A NOVEL APPROACH TO DETECT MALWARE IN SMART DEVICES USING CASCADED MACHINE LEARNING TECHNIQUES

Project Report submitted to
Shri Ramdeobaba College of Engineering & Management,
Nagpur in partial fulfillment of requirement for the award of
degree of

BACHELOR OF ENGINEERING

In

COMPUTER SCIENCE AND ENGINEERING

By

Abhishek Yelne (A-12)
Gagan Badule (B-41)
Hardik Dharmik (B-42)
Shivam Gupta (B-76)

Guide

Prof. P. R. Pardhi

RCOEM
Shri Ramdeobaba College of
Engineering and Management, Nagpur

Department of Computer Science and Engineering

Shri Ramdeobaba College of Engineering and Management, Nagpur 440013

(An Autonomous Institute affiliated to Rashtrasant Tukadoji Maharaj Nagpur University, Nagpur)

April 2023

A NOVEL APPROACH TO DETECT MALWARE IN SMART DEVICES USING CASCADED MACHINE LEARNING TECHNIQUES

Project Report submitted to

Shri Ramdeobaba College of Engineering & Management, Nagpur in partial fulfillment of requirement for the award of degree of

BACHELOR OF ENGINEERING

In

COMPUTER SCIENCE AND ENGINEERING

By

Abhishek Yelne (A-12)

Gagan Badule (B-41)

Hardik Dharmik (B-42)

Shivam Gupta (B-76)

Guide

Prof. P. R. Pardhi

RCOEM

Shri Ramdeobaba College of Engineering and Management, Nagpur

Department of Computer Science and Engineering

Shri Ramdeobaba College of Engineering and Management, Nagpur 440013

(An Autonomous Institute affiliated to Rashtrasant Tukadoji Maharaj Nagpur University, Nagpur)

April 2023

SHRI RAMDEOBABA COLLEGE OF ENGINEERING & MANAGEMENT, NAGPUR

(An Autonomous Institute affiliated to Rashtrasant Tukadoji Maharaj Nagpur University, Nagpur)

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Thesis on **"A Novel Approach To Detect Malware In Smart Devices Using Cascaded Machine Learning Techniques"** is a bonafide work of Abhishek Yelne, Shivam Gupta, Hardik Dharmik and Gagan Badule submitted to the Rashtrasant Tukadoji Maharaj Nagpur University, Nagpur in partial fulfillment of the award of a Degree of Bachelor of Engineering. It has been carried out at the Department of Computer Science and Engineering, Shri Ramdeobaba College of Engineering and Management, Nagpur during the academic year 2022-23.

Date:

Place: Nagpur

Prof. P. R. Pardhi
(Project Guide)
Department of Computer
Science and Engineering

Dr Avinash Agrawal
(H.O.D)
Department of Computer
Science and Engineering

Dr. R. S. Pande
(Principal)

DECLARATION

I, hereby declare that the thesis “A NOVEL APPROACH TO DETECT MALWARE IN SMART DEVICES USING CASCADED MACHINE LEARNING TECHNIQUES” submitted herein, has been carried out in the Department of Computer Science and Engineering of Shri Ramdeobaba College of Engineering & Management, Nagpur. The work is original and has not been submitted earlier as a whole or part for the award of any degree / diploma at this or any other Institute / University.

Date:

Place: Nagpur

Name of the Student	**Roll No.**	**Signature**
Abhishek Yelne	A-12	
Gagan Badule	B-41	
Hardik Dharmik	B-42	
Shivam Gupta	B-76	

APPROVAL SHEET

This report entitled

“A NOVEL APPROACH TO DETECT MALWARE IN SMART DEVICES USING CASCADED MACHINE LEARNING TECHNIQUES”

By

Abhishek Yelne

Gagan Badule

Hardik Dharmik

Shivam Gupta

is approved for the degree of Bachelor of Engineering.

Name & signature of Supervisor(s)   Name & signature of External Examiner(s)

Name & signature of HOD

Date:

Place: Nagpur

ACKNOWLEDGEMENT

The project is a combined effort of a group of individuals who synergize to contribute towards the desired objectives. Apart from the efforts by us, the success of the project shares an equal proportion on the engagement and guidance of many others. We take this opportunity to express our gratitude to the people who have been instrumental in the successful completion of this project.

We would like to show our greatest appreciation towards Prof. P.R Pardhi for providing us with the facilities for completing this project and for his constant guidance.

We would like to express our deepest gratitude to Dr. Avinash Agrawal, Head, Department of Computer Science and Engineering, RCOEM, Nagpur for providing us the opportunity to embark on this project.

Finally, extend our gratitude to all the faculty members of the CSE department who have always been so supportive and providing resources needed in our project development. We are very grateful to all of them who have unconditionally supported us throughout the project.

Date:

— Projectees

ABSTRACT

The growing use of mobile devices has led to an increase in the number of malware attacks targeting Android devices. Malware can cause a range of problems, from stealing personal information to damaging the device or network it is connected to. Therefore, detecting and preventing malware attacks on Android devices is essential for ensuring the security and privacy of users. In this project, we explore various methods and techniques used in Android malware detection, including static analysis, dynamic analysis, and machine learning.

The major aim of this project is to develop an Android malware Detection system by monitoring the feature set requested by application. In such a case when malware is detected, the users can take precautionary measures to safeguard themselves from cyber-threats. This detection system provides a Cascaded Machine Learning based technique for judging different malware families.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Abbreviation	Expansion
SVM	Support Vector Machine
KNN	K Nearest Neighbours
APK	Android Application Package
ML	Machine Learning
RFC	Random Forest Classifier
SHA	Secure Hash Algorithm
CSV	Comma Separated Values
XML	Extensible Markup Language
TP	True Positive
TN	True Negative
FP	False Positive
FN	False Negative

CHAPTER 1
INTRODUCTION

The increasing use of smart devices such as smartphones, tablets, and smartwatches has made them attractive targets for cybercriminals. Malware attacks on smart devices can result in data theft, unauthorized access, and device damage. Therefore, it is important to develop effective techniques for detecting malware in smart devices. In this report, we present a novel approach to detect malware in smart devices using cascaded machine learning techniques. The proposed approach uses multiple machine learning models in a cascaded manner to achieve high detection rates with low false positives. We provide a detailed description of the proposed approach, including the various machine learning models used, the feature extraction process, and the evaluation metrics used to evaluate the performance of the approach. We also present the results of experiments conducted to evaluate the performance of the proposed approach using a dataset of real-world malware samples. The results demonstrate that the proposed approach can achieve high detection rates with low false positives, making it a promising approach for detecting malware in smart devices.

1.1 Problem Definition

To develop an android malware detection system that will use Cascading Machine Learning approaches to detect whether a given Android APK is Malware or not and to protect users from cyber threats. Malware detection is a critical issue and needs to be as accurate as it can be.

1.2 Motivation

The proliferation of smart devices such as smartphones, tablets, and smartwatches has led to an exponential increase in their usage in recent years. These devices have become an essential part of our lives, allowing us to stay connected, access information, and carry out various tasks on the go. However, with the increasing use of these devices comes the risk of cyberattacks, including malware attacks.

Malware attacks on smart devices can lead to data theft, unauthorized access, and even device damage. Cybercriminals are constantly looking for new ways to exploit vulnerabilities in smart devices and install malware. As a result, there is a pressing need to develop effective techniques for detecting malware in smart devices.

1.3 Overview

Android malware detection is the process of identifying malicious software that can affect Android devices. The use of Android devices has become ubiquitous in our daily lives, and with the increasing use of mobile applications, the threat of malware attacks has also grown. Malware can infiltrate Android devices through various means, including app downloads from untrusted sources, phishing attacks, or even through network vulnerabilities.

The detection of Android malware is a critical process in maintaining the security of Android devices. There are various approaches to detect Android malware, including signature-based, behavior-based, and machine learning-based methods. Signature-based detection involves matching the signature of known malware with the code of the target application. Behavior-based detection involves analyzing the behavior of an application and identifying any deviations from expected behavior. Machine learning-based detection involves training ML algorithms to identify patterns in malware and distinguish them from benign software.

The development of new and effective approaches for Android malware detection is a continuous process due to the dynamic nature of malware. Malware creators continuously evolve their techniques to evade detection and exploit vulnerabilities in the Android ecosystem. Therefore, researchers and security professionals must stay abreast of new developments in the field and continue to develop and improve Android malware detection methods. The detection of Android malware is crucial to maintain the security and privacy of Android devices and protect users from the harmful effects of malware attacks.

Machine learning models such as SVM, KNN, Naive Bayes, Random Forest are being used for classification. Random Forest model which is known for performing both regression and classification tasks. In this project, RFC is used to classify whether an application is benign or malware.

1.4 Objectives

The following are the objectives of the project:

- To review the previous studies and efforts put on to build the predictive models.
- To develop an android malware detection system using a cascading approach that combines machine learning, signature-based detection, behavioral analysis, and control flow analysis.
- To provide a comprehensive solution for android malware detection that can detect unknown malware and help to protect users from cyber threats.

CHAPTER 2

LITERATURE SURVEY

The various work on detecting android malware to tackle real life cyber-threats are discussed in this section.

Saleh M. Shehata, et al[1]. This article proposed a powerful Android MAlware detection system. Malware-based Android permissions were used to develop it. The approach that was suggested aimed to identify significant characteristics of malware by analyzing permissions extracted from the AndroidManifest.xml file. These identified features were used to train machine learning classifiers. Through experiments, it was demonstrated that this approach is capable of accurately detecting both benign and malicious applications, with a success rate of 95%. The experiment involved analyzing a sample set of 18,490 applications, consisting of 11,672 benign and 6,818 malignant applications.

Rahman Ali, et al [2] The article offers a comprehensive review of recent studies that investigate the use of deep learning techniques for intrusion and malware detection, as well as their classification. The authors conducted a thorough analysis of 107 papers from five popular digital libraries, examining the types of threats targeted and the effectiveness of deep learning-based systems in identifying new security threats. The review highlights that CNN, LSTM, DBN, and autoencoders were the most commonly used deep learning methods. As a result, this paper is a valuable resource for those interested in researching malware detection using deep learning techniques

Hasan Alkahtani,et al [3] This article utilized various machine learning and deep learning techniques to detect malware attacks on Android devices. The SVM, KNN, LDA, LSTM, CNN-LSTM, and autoencoder algorithms were applied and high correlations were found for important features to protect against attacks. These algorithms successfully detected malware in Android applications with SVM achieving the best accuracy using the CICAndMal2017 dataset, and LSTM achieving high accuracy on the Drebin dataset. The study found a strong relationship between predicted and target values and the SVM, LSTM, and CNN-LSTM algorithms were found to be highly efficient in detecting malware in the Android environment compared to existing security systems.

Jiayin Feng.et al [4] In this paper a two-layer strategy to find malware in Android APPs is suggested in this study. Based on authorization, intent, and component information, the first level paradigm for static malware detection. To detect malware and assess the performance of the system through experimentation, it combines static features with a fully connected neural network; the first layer's detection rate is 95.22%. In the second layer, the outcome (good APPs from the first layer) is then input. A novel technique called CACNN, which cascades CNN and AutoEncoder, is utilized in the second layer to find malware using the network traffic properties of APPs. The binary classification (2-classifier) second layer has a 99.3% detection rate. Additionally, the new two-layer approach can identify malware based on its category (4-classifier) and malicious family.

Laraib U. Memon et.al [5] In this article they thoroughly assess Android applications and categorize them as either benign or malicious applications. Utilized seven different machine learning classifiers on the SherLock dataset, one of the largest datasets available for the detection of malware in Android applications, using a 17-node Apache Spark cluster. They found many appropriate properties of programmes that are involved in recognising malware from the dataset of 83 attributes. In the identification of malware apps, our investigation showed that the gradient boosted trees classification mechanism offers the highest precision, accuracy, and lowest false positive rate. It also used a methodology to create a real-time virus detection system based in the cloud.

Fausto Fasano .et al [6] In this paper, it looks into the possibilities of identifying malicious Android applications and the families to which they belong. In order to achieve this, they used a cascade learner that consists of two classifiers: the first one determines if a sample under investigation is malware or trusted, and the second one determines whether a sample has already been classified as malware in order to identify its family. They tested the suggested strategy using two mobile malware repositories and a real-world dataset of Android applications from Google Play. In order to identify trusted samples, they achieved a good precision to detect malware and about 12 malware families.

Dragos Gavrilut et al. [7] presented a versatile framework that utilizes various machine learning techniques to accurately differentiate between malware and clean files, while minimizing false positives. The authors conducted experiments using three different datasets: a training dataset, a test dataset, and a "scale-up" dataset. They initially employed cascade one-sided perceptrons on the datasets, followed by cascade kernelized one-sided perceptrons, and discussed the framework's underlying concepts. Subsequently, they demonstrated the scalability of the proposed framework by applying it to large datasets of both malware and clean files, after successfully testing it on medium-sized datasets.

Daniel Arp, et al [8] This paper proposes a lightweight method called DREBIN for detecting Android malware that conducts static analysis to collect features of an application. The shared vector space integrates these features, allowing for the identification of typical patterns indicating malware. In an experiment with 123,453 applications and 5,560 malware samples, DREBIN outperformed related algorithms, detecting 94% of malware with few false alarms. The method takes only 10 seconds to check for malware and provides reasons for each detection that indicate important characteristics of the malware found.

Zinal D. Patel [9] The focus of this paper is to present a comprehensive survey of various systems that utilize a static analysis approach for detecting malware. The paper provides a detailed summary of the existing systems such as DREBIN, AndroDialysis, ICCDetector, APKAuditor, DMDAM, DroidNative, and API-based systems, which are explained and analyzed. In addition, a comparative analysis of these detection systems is presented. The paper also aims to give a brief overview of the techniques employed by each of these systems, and they are compared and contrasted.

CHAPTER 3

METHODOLOGY

3.1 Dataset

The Drebin dataset is a widely used dataset for malware detection in Android devices. It contains over 120,000 Android applications, including both benign and malicious applications, and is named after the first malware sample included in it, "Drebin". The dataset is known for its richness and includes real-world malware families, making it an ideal dataset for evaluating the performance of machine learning models. The dataset includes detailed information about permissions requested by each application, as well as other features such as the app's SHA-1 hash, package name, and developer certificate. The dataset contains feature vectors of 3766 attributes for static and 470 attributes for dynamic analysis extracted from 128, 391 applications (5,560 malware apps which have 179 malware families). The features in this dataset are based on Permissions which are in Manifest XML files, API call, signatures, information about Intents. In this project, we used the Drebin dataset to train and evaluate our cascaded machine learning model for detecting malware in smart devices. Figure 3.1 describes the process of custom dataset generation.

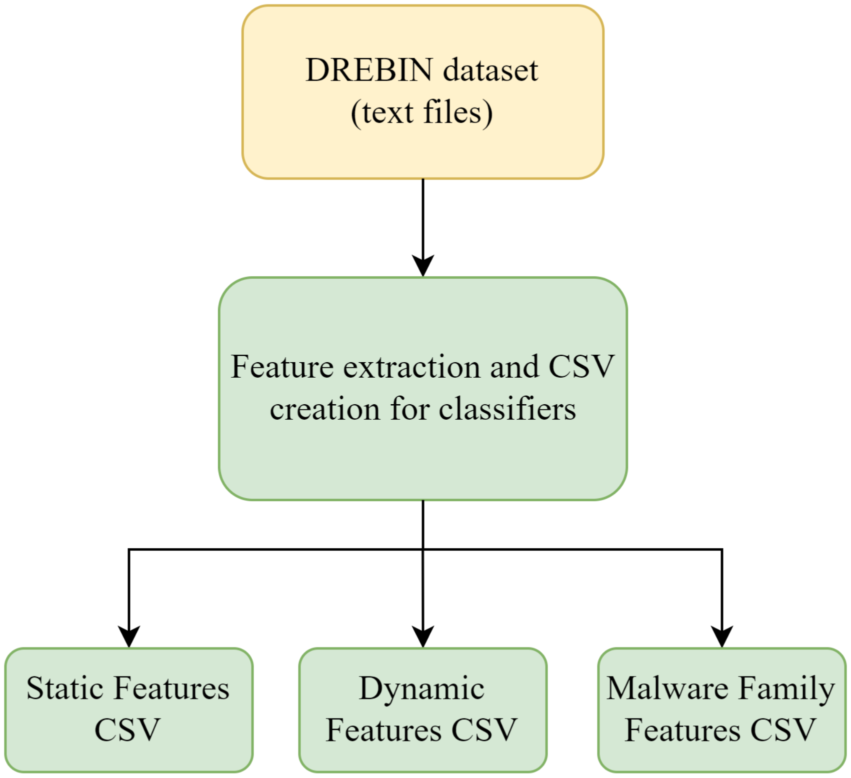

DREBIN dataset (text files)
Feature extraction and CSV creation for classifiers
Static Features CSV
Dynamic Features CSV
Malware Family Features CSV

Fig 3.1 Overview of Dataset collection

The dataset was provided in the form of text files. Each text file corresponded to a specific application and was named after the SHA256 hash of that application. These text files contained the features used by each application, and we parsed them to generate CSV files for analysis. In addition, Drebin provided a text file containing the SHA256 hash of the malware and their corresponding family, which served as ground truth for our analysis and classification of malware.

3.2 CSV Creation for classifiers

The text files were parsed to divide the dataset into three parts: one containing only static features, one containing only dynamic features, and one containing both static and dynamic features used by malware and their corresponding families.

Static features refer to the attributes of an application that remain constant throughout its execution. Some examples of static features include the application's permissions, filtered intents, app components, hardware features, and its digital signature. On the other hand, dynamic features are the attributes of an application that change while it is executing, such as the system calls it makes and the data it accesses.

3.3 Proposed Methodology

The Fig. 3.2 describes an overview of the proposed methodology. Our approach to detecting malware in smart devices using cascaded machine learning techniques involves a multi-step process that starts with data collection. We decompiled the input Android application and extracted its features from sources such as its manifest.xml file, which provided information about the application's permissions, intents, and other hardware features. We then divided the Drebin dataset into three parts, as described in section 3.2.

Using this division, we trained our machine learning models using the static feature data in the first level of our cascaded approach. This classifier was used to classify the input as either benign or malware. If the application was classified as benign by the first classifier, then the input was fed to the second level classifier, which was trained on the dynamic features data. The second level classifier then used the dynamic behavior of the application to classify it as either malware or benign. By doing so, the second layer classifier acted as a cross-check to ensure that the benign prediction made by the first classifier was accurate and not a false positive.

In cases where the input was classified as malware by the first level classifier or the second level classifier, the input was directly fed to the malware family classifier, which was trained on both the static and dynamic features of known malware. This allowed us to determine the specific family of malware to which the application belonged. This information was then used to alert the user and prevent potential cyber threats.

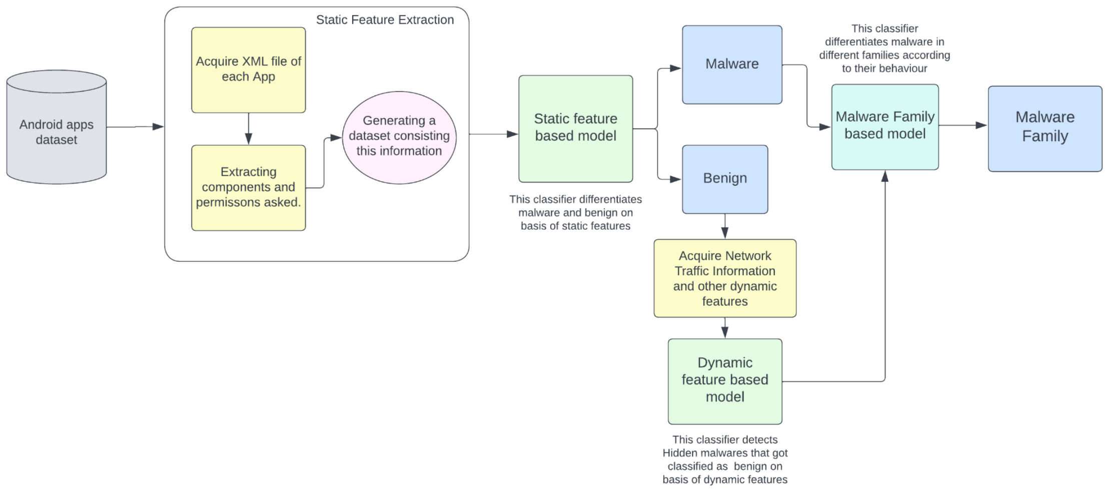

Android apps dataset
Static Feature Extraction
Acquire XML file of each App
Extracting components and permissons asked.
Generating a dataset consisting this information
Static feature based model
This classifier differentiates malware and benign on basis of static features
Malware
Benign
Acquire Network Traffic Information and other dynamic features
Dynamic feature based model
This classifier detects Hidden malwares that got classified as benign on basis of dynamic features
This classifier differentiates malware in different families according to their behaviour
Malware Family based model
Malware Family

Fig 3.2 Proposed Methodology

3.4 Modeling Approach

The modeling approach for Android malware detection using machine learning involves the following steps:

1. Data Collection: Collect a dataset of Android applications, consisting of both benign and malicious samples. The Drebin dataset is in the form of files and folders. It was then converted into CSV files for giving input to the model.

2. Data Preprocessing: Preprocess the dataset to extract features from the Android applications. Features could include permissions requested by the application, API calls made by the application, or system calls made by the application.

3. Feature Selection: Select the most relevant features from the preprocessed dataset. This step helps to reduce the dimensionality of the dataset, making it easier to train the machine learning models.

4. Model Selection: Choose an appropriate machine learning algorithm for the problem of Android malware detection. Popular machine learning algorithms used for this task include Decision Trees, Random Forest, Naive Bayes, and Support Vector Machines (SVM).

5. Model Training: Train the selected machine learning algorithm on the preprocessed and feature-selected dataset. This step involves splitting the dataset into training and testing sets, and then fitting the machine learning model on the training set.

6. Model Evaluation: Evaluate the performance of the trained machine learning model on the testing set. Metrics commonly used for evaluating the performance of the model include accuracy, precision, recall, and F1-score.

Overall, the key to a successful modeling approach for Android malware detection using machine learning is to carefully preprocess the dataset, select relevant features, and choose an appropriate machine learning algorithm while balancing between the model's performance and computational complexity.

3.4.1 Classification Models Used

a. **SVM**

Support Vector Machine (SVM) is a powerful machine learning algorithm used for supervised learning tasks including classification and regression. This type of linear classifier is designed to identify the optimal hyperplane that can separate different classes of data points in a dataset. The key principle of SVM is to identify the hyperplane that maximizes the margin between the two classes, where the margin represents the distance between the hyperplane and the closest data points of each class. This approach is known as "support vector," as it only considers the data points that are closest to the decision boundary. SVM can employ various kernel functions such as linear, polynomial, and radial basis function (RBF) kernels to map the input data to a higher-dimensional space where a hyperplane can be used to classify the data points.

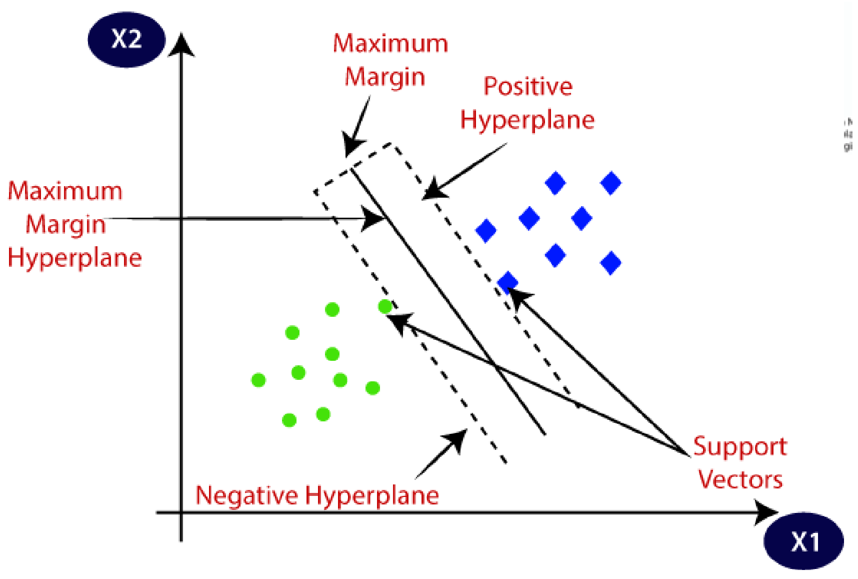


Fig 3.3 Support Vector Machine (SVM)

ALGORITHM: Training Model for SVM

Input: D=[X,Y]; X(array of input with application features), Y(array of class labels (M or B)

```
Y=array(C) //class label
function train_svm(X,Y,number_of_runs)
initialize:learning_rate=Math.random();
for learning_rate in number_of_runs
    for i in X
        if(Y[i]*(X[i]*w))<1 then
        update: w=w+learning_rate*((X[i]*Y[i])*(-2*(1/number_of_runs)*w)
        else
        update: w=w+learning_rate*(-2*(1/number_of_runs)*w)
        end if
    end
end
```

b. **Random Forest Classifier**

The Random Forest Classifier is a commonly used machine learning algorithm suitable for classification and regression tasks. It employs an ensemble learning approach by amalgamating multiple decision trees to produce a more resilient and precise model. The primary concept behind the Random Forest algorithm is to create a multitude of decision trees, where each tree is trained on a random subset of both the original dataset and the features. This technique helps reduce overfitting and enhances the model's overall accuracy. During the training process, each decision tree in the Random Forest is built by recursively splitting the data based on the best split that maximizes the information gain or Gini impurity. Once all the trees are built, the model predicts the class or value of a new input data point by taking the majority vote or average of the predictions made by all the trees in the forest.

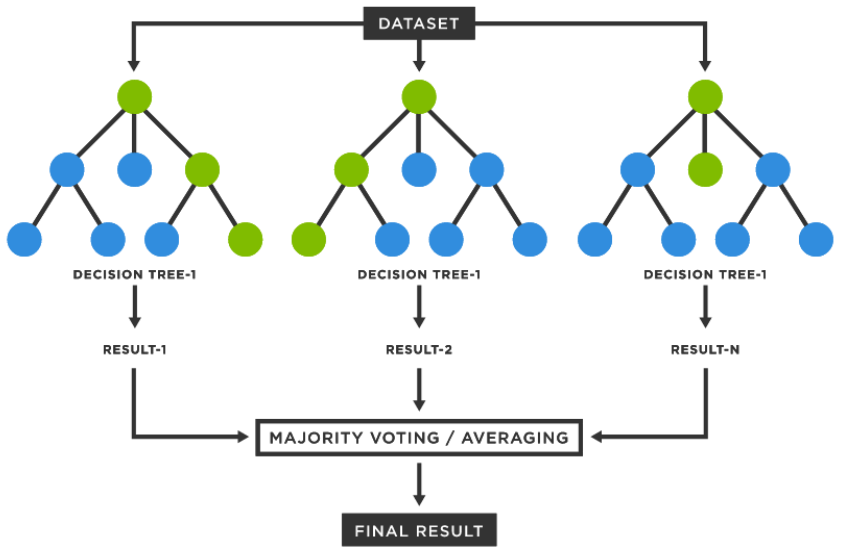


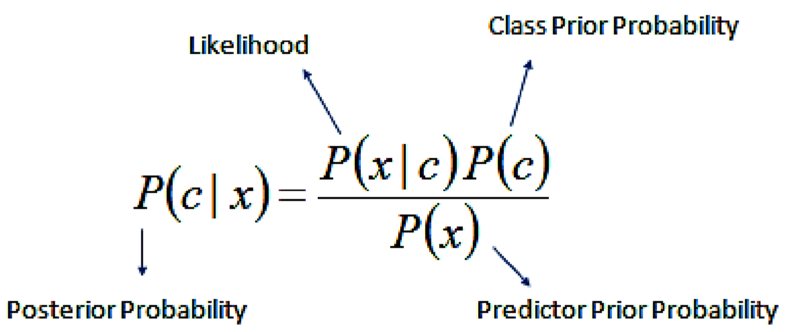
Fig 3.4 Random Forest

ALGORITHM: Training model for Random Forest

```
Precondition: A training set S = (x1,y1),---,(xn.yn), static features F, and number
of trees in forest B.
function RANDOMFOREST(S, F)
H = empty_set
for i = 1 to B do
        S(i) = A bootstrap sample from S
        hi = RANDOMIZEDTREELEARN(S(i), F)
        H = H union {hi}
end for
return H
end function
function RANDOMIZEDTREELEARN(S, F)
At each node:
        f = very small subset of F
        Split on best feature in f
        return The learned tree
end function
```

c. **Naive Bayes**

Naive Bayes is a simple but powerful machine learning algorithm that is widely used for classification tasks. It is based on Bayes' theorem, which is a probabilistic framework used to determine the probability of a hypothesis given evidence. The Naive Bayes algorithm assumes that all features in the dataset are independent of each other, hence the name "naive." This assumption makes the algorithm computationally efficient and enables it to work well with high-dimensional datasets. During the training process, the algorithm calculates the conditional probability of each feature given each class in the dataset. These probabilities are then used to calculate the posterior probability of each class given the input features using Bayes' theorem. Once the model is trained, it can be used to predict the class of new input data points based on their feature values. The algorithm calculates the posterior probability of each class for the input data point, and the class with the highest probability is selected as the prediction.



$$P(c \mid x) = \frac{P(x \mid c)P(c)}{P(x)}$$

$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

Fig 3.5 Naive Bayes

```
ALGORITHM: Training the Naive Bayes Model
Input:
Training dataset T,
F= (f1, f2, f3,.., fn) // value of predictor variable in testing dataset.
Output:
A class of testing dataset.
Step:
1. Read the training dataset T;
2. Calculate the mean and standard deviation of the
predictor variables in each class;
3. repeat
        Calculate the probability of fi using the gauss density equation in each class;
        Until the probability of all predictor variables (f1,f2,..,fn) has been calculated.
4. Calculate the likelihood for each class
5. Get the greatest likelihood.
```

d. **Decision Tree**

Decision Tree is a popular machine learning algorithm used for both classification and regression tasks. It is a type of supervised learning algorithm that learns a hierarchical tree-like structure of decision rules from the training data. During the training process, the algorithm recursively splits the data into subsets based on the feature values that best separates the classes in the dataset. This process continues until a stopping criterion is reached, such as when all the data points in a subset belong to the same class or when a certain depth of the tree is reached. The decision tree model can be visualized as a tree-like structure, where each node represents a decision based on a feature, and each branch represents the possible outcomes or values of the decision. The final decision or prediction is made at the leaf nodes of the tree.

ALGORITHM: GenDecTree(Sample S, Features F)

```
Steps:
1. If stopping_condition(S, F) = true then
        a. Leaf = createNode()
        b. leafLabel = classify(s)
        c. return leaf
2. root = createNode()
3. root.test_condition =findBestSpilt(S, F)
4. V={v| va possible outcomecfroot.test_condition}
5. For each valuev € V:
        a. Sv, = {s | root.test_condition(s) = v and s € S};
        b. Child = TreeGrowth (Sv,F );
        c. Add child as descent of root and label the edge {root ->
child} as v
6. return root
```

e. **K-Nearest Neighbors (KNN)**

K-Nearest Neighbors (KNN) is a simple yet powerful machine learning algorithm utilized for classification and regression tasks. It belongs to the instance-based learning category, indicating that the algorithm predicts results based on the similarity between new data points and the training dataset's data points.During the training process, the algorithm stores all the training data points in memory. When a new input data point is given, the algorithm finds the K-nearest neighbors to the new data point based on a distance metric, such as Euclidean distance or Manhattan distance. The algorithm then assigns the class or value of the new data point based on the majority class or the average value of its K-nearest neighbors. The value of K is an important parameter in the KNN algorithm. A smaller value of K results in a more flexible model, but it may also lead to overfitting. A larger value of K results in a more robust model, but it may also lead to underfitting.

ALGORITHM: k-Nearest Neighbor

```
1. Classify (X, Y, x) // X: training data of application features, Y: class labels
   of X, x: unknown sample
2. for i = 1 to m do
       a. Compute distance d(X,, x)
3. end for
4. Compute set I containing indices for the k smallest distances d(Xi, x).
5. return majority label for {Y, where i € I}
```

3.4.2 System Specification

The requirements to train the model are mentioned in Table 3.1.

Table 3.1 System Specification

Specification	**Systems Configuration**
Operating system	Windows 10
CPU	Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz
RAM	14 GB (usable)
Frameworks	Scikit Learn

3.5 Technology Stack

The project is divided into two parts, one is creating a csv format dataset from file format and then using this newly formed dataset to train above listed models.

a. **Python** is a popular high-level programming language used for a wide range of applications, including web development, scientific computing, data analysis, machine learning, and artificial intelligence. Python is known for its simplicity, readability, and ease of use, making it an ideal choice for beginners to learn programming. Python supports a wide range of programming paradigms, including procedural, object-oriented, and functional programming, and has a vast ecosystem of libraries and frameworks that support various domains and use cases.

b. **Scikit learn** is a popular open-source machine learning library for Python that provides a wide range of tools for various machine learning tasks such as classification, regression, clustering, and dimensionality reduction. Scikit-learn provides a wide range of algorithms and techniques for machine learning, including. Linear regression, logistic regression, and support vector machines (SVM) for classification and regression tasks. Decision trees, random forests, and gradient boosting for ensemble learning. Preprocessing and feature extraction tools for data cleaning and transformation. Scikit-learn also provides several tools for evaluating the performance of machine learning models, including metrics for classification, regression, clustering, and model selection.

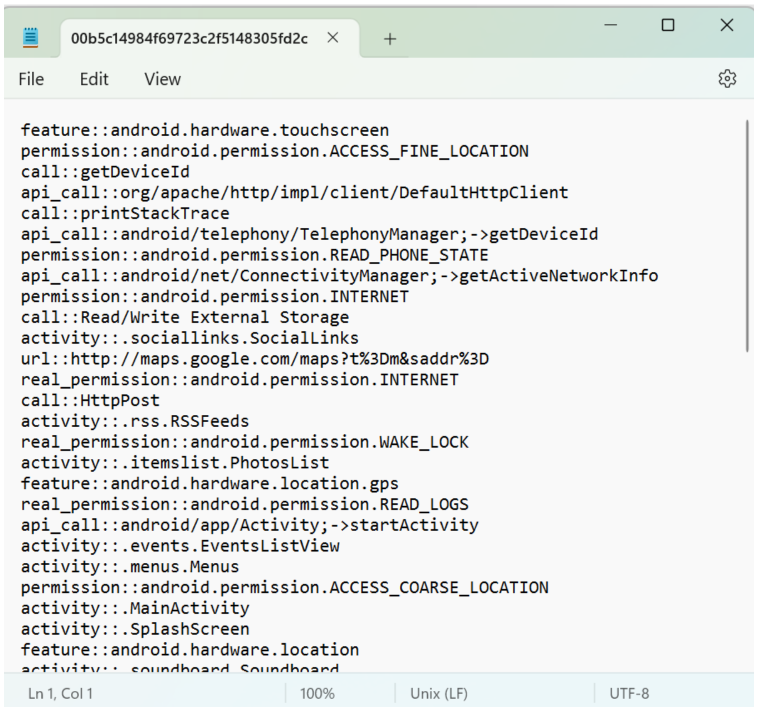
c. **Python Tkinter** is a standard GUI library for creating desktop applications, available on most platforms and included with most Python installations. Its simple syntax and wide range of widgets and tools make it a great choice for rapid application development, with compatibility with other Python libraries allowing for powerful data visualization and analysis applications. Its extensibility and customizability with third-party libraries make it possible to create sophisticated and powerful applications with a straightforward and intuitive user interface.

CHAPTER 4

IMPLEMENTATION

4.1 Classifier-Specific CSV Creation From The Dataset

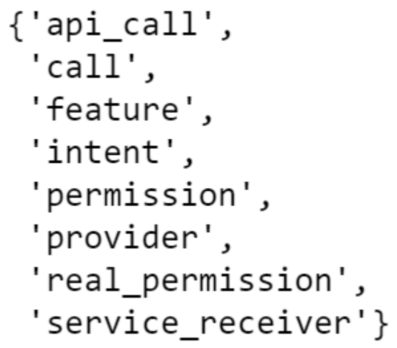
The Drebin dataset consisted of 128, 391 text files each corresponding to a n application. A text file is named after the SHA-256 of the application, and the file consists of the features of that application, one on each line. Figure 4.1 shows a sample file in the dataset. As can be seen, each line of the file is a feature, with "::" separator, in whose left side is the feature type, and the right side is the exact feature.



```
feature::android.hardware.touchscreen
permission::android.permission.ACCESS_FINE_LOCATION
call::getDeviceId
api_call::org/apache/http/impl/client/DefaultHttpClient
call::printStackTrace
api_call::android/telephony/TelephonyManager;->getDeviceId
permission::android.permission.READ_PHONE_STATE
api_call::android/net/ConnectivityManager;->getActiveNetworkInfo
permission::android.permission.INTERNET
call::Read/Write External Storage
activity::.sociallinks.SocialLinks
url::http://maps.google.com/maps?t%3Dm&saddr%3D
real_permission::android.permission.INTERNET
call::HttpPost
activity::.rss.RSSFeeds
real_permission::android.permission.WAKE_LOCK
activity::.itemslist.PhotosList
feature::android.hardware.location.gps
real_permission::android.permission.READ_LOGS
api_call::android/app/Activity;->startActivity
activity::.events.EventsListView
activity::.menus.Menus
permission::android.permission.ACCESS_COARSE_LOCATION
activity::.MainActivity
activity::.SplashScreen
feature::android.hardware.location
activity::.soundboard.Soundboard
```

Fig 4.1 A sample input file from Drebin

After analysis of all the text files using file parsing, it was found that there are a total 10 types of features. However, it was found that the 'url' and 'activity' types had features much larger (as much as 50 times more) than other types and also considering the fact that an application's url access information cannot be a basis for determining whether its malware or not since malicious hackers do not use specific url. Hence after removing these types, 8 types of features were finally used for further analysis, as shown in the Fig. 4.2 (the type ‘feature’ denotes the hardware features).



```
{'api_call',
 'call',
 'feature',
 'intent',
 'permission',
 'provider',
 'real_permission',
 'service_receiver'}
```

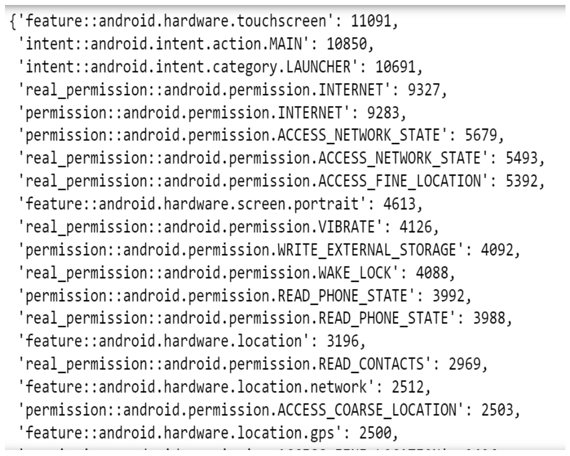
Fig 4.2 All features types

Since the machine learning classifiers cannot work on such semi-structured data in the form of text files, the data needed to be transformed into Comma Separated Values (CSV) format which is the standard dataset type for training these classifiers. Using file parsing in python and the use of the feature types - the following csv files were created:

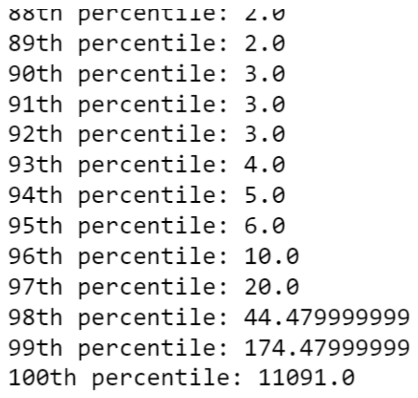
1) **Static features based dataset**

   This csv file was created by taking columns with all the features that are of static type, i.e., which can be known by analyzing the application’s APK file. The last column is the class, denoting whether an application is Benign (B) or Malware (M). The number of columns is 3,766. The 33,330 rows consist of all the malware samples in the dataset (count: 5,555) and benign samples 5 times larger in size than malware rows (i.e. count: 27,775).

   While analyzing the feature usages for benign samples, a map containing the count of applications using specific static features was made, which is shown in Figure 4.x and upon further analysis of the counts, it was found that 92% of the features are being used by less than 3 applications, which strongly suggest that such features may not be suitable for model building and hence these features need to be dropped and only top 8% features will be used for static classifier. The percentile analysis is depicted in figure 4.4.



```
{'feature::android.hardware.touchscreen': 11091,
 'intent::android.intent.action.MAIN': 10850,
 'intent::android.intent.category.LAUNCHER': 10691,
 'real_permission::android.permission.INTERNET': 9327,
 'permission::android.permission.INTERNET': 9283,
 'permission::android.permission.ACCESS_NETWORK_STATE': 5679,
 'real_permission::android.permission.ACCESS_NETWORK_STATE': 5493,
 'real_permission::android.permission.ACCESS_FINE_LOCATION': 5392,
 'feature::android.hardware.screen.portrait': 4613,
 'real_permission::android.permission.VIBRATE': 4126,
 'permission::android.permission.WRITE_EXTERNAL_STORAGE': 4092,
 'real_permission::android.permission.WAKE_LOCK': 4088,
 'permission::android.permission.READ_PHONE_STATE': 3992,
 'real_permission::android.permission.READ_PHONE_STATE': 3988,
 'feature::android.hardware.location': 3196,
 'real_permission::android.permission.READ_CONTACTS': 2969,
 'feature::android.hardware.location.network': 2512,
 'permission::android.permission.ACCESS_COARSE_LOCATION': 2503,
 'feature::android.hardware.location.gps': 2500,
```



```
88th percentile: 2.0
89th percentile: 2.0
90th percentile: 3.0
91th percentile: 3.0
92th percentile: 3.0
93th percentile: 4.0
94th percentile: 5.0
95th percentile: 6.0
96th percentile: 10.0
97th percentile: 20.0
98th percentile: 44.479999999
99th percentile: 174.47999999
100th percentile: 11091.0
```

Fig 4.3 Usage counts of static features

Fig 4.4 Percentile Analysis

Thus, the above methods have helped extract relevant features for the training of static classifiers. This model will check the genuinity of the input application, and decide whether it is malware or benign on the basis of static features from the input.

The steps performed in order to generate the CSV for the static feature based classifier model are:

1. Load the files containing features of applications.
2. Column_set = {}
3. Add all features of the malwares to the Column_set.
4. Perform feature extraction as mentioned above for features of benign applications to consider the relevant features for the model.
5. Add these features extracted above to the Column_set.
6. These columns, along with the rows identified by SHA-256 of the applications form the CSV file for the static feature based model.

2) Dynamic features based dataset

The dynamic features are used to cross-check the results from the first classifier model that worked on the static features, and verifies whether the application is truly benign if it was detected as benign in the previous step. Thus, this model will reduce the false positives by ensuring that no malware gets classified as a benign application.

The csv file on which this model is trained was created by considering columns with all the features that are dynamic, i.e., which can be known by analyzing the application's runtime behaviour. The last column is the class, denoting whether an application is Benign (B) or Malware (M). The number of columns is 469. The number of rows and their distribution are the same as that of static data csv.

The steps performed in order to generate the CSV for the dynamic feature based classifier model are the same as used for the static feature based classifier model except that here only dynamic features are considered:

3) Malware Family feature based dataset

This csv file was created by considering columns with all the features that are static as well dynamic, but which are used only by the malware in the dataset. The last column is the class, denoting the family to which the malware application belongs. In the Drebin dataset, there are malwares from 179 different families The number of rows is 5,555 which is the count of malware samples in the dataset.

The steps performed in order to generate the CSV for the malware family feature based classifier model are:

1. Load the files containing features of malware applications.
2. Column_set = {}
3. Add all features, static as well as dynamic of the malware applications to the Column_set.
4. The Column_set, along with the SHA-256 of the malware applications, form the CSV file for malware family features based classifier model.

4.2 Training Different ML Models On The Dataset

After the data is pre-processed, it is fed to the models on the dataset. Dataset is splitted into 70% training and 30% testing. The training data is given to the model and features are recognized and classified based on labels. Different ML models are trained on all the three CSVs and performance of models are then evaluated.

4.3 Malware Detection and Family Identification

First, features are extracted from Android Manifest-like files, to create an input vector. If the feature is present in Manifest, then it is set as 1 or else 0. This creates a feature vector which is given as input to the model. Then the model first predicts whether the input application is Malware or Benign. If it is found as Malware, then it is again fed to another model for its Malware family detection. If the application is found as Benign, then it is cross checked in another model for Malware again.

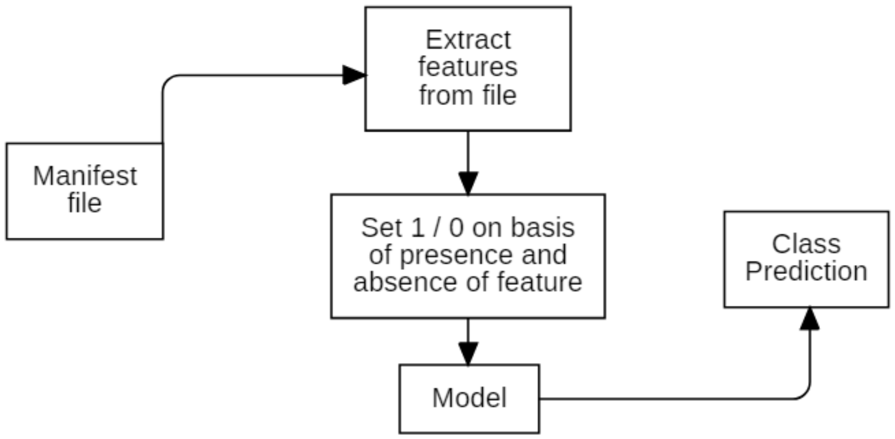

Manifest file
Extract features from file
Set 1 / 0 on basis of presence and absence of feature
Model
Class Prediction

Fig 4.5 Workflow of a Classifier Model

4.4 Evaluation Metrics

As models are classifying applications into Malware and Benign, following metrics of evaluation has been used:

Confusion Matrix : A confusion matrix is a tabular representation of prediction outcomes of any binary classifier, which is used to describe the performance of the classification model on a set of test data when true values are known.

In general, the table is divided into four terminologies, which are as follows:

1. **True Positive(TP):** In this case, the prediction outcome is true, and it is true in reality
2. **True Negative(TN)**: in this case, the prediction outcome is false, and it is false in reality
3. **False Positive(FP)**: In this case, prediction outcomes are true, but they are false in actuality.
4. **False Negative(FN)**: In this case, predictions are false, and they are true in actuality.

Accuracy: The proportion of correctly classified instances in the total number of instances. This is a simple and commonly used metric for classification tasks.

$$Accuracy \ = \ \frac{Number\ of\ Correct\ Predictions}{Total\ Number\ of\ Predictions}$$

Precision: The proportion of true positive instances among all instances predicted as positive. This metric is useful when the goal is to minimize false positives.

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

Recall: The proportion of true positive instances identified among all actual positive instances. This metric is useful since the goal is to minimize false negatives.

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

F1 Score: The harmonic mean of precision and recall, which gives equal weight to both metrics. This metric is useful when both false positives and false negatives are important

$$F1 - score \ = \ 2 * \frac{precision*recall}{precision + recall}$$

CHAPTER 5

RESULTS AND EVALUATION

After dataset creation, feature analysis was performed and generated Heatmap which is Fig 5.1. .From the given heatmap, we can analyze which features are predominantly related to output for Malware detection. *SEND_SMS, android.telephony.SmsManager, READ_PHONE_STATE* are top features contributing to class prediction.

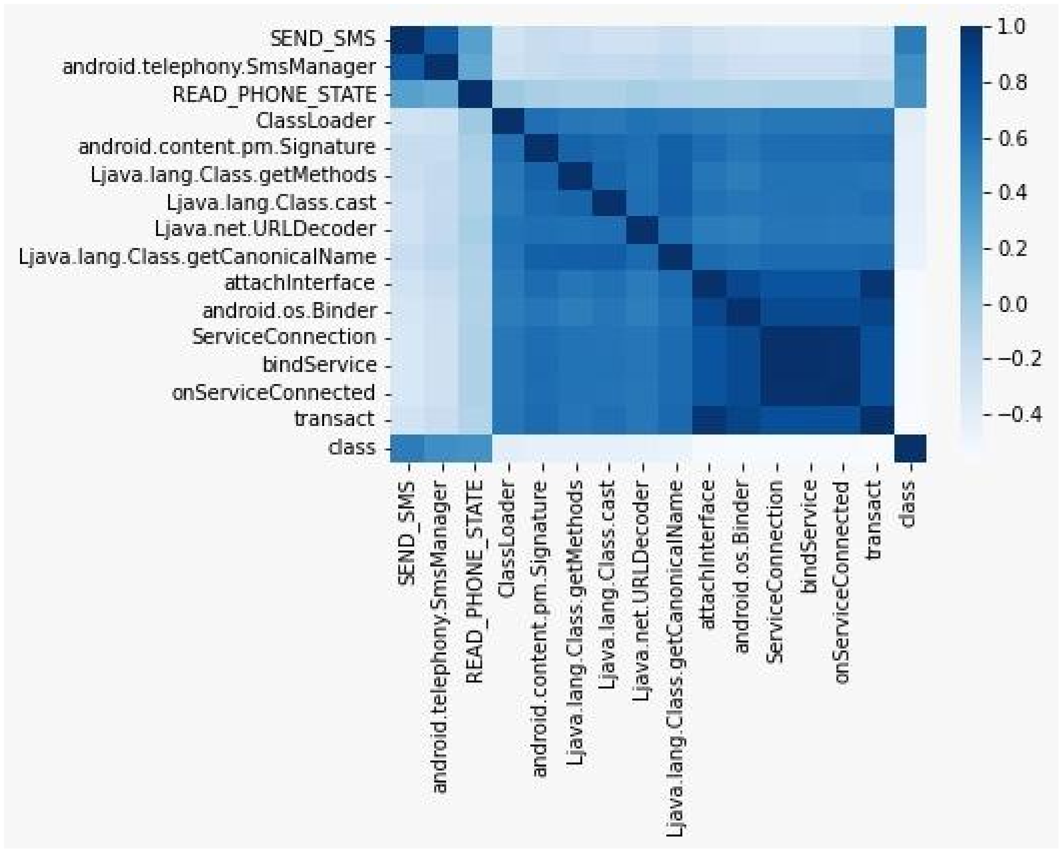


Fig 5.1 Feature analysis

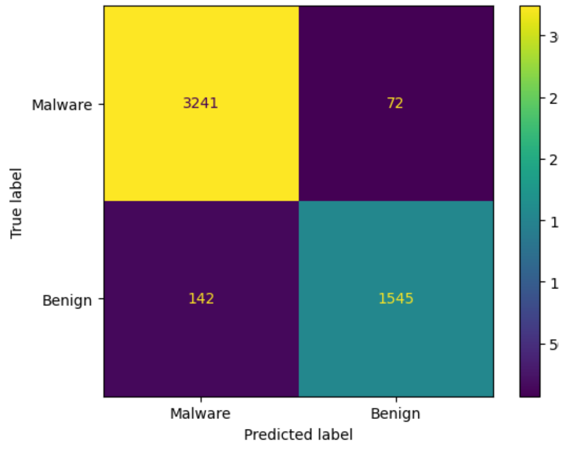
Various ML classification models were being trained on Drebin CSV format dataset with 70% used as training and 30% used as testing.

ALGORITHM for MALWARE DETECTION SYSTEM

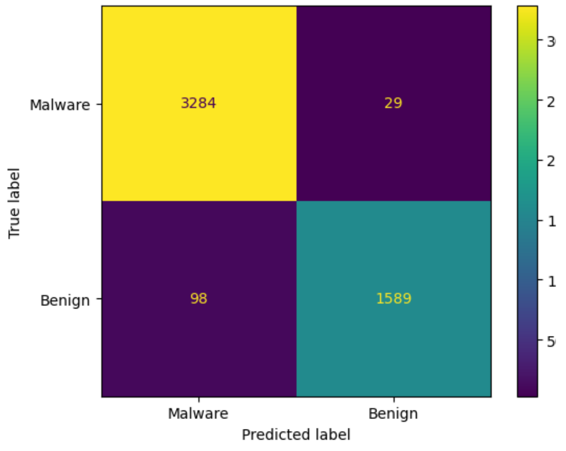
PRE-CONDITION : A input file containing list of features

```
1. function classifyFile(file)
2. Extract list of features from file in featureList
3. Input = []
4. for featureName in columnList:
    a. If featureName in featureList:
    b. Input[featureName] = 1
    c. Else Input[featureName] = 0
5. isMalware = PredictMalwareBenign(input) based on Static features
6. If isMalware = False:
    a. crossChecked = PredictMalwareBenign(input) based on Dynamic
       features
    b. If crossChecked = False: return Benign
    c. Else malwareFamily = predictFamily(input) return malwareFamily
7. Else:
    a. Else malwareFamily = predictFamily(input) return malwareFamily
```

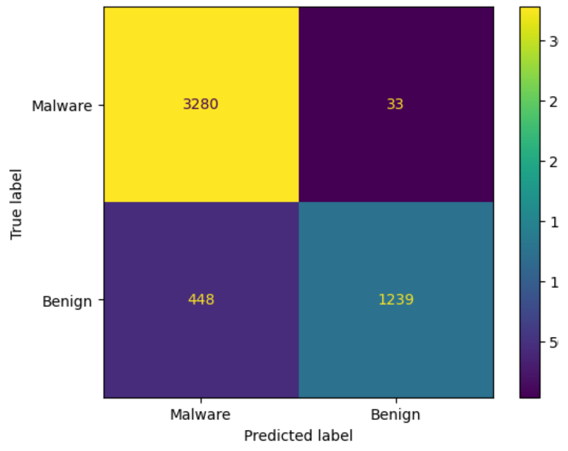
1.**Static Feature based Model :** As discussed in above algorithm, the first level of classifying model is based on static features like PERMISSIONS, INTENTS etc. First file is given as input and based on features contained in file, input feature seat is generated as stated in algorithm above, then this input vector is given to model for training and then prediction purpose. Following confusion matrices were obtained for different ML models like SVM, Random Forest etc.



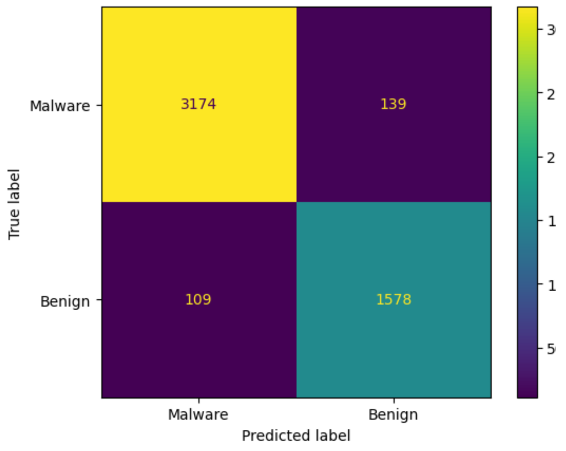
(a) Support Vector Machine (SVM)



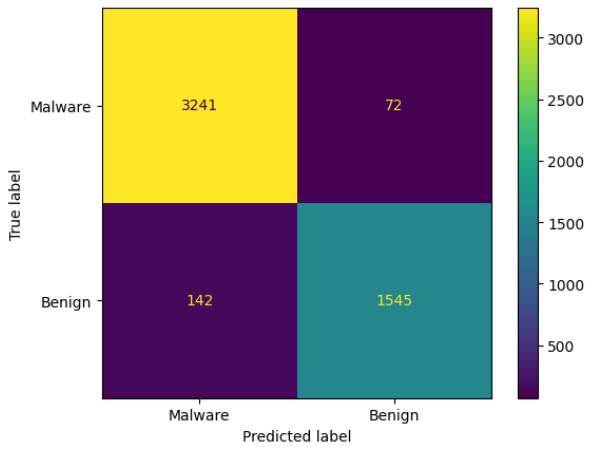
(b) Random Forest Classifier



(c) Naive Bayes



(d) Decision Tree



(e) KNN

Fig 5.2 Confusion Matrices of Static Feature based Models

Above given confusion matrices states that SVM had 3241 True Negatives i.e Prediction and Actual prediction both are Malware (TN), 72 False Positives (FP) indicates that True label is Malware but prediction is Benign. Similarly, 142 False Negatives (FN) indicates that Actual Label is Benign and Prediction is Malware while 1545 True Positives (TP) shows that both actual and predicted value is Benign and similarly for Random Forest, Naive Bayes, Decision Tree and KNN.

Table 5.1 Static Feature Based Models Metrics

Model	**TN**	**FP**	**FN**	**TP**	**Accuracy**	**Precision**	**Recall**	**F1**
SVM	3241	72	142	1545	0.9572	0.9554	0.9158	0.9352
Random Forest	3284	29	98	1589	0.9746	0.9820	0.9419	0.9615
Naive Bayes	3280	33	448	1239	0.9038	0.9740	0.7344	0.8374
Decision Tree	3174	139	109	1578	0.9504	0.919	0.9353	0.9271
KNN	3230	72	153	1545	0.955	0.9554	0.9098	0.9321

In above table, Accuracy, Precision, Recall and F1 is being calculated from confusion matrices and it can be concluded that Random Forest classifier outperformed all other classifiers with accuracy 97.4%, Precision 98.2%, Recall as 94.19% and F1 score as 96.1% So it can be selected for Static feature based classification at Level 1

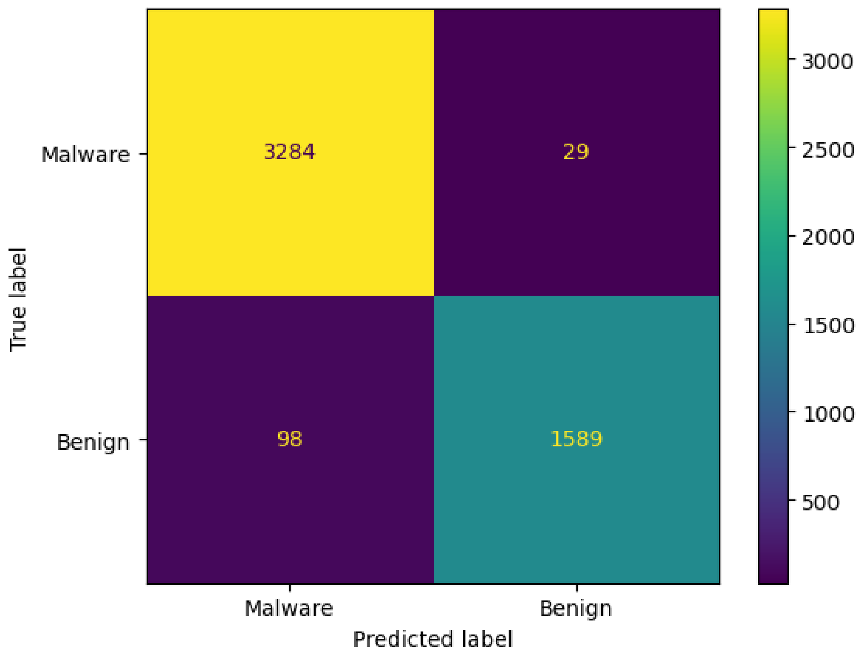
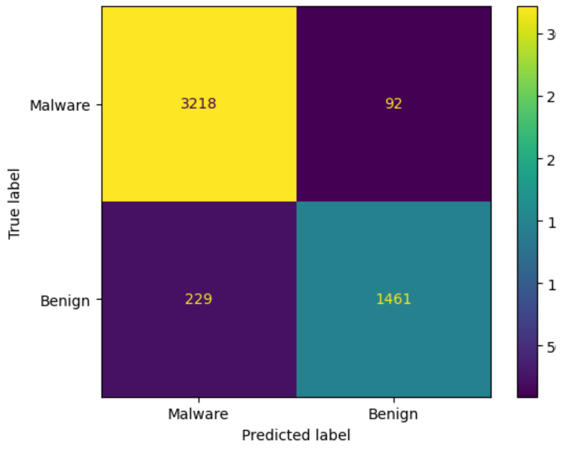


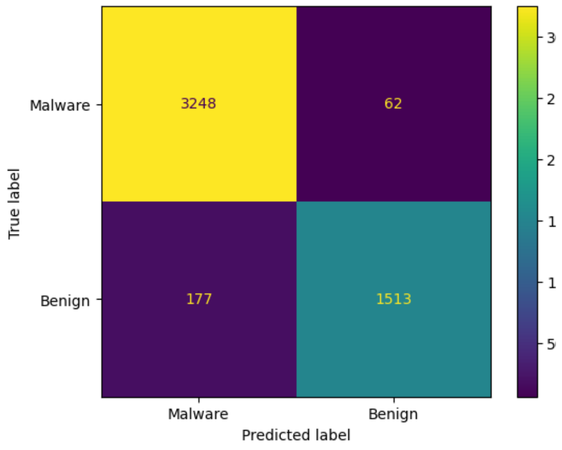
Fig 5.3 Confusion Matrix for Random Forest Static Feature based Model

Above confusion matrix of Random Forest Static Feature based classifier states that 3284 True Negatives i.e Prediction and Actual prediction both are Malware (TN), 29 False Positives (FP) indicates that True label is Malware but prediction is Benign. Similarly, 98 False Negatives (FN) indicates that Actual Label is Benign and Prediction is Malware while 1589 True Positives (TP) shows that both actual and predicted value is Benign.

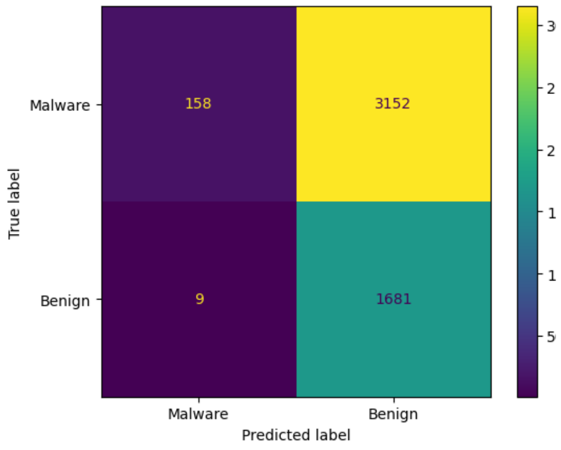
2. **Dynamic Feature based Model :** This is Level 2 classifier which is used to cross check whether the application is Malware or Benign. First file is given as input and based on features contained in file, input feature seat which is dynamic in nature is generated as stated in algorithm above, then this input vector is given to model for training and then prediction purpose. Following confusion matrices were obtained for different ML models like SVM, Random Forest etc.



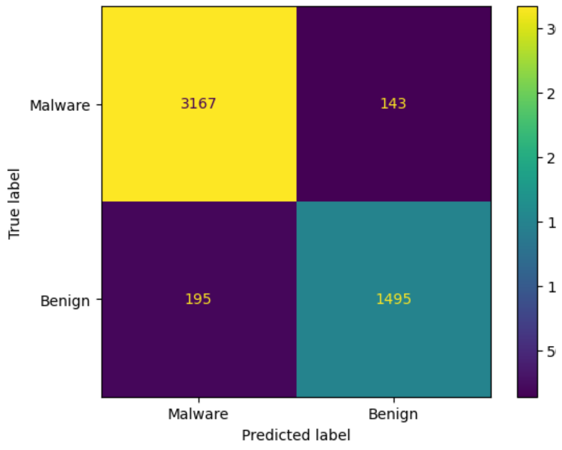
(a) Support Vector Machine



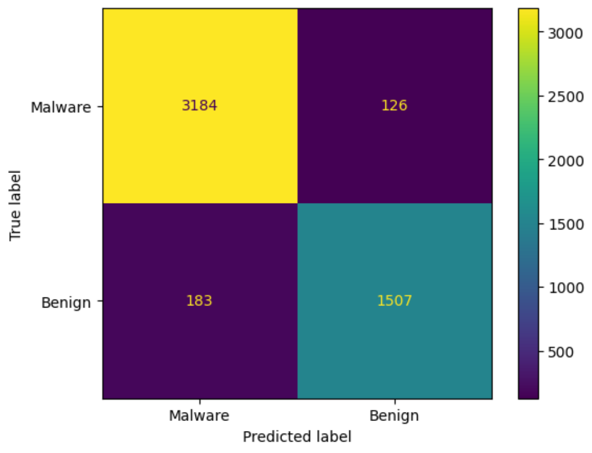
(b) Random Forest Classifier



(c) Naive Bayes



(d) Decision Tree



(e) KNN

Fig 5.4 Confusion Matrices of Dynamic Feature based Models

Above confusion matrices states that (a) SVM had 3218 True Negatives i.e Prediction and Actual prediction both are Malware (TN), 92 False Positives (FP) indicates that True label is Malware but prediction is Benign. Similarly, 229 False Negatives (FN) indicates that Actual Label is Benign and Prediction is Malware while 1461 True Positives (TP) shows that both actual and predicted value is Benign and similarly for Random Forest, Naive Bayes, Decision Tree and KNN.

Table 5.2 Dynamic Features Based Model Metrics

Model	TN	FP	FN	TP	Accuracy	Precision	Recall	F1
SVM	3218	92	229	1461	0.9358	0.9407	0.8644	0.901
Random Forest	3248	62	177	1513	0.9522	0.9606	0.8952	0.9267
Naive Bayes	158	3152	9	1681	0.3678	0.3478	0.9946	0.5154
Decision Tree	3167	143	195	1495	0.9324	0.9126	0.8846	0.8984
KNN	3184	126	183	1507	0.9382	0.9228	0.8917	0.9070

In Table 5.2, Accuracy, Precision, Recall and F1 is being calculated from confusion matrices and it can be concluded that Random Forest classifier outperformed all other classifiers with accuracy 95.2%, Precision 96%, Recall as 89.5% and F1 score as 92.6% So it can be selected for Dynamic feature based classification at Level 2.

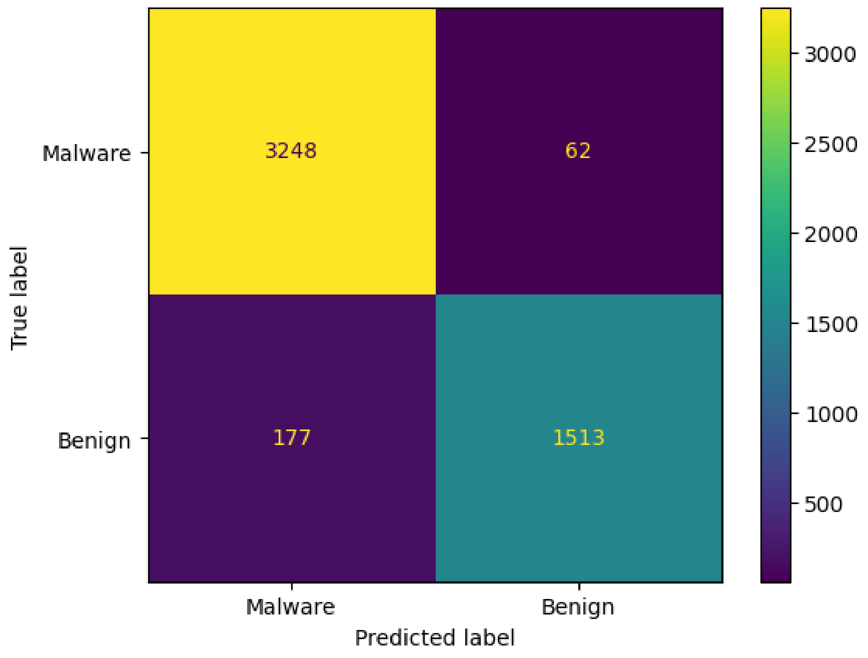


Fig 5.5 Confusion Matrix for Random Forest Dynamic Feature based Model

Above confusion matrix of Random Forest classifiers states that 3248 True Negatives i.e Prediction and Actual prediction both are Malware (TN), 62 False Positives (FP) indicates that True label is Malware but prediction is Benign. Similarly, 177 False Negatives (FN) indicates that Actual Label is Benign and Prediction is Malware while 1513 True Positives (TP) shows that both actual and predicted value is Benign.

3. Combined (Static and Dynamic) Feature based Family Classification Model :

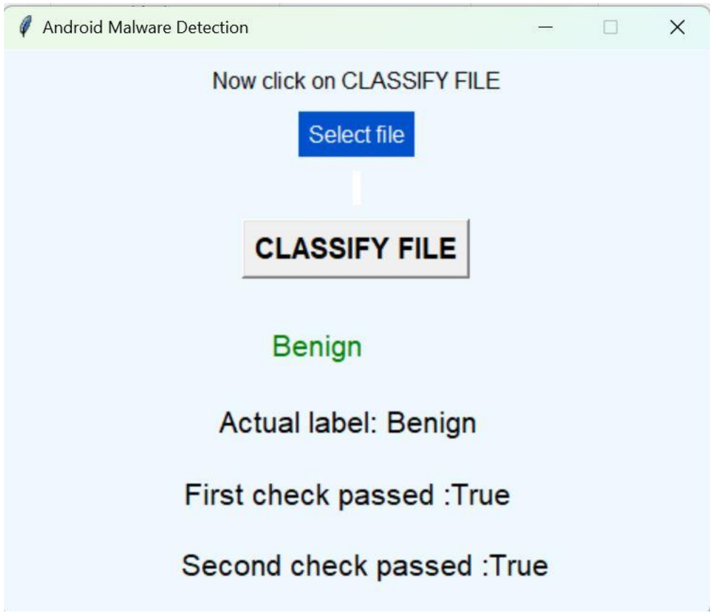
This final level model is used to assign applications to a particular malware family. As discussed in Table 5.3, SVM gave around 87.8% accuracy, Naive Bayes Classifier gave 56.8%, Decision Tree gave 89.86% and KNN gave 89.3% accuracy. Random Forest Classifier outperformed all other classifiers by giving accuracy of around 93.5%

Table 5.3 Combined (Static and Dynamic) Feature based Family Model Metrics

Model	Accuracy	Precision	Recall	F1
SVM	0.8782	0.887	0.854	0.838
Random Forest	0.9358	0.924	0.9101	0.9344
Naive Bayes	0.568	0.5382	0.5027	0.5827
Decision Tree	0.898	0.9021	0.8738	0.8920
KNN	0.8937	0.8357	0.8213	0.857

In Table 5.3, Accuracy, Precision, Recall and F1 is being calculated and it can be concluded that Random Forest classifier outperformed all other classifiers with accuracy 93.58%, Precision 92.4%, Recall as 91% and F1 score as 93.44% So it can be selected for Combined (Static and Dynamic) Feature based Family Classification.

Fig 5.6 shows the output window of the User Interface of the application. Android Manifest like file needs to be uploaded to application then, it is classified as Benign. As the application is Benign, it is passed to classifier 2. Then again it is classified as Benign. So, at both levels it is classified as a Benign application.


Android Malware Detection

Now click on CLASSIFY FILE

Benign

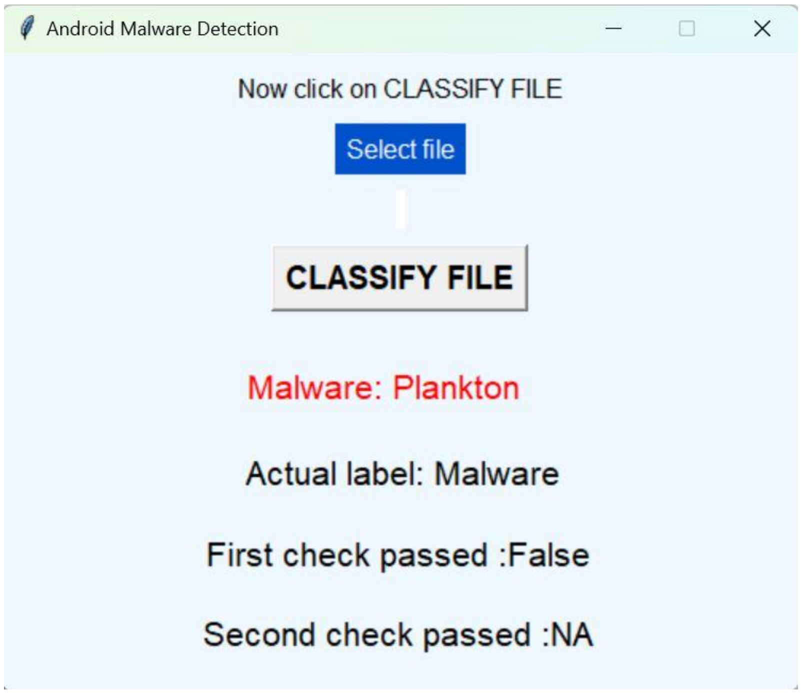
Actual label: Benign

First check passed :True

Second check passed :True

Fig 5.6 Malware Detection System Output 1

Fig 5.7 shows the output window of the User Interface of the application. Android Manifest like file needs to be uploaded to application then, it is classified as Malware. As the application is Malware, it is passed to the Family classifier . Then Malware family to which application belongs is predicted.


Android Malware Detection

Now click on CLASSIFY FILE

Malware: Plankton

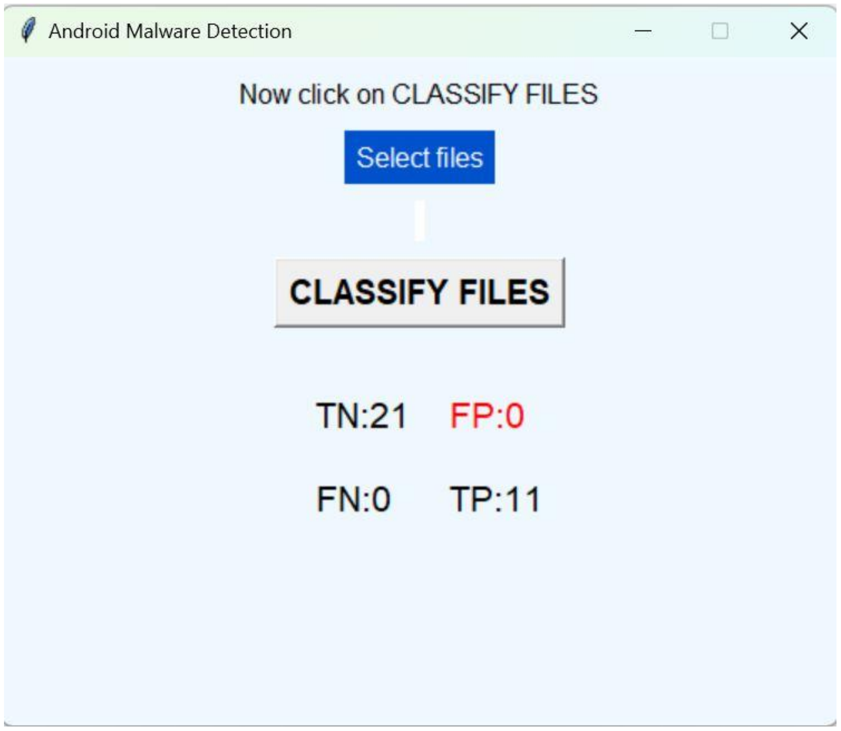
Actual label: Malware

First check passed :False

Second check passed :NA

Fig 5.7 Malware Detection System Output 2

Fig 5.8 shows the output window of the User Interface of the application. In this case,multiple Manifest-like files can be uploaded to the system. As we can infer from the figure, True Negative(TN) indicates applications which are actual Malware and also classified as Malware. Here False Positive (FP) is marked with red because this case is very much dangerous as application is Malware but it is classified as Benign.This needs to be minimized in every case.

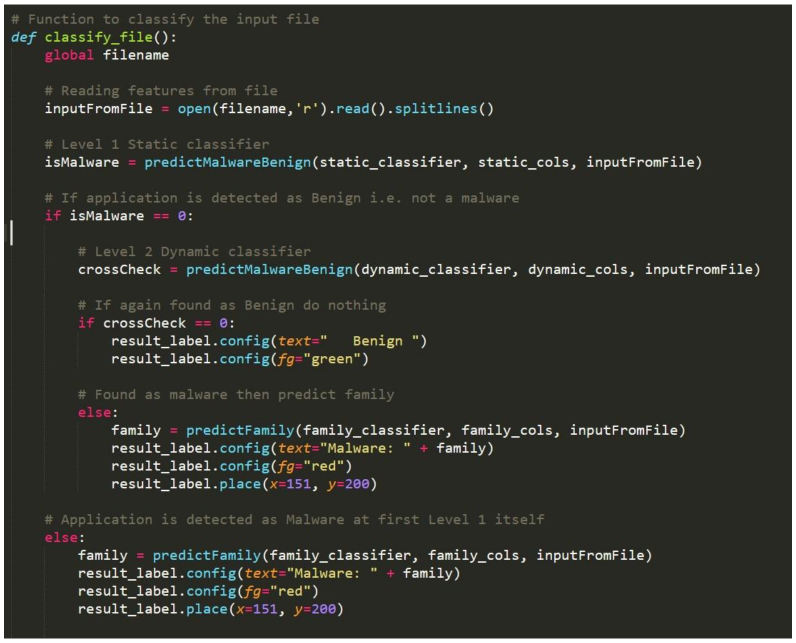


```
TN:21    FP:0

FN:0     TP:11
```

Fig 5.8 Malware Detection System Output 3

Fig 5.9 shows the code for the classification purpose of input.



```python
# Function to classify the input file
def classify_file():
    global filename

    # Reading features from file
    inputFromFile = open(filename,'r').read().splitlines()

    # Level 1 Static classifier
    isMalware = predictMalwareBenign(static_classifier, static_cols, inputFromFile)

    # If application is detected as Benign i.e. not a malware
    if isMalware == 0:

        # Level 2 Dynamic classifier
        crossCheck = predictMalwareBenign(dynamic_classifier, dynamic_cols, inputFromFile)

        # If again found as Benign do nothing
        if crossCheck == 0:
            result_label.config(text="   Benign ")
            result_label.config(fg="green")

        # Found as malware then predict family
        else:
            family = predictFamily(family_classifier, family_cols, inputFromFile)
            result_label.config(text="Malware: " + family)
            result_label.config(fg="red")
            result_label.place(x=151, y=200)

    # Application is detected as Malware at first Level 1 itself
    else:
        family = predictFamily(family_classifier, family_cols, inputFromFile)
        result_label.config(text="Malware: " + family)
        result_label.config(fg="red")
        result_label.place(x=151, y=200)
```

Fig 5.9 Code For The Classification of Input

CHAPTER 6

CONCLUSION AND FUTURE SCOPE

Conclusion

In conclusion, cascading machine learning has shown great potential in detecting Android malware due to its ability to analyze vast amounts of data and detect patterns that may be indicative of malicious behavior. By using techniques such as feature engineering, classification, machine learning algorithms can accurately identify malicious apps and protect users from potential threats. Its continued development will be crucial in ensuring the security of mobile devices in the future.

Future Scope

There is scope for further research in the field of Android malware detection using machine learning techniques. One potential avenue is to explore the use of deep learning models, such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs), for better feature extraction and classification accuracy. Another area of research could be to investigate the effectiveness of transfer learning in the context of Android malware detection, where models trained on one dataset can be fine-tuned on another.

Moreover, the performance of our approach can be further improved by using more sophisticated feature extraction techniques and by incorporating additional sources of information, such as network traffic and user behavior. Additionally, our approach can be extended to other platforms, such as iOS, to provide comprehensive security solutions for smart devices.

REFERENCES

[1] Saleh M. Shehata, Ahmed H. El Fiky, Mohamed Sh. Torky, Tamer H. Farag, Nesrin Ahmed ,Abbas.Android “**Malware Prevention on Permission Based (2017)**” International Journal of Applied Engineering Research 2020.

[2] Rahman Ali , Asmat Ali, Farkhund Iqbal, Mohammed Hussain and Farhan Ullah“ **Deep Learning Methods for Malware and Intrusion Detection**“ 2022.

[3] Hasan Alkahtani and Theyazn H. H. Aldhyani , “**Artificial Intelligence Algorithms for Android Malware Detection** “ MDPI/Sensors journal 2022.

[4] Jiayin Feng , Limin Shen Zhen Chen , Yuying Wang , Hui Li “ **A Two-Layer Deep Learning Method for Android Malware Detection Using Network Traffic** ” IEEE Volume 8 , 2020.

[5] Laraib U. Memon, Narmeen Z Bawany , Jawwad A shamsi ,” **A comparison of machine learning techniques for android malware detection using apache spark** ” Journal of Engineering Science and Technology ,Volume 14, 2019

[6] Fausto Fasano , Fabio Martinelli, Francesco Mercaldo, “ **Cascade learning of machine learning techniques for android malware families Detection through quality and Android metrics** ” .IJCNN 2019. International Joint Conference on Neural Networks , 2019.

[7] Dragoş Gavriluţ , Mihai Cimpoes, Dan Anton, Liviu Ciortuz “ **Malware Detection Using Machine learning** ” International Multiconference on Computer Science and Information Technology (IMCSIT) Volume 4, 2009

[8] Daniel Arp, Michael Spreitzenbarth, Malte Huebner, Hugo Gascon, and Konrad Rieck "**Drebin: Efficient and Explainable Detection of Android Malware in Your Pocket**",21th Annual Network and Distributed System Security Symposium (NDSS), February 2014.

[9] Zinal D. Patel " **Malware Detection in Android Operating System** " International Conference on Advances in Computing, Communication Control and Networking (ICACCCN) , 2018 .

[10] Reference website : https://www.mdpi.com/2079-9292/10/13/1606

[11] DREBIN DATASET : https://www.sec.tu-bs.de/~danarp/drebin/