Information Sciences xxx (xxxx) xxx

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Deep and broad URL feature mining for android malware detection

Shanshan Wang[a], Zhenxiang Chen[a,*], Qiben Yan[b], Ke Ji[a], Lizhi Peng[a], Bo Yang[a], Mauro Conti[c]

[a] *Shandong Provincial Key Laboratory of Network Based Intelligent Computing, University of Jinan, Jinan, China*
[b] *Department of Computer Science and Engineering, Michigan State University, East Lansing, MI, USA*
[c] *Department of Mathematics, University of Padova, Padua, Italy*

A R T I C L E I N F O

Article history:
Received 18 July 2018
Revised 4 October 2019
Accepted 11 November 2019
Available online xxx

Keywords:
Android malware detection
Multi-view neural network
URL feature mining

A B S T R A C T

In recent years, the scale and diversity of malicious software on mobile networks have grown significantly, thereby causing considerable danger to users' property and personal privacy. In this study, we propose a malware detection method that uses the URLs visited by apps to identify malware. 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 inputs automatically and distribute soft attention weights to focus on different features of inputs. Multiple views preserve rich semantic information from inputs 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 achieves robust and timely malware detection. 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.

1. Introduction

With the advent of the mobile Internet era, the popularity of mobile devices has grown significantly. A recent report[1] shows that the number of apps in Google Play Store has been rapidly increasing since December 2009.

However, malicious software or malware has become a major threat to the growing mobile ecosystem. Recently, the quantity and diversity of mobile malware, particularly those targeting Android platforms, have increased dramatically.[2] To make things worse, multi-million dollar virtual bank heists and other extraordinary attacks pose a serious threat to mobile security. Although the Android platform and mobile anti-virus scanners provide security protection mechanisms to Android devices, an increasing number of advanced mobile malware can still penetrate the mobile system by evading these mechanisms. Moreover, the mobile devices are increasingly associated with personal property and privacy information [13,23],

* Corresponding author.
E-mail address: czx@ujn.edu.cn (Z. Chen).
[1] https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/.
[2] https://www.symantec.com/security-center/threat-report.

https://doi.org/10.1016/j.ins.2019.11.008
0020-0255/© 2019 Elsevier Inc. All rights reserved.

which may pose multiple threats to users if compromised. Thus, effective mobile malware detection systems are urgently needed.

A recent survey [10] 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 analysis methods. Numerous previous works use static analysis to detect privacy leakage, malware, and vulnerabilities in Android apps [3,11,20]. However, static analysis is challenged by the code polymorphism and obfuscation of malware [21]. 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 [9]. These techniques are effective [8] 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 and detect malware by their suspicious network behaviors. This type of malware detection approach is promising because most malware executes their malicious behaviors through networks [37]. When executing malicious acts, the malware must communicate with a remote server via the Internet. Specific malware can be easily found through these clues [36]. Moreover, malware detection methods based on network behaviors can be more easily implemented and deployed than dynamic analysis methods. For example, traffic-based detection methods can be deployed at an access point or gateway. These methods only require user-generated network traffic data and do not consume any of the users' mobile resources. Also, these methods do not require user operations other than the granting of permission for the detection service. The core of network traffic-based methods is to find distinctive features for efficient malware classification. However, feature selection is notoriously difficult. Prior work in this area explored the use of statistical features and network traffic contents for understanding the operation mechanism of malware, which is indeed time-consuming and labor-intensive. Therefore, we design a malware detection method that is based on malicious URLs. Our method executes URL segmentation and vectorization operations and does not require a time-consuming feature selection process. Particularly, we use a novel multi-view neural network [14] to achieve the automatic feature selection. This neural network focuses on width and depth and can 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, our key technical contributions can be summarized as follows:

- We perform text-like segmentation and vectorization on the URLs in network traffic. We develop a novel preprocessing method on traffic data which is instructive to malware detection using network traffic.
- We use a multi-view neural network to implement deep and broad feature learning. The novel neural network can produce discriminative features and address the feature selection difficulty in network traffic based malware detection methods.
- 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. In the confirmed 242 malicious wild apps, 90% of scanners in VirusTotal can detect no more than 61 malware samples. On the contrary, our detection model can identify 121 malware samples.

In this paper (which is an extended version of the work in [28]), we add multiple groups of experiments in evaluation section. First, we evaluate the contribution of each view in the multi-view structure to the final detection result. Second, we design two variants of multi-view neural network by changing the way each view connecting with each selector $S+$. And we compare the malware detection performance of multi-view and its two variants. The experimental results confirm the rationality of the multi-view structure. Third, we use a variety of malware variant generation methods to process more than 2000 malware, and obtain different types of malware variants of these malware. At the same time, we use the trained multi-view model to detect the malware and their variants. The experimental results confirm that the multi-view model has the ability to detect malware variants. In addition, we update the related work section to introduce recent works on malware detection.

The remainder of this paper is organized as follows. Related work is introduced in Section 2. Section 3 introduces the methodology of our malware detection method in detail. The experiment and evaluation of our method are discussed in Section 4. Section 5 describes certain limitations of this approach, and Section 6 concludes the paper.

2. Related work

The Android platform uses several security mechanisms, such as permission mechanisms, to prevent target devices from being infected [26,27,34]. However, this permission-based protection requires users to have sufficient security awareness. This over-reliance on the user has limitations, which allow mobile malware to sneak into users' devices. Several anti-virus scanners can be used to uphold mobile device security. Most of such scanners decide whether an app is malicious or not by analyzing certain key security elements, such as permissions, called sensitive APIs and malicious code segments [21]. These anti-virus scanners protect devices to a certain degree. However, malicious apps are constantly evolving, and their quantity and variety continue to grow. Thus, new and effective malware detection methods need to be explored. Our method can identify multiple types of malware variants. To detect them, we only need to expand the data set without adapting our method.

At present, most malware detection methods use machine learning algorithms to build their detection models. The general process of these methods includes analyzing a large number of malware samples, selecting effective features that can distinguish between benign and malicious apps and then trainingmalware classification models using machine learning algorithms. For example, the Drebin project [2] extracts several sensitive APIs and permissions from multiple malware families. These sensitive APIs and permissions are regarded as features and then combined with a support vector machine (SVM) learning algorithm to create a multi-class classification model for different malware families. This method has high accuracy and a low false positive rate. However, this technique also has shortcomings. For example, this detection method is implemented on rooted mobile devices, which consumes significant resources. On the contrary, our method detects on the server side. The mobile side setup is very lightweight and occupies negligible system resources. There are several other works that use machine learning for malware detection, such as [1,6,18,22,32,33]. In [32,33], the authors extract a large number of statistical features and strings from APK files. Based on these features, machine learning algorithms are used to create multiple ensemble machine learning models to detect malicious apps.

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 data is extremely difficult. The feature selection process requires researchers to analyze a large amount of traffic data, which is time consuming. Our method can automatically select important features without manual selection.

Deep learning is a branch of machine learning, which involves a set of algorithms. The biggest advantage of deep learning is that it replaces handcrafted feature selection with the use of effective algorithms for supervised or unsupervised feature learning and hierarchical feature extraction. At present, an increasing number of studies are beginning to apply deep learning to malware detection. One prior work [35] describes a new approach 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. Deep learning algorithms are used to classify images for traffic classification.

A convolutional neural network (CNN) is used in [5] to extract abstract feature representations of HTTP headers and thus map the traffic to the app that generated it. The use of CNN saves time for feature selection and achieves a highly accurate traffic identification performance. In addition, DeepSign [7] 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 [19]. Therefore, the data has to be explored more deeply and widely than before. Portraying network traffic from different perspectives leads to a more accurate understanding than that from a single perspective. 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. We generate multiple views of a URL dataset, and these different views focus on different aspects of the dataset by different weights. The multi-view neural network focuses on the depth and the width, which makes the model suitable for feature selection and extraction. Our experimental results confirm the superior performance of the multi-view neural network.

3. Methodology

We use an Android traffic collection platform to obtain mobile traffic when the apps connect to the internet. With this traffic collection platform, we can obtain actual network traffic generated by apps at the network access point. We focus on the URLs to determine whether the app is malicious or not through the network traffic analysis. We develop a multi-view neural network to automatically discover distinguishable features of malware and benign apps from multiple views of input data. This process is illustrated in Fig. 1 and outlined as follows.

3.1. Traffic collection

The traffic collection platform is used to collect the malicious and benign traffic data generated by malware and benign apps, respectively. A firewall and NAT server are deployed in the gateway to ensure the security of the traffic collection platform. Fig. 1 illustrates our approach, including the traffic collection platform and the malware detection model training.

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 each other via 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. These three components work together to collect Android network traffic effectively.

Here, we provide a detailed explanation of the traffic acquisition module. The 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. The work in [37] suggests that the most common activation mechanism of malware is the restart operation after app installation. 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

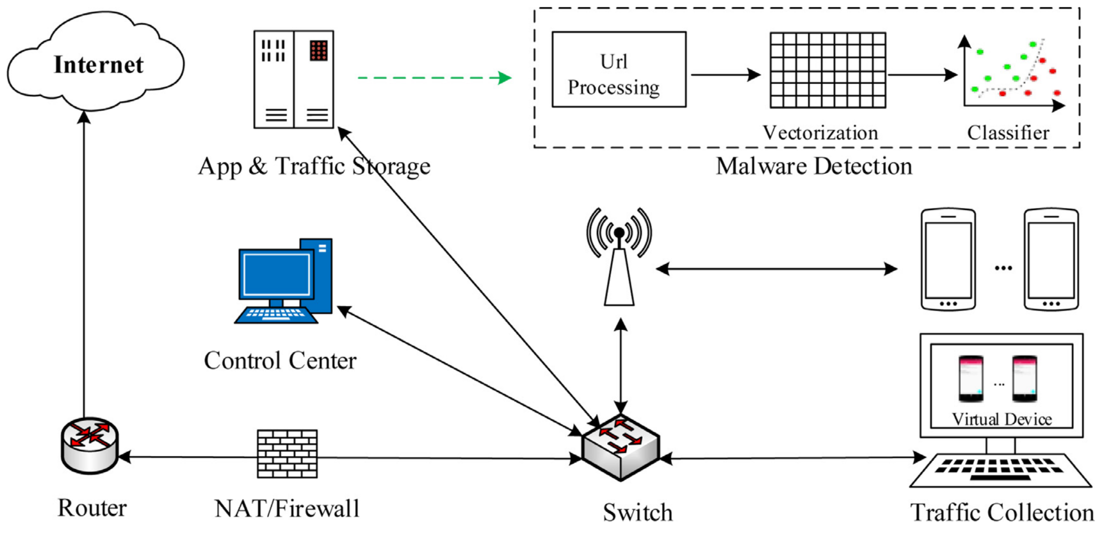

Fig. 1. The framework of traffic collection platform and malware detection model training.

data generated by this app in the first few minutes of operation. The last step is to transfer the collected traffic to the app & traffic storage module.

3.2. URL Processing

In this work, we are concerned only with the URLs in the HTTP traffic. Evidently, once an app visits a malicious URL, it may be infected as malware. In addition, most malware use the parameters in URLs to receive commands to further perform malicious behavior[3]. 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 "tshark -r *filePath* -T fields -e http.host -e http.request.uri -E header=y -E separator=, -E quote=d -E occurrence=f", where *filePath* is the path of each traffic data file. This command can extract the HTTP request host and URL path, and the two components can be concatenated into a complete URL.

Each URL is a string with many characters. We believe that a single character cannot express rich 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. For example, a complete URL string "http://www.example.com:8080/over/there?name=ferret&color=black" can be divided into an ordered set of URL segments, which includes "http example.com:8080 over there name ferret color black".

3.3. URL vector representation

If each segment is encoded directly by one-hot encoding, then the vector length of each segment becomes equal to the segment number that appear in the URLs given that a URL can be divided into several segments. This 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 vectors for each segment through one-hot encoding. We use the skip-gram algorithm [12] to train the vector representation of each segment. Skip-gram is a neural network model that can predict the likelihood of other words appearing 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://www.example.com:8080/overthere?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 is 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

[3] http://www.antiy.net/p/specification-of-malicious-url.

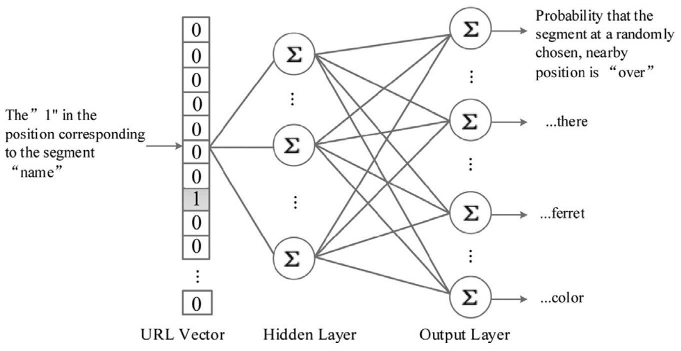
The" 1" in the position corresponding to the segment "name"
Probability that the segment at a randomly chosen, nearby position is "over"
...there
...ferret
...color
URL Vector
Hidden Layer
Output Layer

Fig. 2. The schematic diagram of skip-gram neural network.

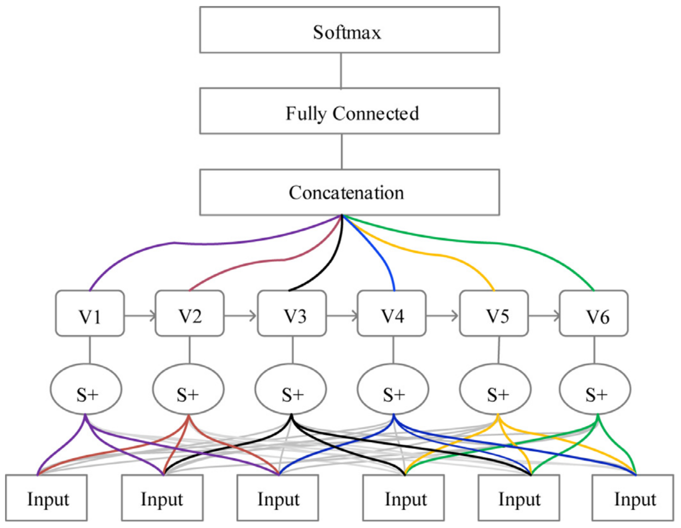

Fig. 3. The structure of the adopted multi-view neural network.

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". Fig. 2 illustrates the skip-gram neural network.

3.4. Multi-view neural network

The structure of the multi-view neural network is depicted in Fig. 3. First, we copy multiple copies of the same URL input and create multiple selectors $S+$. 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.

3.4.1. Multiple attentions for selection

Each selector $S+$ 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 [4]. Because a URL contains H segments, and every segment is represented by a d-dimensional vector, which is provided by a learned embedding table, the URL is represented as a matrix whose shape is $H \times d$. The selector $S+$ for the ith view is the sum of the features of the softmax weights. The formula is as follows:

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

where the B[h:h] matrix represents matrix of every partial words of the URL and the weights $d_{i,h}$ are computed by the following:

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

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

where W_i^s is a parameter matrix that we set, and w_i^s represents each vector of the matrix, which is the parameter that needs to be learned. The selection for each view can focus on different URLs from B by varying the weights $d_{i,h}$, as illustrated by the differently colored curves that connect to $S+$ in Fig. 3.

3.4.2. Aggregating selector into views

After calculating each $S+$ for each view, the formula for calculating each view by $S+$ is introduced as follows:

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

$$v_i = \tanh\left(W_i^v\left(\left[v_1; v_2; \ldots; v_{i-1}; s_i^+\right]\right)\right), 2 <= i <= V - 1, \tag{5}$$

where W_i^v is the parameter that needs to be learned. The matrix of [:;:] represents the concatenated multiple matrix. The content of v_i includes the view of the front $i - 1$ and the ith $S+$. The first and the last views are only determined by their corresponding $S+$. Because v_1 is in the first place, there is no other view in front to provide information. The v_V is also only determined by $S+$ and not affected by other views. The purpose of v_V is to increase the diversity of views [15,24,31]. On the contrary, v_1 forms the basis of multiple views (from v_2 to v_{V-1}). The previous views are stacked together and concatenated with $S+$ for the views. Thus, the subsequent views contain information from the previous views. This structure allows each view to be aware of the information of the preceding views. In addition, W_i^v is applied on the current view and therefore grows with each view.

3.4.3. Classification with views

The final step is to classify 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 [15] is used in this softmax layer, as in [16]. 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].

4. Experiment and evaluation

In this section, we elaborate on the experimental details and evaluate the performance of our model from multiple aspects.

4.1. Data sets

Our malicious apps are downloaded from the ViruShare website,[4] 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. We add a process to select benign samples from these downloaded samples, considering the large number of potential malicious apps in third-party app markets. Each app downloaded from the app market is sent to VirusTotal,[5] which decides whether the app is malicious or not. The app is then added to our benign app set if the test result is benign. Eventually, we obtain a benign app set of 8168 samples. This result ensures a clear boundary between the benign and malicious data sets to increase the credibility of the detection results. This procedure can also be used to expand our malicious data set because the suspicious apps can be included in the malware set.

We collect large-scale traffic data to evaluate the detection performance of our method. Massive traffic data is collected using an automatic mobile traffic collection system (Section 3.1). Finally, we obtain 18.9 GB network traffic data generated by the malware samples. However, not all traffic data generated by malware is malicious. Malware is sometimes represented as a repackaged app by adding malicious code segments to a benign app. Thus, the traffic data is a mixture of benign and malicious traffic. 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. Then, the network traffic is processed to extract URLs. Ultimately, 61,436 benign URLs and 27,500 malicious URLs are extracted. The two types of labeled URLs are used to train the detection model and to test the detection performance of our method.

4.2. Parameter setting

The two parameters that affect the input dimension of the multi-view neural network are the URL segment number and the embedding size of each URL segment. In this section, we use three metrics to evaluate the model and obtain the best parameters of the input dimension.

[4] https://virusshare.com.
[5] https://www.virustotal.com.

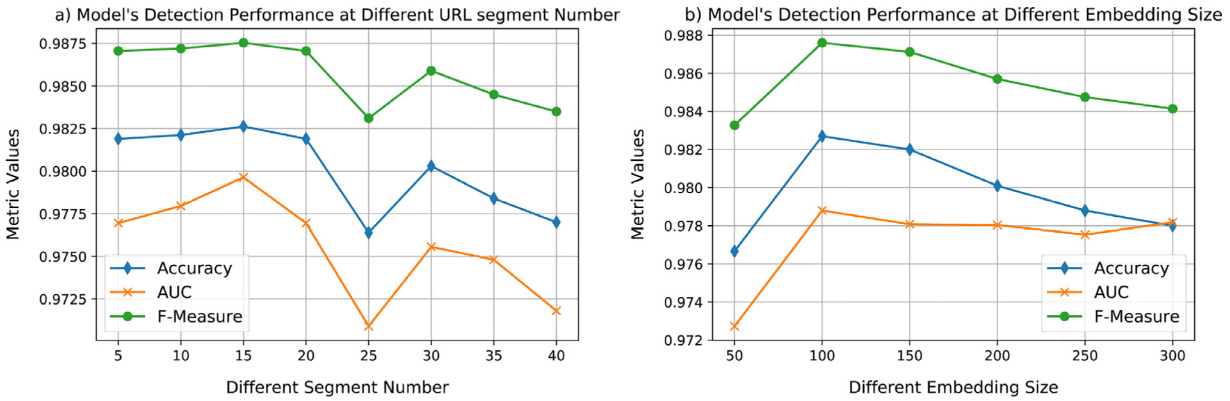

Fig. 4. The impact of segment number and embedding size on malware detection model.

Accuracy refers to the proportion of samples that are accurately predicted by the detection model (e.g., benign and malicious samples). The formula of accuracy is as follows:

$$Accuracy = (TP + TN)/(TP + FN + FP + TN).$$

F-Measure is the harmonic average of the precision and recall, which are used to comprehensively reflect all indicators. The formula of the F-Measure is as follows:

$$F - Measure = \frac{2 * Precision * Recall}{Precision + Recall}.$$

The area under the curve (AUC) is the area under the receiver operating characteristic (ROC) curve, which is a 2D graphical illustration of the tradeoff between the true positive rate (TPR) and its false counterpart (FPR).

4.2.1. URL segment selection

Various URLs have different lengths and can be split into 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.

The horizontal cordinate in the figure represents different URL segment numbers, and the vertical coordinate is the proportion of the segment number in this type of URLs. Dotted lines represent the segment number distribution of the malicious URLs, and the solid line represents the segment number distribution of the benign URLs.

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 Fig. 4(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.

4.2.2. 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 Fig. 4(b).

The horizontal cordinate in Fig. 4(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. The model performs best when the embedding size of each segment is 15. 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.

4.2.3. 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.

In Fig. 5, we present the accuracy, AUC, and F-Measure obtained from the dataset while varying the number of views in our multi-view neural network. The results indicate that predictive accuracy, AUC, and F-Measure can be improved by increasing the number of views. The accuracy, AUC, and F-Measure are remarkably low when the number of views is 1 and increase sharply when the number of views is 2. When the number of views varies from 2 to 10, the accuracy, AUC, and

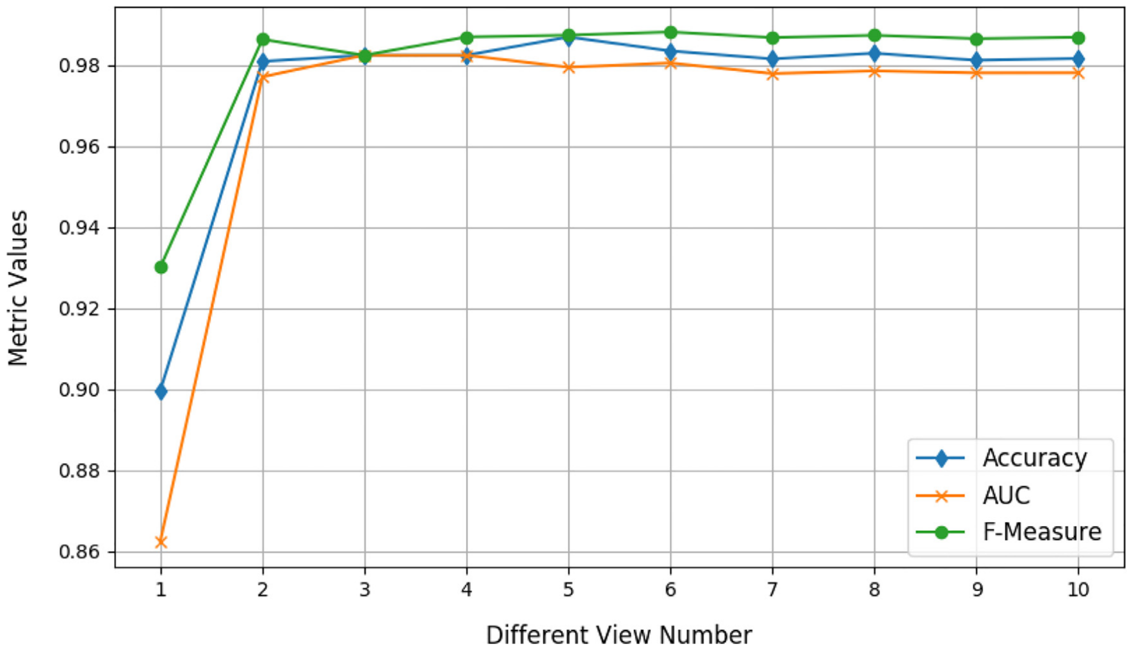

Fig. 5. Accuracy, AUC and F-Measure obtained by varying the number of views.

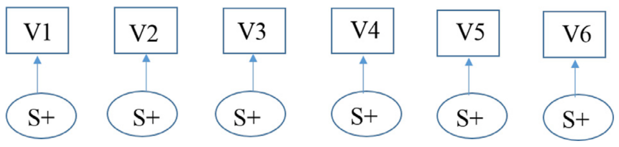

Fig. 6. The connection form between $S+$ and *View* of Multi-view variant 1.

F-Measure show no evident increment. Finally, we select 6 as the view number to obtain the final detection model. The view number of the multi-view neural network should be tuned for each new application scenario, but not too many views are required to achieve optimal performance on the task. An adequate number of views can produce an acceptable detection model.

4.2.4. Contribution of each view

To understand the benefits of the multi-view method, we further analyze the six views constructed by our best model. After training, we obtain the view representation vectors for the training data and then independently train a classifier for each view. We report the accuracy, AUC, and F-Measures for each view in Fig. 9(a). The figure shows that different views present different contributions to the final detection result. v_1, which is only decided by its corresponding S^+. No information for v_1 is available; thus, its performance is the worst. Each view from v_2 to v_5 is aware of the information of all preceding views; thus, their performance is better than that of v_1. v_6 is also determined by S^+ and thus increases the diversity of views; however, its performance is better than that of v_1.

4.3. Two variants of multi-view neural network

According to the connection forms of the selector S+ and view, we design two variants of multi-view neural networks, which simplify the structure complexity of multi-view. We call the two multi-view variants as multi-view-1 and multi-view-2. Here, we first explain the difference between two variants and the initial multi-view. And then, we compare the performance of these three structures.

Multi-view-1: Different from the initial multi-view neural network, each view in multi-view-1 is only related to their current S+, and is not affected by other views. The connection between $s+$ and *view* is shown in the Fig. 6. The formula for calculating each view by $S+$ is introduced as follows:

$$v_i = s_i^+ \quad for \quad i = 1, 2, \ldots, V. \tag{6}$$

Multi-view-2: Different from the initial multi-view neural network, each view (except the v_1) in multi-view-2 is not only affected by the current $S+$, but also affected by a view in front of it rather than all views in front of it. The connection

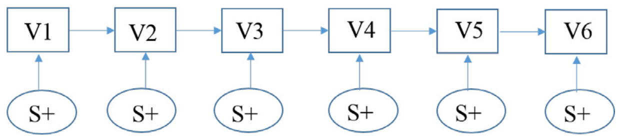

Fig. 7. The connection form between *S*+ and *view* of Multi-view variant 2.

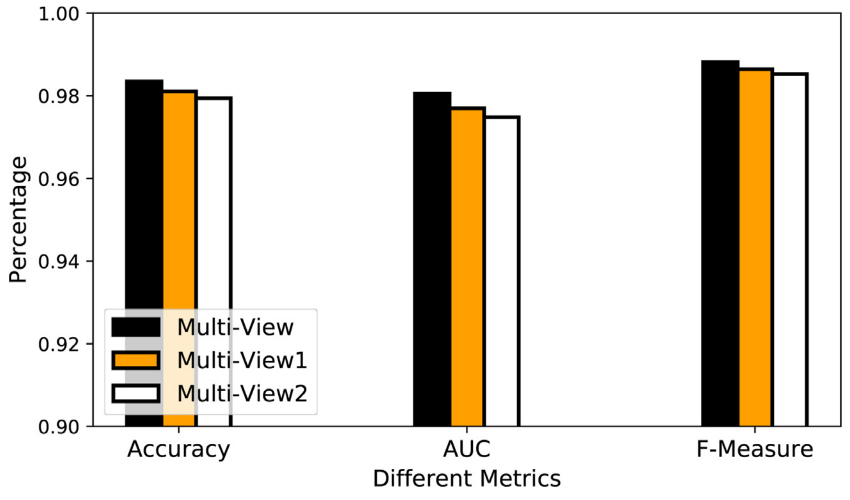

Fig. 8. Performance comparison of different views and machine learning algorithms.

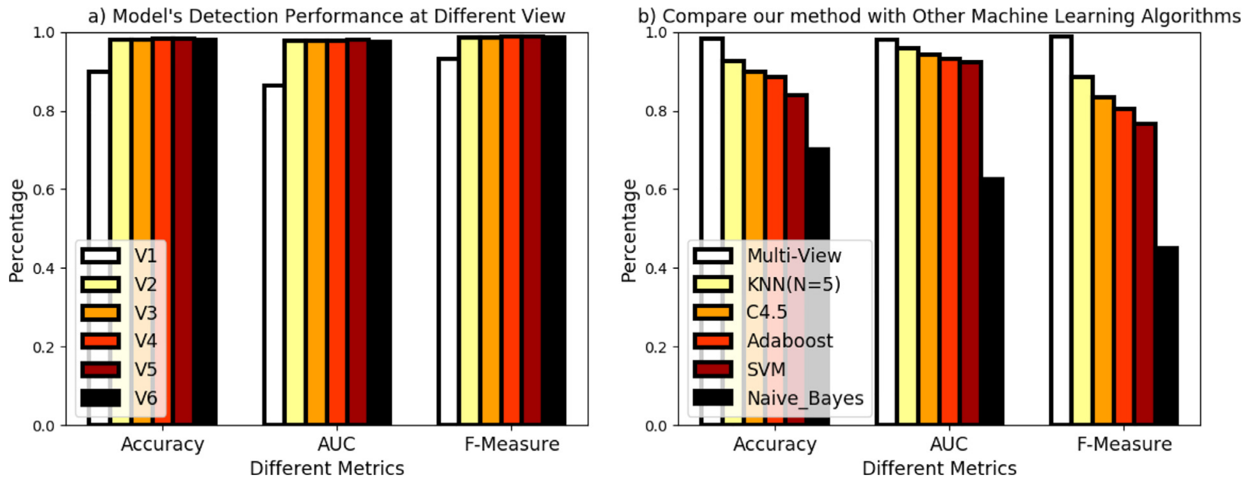

Fig. 9. Performance comparison of different views and machine learning algorithms.

between *s*+ and *view* is shown in the Fig. 7. The formula for calculating each view by *S*+ is introduced as follows:

$$v_i = \tanh\left(W_i^v\left(\left[v_1; s_i^+\right]\right)\right) \quad for \quad i = 2\ldots V-1. \tag{7}$$

Except for the different connection forms of *s* and view, other parameters are consistent with the initial multi-view. We evaluate the performance of the three neural network structures on the same data set. The Accuracy, AUC and F-measure values of 3 structures are shown in Fig. 8. In fact, there are only slight differences in the performance of the three models. Among them, the best performance is the initial multi-view, followed by the multi-view-1, the worst is the multi-view-2. The result also shows that the initial multi-view structure is more reasonable and can preserve more information about URL data.

Table 1
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)

Table 2
Comparison of our method with other traffic-based methods.

Method	Accuracy	AUC	F-Measure
Ours	0.9835	0.9805	0.9885
Statistical Feature-based	0.6688	0.4998	0.8015
Flow header Feature-based	0.9565	0.9612	0.9574

4.4. *Comparison with other methods*

4.4.1. *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. We find that when the number of neighbors is equal to 5 (N = 5), the KNN algorithms achieves the best performance. The final results of the different algorithms are shown in descending order in Fig. 9(b). The experimental results show that selecting features is difficult when the URL vector is trained by these classic machine learning algorithms directly. Compared with other algorithms, the multi-view model performs the best in terms of accuracy, AUC, and F-Measure. The best classifier among the classic machine learning algorithms is KNN (N = 5). However, our multi-view model has better performance than KNN, thereby proving its capability for automatic feature selection.

4.4.2. *Comparison with handcrafted statistical feature-based method*

In the related work section, we introduce several malware detection methods that are based on the statistical features of traffic flows. For example, in [29], 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 1.

We extract these 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 performance of this model is shown in Table 2. We conclude that the features that the multi-view neural network learned are more distinguishable than those learned by handcrafted statistical features. The method based on statistical features performs poorly because these features have poor generalization capability. 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.

4.4.3. *Comparison with handcrafted flow header feature-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 [29] 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 [29] and extract 61,359 benign and 27,498 malicious samples. We also create a classification model using the SVM algorithm, whose performance is shown in Table 2. We found that its performance is better than 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 without carrying out heavy feature engineering.

4.5. *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 use collected traffic data (dated from 2014 to 2016) to train a detection model and test it on a new malware dataset collected from VirusShare in 2017. We download

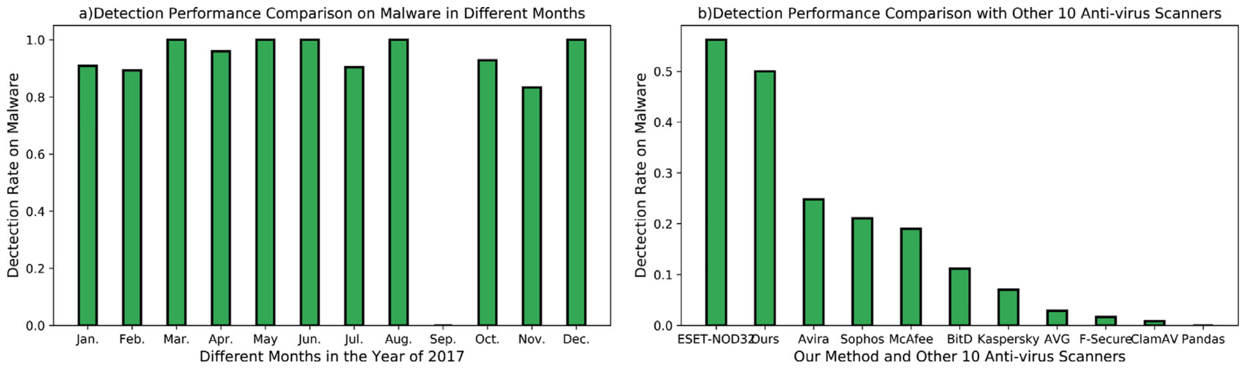

Fig. 10. Performance comparison in different months of 2017 and other 10 anti-virus scanners .

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. There is no samples found in September of 2017. The trained model is applied to detect these URLs, and the detection rates on the malware from different months are shown in Fig. 10(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 evolving characteristic of malware affects the model's detection performance, but our model can effectively detect most new malware. Surprisingly, the experimental results show that our model trained by the malware samples from 2014 to 2016 is also effective in detecting new malware discovered in 2017.

4.6. *Detection of malware varieties*

In this section, we evaluate the ability of multi-view neural networks to identify different types of malware variants. We downloaded 3426 malware samples from virusShare website. After that, we performed different operations of renaming identifier , inserting non-trivial junk instructions, reversing bytecode order, and reordering instructions on these malware samples [25]. The number of valid malware variants obtained after different operations are 697, 1089, 375, and 1061, respectively. These malware variants are all used as testing dataset, and there is no intersection between testing samples and training samples. We then used the Android network traffic collection platform to obtain their network traffic data. The original malware generated 1.79GB network traffic data, while the network traffic sizes of different types of malware variants were 183 MB, 611 MB, 199 MB and 231 MB respectively.

We use the trained multi-view neural network model to detect these initial malware and their malware variants. The results of the model for initial malware and different types of variants malware are shown in Fig. 11. From the figure, the model has the best detection effect on the initial malware detection, and the detection rate reaches 98.81%. At the same time, the model has a good recognition ability for different malware variant. Regardless of malware variants from the operations of renaming identifier , inserting non-trivial junk instructions, reversing bytecode order, and reordering instructions on these malware samples, the detection rates of our model exceed 89%, and the worst case is for detecting rename classes variant, in which the detection rate reaches 89.30%.

4.7. *Detection of malware in the wild*

We use some new apps downloaded from Android app markets to verify the effectiveness of our detection model. These apps have no intersection with those apps used in the training process. The traffic data generated by these wild apps are obtained from 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. On the contrary, our detection model can identify 121 out of 242 apps, thereby verifying its capability in examining wild apps. 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 Fig. 10(b)), Kaspersky, AVG, F-Secure, ClamAV, and Pandas, according to [2]. The detection rates of the scanners are sourced from VirusTotal and vary considerably. Fig. 10(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. For this wild app set, our method follows ESET-NODE32 with a detection rate of 50% and therefore outperforms nine other anti-virus scanners. The reason

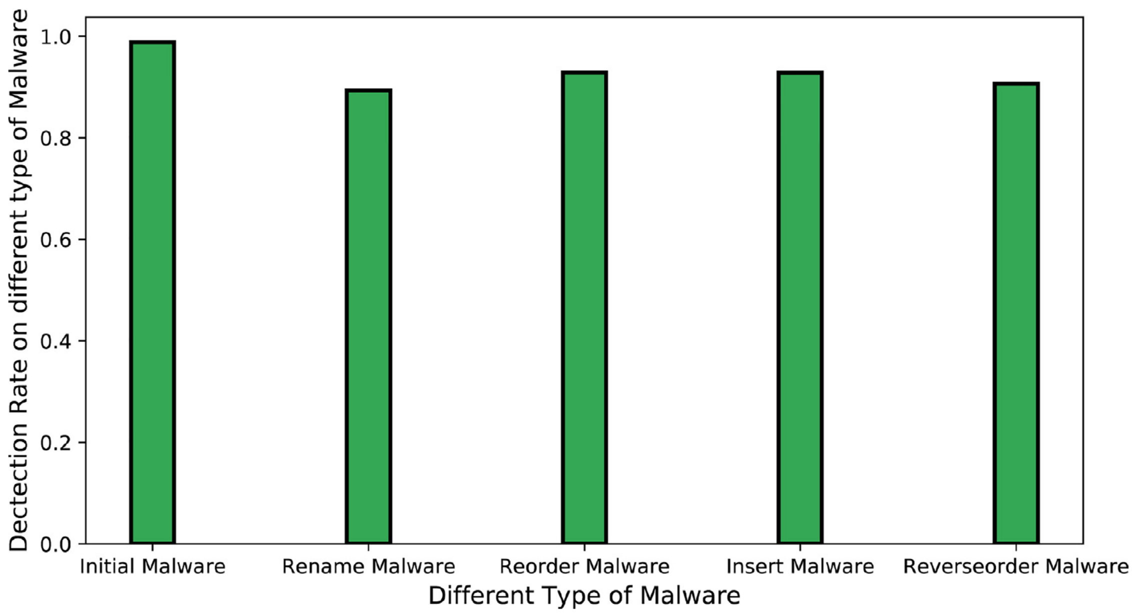

Fig. 11. Performance comparison on malware an their different-type variants.

for ESETNODE-32 to have a better performance is that it integrates dynamic behavior detection, UTFI scanning[6] and other technologies that are more comprehensive than ours.

5. Limitation and discussion

Our method focuses only on URLs in the HTTP traffic, therefore, we cannot perform the identification of malware that use non-HTTP protocols or HTTP encryptions (i.e., HTTPs). We conduct a statistical analysis on the collected traffic, and the proportion of malware samples that use unencrypted HTTP protocol for communication is 83.67%. The statistics show that our method is effective in detecting most of the real-world malware samples. The process of malicious URL identification requires labels for the training data set. Unfortunately, samples with specific labels across the entire network are relatively hard to find. In addition, our method helps expose the malicious network behavior of apps that are unobservable using other tools. Our technique is uniquely suited for scenarios when only URLs are available as features. For instance, some web servers only preserve URL data while discarding all other information in the network traffic.

Moreover, more and more malware prefers to interact with networks using encrypted traffic to bypass detection. However, encrypted malicious traffic is an acknowledged problem that has been investigated by researchers [30]. To deal with encrypted traffic, we have designed a system that leverages the VPN API on mobile devices to provide full access to the network traffic of these devices and identify malware samples with HTTPs traffic. A forwarder in gateway transparently bridges packets on the VPN interface and payload data on the regular socket interface and forwards the traffic to the detection server for analysis. Using this method, we can address the challenge of malware identification with encrypted traffic.

6. 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. Each view focuses on only one portion of the features of the input data. Multiple views conjoin to complete the URL vector classification. The network focuses on depth but also emphasizes width and can complete the automatic selection of features from multiple levels. 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.

[6] https://www.eset.com/us/about/newsroom/corporate-blog/whatis-uefi-scanning-and-why-do-you-need-it-1.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research was partially supported by the National Natural Science Foundation of China under Grants nos. 61672262, 61472164, .61573166, and 61572230, the Shandong Provincial Key R&D Program under Grants nos. 2017CXZC1206 and 2018CXGC0706, the project of Independent Cultivated Innovation Team of Jinan City under Grant no. 2018GXRC002 the Shandong Department of Education Youth Science and Technology Innovation Team grants:2019KJN028. The work of Qiben Yan is supported in part by the NSF grants CNS-1566388 and CNS-1717898.

References

[1] B. Anderson, D. McGrew, Identifying encrypted malware traffic with contextual flow data, in: Proceedings of the 2016 ACM Workshop on Artificial Intelligence and Security (AISec@CCS), ACM, 2016, pp. 35–46.
[2] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, Drebin: effective and explainable detection of android malware in your pocket, 21st Annual Network and Distributed System Security Symposium (NDSS), 2014.
[3] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, P. McDaniel, Flowdroid: precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps, ACM Sigplan Not. 49 (6) (2014) 259–269.
[4] D. Bahdanau, K. Cho, Y. Bengio, Neural machine translation by jointly learning to align and translate, in: 3rd International Conference on Learning Representations, (ICLR), Conference Track Proceedings, 2015.
[5] Z. Chen, B. Yu, Y. Zhang, J. Zhang, J. Xu, Automatic mobile application traffic identification by convolutional neural networks, in: 2016 IEEE Trustcom/BigDataSE/ISPA, 2017, pp. 301–307.
[6] M. Conti, L.V. Mancini, R. Spolaor, N.V. Verde, Analyzing android encrypted network traffic to identify user actions, IEEE Trans. Inf. Forensics Secur. 11 (1) (2016) 114–125.
[7] O.E. David, N.S. Netanyahu, Deepsign: deep learning for automatic malware signature generation and classification, in: International Joint Conference on Neural Networks (IJCNN), 2015, pp. 1–8.
[8] M. Egele, T. Scholte, E. Kirda, C. Kruegel, A survey on automated dynamic malware-analysis techniques and tools, ACM Comput. Surv. (CSUR) 44 (2) (2012) 6.
[9] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L.P. Cox, J. Jung, P. McDaniel, A.N. Sheth, Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones, ACM Trans. Comput. Syst. (TOCS) 32 (2) (2014) 5.
[10] P. Faruki, A. Bharmal, V. Laxmi, V. Ganmoor, M.S. Gaur, M. Conti, M. Rajarajan, Android security: a survey of issues, malware penetration, and defenses, IEEE Commun. Surv. Tutor. 17 (2) (2015) 998–1022.
[11] Y. Feng, S. Anand, I. Dillig, A. Aiken, Apposcopy: Semantics-based detection of android malware through static analysis, in: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (SIGSOFT FSE), ACM, 2014, pp. 576–587.
[12] A. Fonarev, O. Hrinchuk, G. Gusev, P. Serdyukov, I.V. Oseledets, Riemannian optimization for skip-gram negative sampling, in: Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL), 2017, pp. 2028–2036.
[13] C. Gao, S. Lv, Y. Wei, Z. Wang, Z. Liu, X. Cheng, M-SSE: an effective searchable symmetric encryption with enhanced security for mobile devices, IEEE Access 6 (2018) 38860–38869.
[14] H. Guo, C. Cherry, J. Su, End-to-end multi-view networks for text classification, (2017), arXiv:1704.05907.
[15] H. Guo, H.L. Viktor, Multirelational classification: a multiple view approach, Knowl Inf Syst 17 (3) (2008) 287–312.
[16] Y. Kim, Convolutional neural networks for sentence classification, in: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2014, pp. 1746–1751.
[17] D.P. Kingma, J. Ba, Adam: a method for stochastic optimization, in: 3rd International Conference on Learning Representations, (ICLR), Conference Track Proceedings, 2015.
[18] J. Li, L. Sun, Q. Yan, Z. Li, W. Srisa-an, H. Ye, Significant permission identification for machine-learning-based android malware detection, IEEE Trans. Ind. Inf. 14 (7) (2018) 3216–3225.
[19] Y. Liu, J. Ling, Z. Liu, J. Shen, C. Gao, Finger vein secure biometric template generation based on deep learning, Soft Comput. 22 (7) (2018) 2257–2265.
[20] L. Lu, Z. Li, Z. Wu, W. Lee, G. Jiang, Chex: statically vetting android apps for component hijacking vulnerabilities, in: Proceedings of the 2012 ACM conference on Computer and communications security (CCS), ACM, 2012, pp. 229–240.
[21] A. Moser, C. Kruegel, E. Kirda, Limits of static analysis for malware detection, in: Computer Security Applications Conference (ACSAC), IEEE, 2007, pp. 421–430.
[22] H. Ogawa, Y. Yamaguchi, H. Shimada, H. Takakura, M. Akiyama, T. Yagi, Malware originated http traffic detection utilizing cluster appearance ratio, in: Information Networking (ICOIN), IEEE, 2017, pp. 248–253.
[23] T. Peng, Q. Liu, D. Meng, G. Wang, Collaborative trajectory privacy preserving scheme in location-based services, Inf Sci 387 (2017) 165–179.
[24] J. Pennington, R. Socher, C. Manning, Glove: Global vectors for word representation, in: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2014, pp. 1532–1543.
[25] V. Rastogi, Y. Chen, X. Jiang, Catch me if you can: evaluating android anti-malware against transformation attacks, IEEE Trans. Inf. Forensics Secur. 9 (1) (2013) 99–108.
[26] K.A. Talha, D.I. Alper, C. Aydin, Apk auditor: permission-based android malware detection system, Digit. Invest. 13 (2015) 1–14.
[27] P.R.K. Varma, K.P. Raj, K.S. Raju, Android mobile security by detecting and classification of malware based on permissions using machine learning algorithms, in: I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC),, IEEE, 2017, pp. 294–299.
[28] S. Wang, Z. Chen, Q. Yan, K. Ji, L. Wang, B. Yang, M. Conti, Deep and broad learning based detection of android malware via network traffic, in: 26th IEEE/ACM International Symposium on Quality of Service (IWQoS), 2018, pp. 1–6.
[29] S. Wang, Z. Chen, L. Zhang, Q. Yan, B. Yang, L. Peng, Z. Jia, Trafficav: An effective and explainable detection of mobile malware behavior using network traffic, in: 24th IEEE/ACM International Symposium on Quality of Service (IWQoS), IEEE, 2016, pp. 1–6.
[30] S. Wang, Q. Yan, Z. Chen, B. Yang, C. Zhao, M. Conti, Detecting android malware leveraging text semantics of network flows, IEEE Trans. Inf. Forensics Secur. 13 (5) (2018) 1096–1109.
[31] W. Wang, R. Arora, K. Livescu, J. Bilmes, On deep multi-view representation learning, in: International Conference on Machine Learning (ICML), 2015, pp. 1083–1092.
[32] W. Wang, Z. Gao, M. Zhao, Y. Li, J. Liu, X. Zhang, Droidensemble: detecting android malicious applications with ensemble of string and structural static features, IEEE Access 6 (2018) 31798–31807.
[33] W. Wang, Y. Li, X. Wang, J. Liu, X. Zhang, Detecting android malicious apps and categorizing benign apps with ensemble of classifiers, Future Gener. Comput. Syst. 78 (2018) 987–994.

[34] W. Wang, X. Wang, D. Feng, J. Liu, Z. Han, X. Zhang, Exploring permission-induced risk in android applications for malicious application detection, IEEE Trans. Inf. Forensics Secur. 9 (11) (2014) 1869–1882.
[35] Z. Wang, The Applications of Deep Learning on Traffic Identification, BlackHat USA, 2015.
[36] L. Yang, Z. Han, Z. Huang, J. Ma, A remotely keyed file encryption scheme under mobile cloud computing, J. Netw. Comput. Appl. 106 (2018) 90–99.
[37] Y. Zhou, X. Jiang, Dissecting android malware: characterization and evolution, in: Security and Privacy (SP), IEEE, 2012, pp. 95–109.