A

Major Project

On

MALWARE DETECTION: A FRAMEWORK FOR REVERSE ENGINEERED ANDROID APPLICATIONS THROUGH MACHINE LEARNING ALGORITHMS

(Submitted in partial fulfillment of the requirements for the award of Degree)

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE AND ENGINEERING

By

MUGALA YESHWANTH REDDY (207R1A05A1)

VANGALA SHRUTHI (207R1A05B7)

SUNKARA NIKHIL (207R1A05A6)

Under the Guidance of

Dr. B. LAXMAIAH

(Associate Professor)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CMR TECHNICAL CAMPUS

UGC AUTONOMOUS

(Accredited by NAAC, NBA, Permanently Affiliated to JNTUH, Approved by AICTE, New Delhi) Recognized Under Section 2(f) & 12(B) of the UGCAct.1956, Kandlakoya (V), Medchal Road, Hyderabad-501401.

2020-2024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the project entitled **"MALWARE DETECTION: A FRAMEWORK FOR REVERSE ENGINEERED ANDROID APPLICATIONS THROUGH MACHINE LEARNING ALGORITHMS"** being submitted by **MUGALA YESHWANTH REDDY (207R1A05A1), VANGALA SHRUTHI (207R1A05B7) & SUNKARA NIKHIL (207R1A05A6)** in partial fulfillment of the requirements for the award of the degree of B.Tech in Computer Science and Engineering to the Jawaharlal Nehru Technological University Hyderabad, is a record of bonafide work carried out by them under our guidance and supervision during the year 2023-24.

The results embodied in this thesis have not been submitted to any other University or Institute for the award of any degree or diploma.

Dr. B. Laxmaiah
(Associate Professor)
INTERNAL GUIDE

Dr. A. Raji Reddy
DIRECTOR

Dr. K. Srujan Raju
HOD

EXTERNAL EXAMINER

Submitted for viva voice Examination held on _____

ACKNOWLEDGEMENT

Apart from the efforts of us, the success of any project depends largely on the encouragement and guidelines 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 take this opportunity to express my profound gratitude and deep regard to my guide **Dr. B. Laxmaiah**, Associate Professor for his exemplary guidance, monitoring and constant encouragement throughout the project work. The blessing, help and guidance given by him shall carry us a long way in the journey of life on which we are about to embark.

We also take this opportunity to express a deep sense of gratitude to the Project Committee Review (PRC) **G. Vinesh Shankar, Dr. J. Narsimha Rao, Ms. Shilpa, Dr. K. Maheshwari & Saba Sultana** for their cordial support, valuable information and guidance, which helped us in completing this task through various stages.

We are also thankful to **Dr. K. Srujan Raju,** Head, Department of Computer Science and Engineering for providing encouragement and support for completing this project successfully.

We are obliged to **Dr. A. Raji Reddy,** Director for being cooperative throughout the course of this project. We also express our sincere gratitude to Sri. **Ch. Gopal Reddy,** Chairman for providing excellent infrastructure and a nice atmosphere throughout the course of this project.

The guidance and support received from all the members of **CMR Technical Campus** who contributed to the completion of the project. We are grateful for their constant support and help.

Finally, we would like to take this opportunity to thank our family for their constant encouragement, without which this assignment would not be completed. We sincerely acknowledge and thank all those who gave support directly and indirectly in the completion of this project.

MUGALA YESHWANTH REDDY (207R1A05A1)

VANGALA SHRUTHI (207R1A05B7)

SUNKARA NIKHIL (207R1A05A6)

ABSTRACT

Today, Android is one of the most used operating systems in smartphone technology. This is the main reason, Android has become the favorite target for hackers and attackers. Malicious codes are being embedded in Android applications in such a sophisticated manner that detecting and identifying an application as a malware has become the toughest job for security providers. In terms of ingenuity and cognition, Android malware has progressed to the point where they're more impervious to conventional detection techniques. Approaches based on machine learning have emerged as a much more effective way to tackle the intricacy and originality of developing Android threats. They function by first identifying current patterns of malware activity and then using this information to distinguish between identified threats and unidentified threats with unknown behavior. This research paper uses Reverse Engineered Android applications' features and Machine Learning algorithms to find vulnerabilities present in Smartphone applications. Our contribution is twofold. Firstly, we propose a model that incorporates more innovative static feature sets with the largest current datasets of malware samples than conventional methods. Secondly, we have used ensemble learning with machine learning algorithms such as AdaBoost, SVM, etc. to improve our model's performance. Our experimental results and findings exhibit 96.24% accuracy to detect extracted malware from Android applications, with a 0.3 False Positive Rate (FPR). The proposed model incorporates ignored detrimental features such as permissions, intents, API calls, and so on, trained by feeding a solitary arbitrary feature, extracted by reverse engineering as an input to the machine.

LIST OF FIGURES/TABLES

LIST OF SCREENSHOTS

TABLE OF CONTENTS

TABLE OF CONTENTS

1. INTRODUCTION

1.INTRODUCTION

1.1 PROJECT SCOPE

The project aims to enhance Android malware detection by utilizing machine learning, specifically through static analysis of applications. It focuses on developing a robust model trained on latest Android API levels and time-aware malware samples. The project addresses the challenges posed by unnecessary permissions in Android apps, emphasizing automation for efficient static analysis and feature extraction. The scope includes implementing advanced machine learning algorithms to improve the security of Android devices against the rising threat of malware.

1.2 PROJECT PURPOSE

The purpose of this project is to strengthen the security of Android devices against the escalating threat of malware. By leveraging machine learning and advanced static analysis techniques, the project aims to develop a robust model capable of accurately detecting malicious applications. The focus is on addressing the risks associated with unnecessary permissions in Android apps, providing a proactive solution to enhance user protection. The incorporation of the latest Android API levels and time-aware malware samples reflects the project's commitment to staying ahead of evolving threats.

1.3 PROJECT FEATURES

This project introduces a set of key features to bolster Android malware detection. It incorporates a novel subset of features derived from advanced static analysis. The model is trained on the latest Android API levels, utilizing time-aware malware samples to stay abreast of evolving threats. With a primary focus on unnecessary permissions, the project employs machine learning algorithms such as Support Vector Machines (SVM) and ensemble techniques like Adaboost to achieve a remarkable detection accuracy. Automation plays a pivotal role, streamlining the process of static analysis and feature extraction.

2. SYSTEM ANALYSIS

2. SYSTEM ANALYSIS

2.1 PROBLEM DEFINITION

This project defines a comprehensive approach to enhance the security of Android devices by mitigating the threat of malware. Utilizing machine learning and advanced static analysis, the project introduces a novel subset of features to accurately detect malicious applications. The model trained on the latest Android API levels and time-aware malware samples, emphasizes proactive defense against evolving threats. By focusing on unnecessary permissions and employing algorithms like Support Vector Machines and Adaboost, the project achieves a robust detection accuracy. Automation is a key element, streamlining static analysis and feature extraction for efficient and effective malware detection in Android applications.

2.2 EXISTING SYSTEM

The methods proposed in this related work contribute to key aspects and a higher predictive rate for malware detection. Certain research has focused on increasing accuracy, while others have focused on providing a larger dataset, some have been implemented by employing various feature sets, and many studies have combined all of these to improve detection rate efficiency. In the authors offer a system for detecting Android malware apps to aid in the organization of the Android Market. The proposed framework aims to provide a machine learning-based malware detection system for Android to detect malware apps and improve phone users' safety and privacy. This system monitors different permission-based characteristics and events acquired from Android apps and examines these features employing machine learning classifiers to determine if the program is goodware or malicious. The authors present a unique Android malware detection approach dubbed Permission- based Malware Detection Systems (PMDS) based on a study of 2950 samples of benign and malicious Android applications. In PMDS, requested permissions are viewed as behavioral markers, and a machine learning model is built on those indicators to detect new potentially dangerous behavior in unknown apps depending on the mix of rights they require.

This uses two datasets with collectively 700 malware samples and 160 features. Both datasets achieved approximately 91% accuracy with Random Forest (RF) Algorithm. Examines 5,560 malware samples, detecting 94% of the malware with minimal false alarms, where the reasons supplied for each detection disclose key features of the identified malware. Researchers demonstrated the consistency of the model in attaining maximum classification performance and better accuracy compared to two state-of-the-art peer methods that represent both static and dynamic methodologies over for nine years through three interrelated assessments with satisfactory malware samples from different sources. Model continuously achieved 97% F1- measure accuracy for identifying applications or categorizing malware.

2.2.1 DISADVANTAGES OF EXISTING SYSTEM

- The system is not implemented MACHINE LEARNING ALGORITHM AND ENSEMBLE LEARNING.
- The system is not implemented Reverse Engineered Applications characteristics.
- Time-consuming, costly, and prone to errors.

2.3 PROPOSED SYSTEM

We present a novel subset of features for static detection of Android malware, which consists of seven additional selected feature sets that are using around 56000 features from these categories. On a collection of more than 500k benign and malicious Android applications and the highest malware sample set than any state-of-the-art approach, we assess their stability. The results obtain a detection increase in accuracy to 96.24 % with 0.3% false-positives. With the additional features, we have trained six classifier models or machine learning algorithms and also implemented a Boosting ensemble learning approach (AdaBoost) with a Decision Tree based on the binary classification to enhance our prediction rate. Our model is trained on the latest and large time aware samples of malware collected within recent years including the latest Android API level than state-of-the-art approaches.

2.3.1 ADVANTAGES OF THE PROPOSED SYSTEM

- The proposed system chooses the characteristics based on their capability to display all data sets. Enhanced efficiency by reducing the dataset size and the hours wasted on the classification process introduces an effective selection process.
- The system used in this study also incorporates larger feature sets for classification. Although this problem arises in machine learning quite often to some extent choosing the type of model for detection or classification can highly impact the high dimensionality of the data being used.

2.4 FEASIBILITY STUDY

The feasibility of the project is analyzed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

Three key considerations involved in the feasibility analysis are,

- **ECONOMICAL FEASIBILITY**
- **TECHNICAL FEASIBILITY**
- **SOCIAL FEASIBILITY**

2.4.1 ECONOMIC FEASIBILITY

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

2.4.2 TECHNICAL FEASIBILITY

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands on the available technical resources. This will lead to high demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

2.4.3 SOCIAL FEASIBILITY

The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system.

2.5 HARDWARE & SOFTWARE REQUIREMENTS

2.5.1 HARDWARE REQUIREMENTS:

Hardware interfaces specify the logical characteristics of each interface between the software product and the hardware components of the system. For developing the application the following are the Hardware Requirements:

- **System** : Windows.
- **Processor** : Pentium IV or higher
- **Hard Disk** : 40 GB.
- **Ram** : 4 GB.

2.5.2 SOFTWARE REQUIREMENTS:

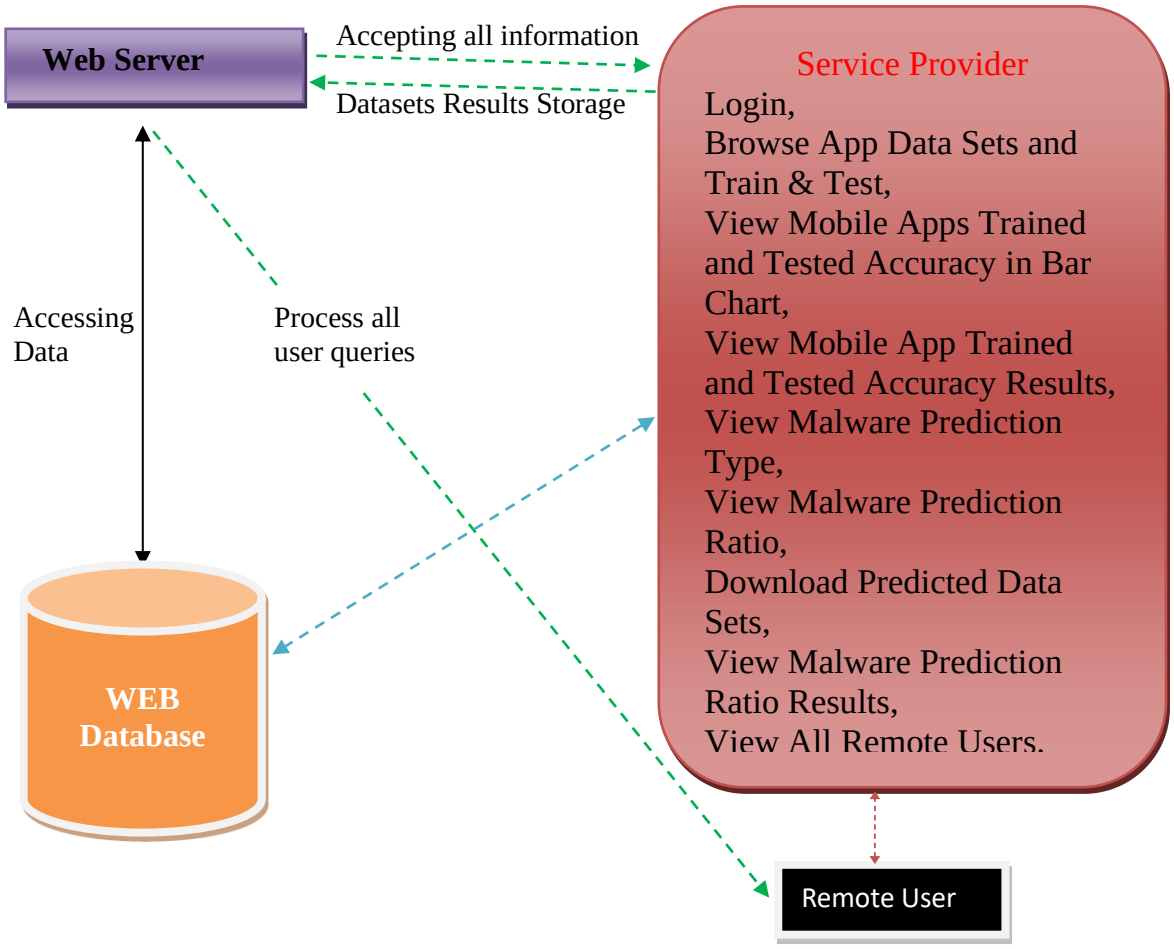
- **Operating system** : Windows 7 or higher.
- **Coding Language** : Python.
- **Framework** : Django.
- **Designing** : Html, CSS, JavaScript.
- **Data Base** : MySQL.

3. ARCHITECTURE

3. ARCHITECTURE

3.1 PROJECT ARCHITECTURE

This project architecture shows the procedure followed for classification, starting from input to final prediction.



SYSTEM ARCHITECTURE OF MALWARE DETECTION: A FRAMEWORK FOR REVERSE ENGINEERED ANDROID APPLICATIONS THROUGH MACHINE

Figure 3.1: Project Architecture of Malware Detection: A Framework for Reverse Engineered Android Applications through Machine Learning Algorithms

3.2 DESCRIPTION

3.2.1 Web Server:

The project utilize a web server for tasks like managing datasets, hosting the machine learning model, and providing an interface for users to interact with the analysis tools and view results.

3.2.2 Web Database:

A web database would store and manage the extensive dataset of Android applications used for training and analysis in the malware detection project.

3.2.3 Service Provider:

The service provider might offer cloud infrastructure and computational resources essential for hosting and processing the machine learning model used in the Android malware detection project.

3.2.4 Remote User:

Remote users would access the Android malware detection system through a web-based interface, enabling interaction with the machine learning model, contributing data, and viewing results from distant locations.

4. DESIGN

4.DESIGN

4.1 USE CASE DIAGRAM

A use case diagram is a graphical depiction of a user's possible interactions with a system. A use case diagram shows various use cases and different types of users the system has. The use cases are represented by either circles or ellipses. The actors are often shown as stick figures.

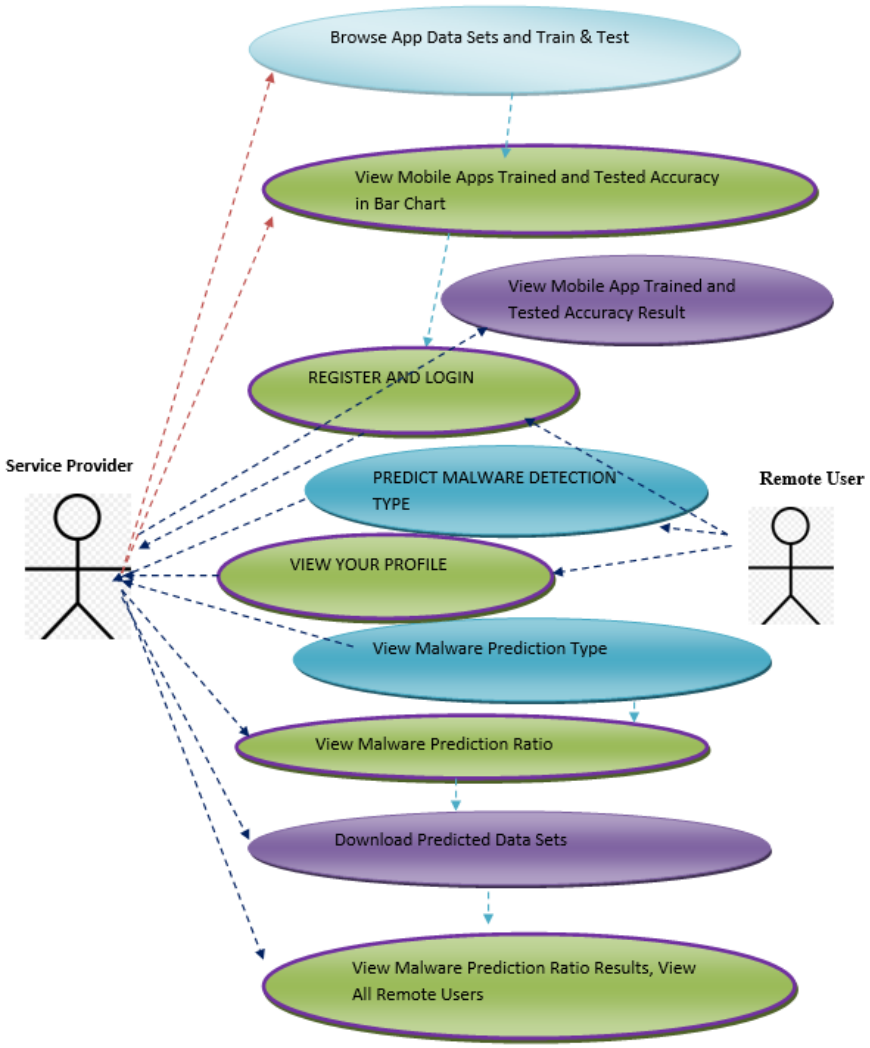


Figure 4.1: Use Case Diagram for Malware Detection: A Framework for Reverse Engineered Android Applications through Machine Learning Algorithms.

4.2 CLASS DIAGRAM

Class diagram is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

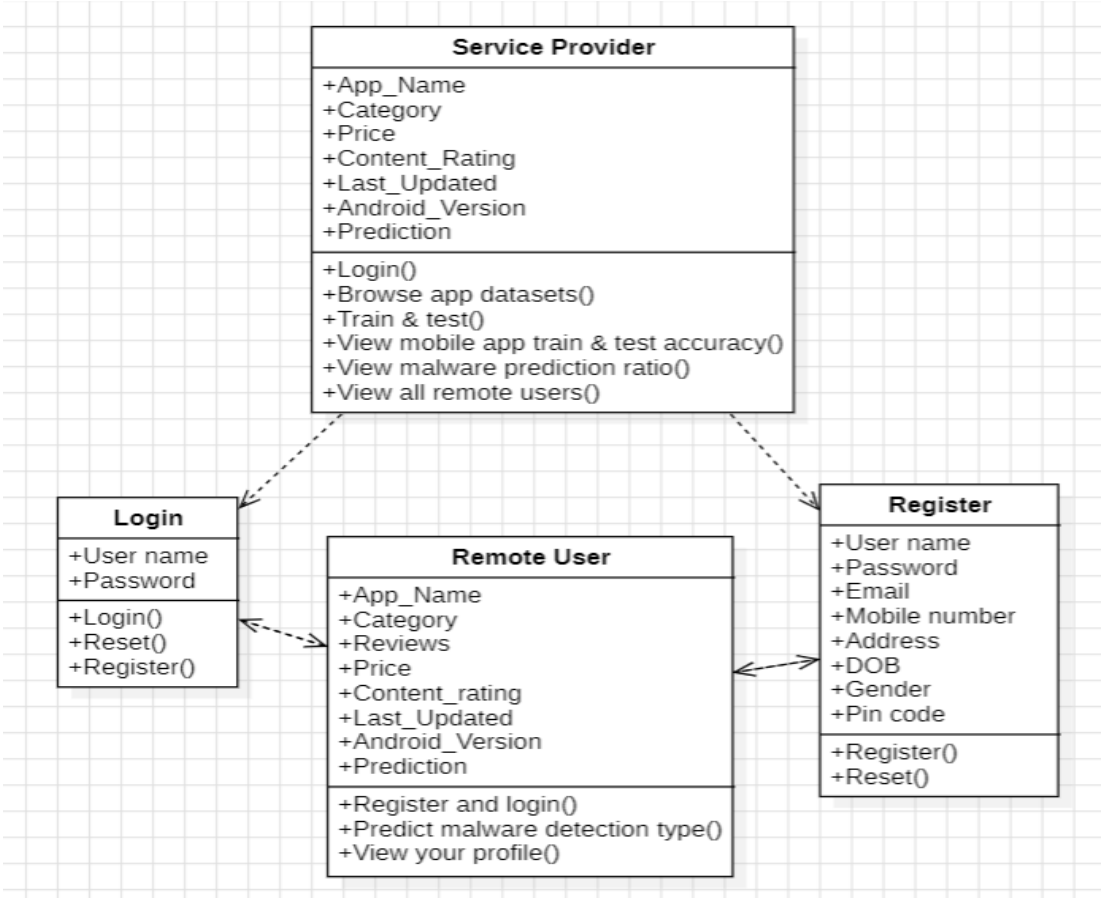


Figure 4.2: Class Diagram for Malware Detection: A Framework for Reverse Engineered Android Applications through Machine Learning Algorithms

4.3 SEQUENCE DIAGRAM

A sequence diagram shows object interactions arranged in time sequence. It depicts the objects involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the logical view of the system under development.

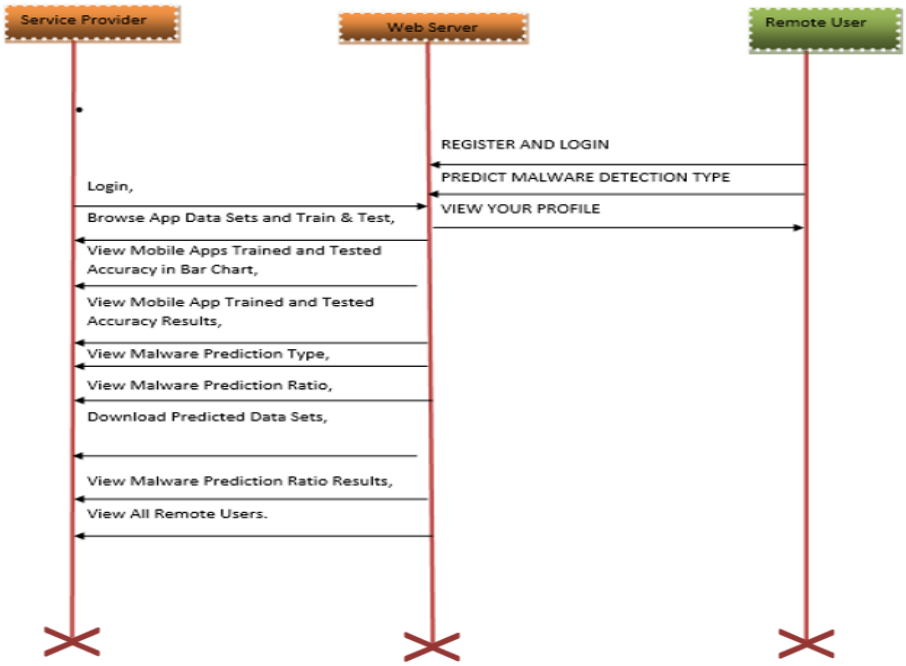
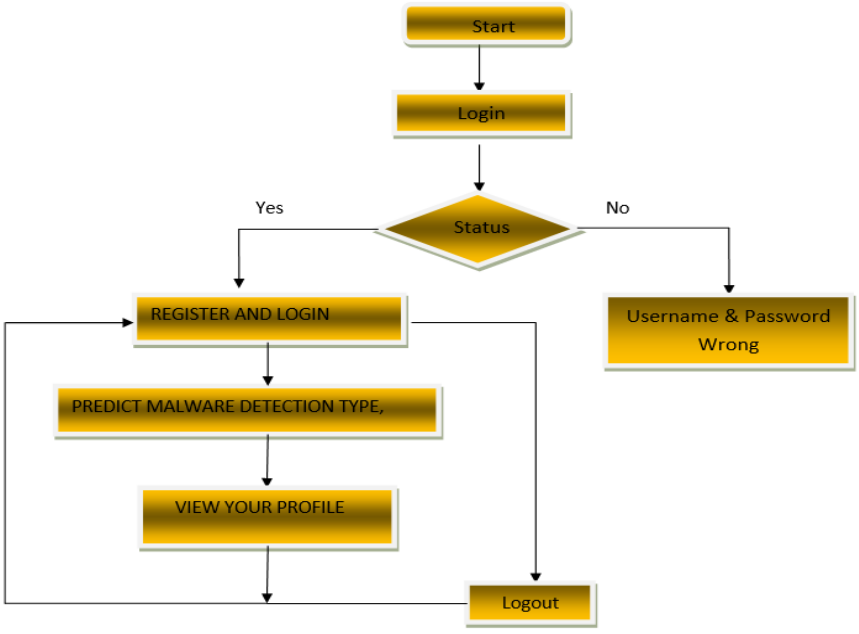


Figure 4.3: Sequence Diagram for Malware Detection: A Framework for Reverse Engineered Android Applications through Machine Learning Algorithms

4.4 ACTIVITY DIAGRAM

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency.

a. Remote User



b. Service Provider

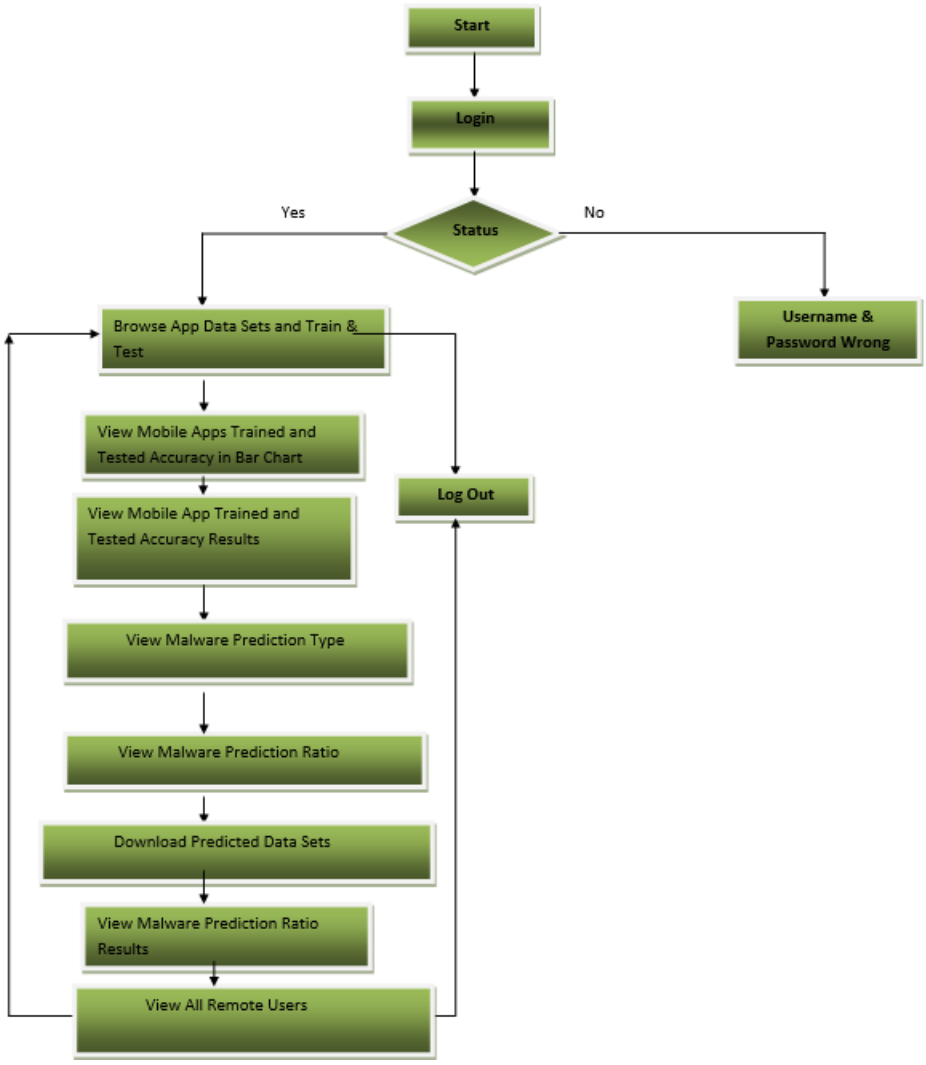


Figure 4.4: Activity diagram for Malware Detection: A Framework for Reverse Engineered Android Applications through Machine Learning Algorithms.

5. IMPLEMENTATION

5. IMPLEMENTATION

5.1 SAMPLE CODE

```
#!/usr/bin/env python
"""Django's command-line utility for administrative tasks."""
import os
import sys


def main():
    """Run administrative tasks."""
    os.environ.setdefault('DJANGO_SETTINGS_MODULE',
'malware_detection.settings')
    try:
        from django.core.management import execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
            "available on your PYTHONPATH environment variable? Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)


if __name__ == '__main__':
    main()
```

```
from django.db.models import  Count, Avg
from django.shortcuts import render, redirect
from django.db.models import Count
from django.db.models import Q

import xlwt
from django.http import HttpResponse

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from sklearn.feature_extraction.text import CountVectorizer

from sklearn.tree import DecisionTreeClassifier
#model selection
from sklearn.metrics import confusion_matrix, accuracy_score, plot_confusion_matrix,
classification_report

# Create your views here.
from Remote_User.models import
ClientRegister_Model,app_type_detection,detection_ratio,detection_accuracy


def serviceproviderlogin(request):
    if request.method  == "POST":
        admin = request.POST.get('username')
        password = request.POST.get('password')
        if admin == "Admin" and password =="Admin":
            detection_accuracy.objects.all().delete()
            return redirect('View_Remote_Users')
    return render(request,'SProvider/serviceproviderlogin.html')

def View_MalwarePrediction_Ratio(request):
    detection_ratio.objects.all().delete()
    ratio = ""
    kword = 'Malware Detected'
    print(kword)
    obj = app_type_detection.objects.all().filter(Q(Prediction=kword))
    obj1 = app_type_detection.objects.all()
    count = obj.count();
    count1 = obj1.count();
    ratio = (count / count1) * 100
    if ratio != 0:
        detection_ratio.objects.create(names=kword, ratio=ratio)
```

```
    ratio1 = ""
    kword1 = 'No Malware'
    print(kword1)
    obj1 = app_type_detection.objects.all().filter(Q(Prediction=kword1))
    obj11 = app_type_detection.objects.all()
    count1 = obj1.count();
    count11 = obj11.count();
    ratio1 = (count1 / count11) * 100
    if ratio1 != 0:
        detection_ratio.objects.create(names=kword1, ratio=ratio1)
    obj = detection_ratio.objects.all()
    return render(request, 'SProvider/View_MalwarePrediction_Ratio.html', {'objs': obj})

def View_Remote_Users(request):
    obj=ClientRegister_Model.objects.all()
    return render(request,'SProvider/View_Remote_Users.html',{'objects':obj})

def ViewTrendings(request):
    topic =
app_type_detection.objects.values('topics').annotate(dcount=Count('topics')).order_by('-
dcount')
    return  render(request,'SProvider/ViewTrendings.html',{'objects':topic})

def charts(request,chart_type):
    chart1 = detection_ratio.objects.values('names').annotate(dcount=Avg('ratio'))
    return render(request,"SProvider/charts.html", {'form':chart1, 'chart_type':chart_type})

def charts1(request,chart_type):
    chart1 = detection_accuracy.objects.values('names').annotate(dcount=Avg('ratio'))
    return render(request,"SProvider/charts1.html", {'form':chart1,
'chart_type':chart_type})

def View_Malware_Prediction_Type(request):
    obj =app_type_detection.objects.all()
    return render(request, 'SProvider/View_Malware_Prediction_Type.html',
{'list_objects': obj})

def likeschart(request,like_chart):
    charts =detection_accuracy.objects.values('names').annotate(dcount=Avg('ratio'))
    return render(request,"SProvider/likeschart.html", {'form':charts,
'like_chart':like_chart})
def Download_Trained_DataSets(request):

    response = HttpResponse(content_type='application/ms-excel')
    # decide file name
```

```
response['Content-Disposition'] = 'attachment; filename="Predicted_Data.xls"'
# creating workbook
wb = xlwt.Workbook(encoding='utf-8')
# adding sheet
ws = wb.add_sheet("sheet1")
# Sheet header, first row
row_num = 0
font_style = xlwt.XFStyle()
# headers are bold
font_style.font.bold = True
# writer = csv.writer(response)
obj = app_type_detection.objects.all()
data = obj  # dummy method to fetch data.
for my_row in data:
    row_num = row_num + 1

    ws.write(row_num, 0, my_row.App_Name, font_style)
    ws.write(row_num, 1, my_row.Category, font_style)
    ws.write(row_num, 2, my_row.Reviews, font_style)
    ws.write(row_num, 3, my_row.Size, font_style)
    ws.write(row_num, 4, my_row.Installs, font_style)
    ws.write(row_num, 5, my_row.Type, font_style)
    ws.write(row_num, 6, my_row.Price, font_style)
    ws.write(row_num, 7, my_row.Content_Rating, font_style)
    ws.write(row_num, 8, my_row.Genres, font_style)
    ws.write(row_num, 9, my_row.Last_Updated, font_style)
    ws.write(row_num, 10, my_row.Current_Ver, font_style)
    ws.write(row_num, 11, my_row.Android_Ver, font_style)
    ws.write(row_num, 12, my_row.Prediction, font_style)

wb.save(response)
return response
```

6. SCREENSHOTS

6. SCREENSHOTS

6.1 User Registration page:

User must register initially by the filling the details such as user name, email id, gender, country, city, password, address, mobile number, state.

FIGURE 6.1: User Registration page

6.2 User Login:

The user must login with username and password.

FIGURE 6.2: User-Login page

6.3 Admin Login:

The admin has to enter the username, password to login. Admin can prepare the data to analyze.

FIGURE 6.3: Admin Login page

6.4 Prediction:

User must enter the app name, total reviews, total installs, app price, genres, current version, app category, size, type, content rating, last updated, android version.

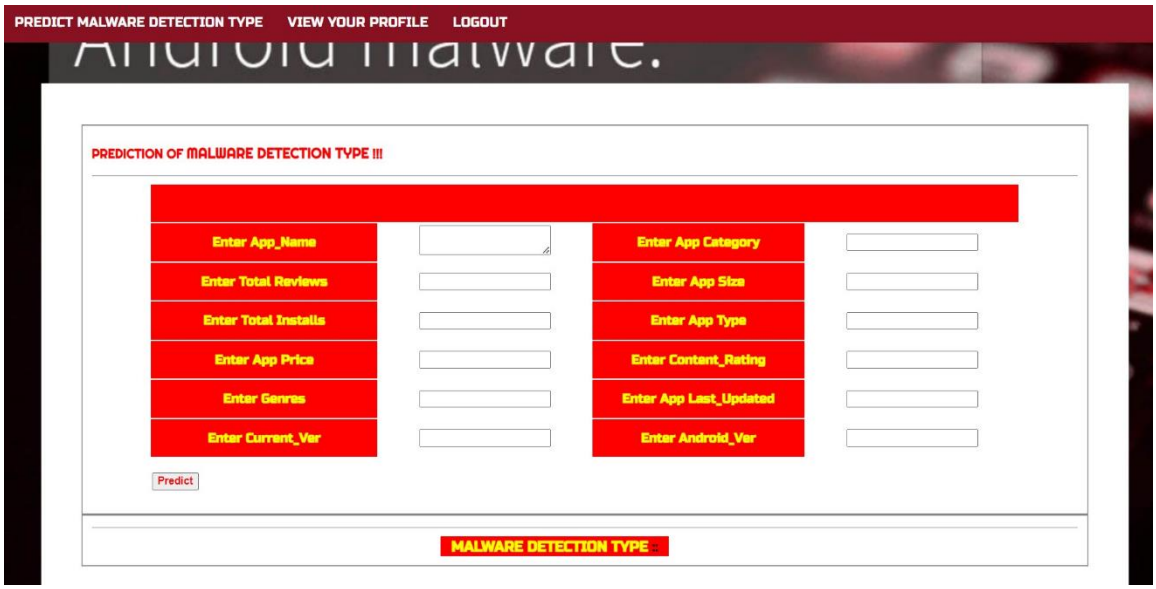


FIGURE 6.4: Prediction

6.5 Analysis of test results:

Analysis of results based on accuracy between the trained and tested results that have been shown below.

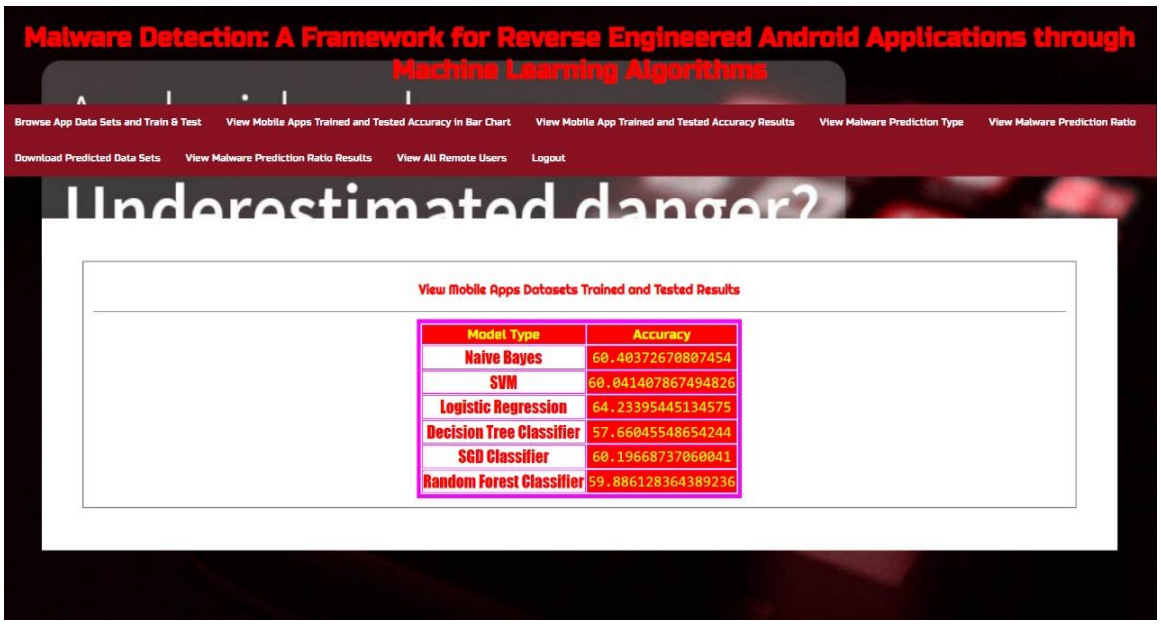


FIGURE 6.5: Analysis of test results

6.6 Analysis using Bar Chart:

After the analysis of the test results, the output can be viewed in the form of charts such as bar chart.

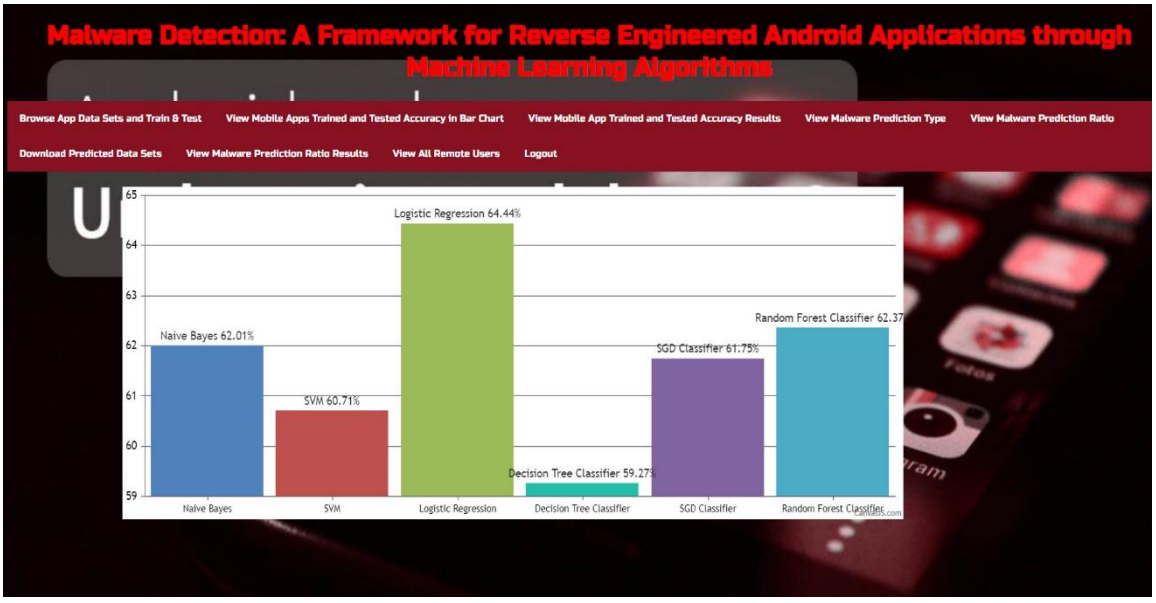


FIGURE 6.6: Analysis using Bar Chart

6.7 Analysis using Line Chart:

After the analysis of the test results, the output can be viewed in the form of charts such as line chart.

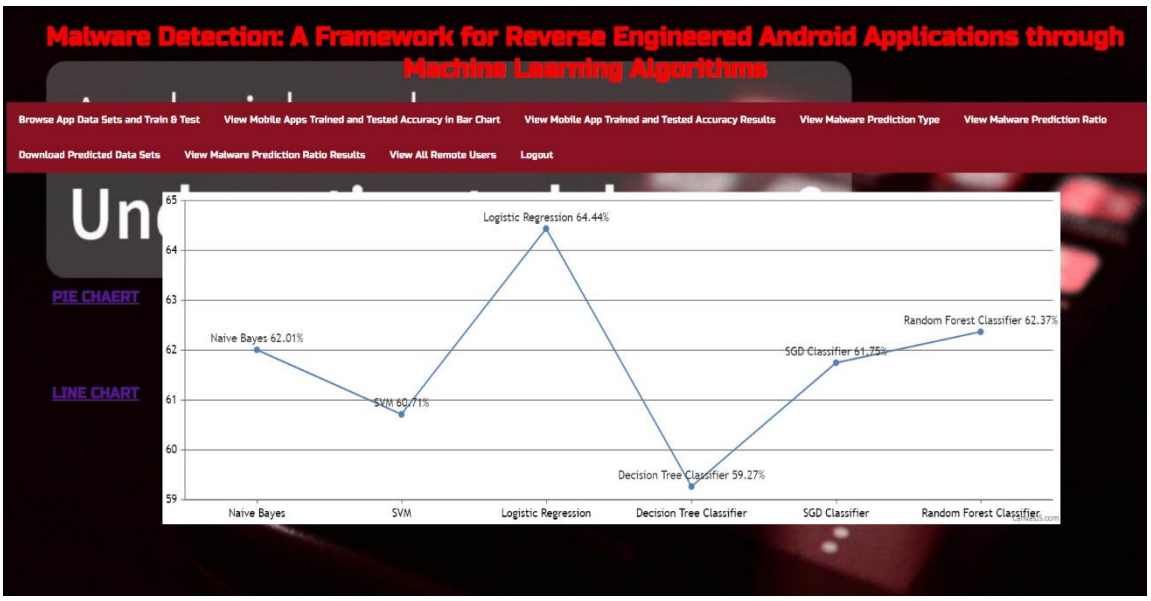


FIGURE 6.7: Analysis using Line Chart

6.8 Analysis using Pie chart:

After the analysis of the test results, the output can be viewed in the form of charts such as pie chart.

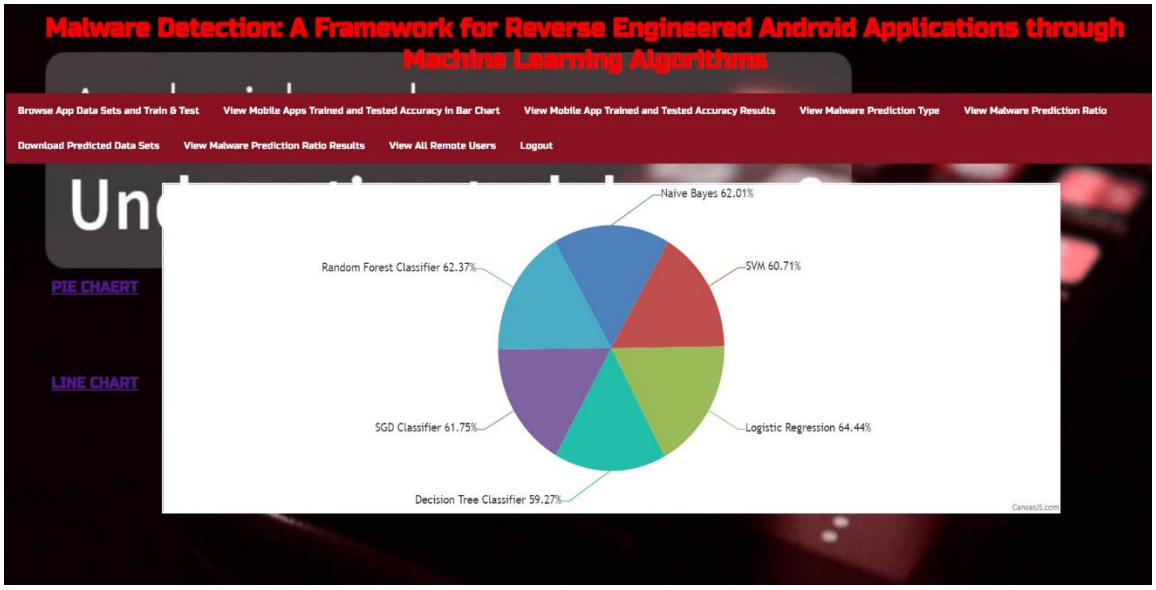


FIGURE 6.8: Analysis using Pie Chart

7. TESTING

7.TESTING

7.1 INTRODUCTION TO TESTING

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, subassemblies, assemblies and/or a finished product. It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of tests. Each test type addresses a specific testing requirement.

7.2 TYPES OF TESTING

7.2.1 UNIT TESTING

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

During this testing, each module is tested individually and the module interfaces are verified for the consistency with design specification. All important processing path are tested for the expected results. All error handling paths are also tested.

7.2.2 INTEGRATION TESTING

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

7.2.3 FUNCTIONAL TESTING

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

Valid Input	: identified classes of valid input must be accepted.
Invalid Input	: identified classes of invalid input must be rejected.
Functions	: identified functions must be exercised.
Output	: identified classes of application outputs must be exercised.
Systems/Procedures	: interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

7.3 TEST CASES

7.3.1 CLASSIFICATION

Test Case ID	Description	Input Data/Scenario	Expected Output/Result	Status (Pass/Fail)
01	Load a benign Android application	Benign Android application	Non-malicious classification	Pass
02	Load a known malware-infected Android application	Malware-infected Android application	Malicious classification	Pass
03	Check model's performance with a large dataset	Large dataset of Android applications	Maintain efficiency and accuracy	Pass
04	Test real-time detection capabilities	Real-time installation of Android application	Quick and accurate real-time detection	Pass
05	Introduce a new variant of known malware	New variant of known malware	Correct adaptation, accurate classification	Pass

8. CONCLUSION

8. CONCLUSION & FUTURE SCOPE

8.1 PROJECT CONCLUSION

In this research, we devised a framework that can detect malicious Android applications. The proposed technique takes into account various elements of machine learning and achieves a 96.24% in identifying malicious Android applications. We first define and pick functions to capture and analyze Android apps' behavior, leveraging reverse application engineering and AndroGuard to extract features into binary vectors and then use python build modules and split shuffle functions to train the model with benign and malicious datasets. Our experimental findings show that our suggested model has a false positive rate of 0.3 with 96% accuracy in the given environment with an enhanced and larger feature and sample sets. The study also discovered that when dealing with classifications and high-dimensional data, ensemble and strong learner algorithms perform comparatively better. The suggested approach is restricted in terms of static analysis, lacks sustainability concerns, and fails to address a key multi collinearity barrier. In the future, we'll consider model resilience in terms of enhanced and dynamic features. The issue of dependent variables or high inter correlation between machine algorithms before employing them is also a promising field.

8.2 FUTURE SCOPE

The future scope of a malware detection project for reverse-engineered Android applications through machine learning involves continuous adaptation to new threats, exploration of deep learning approaches for enhanced accuracy, focus on explainability, incorporation of behavioral analysis, real-time detection capabilities, integration with broader security ecosystems, privacy-preserving techniques, cloud-based solutions for scalability, cross-platform support, user feedback integration, and adherence to regulatory compliance.

9. REFERENCES

9. REFERENCES

9.1 REFERENCES

[1] A. O. Christiana, B. A. Gyunka, and A. Noah, “Android Malware Detection through Machine Learning Techniques: A Review,” Int. J. Online Biomed. Eng. IJOE, vol. 16, no. 02, p. 14, Feb. 2020, doi: 10.3991/ijoe.v16i02.11549.

[2] D. Ghimire and J. Lee, “Geometric Feature-Based Facial Expression Recognition in Image Sequences Using Multi-Class AdaBoost and Support Vector Machines,” Sensors, vol. 13, no. 6, pp. 7714–7734, Jun. 2013, doi: 10.3390/s130607714.

[3] R. Wang, “AdaBoost for Feature Selection, Classification and Its Relation with SVM, A Review,” Phys. Procedia, vol. 25, pp. 800–807, 2012, doi: 10.1016/j.phpro.2012.03.160.

[4] J. Sun, H. Fujita, P. Chen, and H. Li, “Dynamic financial distress prediction with concept drift based on time weighting combined with Adaboost support vector machine ensemble,” Knowl.-Based Syst., vol. 120, pp. 4–14, Mar. 2017, doi: 10.1016/j.knosys.2016.12.019.

[5] A. Garg and K. Tai, “Comparison of statistical and machine learning methods in modelling of data with multicollinearity,” Int. J. Model. Identif. Control, vol. 18, no. 4, p. 295, 2013, doi: 10.1504/IJMIC.2013.053535.

[6] C. P. Obite, N. P. Olewuezi, G. U. Ugwuanyim, and D. C. Bartholomew, “Multicollinearity Effect in Regression Analysis: A Feed Forward Artificial Neural Network Approach,” Asian J. Probab. Stat., pp. 22–33, Jan. 2020, doi: 10.9734/ajpas/2020/v6i130151.

[7] W. Wang et al., “Constructing Features for Detecting Android Malicious Applications: Issues, Taxonomy and Directions,” IEEE Access, vol. 7, pp. 67602–67631, 2019, doi: 10.1109/ACCESS.2019.2918139.

[8] B. Rashidi, C. Fung, and E. Bertino, “Android malicious application detection using support vector machine and active learning,” in 2017 13th International Conference on Network and Service Management (CNSM), Tokyo, Nov. 2017, pp. 1–9. doi: 10.23919/CNSM.2017.8256035.

[9] J. Li, L. Sun, Q. Yan, Z. Li, W. Srisa-an, and H. Ye, “Significant Permission Identification for Machine-Learning-Based Android Malware Detection,” IEEE Trans. Ind. Inform., vol. 14, no. 7, pp. 3216–3225, Jul. 2018, doi: 10.1109/TII.2017.2789219.

[10] G. Suarez-Tangil, J. E. Tapiador, P. Peris-Lopez, and J. Blasco, “Dendroid: A text mining approach to analyzing and classifying code structures in Android malware families,” Expert Syst. Appl., vol. 41, no. 4, pp. 1104–1117, Mar. 2014, doi: 10.1016/j.eswa.2013.07.106.

9.2 GITHUB LINK

https://github.com/yeshwanth72/malware-detection