An efficient Android malware detection method using BorutaShap algorithm

Sandeep Sharma¹*, Prachi¹, Rita Chhikara¹ and Kavita Khanna²

¹Department of Computer Science, The North Cap University, India;
²Delhi Skill and Entrepreneurship University, India

E-mail/Orcid Id:

SS, sandeep19csd004@ncuindia.edu; P, prachi@ncuindia.edu, https://orcid.org/0000-0001-5745-1206
RC, rita.chhikara@ncuindia.edu, https://orcid.org/0000-0001-9537-3907; KA, kavita.khanna@dseu.ac.in, https://orcid.org/0000-0001-5745-1206

Introduction

Android operating system (OS) is the extensively used OS in smart devices such as smartphones, tablets, smart TV’s, Android Auto and Smartwatches. First unveiled in 2007, it quickly became popular because it is open-source, commercially backed, and developed by Google. According to Statista (2003), Android is the main mobile OS with a 71.8% market share in the fourth quarter of 2022. This huge user base makes it a very lucrative target for attackers. In addition to the Google Play Store, users can also get applications from other third-party marketplaces. The open-source nature of Android OS, large user base and availability of applications on various third-party stores make Android users more prone to various malware attacks. Android malware can steal information, cause disruption in services, gain unauthorized access, modify data, spam users, or, in general, interfere with the security of a user’s device. According to Cyber Security News and Insights for Executives (2022), Android malware attacks are over 50 times higher than iOS malware attacks. Google describes Android malware as Potentially Harmful applications (PHAs). In 2018 research report by Google (Android, 2019), the PHAs are classified as Trojan, SMS fraud, phishing, hostile downloaders, spyware, privilege escalation, toll fraud, click fraud, backdoors and commercial spyware. Researchers have investigated several ways of detecting Android malware. These techniques are categorized as static and dynamic analysis-based techniques.

Static analysis-based detection methods reverse engineer the applications to retrieve static features such as permissions, intents, API calls, opcodes, etc (Zhang et al., 2020; Kazmi et al., 2023; Goel et al. 2023). Dynamic analysis-based detection methods execute applications on Android devices or emulators to observe the application’s behavior. Mainly, the dynamic analysis methods (Khan and Jain, 2020; Martinelli et al., 2017; Vinod et al., 2019;
Xu et al., 2019; Li et al., 2019; Kaleem et al., 2023; Vivekanandan and Gunasekaran, 2023) use a simulated environment that triggers the activity of malicious code (Alzaylaee et al., 2020) and monitors the behavior of the application. These techniques are much more effective in the case of obfuscated malware or dynamically loaded code. Thus, it can also detect novel malware based on their real-time activities. However, they fail to offer complete code coverage.

An efficient and robust Android malware solution should consider static and dynamic features because they are key attributes defining an application as benign or malicious. However, the extracted features are frequently afflicted by high dimensionality, leading to a considerable increase in computational overhead. Also, many features are redundant and irrelevant in nature. They increase the complexity and decrease the performance of a classification model. Therefore, this work proposes an Android malware detection model based on a few important static as well as dynamic features that can effectively differentiate malicious applications from benign ones. Contributions to this work are -

1. Extract hybrid features from benign and malicious APKs.
2. Reduce the dimensionality using Boruta, BorutaShap and Gini importance methods.

The rest of the paper is organized as follows:

Section 2 discusses the related work and section 3 presents the proposed methodology, section 4 includes experimental results and discussion and section 5 concludes the paper.

Related Work

Several researchers have explored the application of feature selection in the context of Android malware detection. Here is a summary of their findings and methods:

1. Deepa et al. (2015): They used the correlation feature selection method and achieved an accuracy of 88.75% in Android malware detection by mining the top few features.
2. Zhao et al. (2015): They introduced a novel feature selection technique that considered the frequency of features in benign and malicious Android samples. Their method achieved up to 98% accuracy by selecting distinct features.
3. Wen and Yu (2017): They extracted hybrid features from applications and used Principal Component Analysis and Relief to extract promising features. Their method achieved 95.2% accuracy on malicious applications.
4. Altaher and Barukab (2017): They extracted permissions and API calls and applied filter-based feature selection techniques to rank features based on importance in distinguishing malicious samples from benign ones.
5. Bhattacharya and Goswami (2018): They presented a feature selection method and obtained 87.8% and 97.9% accuracy rates on two different datasets, claiming that their technique can perform well on large datasets.
6. Peynirci et al. (2020): They selected features with the highest Inverse Document Frequency (IDF) in benign applications and the lowest IDF in malicious applications, achieving above 99% accuracy on three different datasets.
7. Dhalaria and Gandotra (2020): They extracted hybrid features from Android applications and used the Chi-Square feature selection method to select important features for detecting malicious Android applications.
8. Sahin et al. (2021): They applied eight different feature selection techniques and demonstrated that feature selection methods from text classification studies can improve classification model performance.
9. Kouliaridis et al. (2021): They used PCA and t-SNE with multiple machine learning algorithms to select features from the entire feature set and achieved an accuracy of 91.7%.

These studies highlight the importance of feature selection in enhancing the accuracy and efficiency of Android malware detection algorithms.

Proposed Method

The method followed in this work to detect malware and measure the accuracy of different algorithms can be divided into 3 steps. First, to create a dataset of static and dynamic features of APKs using tools such as MobSF, Strace and Androguard. Then, a feature set of corresponding features from APKs is created in which if a feature/attribute is present in a particular APK, then it is marked as 1 otherwise 0. Second, Boruta, BorutaShap and Gini Importance feature selection algorithms are applied to extract the important features from the feature set. This improves the performance of the classification model by removing redundant information. Lastly, Random Forest and AdaBoost classifiers are applied to the selected features to find the best possible approach for malware detection.

Dataset Collection

This step collects benign and malicious Android samples from two major sources: malware samples from AndroZoo repository (Allix et al., 2016) and benign
samples from Google Play Store. This work uses AndroZoo to collect malware samples because it is a comprehensive repository containing samples from various sources. Later on, samples are executed within Genymotion emulator and their dynamic features, such as system calls, are retrieved through Mobile Security Framework (MobSF) (Abraham, 2022) and the strace tool. Further, static features are extracted with the help of MobSF and Androguard (Androguard, 2019). The Androguard tool reverse-engineers the samples to obtain manifest.xml and classes.dex files. The following static features are obtained from these samples: opcode, permission, API call and intent filter.

**Feature Selection - Boruta and BorutaShap Algorithm**

Feature selection techniques are one of the best ways to handle the curse of dimensionality and reduce over-fitting in a machine-learning model. It makes the model simple and increases the interpretation of the model, highlighting the most informative features. This research work uses Boruta algorithm to select the most contributory features.

Boruta Algorithm works on the concept of creating duplicates referred to as shadow features of all independent features, as shown in Figure 2. The values are shuffled after that to remove the correlation between independent and dependent features. The algorithm works on the concept of top-down search. It compares the importance of original attributes with shadow features created through permutation and gradually eliminates the features that are not important. The features that are better than shadow features are retained.

The algorithm then computes a Z score, finds the greatest Z score, and tags variables as 'unimportant' or 'important' depending on whether they are significantly lower or higher than the highest Z score. This process is repeated until all attributes are either tagged 'unimportant' or 'important'.

Boruta can be combined with SHAP (SHapley Additive exPlanations) in a machine learning pipeline for feature selection and interpretation of the model. SHAP values are used to explain the contribution of each feature in a predictive model. They are used to give a summary of the importance of features.

**Performance Metrics**

This work uses the following metrics for evaluating the performance of ML algorithms.

**Precision**

This metric shows that out of predicted positive values (predicted malware), how many values were actually positive (actual malware).

\[
\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}
\]

**Recall**

Recall means ratio of predicted positives (predicted malware) out of total actual positives.

\[
\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}
\]

**F1-score**

F1-score is used to get a balanced metric out of precision and recall.

\[
F1 – \text{Score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}
\]

**Accuracy**

Accuracy defines the overall correctness of our model. It is the ratio of correct predictions out of all predictions made by the model.

\[
\text{Accuracy} = \frac{(\text{TP} + \text{TN})}{(\text{TP} + \text{FP} + \text{TN} + \text{FN})}
\]

**Experimental Results and Analysis**

This section includes various experiments and tests conducted on the dataset mentioned in the previous section, which includes 102 dynamic features and 915 static features on 190 APK samples. To maintain uniformity, the dataset is split into train and test for all the experiments at 60%:40%. The feature selection techniques Boruta, BorutaShap and Gini Importance filter the dynamic and static features that can detect Android Malware efficiently. The performance of selected features is compared with Random Forest and AdaBoost. The results and analysis of various experiments are as follows-

**Results of Feature Selection**

As reported in Table 1, the original dimensionality is significantly reduced with all the feature selection techniques with good accuracy. However, the BorutaShap with Random Forest Classifier outperforms all the other techniques. The number of features is significantly reduced from 13102 to 30 for dynamic features and 915 to 74 for static features by Boruta. BorutaShap obtains the best accuracy from Table 1 as a Feature Selector with a Random Forest Classifier. The BorutaShap has the advantage over other methods applied in this study as it projects a strong additive explanation of features, which is the strength of the SHAP method and at the same time, it has the robustness of the Boruta algorithm. This ensures that only significant variables remain on the set.
Figure 1. Proposed model for feature extraction and analysis
Figure 2. Boruta algorithm Steps
Comparison of Random Forest performance with different feature selection techniques

The three feature selection techniques Boruta, BorutaShap and Gini Importance, are evaluated by two popular classification algorithms Random Forest and AdaBoost. As observed from Table 2 Random Forest with BorutaShap feature selector outperforms all other combinations of feature selector and classification algorithm. Random Forest works on the concept of an ensemble of decision trees through bagging. This reduces the variance of the single decision tree and overcomes overfitting in the model. The highest accuracy as reported in Table 2, is 98.9% and Precision, Recall and F1-score of approx. 99%. On the other hand, AdaBoost algorithm with the Gini importance feature selector does not give good results. The lowest accuracy is 70%, reported by Adaboost+Gini Importance.

The Figure 3 displays the feature importance of all features through Boruta with Shap as feature selector. Figure 4 and Figure 5 project the boxplot of top 10 dynamic features and static features respectively. Higher value of z-score represents the significant importance of the feature.
Figure 3. Feature importance graph for all the features
Figure 4. Feature importance graph of selected dynamic features using BorutaShap
Conclusion

This work proposes Boruta and Boruta Shap feature selection algorithm for detecting malicious Android applications. It reduces the count of features and verifies the performance of Boruta and Boruta Shap algorithms in Android malware. The experimental results have been performed under different scenarios, static and dynamic features were evaluated individually as well as in an integrated manner. Results proved that the Boruta Shap algorithm is more effective than other state-of-art algorithms in decreasing feature dimension and increasing accuracy. The performance is determined through various metrics and classification algorithms: Random Forest and AdaBoost.

Further work will implement this framework on a benchmark dataset to compare results with state-of-art works in the literature. Further, in addition to binary classification, multi-class classification will be performed on malicious samples to map them into their respective families.

References


How to cite this Article:

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.