HNSJ Humanitarian & Natural Sciences Journal
ISSN: (e) 2709-0833
www.hnjournal.net
Peer-Reviewed Journal
Indexed at: NSP - SJIF
Arabic Impact Factor 2020 = 0.44

RESEARCH ARTICLE

GRAPH APPROACH FOR ANDROID MALWARE DETECTION USING MACHINE LEARNING TECHNIQUES

Maha Adam Gumaa[1]

[1] Computer Sciences, ALNeelain University

Email: angleeee@live.com

HNSJ, 2021, 2(11); https://doi.org/10.53796/hnsj21115

Published at 01/11/2021 **Accepted at 20/10/2021**

Abstract

Day by day, the number of users of mobile devices such as smartphones and tablets is increasing. the Android operating system is considered one of the most widely used and widespread operating systems. However, Android based apps can be highly vulnerable to various types of malware attacks, due to its popularity in the mobile market and its open nature. These applications need to use a number of sensitive permission files during installation and runtime, which malware developers exploit to launch attacks on users. In this paper, an approach was proposed and made based on the most important permissions and API calls. This is done by selecting and generating features based on the graph and then using machine learning techniques to train and classify the malware detection tool. The results of the experiment show that this approach achieves an accuracy rate of up to 97% when using the algorithm random forest and a recall rate of 96 % which prove the effectiveness and advantages of approach.

Key Words: Android; Malware Detection; Machine Learning; Permission; API calls; Static Analysis

عنوان البحث

نهج الرسم البياني لاكتشاف البرمجيات الضارة لنظام الأندرويد باستخدام تقنيات التعليم الآلي

مها آدم جمعة

[1] علوم الحاسوب، جامعة النيلين، السودان

بريد الكتروني: angleeee@live.com

HNSJ, 2021, 2(11); https://doi.org/10.53796/hnsj21115

تاريخ النشر: 2021/11/01م تاريخ القبول: 2021/10/20م

المستخلص

يومًا بعد يوم ، يتزايد عدد مستخدمي الأجهزة المحمولة مثل الهواتف الذكية والأجهزة اللوحية. يعتبر نظام الاندرويد أحد أكثر أنظمة التشغيل استخداما وانتشارًا. ومع ذلك ،يمكن ان تكون التطبيقات القائمة والمستندة علي الاندرويد معرضة بشدة لأنواع مختلفة من هجمات البرامج الضارة ، نظرًا لشعبيتها في سوق الأجهزة المحمولة وطبيعتها المفتوحة. تحتاج هذه التطبيقات إلى استخدام عدد من ملفات الأذونات الحساسة أثناء التثبيت ووقت التشغيل ، والتي يستغلها مطورو البرامج الضارة لشن هجمات على المستخدمين. في هذه الورقة ، تم اقتراح نهج ووضعه على أساس أهم الأذونات واستدعاءات واجهة برمجة التطبيقات. يتم ذلك بواسطة تحديد وإنشاء الميزات بناءً على الرسم البياني ثم استخدام تقنيات التعلم الآلي لتدريب أداة اكتشاف البرامج الضارة وتصنيفها. تظهر نتائج التجربة أن هذا النهج يحقق معدل دقة يصل إلي 97٪ عند استخدام خوارزمية الغابة العشوائية ومعدل استدعاء 96٪ مما يثبت فعالية ومزايا النهج.

1 Introduction

Smartphones and tablets have grown in popularity as their costs have decreased and their features, capabilities, and service availability have increased. In a variety of fields, Android devices and tablets have become an indispensable part of our daily and practical lives. This increased the level of protection. Operating systems have also played a significant role in the adoption and spread of mobile devices and applications, allowing malicious software to emerge (malware). This is also true of the Android operating system. The Android operating system has long been a major player in the mobile operating system market. As of June 2018, the Google Play store had over 3.3 million apps [1], and Android had 54.2 percent of the global market share as of December 2018. According to the global study, the first quarter of this year saw the sale of approximately 329 million smartphones based on the "Android" operating system, accounting for 86 percent of total sales. Because of its openness and free availability, has not only become a big player in the mobile device market, but also a desirable aim for cybercriminals [2]. Malware or malicious payload exploits Android security problems to launch a variety of attacks, including those for privileges, remote control, unauthorized financial charges, and personal information theft, resulting in non-private financial losses. As a result, there is an urgent need to detect and analyze the malicious payload when installing and using an Android app. Machine learning techniques have been widely used in malware detection. This method approaches malware detection as if it were a binary classification problem (for example, classifying an application as malicious or benign). It can deal with classic pattern recognition and machine learning techniques. This method outperforms traditional dynamic and static methods in terms of scalability and time consumption because it does not thoroughly investigate the substance and details of software. The two most important factors influencing the performance of machine learning-based approaches [2] are feature selection and learning rate [3,4]. APIs and permissions are frequently identified as attributes because they contain detailed security information about critical resources that processes can access. However, in most current works, these features are extracted with the level of accuracy of the entire application, and the contexts associated with them are ignored, resulting in a high rate of false positives in discovery. To address this issue, we propose a static Android malware detection method that improves on existing business by taking feature contexts into account. As primary features, this paper identifies two types of program properties. Permissions that are sensitive to security and the number of API calls, each of which is associated with a different type of context These fundamental features are combined with their contexts to produce new features, which are then used to train and test a classifier model by embedding them in a feature vector space. The main challenge of our approach is feature recovery in order to achieve better performance, lower consumption, and feature generation. We define the program architecture as a collection of graphs and provide a graph reduction transformation to simplify its structures and improve feature generation performance. Based on the reduced graphs, we can create an efficient feature extraction graphing algorithm. In summary, the following are the paper's main contributions:

- propose a graph-based feature selection technique for combining the two kinds of raw features to serve as newly created features. These features outperform traditional machine learning-based approaches because they include rich semantic knowledge about program behaviors.
- propose a graph-based feature generation approach that can safely remove irrelevant graph nodes and edges and minimize the complex structure of a graph to a simpler version because it is only concerned with the APIs and permissions to be called. As a result, an efficient algorithm for feature generation is produced.

3 Background and Related Work

Each and every Android application AndroidManifest.xml is included. File with the necessary permissions and API calls and other parameters application. This list of permissions is given to the user additional resources during installation. The application is downloaded after the user grants all of the permissions. The application's Java code may contain the malicious element of the malware samples. If the manifest file has the necessary permissions, the API calls in the code are invoked. This is why permissions are the most commonly used static feature in Android malware detection.

In recent years, Android malware detection has received a lot of attention, and many techniques and tools have been proposed to detect the rapidly growing Android malware. Android malware detection is classified into three major categories: static detection, dynamic detection, and hybrid detection. In the subsections that follow, this section reviews of related works all of the detection types that have been published in the journal articles.

1-Static Detection: Static methods seek to examine the manifest file elements, Java code, or the sequence of API calls within the code. In [5] Through the supervised learning process, two machine learning (SVM) and (KNN) were applied and evaluated to perform classification of the feature set into either benign or malicious applications (apps). This work entails static analysis of apps, manifest files, and system call logs generated by said app, which can improve results when compared to individual feature analysis. and results was achieved for a dataset of real malware and benign apps indicate the average accuracy rate of 79.08% and 80.50% with average true positive rate of over 67.00% and 80.00% using SVM and KNN, respectively.in [6] Their ML algorithms first individually analyze the intent filters and permissions requested in an app's manifest.xml file. resulted in a significantly large difference in detection accuracy as a result of due to the relatively low relevance of intent filters alone However, combining both static features yielded detection accuracy results of 91.7% percent (SVM) and 91.4% percent (KNN), which is an improvement over the classification results of both individual feature sets. In [7] the work using permissions and APIs as program features are extracted from the manifest.xml and Similar files, respectively. employed and their effectiveness was measured using the Linear-Support Vector Machine (L-SVM) classifier. It was observed that this classifier achieved Android malware detection accuracy of 99.6% for the combined features. In [8] The effectiveness of four different machine learning algorithms in conjunction with features selected from Android manifest file permissions to classify applications as malicious or benign is investigated in this study. And the results, on a test set consisting of 5,243 samples, produce accuracy, recall, and precision rates above 10 80%. Of the considered algorithms (Random Forest, Support Vector Machine, Gaussian Naïve Bayes, and K-Means), Random Forest performed the best with 82.5% precision and 81.5% accuracy. And in [9] The study's goal is to create an efficient system for detecting malware in Application Programmable Interfaces (APIs) and classifying it as worms, viruses, Trojans, or normal. To classify malware that occurred in API call sequences, the Multi-Dimensional Nave Bayes Classification (MDNBS) is used. The performance of existing and proposed techniques is evaluated and compared using True Positive Rate (TPR), False Positive Rate (FPR), precision, recall, f-measure, and accuracy measures. In [10] proposed a framework based on static analysis that extracts a set of features from an app's AndroidManifest.xml and disassembled code to actually create a features vector SVM was used on the dataset to learn a distinction between the two types of apps (benign and malicious).

2-Dynamic Detection: Dynamic analysis methods monitor and inspect the implemented code's interaction with the system. The primary benefit of this technique is that it detects dynamic code loading and records application behavior during running time. They take time, but they are successful against malware obfuscation. In [11] proposes a dynamic analysis approach that mimics human interactions with Monkey Runner and extracts system API calls at runtime. It also suggests SAIL, a new feature selection approach for discovering prominent system calls from application fields, and then employs machine learning techniques to detect potential malicious files. In [13] To detect malicious apps, authors examined dynamic API calls and system calls. In this paper focus on static permissions and api based detection.

Table 1: Brief description of some earlier derived techniques with their accuracy

REF	GOL	METHODOLOGY	DESCRIPTION	ACCURCY
[5]	Detection	Static	manifest file and system call logs	79%
[6]	Detection	Static	intent and permissions	91%
[7]	Detection	Static	permissions and APIs	99%
[8]	Detection	Static	permissions and APIs	80%
[9]	Detection	Static	API call sequences	__
[10]	Detection	Static	AndroidManifest.xml	94%
[11]	Detection	Dynamic	API calls	95%
[13]	Detection	Dynamic	API calls and system calls	96%

4 Implementation and Experiment

This section describes the methodology of the experiment which were performed to the effectiveness of our model for detecting malware and extracting the features by using graph as well as analyzing the performance of the machine learning algorithms. The experimental methodology presented in this paper it is illustrated in the following figure (1). In our detection model, we employ a three-pronged research methodology: The first step is to create the CFG of the Android malware dataset, the second step is to extract features from the CFG and create a training dataset, and the third step is to generate classifiers based on specific machine learning algorithms and then detect malware using these classifiers.

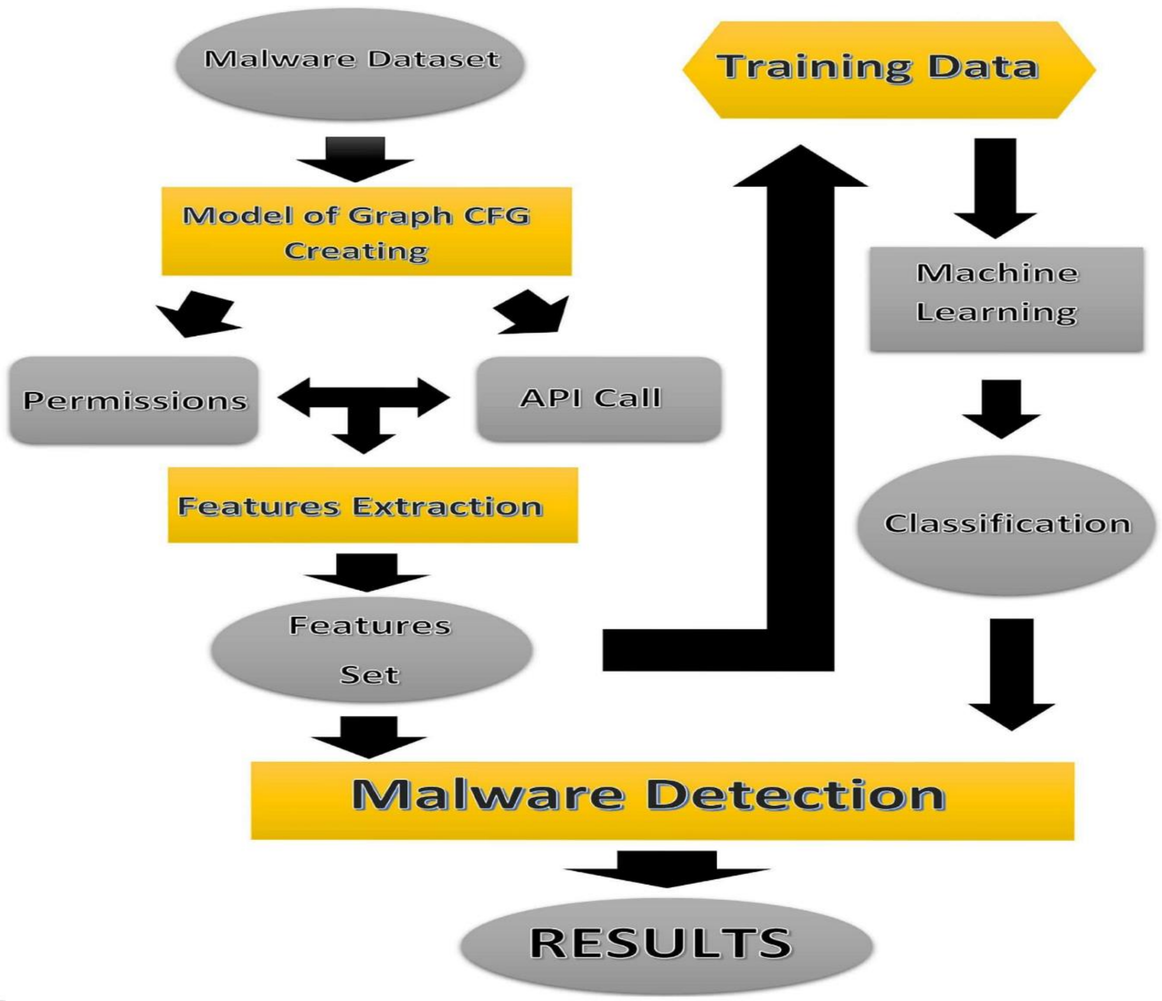


Figure (1) Describe architecture of the module detection.

4.1 Features extraction

The dataset has been processed throw many steps that include data cleaning, and data transformation. Then, the processed dataset is used for extra representative features. extract four types of features by using graph approach which are manifest permissions, API calls signature, intent, and other features commands sign Figure 6 show that. The permissions and API calls are strong features for detecting malware in Android systems and they achieve good performance when they applied with machine learning techniques. The feature extraction stage produces many features These features may include irrelevant elements that increase the risk of overfitting and lengthen model training time. As a result, the dataset should be transformed from a high dimension dataset to a low dimension dataset with no loss of total information. To achieve that, then applied the feature selection stage by selecting the most frequent permissions and API calls in both benign and malware applications.

Algorithm 1 Graph Construction Algorithm

INPUT:

- dataset **(D),** columns represent the features set **(F),** rows represent the Android applications values.
- categories dataset **(D2)** the rows describes each unique feature **(Fi)** category **(Ci).**

Output:

frequency graph **G (V, E)**

```
1 - extract the columns (F1, F2…  Fn) of D in F.
2 - Let C = (C1, C2… Cn) set of categories in (D2).
3 - For each category Ci ∈ D2, add node Vi to the set of nodes V if Vi ∈/ V.
4 - For each feature Fi ∈ D, add edge Ei to set of edges E if Ei ∈/ E.
5 - Return Graph G (V, E).
```

Then, used ANOVA filter (Analysis of Variance) SelectKBest method to select the best n-features according to the k highest scores. The results show that the feature vector with 182 features achieves the best performance. As shown in the following algorithms. Table 7 show the features selection number for each category permissions and API calls.

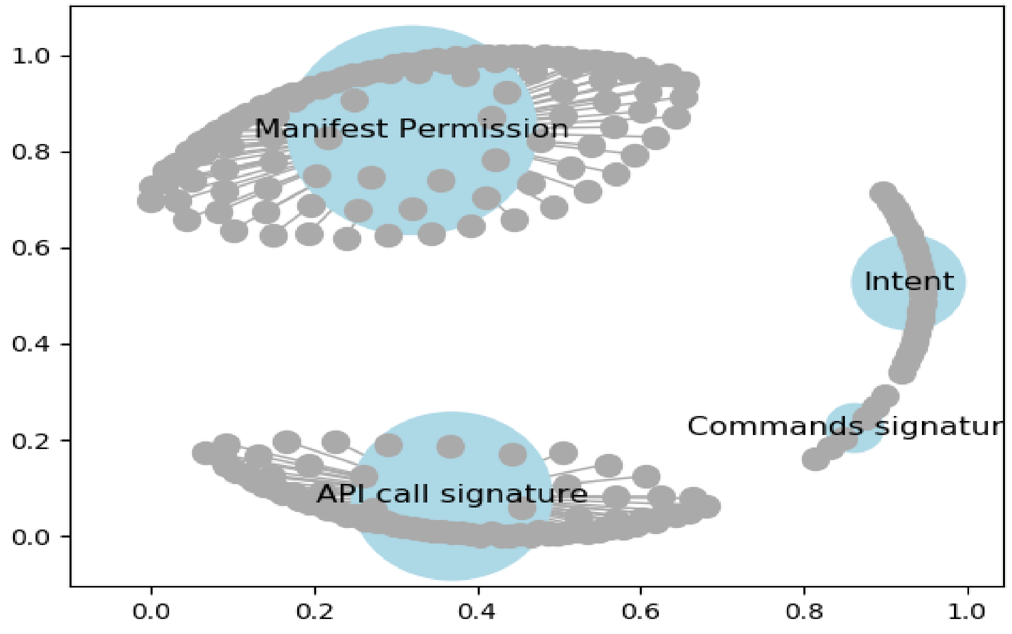


Figure (6) Explain Dataset category

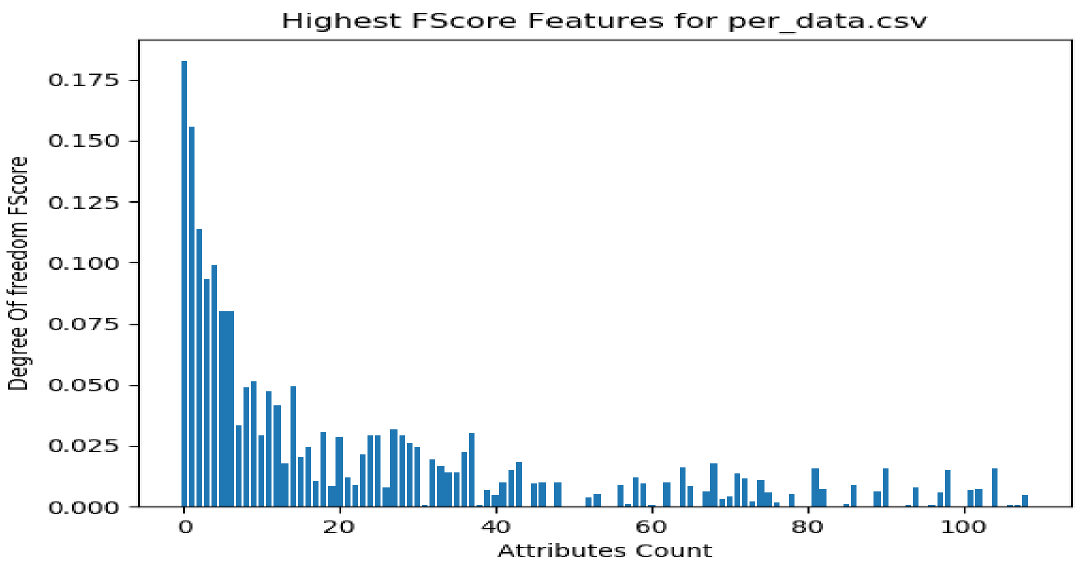


Figure (2) Explain highest F-Score Features for API calls

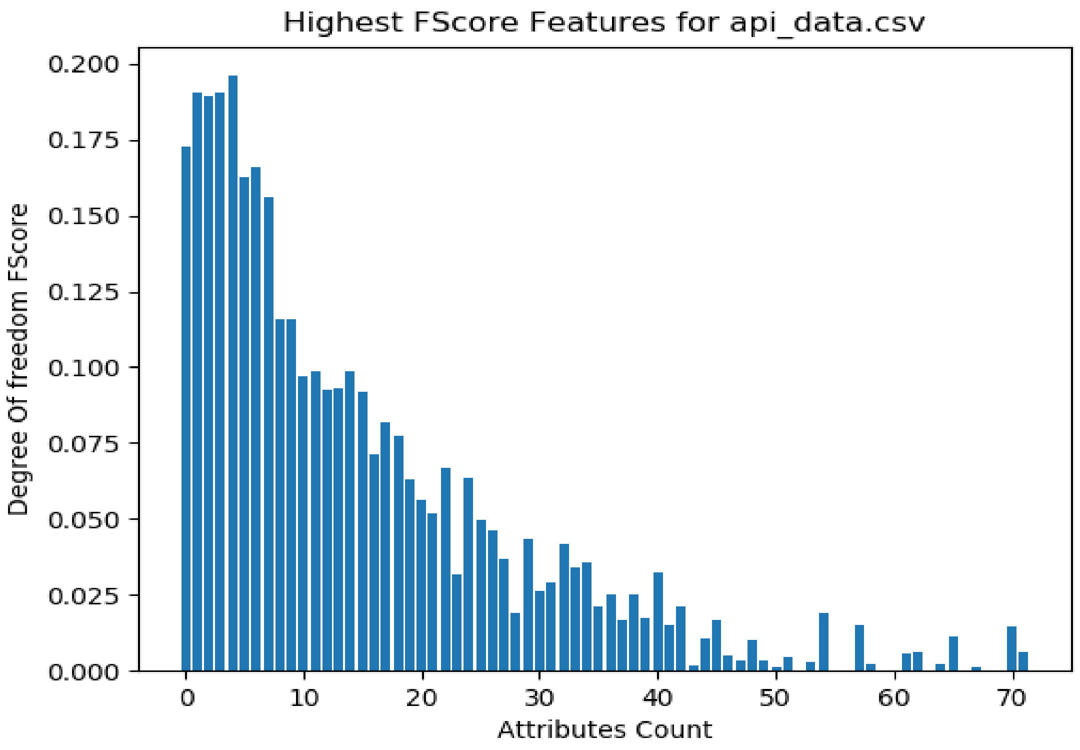


Figure (3) Explain highest F-Score Features for Permission

The suggested framework represents the relationship between various permissions and API calls in each application using graphs. The program utilizes Algorithm 1 to build a graph G (v, e) by representing a single permission or single api calls at vertex V and a couple of permissions and API calls connected by an adjusted edge e. Each edge weight increases when the same permission and API couple appears in various applications. An application with an only one or no permission or api calls actually adds nothing to analysis or detection. And According to the SelectKBest method to select the best features from the permission graph and the API graph for each application, the best features are selected in the following table (2and 3).

Table (2): Show Feature Name for API Calls and Frequency

Feature Name for API Calls	Frequency
['transact']	0.195746
['onServiceConnected']	0.193616
['bindService']	0.193237
['attachInterface']	0.19082
['ServiceConnection']	0.175773
['android.os.Binder']	0.172378
['Ljava.lang.Class.getCanonicalName']	0.167181
['Ljava.lang.Class.getMethods']	0.160376
['Ljava.lang.Class.cast']	0.124507

Table (3): Show Feature Name for Permission and Frequency

Feature Name for Permission	Frequency
['SEND_SMS']	0.183952843
['READ_PHONE_STATE']	0.147901394
['GET_ACCOUNTS']	0.107922138
['RECEIVE_SMS']	0.099231633
['READ_SMS']	0.088655911
['USE_CREDENTIALS']	0.083306345
['MANAGE_ACCOUNTS']	0.079848816
['WRITE_SMS']	0.056954176

4.2 Machine learning classifiers

The algorithms used in our module during the experiment and after the extraction and selection of the best features are Random Forest, K-Nearest Neighbor, Decision Tree, and Logistic Regression. Finally, the all classifier is used to determine whether a file is malware or benign, using two distinct features: API calls and Permission.

4.3 Metrics

For the purpose of evaluating the results, we use confusion matrices that were created for each classifier Five metrics were used for the performance emulation of the detection approaches. These include: true positive rate (TPR), true negative ratio (TNR), false positive ratio (FPR), false negative ratio (FNR), and weighted average F-measure. The definition of these metrics are as follows:

True Positive Rate (TPR)

(Recall) TPR is defined as the correctly predicted value of the malware classifier. It is

It is calculated as follows $TPR = \frac{TP}{TP+FN}$ (1)

False Positive Rate (FPR)

Similarly, the FPR is defined as the incorrectly predicted value of malware. It is calculated as follows: $FPR = \frac{FP}{TN+FP}$ (2)

True Negative Ratio (TNR)

is defined as the ratio of normal that was correctly classified as normal. It is calculated as follows: $TNR = \frac{TN}{TN+FP}$ (3)

False Negative Ratio (FNR)

is defined as the ratio of malware predicted that was incorrectly classified as normal It is calculated as follows: $TNR = \frac{FN}{TP+FN}$ (4)

Precision and recall are the basic measures that are mainly used to evaluate the performance of classification. Precision is defined as the positive predictive value, which is mainly used to return the relevant results. Moreover, it is a measure of accuracy that provides a predicted number of class as follows.

Precision $= \frac{TP}{TP+FP}$ (5) Recall= $\frac{TP}{TP+FN}$ (6)

F-Measure

F-measure is a combination of both precision and recall, when the same weight is given. Moreover, higher values of precision, recall, and f-measure indicates that the efficiency of classification [15]. It is calculated as follows:

$$F - Measure = \frac{2\times(Recall\times Precision)}{Recall+Precision} \quad (7)$$

Accuracy The proportion of the total number of the apps that are correctly classified whether as benign or malware [15]. It is calculated as follows:

$$\text{Accuracy} = \frac{TP+TN}{TP+FP+FN+TN} \quad (8)$$

True positives (TP) are the number of malware samples that are correctly classified, while false negatives (FN) are the number of malware samples that are incorrectly classified. True negatives (TN) are the number of benign samples correctly classified, whereas false positives (FP) are the number of benign samples incorrectly classified. The F-measures are the accuracy metrics that include both recall and precision.

4.4 RESULTS AND DISCUSSIONS

The main objective of this paper is to suggest an approach to detect malware under the Android platform and to achieve this we have taken several steps starting from dataset until machine learning models training, testing, and evaluation. It was used Dataset consisting of feature vectors of 215 attributes extracted from 15,036 applications The collected dataset is good enough to build android malware detection models, but it has an imbalanced class distribution problem. This problem related to the Android malware detection domain where there are many more benign applications than malicious. This causes a problem and affects the model's performance. There are many methods used to deal with the imbalanced dataset such a OrdinalEncoder and transform. The feature-matrix is created during the feature selection stage. The matrix is then used for training, testing, and evaluating machine learning models. use four classifiers to evaluate them. The five metrics T P R, F P R, calculated for this mode of supplying the dataset are shown in Table 4.

Table 4: Explain FPR and TPR of existing proposed techniques

ML Method	TPR	FPR
DT	0.971	0.0470
RF	0.968	0.0162
kNN	0.973	0.0468
LOG	0.962	0.0599

In this research using two different types of evaluation methods 10-fold cross-validation, 66 % split-validation and the results show that the classifier with the best accurse Random Forest, K-Nearest Neighbor, Decision Tree, Logistic Regression

Splitting Dataset

The first method is to divide dataset into percentages, which means that classification results are evaluated on a subset of the original data. then divided dataset by 66 percent for evaluation. Table 5 displays the tow metrics Prec., Recall and F- measure, accuracy calculated for this mode of supplying the data set show in Figure 4. the accuracy for splitting data set is given. The Figure 4 shows the Random Forest is the highest accuracy it has achieved accuracy in detecting malware of 97%. and Logistic Regression lower accuracy it has achieved 95%

Table 5: Algorithms evaluation Splitting dataset validation

ML Algorithms	Recall	Precision
Decision Tree	97	97
Random Forest	96.8	99
K-Nearest Neighbor	97	97
Logistic Regression	96	96

Table 6: Algorithms evaluation 10-fold validation

ML Algorithms	Recall	Precision
Decision Tree	96	97
Random Forest	96.2	98
K-Nearest Neighbor	95	94.9
Logistic Regression	85.2	83

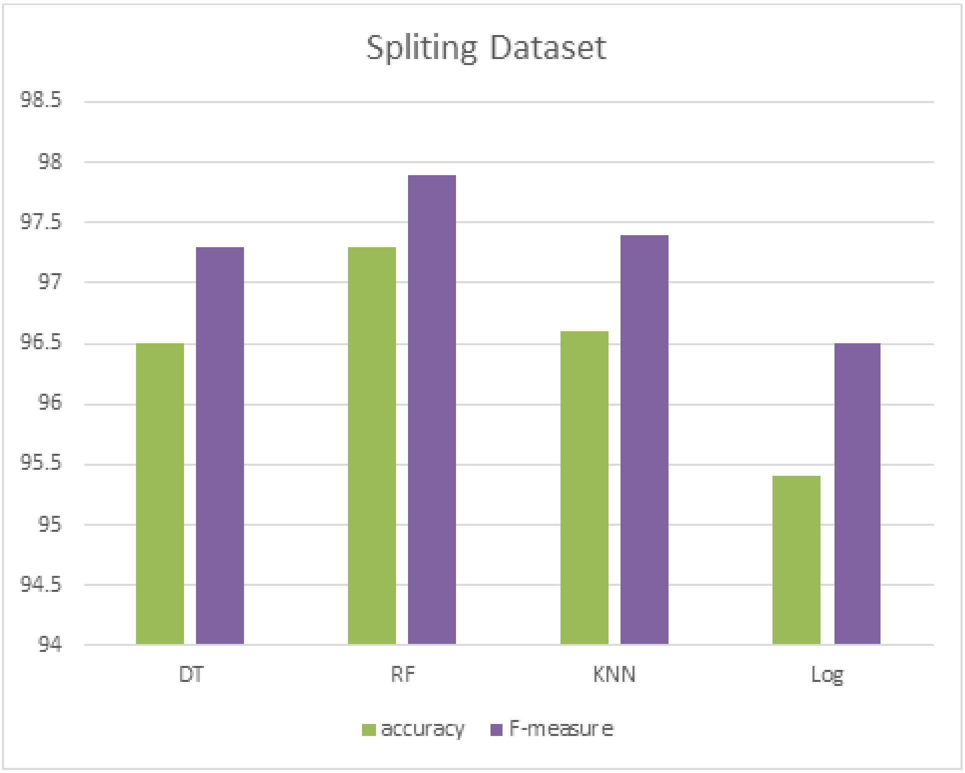
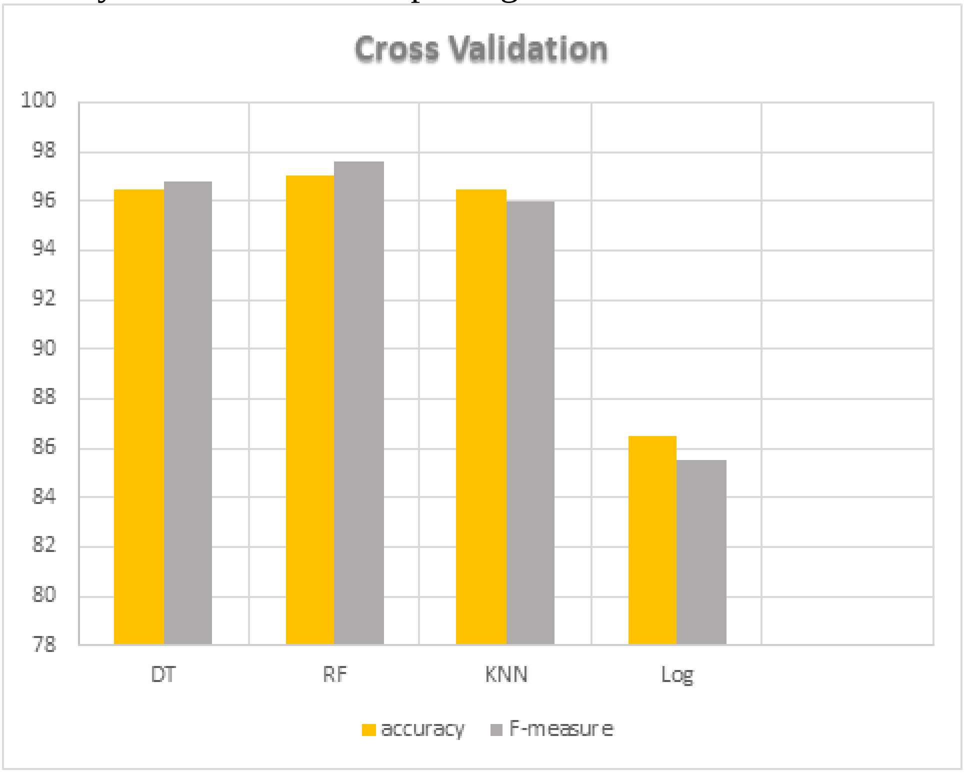
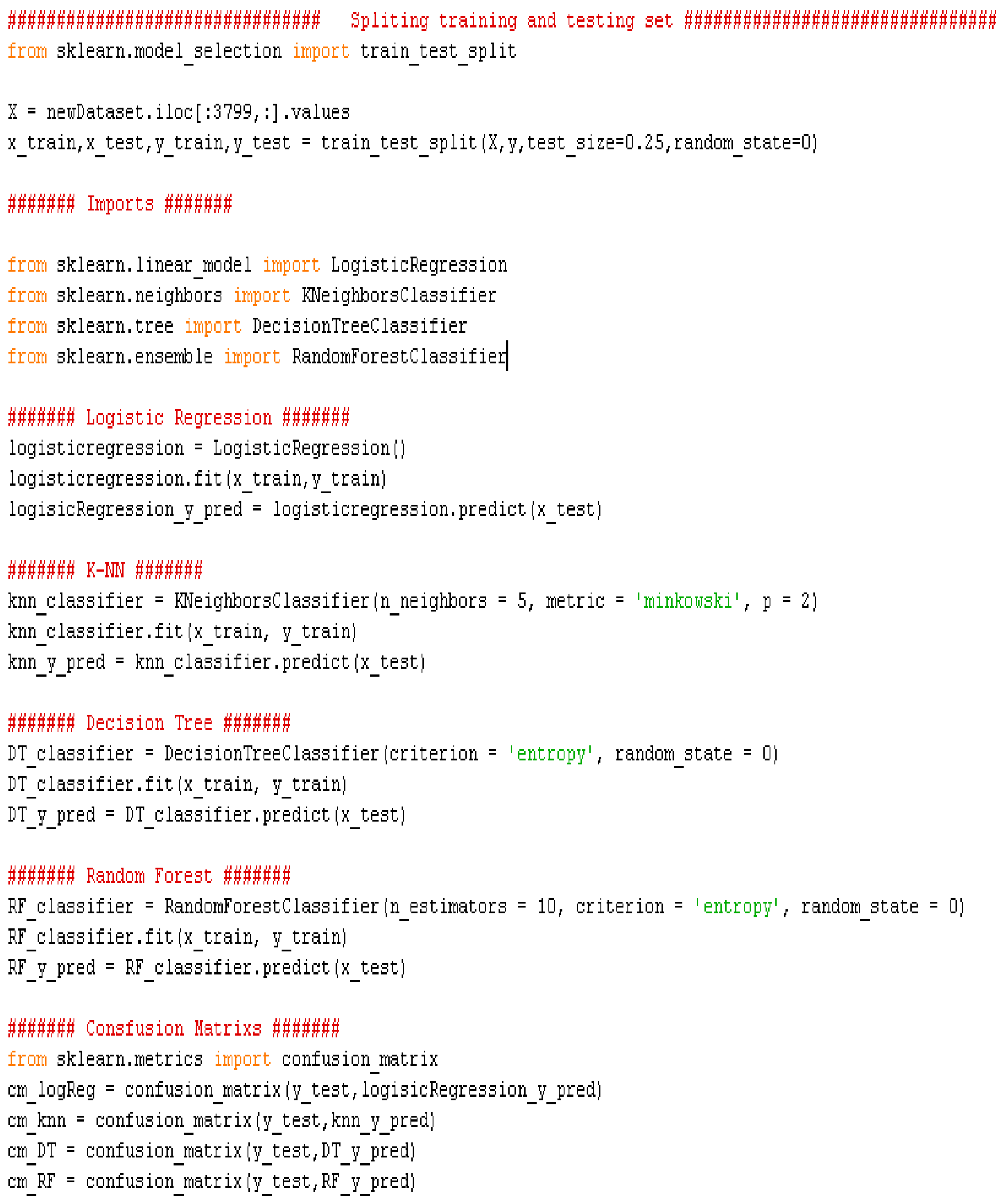


Figure 4: Accuracy and F-measure of splitting data set for all classifiers.



. Figure 5: Accuracy and F-measure of cross validation for all classifiers



```
################################   Spliting training and testing set ################################
from sklearn.model_selection import train_test_split

X = newDataset.iloc[:3799,:].values
x_train,x_test,y_train,y_test = train_test_split(X,y,test_size=0.25,random_state=0)

####### Imports #######

from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

####### Logistic Regression #######
logisticregression = LogisticRegression()
logisticregression.fit(x_train,y_train)
logisicRegression_y_pred = logisticregression.predict(x_test)

####### K-NN #######
knn_classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
knn_classifier.fit(x_train, y_train)
knn_y_pred = knn_classifier.predict(x_test)

####### Decision Tree #######
DT_classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
DT_classifier.fit(x_train, y_train)
DT_y_pred = DT_classifier.predict(x_test)

####### Random Forest #######
RF_classifier = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_state = 0)
RF_classifier.fit(x_train, y_train)
RF_y_pred = RF_classifier.predict(x_test)

####### Consfusion Matrixs #######
from sklearn.metrics import confusion_matrix
cm_logReg = confusion_matrix(y_test,logisicRegression_y_pred)
cm_knn = confusion_matrix(y_test,knn_y_pred)
cm_DT = confusion_matrix(y_test,DT_y_pred)
cm_RF = confusion_matrix(y_test,RF_y_pred)
```

. Figure 6: Explain code for Splitting Dataset by using SKlearn library

Cross Validation

WEKA divides the data set into 10 parts (called "folds"), holds out each part in turn, and then averaged nearly the results to perform 10-fold cross-validation. As a result, each data point in the data set is tested once and trained nine times. Table 6 displays the results obtained by employing four different machine learning classification methods. Figure 5 depicts the percentage accuracy achieved by all four classifiers. The Figure 5 shows the Random Forest is the highest accuracy it has achieved accuracy in detecting malware of 97%. and Logistic Regression lower accuracy it has achieved 86.5%. K-Nearest Neighbor and Decision Tree reports the equal second highest accuracy of 96.5%.

Processing time

Processing time is defined as the amount of time required to complete the task, which is expressed in terms of seconds. Furthermore, it is clear that the time required to build the model with a limited number of features is regarded as effective. According to the results of the analysis, the proposed random forest requires less processing time than the other methods. Because the proposed technique reduces computation complexity to 0.10 s, it takes less time to classify malware.

DISCUSSION

The aim of this paper is design a module to detect the android malware to achieve this It was used Dataset consisting of feature extracted from 15,036 applications (5,560 malware apps from Drebin project and 9,476 benign apps). The dataset has been used to develop and evaluate multilevel classifier fusion approach for Android malware detection, published in the IEEE. The dataset has been many processed. This processed used to extract the best features by using graph approach which are manifest permissions, API calls signature. Evaluate this model by using four machine learning classifier techniques which are Random Frost, K-Nearest Neighbor, Decision Tree, Logistic Regression. In this model interested in the Recall metric than other metrics because the effect of classifying benign applications as malware is less than the effect of classifying malware applications as benign applications table 7 show that.

Table7: Features categories Experiment Results

no	Features categories	No of Features	Recall
1	Permissions only	110	95
2	API calls only	73	96
3	Permissions & API calls	182	97.3

In summarization, the experimental results demonstrate that our detection method can identify Android malware applications with 97.3 percent accuracy and 96.8 percent recall for the best classifier. On average, the proposed method takes 5 seconds to analyze.

5 Conclusion

In this paper, a static approach is proposed to detect Android malware which concentrates on feature selection and feature generation using graphs. Combine the two raw features permissions and API calls to create new features and train the classifier using machine learning techniques. The evaluation results demonstrate that random forest is the best feature set classification technique. Achieving accuracy 97.3% and recall 96.8% the proposed approach requires 10 seconds for analysis on average. At the moment, the approach can only provide a binary classification for an application: malware or benign, but it cannot differentiate the malware family group of the application or reveal the effects of malware payload on the application's behaviors. These two flaws will be handled in the near future by enhancing existing work.

Dataset Availability

http://ieeexplore.ieee.org/document/8245867/ or
https://www.sec.cs.tu-bs.de/~danarp/drebin/download.html

6 References

[1] Sharma, S., Kumar, R. and Rama Krishna, C., 2021. A survey on analysis and detection of Android ransomware. *Concurrency and Computation: Practice and Experience*, p. e6272.
[2] Rodríguez-Mota, A., Escamilla-Ambrosio, P.J. and Salinas-Rosales, M., 2017. Malware analysis and detection on Android: the big challenge. In *Smartphones from an Applied Research Perspective*. IntechOpen.
[3] Mahindru, A. and Singh, P., 2017, February. Dynamic permissions based android malware detection using machine learning techniques. In *Proceedings of the 10th innovations in software engineering conference* (pp. 202-210).
[4] Lee, Y.K. and Kim, D., 2020. A Taxonomy for Security Flaws in Event-Based Systems. *Applied Sciences*, *10*(20), p.7338.
[5] Kakavand, M., Dabbagh, M. and Dehghantanha, A., 2018, November. Application of machine learning algorithms for android malware detection. In *Proceedings of the 2018 International Conference on Computational Intelligence and Intelligent Systems* (pp. 32-36).
[6] Kumaran, M. and Li, W., 2016, November. Lightweight malware detection based on machine learning algorithms and the android manifest file. In *2016 IEEE MIT Undergraduate Research Technology Conference (URTC)* (pp. 1-3). IEEE.
[7] Singh, A.K., Jaidhar, C.D. and Kumara, M.A., 2019. Experimental analysis of android malware detection based on combinations of permissions and API-calls. *Journal of Computer Virology and Hacking Techniques*, *15*(3), pp.209-218.
[8] Mcdonald, J., Herron, N., Glisson, W. and Benton, R., 2021, January. Machine Learning-Based Android Malware Detection Using Manifest Permissions. In *Proceedings of the 54th Hawaii International Conference on System Sciences* (p. 6976).
[9] Jerlin, M.A. and Marimuthu, K., 2018. A new malware detection system using machine learning techniques for API call sequences. *Journal of Applied Security Research*, *13*(1), pp.45-62.
[10] Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K. and Siemens, C.E.R.T., 2014, February. Drebin: Effective and explainable detection of android malware in your pocket. In *Ndss* (Vol. 14, pp. 23-26).
[11] Ananya, A., Aswathy, A., Amal, T.R., Swathy, P.G., Vinod, P. and Mohammad, S., 2020. SysDroid: a dynamic ML-based android malware analyzer using system call traces. *Cluster Computing*, pp.1-20.
[12] Mahindru, A. and Singh, P., 2017, February. Dynamic permissions based android malware detection using machine learning techniques. In Proceedings of the 10th innovations in software engineering conference (pp. 202-210).
[13] Afonso, V.M., de Amorim, M.F., Grégio, A.R.A., Junquera, G.B. and de Geus, P.L., 2015. Identifying Android malware using dynamically obtained features. *Journal of Computer Virology and Hacking Techniques*, *11*(1), pp.9-17.
[14] Mahindru, A. and Singh, P., 2017, February. Dynamic permissions based android malware detection using machine learning techniques. In *Proceedings of the 10th innovations in software engineering conference* (pp. 202-210).
[15] Jerlin, M.A. and Marimuthu, K., 2018. A new malware detection system using machine learning techniques for API call sequences. *Journal of Applied Security Research*, *13*(1), pp.45-62.