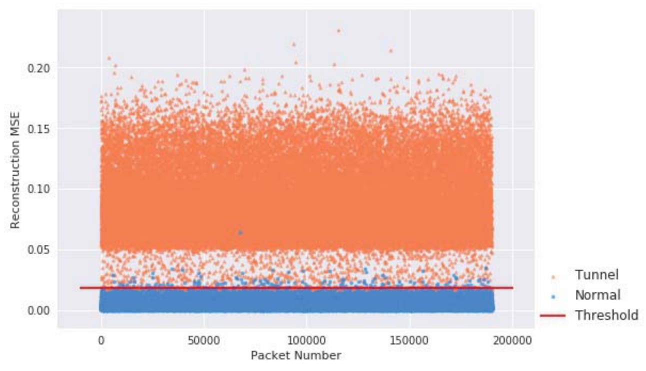

Fig. 3. Reconstruction MSE for normal traffic and DNS tunnel

Through comparison of different data sets, we can find that the Alexa dataset marked as a white domain has a larger F1 score than the unmarked SINGH dataset. The experimental results show that the components of the training set directly apply the model. It is easier to detect the DNS tunnel by marking a large number of whitelisted domain names. At the same time, the DNS traffic of the campus network is more complicated than Alexa because it is not manually marked, but the F1 score is also above 0.9.

By comparing the performance of different models, we can conclude that the simplest model has the best result. Whether increasing the number of layers in the network or adjusting the distribution of the network, the performance of another models has not been improved.

The PR curves and ROC curves are displayed in Fig.4, Fig.5. It can be seen that the AUC of TDAD is both higher than the AUC of the three other classifiers on the two different datasets.

Overall, these results indicate that our proposed approach significantly outperforms other model in DNS tunnel detection from complex real-world environments. In the semi-supervised learning mode, we do not need to learn the characteristics of the DNS tunnel to complete the classification task. Our model

TABLE II
DETECTION PERFORMANCE IN TWO DATASETS USING DIFFERENT MODEL

Model	Alexa				SINGH			
	Accuracy	Presion	Recall	F1	Accuracy	Presion	Recall	F1
TDAE	0.9732	0.9512	0.9834	0.9670	0.8984	0.8736	0.9313	0.9016
TDDAE	0.9556	0.9044	0.9941	0.9471	0.8672	0.8980	0.8282	0.8617
TDSAE	0.6733	0.5513	0.9844	0.7068	0.7741	0.8363	0.6810	0.7507
TDVAE	0.8972	0.9054	0.8297	0.8659	0.8785	0.9030	0.8864	0.8946

can obtain feature vectors from a large number of normal DNS traffic and extract semantic similarities between feature vectors.

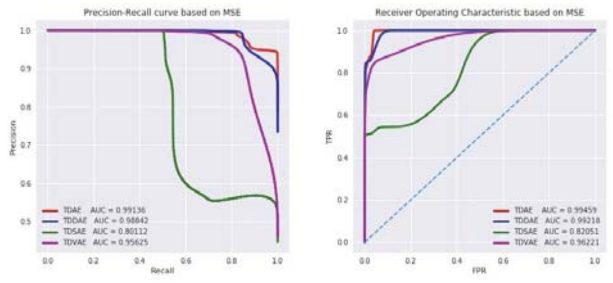

Fig. 4. The PR curves and ROC curves of the four classifiers on Alexa

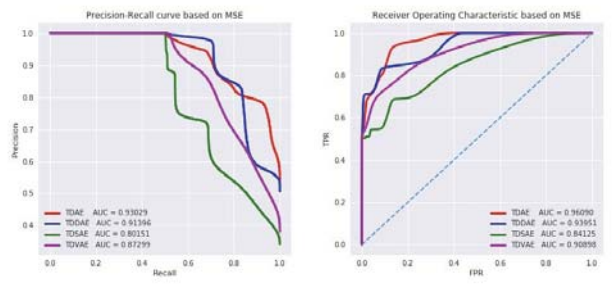

Fig. 5. The PR curves and ROC curves of the four classifiers on SINGH

V. Conclusion

In this paper, we propose a novel and efficient method for detecting DNS tunnels, which is based on semi-supervised learning. This method learn the characteristics of normal DNS traffic through neural networks, and calculate the MSE between different classes of samples to detect DNS tunnels. The experimental results have shown that our method named TDAE can automatically learn the concept of semantic similarity as among features of normal traffic on the campus network traffic.

Two problems require further study in future work. First, in the real world, the amount of DNS tunnel traffic is much more small than the amount of normal traffic, and the proportion of normal traffic to DNS tunnel traffic will vary a lot. The paper [12] attempts to use 1% DNS tunnel traffic for testing, but the results are not ideal. Second, by automatically extracting features that do not contain time-dependent variables, our method becomes undetectable on temporally encoded covert channels. Although temporally encoded covert channels are much smaller in transmission rate of communication, long-term transmission is still a potential threat. To further improve the system performance, our future work will focus on the detection of small-scale DNS tunnels and the time covert channels.

References

[1] M. Singh, "10 days dns network traffic from april-may, 2016," 11 2018. [Online]. Available: http://dx.doi.org/10.17632/zh3wnddzxy.1

[2] P. Bienaime, "Dnscapy," 2012. [Online]. Available: https://code.google.com/archive/p/dnscapy/

[3] E. ANDERSSON, B EKMAN, "iodine," 2011. [Online]. Available: http://code.kiyo.se/iodine/

[4] C. C. Chang. (2006) dns2tcp. [Online]. Available: http://www.hsc.fr/ressources/outils/dns2tcp/index.html.en

[5] D. Kaminsky. (2004) Ozymandns. [Online]. Available: https://room362.com/post/2009/2009310ozymandns-tunneling-ssh-over-dns-html/

[6] PIETRASZEK, "dnscat2," 2010. [Online]. Available: http://tadek.pietraszek.oig/projects/DNScat/

[7] A. Y. Javaid, Q. Niyaz, W. Sun, and M. Alam, "A deep learning approach for network intrusion detection system," in *Eai International Conference on Bio-inspired Information & Communications Technologies*, 2015.

[8] W. Wei, Y. Sheng, J. Wang, X. Zeng, and Z. Ming, "Hast-ids: Learning hierarchical spatial-temporal features using deep neural networks to improve intrusion detection," *IEEE Access*, vol. PP, no. 99, pp. 1–1, 2017.

[9] "Alexa web information company," 2019. [Online]. Available: https://www.alexa.com/topsites

[10] J. Hind, "Catching dns tunnels with ai," *Proceedings of DefCon*, vol. 17, 2009.

[11] M. Aiello, M. Mongelli, and G. Papaleo, "Dns tunneling detection through statistical fingerprints of protocol messages and machine learning," *International Journal of Communication Systems*, vol. 28, no. 14, pp. 1987–2002, 2015.

[12] M. Aiello, M. Mongelli, E. Cambiaso, and G. Papaleo, "Profiling dns tunneling attacks with pca and mutual information," *Logic Journal of IGPL*, vol. 24, no. 6, pp. 957–970, 2016.

[13] M. Aiello, M. Mongelli, and G. Papaleo, "Supervised learning approaches with majority voting for dns tunneling detection," vol. 299, pp. 463–472, 2014.

[14] V. T. Do, P. Engelstad, B. Feng, and T. van Do, "Detection of dns tunneling in mobile networks using machine learning," vol. 424, pp. 221–230, 2017.

[15] A. L. Buczak, P. A. Hanke, G. J. Cancro, M. K. Toma, L. A. Watkins, and J. S. Chavis, "Detection of tunnels in pcap data by random forests," in *Proceedings of the 11th Annual Cyber and Information Security Research Conference*. ACM, 2016, p. 16.

[16] J. Liu, S. Li, Y. Zhang, J. Xiao, P. Chang, and C. Peng, "Detecting dns tunnel through binary-classification based on behavior features," in *2017 IEEE Trustcom/BigDataSE/ICESS*. IEEE, 2017, pp. 339–346.

[17] A. Nadler, A. Aminov, and A. Shabtai, "Detection of malicious and low throughput data exfiltration over the dns protocol," *Computers & Security*, vol. 80, pp. 36–53, 2019.

[18] S. Jaworski, "Using splunk to detect dns tunneling," *SANS Institute InfoSec Reading Room*, 2016.

[19] K. Born and D. Gustafson, "Ngviz: detecting dns tunnels through n-gram visualization and quantitative analysis," *arXiv preprint arXiv:1004.4359*, 2010.

[20] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi, "Exposure: Finding malicious domains using passive dns analysis." in *Ndss*, 2011, pp. 1–17.

[21] S. Lockington. (2012) Detecting the bad from the good. [Online]. Available: http://scottfromsecurity.com/blog/2012/01/16/detecting-the-bad-from-the-good/

[22] A. Das, M.-Y. Shen, M. Shashanka, and J. Wang, "Detection of exfiltration and tunneling over dns," *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pp. 737–742, 2017. [Online]. Available: http://dx.doi.org/10.1109/ICMLA.2017.00-71

[23] M. Himbeault, "A novel approach to detecting covert dns tunnels using throughput estimation," Thesis, 2014.

[24] W. Ellens, P. Żuraniewski, A. Sperotto, H. Schotanus, M. Mandjes, and E. Meeuwissen, "Flow-based detection of dns tunnels," in *IFIP International Conference on Autonomous Infrastructure, Management and Security*. Springer, 2013, pp. 124–135.

[25] R. Salakhutdinov, A. Mnih, and G. E. Hinton, "Restricted boltzmann machines for collaborative filtering," in *International Conference on Machine Learning*, 2007.

[26] G. E. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural Computation*, vol. 14, no. 8, pp. 1771–1800, 2002.

[27] "dns-grind," 2015. [Online]. Available: https://github.com/pentestmonkey/dns-grind

[28] M. Singh, M. Singh, and S. Kaur, "Detecting bot-infected machines using dns fingerprinting," *Digital Investigation*.

[29] "Domain generation algorithm (dga)," 2018. [Online]. Available: https://resources.infosecinstitute.com/domain-generation-algorithm-dga/

[30] Q. V. Le, "Building high-level features using large scale unsupervised learning," in *IEEE International Conference on Acoustics*, 2013.

[31] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P. A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *Journal of Machine Learning Research*, vol. 11, no. 12, pp. 3371–3408, 2010.

[32] E. Bodin, I. Malik, C. H. Ek, and N. D. F. Campbell, "Nonparametric inference for auto-encoding variational bayes," 2017.