Towards a Network-Based Framework for Android Malware Detection and Characterization

Arash Habibi Lashkari, Andi Fitriah A.Kadir, Hugo Gonzalez, Kenneth Fon Mbah, Ali A. Ghorbani
Canadian Institute for Cybersecurity (CIC), UNB, Canada
(A.habibi.l, andi.fitriah, Hugo.gonzalez, kfon, ghorbani)@unb.ca

***Abstract*—Mobile malware is so pernicious and on the rise, accordingly having a fast and reliable detection system is necessary for the users. In this research, a new detection and characterization system for detecting meaningful deviations in the network behavior of a smart-phone application is proposed. The main goal of the proposed system is to protect mobile device users and cellular infrastructure companies from malicious applications with just 9 traffic feature measurements. The proposed system is not only able to detect the malicious or masquerading apps, but can also identify them as general malware or specific malware (i.e. adware) on a mobile device. The proposed method showed the average accuracy (91.41%), precision (91.24%), and false positive (0.085) for five classifiers namely; Random Forest (RF), K-Nearest Neighbor (KNN), Decision Tree (DT), Random Tree (RT) and Regression (R). We also offer a labeled dataset of mobile malware traffic with 1900 applications includes benign and 12 different families of both adware and general malware.**

***Keywords*-Android Malware, Malware Family, Malware Detection, Adware detection, Machine Learning;**

I. Introduction

Nowadays with the dramatic growth of available applications in the smart-phones, the users' behavior to interact and utilize the network has changed significantly for access via smart-phones. Besides, mobile networks traffic substantially increased because of many always-connected applications such as social networks apps. According to the Ericsson mobility report [17], by 2020, the subscription of smart-phones will be more than 6 billion and 80 percent of traffic will be generated by mobile networks. Also, more than 96% of global mobile traffic belongs to Android and Apple, which 50% of this traffic is generated by Android smart-phones [16].

Several issues have risen in smart-phones development. Firstly, with the rapid development of applications on the Android smart-phone due to market demand, an outbreak trend of malicious software has appeared. Secondly, Android is an open source operating system which makes it vulnerable to all sorts of malicious attacks. Furthermore, most of the malwares in smart-phones are connecting to the servers for sending and receiving information for attacks. These abundance malware applications and attacks caused serious damage, such as affecting normal usage, monitoring users' activities, and stealing users' private information on smart-phones. Hence, detecting malware apps before damage occurs is crucial [1] [5].

Most of the malware detection techniques which have been proposed can be categorized into static and dynamic groups [15]. Finding malicious characteristics or bad code segments in an app without running the app is called static detection. Preparing an isolated environment such as an emulator or any devices for running the app and monitoring the app's dynamic behavior is named dynamic detection. This research focuses on the common dynamic behavior of malware and aims to use the generated traffic of malicious apps installed on the real Android device for malware detection and labeling adware apps among others.

Our contribution: The main contribution of this research is to propose an Android Malware detection model based on a new network traffic feature set to expedite the efficiency of traffic classifier. For this purpose, nine new flow-based network traffic features presents for characterizing the benign, adware and apps. It means, not only can the proposed method detect the unknown Malware, but also it can label the type of Malware. Furthermore, a labeled dataset of mobile Malware traffic built, with 1900 benign and malicious apps in 12 different families. Droidkin, an apps similarity detector [25], shows the selected families are not similar and not closely related in our dataset.

The remaining part of this paper is organized as follows: In Section 2, we discuss the previous work on malware detection by network traffic and focuses on the lack of the related research. Section 3 presents our methodology and proposed solution. In Section 4 we describe our new labeled dataset which will be publicly available. Section 5 explains the experiments conducted on the dataset, while section 6 discusses the obtained results for malware detection and labeling. Finally, In section 7 the conclusion and the proposals for future work will be presented.

II. Literature Review

A number of studies have been conducted on area of Android malware detection and characterization from the network traffic perspective. One of the first attempts was presented by Iland *et al.* in 2011 [7], where the authors presented a lightweight approach of detecting Android malware and violations of user privacy through the network traffic

analysis. The authors conducted a series of controlled experiments. They first created the virtualized Android devices with a clean app (i.e. e-commerce app) installed from a third party market, and some dummy user data such as contact data, account passwords, browser history, and credit card information and infected them with 18 malware samples; then they analyzed the collected network traces to find information leakage behavior and identify connection attempts to command and control (C&C) servers; finally, they tested two approaches (i.e IP/DNS blacklisting, string matching) for detecting the malicious behavior of Android malware and defined four features (HTTP header flag, HTTP-GET request, content and pattern of POST request, well-structured identifiers in POST request). However the proposed approaches have several limitations. First, the blacklisting technique heavily relies on static malware behavior which requires frequent updating over time. Also, more complex attacks such as fast-flux and botnet are more difficult to effectively detect. The content matching technique infers information from plain-data transmission. But, this technique cannot be applied in the encrypted traffic.

Another detection technique using a sensor application was presented by Kuhnel & Meyer in 2012 [6]. They analyzed over 30 malware families targeting Android platform, which are divided into four categories: premium calls, SMS only, HTTP only, and Remote Access Tool (RAT). They also utilized the userspace layer of Android Architecture by adding a filtering component, which is responsible for network analysis and can be controlled via a sensor app. The sensor app is also capable of the following tasks: inform a user about suspicious traffic, list all events from a local database, send blocked SMS, and change blocking preferences. With 95% of detection rate, the authors claimed that it is sufficient to detect mobile malware by simply filtering the ingoing and outgoing traffic.

In 2013, Tenenboim *et al.* [11] proposed a novel network-based behavioral analysis for detecting a new group of malware with self-updating capabilities which have been found in the Google Android marketplace. Researches show that this group of malware are not detectable by using available static and dynamic analysis methods or any signature-based approaches. They defined a specific pattern for the generated traffic of each application on the device as a representative model. They used the machine learning to find the deviations of the pattern from normal applications' patterns.

They also have been used nine features such as number of total concurrent connections of the application, number of sent and received TCP or UDP packets, number of sent and received TCP or UDP payload bytes and number of sent and received TCP segments which they defined in their previous publication [12] with five fixed time intervals for calculating the various aggregation functions such as average, standard deviation, minimum, and maximum. For 15 selected applications, they analyzed the original benign and repackaged version which has been created by injecting malware code of five real applications and also ten self-written Trojan malware. Their results showed that in most of the applications the threat was detected in the first five minutes after the infection occurred [11].

In the same year, Dai *et al.* [13] presented a new automatic network profile generator for detecting Android applications in HTTP. They mentioned that regarding the huge number of applications that are appearing every day based on HTTP/HTTPS, traditional method of traffic classification are no longer useful for traffic analysis. In this project, they first ran thousands Android apps in an emulator and collected the network traces. Then for extracting apps' fingerprints, they proposed and developed a light-weight technique that can break the request to "method", "query" that can be split to key-values and "page" that can be broken into "page-components" and "filename". There are two limitations of the proposed technique. First the system needs a user seed path when login is involved and secondly it can not detect apps which have no distinct network behavior and use the same service.

Arora *et al.* [15] proposed an Android malware detection method using its network traffic analysis. They used an Android emulator on a host with public IP and captured the traffic of thirteen malware to create their dataset. Based on previous work, they have chosen sixteen traffic features such as average packet size, average duration of the flow, average number of bytes sent per flow. Then they using machine learning algorithms with their dataset and finalized seven features as the final feature set such as Average number of bytes received per second, average number of packets sent per flow, average number of packets received per flow. In the experiment and analysis section they categorized their dataset to three risky levels of Malware, namely high, medium and low. Their classifier correctly predicted 45 of 48 samples which is around 95% accuracy. The small size and lack of diversity of the Malware in the dataset, could be the major deficiencies of this research.

Similarly, Shabtai *et al.* [4] stated that the best detection technique for malware is using the application's network traffic patterns. They proposed a specific model based on some features such as sent/receive data in byte, network state, send/receive mode, total time in force/background and minutes since last active/modified time along with related aggregation functions such as minimum, maximum and standard deviation to represent a specific traffic pattern for each application. They, also used semi-supervised machine learning methods for defining the normal behavior and calculating the deviation for detecting the abnormal activities. For evaluation, they designed and implemented their method on Android devices with five real malware and ten in-house malware. For benign purpose, the original application were executed and for the malicious purpose, the repackaged versions injected with malware code were operated. The

evaluation results indicated that certain categories of applications can be detected by specific traffic patterns. Also, they realized that malware with self-updating ability have various and diagnosable traffic patterns for a few minutes after starting execution [4].

Li *et al.* [1] proposed a network traffic monitoring system for detecting Android malware in order to improve the Android terminal defense ability against malicious attacks, and Advanced Persistent Threat (APT) attacks. The system consists of four components: traffic monitoring, traffic anomaly recognition, response processing, and cloud storage. Overall, the system will perform the following steps: (1) parses the protocol of data packets, (2) extracts the feature data (ID of the process, the start of the network connection time, end of the network connection time, upward flow, downward flow, packet source IP address, packet destination IP address, protocol type, packet source port, destination port number), (3) uses an SVM classification algorithm for data classification, (4) determines whether the network traffic is abnormal, (5) locates the application that produced abnormal through the correlation analysis. The experiment results show that the monitoring system can effectively detect the Android malware with fewer false positives.

Another analysis focusing on the network-based malware detection system was conducted by Carrasquillo et al. [3] in 2014. The authors combined both the signature-based traffic analysis and the network flows analysis of a virtual private network (VPN) targeting the mobile platform. The goal is to provide alarms and visualization analytics of anomalous behavior to both mobile users and the network administrators. The framework of the detection system uses Snort (for the signature-based) and NetFlow by Cisco (for network flows analysis), which have the following features: Internet Protocol address (IP), the destination IP, the source port, destination port, the sum of the payload size of the packets, and timestamp.

Chen *et al.* [14] believed the main reasons that malware detection research in Android devices still remained theoretical are because of the lack of systematic analysis of traffic features and a large-scale repository of malware. Thus, for capturing the malware generated traffic data in real internet, they proposed and designed a behavior monitoring scheme for Android malware traffic and applied 24 different families of Android malwares. Regarding their analysis of the major compositions of the first 5 minutes of generated traffic, more than 99% of traffic were HTTP and DNS.

They used some common network features such as DNS query, HTTP packet length, HTTP request, and Ad traffic features in the analysis process and discovered that HTTP request can detect malware 40.89% and DNS query can detect 69.55%. Also, they illustrated that more than 70% of malwares started to generate the malicious traffic in the first 5 minutes. As Ad and malware are major HTTP traffic components, so Ad traffic can significantly affect malware detection process [14].

Zaman *et al.* [2] in 2015 demonstrated a work-in progress detection method that was effective against malware communicating with the remote server C&C servers, which is based on the logs of all remote locations. The authors divided their detection procedure into two steps (1) creating the URLs table, and (2) matching the URLs with blacklists. They conducted four main tasks in the first procedure of creating the URLs table: (1) packet dumping - recording incoming and outgoing network packets, (2) netstat logging - checking port numbers, (3) extracting necessary information from packet dump - filtering out all other packets from the packet dump, (4) Aggregating packet dump and netstat logs - creating a time-sequenced log of applications. They also discussed their direction of research toward developing an intelligent malware detection model.

Wang *et al.* [30] presented an effective malware identification and classification method called TrafficAV by combining machine learning algorithm and traffic analysis. In order to get minimum resources on mobile devices without affecting the user experience, the authors performed all data analysis and malware detection on the server side by mirroring the network traffic generated by mobile app from the wireless access point to the server for data analysis. The authors highlighted that with machine learning algorithm (C4.5 decision tree) the detection rates of TCP flow and HTTP models reach 98.16% and 99.65% while the false positive rates are 5.14% and 1.84%.

A recent paper in 2017 [29] presented an algorithm to prioritize network traffic features with minimize number of features to be analyzed for better detection accuracy and processing time. The results showed that 9 features out of 22 are sufficient to give maximum detection accuracy (85% up to 100%). Moreover, the authors claimed that these 9 features also save considerable amount of time for training and testing the dataset. The training time of 300 applications is reduced from 11.7 seconds to 5.8 seconds and testing time of 230 applications is reduced from 25.1 seconds to 17.3 seconds.

In addition to Android studies, we also reviewed some previous work on computer network traffic analysis. In 2005, Karagiannis *et al.* [26] presented BLINC, a multilevel traffic classification of host behavior at the transport layer. This paper was the first attempt in shifting from characterizing flows by application to associating hosts with applications at three different levels: (1) social level- capture source and destination IP addresses, (2) network level - analyze characteristics of the source and destination IP address, and the source port., and (3) the application level - capture additional flow information, such as the transport protocol or the average packet size. A range of application types was studied in this work, including network management traffic, mail, chat, media streaming, web, p2p, data transfer, and gaming. In order to respect the privacy, the authors

operated the study in the dark (having no access to packet payload, no knowledge of port numbers, and no additional information other than what current flow collectors provide). They represented these patterns by using graphs, which called graphlets and matched them with the library of host behavior under examination. The results showed that BLINC is able to classify 80%- 90% of the traffic with more than 95% accuracy.

Later in 2008, Nguyen & Armitage [27] conducted a survey to analyze techniques for Internet traffic classification using machine learning during the period of 2004 to early 2007. They reviewed 18 significant works and discussed the following: (1) the importance of IP traffic classification in operational networks., (2)the limitations of traditional port- and payload-based classification.,(3)the metrics for assessing classification accuracy.,and (4) the key requirements for the employment of ML-based classifiers in operational IP networks. The authors highlighted that most of the machine learning algorithms for offline analysis, such as AutoClass, Expectation Maximisation, Decision Tree, NaiveBayes etc. has demonstrated high accuracy (up to 99%). They also suggested that the promising results of machine learning-based IP traffic classification can open many new avenues for related research areas, such as the application of ML in intrusion detections, anomaly detection, routing traffic, real-time monitoring and management, and greynet or darknet networks.

Recently in 2016, Bartos *et al.* [28] presented a robust representation with a supervised method for classifying malware behaviors. Instead of classifying flows individually, the authors grouped flows into bags, where each bag contains flows that are related to each other. They also developed a novel method that combines the process of learning the representation with the process of learning the classifier. They deployed the system on large corporate networks and evaluated them on real HTTP network traffic with more than 43k malicious samples and more than 15M samples overall. The system detected 2,090 new and unseen variants of malware samples with 90% precision (9 of 10 alerts were malicious).

In summary, most of the related work from 2010 to early 2017 have been selected and evaluated in order to select a collection of commonly used features that could yield high malware detection accuracy based on network traffic. Table I shows the list of features extracted from the previous works that concern network traffic analysis and characterization along with a description. In comparison, the recent paper in 2017 [29] is the closest to our study. We used the same approach of feature set reduction and selection. However, the paper only have 22 total features while we added more features (79 total features with 24 categories) from our experiments and study.

III. Proposed Model

The main contribution of the proposed model is to detect Android malware and label them as specific malware (i.e. adware) or general malware. This operation, which is based on the detection of the network traffic deviation on mobile devices has been divided into three phases (Figure 1).

The first phase focuses on the gathering of benign, general malware, and adware apps for generating the training and testing dataset. Executing these apps on the smart phones, capturing the generated traffic, and labeling them creates the dataset.

The second phase consists of measuring all features including all previous work defined features and our new proposed features. Also, this phase trains the machine learning model with 80% of dataset (Training part) and suggests the most suitable feature set for the detection and labeling process.

The last phase computes the final selected feature set for labeling the new apps. This phase uses the 20% of dataset which has been separated from the whole dataset for testing and evaluating the proposed algorithm and feature set.

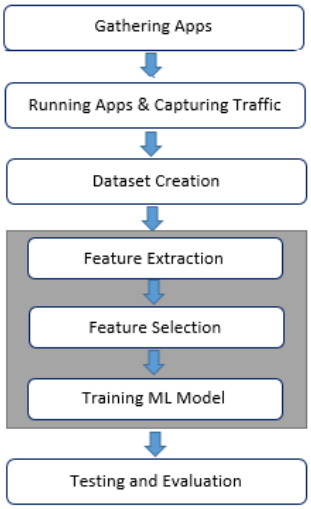

Figure 1: Procedure of Proposed Model

IV. Dataset

One of the contributions of this paper is a labeled Android network traffic dataset that we generated for our experiment. To generate traffic that represents the real world, we used real smart phones (NEXUS 5) instead of emulators or Android Virtual Device (AVD) [19] to ensure that our dataset is rich enough in quality and quantity. Table II shows the yearly breakdown of the collected dataset. We detected the samples creation date by using Androguard [8]. There are 696 samples that cannot be detected by Androguard and we labeled them as *altered*. We further analyzed these *altered* samples using another technique i.e. *Zipinfo* command and

Table I: Lists of network flow-based features fetched from previous works

Behavior-based		
Feature	Feature name	Description
F_1	Duration	The duration of the flow

Byte-based		
F_2	Total Forward Bytes	Total bytes in the forward direction
F_3	Total Backward Bytes	Total bytes in the forward direction
F_4	Forward Header Length	The total bytes used for headers in the forward direction
F_5	Backward Header Length	The total bytes used for headers in the backward direction

Packet-based		
F_6	Total Forward Packets	Total packets in the forward direction
F_7	Total Backward Packets	Total packets in the backward direction
F_8	Forward packet length (Min, Mean, Max, Std)	Min, Mean, Max, and standard deviation of the size of packet in forward direction
F_9	Backward packet length (Min, Mean, Max, Std)	Min, Mean, Max, and standard deviation of the size of packet in backward direction

Time-based		
F_{10}	Forward Arrival Time (Min, Mean, Max, Std)	Min, Mean, Max, and standard deviation time between two packets sent in the forward direction
F_{11}	Backward Arrival Time (Min, Mean, Max, Std)	Min, Mean, Max, and standard deviation time between two packets sent in the backward direction
F_{12}	Idle Time (Min, Mean, Max, Std)	Min, Mean, Max, and standard deviation time a flow was idle before becoming active
F_{13}	Active Time (Min, Mean, Max, Std)	Min, Mean, Max, and standard deviation time a flow was active before becoming idle

Flow-based		
F_{14}	Flow Packet Length (Min, Mean, Max, Std)	Min, Mean, Max, and standard deviation of the length of a flow
F_{18}	Flow Forward Bytes	The average number of bytes in a sub flow in the forward direction
F_{19}	Flow Backward Bytes	The average number of bytes in a sub flow in the backward direction
Flow-based (Proposed)		
F_{15}	Backward Variance Data Byte	Variance of total bytes used in the backward direction
F_{16}	Forward Variance Data Byte	Variance of total bytes used in the forward direction
F_{17}	Flow FIN	Number of packets with FIN
F_{20}	Idle (Max, Min)	Min/Max time a flow was idle before becoming active
F_{21}	Initial Window Forward	The total number of bytes sent in initial window in the forward direction
F_{22}	Initial Window Backward	The total number of bytes sent in initial window in the backward direction
F_{23}	Segment Size Forward (Max, Min)	Max/Min segment size observed in the forward direction
F_{24}	Segment Size Backward (Max, Min)	Max/Min segment size observed in the backward direction

discovered that the dates are altered by the attackers to be inaccurate. For example, some of them are created in 1930, 1931, and some are created in 2020, 2022.

Table II: The yearly breakdown of the collected apps

No	Creation Date (Year)	Benign	Malware
1	2008	18	0
2	2009	9	0
3	2010	2	1
4	2011	6	8
5	2012	7	2
6	2013	5	10
7	2014	59	27
8	2015	236	29
9	2016	765	20
10	Altered	393	303
Total		1500	400

A. Benign Dataset

We have collected 1527 benign apps from Googleplay market in 2015 and 2016. 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 1527 apps, 27 of them were removed (leaving 1500 apps in total) as they were flagged as suspicious or adware by more than two Anti-Virus (AV) products in Virustotal web service [9].

B. Malware Dataset

According to the recent Symantecs Internet Security Threat Report (SISTR), the normal distribution of benign and malware apps in the real world is 80% to 20% [21], so we selected 400 malware apps versus 1500 Benign apps to create a balanced dataset. We have collected these 400 malware from two categories, i.e. adware (250) and general malware (150). The first category, which is an adware consisting the following popular families:

- Airpush: Designed to deliver unsolicited advertisements to the user's systems for information stealing.
- Dowgin: Designed as an advertisement library that can also steal the user's information.
- Kemoge: Designed to take over a user's Android device. This adware is a hybrid of botnet and disguises itself as popular apps via repackaging.
- Mobidash: Designed to display ads and to compromise user's personal information.
- Shuanet: Similar to Kemoge, Shuanet also is designed to take over a user's device. Lookout, an Antivirus

company claimed that the authors of Kemoge and Shuanet used the same pieces of code to build their versions of the auto-rooting adware with 71 percent to 82 percent code similarity [22].

The following families have been chosen for the apps:

- AVpass: Designed to be distributed in the guise of a Clock app.
- FakeAV: Designed as a scam that tricks user to purchase a full version of the software in order to re-mediate non-existing infections.
- FakeFlash/FakePlayer: Designed as a fake Flash app in order to direct users to a website (after successfully installed).
- GGtracker: Designed for SMS fraud (sends SMS messages to a premium-rate number) and information stealing.
- Penetho: Designed as a fake service (hacktool for Android devices that can be used to crack the WiFi password). The malware is also able to infect the user's computer via infected email attachment, fake updates, external media and infected documents [23].

To further analyze our dataset, we employed Droidkin [25], a lightweight detector of Android apps similarity, to those in our dataset. Droidkin is used to investigate the relationships between each apps category (adware, general malware, and benign). Figure 2 visualizes the result of the detection analysis. The big red circle symbolizes apps category and the small black circle represents the apps that belong to its category. Overall, there is a weak-relationship between all three categories. The same weak-relationship also shown between the two malware categories. This result indicates that the selected malware families in our dataset are diverse, which is very important in reducing the unbalanced data for further analysis.

C. Capturing traffic from real smartphone

After defining group of apps, we proceed with the apps installation on the real smartphone, which is on NEXUS 5. We first configured the NEXUS 5 by syncing the already-installed apps such as Gmail, Facebook, and YouTube with the user profile named *iscx*. We created this *iscx* profile to make sure that the smartphone was connected with registered email account.

We then connected the smartphone to the computer by turning on the setting of *USB debugging* and *developer access*. Then we ran a script to install the apps in bulk. We first installed the benign apps in groups of 20 and start running all apps and have some interactions in the following hour. Then the apps were removed and new group of apps were installed. All the traffic was captured in the access point. We only captured the traffic which was coming from the smartphone.

We followed the same procedure for installing the general malware and adware groups. However, in this case, the smartphone was reset and reconfigured after each installation to make sure that the smartphone was clean and not infected with the malware (before installing new malware apps).

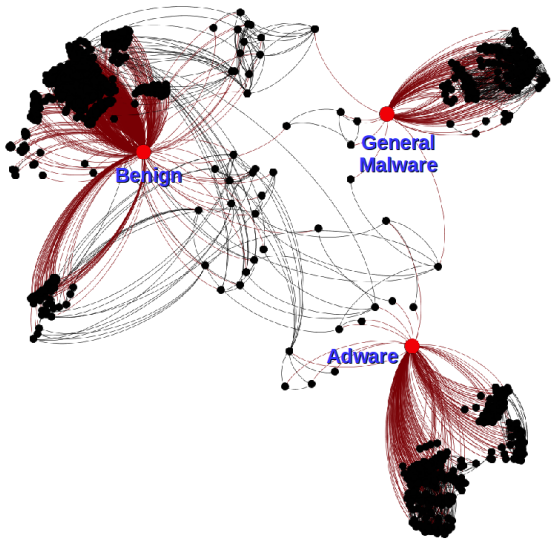

Figure 2: Detection of apps similarity using Droidkin [25]

V. Experiments

There is a common definition for the network flow, where 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 (NEXUS 5) and capturing the generated traffic at the gateway level, all traffics have been labeled in three classes (adware, general malware, and benign). Then we conducted the feature extraction and selection, to find the best set of features, before the testing and evaluation process which explained in the next section.

A. Feature Extraction and Selection

Table I shows the list of features that we found in previous works in this domain (Section II). Since some features were not useful for our classification, so we removed them from the list. For example, most of the Android apps are using HTTP/HTTPS for accessing to the internet, and more than 70% among them are not using HTTPS [13], so we eliminated the *protocol* feature from the list. Besides, as we used the *source port, destination port, Source IP, and Destination IP* for the flow generation and labeling process, so we removed them from the list of gathered features also. We categorized all listed features into four groups: Behavior-based, Byte-based, Packet-based, and Time-based. But, based on our study on the network traffic flows, there are more network traffic features that are not used in the previous work and we want to propose some of them for our experiment (Flow-based section in Table I). Also, to

provide a concise representation of the extracted traffic we used the aggregation modules such as Maximum, Minimum, Mean and standard deviation in the proposed features.

For extracting the feature vector, we used the available flow generator and feature extraction application, CICFlowMeter [18]. As the developer of CICFlowMeter mentioned, their open source application has been developed in Java and it is easy to expand, so anyone can implement his/her own features. Overall, we have extracted our seventeen new features for 395,828 instances of traffic flows using the CICFlowMeter.

As we need two separate datasets for training/testing and evaluation, we divided our datasets accordingly, 80% for training/testing and 20% for evaluation. Therefore, we used the pre-process function available in Weka (resample with noReplacement) [10] [20] 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 three feature selection algorithms, i.e. Information Gain, Cfs Subset, and SVM in Weka over the training dataset. However, Cfs Subset shows the least number of features as listed in Table III. To have the best feature set, we selected the common features of these three algorithms which are:

1) Maximum flow packet length F_{14}
2) Minimum flow packet length F_{14}
3) Backward variance data bytes F_{15}
4) Flow FIN F_{17}
5) Flow forward bytes F_{18}
6) Flow backward bytes F_{19}
7) Maximum Idle F_{20}
8) Initial window forward F_{21}
9) Minimum segment size forward F_{23}

So, for all test and evaluation scenarios we use these 9 common features as the result of features selection process.

Table III: Summary of feature selection results

No	Attribute Evaluator	Search Method	Features
1	CfsSubsetEval	Best First	9
2	InfoGainAttributeEval	Ranker	11
3	SVMAttributeEval	Ranker	79

Three different scenarios have been defined to test and analyze the proposed model. Scenario A: to distinguish the malware apps from benign apps, Scenario B: to distinguish the general malware apps from benign apps and also the adware apps from benign apps, and scenario C: to characterize and label Benign apps, general malware apps, and adware apps. Five common classifiers namely Random Forest (RF), Decision Tree J48 (DT), Random Tree (RT), K-Nearest Neighbors (KNN), and Regression (R) have been executed in all scenarios.

Four common metrics, Precision (Pr) or Positive Predictive, Recall (Rc) or Sensitivity, Accuracy and False Positive (FP) rate have been selected to evaluate the quality of the classification process. The Pr is equal to the ratio of correctly classified instances (TP), let us say X, in front of all the instances classified as X (TP+FP). Whereas the Rc is equal to the ratio of correctly classified instances (TP), let us say Y, in front of all Y instances (TP+FN).

$$Pr = \frac{TP}{TP + FP} \qquad Rc = \frac{TP}{TP + FN}$$

Where the TP is True Positive, FP is False Positive, and FN is the False Negative. In the testing step we used the weighted average of precision, recall and accuracy to choose the best combination of data set and features. In Weka, the weighted average of precision is calculated as:

$$WPr = \frac{\sum Pr_{Ci} * (TP_{Ci} + FP_{Ci})}{(TP + FP + TN + FN)}$$

where Pr_{Ci} is the precision of class Ci, $TP_{Ci} + FP_{Ci}$ is the number of samples classified as Ci, and $TP + FP + TN + FN$ is the total number of samples. The weighted recall and accuracy is calculated following the same procedure.

Scenario A:

In this scenario all general malware and adware are labeled as malware and the dataset has two classes benign and malware. Figure 3 scenario A, shows the results of conducting five common classifiers for this scenario. The results indicate that the RF algorithm using just nine selected features with 93% accuracy has the highest detection rate. The average of all five algorithms shows about 93% detection accuracy and for all of them the probability of success (Precision) is about 92%. The average of FP rate is 0.07% and the maximum rate is 0.08 in Regression algorithm which means the number of incorrectly classified instances is really low and it is good for real world malware detection development.

B. Testing and Analysis

Scenario B:

The first malware characterization is between general malware and benign apps. Figure 3 scenario B1, presents the result result of the first characterization on scenario B for the five selected ML algorithms. The average accuracy for all testing algorithms is more than 99%, with the 0.0065% FP rate, which is showing high correct detection and very low incorrect labeling. The average probability of success in front of all FP (Precision) and the probability of success in front of all FN (Recall) for all five algorithms is more than 99.2% and 99.4% respectively which shows a clear relation between proposed feature set and the efficiency of the algorithm. The best result comes from decision tree algorithm with 99.29% accuracy and 0.007% false positive

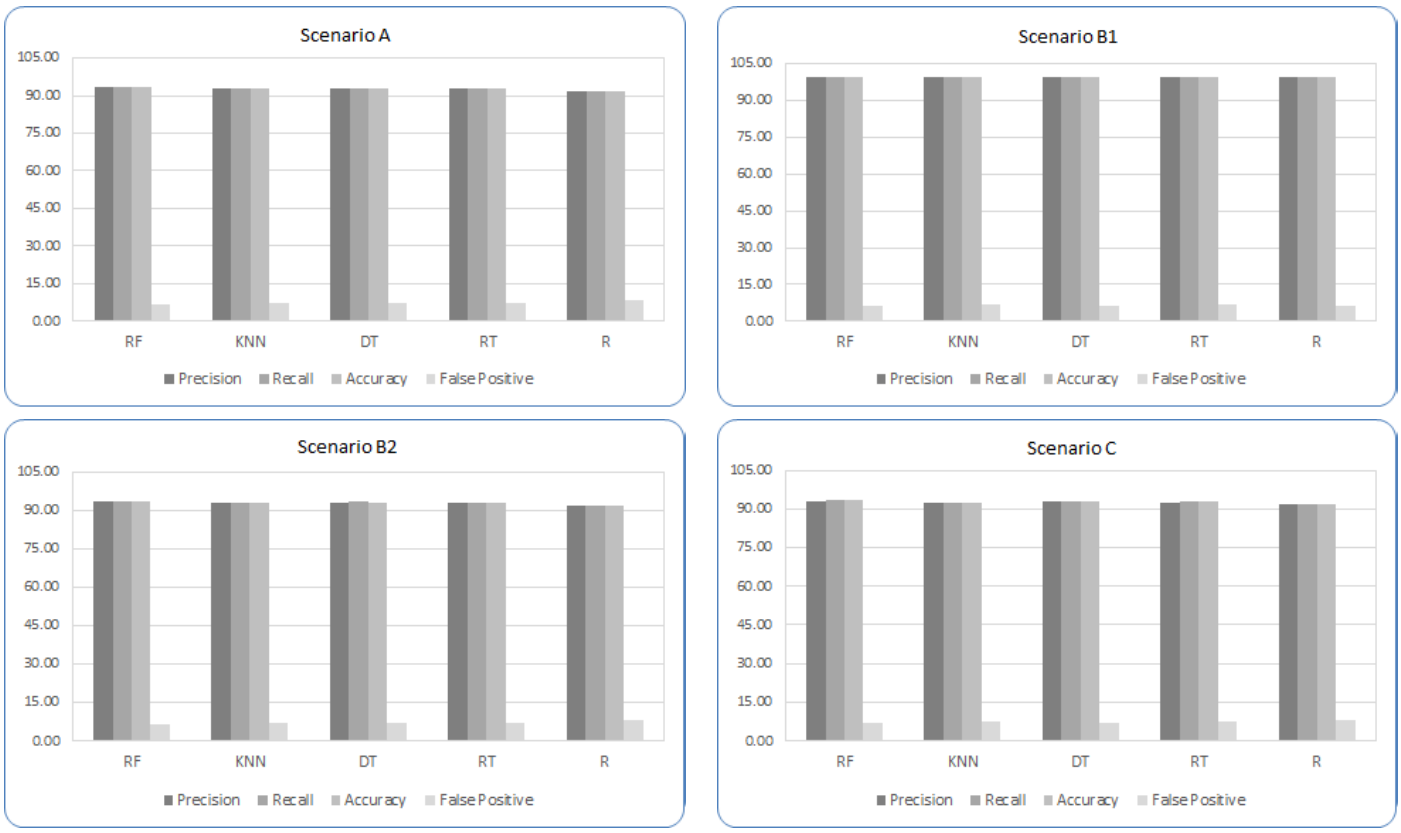

Figure 3: The results of testing process for four scenarios

that present strong detection rate.

On the other hand, Figure 3 scenario B2 presents the result of the second characterization on scenario B, which is detecting adware from benign (B2). The result shows that all five ML algorithms achieved the very high accuracy correct labeling with average 93.3% with the very low FP about 0.06% on average. The probability of success in front of all FP (Precision) on average is 93% while the average on the probability of success in front of all FN (Recall) is 93%. According to robust results on both sub scenarios of scenario B (B1, B2), the proposed feature set is strong enough to go for the complicated experiments level which is labeling all three classes together (Scenario C).

Scenario C:

Figure 3 shows the result of last scenario, which contain the detection and characterization of all categories: adware, general malware, and benign. The results with the proposed nine features present 92% accuracy on average with maximum 0.08% FP rate across five machine learning algorithms. The probability of success for all algorithms on average is 92% which is extremely accurate.

VI. Evaluation and Discussion

According to the experiment, we can conclude that 9 features are sufficient in distinguishing malware from benign and also characterizing the malware apps (general malware, adware). To evaluate the proposed model and address the reliability of the results, we validate the proposed model by running the classifiers over the second dataset (validation dataset) which is completely separated from the training/testing one (20% of original dataset). Secondly, we compare the dataset and results with the other available models which shown high accuracy in their project.

Table IV shows the results of this validation, in detection (Scenario A) and labeling (Scenario C) processes; the minimum accuracy is from Regression (R) algorithm with average accuracy 90.43%, 90.47 and average false positive 0.096%, 0.095% respectively. Of several suitable classification algorithms for handling numerical data that were evaluated, the Random Forest (RF) classifier surpasses the others with average accuracy of 92.18%, 92.09% and average false positive rate 0.07%, 0.07% respectively.

On the other hand, in comparing the proposed model with the previous works, having a comprehensive and complete data set with diverse enough families of instances is one of the major lacks. Much research has used the Android Virtual Device (AVD) for creating their dataset and conducted their experiments based on generating traffic

Table IV: Evaluation results for three scenarios

Scenario:	A (Malware, Benign)					B1 (General Malware, Benign)					B2 (Adware, Benign)					C (Adware, General Malware, Benign)				
Algorithm:	RF	KNN	DT	RT	R	RF	KNN	DT	RT	R	RF	KNN	DT	RT	R	RF	KNN	DT	RT	R
Precision:	0.921	0.914	0.915	0.915	0.903	0.991	0.989	0.992	0.99	0.992	0.924	0.918	0.918	0.92	0.907	0.919	0.912	0.914	0.914	0.903
Recall:	0.922	0.915	0.916	0.916	0.904	0.922	0.991	0.993	0.991	0.993	0.925	0.919	0.919	0.921	0.909	0.921	0.914	0.916	0.915	0.905
Accuracy:	92.18	91.51	91.61	91.59	90.43	99.25	99.05	99.29	99.08	99.28	92.48	91.92	91.93	92.07	90.89	92.09	91.36	91.61	91.54	90.47
False Positive:	0.078	0.085	0.084	0.084	0.096	0.008	0.009	0.007	0.009	0.007	0.075	0.081	0.081	0.079	0.091	0.079	0.086	0.084	0.085	0.095

by the emulators which is not reliable and exactly similar to the real world [14] [5] [7] [15].
Also, the number of benign and malicious families included in the dataset are not diverse and distributed enough, i.e. Zaman *et al.* has two samples [2], Iland *et al.* included 18 instances of malware [7], Chekina *et al.* used 5 benign malware with the injected version of them [12], Tenenboim evaluated his model with 15 malware which included 5 real and 10 self-written apps with the repackaged version of same apps [11], Arora *et al.* used 13 respondent samples from 27 available families [15] and Alam *et al.* mixed 1330 malicious apps with 407 benign apps which is 3 times more and unbalanced [24]. So one of the main reasons for the high accuracy in previous work is that the experiments were based on the incomplete dataset.
By summarizing the results of testing and evaluation of the present research work, we conclude that with the high accuracy (average 93%), high probability of success (PR average 92%) and low FP rate (average 0.07%), the proposed method is really feasible and very effective for Android malware detection and characterization. Our future research direction is developing a malware detection app based on the proposed features for the Android smart-phones.

It is important to highlight that the selected 9 features in our study are all belongs to the flow-based category. In contrast with other categories (i.e. behavior-based, byte-based, packet-based, and time-based), it is impossible for the attacker to modify or mimic their network traffic to evade our proposed system. For example, the attackers can learn the patterns of benign and malicious apps then try to mimic the traffic such as the duration of the flow (behavior-based), but it is impossible for them to modify the variance of total bytes used in the backward direction (flow-based).

Our limitation: One limitation of our approach is that we have a limited user-interaction with the installed apps. Although the apps' installation was done automatically, but the apps' interaction was done manually (i.e. clicking buttons, browsing windows). We aware that this interaction is not the representative of the actual user-interaction.

VII. Conclusion

In this research, we proposed a mobile malware detection model based on 9 traffic features to expedite the efficiency of traffic classifier. Moreover, the model uses classification methods including flow-based, packet-based and time-based features to characterize malware families. The analysis shows the proposed feature set has more than 93% accuracy in the detection and 92% success probability on characterization with less than 0.08 percent false positive rate on average, which is adequately good and necessary for real world malware detection systems. In future work, we plan to include other features such as system and host information together with the traffic features for the malware characterization and propose a complete detection system for smart-phones.

References

[1] Li, Jun and Zhai, Lidong and Zhang, Xinyou and Quan, Daiyong, Research of Android malware detection based on network traffic monitoring, Industrial Electronics and Applications (ICIEA), 2014 IEEE 9th Conference on, p1739-1744, 2014

[2] Zaman, Mehedee and Siddiqui, Tazrian and Amin, Mohammad Rakib and Hossain, Md Shohrab, malware detection in Android by network traffic analysis, 2015 International Conference on Networking Systems and Security (NSysS),p1-5, 2015

[3] Carrasquillo, Abimael and Maldonado, Albert E and Santos, Eric and Ortiz-Ubarri, José, Poster: Towards a framework for Network-based malware detection System, 35th IEEE Symposium on Security and Privacy, 2014

[4] Shabtai, Asaf and Tenenboim-Chekina, Lena and Mimran, Dudu and Rokach, Lior and Shapira, Bracha and Elovici, Yuval, Mobile malware detection through analysis of deviations in application network behavior, Computers & Security journal Elsevier, vol 43, p1-18, 2014

[5] Dai, Shuaifu and Tongaonkar, Alok and Wang, Xiaoyin and Nucci, Antonio and Song, Dong, Networkprofiler: Towards automatic fingerprinting of Android apps, 2013 Proceedings IEEE INFOCOM, p809-817, 2013

[6] Marian Kuhnel and Ulrike Meyer, Detecting malware initiating traffic on mobile device, RWTHAACHEN university, 2012

[7] Danny Iland, Alexander Pucher and Timm Schauble, Detecting Android malware on network level, University of California, Santa Barbara, vol 12, 2011

[8] Androguard, https://github.com/androguard/androguard, Accessed Dec 2016

[9] Virus Total, https://www.virustotal.com/en/, Accessed Dec 2016

[10] The WEKA Data Mining Software: An Update, Hall, Mark and Frank, Eibe and Holmes, Geoffrey and Pfahringer, Bernhard and Reutemann, Peter and Witten, Ian H., ACM SIGKDD Explorations Newsletter, vol 11, no. 1, 2009

[11] Tenenboim-Chekina, L and Barad, O and Shabtai, A and Mimran, D and Rokach, L and Shapira, B and Elovici, Y, Detecting application update attack on mobile devices through network feature, 2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), p91-92, 2013

[12] Chekina, Lena and Mimran, Dudu and Rokach, Lior and Elovici, Yuval and Shapira, Bracha, Detection of deviations in mobile applications network behavior, arXiv preprint arXiv:1208.0564, 2012

[13] S. Dai and A. Tongaonkar and X. Wang and A. Nucci and D. Song, Network Profiler: Towards automatic fingerprinting of Android apps, Proceedings IEEE INFOCOM,p809-817, 2013

[14] Z. Chen and H. Han and Q. Yan and B. Yang and L. Peng and L. Zhang and J. Li, A First Look at Android malware Traffic in First Few Minutes, Trustcom/BigDataSE/ISPA, Vol 1, p206-213, 2015 IEEE

[15] A. Arora and S. Garg and S. K. Peddoju, malware Detection Using Network Traffic Analysis in Android Based Mobile Devices, Eighth International Conference on Next Generation Mobile Apps, Services and Technologies, 2014

[16] Marketing Land report, http://marketingland.com/ report-apple-Android-now-96-percent-smartphones-globally-119487, Accessed Dec 2016

[17] Ericsson Mobility Report,http://www.ericsson.com/res/docs /2015/ericsson-mobility-report-june-2015.pdf, Accessed Dec 2016

[18] Draper-Gil, Gerard and Habibi Lashkari, Arash and Mamun, Moham- mad Saiful Islam and Ghorbani, Ali A., Characterization of Encrypted and VPN Traffic using Time-related Features, Proceedings of the 2nd International Conference on Information Systems Security and Privacy (ICISSP16), 2016, SCITEPRESS

[19] James Talbot and Justin McLean, Learning Android Application Programming: A Hands-On Guide to Building Android Applications (1st ed.), Addison-Wesley Professional, 2014

[20] Hall, Mark and Frank, Eibe and Holmes, Geoffrey and Pfahringer, Bernhard and Reutemann, Peter and Witten, Ian H., The WEKA Data Mining Software: An Update, ACM SIGKDD Explorations Newsletter, v.11, no.1,2009, p10-18, ACM, New York, NY, USA

[21] Symantec Internet Security Threat Report, https://www.symantec.com/security-center/threat-report, Accessed Dec 2016

[22] Lookout discovers new trojanized adware, https://blog.lookout.com/blog/2015/11/04/trojanized-adware/, Accessed Dec 2016

[23] How to Remove Android Penetho, http://www.solvusoft.com/ en/malware/viruses/Android-penetho/, Accessed Dec 2016

[24] M. S. Alam and S. T. Vuong, "Random Forest Classification for Detecting Android malware," 2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing, Beijing, 2013, pp. 663-669.

[25] Gonzalez, H., Stakhanova, N. and Ghorbani, A.A., 2014, September. Droidkin: Lightweight detection of Android apps similarity. In International Conference on Security and Privacy in Communication Systems (pp. 436-453). Springer International Publishing.

[26] Karagiannis, Thomas, Konstantina Papagiannaki, and Michalis Faloutsos. BLINC: multilevel traffic classification in the dark. ACM SIGCOMM Computer Communication Review. Vol. 35. No. 4. ACM, 2005.

[27] Nguyen, T. T., and Armitage, G. (2008). A survey of techniques for internet traffic classification using machine learning. IEEE Communications Surveys & Tutorials, 10(4), 56-76.

[28] Bartos, Karel, Michal Sofka, and Vojtech Franc. "Optimized Invariant Representation of Network Traffic for Detecting Unseen Malware Variants." USENIX Security Symposium. 2016.

[29] Arora, Anshul, and Sateesh K. Peddoju. "Minimizing Network Traffic Features for Android Mobile Malware Detection." Proceedings of the 18th International Conference on Distributed Computing and Networking. ACM, 2017.

[30] Wang, Shanshan, et al. TrafficAV: An effective and explainable detection of mobile malware behavior using network traffic. Quality of Service (IWQoS), 2016 IEEE/ACM 24th International Symposium on. IEEE, 2016.