
# Detecting Malicious Android Application

**Seif ElDein Mohamed, Amr Ehab, Mostafa Ashraf, Omar Shereef**

**Supervised by Dr.Eslam Amer**
**Assistant Supervisors: Eng.Haytham Metawie, Eng.Mostafa Badr**

Submitted July 2021, in partial fulfillment of
the conditions for the award of the degree **BSc Computer Science.**

Faculty of Computer Science
Misr International University, Cairo, Egypt

# Abstract

The mobile industry is in rapid evolution making smartphones available at affordable rates for all segments of society. Smartphones' purposes are not limited to making phone calls or sending messages, users can also take photos, store personal data, do online banking and trace their daily activities. The more applications appear the more security becomes a concern to mobile users. This concern arises from the fear of being subjected to a security breach that jeopardizes confidential personal data such as emails, passwords, credentials, etc. Malware applications that are developed for the sake of compromising users' personal data are also increasing rapidly day after day. In our work, we aim to design an intelligent detection framework for Android malware applications. The framework uses different analysis-based approaches along with various machine learning algorithms to distinguish between benign and malicious applications.

# Acknowledgements

Throughout the writing of this thesis, we have received a great deal of support and assistance.
We would first like to thank our supervisor, Dr.Eslam Amer, whose expertise was invaluable in formulating the research questions and methodology. Your sagacious criticism pushed us to hone our reasoning and carried our work to a more significant level.
We would also like to thank our teaching assistants, Eng.Haytham Metawie, Eng.Mostafa Badr, for their valuable guidance throughout our studies. You provided us with the tools that we expected to pick the correct direction and successfully complete our dissertation.
Also, We might want to thank our parents for their shrewd guidance. You are always there for us. Finally, We couldn't have finished this paper without the help of our friends, who gave stimulating conversations just as glad distractions to rest our minds outside of our research.

# List of Publications

- Published a research paper in the International Mobile, Intelligent, and Ubiquitous Computing Conference 2021 (MIUCC)[14].
- Deployed our application in the google play store [20].

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

Mobile malware regularly adopts one of two strategies, said Adam Bauer, a security scientist for the portable security organization. The main kind of malware fools you into allowing authorizations that lets it access touchy data.

That is the place where the application fits in, and a large number of the permissions it mentioned sound like something genuine needed. Yet, they likewise let the application run continually behind the scenes.

The second sort of malware abuses weaknesses in phones, accessing touchy data by giving itself executive advantages. That diminishes the need to get clients to click "OK" on consents demands, simplifying it for malware to run without clients seeing its quality on the portable.

## 1.1 Motivation

### 1.1.1 Problem Statement

Malicious applications nowadays set out to decimate our information on mobile devices, for example misusing user's private information. Many malware applications develop their methods and use permissions of benign or permissions committed by malicious applications that are similar to those benign, these limitations are considered as too little information for classification. So we propose an application to detect malware applications based on permissions and API calls by static and dynamic behavior.

## 1.2 Aims and Objectives

To examine the best way to execute machine learning to malware detection in addition to detect obscure malware. To create a malware detection program that carries out machine learning to identify obscure malware. To approve that malware detection that executes machine learning will actually achieve a high accuracy rate with a low false-positive rate.

## 1.3 Scope

The proposed system classifies malware behaviors of Android Applications with efficiency and viability, utilizing machine learning techniques and to determine the ability of malware, detect it, and contain it. It additionally helps in determining recognizable examples that can be utilized to fix and prevent future infections.

## 1.4 System Overview

The development of the system is split into 4 stages as follows:

- In the first stage, in which the dataset is priority, collected From DefenseDroid[5] and Malgenome[23] that contains unique Android applications. DefenseDroid[5] have permissions and Malgenome[23] have API Calls.
- In pre-processing phase, Normalization applied on the permissions dataset[5] to extract a binary image from it. And also Normalization applied on the on the API Calls dataset[23].
- In processing phase, both datasets divided into two subsets training and testing respectively. In API Calls we have used Decision Tree Algorithm and in permissions, we have used CNN Algorithm and applied it to binary images that we extract from pre-processing phase then we train and test our model.
- In the output stage, our model classifies the selected application that scanned and detects as either benign or malware.

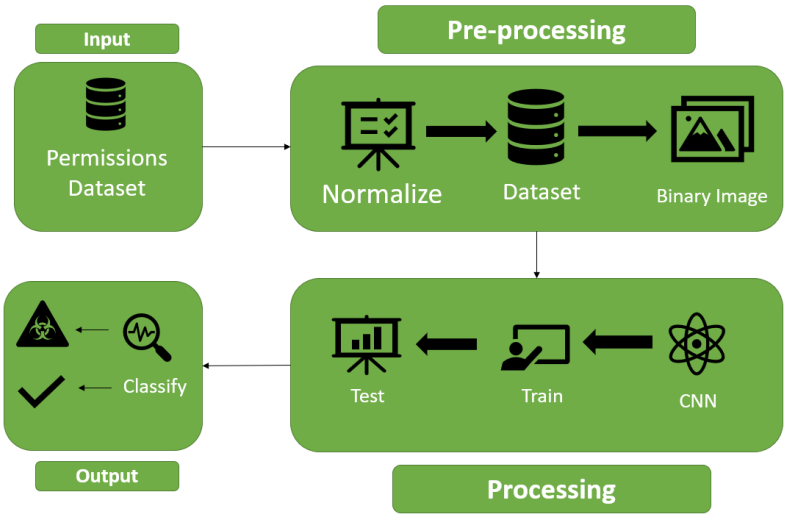


Figure 1.1: System Overview 1

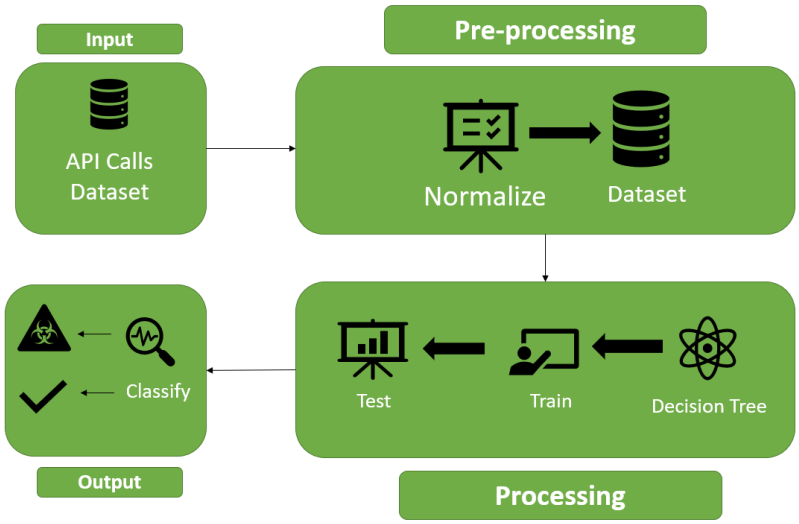


Figure 1.2: System Overview 2

### 1.4.1 Business Context

Malware detection of Android systems is one of the most significant cybersecurity tasks for clients and organizations[3]. Malware Detection provides security from any malware, prevents critical information from getting compromised or altered. Our final goal is to become one of the core pre-installed applications on any new Android device.

### 1.4.2 Users Characteristics

- The end product's user has no special characteristics, but owning an Android device.
- The development team member has to be characterised by having previous experience in implementing machine learning models in Python and dealing with related issues.

## 1.5 Project Management and Deliverables

### 1.5.1 Time Plan

Table 1.1: Project time plan

| Id | Task | Start Date | Days | Team Member |
|---|---|---|---|---|
| 01 | Idea Discussion | 01/08/2020 | 6 | Seif, Mostafa, Amr, Omar |
| 02 | Idea research | 08/08/2020 | 10 | Seif, Mostafa, Amr, Omar |
| 03 | Data-set collection | 19/08/2020 | 30 | Seif, Mostafa, Amr, Omar |
| 04 | Proposal Document Implementation | 19/09/2020 | 37 | Seif, Mostafa |
| 05 | Proposal Code Implementation | 01/10/2020 | 25 | Amr, Omar |
| 06 | Proposal presentation | 27/10/2020 | 0 | Seif,Mostafa, Amr, Omar |
| 07 | SRS Code Implementation | 05/11/2020 | 45 | Seif, Mostafa, Amr, Omar |
| 08 | SRS Document Implementation | 15/11/2020 | 43 | Seif, Mostafa, Amr, Omar |
| 09 | SRS Presentation | 30/12/2020 | 0 | Seif, Mostafa, Amr, Omar |
| 10 | SDD Code Implementation | 15/1/2021 | 41 | Seif, Mostafa, Amr, Omar |
| 11 | SDD Document Implementation | 05/02/2021 | 51 | Seif, Mostafa, Amr, Omar |
| 12 | Writing Paper | 05/02/2021 | 48 | Seif, Amr, Mostafa, Omar |
| 13 | SDD Presentation | 30/03/2021 | 0 | Seif, Mostafa, Amr, Omar |
| 14 | Technical Evaluation Implementation | 15/04/2021 | 55 | Seif, Mostafa, Amr, Omar |
| 15 | Deliver the Paper | 05/05/2021 | 5 | Seif, Mostafa, Amr, Omar |
| 16 | Technical Evaluation Presentation | 13/06/2021 | 0 | Seif, Mostafa, Amr, Omar |
| 17 | Final Thesis Writing | 15/06/2021 | 28 | Seif, Amr, Mostafa, Omar |
| 18 | Final Thesis Presentation | 13/07/2021 | 0 | Seif, Mostafa, Amr, Omar |

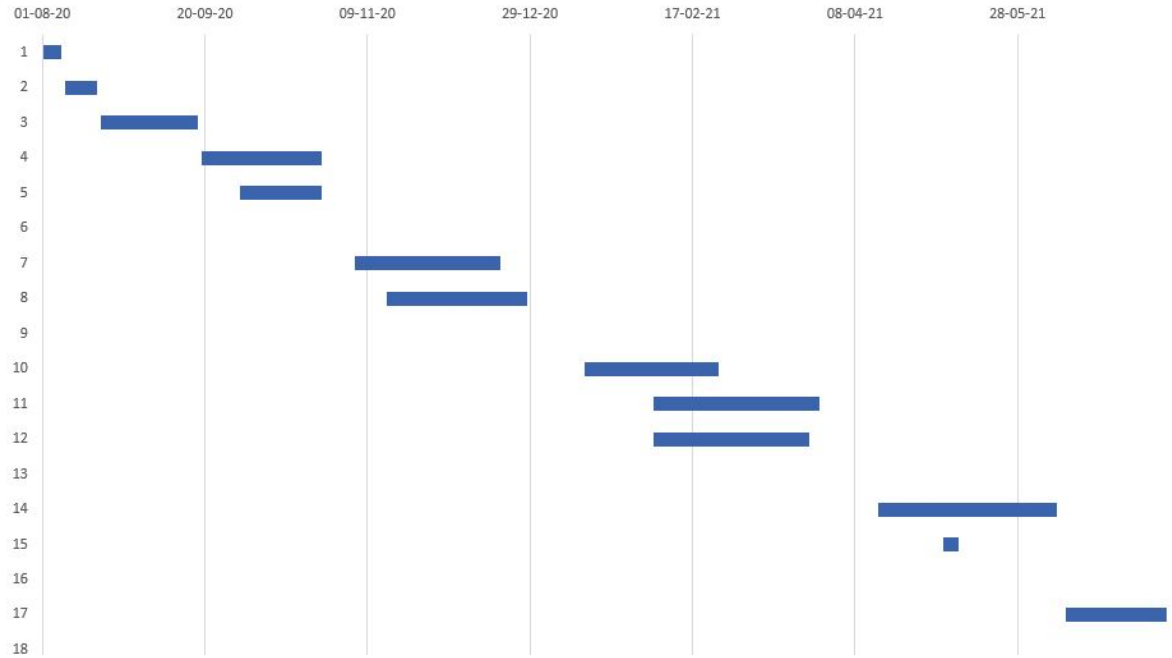


Figure 1.3: GANTT Chart

### 1.5.2 Deliverables

- Software Proposal.
- Software Requirements Specification Document
- Software Design Descriptions Document.
- MIUCC conference paper.
- Thesis Document.
- Python back-end (Model).
- Android application.

## 1.6 Thesis Overview

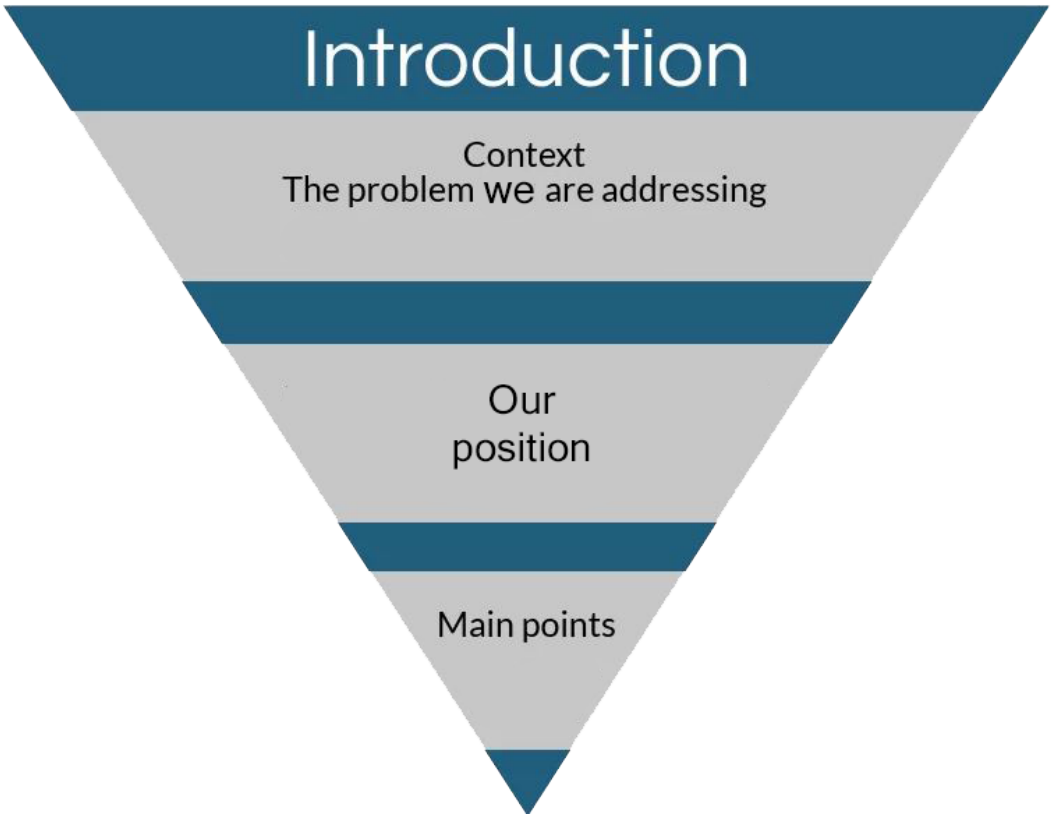


Figure 1.4: Thesis Overview

# Chapter 2

# Background and Related Work

## 2.1 Introduction

Section 2 is spread out as follows. Section 2.2 gives a foundation about malware detection, how it destructive for the security of our devices. Section 2.3 gives a brief outline of related works that spotlights on the techniques for detecting malware by many algorithms. Taking everything into account that section 2.4 sums up this chapter.

## 2.2 Background

Android malware detection has two main identification techniques, namely dynamic analysis and static analysis. Each method has its advantages and disadvantages.Static analysis technology, for example, PApriori [21], and DREBIN [11] browse applications without running them. Nevertheless, the strategy cannot prevent decompilation and confusion.On the other hand, dynamic analysis techniques such as, VetDroid [22] Run the application in real-time to detect malicious behavior, but it is difficult to capture all execution behaviors. With the rapid development of malware, adaptive machine learning technology is used to perform Android malware detection. Therefore, representing malicious behavior accurately requires extracting the most important features to improve malware detection efficiency.

Since most of the applications have corrupted the user's private data such as their location, credit card and contact details. Virtually all applications can access the user's private infor-

mation, although this provides users with better personalized management.It can also lead to private information data overflow and financial misfortunes. Therefore, our project aims to detect any malware in Android device systems based on permissions and API Calls using machine learning techniques. The innovation aspect of the project is to extract images from the permissions dataset then apply the Convolutional neural network (CNN) algorithm. Our system aims to protect our clients' systems from malware applications to protect their personal information from stealing.

## 2.3 Related Systems

**Faiz**[6], A system is proposed to detect Android malware applications through a hybrid classification of K means clustering algorithm and support vector machine (SVM). The authors utilized two datasets[1] [18]. From the first dataset, they generate two datasets Data1 and Data2. Data1 consists of 13,176 applications for training the model. and 1860 for testing. Data2 consists of 12,028 applications for training the model and 3008 for testing. The second dataset consists of 230 conspiring application pairs. The authors assumed that conspiring application-pairs can execute all the threats that are executed by malicious applications. Then they use the parameter vector and a simple decision function to detect application collusion.

**Parnika Bhat**[2], proposed a system that detects malicious Android applications based on the Naïve Bayes model. They got two datasets, DREBIN[1] and PRAGuard[4] data-sets that contain 2870 applications.From 2870 applications they got 1472 pernicious applications and 1398 amiable applications. The scientists tackle the issue by malware identification procedure called MaplDroid utilizes a static investigation approach with Naïve Bayes model examination. Likewise, they applied another characterization strategy like arbitrary backwoods to bring out similar examination. MaplDroid accomplished a score of 99.12%. An obvious shortcoming of this conducted research is the small scale of the used data-set.

**Umme Sumaya Jannat**[10], proposed a framework that investigations and distinguishes Android malware utilizing machine learning. The authors take care of the issue in two different ways by powerful examination and static investigation. They get the best outcome in the unique investigation by applying Random Forest (RF) calculation that is an all-inclusive form from Decision Tree (DT) calculation. The result of dynamic analysis has exceeded the static analysis accuracy scores over 93% accuracy. Also, the researchers have used different datasets for statics analyses and dynamic analyses. They used MalGenome data-set for static analysis and it is composed of roughly 360 applications gathered based on which malware families they belong to, and another dataset from Kaggle that contains 4000 malicious applications in JSON file format. Also, in dynamic examination, they utilized MalGenome[19] dataset which contains 1260 malware applications that have a place with 49 unique malware families.

**Zhuo Ma**[13], proposed a framework that recognizes pernicious Android applications. The analysts utilized control stream charts and machine learning algorithms.They assemble sequential informational indexes and train them by making a Long momentary memory calculation which is a recurrent neural network. The analysts de-incorporate and set up 3 sorts of frameworks: API usage data-sets, API frequency data-sets, and API sequence data-sets. The conducted research achieved 98.98% detection precision. An obvious shortcoming of this conducted research is the small scale of the used dataset.

**Muhammad Murtaz**[16], proposed a framework that extensions to identify Android malware applications. They solve the problem by using 6 algorithms K-Nearest Neighbor (KNN), Support Vector Machine (SVM), Decision Tree (J48), Neural Networks (NN), Naïve Bayes (NB), and Random Forest (RF) on the data-set they acquire and test it on Waikato Environment for Knowledge Analysis (WEKA). They used a data-set called CICAndMal2017 that has 10854 applications (6500 benign and 4354 malware), and the data-set is grouped into four malware families (SMS Malware, Ransomware, Adware, Scareware). They show in this research that they detect malware location in 9 movements to speed up activity classifier productivity. Also, the model makes use of gathering methodologies including time-based, bundle-based, and stream-based highlights to describe malware families. The appraisal displays the proposed fuse set has more than 94 huge for genuine malware affirmation systems.

## 2.4 Summary

Android malware applications can be detected by two techniques static or dynamic. Our project is to detect Android malware applications based on API Calls and permissions using machine learning techniques. And our innovation aspect is to extract images from the permissions dataset then apply the Convolutional neural network (CNN) algorithm. There are different similar systems that used different algorithms one of them like the paper of the author **Umme Sumaya Janna**[10] used the Random Forest (RF) algorithm with an accuracy of 93%.

# Chapter 3

# System Requirements Specification

## 3.1 Introduction

The primary reason for this Software Requirements Specification record is to delineate and layout the prerequisites for a malware detection application that are mainly differentiating, detecting, classifying, analyze various incorrect behaviors in the application analysis.

## 3.2 Functional Requirements

### 3.2.1 System Functions

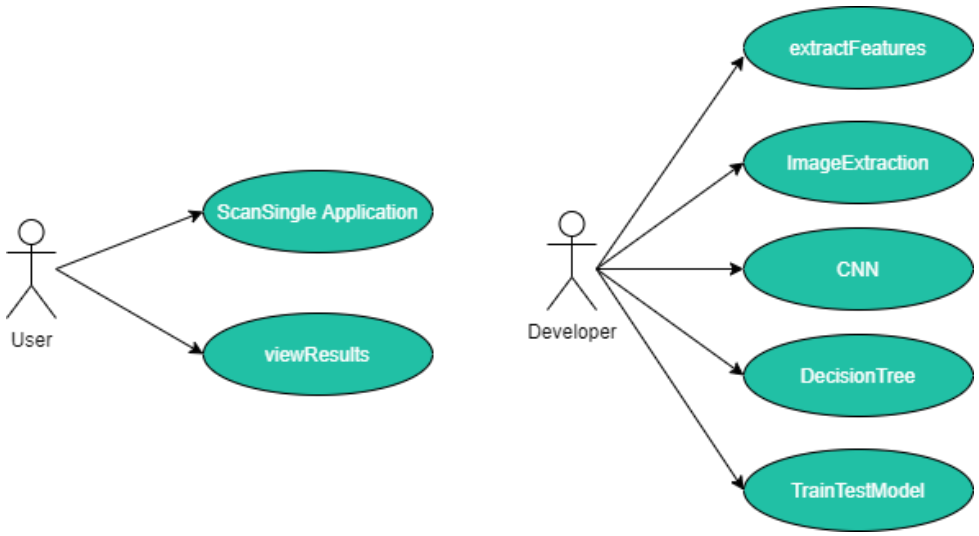


Figure 3.1: Use Case Diagram

### 3.2.2 Detailed Functional Specification

**Building model**

Table 3.1: Extract Features Function Description

| | |
|---|---|
| **Function ID** | FR01 |
| **Name** | extractFeatures |
| **Description** | Reducing the number of features by extracting the highly informative features from the data-set. |
| **Pre-condition** | Having a balanced and correctly collected data-set |
| **Input** | Data-set |
| **Action** | Performing statistical analysis on the data-set's features, and excludes the less informative features in classification decision. |
| **Output** | Reduced data-set in dimensions |
| **Post-condition** | N/A |
| **Dependency** | N/A |

Table 3.2: Image Extraction

| | |
|---|---|
| **Function ID** | FR02 |
| **Name** | ImageExtraction |
| **Description** | Convert matrix into image |
| **Pre-condition** | Binary vector must be applied on the dataset |
| **Input** | Permissions |
| **Action** | Apply image extraction on the Permissions calls matrix |
| **Output** | Image for each row. |
| **Post-condition** | Apply CNN Algorithm on Images |
| **Dependency** | FR01 (extractFeatures) |

Table 3.3: CNN Algorithm in Image Extraction From Dataset

| | |
|---|---|
| **Function ID** | FR03 |
| **Name** | CNN |
| **Description** | Take image and apply CNN Algorithm |
| **Pre-condition** | Image Extraction must be applied on the dataset |
| **Input** | Images |
| **Action** | Apply CNN Algorithm on the images |
| **Output** | Train accuracy for images. |
| **Post-condition** | Extract Model to use in mobile application |
| **Dependency** | FR02 (ImageExtraction) |

Table 3.4: Decision Tree Algorithm

| | |
|---|---|
| **Function ID** | FR04 |
| **Name** | DecisionTree |
| **Description** | Apply Decision Tree on API Calls |
| **Pre-condition** | N/A |
| **Input** | API Calls |
| **Action** | Apply Decision Tree Algorithm on API Calls Dataset[23] |
| **Output** | Train accuracy. |
| **Post-condition** | Extract Model to use in mobile application |
| **Dependency** | FR01 (extractFeatures) |

**End User**

Table 3.5: Scan Selected Application Function Description

| | |
|---|---|
| **Function ID** | FR05 |
| **Name** | scanSelectedApplication |
| **Description** | Scans the application the user selected for any malware behavior |
| **Pre-condition** | N/A |
| **Input** | Applications' package names |
| **Action** | Classify each application as either benign or malware. |
| **Output** | N/A |
| **Post-condition** | Navigates to result page which detect application either malware or benign |
| **Dependency** | N/A |

Table 3.6: View Results Page Function Description

| | |
|---|---|
| **Function ID** | FR06 |
| **Name** | viewResults |
| **Description** | Views the results of scans the user initiated |
| **Pre-condition** | User must scan first |
| **Input** | N/A |
| **Action** | Displays malicious or benign application if any exists. |
| **Output** | Benign or malicious application if any exists |
| **Post-condition** | N/A |
| **Dependency** | FR05 (scanSelectedApplication) |

.

## 3.3 Interface Requirements

### 3.3.1 User Interfaces

#### GUI

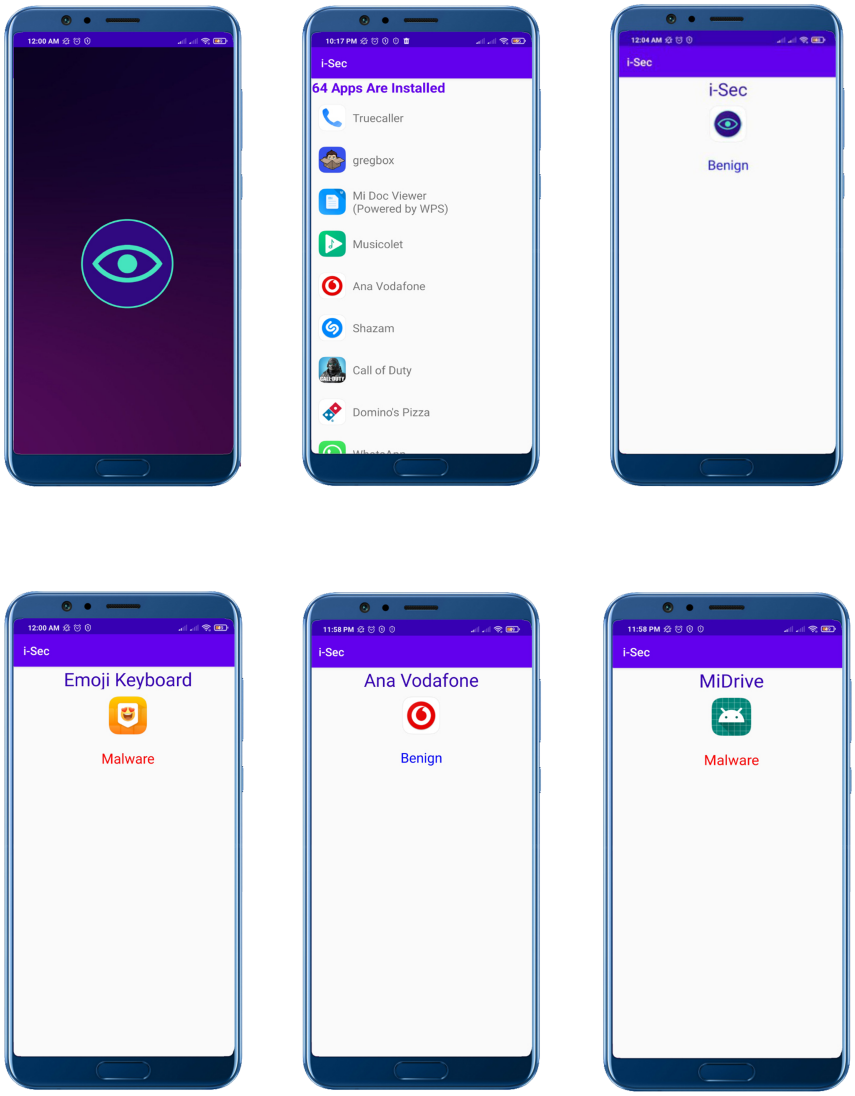


Figure 3.2: GUI

## CLI

```
import numpy as np
from PIL import Image

#Sqaure Image of Train Dataset
count_train = 0
from csv import reader
with open('Train_Perm.csv', 'r') as read_obj_train:
    csv_reader_train = reader(read_obj_train)
    next(csv_reader_train)
    for row_train in csv_reader_train:
        square_train = np.array(row_train).reshape(39, 39)
        print(square_train)
        count_train += 1
        print("Row Number of train dataset", count_train, "Printed")
        array_train = np.array(square_train, dtype=np.uint8)
        print("Array of the image of train dataset:", array_train)
        my_image_train = Image.fromarray(array_train * 255)
        #transpose_img_train = np.transpose(my_image_train)
        #my_image_train.show()
        #my_image2_train = Image.fromarray(transpose_img_train)
        my_image_train.save('Train_Permission_Images/Train_Image'+str(count_train)+'.jpg')
        print("DONE WITH IMAGE SAVE OF TRAIN DATASET", count_train)
```

Figure 3.3: Image Extraction Code 1 - CLI (1)

```
#Sqaure Image of Test Dataset
count_test = 0
from csv import reader
with open('Test_Perm.csv', 'r') as read_obj_test:
    csv_reader_test = reader(read_obj_test)
    next(csv_reader_test)
    for row_test in csv_reader_test:
        square_test = np.array(row_test).reshape(39, 39)
        print(square_test)
        count_test += 1
        print("Row Number of test dataset", count_test, "Printed")
        array_test = np.array(square_test, dtype=np.uint8)
        print("Array of the image of test dataset:", array_test)
        my_image_test = Image.fromarray(array_test * 255)
        #transpose_img_test = np.transpose(my_image_test)
        #my_image_test.show()
        #my_image2_test = Image.fromarray(transpose_img_test)
        my_image_test.save('Test_Permission_Images/Test_Image'+str(count_test)+'.jpg')
        print("DONE WITH IMAGE SAVE OF TEST DATASET", count_test)
```

Figure 3.4: Image Extraction Code 2 - CLI (2)

```
from sklearn.metrics import confusion_matrix, classification_report
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
from PIL import Image
import tensorflow as tf
import numpy as np
import glob
import cv2
import os

# Train Images and label
X_train = []
path_Train = glob.glob("Train_Images/*.png")
for myFile_Train in path_Train:
    image_Train = cv2.imread(myFile_Train)
    X_train.append(image_Train)

y_train = []
y_train = np.loadtxt("Label_TrainDataset80.txt", np.int)

X_train = np.array(X_train, dtype=np.uint8)
y_train = np.array(y_train, dtype=np.uint8)

print("Shape of Train images=(total number, width, height, three channels -> RGB)  :", X_train.shape)
print("Length of Train Label Data-set :", len(y_train))
print("-----------------------------------------------------------------------------------------------------")

# Test Images and label
X_test = []
pathTest = glob.glob("Test_Permission_Images40x40/*.png")
for myFileTest in pathTest:
    imageTest = cv2.imread(myFileTest)
    X_test.append(imageTest)

y_test = []
y_test = np.loadtxt("Label_TestDataset20.txt", np.int)

X_test = np.array(X_test, dtype=np.uint8)
```

Figure 3.5: CNN 1 - CLI (3)

```
import glob
import cv2
import os

# Train Images and label
X_train = []
path_Train = glob.glob("Train_Images/*.png")
for myFile_Train in path_Train:
    image_Train = cv2.imread(myFile_Train)
    X_train.append(image_Train)

y_train = []
y_train = np.loadtxt("Label_TrainDataset80.txt", np.int)

X_train = np.array(X_train, dtype=np.uint8)
y_train = np.array(y_train, dtype=np.uint8)

print("Shape of Train images=(total number, width, height, three channels -> RGB)  :", X_train.shape)
print("Length of Train Label Data-set :", len(y_train))
print("-----------------------------------------------------------------------------------------------------")

# Test Images and label
X_test = []
pathTest = glob.glob("Test_Permission_Images40x40/*.png")
for myFileTest in pathTest:
    imageTest = cv2.imread(myFileTest)
    X_test.append(imageTest)

y_test = []
y_test = np.loadtxt("Label_TestDataset20.txt", np.int)

X_test = np.array(X_test, dtype=np.uint8)
y_test = np.array(y_test, dtype=np.uint8)

print("Shape of Test images=(total number, width, height, three channels -> RGB)  :", X_test.shape)
print("Length of Test Label Data-set :", len(y_test))
print("-----------------------------------------------------------------------------------------------------")
```

Figure 3.6: CNN 2 - CLI (4)

```
# Train CNN Model Algorithm
cnn = models.Sequential([
    layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(40, 40, 3)),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(filters=128, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),


    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    # layers.Dense(128, activation='relu'),
    # layers.Dense(128, activation='relu'),
    layers.Dense(2, activation='softmax')

])

cnn.compile(optimizer='adam',
            loss='sparse_categorical_crossentropy',
            metrics=['accuracy'])

cnn.fit(X_train, y_train, epochs=200)

print("-------------------------------------------------Test-------------------------------------------------")
# Test CNN Model Algorithm
cnn.evaluate(X_test, y_test)

cnn.save("CNN.h5")
```

Figure 3.7: CNN 3 - CLI (5)

```
import pandas as pd
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
import joblib

def DecisionTree(df):
    print("Decision Tree Algorithm: ")
    print("")
    X_train, X_test, y_train, y_test = train_test_split(df.iloc[:, 1:83], df['class'], test_size=0.2, random_state=42)

    clf = DecisionTreeClassifier()
    clf = clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)

    accuracy = accuracy_score(y_pred, y_test)
    print("Accuracy:", metrics.accuracy_score(y_test, y_pred))
    print(classification_report(y_test, y_pred))

    # This Is To Save My Mode
    filename = 'DecisionTree_model.sav'
    joblib.dump(clf, filename)

    # load the model from disk
    loaded_model = joblib.load(filename)
    result = loaded_model.score(X_test, y_test)
    print(result)

df = pd.read_csv("API Calls dataset.csv")
print(df.shape)
apicallsDf = df


DecisionTree(apicallsDf)
```

Figure 3.8: API Calls - Decision Tree Algorithm

### 3.3.2 API

#### External Libraries :

**Python Libraries [12] :**

- NumPy: Support for large multidimensional arrays.
- Pandas: Arrangement of simple information structure and speedier information examination for Python.
- Scikit-learn: It consists of numerous clustering, regression and classification algorithms.
- TensorFlow: focus on enabling fast experimentation.
- CV2: OpenCV loads an image from the specified file.
- Glob: retrieve files/pathnames matching a specified pattern
- PIL: for image extraction.
- Sklearn.metrics: to get accuracy score and classification report for API Calls Model.
- Sklearn.model_selection: to split the dataset into train and test.
- Sklearn.tree: to import a Decision Tree classifier.
- joblib: save API Calls Model in the file.

**Java Libraries (Android Studio) :**

- Tensorflow Lite: Read python TensorFlow machine learning model.

## 3.4 Design Constraints

- This framework should be "user-friendly", easy to use by presenting the user with a clean design and providing them a responsive interface.[8].
- Any smart mobile device that uses the Android operating system with minimum hardware specifications as mentioned in section 3.4.2 later.

### 3.4.1 Standards Compliance

Using the usability ISO 9241 and ISO 25062 standards for Android application[15] to develop our user interface.

### 3.4.2 Hardware Limitations

These are the minimum specifications of an Android smartphone required for our application [7].

- Android 4.4
- Quad Core ARM processor
- 2 GB of RAM.
- 8GB of storage.

## 3.5 Non-functional Requirements

In spite of the fact that this framework may even now work without these non-functional requirements they are as yet expected to improve the framework, so security, reliability, maintainability, and portability are the non-functional requirements that the framework attempts to accomplish.

### 3.5.1 Security

This component is achieved by the framework by hashing private information of the user will be compulsory to guarantee that the put-away information is made sure about.

### 3.5.2 Maintainability

This framework is applied by executing the idea of "MVC" that gives the framework greater adaptability later on while applying any progressions which won't require testing all documents[17].

### 3.5.3 Reliability

The system must be very accurate and error-free since the result provided by the system determines whether the application is malicious or benign.

### 3.5.4 Portability

Portability can be accomplished in this framework by executing a responsive interface to permit the client to associate with the framework using any gadget that has a WiFi connection with the minimum mobile specifications.

## 3.6 Summary

To conclude that there are two types of functional requirements one for building the model and the other one for end-user. Also, We have shown the GUI of mobile application which is very simple for all users to deal with it. Also, we show the CLI of the model. We illustrated the libraries that we used. And there are hardware limitations for minimum specification for Android smartphones. Last but not least the non-functional requirements.

# Chapter 4

# System Design

## 4.1 Introduction

### 4.1.1 Purpose of this section

The purpose of the software design document is to present a detailed description of project system architecture and design. The document will explain the features of the i-Sec system and provides insight into the structure and design of each component. Also, this document will help the project team and the customer to have a full overview of the interface and the functions of the project. The customer can review this document to be aware of the requirements that are finished.

## 4.2 Design viewpoints

### 4.2.1 Context viewpoint

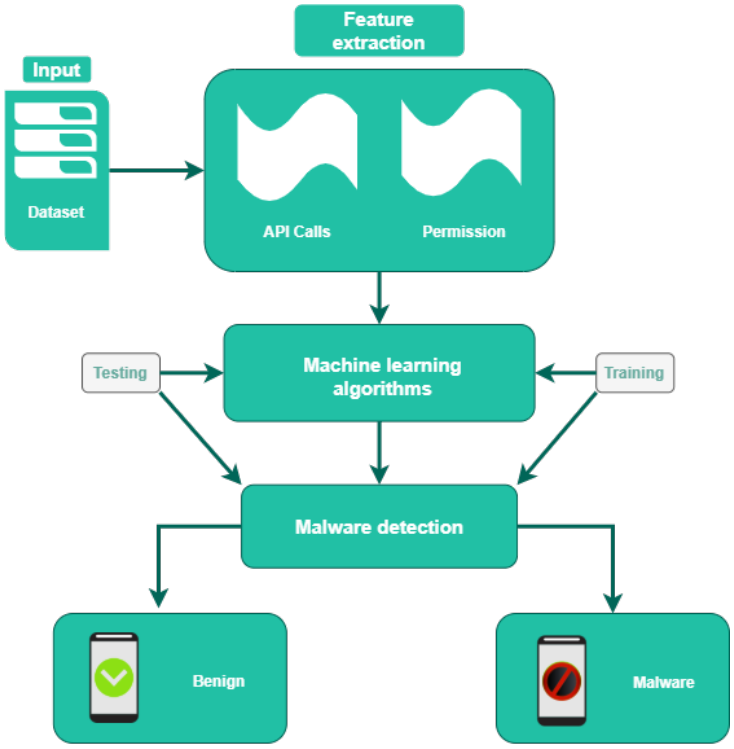


Figure 4.1: System Context diagram

### 4.2.2 Composition viewpoint

By using MVC structure which causes us to isolate our framework to view, model and controller.

- **View:** Firstly, the user has a screen like a welcome screen that contains the logo of our application and appears for about two seconds. The second screen contains a list of all applications that are on the user's phone. If the user selects an application from the list, the user has the ability to scan the application that has been selected then it will navigate to a result page that shows if the application is either malware or benign.

- **Model:** There are eleven libraries that were used. For the Permissions dataset[5] We started with NumPy which supports large multidimensional arrays, then PIL to save images from the Numpy array. Then we go to the second phase that we will apply Convolutional Neural Network (CNN) algorithm. In this phase. Firstly, we used Glob to

select specific folders. Then CV2 loads images from the specified folder. Then we used Pandas analysis data quicker for python. Tensorflow and Scikit-learn which contains the Convolutional Neural Network (CNN) algorithm that uses GPU version to speed up the training and testing. And for API Calls dataset[23] we used Sklearn.metrics to get accuracy and classification report, Sklearn.model_selection to split the dataset into train and test, Sklearn.tree to import a Decision Tree classifier, and joblib to save API Calls Model.

- **Controller:** The controller connects the view with the model. And classify the model by our algorithms.

**Design concerns:**

- For our machine learning model we used image extraction for the permissions of applications in order to train and test these images for the CNN classifier and train the API calls with decision tree classifier and connect the model to the mobile application.
- For our mobile app it uses the pre-trained model to classify the installed applications while using object oriented programming with java.

**Design Rationale**

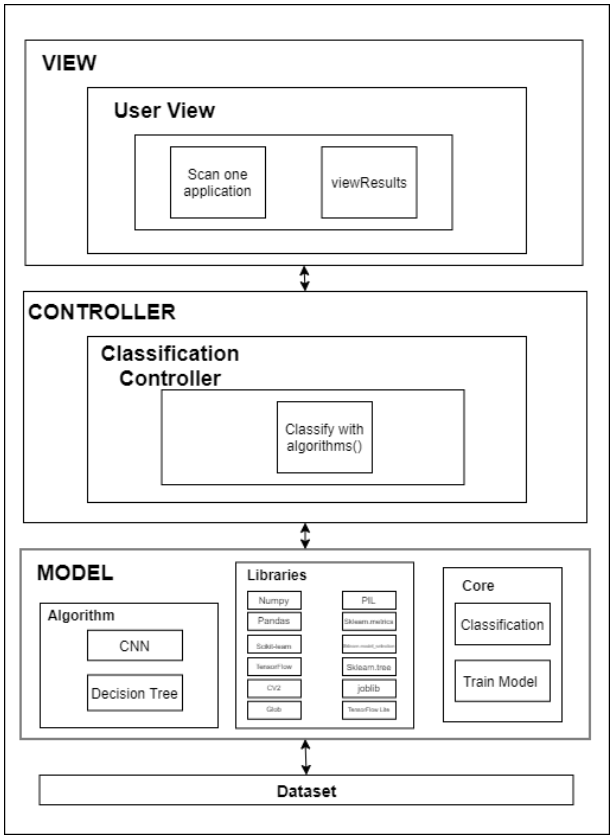


Figure 4.2: Model-View-Controller Diagram

### 4.2.3 Logical viewpoint

Table 4.1: Model Description

| | |
|---|---|
| **Abstract or Concrete:** | Concrete |
| **Superclasses** | Object |
| **Subclasses** | N/A |
| **Purpose** | Balancing and extracting features from datasets and training, testing, and validating the machine learning model. |
| **Collaborations** | N/A |
| **Attributes** | OriginalDataset, newDataset, TrainSize, TestSize. |
| **Operations** | For API Calls, the Decision Tree algorithm is applied. And for Permissions, image Extraction applied from permissions dataset[5] then apply CNN algorithm. Then training and testing the model on both algorithms. |

### 4.2.4 Patterns use viewpoint

**Design Rationale**

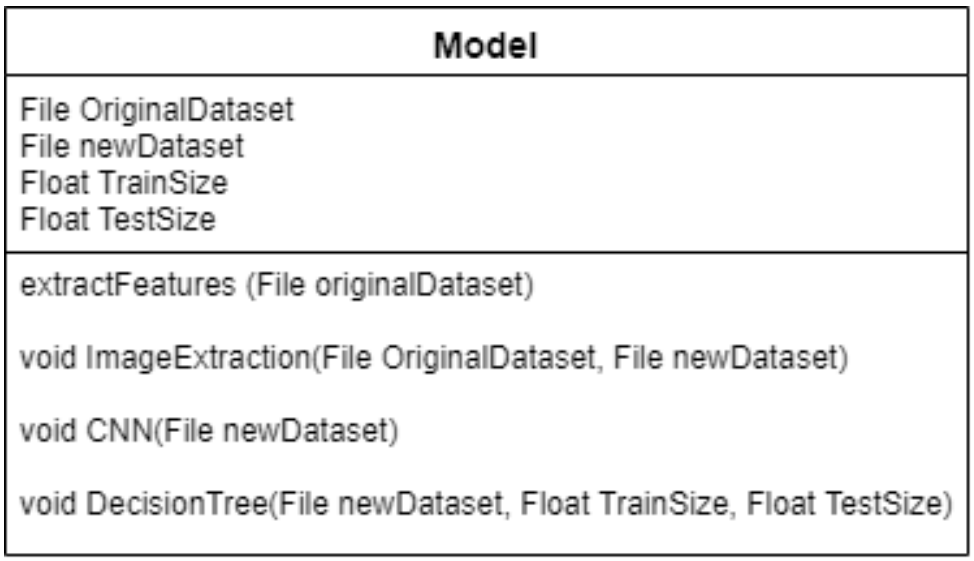


Figure 4.3: Class Diagram

### 4.2.5 Algorithm viewpoint

We follow a two exploration system in the identification model:

(i) Define and extract features from the application and create training data.

(ii) Create classifiers as per explicit machine-learning calculations and afterward identify malware with these classifiers.

The complete architecture of this model is shown in figure 4.4.

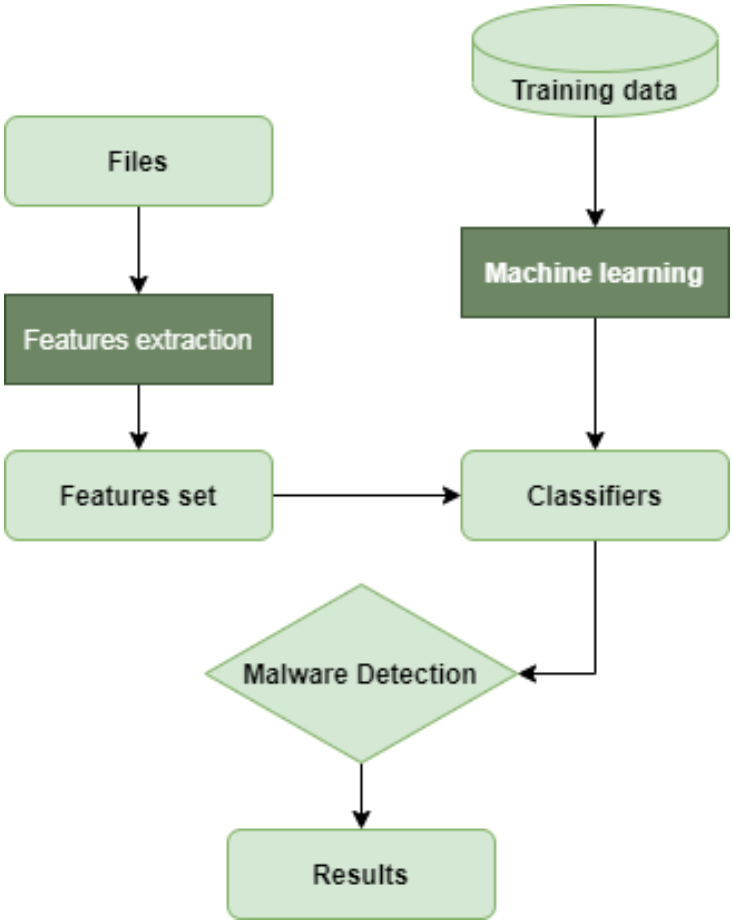


Figure 4.4: Architecture Diagram

### 4.2.6 Interaction viewpoint

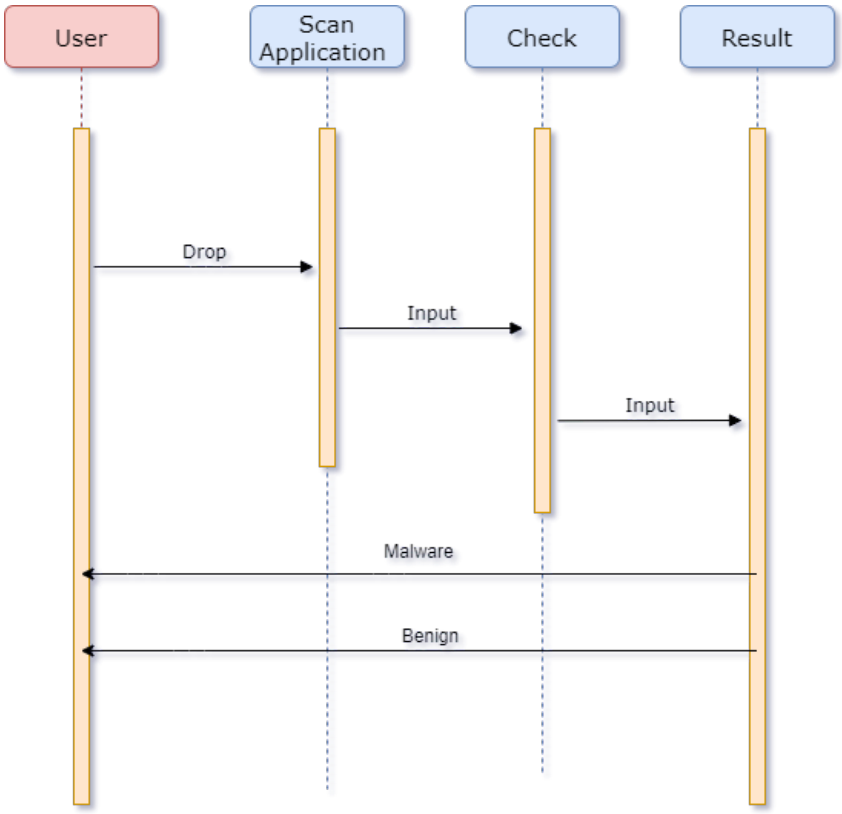


Figure 4.5: Sequence Diagram

### 4.2.7 Interface viewpoint

We build our User interface using java, to introduce mobile applications easy and clear for the user for scanning an application on the users' phone. And shows the malware and benign applications. User interfaces are addressed separately in section 4.4.

## 4.3 Data Design

### 4.3.1 Data Description

The data set used to build the model is called DefenseDroid[5] dataset and Malgenome[23] dataset. DefenseDroid[5] dataset consists of 11975 applications, It has a total of 1490 with a set of Permissions and its number of occurrences on malware and benign. And The Second one for API Calls is called Malgenome[23] dataset that consists of 3799 applications.

### 4.3.2 Dataset Description

Table 4.2: Permissions Dataset

| | |
|---|---|
| Dataset Name | DefenseDroid |
| Link | https://www.kaggle.com/defensedroid/android-malware-detection |
| Number of classes | 2 |
| Notes | Permission |

Table 4.3: API Calls Dataset

| | |
|---|---|
| Dataset Name | Malgenome |
| Link | http://www.malgenomeproject.org/ |
| Number of classes | 2 |
| Notes | API Calls |

## 4.4 Human Interface Design

### 4.4.1 User Interface

Our framework UI is clear. Users can scan only one application. The framework explores you to various screens that rely upon your sort. The results framework has a few obligations. The first screen is like a welcome screen that contains the logo of our application and appears for about two seconds. The second screen has a list of rows each row includes the application name and its logo. If the user selects one application. Our system will navigate the user to a result screen of the application that he/she selected and shows him/here if the application either malware or benign as shown in section 4.4.2

### 4.4.2 Screen Images

Figure 4.6: Welcome Screen

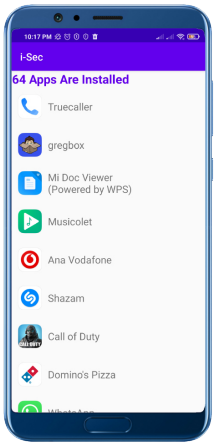


Figure 4.7: All Applications screen


Figure 4.8: Results Screen - Malware Application

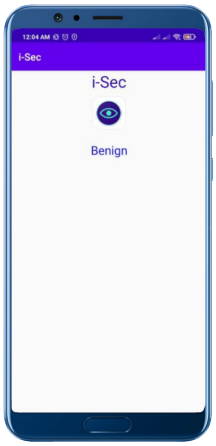


Figure 4.9: Results Screen - Benign Application

### 4.4.3 Screen Objects and Actions

As we mentioned we have a list of applications by click on the application it will show the result on another page that tells the user if it is benign or malware as you can see in 4.10 and 4.11 figures.

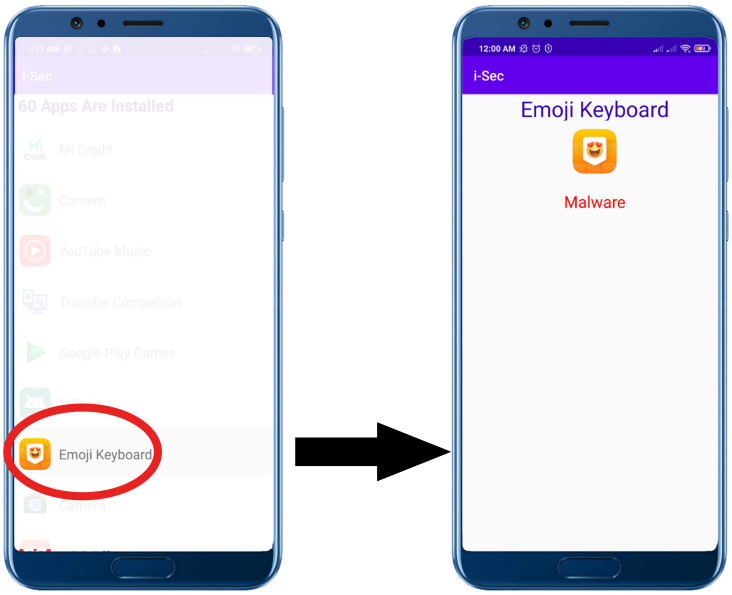


Figure 4.10: When User Select application to scan 1

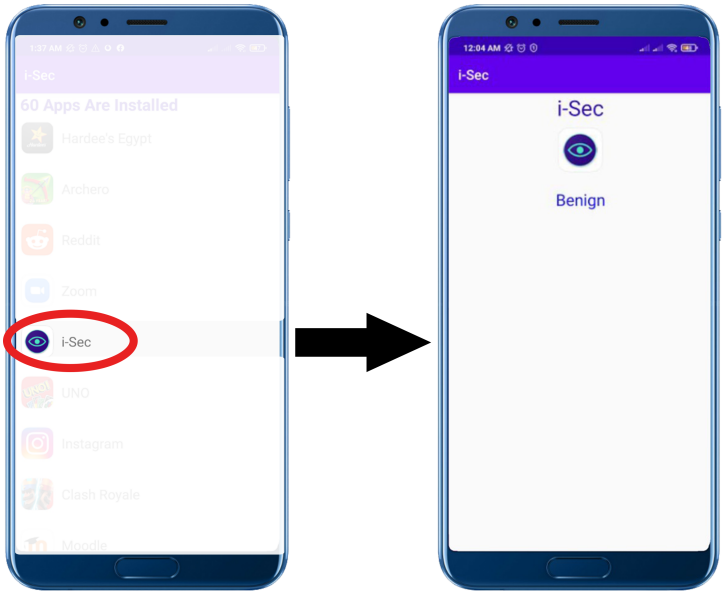


Figure 4.11: When User Select application to scan 2

## 4.5 Implementation

Our System is implemented in two ways:

**1-Model implementation:** We build our model using python programming language using machine learning techniques. We used the Convolutional neural network (CNN) and Decision Tree algorithms.

**2-Mobile application implementation:** We are used java to build the mobile application which is the official language to build android applications

## 4.6 Testing Plan

### 4.6.1 Test Scenario Scan Selected application

On opening the application, the user will select a specific application to scan from their device to discover which applications are benign, and which are malicious.

#### Test Cases

Test Cases for the scenario mention in section 4.6.1 shown in Table 4.4

Table 4.4: Test Scenario Scan Selected application

| Test Case ID | Test Case Desc | Functional Req Code | Test Data | Expected Result |
|---|---|---|---|---|
| TC01 | Selected one App to scan | FR05 | Select 1 App | Malware App |
| TC02 | Selected one App to scan | FR05 | Select 1 App | Benign App |

### 4.6.2 Test Scenario No Permission Found

While scanning the application will not give him results if the application has no permission.

#### Test Cases

Test Cases for the scenario mention in section 4.6.2 shown in Table 4.5

Table 4.5: Test Scenario No Permission Found

| Test Case ID | Test Case Desc | Functional Req Code | Test Data | Expected Result |
|---|---|---|---|---|
| TC03 | Selected one App to scan | FR05 | Select 1 App | No Permission Found |

## 4.7 Requirements Matrix

Table 4.6: Requirements Matrix

| Req. ID | Req Desc | Class | Status |
|---|---|---|---|
| FR01 | Reducing the number of features by extracting the highly informative features from the data-set | extractFeatures | Developed |
| FR02 | Convert matrix into image | ImageExtraction | Developed |
| FR03 | Take image and apply CNN Algorithm | CNN | Developed |
| FR04 | Apply Decision Tree algorithm on API Calls | DecisionTree | Developed |
| FR05 | Scans the application the user selected for any malware behavior | scanSelectedApplications | Developed |
| FR06 | Views the results of scans the user initiated | viewResults | Developed |

## 4.8 System Deployment

- Firstly, We made an account for the developer on the google play console[9].
- Then we create a new app to upload our APK on it.
- Setup the application requirements like put logo of application, application name, privacy policy, and add countries/ regions that can install our application.
- Lastly, we add the APK file that we extracted from the source code.

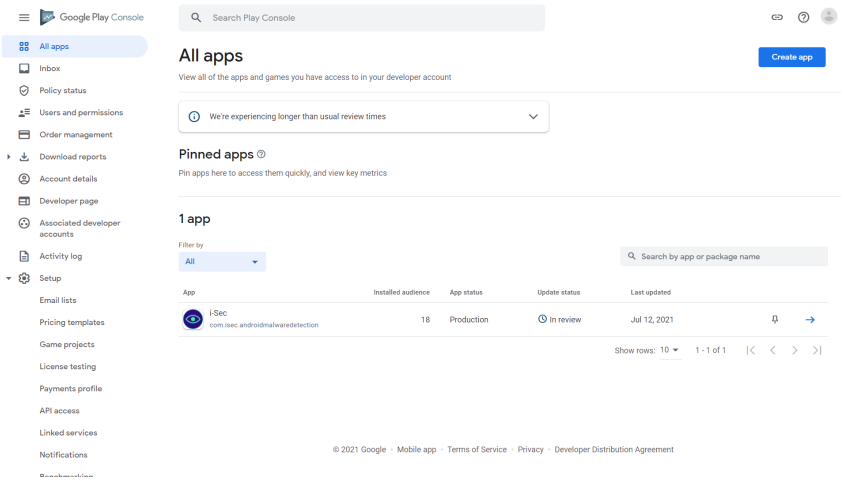


Figure 4.12: System Deployment 1

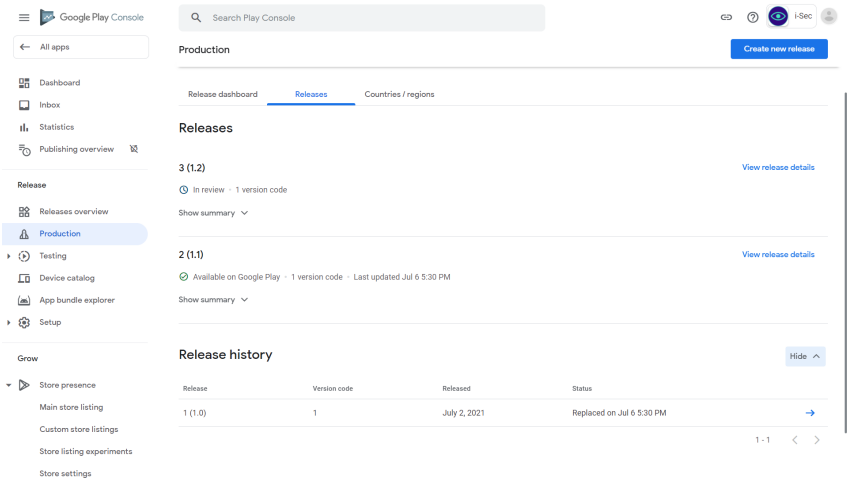


Figure 4.13: System Deployment 2

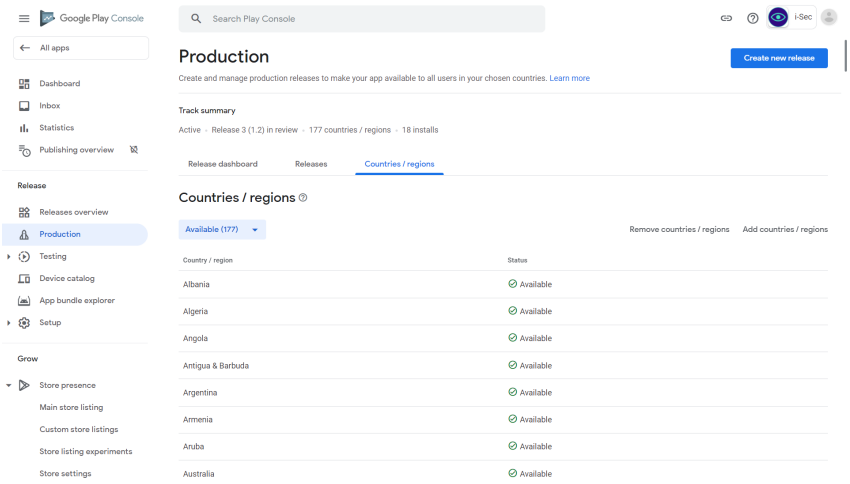


Figure 4.14: System Deployment 3

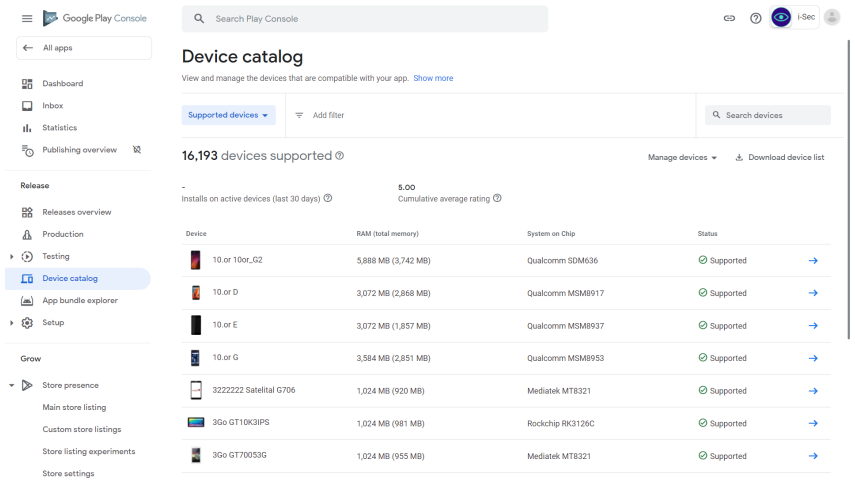


Figure 4.15: System Deployment 4

## 4.9 Summary

This chapter solicitous on determining and analyzing the software design to be developed. It is a rule for execution. It contains all the necessary data to compose a code. It ought to all system architecture that aids in programming by giving details of this project and how it will work. Additionally, it depicts the procedures that will be utilized for testing. It assists with recognizing the function and features that will be tested.

# Chapter 5

# Evaluation

## 5.1 Experiments

- We installed a malware application on our mobile application system and it successfully detect it as a malware application and detect all benign applications as a benign ones.

## 5.2 User evaluation

- The Survey Evaluation:

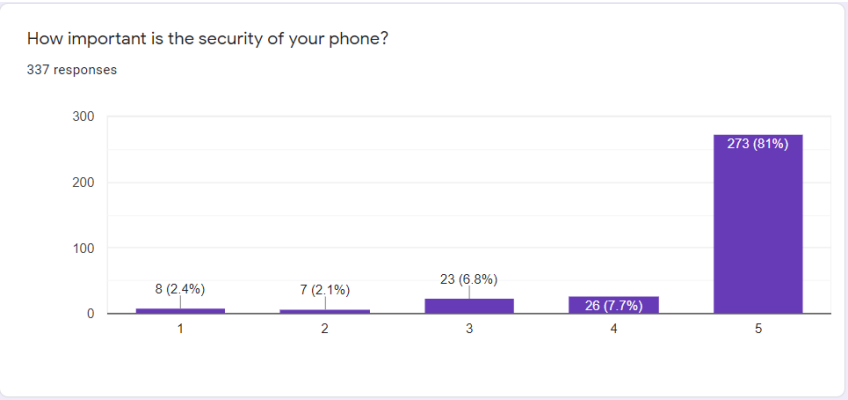


Figure 5.1: Survey First Evaluation

- There are 273 out of 337 responses (81%) of users who are very likely to be important about the security of their phones in fig.5.1

- There are 8 out of 337 responses (2.4%) of who users are not likely to be important about the security of their phones in fig.5.1

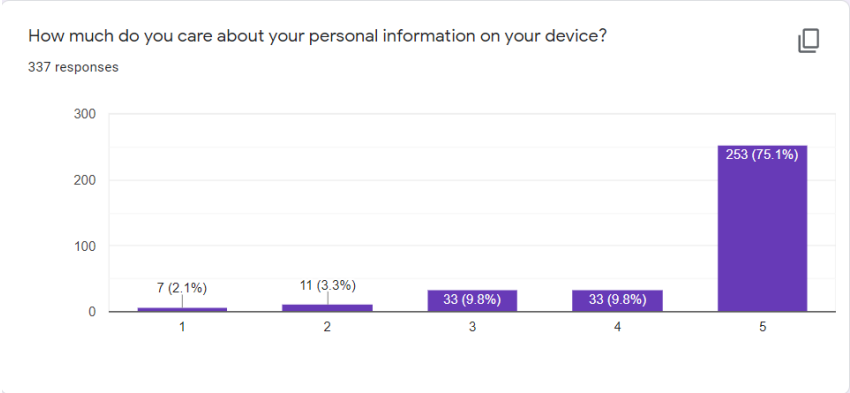


Figure 5.2: Survey Second Evaluation

- There are 253 out of 337 responses (75.1%) of users are very likely to care about their personal information on their device in fig.5.2
- There are 7 out of 337 responses (2.1%) of users are not likely to care about their personal information on their device in fig.5.2

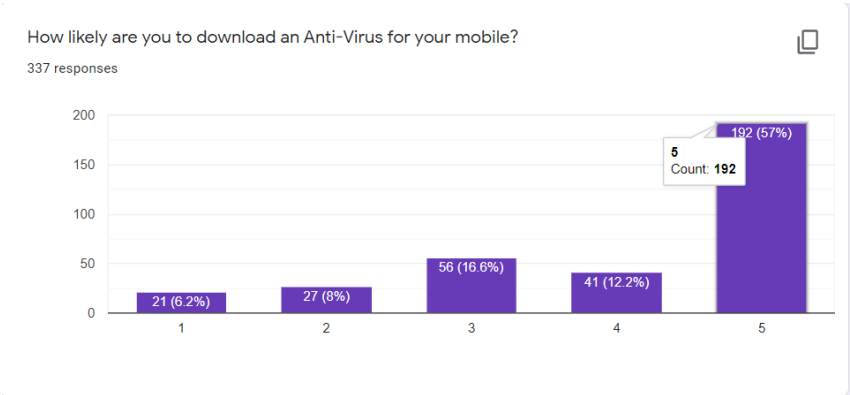


Figure 5.3: Survey Third Evaluation

- There are 192 out of 337 responses (57%) of users are very likely to download an Anti-Virus for their mobile in fig.5.3
- There are 21 out of 337 responses (6.2%) of users are not likely to download an Anti-Virus for their mobile in fig.5.3

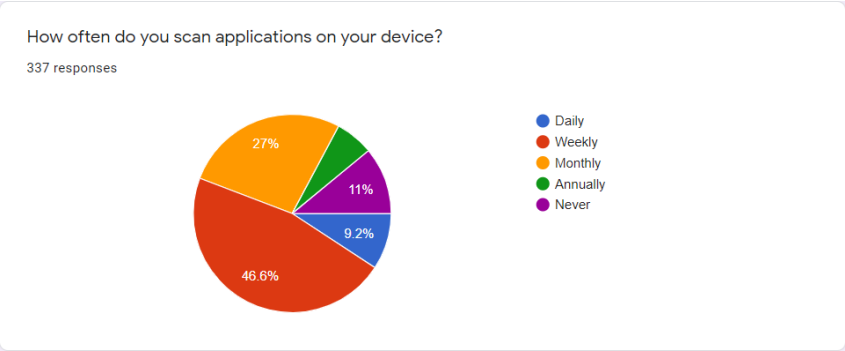


Figure 5.4: Survey Fourth Evaluation

- There are 157 out of 337 responses (46.6%) of users that **Weekly** scan their applications on their device in fig.5.4
- There are 91 out of 337 responses (27%) of users that **Monthly** scan applications on their device in fig.5.4
- There are 21 out of 337 responses (6.2%) of users that **Annually** scan their applications on their device in fig.5.4

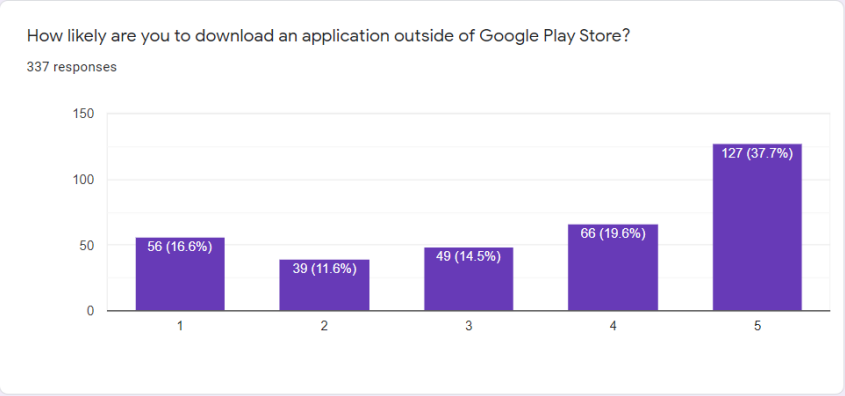


Figure 5.5: Survey Fifth Evaluation

- There are 127 out of 337 responses (37.7%) of users that are very likely to download an application outside the play store in fig.5.5
- There are 56 out of 337 responses (16.6%) of users that are not likely to download an application outside the play store in fig.5.5

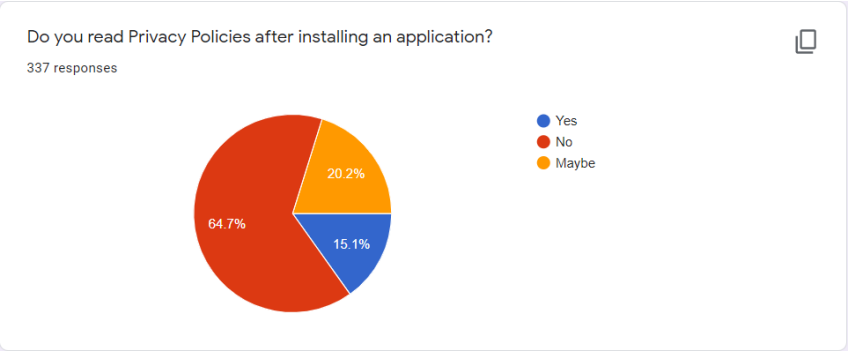


Figure 5.6: Survey Sixth Evaluation

- There are 51 out of 337 responses (15.1%) of users that **Read** Privacy Policies after installing an application in fig.5.6
- There are 68 out of 337 responses (20.2%) of users that **May read** Privacy Policies after installing an application in fig.5.6
- There are 218 out of 337 responses (64.7%) of users that **Do not read** Privacy Policies after installing an application in fig.5.6

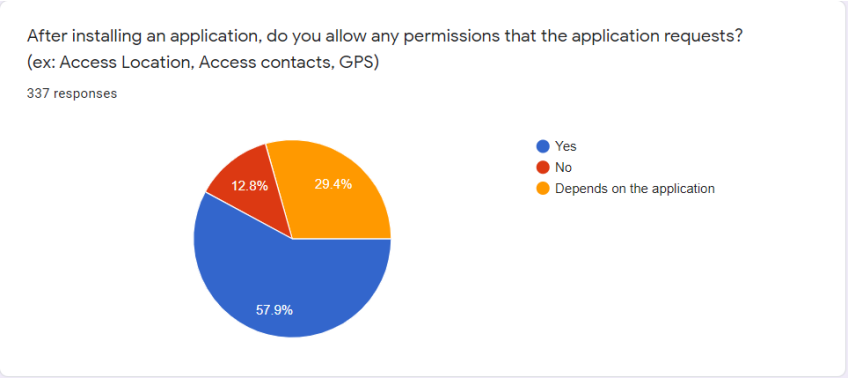


Figure 5.7: Survey Seventh Evaluation

- There are 195 out of 337 responses (57.9%) of users that **Allow** any permissions that the application requests after installing an application in fig.5.7
- There are 99 out of 337 responses (29.4%) of users that **May allow** any permissions that the application requests after installing an application which depends on the application in fig.5.7

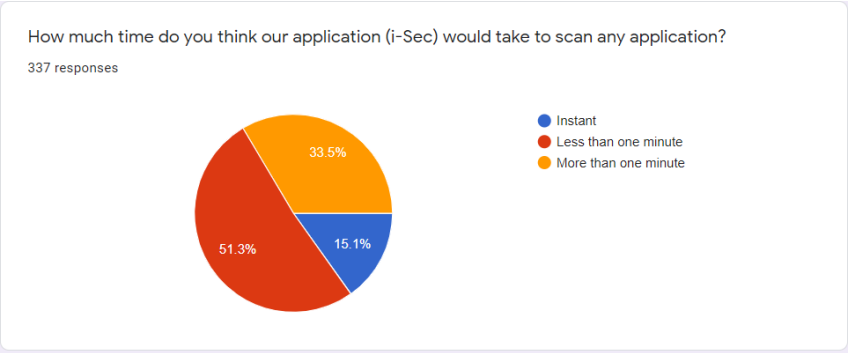


Figure 5.8: Survey Eighth Evaluation

- There are 51 out of 337 responses (15.1%) of users that think that our application would **Instantly** scan any application in fig.5.8
- There are 113 out of 337 responses (33.5%) of users that think that our application would take **More Than 1 minute** to scan any application in fig.5.8
- There are 173 out of 337 responses (51.3%) of users that think that our applications would take **Less than 1 minute** to scan any application in fig.5.8

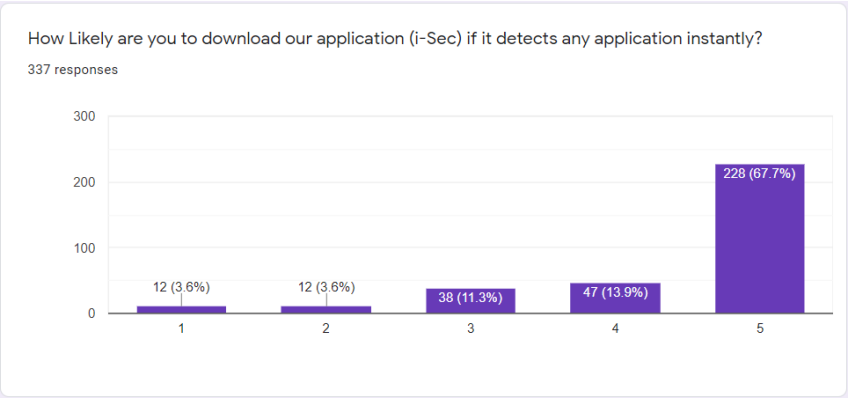


Figure 5.9: Survey Ninth Evaluation

- There are 228 out of 337 responses (67.7%) of users are very likely to download our application (i-Sec) if it detects any application instantly in fig.5.9
- There are 12 out of 337 responses (3.6%) of users are not likely to download our application (i-Sec) if it detects any application instantly in fig.5.9

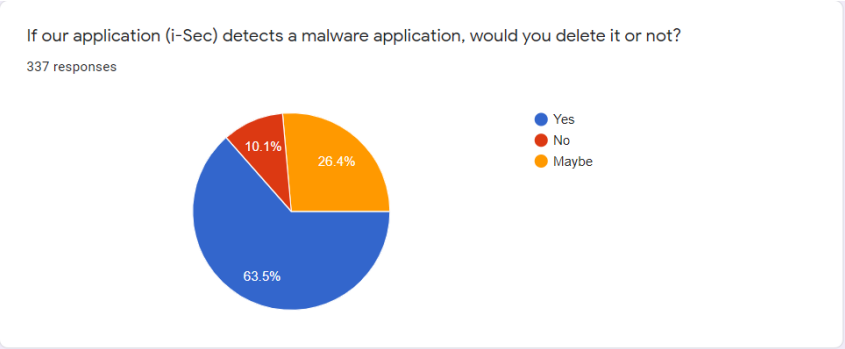


Figure 5.10: Survey Tenth Evaluation

- There are 214 out of 337 responses (63.5%) of users that would **Delete** an application if our application (i-Sec) detects a malware application in fig.5.10
- There are 89 out of 337 responses (26.4%) of users that **May Delete** an application if our application (i-Sec) detects a malware application in fig.5.10
- There are 34 out of 337 responses (10.1%) of users that would **not Delete** an application if our application (i-Sec) detects a malware application in fig.5.10

## 5.3 Performance analysis

- The application shows the user results instantly.
- The application using 140 MB from the mobile ram while scanning an application.

## 5.4 Summary

Our system succeeded in detecting a malware application after installing it on our mobile application, We got good user evaluation feedback in the survey evaluation. Our application shows an instant result of a specified selected application using 140 MB from mobile ram.

# Chapter 6

# Conclusion

## 6.1 Introduction

Our document proposed a system for detecting malicious Android applications by tracking their static and dynamic behavior based on their permissions and API calls. We produce a mobile application combined with our machine learning model that scans all the applications to detect malware and benign ones. We performed a precise feature selection over two datasets. Still, we got an accuracy with **Malgenome** dataset containing API calls with an average of 97% with the decision tree classifier and an accuracy with **Defensedroid** dataset containing permissions with an average 93% with CNN classifier these results were performed due to their high accuracy results and less processing time. The CNN classifier produced a significant training accuracy result with 93% after converting the permissions into images. We produce a mobile application using java programming language.

## 6.2 Contributions and Reflections

Our innovative aspect in this project that we classified our model by converting the permissions of the mobile application into images and train them by CNN classifier with significant accuracy 93%. We linked our model with the mobile app and extracted the permissions of the already installed applications on the device and succeeded in our goal that we detected malware applications on the device. This project has a great impact on us, We benefited from its good

teamwork and a successful classification for malware applications as well as we finished our work in a short period of time, the project went smoothly as we worked as a good team.

## 6.3 Future Directions

Our goal in the future that we aim to enhance our application user interface by adding scanning animation for applications and extra functionalities ex: deleting the malicious apps found after scanning.

## 6.4 Summary

We propose a mobile application for detecting malicious applications by tracking their permissions and API calls, we converted permissions of applications into images and classify it with significant accuracy. We succeeded in our detection for malware applications and we will enhance our application user interface with animation in the future.

# Bibliography


[1] ARP, D., SPREITZENBARTH, M., HUBNER, M., GASCON, H., RIECK, K., AND SIEMENS, C. Drebin: Effective and explainable detection of android malware in your pocket. In *Ndss* (2014), vol. 14, pp. 23–26.

[2] BHAT, P., DUTTA, K., AND SINGH, S. Mapldroid: Malicious android application detection based on naive bayes using multiple. In *2019 2nd International Conference on Intelligent Communication and Computational Techniques (ICCT)* (2019), IEEE, pp. 49–54.

[3] CYBERSECURITY, S. https://new.siemens.com/global/en/company/topic-areas/cybersecurity.html, 2020.

[4] DATASET, A. P. http://pralab.diee.unica.it/en/androidpraguarddataset, 2018.

[5] DEFENSEDROID. https://www.kaggle.com/defensedroid/android-malware-detection, 2021.

[6] FAIZ, M. F. I., AND HUSSAIN, M. A. Hybrid classification model to detect android application-collusion. In *2020 43rd International Conference on Telecommunications and Signal Processing (TSP)* (2020), IEEE, pp. 492–495.

[7] FOR ANDROID, M. S. R. https://www.whistleout.com/cellphones/guides/guide-to-buying-cheap-android-phones-specs-and-tips, 2017.

[8] GARZOTTO, F. A user-friendly enterprise framework for data intensive web applications. In *IRI-2005 IEEE International Conference on Information Reuse and Integration, Conf, 2005.* (2005), IEEE, pp. 415–420.

[9] GOOGLE. https://developer.android.com/distribute/console, 2021.

[10] JANNAT, U. S., HASNAYEEN, S. M., SHUHAN, M. K. B., AND FERDOUS, M. S. Analysis and detection of malware in android applications using machine learning. In *2019 International Conference on Electrical, Computer and Communication Engineering (ECCE)* (2019), IEEE, pp. 1–7.

[11] LI, X., LIU, J., HUO, Y., ZHANG, R., AND YAO, Y. An android malware detection method based on androidmanifest file. In *2016 4th International Conference on Cloud Computing and Intelligence Systems (CCIS)* (2016), IEEE, pp. 239–243.

[12] LIBRARIES, . O.-S. P. https://www.mygreatlearning.com/blog/open-source-python-libraries/l, 2020.

[13] MA, Z., GE, H., LIU, Y., ZHAO, M., AND MA, J. A combination method for android malware detection based on control flow graphs and machine learning algorithms. *IEEE access 7* (2019), 21235–21245.

[14] MOHAMED, S. E., ASHAF, M., EHAB, A., SHEREEF, O., METWAIE, H., AND AMER, E. Detecting malicious android applications based on api calls and permissions using machine learning algorithms. In *2021 International Mobile, Intelligent, and Ubiquitous Computing Conference (MIUCC)* (2021), pp. 1–6.

[15] MOUMANE, K., IDRI, A., AND ABRAN, A. Usability evaluation of mobile applications using iso 9241 and iso 25062 standards. *SpringerPlus 5*, 1 (2016), 548.

[16] MURTAZ, M., AZWAR, H., ALI, S. B., AND REHMAN, S. A framework for android malware detection and classification. In *2018 IEEE 5th International Conference on Engineering Technologies and Applied Sciences (ICETAS)* (2018), IEEE, pp. 1–5.

[17] MVC: MODEL, VIEW, C. https://www.codecademy.com/articles/mvc, 2020.

[18] OF ANDROID MALWARE, S. A. https://www.kaggle.com/goorax/static-analysis-of-android-malware-of-2017.

[19] PROJECT, A. M. G. http://www.malgenomeproject.org/.

[20] SEIF ELDEIN MOHAMED, AMR EHAB, M. A. O. S. https://play.google.com/store/apps/details?id=com.isec.androidmalwaredetection, 2021.

[21] SOMARRIBA, O., ZURUTUZA, U., URIBEETXEBERRIA, R., DELOSIÈRES, L., AND NADJM-TEHRANI, S. Detection and visualization of android malware behavior. *Journal of Electrical and Computer Engineering 2016* (2016).

[22] ZHANG, Y., YANG, M., XU, B., YANG, Z., GU, G., NING, P., WANG, X. S., AND ZANG, B. Vetting undesirable behaviors in android apps with permission use analysis. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security* (2013), pp. 611–622.

[23] ZHOU, Y., AND JIANG, X. http://www.malgenomeproject.org/, 2021.

# Appendix A

# Git Repository

Link: https://github.com/OmarShereef/Graduation-Project

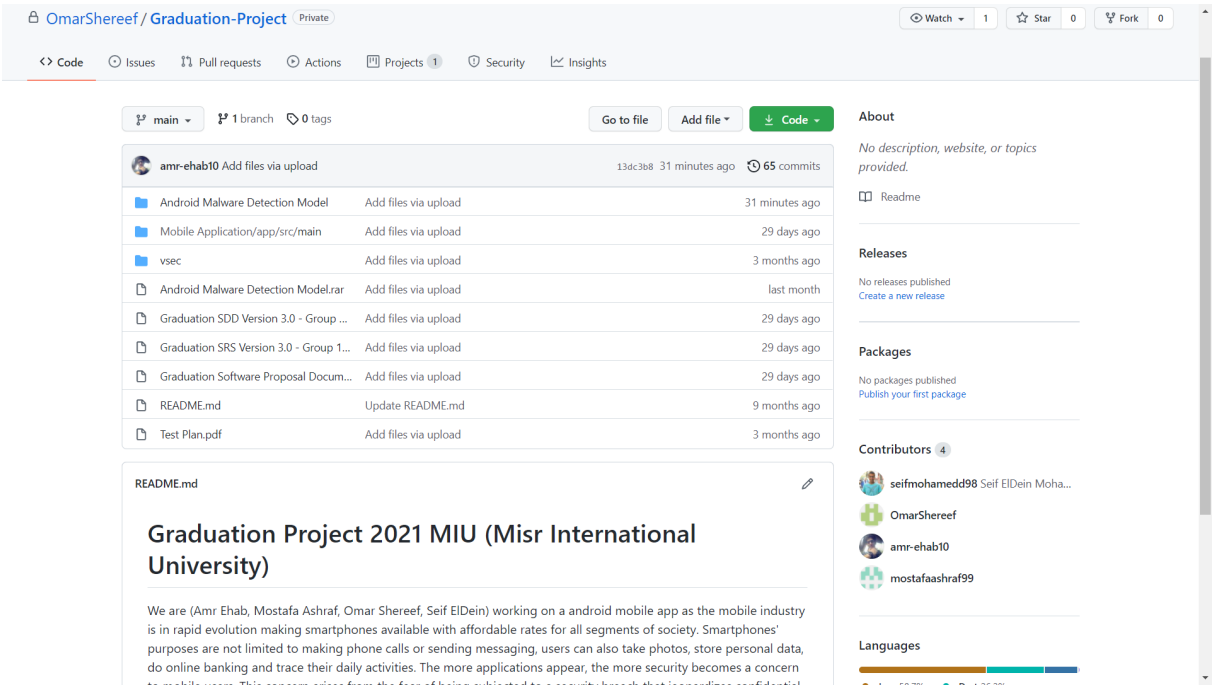


Figure A.1: GitHub Project

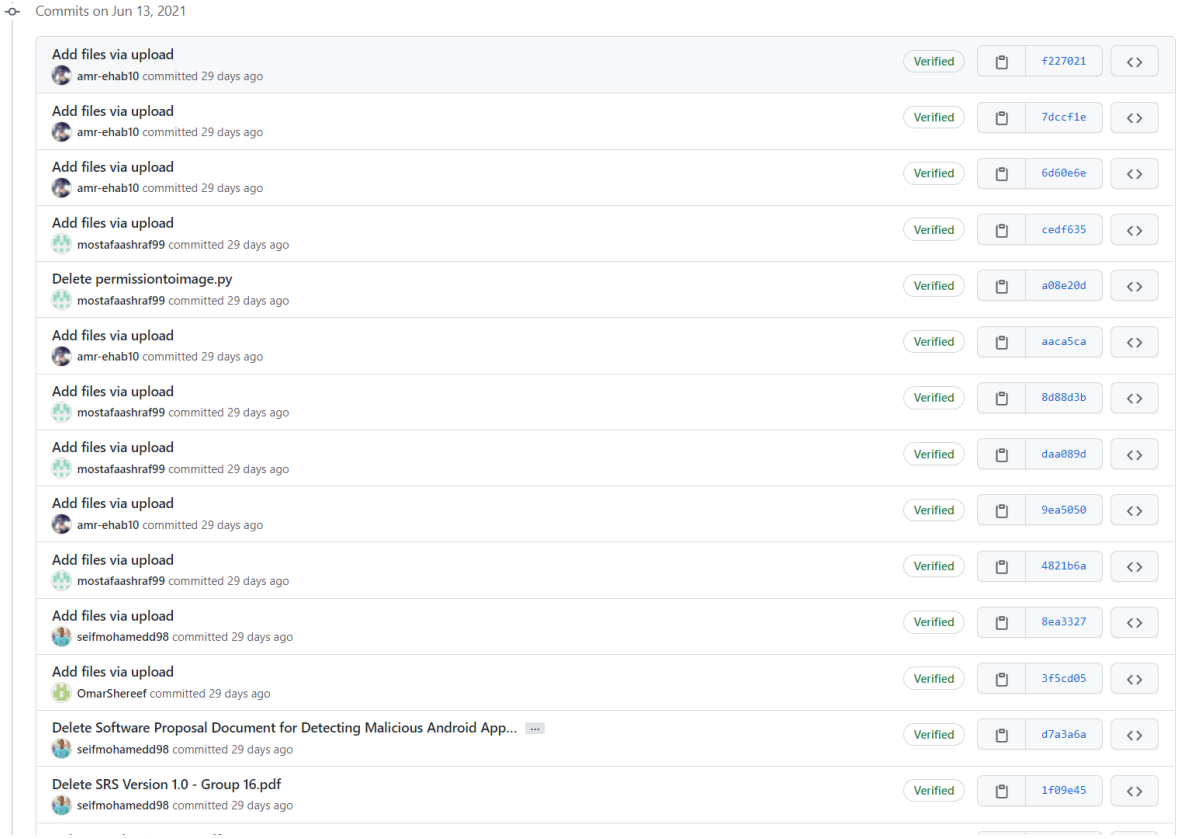


Figure A.2: GitHub Contributors

# Appendix B

# User Manuals

- This framework is "user-friendly", easy to use by presenting the user with a clean design, and providing them a responsive interface.[8].
- Any smart mobile device that uses the Android operating system with minimum hardware specifications as mentioned in section 3.4.2.
- By clicking on the application. Our system will navigate the user to a result screen of the application that he selected and shows him if the application either malware or benign in section 4.4.3.

# Appendix C

# User Evaluation Questionnaire

**1) Suppose you are an android user, do you see that our application is useful to download on your device to protect your own personal information?**

Answer: He said as he android user that our application will be very important because there is a large part in Android not secure with 100% so our application will help.

**2) What is your opinion about the scanning speed for detecting Malware/Benign?**

Answer: As one said the best thing in our application is the performance of detection is fast. Once he selects an application to scan our application shows the result quickly.

**3) Does our application easily to use?**

All the people that we asked them they said that our application is user friendly and they did not face any problem while using.

**4) Is the GUI of our application comfortable for you?**

They said the GUI is good but it can be better by edit the margins and alignments and font to get a better GUI.

**5) What modification can be added for the application from your point of view?**

They said that we can add functionality that if the application that being scanned by the user if it is malware so he can delete the malware application from our application. And there is another opinion that we detect applications once it is installed on the device without he make a scan from our application.

**6) Rate our appliction out of 10**

We get rates from 8 to 9 out of 10.