The Analysis of Feature Selection Methods and Classification Algorithms in Permission Based Android Malware Detection

Uğur PEHLİVAN, Nuray BALTACI, Cengiz ACARTÜRK, Nazife BAYKAL
CyDeS, Cyber Defense and Security Laboratory of METU-COMODO, Informatics Institute
Middle East Technical University (METU), Ankara, Turkey
{ugur.pehlivan, nbaltaci, acarturk, baykal} @metu.edu.tr

***Abstract*—Android mobile devices have reached a widespread use since the past decade, thus leading to an increase in the number and variety of applications on the market. However, from the perspective of information security, the user control of sensitive information has been shadowed by the fast development and rich variety of the applications. In the recent state of the art, users are subject to responding numerous requests for permission about using their private data to be able run an application. The awareness of the user about data protection and its relationship to permission requests is crucial for protecting the user against malicious software. Nevertheless, the slow adaptation of users to novel technologies suggests the need for developing automatic tools for detecting malicious software. In the present study, we analyze two major aspects of permission-based malware detection in Android applications: Feature selection methods and classification algorithms. Within the framework of the assumptions specified for the analysis and the data used for the analysis, our findings reveal a higher performance for the Random Forest and J48 decision tree classification algorithms for most of the selected feature selection methods.**

Keywords—cyber security, android application, machine learning, static analysis, feature selection, classification, malware detection

I. INTRODUCTION

Mobile devices of various operating systems have exhibited a steep increase in the past decade, thus leading to an increase in the number and variety of applications that run on mobile devices. Smartphones are recently used either in a complementary manner to a desktop computer or even to replace it. A closer look at the purpose of using mobile phones reveal that they are mostly used for web browsing, social networking, and online banking. In addition, they are used for mobile-specific functions such as SMS messaging, read-time broadcasting of location, and ubiquitous access [1]. As the capabilities in mobile phone functionality increase (e.g., applications for personal health), mobile phones become more and more attractive for a wider population. Market data surveys reveal that the number of smartphone sales worldwide reached 208 million in 2012. This was a 38.3% increase compared to the sales in 2011 [2].

The development of smartphones and mobile applications has resulted in a major change in people's way of doing tasks in various aspects of daily life, such as making business and conducting social communication. Mobile phone applications exhibit a rich variety not only in common daily life activities but also for users with more specific needs. From games to multimedia applications, navigation systems, and health-related applications, recently available mobile application markets, such as Google's Play Market, Apple's App Store, or Microsoft's Windows Store offer a wide variety of applications to users with different needs [4]. The major application markets have been growing steadily both in terms of the applications offered to the users and the downloads performed by the users (e.g., see [5] for the development of Google Play Market since the past several years).

The fast increase in the popularity of smartphones has led to an increase in their potential as a target for malicious activities [1]. This is mainly because users provide access for various types of private information by means of mobile applications [3]. As a result, some applications in the market have been identified as performing malicious activities. The spread of malicious software is also influenced by the policy of the market providers. For instance, Apple's App Store recently applies a policy that is subject to strict registration and company-issued digital certification before the release of any application, thus providing a security check for the applications listed in their application platform regularly [6]. Others, such as Google's Play Market, introduce more freedom to developers in uploading their own software to the market. The applications are then removed from the market in case of reported malicious activity. In particular, the Android operating system (in its recent form) allows removal of the malicious application from the device remotely. A similar mechanism of removal is also employed by Apple's App Store (namely, by *kill switch*) [7]. The policy of post-detection removal of malicious software brings the need for early detection of malware applications.

978-1-4799-4521-4/14/$31.00 ©2014 IEEE

In the present study, we focus on malware detection in Android operating systems.

Malware detection analysis of Android applications may be performed in two ways: Static analysis and dynamic analysis [8]. Static analysis is conducted by reverse-engineering an application, without running the application.

In particular, this process involves analyzing the file with the *.apk* extension and by focusing on the content of the two files: *AndroidManifest.xml* and *classes.dex* [9]. On the other hand, dynamic analysis (also called behavior-based detection) is conducted by running the application and analyzing its execution traces in a controlled environment [10]. A comparison between the static analysis methods and the dynamic analysis methods reveals that the latter requires more computing resources in terms of memory space and execution time. This is due to the fact that in dynamic analysis, the traces are extracted online during the course of application running on the system, whereas online detection is not necessary in static analysis [11]. The static analysis methods, however, require an intensive analysis of feature selection methods and processing algorithms for malware detection, as we explain in more detail below.

Android applications run by employing functions provided by the operating system. For this, applications need various permissions that should be granted by the mobile device user. Those requests for permissions are usually asked to the user during the course of installation of an application. The list of permissions that may be asked to the user by Android application is provided by [12]. The permissions that are used in the present study are presented in Section V.

The permissions specify affordances of an application in the operating system. Therefore permissions have the potential to allow malicious access of user data by the application. In addition, the analysis of malicious software through the investigation of the permissions is a static analysis, because the application does not run but the permissions are analyzed. In the present study, we use permissions for the backbone of data input to our static analysis.

Two major important aspects of permission-based analysis of malicious software are **feature selection methods** and **classification algorithms**. In general, feature selection methods are used for reducing the dimension size of a dataset. For this, some of the features (attributes) which are not useful in the analysis are removed from the data set. The remaining features are selected by taking into account the representational power of all the features in the dataset.

An efficient feature selection method introduces performance gains in data analysis. Therefore, it is usually conceived as a necessary step for preparing a dataset with a manageable size for the analysis (in terms of the availability of computational resources). Choosing an appropriate feature selection method among alternatives is, however, a challenge. It is not only influenced by the characteristics of the dataset but it also interacts with the selected algorithm for data classification. In the present study, we selected a set of feature selection methods and classification algorithms, as presented in Table 1 below.

TABLE 1. Feature selection methods and classification algorithms for data analysis.

Feature Selection Methods	Classification Algorithms
• *Gain Ratio Attribute Evaluator* • *ReliefF Attribute Evaluator* • *Cfs Subset Evaluator* • *Consistency Subset Evaluator*	• *Bayesian Classification* • *Classification and Regression Tree (CART)* • *J48 Decision Tree* • *Random Forest* • *Sequential Minimal Optimization (SMO)*

The rest of the paper is organized as follows. The next section presents the related work. Section III gives information about the dataset used in the analysis. In Section IV, we introduce the methodology used in the analysis. Section V discusses the issues about the implementation of the methodology for data analysis. In Section VI, we discuss the results and the general findings obtained from the analysis. Finally, Section VII presents conclusions and further research agenda.

II. Related Work

Previous research employed various classification algorithms for the classification of malicious Android applications by static analysis. For instance, [13] performed a permission-based analysis of the applications at Google's Play Market by extracting the permission set from the relevant *AndroidManifest.xml* files, as stated in the previous section. Utilizing the *Information Gain* feature selection method to reduce the size of the feature set, and applying the *J48 Decision Tree*, *Classification and Regression Tree (CART)* and *Random Forest* classification algorithms onto the reduced data set, performances were compared in terms of true positive, false positive, precision and recall rates. According to their results, *J48* and *Random Forest* outperform *CART* by showing higher precision rates and lower false positive rates. In another study, the behavior of selected Android applications were characterized by a feature-based learning framework which included API calls as features combined with permissions. The analysis employed *SVM*, *J48* and *Bagging* classification algorithms. The results revealed that the combination of API calls with the requests for permissions increases the efficiency of malware detection [14]. In another study [15], frequent combinations of request for permissions, intents, broadcast receivers and native code were added to the feature set. The analysis was performed by employing *Random Forest* classification and by using Android Malware Genome Project public data, as well as a set of benign applications obtained from third party markets. The results revealed that the method outperformed conventional anti-malware tools, even for novel malware software. Another study [16] using the Malware Genome Project data classified Android applications with *Bayesian* classifier based on static code analysis. API calls, Linux system commands and permissions were extracted as features by reverse engineering the *.apk* files. *Mutual Information* (MI) *maximization* technique was used to select features considering classes of applications (as malicious or benign). Results reported better detection rates than then popular signature-based antivirus software on the same set of malware samples. In [30], a method was presented by analyzing the information only within manifest files with *J48 Decision Tree* algorithm to detect Android malware.

As introduced in Section I, malware detection methods do not only include static analysis but also include dynamic analysis. Among many others, one approach, called *Crowdroid*, employed crowd-sourcing for obtaining application execution traces. The analysis applied a behavioral framework to distinguish Trojan horses from benign applications which had the same name and the same version but a different dynamic behavior. By uploading the crowd-sourcing application to Google's Play Market and feeding the algorithm by non-personal (but behavior-related) data obtained from the users, an efficient malware detection method was developed. The algorithms was based on the system calls that built feature vectors for *k-means* classification [10]. Another host-based behavioral analysis framework is called *Andromaly*, which continuously monitors features and events obtained from the mobile device. *Andromaly* then applies various classification algorithms, including *k-means*, *logistic regression*, *histogram*, *decision tree*, *Bayesian networks* and *Naïve Bayes* after feature selection (*Chi-square*, *Fisher Score* and *Information Gain* as feature selection algorithms) [17].

In the present study, we focus on static analysis methods by extending the previous studies, in particular, by introducing a broad variety feature selection methods and classification algorithms. Three types of classification algorithms, namely *Bayesian*, *Decision Tree* and *Support Vector Machines (SVM)* are used for a performance comparison. Moreover, two types of feature selection methods, which are attribute-based and subset-based, were evaluated with their performance contribution to the classification algorithms. To the best of our knowledge, some of those feature selection methods (namely *Consistency Subset Evaluator* and *ReliefF Attribute Evaluator*) have not been used previously for the purpose of malware detection in mobile applications. In the following section, we describe the data set before introducing the feature selection methods and the classification algorithms.

III. DATA

The dataset, which was used in the analysis, was provided by COMODO Security Solutions, Inc., a private security company [18]. A branch of COMODO Security Solutions, Inc. is located at the Middle East Technical University (METU) campus, Turkey. The dataset includes 3,784 android applications, 2,338 of which are benign and 1,446 of which are malware applications. The label as 'benign' or 'malware' was provided in the dataset, as specified by the company. This label was used for supervised learning in the feature selection and classification analyses, as described in the following section.

IV. METHODOLOGY

As introduced in Section II, feature selection allows removing some features from the dataset which are already redundant or irrelevant for the analysis. The resulting dataset usually leads to a reduced processing duration and higher accuracy compared to the raw dataset. In the present study, four feature selection methods and five classification algorithms were examined by investigating the accuracy of classification prediction of the applications into 'benign' or 'malware' (Table 1). Weka (Waikato Environment for Knowledge Analysis) software tool was employed for implementing the feature selection methods and the classification algorithms [19]. The evaluation of the feature selection methods and the classification algorithms were performed by using the following measures.

- Overall Accuracy (ACC)
- True Positive (TP) Rate
- False Positive (FP) Rate
- Precision

Android applications (i.e., *.apk* files) include all the data required for the installation of an Android application on the mobile device, including the requests for permission. For example, *android.permission.CALL_PHONE* allows an application to initiate a phone call without going through the Dialer user interface for the user to confirm the call being placed. Similarly, *android.permission. INTERNET* allows applications to open network sockets. The set of all the permissions and their functions in Android applications can be seen in [12]. The features that are used in the present study are presented in the following section. The focus of our analysis is to examine whether there exists a relationship between the requests for permission (as processed by features) and the classification of the application as benign or malicious.

Android applications are developed in Java programming language. The installation package is a compressed (ZIP) bundle of the files including the manifest file (*AndroidManifest.xml*) and *classes.dex*. The components of an Android application, such as the activities, services, broadcast receivers, and content providers that the application is composed of are described in the manifest file [20]. The manifest file specifies the permissions the application must have to access the protected parts of the API and to interact with the applications' components and other applications. Therefore the permissions specified in the manifest file should be granted by the user during the course of installation. Below we present the features that are employed to identify permissions in the present study.

A. Features

Features are the attributes used for defining the permission characteristics of an application. Each feature (e.g., *android.permission.CALL_PHONE*) corresponds to a permission specified by the application. A feature set can be specified as a feature vector, which includes all the permissions that are requested from the user. In the present study, we also added *version code* and *version name* to the feature set, which are included in the manifest file. This resulted in a feature set which was composed of 182 data points. Below is an example of a feature vector (for a single application), which includes binary values for the 182 features. We included the *version code* and the *version name* without conversion (cf. the first two data points in the vector).

*(3,1.2,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,1,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,
1,1,0,1,1,1,1,0,0,0,1,1,1,1,1,1,1,1,1,1,1,0,1,1,1,0,1,1,0,0,0,1,1,0,0,0,1,0,1,0,
1,1,0,0,0,0,0,0,1,1,0,1,0,0,0,0,1,0,0,0,0,1,1,0,1,0,1,1,1,0,0,0,1,0,0,0,0,1,1,1,
1,0,1,1,0,0,1,0,1,0,1,0,1,1,0,1,0,1,1,1,1,1,1,0,0,1,0,0,1,1,1,1,1,0,0,1,1,0,0,1,
0,1,1,1,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0)*

Figure 1: A sample feature vector.

In the following section, we present the feature extraction process.

B. Feature Extraction

In order to reach the permission data of an application, the *apk* file must be extracted from the *.apk* extended file and then the *AndroidManifest.xml* file must be read. We employed the APKTOOL [21] to extract the *apk* file into the *Android Manifest* file and the *Smali* file. Then, the permissions defined in the Android manifest file were examined to extract which type of permissions were required to run the application. Figure 2 depicts the process of *apk* file extraction

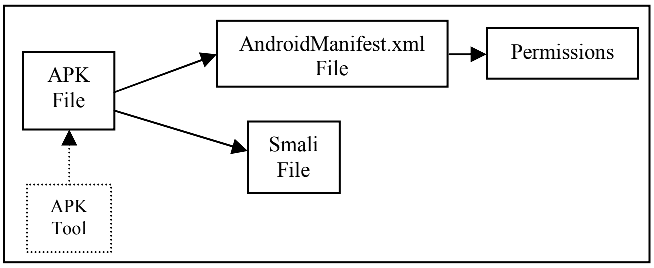

Figure 2: APK File Extraction

C. Feature Selection

Feature selection is performed to reduce the dataset size by removing the features (attributes) which are not beneficial to be used in the analysis. The features are selected according to their representation capability of all the dataset. Efficient feature selection methods introduce performance gains by reducing the dataset size and the time spent in classification analysis. Feature selection methods can be divided into two approaches, called *attribute-based* feature selection methods and *subset-based* feature selection methods. Attribute-based feature selection methods evaluate each feature separately, independent of other features. It is mainly based on class features. Dependencies among the features are ignored, but the relation to the class feature is taken into account. On the other hand, in subset-based feature selection methods, feature subsets are constructed randomly and the subset that best represents the whole features is selected. Dependencies among the features are taken into account. In the present study, we employed the following attribute-based and subset-based feature selection methods.

Gain Ratio Attribute Evaluator: This feature selection method evaluates the worth of an attribute by measuring the gain ratio with respect to the class. It requires a class feature to evaluate features. The benign/malware characteristic is used for the class feature in this method. Also, the number of features has to be specified in this selection method. In the present study, 25 and 50 features were assumed to be sufficient for the representation of 182 features. Hence, the number of features was separately reduced to 25 and 50 features by applying this method, as specified by the following formulation.

$$\begin{aligned}&\mathrm{GainR(Class, Attribute)}\\&\quad = (\mathrm{H(Class)} - \mathrm{H(Class} \mid \mathrm{Attribute)}) / \mathrm{H(Attribute)}\end{aligned}$$

ReliefF Attribute Evaluator: This method evaluates the worth of an attribute by repeatedly sampling an instance and by considering the value of the given attribute for the nearest instance of the same and different class. It can operate on both discrete and continuous class data [22]. The method also requires a class feature and a specified number of features. In the present study, benign/malware status was used as a class feature, and 25 and 50 features were assumed to be satisfactory for representing the whole feature set.

Cfs Subset Evaluator: This method evaluates the worth of a subset of attributes by considering the individual predictive ability of each feature along with the degree of redundancy between them. Subsets of features that are highly correlated with the class, while having low intercorrelation, are preferred [23]. It does not require a class feature because it selects the subset randomly by using a specified number of features. In the present study, 25 and 50 features were assumed to be satisfactory for the purpose of the analysis.

Consistency Subset Evaluator: This method evaluates the worth of a subset of attributes by the level of consistency in the class values when the training instances are projected onto the subset of attributes. Consistency of any subset can never be lower than that of the full set of attributes [24]. It does not require a specified number of features. The method determines the optimum number of features and it returns the optimum feature subset in the result.

The four feature selection methods described above were separately run on the dataset, which included 3,784 android applications with different feature sets.

D. Classification

Classification algorithms were run on the datasets, which were obtained after feature selection. In the present study, we employed the following classification algorithms.

Bayesian Classification: This algorithm is based on so-called Bayes classifier, which employs estimator classes. In the present study, numeric estimator precision values were selected based on the analysis of training data. [28].

Classification and Regression Tree (CART): This algorithm implements minimal cost-complexity pruning [25].

J48 Decision Tree Algorithm: This algorithms generates either a pruned or an unpruned C4.5 decision tree [26].

Random Forests (RF): The idea behind this classification algorithm is to construct a forest of random trees [27].

Sequential Minimal Optimization (SMO): This algorithm presented by John Platt implements training of a support vector classifier [31].

V. Implementation

The data set consisted of 3,784 *apk* installation packages for the Android applications. We developed C# scripts for bunch extraction of the installation packages to avoid manual extraction. After the extraction of the installation packages, we separated the *AndroidManifest.xml* files to identify the permission requests for each package. The permissions are written by standard keywords in Manifest files. For instance, the lines below exemplify the permission request of the application to make a phone call.

```
<uses-permissionandroid:name
    = android.permission.CALL_PHONE />
```

Another example is below, which asks for permission for internet usage.

```
<uses-permissionandroid:name
    = android.permission.INTERNET/>
```

Similar standard statements are used for each permission type in the Manifest files. Therefore, an Android application should involve each and every permission statement within the Manifest file. Accordingly, we searched for those statements in all the Manifest files in the dataset of application packages. If the searched permission statement was found in a manifest file, the corresponding binary feature vector value (in the relevant column) was updated for the relevant application (in the relevant row). This process resulted in a data matrix, which consisted of 3,784 Android applications, represented by the rows, and permissions represented by the columns. This data matrix was then used as input for the analysis.

The selected feature selection methods were employed to identify their performance, as well as their compatibility with the classification algorithms. The *Gain Ratio Attribute Evaluator*, the *ReliefF Attribute Evaluator*, and the *Cfs Subset Evaluator* feature selection methods require predefined number of features as input for further processing. In the present study, we used 25 features and 50 features, and we used more than one feature selection implementation for *Gain Ratio Attribute Evaluator*, *ReliefF Attribute Evaluator* and *Cfs Subset Evaluator* by assuming that 25 features might not have been enough to represent the whole feature set and 50 features might have included redundant features. Furthermore, the *Consistency Subset Evaluator* feature selection method does not require a predefined number of features. Instead, the method determines the optimum number of features itself and after implementation, it returns the best feature set. Consequently, seven different feature selection methods were implemented in feature selection method step to reduce the total of 182 features to an optimum number of features.

The results of the application of the seven feature selection implementations revealed a total of 97 features, which were identified as important in representing the whole feature set. Among the 97 features, one of them (WRITE_APN_SETTING) was identified as important in representing the whole feature set by all four feature selection methods, whereas 34 of them were identified as important in representing the whole feature set by at least three of four feature selection methods. The selected features are presented in Table 2 below.

TABLE 2. Selected features by the feature selection methods

Permission Name	Permission Name
WRITE_APN_SETTINGS*	RECEIVE_WAP_PUSH
ACCESS_WIFI_STATE	WAKE_LOCK
CHANGE_WIFI_STATE	WRITE_EXTERNAL_STORAGE
DELETE_PACKAGES	WRITE_SETTINGS
MOUNT_UNMOUNT_FILESYSTEMS	WRITE_HISTORY_BOOKMARKS
READ_PHONE_STATE	ACCESS_FINE_LOCATION
READ_SMS	BAIDU_LOCATION_SERVICE
RECEIVE_BOOT_COMPLETED	CLEAR_APP_CACHE
RECEIVE_SMS	GET_TASKS
RESTART_PACKAGES	INTERNET
SEND_SMS	READ_EXTERNAL_STORAGE
SYSTEM_ALERT_WINDOW	ACCESS_FIND_LOCATION
INSTALL_SHORTCUT	DEVICE_POWER
CALL_PRIVILEGED	GET_ACCOUNTS
CHANGE_NETWORK_STATE	INTERNAL_SYSTEM_WINDOW
INSTALL_PACKAGES	PROCESS_OUTGOING_CALLS
READ_LOGS	WRITE_SMS

*Only “WRITE_APN_SETTINGS WRITE_APN_SETTINGS” feature was selected by 4, the remaining ones were selected by 3 feature selection algorithms.

For the next step after feature selection, we formed seven datasets with different number of features and feature contents. The number of android applications was the same in each dataset with 3,784 applications. The only difference between the datasets was related to feature dimension. The datasets were used as input to the classification step of the analysis, as described below.

Five classification algorithms (*Bayesian Classification*, *Classification and Regression Tree CART*, *J48 Decision Tree*, *Random Forests RF* and *Sequential Minimal Optimization SMO*) were run on the seven datasets obtained in the feature selection step. This resulted in a total of 35 classification iterations. We employed the default settings as specified by the Weka data mining and classfication software tool.

To improve the validity of the training and testing, we employed k-*fold cross validation* to divide the training and the test data in the datasets, as suggested by the previous research [29]. The steps below are followed for the cross-validation process.

- Step 1: Divide the data randomly into k folds (subsets) of equal size.
- Step 2: Train the model on $k-1$ folds, use one fold for testing.
- Step 3: Repeat this process k times so that all the folds are used for testing.

- Step 4: Compute the average performance on the *k* test sets.

This cross validation process effectively uses all the data for both training and testing. In the present study we assumed *k* = 5. The following section presents the results of the classification process and for each dataset as formed by the feature selection methods.

VI. Implementation Results and Evaluation

The evaluation of the classification algorithm implementations were performed in terms of the following evaluation measures: *Overall Accuracy*, *True Positive Rate*, *False Positive Rate*, and *Precision*. These measures are derived from four basic measures that are described below.

True Positive (TP): This measure specifies the number of correctly identified benign applications.

False Positive (FP): This measure specifies the number of incorrectly identified malware applications. In other words, a false positive classification identifies an originally malicious application incorrectly as a benign application.

True Negative (TN): This measure specifies the number of correctly identified malware applications.

False Negative (FN): This measure specifies the number of incorrectly identified benign applications. In other words, a false negative classification identifies an originally benign application incorrectly as a malicious application.

Table 3 presents four basic classification measures in terms of the relationship between malicious and benign status of the application.

TABLE 3. The summary of four basic measures for evaluation.

		Prediction	
		Malicious	**Benign**
Reality	**Malicious**	TRUE NEGATIVE	FALSE POSITIVE
	Benign	FALSE NEGATIVE	TRUE POSITIVE

The derived evaluation measures are described in terms of the basic measures, as presented below:

Overall Accuracy: The ratio of correctly identified applications. In other words, it shows the ratio of correctly classified instances.

$$Overall\ Accuracy = (TP + TN) / (TP + TN + FP + FN)$$

True Positive Rate: The ratio of correctly identified benign applications. True Positive Rate is also called *Recall*.

$$TPR = TP / (TP + FN)$$

False Positive Rate: The ratio of incorrectly identified malicious applications.

$$FPR = FP / (TN + FP)$$

Precision: It is the ratio of retrieved instances that are relevant. It is also called *positive predictive value*.

$$Precision = TP / (TP + FP)$$

Precision reflects the fraction of correctly classified instances within all classified instances. Precision is a different measure than the accuracy of classification. In the following section, we present the performance evaluation of the selected feature selection methods.

A. The Performance Evaluation of Feature Selection Methods

As presented in the previous section, the number of features in the three feature selection methods were specified as 25 and 50. The fourth feature selection method, namely *Consistency Subset Evaluator*, determines optimum number of feature itself. For this reason, it wasn't possible to use 25 and 50 features for the *Consistency Subset Evaluator*. The feature selection methods applied in the present study produced different prediction output values depending on the classification algorithm used (Table 4). For instance, the *Bayesian* classification algorithm returned the best results for all evaluation parameters (*TPR*, *FPR*, and *precision*) when it was used with the *Cfs Subset Evaluator* (25) feature selection method with 25 features. On the other hand, the *J48 Decision Tree* algorithm revealed the best accuracy when it was used with the dataset formed by the *Gain Ratio Attribute Evaluator* (50) and the dataset formed by the *Cfs Subset Evaluator* (25) feature selection methods. The *Random Forest* (RF) classification algorithm returned the best result with the *Cfs Subset Evaluator* (25) feature selection method in terms of prediction *accuracy*. The *Classification and Regression Tree* (*CART*) algorithm returned close prediction results with *J48 Decision Tree* for all of the selected feature selection methods. In addition, it complied best with the *Gain Ratio Attribute Evaluator* (50) and the *Cfs Subset Evaluator* (25) feature selection methods as the *CART* algorithm did. Finally, the *Sequential Minimal Optimization* (*SMO*) SVM algorithm revealed the best performance for all evaluation parameters when it was used with the *ReliefF Attribute Evaluator* (50) feature selection method.

The *Consistency Subset Evaluator* feature selection method determined the optimum number of features as 36 features. The *Consistency Subset Evaluator* method exhibited the weakest performance in all classification algorithms with the lowest overall accuracy and the lowest true positive rate. Following the *Consistency Subset Evaluator* that, the second weakest feature selection method was the *ReliefF Attribute Evaluator*.

An overall evaluation of the results obtained by the classification algorithms shows that the *Cfs Subset Evaluator* (25) and the *Gain Ratio Attribute Evaluator* (50) gave a good performance in three classification algorithms. Another significant outcome of the analysis is that the *Cfs Subset Evaluator* (25) showed a better performance than the *Cfs Subset Evaluator* (50). In other words, 25 features which were selected for the *Cfs Subset Evaluator* better represented the whole feature set.

TABLE 4. The results of the analysis.

Feature Selection Method	Classification Algorithm	Overall Accuracy (ACC)	TP Rate	FP Rate	Precision
Cfs Subset (25)	Bayesian	89.32%	0.893	0.120	0.893
	CART	93.58%	0.936	0.073	0.936
	J48	93.87%	0.939	0.072	0.939
	Random Forest	94.90%	0.949	0.060	0.949
	SMO	91.62%	0.916	0.082	0.918
Cfs Subset (50)	Bayesian	88.35%	0.883	0.139	0.883
	CART	89.98%	0.900	0.097	0.903
	J48	90.51%	0.905	0.078	0.913
	Random Forest	92.79%	0.928	0.078	0.928
	SMO	92.68%	0.927	0.082	0.927
Consistency Subset	Bayesian	72.89%	0.729	0.367	0.727
	CART	74.05%	0.740	0.373	0.749
	J48	73.39%	0.734	0.384	0.744
	Random Forest	74.21%	0.742	0.369	0.750
	SMO	72.78%	0.728	0.399	0.742
Gain Ratio Attribute (25)	Bayesian	87.66%	0.877	0.150	0.876
	CART	91.99%	0.920	0.081	0.921
	J48	92.23%	0.922	0.078	0.924
	Random Forest	92.15%	0.922	0.081	0.922
	SMO	91.15%	0.911	0.085	0.914
Gain Ratio Attribute (50)	Bayesian	87.95%	0.879	0.146	0.879
	CART	93.60%	0.936	0.071	0.936
	J48	93.90%	0.939	0.069	0.939
	Random Forest	94.50%	0.945	0.062	0.945
	SMO	92.60%	0.926	0.081	0.926
ReliefF Attribute (25)	Bayesian	87.18%	0.872	0.155	0.871
	CART	89.59%	0.896	0.103	0.899
	J48	90.35%	0.904	0.085	0.909
	Random Forest	91.91%	0.919	0.094	0.919
	SMO	92.34%	0.923	0.088	0.923
ReliefF Attribute (50)	Bayesian	87.84%	0.878	0.144	0.878
	CART	89.69%	0.897	0.100	0.900
	J48	91.15%	0.911	0.081	0.915
	Random Forest	93.42%	0.934	0.072	0.934
	SMO	93.18%	0.932	0.077	0.932

B. The Performance Evaluation of Classification Algorithms

The *Random Forest* classification algorithm exhibited the best performance almost in all feature selection methods, except for the *ReliefF Attribute Evaluator* (25), for which the *SMO* classification algorithm returned better accuracy, and *the Gain Ratio Attribute Evaluator* (25), for which *the J48 Decision Tree* classification algorithm returned better accuracy. Moreover, the *J48 Decision Tree* algorithm was the second best because it showed the best results for one (*Gain Ratio Attribute Evaluator* (25)) and the second best results for two feature selection methods (*Gain Ratio Attribute Evaluator* (25) and *Cfs Subset Evaluator* (25)). The *SMO* classification algorithm returned the second best results for two feature selection methods, therefore it can be evaluated as being worse when it is compared to the *J48 Decision Tree* algorithm. The *Bayesian* classification algorithm pointed out the weakest performance nearly for all feature selection methods, except for the *Consistency Subset Evaluator* (36).

The results obtained by the evaluation of the classification algorithms suggest that the *Random Forest* and the *J48 Decision Tree* classification algorithms might be preferred for the task of permission-based Android malware detection. On the other hand, the results suggest that the *Bayesian* algorithm should be avoided in permission based classification analysis of Android applications.

C. The Compatibility of Feature Selection Methods and Classification Algorithms

Bayesian classification showed a relatively high performance (the overall accuracy of 89.32%.) with the *Cfs Subset Evaluator* with 25 features. The *J48 Decision Tree* algorithm returned high accuracy when it was with *Gain Ratio Attribute Evaluator* with 50 features and *Cfs Subset Evaluator* with 25 features, with 93.90% overall accuracy in both cases. The *Random Forest* showed the best performance with the *Cfs Subset evaluator with 25 features and Gain Ratio attribute evaluator* with 50 features, with 94.90% and 94.50% overall accuracies respectively. This finding shows that 25 features and 50 features exhibited close accuracy. These were the highest overall accuracy values obtained in the analysis. The *Classification and Regression Tree* (*CART*) algorithm displayed the best performance with the *Cfs Subset Evaluator* with 25 features and *Gain Ratio Attribute Evaluator* with 50 features. Finally, the *Sequential Minimal Optimization (SMO)* revealed the best compatibility with the *ReliefF Attribute Evaluator* with both 25 and 50 features.

VII. CONCLUSION

In the present study, we performed a permission-based analysis of malware classification for Android applications in two major steps. In the first step, we applied a set of feature selection methods (*Gain Ratio Attribute Evaluator*, *ReliefF Attribute Evaluator*, *Cfs Subset Evaluator* and *Consistency Subset Evaluator*) to reduce the size of the feature dimension of the dataset, which originally had 182 features. We then used the reduced datasets as input sets to five classification algorithms (*J48 Decision Tree*, *Bayesian Classification*, *Random Forest*, *Classification and Regression Tree and SMO*). The performance of the algorithms was evaluated by means of a set of measures, in particular by *Overall Accuracy*, *True Positive Rate*, *False Positive Rate* and *Precision* measures.

The findings revealed that the *Cfs Subset Evaluator* feature selection method gave a good performance when it was used with 25 features. Moreover, increasing the number of features to 50 did not improve accuracy. As for the classification algorithms, the results suggested that the permission based classification analysis of Android applications can be more accurately performed with *Random Forest* and *J48 Decision Tree* classification but not with the *Bayesian algorithm*. The prediction results of the classification algorithms revealed that the permission based Android malware detection model can be performed more accurately by using the *Random Forest* algorithm. Furthermore, the *Random Forest* and *J48 Decision Tree* work best with the *Gain Ratio Attribute Evaluator* with 50 features and with the *Cfs Subset Evaluator* with 25 features, whereas *Bayesian* returned poor performance independent of the feature selection method.

We expect that the results of the current study might provide the basis for future research in malware detection. Our future research will address a finer-grained analysis of the effect of dataset size in terms of the number of data points (i.e., the number of Android applications). We predict that an optimum dataset size can be determined depending on the selected feature selection method and the classification algorithm. In addition, the future research should address other static features existing in an apk files to conduct static analysis of Android malware, including but not limited to specific API calls though the requirements of higher resources for the analysis. Finally, the future research may focus on clustering analysis to identify emergent clusters in the dataset.

Acknowledgment

This study has been conducted by CyDeS Cyber Defense and Security Laboratory of METU-COMODO, launched as a joint venture between the Middle East Technical University (METU) Informatics Institute and COMODO CyDeS Laboratory of COMODO Security Solutions Inc. We thank the reviewers of the IEEE Symposium Series on Computational Intelligence for their valuable comments and suggestions.

VIII. REFERENCES

[1] P. Felt, M. Finifter, E. Chin, S. Hanna and D. Wagner, «A Survey of Mobile Malware in the Wild,» In SPSM '11 Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices, p. 3-14, 2011, Chicago, Illinois.

[2] G. Suarez-Tangil, J. E. Tapiador, P. Peris-Lopez and J. Blasco, «DENDROID: A text mining approach to analyzing and classifying code,» Expert Systems with Applications , 41(4/1), pp. 1104-1117, 2014

[3] Rapid7 Corporate Headquarters, «Mobile Security Guide: Protect Your Organization From Mobile Malware,» Rapid7 Corporate Headquarters, Boston, 2013.

[4] Symantec, «Securing the Mobile App Market: How Code Signing Can Bolster Security for Mobile Applications,» Symantec, 2012.

[5] Statista-The Statistics Portal, 2014. Available: http://www.statista.com/statistics/281106/number-of-android-app-downloads-from-google-play/.

[6] Symantec, «A Window Into Mobile Device Security: Examining the security approaches employed in Apple's iOS,» Symantec, 2011.

[7] Teufl, P., Kraxberger, S., Orthacker, C., Lackner, G., Gissing, M., Marsalek, A., Leibetseder, J., and Prevenhueber, O. (2012). «Android Market Analysis With Activation Patterns,» In Security and Privacy in Mobile Information and Communication Systems (pp. 1-12). Springer Berlin Heidelberg.

[8] Amamra, A., Talhi, C., and Robert, J. (2012, October). «Smartphone malware detection: From a survey towards taxonomy, » In Malicious and Unwanted Software (MALWARE), 2012 7th International Conference on (pp. 79-86). IEEE.

[9] Tchakounté, F., & Dayang, P., «System Calls Analysis of Malwares on Android, » International Journal of Science and Technology, 2(9), pp. 669-674, 2013.

[10] Burguera, I., Zurutuza, U., & Nadjm-Tehrani, S. (2011, October). «Crowdroid: behavior-based malware detection system for android, » In Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices (pp. 15-26). ACM.

[11] Shabtai, A., Fledel, Y., & Elovici, Y. (2010, December). «Automated Static Code Analysis For Classifying Android Applications Using Machine Learning. » In Computational Intelligence and Security (CIS), 2010 International Conference on (pp. 329-333). IEEE.

[12] «Developers,» Available: http://developer.android.com/reference/android/Manifest.permission.html .

[13] Aung, Z., & Zaw, W. (2013). «Permission-based Android malware detection. » International Journal Of Scientific & Technology Research, 2(3), pp. 228-234.

[14] Peiravian, N., & Zhu, X. (2013, November). «Machine Learning for Android Malware Detection Using Permission and API Calls, » In Tools with Artificial Intelligence (ICTAI), 2013 IEEE 25th International Conference on (pp. 300-305). IEEE.

[15] Glodek, W., & Harang, R. (2013, November). «Rapid Permissions-Based Detection and Analysis of Mobile Malware Using Random Decision Forests, » In Military Communications Conference, MILCOM 2013-2013 IEEE (pp. 980-985). IEEE.

[16] Yerima, S. Y., Sezer, S., McWilliams, G., & Muttik, I. (2013, March). «A new android malware detection approach using bayesian classification, » In Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference on (pp. 121-128). IEEE.

[17] Shabtai, A., Kanonov, U., Elovici, Y., Glezer, C., & Weiss, Y. (2012). « "Andromaly": a behavioral malware detection framework for android devices, » Journal of Intelligent Information Systems, 38(1), 161-190.

[18] "COMODO Security Solutions, Inc. Web Site," URL https://tr.comodo.com/

[19] "Weka 3: Data Mining Software in Java," http://www.cs.waikato.ac.nz/ml/weka/

[20] "App Manifest," http://developer.android.com/guide/topics/manifest/manifest-intro.html

[21] "android-apktool, A tool for reverse engineering Android apk files" URL http://code.google.com/p/android-apktool

[22] Kononenko, I. (1994, January). «Estimating attributes: analysis and extensions of RELIEF, » In Machine Learning: ECML-94 (pp. 171-182). Springer Berlin Heidelberg.

[23] Hall, M. A. (1999). «Correlation-based feature selection for machine learning, » (Doctoral dissertation, The University of Waikato).

[24] Liu, H., & Setiono, R. (1996, July). «A probabilistic approach to feature selection-a filter solution, » In ICML (Vol. 96, pp. 319-327).

[25] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, Charles J. Stone (1984). «Classification and Regression Trees, » Wadsworth International Group, Belmont, California.

[26] Quinlan, J.R., (1993). «C4.5: Programs for Machine Learning, » Morgan Kaufmann Publishers, San Mateo, CA.

[27] Breiman, L., (2001). « Random Forests, » Machine Learning. 45(1):5-32.

[28] John, G. H., & Langley, P. (1995, August). « Estimating continuous distributions in Bayesian classifiers, » In Proceedings of the Eleventh conference on Uncertainty in artificial intelligence (pp. 338-345). Morgan Kaufmann Publishers Inc.

[29] F. Keller, "Lecture Notes Evaluation,"

[30] R. Sato, D. Chiba and S. Goto, «Detecting Android Malware by Analyzing Manifest Files,» *Proceedings of the Asia-Pacific Advanced Network 2013 v. 36, p. 23-31.,* 2013.

[31] J. Platt, «Fast Training of Support Vector Machines using Sequential Minimal Optimization,» Advances in Kernel Methods – Support Vector Learning, 1998.