Lexical Mining of Malicious URLs for Classifying Android malware

Shanshan Wang[1] Qiben Yan[2] Zhenxiang Chen*[1] Lin Wang[1] Riccardo Spolaor[3] Bo Yang[1] and Mauro Conti[3]

[1] Shandong Provincial Key Laboratory of Network Based Intelligent Computing, University of Jinan, Jinan, China
[2] Department of Computer Science and Engineering, University of Nebraska-Lincoln, Lincoln, NE, USA
[3] Department of Mathematics, University of Padova, Padua, Italy
*Corresponding author, Email: czx@ujn.edu.cn

Abstract. The prevalence of mobile malware has become a growing issue given the tight integration of mobile systems with our daily life. Most malware programs use URLs inside network traffic to forward commands to launch malicious activities. Therefore, the detection of malicious URLs can be essential in deterring such malicious activities. Traditional methods construct blacklists with verified URLs to identify malicious URLs, but their effectiveness is impaired by unknown malicious URLs. Recently, machine learning-based methods have been proposed for malware detection with improved performance. In this paper, we propose a novel URL detection method based on Floating Centroids Method (FCM), which integrates supervised classification and unsupervised clustering in a coherent manner. The proposed method uses the lexical features of a URL to effectively identify malicious URLs while grouping similar URLs into the same cluster. Our experimental results show that a URL cluster exhibits unique behavioral patterns that can be used for malware detection with high accuracy. Moreover, the proposed behavioral clustering method facilitates the identification of malicious URL categories and unseen malware variants.

1 Introduction

Malicious software, or malware, has become a major threat to the growing mobile ecosystem. Recently, the number and sophistication of mobile malware, particularly those target Android platforms, have increased dramatically [1]. The Android platform and mobile anti-virus scanners provide security protection mechanisms to protect Android devices, yet an increasing number of advanced mobile malware can still penetrate the mobile system by evading these mechanisms. As mobile devices are increasingly associated with personal information, an effective mobile malware detection system is urgently needed.

Malware authors have adopted repackaging and code obfuscation techniques to generate a large number of malware variants. These malware variants exhibit

similar malicious behaviors at runtime, which can be clustered together to identify their common behaviors. The vast majority of malware programs launch their malicious activities through network (e.g., sending spam, exfiltrating private data, and downloading malware updates). Thus, we can use the malware's network behaviors to conduct classification.

Clustering algorithm is an unsupervised learning method, which groups the samples into different families based on their similarities with each other. However, the challenge of the clustering algorithms is to accurately cluster the same family of malware together, while avoiding the inclusion of benign apps. Some clustering algorithms can effectively discover the differences and commonalities between malicious samples, and with these features they can divide malicious samples into multiple categories. However, this approach does not have the ability to efficiently distinguish between benign and malicious samples, i.e., it is highly likely that a benign sample will be included into a malicious cluster when it has some similar characteristics with a certain type of malware.

In this paper, we introduce a novel machine learning technique, Floating Centroid Method (FCM) [2] for mobile malware detection and malware family clustering. FCM can cluster similar samples with the same label, while separating samples with different labels as much as possible that effectively avoids the inclusion of benign samples into a malicious cluster. Note that most malware programs use URLs to execute or transfer commands to support their malicious behaviors [3]. So the method that extracts URLs in HTTP traffic to detect malware can be effective in most cases. Using FCM, malicious URLs can be clustered and identified. By analyzing clustering results, we can find more valuable information about malware's network behaviors. The contribution can be summarized as follows:

- Through the analysis on URLs, we discover the many-to-many relationship between URLs and malware family labels. Based on this observation, we propose a novel network-level behavioral clustering method.
- We use Canopy algorithm [4] to improve the selection of cluster number in FCM. The improved FCM can quickly determine the optimal cluster number. With the improved FCM algorithm, we create a novel model that can cluster and detect malicious URLs based on similar lexical features which has a higher accuracy than traditional clustering algorithms.
- We mine the rich information within each URL cluster and perform statistical and manual analysis to reveal different behavioral patterns in different clusters which helps in finding malicious variants.

The rest of the paper is organized as follows: related works are introduced in Section II. We give a detailed description on the method implementation in Section III. The experimental results and comparative analysis are discussed in Section IV. The limitation of this method is introduced in Section V. The conclusions are provided in Section VI.

2 Related Work

Malware detection has traditionally been implemented based on static and dynamic analysis methods. Static analysis can identify malicious behaviors of suspicious apps without code execution. DroidMat [5], Drebin [6], and DroidMiner [7] are static analysis methods that utilize the machine-learning algorithm to detect anomalies by analyzing permissions, called APIs or bytecode instructions. However, static analysis is challenged by the code polymorphism and obfuscation of malware. In dynamic analysis methods, the app is executed in a sandbox environment. Dynamic analysis systems [8, 9] have been proposed to analyze system calls to detect malicious behaviors. However, these dynamic analysis methods are difficult to deploy due to their complexity.

In addition, suspicious apps can be analyzed by observing their network traffic. We briefly review the mechanisms that use network traffic for malware detection. Some malware detection methods focus on a specific network protocol [10], which considers some basic information about the TCP header. Other studies have focused on the HTTP application-layer traffic between the attackers and victims, such as works [11], [12] and [13]. The flower system [11] is an automatic app signature system that only considers the key value pair and hostname in HTTP header. TrafficAV [12] uses four fields (request method, request host, request URL, and user-agent) in HTTP header and combines the decision-tree algorithm to create an effective malware detection model. Recon [13] reveals privacy leaks in mobile network traffic by observing the keys that appear in the URLs. Work [14] also focus on the URLs. The authors design and implement AURA, a framework for identifying the hosts that an app talks to and evaluating the risks communication entails. Many studies focus on the clustering of malware samples to perform malware detection or explore malicious behavior of a certain type of malware. Shabtai et al. [15] analyzed a large amount of Android network traffic to identify malicious attacks by repackaging. They pointed out that the apps should be grouped into different categories based on the statistical characteristics of network data. They also summarized the deviation between benign and malicious network behaviors. Gorla et al. [16] focused on the market descriptive information of the app to extract keywords for clustering different apps. They identified the most unusual apps in each cluster as suspicious apps because apps in the same cluster would have similar attributes or behaviors, while the suspicious apps are drastically different from other apps. However, this method has a high false-positive rate.

Our proposed method differs from the above classification and clustering methods, as we integrate classification and clustering into a holistic model. The benign URLs are for the convenience of people's memory, while malicious URLs do not want to attract people's attention. Malicious URLs are often filled with a lot of junk characters and change encoding methods, use IP addresses instead of domain names, and randomly generate domain names. From this point of view, lexical mining of malicious URLs is a viable way. So our work clusters and identifies malicious URLs based on their lexical features. FCM algorithm is used

to effectively cluster similar URLs into a group while identifying malicious URLs within the clustered URLs, attaining a better clustering performance.

3 Methodology

Our goal is to cluster similar samples together while keeping benign and malicious samples separated. Figure 1 presents an overview of our method, including URL extraction, feature representation as well as clustering and detection.

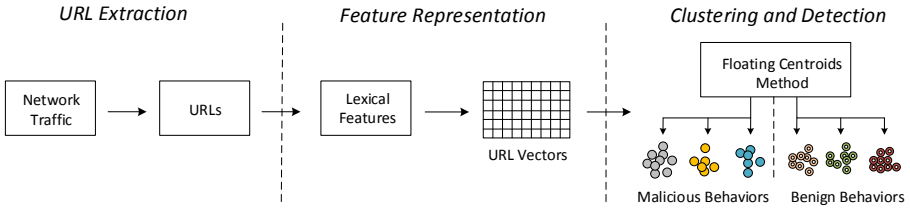

Fig. 1: The overview of URL clustering and detection method

3.1 URL Extraction

We design a traffic collection platform to collect network data generated by Android apps during network interaction. Then, we extract URL samples from the network traffic. A large number of network traffic data generated by both benign and malicious apps is collected. This module consists of two components: app execution and network traffic collection.

We run the apps on multiple Android emulators. Every app is driven by the Android tool Monkey [17], which can randomly send some events to the device during the execution of each app. In the process of traffic collection, additional operations of simulator restart and random event generation are used to trigger the malware's malicious activities as much as possible. To avoid the network traffic mixing by different apps, we only execute one app every time. Before running the next app, the emulator will be destroyed and a new emulator is re-established, which ensures that no app is running in the background when each app is executed. We then extract URLs from traffic data using the tshark tool [18].

3.2 Feature Representation

1) Component Description: We divide each URL into five components and process them separately. The five components are shown in Figure 2. The component m represents the request method, such as "GET", "POST" and "HEAD". The component h represents the hostname. This field is specific for the Internet host and port number of the requested resource. The component p stands

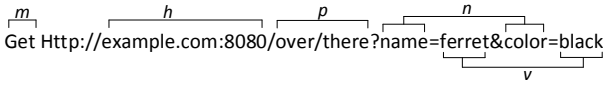

Fig. 2: Example of a URL consisting of five components

for page, which includes the path and page name. We use the "/" character to segment this string and regard each word as a candidate feature after page segmentation. The component n represents the set of parameter names (i.e., n = {name, color}). The parameter names are always followed by the page and start with the character of "?". The component v is the set of parameter values (i.e., v = {ferret, black}). The parameter values are usually followed by the parameter names and connected to the parameter name with "=". Not all of the URLs contain these five components. In general, components m, h, p are common in almost every URL, and only a partial URLs contain components n and v.

2) Feature Selection: We process each component separately. For m component, we save all the request methods appearing in our dataset into a dictionary. For h component, we use another dictionary to save all the different hostnames that appear in our dataset. For the word in components p, n and v, we consider using an automatic feature selection algorithm (chi-square test [19]) to automatically identify meaningful features. This approach accounts for the relevance of a single word to the final category label and ignores the frequency of each feature appears. The formula of the chi-square test is as follows:

$$\chi^2(t,c) = \sum_{e_t \in 0,1} \sum_{e_c \in 0,1} \frac{(N_{e_t e_c} - E_{e_t e_c})^2}{E_{e_t e_c}} \tag{1}$$

where $N_{e_t e_c}$ refers to the occurrence number of feature t and class c, and $E_{e_t e_c}$ is the expected occurrence number of feature t and class c when they are independent of each other. The e_t and e_c are boolean, and value "0" indicates that the feature t is not in the word set from class c, whereas e_t with a value of "1" indicates that word set of class c contains feature t. We use all the lexical features in the components of m ,h and select 100 features with high chi-square test score from components p, n and v respectively.

3) Vectorization: We vectorize the selected features since the adopted machine-learning algorithm can only accept numerical data as input. We use one-hot encoding method to encode selected features obtained in feature selection section. In one-hot encoding, each word will be converted into m bits, among which only one bit is set to 1 and the others are set to 0. Notably, we make a distinction between the words belonging to components m, h, p, n and v. This is done by having a separate dictionary for each component. After encoding words from different components, we also need to vectorize each URL. Given a URL, the resulting vector can be stitched together with multipart vectors. In this vector, the value of 1 indicates that the word appears in the URL, and 0 otherwise.

3.3 Clustering and Detection Model

The original FCM algorithm comprises of two parts. The first part is a three-layer feedforward neural network and the second part is a K-Means clustering algorithm. The K-Means algorithm needs to set the K value in advance. However, the K selection is a difficult but critical issue. Thus, we propose to enhance the FCM by adding the Canopy algorithm [4] for data coarse clustering. The modified FCM uses Particle Swarm Optimization (PSO) [20] algorithm to adjust the parameters of the neural network in accordance with the clustering accuracy of K-Means. The schematic of the modified FCM is shown in Figure 3 and we elaborate on the details of each part in the following.

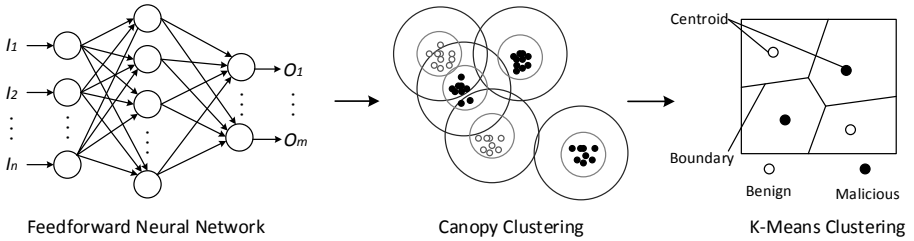

Fig. 3: The schematic diagram of improved FCM algorithm

1) Neural Network Mapping: A mapping relationship refers to the transfer of training data samples from the original data space to the partition space. For this mapping, the input dimension is N and the output dimension is M, so the mapping is from a vector of N-dimensional elements to a vector of M elements. Specifically, the N-dimensional URL vector is fed into the neural network and then is mapped as a vector with M-dimensions. Given that the feedforward neural network can fit any nonlinear function, it is suitable to use feedforward neural network for mapping completion.

2) Canopy Clustering: To determine the optimum K value quickly and accurately, we use Canopy algorithm to cluster data roughly, and then obtain K according to Canopy-clustering results. Specifically, the Canopy algorithm is used to cluster data mapped by the neural network roughly and then calculate the best cluster number for the dataset. Although the Canopy clustering algorithm has low accuracy, it has a great advantage in speed and thus is often used with K-Means.

3) K-Means Clustering: The optimum K is determined by Canopy clustering, and then K-Means algorithm is used to divide the mapped data into K disjoint clusters. The center of each cluster is called centroids. After calculating K centroids, each cluster is marked as malicious or benign in a process called coloring. To prevent coloring bias toward the class with more samples, in practice we balance the problem through setting weights for samples. The sample

Algorithm 1 PSO algorithm optimizes the feedforward neural network

Input: Dataset S and the structure of feedforward neural networks
Output: Best neural network and its corresponding clustering model

```
Initializing 20 different neural networks as individuals
while maximum generation has not been reached do
    for id:=1 to the number of individuals do
        Map dataset S to the partition space
        Canopy algorithm clusters the mapped dataset to obtain the K
        K-Means algorithm clusters the mapped dataset
        According to clustering result to calculate E value as this individual's fitness
    Update individuals by their fitness
return Best neural network and its corresponding clustering model
```

weight belonging to class i with $|S_i|$ samples is defined as follows:

$$W_i = \frac{1}{|S_i|} \tag{2}$$

The principle of coloring is that if the sum of weights of malicious samples in a cluster takes the majority, the cluster is colored as malicious. Otherwise, the cluster is marked as benign.

4) Learning Process: We first define a variable z to evaluate a mapped point in the partition space whose formula is shown in Formula 3. For a point in the partition space, d_{min}^{self} represents the euclidean distance between the point and the closest centroid with the same label. Similarly, d_{min}^{noself} is the distance between the point and the nearest centroid having different labels with it. When no cluster has the same label or has different labels for this point, z is assigned as the *maximum* that is a number. Under a limited condition, if a sample happens to be mapped to a cluster center whose label is consistent with the sample, then z equals 0; if not, then z equals 1.

$$z = \begin{cases} \frac{d_{min}^{self}}{d_{min}^{self}+d_{min}^{noself}} & \exists\, \ell_{self\ class} \bigwedge \ell_{noself\ class} \\ maximum & Else \end{cases} \tag{3}$$

Finally, the optimization target function is defined as follows:

$$E = \sum_{l=0}^{s} \frac{1}{1 + e^{a(1-2Z_l)}}, \tag{4}$$

where Z_l is the z value of the lth sample, s is the total number of samples in the training set, and a is a real constant determining *tortuosity*, and E is the target of neural network optimization. A smaller E indicates a better neural network and partition space.

Based on the optimization target function, the PSO algorithm is used to optimize the neural network and simultaneously obtain the final partition space. The detail of how the PSO algorithm utilizes the clustering result of K-Means

to optimize the feedforward neural network is shown in Algorithm 1. With the optimized neural network, unseen URL vector can be mapped as a new vector. The data space where the vector is located has a clear boundary between benign and malicious samples, and then the vector is clustered by K-Means to a specific cluster. The prediction label of the URL vector is the same as the nearest cluster centroid, and the URL vector shares some attributes with other samples in this cluster.

4 Evaluation

For evaluating the proposed method, we first introduce the dataset used in our method, and then analyze some parameters that affect the model's performance. Next, we compare our performance with other state of the art methods. Some interesting findings are presented. Lastly, we apply the model on the wild apps, and compare the detection results with different anti-virus scanners.

4.1 Data Set

Malicious apps originate from VirusShare website [21]. This library, which is constantly updated, is one dedicated to providing a large number of malware datasets for security researchers. We downloaded 27127 samples from this website dated between July 2014 and September 2016. We collected the network traffic generated by these malicious apps and finally obtained 18.9 GB traffic. We extracted URL samples from the network traffic. Notably, not all URLs requested by malware are malicious, and malicious URLs may only account for a small part. So to let our training set have correct labels, we screened all URLs using the detection report from VirusTotal [22]. Only explicitly malicious URLs were added to the collection of malicious URLs. Eventually, only 11251 explicitly malicious URLs are added to our dataset.

As for the benign data set, we downloaded a total of 6072 apps from multiple third-party application markets (hiapk, wandoujia, and yinyongbao). Similarly, the apps we downloaded from the app markets were not always benign. So we also used VirusTotal to screen these apps. Only apps that VirusTotal confirms benign are added to our benign app collection. And then the traffic-collection platform was used to obtain their traffic data. Ultimately, we obtains 25276 benign URLs from the 14.2GB collected traffic. Although benign URLs are more than twice of malicious URLs, the FCM algorithm sets different weights (see Formula 2) for samples from different class which helps balance the problem.

4.2 Evaluation of Clustering and Detection Model

The parameters of FCM algorithm are related to the structure of the feedforward neural network, which is determined by the number of neurons that are included in the input, hidden, and output layers. According to the empirical analysis of FCM [2], we set the number of hidden-layer neurons in the neural network to 15

and the neurons number in the output layer to 9. Given that the neuron number of the input layer is determined by the data set, we set the neuron number in the input layer to 475 according to the length of URL vector.

1) The Effect of Neural Network Mapping: We use a feedforward neural network to map training data from the initial data space to the partition space. The 475-dimensional data are mapped to 9 dimensional. In theory, a clear division exists among the different categories of mapped data. To verify that our optimal neural network can be helpful for clustering, we display the training data before and after neural-network mapping. We use Principal Component Analysis (PCA) to process and subsequently visualize the data. Figure 4(a) shows the initial data, where different shapes represent different categories (solid points represent benign samples, and hollow points represent malicious samples). We find that a large part of different category data are mixed together. If directly clustering these data, we will end with a false inclusion of benign samples in malicious clusters. A clear boundary is observed between benign and malicious samples after mapping in Figure 4(b), and only a few samples fall into other categories. Thus, from the figures, we can clearly conclude that the data after mapping are more helpful in improving clustering accuracy than data before mapping.

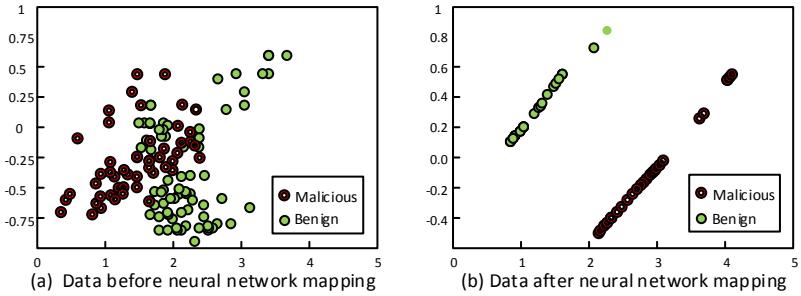

Fig. 4: Training Data distribution before and after neural network mapping

2) The Impact of URL's Different Components on Model: We divide a URL into five components and deal with each component separately. By intuition, each of the five components plays a different role on malicious URL clustering and detection model. In this section, we assess the impact of different components of a URL on final results. In the experiment, we vectorize the components m , h , p, n, and v, respectively. Each component after vectorization is then fed to the improved FCM algorithm to train the corresponding clustering and detection model. Training set and test set are separated randomly and the test set occupy 30% of total samples. According to the accuracy of different models on test set, we plot the line chart (Figure 5).

The horizontal axis represents the optimization generation of PSO optimizing target function (see Formula 4), and the vertical axis represents the Accuracy of

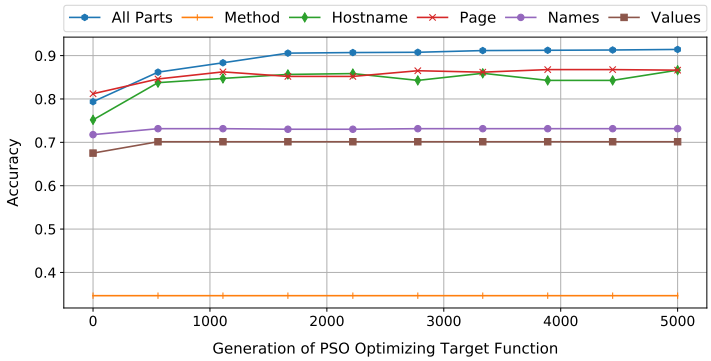

Fig. 5: Different components of URL affect on the accuracy with the optimization generation increasing

the model at different optimization generations. Each type of line represents a different model trained by different components of the URL. We can see that the components p and h have greater contributions to the model than components of m, n and v. Component m of the URL has the poorest recognition because the request method is not diverse enough and almost all of request methods are "GET" and "POST". Each component plays a role in malicious URL detection, so we can derive a better model by combining all URL components.

4.3 Comparison with State of The Art

1) Comparison with Other Clustering Algorithms: FCM can cluster similar samples into the same group. Here, we compare the clustering performance of FCM with other popular clustering algorithms. We have selected several popular clustering algorithms, i.e., K-Means, DBSCAN, Brich, and Hierarchical Clustering. For each algorithm, we attempt to use multiple sets of parameters to maximize the performance of each algorithm. The final results of different algorithms are shown in Figure 6. Regarding malware identification performance, FCM performs best in terms of Accuracy, Precision, F-Measure, and FPR. Only DBSCAN algorithm has higher TPR than FCM, but DBSCAN algorithm has a very high false-positive rate. Regarding clustering performance, we compare the Silhouette Coefficient (SC) [23] of each algorithm. A higher SC score means a model with better-defined clusters. Figure 6 shows that the SC score of FCM is 0.4, which is much higher than that of other algorithms.

2) Comparison with Other URL-similarity Measurement Methods: We claim that we can use the lexical features of URL to cluster similar URLs into a group. The most relevant method on URL-similarity measurement method is described in [24], which considers structural similarity among URL strings.

In their method, URLs are divided into the four components m,p,n,v and does not use the hostname information. However, we believe that regardless of

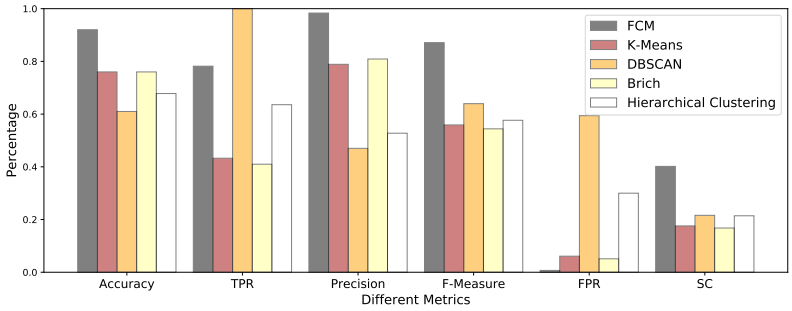

Fig. 6: The model evaluation of FCM with the traditional clustering algorithms

malicious URL detection or clustering, the hostname plays an important role. Our experimental results further prove this point (see Figure 5). In [24], they initially calculate the distance between the m component. If the request method is consistent, the distance d_m is 0; if it is different, the distance d_m is 1. They use the edit distance to count the distance between the p component of two URLs, named as d_p. For component n, they save all the names of a URL to an list and then calculate the jaccard distance between the two lists; this distance is recorded as d_n. For component v, they splice the values of a URL into a string and calculate the editing distance between two strings when two URLs are compared. The distance of this component is d_v. Finally, the distance between two URLs is d, and the formula of d is as follows:

$$d = d_m + d_p + d_n + d_v \tag{5}$$

To compare two URL-similarity measurement methods, we use K-Means algorithm to cluster the data sets obtained by the two methods. Figure 7 shows the clustering results of both methods. We can see that our methods have obvious advantages over structural similarity method [24] in all metrics.

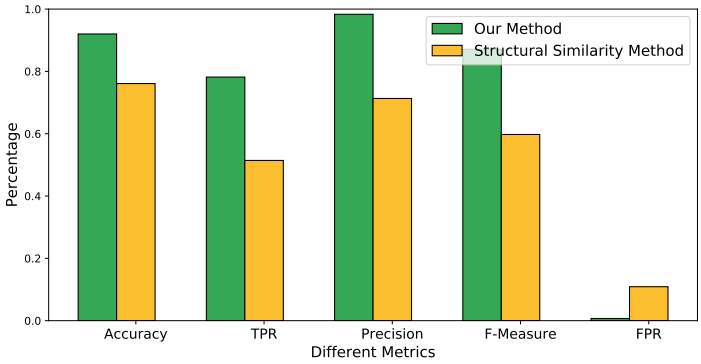

Fig. 7: The clustering results comparison of our method and structured similarity method [24]

4.4 Interesting Findings

FCM algorithm divides our data into 26 small clusters and each cluster is labeled benign or malicious. The URLs in a cluster are grouped together based on similar lexical features. We analyze each cluster and discover that each one has some interesting characteristics. We use DOM Tree technology to visualize each cluster. Here, we show two examples. One cluster is marked as malicious (see Figure 8), and the other is benign (see Figure 9).

Figure 8 shows a total of 114 URLs in the cluster. The request methods in the cluster are all "GET", and contain six unique hostnames. The different paths are followed by the hostnames. Words ending with the symbol "=" are keys in query strings. The corresponding values of the keys are not shown because the values are usually alphanumeric strings that are unique for the app itself or for third-party providers. The hostname "stat.appsgeyser.com" is malicious as validated with VirusTotal. However, no detailed information on malicious behavior about the hostname is found in VirusTotal. We can see that the keys in query strings are "action", "name", "id", "p", "age", "stall" and "system". The values of "name", "id" and "age" are related to the private data of users and devices. The word "ad" appearing in multiple URLs shows that the URLs may be related to the advertising service.

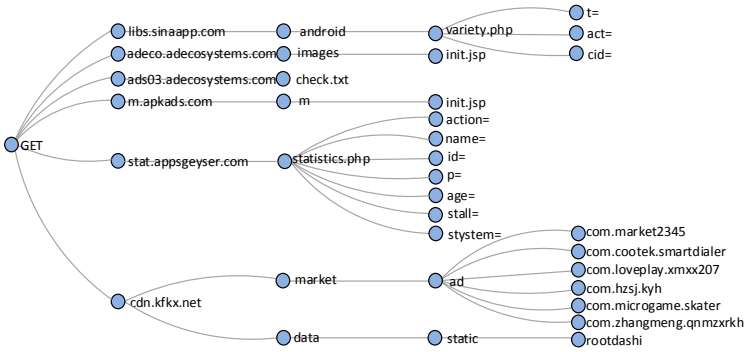

Fig. 8: An example of malicious cluster

The cluster in Figure 9 is a benign cluster in which the method is also "GET" and contains three unique hostnames. Frequent words found in the cluster are "t-ingsh", "service" and "images". Obviously, this cluster gathers a number of flows related to entertainment services. The specific entertainment service is listening to books. Interestingly, the URLs in the cluster do not transmit any parameter to server, and most of the requested resources are images. This phenomenon is in line with common sense because the apps need to load some pictures or other resources when they start.

By comparing multiple clusters, we conclude that the words used in URLs are always related to particular services and can reflect some specific behaviors. In particular, the benign and malicious URLs tend to use different words, so it fur-

ther validates the use of lexical features in performing malicious URL clustering and detection. The analysis on the cluster will help us gain more understanding of malware's network behaviors.

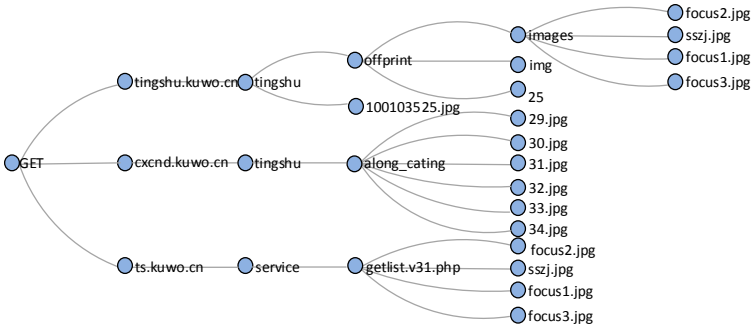

Fig. 9: An example of benign cluster

4.5 URL Detection in the Wild

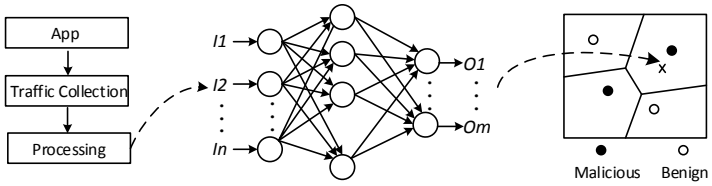

Fig. 10: The detection process for unseen app

The ultimate goal of creating a detection model is to be applied to the real environment to detect malicious apps. To verify the actual detection capabilities of the established model, we download 833 new apps from the app market in December 2017, and extract a total of 10473 URLs from their network traffic. We use bag of words model created in the training phase to vectorize these URLs, and then feed the vectors into the trained neural network. The neural network, which maps the data sample to a partition space, and then the URL (i.e., benign or malicious) is predicted based on the partition space in which the sample is located, i.e., the centroid closest to the sample after the mapping. The app is marked as malware, if it contains malicious URLs. The entire process is shown in Figure 10.

In the end, our new malware dataset consists of 305 malicious apps that are confirmed by VirusTotal reports. The 305 malware are filtered by 59 anti-virus scanners in VirusTotal; however, each scanner in VirusTotal can only detect

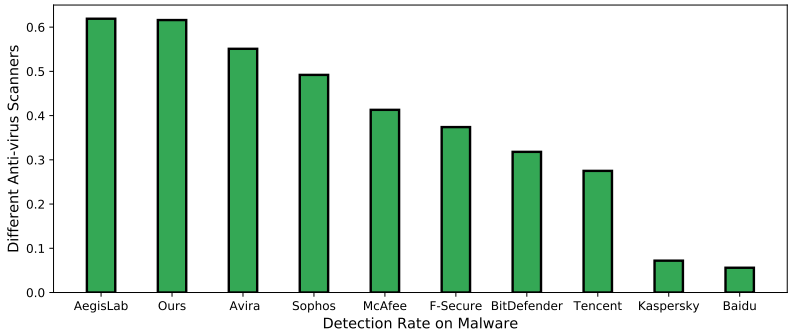

Fig. 11: Detection rate comparison with novel malware in the wild using our method and other anti-virus scanners

part of these malware samples. We select nine popular anti-virus scanners which are AegisLab, Avira, Sophos, McAfee, F-Secure, BitDefender, Tencent, Kaspersky and Baidu respectively. The detection results of scanners are derived from the VirusTotal service, which vary considerably. The best anti-virus scanner is AegisLab which can detect 189 out of 305 malware and the detection rate is 61.9%, whereas the Baidu scanner only discovers 17 malware in the wild app set whose detection rate is only 5.6%. Figure 11 shows the detailed statistics. In contrast, our detection model can identify 188 out of 305 apps and the detection rate is 61.6% that is on par with the best performing scanner, and outperforms eight other anti-virus scanners. Note that detecting novel malicious apps is a notoriously difficult task, and all existing methods are not able to achieve high detection rate due to the malware's high adaptability. Thus, the comparison result validates the capability of our model in scanning wild apps.

5 Limitations

Our method only focuses on the URLs in HTTP traffic which brings its limitation on identifying traffic using non-HTTP protocols or HTTP encryptions (i.e., HTTPs). We have conducted a statistical analysis on the collected traffic, and the proportion of malware samples using unencrypted HTTP protocol for communication is 83.67% [25]. From the statistics, we can conclude that our method will be effective in detecting most of the real-world malware samples. We admit that new types of malware using different URLs or obfuscating URLs can bypass the proposed method. This is a common caveat of supervised learning method. However, when the new malware is added to the training samples, we could re-train and update the classifier for detecting such new type of malware. In addition, the process of malicious URL identification requires the label (benign and malicious) for the training data set. Unfortunately, samples with specific labels across the entire network are relatively hard to find.

6 Conclusion

In this paper, we propose an accurate and efficient malware detection method through malicious URLs clustering and detection from network traffic. To facilitate malicious URL clustering and detection, we enhance the FCM algorithm to render it suitable for finding best cluster number. Using the enhanced FCM algorithm and a real-world dataset, we detect malicious URLs and gather similar URLs into the same cluster. For URL clusters, we discover insightful behavioral difference between benign and malicious URLs using statistical and manual analyses. Specifically, we observe that the words used in URLs have close relationships with the specific services or reflect behaviors pertinent to the malicious activities. Cluster analysis simplifies the analysis on malware and further improves malware detection at the network-level.

Acknowledgement

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. Security threat report 2014. [online]. Available: http://www.sophos.com/en-us/medialibrary/PDFs/other/sophossecurity- threat-report-2014.pdf.
2. L. Wang, B. Yang, Y. Chen, A. Abraham, H. Sun, Z. Chen, and H. Wang, "Improvement of neural network classifier using floating centroids," *Knowledge and Information Systems*, vol. 31, no. 3, pp. 433–454, 2012.
3. Specification of malicious url 2013. [online]. Available: http://www.antiy.net/p/specification-of-malicious-url.
4. Canopy clustering algorithm. [online]. Available: https://en.wikipedia.org/wiki/Canopy_clustering_algorithm.
5. D. J. Wu, C. H. Mao, H. M. Lee, and K. P. Wu, "Droidmat: Android malware detection through manifest and api calls tracing," in *Information Security*, 2012, pp. 62–69.
6. D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck, "Drebin: Effective and explainable detection of android malware in your pocket." in *Proc. The Network and Distributed System Security Symposium (NDSS)*, 2014.
7. C. Yang, Z. Xu, G. Gu, V. Yegneswaran, and P. Porras, "Droidminer: Automated mining and characterization of fine-grained malicious behaviors in android applications," in *European Symposium on Research in Computer Security*. Springer, 2014, pp. 163–182.

8. L. K. Yan and H. Yin, "Droidscope: Seamlessly reconstructing the os and dalvik semantic views for dynamic android malware analysis," in *Proceedings of the 21st USENIX conference on Security symposium*, 2013, pp. 29–29.
9. V. Rastogi, Y. Chen, and W. Enck, "Appsplayground: automatic security analysis of smartphone applications," in *ACM Conference on Data and Application Security and Privacy*, 2013, pp. 209–220.
10. F. A. Narudin, A. Feizollah, N. B. Anuar, and A. Gani, "Evaluation of machine learning classifiers for mobile malware detection," *Soft Computing*, vol. 20, no. 1, pp. 1–15, 2016.
11. Q. Xu, Y. Liao, S. Miskovic, Z. M. Mao, M. Baldi, A. Nucci, and T. Andrews, "Automatic generation of mobile app signatures from traffic observations," in *Computer Communications*, 2015, pp. 1481–1489.
12. S. Wang, Z. Chen, L. Zhang, Q. Yan, and B. Yang, "Trafficav: An effective and explainable detection of mobile malware behavior using network traffic," in *Proc. IEEE/ACM International Symposium on Quality of Service (IWQOS)*, 2016, pp. 1–6.
13. L. Pizzato, T. Rej, T. Chung, I. Koprinska, and J. Kay, "Recon: a reciprocal recommender for online dating," in *ACM Conference on Recommender Systems*, 2010, pp. 207–214.
14. X. Wei, I. Neamtiu, and M. Faloutsos, "Whom does your android app talk to?" in *Global Communications Conference (GLOBECOM), 2015 IEEE*. IEEE, 2015, pp. 1–6.
15. A. Shabtai, L. Tenenboim-Chekina, D. Mimran, L. Rokach, B. Shapira, and Y. Elovici, "Mobile malware detection through analysis of deviations in application network behavior," *Computers & Security*, vol. 43, no. 6, pp. 1–18, 2014.
16. A. Gorla, I. Tavecchia, F. Gross, and A. Zeller, "Checking app behavior against app descriptions," in *Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 1025–1035.
17. Android monkey tool. [online]. Available: http://developer.android.com/tools/help/monkey.html.
18. Tshark - dump and analyze network traffic. [online]. Available:https://www.wireshark.org/docs/man-pages/tshark.html.
19. Y. Yang and J. O. Pedersen, "A comparative study on feature selection in text categorization," in *Fourteenth International Conference on Machine Learning*, 1997, pp. 412–420.
20. Pso tutorial. [online]. Available: http://www.swarmintelligence.org/tutorials.php.
21. Virusshare.com - because sharing is caring. [online]. Available: https://virusshare.com/.
22. Virustotal. [online]. Available: https://www.virustotal.com/.
23. S. Aranganayagi and K. Thangavel, "Clustering categorical data using silhouette coefficient as a relocating measure," in *Conference on Computational Intelligence and Multimedia Applications. International Conference on*, vol. 2. IEEE, 2007, pp. 13–17.
24. R. Perdisci, W. Lee, and N. Feamster, "Behavioral clustering of http-based malware and signature generation using malicious network traces," in *Usenix Conference on Networked Systems Design and Implementation*, 2010, pp. 26–26.
25. S. Wang, Q. Yan, Z. Chen, B. Yang, C. Zhao, and M. Conti, "Detecting android malware leveraging text semantics of network flows," *IEEE Transactions on Information Forensics & Security*, vol. PP, no. 99, pp. 1–1, 2017.