Android Malware Classification based on Mobile Security Framework

Shefali Sachdeva, Romuald Jolivot, and Worawat Choensawat

***Abstract*—In this paper, a machine learning based technique is proposed to classify android applications in three classes based on the confidence level defined as safe, suspicious and highly suspicious. Thirty six features are extracted and selected from Mobile Security Framework based on penetration testing. A set of experiments has been conducted on the scale of 13,850 android applications which includes 8,782 android applications downloaded from apk-dl.com, 3,960 malware and 1,108 benign applications. In order to compare the accuracy of the classification model, a ground truth of the confidence level is created by using VirusTotal. The proposed method can detect and classify android applications into three confidence levels with 81.80% accuracy. Experiment for binary classification, classify as being malware or benign has yielded 93.63% accuracy.**

***Index Terms*—Android Malware, Malware Detection, VirusTotal and Classification Model, Benchmark Creation, Machine Learning.**

I. Introduction

ANDROID has emerged as the most popular mobile operating system in recent years. Android operating system is a freeware, open source and customizable and it offers a great platform to develop and release various type of applications and games. This allows users to perform multiple number of operations including accessing bank, social media and cloud accounts and has replaced computers in many aspects. According to a report from International Data Corporation (IDC) [1], more than 85.00% of the mobile devices are running on android and attracting a large number of mobile application developers.

Among these developers, some have malicious intentions and create malware applications. This has become a global threat [2], [3], [4], [5] and comes in different forms such as banking malware, mobile ransomware, mobile spyware, MMS malware, mobile adware, and SMS Trojans. For example, financial botnets such as ZeuS or ZITMO (ZeuS-in-the-mobile) [6] are getting very popular and can lead to a threat in banking organizations. These malware programs are capable of stealing important information from the devices and can conduct financial fraud.

To protect users against these threats, companies and research scholars have developed malware detection software and keep investigating the subject. Many anti-malware software rely on signature based approaches [7], [8], [9] in which hash code is compared with malicious files. Although signature based approaches are reliable for known malware. These approaches fail against zero day malware attacks [6] or when the malware mutates. The signature based approaches of scanning android applications using antivirus software also fail when the applications are repackaged. To overcome this problem, many researches applied machine learning techniques [10], [11] to detect malware. Currently, most techniques focus on detecting whether an application is malware or benign. There exist multiple methods which are usually based on features extracted from the applications. Different algorithms such as neural network, support vector machine, naive bayes, etc. have been used to detect malware with relatively good accuracy. As malware keeps evolving, a method providing a confidence level on whether the application is safe, suspicious or highly suspicious in terms of being malicious is implemented. This could entrust the user with the choice of installing the applications based on the applications' confidence level.

This paper proposed a classification model capable of determining the confidence level of various android applications. The confidence level refers to the likeliness of an application being malicious. The problem of building a machine learning-based classifier to identify the confidence level presents two main challenges: first, we must extract features representation of the application; second, based on our collected dataset, we have to come up with a method to classify them into each level of confidence. Our contributions can be described in two parts: (1) defining a confidence level ground truth and (2) developing of a machine learning-based classifier to evaluate the confidence level of android applications.

To address the first problem, we use Mobile Security Framework [12], [13] to extract a heterogeneous feature set, and process for feature selection based on statistical analysis (described in Section III-B). To address the second problem, we create a ground truth for labeling each application into one of the three confidence levels, namely safe, suspicious and highly suspicious. The ground truth creation is based on the results of VirusTotal [14]. We believe that the proposed classification model provides a new way of warning users by categorizing android applications based on the confidence level.

II. Literature Review

Following Android popularity, mobile devices are becoming more exposed to malicious attacks. According to a report from Symantec [15] 18.50 million mobile malware were detected in 2016, which is 105.00% more than previous year. Many researchers have proposed different android malware detection techniques. An overview of the latest techniques is presented in this section.

Manuscript received March 06, 2018; revised August, 27, 2018.

Shefali Sachdeva, School of Information Technology and Innovation, Bangkok University, Bangkok, Thailand (Email: 14ssachdeva@gmail.com, shefali.sach@bumail.net).

Worawat Choensawat, School of Information Technology and Innovation, Bangkok University, Bangkok, Thailand (Email: worawat.c@bu.ac.th).

Romuald Jolivot, Department of BU-CROCCS, School of Engineering, Bangkok University, Bangkok, Thailand (Email: romuald.j@bu.ac.th).

A. Malware detection based on static analysis

Static analysis approaches focus on comparing programs to known malware based on the static information such as the program code, permissions, system commands and intents looking for signatures or patterns [16]. Based on the static information, many researchers applied Machine learning techniques for malware detection [17], [10]. A machine learning model ANASTASIA [18] is proposed by extracting features like used permissions, system commands, intents, etc. This approach can classified as benign or malware with 97.30% accuracy. The method is based on static analysis, hence the detection can be affected by repackaging techniques such as code permutation, compression, etc. While K. Zhao et al. [19] have introduced feature based machine learning technique known as FEST: Feature Extraction and Selection Tool and has an accuracy of 98.00%. The method has used dataset of 7,972 applications for feature extraction such as permissions, API, Action, IP and URL features.

Building a classification model for malware detection required known dataset for training stage. Recently, many researches have been conducted by using VirusTotal API to determine whether the application is malware or a benign [20], [21], [22], [23] [24]. VirusTotal uses multiple antivirus scan engines ranges from popular vendors to small companies, the results may vary among scans engines.

Martin et al. [20] have proposed a method to analyze and detect malicious android applications using meta-information: ADROIT, which is based on a text mining process and can be used to extract relevant information from the meta-data. In the malware labeling process, the API provided by the VirusTotal online portal is used which allows analyzing applications with 56 different antivirus or scan engines. If at least one scan engine tested positive the application is marked as malware. However, the trustworthiness of scan engines is not taken into consideration.

B. Malware detection based on dynamic analysis

While static analysis approaches focus on static information of the application, dynamic analysis approaches aim at monitoring usage behaviors during run-time [25]. Several approaches [26], [27] monitor the power usage of applications, and report anomaly consumption. Others [5], [28] [29], [30] monitor system calls and attempt to detect unusual system call patterns. Although dynamic analysis approaches are effective in identifying malicious activity, run-time monitoring often suffer from a large amount of overhead and cannot be directly run on mobile devices. A hybrid method [31] is proposed by combining static and dynamic analysis and performs with 90.00% accuracy. For collecting the system calling data of an application at runtime, dynamic analysis is conducted and for the testing of the same data static analysis is performed. Firstly, a set of patterns is defined for both malware and benign applications and then the applications are tested. However, the approach is limited to the database of already save dataset patterns.

Most of the methods analyze application in binary fashion that is whether the application is malicious or benign. In our study, instead of categorizing the application in two categories, a technique is defined which categorize the confidence level of android applications' likeliness of being malicious.

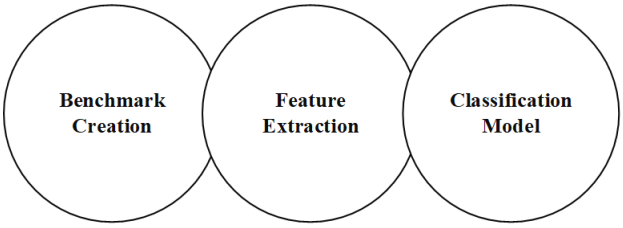

Fig. 1. Methodology for Malware Classification.

The decision of installing application is left to the user based on the confidence level.

III. Methodology

The objective of the proposed approach is to develop a classification model and to determine the confidence level of android applications. The methodology is divided into three modules as shown in figure 1. In this work, we gathered a collection of 8,782 Android applications downloaded automatically from apk-dl.com [32]. In order to label them correctly based on the confidence levels, datasets of known malware and benign are required. For that task, We use 3,960 known malware applications provided by [18], [33] and 1,108 known benign applications. Known benign applications consist of 216 system applications and 892 most popular (most downloaded and has four and above ratings given by the users) applications downloaded manually from Google play store.

A. Benchmark Creation using VirusTotal

The interest of a benchmark creation is to identify the confidence level of android applications. The benchmark is created using VirusTotal. VirusTotal is a subsidiary of Google and is a web based application. It provides free and unbiased service for analyzing and scanning applications, URLs and documents. At the time of writing (December 2017), VirusTotal is using 77 scan engines or antivirus from popular and small companies. Scan engines are developed using different algorithms. The results may vary from one scan engine to other when submitting identical applications. Therefore, the reliability of scan engines has to be taken into account for the benchmark creation and need to be individually weighted. For this purpose a weight is defined by statistical analysis for each scan engine. In order to test the reliability of scan engines used by VirusTotal, a dataset of known malware and benign applications is submitted, verifying that the scan engine can correctly detect malware as malware and benign as benign.

1) Scan-engine weight assignment: A dataset of 3,960 known malware and 1,108 known benign applications is submitted to VirusTotal. By knowing that submitted application is a known malware or benign application, it is possible to determine the accuracy of each scan engine. A true positive rate is defined when the malware is detected as malware and true negative rate is when a benign application is detected as benign. The true positive and negative rate of each scan engine is calculated and equation 1 gives the weight of the scan engine.

$$W_i = \frac{(TP * MS) + (TN * BS)}{MS + BS} \tag{1}$$

where W_i is scan engine weight of scan engine i. TP and TN are true positive rate and true negative rate, respectively. MS and BS are number of malware scanned and number of benign applications scanned, respectively. Example results of scan engine weight is shown in the table I.

In order to perform functional scan engines selection, two conditions are taken into account: first, scan engines with less than 50% true positive and negative rate are discarded, and second, scan engines which are scanning less than 50% of the whole dataset for benign and malware samples are also evicted. Hence, 38 scan engines are selected and an example list is shown in the table I.

2) Confidence Level Determination: After scan engine weight assignment, a score is calculated for known malware and benign applications using equation 2. A range for confidence level is determined by using this score. On the basis of application scores, the confidence level is divided into three classes which are safe, suspicious and highly suspicious. Figure 2 and figure 3 shows the scores for submitted benign and malware applications, respectively. As shown in the figure 3, no malware sample has an application score less than or equal to 15. Therefore, the safe range is empirically defined from 0 to less than and equal to 15.

Let S be a set of the scan engines returning the result in a given time. An application score is calculated from an average of returned results from the scan engines using equation 2.

$$Score = \frac{\sum_{i \in S}(W_i * VTR_i)}{\sum_{i \in S} W_i} * 100 \qquad (2)$$

where *Score* is an application score, W_i is the weight of scan engine i, VTR_i is a returned result of VirusTotal from scan engine i (1 for malware or 0 for benign application).

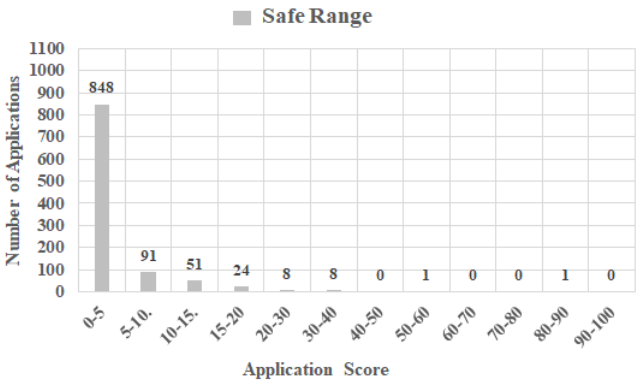

Fig. 2. Application Scores of Submitted Benign Applications.

For the suspicious and highly suspicious range definition, the highly suspicious range is defined for a score greater than 50 since more than the majority of the weighted indicated application as malware. Similarly, based on the application score, the suspicious range is defined from greater than 15 to less than equal to 50 as shown in figure 3. Table II shows the confidence level range defined for safe, suspicious and highly suspicious classes.

3) Applications Labeling: Application scores of all malware and benign applications are calculated and labeled using the confidence level determination process. Similarly, this process of application labeling is performed for a dataset of

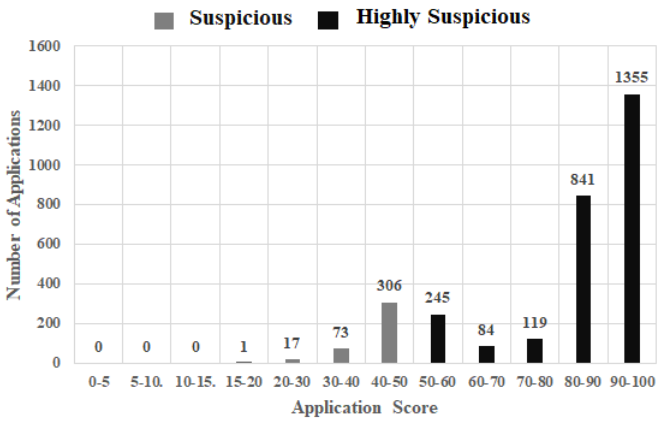

Fig. 3. Application Scores of Submitted Malware Samples.

random unknown applications (8,782) downloaded from apk-dl.com. Figure 4 shows the overall process of the benchmark creation. Table III shows the number of applications in each class after all the applications is submitted for labeling. Based on the benchmark creation, these labeled applications are classified as safe, suspicious and highly suspicious.

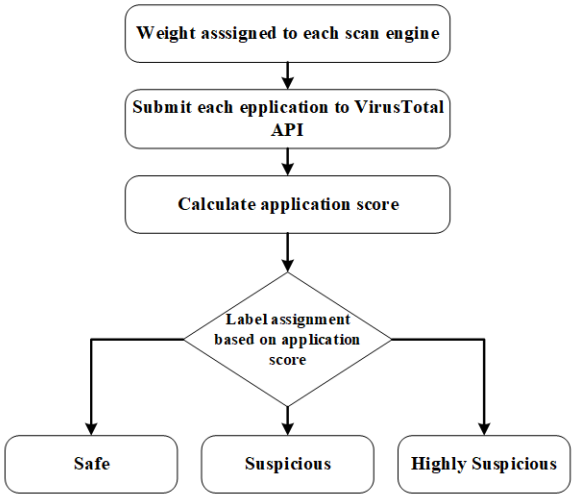

Fig. 4. Methodology of Application Classification into Three Confidence Levels.

B. Feature Extraction using Mobile Security Framework (MobSF)

The interest of the proposed method is to use features to classify applications into confidence level. In order to extract features for all applications, MobSF framework is used. MobSF is an open source mobile testing framework for android, iOS, Windows developed in Python. It is an automated penetration testing framework, capable of detecting the vulnerabilities, which an attacker can exploit in the mobile application. MobSF performs reverse engineering on apk files and extracts data (used as features in our model). MobSF can perform static and dynamic analysis and provides option for scanning bulk applications using command line. The static analyzer can detect insecure permissions.

MobSF stores scanning results in the SQLite database. In order to extract features of android applications a list of

TABLE I
LIST OF SELECTED SCAN ENGINES AND WEIGHT BASED ON THE TRUE POSITIVE AND NEGATIVE RATE.

#	ScanEngine	Weight	True Positive	True Negative	Malware Scanned Out of 3,960	Benign Scanned Out of 1,108
1	ESET-NOD32	98.80	99.00	98.00	3,476	894
2	NANO-Antivirus	98.41	98.00	100	3,473	895
3	SymantecMobileInsight	98.25	99.00	96.00	1,945	647
4	ZoneAlarm	95.48	94.00	100	2,025	661
—	—	—	—	—	—	—
37	TrendMicro-HouseCall	65.00	52.00	95.00	3,426	858
38	Baidu	64.69	56.00	100	2,198	717

TABLE II
CONFIDENCE LEVEL CLASS AND RANGE.

#	Confidence Level Class	Range
1	Safe	$0 \leqslant$ Score $\leqslant 15$
2	Suspicious	15 $<$Score $\leqslant 50$
3	Highly Suspicious	Score $>$50

TABLE III
NUMBER OF APPLICATIONS IN EACH CLASS.

#	Confidence Level Class	Number of Applications
1	Safe	7,197
2	Suspicious	588
3	Highly Suspicious	2,640

recent scans is extracted from MobSF local server (SQLite database) using an in-house developed bot. The results obtained from MobSF is now exploitable as it is, hence a conversion from raw to quantifiable data is required. Therefore, the raw data extracted is transformed to identify the features. For raw data transformation, we identified all the possible values every field can have. Examples of raw and transformed data of permissions, manifest analysis and domains are shown below.

1) Manifest Analysis Feature: There is an xml file named app manifest in every android application in its root directory as per defined by Google. This manifest file contains all the essential application information such as permissions, application activities, services, actions, etc. This manifest analysis is quantified by counting the number of all possible values of manifest analysis which are high, medium and low for each application.

An example of manifest analysis raw data of one application is: ['stat': '**medium**', 'desc': 'This flag allows anyone to backup your application data via adb. It allows users who have enabled USB debugging to copy application data off of the device.', 'title': 'Application Data can be Backed up¡br¿[android:allowBackup=true]', 'stat': '**high**', 'desc': 'An Activity is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device....: '**high**', 'desc': 'An Activity is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device.....']. The transformed and quantified data for manifest analysis is shown in table IV.

TABLE IV
QUANTIFIED DATA AND COUNT FROM MANIFEST ANALYSIS RAW DATA

#	Quantified Data	Total Count
1	Manifest Analysis High	2
2	Manifest Analysis Medium	1
3	Manifest Analysis Low	0

2) Domains Feature: The domain is defined as domain name and ip address and is unique for every device. The domain raw data consists a field named **bad** and the value of field is defined as **no**. An example for one application of domain raw data is:

['u.talkingdata.net': '**bad**': '**no**', 'gaandroid.talkingdata.net': '**bad**': '**no**', '10.10.32.105:9092': '**bad**': '**no**', 'log.intouch.timogroup.com': '**bad**': '**no**', 'log.collection.pplaypro.com': '**bad**': '**no**'].

In order to quantify the raw data, total number of good and bad domains are counted and the result for above mention application is shown in table V.

TABLE V
QUANTIFIED DATA AND COUNT FROM DOMAINS RAW DATA.

#	Domains	Count
1	Good	5
2	Bad	0

3) Permission Feature: A permission is defined as a restriction to protect from the misuse and gives a limited access of the critical data and code. In android the permissions are categorized by Google into four protection levels namely dangerous, normal, signature and signatureOrSystem. These protection levels are examined to improve the accuracy of the model through conducting count permission of known malware and benign applications. For example the permission raw data extracted of one application is:

[u'android.permission.ACCESS_FINE_LOCATION': ['**dangerous**', 'fine (GPS) location', 'Access fine location sources, such as the Global Positioning.....VIBRATE': ['**normal**', 'control.....['**signature**', 'Allows cloud to device messaging', 'Allows the application to receive push notifications.'],.....'android.permission.READ_CONTACTS': ['**dangerous**', 'read contact data'.....].

For quantifying this raw data, the total number of normal, dangerous, signature and signatureOrSystem permissions are counted for every application and the result of above mentioned application is shown in table VI.

TABLE VI
QUNATIFIED DATA AND COUNT FROM PERMISSIONS RAW DATA.

#	Quantified Data	Total Count
1	Normal Permissions	3
2	Dangerous Permissions	10
3	Signature Permissions	1
4	signatureOrSystem Permissions	0

The above explained features are extracted using MobSF. From MobSF the number and type of features acquired are 27 such as permissions, manifest analysis, domains, etc. respectively and are detailed in table VII.

TABLE VII
LIST AND DESCRIPTION OF 27 FEATURES.

#	Feature Name	Description
1	Manifest Analysis High	Counted total number of high manifest analysis used by an each application.
2	Manifest Analysis Medium	Counted total number of medium manifest analysis.
3	Manifest Analysis Low	Counted total number of low manifest analysis.
4	Normal Permission	Counter total number of normal permissions.
5	Dangerous Permission	Counted total number of dangerous permissions.
6	Signature Permission	Counted total number of signature permissions.
7	signatureOrSystem Permission	Counted total number of signatureOrSystem permissions.
8	Cnt_Act	Counted total number of activities performed by each application.
9	Cnt_Pro	Counted total number of providers.
10	Cnt_Ser	Counted total number of services.
11	Cnt_Bro	Counted total number of broadcast receivers.
12	Issued	Certificate status.
13	Native	It verifies if an application is in C/C++.
14	Dynamic	It verifies if an application downloads the executable contents dynamically.
15	Reflect	Reflection provides flexibility to identify API characteristics during run time.
16	Crypto	Cryptographic operations: encryption, key generation and key agreement, etc.
17	Obfus	Obfuscation: to provide the security against reverse engineering.
18	Domains Good	Counted total number of good domains.
19	Domains Bad	Counted total number of bad domains.
20	Dang Info	Code analysis: counted total number of info dang.
21	Dang High	Code analysis: counted total number of high dang.
22	Dang Warning	Code analysis: counted total number of warning dang.
23	Dang Secure	Code analysis: counted total number of secure dang.
24	E_Act	Counted total number of exported activities.
25	E_Ser	Counted total number of exported services.
26	E_Bro	Counted total number of exported broadcast receivers.
27	E_Cnt	Counted total number of exported contents.

4) Permission Feature Analysis: Apart from 27 features extracted from MobSF, we focus on additional features that have high impact to distinguish between benign and malware applications. Therefore, a comparison analysis between malware and benign applications is performed. According to the previous studies [10], [17], permission features are considered to be impacted. Based on all the extracted permissions, there are 141 permissions from four protection levels. Within each level, our aim is to select permissions that are more likely to be used in malware but unlikely to be used in benign applications, and vice versa.

For the selection of permissions, two datasets of known benign and malware are used to compute the frequency of permissions, for example READ_PHONE_STATE permission was found in 93.03% of malware applications and 37.11% of benign applications. The difference between two groups are 55.92%. The percentages of permission counts in malware and benign groups and its difference are shown in Table VIII.

The average of used permissions plus its deviation is used as a threshold for selecting relevant permissions as shown in Equation 3, where $Thres_L$ is a threshold used for the protection level L, Avg_L and Std_L are average and standard deviation of all permissions belonging to the protection level L, respectively.

$$Thres_L = Avg_L + Std_L \tag{3}$$

From four protection levels: normal, dangerous, signature and signatureOrSystem, the obtained value of the threshold for normal, dangerous, signature and signatureOrSystem is 15.24, 20.23, 1.80 and 2.13, respectively. If the threshold value is low, it means either the permissions in these protection levels are rarely used or the frequency of permissions used in malware and benign groups is closed. Since the only permissions which made high impact in differentiating between malware and benign applications are selected, we decided to consider only two protection levels, normal and danger, to select relevant permissions.

To select permissions from normal and danger protection levels, the only permissions are selected which have a differencepermissions which falls under the threshold of the calculated sum which are used as features as shown in the table VIII.

Hence, in this study, a classification model is developed based on 36 features in total which consists of 27 features shown in table VII and nine additional permission features shown in the table VIII.

C. Classification Model

After all applications are labeled, next is to build a classification model. Classification methods require a training phase or may referred as learning phase, where a model is trained from a data set of labeled objects as illustrated in the figure 5. This model can then be applied for predicting class labels on unseen data referred as testing phase. In this paper, three classes are used for developing a classification model to provide the user with a confidence level of how likely applications are malicious rather than providing a binary yes/no answer. Using such model, the user can decide whether or not to proceed with the installation of the applications. For the classification model, we use supervised learning based on the 36 features described previously. Supervised learning is a machine learning task which uses labeled training data that consists of input values and known output values. Figure 6 shows the testing phase where an application with features are fed to the classification model to predict the confidence level of the application as safe, suspicious and highly suspicious.

TABLE VIII
LIST OF SOME OF THE EXTRACTED PERMISSIONS.

#	Permission Name	Protection Level	Malware Count (%)	Benign Count (%)	|Diff| (%)
1	**READ_PHONE_STATE**	**Dangerous**	**93.03**	**37.11**	**55.92**
2	**GET_TASKS**	**Dangerous**	**67.12**	**17.91**	**49.21**
3	**ACCESS_COARSE_LOCATION**	**Dangerous**	**67.94**	**26.57**	**41.37**
4	**SYSTEM_ALERT_WINDOW**	**Dangerous**	**48.67**	**16.14**	**32.53**
5	**GET_ACCOUNTS**	**Normal**	**9.21**	**39.67**	**30.46**
6	**ACCESS_FINE_LOCATION**	**Dangerous**	**57.51**	**28.44**	**29.07**
7	**ACCESS_WIFI_STATE**	**Normal**	**75.53**	**49.90**	**25.63**
8	**WRITE_EXTERNAL_STORAGE**	**Dangerous**	**90.13**	**69.69**	**20.44**
9	**RECEIVE_BOOT_COMPLETED**	**Normal**	**49.69**	**34.15**	**15.54**
10	READ_SYNC_SETTINGS	Normal	0.10	4.33	4.23
—	—	—	—	—	—
140	ACCOUNT_MANAGER	Signature	0.85	8.07	7.22
141	STATUS_BAR	signatureOrSystem	1.22	3.44	2.22

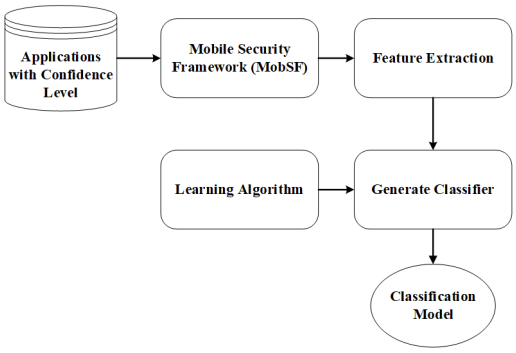

Fig. 5. Classification Model Training.

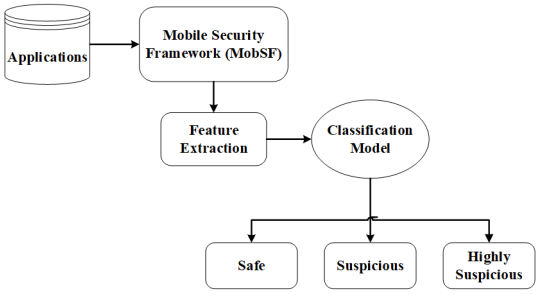

Fig. 6. Classification Model Testing.

IV. EXPERIMENTS AND RESULTS

We conducted two experiments as follows:
Experiment I: We used the dataset of three classes of confidence levels from the benchmark creation as shown in the table III .
Experiment II: We used the dataset containing 3,000 malware and 3,000 benign applications.

For both experiments, we used WEKA to test ten different machine learning techniques using 36 features. The ten classifiers are random forest, random committee, bagging, lmt, random subspace, simple logistic, logistic, classification via regression, kstar and neural network. To evaluate the performance of machine-learning classifiers, k-fold cross validation is used. We performed a k-fold cross validation with k = 10. In this way, our dataset was split 10 times into 10 different sets of training (90% of the dataset were used for training) and testing (10% of the total dataset). We evaluated the following aspects of our method:

i. Accuracy of the classification model.
ii. Influence of different data sizes.
iii. Effectiveness of the extracted features.

A. Experiment I: Confidence Level Classification

This experiment is performed for the classification of android applications in three different classes according to the confidence level. In this experiment, two different balanced applications datasets are used.

Dataset 1: This application dataset contains 900 applications which are 300 applications from each safe, suspicious and highly suspicious class.

Dataset 2: This dataset consists of 1,500 applications that is 500 applications each from safe, suspicious and highly suspicious class.

The both datasets are tested using 10 cross validation. The interest of using different sizes of datasets is to test the dataset size effect on the accuracy. The results in the table IX and X shows that random forest algorithm obtains highest performance in all the other algorithms and as the dataset size increases, the accuracy increases respectively. In the table IX the results for 900 applications are shown. For this dataset Random Forest algorithm obtains highest accuracy of 79.11% in all.

TABLE IX
RESULTS OF DIFFERENT ALGORITHMS WITH 900 APPLICATIONS DATASET FOR CONFIDENCE LEVEL BASED CATEGORIZATION.

#	Algorithm	Accuracy%
1	Random Forest	79.11
2	Random Committee	77.56
3	Bagging	77.33
4	LMT	77.11
5	Random Subspace	76.33
6	Simple Logistic	75.44
7	Logistic	74.44
8	Classification Via Regression	74.33
9	KStar	73.67
10	Neural Network	73.11

In the table X the results of 1,500 applications are shown. The same ten classifiers used above are compared for this dataset and Random Forest attains the highest accuracy of 81.80%. The aim of comparing the results and accuracy of two different size of datasets is to evaluate the effect of the dataset size, to make sure that there is a progressive increase as this will plateau once certain number of applications are submitted.

TABLE X
RESULTS OF ALGORITHMS WITH 1500 APPLICATIONS DATASET FOR CONFIDENCE LEVEL BASED CATEGORIZATION.

#	Algorithm	Accuracy%
1	Random Forest	81.80
2	Random Committee	81.00
3	Random Subspace	78.27
4	Bagging	77.33
5	Classification Via Regression	77.27
6	LMT	77.20
7	Simple Logistic	76.93
8	Logistic	76.27
9	KStar	76.20
10	Neural Network	76.07

The confusion matrix of the Random Forest algorithm for 1,500 applications dataset is depicted in the table XV. This shows that safe class is 89.00% accurate by detecting 445 applications as safe out of 500 safe applications. Hence, the true positive of safe class is 445 and false positive is 31 and 24. Whereas, the suspicious class has gained lowest accuracy of 73.00% and can detect 365 applications as suspicious out of 500. Highly suspicious class has detected 417 applications out of 500 and gained the highest accuracy of 83.40%.

B. Experiment II: Binary Classification

The purpose of this experiment is to test the effectiveness of the features extracted in this study. In this experiment the applications are divided into two categories which are malware and benign. In order to calculate the accuracy of the model, 3,000 applications from benign and 3,000 applications from malware class are used. The accuracy of ten different algorithms is tested and results are shown in table XI. In this experiment the Random forest algorithm has obtained the highest accuracy of 93.63%.

TABLE XI
RESULTS OF DIFFERENT ALGORITHMS WITH 6000 APPLICATIONS DATASET FOR BINARY CATEGORIZATION.

#	Algorithm Used	Accuracy%
1	Random Forest	93.63
2	Random Committee	92.88
3	Random Subspace	91.35
4	Bagging	90.90
5	LMT	90.42
6	Classification Via Regression	90.10
7	Neural Network	89.97
8	KStar	89.85
9	Simple Logistic	88.43
10	Logistic	88.43

C. Experiment III: Principal Component Analysis (PCA)

This experiment is conducted to investigate whether the features extracted inn this study encounter the curse of dimensionality in machine learning. In order to avoid curse of dimensionality dimension reduction is conducted. Curse of dimensionality often occurs when dealing with data in high-dimensional spaces. In our case, the original data consist of 36 features, hence the curse of dimensionality is not encountered. To reduce the dimension experiments are performed on 1,500 applications dataset and explained as below.

PCA is applied with ten principal components and The calculated variance ratio and cumulative sum of various components are detailed in the table XII and shown in figure 7.

TABLE XII
CALCULATED VARIANCE AND CUMULATIVE SUM.

Axis	Variance%	Cumulative%
1	61.45	61.45
2	21.23	82.68
3	9.34	92.02
4	4.71	96.73
5	1.53	98.26
6	0.74	99.00
7	0.23	99.23
8	0.18	99.41
9	0.16	99.57
10	0.09	99.66

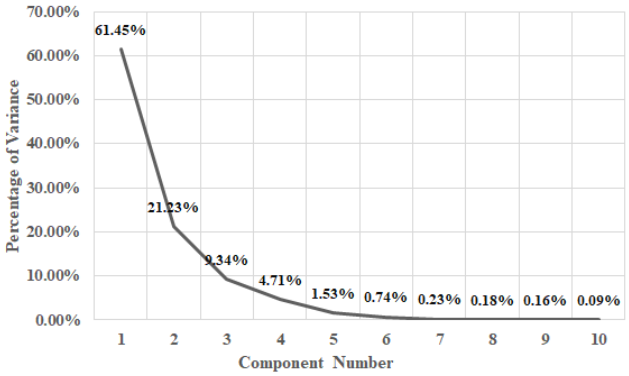

Fig. 7. Graphical Representation of Variance and Cumulative Sum.

From these results it is clearly visible that the cumulative sum after six components is saturated to 99.00%. Hence, the performance of 1,500 applications dataset is tested with six components in which Random Forest has attained highest accuracy of 69.31%. The results of ten tested algorithms are shown in the table XIII. This is concluded from this experiment that dimension reduction is not necessary since it does not improve the overall results and has reduced the accuracy.

TABLE XIII
PCA RESULTS OF TEN ALGORITHMS WITH 1500 APPLICATIONS DATASET.

#	Algorithm Used	Accuracy%
1	Random Forest	69.13
2	Bagging	69.07
3	Random Committee	67.87
4	KStar	67.13
5	Random Subspace	66.80
6	LMT	66.20
7	Classification Via Regression	66.13
8	Neural Network	63.13
9	Logistic	62.60
10	Simple Logistic	62.33

Furthermore, the confusion matrix of the Random Forest algorithm for 1,500 applications dataset for both methods: confidence level classification and PCA is compared and shown in the table XIV. The results show that the performance of all the classes that is safe, suspicious and highly suspicious in case of PCA is much lower than the performance of confidence level classification.

TABLE XIV
CONFUSION MATRIX OF RANDOM FOREST ALGORITHM FOR 1,500 APPLICATIONS DATASET.

Class	Accuracy%	
	Confidence Level Classification	PCA
	(36 Features)	(6 Components)
Safe	89.00	66.00
Suspicious	73.00	61.80
Highly Suspicious	83.40	70.80

V. CONCLUSION AND FUTURE WORK

In this paper, an android malware detection and classification model is developed which uses 36 informative features to categorize malicious applications based on the confidence level of application. The confidence level is categorized into three classes: safe, suspicious and highly suspicious. Static analysis is performed on large scale labeled dataset of 13,850 android applications. This dataset contains known malware, benign applications and random applications downloaded from apk-dl.com. Ten machine learning classification algorithms are tested to determine the most highly performing one in terms of accuracy and speed. The experimental evaluations show that this model is capable of determining the confidence level of android applications with an accuracy of 81.80% using the Random Forest algorithm.

As a future work, first, an online platform can be implemented to identify the confidence level of android application. This online platform can provide a better understanding to the user weather to download the application or not based on the confidence level. Second, a method combining static and dynamic analysis can be implemented for malware detection. Features such as behavior based: memory consumption, CPU usage, network usage, system calls, etc. can be extracted. Third is to increase the dataset both in terms of malware and benign applications so to determine how big the dataset needs to be in order to obtain the best results in terms of high accuracy.

ACKNOWLEDGMENT

This work is supported in part by the National Broadcasting and Telecommunication Commission of Thailand (NBTC) Grant-in-Aid for Scientific Research No. T2-1-0004/57. Authors would like to thank Bangkok University, Thailand for providing scholarship for this program. Authors would like to thank Mr. Hossein Fereidooni and AndroZoo for sharing malware samples with us and VirusTotal for sharing API key for scanning purpose.

REFERENCES

[1] "International data corporation," https://www.idc.com/promo/smartphone-market-share/os, accessed: 2017-August-01.

[2] P. Faruki, A. Bharmal, V. Laxmi, V. Ganmoor, M. S. Gaur, M. Conti, and M. Rajarajan, "Android security: a survey of issues, malware penetration, and defenses," *IEEE communications surveys & tutorials*, vol. 17, no. 2, pp. 998–1022, 2015.

[3] L. Er-Rajy and M. A. El Kiram, "How far android is secure?" in *Complex Systems (WCCS), 2015 Third World Conference on*. IEEE, 2015, pp. 1–6.

[4] E. Fernandes, B. Crispo, and M. Conti, "Fm 99.9, radio virus: Exploiting fm radio broadcasts for malware deployment," *IEEE Transactions on Information Forensics and Security*, vol. 8, no. 6, pp. 1027–1037, 2013.

[5] M. Eslahi, R. Salleh, and N. B. Anuar, "Mobots: A new generation of botnets on mobile devices and networks," in *Computer Applications and Industrial Electronics (ISCAIE), 2012 IEEE Symposium on*. IEEE, 2012, pp. 262–266.

[6] N. Etaher, G. R. Weir, and M. Alazab, "From zeus to zitmo: Trends in banking malware," in *Trustcom/BigDataSE/ISPA, 2015 IEEE*, vol. 1. IEEE, 2015, pp. 1386–1391.

[7] Zheng, S. Min, C. S. L. Mingshen, Bruno, and John, "Droidanalytics: A signature based analytic system to collect, extract, analyze and associate android malware," *12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, pp. 163–171, 2013.

[8] Bläsing, B. Thomas, S. Leonid, A. Aubrey-Derrick, Camtepe an Seyit Ahmet, and Sahin, "An android application sandbox system for suspicious software detection," *5th International Conference on Malicious and Unwanted Software*, pp. 55–62, 2010.

[9] Zhao, Z. Min, G. Tao, Y. Fangbin, and Zhijian, "Robotdroid: A lightweight malware detection framework on smartphones," *Journal of Networks*, vol. 7, no. 4, pp. 715–722, 2012.

[10] F. Ghaffari, M. Abadi, and A. Tajoddin, "Amd-ec: Anomaly-based android malware detection using ensemble classifiers," in *Electrical Engineering (ICEE), 2017 Iranian Conference on*. IEEE, 2017, pp. 2247–2252.

[11] 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.

[12] "Mobile security framework-mobsf," https://hub.docker.com/r/opensecurity/mobile-security-framework-mobsf/., accessed: 2016-08-01.

[13] "Open security mobile security framework," https://hub.docker.com/r/opensecurity/mobile-security-framework-mobsf/, accessed: 2017-02-01.

[14] "Virustotal-free online virus, malware and url scanner," https://www.virustotal.com/en, accessed: 2017-05-01.

[15] "Symantec," https://www.symantec.com/content/dam/symantec/docs/reports/gistr22-government-report.pdf, accessed: 2017-09-01.

[16] J. Kuriakose and P. Vinod, "Unknown metamorphic malware detection: Modelling with fewer relevant features and robust feature selection techniques," *IAENG International Journal of Computer Science*, vol. 42, no. 2, pp. 139–151, 2015.

[17] O. S. Adebayo and N. A. Aziz, "Static code analysis of permission-based features for android malware classification using apriori algorithm with particle swarm optimization." *Journal of Information Assurance & Security*, vol. 10, no. 4, pp. 152–163, 2015.

[18] H. Fereidooni, M. Conti, D. Yao, and A. Sperduti, "Anastasia: Android malware detection using static analysis of applications," in *New Technologies, Mobility and Security (NTMS), 2016 8th IFIP International Conference on*. IEEE, 2016, pp. 1–5.

[19] K. Zhao, D. Zhang, X. Su, and W. Li, "Fest: A feature extraction and selection tool for android malware detection," in *Computers and Communication (ISCC), 2015 IEEE Symposium on*. IEEE, 2015, pp. 714–720.

[20] A. Martín, A. Calleja, H. D. Menéndez, J. Tapiador, and D. Camacho, "Adroit: Android malware detection using meta-information," in *Computational Intelligence (SSCI), 2016 IEEE Symposium Series on*. IEEE, 2016, pp. 1–8.

[21] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, "Crowdroid: behavior-based malware detection system for android," in *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*. ACM, 2011, pp. 15–26.

[22] M. Spreitzenbarth, F. Freiling, F. Echtler, T. Schreck, and J. Hoffmann, "Mobile-sandbox: having a deeper look into android applications," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*. ACM, 2013, pp. 1808–1815.

[23] B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, J. Nieves, P. G. Bringas, and G. Álvarez Marañón, "Mama: manifest analysis for malware detection in android," *Cybernetics and Systems*, vol. 44, no. 6-7, pp. 469–488, 2013.

[24] K. Xu, Y. Li, and R. H. Deng, "Iccdetector: Icc-based malware detection on android," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 6, pp. 1252–1264, 2016.

[25] Y. Otsuki, E. Takimoto, T. Kashiyama, S. Saito, E. W. Cooper, and K. Mouri, "Tracing malicious injected threads using alkanet malware analyzer," pp. 283–299, 2014.

[26] Wu, M. Dong-Jie, L. Ching-Hao, W. Hahn-Ming, and Kuo-Ping, "Droidmat: Android malware detection through manifest and api calls tracing," in *Information Security (Asia JCIS), 2012 Seventh Asia Joint Conference*. IEEE, 2012, pp. 62–69.

[27] K. Shabtai·and Asaf, E. Uri, G. Yuval, W. Chanan, and Yael, ""andromaly": a behavioral malware detection framework for android devices," *Journal of Intelligent Information Systems*, vol. 38, pp. 161–190, 2012.

TABLE XV
CONFUSION MATRIX OF RANDOM FOREST ALGORITHM FOR 1,500 APPLICATIONS DATASET.

		Predicted class			**Total of ground truth applications (%) in class**
		Safe(%)	Suspicious(%)	Highly Suspicious(%)	
Ground Truth	Safe(%)	445(29.70)	31(2.07)	24(1.60)	33.33
	Suspicious(%)	93(6.20)	365(24.33)	42(2.80)	33.33
	Higly Suspicious(%)	17(1.13)	66(4.40)	417(27.80)	33.33
Total of predicted applications (%) in class		37.00	30.80	32.20	100

[28] Bhatia, K. Taniya, and Rishabh, "Malware detection in android based on dynamic analysis," *International Conference on Cyber Security And Protection Of Digital Services (Cyber Security)*, pp. 1–6, 2017.
[29] M. M. Saudi, N. F. Mohd, and N. Basir, "A new mobile malware classification for call log exploitation," *System*, p. 6, 2011.
[30] I. N. Ahmad, F. Ridzuan, M. M. Saudi, S. A. Pitchay, N. Basir, and N. Nabila, "Android mobile malware classification using tokenization approach based on system call sequence," in *Proceedings of the World Congress on Engineering and Computer Science*, vol. 1, 2017.
[31] F. Tong and Z. Yan, "A hybrid approach of mobile malware detection in android," *Journal of Parallel and Distributed Computing*, vol. 103, pp. 22–31, 2017.
[32] "Apk-dl," http://apk-dl.com/, accessed: 2017-02-01.
[33] "Androzoo," https://androzoo.uni.lu/ap, accessed: 2017-08-01.