Android Malware Analysis Using Ensemble Features

Conference Paper · October 2014

DOI: 10.1007/978-3-319-12060-7_20

CITATIONS
6

READS
796

2 authors:

A. M. Aswini

5 PUBLICATIONS **30** CITATIONS

SEE PROFILE

P. Vinod

SCMS School of Engineering & Technology, Ernakulam, Kerala

56 PUBLICATIONS **369** CITATIONS

SEE PROFILE

Android Malware Analysis Using Ensemble Features

A. M. Aswini and P. Vinod

Department of Computer Science & Engineering,
SCMS School of Engineering & Technology, Ernakulam, Kerala, India 683 582
{aswinimohan95,pvinod21}@gmail.com

Abstract. This paper presents a static feature extraction framework for Android malware analysis. The techniques are implemented by extracting prominent features from the components of Android application package i.e. `AndroidManifest.XML` files. Five different types of features likely permissions, count of permission, hardware features, software features as well as API calls from 1175 .apk files are mined for performing the investigation. The objective of this work is to evaluate if independent features are effective in comparison to ensemble features. Feature reduction is performed to investigate the impact of varied feature length on classification accuracy. Feature selection techniques such as *Bi–Normal Separation*, *Mutual Information*, *Relevancy score*, *Kolmogorov dependence* and *Kullback Leibler* are administered to choose the significant attributes. The proposed method introduced here using dimensionality reduction and machine learning algorithms produces an overall classification accuracy of 93.02% with ensemble features. Comparing the empirical results of ensemble features with individual features, the former improved the classification accuracy with Bi–Normal Separation.

Keywords: Android malware, Ensemble features, Feature selection, Static Analysis.

1 Introduction

Smartphones with complete functionalities of a basic phone are equipped with the additional capabilities like web browsing, Wi–fi, digital media access etc. These gadgets have the ability to incorporate small computer programs called apps that can be used for entertainment as well as to perform many other useful tasks.

Android is an OS based on the Linux kernel primarily designed for touchscreen devices. It is the fastest growing mobile operating system that contributes a world–class platform for the development of applications and games for its users and provide an open marketplace for the distribution of these apps [5].

According to the Symantec Corporation Internet Security Threat Report 2014 [6], popular legitimate applications from the Google Play are downloaded by the attackers and are repackaged with additional code thereby generating

R.S. Chakraborty et al. (Eds.): SPACE 2014, LNCS 8804, pp. 303–318, 2014.

third party apps. Trojans mostly disguised as legitimate applications are a part of these malicious codes injected to mobile apps. Such programs uploaded to the mobile marketplaces are downloaded and installed by the users unaware of its maliciousness.

The compromised smartphones are vulnerable to threats like stealing user credentials, stack based buffer overflow resulting in arbitrary code execution, activating unknown services in the device without user's knowledge, denial of service attacks etc. Some forms of attacks exercised by the malware authors are execution of code using Android debugger bridge (adb), cross site scripting for redirecting to vulnerable domains and memory corruption for gaining root privileges. The desktop security solutions in Antivirus based on signature generation cannot detect zero–day malware attacks. These techniques are not completely scalable for smartphones as it require more memory and processing power [13].

Due to these above mentioned drawbacks of the existing detection system, we perform static analysis implementing dimensionality reduction and machine learning algorithms for Android malware analysis. We extract permissions, software/hardware features and Application Programming Interface (API) calls that are significant for mobile malware identification. The contributions of this work are the following:

- Employed attribute ranking methods like *Bi–Normal Separation*, *Mutual Information*, *Relevancy score*, *Kolmogorov dependence* and *Kullback Leibler* to mine precise attributes for classification.
- The most prominent attributes that contribute to the characterization of mobile malware can be determined.
- Optimal feature length, best classifier as well as attribute selection method are found out using this detection mechanism.
- An accuracy of 93.02% is achieved using ensemble features with Bi–Normal Separation feature selection.

The remaining sections are organized as follows: Section 2 includes the related works. Section 3 explains the proposed methodology. Section 4 contains the experiment carried out followed by the results and findings. Section 5 discusses about the inference and finally the conclusion and future work is presented in Section 6.

2 Related Works

The authors [20] used permissions to detect malicious apps in Android OS. A total of 124,769 benign and 480 malicious files were used in their work. The requested and required permissions, number of required permissions, normal, signature, dangerous permissions, number of files with .so extension, number of elf files, count of executable, shared objects were considered as the features. Results showed that a permission–based detector can detect more than 81% of malicious samples.

The authors [21] proposed a machine learning based malware detector to discriminate normal and malware applications. The features used were permissions

mined from 200 .apk samples. The generated models were trained and evaluated using the Area Under ROC Curve (AUC). They obtained an accuracy of 91.58% using Random Forest classifier.

PUMA [22] detects malicious Android apps by extracting permissions. They collected 239 Android malware samples, and obtained a 0.92 of AUC using the Random forest classifier. Except the Bayesian–based classifiers, the methods achieved accuracy rates higher than 80%. The best classifier was Random Forest trained with 50 trees with an accuracy of 86.41%.

This [23] approach used the <uses–permission> and the <uses–feature> tags present in the manifest file. Manhattan, Euclidean and Cosine distance were applied and obtained AUC of 0.88 using Manhattan distance with average as the combination rule (85% accuracy). Using Euclidean distance, they obtained more than 0.90 of AUC with 87.57% of accuracy. The best results of 0.91 of AUC and nearly 90% of accuracy was obtained using Cosine similarity.

MAMA [24] used permissions and feature tags within the manifest file. The best results are obtained with Random Forest, using 100 trees, achieving an accuracy of 87% and an AUC of 0.95 for malware detection.

DREBIN [26] performs static analysis by extracting maximum possible number of features of an app's code from manifest file. The features are grouped in sets of strings (such as permissions, API calls and network addresses) and are embedded in a joint vector space. About 123,453 applications and 5,560 malware samples are used for the investigation and it detects 94% of the malware with relatively less false alarms.

Authors in [16], [17], [19] devised a supervised anomaly detector named Andromaly to extract 88 prominent features. Detection rates were better for the database with benign games than benign tools when used in combination with the 4 malicious apps. The NB and Logistic Regression were found to be the better classifiers.

DroidAPIMiner in [25] was used to extract API calls using a modified Androguard tool and different classifiers were evaluated using the set of features. They achieved an accuracy of 99% and false positive rate of 2.2% using k–NN classifier.

In [27] the authors presented a static analyzer, Droid Permission Miner, that mines prominent permissions present in the .apk files. Feature selection techniques like *Bi–Normal Separation (BNS)* and *Mutual Information (MI)* were used in their work and obtained an accuracy of 81.56% with 15 features proving *MI* to be better method.

3 Proposed Methodology

This section deals with our static framework for Android malware detection that implements machine learning techniques. Different features that belong to distinct feature categories are extracted and dimensionality reduction is employed to prune the feature space. Androguard [1] is used for mining permissions, count of permissions, software/hardware features and API calls for identifying malicious apps. The permissions, software/hardware features and permission count

are mined from the Android Manifest File. However, the API calls from each .apk files are also extracted using the Androguard tool. The individual features are used for classification in the first phase followed by the experimentation with ensemble features. The architecture for our proposed model is shown in Figure 1 for individual features and Figure 2 for ensemble model. These models are briefly described in the following subsections.

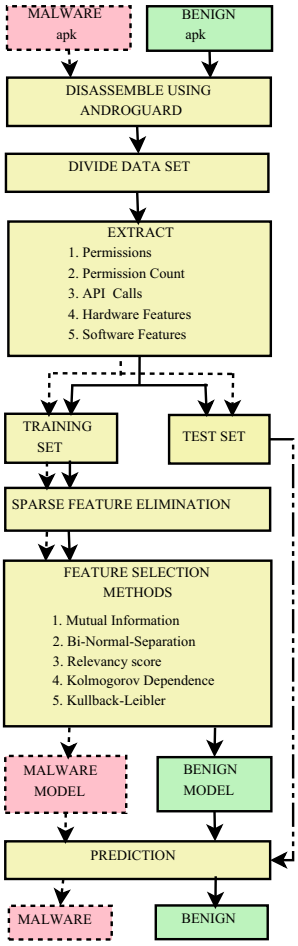

Fig. 1. Individual feature model

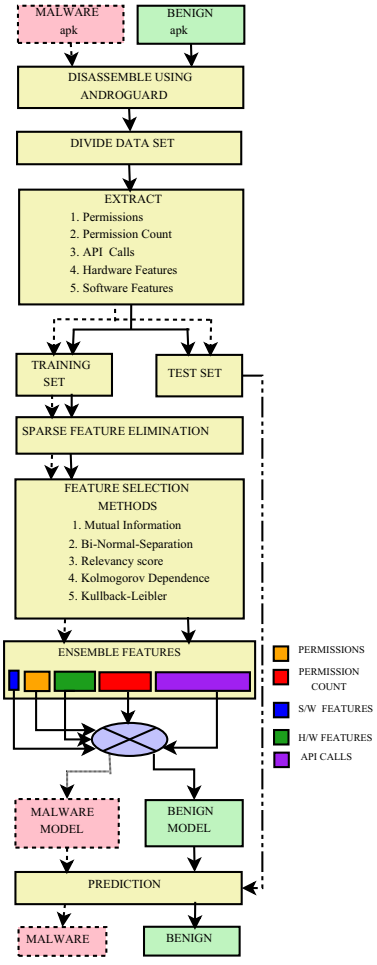

Fig. 2. Ensemble model

3.1 Dataset Preparation

Dataset is prepared using 1175 .apk files comprising of 575 malicious samples collected from Contagiodump [2] and from different user agencies. Also, 600

benign applications were downloaded from various publicly available Internet sources. The benign samples are divided such that 300 files are included in the test set and the other 300 samples are allocated to train set. From the 575 illegitimate samples, 287 .apk files are included in the test set and the remaining 288 files are added to the train set.

3.2 Feature Categories

The individual feature model as well as ensemble model generation requires features that are extracted from five distinct categories. Different categories of features are listed below.

- *Permissions:* The activities of an application depends upon the permissions requested by it. It is declared statically and there is no provision to declare it dynamically. Android architecture provides a well framed permission mechanism to provide security.
- *Permission count:* This feature set is generated by computing the number of permissions requested by an application.
- *Hardware features:* These are the features required by the app for its execution. It provides information about the set of hardware features on which the application depends.
- *Software features:* These are the software features required by the application for its execution.
- *API calls:* The application programming interface calls are invoked at the execution time to perform some specific tasks.

3.3 Feature Extraction

The Android .apk files [18] are provided as input to the disassembler tool Androguard. These files are initially in the binary format. The .xml files are human readable Manifest files generated from the input .apk files using python script `androaxml.py`. The permissions (within <uses-permission> tag) as well as s/w and h/w features (within <uses-feature> tag) are extracted from these xml files. Similarly, the python script `androapkinfo.py` is used to mine the API calls of the samples. After extracting permissions from each .apk file, the number of permissions requested by each file is determined. The count of permissions existing in a file is considered as another feature for generating the classification model.

3.4 Pre–processing Phase

In this phase, feature pruning is carried out to eliminate the attributes that results in misclassification. After removing the irrelevant attributes, features common to both the classes ($M \cap B$) are considered. Common features are given high precedence over other category of attributes such as union of malware and benign features ($M \cup B$), discriminant benign and discriminant malware features as they are reported to be insignificant for the detection of malicious samples [27].

3.5 Feature Selection Techniques

Feature selection is exercised to synthesize the input data into convenient size so as to extract a subset of k prominent features from a set of n features (large feature space). The techniques discussed below implements dimensionality reduction to exclude redundant features. The selected attributes are used to fabricate the classification model to predict unknown samples. Equations of the five feature selection techniques used in this work are discussed in Table 1.

Table 1. Feature selection techniques

Method	Equation	Description
BNS [10] [11] [12]	$BNS = \lvert F^{-1}(tpr) - F^{-1}(fpr)\rvert$ $TruePositiveRate = \frac{(TP)}{(TP+FN)}$ $FalsePositiveRate = \frac{(FP)}{(TN+FP)}$	Selects positive and negative features and is not biased to any class. Its value is determined from the statistical table for Z–Score [4].F^{-1} is the inverse cumulative probability function of standard normal distribution
MI [8]	$MI(f,C) = \sum_{C\in\{M,B\}} \sum_{f} P(f,C) log\left(\frac{P(f,C)}{P(f)P(C)}\right)$	It gives the extent to which an attribute f reduces the uncertainty in determining the appropriate class *C*. *P(f,C)* is the joint probability distribution, *P(f)* and *P(C)* are the marginal probability distributions of variables f and *C*.
RS [7]	$RS(t_k, C_i) = log\left(\frac{(P(t_k\lvert C_i)+d)}{(P(\bar{t}_k\lvert\bar{C}_i)+d)}\right)$	It is based on the conditional probabilities of a feature in the training set. $P(t_k\lvert C_i)$ is the presence of feature t_k in class C_i ,$P(\bar{t}_k\lvert\bar{C}_i)$ is the absence of feature t_k in class C_i and d is the number of samples with feature t_k in class C_i.
KO [14]	$KO(f) = \sum_{j=1}^{\lvert C\rvert} p(f)\ (p(f\lvert C_j) - p(f\lvert\bar{C}_j))$	This method scores each feature f depending on its relation with the classes C_j, $\bar{C}_j$. $P(f\lvert C_j)$ is the presence of feature f in class C_j, $P(f\lvert\bar{C}_j)$ is the presence of feature f in class $\bar{C}_j$, $\lvert C\rvert$ is the total number of classes and $P(f)$ is the probability of feature *f*.
KL [14]	$KL(f) = -P(f\lvert M) log\left(\frac{P(f)}{P(f\lvert M)}\right) - P(f\lvert B) log\left(\frac{P(f)}{P(f\lvert B)}\right)$	$P(f\lvert M)$ is the presence of feature f in class M (malware), $P(f\lvert B)$ is the presence of feature f in class B(benign) and $P(f)$ is the probability of an attribute *f*.

These techniques are applied to the top 78 common permissions and prominent 2166 APIs common to both malware and benign train set to reduce the feature space. The common features ($M \cap B$) are arranged in the decreasing order of their BNS, MI, RS, KO and KL scores respectively. The top BNS scored attributes are not used as it does not provide better accuracy according to the previous work [27]. Hence, we considered bottom BNS and top MI, RS, KO and KL attributes with diverse feature lengths.

3.6 Ensemble Features

This feature space is generated by combining the optimal feature sets of five individual categories of feature (Permissions, count of permissions, s/w and h/w features and API calls). The top ranked features are combined to create ensemble feature space to improve the classification accuracy. The ensemble features are constructed by discarding all extraneous attributes (refer Fig.2).

3.7 Classification

The malware and benign models are developed using classifiers (AdaBoostM1 with J48 as base classifier (ADA) [28], Random Forest (RF)[9] [No: of Trees= 40, seed=3] and J48) implemented in WEKA [15]. Unknown samples are predicted using these learned models.

3.8 Evaluation Parameters

Accuracy [30] gives the degree of correctness of a model in classifying the test samples. Here, FP gives the misclassification of benign samples, TP indicate correctly classified malware instances, FN represents wrongly classified malware samples and TP denotes correctly classified malware files.

$$Accuracy = (TP + TN)/(TP + TN + FP + FN) \tag{1}$$

The investigations carried out in this work are listed below:

1. *Determination of robust feature selection method.*
2. *Evaluate the optimal feature vector length.*
3. *Estimation of best feature category (permissions, permission count, software/ hardware features or API calls).*
4. *Determination of the best classifier that reduced misclassification.*
5. *Comparative study of ensemble and individual features.*

4 Experiments and Findings

The experiments are performed on a computer with Ubuntu 12.04 OS, Intel core i3 CPU and 4GB RAM. The two phases involve (1) considering independent attributes and (2) use of ensemble features. The experiments are carried out in two ways; using test/train set and cross validation.

4.1 Evaluation Based on Train and Test Set

Performance Evaluation with Independent Features. The models for classification are created by considering the frequencies of attributes in each file, and individually recording the presence/absence of an attribute (known as Boolean features). In each sample, the permissions and software/hardware features are declared only once so their presence/absence (0 or 1) are considered in the feature vector. In case of API call, the investigation is carried out in two ways (presence/absence and frequency).

Boolean Feature Vector Table (FVT): From the training samples, a total of 195 and 109 unique permissions are obtained respectively from benign and malware files and 78 common permissions are obtained from these unique lists. These 78 permissions are arranged based on their BNS, MI, RS, KO and KL scores in descending order. BNS features are selected from the bottom and top MI, RS, KO and KL scored attributes are considered. For KO, 34 benign as well as 44 malware prominent features are obtained from the 78 common permissions. Classification models are generated using top 10, 20 $\cdots$ 70 ranked permissions based on the five selection methods. The same activity is carried out with the permission count for the training samples. A total of 7,174 and 29,765 unique API calls are obtained for malware and benign apps. During the pre–processing phase, 50% of infrequent APIs are eliminated thus reducing the feature space to 14,882 benign and 3587 malware APIs respectively. From this pruned feature set, 2166 common APIs are determined. After implementing Kolmogorov dependence (KO), out of 2166 api calls, 786 malware and 1378 benign prominent features are obtained. Classification models are generated using significant API calls (scored on the basis of five feature selection methods), specifying their presence/absence in a sample for variable feature length (i.e. 50, 100, 200 $\cdots$ 1000). The 40 h/w and 7 s/w attributes obtained are used without reducing their feature space.

Table 2. Accuracies for BNS, MI, RS and KL scored permissions

Feature selection Method	**BNS**			**MI**			**RS**			**KL**		
Classifier / Feature Length	J48	ADA	RF	J48	ADA	RF	J48	ADA	RF	J48	ADA	RF
10	82.14	82.31	84.69	84.69	85.03	85.03	83.84	82.31	83.33	85.20	85.71	85.20
20	85.03	85.37	89.62	86.05	87.24	87.58	85.37	86.90	88.77	82.14	86.56	87.07
30	87.92	89.79	**92.51**	84.69	87.24	87.58	88.77	86.56	89.96	83.50	86.73	88.09
40	86.90	88.26	92.00	84.35	88.09	91.49	87.92	88.94	90.98	84.86	88.09	90.98
50	87.24	89.11	92.17	86.56	89.62	91.66	87.07	90.47	91.32	86.56	88.94	92.51
60	87.41	89.28	91.15	87.58	88.94	92	87.41	88.60	91.66	87.58	88.94	92
70	87.41	89.11	91.83	87.41	89.29	92.51	87.41	89.11	91.83	87.41	89.28	92.51

The results of the above experiments are reviewed here. The 30 BNS permission feature resulted in higher accuracy using Random Forest (i.e. 92.51%). It is observed that the classification model generated using MI, RS, KO and KL features does not identify malicious apps effectively as it uses more number of permissions than BNS for improved performance (refer Table 2). Similar experiment is performed for the count of permissions. The optimal feature length is observed to be 71 with an accuracy of 92.34% using BNS/MI/KL with Random Forest (refer Table 3). For API calls, the Boolean features (with feature length of 50) provided improved accuracy of 90.81% using BNS (refer Table 4). The results obtained using the software and hardware features without implementing feature selection technique are shown in Table 6. The 40 h/w and 7 s/w features depicted less accuracy (56.12 and 52.04% respectively).

Due to lack of space, the accuracies for Kolmogorov dependence (KO) with feature lengths that are found to be optimal are only projected in Table 6.

Table 3. Accuracies for BNS, MI, RS and KL scored permission count

Feature selection Method	BNS			MI			RS			KL		
Classifier / Feature Length	J48	ADA	RF	J48	ADA	RF	J48	ADA	RF	J48	ADA	RF
11	84.69	86.90	87.07	86.22	87.41	88.09	85.54	84.01	86.05	86.22	87.07	86.90
21	85.20	87.92	90.81	86.05	87.75	89.62	86.39	87.75	90.13	84.35	87.07	89.96
31	87.41	88.26	91.15	84.69	88.77	89.96	88.26	89.28	90.81	84.35	87.41	90.30
41	86.90	89.11	91.83	84.35	88.77	91.83	87.92	88.94	91.32	84.86	87.41	91.66
51	87.24	89.11	91.83	86.56	87.92	91.83	87.07	89.28	91.83	86.56	89.62	91.34
61	87.41	88.43	92.17	87.58	87.58	92.17	87.41	88.77	90.98	87.58	87.58	92.17
71	87.41	88.43	**92.34**	87.41	88.43	**92.34**	87.41	88.43	92.17	87.41	88.43	**92.34**

Table 4. Accuracies for BNS, MI, RS and KL scored API calls (Boolean features)

Feature selection Method	BNS			MI			RS			KL		
Classifier / Feature Length	J48	ADA	RF	J48	ADA	RF	J48	ADA	RF	J48	ADA	RF
50	84.01	86.22	**90.81**	83.50	82.99	82.48	83.84	85.54	88.94	83.50	82.82	82.99
100	87.24	88.77	90.81	85.54	86.73	88.60	86.39	89.28	91.49	85.30	85.54	87.41
200	87.75	89.45	89.79	85.71	89.79	88.26	87.58	89.28	91.32	85.71	87.24	88.43
300	86.05	89.79	90.81	85.71	90.64	89.79	87.07	90.47	90.47	85.71	88.09	88.09
400	86.05	90.81	89.79	85.54	87.58	90.47	86.73	88.94	91.32	86.05	88.43	88.77
500	86.90	89.79	90.47	87.41	90.30	90.81	86.73	89.79	92.34	86.22	88.77	89.79
600	87.75	90.81	90.13	87.41	90.47	91.15	87.07	90.13	91.15	86.22	88.60	89.45
700	89.11	88.77	90.98	86.22	89.11	90.47	87.07	88.09	91.32	86.22	89.45	89.62
800	88.94	90.13	91.15	87.75	90.81	90.47	87.07	88.77	90.98	87.41	89.11	89.28
900	88.77	90.64	91.15	87.75	90.47	90.81	88.09	91.15	91.49	87.41	89.11	90.47
1000	88.77	90.64	90.64	87.75	90.30	90.47	88.09	91.15	90.47	87.41	89.28	90.30

Frequency FVT: For API calls, the classification model is generated using the frequencies of API in the samples. BNS gives an accuracy of 91.83% with 100 features using Random forest. A minor increase in accuracy is attained using Relevancy Score (Acc. 92.51% with 400 features) but MI, KO and KL shows less performance (refer Table 5 and Table 6). Summarizing the results for independent features, BNS is better for every feature categories as it uses less attributes for classification. Permissions give 92.51% accuracy compared to API calls and permission count (refer Tables 2-6). For all the cases, Random Forest gives better results.

Table 5. Accuracies for BNS, MI, RS and KL scored API calls (Frequency features)

Feature selection Method	BNS			MI			RS			KL		
Classifier / Feature Length	J48	ADA	RF	J48	ADA	RF	J48	ADA	RF	J48	ADA	RF
50	84.69	85.20	90.13	83.50	81.46	82.48	87.58	88.60	90.64	83.50	83.50	82.65
100	87.07	89.62	**91.83**	83.84	86.56	88.94	86.22	89.45	90.47	85.03	86.05	86.90
200	87.24	89.45	91.35	85.37	88.94	90.47	87.58	88.60	90.81	85.54	88.43	88.77
300	86.22	89.28	89.96	85.54	89.62	90.13	88.60	91.49	91.49	85.54	88.09	88.09
400	86.22	88.77	90.98	86.05	90.30	89.79	88.09	91.15	92.51	84.69	85.71	88.77
500	86.39	89.28	91.81	84.86	89.79	90.81	88.09	89.79	92	86.05	89.11	90.30
600	86.73	89.79	91.32	84.86	90.13	90.47	88.09	90.64	90.98	86.05	88.60	90.13
700	86.73	90.81	91.32	87.92	89.45	90.98	88.26	91.15	90.98	86.05	87.92	90.13
800	87.75	90.64	91.66	88.43	90.13	90.81	88.26	91.66	91.32	87.24	88.94	89.28
900	87.41	90.81	91.68	88.26	90.64	90.30	87.41	89.96	91.32	87.24	88.43	90.13
1000	87.41	90.30	91.32	88.26	90.13	91.49	87.41	89.96	91.66	87.24	89.62	90.47

Table 6. Accuracies for software/hardware features and KO features (API calls, permissions and permission count) for benign and malware feature lists (projected the accuracies only for optimal feature lengths of corresponding feature categories due to lack of space)

Features	s/w	h/w	API calls				Permissions & count			
			Boolean		Frequency		Permissions		Permission count	
			Benign	Malware	Benign	Malware	Benign	Malware	Benign	Malware
Feature Length / Classifier	7	40	1000	200	200	600	30	30	35	45
J48	51.19	53.23	83.50	88.26	84.35	85.03	67.68	84.69	82.82	85.88
Adaboost MI (J48)	51.19	54.08	86.90	89.45	87.24	87.58	68.53	84.18	85.37	87.24
Random Forest	51.04	56.12	89.11	90.13	90.64	90.13	70.40	88.09	86.90	89.79

Performance Evaluation with Ensemble Features

Two ensemble models are generated for the five feature selection techniques using (1) the frequencies of prominent API in each file and (2) considering the Boolean value of APIs with the presence/absence of other four categories of features. From Table 7, ensemble model fabricated using frequency with BNS gives an accuracy of 93.87% (for 218 features) with RF classifier. MI (Acc. 94.04%), RS (Acc. 93.87%), KO (93.36% with Benign prominent feature set) and KL(93.53%) depict similar accuracy but employs 1118, 518, 292 and 598 features respectively. The ensemble model designed by Boolean values in FVT of API calls, permissions, count of permissions and software/hardware features is found to be 93.02% with BNS (for 168 features). MI(Acc. 93.53%), RS(Acc. 93.84%), KO(92.85% with malware prominent feature set) and KL(94.21%) have improved accuracies using 718, 618, 292 and 998 features respectively. These two observations indicates that that the ensemble model constructed by employing the Boolean features with BNS provide higher accuracy with 168 features.

Permissions and API calls that are rarely and widely used by malicious and legitimate samples with their descriptions are given in the Appendix A (Tables 10-13).

4.2 Cross–Validation

Cross–validation [29] is implemented to predict the accuracy of a learning model. This approach is significant in the cases where the size of the learning data is very small or when the model is generated with large number of attributes. For a dataset of N specimens, k-fold cross–validation (also known as rotation-estimation) splits the dataset into k mutually exclusive subsets and testing/training are performed k times. In order to estimate the accuracy of a classifier, we performed 10-fold cross–validation (refer Table 8 for the results). In case of KO, 'M' represents malware features and 'B' represents benign attributes .

Table 7. Accuracies for Ensemble features (Boolean and Frequency of features)

Model	Ensemble features (Frequency)					Ensemble features (Boolean)				
Feature selection method	BNS	MI	RS	KO	KL	BNS	MI	RS	KO	KL
Feature Length / Classifier	218	1118	518	292	598	**168**	718	618	292	998
J48	87.92	87.41	87.92	87.75	88.94	88.26	89.28	89.64	88.77	87.92
Adaboost M1(J48)	90.64	91.83	91.49	93.19	90.30	91.15	90.64	91	91.32	93.02
RF (40) Seed 3	93.87	94.04	93.87	93.36	93.53	**93.02**	93.53	93.84	92.85	94.21

Table 8. Accuracies attained by Random forest classifier after cross validation with prominent features of individual feature categories and ensemble features for five feature selection methods; Represented in the form $\alpha/(\beta)$; where, (α) represents accuracy and (β) represents feature space

Selection techniques	Features					
	Permissions	Permission count	API Calls (Frequency)	API Calls (Boolean)	Ensemble (Frequency)	Ensemble (Boolean)
BNS	91.14/(30)	91.23/(71)	90.97/(100)	89.53/(50)	93.53/(218)	92.51/(168)
MI	91.40/(70)	91.65/(71)	91.74/(1000)	90.80/(600)	94.04(1118)	94.12/(718)
RS	91.23/(70)	91.57/(71)	91.82/(400)	92.08/(500)	93.95/(518)	93.36/(618)
KL	91.57/(50)	91.65/(71)	90.72/(500)	91.40/(900)	93.95/(598)	93.61/(998)
KO	86.72/(30,M)	87.82/(45,M)	90.97/(200,B)	89.02/(200,M)	91.23/(292)	92.51/(292)

4.3 Processing Time

The time consumed by prominent BNS features for processing are computed in seconds (secs). This is compared with the time taken by the prominent attribute sets of other feature selection techniques that give improved accuracies with increased feature space (refer Table 9).

Table 9. Processing time (in secs) of prominent BNS features compared with the attribute sets of other feature selection techniques (that exhibit improved accuracy with more features); represented in the form $\delta(\beta, \gamma)$; where, (δ) represents processing time for Random forest, (β) depicts feature space and (γ) gives attribute selection technique

Classification approach	Attributes				
	Permissions	API Calls (Frequency)	API Calls (Boolean)	Ensemble (Frequency)	Ensemble (Boolean)
Test/train set	1.21×10^{-9}[30,BNS]/ 1.45×10^{-9}[50,KL]	1.46×10^{-9}[100,BNS]/ 2.3×10^{-9}[400,RS]	1.28×10^{-9}[50,BNS]/ 2.15×10^{-9}[500,RS]	1.43×10^{-9}[218,BNS]/ 1.77×10^{-9}[1118,MI]	1.33×10^{-9}[168,BNS]/ 1.68×10^{-9}[718,MI]
Cross validation	0.33[30,BNS]/ 0.46[50,KL]	0.52[100,BNS]/ 1.07[400,RS]	0.39[50,BNS]/ 1.38[500,RS]	0.61[218,BNS]/ 2.23[1118,MI]	0.57[168,BNS]/ 1.63[718,MI]

5 Inference

The following are the inferences made from this work:

- For independent features, permissions are found to be the desired attributes as the accuracy attained with BNS is 92.51% (with a small feature length of 30). The functioning of an app is based on the permissions requested by it and all malicious apps need some permissions that are different from the benign .apk files.
- Ensemble models combine the optimal feature space of individual features and gains the strength of this combination. So these models are better than the models built using individual features.
- BNS assigns higher rank to an attribute in comparison with MI, RS, KO and KL (refer Appendix B, Figure.3 and Figure.4).
- Random Forest being an ensemble based learning method aggregated the results from multiple classifiers and performed better in all cases.

- Increase in feature length included the features that are not apt for model generation and reduced the classification accuracy.
- In *PUMA* [22], permissions and count of permissions are used as features to attain 86.41% accuracy. *MAMA* [24] extracted permissions as well as features present in the uses–feature tags and obtained best results with Random Forest (accuracy of 87%). *Droid Permission Miner* [27], extracted permissions and implemented Bi–Normal Separation (BNS) and Mutual Information (MI) to obtain an accuracy of 81.56% (with MI using 15 features). Our work with five feature categories and ensemble features show reasonable performance when compared with [22], [24] and [27]. Here, the accuracy attained is 92.51% with 30 permissions. Using the count of permissions, the accuracy is 92.34%. With 100 API calls, the accuracy is 91.83% for BNS with Random forest classifier. The accuracies of the proposed ensemble model using BNS is 93.02% (with Boolean features using 168 features) and 93.87% (with frequency attributes using 218 features) with Random Forest classifier.

6 Conclusion and Future Scope

We presented a static malware analysis framework using permissions, count of permissions, software/hardware features and API calls by implementing machine learning algorithms. The ensemble model performed better compared to the individual model. In this work, BNS synthesized precise features that improved the classification accuracy. The accuracy for ensemble model with Boolean features is 93.02% with 168 features and individual model with 30 permissions are 92.51% using BNS. Thus, our proposed method can be used for the initial classification of .apk samples with reduced false alarms. In future, features like Dalvik opcode, Java reflection and Android Manifest attributes can also be used individually and as ensemble features. Also, data flow and analysis on API call with and without parameters can be used as in [25] to improve the applicability of our implemented scheme.

Acknowledgements: We would like to thank all the anonymous reviewers for their valuable suggestions that assisted us to enhance the quality of our work.

References

1. Androguard, `http://code.google.com/p/androguard/` (accessed September 12, 2013)
2. Malware apk, `http://contagiominidump.blogspot.in/2011/07/take-sample-leave-sample-mobile-malware.html` (accessed September 22, 2013)
3. 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. Android Developers, `http://developer.android.com/about/index.html` (accessed March 10, 2014)
6. Symantec Corporation, Internet Security Threat Report 2014, vol. 19 (2014)

7. Feature selection, Ling, Fei Xia, http://courses.washington.edu/ling572/winter2013/slides/class7_feature_selection.pdf (accessed April 9, 2014)
8. Battiti, R.: Using Mutual Information for Selecting Features in Supervised Neural Net Learning. IEEE Transactions oN Neural Networks 5(4) (1994)
9. Liaw, A., Wiener, M.: Classification and Regression by Random Forest, 18–22 (December 2002)
10. Forman, G.: An Extensive Empirical Study of Feature Selection Metrics for Text Classification, Special Issue on Variable and Feature Selection. Journal of Machine Learning Research, 1289–1305 (2003)
11. Tang, L., Liu, H.: Bias Analysis in Text Classification for Highly Skewed Data. In: ICDM, pp. 781–784. IEEE Computer Society (2005)
12. Forman, G.: BNS Scaling: A Complement to Feature Selection for SVM Text Classification In Hewlett-Packard Labs Tech Report HPL-2006-19 (2006)
13. Filiol, E., Jacob, G., Le Liard, M.: Evaluation Methodology and Theoretical Model for Antiviral Behavioural Detection Strategies. Journal in Computer Virology (2006); WTCV 2006 Special Issue, Bonfante, G., Marion, J.-Y. (eds.)
14. Bonev, B.I.: Feature Selection based on Information Theory, http://www.dccia.ua.es/~boyan/papers/TesisBoyan.pdf (accessed May 10, 2014)
15. Frank, E., Hall, M.A., Holmes, G., Kirkby, R., Pfahringer, B.: WEKA - A Machine Learning Workbench for Data Mining. In: The Data Mining and Knowledge Discovery Handbook, pp. 1305–1314 (2005)
16. Shabtai, A., Elovici, Y.: Applying Behavioral Detection on Android-Based Devices. In: Cai, Y., Magedanz, T., Li, M., Xia, J., Giannelli, C. (eds.) Mobilware 2010. LNICST, vol. 48, pp. 235–249. Springer, Heidelberg (2010)
17. Shabtai, A.: Malware Detection on Mobile Devices. In: 11th International Conference on Mobile Data Management (2010)
18. Heger, D.A.: Mobile Devices - An Introduction to the Android Operating Environment Design, Architecture, and Performance Implications (2011)
19. Shabtai, A., Kanonov, U., Elovici, Y., Glezer, C., Weiss, Y.: Andromaly: A Behavioral Malware Detection Framework for Android Devices. J. Intell. Inf. Syst. (2012)
20. Huang, C.-Y., Tsai, Y.-T., Hsu, C.-H.: Performance Evaluation on Permission-Based Detection for Android Malware. In: Pan, J.-S., Yang, C.-N., Lin, C.-C. (eds.) Advances in Intelligent Systems & Applications. SIST, vol. 21, pp. 111–120. Springer, Heidelberg (2012)
21. Aung, Z., Zaw, W.: Permission-Based Android Malware Detection. International Journal of Scientific & Technology Research 2, 228–234 (2013)
22. Sanz, B., Santos, I., Laorden, C., Ugarte-Pedrero, X., Bringas, P.G., Álvarez, G.: PUMA: Permission Usage to Detect Malware in Android. In: Herrero, Á., et al. (eds.) Int. Joint Conf. CISIS'12-ICEUTE'12-SOCO'12. AISC, vol. 189, pp. 289–298. Springer, Heidelberg (2013)
23. Sanz, B., Santos, I., Ugarte-Pedrero, X., Laorden, C., Nieves, J., Bringas, P.G.: Instance-based Anomaly Method for Android Malware Detection. In: SECRYPT 2013, pp. 387–394 (2013)
24. Sanz, B., Santos, I., Laorden, C., Ugarte-Pedrero, X., Nieves, J., Bringas, P.G., Álvarez, G.: Mama: manifest Analysis for Malware Detection in Android. In: Cybernetics and Systems, pp. 469–488 (2013)

25. Aafer, Y., Du, W., Yin, H.: DroidAPIMiner: Mining API-Level Features for Robust Malware Detection in Android. In: Zia, T., Zomaya, A., Varadharajan, V., Mao, M. (eds.) SecureComm 2013. LNICST, vol. 127, pp. 86–103. Springer, Heidelberg (2013)
26. Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K.: Drebin: Efficient and Explainable Detection of Android Malware in Your Pocket. In: 17th Network and Distributed System Security Symposium (NDSS) (February 2014)
27. Aswini, A.M., Vinod, P.: Droid Permission Miner: Mining Prominent Permissions for Android Malware Analysis. In: 5th International Conference on the Applications of the Digital Information and Web Technologies (ICADIWT 2014), pp. 81–86 (2014)
28. Freund, Y., Schapire, R.E.: Experiments with a new boosting algorithm. In: Thirteenth International Conference on Machine Learning, pp. 148–156 (1996)
29. Kohavi, R.: A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. In: 14th International Joint Conference on Artificial Intelligence, IJCAI 1995, Canada, August 20-25, pp. 1137–1145 (1995)
30. Tan, P.-N., Steinbach, M., Kumar, V.: Introduction to Data Mining. Addison-Wesley (2005) ISBN 0-321-32136-7

Appendix A

Table 10. Prominent permissions and their description

Permissions	Description
WRITE_EXTERNAL_STORAGE	Permission for an application to write to the external storage
READ_PHONE_STATE	Permission read only access to phone state
CHANGE_WIFI_STATES	Allows changing wi-fi connectivity state
WAKE_LOCK	Allows using PowerManager WakeLocks to keep processor from sleeping or screen from dimming
SEND_SMS	Allows an app to send SMS Permission for the app to access network information
ACCESS_WIFI_STATE	Permission for the app to access network information
ACCESS_COARSE_LOCATION	Permission for the app to access approximate location by means o towers and wi-fi
ACCESS_FINE_LOCATION	Permission for the app to access precise location by means of towers and wi-fi
READ_CONTACTS	To read contact list of the device's user

Table 11. Trivial permissions and their description

Permissions	Description
RECEIVE_WAP_PUSH	Permission to monitor incoming wap push
WRITE_CALL_LOG	Permission for an application only to write user's contact data
READ_CALL_LOG	Permission for an application to read call log
CLEAR_APP_CACHE	Permission for an application to clear the caches of all apps that are installed
UPDATE_DEVICE_STATUS	Permission for an application to update device statistics
DEVICE_POWER	Permission for an application for low-level access to power management
CALL_PREVILEGED	Permission for an application to call any phone number without using dialer user interface to confirm the call
BATTERY_STATS	Permission for an app to collect battery statistics

Table 12. Significant API calls and their description

API calls	Description
onCreateOptionsMenu()	It is called only one time, i.e, 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()	Override these calls to implement custom view. Used when the contents of the view has to be changed
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
onAttachedToWindow()	It is called when the view is window attached
onKeyUp()	Called at the time of an event like a key up event

Table 13. Trivial API calls 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 suit

Appendix B

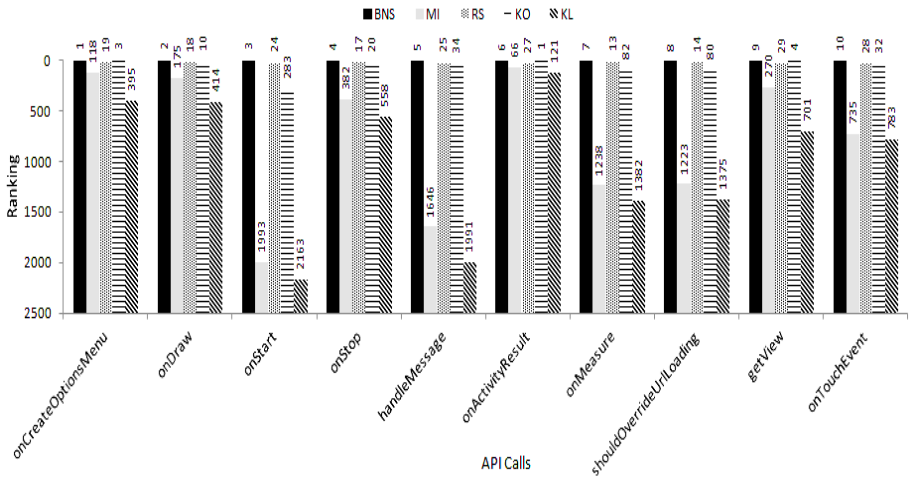

Fig. 3. Comparing the ranks of bottom BNS scored API calls with their MI, RS, KO and KL ranks (Lower ranks indicate high significance)

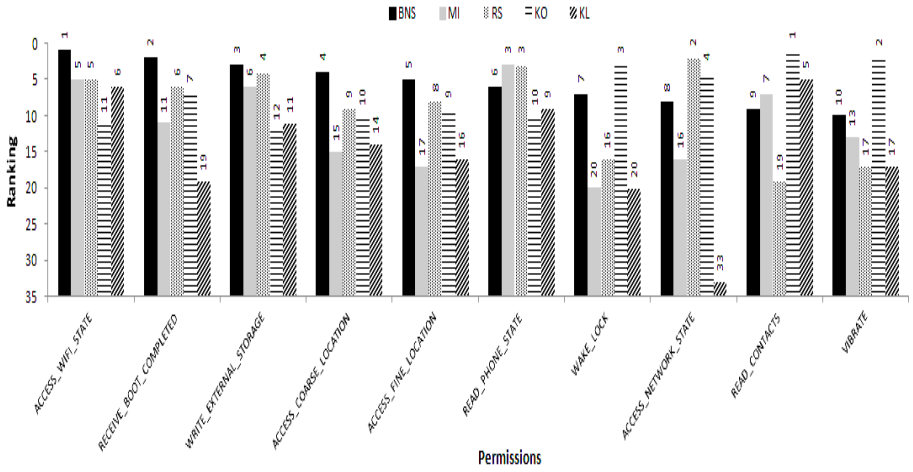

Fig. 4. Comparing the ranks of bottom BNS scored permissions with their MI, RS, KO and KL ranks (Lower ranks indicate high significance)