Journal of Cases on Information Technology
Volume 24 • Issue 3

Malware Detection in Android Apps Using Static Analysis

Nishtha Paul, Jaypee Institute of Information Technology, Noida, India
Arpita Jadhav Bhatt, Jaypee Institute of Information Technology, Noida, India
Sakeena Rizvi, Jaypee Institute of Information Technology, Noida, India
Shubhangi, Jaypee Institute of Information Technology, Noida, India

ABSTRACT

Current studies have revealed startling facts about data harvesting incidents, where user's personal data is at stake. To preserve the privacy of users, a permission-induced risk interface MalApp to identify privacy violations rising from granting permissions during app installation is proposed. It comprises of a multi-fold process that performs static analysis based on app's category. First, the concept of reverse engineering is applied to extract app permissions to construct a Boolean-valued permission matrix. Second, ranking of permissions is done to identify the risky permissions across category. Third, machine learning and ensembling techniques have been incorporated to test the efficacy of the proposed approach on a data set of 404 benign and 409 malicious apps. The empirical studies have identified that the proposed algorithm gives a best case malware detection rate of 98.33%. The highlight of interface is that any app can be classified as benign or malicious even before running it using static analysis.

KEYWORDS

Android OS, Benign, Ensembling, Heuristic Approach, Machine Learning, Malware, Permission Ranking, Permissions, Risk-Relevance

1. INTRODUCTION

It is nothing much surprising that the mobile app industry is flourishing and growing immediately after 2008 when the first app was launched. In today's era, everyone's life revolves around smartphones, which has raised a serious concern for security and privacy. One of the biggest issues being faced by this trending technology is the protection of smartphone devices against various security threats and prevention of user's privacy leaks. One of the most exciting features of smartphones is that their functionalities can be expanded by installing third-party applications called as 'apps'. Android is one of the most popularly used platforms for smartphones with billions of apps available on its official 'Play Store'. As reported by buildfire statistics, there were 2.7 billion smartphone users in the world in 2019 and on an average, a smartphone user uses about 30 apps per month (Blair, n.d.). Apps for the Android OS are mainly written in the JAVA and KOTLIN language (Bose, 2018). With the increase in the development of Android apps since 2008 when the first android app was launched, there are over 2.9 million apps available for download on the Google Play Store globally (Market.us, n.d.). However, this has also resulted in increase of privacy breach of users. App usage rate is increasing at a steady rate with no signs of decline in the future. Forbes stated that the testing specialist researchers

DOI: 10.4018/JCIT.20220701.oa6

at Comparitech had found out that the apps which had been installed for more than 28 million times were detected to show attack paths to threat actors looking to exploit vulnerabilities on the Android platform (Winder, n.d.). The apps were scanned for dangerous permissions and trackers embedded within them.

To summarize on one hand, the smartphones provide the users an open platform to download third party apps from play stores and have fun and enjoyment, on the other hand, they bring loads of unknown risk to the security of the personal data of the users. The users so conveniently download the apps without foreseeing the damage they can bring in their lives. Studies have revealed that the users blindly grant the permissions to run the app in such a hurry that they can't foresee the harm that app can cause. Therefore, it is the responsibility of the developers to request only relevant permissions and they should be transparent so that the users can make informed decisions.

Research studies have also revealed that malware apps can steal sensitive personal information such as login credentials, biometric information, financial and banking information. Additionally, the apps are also capable of accessing gallery images of the owner and other people, videos, important documents, contact details, call logs, messages, emails, location details, IMEI number, IP address, etc. (TermsFeed, n.d.). Hence analysis of apps needs immediate attention. To analyze behaviour of apps lot of researchers have performed static analysis of apps to prompt the users and to warn them even before running the app about its risky behaviour.

Various studies discussed in Section II have detected privacy violations by Android apps data sets comprising thousands of apps irrespective of the category of the app. In this work, we have incorporated app's category which plays a major role in app analysis process to identify malicious behavior. Apps on Android platform are sorted based on category. An app's category is very important as it depicts the general behaviour and function of an app. To substantiate the fact, an app belonging to photo and video category will require access to user's photo album and camera to function well. Likewise, there are several other permissions like location, Bluetooth, SMS read/write permission, microphone etc. Therefore, it is the responsibility of developer to choose app's category wisely before uploading on Play Store. Currently there are 35 categories available on Google Play Store, therefore permissions play a very important role in our research study (Play, n.d.). As use of some permissions such as user's location might be very useful in maps and navigation category while the same permission might be dangerous for some other categories such us. As permissions embedded in app and category plays a very important role to identify basic functionality of an app. In the course of this study, we have performed category-based analysis of the risky permissions requested by android apps when they are being downloaded and before they are run to an Android device.

A brief summary of the prime contributions of the proposed work is listed level by level as follows:

- This work presents an interface developed in JAVA which uses Python at backend to classify unknown malicious apps. The interface takes an unknown android app's .apk file along with its category and predicts whether it is fit to use or not even before running the app.
- Incorporates reverse engineering approach using the 'apktool' to extract app permissions from AndroidManifest.xml to construct a Boolean-valued permission matrix P_{mxn}, where m represents number of apps under analysis from various categories and n is the number of permissions. We have tested about 60 apps for every category over a total of 324 android permissions.
- Applies systematic risk ranking approach to rank every permission from each category based on risk-relevance using Correlation Coefficient. Out of 234 permissions, the top 20 risky ones were chosen in every category.
- Incorporates data heuristic approach that assigns weights to permissions based on category and risk-relevance by assigning higher heuristic value to a more riskier permission. This helps to improve the performance of the model.
- Uses machine learning and data ensembling techniques to analyse the behaviour of 404 benign and 409 malicious apps. Several machine learning classifiers namely naive bayes, SVM, decision

tree (J48), random forest, as well as ensembling techniques such as bagging, boosting, and voting have been employed. The proposed malware detection interface has been evaluated on several performance metrics such as True Positive Rate (TPR), False Positive Rate (FPR), Accuracy and Area Under the Receiver Operating Characteristic (AUROC) curve.

- Applies a comprehensive analysis of performance metrics and chooses the best combination of classifier and permission matrix for each category.
- The proposed interface is capable of testing the new unknown app on the trained model and predicts the final output as Benign or Malware.

As complementary parts to this introduction, Section II briefs the related research papers consulted for the study. Section III describes the data set. Section IV describes the methodology of performing the static analysis with the generation of sparse and heuristic data set followed by classification of malware apps using various classifiers. Section V shows how the proposed interface is used to detect the malicious behaviour of an unknown android app before the whole research work is concluded in Section VI. Section VII finally describes the possible future work.

2. RELATED WORK

This research work is surfeited with white papers, journals, research articles and papers which focus on exploring the risks and vulnerabilities associated with the use of android permissions. This section provides an outline of some related studies in this area which have been referred to carry out behavioural analysis of Android apps.

2.1 Reverse Engineering

The process of reverse engineering is used for analysing the behaviour of an app to determine the potential sources of privacy leaks. Reverse engineering can be applied using static analysis or dynamic analysis. During static analysis install time behaviour of app is analysed while in dynamic analysis run-time behaviour of the app is analysed (Chikofsky & Cross, 1990). Elliot J. Chikofsky describes reverse engineering as a tool used to extract design information or knowledge from anything made by man and use it to reproduce something new and effective (Chikofsky & Cross, 1990). Siegfried Rasthofer discussed that purely automatic analysis is usually not enough for detecting the privacy leaks and malware apps (Rasthofer et al., 2016). To identify the behaviour of the explicit permissions which can be used to leak personal data, human intervention is necessary. This can be done via some tools that can assist manual investigation. CodeInspect, an absolutely innovative reverse engineering tool for the Android apps which can be used to support analysts and investigators was proposed by (Rasthofer et al., 2016). Initially, a variety of mobile OS such as Blackberry, Bada, Windows etc. were designed but got discontinued because they could not compete the momentum already built by Android and iOS. Some of the best reverse engineering tools for iOS include lldb, otool, nm, etc. In this work we have used Apktool for reverse engineering. Taranjeet Kaur Chawla and Aditi Kajala have surveyed many android apps by reverse engineering them and used many types of reverse engineering tools such as Dex2jar, Apktool, JD-GUI etc. (Chawla & Kajala, 2014). May Thu Kyaw used reverse engineering and examined many android apps for checking the malicious activities (Myat, 2019). Their results demonstrated that most malware apps inject the malicious codes and unnecessary permission in the AndroidManifest.xml file. Lipo Wang has adapted three most popular Android app reversing tools, including dex2jar, Apktool, and Soot, which transform the executable source into Jasmin, Smali, and Jimple ILs, respectively (Arnatovich et al., 2018). In this work, reverse engineering in form of static analysis has been used. Some of the prominent work in field of static analysis have been described below.

2.2 Static Malware Analysis

Wei Wang employed three feature ranking methods, namely, correlation coefficient, mutual information, and T-test to analyse the risky behaviour of a group of collaborative permissions and rank them with respect to their risk (Wang et al., 2014). The results of this paper showed that their malapp detectors which were based on risky permissions gave satisfied performance with a fair detection rate of 94.62% with a False Positive Rate of 0.6%. Fauzia Idrees presented a unique Android based malware detection tool, named AndroPIn, whose basic research elements were android permissions and intents (Idrees et al., 2017). Their framework overcame the limitations of surreptitious techniques used by malware by exploiting the usage pattern of permissions and intents. Hyunjae Kang proposed a system which enabled fast detection of malware by using creator information such as serial number of certificates (Kang et al., 2015). Additionally, it analyzed malicious behaviours and permissions to increase detection accuracy. The system could also classify malware apps based on some similarity scoring. The paper concludes that the detection and classification performance of their presented mechanism was 98% and 90% accuracy, respectively.

2.3 Classification Algorithms

Previous work compiled in (Fereidooni et al., 2016; M & M.N, 2015; Rashidi et al., 2017; Sahal et al., 2018; Sun et al., 2018; Tao et al., 2018; Yu et al., 2013) mainly focus on applying various algorithms to analyse the risky permissions in android apps. M. Hossin systematically reviewed many evaluation metrics and analysed that accuracy is generally considered while comparing the results of various classifiers but it has several weaknesses like less discriminability, less distinctiveness, less informativeness and it is bias to majority class data (M & M.N, 2015). The paper also proposed some other metrics that are specifically designed for discriminating the optimal solution. Hossein Fereidooni has proposed a technique to classify apps to benign or malware by employing several algorithms for classification such as SVM, XGBoost, KNN, Logistic Regression, Naive Bayes, Decision Tree, Random Forest classifiers and Deep Learning using the scikit-learn python library (Fereidooni et al., 2016).

2.4 Features of our Interface compared with prior research

Inspired by iABC, a permission induced risk model which had been proposed specifically for iOS devices to detect the privacy breach arising due to permissions granted at the run-time (Bhatt et al., 2018). We have implemented a similar technique in the form of a JAVA interface for apps built for Android OS. As compared to the existing works in the research world of android apps, categorized analysis of malicious behaviour of apps has not been adequately. Therefore, the center of attention which differentiates our work with the prior studies is the app's category which is taken into consideration while detecting malicious apps.

During our study, neither the permission vector sets of android apps were classified on the basis of categories nor was the apps' source code readily available in data repositories. Reverse engineering technique was applied to extract permissions from app's manifest file and analyse their source code. The feature ranking technique discussed in (Wang et al., 2014) was used to rank and weigh permissions according to their risk-relevance. Top 20 risky permissions were selected and then weighing factors were assigned based on their category. New permission matrix P'_{mxn} was constructed after assigning weighing factors based on permission ranking using heuristic approach. After examining various papers on machine learning techniques and research works on static analysis, a few were shortlisted to train the model and then test an unknown app to predict its malicious behaviour.

3. DATA SET

A plethora of research has been performed in the field of malware analysis of android apps but minimal work has been conducted that analyses apps on the basis of their category. In order to conduct an extensive examined research over the risky behaviour of explicit and implicit android permissions, our study required a large data set comprising of permission vectors of each app and a note of their respective categories.

The data set was manually constructed by downloading 30 apks of benign android apps Google's official store Play Store (Play, n.d.) and 30 apks from third party malware stores like vShare (vShare, n.d.) and aptoide (Aptoide, n.d.) for 14 android categories, namely, art & design, augmented reality, auto & vehicles, beauty, books, comics, communication, house & home, lifestyle, libraries & demo, maps & navigation, personalization, videoplayers & editors and weather. A total of 404 benign and 409 malware android apks were reverse engineered for analysis.

In this work, 324 permissions have been considered which can contain implicit as well as explicit permissions provided by the Android system. At the development stage, the app developer has to explicitly request the permissions by including them in the AndroidManifest.xml file. All the permissions are listed in the Manifest file of the app's folder. In this work, static analysis on the android apps has been done which aims to analyse the behaviour of apps before they start running by analysing their code. *apktool* has been utilized to reverse engineer all apks. Later, DOM XML parser was used to read the AndroidManifest.xml file and extract all the permissions written in it. Finally, for every category, each app was denoted by a 324 - dimensional boolean vector wherein 1 denotes that the app requests a particular permission and 0 if it doesn't. This sums up the sparse data set of the research work. After construction of permission matrix, machine learning approach was applied to train classifiers and predict the behaviour of app. Thereafter, a heuristic approach was introduced to redesign these permission vectors in an attempt to improve the malware classification performance which will discussed later in detail based on permission ranking. The following section describes the static analysis approach in detail.

4. STATIC APP ANALYSIS

Static Analysis means an analysis that has been simplified in which the effect of any instantaneous change to a system is calculated without considering the longer-term response of that system to the change. On the other hand, dynamic analysis is done to consider how the particular system is expected to respond to the change over time (Wikipedia, n.d.).

Static analysis performs the exploration of a system or a software without actually executing it. This is achieved through the analysis of the source code or the binaries of the system. This process helps in understanding the code structure, and also helps to ensure that the code adheres to industry standards.

For classifying behaviour of Android apps, different static features such as permissions, Inter-process communication (IPC), developer ID, API calls, intents, hardware, code semantics and components have been used to detect malware (Idrees et al., 2017). However, features like permissions, API calls, and IPC have gained more attention from the researchers. Android apps run as a separate virtual machine on the device, so by default it does not have any authority to perform operations such as accessing resources such as SMS, contacts, gallery etc. or altering the settings of Wi-Fi, Bluetooth etc. It requires the consent or the approval of the user to perform any action on the device. This is achieved via the concept of permissions. Apps can explicitly request these privileges through permissions (Idrees et al., 2017).

As discussed in the prior sections, it is quite evident that misuse of these permissions can cost the users massively. The objective of study in this paper is to perform a static analysis on android apps by examining the permissions requested and the key category factor and finally detecting whether an app is benign or malware without running it.

This section gives an overview of the static analysis performed and the results which will be passed in the interface for testing the malicious behaviour of an unknown android app.

4.1 Static analysis pre work phase - Generating Dataset

As previously discussed, due to the lack of availability of dataset divided on the basis of Android categories, the analysis couldn't begin without generating the whole dataset from scratch. In this stage of the first phase of the model, a technique was implemented to extract a list of permissions requested by an app without actually running the app.

4.1.1 Reverse Engineer Android Apps

An APK (Android Package) is a comprehensive file, most probably a ZIP file that encapsulates the code, resources, signature, manifest and several other files that need to be executed in order to have the complete Android app up and running. AndroidManifest.xml comprises the permissions requested by the app. Being aware of the fact that security is highly correlated to android permissions, the main task was to get hold of this file.

The Android OS works on code from the "Android Open Source Project," or AOSP. This code is open-source and hence, developers can fetch the source code and create their own operating systems from it (Hoffman, n.d.). To substantiate this fact, reverse engineering was applied using 'apktool', which is a java-based platform-independent tool used to decode resources to nearly original form and reconstruct them after making some modifications. This tool was used to reverse engineer the android apks and convert them into their respective folders so as to access the AndroidManifest.xml file.As previously stated, Google Play categorizes android apps under 35 different categories. Aiming to study the behaviour and use of various permissions under different categories, 14 most prominent ones were shortlisted for the research work.

According to the need of training the interface, 30 android apks from Play Store (Play, n.d.) under the 'benign' label and 30 apks from third party malware stores such as vShare (vShare, n.d.) and aptoide (Aptoide, n.d.) under the 'malware' label were downloaded across 14 categories. The list of categories chosen for analysis and the number of apps installed are shown in Table 1. With the objective of performing the study, in total 813 apps were installed and then reverse engineering was performed on every apk individually. Following command was utilized to decompile an apk

```
apktool d name-of-app.apk
```

Figure 1 shows the terminal logs while running apktool on a sample apk.

4.1.2 Construct Permission Matrix P_{mxn}

For the course of the research work, the apps have been trained and analysed on 324 Android permissions (Paul, n.d.). A JAVA code was written that extracted the user-permissions from the Manifest.xml and generated a sparse i.e. binary valued permission matrix , where m denotes the no. of apps (both benign and malware) and n denotes number of android permissions. Each app was represented by a 324-dimensional Boolean vector, where 1 denotes that the app requests the permission and 0 otherwise. Figure 2 gives a snapshot of the sparse dataset for Art & Design category consisting of rows depicting apps and columns depicting permissions and the last column acts as a label which indicates if the app mentioned is benign or malicious.

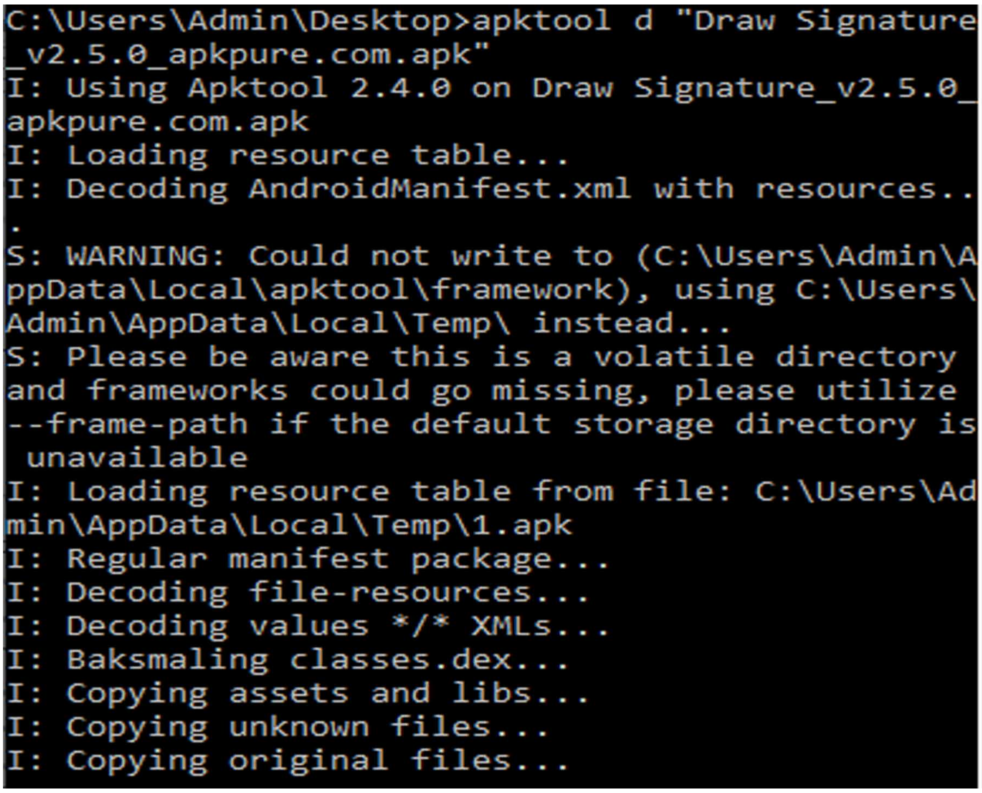
4.2 Sparse Phase

The objective of this phase is to study different machine learning algorithms by applying them on the sparse dataset created in the previous subsection and evaluate the results on the basis of performance metrics. Thereby, the goal was to select the one which gives the best performance so that it can be

Table 1. Category-wise number of applications

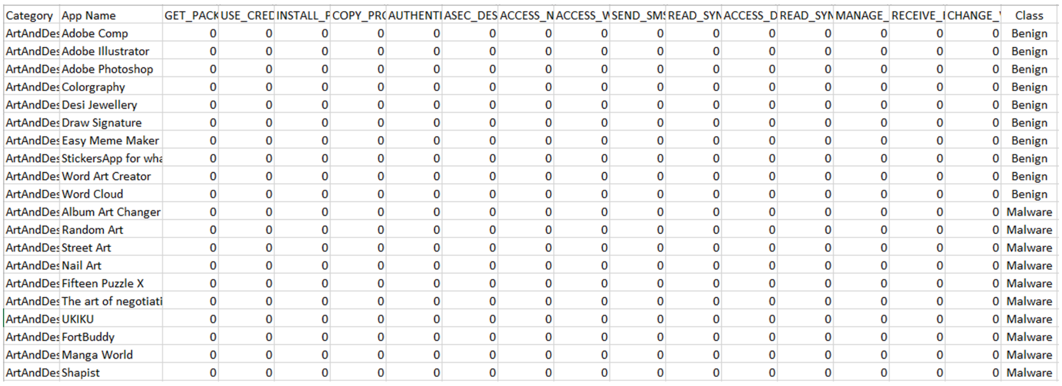
S. No.	Category	Benign	Malware
1	Art&Design	30	30
2	Augmented Reality	22	28
3	Auto&Vehicles	30	29
4	Beauty	30	30
5	Books	30	30
6	Comics	22	22
7	Communication	30	30
8	House&Home	30	30
9	Libraries&Demo	30	30
10	Lifestyle	30	30
11	Maps&Navigation	30	30
12	Personalization	30	30
13	VideoPlayers&Editors	30	30
14	Weather	30	30
Total Number of Applications		404	409

Figure 1. Working of apktool



```
C:\Users\Admin\Desktop>apktool d "Draw Signature
_v2.5.0_apkpure.com.apk"
I: Using Apktool 2.4.0 on Draw Signature_v2.5.0_
apkpure.com.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources..
.
S: WARNING: Could not write to (C:\Users\Admin\A
ppData\Local\apktool\framework), using C:\Users\
Admin\AppData\Local\Temp\ instead...
S: Please be aware this is a volatile directory
and frameworks could go missing, please utilize
--frame-path if the default storage directory is
 unavailable
I: Loading resource table from file: C:\Users\Ad
min\AppData\Local\Temp\1.apk
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Baksmaling classes.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...
```

Figure 2. Snapshot of Sparse Permission Matrix (P_{mXn}) of Libraries & Demo category



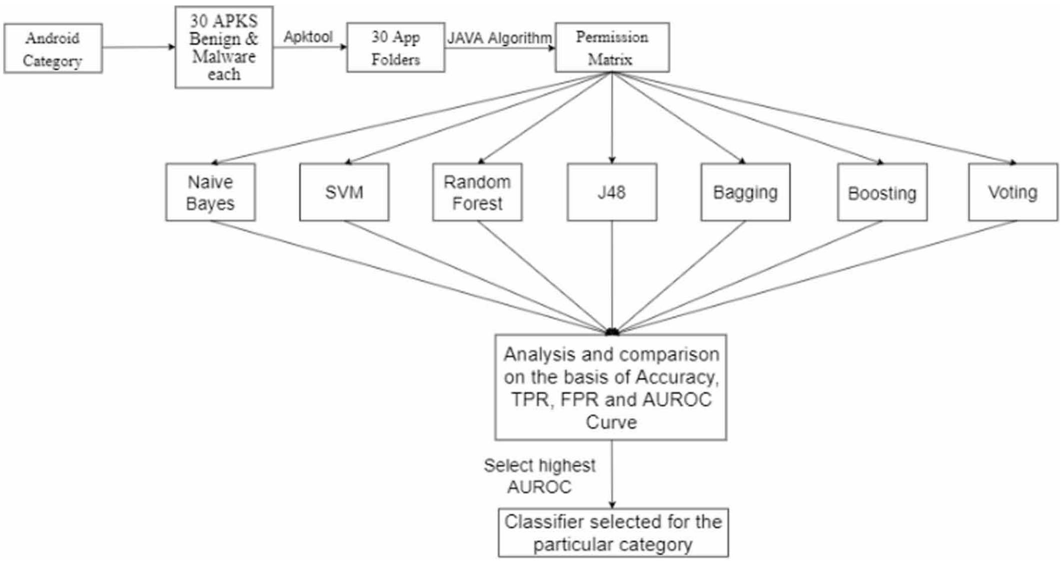
Category	App Name	GET_PACK	USE_CRED	INSTALL_F	COPY_PRC	AUTHENTI	ASEC_DES	ACCESS_N	ACCESS_V	SEND_SMS	READ_SYN	ACCESS_D	READ_SYN	MANAGE_	RECEIVE_I	CHANGE_'	Class
ArtAndDes	Adobe Comp	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Benign
ArtAndDes	Adobe Illustrator	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Benign
ArtAndDes	Adobe Photoshop	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Benign
ArtAndDes	Colorgraphy	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Benign
ArtAndDes	Desi Jewellery	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Benign
ArtAndDes	Draw Signature	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Benign
ArtAndDes	Easy Meme Maker	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Benign
ArtAndDes	StickersApp for wha	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Benign
ArtAndDes	Word Art Creator	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Benign
ArtAndDes	Word Cloud	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Benign
ArtAndDes	Album Art Changer	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Malware
ArtAndDes	Random Art	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Malware
ArtAndDes	Street Art	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Malware
ArtAndDes	Nail Art	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Malware
ArtAndDes	Fifteen Puzzle X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Malware
ArtAndDes	The art of negotiati	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Malware
ArtAndDes	UKIKU	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Malware
ArtAndDes	FortBuddy	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Malware
ArtAndDes	Manga World	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Malware
ArtAndDes	Shapist	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Malware

used to select the best classifier for the testing the apps in the interface. Figure 3 depicts a flowchart describing the course of action of this phase. Following subsection describes the machine learning algorithms for training the classifiers and predict the behaviour of apps.

4.2.1 Machine Learning Algorithms

After extracting the features from apps and constructing the sparse dataset for each category, seven machine learning algorithms were applied to investigate the effectiveness and select the best classifier for every individual category. The classifiers used for this purpose were built employing the following algorithms on python and its library Scikit-learn.

Figure 3. Flowchart depicting the malware classification technique


Android Category
30 APKS Benign & Malware each
Apktool
30 App Folders
JAVA Algorithm
Permission Matrix
Naive Bayes
SVM
Random Forest
J48
Bagging
Boosting
Voting
Analysis and comparison on the basis of Accuracy, TPR, FPR and AUROC Curve
Select highest AUROC
Classifier selected for the particular category

1. Naive Bayes

Naive Bayes classifier originates from the Bayesian theory of probability. The advantage of using Bayesian classifiers is that these classifiers are probabilistic models. They work powerfully when it comes to using real data noise and missing values. It shows a good performance on both weak and strong attribute dependencies. The Gaussian Naive Bayes has been used from Scikit-learn library (De Ferrari & Aitken, 2006).

2. SVM (Support Vector machines)

The Support Vector Machine (SVM) is a statistical learning technique suitable for binary classification techniques. Its main goal is to form a "hyperplane" which divides the data instances into two classes:- positive and negative. The basic technique is to find out the smallest "hypersphere" present in training instances and to determine on which side of the hypersphere the test instance is present. The maximum margin hyperplane has the greatest separation between the classes. The instances closer to the maximum margin hyperplane are known as support vectors (Zareapoor & Shamsolmoali, 2015)

3. Decision Trees (J48)

A decision tree classifier is considered as a non-parametric classifier. It does not have a requirement of any prior statistical assumptions regarding the distribution of data. The data is divided recursively to create a decision tree in accordance with the attributes and the defined classification framework. A decision rule is required at every node which can be implemented using a type of test that splits the data (Otukei & Blaschke, 2010).

4. Random forest

In Random Forest algorithm many CART-like trees are created, that are trained on smaller samples of the original training data. The algorithm searches across a subset that is randomly selected. This is done to find a split. After that, all the trees present in the Random Forest at that time, cast a vote for the most popular attribute at input x. The majority votes determine the output of the classifier. This algorithm handles high dimensional data. Also, the random selection of variables for a split results in minimizing correlation among the trees present in the ensemble which further decrease the error rates (Gislason et al., 2006).

5. Bagging

Bagging is an ensembling technique designed to enhance the stability and accuracy of algorithms used for classification and regression. It combines the result of classifications of randomly generated training sets and forms a final prediction set. It works on a principle which is that many "weak learners" can be used to form a "strong learner". When the classification of a new instance is to be done, it is done repeatedly to each of the week classifiers in the ensemble. Each weak classifier casts a "vote" for a class. The final prediction is gained by that class which has maximum votes (Zareapoor & Shamsolmoali, 2015).

6. Boosting

Adaboost is a type of meta-algorithm. It can coexist with many other learning algorithms so that performance of the algorithm is improved. The working of the algorithm begins by assigning the same weight to all instances in the training data. Then the learning algorithm is called so that it forms a classifier for this data. It reweighs all the instances in accordance with output of the classifier. The weight instances that are classified by the algorithm correctly is decreased, whereas that of misclassified ones is increased due to which a set is created that has easy instances with low weight and another is created that has hard ones with high weight. Once again in the next iteration, a classifier is created and instances are reweighed, This is mainly done to focus on classifying the hard instances correctly. Then the weights assigned to instances are increased or decreased in accordance with the output of this new classifier (Korada et al., 2012).

7. Voting

The simplest ensemble algorithm is the voting method, and is very effective. It can be used for classification and regression problems. It works by making two or more sub-models. Every sub-model makes predictions that are combined in such a way, as by calculating the mean or mode of the predictions, it allows each sub-model to cast a vote on the outcome of the prediction. To really achieve the expected outcome from the voting ensemble, diverse base classifiers should be used (Ledisi & Ugochukwu, 2019).

4.2.2 Evaluation Metrics

The classifiers discussed in the previous subsection had been analysed and compared on the basis of various performance metrics such as the accuracy, TPR, FPR and Area under the ROC (Receiver Operator Characteristic) curve and the results were combined together to have a broader picture of the overall analysis (Neptune.ai, n.d.). This will later help us to determine what combination of i) type of data set and ii) classification algorithm trained would be most suited for each category to perform a predictive analysis of the risky behaviours of a new Android app. Following evaluation metrics were employed (Tharwat, 2018).

1. Classification Accuracy

Classification accuracy can be calculated as the ratio of number of correctly classified instances to overall classifications (Tharwat, 2018).

$$Accuracy = \frac{NumberOfCorrectClassifications}{TotalNumberOfSamples} \times 100 \tag{1}$$

2. True Positive Rate (TPR)/Recall

Performing analysis on ‘imbalanced class’ problems, it was noticed that accuracy results in a high value due to an overwhelming majority of instances belonging to a single class. The metric which should now be used is recall/TPR. It is the ratio of true positives to the data points that actually were positive.

$$TPR = \frac{TruePositives}{TruePositives + FalseNegatives} \times 100 \tag{2}$$

3. False Positive Rate (FPR)

This is the opposite of TPR. It measures the ratio of false positives to the actual number of negatives in a dataset.

$$FPR = \frac{FalsePositives}{FalsePositives + TrueNegatives} \times 100 \quad (3)$$

4. ROC Curve

A Receiver Operator Characteristic (ROC) curve is visualized as a graphical plot which shows the diagnostic ability of the binary classifiers. The ROC curve depicts the trade-off between the sensitivity (or TPR) and specificity (1 – FPR) (Neptune.ai, n.d.). TPR and FPR is calculated for every threshold and plot it on a single graph.

The higher TPR and the lower the FPR would result in a better threshold and hence, the classifiers whose curves are nearer to the top-left corner indicate a finer performance level. If the curve comes closer to the 45-degree diagonal of the ROC space i.e. FPR = TPR, then test or the classifier is considered to be very less accurate. It is not easy to compare different classifiers by solely analysing the ROC curve because there is no scalar value to represent the expected performance (Tharwat, 2018). Therefore, Area under the ROC Curve, or ROC AUC score is calculated to analyse how good the curve or the classifier is. The more top-left cornered the curve is the higher the area and thus higher AUROC score.

During the analysis, it was very difficult to choose a specific classifier for classifying the categories accurately solely depending on any one of the first three evaluation metrics. Therefore, ROC curve which does not depend on the class distribution was chosen for evaluating behaviour of apps. Accuracy, TPR and FPR were not considered as cost sensitive analysis measures. Overall classifiers' accuracy is based on one specific cutpoint, while ROC tries to include all of the cutpoints and plots the sensitivity and specificity curve. So while comparing the overall accuracy, actually, the accuracy is being compared based on a particular cutpoint. The overall accuracy varies from cutpoint to cutpoint.

For checking any classification model's performance, Area under the ROC Curve is one of the most important evaluation metrics to be used and hence this was chosen for proceeding forward (Tharwat, 2018).

4.3 Heuristic Phase

This phase introduces a heuristic approach used to improve the performance of the malware detection using various classifiers.

4.3.1 Google's guideline for app permission and category

The analysis performed till now was dependent solely on the frequency of the permissions requested by malware and benign apps. But, the analysts could not identify or pin point which all permissions were specifically responsible for the malicious behaviour of apps of a particular category. So, before proceeding with the coding of the interface, we studied the Google guidelines which are provided by Google to android developers to choose an appropriate category and the permissions to be requested explicitly. It is the responsibility of the developers to follow Google guidelines for incorporating the allowed user permissions for a category before uploading them to App store (Bhatt et al., 2018). The guidelines provided by Google clearly mentions that permission requests should make sense to users. Therefore, it is expected from the developers that only minimal essential permissions which are required to implement the features or services in the app are promoted in the Play Store listing and they must be requested for authorization. Those permissions which allows the access of user or device data for disallowed, unimplemented or undisclosed features or purposes should not be used. Sensitive or personal user data which can be accessed through permissions should never be sold (Google, n.d.).

Google's guidelines provides basic app functionalities for every category. Any additional features added by the developers to enhance and make it more useful might be detected as malicious or not. These are referred as utility functions. Some developers adopt mal-practices to increase the ASO (App Store Optimization) such as choosing the least competitive category, which might not be related to the app, so that their app gets a better chance to rank closer to the top. It should be kept in mind that putting an app in a blatantly wrong category can lead to serious trouble (MobiLoud, n.d.). The app might not be approved or as far as Google Play is concerned, users can report violations for review. It's only a matter of time before someone points them out.

4.3.2 Pearson's Correlation Coefficient for ranking permissions

Permissions are regarded as basic features that are used to describe functionalities of apps, and their behaviour indicates their attempts to have interactions with the data, the OS or other installed apps. In this work, X stands for a permission variable. To indicate the label of an app as benign or malware, a class variable is defined. In this work, C represents the class variable. The risk-relevance of requesting a permission could be examined by measuring the relevance between that permission or feature variable and the given class variable. A strong correlation between both of these depicts the increase in risk of granting that permission. Feature ranking, which is a measure of the relevance of any feature and class variables, tends to select the most informative features out of all and improves the performance of learned models (Singh et al., 2011; Wang et al., 2014). Pearson CorrCoef is used to measure the relevance between X and C.

$$R\left(X,C\right)=\frac{cov\left(X.C\right)}{\sqrt{var\left(X\right)var\left(C\right)}} \tag{4}$$

which for boolean variable and binary class becomes

$$R\left(X,C\right)=\frac{\sum_{n=1}^{N}\left(X_n-\bar{X}\right)\left(C_n-\bar{C}\right)}{\sqrt{\sum_{n=1}^{N}\left(X_n-\bar{X}\right)^2\sum_{n=1}^{N}\left(C_n-\bar{C}\right)^2}} \tag{5}$$

where $\bar{X}$ (resp. $\bar{C}$) called mean of the values of sample of X (resp. C), X_n (resp. C_n), n = 1...N. R(X,C) which has a value in [−1, 1], where $\mathrm{R}\left(\mathrm{X},\mathrm{C}\right)=0$ depicts the in dependency of X and C, $\mathrm{R}\left(\mathrm{X},\mathrm{C}\right)=1$ indicates the most positive correlation between them and $\mathrm{R}\left(\mathrm{X},\mathrm{C}\right)=-1$ denotes the most negative correlation. In this study, $\mathrm{R}\left(\mathrm{X},\mathrm{C}\right)=1$ means that particular permission that is requested by X makes apps prone to the highest risks, while $\mathrm{R}\left(\mathrm{X},\mathrm{C}\right)=-1$ depicts that permission requests of X makes the apps prone to lower risks.

4.3.3 Ranking Permissions w.r.t Risk-Relevance

Pearson Correlation Coefficient was used on every category to rank all the 324 permissions and a greater value of the coefficient indicates a riskier permission. Category wise permission matrix $\mathrm{P}_{\mathrm{mxn}}$

was used to identify the top 20 risky permissions. After ranking the risky permissions, the results were utilized and Play Store guidelines were used to assign weighing factors using heuristic approach.

To demonstrate the permission ranking results, ranking of the top 20 risky permissions of Libraries & Demo category have been displayed in Table 2 in the decreasing order of correlation coefficient. Similar work was performed for all the other categories.

Table 2. Ranking of Top 20 Risky Permissions of Libraries & Demo Category

Rank	Score	Permissions
1	0.9672	WRITE_SYNC_SETTINGS
2	0.9534	AUTHENTICATE_ACCOUNTS
3	0.9388	MANAGE_ACCOUNTS
4	0.9388	INSTALL_PACKAGES
5	0.8901	READ_SYNC_SETTINGS
6	0.8901	USE_CREDENTIALS
7	0.8050	READ_SYNC_STATS
8	0.8050	CHANGE_WIFI_MULTICAST_STATE
9	0.8050	RECEIVE_BOOT_COMPLETED
10	0.7485	ACCESS_WIFI_STATE
11	0.5517	WAKE_LOCK
12	0.1857	VIBRATE
13	0.1857	ACCESS_NETWORK_STATE
14	0.1857	INTERNET
15	0.1857	WRITE_SETTINGS
16	0.1857	BLUETOOTH
17	0.1857	BLUETOOTH_ADMIN
18	0.1857	CHANGE_WIFI_STATE
19	0.1857	SYSTEM_ALERT_WINDOW
20	0.1857	FLASHLIGHT

Tables depict the results obtained by employing machine learning classifiers without assigning the weighing factors and solely based on permission ranking results. Table 3 depicts the accuracy results for all classifiers for each category. Likewise, Table 4, Table 5, and Table 6 depicts summary of performance metrics TPR, FPR and AUROC for all classifiers and categories.

During analysis, bar graphs were also constructed for every category indicating the frequency of benign and malware apps using those particular risky permissions. Figure 4 represents the occurrence percentage of top 20 permissions of libraries & demo category. The ranking of permissions are done from left to right where left most permission being the most risky one. Length of the solid black bar line depicts the frequency of malware android apps and length of the dash-patterned bar line depicts the frequency of benign ones.

A similar analysis was done on the other categories also, and it was clearly evident from most of the graphs/ categories that the top most risky permissions distinguish mal apps from the benign

Table 3. Compilation of Accuracy for all classifiers for each category (% wise)

Category	Naive Bayes	SVM	Random Forest	J48	Bagging	Boosting	Voting
Art&Design	53.33	78.00	73.33	76.66	78.33	76.66	76.66
Augmented Reality	68.16	60.00	84.00	84.00	82.00	84.00	86.00
Auto&Vehicles	50.84	54.24	77.97	79.66	77.97	76.27	79.66
Beauty	46.67	56.67	56.67	60.00	58.33	61.67	55.00
Books	65.00	56.67	61.67	61.67	63.33	60.00	63.33
Comics	65.91	61.36	65.91	68.18	63.64	63.91	61.36
Communication	80.00	80.00	78.33	73.33	78.33	73.33	75.00
House&Home	98.33	98.33	100.00	100.00	98.33	98.33	98.33
Libraries&Demo	98.33	98.33	98.33	98.33	98.33	96.67	96.67
Lifestyle	75.00	76.67	80.00	76.67	81.67	83.33	78.33
Maps&Navigation	63.33	91.00	83.33	83.33	80.00	86.67	83.33
Personalization	100.00	100.00	100.00	98.33	100.00	100.00	100.00
VideoPlayers&Editors	66.67	60.00	71.67	56.67	65.00	63.33	56.67
Weather	60.00	73.33	81.67	90.00	81.67	83.33	83.33

Table 4. Compilation of TPR for all classifiers for each category

Category	Naive Bayes	SVM	Random Forest	J48	Bagging	Boosting	Voting
Art&Design	0.72	0.77	0.75	0.75	0.77	0.75	0.50
Augmented Reality	0.91	0.91	0.86	0.92	0.90	0.92	0.50
Auto&Vehicles	0.62	0.76	0.75	0.79	0.85	0.85	0.50
Beauty	0.60	0.62	0.58	0.55	0.55	0.60	0.50
Books	0.57	0.65	0.68	0.50	0.63	0.42	0.50
Comics	0.73	0.66	0.64	0.57	0.54	0.54	0.45
Communication	0.82	0.80	0.85	0.85	0.85	0.82	0.50
House&Home	0.98	0.98	1.00	1.00	1.00	1.00	0.50
Libraries&Demo	0.98	0.97	0.98	0.98	0.98	1.00	0.50
Lifestyle	0.77	0.50	0.83	0.85	0.85	0.85	0.50
Maps&Navigation	0.90	0.90	0.90	0.92	0.87	1.00	0.50
Personalization	0.50	0.50	0.50	0.98	0.98	0.85	0.50
VideoPlayers&Editors	0.58	0.67	0.68	0.60	0.67	1.00	0.50
Weather	0.68	0.82	0.83	0.83	0.83	0.82	0.50

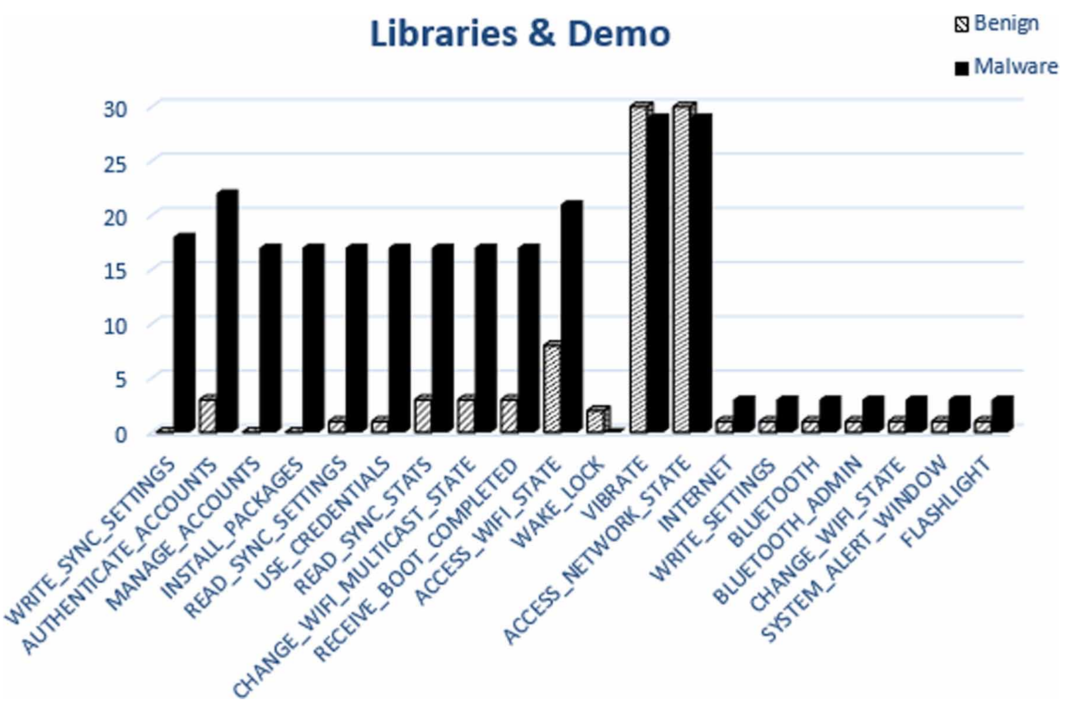
Table 5. Compilation of FPR for all classifiers for each category

Category	Naive Bayes	SVM	Random Forest	J48	Bagging	Boosting	Voting
Art&Design	0.28	0.23	0.25	0.25	0.23	0.25	0.50
Augmented Reality	0.09	0.09	0.14	0.06	0.08	0.06	0.50
Auto&Vehicles	0.37	0.23	0.25	0.20	0.15	0.15	0.50
Beauty	0.40	0.38	0.42	0.45	0.45	0.40	0.50
Books	0.50	0.35	0.32	0.50	0.36	0.58	0.50
Comics	0.27	0.34	0.36	0.43	0.45	0.27	0.55
Communication	0.18	0.20	0.15	0.15	0.15	0.63	0.50
House&Home	0.02	0.02	0.00	0.00	0.00	0.00	0.50
Libraries&Demo	0.02	0.03	0.02	0.02	0.02	0.00	0.50
Lifestyle	0.23	0.50	0.16	0.15	0.15	0.15	0.50
Maps&Navigation	0.10	0.10	0.10	0.08	0.13	0.00	0.50
Personalization	0.50	0.50	0.50	0.02	0.02	0.15	0.50
VideoPlayers&Editors	0.42	0.34	0.32	0.40	0.33	0.00	0.50
Weather	0.32	0.18	0.16	0.17	0.17	0.18	0.50

Table 6. Compilation of AUROC Curve for all classifiers for each category

Category	Naive Bayes	SVM	Random Forest	J48	Bagging	Boosting	Voting
Art&Design	0.71	0.76	0.81	0.73	0.78	0.78	0.50
Augmented Reality	0.89	0.90	0.90	0.87	0.87	0.90	0.43
Auto&Vehicles	0.78	0.76	0.83	0.82	0.83	0.85	0.48
Beauty	0.60	0.67	0.63	0.52	0.62	0.64	0.50
Books	0.65	0.65	0.73	0.57	0.64	0.39	0.50
Comics	0.63	0.65	0.67	0.54	0.54	0.58	0.43
Communication	0.91	0.80	0.93	0.79	0.90	0.88	0.50
House&Home	1.00	0.98	1.00	1.00	1.00	1.00	0.50
Libraries&Demo	0.53	0.79	0.78	0.70	0.78	0.80	0.69
Lifestyle	0.86	0.87	0.93	0.87	0.89	0.87	0.50
Maps&Navigation	0.86	0.90	0.93	0.85	0.86	0.88	0.50
Personalization	1.00	1.00	1.00	0.98	0.96	0.98	0.50
VideoPlayers&Editors	0.69	0.67	0.71	0.64	0.68	0.68	0.50
Weather	0.80	0.81	0.89	0.79	0.85	0.82	0.50

Figure 4. Occurrence percentage of top 20 permissions of Libraries & Demo category



ones in a better manner by the frequency of their appearance as the solid black bar exceeds the dash-patterned one.

To substantiate the above findings, during analysis it was observed that the CHANGE_WIFI_MULTICAST_STATE permission was consistently ranked as the riskiest permission by almost 50% of the categories. The malware apps request this permission to make the apps enter into multicast mode. The other risky permissions considered risky by other categories were INSTALL_PACKAGES, WRITE_SYNC_SETTINGS, READ_SYNC_SETTINGS and SYSTEM_ALERT_WINDOW. The difference in the usage of these permissions was more than 50% in malware apps than the benign ones. It could be easily visualized that the pattern of usage of SYNC_SETTINGS permission was considerably different in benign or in malware apps. The malware apps tend to read and write data from the users' phone via local servers without letting the user know about this hidden action. This data can be in the form of images, videos, important documents, stored settings of the phone etc. A large percentage of permissions requested in the category of video players & editors were benign which means that risk is very less while using apps under this category.

4.3.4 Assign Weighing Factors and Reform Permission Matrix P_{mxn} to Generate P'_{mxn}

In order to improvise the analysis results and performance of the classifiers, a heuristic approach was proposed to redesign the sparse dataset. After listing all the permissions in the descending order of their risk-relevance using Coefficient Correlation method, top 20 permissions were chosen and then, heuristic values were assigned to every permission. The motive was to provide a greater heuristic value to the most risky permissions in each category.

JAVA code written for the sparse approach was improvised to generate a new permission matrix P'_{mXn}, where m denotes the no. of apps (both benign and malware) and n denotes number of android permissions. Earlier, every value of P_{mXn} was a Boolean value i.e. 0 or 1 and now, each value of P'_{mXn}

was in range [0, 0.25, 0.5, 0.75,1] based on permission ranking results and Play Store guidelines for each category. Heuristic values were assigned to the top 20 risky permissions in the following fashion:

- Permission No. 1-5 will be assigned a heuristic value of 1.00
- Permission No. 6-10 will be assigned a heuristic value of 0.75
- Permission No. 11-15 will be assigned a heuristic value of 0.50
- Permission No. 16-20 will be assigned a heuristic value of 0.25
- All the other left 304 permissions will be assigned a heuristic value of 0.00

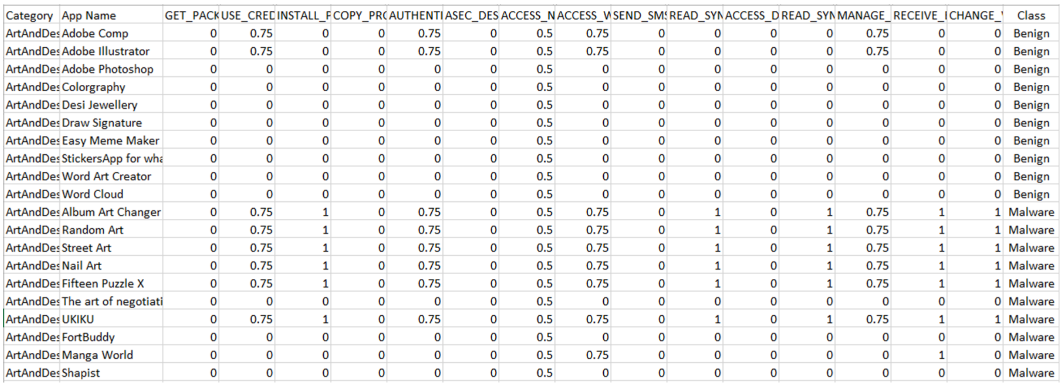
After regenerating the data sets with the newly calculated heuristic values, all the classifiers were applied again on the heuristic data set to check the efficacy of proposed approach.

4.4 Evaluation of Static Analysis approach

The subsection analyses the improvement in the research by pondering over the differences in the static and heuristic results. In this phase, heuristically constructed permission matrix was passed as input to the classifiers. Figure 5 depicts a snapshot of the permission matrix comprising of heuristic values.

The heuristic permission matrices were passed as input to the classifiers and several evaluation metrics were recomputed to check the efficiency of classifiers for app prediction. Following subsections describe the analysis of results obtained.

Figure 5. Snapshot of Heuristic Permission Matrix (P'_{mXn}) of Libraries & Demo category



Category	App Name	GET_PACK	USE_CRED	INSTALL_F	COPY_PR(	AUTHENTI	ASEC_DES	ACCESS_N	ACCESS_V	SEND_SMS	READ_SYN	ACCESS_D	READ_SYN	MANAGE_	RECEIVE_I	CHANGE_	Class
ArtAndDes	Adobe Comp	0	0.75	0	0	0.75	0	0.5	0.75	0	0	0	0	0.75	0	0	Benign
ArtAndDes	Adobe Illustrator	0	0.75	0	0	0.75	0	0.5	0.75	0	0	0	0	0.75	0	0	Benign
ArtAndDes	Adobe Photoshop	0	0	0	0	0	0	0.5	0	0	0	0	0	0	0	0	Benign
ArtAndDes	Colorgraphy	0	0	0	0	0	0	0.5	0	0	0	0	0	0	0	0	Benign
ArtAndDes	Desi Jewellery	0	0	0	0	0	0	0.5	0	0	0	0	0	0	0	0	Benign
ArtAndDes	Draw Signature	0	0	0	0	0	0	0.5	0	0	0	0	0	0	0	0	Benign
ArtAndDes	Easy Meme Maker	0	0	0	0	0	0	0.5	0	0	0	0	0	0	0	0	Benign
ArtAndDes	StickersApp for wha	0	0	0	0	0	0	0.5	0	0	0	0	0	0	0	0	Benign
ArtAndDes	Word Art Creator	0	0	0	0	0	0	0.5	0	0	0	0	0	0	0	0	Benign
ArtAndDes	Word Cloud	0	0	0	0	0	0	0.5	0	0	0	0	0	0	0	0	Benign
ArtAndDes	Album Art Changer	0	0.75	1	0	0.75	0	0.5	0.75	0	1	0	1	0.75	1	1	Malware
ArtAndDes	Random Art	0	0.75	1	0	0.75	0	0.5	0.75	0	1	0	1	0.75	1	1	Malware
ArtAndDes	Street Art	0	0.75	1	0	0.75	0	0.5	0.75	0	1	0	1	0.75	1	1	Malware
ArtAndDes	Nail Art	0	0.75	1	0	0.75	0	0.5	0.75	0	1	0	1	0.75	1	1	Malware
ArtAndDes	Fifteen Puzzle X	0	0.75	1	0	0.75	0	0.5	0.75	0	1	0	1	0.75	1	1	Malware
ArtAndDes	The art of negotiati	0	0	0	0	0	0	0.5	0	0	0	0	0	0	0	0	Malware
ArtAndDes	UKIKU	0	0.75	1	0	0.75	0	0.5	0.75	0	1	0	1	0.75	1	1	Malware
ArtAndDes	FortBuddy	0	0	0	0	0	0	0.5	0	0	0	0	0	0	0	0	Malware
ArtAndDes	Manga World	0	0	0	0	0	0	0.5	0.75	0	0	0	0	0	1	0	Malware
ArtAndDes	Shapist	0	0	0	0	0	0	0.5	0	0	0	0	0	0	0	0	Malware

4.4.1 Results of Analysis Without Weighing Factors vs with Weighing Factors

The empirical results identified that about 60% of the categories showed a remarkable increase in the area under the ROC curve precision metric, when heuristic dataset was used to classify the android apps.

A comparison of results of both the data sets showing AUROC achieved on applying various classifiers on the Libraries & Demo category has been collected in the Table 7. It can be inferred that SVM with the Heuristic Dataset gives the best case result for this category. Similar inferences were drawn for other categories explained in the next subsection.

Table 7. Comparison of AUROC of Libraries & Demo category

S. No.	Category	Sparse Matrix	Heuristic Matrix
1	Naive Bayes	0.53	0.80
2	**SVM**	0.79	**0.98**
3	Random Forest	0.78	0.83
4	Decision Tree	0.70	0.82
5	Bagging	0.78	0.83
6	Boosting	0.80	0.82
7	Voting	0.69	0.83

4.4.2 Best Combination of Dataset and Classifier

After analyzing the results of the classifiers, finally, the best combination of classifier and permission matrix for each category was chosen which gives the largest area under the ROC curve. The combination chosen have been compiled in the Table 8.

The combinations were fed in our proposed interface and were used to classify Android apps based on its category and permission ranking results.

Table 8. Classifier and Permission Matrix to be used while classifying a particular category

S. No.	Category	Classifier	Perm Matrix
1	Art&Design	Random Forest	Heuristic
2	Augmented Reality	SVM	Sparse
3	Auto&Vehicles	Bagging	Sparse
4	Beauty	SVM	Heuristic
5	Books	Random Forest	Sparse
6	Comics	Random Forest	Sparse
7	Communication	Bagging	Heuristic
8	House&Home	Random Forest	Heuristic
9	Libraries&Demo	SVM	Heuristic
10	Lifestyle	Boosting	Heuristic
11	Maps&Navigation	Random Forest	Sparse
12	Personalization	Random Forest	Heuristic
13	VideoPlayers&Editors	SVM	Heuristic
14	Weather	Random Forest	Sparse

5. INTERFACE

Completing all the above phases finally brings us to the main part of our research work. In this section, how the interface can be used in malware identification and detection has been explained in detail. The prime motive of the study was to be able to predict and inform the user whether he/ she should install the app or not without even running the app. We have successfully implemented the interface using the two most popular platform independent languages among developers i.e. Java and Python which makes our interface compatible on all Operating Systems. Android is an open source operating system. A majority of Android apps are developed in Java till date. Therefore, use of platform independent tools makes it easier to analyze malicious Android apps and hence protect user's privacy with the help of static analysis. Moreover, Java and Python are strongly typed and their verbose coding style makes them faster than other languages. Here, Java is used to design the interface (via swing and awt) and extract the new app's permission vector and Python is used to create the prediction model.

5.1 Extraction of Permissions Vector via Reverse Engineering

The interface launches a window asking the user to provide an app and its category for testing. The starting task is to reverse engineer the new unknown app and study the AndroidManifest.xml file. Batch processing has been used to run the 'apktool' on the app's apk. All the user permissions are extracted in the form of a Boolean vector comprising of 1 if the permission is being requested by the app and 0 otherwise. After getting all the information required for testing the app, category and the extracted permissions vector will be injected in a python function which will test the app on the trained dataset.

5.2 Prediction Using Scikit Learn K-Fold Cross Validation

After selecting the area under the ROC Curve as the best evaluation metric and accumulating both the types of data sets, scikit learn python library is used to apply StratifiedKFold cross validation technique to create the prediction model of the interface.

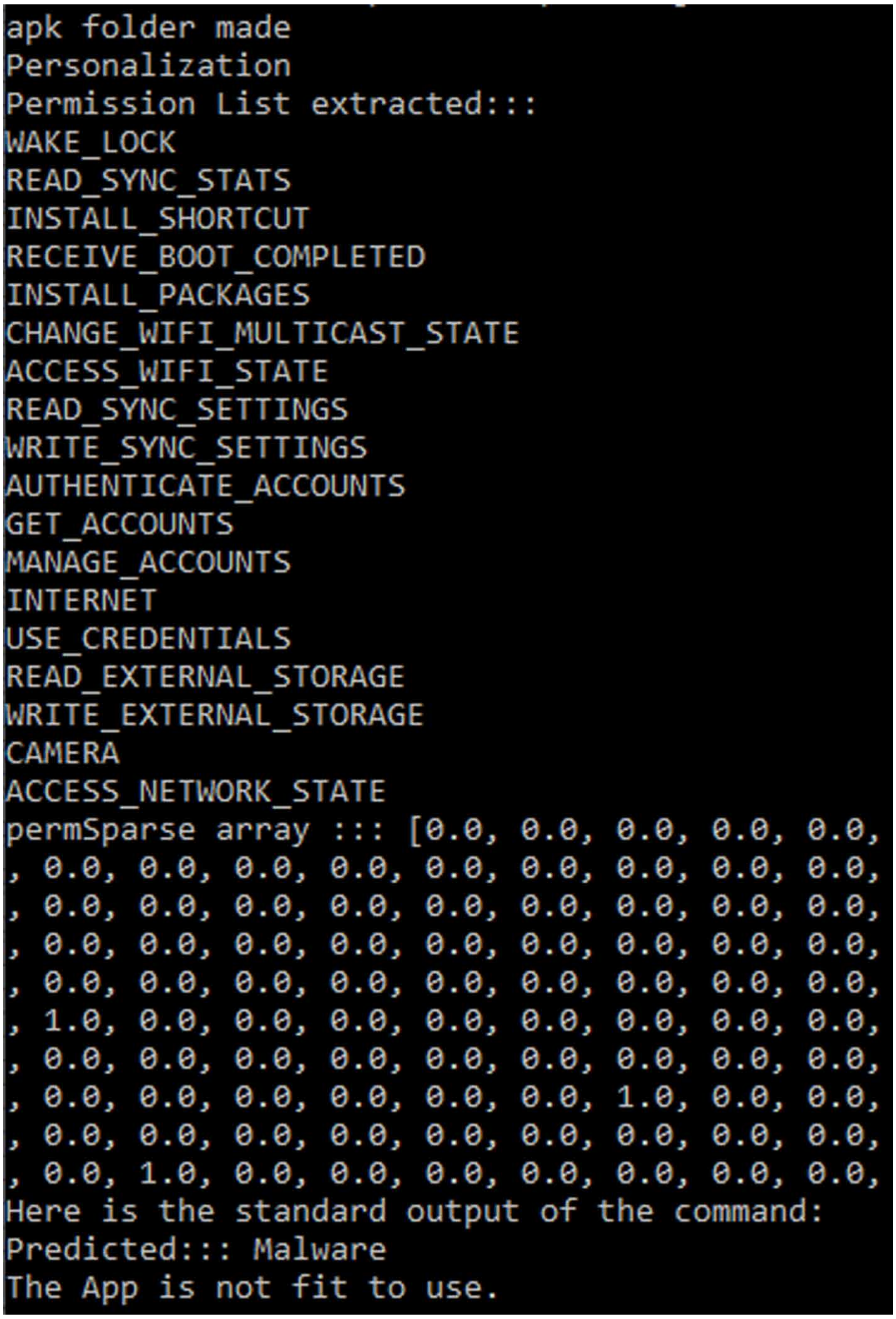
StratifiedKFold: The K fold cross validation technique involves splitting the complete dataset into k folds, with k-1 groups acting as the training sample and the kth i.e. the holdout set as the testing one. The model is fit and evaluated for every fold and the final result is given as the average of all the runs. The problem with K Fold cross validation is that it does not provide satisfactory results for imbalanced classes since it splits the data with the uniform probability. For datasets where there is an extreme abundance of one of the classes, it is possible for one or more splits to not have any values from the other minor population. This implies that the model is being trained for the majority class most of the time and that is what it becomes trained to predict when testing a new data point on it. The stratified K fold, though, again, splits the dataset randomly but in a way that maintains the same ratio of classes in every split as was in the original complete dataset (Brownlee, n.d.).

On the basis of the category specified, the classifier is used to test the malware behaviour of the unknown app on the trained dataset. At last, a prediction is made as whether the app is benign or malicious and the output will be printed on the interface. It will also tell the user that if the app is benign, then he/ she might install and use it because it is risk free but if the app is predicted malware, then it will warn the user not to install the app.

Figure 6 shows a glimpse of all the operations performed in the background of the MalApp Classification Interface which takes an unknown app's apk i.e. Vodel animations of the Personalization category and tests the behavior of the app step by step as described in the paper.

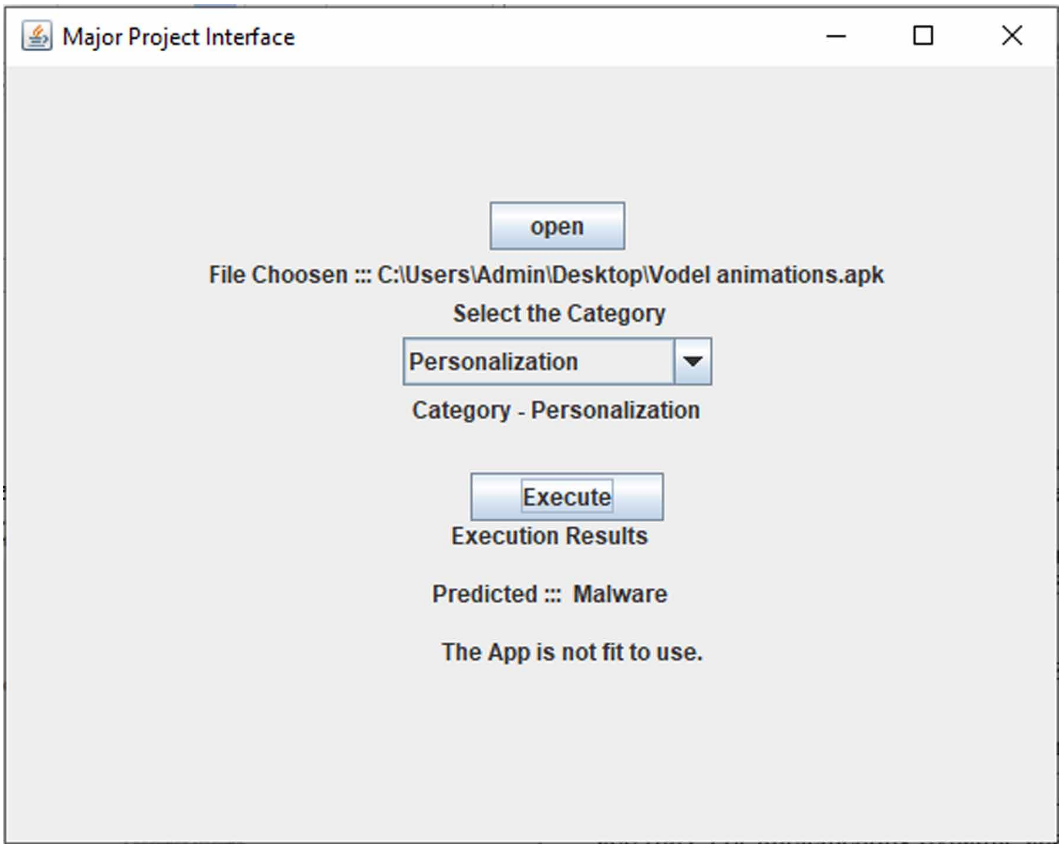
Figure 7 presents the final snapshot of the Interface predicting that the new app is malicious and therefore, it is not fit to use. Thus, this interface can be used to warn the users from using any malicious app beforehand.

Figure 6. Operations performed in the Interface



```
apk folder made
Personalization
Permission List extracted:::
WAKE_LOCK
READ_SYNC_STATS
INSTALL_SHORTCUT
RECEIVE_BOOT_COMPLETED
INSTALL_PACKAGES
CHANGE_WIFI_MULTICAST_STATE
ACCESS_WIFI_STATE
READ_SYNC_SETTINGS
WRITE_SYNC_SETTINGS
AUTHENTICATE_ACCOUNTS
GET_ACCOUNTS
MANAGE_ACCOUNTS
INTERNET
USE_CREDENTIALS
READ_EXTERNAL_STORAGE
WRITE_EXTERNAL_STORAGE
CAMERA
ACCESS_NETWORK_STATE
permSparse array ::: [0.0, 0.0, 0.0, 0.0, 0.0,
, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0,
, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
Here is the standard output of the command:
Predicted::: Malware
The App is not fit to use.
```

Figure 7. Snapshot of MalApp Classification Interface


Major Project Interface

File Choosen ::: C:\Users\Admin\Desktop\Vodel animations.apk

Select the Category

Personalization

Category - Personalization

Execution Results

Predicted ::: Malware

The App is not fit to use.

6. CONCLUSION

In this paper, we have presented a permission induced risk interface, a novel tool for MalApp Classification. In this work, reverse engineering approach in form of static analysis has been employed for mal app classification based on category. To identify the risky permissions across category, permission ranking using correlation coefficient has been employed.

Several machine learning classifiers and ensembling techniques were put to use to check the efficiency of the proposed approach. Heuristic approach was also employed to improve the classification results and develop the malware classification interface. The bounding limits towards the capability of the proposed interface to identify an unknown malicious app are identified with best case detection rate of 98.33% for the Libraries & Demo category classified by SVM on the heuristic dataset and worst case detection rate of 73.33% for the Books category classified by Random Forest on the sparse dataset. Also, the results show an overall average accuracy of 88.2%.

7. FUTURE WORK

The future work can be done by enhancing the performance of the classifiers by improving the heuristic values. Dynamic analysis can be used to analyse the risky behaviour of those permissions which are requested after the app runs. For implementing dynamic analysis, as soon as the app starts to run, an interface can be built which will act as a middle man to stop the app for some moments and meanwhile it will capture its traffic using a proxy server such as Charles Proxy, Fiddler etc. The runtime permissions asked by the app can be tracked. An analysis can also be done on the form of data which is sent on the server - whether the personal details like password, phone no., messages, photos, etc are sent in encrypted form or decrypted form.

After combining the static and dynamic analysis, the interface can prompt the user that the app is safe to run or not.

REFERENCES

Aptoide. (n.d.). *Aptoide Download, find and share the best apps and games for Android*. Retrieved August 6, 2020, from https://en.aptoide.com/

Arnatovich, Y. L., Wang, L., Ngo, N. M., & Soh, C. (2018). A comparison of android reverse engineering tools via program behaviors validation based on intermediate languages transformation. *IEEE Access : Practical Innovations, Open Solutions*, *6*, 12382–12394. doi:10.1109/ACCESS.2018.2808340

Bhatt, A. J., Gupta, C., & Mittal, S. (2018). iABC: Towards a hybrid framework for analyzing and classifying behaviour of iOS applications using static and dynamic analysis. *Journal of Information Security and Applications*, *41*, 144–158. doi:10.1016/j.jisa.2018.07.005

Blair, I. (n.d.). *Mobile App Download and Usage Statistics (2020) - BuildFire*. Retrieved August 6, 2020, from https://buildfire.com/app-statistics/

Bose, S. (2018). a Comparative Study: Java Vs Kotlin Programming in Android Application Development. *International Journal of Advanced Research in Computer Science*, *9*(3), 41–45. doi:10.26483/ijarcs.v9i3.5978

Brownlee, J. (n.d.). *A Gentle Introduction to k-fold Cross-Validation*. Retrieved August 6, 2020, from https://machinelearningmastery.com/k-fold-cross-validation/

Chawla, T. K., & Kajala, A. (2014). International Journal of Computer Science and Mobile Computing Transfiguring of an Android App Using Reverse Engineering. *International Journal of Computer Science and Mobile Computing*, *3*(4), 1204–1208. www.ijcsmc.com

Chikofsky, E. J., & Cross, J. H. (1990). Reverse Engineering and Design Recovery: A Taxonomy. *IEEE Software*, *7*(1), 13–17. doi:10.1109/52.43044

De Ferrari, L., & Aitken, S. (2006). Mining housekeeping genes with a Naive Bayes classifier. *BMC Genomics*, *7*(1), 1–14. doi:10.1186/1471-2164-7-277 PMID:17074078

Fereidooni, H., Conti, M., Yao, D., & Sperduti, A. (2016). ANASTASIA: ANdroid mAlware detection using STatic analySIs of applications. *2016 8th IFIP International Conference on New Technologies, Mobility and Security, NTMS 2016*. doi:10.1109/NTMS.2016.7792435

Gislason, P. O., Benediktsson, J. A., & Sveinsson, J. R. (2006). Random forests for land cover classification. *Pattern Recognition Letters*, *27*(4), 294–300. doi:10.1016/j.patrec.2005.08.011

Google. (n.d.). *Permissions - Play Console Help*. Retrieved August 6, 2020, from https://support.google.com/googleplay/android-developer/answer/9888170?hl=en&ref_topic=9877467

Hoffman, C. (n.d.). *Android Is "Open" and iOS Is "Closed" — But What Does That Mean to You?* Retrieved August 6, 2020, from https://www.howtogeek.com/217593/android-is-open-and-ios-is-closed-but-what-does-that-mean-to-you/

Idrees, F., Rajarajan, M., Chen, T. M., Rahulamathavan, Y., & Naureen, A. (2017). AndroPIn: Correlating Android permissions and intents for malware detection. *2017 8th IEEE Annual Information Technology, Electronics and Mobile Communication Conference, (IEMCON) 2017*, 394–399. doi:10.1109/IEMCON.2017.8117152

Kang, H., Jang, J., Mohaisen, A., & Kim, H. K. (2015). Detecting and Classifying Android Malware Using Static Analysis along with Creator Information. *International Journal of Distributed Sensor Networks*, *11*(6), 479174. doi:10.1155/2015/479174

Korada, N. K., Kumar, N. S. P., & Deekshitulu, Y. V. N. H. (2012). Implementation of Naive Bayesian Classifier and Ada-Boost Algorithm Using Maize. *Expert Systems: International Journal of Knowledge Engineering and Neural Networks*, *2*(3), 63–75.

Ledisi, G., & Ugochukwu, C. (2019). *Comparison of Bagging and Voting Ensemble Machine Learning Algorithm as a Classifier*. Academic Press.

M, H., & M.N, S. (2015). A Review on Evaluation Metrics for Data Classification Evaluations. *International Journal of Data Mining & Knowledge Management Process, 5*(2), 1–11. 10.5121/ijdkp.2015.5201

Market.us. (n.d.). *Mobile App Download and Usage Statistics and Facts - Market.us*. Retrieved August 6, 2020, from https://market.us/statistics/internet/mobile-app-download-and-usage/

MobiLoud. (n.d.). *The Practical Guide to App Store Optimization (ASO)*. Retrieved August 6, 2020, from https://www.mobiloud.com/blog/app-store-optimization/#6

Myat, S. M. (2019). *Analysis of Android Applications by Using Reverse Engineering Techniques*. Academic Press.

Neptune.ai. (n.d.). *F1 Score vs ROC AUC vs Accuracy vs PR AUC: Which Evaluation Metric Should You Choose?* Retrieved August 6, 2020, from https://neptune.ai/blog/f1-score-accuracy-roc-auc-pr-auc

Otukei, J. R., & Blaschke, T. (2010). Land cover change assessment using decision trees, support vector machines and maximum likelihood classification algorithms. *International Journal of Applied Earth Observation and Geoinformation*, *12*(SUPPL. 1), 27–31. doi:10.1016/j.jag.2009.11.002

Paul, N. (n.d.). *A list of all Android permissions- GitHub*. Retrieved August 6, 2020, from https://gist.github.com/nishthapaul/7933e4acbd46163a669ba7b36254d3a6

Play, G. (n.d.). *Android Apps on Google Play*. Retrieved August 6, 2020, from https://play.google.com/store/apps/category/APPLICATION?hl=en_IN

Rashidi, B., Fung, C., & Bertino, E. (2017). Android Malicious Application Detection Using Support Vector Machine and Active Learning. *13th International Conference on Network and Service Management (CNSM)*, 1–9. doi:10.23919/CNSM.2017.8256035

Rasthofer, S., Arzt, S., Miltenberger, M., & Bodden, E. (2016). Reverse engineering Android apps with CodeInspect. *CEUR Workshop Proceedings*, *1575*, 1–8.

Sahal, A. A., Alam, S., & Sogukpinar, I. (2018). Mining and Detection of Android Malware Based on Permissions. *3rd International Conference on Computer Science and Engineering (UBMK 2018)*, 264–268. doi:10.1109/UBMK.2018.8566510

Singh, K. P., Basant, N., & Gupta, S. (2011). Support vector machines in water quality management. *Analytica Chimica Acta*, *703*(2), 152–162. doi:10.1016/j.aca.2011.07.027 PMID:21889629

Sun, L., Srisa-an, W., Ye, H., Li, Z., Li, J., & Yan, Q. (2018). Significant Permission Identification for Machine-Learning-Based Android Malware Detection. *IEEE Transactions on Industrial Informatics*, *14*(7), 3216–3225. doi:10.1109/TII.2017.2789219

Tao, G., Zheng, Z., Guo, Z., & Lyu, M. R. (2018). MalPat: Mining Patterns of Malicious and Benign Android Apps via Permission-Related APIs. *IEEE Transactions on Reliability*, *67*(1), 355–369. doi:10.1109/TR.2017.2778147

TermsFeed. (n.d.). *Android Collection of Data and Sensitive Data - TermsFeed*. Retrieved August 6, 2020, from https://www.termsfeed.com/blog/android-sensitive-data-collection/

Tharwat, A. (2018). *Applied Computing and Informatics Classification assessment methods*. Applied Computing and Informatics. doi:10.1016/j.aci.2018.08.003

vShare. (n.d.). *vShare App (iOS Android and PC)*. Retrieved August 6, 2020, from https://vsharedownload.org/

Wang, W., Wang, X., Feng, D., Liu, J., Han, Z., & Zhang, X. (2014). Exploring permission-induced risk in android applications for malicious application detection. *IEEE Transactions on Information Forensics and Security*, *9*(11), 1869–1882. doi:10.1109/TIFS.2014.2353996

Wikipedia. (n.d.). *Static analysis*. Retrieved August 6, 2020, from https://en.wikipedia.org/wiki/Static_analysis

Winder, D. (n.d.). *28 Million Android Phones Exposed To "Eye-Opening" Attack Risk*. Retrieved August 6, 2020, from https://www.forbes.com/sites/daveywinder/2019/08/03/28-million-android-phones-exposed-to-eye-opening-attack-risk/#350a5cca7b74

Yu, L., Pan, Z., Liu, J., & Shen, Y. (2013). Android malware detection technology based on improved Bayesian classification. *Third International Conference on Instrumentation and Measurement, Computer, Communication and Control, IMCCC 2013*, 1338–1341. doi:10.1109/IMCCC.2013.297

Zareapoor, M., & Shamsolmoali, P. (2015). Application of credit card fraud detection: Based on bagging ensemble classifier. *Procedia Computer Science*, *48*(C), 679–685. doi:10.1016/j.procs.2015.04.201

Nishtha Paul received her BTech degree in Computer Science & Engineering from Jaypee Institute of Information Technology, Noida in 2020. Currently, she is working as an Application Developer in ThoughtWorks Inc. She has 2 years of experience as a Java software engineer and has won $1000 in MongoDB World Hackathon'19 under the category of Best Use of MongoDB Mobile. Her keen interests include 3D modelling and graphic design. A strong believer in the power of positive thinking in the workplace, Nishtha regularly experiments with new technologies and techniques. She has a strong eye for detail and tenacity to never quit on something until it is absolutely perfect.

Arpita Jadhav Bhatt is Assistant Professor (Senior Grade) in the Department of Computer Science & IT from Jaypee Institute of Information and Technology, Noida, India. She obtained her Ph.D. degree in Computer Science from Jaypee Institute of Information and Technology, Noida. She has more than 8 years of teaching experience. She obtained her Masters in Engineering in Software Systems from Birla Institute of Technology and Science, Pilani (BITS Pilani) in 2010 and Bachelor of Technology degree from Rishiraj Institute of Technology, Indore in 2008. Her areas of interest are mobile application engineering, information privacy, software engineering, programming in Android, mobile computing. She has many publications in international journals and conferences to her credit.

Sakeena Rizvi received her Bachelors in Technology from Jaypee Institute Of Information Technology, sector 62, Noida in the year 2020. In the same year, she was hired as programmer analyst trainee at Cognizant Technology Solutions. To her credits, she has various projects on the topics of Android and Machine learning. She has also researched on nature inspired algorithms and blockchain technology and has written on the same.

Shubhangi received her B.Tech degree in Computer science from Jaypee Institute of Information Technology in 2020. Currently, she is working as an Associate application developer in IBM.