A New Approach to Improve Mobile Network's Security Through Android Malware Detection Utilizing Static Analysis

Mahmood Deypir[1], Mani Saffarnia[2],
1- Department of Computer Engineering, South Tehran Branch, Islamic Azad University, Tehran, Iran.
Email: mdeypir@azad.ac.ir (Corresponding author)
2- Department of Electrical and Computer Engineering, Science and Research Branch, Islamic Azad University, Tehran, Iran.
Email: mani.saffarnia@gmail.com

ABSTRACT:
The security of the mobile devices has become a major issue since hackers target them through malwares in order to harm the systems or gather sensitive information and get access to the systems remotely. Recently, new ways have been introduced to confront malwares and other viruses. Two main techniques for recognizing malwares are dynamic analysis and static analysis. This paper proposes a new method using the static analysis to help improve the accuracy of the malwares in detecting threats faster and with lower processing time. For this purpose, our suggested method has utilized the android application's main components to recognize the malwares using the machine learning algorithms. Furthermore, our method has used the feature selection algorithms to reduce the processing overload and to enhance the speed and accuracy. Our method have used the following components as the classification features in our suggested algorithms: API calls, Intents, network address and IPs, services and provider, activities and permissions. In addition to these individual features, our method has also employed complex features to improve malware recognition. We have used 123,446 software and 5,561 malwares to evaluate the accuracy and the precision of the suggested method, demonstrating to be 99.4 percent.

KEYWORDS: Android Security, Malware Detection, Static Analysis, Classification, Machine Learning.

1. INTRODUCTION

In recent years, rapid development of the technology has greatly changed the human lives. As an example, smart phones with their high processing capabilities have become part of our daily lives. Among the smart phones with various operating systems, android-based phones are quite pervasive. The security of the android devices has become a major issue since hackers target them through malwares in order to harm the systems or gather sensitive information and get access to the systems remotely. In this paper, we introduce a new approach for android malware detection based on static analysis. In this approach, we have used android application components (e.g., API calls, Intents, network address and IPs, permissions, activities, services and providers) as classification features, and filter these features by feature selection algorithms. Besides using these simple features, we have introduced complex feature, which is the combination of simple features that were filtered in the feature selection step. Create data model based on the integration of the simple features and complex feature culminate to the higher accuracy and precision in malware detection. The main contributions of this paper can be summarized as new feature selection, introduction of complex features, and extensive experiment on wide range of data mining algorithm for android malware detection.

This paper is organized as follows. In section 2, we review previous works in this field. Then, in section 3, we explain our method. In section 4 and section 5, we present our experimental results and conclusions, respectively

2. BACKGROUND RESEARCH

Two main approaches exist for the malware detection: static analysis and dynamic analysis. Static analysis works on the application codes to detect malwares without running it [1]. Using this approach, the apk file is decompiled and its components are extracted. Features like permissions, hardware components, API call, data flow and etc. have been extracted from the application resources, and then later

تاریخ دریافت: ۱۳۹۷/۱۰/۱۷ تاریخ پذیرش: ۱۳۹۸/۰۲/۱۸

used for the malware detection. Static analysis provides the facility of detecting malwares without running the application on the phone [2]. In the dynamic approach, we install the application in an isolated environment and then run it. The behavior of the application is monitored to detect any malicious behavior. In the dynamic analysis approach, application's dynamic features such as network traffic, system calls, and resource usage are evaluated [3]. In both approaches, the data is sent to the machine learning algorithms so that it can categorize the applications into two classes: malware and benign. [4] In the following, we review related previous studies regarding the static analysis, the core and foundation of our proposed method.

Huang et. al [5] have studied malware detection by analyzing permissions. In this study, all the permissions have been extracted as features, and then have been analized by four machine learning methods: AdaBoost, Native Bayes, Decision tree, SVM. This work has used the mentioned approach to study the applications, and their results show that more than 80 percent of the malwares have been detected by their developed approach.

Talha et. al [6] have proposed a detection system called APK Auditor. This system has used a central server where each user sends a list of permissions to the server or a signature-based datacenter. The APK Auditor uses these information to detect if the program is malicious or not. To keep the system up-to-date, it automatically collects new applications added to the Google play store, and then gathers their information and stores them inside the datacenter.

Arp et. al [7] have proposed Drebin project, a static method for the malware detection in the android. Drebin have extracted permissions, hardware components, software components, API calls and network addresses, and then have used them as features in machine learning. They have collected 5560 malwares in their dataset, and they have used SVM algorithms as a classification method in their detection approach. The proposed method's accuracy is reported to be 94 percent.

Yerima et. al [8] have proposed three classification approaches based on Bayes as follows: Permission-based Bayesian classifier, Code-based properties Bayesian classifier and a combination of both. First, a score is calculated for each feature extracting from AndroidManifest.xml and other code parts. Then, the top 20 highest scored features have been included in the final feature list to be used with the Bayes classification algorithm to detect malware application.

S.Gate et. al [9] have introduced a novel concept for the android application called risk, and then have elaborated on the risk calculation process for each application. In this work, 26 most sensitive permissions have been found to be the most frequently used ones by the malwares, there 26 have been the critical permissions. They have used these critical permissions to calculate the application's risk. Then, they conclude that any application with a high risk score is most probably a malware. They have also defined a numeric risk indicator for the users to realize how harmful any application can be.

Milosevic et. al [10] have used two methods of machine learning using the classification to detect malwares based on the permission analysis. This study suggests that the malware detection based on the signature is not accurate enough. Their proposed method's accuracy based on the permission usage is reported to be 89 percent. Also their method utilizing the API calls has provided 95 percent of accuracy in malware detection.

F.Ghaffari et. al [11] have proposed a new method considering the previous methods' weaknesses. They have introduced the AMD-Ec method where they use the Ensemble classifiers. They have classified each application multiple times to improve the method's accuracy rate to reach 99.47 percent.

Shahriar et. al [12] have introduced a new approach to detect malwares based on the sensitive permissions. The reliable permissions related to each classification are detected by the LSI analysis method. This approach presents potentials for detecting new malwares. They have used two open source datasets for their experiments, and their method has been reported to classify these datasets with 80 percent accuracy rate.

Shang, et al. [13] have used the Naive Bayes machine learning for the malware detection, and then to increase the algorithm's accuracy rate. This work has analyzed the relations among permissions. They have first calculated Pearson correlation coefficient for each permission, and then permissions with lower correlation coefficient have been omitted from the classification process to improve malware detection based on the clustering method. Accuracy rate for this method has been reported to be 87 percent. In [33] features at four levels including kernel, application, user, and package are evaluated and analyzed in order to detect and stop malicious behaviors of an android malware. A user centric approach to analyze security risk of android application is devised in [34]. Moreover, the authors discuss a usable approach to evaluate the trustworthiness of android apps. Over privilege nature of android component is the root cause of many security attacks in android operating system. Therefore, in [35] an automated approach named DELDROID has been proposed to determine and to enforce least privilege architecture in this operating system. Consequently, this approach reduces the attack surface and thus enhances android security.

3. PROPOSED METHOD

In this section, we discuss our proposed method. We have approached malware detection as a machine learning problem. We have extracted features from the applications such as API calls, Intents, network addresses and IPs, permissions and services, activities and so on. We have used these extracted features in

machine learning to improve our algorithm to classify each new applications into two categories of malware and benign. First, we have extracted features from the application resource, and then we have detected the most repetitive features. The extracted features as well as the prevalent complex features, extracted from the applications' resources, have been converted to binary vectors. Then, to reduce the number of the extracted features, we have proposed a feature selection algorithm. Using this feature selection algorithm, we have found the most usable features for malware detection purposes and decreased the overhead and the operation time of the detection. After the selection of the most usable features, we have mapped these features into the binary vectors for each application. Finally, the previously created binary vectors have been sent as inputs to Weka [14] that uses machine learning algorithm to assess and evaluate these features to create a model for the malware detection. In the following sections, we explain each step in detail:

3.1. Feature Extraction

To access information required in detection process, we had to extract apk, access, Android Manifest.xml and classes.dex files, respectively. For this purpose, we have developed a program in Java and have added apktool [15] and dex2jar library to it to make the reverse engineering process automatic. The Java program gets the apk files and with the use of the reverse engineering extracts resources in the apk file. The APK's resources are extracted by the apkkool tool,

As the next step, we have parsed the tags in the manifest file; these tags are related to components such as permissions, activities' services, providers, Intents and Broadcast-receivers. These tags are used as features in the succeeding steps. A sample of these features from the manifest file is shown in Figure (1).

```
permission::android.permission.ACCESS_FINE_LO
CATION
activity::.results.Tabs
activity::SelectMetrics
intent::android.intent.category.LAUNCHER
permission::android.permission.INTERNET
```

Figure 1. A sample file which includes extracted features from application.

Each row defines a feature used inside the apk of an application. Later on, these features are used in the classification process. These features have the following format:

$$\langle FeatureType \rangle :: \langle FeatureName \rangle \quad (1)$$

The saved feature in file are consisted of two parts: (1) the type of feature and (2) the name of the used feature; these two parts are separated with two semicolons to be more readable.

Second feature categories that we use in the malware detection process are extracted from the classes.dex file in the apk package. By the de-compilation of the classes.dex byte code file, we can access the application's Java codes. Then, we can extract the class's names and API calls to be used as features in the classifications. To use a specific method in android applications, they should be called as follows:

$$\langle class \rangle \rightarrow \langle methodName \rangle (\langle paramTypes \rangle) \langle returnType \rangle \quad (2)$$

The Class describes a Java class containing the intended method declared with L(fullClasssName); for example, Lcom/example/MainActivity is a full name of a class. methodName declares an intended method from the classes. paramTypes indicates the type of the input data for the called method, and finally return type indicates expected data type from the intended function. Input and output data type can be of primitive data type or any other types such as a class. If it is primitive type, then it is written in abbreviation form. Character list for each primitive data is shown in Table (1). For example, a method declaration is written as follows:

$$\text{Lcom/example/MainActivity;} \rightarrow \text{setTheme(I)V} \quad (3)$$

This example means the setTheme method from Lcome/example/MainActivity class with an Integer input has been recalled and doesn't have a return value. As explained above, receiving the API calls of an application is not difficult; we cannot use all the called functions as features for the classification process as many classes and functions with similar names but with a different implementation may exist. Therefore, we only use the classes and functions that are native in the android OS. Native classes' name in the android OS starts with Landroid or LJava, hence we have extracted functions with Landroid and LJava in their name. Also we have used regular expression to extract network addresses and IPs from the application's code. Both feature categories extracted from Java classes and manifest are saved in a text file with.txt format so that we can use them later in the machine learning phase.

3.2. Feature Vector Drawing

As the previous step, we have several files with .txt format; each file contains the features used by an application. We have created a second program for reading these txt files. Our script in Java reads all these files and maps them into a vector. For example, by reading a file X, a vector represented as V(x) is created as follows:

$$V: X \rightarrow \{0,1\}^f , V(X) \rightarrow (I(x, f)) \quad (4)$$

In this equation, I(x,f) is described as follows:

$$I(x,f) = \begin{cases} 1 & \text{if the application x contains feature f} \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

Hence, we can show one feature vector as follows:

$$V(x) = \{0,1,1,0,1,0,1, \dots\} \quad (6)$$

"1" indicate that the application uses that feature and "0" indicate that the application doesn't use that feature. In addition to these features, we have added another feature called is_malware to the vectors; this feature is "1" for malwares and "0" for the benign applications. Finally, we have added vectors of each application to an array, so that we have an array of vectors where each cell indicates vector of features related to an application.

3.3. Feature Selection

Large number of features doesn't help the machine learning process, hence leading to more complexity causing less accurate classification algorithm [16]. Hence, we have used feature selection algorithms to reduce the number of features. These algorithms follow three goals: efficiency incensement of the learning algorithm, speeding modeling process by lowering the computational overload, reaching a deep perception from the main processes. [17] In this work, we have used two common methods including Chi-square (x^2) [18] and information gain [19]. In addition, we have also proposed another way to select features; in the following section, we describe our proposed method.

3.4. Proposed Method

As mentioned previously, selecting an exact feature not only reduces the computational overload but also increases the accuracy rate of the modeling and classification phases. We need to select features that can clearly distinguish between the two classes of malware and benign. The simplest way to select such features is choosing the features used most frequently in each class. The number of times or usage frequency f_i for feature i in malware class C_m is represented by $N_{f \in C_m}$ and the number of times this feature used in benign class C_b is represented by $N_{f \in C_b}$. We should select features used the most in their own classes. Otherwise, selecting features with this method is not useful because some features may still be used more often in malwares and benign class simultaneously. Hence, the algorithm may not be able to distinguish them. Therefore, selecting features only based on the frequency of usage is inadequate and impractical.

(1) One other way is to use the ration of the usage of a feature in malware to that of the same feature in benign application, but this criterion does not work appropriate enough either. If we assume the number of using feature f_i in a malware class C_m is shown with $N_{f \in C_m}$ and number of using the same feature in benign class C_b is shown with $N_{f \in C_b}$ and their ratio is D_m , then this ratio is calculated as follows:

$$D_M = \frac{N_{f_i C_m}}{N_{f_i C_b}} \quad (7)$$

As the ratio in relation shown in (7) gets assigned with a larger value, the number of usage times of that feature in malware class is more than the benign class. If this ration are equals 1 ,then it means number of using this feature in both classes is the same. If the ration is less than 1, then it means the feature is most probably in a benign class. This equation is sown in the following:

$$D_M = \begin{cases} 0 \le x < 1, & N_{f_i C_m} < N_{f_i C_b} \\ 1, & N_{f_i C_m} = N_{f_i C_b} \\ x > 1, & N_{f_i C_m} > N_{f_i C_b} \end{cases} \quad (8)$$

Although D_m seems to be a criterion, it doesn't represent the abundance of the feature in one class. For example, although one feature seems to be used more frequently in the malware class than in the benign applications, malwares used it rarely. This circumstance shows that the feature is not qualified to be chosen as a selected feature.

Another parameter that may be used for selecting features is the ratio of using on feature in each class to the ratio of using on feature in whole applications. Considering f_i representing the total number of times feature i is used in both classes, then we have the followings:

$$D_T = \frac{N_{f_i C_{(m,b)}}}{N_{f_i C_T}} = \frac{N_{f_i C_{(m,b)}}}{N_{f_i C_m} + N_{f_i C_b}} \quad (9)$$

This above-mentioned feature is not qualified to be chosen as a selected feature similar to the previous cases because if the number of features in one of the classes is zero, then D_T is always 1 meaning that the algorithm always select that feature that is not considered useful for our purpose.

Therefore, none of these three parameters is proper enough individually for selecting the features. Hence, combining these three parameters can introduce a proper criterion. We suggest the selection of features meeting the following three criteria:

1- The number of times a specific feature is used in a specific class (benign or malware) to the total number all classes provides a ratio more than a special threshold α.

$$D_T = \frac{N_{f_i C_{(m,b)}}}{N_{f_i C_m} + N_{f_i C_b}} > \alpha \quad , \quad \alpha > 0.5 \tag{10}$$

Value of α should be more than 0.5 because if it equals to 0.5, then the number of malwares and benign applications employing this feature is equal, and hence the algorithm cannot properly distinguish.

2- The ratio of the number of using times for that feature in one class to another class is more than a threshold β.

$$\begin{cases} D_{\frac{m}{b}} = \dfrac{N_{f_i C_m}}{N_{f_i C_b}} > \beta \quad , \quad \beta > 1 \\ D_{\frac{b}{m}} = \dfrac{N_{f_i C_b}}{N_{f_i C_m}} > \beta \quad , \quad \beta > 1 \end{cases} \tag{11}$$

The value of β should be more than 1 because if it equals to 1 then it means the number of malwares and benign employing this feature is equal, so the algorithm can't fully distinguish.

3- The percentage of the number of applications, which use that specific feature in one class to number of all applications that does not use that feature, should be more than ϒ.

$$\frac{N_{f_i C_{(m,b)}}}{(\text{total malware} - N_{f_i C_{(m,b)}})} * 100 > \gamma \tag{12}$$

The value of ϒ has direct proportion to the dimensions of the available dataset. Here, the value of ϒ is set to 0.08 which shows that the feature must be appear in at least five application of each group in order to be useful for classification.

The Pseudo-code of the feature selection algorithm is shown below:

Input: F,α,β,γ

Output: SF

1: for $i \rightarrow 0$ to F.size() do
2: $N_T \leftarrow countInTotal(f_i)$;
3: $N_M \leftarrow countInMalware(f_i)$;
4: $N_B \leftarrow countInBenign(f_i)$;
5: $D_{T(m)} \leftarrow \frac{N_M}{N_M+N_B}$;
6: $D_{T(b)} \leftarrow \frac{N_B}{N_M+N_B}$;
7: $D_M \leftarrow \frac{N_M}{N_B}$;
8: $D_B \leftarrow \frac{N_B}{N_M}$;
9: if $D_{T(m)} > \alpha$ && $D_M > \beta$ && $N_M > \gamma$ then
10: $SF \leftarrow f_i$;
11: else $SF \nleftarrow f_i$;
12: Endif
13: if $D_{T(b)} > \alpha$ && $D_B > \beta$ && $N_B > \gamma$ then
14: $SF \leftarrow f_i$;
15: else $SF \nleftarrow f_i$;
16: Endif
17: Endfor
18: return(SF)

Pseudocode 1. feature selection algorithm

The above-mentioned Pseudocode (1) shows how features are selected. This function has 4 inputs including the followings: (1) F, the total number of the extracted features, (2) α, β, ϒ, the thresholds for all three criteria. The output of this function is SF showing that the selected features with fewer dimensions than F because features that are not usable have been deleted. In lines 1 through 4, occurrence of features in total number of applications, in malwares, and in benign apps are computed. In lines 5 through 8, the above mentioned equations are evaluated. In lines 9 through 17, valuable feature based on the pre-determined thresholds as described previously are obtained and added to the set of selected features. Finally the set of selected features is returned in line 18. As shown in the Pseudo-code, the dimensions and the number of the selected features, relate to the thresholds. If the threshold becomes smaller, the accuracy decreases and the number of selected features increases. If the threshold becomes larger, the accuracy increases and the number of the selected features decreases. To compare the impact of the threshold on selecting features and classification, we have used a method called Hill climbing [20] to select random thresholds for the three parameters and to compare them together so that we can see the best threshold. In data analysis section, we elaborate this part. The selected features include those permissions, API calls, Intents, Network addresses, name of hardware and software components which are better distinguish malwares from benign apps. However, by using this approach all features have not equal impacts on the obtained classifier model.

In the implementation of the above feature selection algorithm, counting each feature in each set of applications, i.e., malwares, benign apps, and total apps requires a loop. Larges loop is belong to total apps with the value equal to N_T. Since There are F.Size() features.

The complexity of the above algorithm is equal to $O(N_T \times F.Size())$. If the maximum value of N_T and F.Size() is denoted by N, the complexity is belong to $O(N^2)$.
After selecting features, these features are written in a file and their vectors are made and saved in a file with .csv format. Each row in this file indicates an application and each column indicates the selected features represented with SF_i in Table (2). Each cell in this file includes a zero or one value showing whether this feature is used or not. We use this file as an input for the modeling. Table (2) shows an example of this file.

3.5. Adding Complex Features

After extracting features and selecting the best ones, we have detected the complex features. The goal is to detect a combination of features that may not solely harm but if used together it may harm the device. Hence, using the extracted features from the malwares and using an algorithm, we detect the most common complex features in the malware class. In fact, after feature selection described in the previous section, remaining features are used to determine complex features. A complex feature contains two or more features. Therefore, first, complex features are generated using features were not selected in previous step. We have regarded a complex feature as an individual feature. Then, we calculate how often applications use each complex feature. At this level, we have the data showing the most common complex features in malwares and how many times each complex feature has been used in malwares. Then, according to equations (10), (11), (12), we have evaluated obtained complex features and detected those complex features meeting the above three criteria. In the other words, feature selection described in the previous step is repeated for complex features again. After selecting complex feature, we have added them to previous selected features, and then finally we have created the final vectors of the total extracted features. At this stage, final vectors are as demonstrated in Table (3) and each row indicates vectors related to the applications and each column indicates one selected feature from the collection of the selected feature. In this figure, simple features are represented as SF_i and complex features are shown with CSF_i.

3.6. Modeling

In this step, the created vector from applications is used by the classification algorithm as learning data. For this purpose, we use Weka tool that is an open source application. To achieve a better classification algorithm, considering the amount of the data and features, we could not assess and evaluate each classification algorithms. Hence, we have decided not to run the algorithm on the main dataset, instead we have selected a smaller data subset with 0.1 ration and run the algorithms on this selected data subset to choose the best classification algorithms on this dataset. In the following section, we describe the results in details.

4. EXPERIMENTS

In this section, we explain results from our classification algorithm selection, essential number of features for the classification process and complex features' efficiency in enhancing accuracy. The functionality of our proposed method has also been compared to that of the previous works.

4.1. Dataset and Test Environment

The dataset used in this paper is introduced in [3] which is used to develop a security tool named Drebin. This dataset includes 5561 malwares and 123446 benign applications dating from August 2010 to October 2012. All tests have been performed on a PC with Intel Cori7-4720HQ 2.60GHz processor, 16 GB Ram and a Win 10 operating system. The following Table(4) summarizes simulation settings of our experimentations:

4.2. Evaluation Criteria

To evaluate different classification algorithms in our dataset, we have used the following criteria:

- True Positive:

This criterion indicates malwares that have been detected successfully.

$$TP = \frac{TP}{TP + FN} \tag{13}$$

- False positive:

This criterion indicates the number of benign applications mistakenly detected as malwares.

$$FP = \frac{FP}{FP + TN} \tag{14}$$

- Precision:

The ratio of the number of malwares successfully detected to the total number of all the detected malwares . The equation for this ration is shown as follows (15):

$$Percision = \frac{TP}{TP + FP} \tag{15}$$

- Recall:

The number of malwares that was successfully detected to the number of all the malwares detected successfully or categorized unsuccessfully as harmless ratio:

$$Recall = \frac{TP}{TP + FN} \tag{16}$$

- F-Measure:

Harmonic mean from the precision and recall is the criterion indicating how capable the model act in distinguishing between the two classes.

$$\text{F-Measure} = 2 . \frac{percision . recall}{percision + recall} \quad (17)$$

- ROC Area:

This curve demonstrates sensitivity or successful forecasts in comparison with the unsuccessful ones in a binary classification with a dynamic separation threshold. The ROC curve is created by plotting the true positive rate (TP_{Rate}) against false negative rate (FP_{Rate}) at various threshold settings. Here, true positive rate is a fraction of applications which were truly identified by the model as malwares and false negative rate is a fraction of malwares which were not detected by the model. The area under the curve (AUC) shows a number measuring a part of functionality. AUC has a value ranging from 0 to 1 where 0.5 means a random prediction and 1 means an excellent prediction. [21]

$$AUC = \frac{1 + TP_{Rate} - FP_{Rate}}{2} \quad (18)$$

- Accuracy

This criterion shows the percentage ratio of the number of samples successfully classified to total number of samples.

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \quad (19)$$

4.3. Results of suggested classification algorithm selection

To select an appropriate algorithm for the classification, we have created a subset of the main dataset, so that using different tests, we can find the best algorithm for the main dataset. The subset contains 12000 applications and 500 malwares with 0.1 ration to the original dataset and a 1 to 22 ration among malwares and benign applications in original dataset, this ratio has been checked out in subset as well. Also, to decrease the extracted features, we have used X^2 algorithm and 500 features that had larger X^2 value have been chosen as the selected features for classification. Then, this collection has been evaluated by all classification algorithms in Weka application and the result is shown in Table (5). Various classification algorithm are used here to determine which classifiers are more compliant with our feature selection method in term of classification rate. As shown in Table (5), Kstar have presented the best functionality among all other algorithms. Kstar (K*) is a lazy instance-based classifier in which class label of every test instance is determined by its similarity to its neighbors. However, this is not as impressive as the second algorithm which is IBK. It is another simple instance-based classifier that uses the class of the nearest k training instances for the class of the test data. The superiority of Kstar is only because of 6 extra successful predictions compared to IBK, but the execution time for the Kstar is almost 25 times longer than that of IBK. Therefore, we have removed Kstar algorithm from the selected algorithm list, and finally four algorithms of IBK, SMO, Randomforest and Randomcomittee are selected because of their prominent functionality and execution time. SMO(Sequential Minimal Optimization) is a improved version of SVM(Support Vector Machine) classifier which uses an special optimization approach for training the classifier. Randomforest and Randomcomittee are both ensemble classifiers. Randomforest is an extension of bagging decision trees that can be used for classification. Randomcomittee uses a committee of random classifiers. Random classifiers are generated using a base learner. For more descriptions regarding to classifiers used in this study, interested readers are referred to Weka documentation.

4.4. Feature selection

Because of the high number of features and applications in the dataset, we have required selection on features. Hence, we have employed feature selection algorithms to select these features. These algorithms calculate the distinction of each feature and rate them with a number. These ratings features are sorted and those with higher ratings are selected. Now we should consider how many features is enough to achieve the best accuracy and meanwhile has an appropriate execution time. Hence, we have used Hill climbing method to calculate the number of features. To determine the influence of the selected features, we first sort the extracted features by X^2 and information gain methods. First, we have selected 100 top features and then classified them with two algorithms of randomforest and randomcomittee and saved the results. Next, we have increased selected features by 100 and wrote down the results. We did this till accuracy growth started to reduce. This process has been continued with up to 900 selected features and after that, accuracy haven't presented any change and finally leading to computational overload. The results are shown in Table (6).

Table (6) shows that despite of deleting 98.8 percent of the total extracted features and using only 1000 selected features, we achieve better results compared to those of the Drebin method that employs all the extracted features. Besides this improvement, computational overload and execution time for the process and modeling have decreased significantly. Also, as shown in Table (6), when we increase the number of features, we observe a constant improvement in accuracy for up to using 800 features. After 800th feature, we haven't observed any significant improvement, although in some cases, we had retrogression. Aa a chance to improve the

result, we further increase the features but it won't be significant. For example as we increase features by 100, the final vector will have 129 million cells more than before, and this means computational overload and bigger execution time. Increasing the number of features to more than 1000 is not useful and selecting 700-800 features is more justifiable.

4.5. Complex feature's influence on classification

After the number of features were calculated, we can proceed and use complex features in addition to the simple features. For this purpose, we first extract the most common complex features in malwares and benign applications. Then, we delete the shared complex features among malwares and benign because this leads to worse distinction. Finally, we have selected 89 complex features. By adding this extra 89 features, we have examined our dataset again and the results , which show improvements, are shown in Table (7). By extracting complex features and using them for classification, with only 757 features, all the parameters had improvements. As shown in Table (6), the best result has been achieved by 900 features obtained via Randomforest algorithm. This algorithm uses 143 extra feature causing 18 million extra cells to be calculated. However, this has only detected 29 extra malwares. These tests results show the efficiency of our suggested method.

4.6. Comparison of the proposed method with previous papers

In this section, we discuss differences between our proposed method and other methods based using static analysis method. After declaring the appropriate algorithm for classification and the number of selected features, we have added complex features to get the final classification based on the selected algorithms and the results are presented in Table (8). This comparison is based on the accuracy parameter.As shown in Table (8), the proposed method provides a better accuracy in comparison with all the previous methods. Also, this method provides better results in comparison with all other previous classification methods.

5. CONCLUSIONS

Traditional antiviruses are signature based which are restricted to known malwares. However, by data mining and static analysis of malwares and clean apps, new threats can be found. In this paper, we have proposed a method that detects malwares by static analysis. The suggested method both reduces processing load and increases processing speed; nevertheless, it provides a better accuracy in classification of the applications into benign and malware categories. We have used two main feature groups. First, the simple extracted features from the codes including the followings: permissions, network addresses, API functions and main components of an android application. The second group are complex features made by combining simple features together that are more common in malwares. Because of the diversity of features, we have selected features with Chi-square and Information Gain algorithms. Finally, by the use of classification algorithms such as randomforest and randomcomittee, we have classified the applications and have compared them with the previous works. We have also investigated on the selection of appropriate algorithms for classification. We have also provided experimental results to find out how many features are needed for a classification. We have found out that increasing the number of the selected features doesn't necessarily improve classification, and hence causing computational overload and more modeling time. Therefore, by selecting the right number of features not only the computational overload decreases, but the accuracy of classification increases. Although, the proposed approach has been implemented and tested independently, it can be integrated to android operating system.

REFERENCES

[1] Y. Feng, S. Anand, I. Dillig and A. Aiken, "Apposcopy: semantics-based detection of Android malware through static analysis," *FSE 2014 Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering,* pp. 576-587, 2014.

[2] K. Tam, A. Feizollah, N. B. Anuar, R. Salleh and L. Cavallaro, "The Evolution of Android Malware and Android Analysis Techniques," *ACM Computing Surveys (CSUR),* vol. 49, no. 4, 2017.

[3] M. Y.Wong and D. Lie, "IntelliDroid: A Targeted Input Generator for the Dynamic Analysis of Android Malware," in *Network and Distributed System Security Symposium*, 2016.

[4] M. Fatima and M. Pasha, "Survey of Machine Learning Algorithms for Disease Diagnostic," *Journal of Intelligent Learning Systems and Applications,* pp. 1-16, 2017.

[5] C.-Y. Huang, Y. Tsai and C.-H. Hsu, "Performance Evaluation on Permission-Based Detection for Android Malware," *Advances in Intelligent Systems and Applications,* vol. 2, pp. 111-120, 2013.

[6] K. A. Talha, D. I. Alper and C. Aydin, "APK Auditor: Permission-based Android malware detection," 2015.

[7] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon and K. Rieck, "DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket," 2014.

[8] S. Y.Yerima, S. Sezer and G. McWilliams, "Analysis of Bayesian classification-based approaches for Android malware detection," 2013.

[9] C. S.Gates, N. Li, H. Peng, B. Sarma, Y. Qi, R. Potharaju, C. Nita-Rotaru and I. Molloy, "Generating Summary Risk Scores for Mobile Application," *IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING,* vol. 11, 2014.

[10] N. Milosevic, A. Dehghantanha and K.-K. Raymond choo, "Machine learning aided Android malware

classification," *Computers and Electrical Engineering,* 2017.

[11] F. Ghaffari, M. Abadi and A. Tajoddin, "AMD-EC: Anomaly-based Android Malware Detection using Ensemble Classifiers," in *Iranian Conference on Electrical Engineering*, Tehran, Iran, 2017.

[12] H. Shahriar, M. Islam and V. Clincy, "Android Malware Detection Using Permission Analysis," in *Southeast Conference*, Charlotte, NC, USA, 2017.

[13] F. Shang, Y. Li, X. Deng and D. He, "Android malware detection method based on naive Bayes and permission correlation algorithm," *Cluster Computing,* pp. 1-12, 2017.

[14] U. o. Waikato, "Weka 3: Data Mining Software," University of Waikato,GNU General Public License, 4 9 2018. [Online]. Available: https://www.cs.waikato.ac.nz/ml/weka/.

[15] C. Tumbleson and R. Wiśniewski, Writers, *Apktool.* [Performance]. Apache, 2010.

[16] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer and Y. Weiss, ""Andromaly": a behavioral malware detection framework for android devices," *Journal of Intelligent Information Systems,* vol. 38, no. 1, pp. 161-190, 2012.

[17] J. Bai, J. Wang and G. Zou, "A Malware Detection Scheme Based on Mining Format Information," *The Scientific World Journal,* 2014.

[18] F. Thabtah and M. A. H.Eljinini, "Naïve Bayesian Based on Chi Square to Categorize Arabic Data," *Communications of the IBIMA,* 2009.

[19] A. K. Uysal and S. Gunal, "A novel probabilistic feature selection method for text classification," *Knowledge-Based Systems,* vol. 36, pp. 226-235, 2012.

[20] S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach(2nd ed.), Upper Saddle River, New Jersey: Prentice Hall, 2003.

[21] M. Galar, a. Fernandez, E. Barrenechea, H. Bustince and F. Herrera, "A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches," *IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS,* vol. 42, no. 4, pp. 463-484, 2012.

[22] Y. Aafer, W. Du and H. Yin, "DroidAPIMiner: Mining API-Level Features for Robust Malware Detection in Android," *Security and Privacy in Communication Networks,* pp. 86-103, 2013.

[23] B. Anderson, C. Storlie and T. Lane, "Improving malware classification: bridging the static/dynamic gap," *AISec '12 Proceedings of the 5th ACM workshop on Security and artificial intelligence,* pp. 3-14, 2012.

[24] M. Nezhadkamali, S. Soltani and S. A. H. Seno, "Android malware detection based on overlapping of static features," in *7th International Conference on Computer and Knowledge Engineering (ICCKE 2017)*, Mashhad, 2017.

[25] X. Liu and J. Liu, "A Two-Layered Permission-Based Android Malware Detection Scheme," *Mobile Cloud Computing, Services, and Engineering (MobileCloud),* pp. 142-148, 2014.

[26] M. Schultz, E. Eskin, F. Zadok and S. Stolfo, "Data Mining Methods for Detection of New Malicious Executables," *IEEE Symposium on Security and Privacy,* pp. 38-49, 2001.

[27] I. Firdausi, C. lim, A. Erwin and A. S. Nugroho, "Analysis of Machine learning Techniques Used in Behavior-Based Malware Detection," *Second International Conference on Advances in Computing, Control, and Telecommunication Technologies,* pp. 201-203, 2010.

[28] Z. Yuan, Y. Lu and Y. Xue, "DroidDetector: Android Malware Characterization and Detection Using Deep Learning," *TSINGHUA SCIENCE AND TECHNOLOGY,* vol. 21, no. 1, pp. 114-123, 2016.

[29] M. Siddiqui, M. C. Wang and J. Lee, "Detecting Internet Worms using Data Mining Techniques," *Journal of Systemics, Cybernetics and Informatics,* pp. 48-53, 2010.

[30] S. Hahn, M. Protsenko and T. Müller, "Comparative evaluation of machine learning-based malware detection on android," *Gesellschaft für Informatik,* pp. 79-88, 2016.

[31] A. Sharma and S. K. Dash, "Mining API Calls and Permissions for Android Malware Detection," *Cryptology and Network Security,* pp. 191-205, 2014.

[32] Z. Aung and W. Zaw, "Permission-Based Android Malware Detection," *INTERNATIONAL JOURNAL OF SCIENTIFIC & TECHNOLOGY RESEARCH,* vol. 2, no. 3, pp. 228-234, 2013.

[33] Saracino, A., Sgandurra, D., Dini, G., & Martinelli, F. (2018). Madam: Effective and efficient behavior-based android malware detection and prevention. IEEE Transactions on Dependable and Secure Computing, 15(1), 83-97.

[34] Dini, G., Martinelli, F., Matteucci, I., Petrocchi, M., Saracino, A., & Sgandurra, D. (2018). Risk analysis of Android applications: A user-centric solution. Future Generation Computer Systems, 80, 505-518.

[35] Hammad, M., Bagheri, H., & Malek, S. (2019). DelDroid: An automated approach for determination and enforcement of least-privilege architecture in android. Journal of Systems and Software, 149, 83-100.

Table 1. declaration method for primitive data type when recalling function

Data type	abbreviation
Void	V
Boolean	Z
Byte	B
Short	S
Char	C
Int	I
Long	J
Float	F
Double	D

Table 2. An example of the table made by these features in an excel file

	SF0	SF1	SF2	SF3	…	SFN
App1	0	1	0	1	…	0
App2	1	1	1	1	…	0

Table 3. Binary vector created from apps by simple and complex features

	SF_0	SF_1	SF_2	SF_3	…	SF_N	CSF_0	….	CSF_n
App_1	0	1	0	1	…	0	1	…	1
App_2	1	1	1	1	…	0	1	…	0
App_n	0	0	1	0	…	1	0	…	1

Table 4. Experimental settings

CPU	Intel core i7-4720HQ 2.60GHz
RAM	16GB
OS	Windows 10 64bit
Data mining tool	Weka 8.3
Feature extraction tools	apktool, dex2jar
Evaluation method	10-fold cross validation
Number of Malwares	5561
Number of benign apps	123446

Table 5. The results of different classification algorithms by Weka

	TP rate	*FP rate*	*Percision*	*Recall*	*F-measure*	*ROC*	*Time (s)*	*Correct classified*	*Classified*	*False*
BayesNet	95	23.6	96.7	95	95.6	94.1	0.84	11869	631	
NaiveBayes	95.1	23.6	96.7	95.1	95.7	94	0.08	11883	617	
NaiveBayesUpdateable	95.1	23.6	96.7	95.1	95.7	94	0.04	11883	617	
logistic	98.4	26.9	98.3	98.4	98.3	91.4	30.75	12297	203	
SGD	98.6	27.5	98.5	98.6	98.5	85.6	6.48	12323	177	
SimpleLogistic	98.5	28.4	98.4	98.5	98.4	95.8	41.2	12310	190	
SMO	98.6	28.4	98.5	98.6	98.5	98.5	72.8	12325	175	
votedPerceptron	98.1	36.3	97.9	98.1	97.9	81.1	1.76	12260	240	
IBK	98.8	20.8	98.8	98.8	98.8	95	12.28	12350	150	
Kstar	98.8	19.4	98.8	98.8	98.9	93.9	317	12356	144	
LWL	96	95.2	96.2	96	94.1	93.8	381.56	12004	496	
adaBoostM2	96.6	68.2	95.9	96.6	95.9	92	4.42	12073	427	
AttributeSelectedClassifier	98.1	36.5	98	98.1	98	88.2	16.99	12267	233	
bagging	98.3	31.7	98.2	98.3	98.2	96.3	47.92	12290	210	
ClassificationViaRegression	98.1	34	97.9	98.1	98	94.2	28.08	12259	241	
FilteredClassifier	98.3	31.1	98.2	98.2	98.2	87.8	43.78	12292	208	
IterativeClassifierOptimizer	97.3	48.8	97	97.3	97	93.6	64.55	12162	338	
LogitBoost	97.3	48.8	97	97.3	97	93.6	5.32	12162	338	
MultiClassClassifier	98.4	26.9	98.3	98.4	98.3	91.4	28.07	12297	203	
MultiClassClassifierUpdateable	98.6	27.5	98.5	98.6	98.5	85.6	5.96	12323	177	
RandomCommittee	98.7	26.1	98.7	98.7	98.6	95.2	6.22	12340	160	
RandomizableFilterdClassifier	98.1	30.7	98	98.1	98	91.9	0.23	12261	239	
RandomSubSpace	98.2	37.1	98.2	98.2	98.1	95.9	34.07	12281	219	

Jrip	98.1	26.3	98	98.1	98.1	97.6	26.14	12261	239
OneR	96.4	86.2	96.3	96.4	95	55.1	0.42	12048	452
PART	98.3	23.1	98.2	98.3	98.3	91.5	53.54	12284	216
HeoffdingTree	97.8	42.3	97.5	97.8	97.5	86.1	1.45	12219	281
J48	98.3	31.1	98.2	98.3	98.3	87.8	38.57	12292	208
LMT	98.4	26.9	98.3	98.4	98.3	93.5	1968.24	12299	201
Randomforest	98.7	26.5	98.7	98.7	98.6	97	27.86	12339	161
RandomTree	98.2	23.8	98.1	98.2	98.2	87	0.58	12269	231
REPTree	98	37.6	97.9	98	97.9	93.3	7.01	12252	248

Table 6. Investigating the effect of increasing the number of features on classification accuracy

Feature selection Algorithem	*Number of Features*	*Algorithm*	*TP rate*	*FP rate*	*Percision*	*Recall*	*F-measure*	*ROC*	*Time (s)*	*Correct classified*	*Classified False*
X^2	100	*RandCommittee*	98.9	20	98.8	98.9	98.8	95.9	48.81	127526	1481
		Random forrest	98.9	19.6	98.8	98.9	98.8	96.9	325.4	127542	1465
IG		*RandCommittee*	99.2	14.9	99.1	99.2	99.1	97.9	16.22	127923	1084
		Random forrest	99.2	14.5	99.2	99.2	99.2	99.1	91.13	127952	1055
Proposed		*RandCommittee*	99	17.1	99	99	99	97.6	13.97	127751	1256
		Random forrest	99	17.5	99	99	99	98.8	81.22	127762	1245
X^2	200	*RandCommittee*	99.1	15.3	99.1	99.1	99.1	97.2	106.5	127893	1114
		Random forrest	99.2	14.8	99.2	99.2	99.1	98.3	569.7	127935	1072
IG		*RandCommittee*	99.3	12.8	99.3	99.3	99.3	99.2	24.71	128092	915
		Random forrest	99.3	12.7	99.3	99.3	99.3	99.2	168	128100	907
Proposed		*RandCommittee*	99.2	15.5	99.1	99.2	99.1	97.7	21.87	127922	1085
		Random forrest	99.2	15.9	99.1	99.2	99.1	98.9	135.4	127927	1080
X^2	300	*RandCommittee*	99.3	11.7	99.2	99.3	99.2	97.7	111.9	128049	958
		Random forrest	99.3	11.4	99.3	99.3	99.3	98.5	569.7	128073	934
IG		*RandCommittee*	99.3	12.7	99.3	99.3	99.3	98.2	37.43	128088	919
		Random forrest	99.3	12.5	99.3	99.3	99.3	99.2	240.4	128116	891
Proposed		*RandCommittee*	99.1	16.1	99.1	99.1	99.1	97.8	37.2	127872	1135
		Random forrest	99.2	15.7	99.2	99.2	99.2	99	221.4	127936	1071
X^2	400	*RandCommittee*	99.3	11.4	99.3	99.3	99.4	97.8	125	128092	915
		Random forrest	99.3	11.1	99.3	99.3	99.3	98.8	943.2	128130	877
IG		*RandCommittee*	99.3	12.7	99.3	99.3	99.3	98.2	39.63	128077	930
		Random forrest	99.3	12.6	99.3	99.3	99.3	99.3	287.6	128099	908
Proposed		*RandCommittee*									
		Random forrest									
X^2	500	*RandCommittee*	99.3	11.1	99.3	99.3	99.3	98	134.7	128144	863
		Random forrest	99.4	10.6	99.4	99.4	99.4	99	668.1	128194	813
IG		*RandCommittee*	99.3	12.2	99.3	99.3	99.3	98.2	50.8	128109	919
		Random forrest	99.3	12.5	99.3	99.3	99.3	99.3	322.9	128110	897
Proposed		*RandCommittee*	99.1	15.6	99.1	99.1	99.1	97.7	52.77	127905	1102
		Random forrest	99.2	15.9	99.2	99.2	99.1	99	292.4	127912	1095
X^2	600	*RandCommittee*	99.4	10.9	99.3	99.4	99.3	97.9	134.8	128175	832
		Random forrest	99.4	10.5	99.4	99.4	99.4	99	774.7	128204	803
IG		*RandCommittee*	99.3	12.2	99.3	99.3	99.3	98.2	59.69	128096	911
		Random forrest	99.3	11.9	99.3	99.3	99.3	99.3	386.4	128133	874

Proposed		*RandCommittee*	99.3	12.9	99.3	99.3	99.3	98	62.9	128064	943
		Random forrest	99.3	13.2	99.3	99.3	99.3	99.1	389.1	128065	942
X^2	700	*RandCommittee*	99.4	10.7	99.3	99.4	99.3	98.2	132.5	128181	826
		Random forrest	99.4	10.3	99.4	99.4	99.4	99.1	782.4	128215	792
IG		*RandCommittee*	99.3	12	99.3	99.3	99.3	98.3	71.57	128120	887
		Random forrest	99.3	11.7	99.3	99.3	99.3	99.3	363.1	128150	857
Proposed		*RandCommittee*	99.3	13.1	99.3	99.3	99.3	97.9	78.77	128060	947
		Random forrest	99.3	13.1	99.3	99.3	99.3	99.1	389.1	128072	935
X^2	800	*RandCommittee*	99.4	11	99.3	99.4	99.3	98.2	166.5	128174	833
		Random forrest									
IG		*RandCommittee*	99.3	11.9	99.3	99.3	99.3	98.2	80.2	128118	889
		Random forrest									
Proposed		*RandCommittee*	99.3	12.8	99.3	99.3	99.3	98	78.7	128085	922
		Random forrest	99.3	12.5	99.3	99.3	99.3	99.2	599	128112	895
X^2	900	*RandCommittee*	99.4	10.3	99.4	99.4	99.4	98.2	178.3	128223	784
		Random forrest									
IG		*RandomCommittee*	99.4	11.8	99.3	99.3	99.3	98.3	89.36	128115	892
		Random forrest									
Proposed		*RandCommittee*	99.3	12.7	99.3	99.3	99.3	98	93.6	128092	915
		Random forrest	99.3	12.5	99.3	99.3	99.3	99.2	548	128116	891
X^2	1000	*RandCommittee*									
		Random forrest									
IG		*RandCommittee*	99.3	11.8	99.3	99.3	99.3	98.3	92.3	128133	874
		Random forrest									
Proposed		*RandCommittee*	99.3	12.4	99.3	99.3	99.3	98	101.4	128112	895
		Random forrest	99.3	12.5	99.3	99.3	99.3	99.2	666.8	128119	888

Table 7. results of classification based on complex features

Feature selection Algorithem	Number of Features	TP rate	FP rate	Percision	Recall	F-measure	ROC	Time (s)	Correct classified	Classified False
randomForest	668+89	99.4	10.6	99.4	99.4	99.4	98.6	1924.61	128194	813
randomCommitee		99.3	10	99.3	99.3	99.3	97.7	393.12	128158	849
IBK		99.3	9.4	99.3	99.3	99.3	96.8	478	128097	910
SMO		98.9	21.7	98.9	98.9	98.9	88.6	9727	127571	1436

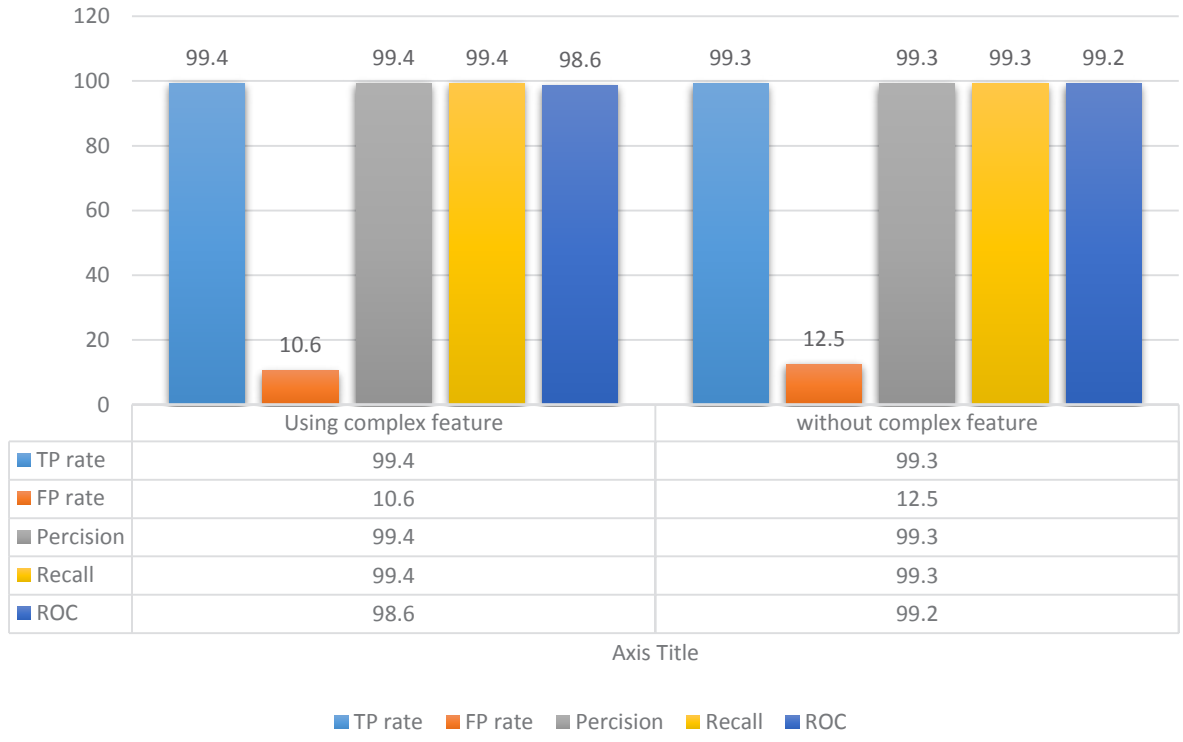

Figure 2. The chart of functionality between classification with complex features and without them

Table 8. Comparison of the proposed method's accuracy

Malware Detection Method [۱]	*Algorithm*	*Accuracy*	*TP rate*	*FP Rate*	*Precision*	*Recall*	*F-measure*	*ROC Area*
This work	*Random Forest*	99.36	99.4	10.6	99.4	99.4	99.4	98.6
Droidapiminer *[22]*	*KNN*	99	97.8	-	-	-	-	-
Anderson et al. *[23]*	*SVM*	98.7	-	-	-	-	-	-
Nezhadkamali et al. *[24]*	*Random Forest*	98.6	98.6	11	99.7	99	98.3	99.7
Xing et al. *[25]*	*J48*	98.6	80.5	5	89.8	55	68.2	-
Schultz et al. *[26]*	*Navies Bayes*	97.11	97.43	3.8	98.7	97.2	97.9	-
Firdausi et al. *[27]*	*J48*	97	90.9	3.8	95.9	-	-	-
DroidDetector *[28]*	*RBM*	96.76	-	-	97.79	95.68	-	-
Siddiqui et al. *[29]*	*Random Forest*	96.6	95.6	3.8	-	95.6	-	-
Sebastian et al. *[30]*	*Random Forest*	96.02	96	3.95	-	-	-	-
Sharma et al. *[31]*	*KNN*	95.1	-	-	94.3	99	95.1	-
Drebin *[7]*	*SVM*	93	-	-	-	-	-	-
Aung et al. *[32]*	*CART*	90.7	97.8	15.7	84.9	97.8	-	87