Mobile Malware Analysis using Independent and Ensemble Features

Aswini, A M, Vinod P
SCMS School of Engineering and Technology
Vidyanagar Karukutty
Ernakulam 683582
India
aswinimohan95@gmail.com

ABSTRACT: *The purpose of this paper is to statically analyze the android application package files to detect zero-day attacks. The methodology deals with attribute extraction using dissemblers, feature reduction by sparse feature elimination, feature selection and ranking by implementing various feature selection techniques, aggregation of attribute categories followed by classification and prediction. Feature selection techniques such as Bi-Normal separation (BNS), Mutual Information (MI), Feature to class correlation (F-CC), Feature to feature correlation (F-FC), combination of feature to class and feature to feature correlation (FCFF), Comprehensive measurement feature selection (CMFS) and Optimal orthogonal centroid feature selection (OCFS) are implemented to choose the significant attributes. Prominent features of five different attribute categories like permissions, count of permissions, hardware features, software features as well as API calls from 1175 application packages are extracted to generate the classification model. Attribute aggregation is performed to build the ensemble model. The intention of this framework is to evaluate the effectiveness of ensemble features with respect to individual features; find out the best feature selection method with fewer feature length and classification algorithm. The framework developed here by implementing dimensionality reduction and machine learning algorithms depicts an overall classification accuracy of 93.02% using ensemble features. Evaluating the performance of ensemble model with independent model, the former provides better results with Bi-Normal separation.*

Keywords: Android Malware, Feature Extraction, Feature Selection, Static Analysis

Received: 11 September 2015, Revised 23 October 2015, Accepted 29 October 2015

1.Introduction

According to the global provider of market intelligence International Data Corporation (IDC) [7], Android is the most popular operating system [21] which is ranked top among the available mobile OSs on the basis of shipment volumes and market share. Their widespread use and open source nature motivated the malware writers to focus on developing the apps with malicious intents. Third party app stores with niche apps offer .apk files with more features than those provided by the official market like Google play [8]. Such apps with improved characteristics encourage the users to be the regular customers of these repositories that lack security checks. These third-party apps developed by injecting malicious code to the legitimate apps seems to be useful at first sight as it provide additional functionalities to the user. The user unaware of its malicious activities fall prey to such apps. The device with malicious app installed in it is susceptible to threats which may result in loss of personal data, charging cost of a premium-rate call or SMS, DOS attacks etc [20]. According to the Symantec Corporation Internet Security Threat Report 2014 [5], benign applications are downloaded by the malicious authors frequently from Google play store and are modified to third party apps containing illegitimate program. Mobile threat report 2014 of F-Secure labs [6] reported that Trojans are capable of tracing locations, sending SMS's, link click, create fraudulent transaction, steal data etc.

Due to the above stated issues and threats; smartphone security should be given higher priority. The security solution like Antivirus mainly employs signatures [16]. These signatures can identify known malicious specimens that are already detected. It is a tedious process and cannot detect novel malware as the signatures cannot generalize to identify zero day malware. Also technical expertise is required to create signatures and the repository has to be updated frequently. Therefore, it is required to find out a better method to detect the malicious .apk files.

The proposed framework is designed to detect illegitimate .apk files. Here, five different attributes are extracted and feature selection techniques are applied to find out the significant attributes. Classification models are generated using these prominent attributes. Ensemble model is designed by aggregating the significant attributes from each category of feature. Finally, prediction is carried out using these models to evaluate the classification accuracy. We evaluate the performance of both independent and ensemble features.

The remaining sections are organized as follows: Section 2 deals with some related works which implemented static analysis in mobile malware. The proposed method is detailed in Section 3 and Section 4 comprises of experiments and results. Section 5 includes inference of this work. Section 6 presents the comparison with previous works and finally the conclusion and the future work are presented in sections 7 and 8 respectively.

2. Related Works

DroidAPIMiner in [24] is used to extract Application programming interface calls using a modified python tool named Androguard and different classifiers are evaluated using the feature set. They achieved an accuracy of 99% and false positive rate of 2.2% using k-NN classifier.

The authors [29] carried out the experiments using 124,769 benign and 480 malicious applications. They used permissions declared in the specimens to detect malicious apps in Android OS. The attributes were requested and required permissions, number of required permissions, normal, signature, dangerous permissions, number of files with .so extensions, number of elf files, count of executables, shared objects etc. Performance of this system is evaluated using various machine learning techniques. Results showed that a permission-based detector can detect more than 81% of illegitimate .apk files.

MAMA [32] utilized the permissions and feature tags within the Android manifest file. Their dataset comprised of 333 benign and 333 malware samples. Their investigations provided 87.41% accuracy for permission based model, 86.09% for feature model and 94.83% for permission and feature combined model. The improved results were obtained with Random Forest, using 100 trees in all the cases.

In [26], the authors implemented dimensionality reduction and machine learning techniques to find out the attributes that contributed to the identification of malicious files. They presented a static analyzer named Droid Permission Miner that mines permissions from the manifest file. Feature selection methods like Bi-normal Separation (BNS) and Mutual Information (MI) were used and obtained an accuracy of 81.56% with 15 features using MI feature selection approach.

The authors [33] carried out the experiment with 200 .apk samples. They proposed a machine learning based malware detection framework to distinguish benign and malicious samples. The attributes used were permissions extracted from the .apk samples. The generated models were trained; and are evaluated using the Area Under ROC Curve (AUC). They recorded an accuracy of 91.58% using Random Forest classifier.

PUMA [25] deploys permissions from the .apk samples for classification and implements machine learning techniques. They used 239 Android malware samples and mined the attributes, trained the models and evaluated each configuration using the Area Under ROC Curve (AUC). They obtained a 0.92 of AUC using the Random Forest classifier. All the classifiers resulted in accuracy higher than 80% except Bayesian classifier. The best classifier was observed to be Random Forest (50 trees) with an accuracy of 86.41%.

This [31] approach is carried out with the uses-permission and the uses-feature tags extracted from the manifest file. Using the features of benign samples they created an instance based anomaly detection model to detect malware applications. Manhattan distance, Euclidean distance and Cosine distance was implemented in this work. In case of Manhattan distance, AUC value of 0.88 was obtained for using average as a parameter in the combination rule resulted in 85% accuracy. Using Euclidean distance, 0.90 of AUC and 87.57% accuracy was achieved. The best results of 0.91 of AUC and nearly 90% of sureness is obtained using Cosine distance.

DREBIN [30] performs static analysis by extracting the maximum possible number of features of an application's code and manifest file. The features are grouped in sets of strings of permissions, API calls and network addresses and are embedded in a joint vector space by applying machine learning strategies. The dataset for experiment includes 123,453 benign and 5,560 malware samples. It detects 94% of the malicious specimens with relatively less false positive rate.

Authors in [18], [19], [22] proposed a supervised anomaly based method named Andromaly to extract 88 contributing features. Detection rates were better for the database with benign games than benign tools when used in combination with the 4 malicious apps. Naïve Bayes and Logistic Regression were found to be the better classifiers with reported results.

The authors in [20] proposed permission based static mechanism for detecting the malicious applications. The experiment was performed on 46 pieces of iOS, Symbian 9.x and Android malware. Most common malware activities were found to be disposing user information (61%) as well as sending SMS (52%). They observed that four pieces of malicious files used root exploits to perform attack on the Android device. They surveyed the act of non-malicious smartphone tinkerers in publishing root exploits and surveyed the availability of root exploits.

MADAM [23], a real-time multilevel anomaly detector monitors Android both at kernel as well as user level. They used 13 features and implemented machine learning techniques to detect unknown malware. The results depicted an accuracy of 100%. It monitors system calls to monitor energy consumption user activity, incoming/outgoing traffics files, memory access and sensors status. MADAM detected all the 10 monitored real malicious software.

3. Proposed Methodology

Androguard [1] is used to extract permissions, count of permissions, API calls and software/hardware feature for identifying illegitimate samples. The permissions, software/hardware features and count of permissions are obtained from the Android manifest file. The API calls from each .apk files are also extracted using the same tool. Experiments are carried out on the basis on two aspects (1) individual features and (2) ensemble features. Figure 3 and Figure 4 shows the architecture of our proposed model. These classification models are briefly explained in the coming sections. Table 1 and Figure 2 include the .apk file components and its descriptions.

3.1 Datasets

The investigations are carried out using 1175 .apk files collected from various sources. It comprises of 575 malicious samples downloaded from Contagiodump [2] and received from various user agencies. The rest 600 benign applications are downloaded from various publicly available internet sources that allows free downloads of mobile applications for Android OS. The benign samples are allocated such that 300 files are included in the test set and the remaining 300 samples are added to the training set. From the 575 illegitimate .apk files, the test set is provided with 287 .apk files and the rest 288 files are added to the train set.

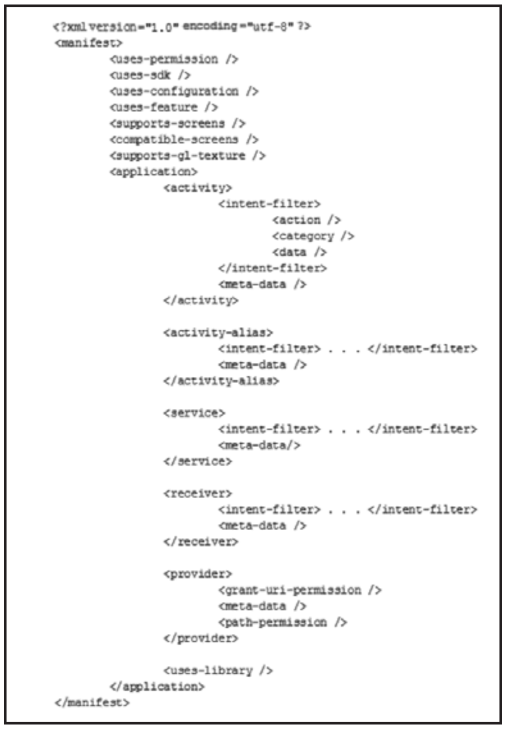
3.2 Feature Categories

To statically examine each Android application as benign or malicious; the attributes are mined without executing the apps. The descriptions about these attributes are given below   :

i.**Permissions:** The Android OS provides a well framed permission mechanism incorporated within its security model. The functionalities of the .apk file depend upon the permissions requested by it in the Android manifest file. It is declared statically and there is no provision to declare it dynamically. Permissions are declared in the <uses-permissions> tag in the Manifest file (Refer Figure 5). It includes an android:name attribute that gives the permission name. For example RECEIVE_BOOT_COMPLETED, GET_ACCOUNTS, SEND_SMS, READ_SMS etc are some of the permissions.

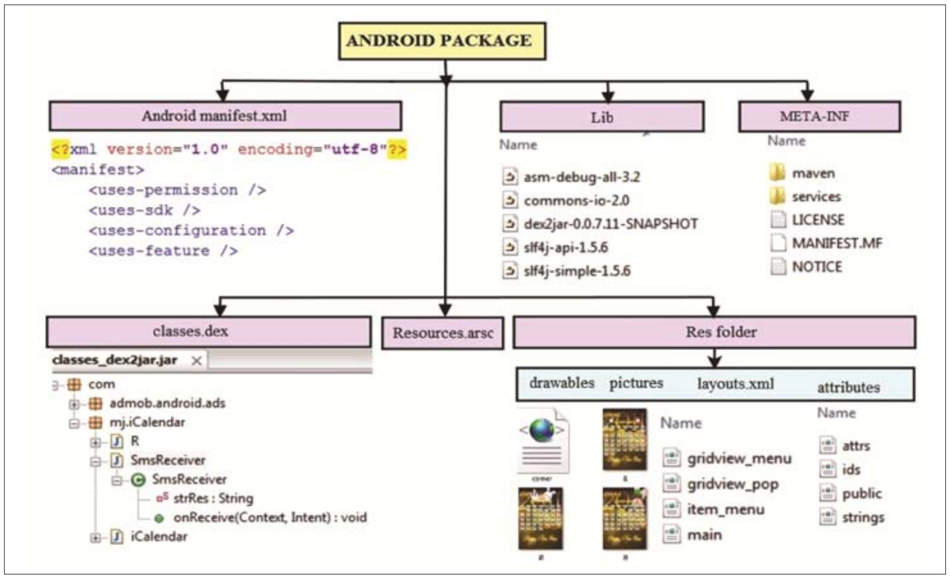
Components	**Description**
Android manifest file:	An XML file that provides the details about hardware and software features required by application, version, permissions provided by the .apk file, metadata, package name etc (Refer Figure 1).
Res folder:	It contains the resources of application package file that defines languages, sound settings, graphics layout, attributes, drawables etc. The res directory comprises of resources that are not compiled into resources.arsc
Classes.dex:	These are Dalvik virtual machine executables that execute in Dalvik machine. It includes the compiled Java source code in .dex format. The res directory comprises of resources which are not compiled into resources.arsc.
Lib:	It consists of native libraries (for C or C++) that can be used through NDK (Native Development Kit).
META-INF:	It includes application's signature mainly aimed at its security and integrity. It comprises of application's certificate CERT.RSA, manifest file MANIFEST.MF and CERT.SF that includes the list of resources and SHA-1 digest of the MANIFEST.MF file
Resources.arsc	It is obtained after the compilation.

Table 1. Components of Android application package file



```
<?xml version="1.0" encoding="utf-8" ?>
<manifest>
        <uses-permission />
        <uses-sdk />
        <uses-configuration />
        <uses-feature />
        <supports-screens />
        <compatible-screens />
        <supports-gl-texture />
        <application>
                <activity>
                        <intent-filter>
                                <action />
                                <category />
                                <data />
                        </intent-filter>
                        <meta-data />
                </activity>

                <activity-alias>
                        <intent-filter> . . . </intent-filter>
                        <meta-data />
                </activity-alias>

                <service>
                        <intent-filter> . . . </intent-filter>
                        <meta-data/>
                </service>

                <receiver>
                        <intent-filter> . . . </intent-filter>
                        <meta-data />
                </receiver>

                <provider>
                        <grant-uri-permission />
                        <meta-data />
                        <path-permission />
                </provider>

                <uses-library />
        </application>
</manifest>
```

Figure 1. Android manifest file format



```
<?xml version="1.0" encoding="utf-8"?>
<manifest>
    <uses-permission />
    <uses-sdk />
    <uses-configuration />
    <uses-feature />
```

Figure 2. Android package file format

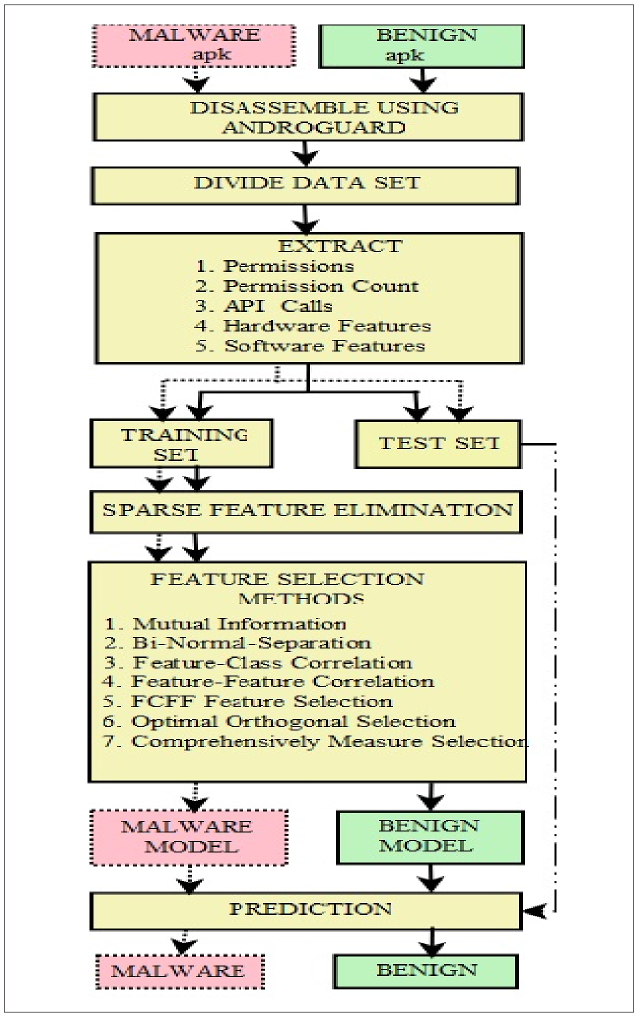


Figure 3. Architecture of Individual feature model

i. **Count of permission in each file:** This attribute set is generated by computing the number of permissions requested by an Android application

ii. **Application Programming Interface (API) calls:** API calls are employed so as to request with the user queries to the repository and respond back to the user. The application programming interface calls should be invoked at the run time to perform some specific tasks required by the user. Some examples are `onCreateOptionsMenu()`, `onDraw()`, `onActivityResult ()`, `setSpeechRate()`, `setPitch()`, `addSpeech()` etc. Refer Figure 7 for the snippet of API calls obtained using the Androguard tool

iii. **Software Features:** The software features required by the application for its execution is defined within the `<uses-feature>` tag in the Manifest file. It includes an android: required attribute that chooses true or false options to determine whether the app could or could not function without it. For example `device_admin`, `app_widgets`, `home_screen etc`. (refer Figure 6).

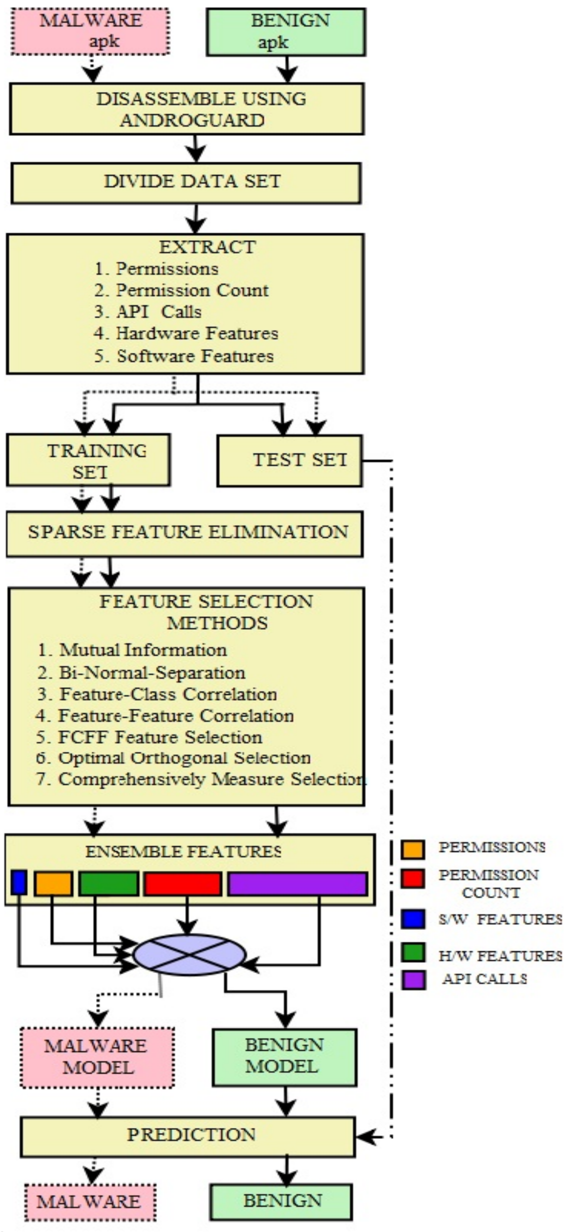
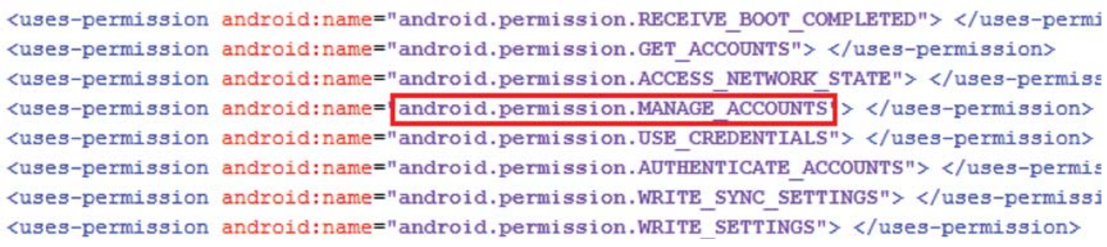
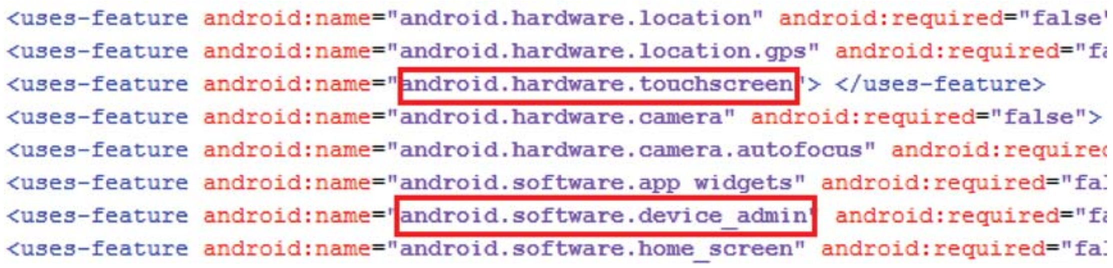


Figure 4. Architecture of Ensemble model



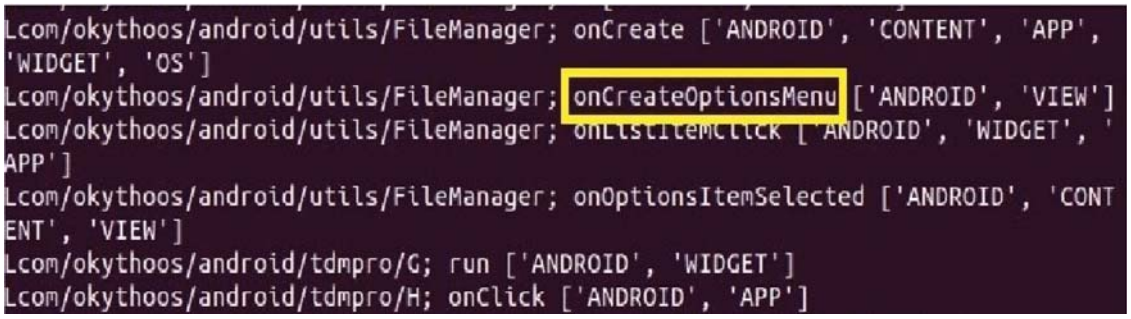
```
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"> </uses-permi
<uses-permission android:name="android.permission.GET_ACCOUNTS"> </uses-permission>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"> </uses-permiss
<uses-permission android:name="android.permission.MANAGE_ACCOUNTS"> </uses-permission>
<uses-permission android:name="android.permission.USE_CREDENTIALS"> </uses-permission>
<uses-permission android:name="android.permission.AUTHENTICATE_ACCOUNTS"> </uses-permis
<uses-permission android:name="android.permission.WRITE_SYNC_SETTINGS"> </uses-permissi
<uses-permission android:name="android.permission.WRITE_SETTINGS"> </uses-permission>
```

Figure 5. Android Permissions declared within <uses-permission> tag



```
<uses-feature android:name="android.hardware.location" android:required="false'
<uses-feature android:name="android.hardware.location.gps" android:required="fa
<uses-feature android:name="android.hardware.touchscreen"> </uses-feature>
<uses-feature android:name="android.hardware.camera" android:required="false">
<uses-feature android:name="android.hardware.camera.autofocus" android:require
<uses-feature android:name="android.software.app widgets" android:required="fa
<uses-feature android:name="android.software.device_admin" android:required="fa
<uses-feature android:name="android.software.home_screen" android:required="fa
```

Figure 6. Software and Hardware features within<uses-feature> tag



```
Lcom/okythoos/android/utils/FileManager; onCreate ['ANDROID', 'CONTENT', 'APP',
'WIDGET', 'OS']
Lcom/okythoos/android/utils/FileManager; onCreateOptionsMenu ['ANDROID', 'VIEW']
Lcom/okythoos/android/utils/FileManager; onListItemClick ['ANDROID', 'WIDGET', '
APP']
Lcom/okythoos/android/utils/FileManager; onOptionsItemSelected ['ANDROID', 'CONT
ENT', 'VIEW']
Lcom/okythoos/android/tdmpro/G; run ['ANDROID', 'WIDGET']
Lcom/okythoos/android/tdmpro/H; onClick ['ANDROID', 'APP']
```

Figure 7. API calls obtained using Androguard tool

i.**Hardware Features:** It gives information about the set of hardware features on which the application depends. For example: touchscreen, camera, camera.autofocus etc. (refer Figure 6).

3.3 Feature Extraction

The .apk files which are originally in binary format are provided as input to the disassembler tool. The tool component androaxml.py is used to convert the manifest file which is originally in binary format to human readable .xml form. This file includes the permissions (within <uses-permission> tag) as well as s/w and h/w features (within <uses-feature> tag). Similarly, the python script androapkinfo.py is implemented to extract the API calls of the specimens. Refer Figure 8 and Figure 9 for the commands executed to obtain the output. The number of permissions in each file is considered as another attribute to foster the classification model.

3.4 Data Pre-Processing

During this phase, the attributes are mined from the test and train set. After feature extraction, (which is an initial step to data pre-processing) the redundant, irrelevant information that appears as noise in the training set should be filtered off. For this, feature pruning is carried out to eliminate the features (sparse attributes) that do not contribute to the classification of malicious samples.

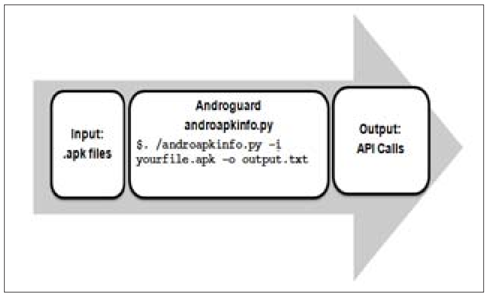
After removing the unreliable information, features common to both the classes ($M \cap B$) are taken into consideration. This feature set is given more importance than other category of attribute sets like malware and benign features ($M \cup B$), discriminant benign features ($B \setminus M$) and discriminant malware ($M \setminus B$) features as they give improved accuracy according to our previous work [26].

3.5 Attribute Selection Techniques

Feature selection in data mining is the technique to select significant features for model creation. The input feature list obtained as a result of initial data pre-processing steps still contains noisy attributes as well as highly correlated features. These irrelevant features have to be eliminated by selecting only the required features as their presence increases the time and processing power.

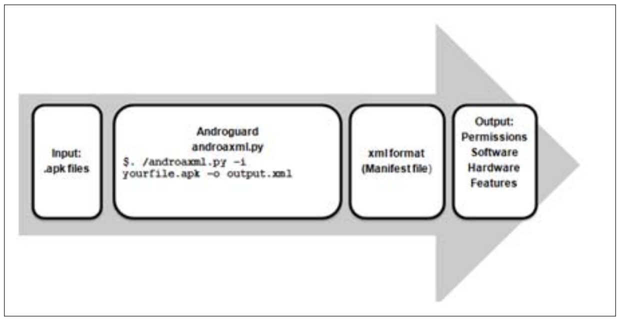
Bi-Normal Separation (BNS) (based on Z-Score), Mutual Information (MI), feature to class correlation (F-CC), feature to feature correlation (F-FC), FCFF feature selection (FCFF), comprehensively measure feature selection (CMFS) and optimal orthogonal centroid feature selection methods (OCFS) are implemented to build the classification model in our proposed method. The mathematical formulas of these variable selection techniques are included in Table 2.

Bi-Normal Separation (BNS) chooses the positive and negative attributes and is not biased to the malware or benign class. The score is determined using the statistical table for Z-score [4]. Value of F^{-1} is obtained from this table. F^{-1} values of both true positive rate (TPR) as well as false positive rate (FPR) are computed and the absolute differences of these values are treated as the BNS score (Refer equations in Table 2).



```
$. /androapkinfo.py -i
yourfile.apk -o output.txt
```

Figure 8. Androguard command to output API Calls



```
$. /androaxml.py -i
yourfile.apk -o output.xml
```

Figure 9. Command to output Permissions and software/hardware features

The attributes are ranked on the basis of this score in order to generate different feature sets of varying attribute lengths.

Mutual Information (MI) provides the knowledge about the extent to which an attribute *F* reduces the uncertainty in identifying the right class *C*. It computes the mutual dependence of two random variables.

Feature to Class Correlation (F-CC) measures the correlation between a feature and a class. It is based on the joint probability distribution function between the attribute and its associated class (malware and benign class). The marginal probability distribution of feature and class is also computed to find out the score of an attribute. It chooses the attribute that highly represents its membership in a class.

Feature to Feature Correlation (F-FC) is also known as Pearson's correlation coefficient. It measures the strength, correlation and direction of a linear relationship between two attributes. The correlation coefficient r has the value between +1 and -1. i.e. $-1.00 \leq r \leq +100$. The value $r = 0.0$ represents the absence of a linear relationship (weak correlation), $r = +1$ 0 or -1.0 represents the presence of a perfect linear relationship (perfect correlation) [37].

FCFF is a combination of feature to class and feature to feature correlation techniques that are mentioned above. The attributes are initially scored on the basis of their F-CC value. The scored features are ranked in the descending order of their scores to select the top 50% attributes from the ranked set. The F-FC values of these selected attributes are computed with each and every other attributes in the selected list. Correlation of a feature to itself generates a value of 1.00 as they are highly correlated to one another. From this list, every attribute pairs with $0 \geq r \geq -0.7$ (negatively correlated) and $0 \leq r \leq 0.7$ (positively correlated) are considered in our evaluation (the pairs with *r* value beyond these values approaches to more correlated features). These feature pairs are arranged in the descending order of values of their Pearson's correlation coefficient.

Comprehensive Measurement Feature Selection (CMFS) measures the importance of an attribute by calculating its significance from both inter-category and intra-category. To measure the goodness of a feature, the class specific scores of an attribute can be combined in the following ways (using equation 1 or 2):

$$CMFS_{average}(t_k) = \sum_{i=1}^{|C|} P(C_i) \quad CMFS(t_k, C_i) \tag{1}$$

$$CMFS_{max}(t_k) = \max_{i=1}^{|C|} CMFS(t_k, C_i) \tag{2}$$

Optimal orthogonal centroid feature selection method (OCFS) is based on a supervised feature extraction algorithm known as the orthogonal centroid algorithm [38]. The steps involved in the implementation are described below:

Step 1: Compute the centroid for attributes in benign class and malware class using equations 3 and 4.

$$m_B^i = \frac{1}{n_B} \sum_{x_i \in class\ B} x_i \ , \tag{3}$$

$$m_M^i = \frac{1}{n_M} \sum_{x_i \in class\ M} x_i \tag{4}$$

Where *i* is the set of features, x_i is the frequency of feature *i*

Step 2: Compute the centroid for all attributes in the training sample using equation 5.

$$m^i = \frac{1}{N} \sum_{i=1}^{N} x_i \tag{5}$$

Step 3: Calculate the feature score using the equation 6.

$$OCFS(t_k) = \sum_{i=1}^{|C|} \frac{n_i}{n} (m_i^k - m^k)^2 \tag{6}$$

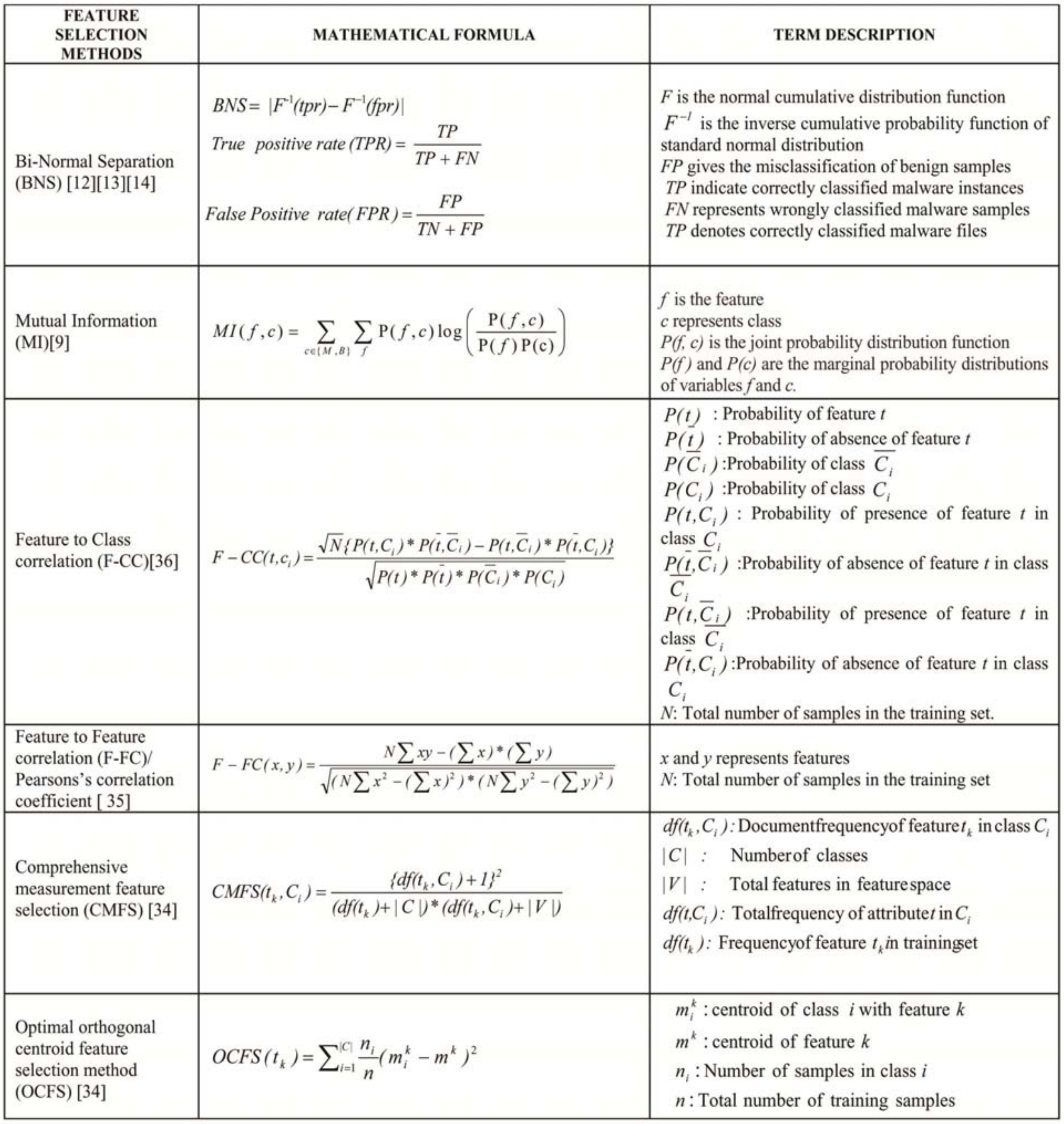
These feature selection techniques are applied to select the required features for classification based on the scores assigned to each attribute.

3.6 Feature Aggregation

The optimal feature space of the five individual attribute categories are aggregated to form a new feature type referred as ensemble features. It may improve the detection rate as extraneous attributes would be already eliminated (refer Figure 4 and 10).

3.7 Classification and Prediction

The class labels are assigned in advance to the learning set in case of supervised learning. The unclassified specimens in the test set are finally assigned with the predicted class labels (malware/benign). The input to the classification algorithms used in this work includes feature vector tables with prominent features of varying attribute lengths (refer steps 1 and 2 in Figure 10). For each feature category, optimal feature space is found out by comparing the classification accuracies and this process is followed by generating ensemble models with the combined feature space for classification (refer step 3, 4 and 5).



FEATURE SELECTION METHODS	MATHEMATICAL FORMULA	TERM DESCRIPTION
Bi-Normal Separation (BNS) [12][13][14]	$BNS = \lvert F^{-1}(tpr) - F^{-1}(fpr) \rvert$ $True\ \ positive\ rate\ (TPR) = \frac{TP}{TP + FN}$ $False\ Positive\ \ rate(FPR) = \frac{FP}{TN + FP}$	F is the normal cumulative distribution function F^{-1} is the inverse cumulative probability function of standard normal distribution FP gives the misclassification of benign samples TP indicate correctly classified malware instances FN represents wrongly classified malware samples TP denotes correctly classified malware files
Mutual Information (MI)[9]	$MI(f,c) = \sum_{c \in \{M,B\}} \sum_{f} \mathrm{P}(f,c) \log\left(\frac{\mathrm{P}(f,c)}{\mathrm{P}(f)\mathrm{P}(c)}\right)$	f is the feature c represents class $P(f, c)$ is the joint probability distribution function $P(f)$ and $P(c)$ are the marginal probability distributions of variables f and c.
Feature to Class correlation (F-CC)[36]	$F - CC(t,c_i) = \frac{\sqrt{N}\{P(t,C_i) * P(\bar{t},\bar{C}_i) - P(t,\bar{C}_i) * P(\bar{t},C_i)\}}{\sqrt{P(t) * P(\bar{t}) * P(\bar{C}_i) * P(C_i)}}$	$P(t)$: Probability of feature t $P(\bar{t})$: Probability of absence of feature t $P(\bar{C}_i)$:Probability of class $\bar{C}_i$ $P(C_i)$:Probability of class C_i $P(t,C_i)$: Probability of presence of feature t in class C_i $P(\bar{t},\bar{C}_i)$:Probability of absence of feature t in class $\bar{C}_i$ $P(t,\bar{C}_i)$:Probability of presence of feature t in class $\bar{C}_i$ $P(\bar{t},C_i)$:Probability of absence of feature t in class C_i N: Total number of samples in the training set.
Feature to Feature correlation (F-FC)/ Pearsons's correlation coefficient [35]	$F - FC(x,y) = \frac{N\sum xy - (\sum x) * (\sum y)}{\sqrt{(N\sum x^2 - (\sum x)^2) * (N\sum y^2 - (\sum y)^2)}}$	x and y represents features N: Total number of samples in the training set
Comprehensive measurement feature selection (CMFS) [34]	$CMFS(t_k, C_i) = \frac{\{df(t_k, C_i) + 1\}^2}{(df(t_k) + \lvert C \rvert) * (df(t_k, C_i) + \lvert V \rvert)}$	$df(t_k, C_i)$: Documentfrequencyof feature t_k in class C_i $\lvert C \rvert$: Numberof classes $\lvert V \rvert$: Total features in featurespace $df(t, C_i)$: Totalfrequency of attribute t in C_i $df(t_k)$: Frequencyof feature t_k in trainingset
Optimal orthogonal centroid feature selection method (OCFS) [34]	$OCFS(t_k) = \sum_{i=1}^{\lvert C \rvert} \frac{n_i}{n}(m_i^k - m^k)^2$	m_i^k : centroid of class i with feature k m^k : centroid of feature k n_i : Number of samples in class i n : Total number of training samples

Table 2. Attribute selection techniques

The classifiers implemented in WEKA [17] such as AdaBoostM1 with J48 as base classifier (ADA) [10][28], Random Forest (RF) [11][27] [No: of Trees = 40, seed = 3] and J48 are used for classification.

AdaBoost.M1 with J48 as base classifier (ADA): It incorporates the base classifier J48 as the accuracy can be improved by using multiple classifiers than a single classifier. Boosting enhances the potential of a weak classifier. The classification models generated from the training data by the process of repeated learning by weak learners are combined to a single strong model.

Random Forest (RF): The RF classifier grows an ensemble of decision trees. These are generated at the training time by creating random vectors that deal with the growth of each tree. Each tree vote for the most favoured class to obtain the final output class. It implements the bagging technique to resample the training data and train the weak learners on this data. Its randomness allows the random selection of features for tree construction.

J48: It is a decision tree based classification algorithm that implements C4.5 algorithm in WEKA. The classification process is modelled by constructing a decision tree traversed in top-down approach. The tree traversals generate rules that contribute to classification.

3.8 Evaluation Parameters

Accuracy gives the degree of correctness of the model in classifying the test samples [15]. It specifies how close to the actual class the predicted class is (equation 7 is used to compute accuracy).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \qquad (7)$$

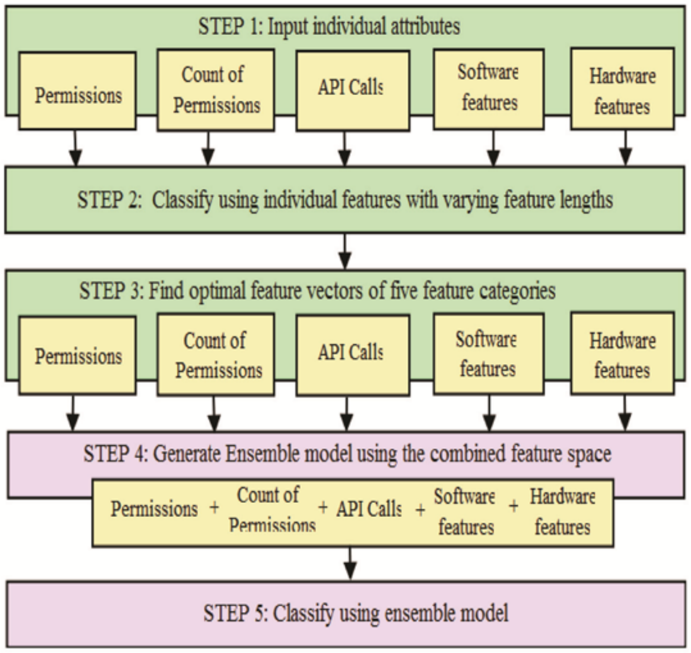

STEP 1: Input individual attributes
Permissions
Count of Permissions
API Calls
Software features
Hardware features
STEP 2: Classify using individual features with varying feature lengths
STEP 3: Find optimal feature vectors of five feature categories
Permissions
Count of Permissions
API Calls
Software features
Hardware features
STEP 4: Generate Ensemble model using the combined feature space
Permissions + Count of Permissions + API Calls + Software features + Hardware features
STEP 5: Classify using ensemble model

Figure 10. Steps to generate ensemble model from individual

Accuracy does not indicate the proportion of malware samples that are correctly identified as malware and misclassified as benign samples. To have knowledge about these proportions, the true positive rates (TPR) and false positive rates (FPR) should be evaluated. The equations for TPR as well as FPR are included in Table 2.

The classification accuracies of models built with distinct feature sets are compared to find out the:

1) Best feature selection method.
2) Optimal feature vector length.
3) Best feature category (permissions, count of permissions, software/hardware features or API calls).
4) Best classifier
5) Comparative analysis of ensemble and individual features.

4. Experiments and Results

The experimental environment is set up with Ubuntu 12.04 OS, Intel core i3 CPU and 4GB RAM. The investigations are carried out in two phases (1) considering independent attributes and (2) ensemble features. These two steps are discussed briefly in the following subsections.

4.1 Evaluation with Independent Features

A given permission, software and hardware feature is declared only once in an apk file. So the presence/absence (1 or 0) of these attributes represented in the Boolean format are considered to generate the feature vector tables (FVT). API calls in most cases are invoked more than once in an application. In this case, the classification models can be generated using Boolean FVT as well as frequency FVT. The former incorporates the presence/absence of an API in a sample and the latter considers the frequency of each attribute in the sample.

Performance Evaluation with Boolean FVT:

The benign and malware train samples respectively contributed 195 and 109 unique permissions. About 78 common permissions ($M \cap B$) are found within these unique lists. These are arranged based on their BNS, MI, F-CC, F-FC, CMFS, OCFS and FCFF scores in descending order.

Classification models are generated using top 10, 20..70 ranked permissions based on the feature selection methods. For BNS, the bottom ranked permissions are considered as the top ranked BNS attributes give less classification accuracy according to our prior work [26]. The same activity is carried out for permissions with permission count for the training samples. In case of API calls, 7,174 and 29,765 unique attributes are obtained for malware and benign train samples. In order to reduce the feature space, 50% of irrelevant API calls are filtered off to obtain rest 14,882 benign and 3587 malware API's. From this pruned feature set, 2166 common API's (MB) are determined. BNS API calls are selected from bottom (bottom scored) whereas top scored APIs are selected for rest of the feature selection techniques. Boolean FVT's are generated for variable feature length (i.e. 50, 100, 200.....1000).

The experiment results depict that 30 BNS permissions give 92.51% accuracy using Random Forest classifier. It is observed that the classification model generated using MI, F-CC, F-FC, CMFS, OCFS and FCFF features uses more number of permissions than BNS to generate the better model. For the count of permissions, the optimal feature length is observed to be 41 with an accuracy of 92.50% using OCFS with Random Forest. BNS, MI, F-CC, F-FC and FCFF give improved accuracy with 71 features whereas CMFS uses 10 features more than OCFS to provide 92.16% accuracy (refer Table 3).

For API calls with Boolean FVT, model built with 50 APIs provided an accuracy of 90.81% using BNS. MI gives an accuracy of 91.15% using 550 features more than BNS. FCC uses extra 850 features to gain 90.80% accuracy, FFC with 900 features give 90.80% accuracy, CMFS gives 92.33% accuracy with 400 APIs, OCFS uses 950 features more BNS to attain 91.82% accuracy. So, BNS with 50 APIs take less processing time and uses precise attributes compared to the features selected with other feature selection techniques (refer Table 3).

The feature space for software and hardware features are not pruned in the feature pre-processing stage and feature selection techniques are not applied to these attributes. The 40 h/w and 7 s/w features depicted 56.12% and 52.04% accuracies respectively (refer Table 4).

Performance Evaluation with Frequency FVT:

Classification models are generated using the frequencies of APIs in the samples. Bottom 100 BNS APIs are optimal as they provide an accuracy of 91.83% with Random forest. CMFS APIs provide 92.84% accuracy but considers 400 features more than BNS to have 1.01% increase in accuracy (refer Table 3).

From the results for evaluation with Boolean and frequency FVT of independent features, BNS is better for every feature categories as it uses precise attributes and consumes less processing time for classification. BNS Permissions are found to be the best feature category when compared with API calls and count of permissions as these models depict 92.51% accuracy with 30 features. For all the cases, Random Forest gives better detection rate (refer Table 5).

4.2 Evaluation with Ensemble Features

For each feature selection technique, two ensemble models are generated using (1) the frequencies of prominent APIs in each file and (2) considering the Boolean value of APIs. These two categories of API calls are separately aggregated with the presence/absence of other four categories of features.

From Table 6, ensemble model fabricated using frequency FVT gives an accuracy of 93.87% (for 218 features) with Random

Forest classifier for BNS method. Even though, the highest accuracy is attained using FCFF technique (94.37%), this method uses extra 680 attributes that increases the processing time.

CMFS employs 508 features to attain highest accuracy of 94.03% in the category of Boolean ensemble models but requires more processing time. So, 168 BNS attributes providing 93.02% accuracy are found to be the optimal features (refer Table 7).

Summarizing the results for ensemble feature category, Boolean features with BNS that uses 168 features are the preferred categories as they require less number of features and less processing time (refer Table 6 and 7).

Trivial and significant permissions/API calls that are used by malicious and legitimate .apk files with their descriptions are given in Tables 9, 10, 12 and 13.

For an ideal malware analyzer, the TPR should be higher with minimum FPR rates. Figure 11 depicts the 30 BNS permissions with improved accuracy (92.51%), TPR (88.50%) and minimum FPR (3.6%) when compared with rest of the feature space. For permissions with permission count, highest accuracy of 92.50% is achieved with OCFS using 41 attributes. This feature space has depicted a minimum false positive rate of 3.33% with higher true positive rate of 88.1% (refer Figure 12).

For frequency FVT, 900 attributes give 4% FPR with reduced TPR of 89.19% and accuracy of 91.68% (refer Figure 13). But, BNS with 100 API calls (giving 91.83% accuracy, 90.24% TPR with 6.66% FPR) are selected as the best case. The reason behind this is that they require less attributes to achieve higher accuracy and TPR compared to the other feature spaces thus avoiding irrelevant features. BNS Boolean FVT with 50 API calls provides 90.81% accuracy, 88.15% TPR with 6.6% FPR (refer Figure 14).

4.3 Processing Time

In this subsection, we discuss with the processing time for testing unseen samples (in seconds) with the Random forest classifier. In all the cases, except for permission count, BNS with less number of attributes are found to be the best method. Table 8 provides information about the comparison of processing overhead of BNS attributes with other feature selection techniques. The other attribute sets are selected for comparison based on the criteria that they exhibits improved accuracy with more number of features in the corresponding feature category.

For permissions with permission count, time consumed by OCFS technique for testing the samples is compared with F-CC method.

In case of ensemble features, the processing overhead of 168 BNS attributes for Boolean model are compared with 508 CMFS attributes. Similarly, 218 BNS attributes for frequency model are compared with 898 FCFF features.

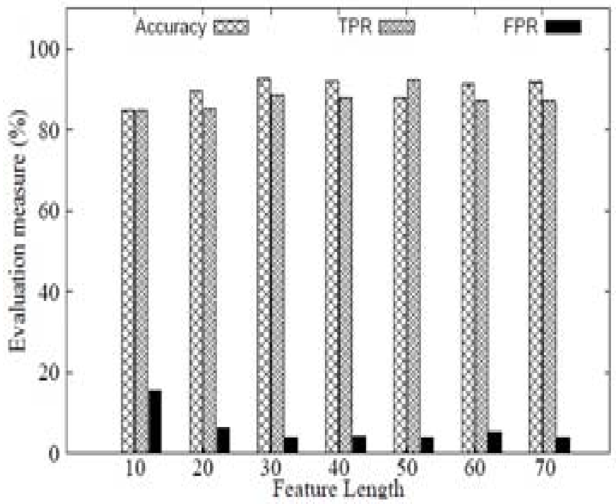


Figure 11. Evaluation measures of BNS scored permissions

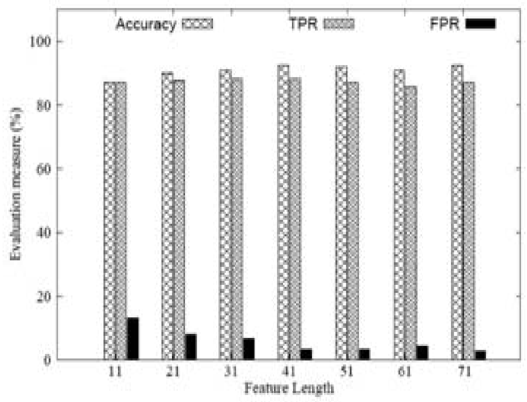


Figure 12. Evaluation measures of OCFS scored permission count

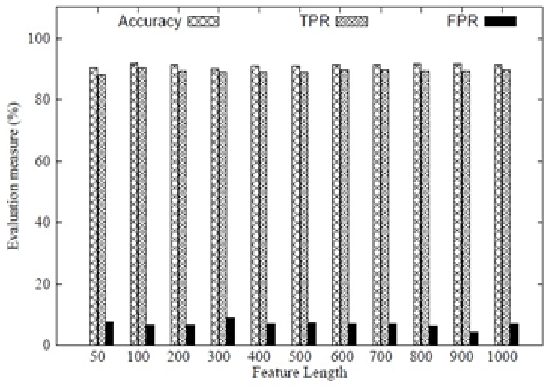


Figure 13. Evaluation measures of BNS scored API calls (frequency features)

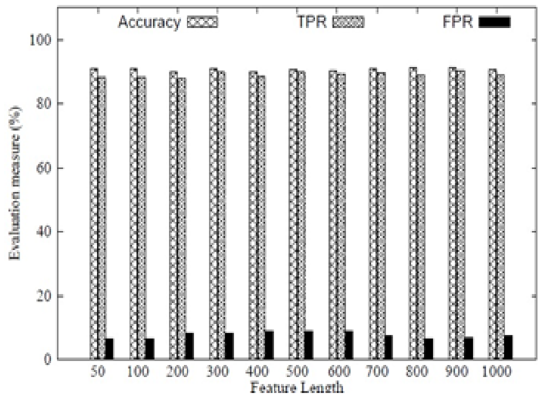


Figure 14. Evaluation measures of BNS scored API calls (Boolean features)

Feature selection	**BNS**	**MI**	**F-CC**	**F-FC**	**FCFF**	**CMFS**	**OCFS**
Evaluation measure / **Features**	$\alpha/(\beta)$	$\alpha/(\beta)$	$\alpha/(\beta)$	$\alpha/(\beta)$	$\alpha/(\beta)$	$\alpha/(\beta)$	$\alpha/(\beta)$
Permissions	**92.51 / (30)**	92.51 / (70)	90.30 / (70)	91.65 / (70)	88.92 / (40)	92.33 / (60)	92.50 / (50)
Permission count	92.34 / (71)	92.34 / (71)	90.30 / (71)	92.50 / (71)	89.77 / (51)	92.16 / (51)	**92.50 / (41)**
API calls	**91.83 / (100)**	91.49 / (1000)	90.11 / (900)	90.46 / (900)	90.80 / (800)	92.84 / (500)	91.48 / (500)
API (Boolean)	**90.81 / (50)**	91.15 / (600)	90.80 / (900)	90.80 / (900)	91.65 / (800)	92.33 / (400)	91.82 / (1000)

Table 3. Accuracies for optimal feature space of individual features for the corresponding feature selection technique represented in the $\alpha/(\beta)$ form, where α represents accuracy attained using Random forest classification algorithm and β depicts feature length

Features	Software Features (7)	Hardware Features (40)
Evaluation measure / Classifiers	Acc.	Acc.
J48	51.19	53.23
Adaboost M1(J48)	51.19	54.08
RF(40) Seed 3	51.04	56.12

Table 4. Accuracies for software and hardware attributes

Features		Feature length	Classifiers		
			ADA	J48	RF
BNS	Permissions	30	89.79	87.92	**92.51**
	API calls	100	89.62	87.07	**91.83**
	API (Boolean)	50	86.22	84.01	**90.81**
OCFS	Permission count	41	87.73	86.54	**92.50**

Table 5. Summary table for accuracies of individual features (results for optimal feature sets are only projected here)

Feature selection	BNS	MI	F-CC	F-FC	FCFF	CMFS	OCFS
Feature length / Classifier	218	1118	1018	1018	898	608	588
J48	87.92	87.41	89.45	87.56	88.41	89.26	87.39
AdaBoost M1 (J48)	90.64	91.83	89.45	92.67	90.46	91.48	92.33
Random Forest	**93.87**	94.04	92.85	93.35	94.37	93.18	94.03

Table 6. Accuracies for ensemble features with frequency of attributes

Feature selection	BNS	MI	F-CC	F-FC	FCFF	CMFS	OCFS
Feature length / Classifier	168	718	1318	1018	898	508	1388
J48	88.26	89.28	89.77	87.73	89.60	88.24	92.33
AdaBoost M1 (J48)	91.15	90.64	91.82	91.99	92.50	92.50	91.14
Random Forest	**93.02**	93.53	92.50	93.52	92.50	94.03	93.35

Table 7. Accuracies for ensemble features with Boolean value of attributes

Attributes					
Permissions	**Permission Count**	**API call (Frequency)**	**API call (Boolean)**	**Ensemble Features (Boolean)**	**Ensemble Features (Frequency)**
1.21×10^{-9} [BNS,30]	1.3×10^{-9} [OCFS, 41]	1.46×10^{-9} [BNS, 100]	1.28×10^{-9} [BNS, 50]	1.33×10^{-9} [BNS,168]	1.43×10^{-9} [BNS, 218]
1.41×10^{-9} [MI, 70]	1.46×10^{-9} [F-CC, 71]	1.55×10^{-9} [CMFS, 500]	1.53×10^{-9} [CMFS, 400]	1.62×10^{-9} [CMFS, 508]	1.56×10^{-9} [FCFF, 898]

Table 8. Processing time (in seconds) of optimal attribute sets of each feature category compared with the feature sets of other feature selection techniques (that exhibit improved accuracy with more features); represented in the form $x[y, z]$; where, x represents processing time for Random forest, z depicts feature space and y gives attribute selection technique

Permissions	Description
`WRITE_EXTERNAL_STORAGE`	Permission for an app to write to the external storage
`READ_PHONE_STATE`	Read only access permission to phone state
`CHANGE_WIFI_STATES`	Permission to change Wi-fi connectivity state.
`WAKE_LOCK`	Using PowerManager WakeLocks to keep processor from sleeping or screen from dimming
`ACCESS_NETWORK_STATE`	To access network information
`RECEIVE_BOOT_COMPLETED`	Allows an application to receive `ACTION_BOOT_COMPLETED` that is broadcasted after booting.
`SEND_SMS`	Allows an application to send SMS
`ACCESS_WIFI_STATE`	Allows the app to access network information
`ACCESS_COARSE_LOCATION`	Permission for the app to access approximate location by means of towers and wi-fi
`ACCESS_FINE_LOCATION`	Permission for the app to access precise location by means of towers and Wi-Fi
`READ_CONTACTS`	Permission to read contact list of the device's user

Table 9. Prominent permissions and their description

5. Inference

In this section we discuss with the inferences made from this work:

1. BNS Permissions are found to be the desired attributes in case of independent features as it gives 92.51% accuracy with 30 features. The functions of an app are based on the permissions requested by it and all illegitimate .apk files need some permission that is different from the benign samples.

2. Ensemble models are found to be better than independent models as they combine the optimal feature space of individual features like permissions, count of permissions, API calls, s/w and h/w features and aggregates the strength of this combination.

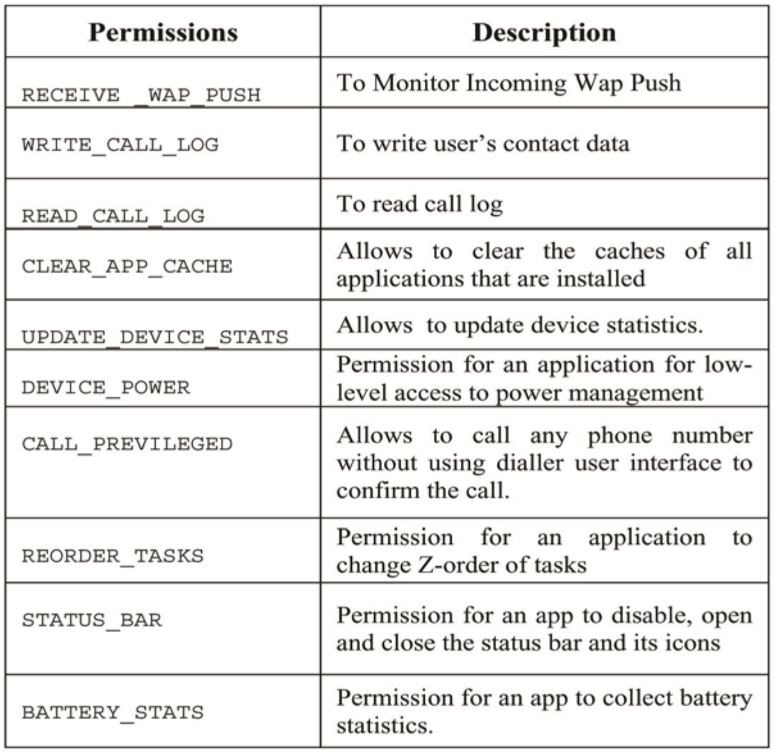
3. Boolean ensemble model with BNS gives 93.02% accuracy with 168 features whereas frequency ensemble framework uses extra 50 attributes to gain 93.87% accuracy.

4. In all the cases, Random Forest classifier performed better than other classifiers as it is an ensemble based learning method. Results from multiple classifiers are aggregated to assign the class label to an unseen sample.

5. J48 classifier gives less accuracy in all the cases as it is a decision based classifier and causes overfitting of training data.

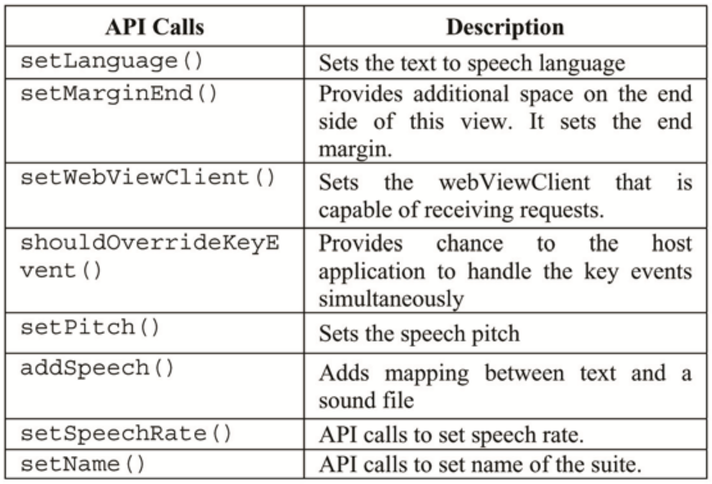
6. Increase in feature length included redundant and irrelevant features that are not necessary for model generation and reduced the classification accuracy. Reducing the feature space beyond certain feature length degraded the performance of our analyzer. The optimal feature sets with precise attributes are selected such that they could represent the features of the entire attribute set.

7. Permissions and API calls assigned with lesser score by the attribute selection methods are filtered off in order to reduce the dimensionality of feature space. In case of BNS, the less scored features are selected for classification by avoiding the high scored attributes



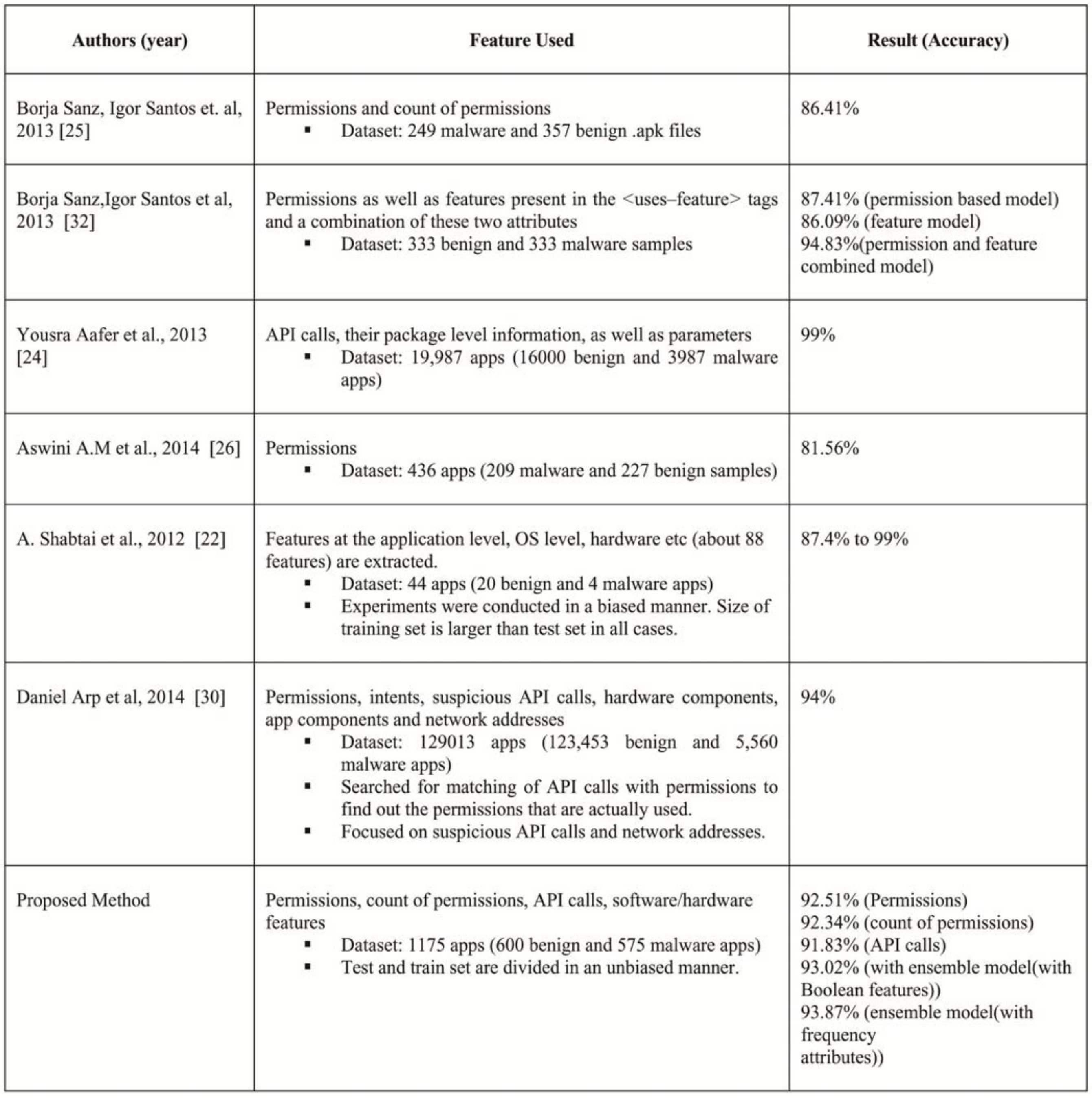
Permissions	Description
RECEIVE _WAP_PUSH	To Monitor Incoming Wap Push
WRITE_CALL_LOG	To write user's contact data
READ_CALL_LOG	To read call log
CLEAR_APP_CACHE	Allows to clear the caches of all applications that are installed
UPDATE_DEVICE_STATS	Allows to update device statistics.
DEVICE_POWER	Permission for an application for low-level access to power management
CALL_PREVILEGED	Allows to call any phone number without using dialler user interface to confirm the call.
REORDER_TASKS	Permission for an application to change Z-order of tasks
STATUS_BAR	Permission for an app to disable, open and close the status bar and its icons
BATTERY_STATS	Permission for an app to collect battery statistics.

Table 10. Trivial permissions and their description



API Calls	Description
setLanguage()	Sets the text to speech language
setMarginEnd()	Provides additional space on the end side of this view. It sets the end margin.
setWebViewClient()	Sets the webViewClient that is capable of receiving requests.
shouldOverrideKeyEvent()	Provides chance to the host application to handle the key events simultaneously
setPitch()	Sets the speech pitch
addSpeech()	Adds mapping between text and a sound file
setSpeechRate()	API calls to set speech rate.
setName()	API calls to set name of the suite.

Table 11. Trivial API's and their description



Authors (year)	Feature Used	Result (Accuracy)
Borja Sanz, Igor Santos et. al, 2013 [25]	Permissions and count of permissions ▪ Dataset: 249 malware and 357 benign .apk files	86.41%
Borja Sanz,Igor Santos et al, 2013 [32]	Permissions as well as features present in the <uses–feature> tags and a combination of these two attributes ▪ Dataset: 333 benign and 333 malware samples	87.41% (permission based model) 86.09% (feature model) 94.83%(permission and feature combined model)
Yousra Aafer et al., 2013 [24]	API calls, their package level information, as well as parameters ▪ Dataset: 19,987 apps (16000 benign and 3987 malware apps)	99%
Aswini A.M et al., 2014 [26]	Permissions ▪ Dataset: 436 apps (209 malware and 227 benign samples)	81.56%
A. Shabtai et al., 2012 [22]	Features at the application level, OS level, hardware etc (about 88 features) are extracted. ▪ Dataset: 44 apps (20 benign and 4 malware apps) ▪ Experiments were conducted in a biased manner. Size of training set is larger than test set in all cases.	87.4% to 99%
Daniel Arp et al, 2014 [30]	Permissions, intents, suspicious API calls, hardware components, app components and network addresses ▪ Dataset: 129013 apps (123,453 benign and 5,560 malware apps) ▪ Searched for matching of API calls with permissions to find out the permissions that are actually used. ▪ Focused on suspicious API calls and network addresses.	94%
Proposed Method	Permissions, count of permissions, API calls, software/hardware features ▪ Dataset: 1175 apps (600 benign and 575 malware apps) ▪ Test and train set are divided in an unbiased manner.	92.51% (Permissions) 92.34% (count of permissions) 91.83% (API calls) 93.02% (with ensemble model(with Boolean features)) 93.87% (ensemble model(with frequency attributes))

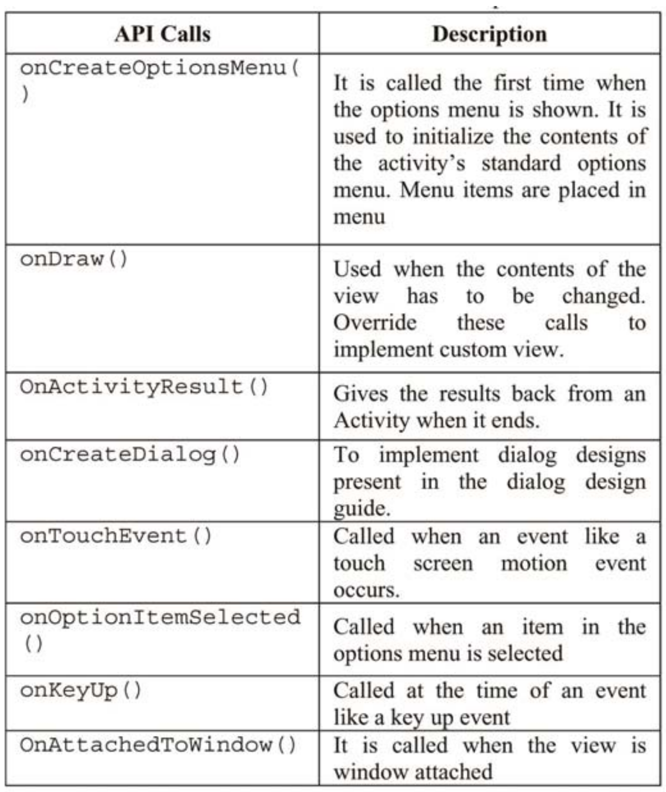
Table 12. Comparison with previous works

6. Comparison with Previous Works

Table 12 shows the comparison of our result with the results of some previous works.

7. Conclusion

We implemented machine learning and dimensionality reduction techniques to perform static analysis of Android malware files. Permissions, count of permissions, software/hardware features from manifest file and API calls are used as attributes to generate the classification model. Results suggest that ensemble model performed better than individual features. Ensemble



API Calls	Description
onCreateOptionsMenu()	It is called the first time when the options menu is shown. It is used to initialize the contents of the activity's standard options menu. Menu items are placed in menu
onDraw()	Used when the contents of the view has to be changed. Override these calls to implement custom view.
OnActivityResult()	Gives the results back from an Activity when it ends.
onCreateDialog()	To implement dialog designs present in the dialog design guide.
onTouchEvent()	Called when an event like a touch screen motion event occurs.
onOptionItemSelected()	Called when an item in the options menu is selected
onKeyUp()	Called at the time of an event like a key up event
OnAttachedToWindow()	It is called when the view is window attached

Table 13. Prominent API's and their description

model aggregates the strength of individual features by combining the optimal attributes of each feature category. These models with Boolean features depict 93.02% accuracy with 168 features and individual model with 30 permissions depict 92.51% accuracy using BNS.

8. Future Scope

Features like Dalvik opcode, Java reflection and Android Manifest attributes can be used individually and as ensemble features to extend this work. In future, the parameters of API calls, suspicious API calls, permissions that are actually used etc. can be selected to improve the performance of our proposed system. Dynamic analysis can be carried out to extract run-time features to generate an expanded framework for malware detection. Thus the proposed system can also be extended to a hybrid model as a future work.

References

[1] Androguard.http://code.google.com/p/androguard/, (accessed September 12, 2013)

[2] Malware application package files download : http://contagiominidump.blogspot.in/2011/07/take-sample-leave-sample-mobile-malware.html, (accessed September 22, 2013)

[3] Application package file (.apk file) format. http://www.file-extensions.org/article/android-apk-file-format-description, (accessed October 5, 2013)

[4] Z-Score Table: http://www.stat.tamu.edu/~lzhou/stat302 /standardnormaltable.pdf, (accessed October 13, 2013)

[5] Symantec Corporation, Internet Security Threat Report 2014:http://www.symantec.com/content/en/us/enterprise/other_resources/b-istr_main_report_v19_21291018.en-us.pdf (accessed June 10, 2014)

[6]F-Secure Labs Mobile Threat Report Q1 2014:http://www.f-secure.com/static/doc/labs_global/Research/Mobile_Threat_Report_Q1_2014.pdf (accessed June 10, 2014)

[7] IDC, International Data Corporation: http://www.idc.com/getdoc.jsp?containerId=prUS24676414, (accessed June 10, 2014)

[8] Google Play Market: https://play.google.com/store?hl=en (accessed September 5, 2013)

[9] Battiti, R. (1994). Using Mutual Information for Selecting Features in Supervised Neural Net Learning, *IEEE Transactions On Neural Networks*, 5 (4), July.

[10]Freund, Y., Schapire, R. E. (1996). Experiments with a new Boosting Algorithm, Machine Learning *In*: Proceedings of the Thirteenth International Conference, 148–156.

[11] Liaw, A., Wiener, M. (2002). Classification and Regression by Random Forest, R News, 8-22, December.

[12] Forman, G. (2003). An Extensive Empirical Study of Feature Selection Metrics for Text Classification. special Issue on Variable and Feature Selection, *Journal of Machine Learning Research*, 3 (Mar), 1289-1305.

[13] Tang, Lei., Liu, Huan. (2005). Bias Analysis in Text Classification for Highly Skewed Data, ICDM, IEEE Computer Society, 781-784.

[14] Forman, G. (2006). BNS Scaling: A Complement to Feature Selection for SVM Text Classification. Hewlett-Packard Labs Tech Report HPL-2006-19.

[15] Tan, Pang-Ning., Steinbach, Michael and Kumar, Vipin, Introduction to Data Mining, First Edition, 2005, Addison-Wesley Longman Publishing Co., Inc., Boston, MA USA

[16] Filiol, E., Jacob, G., Le Liard, M. (2006). Evaluation Methodology and Theoretical Model for Antiviral Behavioural Detection Strategies, WTCV06 Special Issue, G. Bonfante J.-Y. Marion eds, *Journal in Computer Virology*, 2 (4).

[17] Hall, M., Frank, E., Holmes, G.., Pfahringer, B., Reutemann, P., Ian H. Witten. (2009). The WEKA Data Mining Software: An Update, *SIGKDD Explorations*, 11 (1).

[18] Shabtai, A., Elovici, Y. (2010). Applying Behavioral Detection on Android-Based Devices. *In*: The proceedings of Third International Conference, Mobilware, Chicago, IL, USA, June 30 - July 2, 235-249.

[19] Shabtai, A. (2010). Malware Detection on Mobile Devices, *In*: The Proceedings of 11th International Conference on Mobile Data Management.

[20] Felt, A. P., Finifter, M., Chin, E., Hanna, S., Wagner, D. (2011). A Survey of Mobile Malware in the Wild, *In* : the proceedings of SPSM11, October 17.

[21] Heger, D. A. (2011). Mobile Devices - An Introduction to the Android Operating Environment Design, Architecture, and Performance Implications.

[22]Shabtai, A.., Kanonov, U., Elovici, Y., Glezer, C., Weiss, Y. (2012) . Andromaly: A Behavioral Malware Detection Framework for Android Devices, *J. Intell. Inf. Syst*. 38 (1), February.

[23] Dini, G., Martinelli, F., Saracino, A., Sgandurra, D. (2012). MADAM: a Multi-Level Anomaly Detector for Android Malware, *In*: The proceedings of 6th International Conference on Mathematical Methods, Models and Architectures for Computer Network Security, MMM-ACNS 2012, St. Petersburg, Russia, October 17-19, 240-253.

[24] Aafer, Y., Du, W., Yin, H. (2013). Droid APIMiner: Mining API-Level Features for Robust Malware Detection in Android.

[25] Sanz, Borja., Santos, Igor., Laorden, Carlos., Ugarte-Pedrero, Xabier., Bringas, Pablo Garcia., Marañón, Gonzalo Álvarez (2012). PUMA: Permission Usage to Detect Malware in Android. CISIS/ICEUTE/SOCO Special Sessions. 289-298,

[26] Aswini, A M., Vinod, P. (2014). Droid Permission Miner: Mining Prominent Permissions for Android Malware Analysis, *In:* The proceedings of 5*th* IEEE International Conference on the Applications of the Digital Information and Web Technologies (ICADIWT).

[27] Breiman, Leo. (2001). Random Forests, *Machine Learning*, 45 (1), 5-32.

[28] Breiman, Leo. (1996). Bagging Predictors, *Machine Learning*, 24 (2), 123-140.

[29] Huang, Chun-Ying., Tsai, Yi-Ting., Hsu, Chung-Han. Performance Evaluation on Permission-Based Detection for Android Malware, *In*: Proceedings of International Computer Symposium, December.

[30] Arp, Daniel., Spreitzenbarth, Michael., Hubner,Malte., Gascon, Hugo., Rieck, Konrad. (2014). Drebin: Efficient and Explainable Detection of Android Malware in Your Pocket, *In*: Proceedings of 17th Network and Distributed System Security Symposium (NDSS).

[31] Sanz, Borja., Santos, Igor., Ugarte-Pedrero, Xabier., Laorden, Carlos., Nieves, Javier., Bringas, Pablo Garcia. (2013). Instance-based Anomaly Method for Android Malware Detection. *In*: Proceedings of SECRYPT 2013, 387-394.

[32] Sanz, Borja., Santos, Igor., Laorden, Carlos., Ugarte-Pedrero, Xabier., Nieves, Javier., Bringas, Pablo Garcia., Mara, Gonzalo lvarez. (2013). Mama: manifest Analysis for Malware Detection in Android, Cybernetics and Systems, p.469-488.

[33] Aung, Zarni., Zaw, Win. (2013). Permission-Based Android Malware Detection, *In*: *Proceedings of International Journal of Scientific & Technology Research*, 2, 228-234.

[34] Yang, Jieming., Qu, Zhaoyang., Liu, Zhiying. (2014). Improved Feature-Selection Method Considering the Imbalance Problem in Text Categorization, *The Scientific World Journal*, Article ID 625342, 17.

[35] Egghe, Leo., Leydesdorff, Loet. (209). The relation between Pearson's correlation coefficient r and Salton's cosine measure. *JASIST,* 60 (5), 1027-1036.

[36] Zheng, Zhaohui., Wu, Xiaoyun., Srihari, Rohini. (2004). Feature selection for text categorization on imbalanced data, SIGKDD Explor. Newsl. 6, 1, 80-89, June .

[37] Correlation: http://academic.macewan.ca/burok/Stat151/notes/regression.pdf (accessed May 12, 2013)

[38] Yan, Jun., Liu, Ning., Zhang, Benyu., Yan, Shuicheng., Chen, Zheng., Cheng, Qiansheng., Fan, Weiguo., Ma, Wei-Ying (2005). OCFS: optimal orthogonal centroid feature selection for text categorization, *In*: *Proceedings of 28th Annual International ACM SIGIR conference on Research and development in Information Retrieval*, 122-129, January.