

# Android Malware Detection System Using Machine Learning and Random Forest Classification

Gaytri Ganesh Jadhav<sup>1</sup> and Ms. Asha Mane<sup>2</sup>

Student, Master of Computer Applications<sup>1</sup>

Assistant Professor, Master of Computer Applications<sup>2</sup>

Sadhu Vaswani Institute of Management Studies for Girls, Pune

Savitribai Phule Pune University, Pune, Maharashtra, India.

**Abstract:** *Malware on the Android platform has become a significant problem in terms of security because the malicious the apps can misuse the resources of the phone. In this study, we create a machine learning-based solution that distinguishes between benign and malware Android apps by analysing the DREBIN-215 dataset.*

**Keywords:** Android Malware, Malware Classification, Machine Learning, Random Forest Algorithm, DERBIN Dataset, Static Analysis, Feature Selection, Cyber security, APK Files

## I. INTRODUCTION

Android Applications Applications that are available on the android platform have become an essential element of our day-to-day activities. These include communication, financial transactions, learning, entertainment and information sharing among others. Due to the increase in the use of such applications, the issue of security becomes paramount due to potential misuse of sensitive information and other resources by malicious applications.

Signature based detection system is based on the use of previously known malicious software pattern. While this method is quite useful in the case of known malware, it will prove less effective when it comes to malware that has been altered or a new variant of malware emerges. Machine learning offers an alternative solution through the learning of patterns from labelled application data.

DREBIN technique showed how static attributes of the Android apps could be analysed. DREBIN data set has hence become an extensively utilized benchmark for the analysis of Android malware.

Subsequent studies have considered classifiers, ensembles, features section, and deep learning based techniques using Android malware data sets.

The feature obtained using DREBIN-215 are then applied in building a binary classifier of Android malware. Various machine learning algorithms were evaluated, but Random Forest was chosen as the final classifier through experiments. Besides the work not only emphasized the process of classifying malware but also the pre-processing, model evaluation, confusion matrix, and feature selection processes. This makes it an easy way to understand the topic.

## II. LITERATURE REVIEW

Malware Detection in Android has progressed from conventional signature-based systems to static, dynamic analysis, Machine Learning, and Deep Learning approaches. Research by Arp et al. (2014) titled DREBIN demonstrated that static features of an app could be used to identify malware. Reviews conducted later pointed towards the significance of feature extraction, pre-processing, feature selection, classifier selection, and evaluation in malware detection using machine learning in android (Liu et., 2020; Muzaffar et al., 2022).



Some of the researches have focused on the use of supervised machine learning techniques through static Android features. For example, Syrris and Geneiatakis (2021) compared various classifiers with static features obtained from DREBIN data set and demonstrated that effective classification is possible with an appropriate selection of features. In addition, the study by Al Sarah et al. (2021) also compared various ensemble techniques and obtained good results through the use of reduced feature sets such as Random Forest.

Feature selection and feature importance have been studied extensively due to existence of numerous non-equivalent features in the Android datasets. There have been studies conducted on information based feature selection, feature importance-based feature selection, and automated feature selection techniques. For instance, Wu et al.(2023) developed an automated feature selection technique known as DroidRL, whereas Pathak et al.(2025) considered feature-importance based feature selection technique for Android malware detection.

There have also been recent attempts at using deep learning and hybrid methods for Android malware detection. While these techniques can extract sophisticated patterns from large dimensional space, they may need higher computational requirements and may not always be popular by popularity of deep learning and machine learning techniques along with ongoing issues regarding datasets, feature extraction, and evaluation. All the same, the previous studies provide an excellent base for this study, and there is still room for experimentation with the DREBIN-215 feature set.

### **Research Gap**

The above literature review, clearly proves that machine learning techniques can be efficiently employed in detecting Android malware. Nevertheless, there are still some problems worth considering..

To begin with, many papers focus mainly on enhancing the classification score by means of feature selection, ensemble learning, or deep learning. This may lead to a situation where the full process needed to go from the Android app to the detection system is not clear.

In addition to this, varying pre-processing techniques, different sets of features, different training and testing approaches and various performance measurement techniques are used in various research works. Therefore, the performance values obtained by various research works cannot be considered equal.

Thirdly, there are different kinds of application static characteristics present in the DREBIN feature representation. Hence, a good APK detection method needs to guarantee that the features employed in training the model. An already trained model will not be able to detect any APK when the input feature representation is generated from other features.

Fourth, importance of features will be very helpful in understanding which feature affects a Random Forest model. Yet, most application-based prototypes concentrate primarily on the final output classification, not on the behaviour of the model based on its features.

With these considerations in mind, the objective of this research is to design a complete experimental pipeline of data pre-processing, Random Forest learning, performance evaluation, feature selection, and prototype-based APK analysis.

### **III. METHODOLOGY**

Android malware detection system is designed based on a machine learning technique. The approach includes many steps like collecting the data set, inspecting the data, cleaning the data, pre-processing, splitting the data set, developing the machine learning model, comparing the models, choosing the final model and performance evaluation. The entire process is performed using python in Google Colab environment.

#### **A. Dataset Collection**

The DREBIN-215 dataset was used for developing the proposed malware detection model. This dataset consisted of 15,036 samples of Android applications which had a total of 215 features where there were 5,560 samples of malware and 9,476 samples of benign applications. In the first step, the dataset was imported to the python environment and details about it were analysed to get an idea about the quality and structure of the dataset.



**Table 1. Dataset Features**

| Feature_Name                      | Feature_Category    | Feature_Name                                | Feature_Category    |
|-----------------------------------|---------------------|---------------------------------------------|---------------------|
| transact                          | API call signature  | android.telephony.SmsManager                | API call signature  |
| onServiceConnected                | API call signature  | READ_PHONE_STATE                            | Manifest Permission |
| bindService                       | API call signature  | getBinder                                   | API call signature  |
| attachInterface                   | API call signature  | ClassLoader                                 | API call signature  |
| ServiceConnection                 | API call signature  | Landroid.content.Context.registerReceiver   | API call signature  |
| android.os.Binder                 | API call signature  | Ljava.lang.Class.getField                   | API call signature  |
| SEND_SMS                          | Manifest Permission | Landroid.content.Context.unregisterReceiver | API call signature  |
| Ljava.lang.Class.getCanonicalName | API call signature  | GET_ACCOUNTS                                | Manifest Permission |
| Ljava.lang.Class.getMethods       | API call signature  | RECEIVE_SMS                                 | Manifest Permission |
| Ljava.lang.Class.cast             | API call signature  | Ljava.lang.Class.getDeclaredField           | API call signature  |
| Ljava.net.URLDecoder              | API call signature  | READ_SMS                                    | Manifest Permission |
| android.content.pm.Signature      | API call signature  | getCallingUid                               | API call signature  |

## B. Data Cleaning

Data cleaning was done before building the classifier model to avoid any possible data quality problems. Class column was used for the target variable and the rest for the feature variable. Labels of class were coded into numeric binary, where by B=0 and M=1.

The values of the features were transformed into numeric form in order to allow the use of machine learning models. During this transformation, the values which could not be transformed into numeric form were transformed to missing values. These missing values further imputed with the value of zero.

```
Dataset loaded successfully!
Dataset Shape: (15036, 216)

===== BEFORE CLEANING =====
Total Rows: 15036
Total Columns: 216

Class Distribution:
class
B      9476
S      5560
Name: count, dtype: int64

Missing Values: 0
Duplicate Rows: 7775
```

## C. Feature and Target Saperation

Following the process of cleaning, the dataset was split into two parts, nearly input feature matrix(x), and target variable(y). It was ensured that the class variable was not included in the input features, as it would helped in making sure that the feature matrix consisted of 215 DREBIN features.

```
===== FEATURE AND TARGET SEPARATION =====
Feature Matrix Shape : (15036, 215)
Target Variable Shape: (15036,)
```

## D. Training and Testing Data

The data after pre-processing was split in 80%20% for training and testing respectively. While the training data was used to build the ML models, the testing data served to evaluate the performance of the said ML models on unseen samples. For this purpose, the stratified sampling methods was employed to maintain the ratio of benign and malware samples in both sets the same.

training and testing dataset.



```

===== TRAIN TEST SPLIT =====
Total Records      : 15036
Training Records   : 12028
Testing Records    : 3008

Training Shape: (12028, 215)
Testing Shape  : (3008, 215)

Training Class Distribution:
Class (0 = Benign, 1 = Malware)
class
0      7580
1      4448
Name: count, dtype: int64

Testing Class Distribution:
Class (0 = Benign, 1 = Malware)
class
0      1896
1      1112
Name: count, dtype: int64

```

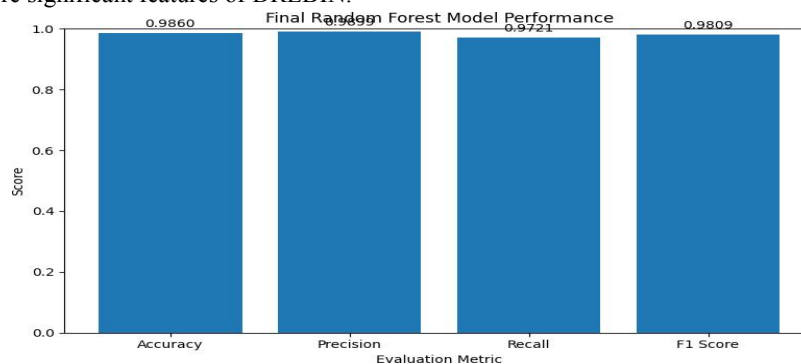
### E. Machine-Learning Model Development

Various supervised machine learning methods were developed and analysed after pre-processing to analyse their effectiveness in classifying Android malware. This gave a uniform ground for evaluating their classification abilities. The following measures were used to evaluate these models: Accuracy, Precision, Recall, F1-score, and ROC-AUC.

|   | Model               | Accuracy | Precision | Recall   | F1 Score |
|---|---------------------|----------|-----------|----------|----------|
| 0 | Logistic Regression | 0.978391 | 0.973756  | 0.967626 | 0.970681 |
| 1 | Decision Tree       | 0.977061 | 0.967713  | 0.970324 | 0.969017 |
| 2 | Random Forest       | 0.988032 | 0.990876  | 0.976619 | 0.983696 |
| 3 | SVM                 | 0.982048 | 0.980909  | 0.970324 | 0.975588 |
| 4 | KNN                 | 0.979056 | 0.973803  | 0.969424 | 0.971609 |

### F. Final Random Forest

Following the analysis of various machine learning algorithms, Random Forest was chosen for implementing malware detection system. Training of the algorithm was done with 300 decision trees under specific parameters. The algorithm learned from the input data and helped classify an applications as either benign or malware. It was also used for discovering the more significant features of DREBIN.



### G. Model Evaluation

The trained model was validated by using new unseen data. Accuracy, Precision, Recall, F1-score, and ROC-AUC were used for validating the model performance. Confusion matrix was used to analyse the positives and false



negatives. ROC curve was also generated to analyse the model's ability to differentiate between benign software and malware. Lastly feature importance was analysed to know about the features playing an important role in the decision-making process of the model.

#### **IV. FINDINGS AND RESULTS**

Based on the findings of the experiments, the application of the machine learning technique succeeded in differentiating the begin from the malware Android apps based on the DREBIN-215 features. In terms of choosing among the tested models, the Random Forest turned out to be the chosen final model due to the obtained evaluation results. Evaluation was performed using such metrics as Accuracy, Precision, Recall, F1-score, and ROC-AUC

##### **Limitations**

There exist certain limitations in this study. First of all, the model was designed based on the dataset called DREBIN-215, containing old examples of Android malware. Thus, the outcomes of the research might not be applicable to present-day Android malware. Besides, the model is based on the use of 215 predefined static features utilized during the training process. Furthermore, there exist the problem of the inability to classify the behaviour of applications is full measure.

#### **V. CONCLUSION**

This study developed an Android malware detection system using machine learning and the DREBIN-215 dataset. Different stages, including data pre-processing, model training, testing, and performance evaluation, were carried out. Random Forest was used as the final model and its performance was analysed using Accuracy, Precision, Recall, F1-Score, ROC-AUC and confusion-matrix. Feature-importance analysis was also used to understand the features influencing model. Overall, the study demonstrates the usefulness of machine learning for android malware classification, while future work can focus on newer datasets and more complete APK extraction.

#### **REFERENCES**

1. Kaggle, "DREBIN Android Malware Dataset," Kaggle Datasets, accessed 2026. Kaggle Datasets
2. Al Sarah, N., Rifat, F.Y., Hossain, M, S., and Narman, H. S.(2021). An efficient Android malware prediction using ensemble machine learning algorithms. *Procedia Computer Science*, 191, 184-191. <https://doi.org/10.1016/j.procs.2021.07.023>
3. Arp, D., M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck(2014), "DREBIN: Effective and explainable Detection of Android Malware in Your Pocket," *Network and Distributed System Security Symposium(NDSS)*. <https://doi.org/10.14722.ndss.2014.23247>
4. Alqahtani, E.J., Zagrouba, R., and Almuhaideb, A.(2019). A survey on Android malware detection techniques using machine learning algorithms. In 2019 sixth International conference on software defined Systems (SDS)(pp. 110-117).IEEE.
5. Muzaffar, A., Ragab Hassan, H., Lones, M.A., and Zantout, H.(2022). An in – depth review of machine learning based Android malware detection. *Computers and security*, 121,102833. <https://doi.org/10.1016/j.cose.2022.102833>
6. Pathak, A., Barman, U., and Kumar, T.S.(2025). Machine Learning approach to detect Android malware using feature-selection based on feature importance score. *Journal of Engineering Research*, 13(2), 712-720. <https://doi.org/10.1016/j.jer.2024.04.008>
7. Sharma, M., and Kaul, A.(2024). A review of detecting malware in Android devices based on machine learning techniques. *Expert Systems*, 41(4), e13482. <https://doi.org/10.1111/exsy.13482>
8. Syrris, V., and Geneiatakis, D.(2021). On machine learning effectiveness for malware detection in Android OS using analysis data.

