Deep and Broad Learning based Detection of Android Malware via Network Traffic

Shanshan Wang[†] Zhenxiang Chen[†*] Qiben Yan[‡] Ke Ji[†] Lin Wang[†] Bo Yang[†] Mauro Conti[§]
[†] Shandong Provincial Key Laboratory of Network Based Intelligent Computing, University of Jinan, Jinan, China
[‡] Department of Computer Science and Engineering, University of Nebraska-Lincoln, Lincoln, NE, USA
[§] Department of Mathematics, University of Padova, Padua, Italy
*Corresponding author, Email: czx@ujn.edu.cn

***Abstract*—In recent years, the scale and diversity of malicious software on mobile networks are constantly increasing, thereby causing considerable danger to users' property and personal privacy. In this study, we devise a method that uses the URLs visited by applications to identify malicious apps. A multi-view neural network is used to create a malware detection model that emphasizes depth and width. This neural network can create multiple views of the input automatically and distribute soft attention weights to focus on different features of input. Multiple views preserve rich semantic information from input for classification without requiring complicated feature engineering. In addition, we conduct comprehensive experiments to compare the proposed method with others and verify the validity of the detection model. The experimental results show that our method has a certain timeliness. It can not only effectively detect malware discovered in different months of a certain year, but also detect potentially malicious apps in the third-party app market. We also compare the detection results of the proposed method on wild apps with 10 popular anti-virus scanners, and the final result shows that our approach ranks second in terms of detection performance.**

I. Introduction

The popularity of mobile devices has driven the advent of the mobile internet era. A recent report [1] shows that the number of available apps in Google Play Store exceeded 1 million in July 2013 and reached 3.5 million in December 2017. However, the quantity and diversity of mobile malware, particularly those that target Android platforms, have increased dramatically [2]. Mobile devices are increasingly associated with personal property and privacy information, which may pose multiple threats to users if compromised. Thus, effective mobile malware detection systems are urgently needed.

A recent survey [3] studies the behavior of a wide variety of malware and classifies existing mobile malware detection methods into three categories, i.e., static analysis, dynamic analysis and traffic-based methods. However, static analysis is challenged by the code polymorphism and obfuscation of malware [4]. These mechanisms are used to generate variants of malware to evade detection. Other dynamic analysis methods attempt to modify the device OS to track and access sensitive information at runtime [5]. These techniques are effective but require a sufficiently large set of executions to cover app behaviors. Thus, performing dynamic analysis on resource-constrained smart devices is challenging.

In addition, many malware detection methods focus on the network traffic generated by mobile apps. The core of network traffic-based methods is to find distinctive features for efficient malware classification. However, feature selection is notoriously difficult. For example, in order to extract diacritical features from network traffic, researchers need to observe a large amount of network traffic data and understand the operating mechanism of malware, which is indeed time-consuming and labor-intensive. Through URL analysis, we find that "imei","longitude","latitude", "apikey" and other dubious words are frequently used in malicious URLs. However, benign URLs usually do not contain these words. Therefore, we design a malware detection method that is based on URLs. The proposed method, which executes URL segmentation and vectorization operations and does not require a time-consuming feature selection process. We use a novel multi-view neural network [6] that focuses on width and depth of neures to mine URL feature information from multiple levels automatically to address the challenge of feature selection, while retaining the rich semantic from the input data.

In summary, we provide the following contributions to the literature on Android malware detection.

- We perform text-like segmentation and vectorization on the URLs which is instructive to malware detection using network traffic.
- We use a multi-view neural network to implement deep and broad discriminative feature learning that addresses the feature selection difficulty in malware detection via network traffic.
- Multi-group experiments are performed to evaluate different influential factors on malware detection model. Meanwhile, we compare our method with other methods and several anti-virus scanners, and the detection model has a good performance on wild malware in the application market.

The remainder of this paper is organized as follows. Related work is introduced in Section II. Section III introduces the methodology of our malware detection method in detail. The experiment and evaluation of our method are discussed in Section IV. Section V concludes the paper.

II. Related Work

At present, most malware detection methods based on network traffic use machine learning algorithms to build their detection models. The general processes of these methods include analyzing a large number of traffic data, selecting effective features that can distinguish between benign and malicious or different malware families, and then training malware classification models using machine learning algorithms. The works [7–9] all apply machine learning algorithms with traffic data to implement malware detection. In general, most malware identification methods based on network traffic and machine learning algorithms are excessively feature-dependent. These features may be specific traffic fields, static signatures, and statistics characteristics. However, identifying these effective features from network traffic is extremely difficult.

Deep learning is a branch of machine learning, which is a set of algorithms. The biggest advantage of deep learning is that it replaces handcrafted feature selection with the use of effective algorithms. One previous work [10] described a new idea for traffic identification and shows that each payload byte in a TCP session is in the range of 0 to 255, which is consistent with the range of each pixel in one picture. The TCP session is presented as a picture and each byte as a pixel. A convolutional neural network (CNN) is used in [11] to extract the abstract feature representations of HTTP headers and thus map the traffic to the app that generated it. In addition, [12] uses deep belief networks to generate invariable compact representation for malware behavior, thereby potentially identifying most variants of existing malware effectively.

Deep learning is effective in selecting features, and many malware detection studies based on deep learning have achieved excellent performance. In this paper, we use a multi-view neural network to mine the multi-level features in URLs and further identify malware through malicious URLs. This structure focuses on the depth and the width of data. Therefore, in theory, this model has strong feature selection and extraction capabilities, and our experimental results confirm this assumption.

III. Methodology

We have designed an Android traffic collection platform, from which we can obtain actual network traffic generated by apps at network access point. A multi-view neural network is used to automatically discover discriminative features of malware and benign apps from multiple views of URLs.

A. Traffic Collection

The traffic collection platform comprises three components, i.e., the control center, traffic acquisition module, and app & traffic storage module. The three components communicate with one another by a LAN switch. The control center is used to allocate traffic acquisition tasks to the machines in the traffic acquisition module, which then collects the traffic data generated by specific apps. The app & traffic storage module is used to store apps and the network traffic data they generate. These three components work together to effectively collect Android network traffic.

Traffic acquisition module module consists of multiple machines, and every machine has several Android emulators. Each app is installed on an emulator, which is restarted to stimulate the malware into performing malicious behavior. After restarting each emulator, we also start the interface traversal program to allow each app interface to simulate human actions as much as possible. In the meantime, we collect the traffic data generated by this app in the first few minutes of operation. The last step is transferring the collected traffic to the app & traffic storage module.

B. URL Processing

1) URL Extraction: In this work, we are concerned only with the URLs in the HTTP traffic. Evidently, once an app visits a malicious URL, it may become a malware. In addition, most malware use the parameters in URLs to receive commands to further perform malicious behavior [13]. Therefore, malware detection based on URLs is effective. The first step in traffic processing is extracting the URL string from the network traffic data using the tshark command.

2) URL Segmentation: Each URL is a string with many characters. We believe that a single character cannot express valuable information. For example, extracting only one character from the host name "www.baidu.com", does not make any sense; only when it is regarded as a unit can it express a complete domain name. Then, we split each URL into different components, where every segment represents a unit that expresses certain information. We use special characters, such as "/", "&" ,":" , to divide each URL into multiple segments, each of which is considered a URL segment.

C. URL Vector Representation

Common one-hot encoding method causes serious data sparsity problems, which pose challenges for follow-up calculation. Moreover, the related information between segments and the semantic information of context are lost with this encoding method. Thus, we train low-dimensional dense vector for each segment through one-hot encoding. We use the skip-gram algorithm [14] in word2vect to train the vector representation of each segment. Skip-gram is a neural network model that allows the prediction of the likelihood that other words appear near the center word. Nearby words can be measured with a window size. If the window size is 2, then the first two input words before the center word and the two succeeding words are the nearby words of this word. Thus, the skip-gram model is given a central word to predict the context.

In our scenario, suppose the URL string is "http://example.com:8080/over/there?name=ferret&color=black", and the ordered segment set that is divided by the URL includes "http example.com:8080 over there name ferret color black". If we set the window size to 2, then the segments in the vicinity of "name" include "over","there","ferret" and "color". The training goal of the skip-gram model to obtain the maximum probability of nearby segments. The input layer of the model is the one-hot encoding for segment "name". The output layer is the probability of other segments, according to the center segment, and the connection weights from the input layer to the hidden layer is an embedding table for segment vector representation. In the learned embedding table, each row is a vector representation of one segment, whose current position is equal to "1". Figure 1 illustrates the skip-gram neural network.

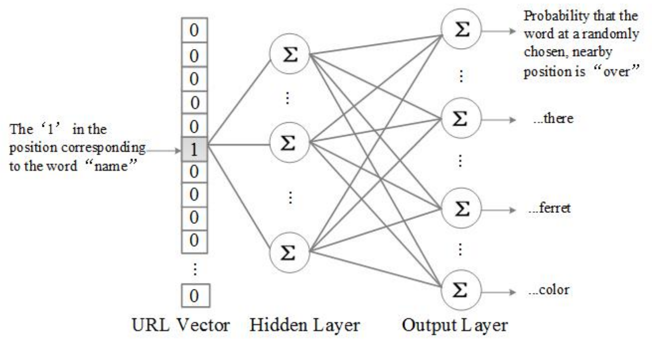

Probability that the word at a randomly chosen, nearby position is "over"
...there
...ferret
...color
The '1' in the position corresponding to the word "name"
URL Vector
Hidden Layer
Output Layer

Fig. 1: The schematic diagram of skip-gram neural network

D. Multi-view Neural Network

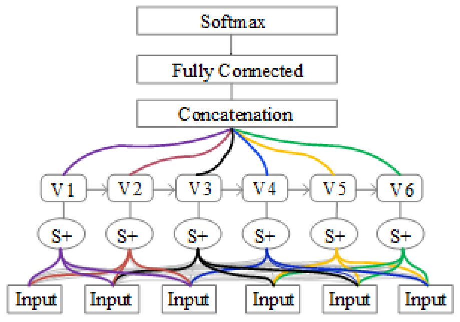


Fig. 2: The structure of used multi-view neural network

The structure of the multi-view neural network is depicted in Figure 2. We create multiple $S+$ based on the input URL vector. Each $S+$ is derived from the sum of the softmax weights of the input URL vector. Then, each $S+$ is converted to a view, and every view, except the first and last ones, is affected by all the previous views. All views are fed into a fully connected layer, followed by a softmax classifier.

1) Multiple Attentions for Selection: Each selection $S+$ is created by focusing on different subsets of the URL vectors. It is determined by the sum of the input data of the softmax weights [15]. Considering that a URL contains H segments, we represent the URL as a matrix whose shape is $H \times m$. Each row in the matrix corresponds to a URL segment, which is represented as a m-dimension vector, and is provided by a learned embedding table. The selection $S+$ for the ith view is the sum of the features of the softmax weights.

$$s_i^+ = \sum_{h=1}^{H} d_{i,h} B[h:h] \tag{1}$$

where the weights $d_{i,h}$ are computed by the following:

$$d_{i,h} = \frac{e^{m_{i,h}}}{\sum_{h=1}^{H} e^{m_{i,h}}} \tag{2}$$

$$m_{i,h} = w_i^s \tanh(W_i^s B[h:h]) \tag{3}$$

Here, w_i^s (a vector) and W_i^s (a matrix) are learned selection parameters. The selection for each view can focus on different URLs from B (URL vectors) by varying the weights $d_{i,h}$, as illustrated by the differently colored curves that connect to $S+$ in Figure 2.

2) Aggregating Selections into Views: After calculating each $S+$ for each view, the formula for calculating each view by $S+$ is introduced as follows:

$$v_1 = s_1^+; v_V = s_V^+ \tag{4}$$

$$v_i = \tanh(W_i^v([v_1; v_2; ...; v_{i-1}; s_i^+])) \quad for \quad i = 2...V-1 \tag{5}$$

where W_i^v is the parameter to be learned and [:;:] represents the concatenated multiple matrix. The first and the last views are only determined by their corresponding $S+$, but $v1$ and v_V play different roles. v_V is only determined by $S+$ and not affected by other views. The purpose of v_V is to increase the diversity of views [16]. On the contrary, $v1$ forms the basis of multiple views (from $v2$ to v_{V-1}). The previous views are stacked together and concatenated with $S+$ for the views. Thus, this structure allows each view to be aware of the information of the preceding views.

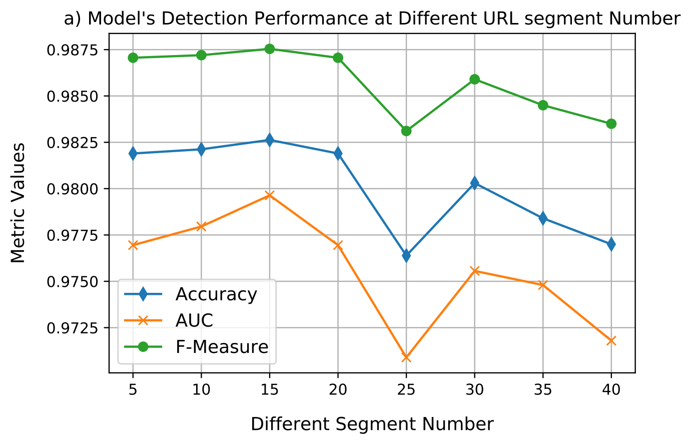
3) Classification with Views: The final step is classifying the views generated by the URL vectors. Our model first combines multiple views and feeds them into a fully connected layer, followed by a softmax classifier, which produces a probability distribution for different classes (i.e., benign and malicious). Dropout regularization [16] is used in this softmax layer. In addition, we use cross-entropy as the loss function to guide the model's training process. Adam, which is an algorithm for first-order gradient-based optimization of stochastic objective functions, is used as the specific optimization algorithm [17].

IV. Experiment and Evaluation

In this section, we elaborate on the experimental details and evaluate the performance of our model. We focus on the following four aspects:

A. Data Sets

1) ***URL Segment Selection:*** Our malicious apps are downloaded from the ViruShare website [18], which shares samples of suspicious apps to security researchers. We obtain 40,751 malicious apps that were discovered from 2014 to 2016. Our benign apps are downloaded from multiple popular app markets by an app crawler. More than 10,000 benign samples are initially obtained. Each app downloaded from the app market is sent to VirusTotal [19], which decides whether the app is malicious or not. The app is then added to our benign



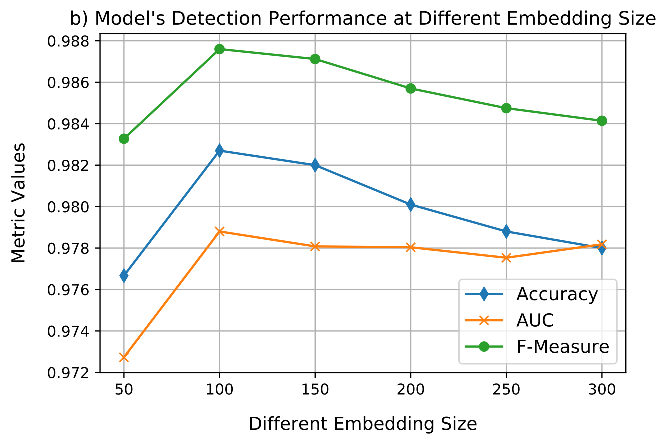


Fig. 3: The impact of segment number and embedding size on malware detection model

app set if the test result is benign. Eventually, we obtain a benign app set of 8,168 samples. This result ensures a clear boundary between the benign and malicious data sets to increase the credibility of the detection results.

Massive traffic data are collected using an automatic mobile traffic collection system (Section III-A). Finally, we obtain 18.9 GB network traffic data generated by the malware samples. However, not all traffic data generated by malware is malicious. We extract 968 MB malicious behavior traffic from this data by a domain list named “blacklist”. Similarly, we obtain 14.2 GB data generated by the benign apps. From the traffic data, 61,436 benign URLs and 27,500 malicious URLs are extracted.

B. Parameter Setting

The three parameters that affect the multi-view neural network are the URL segment number, the embedding size of each URL segment and the view number.

*1) **URL Segment Number:*** Various URLs have different lengths and can be split into varying segments. However, our multi-view neural network requires a unified input shape. Thus, we should fix a segment number H. The URL segment that exceeds the fixed number H is discarded, and that whose number is less than H is padded with a specific segment (blank space). A statistical analysis is performed on the segment number of the malicious and benign URLs and we find that the segment number of most URLs is within 40.

We vary the value of H from 5 to 40 and the interval is set to 5 to select the best segment number. We develop different models with different segment numbers, and the result is shown in Figure 3(a). The experimental result shows that the segment number only slightly affects the final detection model. In addition, the training time of the multi-view model increases with the fixed segment number H. Considering the performance and calculation time, we set H as 15.

*2) **Embedding Size Selection:*** We regard each segment as a unit and use the skip-gram algorithm to train the vector representation of each segment. We change the segment dimension from 50 to 250 and the interval to 50 to explore the ultimate effect of the vector dimension on the model. The final result of the detection model under different segment embedding sizes is shown in Figure 3(b).

The horizontal ordinate in Figure 3(b) represents different embedding sizes of each segment, and the vertical coordinate represents the accuracy rate, AUC, and F-Measure of the model at different segment embedding sizes. The experimental results show that the dimension of the segment vector only slightly influences the final training result. A long embedding size does not always improve the effect. Once the embedding size is greater than a certain value, the effect of the model shows a downward trend. We set the embedding size to 100, considering the performance of the model under different segment embedding sizes and execution efficiencies.

*3) **View Number Selection:*** We change the number of views from 1 to 10 and train 10 different models to explore the influence of view number on the final model. The result indicates that predictive accuracy, AUC, and F-Measure can be improved by increasing the number of views. However, not too many views are required to achieve optimal performance on the task. In this work, we set the view number is 6. For different application scenario, the view number of the multi-view neural network should be tuned.

C. Comparison with Other Methods

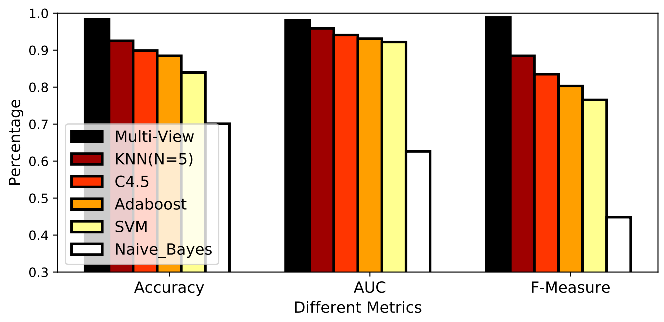


Fig. 4: The detection performance comparison of different machine learning algorithms

*1) **Comparison with Other Machine Learning Methods:*** Here, we compare the detection performance of the proposed multi-view neural network with that of popular machine learning algorithms. We select SVM, KNN (N=5), C4.5, Adaboost, and naive Bayes. We attempt to use multiple sets of parameters to maximize the performance of each algorithm. The final results of the different algorithms are shown in descending order in Figure 4. The best classifier among the classic machine learning algorithms is KNN (N=5). Compared with them, the multi-view model performs the best in terms of accuracy, AUC, and F-Measure, thereby proving its capability for automatic feature selection.

*2) **Comparison with Statistical Feature-based Method:*** In work [20], the author analyzes a large number of benign and malicious traffic data and designs six TCP statistical features to distinguish between benign and malicious traffic. The specific features and their descriptions are shown in Table I.

We extract six features and finally obtain 64,423 benign and 31,859 malicious feature samples and train an effective detection model that are based on these samples using our traffic data and the C4.5 algorithm. The F-Measure of model is only 80.15% which is lower than multi-view model. We conclude that the features that the multi-view neural network learned are more distinguishable than those learned by hand-crafted statistical features. The method based on TCP flow statistical features performs poorly because the author focuses only on the traffic data and the features cannot effectively represent other traffic characters. This result also shows that the features automatically selected by the multi-view neural network provide remarkable flexibility. Regardless of the data set, we can directly feed the data to the multi-view neural network for automatic feature selection after pretreatment.

TABLE I: Network-level statistical features that can distinguish between benign and malicious traffic

Id	Feature Description
1	Uploading bytes(client->server)
2	Downloading bytes(server->client)
3	Total upload packet number in a session(client->server)
4	Total download packet number in a session(server->client)
5	Average bytes of upload packets(client->server)
6	Average bytes of download packets(server->client)

*3) **Comparison with Flow header-based Method:*** Our model focuses only on URL strings in HTTP traffic; likewise, many previous works also focus on HTTP traffic for malware detection. For example, the work in [20] develops another malware detection model that is based on the traffic data, i.e., network-level attribute field features that can distinguish between benign and malicious traffic in the HTTP request header. Attribute field features include the host, request method, URL path, and user-agent. Different fields are preprocessed with different methods. Then, the SVM algorithm is used to create the detection model.

On our traffic data, we follow the steps in [20] and extract 61,359 benign and 27,498 malicious samples. We also create a classification model using the SVM algorithm. The F-Measure of this model is 95.74% which is better than that of TCP flow statistic features but cannot surpass that of our method. This experimental result indicates that our method can achieve an acceptable performance by using only URLs and without heavy feature engineering.

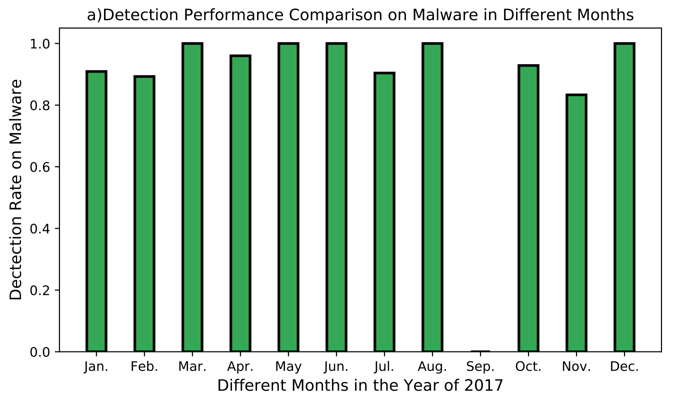
D. Detection of Malware over Different Periods

To bypass anti-virus scanners, attackers may attempt to produce malware variants to poison and cheat the detection model. To evaluate the effectiveness of our model on new malware samples, we download 430 malware that were newly discovered in 2017 from VirusShare and divide the samples by month. A total of 83.5 GB network traffic from the malware samples are collected. We extract 3890, 3421, 179, 892, 418, 1390, 1612, 27, 0, 619, 418, and 321 URLs from the traffic data generated by the malware from January to December 2017. The trained model is applied to detect these URLs, and the detection rates on the malware from different months are shown in Figure 5(a).

Our model achieves the best detection performance (100%) on malware from March, May, June, August, and December but discovers merely 83.33% of the malware samples from November. The result shows that the time characteristic of malware affects the models detection performance, but our model can effectively detect most new malware which proves its time-based effectiveness.

E. Detection of Malware in the Wild

We use some new apps downloaded from Android app markets to verify its effectiveness in the wild. These apps have no intersection with those apps used in the training process. The traffic data generated by these wild apps are obtained from the traffic generation and collection platform. These traffic data are processed (i.e., URL extraction, segmentation, and vector representation) and then fed into the multi-view neural network. Each URL label (i.e., malicious or benign) is determined using the trained detection model. An app is deemed to be malware if it visits a malicious website though a URL. In the wild app set of 337 apps, 242 are confirmed malicious apps by the detection report of VirusTotal. These 242 malicious apps are filtered by approximately 60 anti-virus scanners in VirusTotal. However, each scanner in VirusTotal can detect only a portion of these malware samples. We compare the performance of our model against ten selected anti-virus scanners, i.e., ESET-NODE32, Avira, Sophos, McAfee, BitDefender (abbreviated as BitD in Figure 5(b)), Kaspersky, AVG, F-Secure, ClamAV, and Pandas. Figure 5(b) shows the detection performance of each scanner in descending order. The best scanner is ESET-NODE32, which can detect 56.20% of malware, whereas Pandas discovers 0 malware. Our method follows ESET-NODE32 with a detection rate of 50% and outperforms nine other anti-virus scanners. The reason for ESET-NODE32 has a better performance is that it integrates dynamic behavior detection, UTFI scanning and other technologies that are more comprehensive than ours.



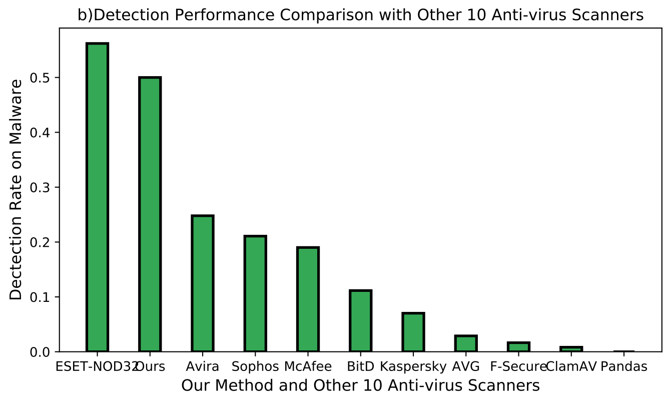


Fig. 5: The detection performance comparison in different months of 2017 and other 10 anti-virus scanners

V. Conclusion

We present an approach to detecting malware based on malicious URLs. Our method divides each URL into several segments using specific characters and then uses the skip-gram algorithm to train the embedding for each segment. This vectorization method solves the data sparsity and semantic loss problems that common encoding methods cause. We feed the URL vector into a multi-view neural network, which can automatically create multiple $S+$ using the input data and generate multiple views. The network focuses on depth but also emphasizes width and can complete the automatic selection of features from multiple views. We design a set of experiments to verify the effectiveness of our method and compare our technique with several other methods. The experimental result shows that our model performs well on the test set. In addition, we apply this model for wild malware detection and find that it has excellent detection capability.

Acknowledgment

This work was supported by the National Natural Science Foundation of China under Grants No.61672262, No.61573166, No.61472164, No.61572230, No.61702218 and No.61702216, the Natural Science Foundation of Shandong Province under Grants No.ZR2012FM010 and No.ZR2017BF001, the Shandong Provincial Key R&D Program under Grant No.2016GGX101001, CERNET Next Generation Internet Technology Innovation Project under Grant No.NGII20160404. This work is also supported in part by NSF grant CNS-1566388.

References

[1] “Number of available applications in the google play store from december 2009 to december 2017,” https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/, Tech. Rep., 2017.
[2] “2017 internet security threat report,” https://www.symantec.com/security-center/threat-report, Tech. Rep., 2017.
[3] P. Faruki, A. Bharmal, V. Laxmi, V. Ganmoor, M. S. Gaur, M. Conti, and M. Rajarajan, “Android security: a survey of issues, malware penetration, and defenses,” *IEEE communications surveys & tutorials*, vol. 17, no. 2, pp. 998–1022, 2015.
[4] A. Moser, C. Kruegel, and E. Kirda, “Limits of static analysis for malware detection,” in *Computer security applications conference, 2007. ACSAC 2007. Twenty-third annual*. IEEE, 2007, pp. 421–430.
[5] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, “Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones,” *ACM Transactions on Computer Systems (TOCS)*, vol. 32, no. 2, p. 5, 2014.
[6] H. Guo, C. Cherry, and J. Su, “End-to-end multi-view networks for text classification,” *arXiv preprint arXiv:1704.05907*, 2017.
[7] H. Ogawa, Y. Yamaguchi, H. Shimada, H. Takakura, M. Akiyama, and T. Yagi, “Malware originated http traffic detection utilizing cluster appearance ratio,” in *Information Networking (ICOIN), 2017 International Conference on*. IEEE, 2017, pp. 248–253.
[8] B. Anderson and D. McGrew, “Identifying encrypted malware traffic with contextual flow data,” in *Proceedings of the 2016 ACM Workshop on Artificial Intelligence and Security*. ACM, 2016, pp. 35–46.
[9] M. Conti, L. V. Mancini, R. Spolaor, and N. V. Verde, “Analyzing android encrypted network traffic to identify user actions,” *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 1, pp. 114–125, 2016.
[10] Z. Wang, “The applications of deep learning on traffic identification,” *BlackHat USA*, 2015.
[11] Z. Chen, B. Yu, Y. Zhang, J. Zhang, and J. Xu, “Automatic mobile application traffic identification by convolutional neural networks,” in *Trustcom/bigdatase/i?spa*, 2017, pp. 301–307.
[12] O. E. David and N. S. Netanyahu, “Deepsign: Deep learning for automatic malware signature generation and classification,” in *International Joint Conference on Neural Networks*, 2015, pp. 1–8.
[13] Specification of malicious url 2013. [online]. Available: http://www.antiy.net/p/specification-of-malicious-url.
[14] A. Fonarev, O. Hrinchuk, G. Gusev, P. Serdyukov, and I. Oseledets, “Riemannian optimization for skip-gram negative sampling,” pp. 2028–2036, 2017.
[15] D. Bahdanau, K. Cho, and Y. Bengio, “Neural machine translation by jointly learning to align and translate,” *arXiv preprint arXiv:1409.0473*, 2014.
[16] H. Guo and H. L. Viktor, “Multirelational classification: a multiple view approach,” *Knowledge and Information Systems*, vol. 17, no. 3, pp. 287–312, 2008.
[17] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.
[18] “Virusshare.” [Online]. Available: https://virusshare.com/
[19] “Virustotal.” [Online]. Available: https://www.virustotal.com/
[20] S. Wang, Z. Chen, L. Zhang, Q. Yan, B. Yang, L. Peng, and Z. Jia, “Trafficav: An effective and explainable detection of mobile malware behavior using network traffic,” in *Quality of Service (IWQoS), 2016 IEEE/ACM 24th International Symposium on*. IEEE, 2016, pp. 1–6.