TextDroid: Semantics-based Detection of Mobile Malware Using Network Flows

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

***Abstract*—The wide-spreading mobile malware has become a dreadful issue in the increasingly popular mobile networks. Most of the mobile malware relies on network interface to coordinate operations, steal users' private information, and launch attack activities. In this paper, we propose TextDroid, an effective and automated malware detection method combining natural language processing and machine learning. TextDroid can extract distinguishable features (n-gram sequences) to characterize malware samples. A malware detection model is then developed to detect mobile malware using a Support Vector Machine (SVM) classifier. The trained SVM model presents a superior performance on two different data sets, with the malware detection rate reaching 96.36% in the test set and 76.99% in an app set captured in the wild, respectively. In addition, we also design a flow header visualization method to visualize the highlighted texts generated during the apps' network interactions, which assists security researchers in understanding the apps' complex network activities.**

I. Introduction

With the proliferation of mobile devices, we have entered the mobile era, witnessing a rapidly growing popularity of mobile apps. A recent report [1] shows that the number of apps in the Google Play Store has risen from $16K$ in December 2009 to more than $2M$ in February 2016. However, the uprising of the Android system is greatly impaired by the prevalent Android malware – it is reported that 90% of the 126 apps tested contain at least two vital security vulnerabilities [2]. This frightening statistic reveals the urgency of enforcing mobile app security.

A. Limitation of Prior Art

Existing mobile malware detection methods include static analysis and dynamic analysis. Previous work has employed static analysis to detect privacy leakage, malware, and vulnerabilities in Android apps. However, static analysis is challenged by malware's code polymorphism and code obfuscation, which are used to generate variants of malware to evade detections. Dynamic analysis executes the code to track the malicious behaviors at runtime. Dynamic analysis seems promising, but it is challenging to perform dynamic analysis on resource-constrained smart devices.

In their seminal work, Zhou et al. [3] shows that more than 90% of malware-infected mobile terminals are controlled by botnets through network or SMS commands. Their observations give us insights on exploring apps' network behaviors to identify mobile malware.

B. Motivation

The performance of network traffic based malware detection mechanisms relies on selected traffic features for distinguishing between benign and malicious traffic. Moreover, the selection of features has been conducted based on the empirical knowledge of a specific set of traffic data [4]. However, malicious apps can be strategic and adaptive in generating network traffic, which makes the distinction between benign and malicious traffic a challenging task.

We find great similarities between mobile traffic flow and natural language. HTTP flows can be regarded as a communicating language between client and server. Similar to the analysis of text documents using NLP, we rely on the semantic information extracted from the traffic flows. The N-gram sequence of words in the flow, just like those of natural language, exhibits a highly skewed frequency-rank distribution. Hence, the N-gram sequence can be used as features to identify malicious traffic.

C. Key Contribution

In this paper, we handle the traffic flow's header using two well-known natural language process (NLP) methods: word segmentation and N-gram model. Then, we use a feature selection algorithm to identify meaningful features. These selected features are used to build an SVM classifier for malware detection. To sum up, the main contributions of this paper are listed as follows:

- We develop TextDroid to perform malware detection using NLP methods by treating mobile traffic as documents. TextDroid utilizes an automatic feature selection algorithm based on the N-gram sequences to extract meaningful features.
- TextDroid trains SVM classifier using the extracted features. It achieves an excellent performance with 96.36% detection rate in training data and 76.99% detection rate in testing data collected in the wild.
- We design a word cloud representation of network flows, which allows researchers to visualize flow's head

contents and quickly identify the keywords embedded in the network flows of mobile malware.

II. Related Work

Due to the previously mentioned limitations of static detection and dynamic analysis methods, researchers resort to network traffic generated by malicious apps to analyze and identify these apps.

AntMonitor [5] uses the Android VPN service API to intercept traffic on Android devices and perform traffic analysis. Xu et al. [6] propose a system that can automatically identify mobile apps by continually learning the apps' distinguishable features via HTTP traffic observations.

Several studies utilize text analysis for the purpose of identifying malicious behaviors. Asdroid [7] detects stealthy behaviors in Android apps by UI textual semantics and program behavior contradiction. However, it only uses a few keywords to cover sensitive operations such as "send sms","call phone". WHYPER [8] uses NLP techniques to identify sentences that describe the need for a given permission in the app description. Nan et al. propose a framework called UIPicker [9] for identifying users' personal information on a large scale which is based on a novel combination of NLP, machine learning and program analysis techniques. As for the traffic analysis, an N-gram model in NLP has been used in an automatic network protocol identification system [10]. The proposed system first extracts statistic message format by clustering the N-grams with the same semantics, and then the statistic format is used to classify the raw traffic data. Note that none of the above work focuses on mobile malware detection using network flows.

We utilize an N-gram model in NLP and the semantic correlation in HTTP flow header for mobile malware identification. Among all the features generated by the N-gram sequencing method, a feature selection algorithm is applied to automatically select features with high correlations to malware, which requires no prior knowledge of the HTTP flows.

III. Methodology

We propose a TextDroid system for detecting malicious apps using mobile traffic. The complete process is illustrated in Figure 1. TextDroid segments each flow into a word set (step (a)), and then N-gram generation (step (b)) is adapted from NLP to generate features that can effectively characterize a specific HTTP flow. We then propose a feature selection algorithm to select meaningful features in an automated manner (step (c)) and train an SVM classifier (step (d)) that can automatically determine whether the unknown traffic is benign or malicious (step (e)). Finally, we integrate a flow visualization method by building the word clouds to visualize the headers of HTTP flows from the identified mobile malware (step (f)).

A. Flow Extraction and Segmentation

1) Flow Extraction: We use the methods described in [11] to set up a mobile traffic generation and collection platform. This platform will be used for collecting two types of data: malicious traffic generated by malicious apps, and benign network traffic generated by benign apps.

Since multiple HTTP flows generated by an app are mixed together, we need to extract the individual HTTP flow from the traffic file to facilitate flow analysis. We design an algorithm that can extract each flow and export each flow's header to a document, as described in algorithm 1.

Data: Network traffic data (a pcap file that contains multiple HTTP flows) generated by one app.
Result: Text documents that hold HTTP flows' header.
Initialize the index of HTTP flow to 0;
while *true* **do**
  Use T-shark command to export the header of the HTTP flow of the current index to a text document;
  Get the byte number of this text document;
  if *the byte number of the text document is smaller than a threshold* **then**
    break;
  else
    Save this text document to the target folder;
    Delete the text document at the current location;
  end
  Increase the index of HTTP flow;
end

Algorithm 1: The algorithm for extracting HTTP flow header

2) HTTP Flow Segmentation: The segmentation of the flow header is a challenging task, because there is no standard token (e.g. whitespace or punctuation) for cutting up flows into words. In fact, HTTP flow's header contains many information, including but are not limited to the requesting method, encoding type, requested url, and browser information. Meanwhile, these information is arranged in a particular order. Moreover, many special characters can be included in the HTTP flow, such as " ","/", ":", ";", "&" in the flow. We can split such strings into separated words using these special characters, and finally the header of this flow is separated into a word set of "GET", "dm", "ad-maker.info", "Acc", etc.

B. N-Gram Generation

In the field of NLP, the N-gram is a sub-sequence composed of N elements that are included in a particular sequence which has at least N elements.

N-gram generation module is designed to provide more semantic information for the HTTP flow's header. Figure 2 presents several examples to turn a word set (derived from a flow segmentation) of flows into N-gram sequences. The leftmost part is the initial word set (obtained by splitting the header of flow), where each row represents a flow. At the same time, these words form 1-gram sequence. The middle and rightmost portions are N-gram sequences generated by corresponding word set in each flow. The middle portion represents the 2-gram sequence, and the rightmost portion represents the 3-gram sequence. The N-gram provides contextual information that captures the relationship among multiple words, which helps extract meaningful word sets as features.

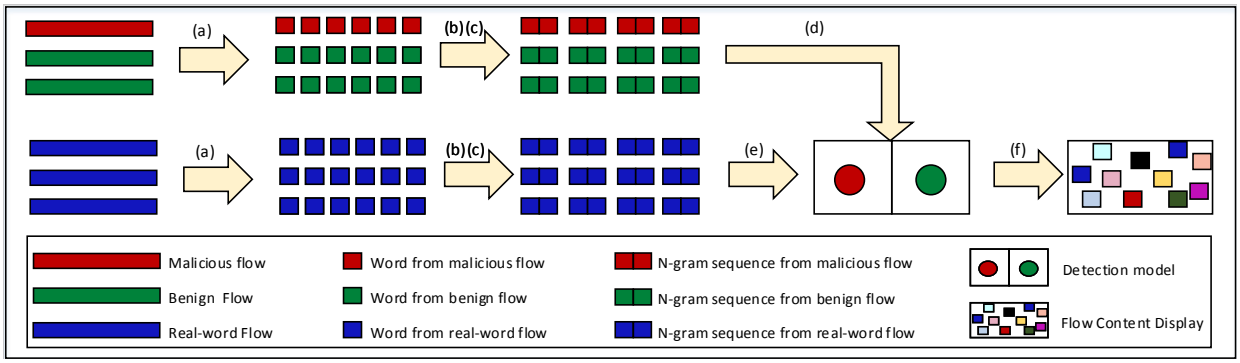


Fig. 1: A schematic depiction of TextDroid

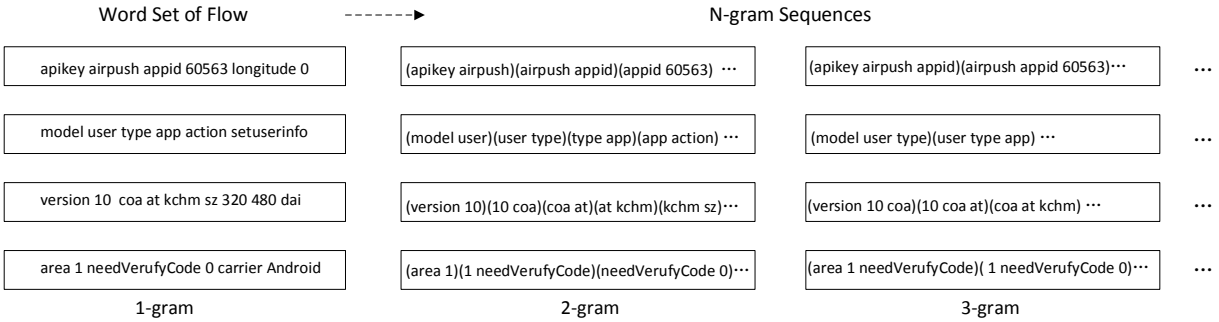


Fig. 2: Example of translating a word set of flow to N-gram sequences

C. Feature Selection

In our feature set, not all N-gram sequences make a contribution to the malware detection model. Those who are not essential will disrupt the normal operation of the algorithm negatively. So we propose to filter out features using Chi-Square test [12], which is a univariate feature selection algorithm. Chi-Square (Chi2) test assigns the features a certain weight to characterize the correlation between the features and the category (i.e., benign or malicious). The higher the test result is, the more relevant the feature is with the particular category.

In this paper, we test whether a specific feature in N-gram sequence set is significant or not based on its occurrences in two opposite data sets (N-gram sequence set from malicious flows and N-gram sequence set from benign flows). More specifically, we are mainly concerned about whether a feature t (an N-gram sequence) and a class c (malicious) are independent of each other. If so, we regard feature t as insignificant to classify class c. In other words, we cannot determine whether a flow belongs to class c using feature t. The formula for calculating the Chi-Squared value of a feature t and class c is defined as follows:

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

where the $N_{e_t e_c}$ refers to the number of occurrences of feature t and class c, $E_{e_t e_c}$ is the expected number of occurrences of feature t and class c when they are independent of each other, and e_t and e_c are set as '1' or '0'. For example, if e_t is taken as '0', it means that the incident does not occur (no feature t exists in one flow), while if it is taken as '1', it indicates that the incident has occurred (a flow contains feature t). After that, we can select top K features (K is an adjustable parameter) for the modeling training.

D. Detection Model Training

1) Words Vectorization. After the feature selection, we will get a bag-of-words as a feature set. Utilizing this bag-of-words, binary values for each flow can be produced. If the given N-gram sequence does exist in the bag-of-words, the value corresponding with the N-gram sequence will be '1'. Otherwise, the value is '0'. This encoding type is also known as one-hot encoding. Through this encoding method, each HTTP flow will be transformed into a numeric vector whose dimension is equal to the length of the bag-of-words.

2) Model Training. To identify malicious traffic and further pinpoint malware by tracing the source app of the malicious traffic, we consider linear Support Vector Machines (SVM) which learns a hyperplane with maximum margin for classification.

The detection model of linear SVM simply maps the feature vector V of a HTTP flow x to the direction of the hyperplane. The corresponding detection function F is given as:

$$F_{(x)} =< W, V >,$$

where W denotes the weight vector, which should be continuously adjusted during the training process. We arrive at a

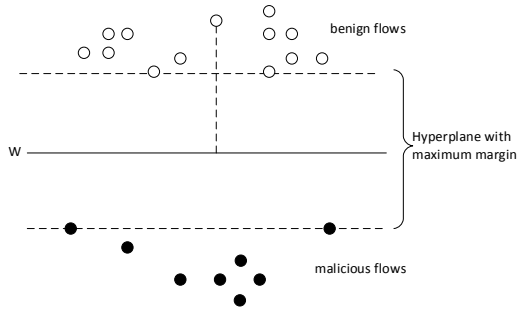


Fig. 3: The schematic diagram of SVM algorithm

final conclusion based on the value of $F_{(x)}$. That is, $F_{(x)} > 0$ (a given value) indicates malicious activity, whereas $F_{(x)} < 0$ corresponds to benign traffic flows, and $F_{(x)} = 0$ indicates the separating hyperplane.

E. Classification in the Wild

The derived SVM model can be used for malware classification. When an unknown traffic file (mixed flows) arrives, unknown traffic file is processed using the method described in Section 3.1 (flow extraction and segmentation) and Section 3.2 (N-gram generation). After vectorizing the features with the bag-of-words feature set, every HTTP flow is transformed into a numeric vector. The numeric vector will be used as inputs to the detection model. Once malicious flow is discovered, the source app of this traffic flow will be tagged as a malware.

F. Flow Header Visualization

The main purpose of this module is to visualize the flow's header of mobile malware to allow researchers to mine more information from the flow. There are three steps:

1) Remove Useless Word. We set up several filtering rules to remove useless words from flow word set: (1) low-frequency words removal. Words appearing only once or rarely involve a great deal of randomness, and will be discarded; (2) high-frequency but common words removal. Some words such as "content-length", "en-us", etc., appear in almost every HTTP flow, and they will also be discarded; (3) stop words removal. We remove stop words (such as "the", "is", "were"), since they do not provide any meanings for understanding the HTTP flows.

2) Calculate Word Weights. We consider the term frequency (TF) as the word weight in the HTTP flow. Term frequency refers to the occurrence number of a given word appearing in one document. As for one word t_i in a specific file f_j, its term frequency TF can be expressed as follows.

$$TF_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}},$$

where the $n_{i,j}$ is the number of word i appearing in the file j, and the denominator $\sum_k n_{k,j}$ is the number of all words appearing in all malicious flows.

3) Visualize Flow's Header. HTTP flow's header is represented in a word cloud format. Word cloud is a set of related words with the corresponding weights. Specifically in our approach, the font size of high term frequency word will be larger than low term frequency one, and the font color of high term frequency word is more vivid than that of low term frequency one.

In order to articulate the visualization process, we give an example word cloud, which is from a malware sample from DroidkungFu family. Figure 5 shows the words in HTTP flow's header and their corresponding term frequency. According to these words' term frequency, we use different colors and fonts to demonstrate all flows' header information, which is shown in Figure 4.

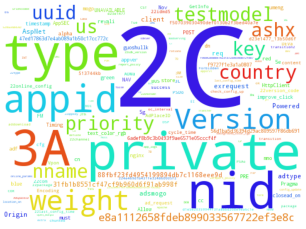


Fig. 4: Flow header visualization

Term	#	TF
2C	13	0.112
type	11	0.095
3A	11	0.095
private	8	0.069
nid	8	0.069
version	7	0.060
...	...	...

Fig. 5: Terms and corresponding TF values

IV. Evaluation

A. Training Data Set

Our initial malicious app set contains 3142 malwares, which is provided by the Drebin project [13]. The initial 4791 benign apps are all from several famous Android application market, such as Google Play Store and third-party application market in China. With a mobile traffic generation and collection platform [11], 48,870 benign HTTP flows generated by these benign apps are obtained. We also get 34,205 HTTP flows generated by malware samples, and then extract 19,881 malicious flows from it according to the malicious destination IP or domain name.

The complete set of malicious apps are divided into 94 families. Table I only shows 10 families' app number, HTTP flow number as well as HTTP flow size generated by these malware samples in each family. As for the benign app samples, Android markets have given them a specific category according to their functions or other rulers. These 4,791 apps are classified into 10 categories. Table II shows the number of samples in each category, the number and the size of HTTP flows generated by these samples.

B. Parameter Settings

There are two main parameters that affect the detection model. One important parameter is the N value in N-gram, and the other parameter is the feature number in feature selection algorithm. The 10-fold cross validation method is used in our experiments to get credible results.

Figure 6 shows the TPR (True positive rate on malicious flows), FPR (False positive rate on malicious flows) and AUC

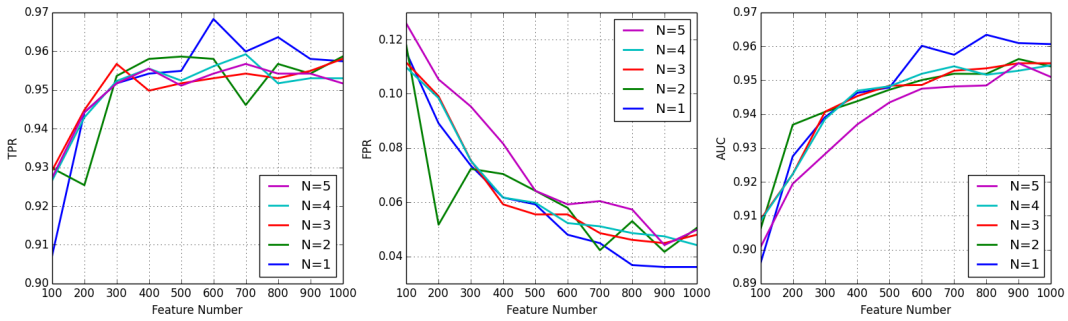


Fig. 6: The TPR, FPR and AUC values using different N values and features numbers

TABLE I: The sample number, HTTP flow number and HTTP flow size of 10 malware families

Family	Malware #	Flow #	Size
Plankton	483	3056	14.9MB
DroidKungFu	427	3708	245MB
GinMaster	302	2138	4.9MB
Opfake	286	200	857KB
BaseBridge	272	275	1.72MB
FakeInstaller	214	188	1.91MB
FakeRun	143	692	3.2MB
FakeDoc	127	663	1.08MB
Adrd	81	204	518KB
Gappusin	57	103	251KB

TABLE II: The app number, HTTP flow number and HTTP flow size of benign apps

Category	App #	Flow #	Size
Reading	744	7653	124MB
NewsAndMagazines	379	7945	188MB
Photography	353	2806	128MB
MediaAndVideo	340	9255	60.7MB
Productivity	258	1730	186MB
Input	146	994	10MB
Tools	102	1115	17.1MB
Social	97	976	61.6MB
Shopping	88	1229	110MB
Sport	19	149	922KB

values at different N values and feature number. From the leftmost sub-graph, we can see that TPR value reaches the highest point, when N equals to 1 and the feature number is 600. However, in order to take both the TPR and FPR values into account, the rightmost sub-graph can help us choose the best model. When the features number is 800, the value of AUC is the maximal using 1-Gram. So the N value is set as 1 and feature number is 800. At this time, model' TPR reaches to 96.36% and FPR is 3.68%.

In addition, it takes about 31.27 seconds to training this detection model with 48,870 benign flows and 19,881 malicious flows. In the identification stage, 84 traffic flows are imported into the model, and 0.33 seconds later, we get the final detection results of all flows. Therefore, the average detection time for each flow is only 0.0039 seconds.

C. Compare with Other Methods

We have selected the static detection method and the mobile traffic based detection methods for comparison with TextDroid.

1) Compare with Other Static Detection Methods

The malware samples we used for training come from the Drebin project [13]. We extract 85 malware samples from this list those are not correctly classified by Drebin. Then, we use our trained model to classify the traffic data generated by these 85 malware samples. In the end, we correctly identify 12 samples. We further analyze these 12 detected samples and find that they are from different malware families, respectively Gapussin, SpyPhone, Glodream, Plankton, DroidSheep and Sdisp. This shows the superior performance of TextDroid compared with Drebin.

2) Compare with Other Traffic-based Detection Methods

We select the identified features from HTTP request header according to TrafficAV [14] and then train an SVM model. Eventually, we get 24,098 malicious HTTP requests and 94,898 benign HTTP requests from our traffic data. Using 10-fold cross validation, TrafficAV achieves 91.01% malware detection rate.

In addition, we also compare TextDroid with DroidClassifier [15], which is another malware detection method using mobile network traffic. DroidClassifier designs a score-based classification method according to these HTTP header fields and the malware detection rate is 94.33% on our dataset. Note that TextDroid achieves 96.36% detection rate as shown previously. The results show that TextDroid can achieve a higher detection rate than both DroidClassifer and TrafficAV. The reason for this result may be that TextDroid considers all HTTP flow header's information, rather than a few request fields.

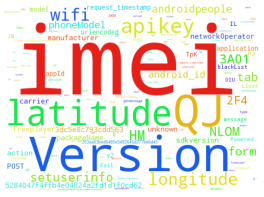
D. Evaluation on Wild Apps

Using a crawler, multiple apps are downloaded from several Android application markets. Traffic data generated by these apps in first five minutes is collected by a traffic generation and collection platform. These traffic data is processed and then fed into the detection model, which produces the test result, and an app will be marked as malware if it contains malicious flows.

Among 752 apps in the test set, 452 apps are malicious as confirmed by the VirusTotal's detection report. In the end, our detection model can identify 348 apps out of the 452 apps, which verifies our model's strength in identifying malware from wild apps.

TABLE III: Detection rates of TextDroid and other anti-virus scanners

	TextDroid	AegisLab	Antiy-AVL	NANO-Antivirus	McAfee	F-Secure	BitDefender	Sophos
Wild apps set	76.99%	84.96%	71.46%	69.25%	66.81%	55.97%	44.47%	41.15%



(a) Example of a malware's flow header

(b) Example of a benign app's flow header

Fig. 7: Examples of malware and benign app's network traces

We also compare TextDroid's performance against seven selected anti-virus scanners on our test set. The detection rates of each scanner are presented in Table III. On this wild app set, TextDroid provides the second best performance with a detection rate of 76.99% (348/452) and outperforms 6 out of the 7 anti-virus scanners.

E. Flow Header Visualization

We implement the visualization of malicious flows' header using Python. Researchers can gain a better understanding of the information in the flow using such visualization. To show the benefits of visualizing malware's flow header, we illustrate the network traces from two apps. Figure 7(a) shows the flows' header from a malicious app, while Figure 7(b) comes from one benign app. The most prominent word in Figure 7(a) is the red word "imei" and blue "version". The "imei" is the unique identifiers of the phone. In addition, there are some other conspicuous words, such as "longitude", "latitude", "wifi", "apikey" and so on. These terms are closely related to the user's device information and personal information. However, the benign network traffic only contains some common words, such as "mobile", "application", "web51net" and so on. Surprisingly, researchers can even infer the maliciousness of an app simply by examining the visualization of the headers of HTTP flows.

V. Conclusion

In this paper, we present TextDroid, a system for mobile malware detection using network traffic flows. TextDroid treats every HTTP flow as one document. Word segementation, N-gram sequence and feature selection algorithm are utilized to facilitate the extraction of meaningful features from traffic flows. Then, we created a detection model with SVM algorithm that achieves a malicious traffic detection rate of 96.36% in the test set. While applying the detection model into a real world environment, TextDroid can detect 76.99% of all the malicious apps. Moreover, we designed a visualization tool using word cloud, which uses the term frequency value of every word as a weight to represent the malware's flow header in a meaningful manner.

Acknowledgement

This work was supported by the National Natural Science Foundation of China under Grants No.61672262, No.61573166, No.61472164 and No.61572230, the Natural Science Foundation of Shandong Province under Grants No.ZR2014JL042 and No.ZR2012FM010, the Shandong Provincial Key R&D Program under Grants No.2016GGX101001. This work is also supported in part by NSF grant CNS-1566388.

References

[1] "Google play: number of available apps 2009-2016," http://www.statista.com/statistics/266210/, Accessed at Jan. 15, 2017.

[2] "Infographic report: State of application security," https://securityintelligence.com/media/infographic-report-state-of-application-security/, Accessed at Jan. 15, 2017.

[3] X. Jiang and Y. Zhou, "Dissecting android malware: Characterization and evolution," in *IEEE Symposium on Security & Privacy*, 2012, pp. 95–109.

[4] 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.

[5] A. Le, J. Varmarken, S. Langhoff, A. Shuba, M. Gjoka, and A. Markopoulou, "Antmonitor: a system for monitoring from mobile devices," in *Proceedings of the 2015 ACM SIGCOMM Workshop on Crowdsourcing and Crowdsharing of Big (Internet) Data*. ACM, 2015, pp. 15–20.

[6] Q. Xu, Y. Liao, S. Miskovic, and Z. M. Mao, "Automatic generation of mobile app signatures from traffic observations," in *Proc. of INFOCOM*, 2015.

[7] J. Huang, X. Zhang, L. Tan, P. Wang, and B. Liang, "Asdroid: detecting stealthy behaviors in android applications by user interface and program behavior contradiction," in *Prof. of ICSE*, 2014, pp. 1036–1046.

[8] R. Pandita, X. Xiao, W. Yang, W. Enck, and T. Xie, "Whyper: towards automating risk assessment of mobile applications," in *Usenix Conference on Security*, 2013, pp. 527–542.

[9] Y. Nan, M. Yang, Z. Yang, S. Zhou, G. Gu, and X. F. Wang, "Uipicker: user-input privacy identification in mobile applications," in *Usenix Conference on Security Symposium*, 2015.

[10] X. Yun, Y. Wang, Y. Zhang, and Y. Zhou, "A semantics-aware approach to the automated network protocol identification," *IEEE/ACM Transactions on Networking*, vol. 24, no. 1, pp. 1–1, 2015.

[11] D. Cao, S. Wang, Q. Li, Z. Cheny, Q. Yan, L. Peng, and B. Yang, "Droidcollector: A high performance framework for high quality android traffic collection," in *2016 IEEE Trustcom/BigDataSE/ISPA*, Aug 2016, pp. 1753–1758.

[12] Y. Yang and J. O. Pedersen, "A comparative study on feature selection in text categorization," in *Fourteenth International Conference on Machine Learning*, 1998, pp. 412–420.

[13] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck, "Drebin: Effective and explainable detection of android malware in your pocket." in *Prof. of NDSS*, 2014.

[14] 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 *Prof. of IWQoS*, 2016, pp. 1–6.

[15] Z. Li, L. Sun, Q. Yan, and W. Srisa-an, "Droidclassifier: Efficient adaptive mining of application-layer header for classifying android malware." in *Proc. of Securecomm*, 2016.