Permission-Based Malware Detection in Android Using Machine Learning

Nishant Rawat
Computer Science & Engineering, Sharda School of Engineering & Technology, Sharda University, Greater Noida, Uttar Pradesh, India,
nishantrawat575@gmail.com

Amrita
Computer Science & Engineering, Sharda School of Engineering & Technology, Sharda University, Greater Noida, Uttar Pradesh, India.
amrita@sharda.ac.in

Avjeet Singh
Computer Science & Engineering, Sharda School of Engineering & Technology, Sharda University, Greater Noida, Uttar Pradesh, India
avijeet.mnnit.cs@gmail.com

Abstract

Mobile devices are increasingly vulnerable to malicious programs or apps that threaten the privacy of users' data. Malicious apps are more intrusive than necessary, as they require fewer overall permissions to operate. The open-source nature of the Android platform, its acceptance of third-party app stores, and the range of app vetting make it more susceptible to attacks. To address this issue, a malware detection system has been developed that analyzes an app's permission requests and categorizes it as either benign or malware. The system uses a multi-level-based approach that involves collecting a dataset of 10,000 apps and identifying various aspects such as permission, small size, and permission rates. The apps are then classified as malware or benign using machine learning algorithms. The proposed technique achieved higher accuracy in detecting malware compared to existing methods, with accuracies of 91.54 % for Support Vector Machine, 92.04 % for Random Forest, and 91.11 % for Naive Bayes models. The proposed model showed a great balance between detecting malware and benign applications. The methodology also shows promise as a low-cost alternative to existing methods for detecting malware in Android apps, especially those that have been repackaged.

Keywords: Android, Malware Detection, Machine Learning

1. Introduction

Android has quickly become the dominant smartphone OS, with a current market share of around 70.97%. Based on these numbers, it's clear that Android is dominating the smartphone app industry. The availability of robust apps for a diverse user base is Android's strongest selling point. Regrettably, many cybercriminals throughout the world have taken notice due to the Apps' widespread use and accessibility. About 97% of mobile malware, according to reports, targets Android handsets. Around, 1.45 million new Android malware Apps were discovered in the second quarter of 2021, indicating that new malware is being developed every few seconds [1]. Several of these apps are launched with many versions with the express purpose of evading detection and reaching the largest possible audience. Academia and business have given serious thought to the problem of harmful apps on mobile devices. In response to these growing worries, researchers and

analysts from across the globe have devised and used a wide range of strategies for the creation of effective android malware detection technologies [2]. Both dynamic and static analysis is used to identify malicious APKs. The dynamic part of the analysis compares the behavior of the program while it runs against predetermined test scenarios. Static analysis, on the other hand, involves looking for security flaws in the meta. The dynamic technique is a reliable detection method; nevertheless, it is computationally expensive due to its emphasis on in-depth application analysis. In addition, unlike the static method, the analysis is done after the APKs have already been run [3][4]. This is why it is generally agreed that static analysis is superior for quickly gaining an overview of the APKs based on their predicted behavior. With an exhaustive range of approaches and methods, the static analysis attempts to identify the behavior of software in its runtime state before it is executed.

The goal in a secure environment is to identify and remove obviously harmful or repackaged applications before they can be installed and used. Using an authoritative estimate of the app's likely runtime behavior, static analysis may identify an app as malicious. When it comes to protecting user data or preventing apps from gaining access to private data, Android often uses the permission-based security approach [5]. To get unauthorized access to sensitive information, hackers often target Android apps by abusing the permissions they have been granted. As a result, without being given explicit permission, it is next to impossible to conduct a plan of action, making permission screening a crucial step for malware identification. In order to utilize an Android app, users must first agree to provide the app access to certain parts of the user’s device. Certain undesirable actions may be reflected by a combination of many permits [6]. When an app requests network authorization in addition to SMS access permission, for instance, it may collect the users' SMS information and then broadcast it over the Internet. This indicates that permissions are one of the most popular and useful permanent features in Android. Cloud computing has also been used in the recent evaluation of a detection method. SWORD [7] is a semantic analysis-based method for discovering evasion-aware malicious apps by using the Asymptotic Equipartition Property (AEP).

The primary focus of this research is on the permission-based detection of malware. The suggested method's goal is to enhance malware detection APKs' functionality while simultaneously decreasing the number of permissions used for categorization. Nevertheless, the top classifiers currently in use, such as Random Forest and Naive Bayes, were discovered and improved upon using the suggested Permission-based Malicious Applications detection technique. The proposed research aims to determine the smallest set of permissions necessary to accomplish the desired categorization accuracy, in comparison to current approaches. The proposed approach begins with decompiling Applications using AndroGuard to extract permissions. In addition, programs from VirusShare and the Official Play Store are used to compile a dataset from which the permissions may be retrieved. After the permissions are retrieved, they are coupled with other features, such as the permission rates and the App size metrics, before being filtered using the feature significance approach. When the authorization feature set has been generated, many machine learning models are used to identify dangerous apps. These are some of the ways in which the suggested strategy helps:

- Created a lightweight malware detection algorithm.
- Compiled a dataset of about 10,000 malware and benign APKs.
- We have optimized the permission feature set (77%) in comparison to current methods.
- Upped detection precision to 97% utilizing common ML methods, including support vector machine (SVM), random forest, and Naïve Bayes (NB)-based classifiers.

2. Literature Survey

The literature review section of our work examines various papers that have been published on the topic of malware detection using static analysis. One such paper by Suarez-Tangil et al. [9] proposes an Android malware detection method that utilizes explicit and implicit intentions. By using intentions alone for categorization, the system achieves an accuracy of 91%, while combining intent and permission of the program results in 95% accuracy. This study suggests that intent, in addition to permissions, can be used to identify malware.

Mary et al. [10] proposed a method for identifying repackaged Android apps by comparing the permission lists of the original and repackaged apps. The Random Forest classifier was found to have the highest accuracy at 88.53%. Hr Sandeep's [11] study utilized an Exploratory Data Analysis technique with deep learning methods to detect malicious apps based on their permissions. The proposed framework achieved a 94.6% success rate in identifying malware when using RF as a classifier. Srisa-An et al. [12] presented a permission-based approach that selected the most relevant permissions from the android manifest file to detect malware. Their multi-level data pruning method achieved a classification accuracy of over 90% using only 22 permissions. Fukuda et al. [13] proposed a method called Multilevel Permission Extraction (MPE), which first extracts and identifies the permissions of the android application and then uses that information to determine if the app is malicious. Their method achieved a 97.88% detection rate of malware.

Ibrahim et al. [14] proposed a system called APKauditor, it makes use of permissions to distinguish between safe and malicious apps. The system consists of a database, which stores the permissions or signature of the database, an android app, which is utilized by the end user, and a central server, which acts as a go-between between the android app and the database. The end-user program simply connects to the server where the analysis is stored and retrieves the results. They run 8762 programs through their system and catch malware 88% of the time.

According to the study by Cheng et al. [15], a framework was presented to calculate the application permission rate depending on the feature set by using system events, sensitive APIs, and authorization from the application. Cheng's methodology is efficient in terms of both time and money, correctly detecting malware 88.6 percent of the time. The classification model used to distinguish between safe and harmful files is constructed using the Rotation Forest method. They only utilize a small dataset, but by experimenting with various machine learning classifiers, they may increase their accuracy. Similarly, Lui et al. [16], proposed a system that decompiles android apps to get at their dex byte code, which is then transformed to java source code to get at their API calls, this approach involves many steps. Malware often asks for more permission than is really necessary, so they realize the manifest file of the Android program may appear like a rough approximation. When they combine Random Forest, SVM, and ANN, their detection accuracy rises to 94.98%.

In Mao et al. [17], singular value decomposition is used for clustering, the K-means algorithm is used to construct a classification model, and the KNN algorithm is used to categorize malware. The solution utilizes permissions in addition to intent and API requests to ensure a 97.87% success rate. Nevertheless, the approach is limited to a dataset of just 1738 android apps. Mansour et al. [18] presented a system known as DroidSeive. The system classifies malware based on a variety of characteristics, including API calls, code structure, permissions, etc. The framework is able to identify malicious software and then provide classification to the family to which that software belongs. The system

employs over a hundred thousand Android apps, with which it detects malware with 99.44% accuracy, classifies it with 99.265 accuracies, and experiences no false positives.

The PUMA (Permission Use to Detect Malware on Android) approach, suggested by Carlos et al. [19], makes use of permission in order to identify malicious apps. Using a classifier such as RandomForest, RandomTree, NaiveBayes, etc. on 4031 samples of android applications, they achieve a detection accuracy of 86.41%. For static analysis, Ming-Yang Su et al. [20] suggested a framework to extract native permissions and intent priority in addition to permission and sensitive functions. It launches the programme in a safe environment called a "sandpit," where its actions may be monitored and recorded. Naive Bayes, SVM, KNN, logistic regression, and other classifiers are used to correctly categories malware, allowing the system to identify malicious software with a 93% success rate. Weng et al. [21] suggested a tool they called Waffle Detector, which requires nothing in the way of human involvement. They've used permissions and API requests that need special handling to obtain an accuracy of 97.14%.

Daiki et al. [22] utilized the Androidmanifest.xml file for analysis and extracted six pieces of data, including authorization, intent filter (action), and process name. By evaluating the manifest file of the program cost-effectively, they achieved a 90% accuracy rate in identifying new viruses using a database of 365 Android applications. Many of the methods discussed above use permission-based feature sets to differentiate between benign and malicious APKs due to their high speed and nearly perfect detection accuracy. However, basic permissions are often not utilized to their full potential, and the number of inefficient authorization characteristics may increase computational complexity. To address these issues, a more efficient and faster detection method is needed. In the following section, we will explore the proposed solution to address the limitations of current approaches.

3. Related Background

3.1 Feature Selection Method

Eliminating unnecessary or irrelevant characteristics from a dataset is a frequent use case for feature selection, which is then used to determine the ideal feature subset to boost the system's performance. It also aids in resolving "the curse of dimensionality." For selecting features, we can use either a filter, a wrapper, or a hybrid. The performance of chosen features is evaluated by an external classifier in the filter technique. The wrapper method "wraps around" the existing classifier in order to evaluate an abridged collection of characteristics [23]. The computational overhead of this approach is higher than that of the filter method. In order to maximize efficiency with a given classifier, the hybrid strategy combines filtering and wrapping.

3.2 Classification Method

In this study, we use 3 different classifiers for the detection of malware. When these malwares were put through their paces, the primary objective was to evaluate how well they identify malware applications when combined with data preparation and feature selection strategies. These ML classifiers are as follows:

3.2.1 SVM

SVM is one of the most commonly and widely used classifiers of Machine Learning. This classifier's major goal is to build a judgement boundary or ideal line for classifying n-dimensional space so that subsequent data points may be rapidly allocated. Hyperplanes are optimum decision boundaries [24].

3.2.2 Random Forest

The random forest algorithm is an ensemble approach comprised of many decision trees and bagging processes. Each decision tree is trained on a subset of the whole dataset using bagging. Every tree is classified and the categorization is completed by majority voting on the finding of the decision trees. The most important parameters are max Depth, which says how deep the tree can grow and estimators, which define the number of trees in the forest [25] [26]. In our research, the number of trees in the forest was 100.

3.2.3 Naïve Bayes

It is a classifier based on the Bayes theorem. It's mainly used to solve classification problems. It is one of the probabilistic classifiers which uses probability to predict the class of the new sample [27].

4. Proposed Methodology

In this section of the paper, we discuss our proposed methodology. Our proposed framework outweighs all the above-mentioned paper in every aspect such as accuracy, runtime, etc. It enhances the performance of malware detection. Above mentioned paper's drawbacks were tackled by our proposed framework. In the proposed framework of malware detector, the system extracts only that permission set from the permission list of the android which is relevant and significant in the detection of the malware rather than dissecting the whole application. Our proposed system mainly focuses on those permissions combinations which were commonly found in malware rather than including all the permissions in the manifest file.

The proposed framework design was divided into steps: -

4.1 Creating Dataset

This is the first step in designing the framework. In this part of designing, we first collect the collection of malwares as well as benign applications from the internet. For the benign application, we could go to the official play store and collect the different categories of benign applications for our system database. For the malware application sample, we could use third-party websites like virus share, Virus Total, etc. These websites contain thousands of malwares that are already on the internet and get detected. After the collection of both types of samples, we create our dataset by considering a 50:50 ratio of both types of samples in the database. By doing this our dataset should be balanced and give less false positive rate. After successfully creating a balanced dataset our first step is completed.

Sample Size	Malware	Benign
No. of samples used	5000	5000

Table 1- No. of Data samples

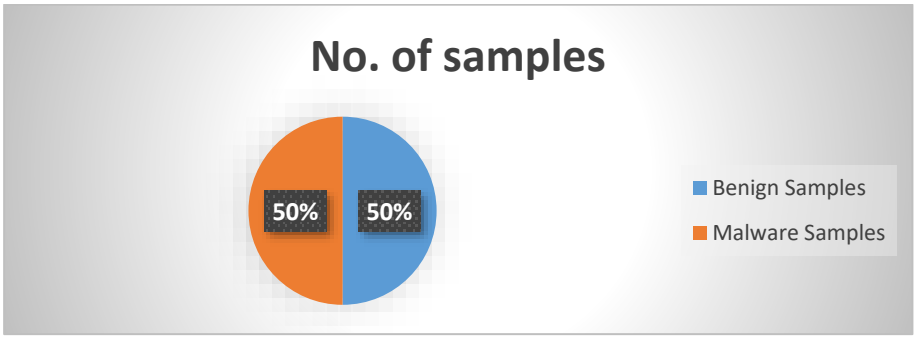


Figure 1 Ratio of Benign Samples to Malware Samples in the dataset.

4.2 Extracting Permissions

This is the next step in designing the framework which consists of extracting permissions from the collected applications dataset. All the permissions of an android APK are written in the android.manifest.xml file. These permission help us to do static analysis of the applications. In this file, all the permission requested by the application were written. By using software called AndroGuard [28] we can decompile the android application one by one and extract their permissions from their manifest.xml file. After extraction of permissions from the android.xml file of each APK in the dataset we create a feature for each APK in the dataset we create a feature set. This feature set contains the extracted permission of the applications.

4.3 Data Preprocessing on The Feature Dataset

In this step of the framework, we apply data preprocessing techniques on the feature dataset to reduce its size of the feature dataset. This step is the most important step among the other steps of designing the framework. In this step, we select only the most significant permission which helps the system in the classification of malware and benign application [10]. By applying techniques to fill the missing spaces, and removing duplicate permission in the feature dataset, we complete the feature dataset first. Then after we select the most significant permission out of others which helps the system in faster classification of malware and benign application. Selecting the most effective permissions out of the extracted permissions is given in the next part.

4.3.1 Finding the Effective Permissions for the Model

We exclude the less important permissions from consideration when choosing the minimal set of permissions to use for detection. We did this by using the feature significance attribute with several permutations of Google's risky permission list [10]. An indicator of feature relevance may be used to create less complex and quicker prediction models with fewer parameters. To get the necessary authorizations, we use a Random Forest-based feature significance approach (Table 2) [29][30]. We have chosen a cutoff value of 0.1 to filter out the least important permissions so that we may focus on the most consequential ones. Table 3 displays the results of our studies with several feature sets, which allow us to choose the permissions list that will have the most impact.

4.4 Applying Machine Learning Classifiers for classification

In this final step in designing the framework, we use a supervised ML algorithm for the classification of malware and benign application with a low false positive rate. For that first, we divide the sample set into 2 parts one is training data and the other one is test data. With the help of the training data we train our machine learning model and we use test data to check our model accuracy. The dataset used in the system to train and the test consists of 10,000 samples of applications both benign and malware both. In the traditional spilt model can overfit so to overcome that we use 10-fold cross-validation.

4. Experimental Result

The approach we have suggested distinguishes the malware application from the benign application based on their permissions. In order to demonstrate our work is successful we carried out many tests under the below-mentioned environment and the below-mentioned performance metrics we use to evaluate the performance of our system.

S. No.	Permissions	Feature Importance
1.	READ_PHONE_STATE	0.31764
2.	Smali Size	0.23495
3.	Permission Rate	0.21749
4.	WRITE_EXTERNAL_STORAGE	0.06948
5.	ACCESS_COARSE_LOCATION	0.05323
6.	READ_EXTERNAL_STORAGE	0.01588
7.	GET_ACCOUNTS	0.00953
8.	READ_CONTACTS	0.00650
9.	ACCESS_NOTIFICATION_POLICY	0.00337
10.	WRITE_CONTACTS	0.00265
11.	READ_CALL_LOG	0.00200
12.	INSTALL_PACKAGES	0.00151
13.	BODY_SENSORS	0.00080
14.	WRITE_CALL_LOG	0.00069
15.	READ_HISTORY_BOOKMARKS	0.00002

Table 2 List of different dangerous permissions with their importance

Type	S. No.	Permission
Permissions	1.	android.permission. ACCESS_COARSE_LOCATION
	2.	android.permission.READ_PHONE_STATE
	3.	android.permission.WRITE_EXTERNAL_STORAGE
Matrices	1.	Smali Size
	2.	Permission Rate

Table 3 Effective permissions for classification

4.1 Experiment Environment

While conducting the experiment, we make use of a computer that has a CPU of intel core i5 processor, 8 GB of RAM, and windows 11 as the primary operating system.

4.2 Evaluation Metrics

To measure how well our proposed system detects malware we evaluate it on some metrics which are as follows,

- Accuracy- To calculate the accuracy we use,

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \quad (1)$$

- Sensitivity- To calculate how well our system detects the positive(malware) instances.

$$Sensitivity = \frac{TP}{TP + FN} \quad (2)$$

- Precision- To calculate precision we use,

$$Precision = \frac{TP}{TP + FP} \quad (3)$$

- F-1 Score- For f-1 score we use,

$$F1 - Score = \frac{TP}{TP + \frac{1}{2}(FP + FN)} \quad (4)$$

4.3 Results and Discussions

In this section, we see the results of different classifiers on our model and based on that we evaluate the performance of the proposed system. However, many works have been done for malware detection. That's why in our work we did classification two times. For the first classification, we divide the dataset into two separate datasets. One is called the training set and another one is called the test dataset and the ratio of splitting the main dataset is 75:25. We compare the accuracy we got during the first classification with the already published systems.

Ref No.	Classifier	Features	Accuracy
[1]	Bayesian	Intent & Permission	95.5%
[2]	KNN, SVM, RF	Permission	92.94%
[3]	RF	Permissions	94.65%
[4]	SVM	Permissions	>90%
[5]	SVM, RF	Permission	97%
Our	SVM, RF, NB	Permission	97.3%

Table 4 Comparatively analysis of accuracy

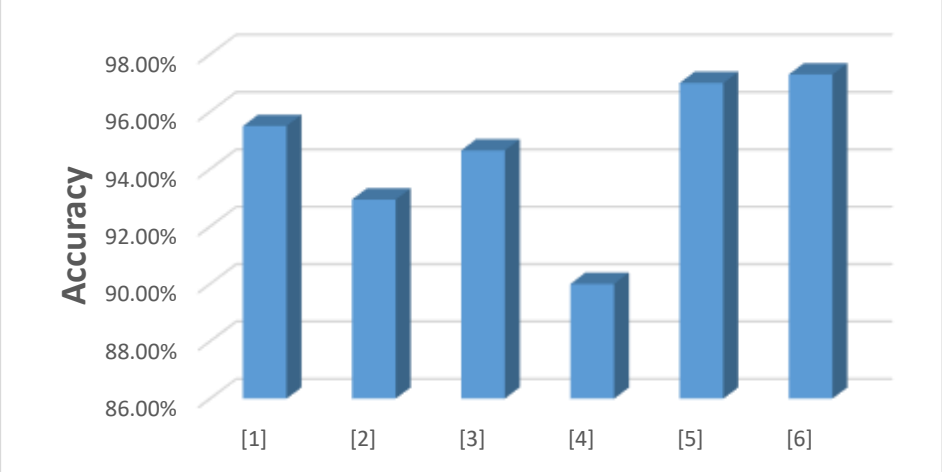


Figure 2 Comparing the accuracy of other work with our proposed model

Now, we perform the second classification based on the 10-fold cross Validation technique. The reason why we choose 10-fold cross-validation is that, in the conventional

dataset split sometime the model undergoes overfitting which affects the system accuracy. Table 5 and Figure 3 shows the result of the different classifier using 10-fold cross-validation.

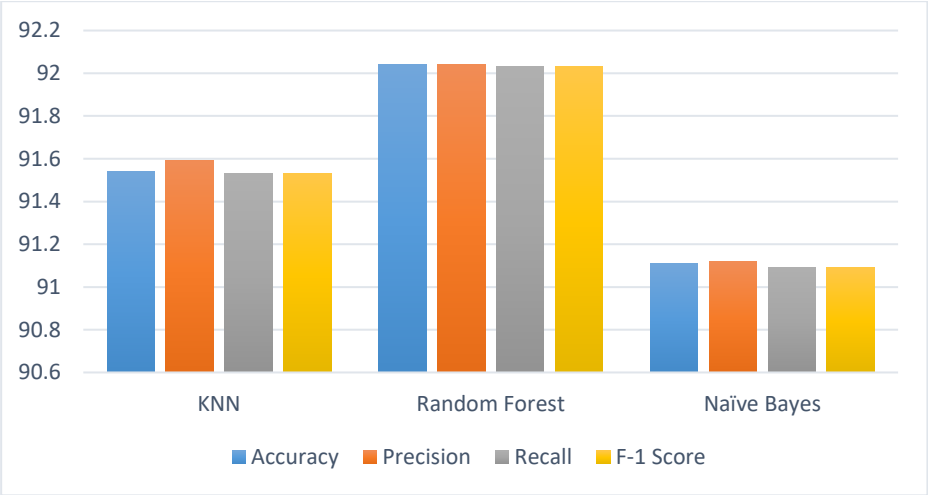
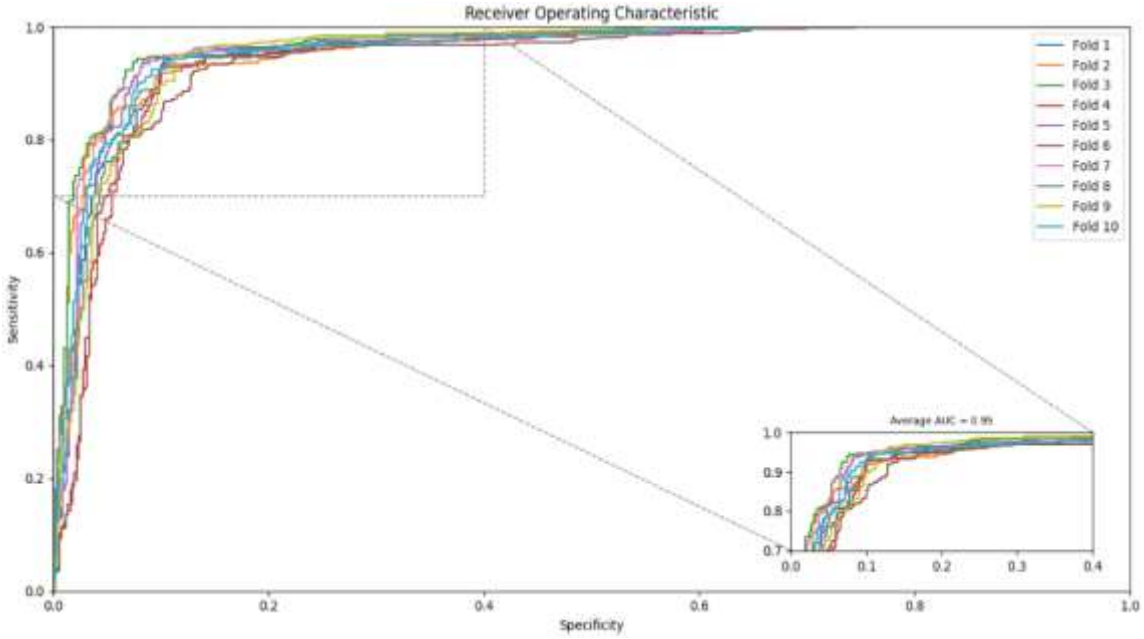


Figure 3 Comparison of different ML methods

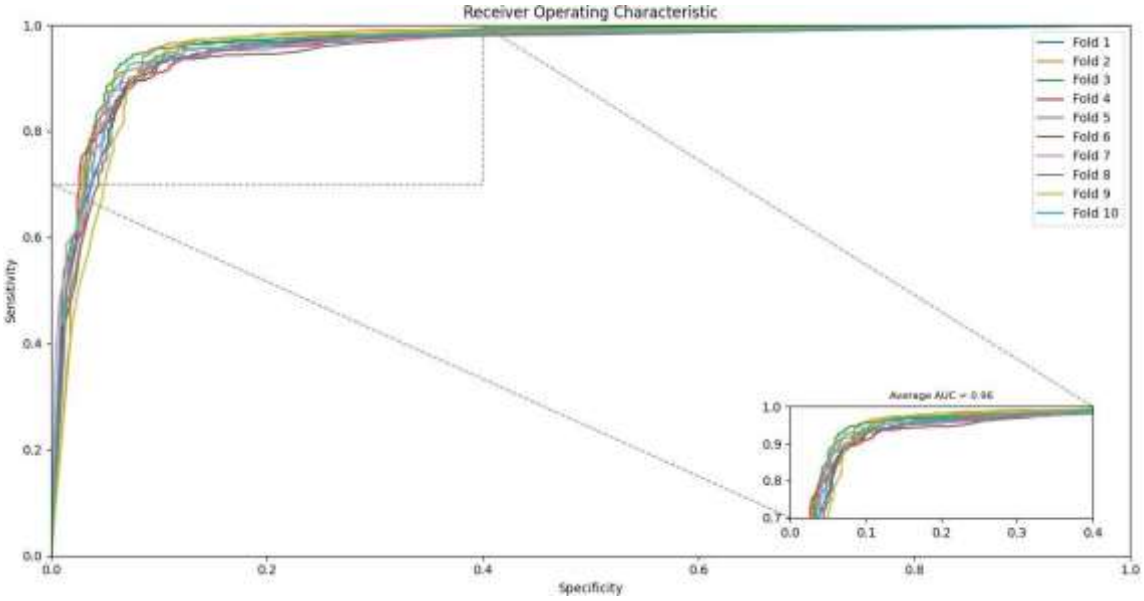
As we can see in the result the random forest algorithm gets the best accuracy which is 92.04%. The ROC curve is a likelihood plot that illustrates how well a classification model performs at various thresholds. Based on the result of 10-fold cross-validation, we draw a ROC curve for each fold. We show different ROC curves for different ML classifiers

Classifiers	Precision	Accuracy	F-1 Score	Recall	TPR	FPR
SVM	91.59	91.54	91.53	91.53	0.89	0.051
Random Forest	92.04	92.04	92.03	92.03	0.963	0.018
Naïve Bayes	91.12	91.11	91.09	91.09	0.846	0.075

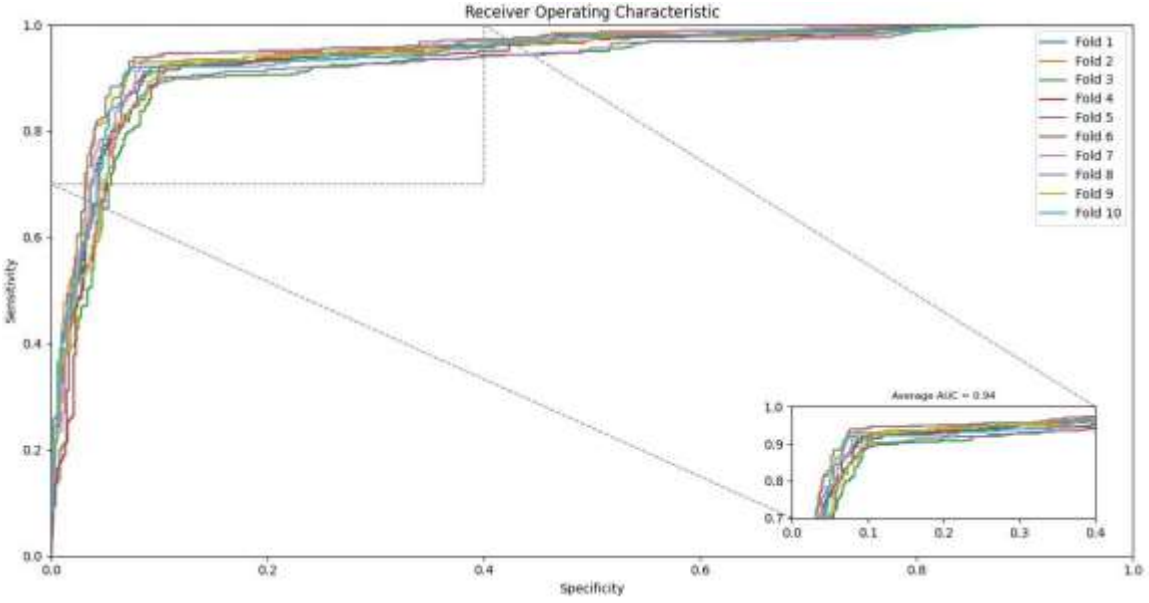
Table 5 ML classifier evaluation using 10-fold Cross-validation technique


Receiver Operating Characteristic
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5
Fold 6
Fold 7
Fold 8
Fold 9
Fold 10
Sensitivity
Specificity
Average AUC = 0.95

(a)


Receiver Operating Characteristic
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5
Fold 6
Fold 7
Fold 8
Fold 9
Fold 10
Sensitivity
Specificity
Average AUC = 0.96

(b)



(c)

Figure 4 (a), (b), (c) comparison of ROC curves of SVM, Random Forest and Naïve Bayes.

The AUC is a statistic that is used to summarize the ROC curve. This measure is used to determine how well a binary classifier can differentiate between different classes. When AUC is equal to one, the classifier is able to make an accurate distinction between all of the positive and negative class values. If the AUC has been 0, then all negatives were positives and all positives were negatives when making its predictions. When 0.5<AUC<1, there is a strong likelihood that the classifier can distinguish positive class values from negative class values. Whenever the AUC is higher, it shows that the model is more capable of distinguishing between the positive and negative classes.

- **Discussion**

Improving the dataset-gathering method or the chosen characteristics may help malware detection systems meet the need for a higher performance measure. The suggested strategy relies heavily on the careful selection of crucial permissions, which the research has shown to be a critical factor in ensuring excellent performance. Making permissions is the primary topic of our research since an App has to get user approval to do the essential operations. The relevance of the suggested technique lies in the meticulous selection of the dataset samples, which is the other part of assessment performance that it focuses on. Moreover, 10,000 APKs are used to train and verify the proposed technique. These APKs were obtained from legitimate Google Play Store and VirusShare sources. In order to develop a low-cost malware detection solution, it is important to determine the minimum amount of feature sets that will allow the classifier to achieve high detection accuracies.

5. Conclusion

The prevalence of mobile malware has become a major security concern in the mobile ecosystem, and using machine learning algorithms to address these concerns requires a more precise selection of features. In this study, we conducted a statistical analysis of the Android ecosystem and found that a higher degree of accuracy can be achieved by reducing the number of permissions while maintaining high efficiency and effectiveness. We tested Random Forest, Support Vector Machine, and Naive Bayes classifiers on the same dataset, and our experimental results showed that all classifiers achieved an accuracy of 91% or higher when using the proposed permission dataset. We also found that a smaller

set of major permissions was sufficient for achieving an adequate level of detection accuracy. In the future, expanding the dataset and testing the proposed method with more supervised and unsupervised machine learning classifiers may lead to improved detection accuracies.

References

[1] *Chebyshev,.Victor. "It threat evolution in Q2 2021. mobile statistics, Securelist English Global securelistcom". Available at: https://securelist.com/it-threat-evolution-q2-2021-mobile-statistics/103636/.*

[2] *Fidelis Threat Intelligence Report - february/march 2021 Fidelis Cybersecurity. Available at: https://fidelissecurity.com/resource/report/fidelis-threat-intelligence-report-february-march-2021.*

[3] *Sihag, V.; Vardhan, M.; Singh, P. "A survey of android application and malware hardening". Comput. Sci. Rev. 2021, 39, 100365.*

[4] *Zhang, W.; Luktarhan, N.; Ding, C.; Lu, B., "Android Malware Detection Using TCN with Bytecode Image. Symmetry", 13, 1107, 2021.*

[5] *Wang, W.; Zhao, M.; Gao, Z.; Xu, G.; Xian, H.; Li, Y.; Zhang, X., "Constructing Features for Detecting Android Malicious Applications: Issues", Taxonomy and Directions. IEEE Access, 7, 67602–67631, 2019.*

[6] *Jannath Nisha, O.S.; Mary Saira Bhanu, S., "Detection of malicious Android applications using Ontology-based intelligent model in mobile cloud environment". J. Inf. Secur. Appl., 58, 102751, 2021.*

[7] *Bhandari, S.; Panihar, R.; Naval, S.; Laxmi, V.; Zemmari, A.; Gaur, M.S. Sword: Semantic aware android malware detector. J. Inf. Secur. Appl., 42, 46–56, 2018.*

[8] *VirusShare.com. Available at: http://virusshare.com/ (Accessed: September 24, 2022).*

[9] *Feizollah, Ali, Nor Badrul Anuar, Rosli Salleh, Guillermo Suarez-Tangil, and Steven Furnell. "Androdialysis: Analysis of android intent effectiveness in malware detection." computers & security 65 (2017): 121-134.*

[10] *Nisha, OS Jannath, and S. Mary Saira Bhanu. "Detection of repackaged Android applications based on Apps Permissions." In 2018 4th International Conference on Recent Advances in Information Technology (RAIT), pp. 1-8. IEEE, 2018.*

[11] *Sandeep, H.R. Static analysis of android malware detection using deep learning. In Proceedings of the 2019 International Conference on Intelligent Computing and Control Systems (ICCS), Secunderabad, India, 15–17 May 2019; pp. 841–845.*

[12] *Li, Jin, Lichao Sun, Qiben Yan, Zhiqiang Li, Witawas Srisa-An, and Heng Ye. "Significant permission identification for machine-learning-based android malware detection." IEEE Transactions on Industrial Informatics 14, no. 7 (2018): 3216-3225.*

[13] *Wang, Zhen, Kai Li, Yan Hu, Akira Fukuda, and Weiqiang Kong. "Multilevel permission extraction in android applications for malware detection." In 2019 International Conference on Computer, Information and Telecommunication Systems (CITS), pp. 1-5. IEEE, 2019.*

[14] *Talha, Kabakus Abdullah, Dogru Ibrahim Alper, and Cetin Aydin. "APK Auditor: Permission-based Android malware detection system." Digital Investigation 13 (2015): 1-14.*

[15] *Zhu, H.-J.; You, Z.-H.; Zhu, Z.-X.; Shi, W.-L.; Chen, X.; Cheng, L. DroidDet: Effective and robust detection of android malware using static analysis along with rotation forest model. Neurocomputing 2018, 272, 638–646.*

[16] *Qiao, Mengyu, Andrew H. Sung, and Qingzhong Liu. "Merging permission and api features for android malware detection." In 2016 5th IIAI international congress on advanced applied informatics (IIAI-AAI), pp. 566-571. IEEE, 2016.*

[17] *Wu, Dong-Jie, Ching-Hao Mao, Te-En Wei, Hahn-Ming Lee, and Kuo-Ping Wu. "Droidmat: Android malware detection through manifest and api calls tracing." In 2012 Seventh Asia joint conference on information security, pp. 62-69. IEEE, 2012.*

[18] *Suarez-Tangil, Guillermo, Santanu Kumar Dash, Mansour Ahmadi, Johannes Kinder, Giorgio Giacinto, and Lorenzo Cavallaro. "Droidsieve: Fast and accurate classification of obfuscated android malware." In Proceedings of the seventh ACM on conference on data and application security and privacy, pp. 309-320. 2017.*

[19] *Borja Sanz et al.,2019, PUMA: Permission Usage to Detect Malware in Android. Springer Berlin.*

[20] *Su, Ming-Yang, and Kek-Tung Fung. "Detection of android malware by static analysis on permissions and sensitive functions." In 2016 Eighth International Conference on Ubiquitous and Future Networks (ICUFN), pp. 873-875. IEEE, 2016.*

[21] *Sun, Yuxia, Yunlong Xie, Zhi Qiu, Yuchang Pan, Jian Weng, and Song Guo. "Detecting android malware based on extreme learning machine." In 2017 IEEE 15th Intl Conf on Dependable, Autonomic and Secure Computing, 15th Intl Conf on Pervasive Intelligence and Computing, 3rd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech), pp. 47-53. IEEE, 2017.*

[22] *Sato, Ryo, Daiki Chiba, and Shigeki Goto. "Detecting android malware by analyzing manifest files." Proceedings of the Asia-Pacific Advanced Network 36, no. 23-31 (2013): 17.*

[23] *Kohavi, R., and G. H. John. "Wrappers for feature subset selection, Artificial Intelligence, vol. 97, no. 1-2." (1997): 273324.*

[24] *Sckit learn. Support Vector Machines, by sci-kit learning. Available: https://sckit-learn.org/stable/modules/svm.html.*

[25] *J. Saxe and H. Sanders, Malware data science: attack detection and attribution.*

[26] *F. A Narudin, A.Feizollah, N. B. Anuar, and A. Gani, 2016, "Evaluation of machine learning classifiers for mobile malware detection", Soft comput, vol.20, no.1, doi: 10.1007/s00500-014-1511-6.*

[27] *H. Zhang, 2004, "The optimality of Naïve Bayes," in Proceedings of The Seventeenth International Florida Artificial Intelligence Research Society Conference, pp. 17{19, Miami Beach}.*

[28] *Androguard: Reverse Engineering, Malware Analysis of Android Applications. Available online: https://github.com/androguard.*

[29] *Sotiroudis, S.P.; Goudos, S.K.; Siakavara, K. 2020, Feature Importances: A Tool to Explain Radio Propagation and Reduce Model Complexity. Telecom, 1, 114–125.*

[30] *Nasir, M.; Javed, A.R.; Tariq, M.A.; Asim, M.; Baker, T., 2022, Feature engineering and deep learning-based intrusion detection framework for securing edge IoT. J. Supercomput., 78, 1–15.*