Toward Developing a Systematic Approach to Generate Benchmark Android Malware Datasets and Classification

Arash Habibi Lashkari, Andi Fitriah A.Kadir, Laya Taheri, and Ali A. Ghorbani
Canadian Institute for Cybersecurity (CIC)
University of New Brunswick Fredericton, NB, Canada
(a.habibi.l; andi.fitriah; ltaheri; ghorbani)@unb.ca

***Abstract*—Malware detection is one of the most important factors in the security of smartphones. Academic researchers have extensively studied Android malware detection problems. Machine learning methods proposed in previous work typically reported high detection performance and fast prediction times on fixed and defective datasets. Therefore, based on these shortcomings most datasets are not suitable for real-world deployment. The main goal of this paper is to propose a systematic approach to generate Android malware datasets using real smartphones instead of emulators and develop a new dataset, namely CICAndMal2017, which covers all the shortcomings and limitations of previous datasets. Also, we offer 80 traffic features to select the best feature sets for detecting and classifying the malicious families just by traffic analysis. The proposed method showed an average precision of 85% and recall of 88% for three classifiers, namely Random Forest(RF), K-Nearest Neighbor (KNN), and Decision Tree (DT).**

***Index Terms*—Android, Malware, Dataset, Network Traffic**

I. INTRODUCTION

In recent years, smartphones have become the main medium of communication. The number of Smartphone users is now greater than ever because of its available features. Currently, different operating systems such as iOS, Windows, Blackberry OS, and Android are available for smartphones. Among them Android is the most proper platform. Worldwide mobile app downloads in 2017 were 197 billion and are expected to reach 352.9 billion in 2021 [1]. Also, according to Ericsson mobility report [2], by 2020, the subscription of smartphones will be more than 6 billion and 80% percent of network traffic will be generated by mobile networks. Meanwhile, more than 96% of global mobile traffic belongs to Android and Apple, of which 50% is generated by Android smartphones [3].

Most of the existing malware detection techniques can be categorized into static and dynamic groups [4]. Static detection is the process of finding malicious characteristics or bad code segments in an app without running the app, whereas dynamic detection is done monitoring the real behaviour of app by running it in a sandbox as an isolated environment. Isolated environment is difficult and not possible for all researchers, using the available datasets is one of the best solutions for dynamic analysis. Currently there are numerous datasets available for Android malware, but all of them suffer from shortcomings and limitations.

Our contributions in this paper are twofold. Firstly, we analyze the existing datasets to find their main shortcoming and limitations. Then, we present our approach for generating a new Android malware dataset called CICAndMal2017, which covers all the shortcomings and limitations of previous datasets. The dataset is completely labelled and includes network traffic, logs, API/SYS calls, phone statistics, and memory dumps of 42 malware families. Droidkin, an apps similarity detector [5], shows the selected families are not similar and not closely related in our dataset, called CICAndMal2017. Secondly, the paper extracts and calculates more than 80 network traffic features for all benign and malware Biflows by using CICFlowMeter software which is publicly available on Canadian Institute for Cybersecurity website [6]. Finally, we analyze the generated dataset with three defined scenarios: malware binary detection, malware category classification, and malware family characterization. We build, train, and evaluate our model via three common machine learning algorithms, namely RF, KNN, and DT.

The remind part of this paper organized as follow: section II, Available Datasets which compares the currently available datastes, section III namely Experiments explains the configuration architecture of the dataset, section IV, Analysis which analyze the proposed dataset based on network traffic and finally section V is the Conclusion.

II. AVAILABLE DATASETS

In this section we explained the need for a comprehensive and reliable dataset to test and validate android malware detection system. We evaluate several publicly available Android malware datasets spanning from 2012 to 2017.

The Genome project is one of the first publicly available attempts in 2012 [7]. In this project, the authors collected 1260 malware samples from 2010 to 2011 of Android malware vendors. They use static analysis methods to define malware behavior by evaluating the installation, activation and payload of samples. These methods only focus on static test and statically scanned components of the malicious source code, tracked API calls and studied permission lists. They also used their proposed dataset on real smart phones to examine the effectiveness of the existing anti-virus software. After that, the Drebin dataset (2014) [8], was created which provided

978-1-5386-7931-9/18/$31.00 ©2018 IEEE

5560 malware samples across 20 families and 123,453 benign samples from 2010 to 2012. They trained their classification system based on static features include hardware components, requested permissions, App components, filtered intents (Extracted from manifest files and suspicious/restricted API calls and used permissions), and network addresses (Extracted from disassembled code), to evaluate their dataset.

The Hacking and Countermeasure Research Lab (HCRL) [9], produced the following four Android malware datasets: AndroTracker, SAPIMMDS, Andro-Dumpsys, Andro-Profiler. In the AndroTracker dataset [10] (2015), each malware family has been written by the same creator and each creator has used his/her own certification (serial number of certificate) to develop malware apps. They used a similarity scoring associated with creator information and other static features such as intent, critical permissions and suspicious API calls to classify malware samples.

Creator based system score

Dataset SAPIMMDS [11] from the Korea Internet Security Agency (KISA), consists of 13 malware families with 906 samples and 1776 benign samples. this dataset was generated from March to December in 2014. They tracked suspicious sequences of API calls from bytecode with memory dump techniques in order to extract API call patterns for certain malicious functionality through their dataset.

Api calls & memory dumping techniques

The contribution in gathering Andro-Dumpsys dataset[12] (2016), combines malware-centric attributes with intent-based features for malware detection and classification. It includes 906 malware samples and 1776 benign samples. The authors adopted the serial number of a certificates, suspicious API call sequences, permission distributions, intents, and the usage of system commands for executing forged files as feature vectors. Based on these feature vectors, they conducted profiling patterns according to the relationships between opcode and bytecode (captured by memory dumping techniques) for each malware families. In the detection and classification engines on a server side, they calculated a similarity score among each apk request from a client side and their profiling patterns.

Similarity score

In the Andro-Profiler dataset[13] (2016), the authors used the concept of behavior profiling with extracting system calls and system logs during malicious application executions on an emulator. They developed a hybrid anti-malware system with client and server sections.

The Kharon dataset [14] (2016), was generated by running 7 malware samples on a real mobile device and made use of AndroBlare tool[1], to track information flow between system objects such as files, processes or sockets at system level and provide a human-readable directed graph for each malware sample. The authors documented the analysis behavior of each graph in great detail.

Dynamic

In 2017 Lashkari *et al.* [15], provided the AAGM dataset with 1500 benign samples plus 400 malware samples from 2008 to 2016 in 12 families of 3 categories (benign, general and adware). They installed these samples on real smartphones and started performed user-interaction scenarios to capture network traffic. Their analysis method was based on machine learning algorithms. Also, they used the Droidkin project [5], for finding relationships between various apps and clustering them based on source code component's similarity. They released their validation dataset with known relationships between apps.

Dynamic

Static

[1]http://www.blare-ids.org/

Besides this dataset, the AMD dataset[16] is another publicly available dataset since 2017 that includes profile information of Android malware. They gathered 405 android malware samples of 4 categories within 71 families. Their systematic methodology lies in evaluating malware behavior with analysis of malicious components based on their priorities.

The aforementioned datasets remedied researchers' approaches in the Android malware detection domain, however they have still struggled with serious drawbacks of their contents. A comprehensive Android malware dataset must contain diverse categories and families alongside a large number of malware samples, which the stated datasets lacked. In general, malware will not express their actual malicious behavior, unless they are installed on real devices. Even though some of the datasets captured dynamic features of their samples, they only executed the malware on emulators and Android Virtual Device (AVD) and avoided to running the apps on real smartphones.

1.

2.

3.

According to Malton [17], running samples on real devices enables us to circumvent many anti-emulator techniques. Most of the malware applications are sensitive to a trigger point of malicious intended activation and need user-interaction scenarios to be tricked, e.g. BOOT-COMPLETED. In addition, an inclusive dataset may gather as many features as it can, both static and dynamic features, for more desirable usage.

However, none of the mentioned datasets have captured all variety of malware behavioral features, including continuous features (API calls, system calls, logs, network traffic) and discrete features (memory dump, permissions, memory usage, battery usage and network usage). In order to be able to trust research results, its used dataset must balance the number of malware samples and the number of benign samples in its evaluation. Otherwise, reaching a high accuracy may not be a valid proof of their work. As we mentioned in our previous paper[15], the normal distribution of benign and malware apps in the real world is 80% to 20% according to Symantecs Internet Security Threat Report (SISTR) [18]. To put it briefly, all of the above datasets lack real-phone installation and user-interaction scenarios, contain insufficiently diverse categories & families and failed to balance the number of malware samples vs. benign samples.

III. EXPERIMENTS

In this section, we first describe the benign and malware families of the CICAndMal2017 dataset and then explain the configuration architecture of the experiment, followed by the activation scenarios, data installation and capturing, and feature extraction and selection.

A. Benign and malware apps

We collected more than 10,854 samples (4,354 malware and 6,500 benign) from several sources such as VirusTotal service [19], Contagio security blog [20], previous researchers [21], [15], [5]. However, we have only managed to install 5 thousands of them (malwware 429 and benign 5,065) mainly due to the following two reasons:

1) *Sample error*. Most of the samples, then we have collected are of bad quality, i.e. Dex error, unsigned and corrupted apps. For instance, the unsigned apps in the CICAndMal2017 dataset have reduced the number of samples to be installed. Significantly, all applications must be signed in order to be installed on an emulator or a real device.
2) *Inconsistent malware labelling*. The naming convention of malware labelling (i.e. Family name) is inconsistent among both academic and industry fields. The inconsistent labelling between different anti-virus vendors and researchers leads to confusion and is time-consuming for reorganization. In this case, we have to compare the malware labelling from several anti-virus vendors and follow the majority numbers of the most frequent label of the specific malware family.

We have collected over six thousand of benign apps from Googleplay market published in 2015, 2016, 2017. These apps were collected based on their popularity (i.e., top free popular and top free new) for each category available on the market. Out of 6,500 apps, 1,435 of them were removed (leaving 5,065 apps in total) as they were flagged as suspicious or adware, by more than two antivirus (AV) products in Virustotal web service [19] [2].

Table I shows the yearly breakdown of the collected dataset (42 malware families), which is based on the following four categories, i.e., adware, ransomware, scareware, SMS malware.

B. *Configuration Architecture*

Figure 1 shows the network architecture for the experiment set up and configuration. In this experiment, we used three laptops which are connected to three smart-phones respectively:

1) Laptop-1 with smart-phone-1: a 64-bit Ubuntu 16.04 machine with Intel CPU and 8GB RAM was connected by a NEXUS 5 smart-phone
2) Laptop-2 with smart-phone-2: a 64-bit Ubuntu 16.04 machine with Intel CPU and 8GB RAM was connected by a NEXUS 4 smart-phone
3) Laptop-3 with smart-phone-3:a 64-bit Ubuntu 16.04 machine with Intel CPU and 8GB RAM was connected by a NEXUS 5 smart-phone

We defined three states of data capturing in order to overcome the stealthiness of an advanced malware. Most of the advanced malware employed the evasion or transformation technique to dodge detection (i.e code permutation, register renaming, idle activation). Thus, in this stage we consider some behaviour of malware that only triggered over time after the restart process. In a preliminary analysis, we have tested small samples and discovered that the following states are able to activate and trigger the malware behaviour at run-time during the execution in dynamic analysis:

1) Installation: The first state of data capturing which occurs immediately after installing malware (1-3 min).

[2]The scanning process dated June 2017

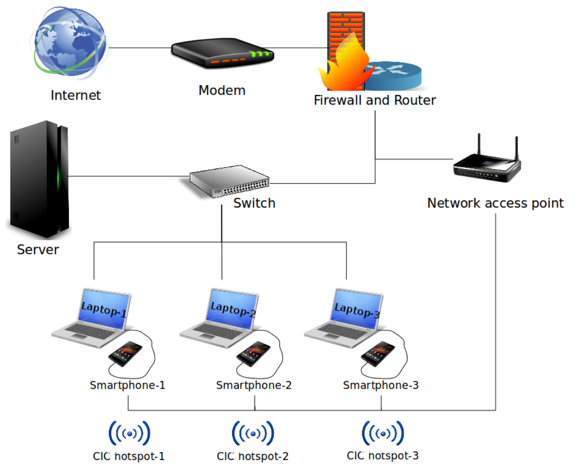

Figure 1: The network architecture

2) Before restart: The second state of data capturing which occurs 15 min before rebooting phones.
3) After restart: The last state of data capturing which occurs 15 min after rebooting phones.

C. *Malware Activation Scenario*

In order to create a comprehensive scenario for data capturing, we first created a taxonomy to understand the malware behavior and its characteristics. We classified the taxonomy into 20 types of attacks (A1-A20) and 4 types of C&C communications (C1-C4) as shown in the Table I. In this phase, we gathered all reports from multiple antivirus such as Kaspersky, Fortinet, Dr.Web, and to name a few, which is based on the family label listed in the table. After that, we created the user profile for each smartphones and defined the user interaction with different tasks automatically.

1) User Profile: We created three user profiles for each smart-phones. Each user is registered to Gmail, Facebook, Skype, and Whatsapp with a valid email address and phone number as listed in Table III.

2) User Interaction: In order to have a reliable data output, we created a specific scenario for each of the malware category. Table II shows the list of the scenario for each malware category. The following explains the detail description of the scenarios:

Sending SMS: To automate the sending of messages on the Nexus 4, we were able to use the messaging app that came installed on the phone with following commands.

```
adb shell am start −a android.intent.
action.SENDTO −d sms:\<phone_number\>
−−es sms_body ”\<message body\>”
−−ez exit_on_sent true
adb shell input keyevent 22
adb shell input keyevent 66}
```

However, with the Nexus 5 this only type out a message and starts a new line in the same message. So we had to use Android Studios UI Automator to create a test apk for the messaging app that we were then able to call whenever we wanted to send a message. We only put a SIM card in the SMS

Table I: Description of malware dataset (Available at http://www.unb.ca/cic/datasets/CICAndMAl2017.html)

	Family	Year	AV Labelled	Total Collected	Total Captured	Attack																				Communication			
						A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	A16	A17	A18	A19	A20	C1	C2	C3	C4
	Dowgin	2016	Gdata	50	10		✓																						
	Ewind	2017	Koodous	50	10	✓		✓		✓		✓														✓	✓		
A	Feiwo	2016	Fortinet	100	15	✓																							
D	Gooligan	2016	Fortinet	43	14	✓				✓	✓	✓														✓		✓	
W	Kemoge	2015	Lookout	35	11							✓																	
A	koodous	2017	Koodous	50	10																								
R	Mobidash	2015	Enet32	32	10				✓																	✓	✓	✓	
E	Selfmite	2014	AntiVirus	6	4		✓	✓				✓																	
	Shuanet	2015	Lookout	24	10					✓	✓																		
	Youmi	2015	Gdata	50	10	✓						✓																	
R	Charger	2017	Sophos	150	10	✓						✓			✓	✓		✓											
A	Jisut	2017	ESET	150	10										✓			✓			✓		✓	✓	✓	✓			
N	Koler	2015	Avast	135	10		✓	✓								✓	✓	✓											
S	LockerPin	2015	ESET	150	10					✓					✓			✓	✓		✓								
O	Simplocker	2015	Symantec	146	10	✓						✓						✓	✓							✓	✓	✓	
M	Pletor	2014	Alibaba	112	10											✓		✓	✓	✓		✓				✓			✓
W	PornDroid	2016	Ikarus	113	10		✓											✓		✓						✓			
A	RansomBO	2017	Fsecure	100	10																								
R	Svpeng	2014	Sophos	120	11	✓												✓	✓							✓			
E	WannaLocker	2017	Avast	18	10											✓			✓	✓									
	AndroidDefender	2013	Fortinet	150	17						✓	✓	✓																
S	AndroidSpy.277	2016	DrWeb	150	6	✓						✓														✓			
C	AV for Android	2015	Mixed AVs	83	10								✓																
A	AVpass	2013	Kaspersky	150	10		✓					✓			✓														
R	FakeApp	2015	Kaspersky	150	10								✓	✓		✓													
E	FakeApp.AL	2015	Mixed AVs	150	11			✓					✓	✓		✓													
W	FakeAV	2013	Kaspersky	150	10	✓				✓					✓														
A	FakeJobOffer	2013	Fsecure	9	9	✓																							
R	FakeTaoBao	2013	Mixed AVs	150	9	✓						✓																	
E	Penetho	2012	Kaspersky	150	10	✓	✓			✓																✓			
	VirusShield	2014	Mixed AVs	150	10								✓																
S	BeanBot	2011	Sophos	85	10	✓		✓					✓																
M	Biige	2012	Alibaba	147	11		✓	✓		✓		✓																	
S	FakeInst	2013	BkAv	150	11			✓																					
	FakeMart	2012	Alibaba	63	10			✓	✓				✓																
M	FakeNotify	2012	Fsecure	150	10			✓				✓		✓															
A	Jifake	2012	Fsecure	149	10		✓	✓				✓	✓																
L	Mazarbot	2016	ESET	22	10	✓		✓				✓	✓	✓		✓													
W	Nandrobox	2012	NANO	150	11		✓	✓																					
A	Plankton	2011	Ikarus	150	10		✓					✓																	
R	SMSsniffer	2011	Ikarus	53	9	✓		✓																					
E	Zsone	2011	Avira	150	10			✓					✓																
	Total			**4 354**	**429**	**14**	**11**	**14**	**2**	**8**	**3**	**15**	**8**	**4**	**6**	**7**	**1**	**9**	**5**	**3**	**2**	**1**	**1**	**1**	**1**	**10**	**3**	**3**	**1**

Legend:
A1: Steal information (contacts and sms messages, credit card credentials)
A2: Harvest information (text messages, bookmark history and mobile accounts)
A3: Send SPAM SMS / Phishing to the contact list
A4: Make silent calls in background
A5: Exploit root (Privilege escalation)
A6: Inject malicious code
A7: Download malicous software
A8: Premium subscription
A9: Direct user to malicious web site
A10: Prevalence of the malware (uninstall AVs, encrypt source, load dinamic code)
A11: Geographic location attack
A12: Send shortened URLs

Legend:
A13: Lock the device
A14: Encrypt the user files
A15: Modify contents,(SD card)
A16: Change the PIN number
A17: Use the Video Camera
A18: Voice,message to unlock the screen
A19: Fake login screen
A20: Change wallpaper
C1: C&C servers
C2: Start after time delay
C3: Update with new features
C4: Using TOR

malware category to activate the SMS premium-rated malware. Thus, these messages were not actually being sent except for SMS malware. We also sent messages using WhatsApp which, unlike the previous messages, these ones were successfully sent. Because of that, we only sent messages between our test phones. In order to send these messages, we used Android Studios UI Automator again to create a second test.

Sms & WhatsApp msg automation

Sending Calls: To automate phone call events we used the adb command:

```
adb shell am start −a android.intent.
action.CALL %−d tel:<phone_number>.
```

However our phones did not have SIM cards so no calls were actually made. When we tested with SIM cards we modified that line to be:

```
adb shell am start −a android.intent.
action.CALL −d tel:
```

GPS spoofing: We modified an open source Android application that can fake the location of the phone[22]. we made a small modification so we could run it on API 23. Then, we took a backup so we do not need to install it every time we restore the phones.

Reboot phone: Running a factory reset after each test was not a viable solution because it requires a number of manual steps to set up the phone again. Rather than a factory reset, we decided to use a custom recovery tool (TWRP) to create a full backup that we can restore it between the tests. With our Nexus phones running Android 5.1.1 (Lollipop), we were able to create a backup, and then using only ADB we were able to successfully restore the backup and reboot the phone. However, on our Samsung Galaxy A5 (Model number: SM-A510FD) running Android 6.0.1 (Marshmallow), we were unable to access the device with ADB while it was in recovery mode and because of this we were unable to automate restoring the backup.

On our Google Pixel phone, we used TWRP 3.1.0-RC2 and whenever we tried to restore a backup the phone was wiped as if we had performed a factory reset. The team behind TWRP knew about this issue in a previous version of Google Pixel but it was supposedly fixed in 3.1.0-RC1 and 3.1.0-RC2. As an alternative to the TWRP backup we tried using ADB's built-in backup tool. However, we found many reports of people having issues with this tool and after trying it ourselves, we determined the backup did not cover as much as we needed it to.

Internet browsing (loading urls): We are able to use the browser to navigate to a given URL using the command below. If this command is run and you have multiple browsers installed, but do not have one selected as the default, it will bring up a prompt to select one of the browsers.

```
adb shell am start −a android.intent.
action.VIEW −d url
```

Table II: User interaction scenario

Category	Scenario	Rem.
Benign	Send Message, Make Call, Enable GPS, Browse Internet	SIM card disable
Adware	Send Message, Make Call, Enable GPS, Browse Internet	SIM card disable
Scareware	Send Message, Make Call, Enable GPS, Browse Internet, Click/follow popup	SIM card disable
Ransom	Send Message, Make Call, Enable GPS, Browse Internet, Click/follow popup, Set the four-digit PIN and lock the phone, Click/interact with any popup message, Save more than 10 contacts in the contact list, Save the following documents in both internal and external storage: jpeg, jpg, png, bmp, gif, pdf, doc, docx, txt, avi, mkv, 3gp, mp4 (size more than 10 KB)	SIM card disable
SMS malware	Send Message and SMS, Make Call Enable GPS, Browse Internet, Install AV(AVG, Avast, BitDefender), Save more than 10 contacts in the contact list	SIM card enable

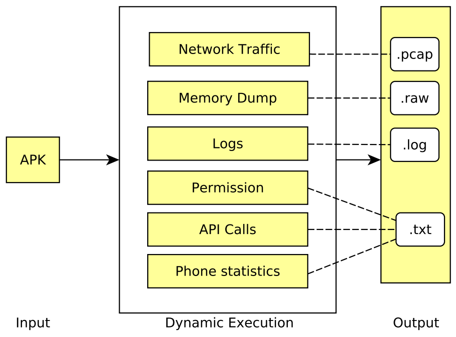

Figure 2: Captured and monitored data sources

D. *Feature Extraction and Selection*

We have two groups of features namely continuous and discrete. The continuous features such as API/system calls, logs and network traffic are extracted at all three states that we have defined in Section 4.1 (installation, before restart, and after restart) while discrete features such as memory dumps, permissions, battery usage, and network usage are extracted in the last state of data capturing which occurs 15 min after rebooting phones. Figure 2 shows the list of all captured and monitored data sources in this dataset.

1) Continuous Features.

a) API calls:: We initially tried to use Frida to trace the api calls of a process[23]. By pushing the executable for frida-server onto the device and executing it as the superuser, we can use frida-trace on the computer to capture the API calls. However, it doesn't seem possible to use both this and strace on the same process, we believe this is because they both use ptrace. Also in order to monitor any function with frida-trace you have to pass the name of the function you want to trace as an argument. Because of this we decided to use Appmon[24], as an alternative to frida-trace.

We were able to get the API information by using appmon.py after starting frida-server on the device and saving the output to a file. However some malware samples kill their processes immediately, therefore we were not able to trace the API calls because there is no process to trace. It is also important to note that Appmon takes a single process name as its argument, so it will only capture the API calls from the malware process.

b) System calls:: We found a version of strace made for Android; we can push this executable to the device and can use it to trace the system calls of a given process. Because we cannot use this simultaneously with our method of capturing the API calls, we decided to capture the API calls and not the system calls. That version worked on the google Pixel and Samsung phones we had originally. We would have had to make our own executable to be able to use strace on the Nexus phones.

c) Logs:: ADB has a built in function, logcat, that we can use to get the logs of the entire device.

d) Network traffic:: By creating a hotspot with the computer and connecting to that hotspot with the phone we are able to use tcpdump on the computer to capture all of the network traffic from the phone.

2) Discrete Features.

e) Memory dump:: LiME[25], requires the device to allow Loadable Kernel Modules (LKM). Because of this we had to compile a kernel for our Nexus 5 that supports LKMs, we were successful following the guide in the appendix of this paper[26].

There are a number of other memory dumping tools that we found, such as Silensec's AProcDump [3], and memfetch [4]. However these tools only dump the memory related to a single process. Also many of these tools need to be remade as Position Independent Executables before they can be run on newer devices. We were able to do this by modifying the Android.mk file to include the following lines:

LOCAL_CFLAGS += -fPIE
LOCAL_LDFLAGS += -fPIE -pie
before the line:
include $(BUILD_EXECUTABLE)

f) Permissions:: Initially we managed to get the permission information from the .apk file using aapt d permission ¡file.apk¿ We also managed to get the permission information using dumpsys package which gives the permissions requested, the permissions granted and some other useful information like the uid.

g) Memory usage:: To get the ram usage we used the ADB command dumpsys meminfo -a We also used dumpsys meminfo -a $package to get only the infomation related to the malware sample

h) Battery usage:: To get the battery usage we used the ADB command dumpsys batterystats

i) Network Usage:: To get the network usage we used the ADB command dumpsys netstats detail This lists usage

[3]http://www.silensec.com/downloads-menu/software/aprocdump
[4]https://github.com/citypw/lcamtuf-memfetch

by uid. The uid of an app is obtained as part of dumpsys package

Table III: User profile description

User	Account	Email/Phone No.
1	Gmail	cic.googl.01@gmail.com
	Facebook	cic.googl.01@gmail.com
	Skype	cic.googl.01@gmail.com
	Whatsapp	+6*******0062
2	Gmail	cic.googl.02@gmail.com
	Facebook	cic.googl.02@gmail.com
	Skype	cic.googl.02@gmail.com
	Whatsapp	+6*******6315
3	Gmail	cic.googl.03@gmail.com
	Facebook	cic.googl.03@gmail.com
	Skype	cic.googl.03@gmail.com
	Whatsapp	+6*******2140

IV. ANALYSIS

In this paper, we only present the analysis of one of the six feature categories (Figure 2), which is network traffic. Previous works have shown that the network traffic can be used to detect and classify the Android malware accurately. However, most of the papers have used an old dataset in their analysis, such as the 2012 Genome datasets [27], [4], [28] and 2014 Drebin datasets [29]. As the behavior of malware becomes sophisticated over time, we developed all features presented by previous researchers and tested our framework on the pervasive and comprehensive dataset that we have generated (as mentioned in Section 3). We evaluated the proposed dataset in three scenarios: (1) Scenario A: malware binary detection, which is to distinguish the malware apps from benign apps, (2) Scenario B: malware category classification, which is to classify the malware apps category and (3) Scenario C: malware families characterization, which is to label malware families.

Testing. Scenario

A. *Feature Extraction and Selection*

We began to extract the 80 network-flow features from the dataset using CICFlowMeter[30]. A flow is defined by a sequence of packets with the same values for five attributes, namely *Source IP*, *Destination IP*, *Source Port*, *Destination Port* and *Protocol*.

After running the apps on the real Android smart phones and capturing the generated traffic at the gateway level, all traffics have been labelled in five classes (adware, scareware, ransomware, SMS malware, and benign). Then we conducted the feature extraction and selection to find the best set of features, before the training and testing process.

As we need two separate datasets for training/testing and evaluation, we divided our samples of CICAndMal2017 dataset accordingly, 60% for training/testing and 40% for evaluation. Therefore, we used the pre-process function available in Weka [31] (Resample with noReplacement)to divide the different datasets proportionally within the testing and evaluation datasets. After that, we implemented the 10-fold cross validation in our experiment.

For feature selection, we conducted two feature selection algorithms, namely *CfsSubsetEval with the Best First search method* and *Infogain with the Ranker search method* in Weka over the training dataset [31]. We then used the following common features as the result of features selection process.

1) *Fwd_Packet_Length_Std*: standard deviation of the size of packet in forward direction
2) *Init_Win_bytes_forward*: the total number of bytes sent in the initial window in the forward direction
3) *Init_Win_bytes_backward*: the total number of bytes sent in the initial window in the backward direction
4) *min_seg_size_forward*: minimum segment size observed in the forward direction
5) *Bwd_Packet_Length_Min*: minimum of the size of packet in backward direction
6) *Bwd_Packet_Length_Std*: standard deviation of the size of packet in backward direction
7) *Total_Length_BwdPackets*: the total size of the packet in backward direction
8) *Flow_IAT_Max*: maximum inter-arrival time of the packet
9) *Fwd_IAT_Min*: minimum time between two packets sent in the forward direction

Two common metrics, Precision (Pr) and Recall (Rc) have been selected to evaluate the quality of the classification process.

B. *Training, Testing, and Evaluation*

We built and trained our model with 10-fold cross validation on the 60% of training data, and evaluated our model on the 40% of testing data via three common classifiers namely Random Forest (RF), K-Nearest Neighbors (KNN), and Decision Tree J48 (DT) accordingly based on the three defined scenarios:

Better models in other papers

- Scenario A: the results indicate that all three algorithms returned similar results (Table 4 Scenario A). The average of the probability of success (precision) is about 85% with 88% recall for both the training and evaluation set.
- Scenario B: Table 4 (Scenario B) presents the results of the malware category classification. The average precision and recall for all testing algorithms is less than 50%. The same procedure was used in the evaluation test.
- Scenario C: this scenario also exhibits low detection results as Scenario B with the average precision and recall of 20% for both training and evaluation set (Table 4 Scenario C).

The analysis results (Scenario A) indicate that network traffic feature is acceptable for real world malware detection as the dataset used in this analysis is comparable to real-world network traffic. However, the Scenario B and C show that network traffic feature is not sufficient to characterize the type of malware (*i.e. Adware, ransomware, SMS malware, scareware*) and the variety of malware family (*i.e. Shuanet.Adware, Koler.Ransomware, Biige.SMSmalware, AVpass.scareware*). Therefore, for the future work, we need to consider other types of feature such as memory dump, logs, permission, API calls, phone statistics (Figure 2) in developing a comprehensive and pervasive Android malware detection framework.

V. CONCLUSION

In this paper we have reviewed the most popular datasets, such as Genome, Drebin, AAGM and AMD, to find the serious drawbacks and shortcomings. The results show that malware

Table IV: Analysis results with 3 classifiers

Dataset:	**Training (10-fold cross validation)**									**Evaluation (Testing set)**								
Scenario:	**A (Malware Binary)**			**B (Malware Category)**			**C (Malware Families)**			**A (Malware Binary)**			**B (Malware Category)**			**C (Malware Families)**		
Algorithm:	RF	KNN	DT	RF	KNN	DT	RF	KNN	DT	RF	KNN	DT	RF	KNN	DT	RF	KNN	DT
Precision (%):	84.00	83.60	85.10	46.50	45.70	46.50	22.00	21.50	21.00	85.80	85.40	85.10	49.90	49.50	47.80	27.50	27.24	26.66
Recall (%):	87.50	87.30	88.00	45.50	44.80	44.70	21.50	21.60	21.40	88.30	88.10	88.00	48.50	48.00	45.90	25.50	23.74	20.06

will not express their actual malicious behaviour, unless they are installed on real devices. Also, a lot of malware applications are sensitive to activation trigger events and need user-interaction scenarios to be tricked, e.g. BOOT-COMPLETED. So, based on the aforementioned shortcomings, we proposed a systematic approach to generate an Android malware dataset using real smartphones instead of emulators and design different activation scenarios to trigger different families separately. A new dataset namely CICAndMal2017, generated includes 5,065 benign apps and more than 43 families of malware alongwith network traffic, memory dump, logs, permission, API calls, and phone statistics data. Many researchers are already focused on the network traffic analysis to detect and classify the malware. In this paper, we developed and extracted more than 80 network traffic features based on previous research for our dataset, the CICAndMal2017, and demonstrated that network traffic can be useful for detection (Binary classification), but can not be a good classifier alone. In future work, we will find and extract the useful features from other data sources such as memory dumps, logs, and API calls and will propose a complete malware family classifier and characterizer.

References

[1] Statistica, *Number of mobile app downloads worldwide in 2016, 2017 and 2021*, accessed by Jan 2018. https://www.statista.com/statistics/271644/worldwide-free-and-paid-mobile-app-store-downloads/.

[2] P. Cerwall, *Ericsson Mobility Report*, accessed by Jan 2018. http://www.ericsson.com/res/docs/2015/ericsson-mobility-report-june-2015.pdf/.

[3] G. Sterling, *Marketing Land report:Apple, Android Now 96 Percent Of Smartphones Globally*, 2015. http://marketingland.com/report-apple-Android-now-96-percent-smartphones-globally-119487.

[4] A. Arora, S. Garg, and S. K. Peddoju, "Malware detection using network traffic analysis in android based mobile devices," in *Next generation mobile apps, services and technologies (NGMAST), 2014 eighth international conference on*, pp. 66–71, IEEE, 2014.

[5] H. Gonzalez, N. Stakhanova, and A. A. Ghorbani, "Droidkin: Lightweight detection of android apps similarity," in *International Conference on Security and Privacy in Communication Systems*, pp. 436–453, Springer, 2014.

[6] A. H. Lashkari, G. Draper-Gil, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of tor traffic using time based features.," in *ICISSP*, pp. 253–262, 2017.

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

[8] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "Drebin: Effective and explainable detection of android malware in your pocket.," in *Ndss*, vol. 14, pp. 23–26, 2014.

[9] HCLR, *Hacking and Countermeasure Research Lab*, accessed by Jan 2018. http://ocslab.hksecurity.net.

[10] H. Kang, J.-w. Jang, A. Mohaisen, and H. K. Kim, "Detecting and classifying android malware using static analysis along with creator information," vol. 11, p. 479174, SAGE Publications Sage UK: London, England, 2015.

[11] J.-w. Jang and H. K. Kim, "Function-oriented mobile malware analysis as first aid," vol. 2016, Hindawi, 2016.

[12] J. wook Jang, H. Kang, J. Woo, A. Mohaisen, and H. K. Kim, "Andro-dumpsys: Anti-malware system based on the similarity of malware creator and malware centric information," vol. 58, pp. 125 – 138, 2016.

[13] J.-w. Jang, J. Yun, A. Mohaisen, J. Woo, and H. K. Kim, "Detecting and classifying method based on similarity matching of android malware behavior with profile," *SpringerPlus*, vol. 5, no. 1, p. 273, 2016.

[14] E. CIDRE, "Kharon dataset: Android malware under a microscope," *Learning from Authoritative Security Experiment Results*, p. 1, 2016.

[15] A. H. Lashkari, A. F. A. Kadir, H. Gonzalez, K. F. Mbah, and A. A. Ghorbani, "Towards a network-based framework for android malware detection and characterization," in *Proceeding of the 15th international conference on privacy, security and trust*, 2017.

[16] F. Wei, Y. Li, S. Roy, X. Ou, and W. Zhou, "Deep ground truth analysis of current android malware," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pp. 252–276, Springer, 2017.

[17] L. Xue, Y. Zhou, T. Chen, X. Luo, and G. Gu, "Malton: Towards on-device non-invasive mobile malware analysis for art," in *In 26th USENIX Security Symposium (USENIX Security 17). ACM*, 2017.

[18] Symantec, *2017 Internet Security Threat Report*, accessed by Nov 2017. https://www.symantec.com/security-center/threat-report.

[19] V. Total, "Virustotal-free online virus, malware and url scanner," *Online: https://www. virustotal. com/en*, 2012.

[20] V. Total, "Contagio mobile malware mini dump," *Online: http://contagiominidump.blogspot.ca/*, 2016.

[21] A. F. A. Kadir, N. Stakhanova, and A. A. Ghorbani, "Android botnets: What urls are telling us," in *International Conference on Network and System Security*, pp. 78–91, Springer, 2015.

[22] Amotzete, *Android mock location for development*, accessed by June 2017 2017. https://github.com/amotzte/android-mock-location-for-development.

[23] Oleavr, *Android Setting up your Android device*, accessed by June 2017 2017. https://www.frida.re/docs/android/.

[24] J. Long, *Welcome to AppMon!*, accessed by June 2017 2016. https://dpnishant.github.io/appmon/.

[25] Jtsylve, *LiME Linux Memory Extractor*, accessed by June 2017 2017. https://github.com/504ensicsLabs/LiME.

[26] J. Selvi, *Forensic Analysis On Android: A Practical Case*, accessed by June 2017 2015. https://www.sans.org/reading-room/whitepapers/forensics/forensic-analysis-android-practical-case-36317.

[27] A. Arora and S. K. Peddoju, "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.

[28] J. Li, L. Zhai, X. Zhang, and D. Quan, "Research of android malware detection based on network traffic monitoring," in *Industrial Electronics and Applications (ICIEA), 2014 IEEE 9th Conference on*, pp. 1739–1744, IEEE, 2014.

[29] S. Wang, Z. Chen, L. Zhang, Q. Yan, B. Yang, L. Peng, and Z. Jia, "Trafficav: An effective and explainable detection of mobile malware behavior using network traffic," in *Quality of Service (IWQoS), 2016 IEEE/ACM 24th International Symposium on*, pp. 1–6, IEEE, 2016.

[30] G. D. Gil, A. H. Lashkari, M. Mamun, and A. A. Ghorbani, "Characterization of encrypted and vpn traffic using time-related features," in *Proceedings of the 2nd International Conference on Information Systems Security and Privacy (ICISSP 2016)*, pp. 407–414, 2016.

[31] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: an update," *ACM SIGKDD explorations newsletter*, vol. 11, no. 1, pp. 10–18, 2009.