Software Requirement Specification Document for Detecting Malicious Android Applications

Seif ElDein Mohamed, Amr Ehab, Mostafa Ashraf, Omar Shereef
Supervised by: Dr.Eslam Amer
Assistant Supervisors: Eng.Haytham Metawie, Eng.Mostafa Badr

December 28, 2020

Version	Date	Reason for Change
1.0	17-Dec-2020	SRS First version's specifications are defined.
1.1	25-Dec-2020	Update Functional Requirements. Class Diagram identified
1.2	27-Dec-2020	CLI Updated.

Table 1: Document version history

GitHub: https://github.com/OmarShereef/Graduation-Project.git

Abstract

Nowadays, the mobile industry is in rapid evolution making smartphones available with affordable rates for all segments of society. Smartphones' purposes are not limited to making phone calls or sending messaging, 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, location, credentials etc. Malware applications which 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 different machine learning algorithms to distinguish between benign and malicious

1 Introduction

Using smartphones has become an indispensable part of our daily lives. By December 2018, Android occupies 75.16% of the market share of mainstream portable working framework[1].

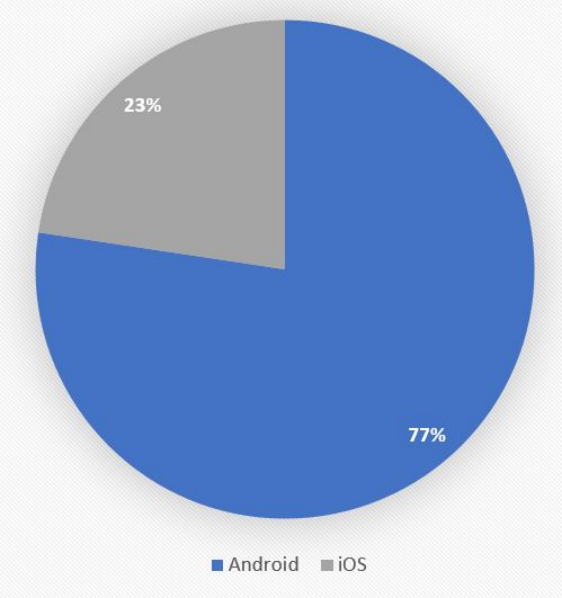

Figure 1: Most popular mobile operating system

More than one million Android applications such as We-Chat, Tik-Tok, and mobile banking applications are commonly used daily[2], that keep on playing a progressively significant role found in Android markets such as Google Play. The vast majority of these applications have breached the users' private data, for example, their location, credit cards, and contacts data. Practically all applications can access the users' private information, in spite of the fact that this gives users with better-customized administrations. It might likewise result in data spillage of private information and financial misfortune. Moreover, Android malware applications continue rising perpetually, this security issue has been broadly expanding in the business and academic fields. A huge group

of researches against malware have been conducted. As of now, the two primary kinds of identification techniques are dynamic and static analyses. Each approach has benefits and deficiencies. Static analysis techniques, for example, PApriori [3], and DREBIN [4] examine applications without executing them. Be that as it may, the strategies cannot protect against anti-decompiling and obfuscation. On the other hand, dynamic analysis techniques for example, VetDroid, [5] run the applications in real time to detect malicious behaviour, yet it is hard to capture all the execution behaviour. With malware being quickly developing, adaptive machine learning techniques are utilized to perform Android malicious detection. Therefore, representing malicious behavior accurately requires extracting the most important features to improve malware detection efficiency.

1.1 Purpose of this document

The main purpose of this Software Requirements Specification document is to outline the fundamental requirements of our system **'Detecting Malicious Android Applications'**. The system aims to detect any malware in Android device systems to protect our clients' system. Our model will be implemented in a mobile application that runs on Android systems. We also provide a fulfilled description of each single stage input, Post-Condition, and algorithms used in this stage. Along with a full illustration for each stage's requirements and development process.

1.2 Scope of this document

This document targets the clients' and companies' administration which has a role in the companies' business flow. The document provides the detailed functional and non-functional requirements, as well as the main functionalities of our system presented in the different stages of the system development.

1.3 System Overview

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

- The input stage, in which the data is priorly, collected From Androzoo[6] that contains more than seven million unique Android applications. Also, it has permissions and API calls for benign and malicious applications.
- In pre-processing phase, feature extraction is performed to reduce the dimensionality of the raw data in order for our model to require less computation power. Normalization, and standardization will also be applied on the data-set if needed.
- In processing stage, the data-set is split into three subsets training, validation and testing respectively. We focus on training the different models using training set, our malware detection model will be trained and tested on the data-set by some algorithms such as KNN, and SVM[7],by using dynamic analysis which is executed during the applications' runtime.
- In the last stage, we will classify each selected application as either benign, or which of the nine malware families (virus, worms, ransomware, spyware, etc.) these applications belong to.

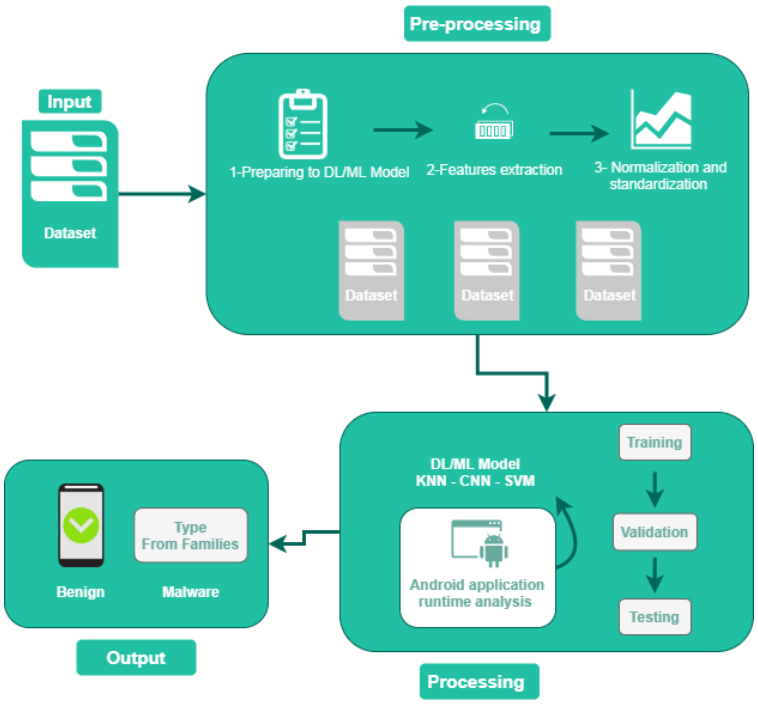

Figure 2: System Overview

1.4 System Scope

The proposed system classifies malware behaviours of Android Applications with efficiency and viability, utilizing machine learning techniques and dynamic behaviour analysis to naturally identify malware in all file types.

1.5 Business Context

Malware detection of Android systems is one of the most significant cybersecurity tasks for clients and organizations[8]. 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.

2 Similar Systems

2.1 Academic

Faiz[9], proposed a system that detects Android malware applications by hybrid classification with K-means clustering algorithm and support vector machine (SVM). The researchers used two data-sets[10] [11]. From the first data-set, they generate two data-sets 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 data-set 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[12], proposed a system that detects malicious Android applications based on Naïve Bayes model. They got two data-sets, DREBIN[10] and PRAGuard[13] data-sets that contain 2870 applications. From 2870 applications they got 1472 malicious applications and 1398 benign applications. The researchers solve the problem by malware detection technique called MaplDroid uses a static analysis approach with Naïve Bayes model analysis. Also, they applied another classification technique such as random forest to bring out comparative analysis. MaplDroid achieved a score of 99.12%. An obvious shortcoming of this conducted research is the small scale of the used data-set.

Umme Sumaya Jannat[14], proposed a system that analyses and detects Android malware using machine learning. The researchers solve the problem in two ways by dynamic analysis and static analysis. They get the best result in the dynamic analysis by applying Random Forest (RF) algorithm that is an extended version from Decision Tree (DT) algorithm. The result of dynamic analysis has exceeded the static analysis accuracy scores over 93% accuracy. Also, the researchers have used different data-sets for static analysis and dynamic analysis. 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 data-set from Kaggle that contains 4000 malicious applications in JSON format. And in dynamic analysis, they used MalGenome[15] data-set which contains 1260 malicious applications that belong to 49 different malware families.

Zhuo Ma[16], proposed a system that detects malicious Android applications. The researchers used control flow graphs and machine learning algorithms. They build chronological data-sets and train them by making a Long short-term memory algorithm which is a recurrent neural network. The researchers de-compile and set up 3 kinds of systems: 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 data-set.

Muhammad Murtaz[17], proposed a system that scopes to detect 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 assessment exhibits the proposed incorporate set has more than 94 significant for real malware acknowledgment frameworks.

2.2 Business Applications

1. **Malware Bytes[18] for Android malware detection[19]**: Scan blocker and privacy protector. The application scan malware, and identifies ransomware, PUP's, and phishing scams.

    - Distinguishes ransomware.
    - Behaviors protection review for all applications.
    - Search for malware and adware and remove them.

Figure 3: Malware bytes application

2. **Norton 360[20] for Android malware detection[21]**: Provides powerful layers of security to Android mobiles against infections, ransomware, malware, and others like online shopping.

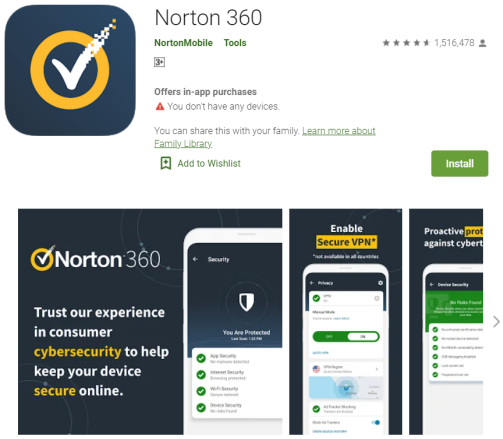

Figure 4: Norton 360 application

3 System Description

3.1 User Problem Statement

Malicious applications nowadays set out to decimate our information on mobile devices, for example misusing user's private information. Many malware detection applications develop their methods based on permissions or when the harmful permissions committed by malicious applications are similar to those benign, this limitations is considered as too few information for classification. We propose detecting the malicious applications by analysing their dynamic behavior and system calls.

3.2 User Objectives

- To use different machine learning algorithms to detect malicious Android applications based on their run-time behaviour.
- To provide firewall from breaching Android users' critical data.
- To secure the personal data information.
- Malware detection of Android applications based on their permissions, and API call sequence. In case of malicious applications, specify the malware family.

3.3 User Characteristics

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

3.4 System Context

- The End User triggers the application and Scan the device apps.
- The system identify the permissions, API calls and system calls.
- The system apply the ML algorithms of the Model and Test the apps whether there is Benign or Malicious applications.

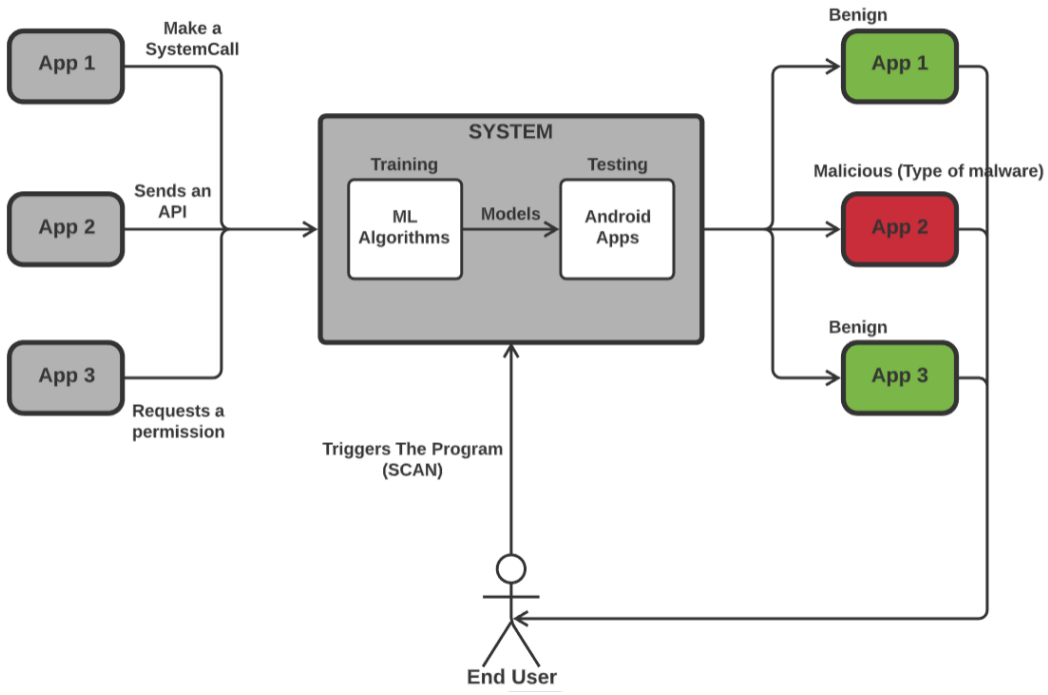

Figure 5: System Context

4 Functional Requirements

4.1 System Functions

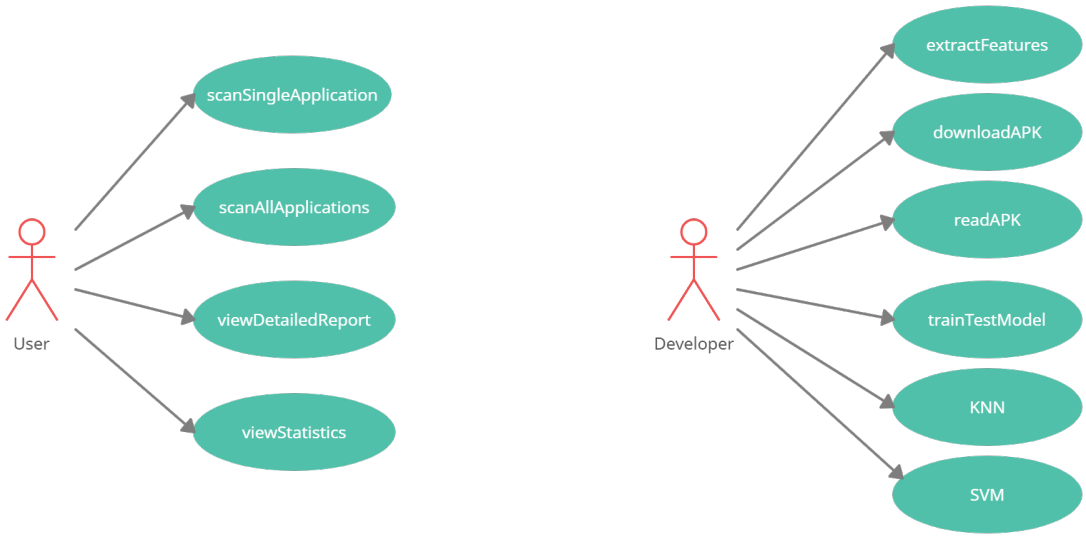

Figure 6: Use Case Diagram

4.2 Detailed Functional Specification

4.2.1 Building model

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 (Androzoo)
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 2: Extract Features Function Description

Function ID	FR02
Name	downloadAPK
Description	Downloads an APK from the data-set
Pre-condition	Internet connection
Input	Sha256 (from data-set), API key of the APK, download file path
Action	Uses the sha256 of APK in the data-set, and the given API key to download APK to specified location
Output	Boolean
Post-condition	N/A
Dependency	N/A

Table 3: Download APK Function Description

Function ID	FR03
Name	readAPK
Description	Reads and saves APK data to a new data-set
Pre-condition	APK with corresponding SHA-256 must exist
Input	SHA-256 (from data-set), APK location, file save location
Action	Uses the SHA-256 of APK in the data-set, and the APK location to read APK's data (Permissions, Package name, API calls, etc.) Then, save this data to a new data-set in specified location
Output	Boolean
Post-condition	N/A
Dependency	FR02 (downloadAPK)

Table 4: Read APK Function Description

Function ID	FR04
Name	trainTestModel
Description	Training and testing the new data-set
Pre-condition	Saving the APK data from old data-set to new data-set
Input	Data-set, train size
Action	Splits the new data-set into training and testing subsets, and applies machine learning algorithms to train and test the data
Output	Benign or Malicious (Android malware family)
Post-condition	N/A
Dependency	FR01 (extractFeatures), and FR03 (readAPK)

Table 5: Train and Test Model Function Description

Function ID	FR05
Name	KNN
Description	Checks if application is malicious or benign
Pre-condition	Data-set must be trained and tested
Input	Permissions, and API call sequence
Action	Applies KNN (K-Nearest Neighbors) algorithm to categorize the APK as either malicious or benign, if malicious, specifies which Android malware family the application belongs to
Output	Benign or Malicious (Android malware family)
Post-condition	N/A
Dependency	FR03 (readAPK), and FR04 (trainTestModel)

Table 6: Classify APK using KNN Function Description

Function ID	FR06
Name	SVM
Description	Checks if application is malicious or benign
Pre-condition	Data-set must be trained and tested
Input	Permissions, and API call sequence
Action	Applies SVM (Support Vector Machine) algorithm to categorize the APK as either malicious or benign, if malicious, specifies which Android malware family the application belongs to
Output	Benign or Malicious (Android malware family)
Post-condition	N/A
Dependency	FR03 (readAPK), and FR04 (trainTestModel)

Table 7: Classify APK using SVM Function Description

4.2.2 End User

Function ID	FR07
Name	scanSelectedApplications
Description	Scans the applications the user selected for any malicious behavior
Pre-condition	N/A
Input	Applications' package names
Action	Classify each application as either benign or malicious, if malicious, specify which malware families it belongs to
Output	N/A
Post-condition	Navigates to results page
Dependency	N/A

Table 8: Scan Selected Applications Function Description

Function ID	FR08
Name	scanAllApplications
Description	Scans the user's device for any malicious applications
Pre-condition	N/A
Input	N/A
Action	Classify each application as either benign or malicious, if malicious, specify which malware families it belongs to
Output	N/A
Post-condition	Navigates to results page
Dependency	N/A

Table 9: Scan All Applications Function Description

Function ID	FR09
Name	viewResults
Description	Views the results of scans the user initiated
Pre-condition	User must scan first
Input	N/A
Action	Displays a list of malicious applications if any exists, and their corresponding malware family type
Output	List of malicious applications if any exists
Post-condition	N/A
Dependency	FR07 (scanAllApplications) or FR08 (scanSelectedApplications)

Table 10: View Results Page Function Description

Function ID	FR10
Name	viewAppDetails
Description	Views the details of a malicious application
Pre-condition	User must scan first
Input	N/A
Action	Displays all malicious application information
Output	Displays important information about the malicious application such as malware family type, Android permissions, activities, etc.
Post-condition	N/A
Dependency	FR09 (viewResults)

Table 11: View Application Details Page Function Description

Function ID	FR11
Name	viewStatistics
Description	Views statistics about the application life time
Pre-condition	User must scan first
Input	N/A
Action	Navigates to a page that displays graphs and charts representing important statistics of the application activity on user's device
Output	Displays various statistics such as no. of scans done, no. of malicious applications found, most common malware families, etc.
Post-condition	N/A
Dependency	FR07(scanSelectedApplications) or FR08(scanAllApplications)

Table 12: View Statistics Function Description

Function ID	FR12
Name	viewMalwareInformation
Description	Views general information about Android malware to make users aware and careful about which applications to install
Pre-condition	N/A
Input	N/A
Action	Displays a list of Android malware families, and general Android malware information
Output	Displays the types of the different Android malware families, their dangers, what data they can access, and general Android malware information.
Post-condition	N/A
Dependency	N/A

Table 13: View Malware Information Function Description

5 Interface Requirements

5.1 User Interfaces

5.1.1 GUI

Figure 7: Home Screen

Figure 8: Scanning applications screen

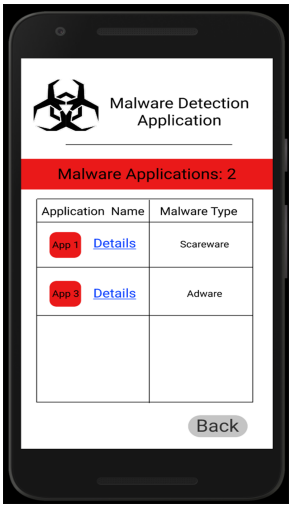

Figure 9: Details screen

Figure 10: Statistics screen

5.1.2 CLI

```
def isPermission(columnName):
    flag = False
    for character in columnName:
        if character.isupper():
            flag = True
        elif character == '_':
            flag = True
        else:
            return False
    return flag

def plotGraph(df, type):
    ig, axs = plt.subplots(ncols=2, sharex=False, figsize=(20, 20))
    pd.Series.sort_values(df[df['class'] == 0].sum(axis=0), ascending=False)[:10].plot.bar(ax=axs[0])
    pd.Series.sort_values(df[df['class'] == 1].sum(axis=0), ascending=False)[1:11].plot.bar(ax=axs[1], color="red")

    axs[0].set_title("Benign")
    axs[1].set_title("Malicious")

    axs[0].set(xlabel="Most common " + type, ylabel="Number of occurences")
    axs[1].set(xlabel="Most common " + type, ylabel="Number of occurences")

    fig = plt.gcf()
    fig.set_size_inches(15.5, 8)
    plt.gcf().subplots_adjust(bottom=0.5)
    pylab.show()
```

Figure 11: CLI (1)

```
def trainTestAlgorithms(df):
    # Splitting dataset into train and test for both algorithms
    X_train, X_test, y_train, y_test = train_test_split(df.iloc[:, 1:3799], df['class'], test_size=0.20,
                                                        random_state=42)

    # Naive Bayes algorithm
    gnb = GaussianNB()
    gnb.fit(X_train, y_train)

    # prediction
    pred = gnb.predict(X_test)

    # accuracy
    accuracy = accuracy_score(pred, y_test)
    print("naive_bayes")
    print(accuracy)
    print(classification_report(pred, y_test, labels=None))

    # kneighbors algorithm
    for i in range(3, 11, 2):
        neigh = KNeighborsClassifier(n_neighbors=i)
        neigh.fit(X_train, y_train)

        # prediction
        pred = neigh.predict(X_test)

        # accuracy
        accuracy = accuracy_score(pred, y_test)
        print("kneighbors {}".format(i))
        print(accuracy)
        print(classification_report(pred, y_test, labels=None))
        print("")


df=pd.read_csv("malgenome-215-dataset-1260malware-2539-benign.csv")
```

Figure 12: CLI (2)

```
#Creating new data frames for each of permissions and API calls

apiCallsDF = df
permissionDF = df

for col in df.columns:
    if col=="class":
        break;
    elif not isPermission(col):
        permissionDF = permissionDF.drop([col],axis=1)
    elif isPermission(col):
        apiCallsDF = apiCallsDF.drop([col],axis=1)

print("Most common benign permissions")
print(pd.Series.sort_values(permissionDF[permissionDF['class']==0].sum(axis=0), ascending=False)[:10])

print("Most common malicious permissions")
print(pd.Series.sort_values(permissionDF[permissionDF['class']==1].sum(axis=0), ascending=False)[1:11])

print("Most common benign API calls")
print(pd.Series.sort_values(apiCallsDF[apiCallsDF['class']==0].sum(axis=0), ascending=False)[:10])

print("Most common malicious API calls")
print(pd.Series.sort_values(apiCallsDF[apiCallsDF['class']==1].sum(axis=0), ascending=False)[1:11])

#Plotting most common used permissions and API calls in both benign and malicious apps

plotGraph(permissionDF, "permissions")
plotGraph(apiCallsDF, "API calls")

#Training and Testing algorithms

print("Permissions Training and Testing")
trainTestAlgorithms(permissionDF)

print("API calls Training and Testing")
trainTestAlgorithms(apiCallsDF)
```

Figure 13: CLI (3)

5.2 API

Python Libraries[22] :

- NumPy: Support for large multidimensional arrays.
- Pandas: Provision of easy data structure and quicker data analysis for Python.
- Scikit-learn: It consists of numerous clustering, regression and classification algorithms.
- apkparser: Reads the content of the APK files[23].

6 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.[24].
- Any smart mobile device that uses the Android operating system with minimum hardware specifications as mentioned in **section 6.2** later.

6.1 Standards Compliance

Using the usability ISO 9241 and ISO 25062 standards for Android application[25] to develop our user interface.

6.2 Hardware Limitations

These are the minimum specifications of an Android smartphone required for our application [26].

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

7 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, maintainability, and portability are the non-functional requirements that the framework attempts to accomplish.

7.1 Security

This component is achieved by the framework by encrypting all the information before being put away in the database. In addition, hashing private information will be compulsory to guarantee that the put-away information is made sure about.

7.2 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 to test all documents[27].

7.3 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.

7.4 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.

8 Data Design

8.1 Data Description

The data-set in bulding the model is called "Androzoo"[28], it is a collection of hash codes(SHA-256) of APKs, that we can use to download each APK in the data-set. Each APK consists of many data, such as Package Name, Permissions, API calls[6], verification code, etc.

- The data-set is originated from a paper called "Androzoo". It's an excel file with 7M (million) APK samples.
- We used a Python library called "apkparser" to read, extract, and save the data from each APK.
- The data-set consists of 11 columns excluding the index (sha256, sha1, md5, dex_date, apk_size, pkg_name, vercode, vt_detection, vt_scan_date, dex_size ,markets).
- Anyone with an Android device is a possible user.

9 Preliminary Object-Oriented Domain Analysis

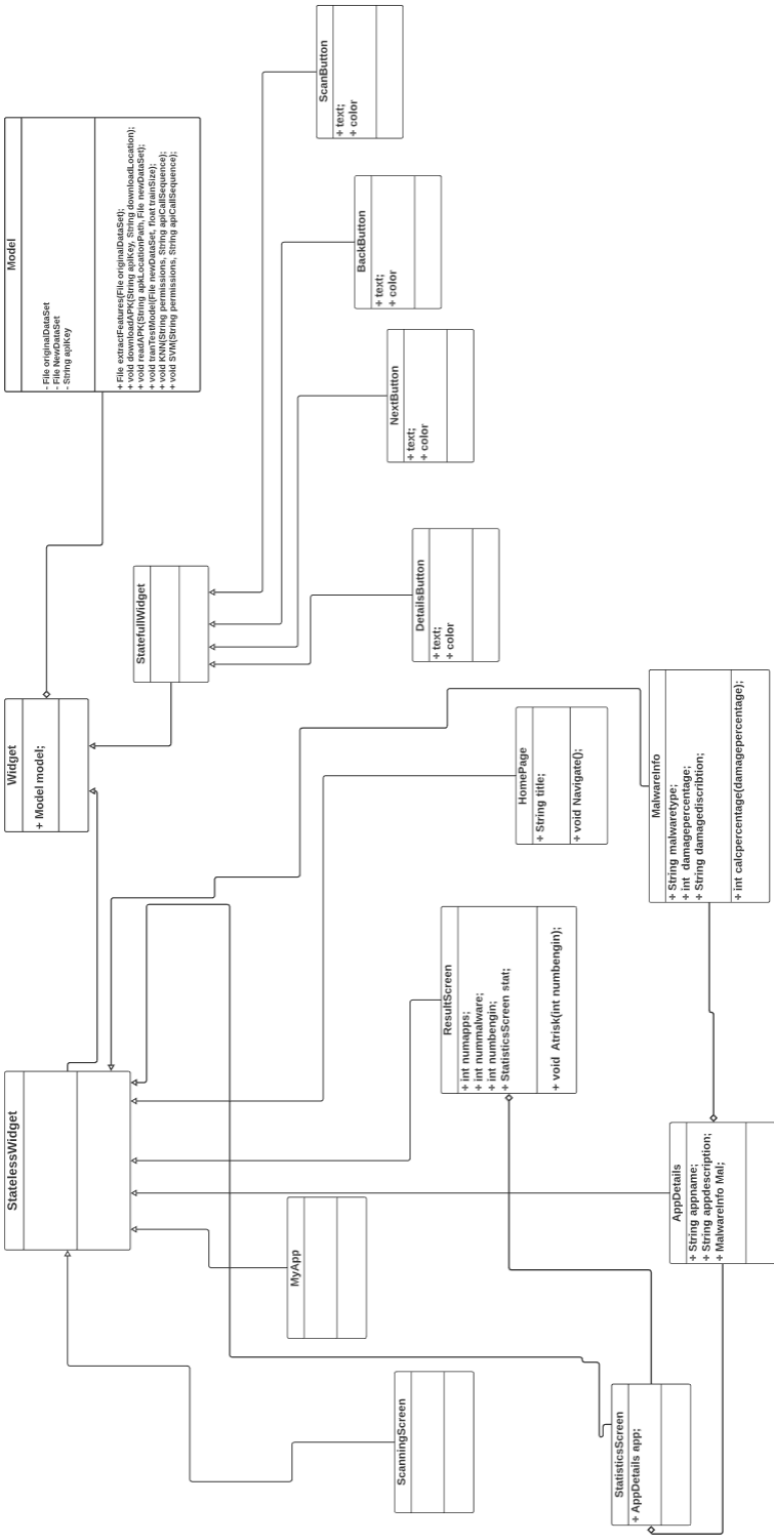

Figure 14: Class Diagram

9.1 Inheritance Relationships

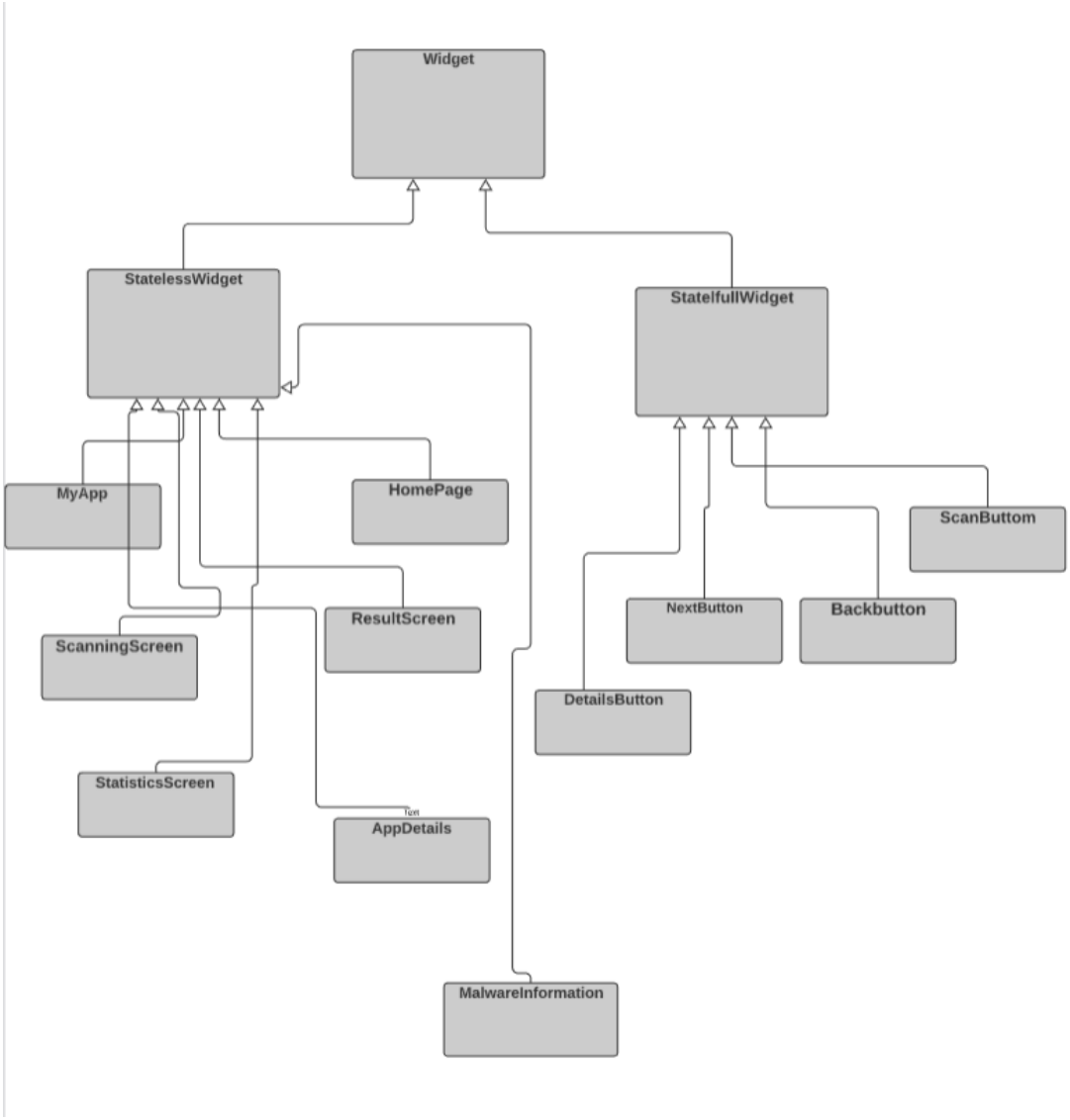

Figure 15: Mobile app Inheritance Diagram

9.2 Class descriptions

Abstract or Concrete:	Concrete Class
List of Superclasses	N/A.
List of Subclasses	StatelessWidget / StatefullWidget Classes.
Purpose	Building the core of the application.
Collaborations	Collaborates with(Model) by Aggregation To this class.
Attributes	(Model model) is instance of class Model.
Operations	N/A.
Constraints	N/A.

Table 14: Widget Class

Abstract or Concrete:	Concrete Class
List of Superclasses	Widget Class.
List of Subclasses	ScanningScreen/ MyApp/ AppDetails/ ResultScreen/ HomePage/ StatisticsScreen/ MalwareInfo Classes.
Purpose	Building the UI (User Interface) of the application.
Collaborations	N/A.
Attributes	N/A.
Operations	N/A.
Constraints	N/A.

Table 15: StatelessWidget Class

Abstract or Concrete:	Concrete Class
List of Superclasses	Widget Class.
List of Subclasses	DetailsButton/ QuitButton/ BackButton/ ScanButton Classes.
Purpose	Navigates through the pages of the application.
Collaborations	N/A.
Attributes	N/A.
Operations	N/A.
Constraints	N/A.

Table 16: StatefullWidget Class

Abstract or Concrete:	Concrete Class
List of Superclasses	N/A.
List of Subclasses	N/A.
Purpose	Constructs the machine learning algorithms for testing the applications to set if it is Malicious or Benign.
Collaborations	Collaborates with Widget Class by Aggregation from this class.
Attributes	String apikey/ File originaldata-set/ File newdata-set.
Operations	File extractFeatures(File originaldata-set)/ void downloadAPK(String apiKey, String downloadLocation)/ void readAPK(String apkLocation-Path, File newdata-set)/ oid tranTestModel(File newdata-set, float train-Size)/ void KNN(String permissions, String apiCallSequence)/ void SVM(String permissions, String apiCallSequence).
Constraints	N/A.

Table 17: Model Class

Abstract or Concrete:	Concrete Class
List of Superclasses	StatelessWidget Class.
List of Subclasses	N/A.
Purpose	It is the main class for Building the UI(User Interface) of the application.
Collaborations	N/A.
Attributes	N/A.
Operations	N/A.
Constraints	N/A.

Table 18: MyApp Class

Abstract or Concrete:	Concrete Class
List of Superclasses	StatelessWidget Class.
List of Subclasses	N/A.
Purpose	It is used for showing the scanning results of the applications.
Collaborations	N/A.
Attributes	N/A.
Operations	N/A.
Constraints	N/A.

Table 19: ScanningScreen Class

Abstract or Concrete:	Concrete Class
List of Superclasses	StatelessWidget Class.
List of Subclasses	N/A.
Purpose	It is used for showing the statistics of the Scanned applications.
Collaborations	Collaborates with ResultScreen by Aggregation From this Class and with AppDetails by Aggregation To this class.
Attributes	(AppDetails app) is instance of Class AppDetails.
Operations	N/A.
Constraints	N/A.

Table 20: StatisticsScreen Class

Abstract or Concrete:	Concrete Class
List of Superclasses	StatelessWidget Class.
List of Subclasses	N/A.
Purpose	It is used for showing the Details of the applications.
Collaborations	Collaborates with StatisticsScreen by Aggregation From this Class and with MalwareInfo by Aggregation To this class.
Attributes	String appname/ String appdescribtion/ (MalwareInfo Mal) is instance of class MalwareInfo.
Operations	N/A.
Constraints	N/A.

Table 21: AppDetails Class

Abstract or Concrete:	Concrete Class
List of Superclasses	StatelessWidget Class.
List of Subclasses	N/A.
Purpose	It is used for showing all the results of the Scanned applications.
Collaborations	Collaborates with StatisticsScreen by Aggregation To this Class.
Attributes	int numapps/ int nummulware/ int numbenign/ (StatisticsScreen stat) is instance of Class StatisticsScreen.
Operations	void Atrisk(int numbengin).
Constraints	N/A.

Table 22: ResultScreen Class

Abstract or Concrete:	Concrete Class
List of Superclasses	StatelessWidget Class.
List of Subclasses	N/A.
Purpose	To Identify the type and damage percentage represented by the malware and the discribtion.
Collaborations	Collaborates with AppDetails by Aggregation from this Class.
Attributes	String malwaretype/ int damagepercentage/ String damagediscribtion.
Operations	int calcpercentage(damagepercentage).
Constraints	N/A.

Table 23: MalwareInfo Class

Abstract or Concrete:	Concrete Class
List of Superclasses	StatelessWidget Class.
List of Subclasses	N/A.
Purpose	To present the UI of the application homepage and start scanning.
Collaborations	N/A.
Attributes	String title.
Operations	void Navigate().
Constraints	N/A.

Table 24: HomePage Class

Abstract or Concrete:	Concrete Class
List of Superclasses	StatefullWidget Class.
List of Subclasses	N/A.
Purpose	To navigate through the DeatailsScreen page and showing the Scanned results.
Collaborations	N/A.
Attributes	text/ color.
Operations	N/A.
Constraints	N/A.

Table 25: DetailsButton Class

Abstract or Concrete:	Concrete Class
List of Superclasses	StatefullWidget Class.
List of Subclasses	N/A.
Purpose	To Execute the application.
Collaborations	N/A.
Attributes	text/ color.
Operations	N/A.
Constraints	N/A.

Table 26: NextButton Class

Abstract or Concrete:	Concrete Class
List of Superclasses	StatefullWidget Class.
List of Subclasses	N/A.
Purpose	To Return to the previous page.
Collaborations	N/A.
Attributes	text/ color.
Operations	N/A.
Constraints	N/A.

Table 27: BackButton Class

Abstract or Concrete:	Concrete Class
List of Superclasses	StatefullWidget Class.
List of Subclasses	N/A.
Purpose	To Scan the applications to identify if it is malware or benign and navigates through Scanning Screen page.
Collaborations	N/A.
Attributes	text/ color.
Operations	N/A.
Constraints	N/A.

Table 28: ScanButton Class

10 Operational Scenarios

Our application starts with the home page. The home page contains a button called start scan to initiate a scan on all applications on Android mobile, then this button will navigate the user to a page called scanning page. The scanning page will start scanning on all applications present on the Android device to detect benign and malware applications. When the application begins scanning, it will count all benign and malware applications. The number of benign applications will appear in a green box and number of malware applications will appear in a red box. Also, there will be a table that has two columns. The first column will contain the names of malicious applications, and hypertext called "details" which will navigate to third screen that is called details screen. The second column contains names of all benign applications and the percentage of them being malicious. If the user does not want to know the type of malware, the user will have the ability to go back through a button called done in the bottom right of the screen. The user can navigate to the details screen by pressing on the details word that is in column of malware applications or by pushing the arrow that is inside the red box that counts number of malware applications. The details screen also contains a red box that counts malicious applications, and a table that contains two columns, application name and malware type. In each row of the table, malicious applications' name will be shown, and in the other column, the malware type of this application. When the user knows the malware type also they can know its harmful effect percentage by press on the details button which will show the user the statistics of the application whether it is malicious or benign, they can also go back to scanning page by pushing the back button in the bottom right of the screen.

11 Project Plan

Id	Task	Start Date	Number of Days	Team Member
1	Collect data-set	20/10/2020	10	Omar
2	Complete proposal document	27/10/2020	10	Seif, Mostafa
3	Proof of concept (10%)	30/10/2020	15	Omar, Amr
4	Submit survey paper	01/11/2020	07	Seif
5	Complete SRS document	01/12/2020	25	Seif, Amr, Mostafa, Omar
6	Complete SDD document	01/02/2021	25	Seif, Amr, Mostafa, Omar
7	System prototype	11/02/2021	30	Seif
8	Submit contribution paper	14/02/2021	20	Seif, Amr, Mostafa, Omar
9	Technical evaluation	15/03/2021	45	Seif, Amr, Mostafa, Omar
10	Final thesis	01/05/2021	50	Seif, Amr, Mostafa, Omar
11	Ceremony	17/06/2021	07	Seif, Amr, Mostafa, Omar

Table 29: Project time plan

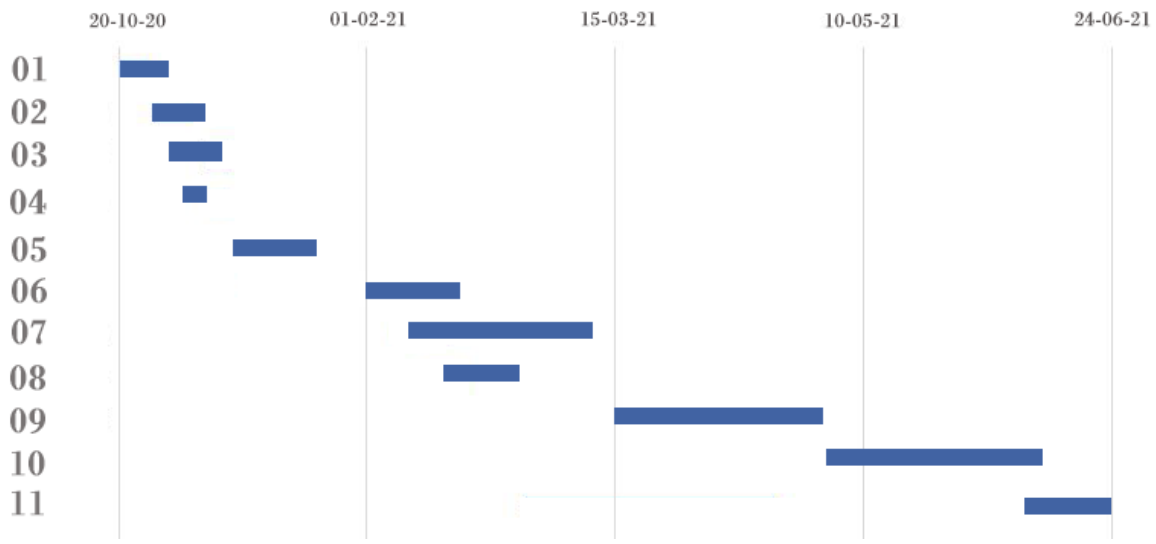

Figure 16: GANT Chart

12 Appendices

12.1 Definitions, Acronyms, Abbreviations

no.	Acronym	Definition
1	SRS	Software Requirement Specification
2	SHA	Secure Hash Algorithm
3	SVM	Support Vector Machine
4	GUI	Graphical User Interface
5	API	Application Programming Interface
6	FR	Functional Requirements
7	MVC	Model - View - Controller
8	DL	Deep Learning
9	ML	Machine Learning
10	KNN	K-Nearest Neighbors
11	SVM	Support Vector Machine
12	DT	Decision Tree
13	RF	Random Forest
14	NN	Neural Networks
15	APK	Android Application Package
16	ISO	International Organization for Standardization

12.2 Supportive Documents

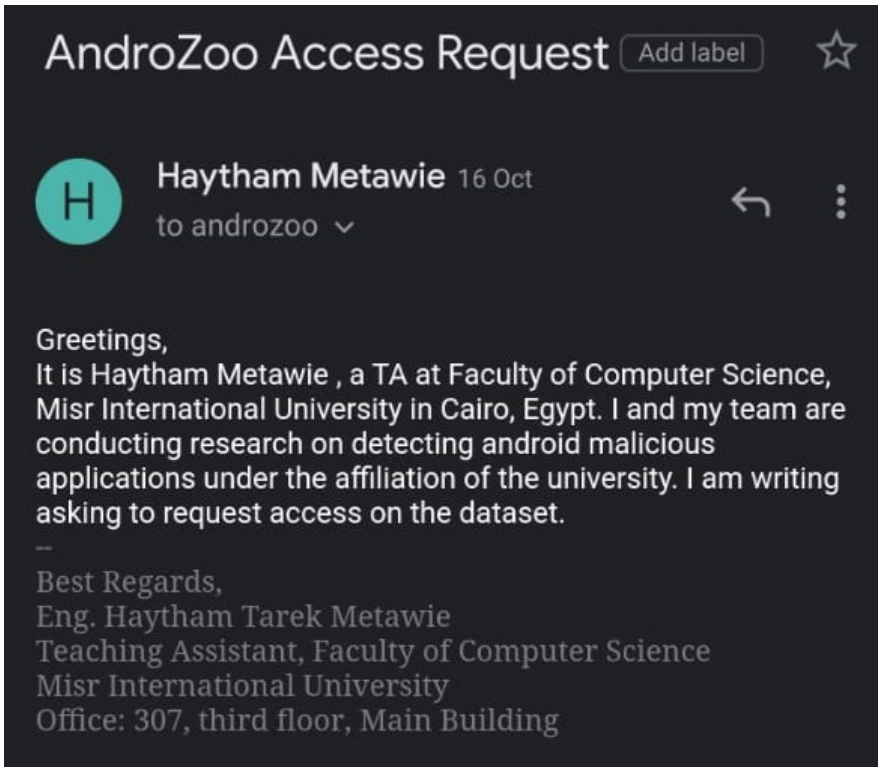

Figure 17: data-set Request

References

[1] Dimitropoulou. https://ceoworld.biz/2019/01/18/worlds-most-popular-mobile-operating-systems-android-vs-ios-market-share-2012-2018/, 2019.

[2] Mobile App Statistics To Know In 2020. https://mindsea.com/app-stats/, 2020.

[3] Oscar Somarriba, Urko Zurutuza, Roberto Uribeetxeberria, Laurent Delosières, and Simin Nadjm-Tehrani. Detection and visualization of android malware behavior. *Journal of Electrical and Computer Engineering*, 2016, 2016.

[4] Xiang Li, Jianyi Liu, Yanyu Huo, Ru Zhang, and Yuangang Yao. An android malware detection method based on androidmanifest file. In *2016 4th International Conference on Cloud Computing and Intelligence Systems (CCIS)*, pages 239–243. IEEE, 2016.

[5] Yuan Zhang, Min Yang, Bingquan Xu, Zhemin Yang, Guofei Gu, Peng Ning, X Sean Wang, and Binyu Zang. Vetting undesirable behaviors in android apps with permission use analysis. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 611–622, 2013.

[6] API Documentation. https://androzoo.uni.lu/api$_doc$, 2016.

[7] Barath Narayanan Narayanan, Ouboti Djaneye-Boundjou, and Temesguen M Kebede. Performance analysis of machine learning and pattern recognition algorithms for malware classification. In *2016 IEEE National Aerospace and Electronics Conference (NAECON) and Ohio Innovation Summit (OIS)*, pages 338–342. IEEE, 2016.

[8] Siemens cybersecurity. https://new.siemens.com/global/en/company/topic-areas/cybersecurity.html, 2020.

[9] Md Faiz Iqbal Faiz and Md Anwar Hussain. Hybrid classification model to detect android application-collusion. In *2020 43rd International Conference on Telecommunications and Signal Processing (TSP)*, pages 492–495. IEEE, 2020.

[10] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, and CERT Siemens. Drebin: Effective and explainable detection of android malware in your pocket. In *Ndss*, volume 14, pages 23–26, 2014.

[11] Static Analysis of Android Malware. https://www.kaggle.com/goorax/static-analysis-of-android-malware-of-2017.

[12] Parnika Bhat, Kamlesh Dutta, and Sukhbir Singh. Mapldroid: Malicious android application detection based on naive bayes using multiple. In *2019 2nd International Conference on Intelligent Communication and Computational Techniques (ICCT)*, pages 49–54. IEEE, 2019.

[13] Android PRAGuard Dataset. http://pralab.diee.unica.it/en/androidpraguarddataset, 2018.

[14] Umme Sumaya Jannat, Syed Md Hasnayeen, Mirza Kamrul Bashar Shuhan, and Md Sadek Ferdous. Analysis and detection of malware in android applications using machine learning. In *2019 International Conference on Electrical, Computer and Communication Engineering (ECCE)*, pages 1–7. IEEE, 2019.

[15] Android Malware Genome Project. http://www.malgenomeproject.org/.

[16] Zhuo Ma, Haoran Ge, Yang Liu, Meng Zhao, and Jianfeng Ma. A combination method for android malware detection based on control flow graphs and machine learning algorithms. *IEEE access*, 7:21235–21245, 2019.

[17] Muhammad Murtaz, Hassan Azwar, Syed Baqir Ali, and Saad Rehman. A framework for android malware detection and classification. In *2018 IEEE 5th International Conference on Engineering Technologies and Applied Sciences (ICETAS)*, pages 1–5. IEEE, 2018.

[18] Websites of malware bytes. Anti-malware malwarebytes.

[19] Malwarebytes android mobile application. Play store - malwarebytes security: Virus cleaner, anti-malware fro android mobile.

[20] Website of US Norton.

[21] NortonMobile. Play store - norton 360 for android mobile.

[22] 34 Open-Source Python Libraries. https://www.mygreatlearning.com/blog/open-source-python-libraries/l, 2020.

[23] apk parser. https://pypi.org/project/apk-parse/, 2018.

[24] Franca Garzotto. A user-friendly enterprise framework for data intensive web applications. In *IRI-2005 IEEE International Conference on Information Reuse and Integration, Conf, 2005.*, pages 415–420. IEEE, 2005.

[25] Karima Moumane, Ali Idri, and Alain Abran. Usability evaluation of mobile applications using iso 9241 and iso 25062 standards. *SpringerPlus*, 5(1):548, 2016.

[26] Minimum System Requirements for Android. https://www.whistleout.com/cellphones/guides/guide-to-buying-cheap-android-phones-specs-and-tips, 2017.

[27] Controller MVC: Model, View. https://www.codecademy.com/articles/mvc, 2020.

[28] Kevin Allix, Tegawendé F Bissyandé, Jacques Klein, and Yves Le Traon. Androzoo: Collecting millions of android apps for the research community. In *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*, pages 468–471. IEEE, 2016.