A Fast and Effective Detection of Mobile Malware Behavior Using Network Traffic

Anran Liu[1,2], Zhenxiang Chen[1,2(✉)], Shanshan Wang[1,2], Lizhi Peng[1,2], Chuan Zhao[1,2], and Yuliang Shi[3]

[1] School of Information Science and Engineering, University of Jinan, Jinan 250022, Shandong, China
czx@ujn.edu.cn

[2] Shandong Provincial Key Laboratory of Network Based Intelligent Computing, Jinan 250022, Shandong, China

[3] School of Software, Shandong University, Jinan 250100, Shandong, China

Abstract. Android platform has become the most popular smartphone system due to its openness and flexibility. Similarly, it has also become the target of numerous attackers because of these. Various types of malware are thus designed to attack Android devices. All these cases prompted amounts of researchers to start studying malware detection technologies and some of the groups applied network traffic analysis to their detection models. The majority of these models have considered the detection primarily on network traffic statistical features which can distinguish malicious network traffic from normal one. However, when faces a large amount of network traffic on the detection stage, especially some of the network flows are quite huge as a result of containing too many packets, feature extraction can be extremely time consuming. Therefore, we propose a malware detection approach based on TCP traffic, which can quickly and effectively detect malware behavior. We first employ the traffic collection platform to collect network traffic generated by various apps. After preprocessing (filtering and aggregating) the collected network traffic data, we get a large number of TCP flows. Next we extract early packets' sizes as features from each TCP flow and then send it to detection model to get the detection result. In our method, the time it takes to extract features from 53108 network flows is reduced from 39321 s to 18041 s, which is a reduction of 54%. Meanwhile, our method achieves a detection rate of 97%.

Keywords: Malware detection · Network traffic · Machine learning

Supported by the National Natural Science Foundation of China under Grants No. 61672262, No. 61573166 and No. 61572230, the Shandong Provincial Key R&D Program under Grant No. 2016GGX101001, No. 2016GGX101008, No. 2018CXGC0706 and No. 2016ZDJS01A09, the TaiShan Industrial Experts Programme of Shandong Province under Grants No. tscy20150305, CERNET Next Generation Internet Technology Innovation Project under Grant No. NGII20160404.

J. Vaidya and J. Li (Eds.): ICA3PP 2018, LNCS 11337, pp. 109–120, 2018.
https://doi.org/10.1007/978-3-030-05063-4_10

1 Introduction

Due to the portability of smartphones, more and more people prefer to using smartphones to handle everything that can be handled, and smartphones therefore carry a large amount of users' privacy information. Attackers thus began to attack smartphones to steal users' private information, and use it to make profits. The most common way they use to attack smartphones is to design various types of malware, and the way malware is developed differs depending on the operating system.

Smartphones can be classified into several different categories according to their operating systems. Among them, smartphones equipped with Android operating systems have become the most widely accepted and used devices because of their openness and the diversity of apps they provided. According to the report released by Gartner in May 2017 [2], the market share of Android platform has reached 86.1%. In order to attack more devices and obtain more users' privacy information, most attackers thus consider Android platform as their target. Recent report has pointed that more than 99% of malware designed for mobile devices targets Android devices [1].

In March 2018, the number of the apps in Google Play has reached 3.6 million [3], which will only be much larger in third-party market. It is very difficult for users to identify whether an app is malicious or not when faced with such a large number of apps. Therefore, the need to detect Android malware has risen sharply. Amounts of researchers have thus studied malware detection techniques which can be roughly categorized into static and dynamic groups. In addition, with more and more malware relying on network interfaces to interact with attackers, network traffic based detection starts to be used to identify Android malware. Compared to static analysis detection, like [6,11,13], which is difficult to identify malicious variants due to code obfuscation [15], and dynamic analysis detection, like [10] and [21], which needs to deploy a detection environment on the mobile phone, detection based on network traffic doesn't need to deploy on the Android devices and is adaptable to the variants. Network traffic based detection extracts features from network traffic and employs machine learning method to identify mobile malicious behavior. But when faces a large amount of network traffic on the detection stage, especially some of the network flows are huge as a result of containing too many packets, feature extraction can be extremely time consuming. And this will directly increase the time overhead of detection stage. Therefore, we propose our method, a fast and effective detection approach of mobile malware behavior using network traffic.

In our method, we first collect a large amount of network traffic using a traffic collection platform and then filter all the non-TCP traffic out. After that, we aggregate the remaining traffic into flows and perform feature extraction on each flow. Finally, we send the extracted feature tuple to detection model to get the detection result.

The detection model used in our method is a model based on TCP packet size feature. In consideration of the fact that encrypting network traffic between application layer and transport layer is gradually becoming a popular trend, we

choose TCP, one of the most popular transport layer protocols, to analyze. And in order to deal with the time consumption problem we pointed out above, we chose the sizes of the first few packets of a network flow and some statistics of those sizes as features since we won't have to deal with the entire network flow when we extract a feature tuple from it. Furthermore, for encrypted network traffic, we won't have to separate them out and decrypted them for analysis, but to treat them equal to unencrypted network traffic and then extract the same features from them. This is to some extent also reduces the overall detection time as well.

In this paper, we mainly make the following contributions:

- **Network traffic based malware detection in a time saving way.** Our experimental results show that our method can effectively identify mobile malicious behavior with a low time consumption.
- **A lightweight detection system.** We proposed a prototype system which allows us to identify mobile malicious behavior on the server side without consuming the resources of users' mobile devices.
- **An effective encrypted traffic classification method without infringing users' privacy.** Our method can identify encrypted network traffic quite precisely without decrypted those encrypted network flows, which can save a lot of time and also protects users' privacy.

The rest of the paper is organized as follows: related work is introduced in Sect. 2. Section 3 presents the methodology of our method in detail and the evaluation of our method is introduced in Sect. 4. Section 5 concludes the paper.

2 Related Work

The network traffic analyzed by the researchers in traffic-based malware detection method is roughly divided into two parts, namely application layer traffic and transport layer traffic.

Application Layer Traffic Based Analysis. In mobile devices, most apps use DNS protocol for domain name resolution and HTTP protocol for data transfer. Therefore, almost all application-level traffic based malware detection methods are implemented by analyzing DNS traffic or HTTP traffic. Wei et al. [20] proposed a malware detection method based on domain name resolution behavior. Their method can analyze DNS flows generated by apps automatically and determine whether the app is malicious based on the geospatial information of the resolved IP address. Zaman et al. [22] record the domain names accessed by the app through the URL in HTTP request message and then compare them with the known domain name blacklist. The app that communicates with the domain name in the blacklist is considered as a malicious app. Wang et al. [19] treat each HTTP flow generated by an app as a text file, and then leverage natural language processing to extract text-level features that can

distinguish between normal apps and malware. Although these methods achieve high detection rates, they are severely limited by SSL/TLS encryption. Attackers can use SSL/TLS encryption to hide the information carried by application layer traffic easily.

Transport Layer Traffic Based Analysis. The two most important protocols at transport layer are TCP and UDP, and the majority of the transport layer traffic is transmitted over TCP. Thus, researchers usually leverage TCP traffic to explore malware detection method, and most of them analyze the statistical features of TCP flows [4]. Shabtai et al. [17] proposed anomaly-based malware detection method with automatic update capability. This method uses statistical features extracted from TCP flows to generate normal app traffic patterns and then identify all the apps whose traffic patterns are deviated from the normal pattern as malware. Wang et al. [18] use six TCP statistical features and combined with machine learning algorithm to detect Android malware. However, none of these methods takes into account the time consumption problem in feature extraction phase in malware detection process.

Arora et al. [5] proposed an algorithm to prioritize network traffic features, which gave a high detection accuracy and reduced both training and testing time. However, some of the final 9 features chosen in that paper, such as Average Packet Size, Packets Sent per Flow and Maximum Packet Size, still are flow statistic features which need to traverse the whole network flow to be extracted, and this will increase the time cost of feature extraction phase in detection process. Lizhi et al. [14] pointed out that the sizes of early stage packets are effective enough to achieve ideal traffic identification performances. Bernaille et al. [7] used the first four to five packets to classify TCP-based apps and achieved high accuracy. Since malware detection is a subcategory in traffic classification, we decided to use early packet sizes as features to detect Android malware behavior.

3 Methodology

In our method, network traffic generated by the apps are collected from the traffic collection platform. After preprocessing the collected network traffic data, we get a large number of TCP flows. Next, we extract the sizes of the early packets of a network flow and some statistics of those sizes as features and then send the feature tuple to detection model to get the detection result. The whole detection approach is detailed in the following sections.

3.1 Traffic Collection

Network Trace Capture. In our method, we use the active traffic generation and collection platform proposed in [9] to collect traffic data. The platform consists of four parts, namely foundation platform, traffic generator, traffic collector and network proxy/firewall respectively. Foundation platform is built based on Android Virtual Device (AVD), and traffic generator is designed to install and

activate malware samples to generate network traffic automatically. The function of traffic collector is to collect inbound and outbound network traffic with tcpdump tool, and then the traffic mirroring technology is employed to mirror traffic to the server side. During the whole process, the attack behavior is carefully monitored and controlled by proxy/firewall. On the server side, we collect traffic traces and store them in pcap format for the next preprocessing step.

Traffic Data Preprocessing. Since our work only focuses on TCP traffic, we need to filter the traffic data collected in the previous step and remove all packets whose transport layer protocol is not TCP. After that, we aggregate the packets in each traffic trace according to the definition of TCP flow, and store all flows in pcap format separately. The whole process is implemented using a combination of Python script and T-shark command.

3.2 Feature Extraction

The features used in our method are the sizes of the first n packets of a TCP flow and 4 statistics (i.e. average, standard deviation, minimum and maximum values) of the sizes. The number of selected packets is a parameter that is to be determined and we will describe the determination of this parameter in detail in Sect. 4.2. It should be noted that the order of the features extracted from a TCP flow must be consistent with the order of the packets. That is, the first feature in the feature tuple is the size of the first packet in this TCP flow, the second feature is that of the second packet, and so on. Beyond this, these four statistical features are also arranged in order in the feature tuple. Namely, the (n+1)st feature in the feature tuple is the average of the first n features, the (n+2)nd feature is the standard deviation of the first n features, and so on. In addition, if a TCP flow contains fewer than n packets, we will use an integer 0 to fill in the feature tuples extracted from this flow (the last 4 features of the feature tuple are still the values of these 4 statistics). Thus, the length of the feature tuples extracted from each network flow is n+4. The entire feature extraction process is illustrated in Fig. 1.

We chose the first few packets sizes of a TCP flow as features for the following reasons. On the one hand, Este et al. [12] had proved that packet size is the most effective feature for early stage network traffic identification; on the other hand, we only have to traverse the first few packets of a TCP flow and ignore the rest part of the TCP flow to get features, which will significantly reduce the time cost of feature extraction process.

3.3 Learning-Based Detection

Machine learning can be used to automatically discover the rules by analyzing data, and the rules are helpful for us to predict unknown data. In our research, we leverage machine learning method to analyze features extracted from TCP flows and generate the rules which can determine whether a flow is malicious or

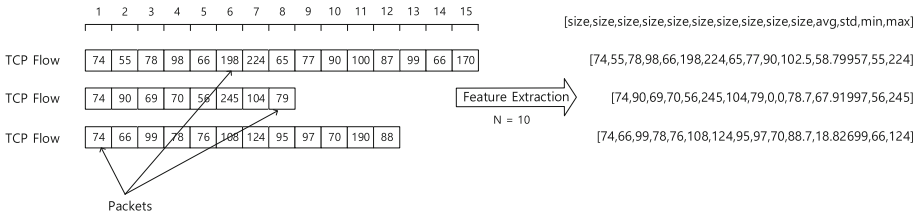

Fig. 1. Feature extraction process.

not. If we determine that the flow is malicious based on the generated rules, the app corresponding to this flow is then considered to be malware. Through this process, we can achieve the purpose of malware detection.

The learning method we chose is Random Forest [8], which uses several decision trees to train the data set and then gets the prediction rule. The construction process of Random Forest is roughly as follows. First, bootstrapping method [16] is used to randomly select samples from the original training set, which is a resampling with replacement. This procedure will be repeated several times to generate multiple training sets. The number of the training sets is a parameter. And then for each training set, a decision tree will be generated. It should be noted that, for each node of the decision tree in a Random Forest, the feature set used for the selection of the optimal feature is only a subset of the feature set of this node. Finally, each decision tree will vote for the prediction result and the result is a majority vote of all single predictions given individually by each decision tree. In our method, we built the classifiers in Python using the scikit-learn machine learning libraries, using $criteria = gini$, $max\_features = auto$, and $n\_estimators = 10$ as parameters.

3.4 Lightweight Detection Architecture

The prototype system shown in Fig. 2 is a lightweight detection system, which only requires users to install an app on their mobile devices and won't need to get root permissions for we just identify the malicious behavior by the network traffic mirrored from the user side. After the detection, the final result will be returned to the app.

Additionally, in order to improve model's adaptability to new types of malware, we designed a model updating framework. We store the traffic and the labels we predicted on the storage server. After collecting a certain amount of labeled flows, we cluster these flows and check the labels of each cluster. If all the flows in a cluster belong to the same category, we will randomly select several flows from this cluster and then detect them manually. If all manually detected flows also belong to this category, then we add all the flows in the cluster to training set and update the model. It should be noted that the model updating process is off-line and therefore does not occupy any user's mobile device resources.

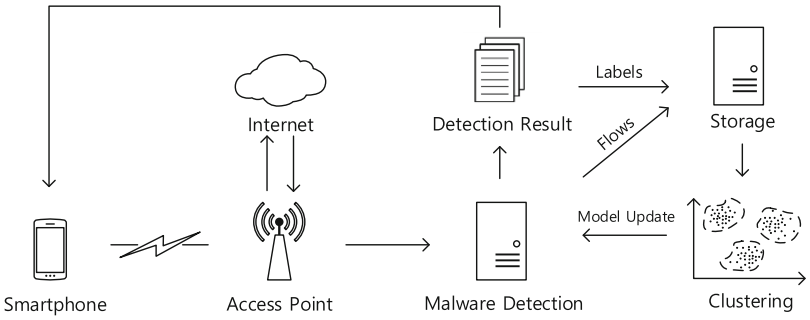

Fig. 2. Lightweight detection architecture.

By using this detection architecture, we can detect multiple apps on a single device, as well as multiple apps on multiple devices at the same time. Within the LAN (local area network), diverse apps might execute network behavior simultaneously, therefore we can collect the network traffic generated from them and then identify these apps in a short time. In addition, the features we used in our method can also guarantee the real-time performance of the detection, for we only need the sizes of early packets in a TCP flow.

4 Evaluation

4.1 Data Sets

Our malicious apps are downloaded from public malware database, VirusShare. Our normal apps are downloaded from multiple popular app markets by app crawler. Consider there might be some potential malicious apps in app markets which can cause the impurity of our normal app set, thus all the normal apps we downloaded from the markets are sent to VirusTotal to test whether these apps are malicious or not and then the apps whose test results are normal will be added to our normal app set. We get a normal app set of 8321 samples and a malware set of 2839 samples.

In order to train and evaluate the detection performance of our model, we use the automatic mobile traffic collection system which is introduced in Sect. 3.1 to collect traffic data. For traffic generated by normal apps, we label it as benign. Since most malware are taking the form of repackaging malicious behavior into normal apps, the traffic generated by malware contains both malicious traffic and normal traffic (most of which is normal traffic and only a small amount of traffic is malicious). In order to improve the accuracy of our labels, we extract the target IP filed of each flow generated by malware and upload it to VirusTotal. If the filed is detected as malicious, we will label the corresponding flow as malicious.

Finally, we get 1.03 GB network traffic data generated by normal apps as well as 219 MB malicious network traffic data, and then we extract features from those traffic data. Finally, 53108 feature tuples are extracted from the traffic we collected to train and test the model.

4.2 Parameter Determination

There is a parameter which have to be determined in feature extraction stage, namely how many packets are needed in a network flow can make the model trained by packet size feature get the best performance at a low time consumption. If the parameter value is too small, then the detection model may suffer from under-fitting; if the parameter is too large, it will certainly increase the time consumption. Therefore, the determination of the parameter is a trade-off between time consumption and detection performance. Thus, we conducted several sets of experiments to determine the optimal value of this parameter.

The first set of experiments we conducted was to determine a rough range of the parameter. Firstly, for each instance in the data set, we extract the size of the first 10, 30, 50, 100, and 200 packets separately and record the time cost of each extraction process. Considering that for most TCP flows, the number of packets contained in it is usually less than 200, so as the value of the parameter (number of packets) further increased, early packet feature extraction will be the same with the extraction of the entire flow. And thus, we only took a maximum of 200.

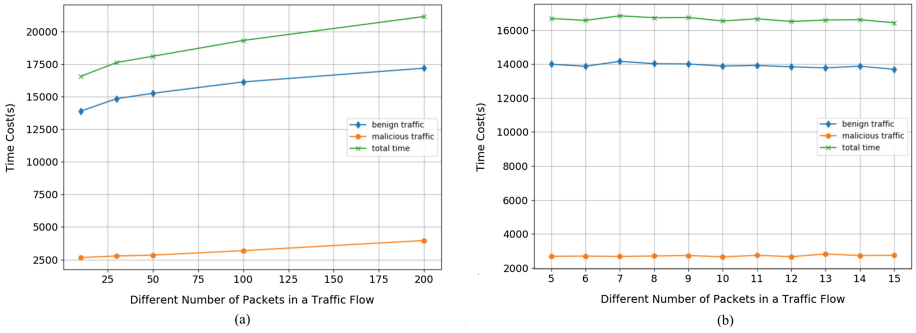

Fig. 3. Time costs of feature extraction process. (a) Time costs of the extraction of the packet sizes of first 10, 30, 50, 100 and 200 packets. (b) Time cost of the extraction of the packet sizes of first 5–15 packets.

The time we recorded is shown in Fig. 3(a). From this figure we can clearly see that there is a growing trend with the increment of the parameter and when the number of packets is 10, the time cost is the minimum. Thus in the next set of experiments, we extracted the size of the first 5–15 packets from each instance in the data set separately, and generated 11 feature sets. Choosing to extract the size of the first 5–9 packets is due to time considerations while the extraction of the size of the 11–15 packets is in consideration of the model performance. Similarly, we also recorded the time required to generate each feature set and the results are shown in Fig. 3(b).

From Fig. 3(b) we can find out that there is no obvious trend in this figure, and the data is fluctuating up and down without an optimal value. Additionally,

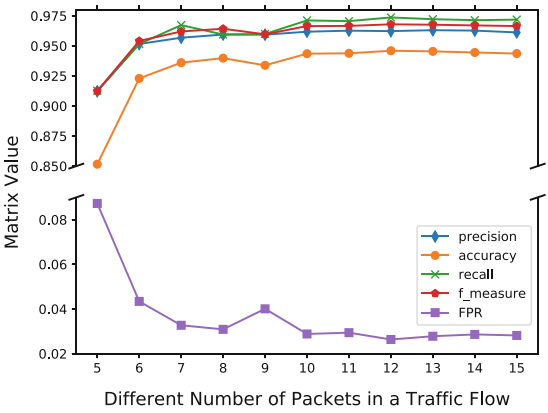

Fig. 4. Matric values of each parameter.

the time cost of generating the sizes of first 5 packets feature set even is a little larger than the time cost of generating the sizes of first 15 packets feature set. Thus, in this session, the time cost indicator is not taken into account.

Next, we trained and tested 11 detection models using ten-fold crossover method on the extracted 11 feature sets. We measured the results in terms of five indicators, including accuracy, precision, recall, F-measure score and FPR, and the detection performance is illustrated in Fig. 4. We can see from the Fig. 4 that when the value of the parameter is 12, five indicators have all reached the optimal value (the first four indicators are the highest under all parameters, the last indicator is the lowest under all parameters). Therefore, we chose 12 as the value of the parameter.

4.3 Comparative Evaluation

In order to evaluate the performance of our detection model, we reconstructed the flow statistics model proposed by TrafficAV [18] to compare with the proposed model, for the statistical features it used are typical and required to traverse the entire flow to be extracted. Our evaluation is conducted from two aspects of time performance and detection performance.

Time Performance. We extract packet size feature set from the whole data set and recorded the time consumption in the meanwhile. Additionally, we extract the flow statistic features used in TrafficAV from our data set and also record time consumption as a control. We didn't add the time it took to train and test the detection model because when compared with the time cost in feature extraction phase, the time cost in training phase and testing phase only account for a small proportion.

These two time records are shown in Fig. 5(a). We can see that our method greatly reduces the time required for feature extraction. It takes us 27324 s to

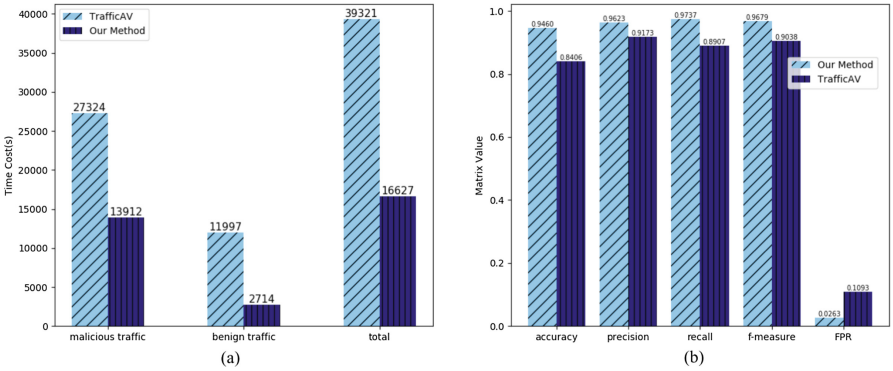

Fig. 5. Comparative evaluation. (a) Time performance. (b) Detection performance.

extract flow statistic features used in TrafficAV from the whole data set, and the number is only 15009 in our method, which is a reduction of 54%.

We believe that when there are more huge flows which contains great amounts of packets, the advantage of our method will be more prominent since if we want to extract the flow statistic features used in TrafficAV from a network flow, we have to traverse the entire flow, while extracting the features used in our method only requires traversing the first few packets of this network flow.

Detection Performance. We employed ten-fold crossover method to evaluate the detection performance of our detection model. Next, we implemented the flow model proposed in TrafficAV based on our data set, and employed ten-fold crossover method to evaluate this detection model as well. The detection performance of these two models is shown in Fig. 5(b). It shows that our method also get a better detection performance than TrafficAV does.

4.4 Encrypted Network Traffic Classification

We collected 735 benign TLS flows and 4 malicious TLS flows as test data set in order to estimate our method's ability to identify encrypted network flows. We extracted feature tuples from these encrypted network flows and then use the model we have trained by the data set we introduced in Sect. 4.1 to classify these network flows. These two data sets are non-overlapping.

At last, our model identified 733 encrypted network flows correctly, including 730 benign TLS flows and 3 malicious TLS flows. This illustrates our method has a certain ability to classify encrypted network flows. In addition, through our method, we won't have to separate the encrypted traffic from a pile of traffic and then analyze them separately in detection stage. Instead, we treat these network traffic equal to unencrypted traffic, extract the same features from them and then send feature tuples to the model for detection. In this respect, our method saves the time as well.

5 Conclusion

The increasing number of Android malware brings mobile users a elevating security risk, and makes the detection of mobile malware a greater challenge. Multiple methods are thus proposed to identify malicious behavior of Android apps and network traffic based method is one of the most popular methods. However, when faces a large amount of network traffic, feature extraction phase in detection process can be extremely time consuming. In this paper, we propose a method which is able to identify malware without violating user's privacy, and can achieve relatively rapid detection on the premise of ensuring a high detection rate. We proved that: by using packet size based analysis, we can quickly detect Android malware behavior with a high accuracy. Furthermore, we designed a device-level lightweight prototype system which can identify a collection of malware on multiple mobile devices in a short time and have the self-update ability. In the future, we will make our efforts to implement the prototype system and try to extend our work to recent Android malware samples.

References

1. Another reason 99% of mobile malware targets androids (2017). https://safeandsavvy.f-secure.com/2017/02/15/another-reason-99-percent-of-mobile-malware-targets-androids/
2. Gartner: Q1 worldwide smartphone sales growth 9% (2017). http://smartcity.asmag.com.cn/xfdz/4345.html
3. Number of available applications in the google play store from December 2009 to March 2018 (2018). https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/
4. Arora, A., Garg, S., Peddoju, S.K.: Malware detection using network traffic analysis in android based mobile devices. In: 2014 Eighth International Conference on Next Generation Mobile Apps, Services and Technologies (NGMAST), pp. 66–71. IEEE (2014)
5. Arora, A., Peddoju, S.K.: Minimizing network traffic features for android mobile malware detection. In: Proceedings of the 18th International Conference on Distributed Computing and Networking, p. 32. ACM (2017)
6. Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K., Siemens, C.: DREBIN: effective and explainable detection of android malware in your pocket. In: Ndss, vol. 14, pp. 23–26 (2014)
7. Bernaille, L., Teixeira, R., Akodkenou, I., Soule, A., Salamatian, K.: Traffic classification on the fly. ACM SIGCOMM Comput. Commun. Rev. **36**(2), 23–26 (2006)
8. Breiman, L.: Random forests. Mach. Learn. **45**(1), 5–32 (2001)
9. Chen, Z., et al.: A first look at android malware traffic in first few minutes. In: 2015 IEEE Trustcom/BigDataSE/ISPA, vol. 1, pp. 206–213. IEEE (2015)
10. Enck, W., et al.: TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. ACM Trans. Comput. Syst. (TOCS) **32**(2), 5 (2014)
11. Enck, W., Ongtang, M., McDaniel, P.: On lightweight mobile phone application certification. In: Proceedings of the 16th ACM Conference on Computer and Communications Security, pp. 235–245. ACM (2009)

12. Este, A., Gringoli, F., Salgarelli, L.: On the stability of the information carried by traffic flow features at the packet level. ACM SIGCOMM Comput. Commun. Rev. **39**(3), 13–18 (2009)
13. Felt, A.P., Chin, E., Hanna, S., Song, D., Wagner, D.: Android permissions demystified. In: Proceedings of the 18th ACM Conference on Computer and Communications Security, pp. 627–638. ACM (2011)
14. Lizhi, P., Bo, Y., Yuehui, C., Tong, W.: How many packets are most effective for early stage traffic identification: an experimental study. China Commun. **11**(9), 183–193 (2014)
15. Moser, A., Kruegel, C., Kirda, E.: Limits of static analysis for malware detection. In: 2007 Twenty-Third Annual Computer Security Applications Conference, ACSAC 2007, pp. 421–430. IEEE (2007)
16. Opitz, D.W., Maclin, R.: Popular ensemble methods: an empirical study. J. Artif. Intell. Res. (JAIR) **11**, 169–198 (1999)
17. Shabtai, A., Tenenboim-Chekina, L., Mimran, D., Rokach, L., Shapira, B., Elovici, Y.: Mobile malware detection through analysis of deviations in application network behavior. Comput. Secur. **43**, 1–18 (2014)
18. Wang, S., et al.: TrafficAV: an effective and explainable detection of mobile malware behavior using network traffic. In: 2016 IEEE/ACM 24th International Symposium on Quality of Service (IWQoS), pp. 1–6. IEEE (2016)
19. Wang, S., Yan, Q., Chen, Z., Yang, B., Zhao, C., Conti, M.: Detecting android malware leveraging text semantics of network flows. IEEE Trans. Inf. Forensics Secur. **13**(5), 1096–1109 (2018)
20. Wei, T.E., Mao, C.H., Jeng, A.B., Lee, H.M., Wang, H.T., Wu, D.J.: Android malware detection via a latent network behavior analysis. In: 2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), pp. 1251–1258. IEEE (2012)
21. Yan, L.K., Yin, H.: DroidScope: seamlessly reconstructing the OS and dalvik semantic views for dynamic android malware analysis. In: USENIX Security Symposium, pp. 569–584 (2012)
22. Zaman, M., Siddiqui, T., Amin, M.R., Hossain, M.S.: Malware detection in android by network traffic analysis. In: 2015 International Conference on Networking Systems and Security (NSysS), pp. 1–5. IEEE (2015)