DroidCollector: A High Performance Framework for High Quality Android Traffic Collection

Dong Cao†, Shanshan Wang†, Qun Li†, Zhenxiang Chen†*, Qiben Yan‡, Lizhi Peng†and Bo Yang†
†School of information science and engineering University of Jinan, Jinan, Shandong, China, 250022
‡University of Nebraska Lincoln, NE, USA, 68588
*Corresponding author, Email: czx@ujn.edu.cn

***Abstract*—In this mobile era, people have become increasingly dependent on smart devices. Smartphones have emerged as the most popular smart computing device. However, numerous security issues affecting smartphones have been exposed. In recent years, mobile network traffic based approaches have been proposed to identify malware malicious behaviors, but these approaches, especially the approaches using machine learning methods are largely constrained by the difficulty of mobile traffic dataset collection. Without sufficient and effective mobile traffic dataset, research focusing on mobile network traffic will be hindered. This study introduces DroidCollector, a high performance framework for high quality Android traffic collection. This framework leverages multithreading to perform active and automatic network traffic collection. Using this framework, we collect 808 MB and 330 MB traffic data generated by 6000 benign apps and 5560 malicious apps in a short period of time, respectively. The collected high quality traffic is mostly generated from apps and with little irrelevant traffic. We also apply machine learning algorithm on the extracted traffic features to identify malicious network behaviors. The experimental result shows that it can achieve a malicious traffic detection rate of 98% on average.**

I. Introduction

The daily use of mobile phones has increased significantly since the dawn of mobile era. However, mobile security can be easily compromised by the malicious software, i.e. mobile malware. Thus, the mobile phones should be effectively protected against mobile malware. Researchers have made great efforts on identifying malware in mobile app markets [1], but a wide variety of mobile malware still find their way into the mobile app markets, especially the third-party mobile app markets.

It is reported that more than 90% malware performs malicious behaviors via network interface [1]. For instance, Sarma et al. [2] analyzed the permission requests of Android apps among 150,000 apps and discovered that 68.50% of benign apps require network access, whereas 93.28% of malware require network access. Similarly, [3] also shows that 93% of malware requests network connectivity among 2000 malware samples. Therefore, researchers can resort to network traffic to detect Android malware. Many researches Start from the mobile network traffic, and then extract malware's behavioral characteristics in network traffic to distinguish them from a large amount of benign traffic. Nevertheless, research efforts have been limited by the lack of sufficient mobile malware traffic datasets.

In this paper, an Android malware traffic generation and collection framework called DroidCollector is designed and implemented. This framework captures traffic from mobile apps, which are actively installed on each collection machine. The framework controls traffic collection via a hierarchical management mechanism. Multiple traffic collection machines work simultaneously. Each collection machine in this framework runs multiple threads to collect mobile network traffic. We capture packet-level traces in the first five minutes and finally attain a traffic dataset with 808 MB benign traffic data and 330 MB malicious traffic data. We further investigate the traffic dataset by extracting traffic features and utilizing these features to train a machine learning classifier, which can be used to identify malicious network behaviors. This paper mainly has the following contributions:

- **Active and automatic Android traffic collection:** using this designed framework, we can collect Android traffic data actively and automatically, which can solve the difficulty in acquiring traffic data effectively.
- **High performance framework:** With multithreading parallel technology, DroidCollector can collect mobile traffic data from a bunch of apps quickly.
- **High quality Android traffic data:** Using this framework, we collect a high quality traffic dataset, which contains traffic mainly generated by the sample apps with little OS generated background traffic. We have shared and keep updating the dataset to stimulate researches in this area.

The paper is organized as follows. Section II highlights related work and motivation. Section III discusses the architecture of traffic collection framework. Section IV presents our evaluation. Section V discusses the application of traffic data to detect malware behaviors. Section VI analyzes the limitations of this methodology. Section VII concludes the paper.

II. Related work and motivation

Generally speaking, now traffic collection methods can be classified as three. The first method is passively obtain users' mobile network traffic from network service providers. For instance, Trestian et al. [4] applied rule mining spectral clustering to investigate the relationships of more than 280,000 users in a 3G mobile network by using the data set from their mobile carrier. Similarly, Maier et al. [5] focused on anonymized packet-level data of over 20,000 DSL customers. Obtaining mobile network traffic from network service providers is straightforward, but it is also dependent on the service providers' willingness to share the data, and may be

2324-9013/16 $31.00 © 2016 IEEE
DOI 10.1109/TrustCom.2016.267

confined by stringent regulatory rules. The second method is to manually collect mobile traffic. [6] designed SmartSiren to collect communication activity information from smartphones. Moreover, running an agent on a smartphone highly consumes resources on user terminals. This method is difficult to be implemented in a large scale system. Zhou et al. [7] manually analyzed malware samples in their collection. Falaki [8] studied traffic on the basis of two sets of traces from 43 users across different platforms. The first data set is collected using Netlog on mobile phones operating on Windows, and the second data set is from the tcpdump tool on mobile phones running on Android. Nevertheless, these studies have implemented manual traffic collection, which cannot obtain suffient traffic data sets. Furthermore, some researchers can capture network traffic proactively and automatically. Chen et al. [9] designed and implemented an Android malware traffic generation & collection scheme and captured network traffic traces in the first five minutes. However, the collected network traffic contains lots of irrelevant background traffic and its collection efficiency is not satisfactory.

Although considerable efforts have been made to collect network traffic, other issues, such as efficiency of traffic collection, quality of the collected network traffic data, have yet to be addressed. Compared with previous studies, this paper employs an approach that captures traffic using multiple collection machines and multithreading in each collection machine. This collection method ensures that the DroidCollector framework collects mobile app traffic with high efficiency. Therefore, we can obtain enormous and high quality traffic data through this framework in a short period of time.

III. Methodology

A. The Architecture Diagram of DroidCollector

The traffic collection framework is deployed in the UJN (University of Jinan) campus network. At the gateway of the campus, a firewall and NAT server are present to ensure the safety of the traffic collection framework. As shown in Figure 1, DroidCollector consists of the following three parts: control unit, data storage unit (including traffic storage server and app storage server), and traffic generation & collection unit. The control unit connects with the traffic generation & collection unit and the storage unit via LAN switch. The control unit is responsible for scheduling task. It assigns Android apps from the app storage server to a traffic collection machine in the traffic generation & collection unit. All collection machines in the generation & collection unit work together to complete the traffic collection task. Then, collected traffic data files are transferred to the traffic storage server.

B. The Control Unit

The control unit receives apps sample resouces and the task requirement from the system user. The control unit estimates the desired number of collection machines according to the number of apps and collection time. The control unit input apps to the APK queues in different collection machines for traffic collection. During the traffic collection period, the control unit monitors all collection machines statuses. When all of the collection machines complete the tasks assigned by the control unit, the final traffic data is returned to the user.

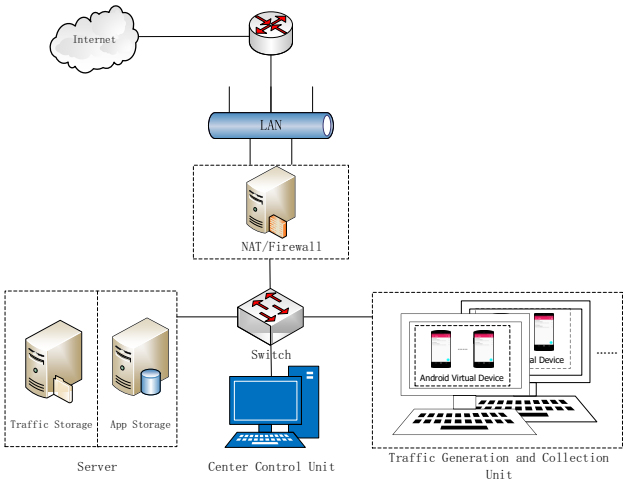

Fig. 1: The architecture diagram of DroidCollector

C. The Traffic Generation & Collection Unit

The traffic generation & collection unit includes multiple collection machines that work in parallel. Each collection machine consists of three parts, namely, pretreatment, regulation, and collection. The work flow of each collection machine is shown in Figure 2.

1) Pretreatment: The purpose of the APK pretreatment part is to preprocess these APK files to install these apps automatically. DroidCollector framework modifies the command of *aadp dump badging* to parse AndroidManifest.xml file. From the AndroidManifest.xml files, we can obtain some essential information, such as package name, MainActivity, and storage path which are required by an installer.

2) Regulation: This part controls and manages multithreading in the traffic collection program. Regulation consists of three components which work together. These components are the critical resource lock, thread controller, and APK queue. The critical resource lock controls critical resources to ensure the correctness of the corresponding programs. Critical resources include the rights to use adb command and to control the APK queue. In addition, the right to read and write thread information resources is also included. The thread controller is responsible for controlling thread starting, thread input, thread output and terminating one thread. When a thread starts, the thread controller needs to receive an APK message from the APK queue and relay this message as a set of parameters to the specific thread. The APK queue is used to store APK information whose network traffic data we wish to obtain.

3) Collection: The collection part consists of the specific functional modules. The function modules include the AVD (Android Virtual Device) controller, traffic generator, and exception handler. When a thread begins to execute, the collection machine uses the command of the adb and emulator to control the AVD's creation and launch. After starting AVD, the APK is installed on the emulator by the adb shell command. If the APK is installed successfully, then the AVD is shut down. The next step is to restart the emulator and use the

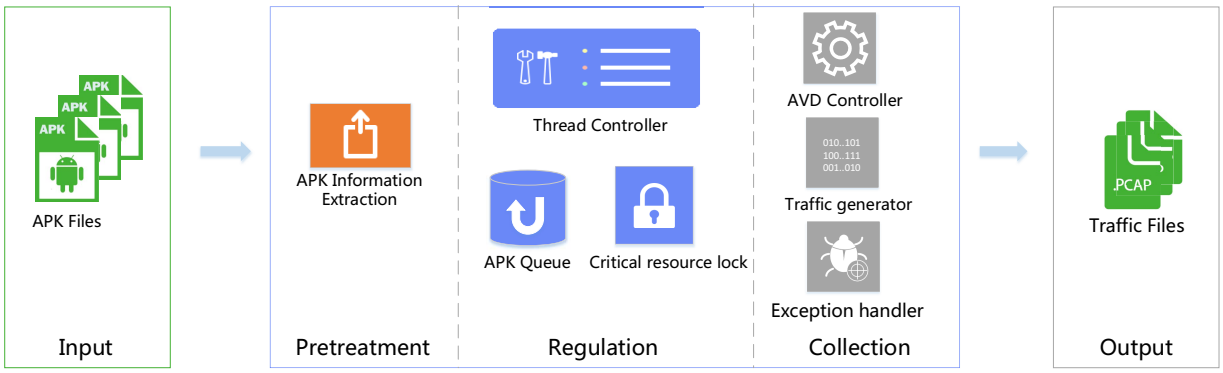

Fig. 2: The work flow of traffic generation & collection unit

tcpdump tool to collect traffic. We added the step of restarting the emulator because 83.3% of malware samples are activated by this restarting method, according to [1]. In addition, we find that background traffic will be included in the traffic collected by our method. Fortunately, the background traffic is controllable. The background traffic consists of those TCP packets that have specific source and destination IP addresses. Thus, we can filter out this part of the background traffic by applying a simple traffic filter. Android traffic generation and collection algorithm described above can be illustrated in Algorithm 1.

Data: Android APK files in app storage server and APK information in APK information queue
Result: Traffic files
while *The APK information queue is not null* **do**
  Read one piece of APK information in the queue;
  if *The adb is in use* **then**
    Wait 5 seconds;
    Continue;
  else
    Create and start AVD;
  end
  Get the PID and the name of AVD;
  Install the app;
  Restart AVD;
  Start Tcpdump tool;
  Start the App;
  if *Starting the app fails* **then**
    Log the error messages;
    Shut down the AVD;
  else
    Run AVD 300 seconds;
    Filter out background traffic;
    Read next piece of APK information in APK information queue;
  end
end

Algorithm 1: Android traffic generation and collection algorithm

IV. Evaluation

After presenting the traffic collection framework in detail, we evaluate its efficiency. All experiments were run on collection machines with an 8GB memory and Intel Core i7-4790 3.60GHz. We evaluate the framework from collection efficiency and traffic quality two aspects.

A. Collection Efficiency for Multithreading

As shown in Figure 3, we evaluate the time spent on Android traffic collection for multithreading on the settled number of apps. Given that collection machines need to perform some preparation work before collecting network traffic from apps, the time difference is not notable at the beginning of the collection process. Once the pretreatment is completed, the advantage of multithreading starts to show. According to Figure 3, the collection machine with three threads takes 2,087 minutes to collect the traffic of 800 apps. For the collection machines using six threads, nine threads, and twelve threads, the collection of traffic of 800 apps takes 1,260, 829, and 551 minutes, respectively. Clearly, using multithreading technique can drastically improve the collection efficiency.

B. The Quality of Collected Traffic

Through this traffic collection framework, we collect 330 MB traffic data generated by 5,560 malicious apps and 808 MB traffic data generated by 6,000 benign apps. It takes us about 140 hours (5.7 days) to complete the network traffic collection. In [9], Chen et al. proposed a traffic collection platform which we call Chen's platform. Here, we comparatively analyze the quality of two traffic data sets collected using their platform and the DroidCollector platform. The size of traffic data generated by 5,560 malicious apps during the first 5 minutes is 1,702 MB using Chen's platform, whereas the traffic data is only 330 MB with DroidCollector. The packet number is also clearly smaller using DroidCollector. The comparison charts of traffic size and packet number are shown in Figure 4, respectively. Apparently, our collected traffic is much less than Chen's collected traffic data in terms

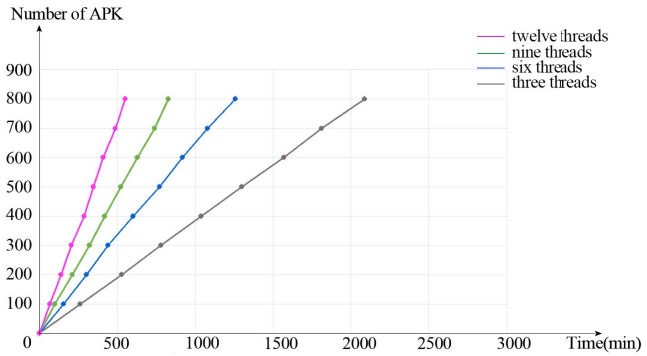

Fig. 3: Collection efficiency for multithreading on one machine

of data size or packet number because we have filtered out irrelevant traffic. We also comparatively analyze the protocol distribution of the two traffic data sets using the two methods.

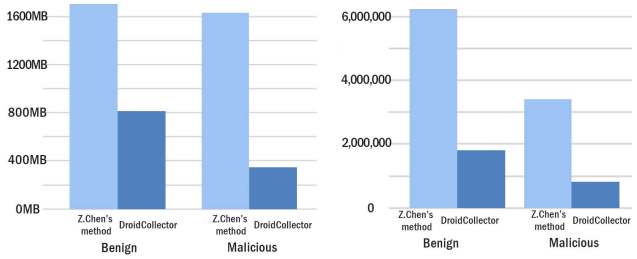

Fig. 4: Traffic data size and packet number using two different methods

Figure 5 shows the comparison pie charts of various protocols in benign traffic data, and Figure 6 shows the proportions of different protocols of traffic in malicious traffic data. Using Chen's method, the collected traffic contains mainly Windows OS and background traffic generated by Android emulator, whereas only a small portion of the traffic generated by the apps installed on the emulator. On the other hand, using DroidCollector to collect traffic, most of the traffic data is valuable HTTP traffic. Considering the superior performance of DroidCollector in terms of the traffic size, packet number, and traffic protocol distribution, we can conclude that the validity of collected traffic has been significantly improved.

Looking into details of the protocol distribution ratios of two Android traffic datasets, we can see that HTTP protocol packet number, DNS protocol packets number and HTTPS protocol packets number account for 99.98% Android traffic data collected by DroidCollector framework. Using Chen's method these three main protocols' ratio is only 9.38%, while the remaining protocol types of traffic data are LLMNR protocols, SSDP protocols, NBNS protocols. We consider these types(LLMNR, SSDP, NBNS) of traffic as *background traffic* because these types of background traffic will interfere with the analysis of apps' traffic data.

In addition, to further improve the traffic quality, we start the Android virtual machine in every collection machine 10 times continuously when no app is running. During this process, we capture the traffic of the virtual machine within 10 minutes. We start the collection of apps' traffic only when there is no traffic coming out of the virtual machine. This procedure will ensure that all the traffic data from the Android virtual machine are generated by the app.

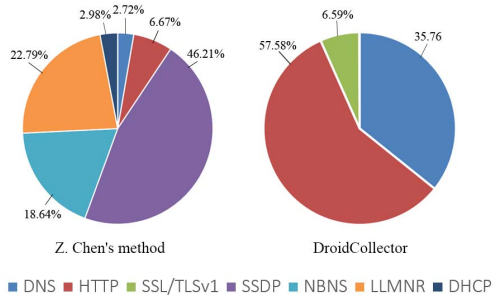

Fig. 5: Application layer protocols on benign App's traffic using two methods

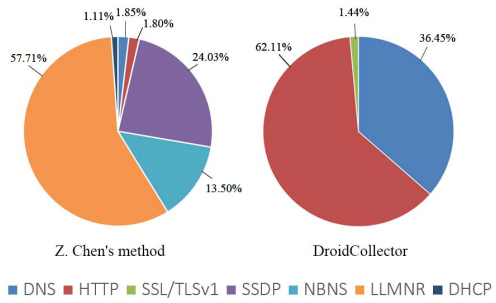

Fig. 6: Application layer protocols on malicious App's traffic using two methods

V. Malware Behavior Detection Using Collected Traffic

Based on the network traffic collection framework DroidCollector mentioned above, we have collected a significant amount of network traffic data including both benign traffic generated by benign apps and malicious traffic generated by malicious apps. We apply the machine learning algorithm: SVM (Support Vector Machines) [10] on traffic features to identify malicious network behaviors. The work flow is shown in Figure 7, which contains four major steps: network traffic collection, feature extraction, vector space embedding and learning-based detection.

A. Network Traffic Dataset

According to [1], more than 80% of malware utilize repackaging technique to embed malicious codes to normal apps. This type of malware not only performs malicious behaviors, but also runs normal functions. Therefore, the traffic dataset generated by malware is a mixture of benign and malicious traffic data. To improve the accuracy of the detection mechanism, we extract malicious traffic flows according to malicious destination IPs or domain names published by Virustotal [11].

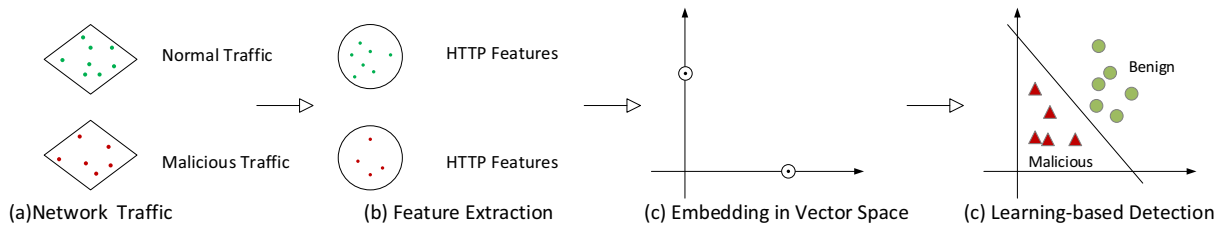

Fig. 7: The work flow of malicious behavior detection

In this paper, we only focus on the HTTP packets, since it contains lots of useful information for malware detection [12]. In Table I, we show the number of collected HTTP packets for each family of malware.

TABLE I: HTTP packet number of each family's malware

Id	Family	Number	Id	Family	Number
A	plankton	3846	F	Gappusin	247
B	DroidKungFu	657	G	Sendpy	85
C	FakeRun	391	H	ExploitLinuxLotoor	20
D	Opfake	317	I	Ginmaster	19
E	Glodream	268	J	Gamex	9

B. Network Traffic Feature Extraction

We extract traffic features from the HTTP header of benign traffic and malicious traffic, given that HTTP is the predominant protocol adopted by most mobile apps, and metadata information in HTTP request headers always contains valuable information [12]. These extracted features include the field of host, request-uri field, request method field, and user-agent field. A detailed explanation of these fields is presented in Table II. Extracted HTTP request fields from a HTTP request form a feature tuple which can represent this HTTP packet. Every feature tuple has a specific categorical label which corresponds to a benign app or malware family name.

TABLE II: Features extracted from HTTP request header

Id	Feature	Description
1	Host	This field specifics to the Internet host and port number of the requested resource.
2	Request-Uri	The URI is from the request source.
3	Request-Method	The method from HTTP indicates the action to be performed on the identified resource.
4	User-Agent	This field contains information about the user agent originating the request.

C. Embedding in Vector Space

The most common shortcoming of machine leaning methods is the black-box property of these methods [13]. We add this embedding step specifically to demonstrate the process of malware detection more clearly. We define four sets for each feature extracted from malicious HTTP requests. S_1 set saves the values of all host strings appearing in malicious HTTP requests; S_2 represents the keys in request-uri; the request-method set and user-agent strings are stored in S_3 and S_4, respectively. We define a feature vector where each feature is either 0 or 1. Every specific HTTP request feature tuple is mapped to a feature vector. The mapping relationship is defined as follows:

$$U_i = \begin{cases} 1 & \text{If the } ith \text{ feature exists in } S_i \\ 0 & \text{Otherwise} \end{cases}$$

We give an example to clarify the calculation process. A malware sample connects to a host which does not exist in S_1, however, the request-uri, request-method and user-agent of this HTTP packet appear in S_2, S_3, S_4,respectively. An example of the feature vector U is shown below:

$$U = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \end{bmatrix} \begin{matrix} Host & S_1 \\ Request-Uri & S_2 \\ Request-Method & S_3 \\ User-Agent & S_4 \end{matrix}$$

D. Learning-based Detection

We use machine learning techniques to automatically learn the separating hyperplane between benign traffic and malicious traffic. We comparatively analyze several machine learning algorithms, and finally, SVM classifier is selected because it achieves the best performance. By learning the features of the traffic data, it can divide the traffic data into two categories, namely, the benign traffic and malicious traffic. The detection function is given by:

$$F(x) =< W, U >$$

The training process of SVM involves the continuous adjustment of the weight vector W. We arrive at a final conclusion bas ed on the value of $F(x)$. That is, $F(x) > t$ (a given value) indicates malicious activity, whereas $F(x) < t$ corresponds to benign apps. The $F(x) = t$ represents the separating hyperplane.

E. Analysis of The Detection Results

To ensure the reliability of the detection model, 66% of feature vectors are applied to train detection model while 34% of the feature vectors are used to test the detection model. With the separating hyperplane, we can give an unknown HTTP packet a benign or malicious label. The detection rates of 10 different families' malware and benign apps are shown in Figure 8. From the histogram, we can see that 5 malware

families, such as Opfake Family (D), and Glodream Family (E), yield detection rates of 100%. Moreover, the False positive rate (FPR) is very low, and the maximum false positive rate (FPR) is only 1.15%.

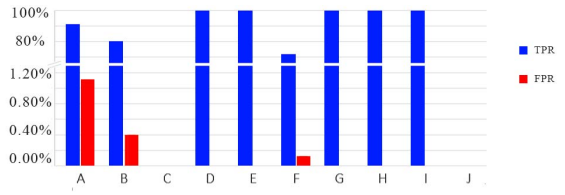

Fig. 8: The TPR and FPR on malicious traffic in each family

However, we should also note an unexpected phenomenon: the detection model cannot classify malware from FakeRun Family (C) and Gamex Family (J) to the correct category. To find the cause of this problem, we carefully analyze these two families' malware samples and their classification details. We find that extracted features from the FakeRun Family are similar with that of the Plankton family, the detection model falsely regards these samples as Plankton Family. As for the Gamex Family, the total sample number is only 9, which is too small for SVM to train an effective detection model. On average, we achieve about 98% confidence that these unknown HTTP request packets are classified by the detection model into their corresponding categories correctly.

VI. Limitations

Although this platform is able to launch automated network traffic collection, and the traffic collected has high quality, there are still some limitations in certain scenarios. Here, we will discuss two major weaknesses of DroidCollector that could be improved.

A. Incomplete Malicious Traffic Data

According to [1], *BOOT COMPLETED* is the most commonly used malware activation method. In this paper, we use the activation method based on emulator restarting. However, other types of activation methods such as receiving a call, accessing a website, are not implemented. Therefore, the collection of malicious traffic data may not be complete and comprehensive. To ease this problem, we will add activation methods and collect more malicious traffic in the future.

B. Advanced Malware with The Ability of Hiding Malicious Behaviors

To evade advanced malware detection methods, advanced malware executes malicious behaviors in subtle ways. Malware may attempt to detect the current operating environments before it runs its malicious code [14]. Our method can also enable the collection of traffic data on real smartphones, which is one of our future directions.

VII. Conclusion

Android malware has emerged as a new and rapidly growing threat to the mobile ecosystem. Malware can be identified by analyzing mobile network traffic traces. However, research initiatives have been constrained by the lack of a comprehensive mobile malware traffic dataset. To address this issue, we introduce DroidCollector, which is an effective Android traffic generation and collection framework to collect traffic generated by Android apps while removing irrelevant background traffic. The multiple collection machines enabled by the multithreading mechanism significantly improve the collection efficiency. To stimulate research in the area of malware detection, we publish the traffic data collected from our framework to other researchers under $http://sec.ujn.edu.cn/DroidCollector$ and we will keep updating the dataset. We also apply SVM machine learning algorithm on the traffic features extracted from the traffic data to identify malicious network traffic. The final results show that we can achieve 98% confidence, which indicates that most of malicious HTTP traffic is correctly classified.

Acknowledgement

This work was supported by the National Natural Science Foundation of China under Grants No.60903176 and No.61472164, the Natural Science Foundation of Shandong Province under Grants No.ZR2014JL042 and No.ZR2012FM010.

References

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

[2] B. P. Sarma, N. Li, C. Gates, R. Potharaju, C. Nita-Rotaru, and I. Molloy, "Android permissions: a perspective combining risks and benefits," in *Proceedings of the 17th ACM symposium on Access Control Models and Technologies*. ACM, 2012, pp. 13–22.

[3] S. Y. Yerima, S. Sezer, and G. McWilliams, "Analysis of bayesian classification-based approaches for android malware detection," *Information Security, IET*, vol. 8, no. 1, pp. 25–36, 2014.

[4] I. Trestian, S. Ranjan, A. Kuzmanovic, and A. Nucci, "Measuring serendipity: connecting people, locations and interests in a mobile 3g network," *Acm Sigcomm*, pp. 267–279, 2009.

[5] G. Maier, F. Schneider, and A. Feldmann, *A First Look at Mobile Hand-Held Device Traffic*. Springer Berlin Heidelberg, 2010.

[6] J. Cheng, S. H. Y. Wong, H. Yang, and S. Lu, "Smartsiren: virus detection and alert for smartphones." in *Proceedings of the 5th International Conference on Mobile Systems, Applications, and Services (MobiSys 2007), San Juan, Puerto Rico, June 11-13, 2007*, 2007, pp. 258–271.

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

[8] H. Falaki, D. Lymberopoulos, R. Mahajan, S. Kandula, and D. Estrin, "A first look at traffic on smartphones," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. ACM, 2010, pp. 281–287.

[9] Z. Chen, H. Han, Q. Yan, B. Yang, L. Peng, L. Zhang, and J. Li, "A first look at android malware traffic in first few minutes," in *IEEE Trustcom/bigdatase/ispa*, 2015.

[10] R. E. Fan, P. H. Chen, and C. J. Lin, "Working set selection using second order information for training svm," *Journal of Machine Learning Research*, vol. 6, no. 4, pp. 1889–1918, 2005.

[11] "virustotal." [Online]. Available: https://www.virustotal.com/

[12] 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 (INFOCOM), 2015 IEEE Conference on*. IEEE, 2015, pp. 1481–1489.

[13] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *Security and Privacy (SP), 2010 IEEE Symposium on*. IEEE, 2010, pp. 305–316.

[14] "Android malware detecting emulator." [Online]. Available: http://c0defreak.blogspot.hk/2014/02/android-malware-detecting-emulator.html