A Project Report for IT 794 (Minor Project) of Autumn Semester 2021-2022

On

Android Malware Detection

Submitted by

Anshika Agrawal(18118008)

anshika04agrawal@gmail.com

Bandaru Uma Maheswari(18118015)

maheswaribandaru28@gmail.com

Kolisetty Pavan Kalyan(18118033)

pkalyan264@gmail.com

Shraddha Patel(18118074)

shraddhacute1230@gmail.com

Supervised by

Dr. Sanjay Kumar

Department of Information Technology

National Institute of Technology (NIT) Raipur

December 1, 2021

Abstract

The Android smartphone, with its wide range of uses and excellent performance, has attracted numerous users. Still, this domination of the Android platform also has motivated the attackers to develop malware. The traditional methodology which detects the malware based on the signature is unfit to discover unknown applications. In this paper, we tried to detect whether an application is malware or not using Static Analysis. We considered all the permissions that an application asks for and took them as input to feed our machine learning models. We built different classifiers using different machine learning algorithms such as Support Vector Machine(Linear and RBF), Logistic Regression,Random Forest Algorithm, Gaussian Naive-Bayes, Decision Tree Method etc., and we compared their performances.

Keywords:Static Analysis; Dynamic Analysis; Feature Selection; Decision tree Classifier.

Contents

1 Introduction

1.1 Objective

To implement a machine learning model that can detect the presence of any malware in an android application accurately.

1.2 Motivation

In June 2021, Android retained its status as the world's most popular mobile operating system, with a market share of about 73 percent. As Android has grown in popularity, so has the amount of malware targeting the platform. Android has been the most popular target for malware makers since 2010, accounting for more than 90% of all mobile malware.

Android applications can be downloaded from a lot of sources including third-party app stores. These application stores serve as a point of entry for malware to spread quickly. The most common strategy used by malware creators is Repackaging, which involves embedding the harmful code segment within a legal application making the application malicious, and distributing it through third-party stores. Their malware programs are designed to get root access, stealing personal information, setting up mobile botnets and many more. So far, there are a large number of malware discovered, some of which are more dangerous than others. They are harmful because they have the potential to steal user's personal information, as well as sending this data to remote servers.

So, there is a need to develop models that can detect the presence of Android malware with accuracy and that is our motivation.

1.3 Problem Statement

Smartphones have become the most used device in one's day to day life. They facilitate users with a variety of applications that are enriched with powerful features. It is almost impossible for anyone these days to spend a day without their smartphones.

Out of all smartphones, Android smartphones are the ones that are widely used. This increasing popularity of Android smartphones has also attracted malicious attackers. This malicious activity can be done by either a single application or a group of applications working together. The objective of this project is to create a model that can detect such malicious applications.

2 Literature Survey

We studied the techniques that are proposed to identify Android malwares. In his work, Anshul et al. [1] presented an idea to detect Android Malwares by Network traffic analysis. Their approach is used to identify malware on Android that is operated by a remote server. These malwares either accept orders from the server or leak sensitive data to it. First, they analyzed the network traffic of android malwares and then the traffic of normal applications. They discovered the characteristics that distinguish malware traffic from non-malware traffic.. And in the second phase, they built a classifier using these network traffic features which can detect the malwares.

In another work, Anshul et al. [2] proposed a technique called the PermPair method. They approached the goal by considering every pair of permissions as the possible input feature and finally decided on each pair, if that combination is vulnerable. Their method includes data sets from 3 different sources called Genome ,Debris and Koodoos. Their approach had 3 phases. In the first phase, they constructed 4 different graphs by extracting permission pairs from each application. Out of the 4 graphs, 3 graphs are for malwares and 1 graph is for benign applications. In the second phase, they dealt with merging 3 malicious graphs into a single malicious graph. At the end of this phase, they ended up with two graphs, one for malicious and one for benign. In the third and final phase of their method, they developed a model that calculates two scores called normal score and malicious score for every application and decides whether a particular application is malware or not.

The most commonly used properties in static and dynamic Android malware detection are permissions and network traffic features respectively. Static permissions cannot identify sophisticated malware, which is capable of update attacks. And coming to dynamic network traffic, it cannot detect malware samples without a network connection. Therefore, a hybrid model integrating both of these properties is proposed. They extracted both permissions and network traffic features and made them into a single vector. Using the K-medoids method, they partitioned the vectors into K clusters. And they used the K-Nearest Neighbours method, to classify whether a particular application is malicious or not. They made sure that K is odd, just to make sure out of K nearest neighbours, the count of malicious and benign neighbours is not the same.

In another work, Zhenlong Yuan et al. [3] proposed a technique to associate static features with dynamic features and then classify the given android applications as malicious or safe. They got the features they used as input to their model in three stages:

- Static Phase
- Sensitive APIs
- Dynamic Phase

Static phase includes the permissions that are obtained by unzipping the apk file and parsing xml files obtained later. Another file classes.dex accounts for the sensitive api calls . To obtain the dynamic features, they used to install the apk files on droidbox , which is an extended sandbox of Taintdroid, which is itself capable of recording the network traffic,information leak and other run time analytics that could play a major role in determining an app as malware or not. They implemented an online android detection engine based on deep learning models.They are able to develop a deep learning model that outperforms many state of art machine learning techniques.

In another work, Suleiman et al. [4] examined the use of varied machine learning techniques in a parallel classification approach to Android malware detection .The proposed method made use of a variety of properties,API calls, instructions, and other API-related topics were provided.The recent rise of Android usage Malware and its ever-increasing ability to be detected ,current signature-based techniques are avoided.The classification approach adapted by them is a realistic option.The method that not only gives a supplemental tool but also it has the potential to improve Android malware detection, but it also has some drawbacks.It only permits different classifiers' strengths to be combined. Assisting with further phases of analysis Moreover the planned from a performance standpoint, the technique is great since it is economical when it comes to classifying an unknown instance because the app's static features are consumed and also chosen features are consumed. Models for constituent categorization have a minimal computational cost before making a categorization judgement.

Xingquan Zhu et al. [5] proposed a feature based learning method that takes input as the permissions and api calls an app is intended to make during its life time. The proposed frame work consists of 4 different phases. They are:

- App Analyser (decompresses) the apk file
- Fetches permissions and api calls
- Feature generator (Generate binary datasets)
- Data mining models(SVM, Bagging, Decision Tree)

Though this approach doesn't involve any dynamic analysis , this approach of complimenting the permissions with the api calls improves the performance of the model and helps in the better performance of the model.

To obtain input parameters from bundled Android apps, Justin et al. [6] used an apk called Androguard. This androguard is used to obtain permission features from the jar files included in the android apk file. They then utilised the Scikit learn framework, a python library. It gives a straightforward interface to LIBSVM , to train a SVM having 1 class, utilising these obtained features. They aimed to reduce the high false positive rates generally seen in these android malware detection techniques.They are only reported after testing data as malware if it differs sufficiently from the training data, which is ideal for their purposes because the range of benign applications are more widely available than malicious applications.For the Android operating system, they introduced a revolutionary malware detection solution which works on the principle of ML.

In Zi Wang et al. [7], they offered DroidDeepLearner, an Android malware characterization and identification strategy that uses a deep learning algorithm to distinguish malicious Android apps from benign ones.They carried out a series of tests to ensure that the suggested technique was both legitimate and effective,and compared the performance of the proposed DroidDeepLearner with that of SVM, which is one of the most effective and commonly used algorithms for malware detection.They ran a variety of experiments using real Android app datasets to validate the malware detection technique, and the findings suggest that the scheme can accurately identify malware in the Android.When compared to traditional solutions which works on SVM as backbone, experiment results suggest that the proposed system is capable of accurately identifying malware.

In Hyo-Sik Ham et al. [8], a study is undertaken on the detection of malware using an Android device, which is the most commonly used as an attack point by attackers. This paper presents a feature selection and experimentation technique for lowering the number of malware detections that are false positives and improving device performance degradation that has been identified as a problem in the current system Machine learning-based mobile malware detection.This model also analyses malware detection methods and performance of machine learning classifiers.First, they must compare the agent's resource consumption to that of an existing agent that is monitoring all features. In the future, more improved variations of Android malware could be collected to investigate their attributes. Also, a research could be conducted on identifying new malwares that doesn't exist in the known directory.

As there are a many number of permissions an application requests for during the installation , It is important to pick out the most prominent permissions among those that affect the malicious activity of the application. Feature selection is the phase that facilitates the developers with this task. The dataset generated after this feature selection accounts for the more accurate results. In this paper, Pehlivan et al. [9] used a tool called Waikato Environment for Knowledge Analysis (WEKA) to achieve feature selection. The metrics that are employed in evaluation of effective feature selection are TruePositiveRate, FalsePositiveRate and Precision. They also took accuracy as one of their metrics. A feature is employed in the classification dataset on the basis of Gain Attribute Evaluation attribute. The higher is this attribute, the more impact the feature keeps on the output. They considered this gain attribute to decide whether a feature is to be included in the training data set or not.

In another work, Fereidooni et al. [10] used a tool named unitPdroid written in python 3 to extract the prominent features i.e.,a technique used in feature engineering . The permissions have been extracted from the class.xml file unzipped from the apk file of the android application. The permissions are then processed and modified into binary features so that it could be easier to train our classification models. This approach also incorporates the suspicious API calls as a feature that can affect the malicious activity of the application. This paper employed an approach called dalvik bytecode analysis to determine if an application is malware. This analysis is based on the kind of information that the application is forwarding to the remote servers. If the application forwards the sensitive information or anything serious, it's rated more probably as the malware.They are able to achieve results more efficiently than many conventional state of art techniques.

3 Methodology

3.1 Dataset

Any machine learning model needs a dataset over which it can be trained. So data collection is one of the most important steps. We've worked on 3 different datasets and compared their result against each other

- 1st dataset is collected from google comprised of 70 different application each having a set of 17 permission
- 2nd is downloaded from Kaggle which has 184 different permissions or we can say features list for 29999 apps individually.
- 3rd one is downloaded from Kaggle. It has 138047 records with each record consisting 57 columns(permissions)

For training and testing purposes, we split the dataset into two parts. We used 80% of the dataset for training the machine learning model and the remaining 20% dataset was used for testing every machine learning model and calculating the performance of each model with metrics such as accuracy, f1-score, precision and recall.

3.2 Feature Engineering

The feature set used for training has a big impact on machine learning. Several research have found that certain features are helpful in training machine learning-based malware classifiers. That is the reason we have used feature engineering in our implementation. In supervised learning, we will use Feature engineering, which is the process of selecting, manipulating, and changing raw data into features.

We use Feature Engineering for the following reasons:

- To remove imputation
- Handling outliers
- To achieve Normalization
- Null Value Handling
- To remove invalid data
- To achieve scaling

The task of feature engineering is divided into to 2 parts:

3.2.1 Data Preprocessing

The process of converting raw data into a comprehensible format is known as data preparation. We can't work with raw data, thus this is a key stage in machine learning. Before using machine learning or data mining methods, make sure the data is of good quality.The purpose of data preprocessing is to ensure that the data is of good quality. The following criteria can be used to assess quality accuracy, completeness, consistency, trustworthy, understandablity. Data Preporocessing involves the following steps:

- Data Cleaning: Correcting or deleting incorrect, corrupted, improperly formatted, duplicate, or incomplete data.We have removed those columns having missing values. We've removed any undesirable observations from our datasets, such as duplicates or irrelevant observations
- Data Transformation: Changing data from one format to another. For string columns and decimal columns such as price, they're converted to binary.
- Data integration: combining data from a variety of sources, including databases (both relational and non-relational), data cubes, files, and so on.
- Data reduction: It is possible to reduce the amount of records, characteristics, or dimensions. It is carried out during feature selection using correlation matrix.

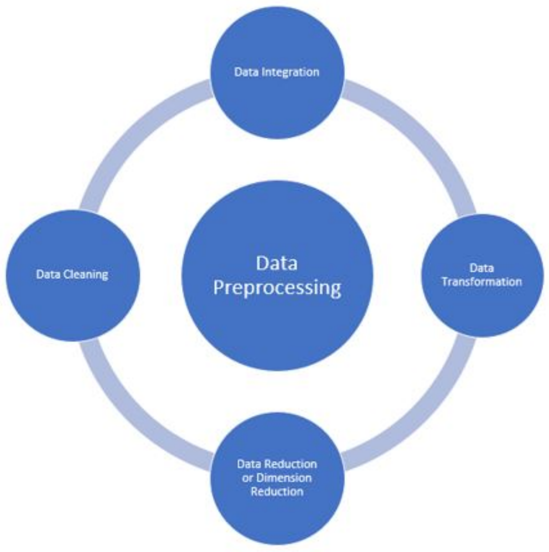


Figure 1: Steps involved in Data Preprocessing

3.2.2 Feature Selection

Feature Selection is a dimensionality reduction procedure that reduces a large set of raw data into smaller groups for processing. Feature selection aims to maximise relevance while reducing repetition. To accomplish efficient data reduction, feature selection approaches can be utilised in data pre-processing. This is accomplished by Correlation Coefficient Matrix.

We need to establish an absolute value as the variable selection threshold. If the predictor variables are found to be associated, we can exclude the variable having the lowest correlation coefficient value with the target variable.

After the process of feature selection, we are left with 54 columns which we are going to consider as features that we use for training the machine learning models.

We computed the correlation coefficients for each feature pair. Correlation coefficient between two variables tells us how those two variables are dependent on each other. Heatmap is a pictorial representation of the correlation matrix. In figure-1, we can see the heatmap of 54 features.

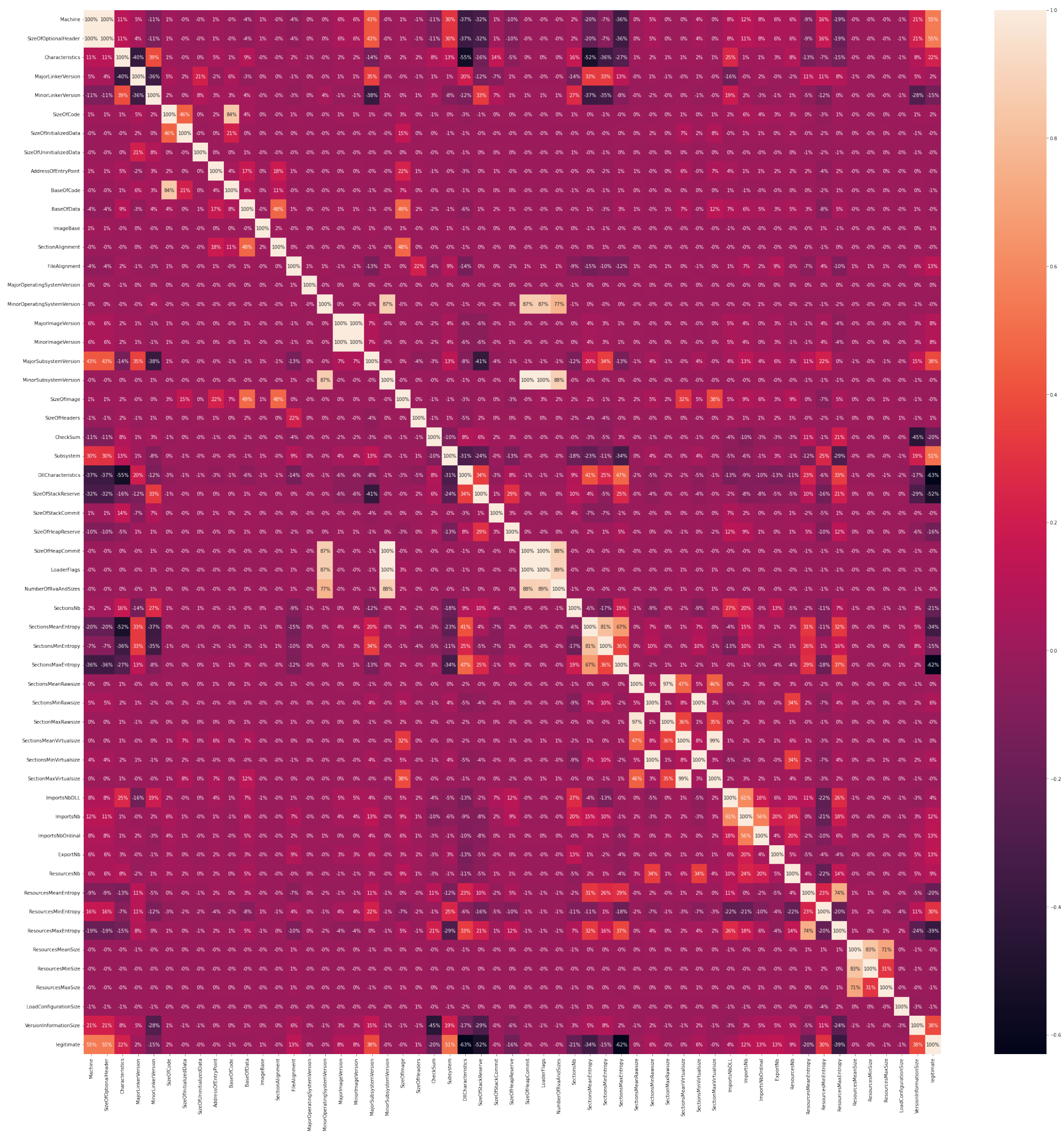


Figure 2: Heatmap showing the correlation between feature pairs

4 Experimentation and Discussion

We have worked on last dataset in detail and for remaining we have tested accuracy and compared its result. We made sure that the dataset contained enough examples for both the malware and benign applications. There are 41323 examples for benign applications and 96724 applications for malicious applications. So, we can say that the dataset is not skewed. Each permission column is a binary column,indicating a permission is asked or not.

4.1 Designing algorithmic Model

For Implementing our project we have used 7 different predefined models present in various libraries so that we can compare their working and result. Apart from this we have also designed our Recurrent Neural Network model with base as sequential model having LSTM layer and various dense layers. After Implementation we have compared the results of all models in the result section.

4.1.1 Logistic Regression

Logistic Regression is a predictive analytic technique built on the notion of probabilities. It is a Machine Learning algorithm that would be used for problems which involves classifying. Here, we will have a hypothesis function which will tell us the probability that a sample belongs to a particular class. In logistic regression, we use sigmoid function as hypothesis function.

$$h_\theta(x) = \sigma(x^T\theta)$$
$$\sigma(x^T\theta) = \frac{1}{1 + e^{-x^T\theta}}$$

Ideally we will want the logistic regression classifier to give an accurate prediction. For that, we have to find θ that minimizes the cost function J(θ). We do that using gradient descent algorithm

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[(1 - y^i)log(1 - p^i) + y^i log(p^i)]$$

where p^i is the probability that i is a malicious application and y^i is the labelled class for that application.

4.1.2 K-Nearest Neighbours

K-Nearest Neighbors is a straightforward machine learning approach that may be applied to regression problems as well as tasks involving classifying. In KNN, we look for K applications that are nearest to the given instance application. We note down the classes of all those K applications and count how many times each class appeared. The class that appeared the most will be the class of our instance application. Here the K nearest neighbours are chosen using the euclidean distance.

Here we have to choose the K value that gives the least training error rate. There isn't any particular method to do that but when we plot error rate and k on a 2D-plane, we can take the K value that gives us least error.

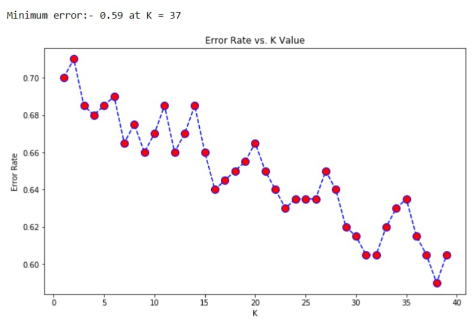


Figure 3: Plot between error rate and K

4.1.3 SVM Linear

A SVM Linear classifier is built to fit the data you supply and provide a hyperplane that fits well and classify your data into different classes. Following that, you may input some attributes to your classifier to check what the projected class is once you've obtained the hyperplane.

Support Vectors are the points which can be considered as edge cases. They are very nearer to the hyperplane. The two support vectors corresponding to either classes benign and malicious respectively are equidistant to the hyperplane with maximum margin possible.

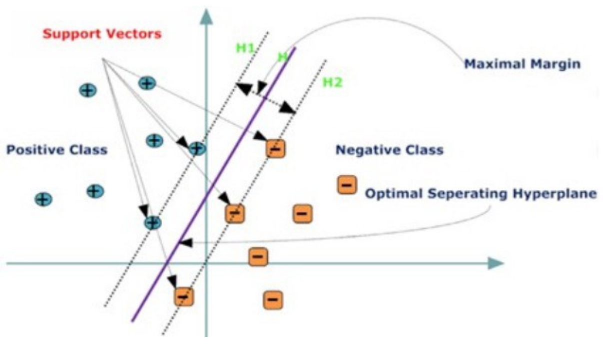


Figure 4: Support Vector Machine Terminology

4.1.4 SVM RBF

In classification using Support vector machines, the model with polynomial kernel performs merely when the positive examples and negative examples in the data are overlapping. One way to deal with this overlapping data is to use a support vector machine with a radial kernel generally known as Radial Basis Function(RBF). When RBF kernel is used in SVM, radial kernel behaves like a weighted nearest neighbor model. In other words, the nearest observation has a lot of influence on how we classify the new example. The value obtained after substituting in radial kernel function is inversely proportional to the closeness. The radial kernel function of two data observations a,b is as below.

$$RBF(a,b) = e^{-\gamma(a-b)^2}$$

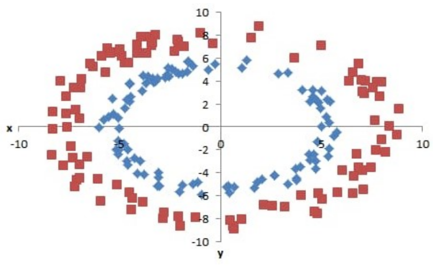


Figure 5: Example of a dataset that isn't linearly seperable

4.1.5 Decision Tree

Decision Trees are a non-parametric supervised learning approach which can be used for classifying problems as well as regression problems. The sole objective is to construct a machine learning model that guesses the class of a given instance by learning basic decision rules from feature values.

First, we will calculate the entropy. It is also known as measure of uncertainity. Then for each attribute A, we calculate information gain. The attribute with maximum value for information gain will be selected as the root node and this process continues. The formulas for entropy and Information gain are:

$$E(S) = -[plog(p) + (1-p)log(1-p)]$$

$$Gain(S, A) = E(S) - \sum_{v} p_v E(S_v)$$

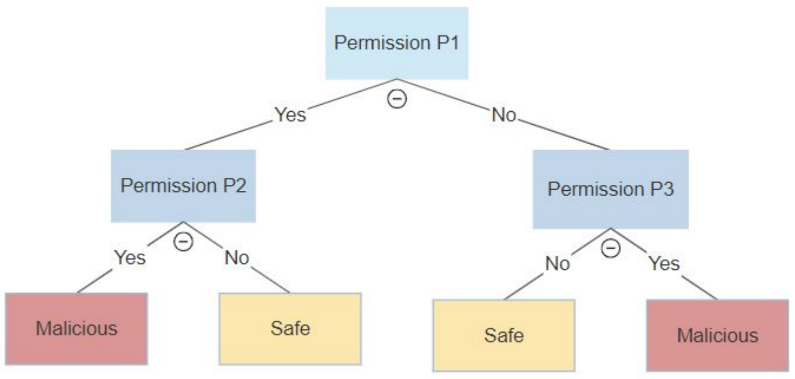


Figure 6: Illustration of Decision Tree

4.1.6 Random Forest Method

Random forest algorithm is a supervised learning method of machine learning. It produces a "forest" out of a collection of multiple decision trees. These trees are often trained using an approach called bagging approach. The main idea of this approach is that combining many learning models enhances total output. In simple words, random forest is a method of combining many decision trees to get a more accurate and reliable prediction.

Random Forest is a Machine learning classifier that has various decision trees on different subsets of input datasets and takes the majority of them to improve the performance of the model. The more number of decision trees we take, the higher the performance of our model will be.

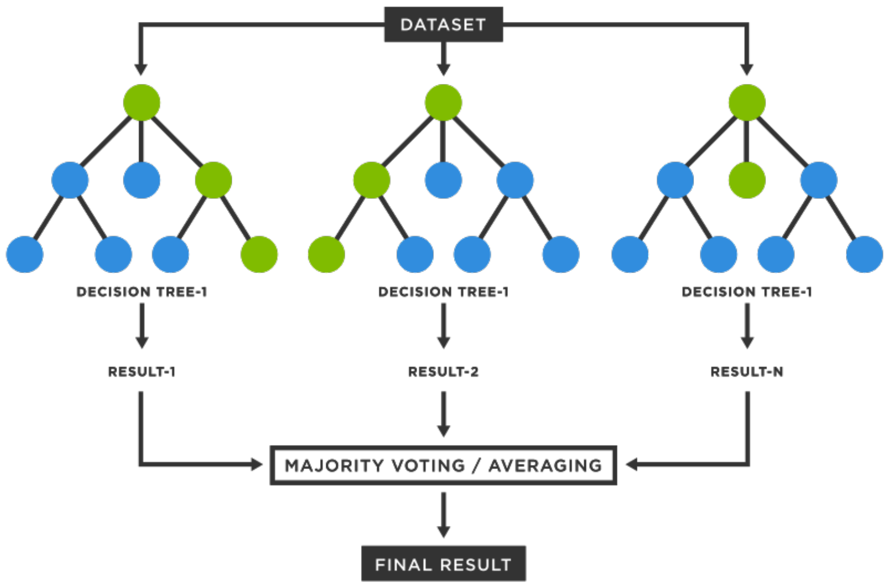


Figure 7: Illustration of Random Forest

4.1.7 Gaussian Naive Bayes

A collection of classification algorithms which are based on a theorem named after Bayes together forms a Naive Bayes classifier. It isn't a single algorithm. It is a group of algorithms sharing one common principle i.e., every pair of features which are used in classification are independent of each other.

For two events E_a, E_b Bayes theorem tells that:

$$P(E_a|E_b) = P(E_b|E_a) * \frac{P(E_a)}{P(E_b)}$$

Using this principle for classification task, we can say that:

$$Pr(class|attributes) = Pr(attributes|class) * \frac{Pr(class)}{Pr(attributes)}$$

As we have two classes malicious and safe, we will calculate the probabilities for the application to be in malicious class and to be in safe class. The class for which the probability value is higher, is the class to which the application belongs to.

$$\hat{class} = \underset{class}{\operatorname{argmax}} P(class|attribute)$$

4.1.8 Sequential Neural Network

We have a sequential neural network with 3 hidden layers. It has 1 input layer and 1 output layer also. The input layer consists of 54 features. The first hidden layer has 16 nodes. Second layer has 8 nodes. And the final hidden layer has 4 nodes. The three hidden layers have relu activation function and output layer has sigmoid function.

We have divided the input data into batches of size 32. It is called mini-batch mode. We used Rmsprop optimizer for the optimization to preventing the decline of learning rate of our model. We took 30 epochs. Epoch means that we passed the whole training data into our neural network once. In 10 epochs, we pass the whole training data 10 times.

After every epoch, the average loss is computed and the neural network back-propagates to the first hidden layer and updates the weights such that the loss is minimized. We used cross entropy as our loss function.

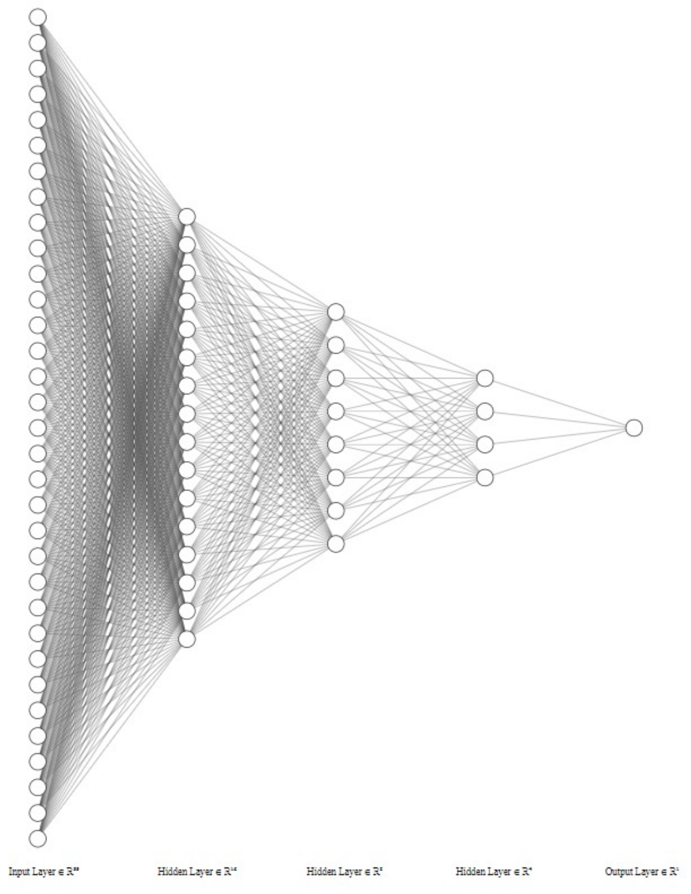


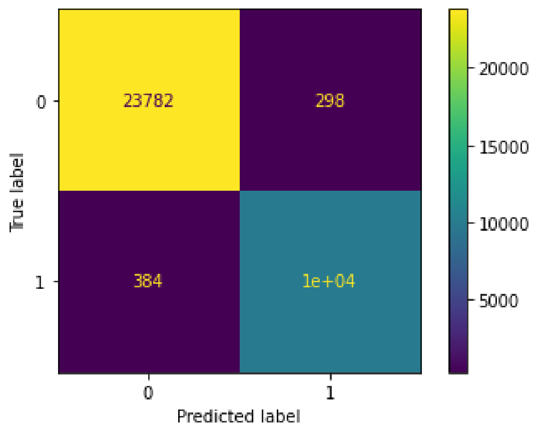
Figure 8: Illustration of Sequential Neural Network showing all its layers

5 Results

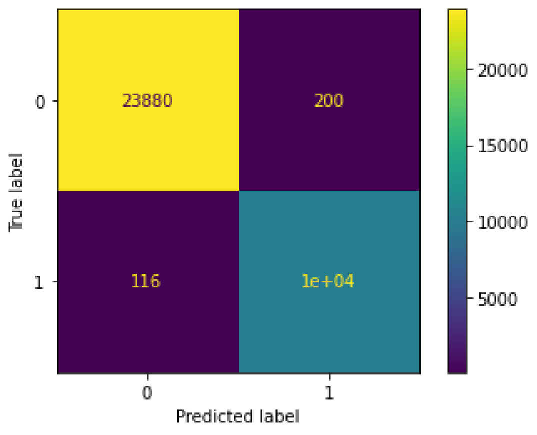
In Table-1, we can see the performance metrics for various models. The performance metrics include accuracy, precision as well as recall. It also has f1-score.

S.No	Classifier Type	Accuracy	Precision	Recall	F1-score
1	Logistic Regression	91.02	0.91	0.91	0.91
2	K-Nearest Neighbours	92.56	0.90	0.93	0.92
3	SVM Linear	93.85	0.94	0.92	0.93
4	SVM RBF	94.79	0.94	0.93	0.93
5	Gaussian Naive Bayes	53.11	0.71	0.64	0.52
6	Decision Tree	97.14	0.95	0.96	0.96
7	Random Forest	97.45	0.97	0.97	0.97

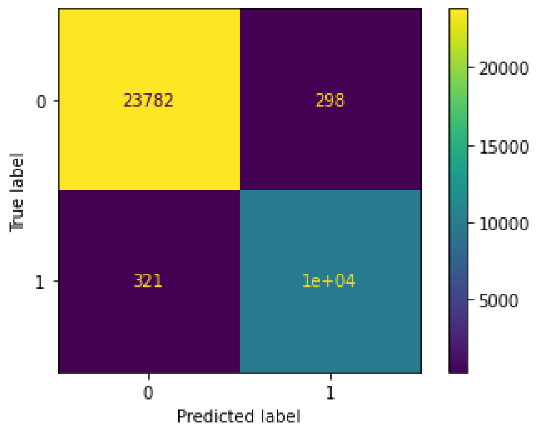
Table 1: Performance metrics of different models



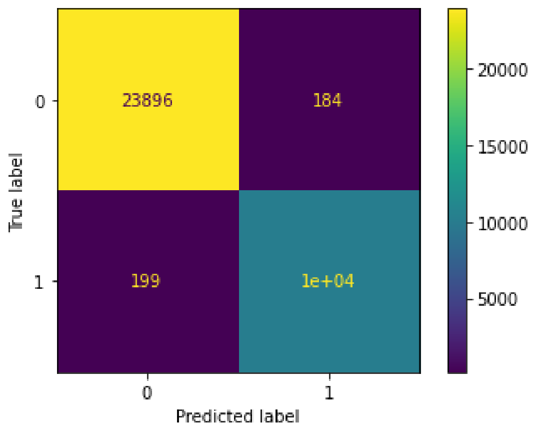
(a) Logistic Regression



(b) KNN

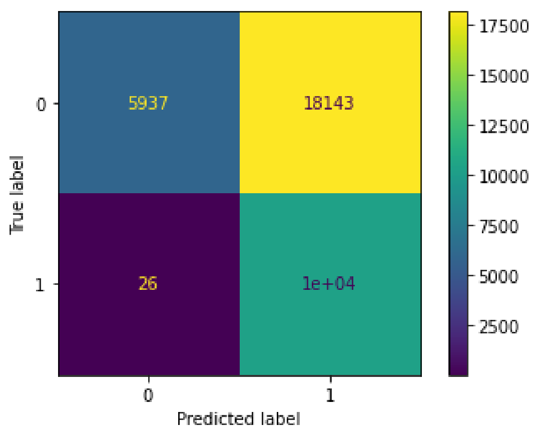


(c) SVM Linear

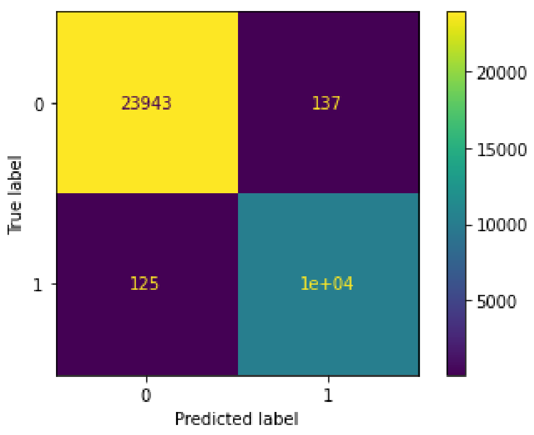


(d) SVM RBF

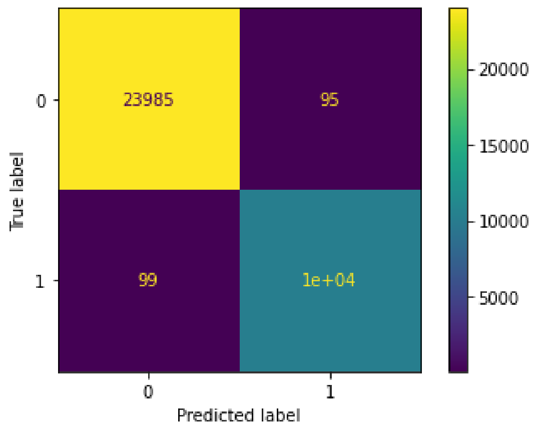
Figure 9: Confusion Matrices



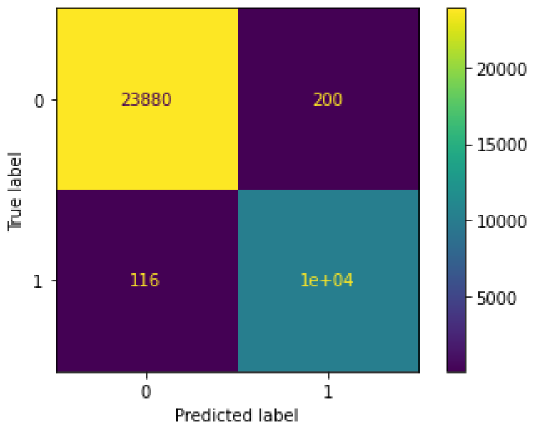
(a) Gaussian NB



(b) Decision Tree



(c) Random Forest



(d) Sequential Neural Network

Figure 10: Confusion Matrices

In Figure-9 and Figure-12, we can see the confusion matrices for various models. Confusion matrix compares the predicted classes with actual classes of applications. We can obtain the values of True Positive, False Positive, True Negative and False Negative from these confusion matrices.

Accuracy can be computed by dividing the number of correctly classified applications by total number of applications.

$$Accuracy = \frac{No.of correctlyclassified applications}{Totalno.of applications}$$

Precision can be computed by dividing the number of applications that are predicted correctly that they belong to a particular class by the number of applications that are predicted to be in that particular class.

$$Precision = \frac{TP}{TP + FP}$$

Recall can be computed by dividing the number of applications that are predicted correctly that they belong to a particular class by the number of applications that should actually be in that particular class.

$$Recall = \frac{TP}{TP + FN}$$

F1-score is a metric involving both Precision and Recall and can be computed as:

$$F1 - score = \frac{2 * Precision * Recall}{Recall + Precision}$$

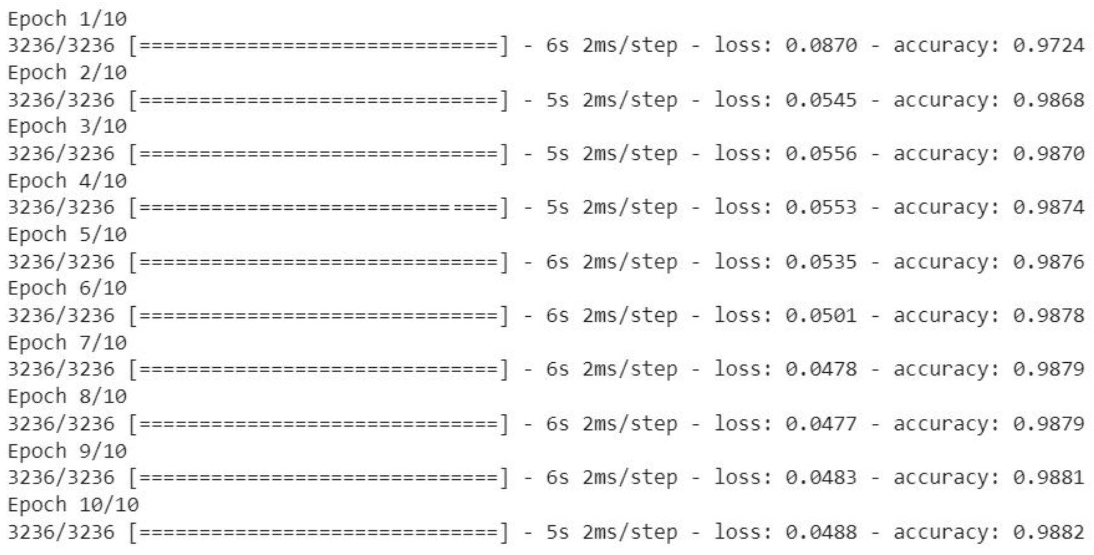
For Sequential Neural Network, we took 3 hidden layers of length 16, 8 and 1.

We used relu activation function for the hidden layers and sigmoid function for the output layer.

Layer	Size	No. of Param
dense	16	880
dense-1	8	136
dense-2	4	36
dense-3	1	5

Table 2: Summary of Sequential Neural Network

We took cross entropy value for our loss function and chose rmsprop as our optimizer. After 10 epochs, we got an accuracy of 98.82 %.



```
Epoch 1/10
3236/3236 [==============================] - 6s 2ms/step - loss: 0.0870 - accuracy: 0.9724
Epoch 2/10
3236/3236 [==============================] - 5s 2ms/step - loss: 0.0545 - accuracy: 0.9868
Epoch 3/10
3236/3236 [==============================] - 5s 2ms/step - loss: 0.0556 - accuracy: 0.9870
Epoch 4/10
3236/3236 [==============================] - 5s 2ms/step - loss: 0.0553 - accuracy: 0.9874
Epoch 5/10
3236/3236 [==============================] - 6s 2ms/step - loss: 0.0535 - accuracy: 0.9876
Epoch 6/10
3236/3236 [==============================] - 6s 2ms/step - loss: 0.0501 - accuracy: 0.9878
Epoch 7/10
3236/3236 [==============================] - 6s 2ms/step - loss: 0.0478 - accuracy: 0.9879
Epoch 8/10
3236/3236 [==============================] - 6s 2ms/step - loss: 0.0477 - accuracy: 0.9879
Epoch 9/10
3236/3236 [==============================] - 6s 2ms/step - loss: 0.0483 - accuracy: 0.9881
Epoch 10/10
3236/3236 [==============================] - 5s 2ms/step - loss: 0.0488 - accuracy: 0.9882
```

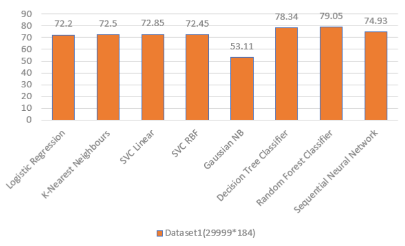
Figure 11: Loss and Accuracy after every epoch

Inorder to compare the performances of different models, we thought it's not fair to judge them based on the results of a single dataset. So, we took 3 different datasets of different input dimensions and trained the models accordingly. The results obtained after training are noted down in Table 3

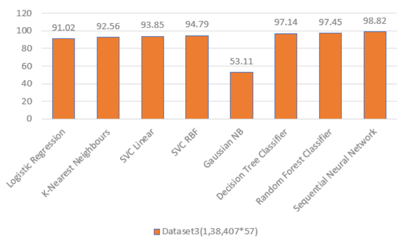
S.No	Model Name	Testing Accuracy		
		Dataset 1 (29999*184)	Dataset 2 (70*17)	Dataset 3 (138047*57)
1	Logistic Regression	72.2	75.13	91.02
2	K Nearest Neighbors	72.5	83.3	92.56
3	SVM Linear	72.85	83.3	93.85
4	SVM RBF	72.45	66.6	94.79
5	Gaussian NB	53.11	62.5	53.11
6	Decision Tree	78.34	91.6	97.14
7	Random Forest	79.05	98.21	97.45
8	Sequential Neural Network	74.93	70.83	98.82

Table 3: Performance of different models on different datasets

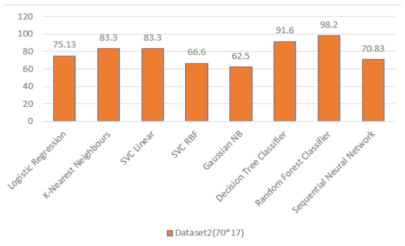
From the results obtained, we can observe that Random Forest outperforms all other algorithms, nevertheless it only marginally performs better than decision trees. For dataset 2 also, we find random forest giving best possible result(OA 98.21) to be noted, even here decision tree proved to be the second best algorithm. And for the final data set, a huge one with around 138000 records, the sequential neural network model we trained gave the best results. Random forest here takes the second place followed by decision trees.



(a) Dataset 1



(b) Dataset 2



(c) Dataset 3

Figure 12: Graphs based on testing accuracies of different models

6 Conclusion and Future Work

As android malwares are increasing day to day, we wanted to present a model that can detect malwares accurately. We considered the permissions asked by an android application as the features to our machine learning model. This comes under Static Analysis. The idea of considering these permissions as features worked well. We got good accuracy from most of the machine learning models that were implemented. The Sequential Neural Network got the highest accuracy of 98.82%. Out of all the remaining models, Random forest and Decision tree performed well. And Gaussian Naive Bayes model got the least accuracy among all the models that we implemented.

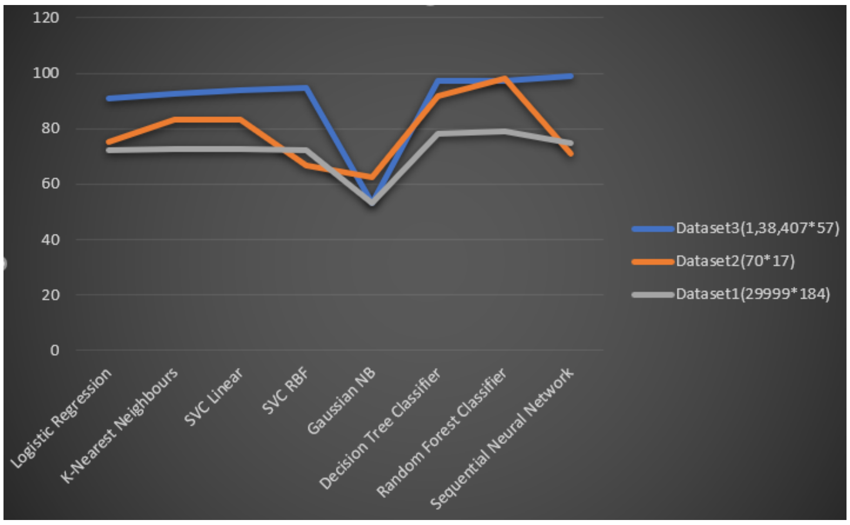


Figure 13: Comparisons of Different models on 3 datasets used

But, a set of android applications operating together can carry out a malicious activity. We call them colluding apps. In this, the malicious activity is carried out by more than one application. Each application participating in collusion does a small part of the malicious action. These applications communicate with each other through covert channels. Sometimes when a malicious activity cannot be performed by a single application, it might be possible that a group of applications coordinating with each other can perform that malicious activity. This phenomenon is called Application Collusion. It is an emerging threat.

The reason behind why we are calling this as an emerging threat is because most of the android malware detectors scan the applications individually when determining whether it is a malware or not. But as the malicious activity here is being carried out by a group of applications, those traditional detectors cannot detect this. So, we need a model to detect these colluding applications. Till now, very little research has been done on this and there is scarcity for datasets.

We are trying to create or obtain a few applications that can perform collusion, so that we can do some research on them which may eventually help in creating a model that can detect colluding applications. Firstly, we have to obtain a template for collusion. Then, we have to try to split a malicious task into various steps and make each application perform one of the steps. By doing this, we can achieve collusion.

References

[1] A. Arora, S. Garg, and S. K. Peddoju, "Malware detection using network traffic analysis in android based mobile devices," in *2014 Eighth International Conference on Next Generation Mobile Apps, Services and Technologies*, pp. 66–71, 2014.

[2] A. Arora, S. K. Peddoju, and M. Conti, "Permpair: Android malware detection using permission pairs," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 1968–1982, 2020.

[3] Z. Yuan, Y. Lu, and Y. Xue, "Droiddetector: android malware characterization and detection using deep learning," *Tsinghua Science and Technology*, vol. 21, no. 1, pp. 114–123, 2016.

[4] S. Y. Yerima, S. Sezer, and I. Muttik, "Android malware detection using parallel machine learning classifiers," in *2014 Eighth International Conference on Next Generation Mobile Apps, Services and Technologies*, pp. 37–42, 2014.

[5] N. Peiravian and X. Zhu, "Machine learning for android malware detection using permission and api calls," in *2013 IEEE 25th International Conference on Tools with Artificial Intelligence*, pp. 300–305, 2013.

[6] J. Sahs and L. Khan, "A machine learning approach to android malware detection," in *2012 European Intelligence and Security Informatics Conference*, pp. 141–147, 2012.

[7] Z. Wang, J. Cai, S. Cheng, and W. Li, "Droiddeeplearner: Identifying android malware using deep learning," in *2016 IEEE 37th Sarnoff Symposium*, pp. 160–165, 2016.

[8] H.-S. Ham and M.-J. Choi, "Analysis of android malware detection performance using machine learning classifiers," in *2013 International Conference on ICT Convergence (ICTC)*, pp. 490–495, 2013.

[9] U. Pehlivan, N. Baltaci, C. Acartürk, and N. Baykal, "The analysis of feature selection methods and classification algorithms in permission based android malware detection," in *2014 IEEE Symposium on Computational Intelligence in Cyber Security (CICS)*, pp. 1–8, 2014.

[10] H. Fereidooni, M. Conti, D. Yao, and A. Sperduti, "Anastasia: Android malware detection using static analysis of applications," in *2016 8th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, pp. 1–5, 2016.