An Intelligent Methodology for Malware Detection in Android Smartphones Based Static Analysis

Ahmed H. Mostafa, Marwa M. A. Elfattah and Aliaa A. A. Youssif

***Abstract*—** Recently**,** a lot of mobile phone users are rapidly switching to smartphones, and, many users download mobile applications without any thought of security. Therefore, smartphones are interesting target for malware, especially with Android devices. So, it is too important to use a methodology to detect the malware applications before installing it on the phones.

In this paper we propose an effective methodology to detect Android malware using static code analysis based models. The proposed models are built to extract features relevant to malware based on extracted permissions from AndroidManifest.xml file as well as extracted methods and APIs from disassembled code to be used as features for training machine learning classifiers.

***Keywords*—** Android, Malware Detection, Machine learning, Smartphones, Pattern Recognition.

I. INTRODUCTION

Malware is an abbreviation for two words malicious and software. Actually, it is software that included in the computer system for malicious purposes, without any knowledge from the computer owner. It may be used to collect important information, or gain access to computer systems. The seriousness of malicious software ranges from hurt the users with annoying Ads to steal important data such as credit card numbers and format the phone's memory.

Smartphones have become pervasive due to the availability of office applications, Internet, games, vehicle guidance using location-based services apart from conventional services like voice calls, SMS and multimedia services. [2]

A lot of mobile phone users are rapidly switching to smartphones. According to eMarketer [3], it is expected that around 49% of the mobile phone users globally are likely to use smartphones by 2017.

Many users download mobile applications without any thought of security. Whereas, with exponentially increasing in downloading mobile application, according to PortioResearch [4] downloading of mobile applications will continue to grow to exceed 200 billion applications by the end of year 2017. The number of markets which allow users to download applications are increasing and the number of non-official markets are also increasing, but non-official market do not impose security measures on the phone applications that are being uploaded by developers so many hackers upload malicious applications to these markets. Therefore, it is important to develop a methodology to detect the malware applications before installing it on the phone.

Actually, most current existing commercial anti-malware solutions employ signature based detection due to its implementation simplicity, but the major drawback of these techniques is that they cannot detect new malwares [2].

On other hand, the other set of techniques depend on monitoring the behavior of malware during the run time but monitoring can be a very heavy consuming task also the malware is detected after it was installed . [2], [5].

The most common mobile operating systems are Android, Blackberry, iOS, Windows Phone and Symbian.

Statista [6] expected that Android is expected to account for 62.4 percent of global tablet shipments in 2017, thus taking over as the market leader. Statista also expected that the smartphones deploying Android as operating system are forecast to reach around 1.5 billion units by 2018 [7]. Cisco security report for 2014 finds 99% of all new mobile malware is targeting Android [8]. Android's Google Play store has officially reached over 1 million applications, and applications download have also grown to over 50 billion [9]. Several third-party Android Marketplaces exist without restricted security rules for submit applications.

In this paper we extend our previous work in [1] by proposed an effective approach of detecting malwares in android smartphones before installing it, based on static code analysis. It takes into account various features based on permissions declared in AndroidManifest.xml file as well as extracted methods and APIs from disassembled code from dex file to be used as features for training machine learning classifiers.

The rest of paper is structured in the following way: We start in section II with a brief background on malware detection techniques, in section III a survey of previous relevant studies, in section IV the architecture of android application, section V describes the methods used to collect

This work was supported by Computer Science Department, Faculty of computers and information, Helwan University.

Ahmed Hesham Mostafa, Computer science Department, Faculty of computer and information, Helwan university, Cairo, Egypt (phone: +2001095906541; e-mail: ahmed.hisham@fci.helwan.edu.eg).

Dr. Marwa M. A. Elfattah, Computer science Department, Faculty of computer and information, Helwan university, Cairo, Egypt (e-mail: marwa26880@gmail.com).

Prof. Aliaa Youssif , Computer science Department, Faculty of computer and information ,Helwan University ,Cairo , Egypt (Phone: +202 27644827 ; Fax : +202 25547975 ;e-mail: aliaay@yahoo.com)

data , extract features and building the dataset, in sections VI we describe the extracted permissions ,methods and APIs , section VII presents the experiments setup and the evaluation measures. In section VIII discuss the classification results in details and finally in section IX the conclusion.

II. Malware Detection Background

Mainly, there are two categories of smartphones malware detection techniques, which are detection techniques based on static analysis and detection techniques based on dynamic analysis [10]-[13]. The major difference between static and dynamic analysis is how the data is acquired.

Static detection represents an approach of checking source code or compiled code of applications before it gets executed. It identifies malicious code by unpacking and disassembling the application to extract some features. There exists several works that apply static analysis such as [14]-[20].

On other hand, dynamic set of techniques identify malicious behaviors after executing the application on an emulator or controlled environment. Monitoring can be performed in cloud systems. But, it is dependent on external server, which means there can be server down problems and network congestion. Also, there exists several works that apply dynamic analysis such as [21]-[25].

Obviously, static based techniques are fast, flexible and easy to be automated, which means, they are suitable for mobile devices. Whereas, in dynamic based analysis the monitoring can be a very heavy consuming task. Also, in dynamic based analysis, the malware can change his behavior during the run time and cannot be detected [10].

From other point of view, there are two main different malware identification techniques which are anomaly-based identification techniques and signature based identification techniques. [10]

Anomaly-based identification attempts to model normal and non-normal behaviors during a training phase. Anomaly detection techniques have the potential to detect newfangled malware. However, they are prone to detect rare legitimate behaviors as malicious [12].

Signature-based identification aims at identifying known malicious by means of predefined patterns of signatures. The main benefit of signature detection lies in its accuracy detecting well-known attacks [2], [10].

Static signature-based technique is very efficient and reliable to identify known malwares; otherwise, they cannot detect unknown malwares. Also, Signatures must be up-to-date that lead to a massive amount of signatures. On other hand, anomaly based techniques have the ability to detect unknown malwares [10]-[13].

III. Related work

Several studies have been done in the field of Android malware detection. As mentioned early, the main malware detection techniques are dynamic and static techniques.

For dynamic techniques, the Crowdroid [22] and MADAM [23] propose android malware detection by monitoring the malware behavior through analyzing the application system call. Adas, et al [26] proposed methods to extract network and URL inspection feature and these methods based on cloud computing platform,. Also, Marengereke, et al [27] is a cloud based. Alterdroid [21] is a framework for malware dynamic analysis, based on the notion of differential fault analysis. . On other hand, for static analysis techniques, Sanz, Borja, et al[17],[20] proposed a new method that, based on several features that are extracted from the Android manifest file of the legitimate applications. Also, Yerima [18] proposed approach based on Bayesian classification models obtained from static code analysis to detect android malware using malware dataset from 49 android families .Gunjan Kapse [14] proposed a method of detection of malware on Android based on application features which make use of Permissions, API calls, and Intent filters. Ming-Yang Su [15]proposed to use the permission combinations of the application detect a malware Sato, et al [16] proposed a method for Android malware by using only manifest files to detect malware and the proposed method extracts six types of information from manifest files and uses them to detect Android malware.

IV. Android Application

Android [28] is an open source OS built on Linux for mobile devices.

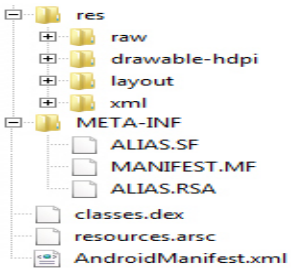

Fig. 1 APK Architecture

Android applications are developed with Google Android SDK [29] and written in Java language. The Android application source code is compiled into .dex file. The dex file, shared libraries and any other resources, including the AndroidManifest.xml file that describes the App, are packaged together into an APK (Android application package).

As shown in Fig. 1 the APK consist of folder res stores icons, images, string/numeric/color constants, UI layouts, menus, animations compiled into the binary. Folder assets contains non-compiled resources META-INF stores the signature of the app developer certificate to verify the third party developer identity.

AndroidManifest.xml stores the meta-data such as package name, permissions, definitions of one or more components like Activities, Services, Broadcast Receivers or Content Providers, minimum and maximum version support... etc

V. Feature Extraction

This section mainly aims to describe the collecting and building the dataset and the extraction process for features

from android application based on dimensionality reduction technique that extracts a subset of new features from the original set of features by means of some functional mapping keeping as much information in the data as possible.

A. Collect Data

To conduct experiments, a dataset of real Android applications and real malware in considered. In particular, an initial dataset of 650 applications divided to 325 malware and 325 benign android applications is acquired. The malwares are collected from Contagio Malware Dump [30] Android Malware Dump [31] and MalShare [32].

Malware applications represent more than 89 android malware families [33] - [35] and it is listed in Table I. A malware family is basically a collection of malware presenting similar behaviour. Whereas, the benign applications cover all android categories in Google play store [36].

Table I - Malware Families

Malware Family	#	Malware Family	#
Airpush,StopSMS,Minimob	13	NotCompatible	1
CI4 SMS Bot	1	PJAPPS	15
Opfake aka	3	Plankton	26
Stiniter TGLoader	1	Repane	1
AVPass	1	Roidsec Sinpon	1
Backflash Crosate	1	Scavir	1
BadNews	1	Scipiex	1
BaseBridge	4	Simhosy	3
Beita	1	Skullkey	1
Carberp	2	SMSilence	2
Ccounterclank	6	SPPush	1
DougaLeaker(Dougalek)	6	Ssucl	2
DroidKungFu	4	SteeK Fatakr, fakelottery	19
Dropdialerab	2	Tascudap	1
Extension	2	Tigerbot-Spyera	1
fakeAV	9	Uracto	3
Fakebank	4	UsbCleaver	2
Fakedaum	1	Uten	1
Fakedefender	1	VDLOADER	2
FAKEINST	6	YZHC	2
FakeJobOffer	1	Zertsecurity	1
FakeMarket	1	Zitmo2012	7
FakeMart	3	Lotoor	2
Fakenotify	1	Boxer	1
Fakeplay	1	FakeTimer	2
FakeRegSMS	1	Tetus	4
Faketaobao	1	Android iBanking	3
FakeToken	1	Foncy	1
Fakeupdate	1	Adsms	1
FakeVertu	2	Arspam	1
finfisher_finspy	1	Cosha	1
GAMEX	1	DroidDream	1
GEINIMI	38	FakeAngry	1
Godwon	3	Fjcon	1
GoldDream	2	GGTracker	2
Jollyserv	1	GingerBreak	1
KMIN	47	HippoSMS	1
LeNa	4	Lien	1
Loozfon	2	MMarketPay	1
LUCKYCAT	1	Qicsomos	1
Malap	1	SMSspy	1
Moghava	1	Spitmo	1
MouaBad	1	Tapsnake	1
Nickyspy	1	Spayoo	1
Unclassified	15		

B. Features extraction methodology

We focus on extracting features from manifest.xml file and the disassembled dex code of the application which are extracted from the application APK file. Those features are extracted statically using python [37] script that is developed based on AndroGuard APIs [38] and wxpython [39]. A python script is developed to automate the extraction of the features for data analyzing. The script is used to extract features from all APK files at once without needing to analysis each apk file alone. The developed script filters permissions, strings, and APIs, methods and classes paths. All extracted features are represented as sets of strings, then, they are saved in list format or binary vector format to have a binary matrix and all data saved in CSV files.

The developed script unpacks the APK files to classes.dex, and the manifest file to binary format. Then, it converts the manifest to xml file, where, all permissions used by the application can be extracted, and, the dex file is disassembled to extract the methods names and the path of APIs including packages and classes. All features from manifest files are extracted based on the following generic methodology:

Vector V contains all features (android system permissions or methods, APIs or combination of both). For each application there is features vector V_i contains all features for such application, where, the feature vector represents all features. So, for each application a_i in the Applications set A there is binary vector $V_i = \{v_1, v_2, v_3, ..., v_n\}$ where, n is number of features, and,

$$v_i = \begin{cases} 1, & \textit{if exxtracted feature } v_n \textit{ exist in } V \\ 0, & \textit{else} \end{cases}$$

- The variable C is the type of the applications to be benign or malware where $C \in \{Malware, Benign\}$
- The creating of matrix M process is described by following algorithm :

Input: *set A contain all apk files and vector V contain all features (android system permissions or methods API or Both)*
Output: *matrix M contain all vectors* V_i

for each a_i *in* A ***do***
  Extract all features from a_i *and set it to set* S_i
  for each $s_j \in S_i$ ***do***
    if $s_j \in V$ ***do***
      $v_n \in V_i = 1$
    else
      $v_n \in V_i = 0$
    end if
  end for
  Set V_i *in* M
end for

C. Choosing informative features

A key idea behind selecting the most informative features is to find a minimum set of features such that the resulting probability distribution of the data classes is as close as possible to the original distribution obtained using all features and removing as much irrelevant and redundant information as possible.

Using the reduced set of features has additional benefits. It reduces the number of features appearing in the discovered patterns, which helps in making the patterns easier to be understood. Further it enhances the classification accuracy and learning runtime.

In the conducted experiments, the previously mentioned methodology for feature extraction is applied to produce a matrix M, which contains the binary vectors of the features from all collected applications.

Then, to select the most informative feature set, the feature selection GainRatio are used to select the most informative features then rank the features from highest to lowest feature using ranking algorithm then drop or remove all features have zero score to have finally an ordered set of features.

The Gain Ratio score is calculated for feature vectors based on the following formula [40], [41]:

$$GainRatio\ (C, v_n) = \frac{(H(C) - H(C \mid v_n))}{H(v_n)} \quad (1)$$

Where H is the information entropy, Y and X are random variables and P is the probability.

$$H(X) = \sum_i P(x_i) I(x_i) = -\sum_i P(x_i) \log_b P(x_i) \quad (2)$$

Where the conditional entropy of two events X and Y

$$H(X \mid Y) = \sum_{i,j} P(x_i, y_j) \log \frac{P(y_j)}{P(x_i, y_j)} \quad (3)$$

VI. Features

To classify smartphone applications as malicious or benign, it is widely required for each application to extract its features which help in identifying the application as malware or benign application. Those extracted features are required to be informative to produce an accurate decision.

In this section, we describe the features which are proposed to be extracted by applying the discussed features extraction and filtering methodology.

A. Android Permission

Android permissions control the access to sensitive resources and functionalities. Permissions allow an application to access potentially dangerous API functionality. Many applications require several permissions to do its function properly. Also, android systems allow developer to create his own permissions that allow other application to access some activities without developed it again. All these permissions whether android system permission and user permission must be listed explicitly in the application's Manifest.xml file, and each application must have an android Manifest.xml in its root directory [42], [43].

For example, applications that want to connect to internet need to use the permission INTERNET, to call phone CALL_PHONE permission is used, to send SMS message SEND_SMS permission is used, to access camera CAMERA permission is used and so on.

Using the permissions as features for machine learning classifiers can help to detect the malware before the installation. So, analyzing the android applications manifest files to identify the permission set requested by that application can considered as an informative methodology for anomaly based feature extraction in static manner.

First, we extracted all permissions from all of the collected dataset whether android system permissions or permissions developed by developer of applications, we noted, that the benign application use 1141 permissions and malware application use 4882 permissions. Approximately, malware applications use nearly three times permissions more than benign applications, that means malwares actually use permissions to access functions the benign applications not use.

We focused on the android system permissions ,so we found that android 5.0 Lollipop with API level-21 [43],[44] provide 151 android system permissions , according to considering all of android system permissions as a feature set will produce an enormous feature vector for each application. So it is required to reduce the number of the selected features.

For reducing the feature set, a preprocessing step has been performed, which is removing all zero-frequency-permissions in the binary matrix M . The permissions that its frequency is zero are those which are not used by any malware or benign applications, the number of features were reduced to 114 features.

Then, we calculated GainRatio score for each permission. All permissions that its score is zero are removed. A set of ordered features (permissions) from highest to lowest gainratio score have been produced as shown Fig. 2.

B. Methods and APIs

API refer to application programming interface, it consists of packages, classes and methods to help developer to develop applications. Android provides developer with APIs to develop android applications, and developer also can use externals APIs such as others Java and Jason APIs.

Android applications are developed in Java and compiled into optimized bytecode for the Dalvik virtual machine. This bytecode can be disassembled and provides information about packages, classes, methods, parameters and data used in the application.

As mentioned before we developed python script based on Androguard APIs to extract information from dex files, the script extracts both the method name and its class path. The extracted methods and its class path are concatenated in one string separated by semi coma, as, for example:

"value;Landroid/annotation/SuppressLint;"

Where "value"is the method name or API Call and

"Landroid/annotation/SuppressLint" is the class path and "SuppressLint" is the class name where the "SuppressLint" belongs to the package android.annotations.

We extracted all the methods names and its associated class paths from all malware and we found that the number of extracted methods and its class paths are more than 911967 string. Actually, there are a lot of repetitions. So, we calculated the frequency for each method associate with its class path. The number of features has been more than 16356 methods and their class paths. We selected the top 500 features with top frequency.

The feature extraction methodology described before is applied for each application to build the binary matrix M of feature vectors. Then, the gainratio for each feature is calculated, and, the features with zero weight are removed to have only 346 features ranked from highest to lowest from 16356 features.

We noted that the external APIs have more informative values than the Android APIs. Therefore in this study we focused in the external APIs more than Android APIs - Fig. 3 show only 30 ranked features.

C. Combination of features

We combined the reduced permissions and the APIs methods features, and calculated the gainratio for each feature then ranking the gainratio scores from the highest to lowest to have an ordered list consist of 404 features based on combination of permissions, methods and its class paths.

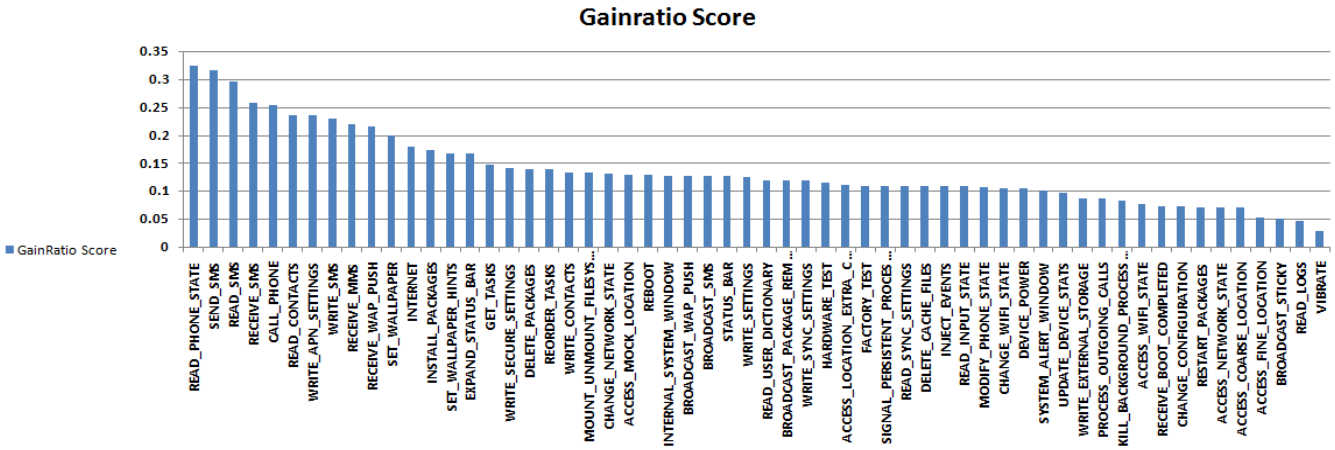

Fig.2 Top 58 ranked permissions

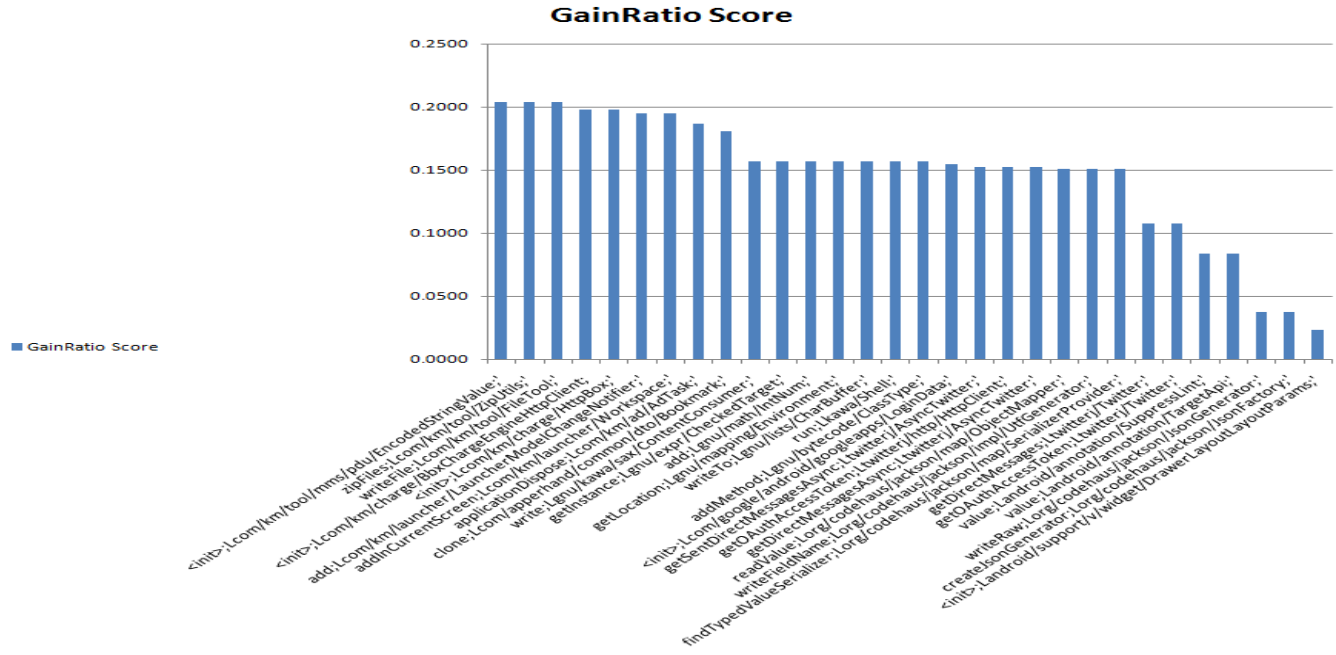

Fig.3 30 ranked methods and APIs

VII. Experiments Setup

In our experiments, the different feature sets obtained from previously mentioned feature extraction techniques are categorized using more than one classifier from WEKA tool [45].

A. Classifiers

In the conducted experiments, thirteen classifiers from WEKA have been used to choose the best features set. Where, some of classifiers were tree based classifiers, such as, C4.5 algorithm (J48) ExtraTree , J48Consolidated(J48C) , RandomForest Tree(RFT), RandomTree (RT) and best-first decision tree (BFT), some of classifiers were mathematical based classifiers, such as feed forward neural network (MultilayerPerceptron MLP) with .001 learning rate , Support vector machine (SVM) , Radial Basis Function Network (RBF) , stochastic gradient descent (SGD),Logistic Regression (Log), and some of classifiers were lazy based classifiers, such as, K- nearest neighbours classifier *(*IBk) and KStar .

B. Learning and Testing Options

WEKA has different mechanisms to divide the experimental dataset into training dataset and testing dataset testing that is used to train and test the classifiers models. The first methodology is k-cross validation [46]. In k-fold cross-validation, the dataset is randomly partitioned into k equal size subsamples. One subsample is used as the validation data for testing the model, and the remaining k-1 subsamples are used as training data. Then repeated k times, with each of the k subsamples used exactly once as the validation data. The k results from the folds can then be averaged to produce a single estimation. In the conducted experiments, two k values have been chosen, k=10 and k=3 folds.

Another mechanism is simply to divide dataset into two portions in a random manner. Here, 66% of the original dataset are randomly chosen for training, and the remaining 34% of the data are used for testing, the last testing option is using same data for training and this option useful to give us an intuition on how the selected features are good to classify the selected data but it does not have any indication of how the classification model will perform on new data.

This study will choose the best features sets combined with machine learning classifiers based on the results of testing options for the classification models based on this order of the testing options first testing option with 10 folds will be the main testing option to distinguish between classification models results, then 3 fold, then split data to 66% and 34 % and in the last option is using same data for training and testing

C. Measure of classifiers

The evaluation was performed by following measures [46]:

$$Accuracy = \frac{(TP + TN)}{(TP + TN + FP + FN)} \quad (4)$$

The *Accuracy* is the percentage of predictions that is correct, where TN is the number of benign applications correctly classified, TP is the number of malware cases correctly classified, FP is the number of benign applications incorrectly detected as malware, and FN is the number of malware incorrectly classified as benign applications .

$$TPRate = \frac{TP}{(TP + FN)} \quad (5)$$

TP Rate The percentage of positive labeled instances that were predicted as positive.

$$FP\ Rate = \frac{FP}{N} \quad (6)$$

FP Rate is the percentage of positives cases that were incorrectly classified as negative, and where the N is the number of all benign.

ROC curve is obtained by plotting the TP Rate (TPR) against the FP Rate (FPR). An ROC curve plots the TPR against FPR for every possible detection cut-off. The total area under the ROC curve (AUC) indicates the classifier's predictive power. If the AUC value is 1, that implies perfect classification. If the AUC value is close to 1, that denotes good classier predictive power [47].

D. Extraction processing Time

To analyze the time for extracting different set of features, the experiments for extracting and building the dataset were performed on a desktop computer with Intel core i5 and 4GB of RAM, The operating system of this machine is Microsoft Windows 7 and the experiment is running on Ubuntu-Linux installed on virtual machine Oracle VM VirtualBox.

Table II represents the average extraction time for the features of the three feature models for, where, Model 1 represents the features set based on permissions, Model 2 represents the features based on methods and APIs and the Model 3 represents the features set based on the combination of features.

It is widely noted from table II that the average extraction time for the models, which is based on API methods, is larger than the model which is based on permissions features only.

Table II Features extraction time

	Model 1	**Model 2**	**Model 3**
Type of features	Permissions	API methods	Combination
Features number	58	346	404
Extraction time for 325 malware	10.37 sec	972.16 sec	978.47 sec
Average	0.0319 sec	2.9912 sec	3.0106 sec
extraction time for 325 benign	9.24 sec	2100.23 sec	2115.26 sec
Average	0.02843 sec	6.4622 sec	6.5084 sec

Also, as noted, the extraction time depend on the amount of data extracted from the Mainfeast.xml file and the dissemble dex files, where, the number of permission extracted from Mainfeat.xml files for benign is 1141 permission and for malware is 4882 permission, while the number of extracted methods and its class paths as one sting from malware is 911967 string, and for benign is 1185635 string.

VIII. Experiments Results

In this section, we show the classification accuracy results in details for each of the three previously mentioned models.

A. Reduced Permission

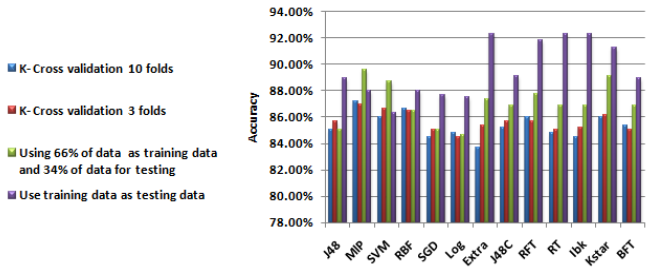

Fig. 4 Accuracy for features based permissions

From Fig. 4, we can note that the best accuracy results for reduced permissions as features combined with machine learning classifiers are 87.2308% with testing option 10 folds using MLP as classifier ,with 3 folds is 86.9231% ,and using 34%of data for testing is 89.5928% also using MLP as classifier. While, with training data as testing data is 92.3077% using classifiers RT, Extra tree and IBk, but the testing using same training data give us how this features are good for this data but not give any induction about future or new data.

As mentioned earlier, an AUC value closer to 1 denotes better classifier predictive power and it can be observed from Fig. 5 AUC for reduced permission combined with MLP classifier using 10 folds as testing is 0.9143 of area under the ROC curve.

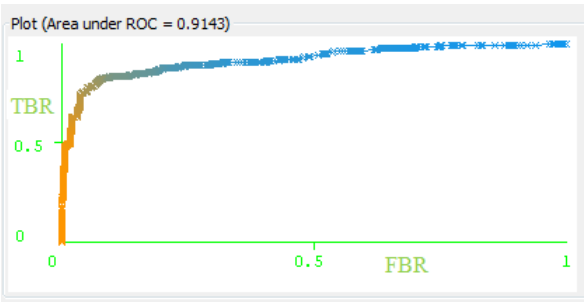

Fig. 5 Roc Curve –reduced permission 10 folds

so we can conclude that the best accuracy result obtained by using reduced permissions as features is 87.2308% and with weighted averages for FP Rate and TP Rate are 0.128, 0.872 respectively

B. Methods and APIs

From Fig. 6, we can note that, the best accuracy results for Methods and APIs as features combined with machine learning classifiers are 74.9231 % for testing option 10 folds using RFT classifier, and with testing option 3 folds is 74.9231 % using RFT and Log classifiers, and with testing with 34% of data is 76.9231 % with all classifiers expect RBF and RFT are 76.4706 % and with testing using same data is 74.9231 with all classifiers expect the SVM.

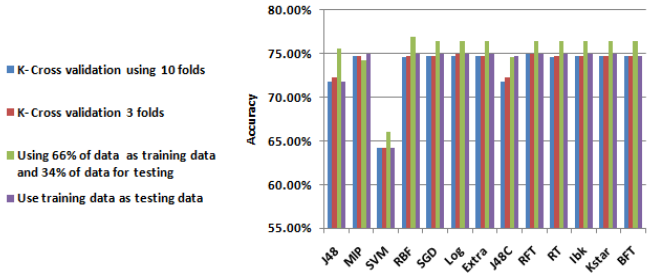

Fig. 6 Accuracy for features based methods and APIs

It can be observed from the Fig. 7 that represents the ROC curve for Methods and APIs as features combined with RFT using 10 folds that the curve is going away from the left-hand border that represent the true positive rate also, also the AUC is 0.784 of area under the ROC curve, so we can conclude that the features based methods an APIs only give low accuracy and high false positive rate.

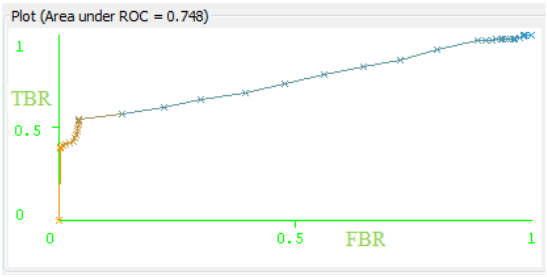

Fig. 7 ROC Curve –methods, APIs 10 folds

So from comparing the results, we can conclude that the best accuracy based on methods and APIs as features is 74.9231 % and with weighted averages for FP Rate, and TP Rate 0.251, 0.749 respectively

C. Combination between features

We can note from Fig. 8 that , the best accuracy results for combination between features reduced permissions methods and APIs combined with machine learning classifiers are for testing option 10 folds is 91.5385 % using RBF classifier ,and with 3 folds is 91.2308 %% with RBF , KStar and BFT classifiers ,and with testing with 34% of data the best accuracy result is 92.3077 % with RBF and KStar while with testing using same data, the best accuracy result is 96.7692 % with RT and IBK classifiers , from comparing the results we can note that the best accuracy is 96.7692 % but the testing using same training data give us how this features are good for this data but not give any induction about new data , also we can note that the combined features combined with RBF classifier using 10 folds , 3folds and 34% testing option give higher accuracy results than using combination features with others classifiers especially RT and IBK classifiers.

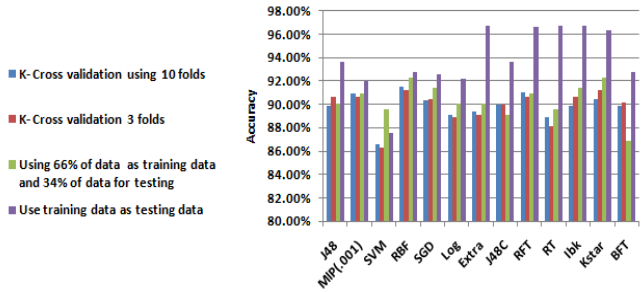

Fig. 8 Accuracy for combination features

.

Also the ROC curve in Fig. 9 show that the area under the curve is 0.953 for the combination features with RBF classifier using 10 folds ,we can note form the Fig.9 that the curve follows and closer to the left-hand border and then the top border of the ROC space, that mean more accurate the test.

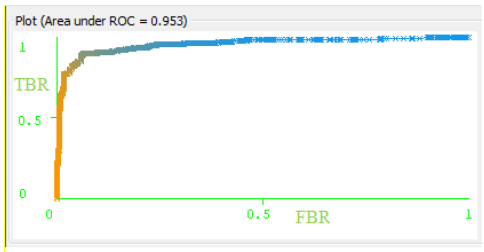

Fig. 9 ROC Curve – Combination 10 folds

So we can conclude that the best accuracy result obtained by using the combination of features, the reduced permissions, Methods and APIs is 91.5385 % and with weighted averages for FP Rate and TP Rate 0.085, 0.915 respectively.

D.Comparing between best results

The comparison between results obtained from the three models of feature extractions is shown in Fig. 10 and table III

The figure shows that the highest accuracy, which is 91.5385 % with high TP rate 0.915, high AUC value 0.953, and the lowest FP rate 0.085, belongs to the ordered features set based on the combination of features with Radial basis function (RBF) as machine learning classifier. While, coming in the second place is the ordered features set based on reduced permissions combined with MultilayerPerceptron as machine learning classifier with high detection accuracy 87.2308%, high TP rate 0.872, high AUC value 0.9143 and low FP rate 0.128.

Table III –AUC Comparison

Features	Permissions	Methods,APIs	Combination
AUC	0.9143	0.784	0.953

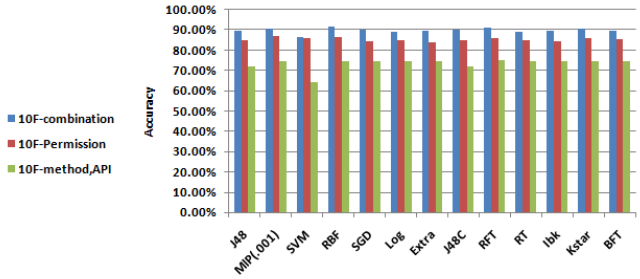

Fig. 10 Comparison for Classifiers Accuracy for different features sets

Although as shown in table II the features set for model 3 gives better result than features set for model 1, the extraction time for features of model 3 is larger than the extraction time for features of model 1. Where, the extraction time for features of model 3 is ranged between 3.0106 and 6.5084 seconds per application, while the extraction time for features of model 1 is ranged between 0.02843 and 0.0319 seconds per application as mentioned early in table II. That means, models based on reduced permissions features will be faster, but with lower accuracy.

IX. Conclusion

In this paper, we have proposed an effective approach of detecting malwares before installing it using static code analysis. It takes into account various features based on permissions declared in AndroidManifest.xml file and methods and APIs used in the applications. We have extracted the features from 650 application divided into 325 for malware representing 89 malware families and 325 benign applications.

The applied experiments concluded that models based on using reduced permissions as feature set is faster than other models with average extraction time ranged between 0.02843 and 0.0319 seconds per application and with AUC value 0.9143 and classification accuracy 87.2308%. While, models based on using a combination between permissions and API methods as a feature set is more accurate in classification with AUC value 0.953 and classification accuracy 91.5385%, but it need 3.0106 and 6.5084 seconds per application for extract features so it consumes more time to extract the needed features.

Acknowledgment

We would to thanks everyone help us to complete this research and i would to thanks my supervisors Prof. Aliaa and Dr. Marwa for their excellent guidance.

References

[1] Ahmed H. Mostafa, Marwa M. A. Elfattah and Aliaa A. A. Youssif. "Reduced Permissions Schema for Malware Detection in Android Smartphones". In Proc. Recent Advances in Computer Science, 19th Int. Conf. on Circuits, Systems, Communications and Computers (CSCC 2015), July 16-20, 2015, Zakynthos Island, Greece,, ISBN: 978-1-61804-320-7, pp. 406-413

[2] Faruki, P.; Bharmal, A.; Laxmi, V.; Ganmoor, V.; Gaur, M.S.; Conti, M.; Rajarajan, M., "Android Security: A Survey of Issues, Malware Penetration, and Defenses," *Communications Surveys & Tutorials, IEEE* , vol.17, no.2, pp.998,1022, Secondquarter 2015

[3] Smartphone Users Worldwide Will Total 1.75 Billion in 2014 [Online]. Available: http://www.emarketer.com/Article/Smartphone-Users-Worldwide-Will-Total-175-Billion-2014/1010536

[4] Mobile Application Futures 2013-2017 [Online]. Available: http://www.portioresearch.com/en/mobile-industry-reports/mobile-industry-research-reports/mobile-applications-futures-2013-2017.aspx

[5] Skovoroda, A.; Gamayunov, D., "Review of the Mobile Malware Detection Approaches," *Parallel, Distributed and Network-Based Processing (PDP), 2015 23rd Euromicro International Conference on* , vol., no., pp.600,603, 4-6 March 2015

[6] Worldwide market share forecast of smartphone operating system from 2010 to 2015 [Online]. Available: http://www.statista.com/statistics/266970/market-share-forecast-of-smartphone-operating-systems-from-2010-to-2015/

[7] Global Smartphone unit shipments forecast by operating system 2014 and 2018 [Online]. Available : http://www.statista.com/statistics/309448/global-smartphone-shipments-forecast-operating-system/

[8] Cisco: 2014 Cisco Annual Security Report [Online]. Available: http://www.cisco.com/web/offers/lp/2014-annual-security-report/index.html

[9] Android's Google Play beats App Store with over 1 million apps, now officially largest [Online]. Available: http://www.phonearena.com/news/Androids-Google-Play-beats-App-Store-with-over-1-million-apps-now-officially-largest_id45680.

[10] Abdelfattah Amamra, Chamseddine Talhi, and Jean-Marc Robert," Smartphone malware detection: From a survey towards taxonomy". In Malicious and Unwanted Software (MALWARE), 2012 7th International Conference on, pages 79–86. IEEE, 2012.

[11] Mariantonietta La Polla, Fabio Martinelli, and Daniele Sgandurra," A survey on security for mobile devices" Communications Surveys & Tutorials, IEEE, 15(1):446–471, 2013.

[12] Suarez-Tangil, Guillermo, et al. "Evolution, detection and analysis of malware for smart devices." Communications Surveys & Tutorials, IEEE 16.2 (2014): 961-987

[13] Yan Ma and Mehrdad Sepehri Sharbaf. "Investigation of static and dynamic android anti-virus strategies". In Information Technology: New Generations (ITNG), 2013 Tenth International Conference on, pages .403–398IEEE, 2013

[14] Gunjan Kapse et al, Detection of Malware on Android based on Application Features , (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 6 (4) , 2015, 3561-3564

[15] Ming-Yang Su; Wen-Chuan Chang, "Permission-based malware detection mechanisms for smart phones," *Information Networking (ICOIN), 2014 International Conference on* , vol., no., pp.449,452, 10-12 Feb. 2014

[16] Sato, Ryo, Daiki Chiba, and Shigeki Goto. "Detecting android malware by analyzing manifest files." *Proceedings of the Asia-Pacific Advanced Network*36 (2013): 23-31.

[17] Sanz, Borja, Igor Santos, Xabier Ugarte-Pedrero, Carlos Laorden, Javier Nieves, and Pablo Garcia Bringas. "Instance-based Anomaly Method for Android Malware Detection." In *SECRYPT*, pp. 387-394. 2013.

[18] Yerima, Suleiman Y., Sakir Sezer, Gavin McWilliams, Igor Muttik. "A new android malware detection approach using bayesian classification." *Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference on*. IEEE, 2013.

[19] Vala, Sarga and Benda. "Security Reverse Engineering of Mobile Operating Systems: A Summary." WSEAS, Recent Advances in Computer science. Proceedings of the 17th International Conference on computers. Rhodes Island, Greece. 2013.

[20] Sanz, Borja, et al. "Puma: Permission usage to detect malware in android." International Joint Conference CISIS'12-ICEUTE´ 12-SOCO´ 12 Special Sessions. Springer Berlin Heidelberg, 2013.

[21] Suarez-Tangil, G.; Tapiador, J.E.; Lombardi, F.; Di Pietro, R., "Alterdroid: Differential Fault Analysis of Obfuscated Smartphone Malware," *Mobile Computing, IEEE Transactions on* , vol.PP, no.99, pp.1,1,2015

[22] Burguera, Iker, Urko Zurutuza, and Simin Nadjm Tehrani. "Crowdroid: behavior-based malware detection system for android." Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices. ACM, 2011.

[23] Dini, Gianluca, et al. "Madam: a multi-level anomaly detector for android malware." Computer Network Security. Springer Berlin Heidelberg, 2012. 240-253.

[24] Naqliyah BT Zainuddin, Mohd.Faizal Bin Abdollah, and Shahrin Bin Sahib. "Framework of Analysis Technique for Abnormal Behavior in Mobile Application (FATABMA).", WSEAS

[25] Naqliyah BT Zainuddin , " A Study on Android-A Proposal for Cost-sensitive Based Intrusion Response System based IDS",WSEAS,Advances in Remote Sensing, Finite Differences and Information Security , ISBN: 978-1-61804-127-2 , WSEAS

[26] Adas, Husam, Sachin Shetty, and Waled Tayib. "Scalable detection of web malware on smartphones." In *Information and Communication Technology Research (ICTRC), 2015 International Conference on*, pp. 198-201. IEEE, 2015.

[27] Marengereke, Tendai Munyaradzi, and K. Sornalakshmi. "Cloud based security solution for android smartphones." *Circuit, Power and Computing Technologies (ICCPCT), 2015 International Conference on*. IEEE, 2015.

[28] Android Operating System [Online]. Available : https://www.android.com/

[29] Android SDK [Online]. Available : http://developer.android.com/sdk/index.html

[30] Contagio Mobile Mini Malware Dumb [Online]. Available : http://contagiominidump.blogspot.com/

[31] Android Malware Dump [Online]. Available: http://androidmalwaredump.blogspot.com/

[32] MalShare [Online]. Available : http://malshare.com/

[33] Cooper, Vanessa N., Hossain Shahriar, and Hisham M. Haddad. "A Survey of Android Malware Characterisitics and Mitigation Techniques." Information Technology: New Generations (ITNG), 2014 11th International Conference on. IEEE, 2014.

[34] Le Thanh, Hieu. "Analysis of Malware Families on Android Mobiles: Detection Characteristics Recognizable by Ordinary Phone Users and How to Fix It." Journal of Information Security 4.04 (2013): 213.

[35] Current Android Malware [Online] Available : http://forensics.spreitzenbarth.de/android-malware/

[36] Google Play store [Online]. Available : https://play.google.com/store

[37] Python 2.7 [Online]. Available : https://www.python.org/download/releases/2.7/

[38] Androguard Project [Online]. Available : https://code.google.com/p/androguard/

[39] wxpython [Online] Available : http://www.wxpython.org/

[40] Karegowda, Asha Gowda, A. S. Manjunath, and M. A. Jayaram. "Comparative study of attribute selection using gain ratio and correlation based feature selection." International Journal of Information Technology and Knowledge Management 2.2 (2010): 271-277.

[41] T. M. Cover, J. A. Thomas, Elements of Information Theory, Ed. Wiley, 1991.

[42] Android system Permissions [Online]. Available : http://developer.android.com/guide/topics/security/permissions.html

[43] List of Android Manifest Permissions [Online]. Available : http://developer.android.com/reference/android/Manifest.permission.html

[44] Android Lollipop 5 [Online]. Available : http://www.android.com/versions/lollipop-5-0/

[45] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, Ian H. Witten (2009); The WEKA Data Mining Software: An Update; SIGKDD Explorations, Volume 11, Issue 1.

[46] Kohavi, Ron. "A study of cross-validation and bootstrap for accuracy estimation and model selection." In *Ijcai*, vol. 14, no. 2, pp. 1137-1145. 1995.

[47] Powers, David Martin. "Evaluation: from precision, recall and F-measure to ROC, Informedness, Markedness and Correlation." (2011).