[2021/2022]

Computer Engineering Department

Faculty Of Engineering

Helwan University

Android Malware Detection System

Peresented By:

Kareem Atif Mohamed Mohamed

Salma Seif Eldin Elsayed Salem

Ahmed Mohamed Ahmed Mazroua

Mayar AlaaEldin AbdElnaby Ahmed

Abdullah Ibrahim Nady Soliman

Supervised by:

Dr. Samir Gaber

Dr. Mahmoud Zaki

Abstract

Android is a free open-source operating system (OS), which allows an in-depth understanding of its architecture. Therefore, many manufacturers are utilizing this OS to produce mobile devices (smartphones, smartwatch, and smart glasses) in different brands, including Google Pixel, Motorola, Samsung, and Sony. Notably, the employment of OS leads to a rapid increase in the number of Android users. Android OS is one of the widely used mobile Operating Systems.

Android applications are developing rapidly across the mobile ecosystem, but Android malware is also emerging in an endless stream. Many researchers have studied the problem of Android malware detection and have put forward theories and methods from different perspectives.

Existing research suggests that machine learning is an effective and promising way to detect Android malware. Notwithstanding, there exist reviews that have surveyed different issues related to Android malware detection based on machine learning. We believe our work complements the previous reviews by surveying a wider range of aspects of the topic. This paper presents a comprehensive survey of Android malware detection approaches based on machine learning.[1]

We briefly introduce some background on Android applications, including the Android system architecture, security mechanisms, and classification of Android malware. Then, taking machine learning as the focus, we analyze and summarize the research status from key perspectives such as sample acquisition, data preprocessing, feature selection, machine learning models, algorithms, and the evaluation of detection effectiveness. Finally, we assess the future prospects for research into Android malware detection based on machine learning.

This review will help academics gain a full picture of Android malware detection based on machine learning. It could then serve as a basis for subsequent researchers to start new work and help to guide research in the field more generally.[2]

Acknowledgment

First and foremost, praises and thanks to Allah, the almighty, for his showers of blessings throughout our whole journey of finding a real beneficial idea, and working as a great team on the project.

We would like to express our deep and sincere gratitude to our project supervisor, **Dr.Mahmoud Zaki** for providing invaluable guidance throughout this project. He has always supported, motivated, and lift our spirits to give our maximum effort for this project and for its noble purpose. It was a great privilege and honor to work and study under his guidance. We are extremely grateful for what he has offered us. We truly hope that our outcome can make him proud of us.

Besides our supervisor. We would like to thank **Dr.Samir Gaber** for providing invaluable guidance throughout this project. He supported us, motivated us, and inspired us with so many ideas. He helped us understand so many aspects of the project. A lot couldn't be possible without his guidance, he truly made a lot of things easier for us.

Last but not least, we members of this team would like to thank each other for always supporting and motivating each other. Always being there for each other. Understanding and considerate with each other. Appreciating each and every effort by each member. And always learning from each other.

Contents

Abstract 2
Acknowledgment 3
Table of Contents 4
Table of Figures 7
Table of Tables 9

Chapter (1):
Introduction 10
1.1 Introduction 11
1.2 Literature Review 13
1.2.1 the prior work 13
1.2.2 our improvement 14
1.3 Motivation 15
1.4 Problem Statement 16
1.5 Project Diagram 18
1.5.1 Unpackaging & Decompiling 18
1.5.2 Features Extraction 18
1.5.3 Features Vector 18
1.5.4 Classifier 18
1.6 Project Phases 19
Chapter (2):
Reverse Engineering 20
2.1 Disassembling and Decompiling 21
2.2 Android Application Contents 22
2.3 Practical Overview 22

Chapter (3):
Malware Intrusion Detection Systems 27
3.1 Android Malware Features 29
3.1.1 Android - Permissions 31
3.1.2 Android - Broadcast Receivers 33
3.1.3 Android - Intent Services 36
3.1.4 API Calls 37
3.2 Feature Extraction 38

Chapter (4):
Machine Learning Techniques 39
4.1 Dataset Description 41
4.2 ML in Permission Feature 42
4.2.1 Support Vector Machine (SVM) result 44
4.2.2 Decision Tree 45
4.2.3 Random Forest 46
4.3 ML in Receiver Feature 47
4.3.1 Support Vector Machine (SVM) result 47
4.3.2 Decision Tree 48
4.3.3 Random Forest 49
4.4 ML in Services Feature 50
4.4.1 Support Vector Machine (SVM) result 50
4.4.2 Decision Tree 51
4.4.3 Random Forest 52

Chapter (5):
Deep Learning Techniques 53
5.1 ANN 54
5.1.1 ANN technique on permission feature 55
5.1.2 Summarize history for Loss & Accuracy 56
5.2 LSTM 57
5.2.1 LSTM technique on permission feature 58
5.2.2 LSTM technique on Receiver feature 59
5.2.3 LSTM technique on Services feature 60
Difference between Machine and Deep Learning 61

Chapter (6):
Front End 63

Chapter (7):
Back End 69
7.1 The Website Page 71
7.2 Making Apk.txt 72
7.3 Backend Testing Code 73
7.4 Implementation of the server 78
7.4.1 Permissions 78
7.4.2 Services 78

7.4.3 Receivers 78
7.4.4 API Calls 82

Chapter (8):
Evaluation 84

Chapter (9):
Design specifications 86
9.1 System requirements 87
9.1.1 Functional requirements 87
9.1.2 Non-functional requirements 88
9.1.3 Functional requirements specification 89
9.1.3.1 Stakeholders 89
9.1.3.2 Use cases Diagram: 90
9.1.3.3 Sequence Diagrams 91
9.1.3.4 Class Diagram 92
9.2 Tools and technologies: 93
9.2.1 Tools 93
9.2.2 Technologies 93

Chapter (10):
10.1 Conclusion 95
10.2 Future work 96
10.3 References 97

Table of Figures

Fig (1): Smartphone subscriptions worldwide 2016-2021 11
Fig (2): Malware installation Packages for smartphone devices 12
Fig (3): The no. of detected malicious installation packages for recent years ... 16
Fig (4): The user has a choice to install 17
Fig (5): System overview and components 18
Fig (6): Project Phases 19
Fig (7): APK Files 22
Fig (8): Using APKTool to decode the files of an APK 23
Fig (9): AndroidManifest.xml opened from Apk file 23
Fig (10): Permissions required an App to do its function 24
Fig (11): Services using in an android App 24
Fig (12): Receivers using in an android App 25
Fig (13): A Class. Dex file 26
Fig (14): Android Malware Features 30
Fig (15): Broadcast-Receiver 33
Fig (16): Code:1 Broadcast-Receiver 33
Fig (17): Code:2 Broadcast-Receiver 35
Fig (18): Code Intent-Services 36
Fig (19): Android Malware Detection 40
Fig (20): Malicious Apps 41
Fig (21): Benign Apps 42
Fig (22): SMS permission 42
Fig (23): SVM of permission 44
Fig (24): Decision of permission 45
Fig (25): Random Forest of permission 46
Fig (26): SVM of Receiver 47
Fig (27): Decision Tree of Receiver 48
Fig (28): Random Forest of Receiver 49
Fig (29): SVM of Services 50
Fig (30): Decision Tree of Services 51
Fig (31): Random Forest of Services 52

Fig (32): ANN of permission 55
Fig (33): Model of Loss & accuracy 56
Fig (34): History of Loss & accuracy 57
Fig (35): LSTM of permission 58
Fig (36): LSTM of Receiver 59
Fig (37): LSTM of Services 60
Fig (38): There are many differences between these two subsets of AI 61
Fig (39): Apk path of Apps 64
Fig (40): Pubspec.yaml using Chaquopy 66
Fig (41): top-level build.gradle file 66
Fig (42): Python script to extract features from Apk 67
Fig (43): Neuron User Interface 68
Fig (44): Network Overview 70
Fig (45): Choose apk.txt 71
Fig (46): Click submit 71
Fig (47): Result of prediction 71
Fig (48): Jupiter 72
Fig (49): extracted permission feature code 72
Fig (50): Backend testing code 73
Fig (51): model_predict function 74
Fig (52): permissions of app as list of 0s and 1s 74
Fig (53): permissionListForPrediction.txt 75
Fig (54): Post request body 76
Fig (55): GET or POST 77
Fig (56): Model prediction of permission 78
Fig (57): Model prediction of Services 78
Fig (58): Model prediction of Receiver 78
Fig (59): Send result of client 80
Fig (60): The response from the server to the client 81
Fig (61): Http API 82
Fig (62): App scan API 82
Fig (63): Extracted features in JSON 83
Fig (64): Use Case diagram 90
Fig (65): Sequence diagram 91
Fig (66): Class diagram 92

Table of Tables

Table (1): Important system events 34
Table (2): Dataset between Benign and Malware 41
Table (3): Evaluation 1 85
Table (4): Evaluation 2 85
Table (5): Functional Requirements 87
Table (6): Non-Functional Requirements 88

Chapter (1): Introduction

Android OS was designed in 2007 as a modified version of the Linux kernel for mobile devices. The Android OS obtained 70% of the market share worldwide compared to other OS in 2021. Over the last years, smartphones have become one of the most used devices worldwide with nearly 6.259 billion users in 2021 according to Statista report [3] and shown Fig (1).

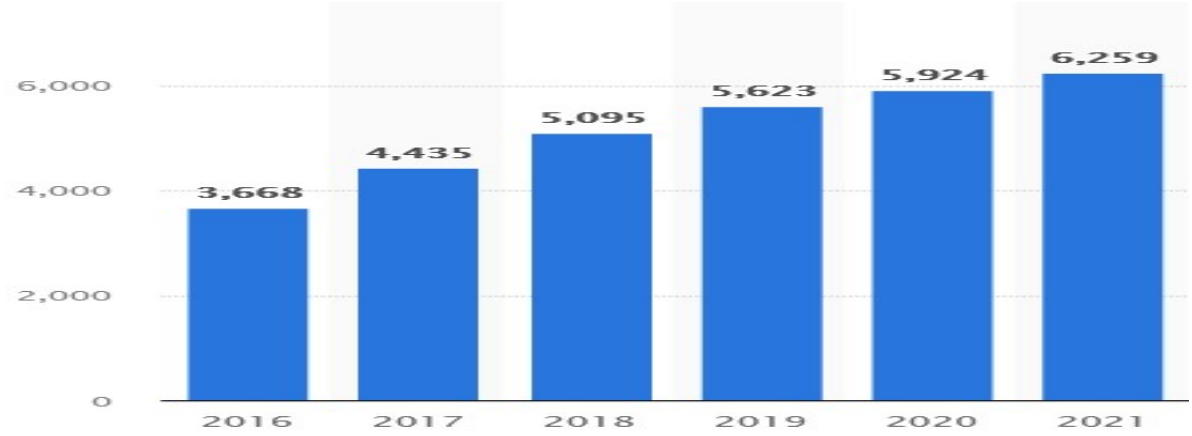


Fig (1): Smartphone subscriptions worldwide 2016-2021.

In recent years, the popularity of the Android operation system has attracted the attention of malware developers, whose work has grown rapidly [4,5]. Many malware developers focus on hacking mobile devices and changing them into bots. This allows hackers to access the infected device and other connected devices and form botnets. Botnets are used to execute different malicious attacks, such as distributed denial-of-service (DDoS) attacks, sending spam, data theft, etc. The malicious botnet attacks are developed with advanced techniques (e.g., multi-staged payload or self-protection), making it difficult to identify the malware. This, in turn, poses major threats that require the design of effective approaches to detect these attacks [6].

Haystack [7] reported that a third part of software-development companies manage 70% of the mobile application and control the personal data of users. According to the AVTEST Security Institute [8], malicious programming increased, with 5.7 million malware Android packages detected by Kaspersky in 2020, three times more than in 2019 (2.1 million).

Fig (2) summarizes the increase of malware installation packages for smartphone devices in the last five years.

Therefore, signature-based malicious installation packages for the extraction of malware patterns relying on their characteristics can be an effective strategy to secure mobile applications.

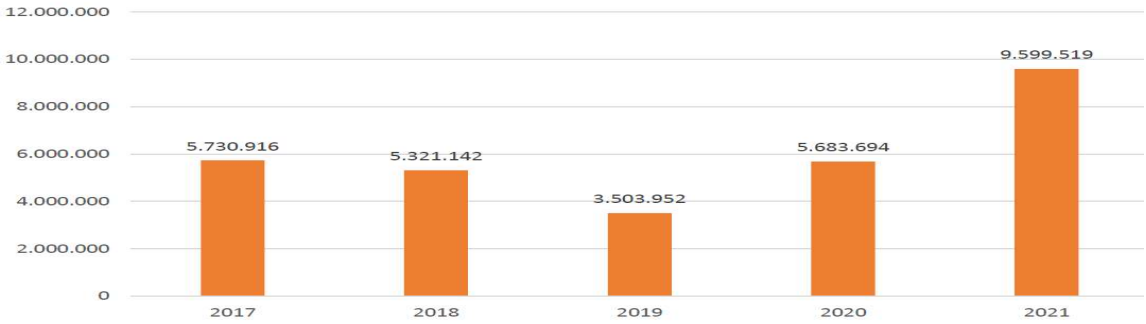


Fig (2): Malware installation Packages for smartphone devices.

The present project aimed to extract static features from unknown applications; these features show if a particular application is "Benign", "Low-Risk", "Medium-Risk" or "High-Risk" Android Application. These features are used to examine the performance of several machine learning and deep learning models, including the k-nearest neighbors (KNN), support vector machine (SVM), random forest (RF), decision tree, adaboost [9], multi-layer perceptron, long short-term memory (LSTM) [10] methods.

In this project, we investigated and estimated the performance of various machine learning and deep learning algorithms in the detection of mobile malware applications. This project offers the optimal algorithms for the monitoring of Android applications against malicious static features.

Thus, our research aims to contribute to this field with the following:

1. The development of intrusion detection in the Android system using various machine learning and deep learning algorithms.
2. The proposed system was tested and evaluated using two standard Android datasets.
3. A comparison between the tested algorithms and different state-of-the-arts models is presented.
4. The sensitivity analysis was used to find significant relationships between dataset features and the proposed classes of the datasets.

Literature Review

The initial studies on smartphone malware were chiefly targeted on understanding the threats and behaviors of rising malware. There has been vital work on the matter of detecting and eliminating malware on mobile devices. Signatures primarily based ways, introduced within the mid-90s area unit ordinarily employed in malware detection. The main weakness of this kind of approach is its weakness in recognizing updated and unseen malware. Rather than victimizing predefined signatures for malware detection, data processing and machine learning techniques give a good thanks to dynamically extract malware patterns [11].

- **The prior work:**

1. They collect several malicious and benign android applications by retrieving their APKs online.

2. they implement feature extraction by enumerating the API calls, permissions, and activity of the APKs to get a fair idea of the behavior of the apps. Extract the following things in the analysis phase:

    a. Permissions

    b. Intent-receivers

    c. Intent-services

    d. API Calls

3. Then create a dataset based on the extracted features by compiling them into CSV files for future use and reference [11].

4. After that selecting specific features which seem like an anomaly or a red flag, collect and store them in new CSV files and train our Machine Learning model according to them. SVM is known for classification, so it was the first algorithm to go towards but during the testing it found out that the novel SVM approach had low accuracy on real time applications.

    They mainly compared these two algorithms GRU (Gated recurrent units) and LSTM (Long Short-Term Memory Network) where we found LSTM to have

an edge with respect to accuracy and hence, we preferred it as our prime algorithm for our model so, the model used is Long Short-Term Memory Network [11].

5. Finally, the last phase is testing the model with the data and evaluate our findings according to the results.

6. At the application the user is allowed to choose deep scan or not, in deep scan we analyze API Calls hence it takes more time.

Once the user sends the desired application to test it, a backup is taken, and it is uploaded to the eternal storage. The next phase involves the server receiving a request as a trigger message along with a token number which identifies the device from which the App was sent while the App is stored on the storage along with the user's scan choice as its name. Now the server conducts the analysis via our trained model where the applied LSTM algorithm plays the main role. Here standard analysis is done based on 3 factors, namely Permissions, Receivers and Services and deep analysis is done based on API Calls.

- **Our improvements:**

1. **Reduce time of deep scan**: Since the Deep scan analysis takes a while, we can optimize it by reframing the algorithm and increase its efficiency and reduce time.

2. **Increase scanning ability and test apps on real time basics:** using an Android Sandbox environment. This would increase the scanning ability of our detection system as well as it would test the apps on a real time basis. In real time, some apps behave differently than they are supposed to; hence a Sandbox would catch such apps and provide an added layer of accuracy and protection.

3. **enable users to simultaneously send multiple or all apps for testing at one time**: by using multithreading.

4. **integrating machine learning techniques:** for better enhanced performance.

Motivation

Our motivation for project is to protect smartphones from attacks by building a system which scan the android applications for detecting the malicious programs and able to scan new malwares from different families that don't has a signature before by using static analysis features of android applications such as permission, Services, Receivers, API calls, etc.

We will use machine learning techniques such as support vector machine, Random Forest, Decision Tree, etc. The expected outcome is to build a system can used in smartphones and online Web Service to analyze android applications and tell the user if this android application is a Benign, Low-Risk, Medium-Risk, or High-Risk android application.

Also, making a database to improve our system speed as a cache of the previous analyzed android applications and in the future with new types of malwares moreover the using of this combination between deep learning techniques and machine learning techniques on the static features of android application make the system scan slowly so to raise the speed of scan, we will use sandbox environment or modify the algorithm which we build.

Problem statement

Increasing the number of attacks that targets mobile users especially android mobile users as the android operating system is the most used operating system for mobile devices around the world nowadays [12].

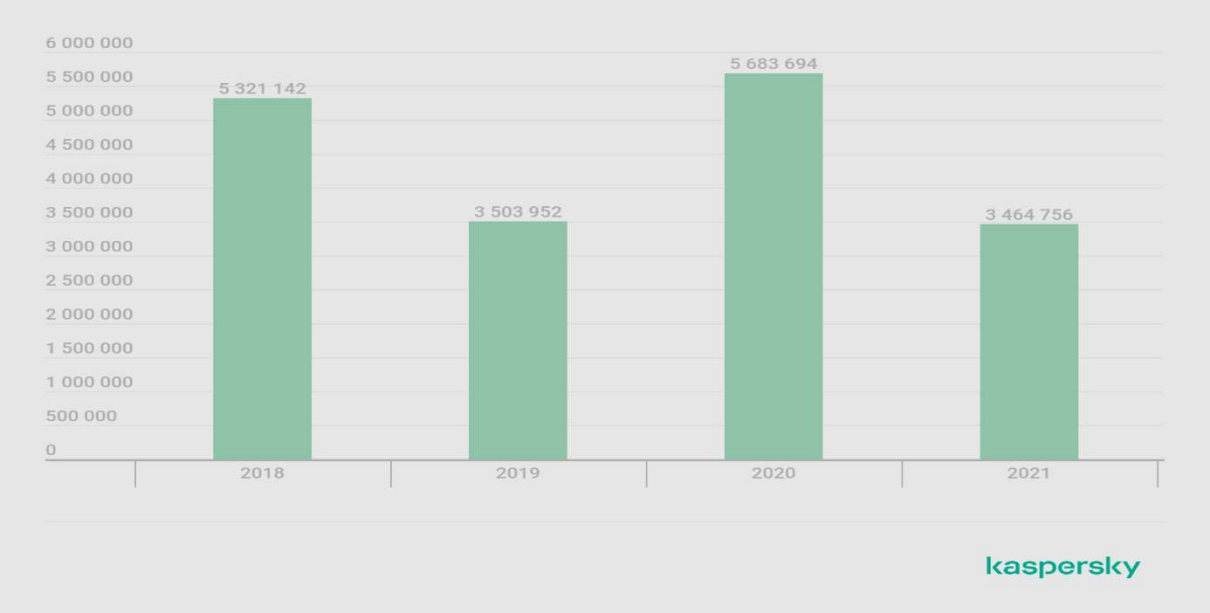


Fig (3): The number of detected malicious installation packages for recent years - Source: Kaspersky [10].

Fig (3) [12], represents that Nearly 3,5 million malicious installation packages have been detected by its products in 2021. As the number of malicious packages over the years not decreased continually, and may be the next year, the number of malicious installation packages will increase than before. Therefore, we hope that with each year the rate of attacks targeting mobile users decreases and help to identify malicious applications to protect the users from them.

Existing approaches evolved to address issues of the permission system are fundamentally limited by the categorisation of permissions done by Android. Permissions are grouped into four categories according to Google: normal, dangerous, signature, and SignatureOrSystem. The first category cannot impart real harm to the user; it is automatically granted without the user while the second one can have a negative impact if used incorrectly.

In the second one (dangerous) the user probably grants the requested permissions. Most users do not understand what each permission means and blindly grant them, allowing the application to access the user's sensitive information, according to [13] and [14]. This conceptual categorisation fixed by Google however becomes false when the association of many permissions is considered. For instance, the permission READ_PHONE_STATE (which is normal according to Google) allows applications to get some information (identifiers, phone numbers), but the permission itself taken individually is not risky concerning personal data.

When, however, the application requests also several other permissions such as INTERNET (which allows accessing the Internet) personal data can be transmitted to a remote server for bad usage.

This mechanism gets the user confused and it probably will give more authorizations than it should be allowed Fig (4).

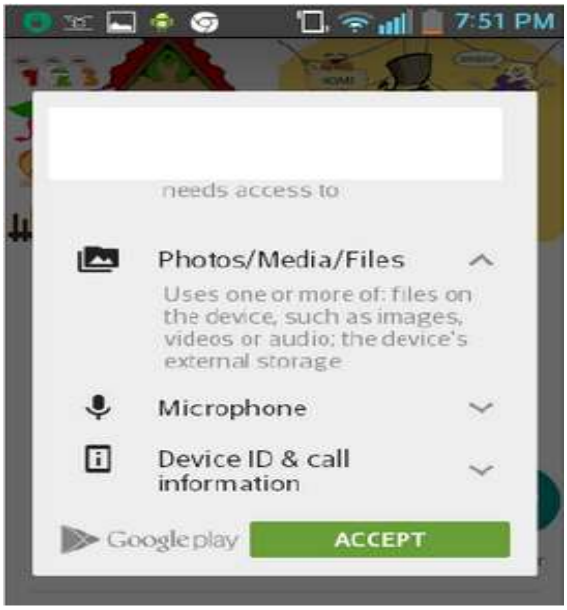


Fig (4): The user has a choice to install.

The research problem provides detection models of Android malware, which consider combinations of permissions, services, intent receivers, API Calls, and security risks generated by these resources, while considering any of them as risky.

Project Diagram

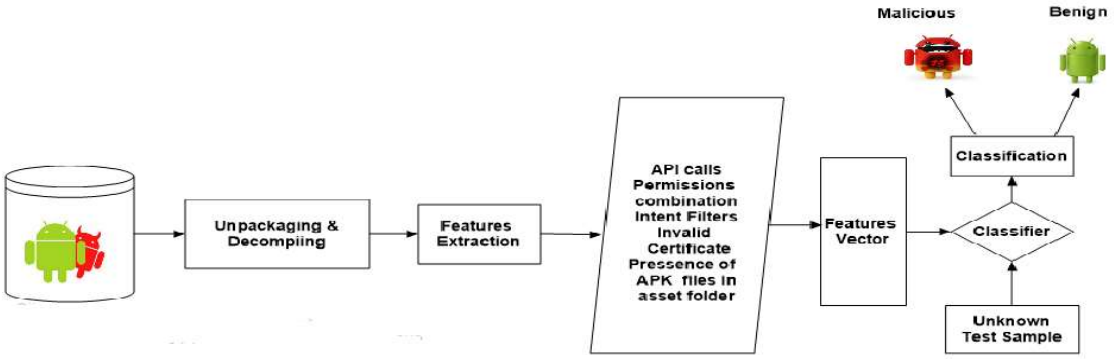


Fig (5): System overview and components.

- **Unpackaging & Decompiling:**
  Decompiling is a process in which a mobile app is analysed and dissected to its components like its resources, permissions, etc. A decompile program examines the binary and translates its contents from a low-level abstraction to a higher-level abstraction.

- **Features Extraction:**
  After decompiling the application and dissecting to its components. Now, Features is extracted from the application data and components like the app permissions, API calls, Intent Filters, etc.

- **Features Vector:**
  Generating a vector from the Features which extracted from the application which analysis it

- **Classifier:**
  A classifier (algorism) that automatically orders or categorizes the given application into two classes (Malware or benign).

So, Malware classifier System will scan apps by different machine learning and deep learning techniques to filter them by class label: Malware or Benign and as we will use a combination of more static features of android application which pass to the model to classify it. A combination between permissions, services, intent receivers, and API Calls. So, the system will classify the given app to four Classes: Benign (Safe), Low-Risk, Medium-Risk, or High-Risk android application.

Project phases

The User selects the APK file which is to be tested and it is sent to the cloud storage. Over there the applications are stored in sequence and wait to be executed. The APK is then sent to the Cloud Engine. The features of the APK such as permissions and API calls are extracted and sent to the Machine Learning model. Thus, the features of the APK are analyzed and based on the findings a report is generated and sent back to the User.

Fig (6): Project Phases.

Firstly, we collect several malicious and benign android applications by retrieving their APKs online. Next, we implement feature extraction by enumerating the API calls, permissions, and activity of the APKs to get a fair idea of the behavior of the applications. Then we create a dataset based on the extracted features by compiling them into CSV files for future use and reference.

After that we select specific features which seem like an anomaly or a red flag, collect and store them in new CSV files and train our Machine Learning model according to them.

Finally, our last phase is to test our model with the data and evaluate our findings according to the results.

Chapter (2): Reverse Engineering

Reverse Engineering: it is the process of taking an app apart to find out how it works. You can do this by examining the compiled app (static analysis), observing the app during runtime (dynamic analysis), or a combination of both [13].

1. Disassembling and Decompiling

In Android app security testing, if the application is based solely on Java and doesn't have any native code (C/C++ code), the reverse engineering process is relatively easy and recovers (decompiles) almost all the source code. In those cases, black-box testing (with access to the compiled binary, but not the original source code) can get pretty close to white-box testing [14].

Android decompilers go one step further and attempt to convert Android bytecode back into Java source code, making it more human-readable. Fortunately, Java decompilers generally handle Android bytecode well. The above-mentioned tools embed, and sometimes even combine, popular free decompilers such as:

- JD
- JAD
- Jadx

2. Android Application Contents

Android application is a compressed file that can be unzipped to reveal its files.

Below are the files you should expect when you decompress an APK file:

- **AndroidManifest.xml**: here are everything about the application like:
  - Package name
  - Target and minimum API level
  - App configuration
  - App components
  - Permissions

- **META-INF**: contain metadata about the application:
  - classes.dex: this file contains the compiled source code.
  - assets: a folder containing the following.
  - Pictures
  - Fonts
  - Videos files

- **Lib**: include the native library that the application uses.

- **Res**: this is the resources folder, and you can find in it stuff like:
  - Shapes
  - Colors
  - Pictures

3. Practical Overview

As an example of apk file analysis, An APK is an archive file, meaning that it contains multiple files, plus some metadata about them. You're probably familiar with other types of archive files, like ZIP and RAR [14].

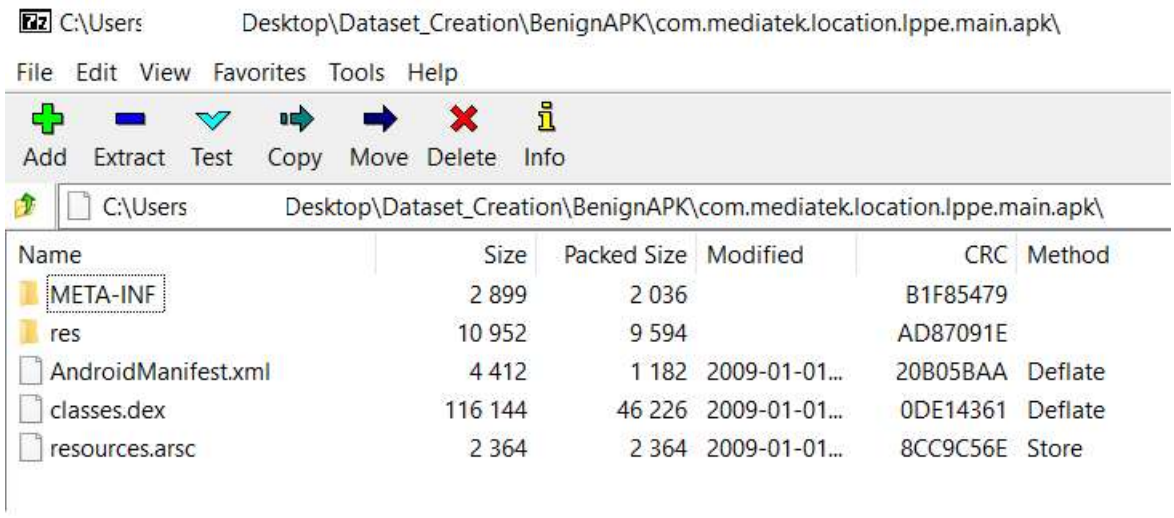


Fig (7): APK file opened as an archive and shown its contents from files & folders.

Now, if we opened the AndroidManifest.xml file from the above archive, it will not be understandable, because it encoded as shown in Fig (8).

So, we will use tool like **APKTOOL** **to make decoding to the files to show the contents in a readable and understandable format.**

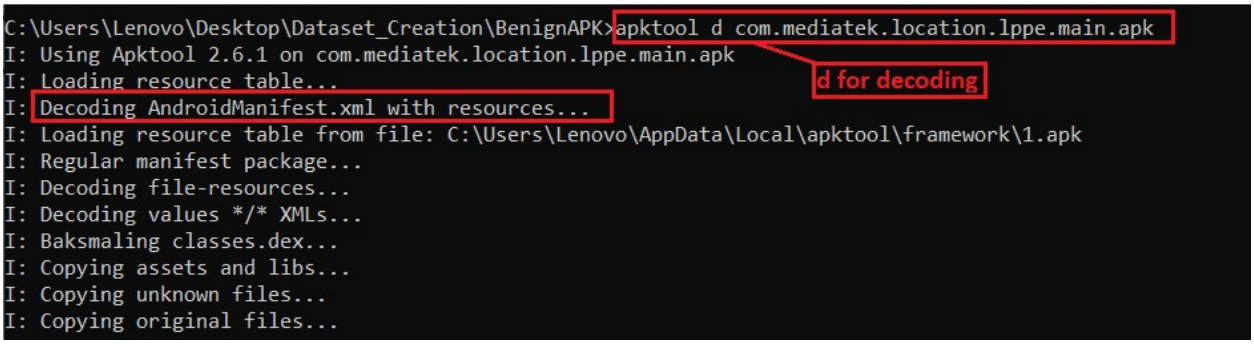


Fig (8): using APKTOOL to decode the files of an apk.

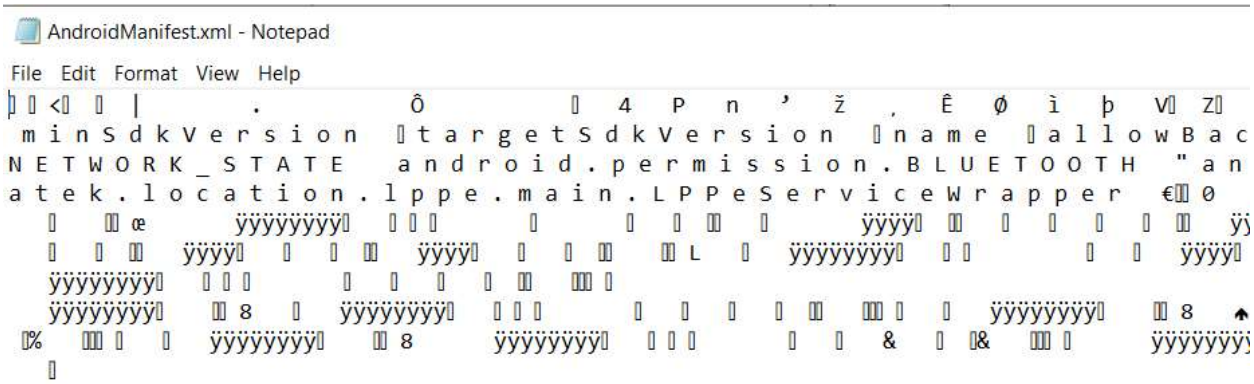


Fig (9): AndroidManifest.xml opened from apk file archive in encoded format.

This manifest file is normally a binary file within the APK or app bundle, but when selected in the APK Analyzer, the XML form of this entity is reconstructed and presented. This viewer allows you to understand any changes that might have been made to your app during the build. For example, you can see how the AndroidManifest.xml file from a library your application depends on was merged into the final AndroidManifest.xml file [16].

And as shown in Fig (10) the permissions required by an android application to do its functions.

```
<?xml version="1.0" encoding="utf-8" standalone="no"?><manifest xmlns:android="http
    <uses-permission android:name="android.permission.LOCATION_HARDWARE"/>
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
    <uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>
    <uses-permission android:name="android.permission.CHANGE_NETWORK_STATE"/>
    <uses-permission android:name="android.permission.BLUETOOTH"/>
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
    <uses-permission android:name="android.permission.READ_SMS"/>
    <uses-permission android:name="android.permission.RECEIVE_SMS"/>
    <uses-permission android:name="android.permission.RECEIVE_WAP_PUSH"/>
    <uses-permission android:name="android.permission.WAKE_LOCK"/>
    <uses-permission android:name="com.mediatek.permission.CTA_ENABLE_WIFI"/>
    <uses-permission android:name="android.permission.PROCESS_OUTGOING_CALLS"/>
```

Set of permissions requred by the application to make its function.

Fig (10): Permissions required by an application to do its function.

A Service is an application component that can perform long-running operations in the background. It does not provide a user interface. Once started, a service might continue running for some time, even after the user switches to another application. Additionally, a component can bind to a service to interact with it and even perform interprocess communication (IPC) [15]. For example, a service can handle network transactions, play music, perform file I/O, or interact with a content provider, all from the background. And services of an android application is shown in Fig (11).

```
<service android:name="com.etisalat.utils.services.RichNotificationService">
    <intent-filter>
        <action android:name="com.google.firebase.MESSAGING_EVENT"/>
    </intent-filter>
</service>
<service android:name="com.etisalat.view.authorization.pushnotification.GCMJobService" android:permission="android.permission.BIND_JOB_SERVICE"/>
<service android:name="com.etisalat.view.authorization.pushnotification.GCMIntentService"/>
<uses-library android:name="android.test.runner"/>
```

Fig (11): Services used in an android application.

A broadcast receiver (a BroadcastReceiver subclass) as one of the application's components. Broadcast receivers enable applications to receive intents that are broadcast by the system or by other applications, even when other components of the application are not running.

There are two ways to make a broadcast receiver known to the system: One is declared it in the manifest file with this element. The other is to create the receiver dynamically in code and register it with the Context.registerReceiver() method. And a Receivers used in an android application is shown in Fig (12).

```
<receiver android:enabled="true" android:exported="false" android:name="com.google.android.gms.measurement.AppMeasurementReceiver"/>
<service android:enabled="true" android:exported="false" android:name="com.google.android.gms.measurement.AppMeasurementService"/>
<service android:enabled="true" android:exported="false" android:name="com.google.android.gms.measurement.AppMeasurementJobService" a
<receiver android:exported="true" android:name="com.google.firebase.iid.FirebaseInstanceIdReceiver" android:permission="com.google.ar
    <intent-filter>
        <action android:name="com.google.android.c2dm.intent.RECEIVE"/>
    </intent-filter>
</receiver>
```

Fig (12): Receivers used in an android application.

Application code in the Dex file format. There can be additional .dex files (named classes2.dex, etc.) when the application uses multidex.

This is the actual code of the app. “dex” is the short form of Dalvik Executable. You already know that the android apps are coded in Java (in most cases). The source code will be in the extension “.java”. When it is compiled, it will become “.class”. But in android all these class files are further optimized and packed into dex file for running easily in the android run time. It also provides protection to the code so that no one can steal it easily.

The APK Analyzer provides nice support for decoding Dex files. The most useful feature here is the Referenced Methods column that shows counts for all method references. This information comes in handy when analyzing the number of references in apps in order to stay below the 64K methods limit.

We can navigate through the entire code and search for usage of classes and methods by right-clicking on the class or method names and selecting the Find Usages option.

We can also decompile classes and methods into smali bytecode by selecting the Show Bytecode option in the same context menu as shown in Fig (13).

What is smali bytecode, you ask? Well, it's a disassembler syntax for Dalvik bytecode.

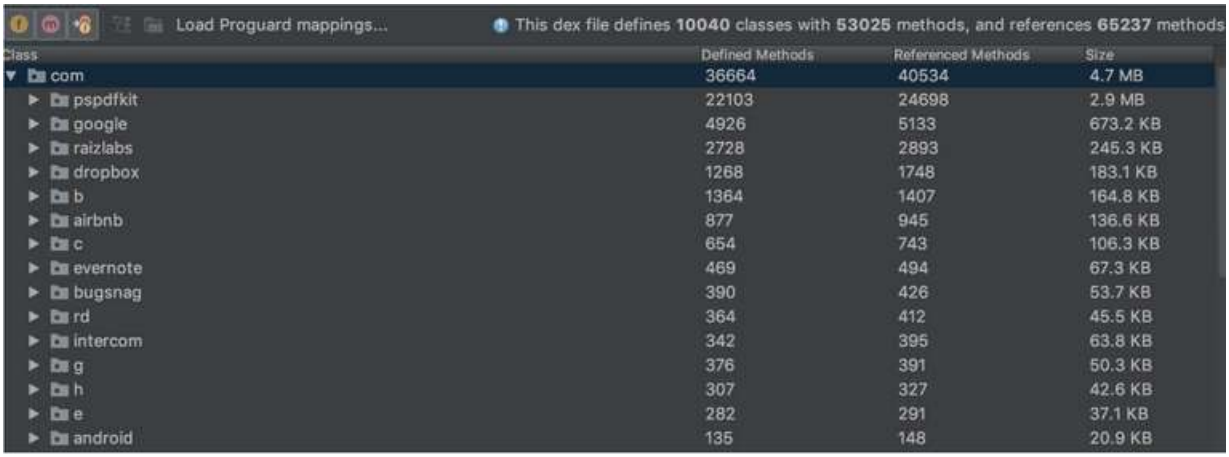


Fig (13): A class.dex file decoded and shown the classes used in the code.

Chapter(3): Malware intrusion detection systems

Although there are security mechanisms, such as firewalls, antivirus software, and IDSs, to secure mobile devices, there is still a need to develop a novel approach towards detecting malware. Commercial antivirus mechanisms can effectively detect known malware, but they are incapable of detecting unknown malware.

Therefore, malware detection techniques with high levels of accuracy and speed are important to ensure the effectiveness of such malware applications. The IDSs were developed to protect devices from attackers. If an intrusion is detected, then it is logged, and an alert is generated. The malware IDSs are divided into three classifications, as illustrated in Analysis technique. The two most common techniques used in this field are static and dynamic analysis techniques. The static analysis technique analyses the programs without executing them.

The static examination is possible by using the program analyzer, debugger, and disassembler. The features commonly used in the static analysis are permission, intent filters, Java code, network address, strings, and hardware components. Android applications use permissions to protect the mobiles from malware. The application requests permission to access the mobile device during the installation process [17].

Meanwhile, Android applications use API calls to communicate with the devices. Dishonest programmers normal change the sequence and then rename the API calls to evade the detection system. This is known as code obfuscation. It is then saved as a Dalvik format. Fraudulent programmers are able to modify the code-base and inject malicious code into Android applications.

All the static features are accessible in the AndroidManifest.xml file or Java code file. Table 1 shows the related work in static analysis. In contrast, the dynamic analysis runs the application in a safe environment while observing the malware behavior. It is capable of detecting malware when the obfuscation technique is applied. The features most commonly selected for detecting malware in the dynamic analysis were memory and registry usage, instruction traces, network traffic, and API call traces.

More than 1,000 datasets were used in these studies. The accuracy rate of the detection was more than 90%. Various tools used in the dynamic analysis include CrowDroid, TaintDroid, ParanoidAndroid, Aurasium, Appfence, and DriodScope [18].

The hybrid analysis combines both static and dynamic analyses. In the hybrid analysis, user behaviors and permission intent were the most common features selected for detecting malware. 4,000 datasets extracted from MARVIN, Drebin, Genome and Virus hare were used in these studies. The accuracy rate of the detection was more than 90%.

Android malware features

Android malware features include static, dynamic, hybrid, and application metadata features **Fig:(14)** shows the taxonomy of Android malware features. The static features are available in the AndroidManifest.xml or Java code file. The common static features used are permission, Java code, intent filters, network address, strings, and hardware components. In comparison, dynamic features refer to the application behaviors that communicate between the operating systems or networks. A previous study used network.

A previous study used network traffic, system calls, and API calls as dynamic features to detect malware. Hybrid features are those derived from a combination of static and dynamic features. They are used in the detection system as an additional measure to increase accuracy. The application metadata features refer to the information that appears before a download installs the application.

They include the application description, rating, requested permission, and developer information. The selection of features is a crucial part of the classification process. Some of the features used in previous studies were permission-based features that used the API call sequence features and intent [19].

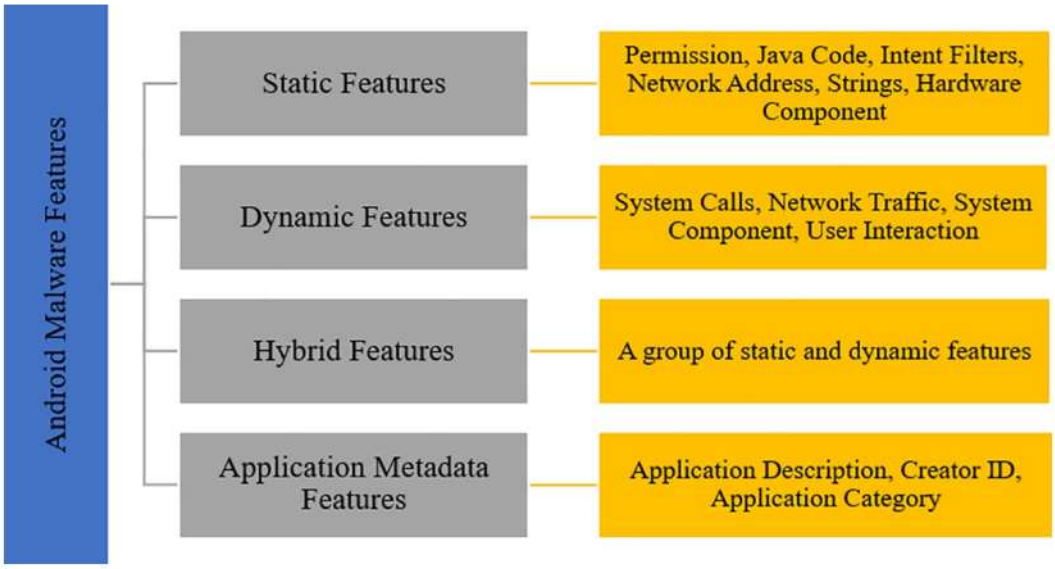


Fig (14): Android malware features

Selected Features for Neuron

Android - Permissions

- syntax:

```
<uses-permission android:name="string"
        android:maxSdkVersion="integer" />
```

- description:

Specifies a system permission that the user must grant in order for the app to operate correctly. Permissions are granted by the user when the application is installed (on devices running Android 5.1 and lower) or while the app is running (on devices running Android 6.0 and higher).

For more information on permissions, see the Permissions section in the introduction and the separate System Permissions API guide. A list of permissions defined by the base platform can be found at android.Manifest.permission.

- attributes:

android:name:

The name of the permission. It can be permission defined by the application with the <permission> element, permission defined by another application, or one of the standard system permissions (suchas "android.permission.CAMERA" or "android.permission.READ_CONTACTS").

As these examples show, a permission name typically includes the package name as a prefix.

android:maxSdkVersion:

The highest API level at which this permission should be granted to your app. Setting this attribute is useful if the permission your app requires is no longer needed beginning at a certain API level.

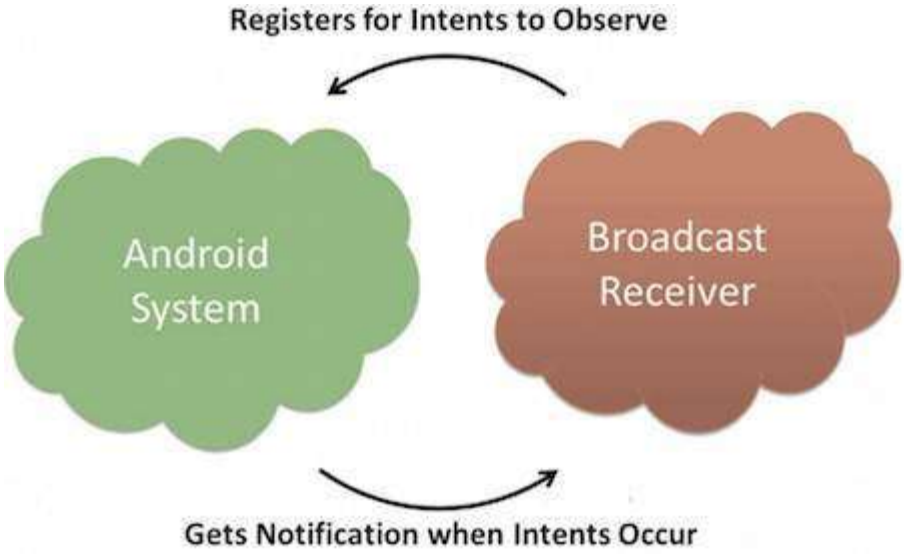
For example, beginning with Android 4.4 (API level 19), it's no longer necessary for your app to request the WRITE_EXTERNAL_STORAGE permission when your app wants to write to its own application-specific directories on external storage (the directories provided by getExternalFilesDir()). However, the permission *is required* for API level 18 and lower. So you can declare that this permission is needed only up to API level 18 with a declaration [20] such as this:

```
<uses-permission
    android:name="android.permission.WRITE_EXTERNAL_STORAGE"
    android:maxSdkVersion="18" />
```

Android - Broadcast Receivers

Broadcast Receivers simply respond to broadcast messages from other applications or from the system itself. These messages are sometime called events or intents.

For example, applications can also initiate broadcasts to let other applications know that some data has been downloaded to the device and is available for them to use, so this is broadcast receiver who will intercept this communication and will initiate appropriate action.



Fig(15): Broadcast-Receiver

```
<application
  android:icon="@drawable/ic_launcher"
  android:label="@string/app_name"
  android:theme="@style/AppTheme" >
  <receiver android:name="MyReceiver">

    <intent-filter>
      <action android:name="android.intent.action.BOOT_COMPLETED">
      </action>
    </intent-filter>

  </receiver> </application>
```

Fig (16): Code:1 Broadcast-Receicer

The Following Table Lists a Few Important System Events.

Sr.No	Event Constant & Description
1	**android.intent.action.BATTERY_CHANGED** Sticky broadcast containing the charging state, level, and other information about the battery.
2	**android.intent.action.BATTERY_LOW** Indicates low battery condition on the device.
3	**android.intent.action.BATTERY_OKAY** Indicates the battery is now okay after being low.
4	**android.intent.action.BOOT_COMPLETED** This is broadcast once, after the system has finished booting.
5	**android.intent.action.BUG_REPORT** Show activity for reporting a bug.
6	**android.intent.action.CALL** Perform a call to someone specified by the data.
7	**android.intent.action.CALL_BUTTON** The user pressed the "call" button to go to the dialer or other appropriate UI for placing a call.
8	**android.intent.action.DATE_CHANGED** The date has changed.
9	**android.intent.action.REBOOT** Have the device reboot.

Table (1): Important system events

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="Dummy Music"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/Theme.HelloWorld">
    <activity android:name=".MainActivity">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <receiver android:name="android.support.v4.media.session.MediaButtonReceiver">
        <intent-filter>
            <action android:name="android.intent.action.MEDIA_
        </intent-filter>
    </receiver>
    <service android:name="com.example.android.MediaPlaybackService" >
        <intent-filter>
            <action android:name="android.intent.action.MEDIA_BUTTON" />
        </intent-filter>
    </service>
</application>
```

Unresolved class 'MediaButtonReceiver'

Fig (17): Code:2 Broadcast-Receiver

Android-Intent-services

Intent Service is an extension of the Service component class that handles asynchronous requests (expressed as Intents) on demand. Clients send requests through Context.startService(Intent) calls; the service is started as needed, handles each Intent in turn using a worker thread, and stops itself when it runs out of work. This "work queue processor" pattern is commonly used to offload tasks from an application's main thread. The IntentService class exists to simplify this pattern and take care of the mechanics. To use it, extend IntentService and implement onHandleIntent(android.content.Intent). IntentService will receive the Intents, launch a worker thread, and stop the service as appropriate.

All requests are handled on a single worker thread -- they may take as long as necessary (and will not block the application's main loop), but only one request will be processed at a time[21].

Fig (18): Code Intent-Services

API calls

Application programming interfaces (APIs) are a way for one program to interact with another. API calls are the medium by which they interact. An API call, or API request, is a message sent to a server asking an API to provide a service or information.

If Jan is hosting a lot of guests for dinner, she might call a catering company and ask that they prepare food for the party. This saves her a great deal of time and effort preparing food herself. Similarly, one application can "call" another for needed data or functionality. This ensures developers do not have to spend time and effort building application capabilities that can be integrated via API.

Because APIs are integrated into almost all web applications today, API calls take place behind the scenes all the time. Suppose someone searches for bus tickets on a travel website. The travel website sends an API call to the various bus companies' servers and receives back information about what rides are available and how much they cost. From the user's perspective, this process should be almost instantaneous[22].

Feature Extraction

Collect all applications in separate folders which contain benign as well as suspicious applications respectively. Using “Glob” framework in python create an array of files is for further processing. Analyze each application in the array using pyaxmlparser [19] and Androguard [18] framework.

Malscan [20] is a framework which operates on API calls. It extracts the API Calls from .dex files which is obtained by Androguard. It extracts and saves the API calls in .gexf file format. It takes a set of sensitive API Calls commonly used by malicious apps and generates a CSV file in vector format as previously used. Different CSVs are made for various centrality types such as degree, katz and closeness.

Extract the following things in the analysis phase:

a. Permissions

b. Intent-receivers

c. Intent-services

d. API Calls

Taking these four attributes into consideration a program maps all attributes to a CSV file and mentions a class for each application. Once CSV files are generated, analyze them for any redundancy present, and if found, eliminate the entire row. Another program extracts the total permissions from these APK files. These permissions will work as attributes in the Dataset CSV File (Here if permission is present it is marked as 1 else it is marked as0). An N-bit Vector extracts search line in the CSV file, these vectors work as input to the machine learning algorithm[23].

Chapter (4): Machine Learning Techniques

ML is a branch of artificial intelligence that focuses on developing applications by learning from data without explicitly programming how the learned tasks are performed. The traditional ML methods make predictions based on past data. ML process lifecycle consists of multiple sequential steps. They are data extraction, data preprocessing, feature selection, model training, model evaluation, and model deployment. Supervised learning, unsupervised learning, semi-supervised learning, reinforcement learning, and deep learning are the different subcategories of ML. The supervised learning approach uses a labeled dataset to train the model to solve classification and regression problems depending on the output variable type (continuous or discreet). Unsupervised learning is used to identify the internal structures (clusters), and the characteristics of a dataset, and a labeled dataset is not required to train the model. The learning model and the data used for training are inferred. The model parameters are updated with the received feedback from the environment in reinforcement learning where no training data is involved. This ML method proceeds as prediction and evaluation cycles. To analyze the behavior of applications in the Android platform, two methods of analysis are commonly used. These are Dynamic and Static analysis.

We will focus on static analysis, The static analysis is mostly applicable for analyzing applications by disassembling packages and extracting source. Like dynamic analysis, different approaches exist in static analysis. Some researches focused on finding a sequence of op-code that reflects the malicious activity. The reason why op-code can be named as a feature is that every instruction in the computer program follows a specific structure. In this structure, op-code determines the action of this instruction and the rest is an address in memory. Malicious packages may follow the same action in instruction, and so, op-code can be a good classifier for detection[24].

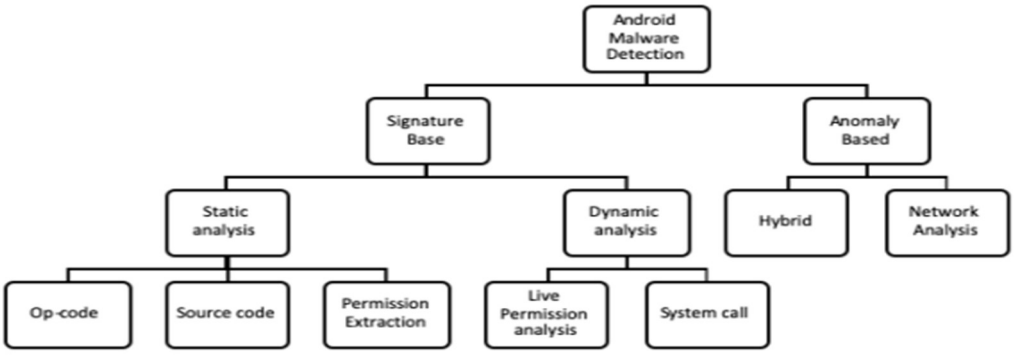


Fig (19): Android Malware Detection

Dataset Description

The Dataset used in this study is from an online project which tend to gift a machine learning-based system for the detection of malware on android devices.

The project which has the dataset is mainly focused on Permissions, Receivers, Services and API Calls. Permissions, Receivers and Services deals with features of android phone like WiFi, Bluetooth, Internet, etc which can share phone's information and data to different applications or over network. API Calls of an application gives information about how application is behaving when run on an android platform. All these features help us to know how an application is working and is this safe for our android device or not.

Dataset Location	Android Malware Detection Dataset used in DefenseDroid
Dataset Status	Balanced
Malware Samples	6000
Benign Samples	5975

Table (2): Dataset between Benign and Malware

Using our dataset we search for the most fifteen malicious permission in our data and how many it’s found in our data.

```
# Malicious
pd.Series.sort_values(permissions_data[permissions_data['class']==1].sum(axis=0), ascending=False)[:15]
```

```
class                                 6000
permission.INTERNET                   5872
permission.ACCESS_NETWORK_STATE       5755
permission.WRITE_EXTERNAL_STORAGE     5678
permission.READ_PHONE_STATE           5468
permission.ACCESS_WIFI_STATE          5370
permission.GET_TASKS                  4087
permission.WAKE_LOCK                  4079
permission.VIBRATE                    3836
permission.ACCESS_COARSE_LOCATION     3822
permission.READ_EXTERNAL_STORAGE      3562
permission.MOUNT_UNMOUNT_FILESYSTEMS  3555
permission.SYSTEM_ALERT_WINDOW        3547
permission.CHANGE_WIFI_STATE          3540
permission.ACCESS_FINE_LOCATION       3485
dtype: int64
```

Fig (20): Malicious Apps

```
pd.Series.sort_values(permissions_data[permissions_data['class']==0].sum(axis=0), ascending=False)[:15]
```

```
permission.INTERNET                   5846
permission.ACCESS_NETWORK_STATE       5656
permission.WRITE_EXTERNAL_STORAGE     4446
permission.WAKE_LOCK                  3976
permission.RECEIVE                    3106
permission.C2D_MESSAGE                2841
permission.ACCESS_WIFI_STATE          2754
permission.VIBRATE                    2666
permission.GET_ACCOUNTS               2387
permission.READ_PHONE_STATE           2161
permission.ACCESS_FINE_LOCATION       1896
permission.RECEIVE_BOOT_COMPLETED     1824
permission.READ_EXTERNAL_STORAGE      1794
permission.ACCESS_COARSE_LOCATION     1756
permission.CAMERA                     1256
dtype: int64
```

Fig (21): Benign Apps

1. ML in permission feature

Permission is a famous feature for Android malware detection, and it exposes the exact intent of the application. It is designed to protect user privacy on the Android platform. Permission is requested by an application to authorize itself to access specific hardware or software resource. For instance, the sensitive user data (such as SMS or contacts) or system features (such as a camera or internet). Based on the application nature, the system may grant permission or ask the user to grant it. Basically, no application has the authorization to access or read other application data, system files, and user private data (such as SMS).

In this study, we used the permission only to detect if an android application is Malicious or Benign based on Machine-Learning Algorisms and make an evaluation to select the best Machine-Learning Algorism to use in our case [25].

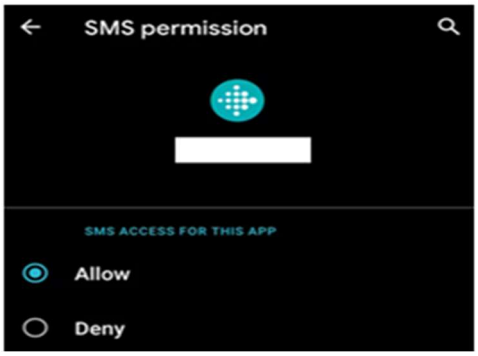


Fig (22): SMS permission

App permissions help support user privacy by protecting access to the following:

- Restricted data, such as system state and a user's contact information.
- Restricted actions, such as connecting to a paired device and recording audio.

Android categorizes permissions into different types, including install-time permissions, runtime permissions, and special permissions. Each permission type indicates the scope of restricted data that your app can access, and the scope of restricted actions that your app can perform, when the system grants your app that permission.

In our project, we use some ML learning techniques based on permission feature extraction. proposed a model for Android malware detection using machine learning. The authors used the ensemble technique to get more accurate results. The results for classification with the SVM algorithm were 90.5%, Random Forest was 91.06% and decision tree was 88.6%

We used several standard evaluation metrics to evaluate the performance of our model. The most standard metrics include accuracy, precision, sensitivity (recall), specificity, and fscore (F1). They are calculated as follows:

$$\text{accuracy} = \frac{TP+TN}{TP+FP+TN+FN}$$

$$\text{precision} = \frac{TP}{TP+FP}$$

$$\text{recall} = \frac{TP}{TP+}$$

I- Support Vector Machine (SVM) result

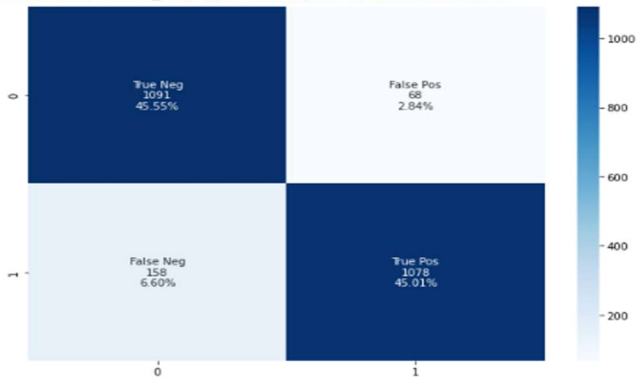
```
from sklearn import svm
clf = svm.SVC()
clf.fit(x_train, y_train)
pred = clf.predict(x_test)
cm=confusion_matrix(y_test, pred)
accuracy = accuracy_score(pred, y_test)
print(accuracy)
print(classification_report(pred, y_test, labels=None))
filename = 'svm.joblib'
joblib.dump(clf,filename)
```

```
0.9056367432150313
              precision    recall  f1-score   support

           0       0.94      0.87      0.91      1249
           1       0.87      0.94      0.91      1146

    accuracy                           0.91      2395
   macro avg       0.91      0.91      0.91      2395
weighted avg       0.91      0.91      0.91      2395
```

And Confusion Matrix

```
plt.figure(figsize = (10,7))
group_names = ['True Neg','False Pos','False Neg','True Pos']
group_counts = ["{0:0.0f}".format(value) for value in
                cm.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in
                     cm.flatten()/np.sum(cm)]
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sn.heatmap(cm, annot=labels, fmt='', cmap='Blues')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2033763110>
```




```
loaded_model = joblib.load('svm.joblib')
loaded_model.predict(x_test)
```

```
array([0, 0, 1, ..., 1, 1, 0])
```

Fig (23): SVM of permission

II- Decision tree

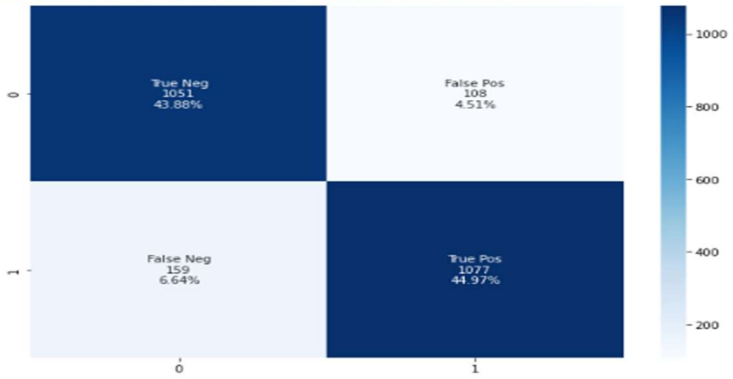
```
from sklearn import tree
clf = tree.DecisionTreeClassifier()
clf.fit(x_train, y_train)
pred = clf.predict(x_test)
cm=confusion_matrix(y_test, pred)
accuracy = accuracy_score(pred, y_test)
print(accuracy)
print(classification_report(pred, y_test, labels=None))

filename = 'decision_tree.joblib'
joblib.dump(clf,filename)
```

```
0.886847599164927
              precision    recall  f1-score   support

           0       0.91      0.87      0.89      1212
           1       0.87      0.91      0.89      1183

    accuracy                           0.89      2395
   macro avg       0.89      0.89      0.89      2395
weighted avg       0.89      0.89      0.89      2395

['decision_tree.joblib']
```

And Confusion Matrix

```
plt.figure(figsize = (10,7))
group_names = ['True Neg','False Pos','False Neg','True Pos']
group_counts = ["{0:0.0f}".format(value) for value in
                cm.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in
                     cm.flatten()/np.sum(cm)]
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sn.heatmap(cm, annot=labels, fmt='', cmap='Blues')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2033126310>
```




```
loaded_model = joblib.load('decision_tree.joblib')
loaded_model.predict(x_test)
```

```
array([0, 1, 1, ..., 1, 1, 0])
```

Fig (24): Decision of permission

III- Random Forest

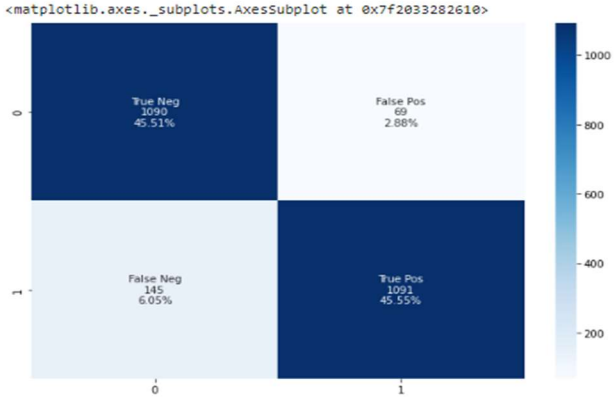
```
from sklearn.ensemble import RandomForestClassifier
rdF=RandomForestClassifier(n_estimators=250, max_depth=50,random_state=45)
rdF.fit(x_train,y_train)
pred=rdF.predict(x_test)
cm=confusion_matrix(y_test, pred)
accuracy = accuracy_score(y_test,pred)
print(accuracy)
print(classification_report(y_test,pred, labels=None))


filename = 'random_forest.joblib'
joblib.dump(clf,filename)
```

```
0.9106471816283925
              precision    recall  f1-score   support

           0       0.88      0.94      0.91      1159
           1       0.94      0.88      0.91      1236

    accuracy                           0.91      2395
   macro avg       0.91      0.91      0.91      2395
weighted avg       0.91      0.91      0.91      2395

['random_forest.joblib']
```

And Confusion Matrix

```
plt.figure(figsize = (10,7))
group_names = ['True Neg','False Pos','False Neg','True Pos']
group_counts = ["{0:0.0f}".format(value) for value in
                cm.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in
                     cm.flatten()/np.sum(cm)]
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sn.heatmap(cm, annot=labels, fmt='', cmap='Blues')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2033282610>
```




```
[ ]  loaded_model = joblib.load('random_forest.joblib')
     loaded_model.predict(x_test)
```

```
array([0, 1, 1, ..., 1, 1, 0])
```

Fig (25): Random Forest of permission

2. ML in Receiver feature

In our project, we use some ML learning techniques based on permission feature extraction. proposed a model for Android malware detection using machine learning. The authors used the ensemble technique to get more accurate results[26]. The results for classification with the SVM algorithm were 78.45%, Random Forest was 77.47% and decision tree was 77.99%

I- Support Vector Machine (SVM) result

```
from sklearn import svm
clf = svm.SVC()
clf.fit(x_train, y_train)
pred = clf.predict(x_test)
cm=confusion_matrix(y_test, pred)
accuracy = accuracy_score(pred, y_test)
print(accuracy)
print(classification_report(pred, y_test, labels=None))
filename = 'svm.joblib'
joblib.dump(clf,filename)
```

```
0.7849686847599165
              precision    recall  f1-score   support

           0       0.64      0.88      0.74       842
           1       0.92      0.73      0.82      1553

    accuracy                           0.78      2395
   macro avg       0.78      0.81      0.78      2395
weighted avg       0.82      0.78      0.79      2395


['svm.joblib']
```

And Confusion Matrix

```
plt.figure(figsize = (10,7))
group_names = ['True Neg','False Pos','False Neg','True Pos']
group_counts = ["{0:0.0f}".format(value) for value in
                cm.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in
                     cm.flatten()/np.sum(cm)]
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sn.heatmap(cm, annot=labels, fmt='', cmap='Blues')
```

```
<AxesSubplot:>
```

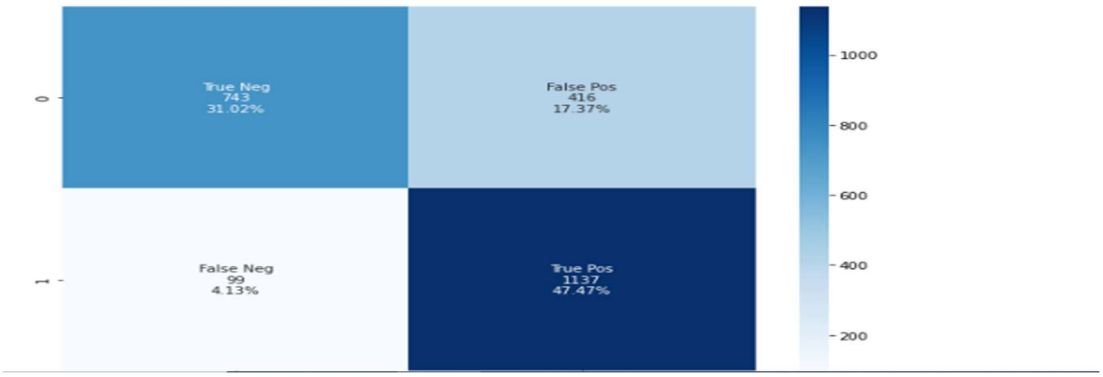


Fig (26): SVM of Receiver

II- Decision tree

```
from sklearn import tree
clf = tree.DecisionTreeClassifier()
clf.fit(x_train, y_train)
pred = clf.predict(x_test)
cm=confusion_matrix(y_test, pred)
accuracy = accuracy_score(pred, y_test)
print(accuracy)
print(classification_report(pred, y_test, labels=None))

filename = 'decision_tree.joblib'
joblib.dump(clf,filename)
```

```
0.7799582463465553
              precision    recall  f1-score   support

           0       0.65      0.86      0.74       866
           1       0.91      0.73      0.81      1529

    accuracy                           0.78      2395
   macro avg       0.78      0.80      0.77      2395
weighted avg       0.81      0.78      0.78      2395
```

And Confusion Matrix

```
plt.figure(figsize = (10,7))
group_names = ['True Neg','False Pos','False Neg','True Pos']
group_counts = ["{0:0.0f}".format(value) for value in
                cm.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in
                     cm.flatten()/np.sum(cm)]
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sn.heatmap(cm, annot=labels, fmt='', cmap='Blues')
```

```
<AxesSubplot:>
```

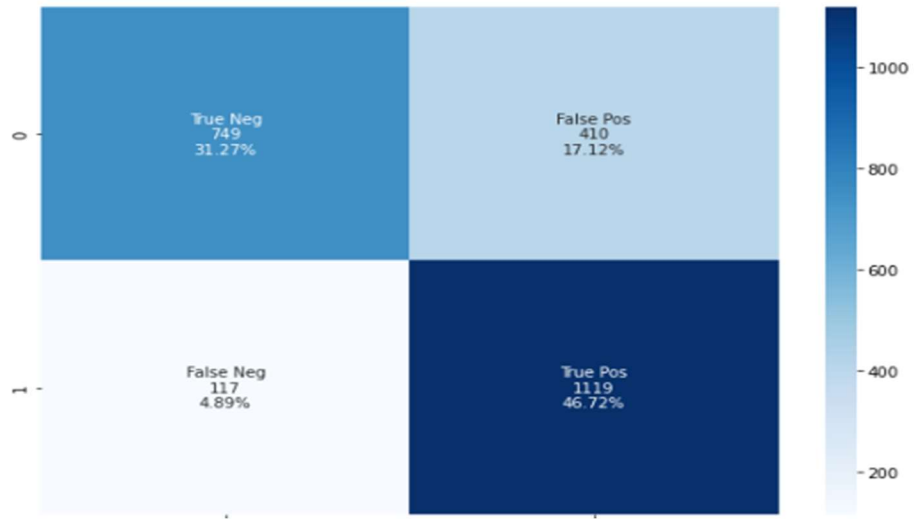


Fig (27): Decision tree of Receiver

III- Random Forest

```
from sklearn.ensemble import RandomForestClassifier
rdF=RandomForestClassifier(n_estimators=250, max_depth=50,random_state=45)
rdF.fit(x_train,y_train)
pred=rdF.predict(x_test)
cm=confusion_matrix(y_test, pred)
accuracy = accuracy_score(y_test,pred)
print(accuracy)
print(classification_report(y_test,pred, labels=None))


filename = 'random_forest.joblib'
joblib.dump(clf,filename)
```

```
0.7774530271398747
              precision    recall  f1-score   support

           0       0.90      0.61      0.73      1159
           1       0.72      0.93      0.81      1236

    accuracy                           0.78      2395
   macro avg       0.81      0.77      0.77      2395
weighted avg       0.80      0.78      0.77      2395


['random_forest.joblib']
```

And Confusion Matrix

```
plt.figure(figsize = (10,7))
group_names = ['True Neg','False Pos','False Neg','True Pos']
group_counts = ["{0:0.0f}".format(value) for value in
                cm.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in
                     cm.flatten()/np.sum(cm)]
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sn.heatmap(cm, annot=labels, fmt='', cmap='Blues')
```

```
<AxesSubplot:>
```

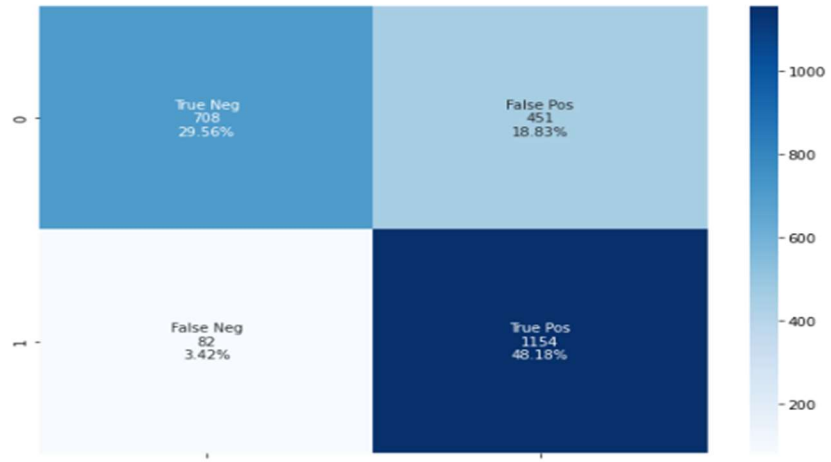


Fig (28): Random Forest of Receiver

3. ML in Services feature

In our project, we use some ML learning techniques based on Services feature extraction. proposed a model for Android malware detection using machine learning. The authors used the ensemble technique to get more accurate results[27]. The results for classification with the SVM algorithm were 80.70%, Random Forest was 80.66% and decision tree was 80.83%

1) Support Vector Machine (SVM) result

```
from sklearn import svm
clf = svm.SVC()
clf.fit(x_train, y_train)
pred = clf.predict(x_test)
cm=confusion_matrix(y_test, pred)
accuracy = accuracy_score(pred, y_test)
print(accuracy)
print(classification_report(pred, y_test, labels=None))
filename = 'svm.joblib'
joblib.dump(clf,filename)
```

```
0.807098121085595
              precision    recall  f1-score   support

           0       0.95      0.73      0.83      1511
           1       0.67      0.94      0.78       884

    accuracy                           0.81      2395
   macro avg       0.81      0.83      0.80      2395
weighted avg       0.85      0.81      0.81      2395


['svm.joblib']
```

And Confusion Matrix

```
plt.figure(figsize = (10,7))
group_names = ['True Neg','False Pos','False Neg','True Pos']
group_counts = ["{0:0.0f}".format(value) for value in
                cm.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in
                     cm.flatten()/np.sum(cm)]
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sn.heatmap(cm, annot=labels, fmt='', cmap='Blues')
```

```
<AxesSubplot:>
```

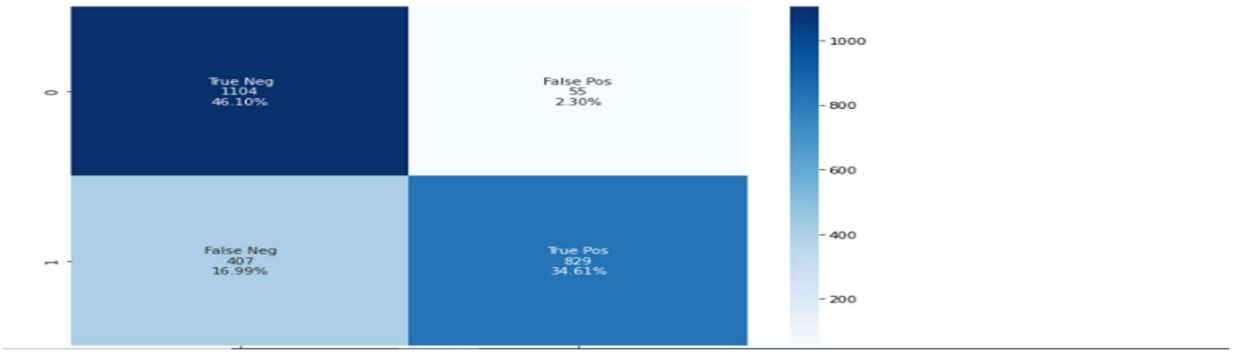


Fig (29): SVM of Services

2) Decision tree

```
from sklearn import tree
clf = tree.DecisionTreeClassifier()
clf.fit(x_train, y_train)
pred = clf.predict(x_test)
cm=confusion_matrix(y_test, pred)
accuracy = accuracy_score(pred, y_test)
print(accuracy)
print(classification_report(pred, y_test, labels=None))

filename = 'decision_tree.joblib'
joblib.dump(clf,filename)
```

```
0.8083507306889353
              precision    recall  f1-score   support

           0       0.96      0.73      0.83      1514
           1       0.67      0.94      0.78       881

    accuracy                           0.81      2395
   macro avg       0.81      0.84      0.81      2395
weighted avg       0.85      0.81      0.81      2395
```

And Confusion Matrix

```
plt.figure(figsize = (10,7))
group_names = ['True Neg','False Pos','False Neg','True Pos']
group_counts = ["{0:0.0f}".format(value) for value in
                cm.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in
                     cm.flatten()/np.sum(cm)]
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sn.heatmap(cm, annot=labels, fmt='', cmap='Blues')
```

```
<AxesSubplot:>
```

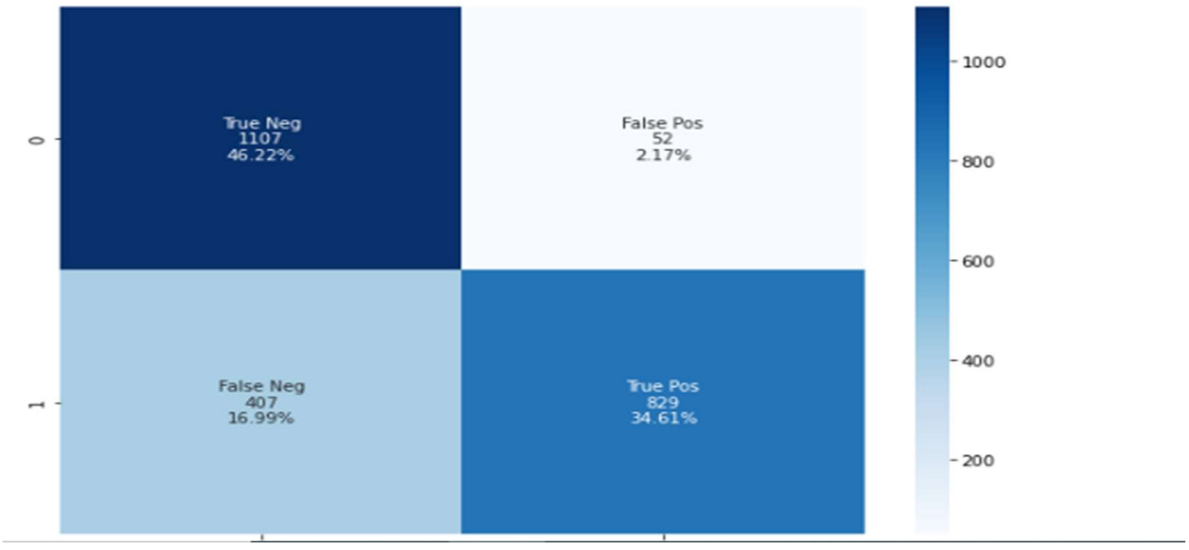


Fig (30): Decision tree of Services

3) Random Forest

```
from sklearn.ensemble import RandomForestClassifier
rdF=RandomForestClassifier(n_estimators=250, max_depth=50,random_state=45)
rdF.fit(x_train,y_train)
pred=rdF.predict(x_test)
cm=confusion_matrix(y_test, pred)
accuracy = accuracy_score(y_test,pred)
print(accuracy)
print(classification_report(y_test,pred, labels=None))


filename = 'random_forest.joblib'
joblib.dump(clf,filename)
```

```
0.8066805845511482
              precision    recall  f1-score   support

           0       0.72      0.97      0.83      1159
           1       0.96      0.65      0.78      1236

    accuracy                           0.81      2395
   macro avg       0.84      0.81      0.80      2395
weighted avg       0.85      0.81      0.80      2395


['random_forest.joblib']
```

And Confusion Matrix

```
plt.figure(figsize = (10,7))
group_names = ['True Neg','False Pos','False Neg','True Pos']
group_counts = ["{0:0.0f}".format(value) for value in
                cm.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in
                     cm.flatten()/np.sum(cm)]
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sn.heatmap(cm, annot=labels, fmt='', cmap='Blues')
```

```
<AxesSubplot:>
```

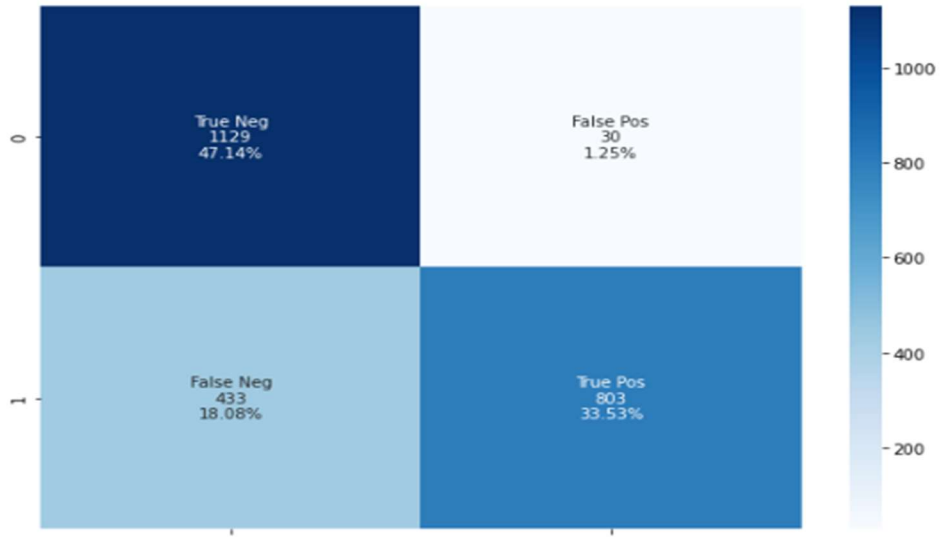


Fig (31): Random Forest of Services

Chapter (5): Deep learning techniques

Deep learning is a subset of machine learning, which is essentially a neural network with three or more layers. These neural networks attempt to simulate the behavior of the human brain allowing it to "learn" from large amounts of data. While a neural network with a single layer can still make approximate predictions, additional hidden layers can help to optimize and refine for accuracy. Deep learning drives much artificial intelligence (AI) applications and services that improve automation, performing analytical and physical tasks without human intervention.

The application of deep learning to Android malware detection has become a significant trend. A typical deep learning model is a very deep neural network that uses multiple hidden layers of many interconnected neurons to process data. Each layer comprises many different neurons in deep neural networks, each with different weights and possibly different activation functions. When the data is applied to a neural network, the loss function calculates the prediction error. The optimizer is used to update the weights to reduce the loss function error and improve the accuracy gradually. It trains the data and evaluates the accuracy of the test set.

- **ANN:** An artificial neuron network (neural network) is a computational model that mimics the way nerve cells work in the human brain.

Artificial neural networks (ANNs): use learning algorithms that can independently make adjustments - or learn, in a sense - as they receive new input. This makes them a very effective tool for non-linear statistical data modeling.

An artificial neural network has three or more layers that are interconnected. The first layer consists of input neurons. Those neurons send data on to the deeper layers, which in turn will send the final output data to the last output layer.

All the inner layers are hidden and are formed by units which adaptively change the information received from layer to layer through a series of transformations. Each layer acts both as an input and output layer that allows the ANN to understand more complex objects. Collectively, these inner layers are called the neural layer.

The units in the neural layer try to learn about the information gathered by weighing it according to the ANN's internal system. These guidelines allow units to generate a transformed result, which is then provided as an output to the next layer[28].

ANN technique on Permission feature:

In our project, we use some DL learning techniques based on Services feature extraction. proposed a model for Android malware detection using DL. The authors used the ensemble technique to get more accurate results. The results for classification with the ANN algorithm were 90.5%, LSTM was 96.85%

```
[ ] from tensorflow import keras
    from tensorflow.keras import layers

    model = keras.Sequential([
        keras.layers.Dense(32, input_shape=(1490,), activation='relu'),
        keras.layers.Dense(16, activation = 'relu'),
        keras.layers.Dense(2, activation = 'sigmoid')
    ])

    # Compiling the model
    model.compile(optimizer=keras.optimizers.RMSprop(learning_rate=1e-3),
                  loss=keras.losses.SparseCategoricalCrossentropy(),
                  metrics=['accuracy'])

    # fitting the model
    history = model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=10, batch_size=8)
```

```
Epoch 1/10
1198/1198 [==============================] - 5s 4ms/step - loss: 0.3027 - accuracy: 0.8895 - val_loss: 0.2689 - val_accuracy: 0.9044
Epoch 2/10
1198/1198 [==============================] - 3s 3ms/step - loss: 0.2631 - accuracy: 0.9043 - val_loss: 0.2643 - val_accuracy: 0.9035
Epoch 3/10
1198/1198 [==============================] - 3s 3ms/step - loss: 0.2592 - accuracy: 0.9063 - val_loss: 0.2643 - val_accuracy: 0.9035
Epoch 4/10
1198/1198 [==============================] - 3s 3ms/step - loss: 0.2550 - accuracy: 0.9087 - val_loss: 0.2655 - val_accuracy: 0.9027
Epoch 5/10
1198/1198 [==============================] - 3s 3ms/step - loss: 0.2526 - accuracy: 0.9102 - val_loss: 0.2727 - val_accuracy: 0.9081
Epoch 6/10
1198/1198 [==============================] - 3s 2ms/step - loss: 0.2484 - accuracy: 0.9117 - val_loss: 0.2765 - val_accuracy: 0.9048
Epoch 7/10
1198/1198 [==============================] - 3s 3ms/step - loss: 0.2453 - accuracy: 0.9149 - val_loss: 0.2694 - val_accuracy: 0.9052
Epoch 8/10
1198/1198 [==============================] - 3s 3ms/step - loss: 0.2416 - accuracy: 0.9152 - val_loss: 0.2719 - val_accuracy: 0.9031
Epoch 9/10
1198/1198 [==============================] - 3s 3ms/step - loss: 0.2412 - accuracy: 0.9163 - val_loss: 0.2718 - val_accuracy: 0.9035
Epoch 10/10
```

Fig (32): ANN of permission

Summarize history for loss & accuracy

```
[ ]  model.evaluate(x_test, y_test, batch_size=8)
```

```
300/300 [==============================] - 1s 2ms/step - loss: 0.2808 - accuracy: 0.9035
[0.2807779610157013, 0.903549075126648]
```

```
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

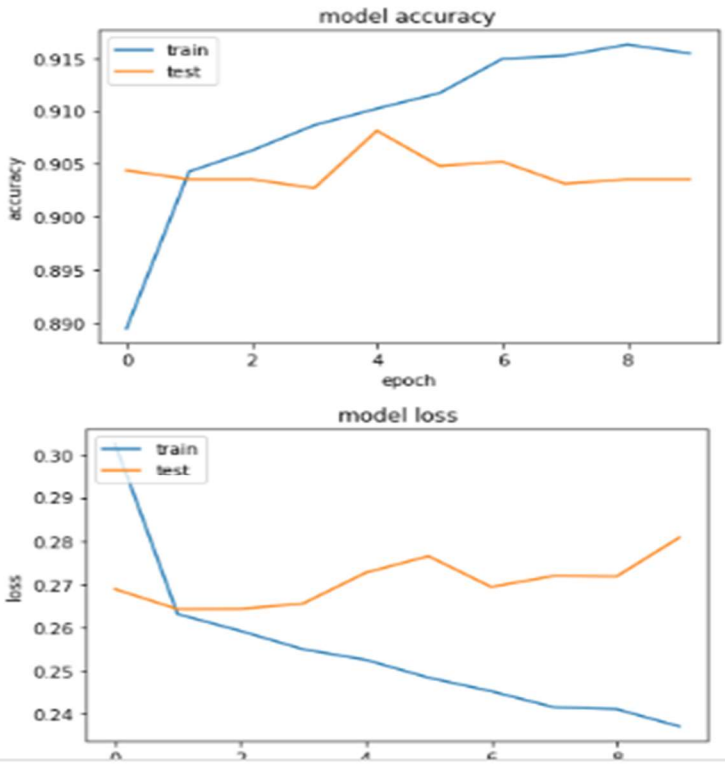


Fig (33): Model of Loss & accuracy

And Confusion Matrix

```
[ ]  cm=confusion_matrix(y_test, pred)
     plt.figure(figsize = (10,7))
     group_names = ['True Neg','False Pos','False Neg','True Pos']
     group_counts = ["{0:0.0f}".format(value) for value in
                     cm.flatten()]
     group_percentages = ["{0:.2%}".format(value) for value in
                          cm.flatten()/np.sum(cm)]
     labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
               zip(group_names,group_counts,group_percentages)]
     labels = np.asarray(labels).reshape(2,2)
     sn.heatmap(cm, annot=labels, fmt='', cmap='Blues')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd88de9e550>
```

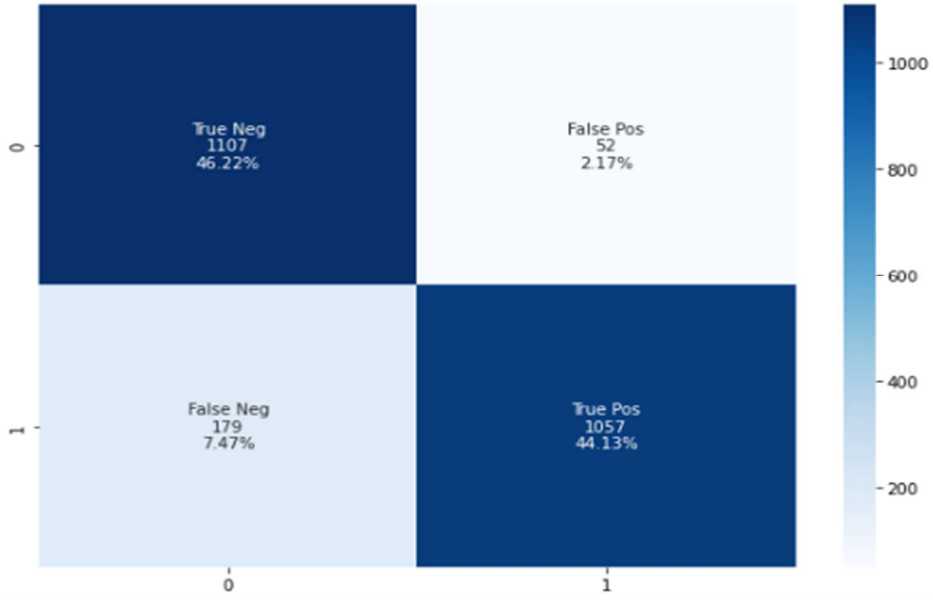


Fig (34): History of Loss & accuracy

- **LSTM**

Long short-term memory (LSTM) units or blocks are part of a recurrent neural network structure. Recurrent neural networks are made to utilize certain types of artificial memory processes that can help these artificial intelligence programs to more effectively imitate human thought.

The recurrent neural network uses long short-term memory blocks to provide context for the way the program receives inputs and creates outputs. The long short-term memory block is a complex unit with various components such as weighted inputs, activation functions, inputs from previous blocks and eventual outputs.

I- LSTM technique on Permission feature

```
#epoch-check
#path-check

model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
model.fit(X_train, y_train, batch_size=batch_size, epochs=10, validation_data=(X_test, y_test))
model.save("PermissionModel.hdf5")
```

```
Epoch 1/10
7/7 [==============================] - 14s 468ms/step - loss: 0.6932 - accuracy: 0.4620 - val_loss: 0.6932 - val_accuracy: 0.4975
Epoch 2/10
7/7 [==============================] - 0s 26ms/step - loss: 0.6936 - accuracy: 0.4662 - val_loss: 0.6932 - val_accuracy: 0.4975
Epoch 3/10
7/7 [==============================] - 0s 24ms/step - loss: 0.6932 - accuracy: 0.4849 - val_loss: 0.6931 - val_accuracy: 0.5025
Epoch 4/10
7/7 [==============================] - 0s 24ms/step - loss: 0.6932 - accuracy: 0.5005 - val_loss: 0.6931 - val_accuracy: 0.5025
Epoch 5/10
7/7 [==============================] - 0s 26ms/step - loss: 0.6931 - accuracy: 0.5094 - val_loss: 0.6931 - val_accuracy: 0.5025
Epoch 6/10
7/7 [==============================] - 0s 25ms/step - loss: 0.6931 - accuracy: 0.5271 - val_loss: 0.6931 - val_accuracy: 0.5025
Epoch 7/10
7/7 [==============================] - 0s 24ms/step - loss: 0.6932 - accuracy: 0.5113 - val_loss: 0.6931 - val_accuracy: 0.5025
Epoch 8/10
7/7 [==============================] - 0s 30ms/step - loss: 0.6932 - accuracy: 0.4998 - val_loss: 0.6931 - val_accuracy: 0.5025
Epoch 9/10
7/7 [==============================] - 0s 24ms/step - loss: 0.6931 - accuracy: 0.5138 - val_loss: 0.6931 - val_accuracy: 0.5025
Epoch 10/10
7/7 [==============================] - 0s 24ms/step - loss: 0.6934 - accuracy: 0.4831 - val_loss: 0.6931 - val_accuracy: 0.5025
```

ACCURACY CHECK

```
accuracy_score(y_test, y_pred)
```

```
0.9658426590947052
```

And Confusion Matrix

```
from sklearn.metrics import confusion_matrix
cf = confusion_matrix(y_test, y_pred)
print(cf)
#sns.heatmap(cf, annot=True)
sns.heatmap(cf/np.sum(cf), annot=True, fmt='.2%', cmap='Blues')
print(precision_score(y_test, y_pred, average=None))
```

```
[[5794  181]
 [ 228 5771]]
[0.962 0.97 ]
```

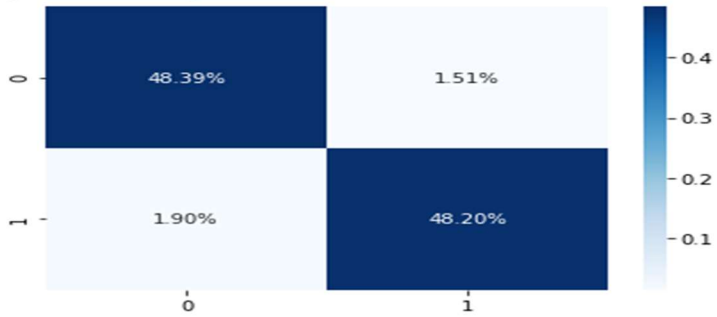


Fig (35): LSTM of permission

II- LSTM technique on Receiver feature

In our project, we use some DL learning techniques based on Receiver feature extraction. proposed a model for Android malware detection using DL. The authors used the ensemble technique to get more accurate results. The results for classification with the LSTM was 85.48%

```
#epoch-check
#path-check

model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
model.fit(X_train, y_train, batch_size=batch_size, epochs=500, validation_data=(X_test, y_test))
model.save("/content/drive/MyDrive/Project BE 2020-2021/Quarter Final/Receivers/Receivers_Model_LSTM_500ep_softmax.hdf5")
```

```
375/375 [==============================] - 4s 12ms/step - loss: 0.2669 - accuracy: 0.4984 - val_loss: 0.2689 - val_accuracy:
0.5010
Epoch 179/500
375/375 [==============================] - 4s 12ms/step - loss: 0.2744 - accuracy: 0.4980 - val_loss: 0.2689 - val_accuracy:
0.5010
Epoch 180/500
375/375 [==============================] - 4s 11ms/step - loss: 0.2689 - accuracy: 0.4998 - val_loss: 0.2688 - val_accuracy:
0.5010
Epoch 181/500
375/375 [==============================] - 4s 12ms/step - loss: 0.2723 - accuracy: 0.5059 - val_loss: 0.2688 - val_accuracy:
0.5010
Epoch 182/500
375/375 [==============================] - 5s 12ms/step - loss: 0.2765 - accuracy: 0.5047 - val_loss: 0.2689 - val_accuracy:
0.5010
Epoch 183/500
375/375 [==============================] - 5s 12ms/step - loss: 0.2689 - accuracy: 0.5036 - val_loss: 0.2687 - val_accuracy:
0.5010
Epoch 184/500
375/375 [==============================] - ETA: 0s - loss: 0.2663 - accuracy: 0.4895
```


ACCURACY CHECK

```
accuracy_score(y_test, y_pred)
```

```
0.8242024386170035
```

Fig (36): LSTM of Receiver

III- LSTM technique on Services feature

In our project, we use some DL learning techniques based on Services feature extraction. proposed a model for Android malware detection using DL. The authors used the ensemble technique to get more accurate results. The results for classification with the LSTM was 85.48%

```
In [12]: #epoch-check
         #path-check

         model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
         model.fit(X_train, y_train, batch_size=batch_size, epochs=500, validation_data=(X_test, y_test))
         model.save("/content/drive/MyDrive/Project BE 2020-2021/Quarter Final/Services/Services_Model_LSTM_500ep.hdf5")
```

```
0.8549
Epoch 495/500
375/375 [==============================] - 5s 12ms/step - loss: 0.2530 - accuracy: 0.8564 - val_loss: 0.2510 - val_accuracy:
0.8549
Epoch 496/500
375/375 [==============================] - 5s 12ms/step - loss: 0.2493 - accuracy: 0.8617 - val_loss: 0.2510 - val_accuracy:
0.8549
Epoch 497/500
375/375 [==============================] - 5s 12ms/step - loss: 0.2428 - accuracy: 0.8586 - val_loss: 0.2509 - val_accuracy:
0.8549
Epoch 498/500
375/375 [==============================] - 5s 12ms/step - loss: 0.2564 - accuracy: 0.8498 - val_loss: 0.2510 - val_accuracy:
0.8549
Epoch 499/500
375/375 [==============================] - 5s 12ms/step - loss: 0.2504 - accuracy: 0.8528 - val_loss: 0.2509 - val_accuracy:
0.8549
Epoch 500/500
375/375 [==============================] - 5s 12ms/step - loss: 0.2527 - accuracy: 0.8543 - val_loss: 0.2509 - val_accuracy:
0.8549
```

ACCURACY CHECK

```
: accuracy_score(y_test, y_pred)

: 0.8548521797227326
```

Fig (37): LSTM of Services

Difference between Deep Learning and Machine Learning

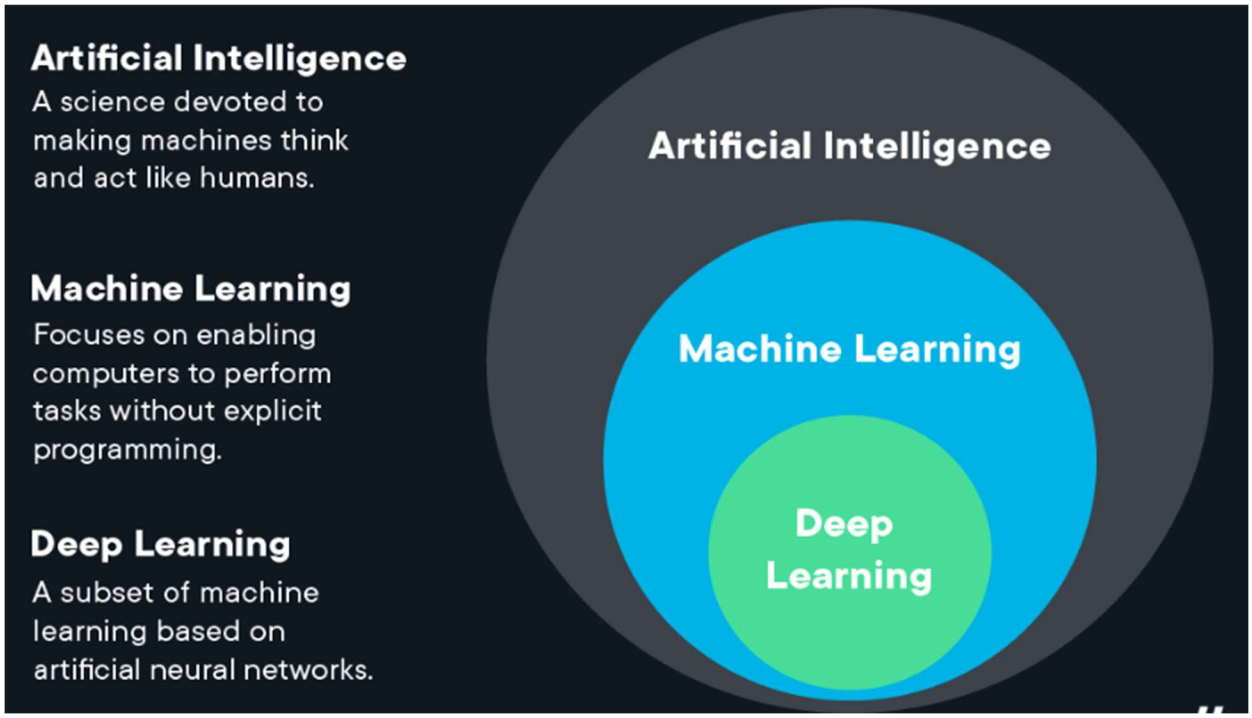


Fig (38): There are many differences between these two subsets of artificial intelligence, here are five of the most important:

1. Human Intervention: Machine learning requires more ongoing human intervention to get results. Deep learning is more complex to set up but requires minimal intervention thereafter.

2. Hardware: Machine learning programs tend to be less complex than deep learning algorithms and can often run on conventional computers, but deep learning systems require far more powerful hardware and resources. This demand for power has driven has meant increased use of graphical processing units. GPUs are useful for their high bandwidth memory and ability to hide latency (delays) in memory transfer due to thread parallelism (the ability of many operations to run efficiently at the same time.)

3. Time: Machine learning systems can be set up and operate quickly but may be limited in the power of their results. Deep learning systems take more time to set up but can generate results instantaneously (although the quality is likely to improve over time as more data becomes available).

4. Approach: Machine learning tends to require structured data and uses traditional algorithms like linear regression. Deep learning employs neural networks and is built to accommodate large volumes of unstructured data.

5. Applications: Machine learning is already in use in your email inbox, bank, and doctor's office. Deep learning technology enables more complex and autonomous programs, like self-driving cars or robots that perform advanced surgery[29].

Chapter(6): Front End

Firstly, there were many challenges. For example, the model dealt with apks to extract selected features from it and even after getting apk of the app chosen, it is not practically useful to upload the apk file of the app to the server to do classifications on it due to time taken to upload files. For example, the WhatsApp apk file size is nearly 50 megabytes so to upload it, the user can uninstall the application immediately.

As a result, To solve uploading all the apk to the server, it succeeded to extract features including permission, receivers, and services on the front-end side then sending the extracted features as a string to the server in JSON format then the server receives the data and does its classifications and send the result to the app.

Using installed_apps plugin in Neuron

```
54
55      installed_apps: ^1.3.1
56
```

In the project's pubspec.yaml:

Getting apk path of apps:

```
Future apps() async {
  List<Application> apps = await DeviceApps.getInstalledApplications( includeAppIcons: true,
      onlyAppsWithLaunchIntent: true, includeSystemApps: true);

  var output;


  final items = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,3

  //[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,3

  for (int item in items) {
    String path = apps[item].apkFilePath.toString();
    //print(path);
    //print(apps[item].appName);

    await Future.delayed(const Duration(seconds: 0));


    output = await Runpython(path);
    //print("samereeeeee");

    //var ss=output.toString();


    Application app11 =  apps[item];
    app11 is ApplicationWithIcon?
    picture.add({"id": item , "img":app11.icon}) : picture.add({"id": item , "img": "assets/img/bo.PNG "});


    a.add({'id': item, 'name': apps[item].appName, 'features': output});
  }


  print(picture);

  return a;
```

Fig (39): Apk path of Apps

Chaquopy

The easiest way to use Python in an Android app. Chaquopy provides everything you need to include Python components in an Android app, including:

- Full integration with Android Studio's standard Gradle build system.
- Simple APIs for calling Python code from Java/Kotlin, and vice versa.
- A wide range of third-party Python packages, including SciPy, OpenCV, TensorFlow and many more.

Activating a license:

Once you have a license key, activate it by adding the following line to your project's local.properties file:

```
chaquopy.license=your_license_key
```

The notification and time limit will then be removed the next time you build the project.

Licensing an AAR:

If you're using Chaquopy in an Android library module (AAR), you'll also need to add this line to identify the app which the AAR will be built into:

```
chaquopy.applicationId=your.application.id
```

If you want to build a single AAR which is usable in multiple apps, please contact us to discuss your use case.

Using Chaquopy in neuron:

In the project's pubspec.yaml:

```
  http: ^0.13.4
  #installed_apps: ^1.3.0
  #device_apps: ^2.1.1
  #firebase_core: ^0.7.0
  #firebase_storage: ^7.0.0
  #file_picker: ^2.1.7
  flutter_search_bar: ^3.0.0-dev.1
  chaquopy: ^0.0.15
```

Fig (40): Pubspec.yaml using Chaquopy

In the project's top-level build.gradle file :

```
1
2
3   apply plugin: 'com.android.application'
4   apply plugin: 'com.chaquo.python'
5
```

```
android {
    compileSdkVersion 32

    defaultConfig {

        ndk {
            abiFilters "armeabi-v7a", "arm64-v8a"
        }

        python {

            buildPython "C:\\Users\\kareem\\AppData\\Local\\Programs\\Python\\Python38\\python.exe"
            //buildPython "C:\\Users\\kareem\\AppData\\Local\\Programs\\Python\\Python38\\python.exe", "-3.8"
            pip {
                //install "scripts"
                //install "scipy"
                install "numpy"
                install "keras.models"
                install "pyaxmlparser"
                //install "os"
                //install "pyrebase"
                //  specify any other package to install.
            }
```

Fig (41): top-level build.gradle file

Python script to extract features from apk:

```
final _result = await Chaquopy.executeCode(s2);


setState(() {
  _outputOrError = _result['textOutputOrError'] ?? '';
  //print(_outputOrError);

});



/*

final url = Uri.parse('http://192.168.1.6:5000//api');
final headers = {"Content-type": "application/json"};
final json = '{"title": ${_outputOrError}}';
final response = await post(url, headers: headers, body: json);

print('Status code: ${response.statusCode}');
print('Body: ${response.body}');



*/



return _outputOrError;
```

```
    String s2 = """
import os
from pyaxmlparser import APK
import numpy as np
import ast
import json

def process(arr1):
  res = []
  for i in arr1:
    temp = i.split('.')
    if 'permission' in temp:
      ind = temp.index('permission')
      res.append(".".join(temp[ind:]))
  return res
def reriveprocess(arr2):
  rest = []
  for i in arr2:
    temp2= i.split('.')
    if 'service' in temp2:
      indd = temp2.index('service')
      rest.append(".".join(temp2[indd:]))
    elif 'intent' in temp2:
      indd = temp2.index('intent')
      rest.append(".".join(temp2[indd:]))
    else:
      rest.append(i.split(".")[-1])
  return rest
def serviceprocess(arr3):
  rest = []
  for i in arr3:
    temp2= i.split('.')
    if 'service' in temp2:
      indd = temp2.index('service')
      rest.append(".".join(temp2[indd:]))
    elif 'intent' in temp2:
      indd = temp2.index('intent')
      rest.append(".".join(temp2[indd:]))
    else:
      rest.append(i.split(".")[-1])
  return rest

appundertest ='${path}'

a = APK(appundertest)
perms=process(a.get_permissions())
reci = reriveprocess(a.get_receivers())
se = serviceprocess(a.get_services())
quo={'perms':perms , "recievers":reci , "services":se}
fr=json.dumps(quo)



print(fr)""";
```

Fig (42): Python script to extract features from Apk

Neuron User Interface:

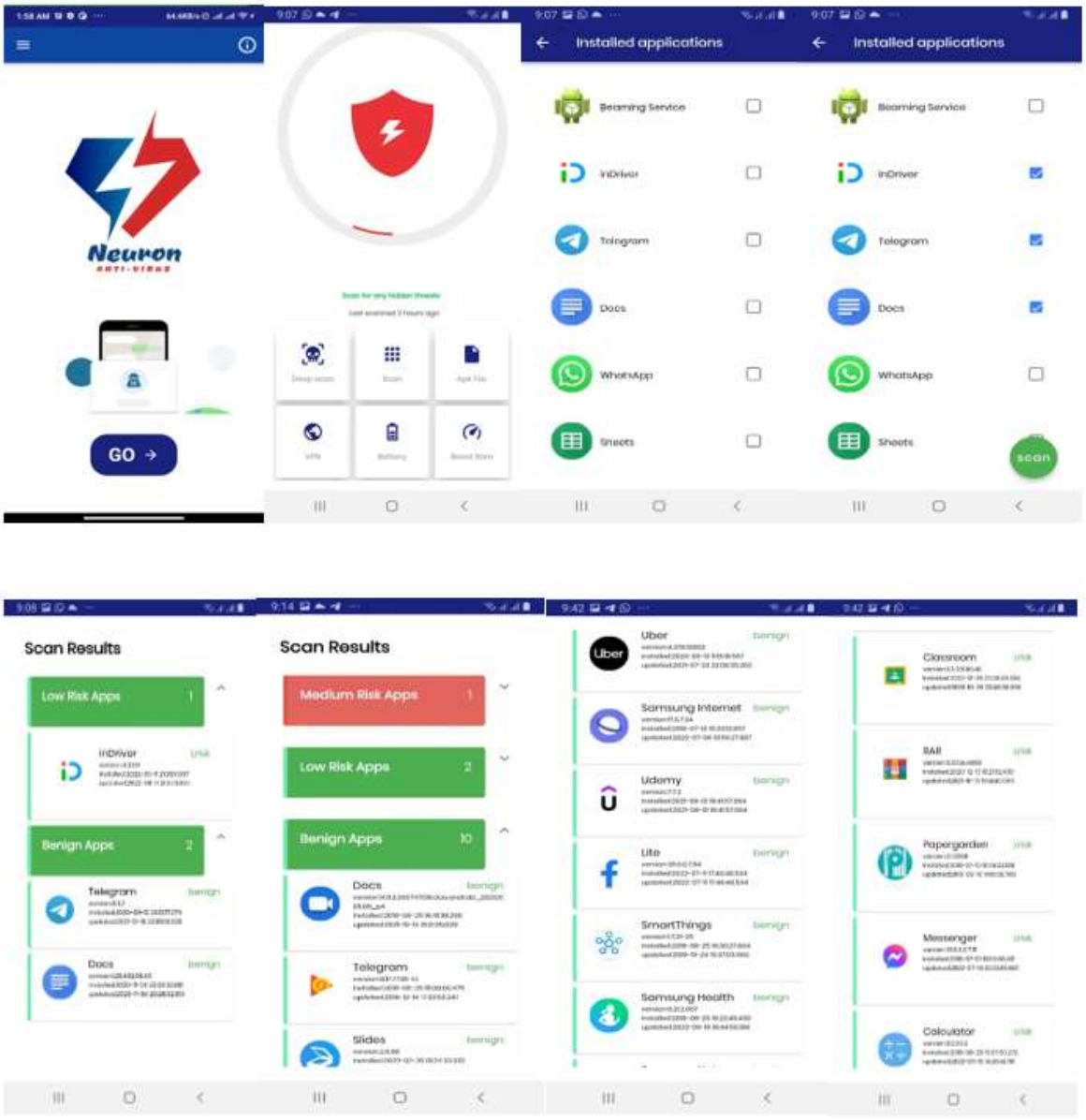


Fig (43): Neuron User Interface

Chapter (7): Back End

In the back-end side of the project we used Flask to program our server.

Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries. It has no database abstraction layer.

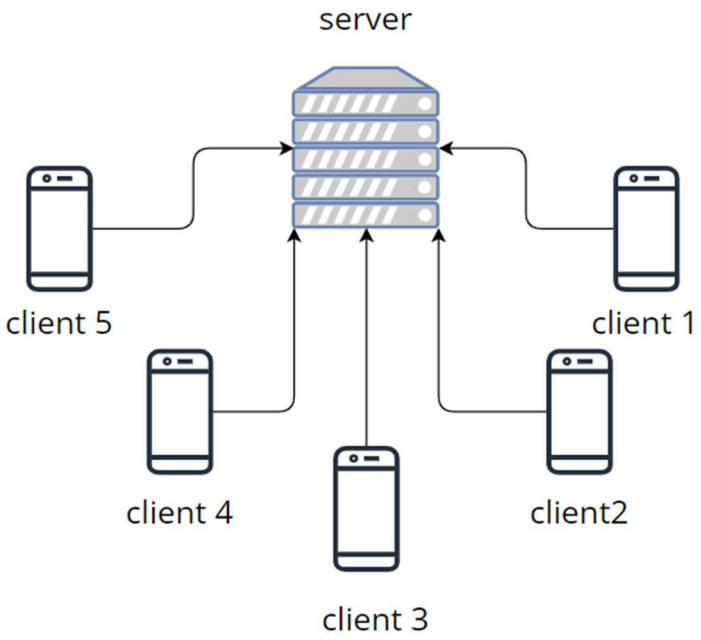


Fig (44): Figure network overview

The above image shows an abstract representation of the system as a whole.

Clients communicate with the server through **APIs**, clients are **mobile applications** sending information or receiving information .

The server then validates these requests then it reformats data to the suitable form expected by the client, then it sends it to the client.

At first to test the backend we make a website page to upload any txt file has the permissions granted by the app we want to scan and then after submit the result of prediction is given in shown in form of json.

The website page

Steps to test that our backend is working right:

1- Choose apk.txt(has permissions granted to the app)

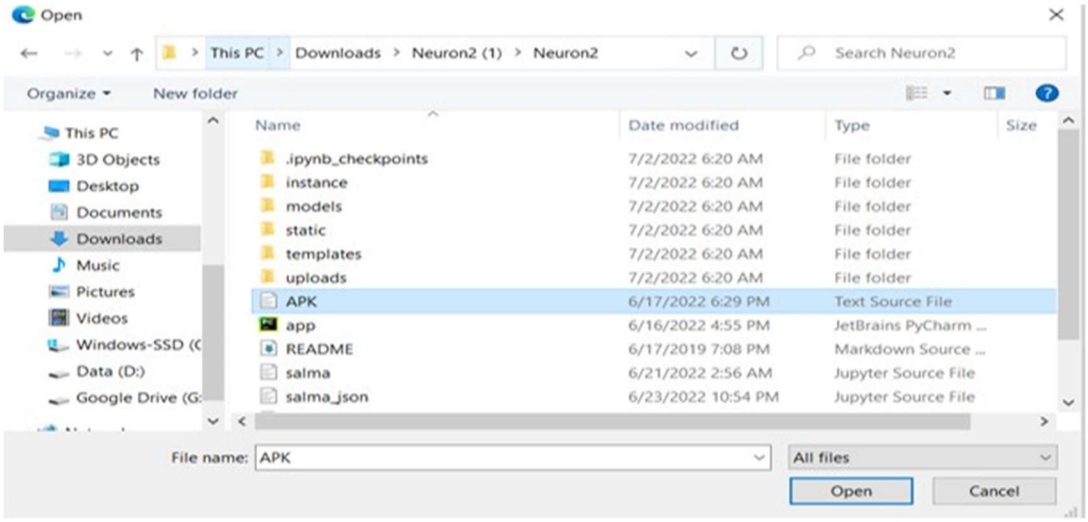


Fig (45): Choose apk.txt

2- Click submit

Fig (46): Click submit

3- after click submit the result of prediction is given in form of json

Fig (47): Resuly of prediction

as shown in the below image and the chosen txt file is save in uploads directory

Fig (48): Jupyter

making apk.txt:

Apk.txt has all the permissions granted by the apk we want to test we use the extracted permission feature code to extract permissions from this apk and put these permissions in apk.txt file

```
import os
from pyaxmlparser import APK
import numpy as np


def process(arr1):
  res = []
  for i in arr1:
    temp = i.split('.')
    if 'permission' in temp:
      ind = temp.index('permission')
      res.append(".".join(temp[ind:]))
  return res

appUndertest ='/storage/emulated/0/Download/facebook_lite_v299.0.0.11.111.apk'

a = APK(appUndertest)
perms=process(a.get_permissions())
print(perms)""";
```

Fig(49): extracted permission feature code

Backend testing code:

```
MODEL_PATH = 'models/PermissionModel.hdf5'
predictionModel = load_model(MODEL_PATH)

@app.route('/uploader', methods = ['GET', 'POST'])
def upload():

    if request.method == 'POST':
        f = request.files['file']
        filename = werkzeug.utils.secure_filename(f.filename)
        f.save("./uploads/"+filename)
        m="./uploads/"+filename


        preds = model_predict(m, predictionModel)

        print(preds)
        s=str(preds)
        result=""
        for i in s:
            if i=='[':
                pass

            elif i==']':
                pass
            else:
                result+=i
        print(s)
        print(result)
        print(m)
        json_dump = json.dumps({'num': s} , cls=NumpyEncoder)
        d={}
        d['num']=str(result)
    return d
```

Fig (50): Backend testing code

The model_predict function takes 2 parameter the permissions as a string and the prediction model(the model in hd5 format)

```
In [3]: def model_predict(perms2, model):
            import ast
            f1 = open('models/APK.txt')
            f1 = open(apk_path)
            perms2= f1.read()
            perms2= ast.literal_eval(perms2)
            predictionList1 = [0] * len(setPerms)
            for i in perms2:
                if i in setPerms:
                    predictionList1[setPerms.index(i)] = 1



            c=predictionModel.predict(np.array(predictionList1, ndmin=3))
            return c
```

Fig(51): model_predict function

In the model predict function, it reads the txt file which have the permissions(as a list) and start to convert it list of zeros and ones that describe the app permissions with respect to the dataset as if the app has the permission in the dataset we represent that as (1)or not we represent that as (0) as shown in fig2 , we do that as we can be able to represent the app permissions in the correct format for the model prediction


```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```


Fig(52): permissions of app as list of 0s and 1s

We use permissionListForPrediction.txt file to compare between permissions in the dataset to permissions in the app to make list of 0s and 1's represents the app permissions in the same shape and order of the dataset samples.

- **permissionListForPrediction.txt file contains permissions in our dataset**

```
setPerms = []
f1 = open('models/PermissionListForPrediction.txt')
temp = f1.read()
setPerms = temp[2:-2].split("', '")
setPerms
```

Fig(53): permissionListForPrediction.txt

Client

In our project, we expect the client to be a mobile application

client is can send login ,Register, apps.

The JSON body send from the user to the server :

Fig(54): Post request body

As from the above image , In the client side we store information for each app the user wants to scan in json form so json body has the following keys

- Id (identifier for the app)
- Name (name of the app)
- Features (has the static features extracted from the app , the value of this key is JSON it has three keys permissions , Receivers, Services[30].

Server

We used Flask package to create our server and Application Programming Interface (API)s. To access the incoming data in Flask, you have to use the request object. The request object holds all incoming data from the request, which includes the raw data, and we use it to check if the request method is GET or POST

```
import ast
@app.route('/appi', methods = ['GET', 'POST'])
def list_get():

    if request.method == 'POST':

        data = json.loads(request.data)
        print(data)
        qq=data["num"]
```

Fig (55): GET or POST

As we use a local server , when we run the flask backend it gives us a link to the de ployment
local server http://192.168.100.14:5000/ 192.168.100.14 is the address of the local host (computer) where we run the flask and 5000 is the port number By default, Fla sk runs on **port 5000** in development mode.

```
:
if __name__ == '__main__':
    app.run(host='0.0.0.0')
```

```
 * Serving Flask app "__main__" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on all addresses.
   WARNING: This is a development server. Do not use it in a production deployment.
 * Running on http://192.168.100.14:5000/ (Press CTRL+C to quit)
```

Implementation of the server

- **model prediction for each static features**

1) permission

```
In [25]: def model_predict1(rec, model):
             import ast
             f1 = open('models/APK.txt')

             perms2= f1.read()
             perms2= ast.literal_eval(perms2)
             predictionList1 = [0] * len(setPerms)
             for i in perms2:
                 if i in setPerms:
                     predictionList1[setPerms.index(i)] = 1
             c=predictionModel.predict(np.array(predictionList1, ndmin=3))
             print( predictionList1)
             return c
```

Fig (56): Model prediction of permission

2) services

```
In [18]: def model_predict2(rec, model):
             import ast
             predictionList2 = [0] * len(setRecievers)
             for i in rec:
                 if i in setRecievers:
                     predictionList2[setRecievers.index(i)] = 1
             c=predictionModel2.predict(np.array(predictionList2, ndmin=3))
             return c
```

Fig (57): Model prediction of Services

3) receivers

```
In [19]: def model_predict3(rec, model):
             import ast
             predictionList3 = [0] * len(setServices)
             for i in rec:
                 if i in setServices:
                     predictionList3[setServices.index(i)] = 1


             cc=predictionModel3.predict(np.array(predictionList3, ndmin=3))
             return cc
```

Fig (58): Model prediction of Receiver

As shown in the above images each feature has its model predict function which takes two parameters the extracted feature as string(permiisions or services or recivers) and the prediction model as hd5 format

Each of these fuctions reads the txt file which have the feature (as a list) and start to convert it list of zeros and ones that describe the app feature with respect to the dataset as if the app has the feature in the dataset we represent that as (1)or not we represent that as (0), we do that as we can be able to represent the app feature in the correct format for the model prediction

Then we use method predict in the desired prediction model for the feature and give the predictionList which represent the existance or absence of acertain example of this feature which exist in the dataset

This method returns the prediction result as a number its range is between 0(benign) and 1 (malicious)[31].

- **send result to the client**

In [9]:
```
import ast
@app.route('/appi', methods = ['GET', 'POST'])
def list_get():

    if request.method == 'POST':
        data = json.loads(request.data)
        print(data)
        qq=data["num"]
        aii=[]
        aii=qq
        for p in aii:
            gg=p["features"]
            yy=json.loads(gg)
            pp=yy["perms"]
            rr=yy["recievers"]
            ss=yy["services"]
            preds = model_predict(pp, predictionModel)
            preds2 = model_predict2(rr, predictionModel2)
            preds3 = model_predict3(ss, predictionModel3)
            ss=str(preds)
            ss2=str(preds2)
            ss3=str(preds3)
            d={}
            result=""
            resultt=""
            resulttt=""
            for i in ss:
                if i=='[':
                    pass

                elif i==']':
                    pass
                else:
                    result+=i
```

```
                pass

            elif i==']':
                pass
            else:
                resulttt+=i
        count = 0
        bm=""
        if preds > threshold:
            count += 1
        if preds2 > threshold:
            count += 1
        if preds3 > threshold:
            count += 1

        if count == 0 or count == 1:
            bm="benign"
        elif count == 2:
            bm="Mrisk"
        elif count == 3:
            bm="Hrisk"
        print(type(fl))
        d={"result_p":result ,"result_R":resultt,"result_S":resulttt, "id":idd , "name":name , "final":bm}
        a.append(d)
    return str(a)
```

Fig (59): Send result of client

In the above images we get the result of prediction by calling model_predict of each feature then after get the prediction result based on each feature we compare for each model prediction result with the threshold (we chose it to be 0.51).

we consider if the prediction model result less than threshold then app considered benign else malicious but we have three prediction model results so how to take decision the app is benign or malicious ?

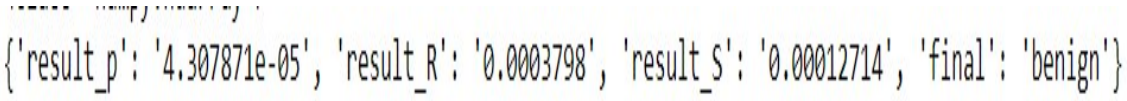
so we will take the majority opinion , we use count as indication to the number of opinions tells that the app is malicious (we consider each feature prediction model result as separate opinion) that's will be a good indication to us of the majority opinion.

if the model prediction result number is more than that threshold(malicious) we increment the count then if the count is equal to 0 then the 3 features prediction model says the app is benign else if count is equal to 1 then one of the decision makers says the app is malicious so we consider that as allow risk else if count is equal to 2 then two of the decision makers says the app is malicious so we consider that as a medium risk , else if count is equal to 3 then three of the decision makers says the app is malicious so we consider that a high risk.

Then we send the result to the client in list of JSON format each JSON in the list describes an app.

The JSON has 6 key :

- **result_p :** result of the permission model prediction based on permission feature
- **result_R:** result of the Receiver model prediction based on Receiver feature
- **result_S:** result of the service model prediction based on service feature
- **id:** identification number for the app
- **name:** app name
- **final:** the final result of the app as malicious or benign



Fig(60): The response from the server to the client

4) Application Programming Interface (API)s

In our server we exposed API for the Android mobile users to consume. We use http API as shown in the following image. An HTTP API is an API that uses Hypertext Transfer Protocol as the communication protocol between the two systems.

```
final response = await http.post(
  Uri.parse('http://192.168.100.14:5000//appi'),
```

Fig(61): Http API

We send in the body of the Post request a json with string key and value list :

```
final response = await http.post(
  Uri.parse('http://192.168.100.14:5000//appi'),
  headers: <String, String>{
    'Content-Type': 'application/json; charset=UTF-8',
  },
  body: jsonEncode(<String, List>{'num':vv}),
```

Fig(62): App scan API

As shown in the above image we expose an endpoint "/appi", this endpoint uses a POST method. In then, firstly we put the request data in the body key then we send it to the json wait until the prediction result (benign or malicious) is send to the server .the response code is set to 200 which indicate that the prediction is finished then we send the return the result of prediction to the client as the response.

For example, to scan an app, we firstly select the method type as POST and write the endpoint as localhost: /appi as shown below the method's body is a json object

If we send the request, we'll get this response. The response code is 200 which means that the prediction result created and returned to the client without any problems, and the response body has the app id , name and prediction result(Low risk or Medium risk or High risk or benign[32].

Extracted features in JSON:

{'id': 16, 'name': 'Telegram', features : '{"perms": ["permission.CAMERA", "permission.INSTALL_SHORTCUT", "permission.USE_FULL_SCREEN_INTENT", "permission.UPDATE_SHORTCUT", "permission.CALL_PHONE", "permission.INSTALL_SHORTCUT", "permission.READ", "permission.MODIFY_AUDIO_SETTINGS", "permission.MANAGE_ACCOUNTS", "permission.READ_PHONE_NUMBERS", "permission.READ_EXTERNAL_STORAGE", "permission.SYSTEM_ALERT_WINDOW", "permission.ACCESS_BACKGROUND_LOCATION", "permission.WRITE_SETTINGS", "permission.WRITE_SETTINGS", "permission.UNINSTALL_SHORTCUT", "permission.READ_CONTACTS", "permission.MAPS_RECEIVE", "permission.MANAGE_OWN_CALLS", "permission.ACCESS_FINE_LOCATION", "permission.REQUEST_INSTALL_PACKAGES", "permission.FOREGROUND_SERVICE", "permission.READ_SYNC_SETTINGS", "permission.READ_GSERVICES", "permission.READ_APP_BADGE", "permission.USE_BIOMETRIC", "permission.WRITE_EXTERNAL_STORAGE", "permission.RECORD_AUDIO", "permission.ACCESS_NETWORK_STATE", "permission.BADGE_COUNT_WRITE", "permission.WRITE_CONTACTS", "permission.AUTHENTICATE_ACCOUNTS", "permission.GET_ACCOUNTS", "permission.BROADCAST_BADGE", "permission.WRITE", "permission.BADGE_COUNT_READ", "permission.READ_CALL_LOG", "permission.ACCESS_WIFI_STATE", "permission.VIBRATE", "permission.READ_PROFILE", "permission.BLUETOOTH", "permission.READ_SETTINGS", "permission.UPDATE_COUNT", "permission.RECEIVE", "permission.WRITE_SYNC_SETTINGS", "permission.RECEIVE_BOOT_COMPLETED", "permission.UPDATE_BADGE", "permission.INTERNET", "permission.READ_SETTINGS", "permission.CHANGE_BADGE", "permission.USE_FINGERPRINT", "permission.PROVIDER_INSERT_BADGE", "permission.WAKE_LOCK", "permission.READ_SETTINGS", "permission.READ_PHONE_STATE", "permission.ACCESS_COARSE_LOCATION"], "recievers": ["AppMeasurementReceiver", "AutoMessageHeardReceiver", "NotificationsDisabledReceiver", "AutoMessageReplyReceiver", "SmsReceiver", "CallReceiver", "MusicPlayerReceiver", "VolPMediaButtonReceiver", "AppStartReceiver", "RefererReceiver", "WearReplyReceiver", "StopLiveLocationReceiver", "PopupReplyReceiver", "NotificationCallbackReceiver", "ShareBroadcastReceiver", "CustomTabsCopyReceiver", "NotificationDismissReceiver", "VolPActionsReceiver", "ChatsWidgetProvider", "ContactsWidgetProvider", "FirebaseInstanceIdReceiver", "AlarmManagerSchedulerBroadcastReceiver"], "services": ["GcmPushListenerService", "AppMeasurementService", "AuthenticatorService", "ContactsSyncAdapterService", "KeepAliveJob", "BringAppForegroundService", "NotificationsService", "NotificationRepeat", "VideoEncodingService", "ImportingService", "LocationSharingService", "VolPService", "GoogleVoiceClientService", "MusicPlayerService", "MusicBrowserService", "TelegramConnectionService", "ChatsWidgetService", "ContactsWidgetService", "FilesMigrationService", "ChooserTargetServiceCompat", "FirebaseMessagingService", "ComponentDiscoveryService", "RevocationBoundService", "MlKitComponentDiscoveryService", "TransportBackendDiscovery", "JobInfoSchedulerService"]}\n'}

Fig (63): Extracted features in JSON

Chapter (8): Evaluation

Sample 2017

Sample Name	Sample No.	Benign	LRisk	MRisk	HRisk
Adware	100	5	19	39	37
Ransomware	84	2	27	40	15
SMSmalware	91	1	17	58	15
Scareware	107	10	26	40	31
Benign	68	44	21	3	-

Table(3):Evaluation 1

Sample 2021[33]

Sample Name	Sample No.	Benign	Malware
Adware	1500	285	1215
SMSware	4560	904	3656
Drebin	280	55	225
Bengin	222	182	40

Table(4):Evaluation 2

Chapter (9): Design Specifications

Functional Requirements:

Requirements	Requirements brief description
REQ-1	user can choose multiple apps to be scanned
REQ-2	user can scan installed or uninstalled app
REQ-3	the anti-malware app has 2 levels of risk if the app is malicious (low risk, Medium risk, High risk)
REQ-4	The app the user select to be scanned identified as benign or malicious based on 3 static features (permissions, services and features)
REQ-5	the user has two options to scan • Scan (features used are permissions, services and receivers) • Deep scan (API calls)
REQ-6	User can display all the installed apps in his mobile and choose one or more app to be scanned
REQ-7	A detailed report of the results of scanning is provided to the user in a organized manner
REQ-8	high accurate scanning results

Table(5): Functional Requirements

Non Functional Requirements

Requirements	Requirements brief description
REQ-1	App can adapt to detect different type of malwares
REQ-3	App help users to protect their mobiles from malwares that may cause a high risk to the user data
REQ-4	The app is safe, it doesn't take any sensitive data from the user
REQ-5	Time taken to deliver server response is so small in case of normal scan.
REQ-6	The app made for android phones
REQ-7	The scanning time is small in case of normal scan
REQ-8	Simple UI to guide user through the application

Table(6): Non-Functional Requirements

Functional requirements specification

1. Stakeholders

- **User**

Definition: A person who installs the app and uses it on his phone.

Interest: scan apps on his phone to make sure that the apps are not malicious if they are that would harm his phone and expose his files to the risk of being stolen or corrupted.

2. Actor and Goals

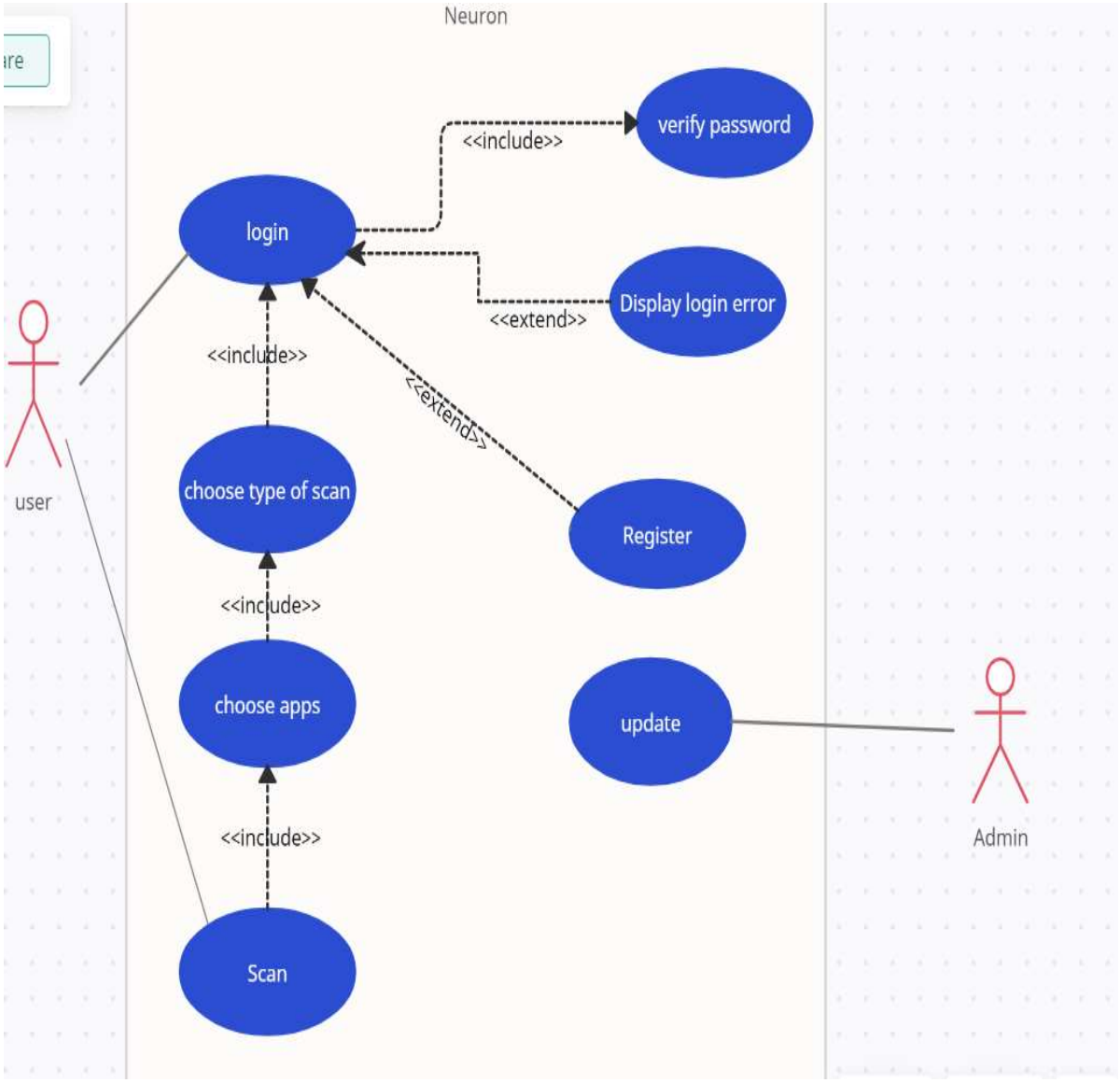
1- **User:** The main role of the user is to choose the apps he wanted to scan.
2- **Admin:** it can update the model in the server or update the dataset to be compatible with the continuous updates in malware.

3. Use cases

- **Use cases description:**

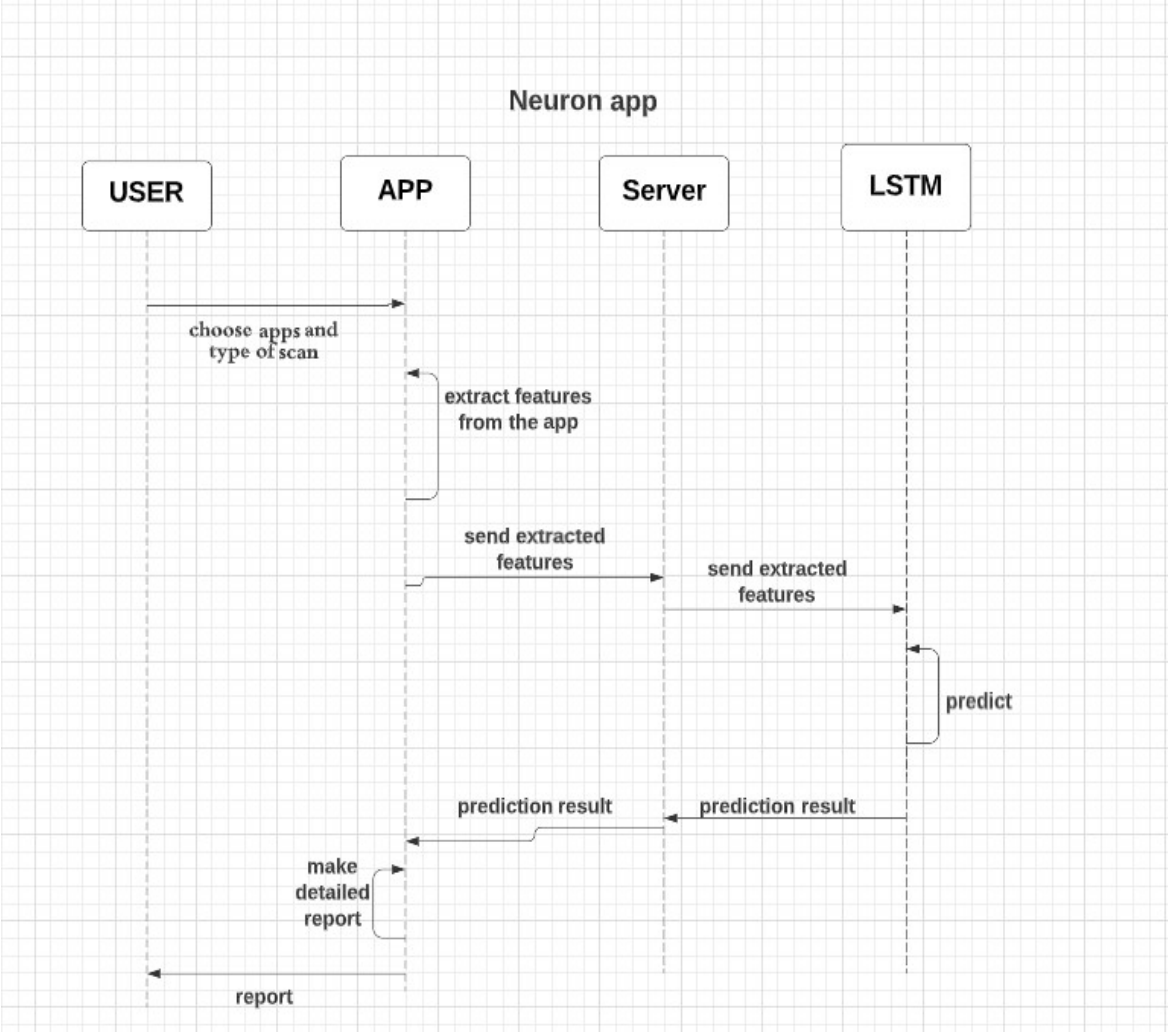
1- **Login:** The user logins with an email and password.
2- **Verify password:** The firebase verifies the password and email entered by the user.
3- **Display login Error:** When a user enters the wrong password or email an error display.
4- **Register:** User registers with name, email, password, and phone.
5- **Choose type of Scan:** The user must choose type of scan whether scan or deep scan.
6- **Choose apps:** The user can choose the apps he wanted to be scanned.
7- **Scan:** The user scan apps to know if these apps are benign or malicious.
8- **Update:** The Admin can update the model or the dataset in the server.

- Use case diagram



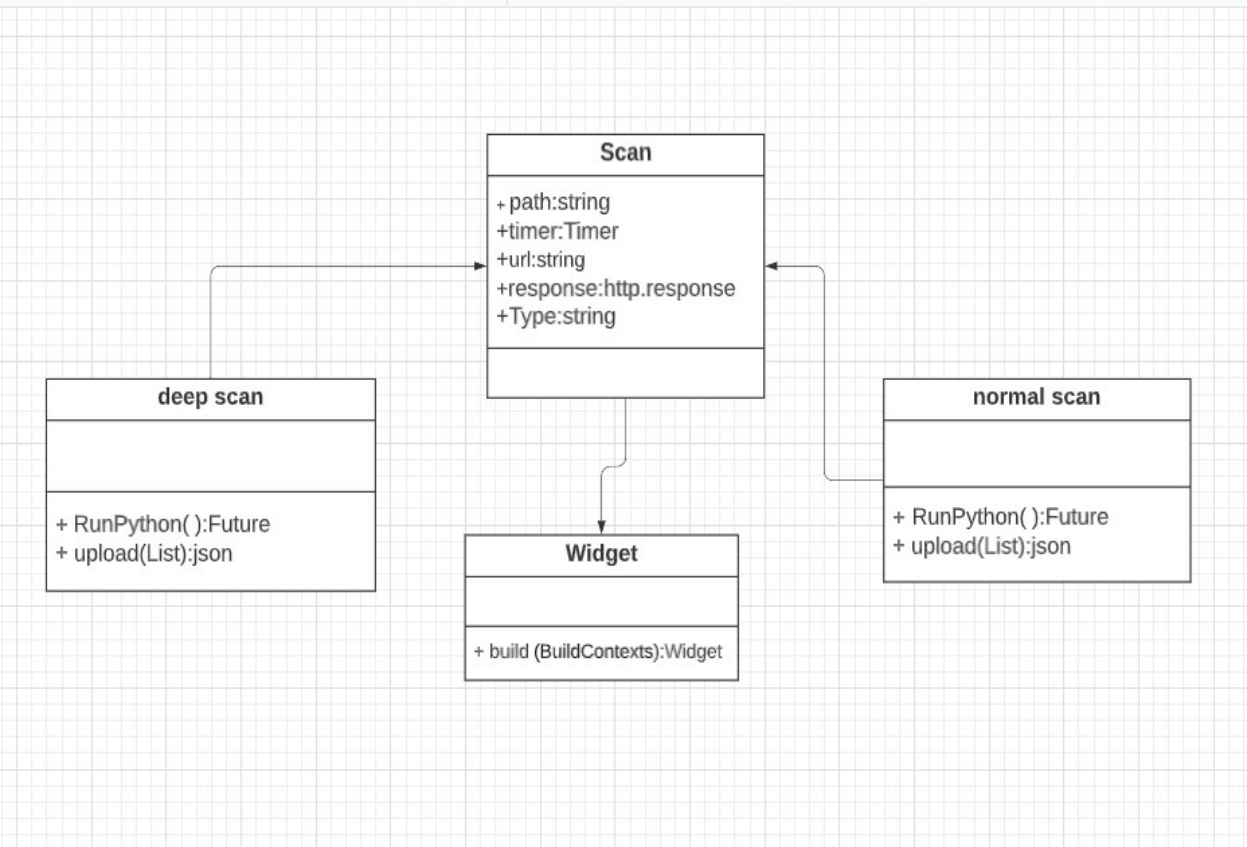
Fig(64): Use Case diagram

- Sequence Diagrams



Fig(65): Sequence diagram

- Class diagram



Fig(66): Class diagram

Tools and technologies:

Apk tool:
It is a tool used for decoding android applications.

dex2jar:
it is converts the file classes.dex of an application to classes.jar files.

jd gui:
it is a standalone graphical utility that displays java source code of .class files.

Spider IDE:
It is a free and open-source scientific environment written in *Python*, for *Python*, and designed by and for scientists, engineers, and data analysts.

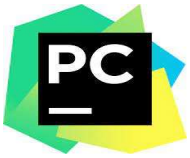
Android Studio:
It is the official IDE for Google's Android operating system, built on JetBrains' IntelliJ IDEA software and designed specifically for Android development.

Flutter:
That if your app is being run on iOS. Then. They should look like this making your app look like it was created in a native way. This is how flutter works you get access to a whole bunch of widgets

Jupiter:
It's flexible interface allows users to configure and arrange workflows in data science, scientific computing, computational journalism, and machine learning.

Flask :
it is a web framework. This means flask provides you with tools, libraries and technologies that allow you to build a web application. This web application can be some web pages, a blog, awiki or go as big as a web-based calendar application or a commercial website.

Languages:
Python (Machine Learning)
Java (User Interface) [34]



Chapter (10)

Conclusion

Hence, we have successfully proposed to use permissions, receivers and services of Android applications to detect malware and malicious codes in Android based mobile platform. Ours is a novel approach to distinguish and detect Android malware with different intentions. It is effective, that is, it can distinguish variant of Android malware between distinct purposes of them. The proposed framework extracts permissions from Android applications and further combines the API calls to characterize each application as a high dimension feature vector. By applying learning methods to the collected datasets, we can derive classification models to classify Apps as benign or malware. Experiments on real world data demonstrate the good performance of the framework for malware detection.

We briefly introduced the background to Android malware and gave a comprehensive review of machine learning-based approaches for detecting Android malware, arranged roughly in the order of the machine learning development pipeline. A range of alternative approaches were considered at each stage with a detailed consideration of their advantages in specific contexts. Some key content was summarized in the form of diagrams and tables to provide a better understanding and facilitate comparative analysis.

Finally, we isolated five topics to be studied in future research: the establishment of the sample set, data optimization and processing, feature extraction and establishment, application of machine learning, and classifier evaluation.

This work is different from previous surveys on Android malware detection, focusing on more aspects of machine learning methods. We believe this work complements previous reviews by filling some research gaps and putting forward some open issues in this field. We hope this review will provide a foundation for interested readers and inspire them to pursue new research avenues.

Future work

Since the Deep scan analysis takes a while, we can optimize it by reframing the algorithm and increase its efficiency and reduce time. Also, we can include an advanced analysis using an Android Sandbox environment.

This would increase the scanning ability of Neuron as well as it would test the apps on a real time basis. In real time, some apps behave differently than they are actually supposed to; hence a Sandbox would catch such apps and provide an added layer of assurity, accuracy and protection.

Since the server is local, premium flask quote is needed then the application can be at google play then app store.

References

[1] A Review of Android Malware Detection Approaches based on Machine Learning. https://ieeexplore.ieee.org/document/9130686

[2] DefenseDroid: A modern approach to Android Malware Detection. 27-May-3401.pdf - Google Drive

[3] https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/

[4] McAfee Mobile Threat Report Q1. 2020. Available online: https://www.mcafee.com/en-us/consumer-support/2020-mobilethreat-report.html (accessed on 2 December 2021).

[5] Yerima, S.Y.; Khan, S. Longitudinal Performance Analysis of Machine Learning based Android Malware Detectors. In Proceedings of the 2019 International Conference on Cyber Security and Protection of Digital Services (Cyber Security), Oxford, UK, 3–4 June 2019.

[6] Grill, B.B.; Ruthven, M.; Zhao, X. "Detecting and Eliminating Chamois, a Fraud Botnet on Android" Android Developers Blog. March 2017. Available online: https://android-developers.googleblog.com/2017/03/detecting-and-eliminating-chamois-fraud. html (accessed on 12 December 2021).

[7] Haystack. Mobile Issues. Available online: https://safeguarde.com/mobile-apps-stealing-your-information. (accessed on 14 January 2022).

[8] AV-TEST. Security Institute. Available online: https://www.av-test.org/en/statistics/malware/ (accessed on 14 January 2022).

[9] Goeschel, K. Reducing False Positives In Intrusion Detection Systems Using Data-Mining Techniques Utilizing Support Vector Machines, Decision Trees, And Naive Bayes for Off-Line Analysis. In Proceedings of the SoutheastCon 2016, Norfolk, VA, USA, 30 March–3 April 2016; pp. 1–6.

[10] Mehedi, S.T.; Anwar, A.; Rahman, Z.; Ahmed, K. Deep Transfer Learning Based Intrusion Detection System for Electric Vehicular Networks. *Sensors* 2021, *21*, 4736.

[11] DefenseDroid: A modern approach to Android Malware Detection. https://drive.google.com/file/d/1Rwbud-M8VNHi8ptna0UX0XVILe0V8oql/view

[12] https://securelist.com/mobile-malware-evolution-2021/105876/

[13] Felt, A.P., S. Egelman, M. Finifter, D. Akhawe, and D. Wagner: How to ask for permission. In: Proceedings of USENIX Workshop on Hot Topics in Security. USENIX Association, Berkeley, CA 2012, pp. 7-7.

[14] Kelley, P.G., S. Consolvo, L.F. Cranor, J. Jung, N. Sadeh, and D. Wetherall: A conundrum of permissions: Installing applications on an android smartphone. In: Proceedings of Fi- nancial Cryptography and Data Security. Springer, Berlin 2012, pp. 68-79.

[15] Tampering and Reverse Engineering on Android.https://mobile security.gitbook.io/mobile-security-testing-guide/android-testing-guide/0x05c-reverse-engineering-and-tampering

[16] Reverse Engineering an Android Application. https://cybersync.org/blogs-en/reverse_engineering_an_android_application.

[17] Shabtai, A., Kanonov, U., Elovici, Y. et al. "Andromaly": a behavioral malware detection framework for android devices. J Intell Inf Syst 38, 161–190 (2012). https://doi.org/10.1007/s10844-010-0148-x

[18] Justin Sahs and Latifur Khan. 2012. A Machine Learning Approach to Android Malware Detection. In Proceedings of the 2012 European Intelligence and Security Informatics Conference (EISIC'12). IEEE Computer Society, USA, 141–147. DOI: https://doi.org/10.1109/EISIC.2012.34

[19] Zhao M., Ge F., Zhang T., Yuan Z. (2011) AntiMalDroid: An Efficient SVM-Based Malware Detection Framework for Android. In: Liu C, Chang J, Yang A (eds) Information Computing and Applications. ICICA2011. Communications in Computer and Information Science, vol 243. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-27503-622

[20] Seung-Hyun Seo, Aditi Gupta, Asmaa Mohamed Sallam, Elisa Bertino, Kangbin Yim, Detecting mobile malware threats to homeland security through static analysis, Journal of Network and Computer Applications, Volume 38, 2014, Pages 43-53, ISSN 1084-8045,https://doi.org/10.1016/j.jnca.2013.05.008

[21] Arp, Daniel & Spreitzenbarth, Michael & Hubner, Malte & Gascon, Hugo & Rieck, Konrad.(2014). DREBIN: Effective and Explainable Detection of Android

Malware in Your Pocket. Symposium on Network and Distributed System Security (NDSS). 10.14722/ndss.2014.23247

[22] D. Wu, C. Mao, T. Wei, H. Lee and K. Wu," DroidMat: Android Malware Detection through Manifest and API Calls Tracing," 2012 Seventh Asia Joint Conference on Information Security, Tokyo, 2012, pp. 62-69, DOI: 10.1109/AsiaJCIS.2012.18. https://ieeexplore.ieee.org/document/6298136/

[23] William Enck, Machigar Ongtang, and Patrick McDaniel. 2009. on lightweight mobile phone application certification. In Proceedings of the 16th ACM conference on Computer and communications security CCS'09. Association for Computing Machinery, New York, NY, USA, 235–245. DOI: https://doi.org/10.1145/1653662.1653691. https://doi.org/10.37896/sr8.5/027281 ISSN: 0039-2049 http://stradresearch.org/ Strad Research VOLUME 8, ISSUE 5, 2021

[24] Sanz, Borja & Santos, Igor & Laorden, Carlos & Ugarte Pedrero, Xabier & Bringas, Pablo. (2012). On the Automatic Categorization of Android Applications. 10.1109/CCNC.2012.6181075. https://ieeexplore.ieee.org/document/6181075

[25] H.Wang, J.Si, H.Li and Y.Guo, "RmvDroid: Towards A Reliable Android Malware Dataset with App Metadata," 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR), Montreal, QC, Canada, 2019, pp.404-408, DOI: 10.1109/MSR.2019.00067. https://ieeexplore.ieee.org/document/8816783

[26] X. Li, J. Liu, Y. Huo, R. Zhang and Y. Yao, "An Android malware detection method based on Android Manifest file," 2016 4th International Conference on Cloud Computing and Intelligence Systems(CCIS), Beijing, 2016, pp.239-243, DOI: 10.1109/CCIS.2016.7790261. https://ieeexplore.ieee.org/document/7790261

[27] Mohammed K. Alzaylaee ,Suleiman Y. Yerima, Sakir Sezer, DL-Droid: Deep Learning Based Android Malware Detection Using Real Devices, Computers & Security (2019), DOI:https://doi.org/10.1016/j.cose.2019.101663

[28] N.Peiravian and X.Zhu, "Machine Learning for Android Malware Detection Using Permission and API Calls, "2013 IEEE 25th International Conference on Tools with Artificial Intelligence, Herndon, VA, 2013, pp. 300-305, DOI:10.1109/ICTAI.2013.53. https://ieeexplore.ieee.org/document/6735264

[29] The Complete Android Oreo Developer Course - Build 23 Apps! Created by Rob Percival, Nick Walter. https://www.udemy.com/course/the-complete-android-oreo-developer-course/

[30] Androguard: A Full Python Tool to play with Android files. https://androguard.readthedocs.io/en/latest/

[31] Pyaxmlparser: Python3 Parser for Android XML file. https://pypi.org/project/pyaxmlparser/

[32] MalScan: Fast Market-Wide Mobile Malware Scanning by Social-Network Centrality Analysis. Yueming Wu, XiaoDi Li, Deqing Zou, Wei Yang, Xin Zhang, Hai J http://youngwei.com/pdf/MalScan.pdf

[33] Dataset to testing Apps including Apk. http://205.174.165.80/CICDataset/MalDroid-2020/Dataset/APKs/

[34] A static analysis approach for Android permission-based malware detection systems Juliza Mohamad ArifID, Mohd Faizal Ab Razak*, Suryanti Awang, Sharfah Ratibah Tuan Mat, Nor Syahidatul Nadiah Ismail, Ahmad Firdaus Faculty of Computing, Universities Malaysia Pahang, Pekan, Pahang, Malaysia. A static analysis approach for Android permission-based malware detection systems - PubMed (nih.gov).