DROIDANALYZER: EFFICIENT FRAMEWORK FOR ANDROID MALWARE DETECTION

MAJOR PROJECT SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF DEGREE OF

Master of Technology

In

Information Systems

Submitted By:

PARVESH

MAMGAIN

(2K17/ISY/18)

Under the Guidance

Of

Mr Jasraj Meena

(Assistant Prof., Department of IT)

DEPARTMENT OF INFORMATION TECHNOLOGY

DELHI TECHNOLOGICAL UNIVERSITY

(2017-2019)

CANDIDATE'S DECLARATION

I Parvesh Mamgain, Roll No. 2K17/ISY/18 student of M.Tech Information Systems, hereby declare that the project Dissertation titled "DROIDANALYZER: EFFICIENT FRAMEWORK FOR ANDROID MALWARE DETECTION" which is submitted by me to the Department of Information Technology, Delhi Technological University, Delhi in partial fulfilment of the requirement for the award of the degree of Master of Technology, is original and not copied from any source without proper citation. This work has not previously formed the basis for the award of any degree, Diploma Associateship, Fellowship or other similar title or recognition

Place: Delhi Parvesh Mamgain

Date:

CERTIFICATE

I hereby certify that the Project Dissertation "DROIDANALYZER: EFFICIENT FRAMEWORK FOR ANDROID MALWARE DETECTION" which is submitted by Parvesh Mamgain, Roll No 2K17/ISY/18 under Department of Information Technology, Delhi Technological University, Delhi in partial fulfilment of the requirement for the award of the degree of Master of Technology, is a record of the project work carried out by the student under my supervision. To the best of my knowledge this work has not been submitted in part or full for any Degree or Diploma to this University or elsewhere.

Place: Delhi **Mr. Jasraj Meena**

Date: **SUPERVISOR**

ACKNOWLEDGEMENT

I take the opportunity to express my sincere gratitude to my project mentor Mr Jasraj Meena, Assistant Prof., Department of Information Technology, Delhi Technological University, Delhi, for providing valuable guidance and constant encouragement throughout the project. It is my pleasure to record my sincere thanks to him for his constructive criticism and insight without which the project would not have shaped as it has.

It humbly extends my words of gratitude to other faculty members of this department for providing their valuable help and time whenever it was required.

Parvesh Mamgain

Roll No. 2k17/ISY/18

M.Tech (Information Systems)

E-mail: mamgainparvesh@gmail.com

ABSTRACT

Mobile Application Market has evolved and is continuously expanding with over millions of applications. Many mobile operating systems are available, most popular among them is Android. Due to its popularity and reach, malware developers are targeting android markets for distributing malware. This has led to an increase in risk associated with Android devices. The growth of mobile malware is so huge that traditional techniques for malware detection are inefficient. Therefore, effective and robust malware detection techniques are required. Many researchers have proposed static and dynamic approaches for effective Android malware detection. In this research, we have proposed a fine-grained hybrid model for efficient android malware detection using multi-modal learning. We have extracted static and dynamic features from a set of 4000 applications. We have used multi-modal learning to better classify the samples. We have compared our implementation with other techniques. Our analysis suggest that multi-modal learning outperforms other state of the art techniques.

Table of Contents

List of Tables

List of Figures

Chapter 1

INTRODUCTION

Android is a popular mobile platform among other platforms like iOS and windows. As of now, the majority of devices run on Android [1]. Users mostly perform business transactions, Important task, etc. usually through their mobile devices. Therefore, Smartphones have become a central point for sensitive information. Due to this, malware developers are targeting Android devices. Malware refers to a variety of unwanted software that is used for performing malicious activities. Few commonly used malware are viruses, spyware, Trojan horses, rootkits, backdoors, etc. There are various ways by which malware gets distributed to the end-user like downloading and installing repacked apps, downloading the apps from the malicious website, third-party app store, play store, etc. Android allows installing apps from other third-party app stores which further increase the risk of being infected by malware as most of these sources are unverified. Google Play is the official service provided by Google for downloading apps. However, it is found that it is also not secure. There is an enormous growth in malware from 2014 to 2019 (up to quarter 1) as shown in Fig. 1.1 [2].

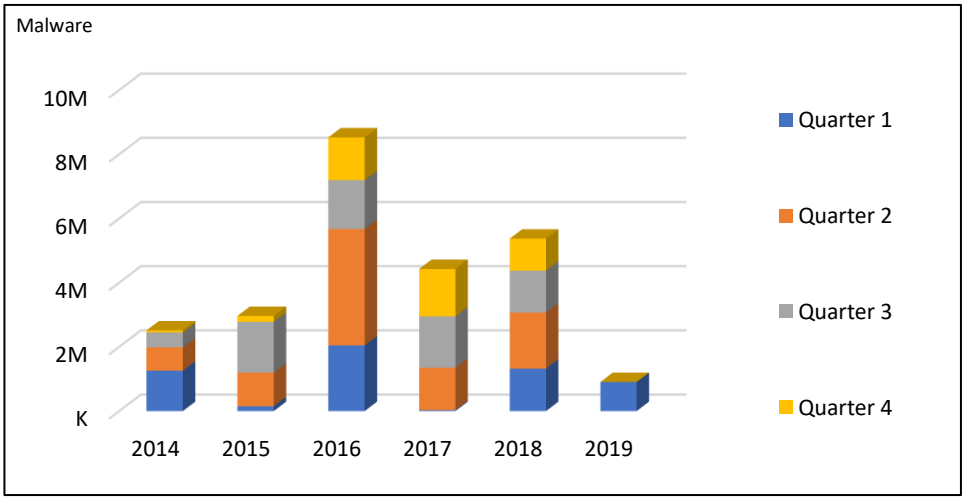

Fig. 1.1: Mobile Malicious Installation packages detected by Kaspersky Lab [2]

1.1 MOTIVATION

Android is a rich target environment for malware because of Larger Attack Surface, Open Source and loose security in Google play store. These shortcomings have led to an increase in the growth of Android malware. According to Quick Heal annual threat report 2019, there is a huge increase in Android samples detection count from 2016 to 2019 [3]. Reports suggest there is an increase in profit-driven malware like ransomware, adware, potentially unwanted programs (PUP), premium SMS, etc. The report also suggests that malware infection rate is not uniform across the world rather it varies on the basis of geological areas like mobile malware infections in Iran, Bangladesh, China, India, and Nepal are 35.12%, 28.3%, 27.38%, 21.91%, and 20.78% respectively. This data also suggests that there exist some countries that may become a victim of malware attack due to poor security infrastructure.

Therefore, the malware is increasing with so rapid growth that the need for an effective and efficient framework for malware detection is required. As a result, many heuristic-based states of the art techniques are used for learning classifiers for effective android malware detection which can cope up with the current growth of malware.

1.2 OBJECTIVE OF MASTERS THESIS

In this thesis, we have proposed an efficient technique for effective android malware detection using a multi-modal deep neural network. We have compared our technique with other approaches [4], [5]. We have also extended these approaches for dynamic and hybrid analysis. We have performed the analysis by extracting static and dynamic features from a large dataset of 4000 applications. We have reduced the features by using Information Gain Attribute selection algorithm. The analysis has been carried out on a large dataset of 4000 applications with 2000 benign and 2000 malware applications. The benign applications are collected from AndroZoo Repository [6] and Malware applications are collected from VirusShare repository [7]. Our result shows that multi-modal neural network is an effective alternative and can be used for android malware detection. Our analysis also suggests that machine learning models yield a better result when features are provided in the form of different modalities separately rather than combining all the features into one long feature vector.

1.3 ORGANIZATION OF DISSERTATION

The dissertation is structured as follows: -

- Chapter two give you a brief overview of the topic. It discusses malware, its types and different propagation techniques used by malware. It also discusses the android framework and literature review.
- Chapter three discusses general framework for android malware detection.
- Chapter four presents the proposed work.
- Chapter fifth contains the results of our approaches on a dataset.
- Chapter sixth thesis is directed towards a conclusion and further ideas for future work have been proposed.

Chapter 2

BACKGROUND

Android Malware and its detection techniques are the key areas of focus for this section. This chapter is organized as follows: -

In section 2.1, we have discussed an overview of Malware and its types.

In section 2.2, we have given a general brief of the android architecture.

In section 2.3, We have discussed techniques used for malware detection.

In section 2.4, We have discussed Literature overview.

2.1 ANDROID MALWARE

Malware is the malicious application that is used by hackers for committing unethical activities like fraud, information leak, Stealing user private data, Identity Theft, etc. Malware applications are used by hackers to steal user's private information or to obtain unauthorized access to a mobile device. Android has witnessed a lot of malware that has affected millions of devices worldwide. In 2018, A new type of malware was detected by Trend Micro which was developed using Kotlin (It is a new Programming language, officially confirmed by Google, specifically designed for android application development) [8]. It can infect mobile devices with ads and secretly subscribe them to premium SMS numbers. Other features of this malware include remote code execution, SMS sending, URL forwarding, etc. Hummingbird is another Android malware discovered by Checkpoint in 2016. This malware is responsible for installing numerous fraudulent apps each day, displays millions of malicious advertisements, and generates a lot of revenue from these ads and apps. Most of the malware is developed with some intent of profit or gain. However, Malware development may or may not be profit-driven. Initially, the malware was developed to showcase the proficiency of one's technical ability. However, these days trends have changed, the number of profit-driven malware has increased enormously. Antivirus vendors have implemented many frameworks for detecting Malware applications. However, Malware

often uses different techniques for evading anti-malware scanners.

- **Encryption** – In this technique, the application is encrypted with a key so that static scanners can't parse it directly. The application is decrypted only during execution at runtime. Thus evading all the static scanners.

- **Obfuscation** – In this technique, malware behavior is hidden by using garbage commands and jumps. Dead or irrelevant codes are used for evading the malware detection. A condition is set on the malware like execute malware module during night time only.

- **Dynamic Loading** – This technique can be used to deter static detection techniques. In this technique, modules are loaded dynamically at runtime. However, this can be detected by using a behavior-based detection technique.

Security firms are witnessing a drastic change in malware growth. Various security reports from leading anti-malware products suggest the following trends and predictions –

- Development of more sophisticated ransomware.
- Distribution of malware through exploits.
- Security concern with IoT Industry.
- Android Vulnerabilities.
- Machine learning Based malware.
- The growth of Potentially unwanted programs.
- Growth in the payment system and banking malware.

Quick Heal Report 2019 suggest that adware and premium SMS service based infections have dominated the year 2018 as shown in the fig. 2.1 [3]. The report indicates that the growth of Banking Trojans and ransomware have increased. The report also mentions the security vulnerability of the Android operating system like Code execution flaws, Information gain attacks, etc.. According to Quick heal threat report 2019 trends and predictions, android vulnerability and potentially unwanted programs are a major concern in the coming days. It also suggests that PUA's

are increasing rapidly.

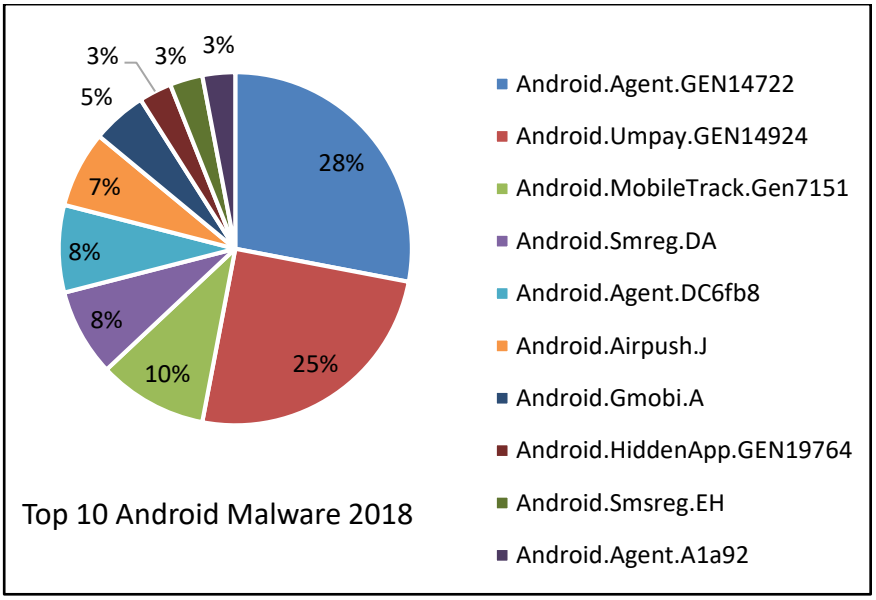

Fig. 2.1: Top 10 Android Malware 2018 [3]

2.1.1 MALWARE CATEGORIES

On the basis of the behavior of the malware, most common mobile malware types are – Expander, Worm, Trojan, and Spyware.

1. **Expander**

   It is a type of mobile malware that targets mobile meter for additional billing by subscribing the user to premium services. Most types of malware that fall under this category targets services for profit. Ransomware is a popular malware which installs covertly on a victim's computer. It executes an encryption attack that encrypts the system with a key. The computer is locked down until the decrypting key is used for decrypting the system back to its original state. Once the victim is infected by this malware, he is demanded to pay a ransom to the attackers to decrypt his system. WannaCry is popular ransomware that has affected thousands of computers by encrypting their data and demanding ransom payments in the form of Bitcoin cryptocurrency in exchange for decrypting their data back [9].

2. Worms

The worm is the type of malware that contains harmful instructions that do not require user interaction for executing its malicious behavior. It uses a computer network as a medium to spread itself across millions of devices. It relies on security failures of the target device to access its resources. They always cause at least some damage to the network, even if only by consuming bandwidth. The Morris worm (released in 1988) was the first publicly known worm. It is a pay-load free worm. A Payload consists of the code that is designed to do some malicious behavior rather than spreading the worm. The cost of removing the Morris worm was in millions. It is reported that the Morris worm has affected a lot of UNIX machines. Samsapo is another android worm that spread itself by sending an SMS message to all the contacts that are listed in the device [10]. The message contains a malicious link to a malicious APK package. Abilities of this worm include multiple malicious routines, gathering and sending information from the mobile device to a remote server, registering the phone number to a premium service, etc. Mitigation techniques that are used as a countermeasure to reduce the effect of the worm are Packet Filtering, Security Patches, Null route (A network route that goes nowhere), etc.

3. Trojans

A trojan is the type of mobile malware that disguises itself as a normal application, However, executes malicious behavior in the background. They are the executables that required user interaction, once activated they can cause serious damage to the victim device. They behave to be legit but performs malicious action in the background. Trojans are usually spread through social engineering techniques. Zeus is a Trojan horse malware package that can be used to carry out many malicious tasks. Few features of this malware include stealing financial information, keylogging, form grabbing, etc. It is widely spread by downloads and phishing schemes. Detection of Zeus malware is somewhat difficult as it uses advanced stealth techniques to avoid detection. In 2017, a new mobile Trojan was found by Malwarebytes called Android/Trojan.AsiaHitGroup which masquerades as multiple apps which we use in our day to day life like an alarm clock, QR scanner, compass, photo editor, Internet speed test, and file explorer. In this malware, malicious payloads are distributed over these apps [11].

4. **Spyware**

   It is used for monitoring users personal and sensitive activities without the user's consent. Its main objective is to gather information about a person or organization without their knowledge. Malware like Premium SMS, RATs, Financial Fraud, Botnets, and Cryptolocker are mostly used by hackers for gaining profit. GO Keyboard virtual Android keyboard android app is a malware application that transmits user's personal information without user's consent [12]. This information includes the user's account details, location, network details, Android version and build, and device's model and screen size. This malware supports remote code execution attack. This app has affected millions of users worldwide. It was detected in 2017 by security researchers from AdGuard.

2.1.2 MALWARE PROPAGATION

In this section, we will briefly discuss what are the possible ways through which malware attack the victim device. We have discussed general approaches that are widely used for attacking the victim device to gain unauthorized access or steal user information [13]. The attacking environment can be summarized in figure 2.2. The hacker or attacker tries to distribute the malware by sending a malicious link via SMS, Exploiting an existing vulnerability in the Android architecture or through malware-hosting websites. The malware once installed into the system executes its malicious behavior and gains unauthorized access to various system resources. Some possible attacks after infecting the device with malware are accessing device resources, Monitoring user activities, sending SMS to premium numbers, stealing user's personal information, etc. The malware is usually hidden from the user interface and hence it is utmost important to install a good anti-malware software on your device in order to protect yourself from harmful malware. The malware often uses stealth technology to avoid detection by anti-malware. If malware is zero-day malware then this detection technique cannot detect this type of malware. Therefore, heuristic-based detection is widely used for detecting such kind of malware which are zero-day.

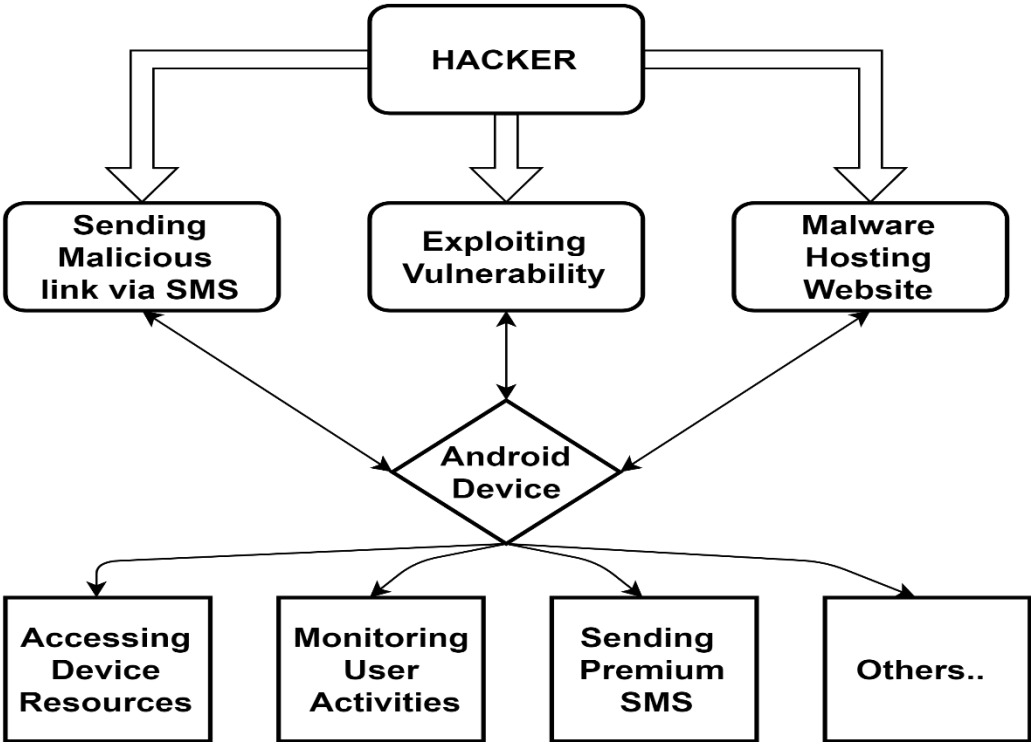

Fig. 2.2: Malware Attacking Environment.

2.2 OVERVIEW OF ANDROID STACK

Android is a popular mobile platform. It has a large market share of 86.1% as of 2017. Android architecture can be described as a stack of components categorized into five layers as shown in Fig. 2.3. The top layer consists of core applications that are included in the Android platform along with other apps. Java API framework acts as an interface between systems apps and native library and android runtime. It provides a framework for developing an android application. Native libraries consist of core features and services that are written in C and C++. Android runtime provides a runtime environment for executing dex files. HAL consists of several interfaces that are used for interacting with the corresponding hardware component. The Linux kernel forms the core of the Android platform which manages core functionalities. All applications execute separately within their own Dalvik Virtual Machine instance in ART (Android Runtime). However, After Android 5.0, this was changed to Ahead of Time compiler as opposed to dalvik just in time compiler (JIT). User-defined applications are executed on the top layer of Android architecture. They interact with each other through intents. The process is known as Interprocess communication. Intents can be implicit or explicit. They are used by

various Android components for interacting with one another. Android application is packed into a special compressed file known as APK File.

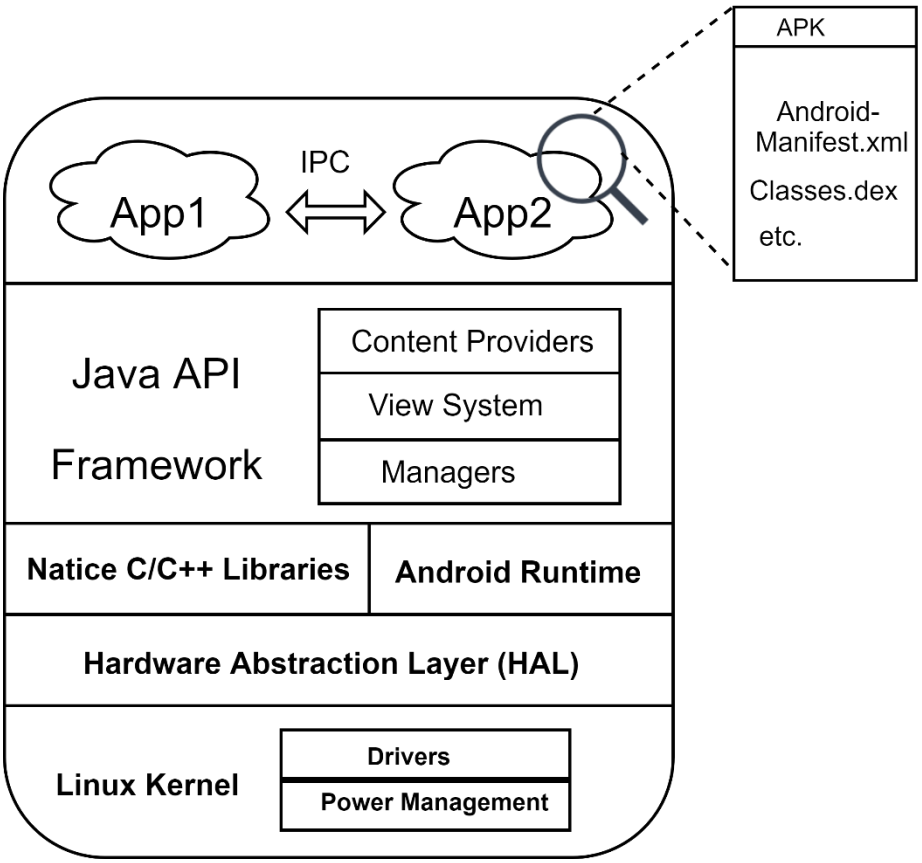

Fig. 2.3: Android Stack.

APK is the standard file format for the Android application. Android Package Kit (APK) is a type of archive file with a *.**apk** extension. The content of the APK file is described below.

- **META-INF** – This folder contains the metadata about the java package. It contains the manifest file, list of files along with SHA-1 digest and certificate.
- **ASSETS** – This folder contains the files or resources that can be used by the application. AssetManager class provides access to these files or resources.
- **AndroidManifest.xml** – It is an important XML file that describes the name of the packages, permissions, version, referenced library and application components that are used in the application.

- **Classes.dex –** Android apps are written in Java which upon compilation produces bytecode which is further converted into a dalvik executable file which is interpreted by dalvik virtual machine. After lollipop update, this is changed to ART (Android Runtime) which uses AOT Compiler instead of JIT Compiler as in Dalvik Virtual machine.
- **Lib –** This directory contains processor-specific compiled code.
- **Res –** It holds the raw files that are not compiled such as drawable, media, etc.

Among them, Classes.dex and androidmanifest.xml is widely used for extracting malware behavior. We have also used these two files for feature extraction. Following features can be extracted from these files –

- **Permissions** - In Android, access to a resource is granted by Android Permission System if the permission is defined in the manifest file.
- **Activities –** It defines the UI and handles the user interaction.
- **Services** – It is used for background processing within an app.
- **Broadcast Receivers** - It handles communication between OS and app.
- **Content Providers** – It provides data related solutions in the application.
- **Intents** - It is a data structure that holds the information of the operation to be performed. There are two types of intents –
    - **Explicit Intents** - It is used to connect the internal application. Suppose if you want to connect one activity to another activity, then explicit intents are used.
    - **Implicit Intents** – In this, the target application is not defined. Implicit intents are often used to activate components in other applications.
- **API** - A set of methods that helps in the development of applications by allowing access to the resources, apps, or services.

In Android, access to a resource is granted by Android Permission System if the corresponding permission is listed in the manifest file. Android permission system acts as a primary defense against malware applications by preventing an application from gaining access to an unauthorized resource. Permissions can be further categorized into the following categories based on the risk associated with them.

1. **Normal**: A low-risk permission that is automatically granted to an application on request. They provide access to features having minimal risk.
2. **Dangerous**: A high-risk permission that can grant access to user private data. It is granted by the user during installation.
3. **Signature**: This permission is only granted by the system if and only if the application that is requesting for the permission is signed.
4. **Signatureorsystem**: This permission is granted by the system only to those apps that either belongs to Android system image or that are signed.

2.3 ANDROID MALWARE DETECTION TECHNIQUES

In this section, we will briefly discuss different malware detection techniques and issues associated with these techniques [13]. Malware detection is the process of analyzing the application with the intent of determining whether the application is benign or malware. These are some issues which one should keep in mind while designing a malware detector. Mobile devices have limited resources such as battery life and therefore on-device analysis is somewhat difficult. Also, Android Permission System prevents antivirus applications to introspect other application, which further limit the extent of the analysis. Therefore, most of the detection techniques are cloud-based. Considering these issues, the following detection techniques are used –

1. **Signature Based Detection**

   Traditional techniques for malware detection use signatures for detecting malware applications. In this technique, A signature is computed for an application by extracting binary patterns or snippets from the code such that the signature is unique to the application. The signature is then matched with the dataset consisting of malware signatures. If a match occurs then the application is malware else benign. To further improve the detection accuracy, cloud-based detection methods were developed in which dataset is maintained at the server end and a signature is matched with the server-side dataset. The central repository is regularly updated so that the accuracy of detection can be improved. This technique is highly inefficient while considering the current pace of malware development. Limitations

of signature-based detection technique are summarized below –

- Ineffective for detecting the zero-day vulnerability.
- Obfuscation attacks go undetected.
- It cannot detect encrypted malware.

2. Static Analysis Based Detection

In this technique, the application is examined without executing the code [4], [5], [14]–[16]. AndroidManifest.xml and Classes.dex file is widely used for feature extraction. The manifest file is decompiled and then features are extracted by developing a custom XML parser. Classes.dex is a compiled binary which needs to be decompiled before using it. Various tools are available for decompiling dex files like APK Tool [17] and Androguard [18]. The following kinds of features can be extracted during the static analysis phase.

- **Permissions –** Android required applications to request permissions before accessing some resources or features. Permissions are further categorized into normal permissions, system permissions, signature permissions and dangerous permissions.
- **Intents –** These are the objects that contain some information about an operation that needs to be accomplished by an application component. Intents can be explicit or implicit.
- **Hardware Details –** listed hardware components in the manifest.
- **Dex File –** It consists of classes, APIs, methods, structure sequences, program dependency graph, inter process communication etc.

Static analysis-based detection is fast and yields a better result with good accuracy. However, it has the following limitations-

- Vulnerable to obfuscation, reflection, and encryption.
- It cannot analyze dynamically loaded code

Some of the static analysis tools that are widely used for analyzing applications are –

- **Smali and Baksmali** - Smali/baksmali is a reverse engineering tool that is used for converting Apk binaries to a human-readable format.
- **Androguard –** Androguard is a python tool that is widely used for analyzing APK's. Its features are Disassembler, De-compiler, support dex, and odex file format, etc. [18].
- **ApkTool –** It is a reverse engineering tool for Android. It is widely used for decompiling APK into the human-readable format. [17].

3. Dynamic Analysis Based Detection

In dynamic analysis technique, Application is examined at runtime by executing the application [19]–[22]. The runtime environment can be a virtual machine, Sandbox or a real mobile device. This technique is not vulnerable to obfuscation and reflection attacks. However, malware is becoming more sophisticated and can detect an emulated environment and choose not to exhibit malicious behavior. Few limitations of this technique are -

- Limited coverage as the user has to interact with the application to execute different modules. However, stimulation techniques can be used to improve coverage
- Executing a lot of paths is a costly process.
- No guarantee that all execution paths are executed
- Smartphones accept a wide set of touch commands which further complicates the analysis.
- Difficult to detect VM-Aware malware.

Some of the dynamic analysis tools that are widely used for analyzing applications are –

- **DroidBox –** DroidBox is used for dynamic analysis of Android applications. It provides a detailed analysis of Android applications. Few parameters detected by

DroidBox includes network streams, File streams, services, Information leaks via the network, Android APIs, and SMS and phone call logs [23].

- **CuckooDroid –** CuckooDroid is also used for behavior analysis of Android applications. It supports automatic analysis of applications. It is open-source software.

4. **Hybrid Analysis Based Detection**

   Static analysis is fast but is vulnerable to obfuscation and reflection attacks. Dynamic analysis is robust to these attacks but is time-consuming. Therefore, a combination of these two approaches is used for detecting malware [24]–[26]. Hybrid analysis has the advantage of both techniques. Combination of Virtualization technique and real devices can also be used for detecting advance sophisticated Virtual machine aware malware. Experiments show that the combination of static and dynamic analysis yields a better result.

2.4 LITERATURE REVIEW

Traditional malware detection methods are signature-based. A unique signature is generated for each malware found by extracting binary patterns or snippets from code. Its drawbacks are Ineffective to detect a zero-day error, Distribution of newly generated malware is a costly process and obfuscation attacks can bypass it. This leads to the development of heuristic-based detection approaches. Machine learning-based detection methods provide reliable and robust malware detection by extracting static and dynamic features. Static analysis-based approaches are widely popular. Arp et al. proposed a static framework that uses permissions, activities, services, system events, etc. as features for classifying malware [5]. The research was based on support vector machine for model creation. Yuan et al. proposed a static framework using rotation forest model to better classify the dataset [4]. However, static approaches are inefficient to determine the obfuscated code. To detect obfuscated code, dynamic analysis approaches are used. Feng et al. implemented a dynamic framework based on ensemble learning techniques [20]. They have used system calls, network logs, API Calls, etc. as features to better classify malware. Many types of research are carried out by using a combination of both techniques. It was found that the hybrid approach yields the best result. Yuan et al. implemented a hybrid approach using both static and dynamic features and was able to achieve good accuracy [25]. Deep learning is a relatively new

field that is becoming popular now a days. Many researchers have used DNN to solve the problem of malware detection and achieved good results [27]–[29]. In comparison, our work is motivated by some of the above techniques and approaches, but with a focus on multi-modal learning using a deep neural network. We have used static and dynamic features extracted from a large dataset of 4000 applications, collected from AndroZoo [6] and VirusShare repository [7]. Our result shows that recent advances in neural network approaches can be used as a better classifier when compared with traditional approaches.

Chapter 3

RESEARCH METHODOLOGY

In this chapter, we have discussed the general framework that is widely used by machine learning-based approaches for effective malware detection. Most android analysis techniques follow this general framework with minor modification. The general framework is shown in fig. 3.1. The research methodology can be broadly classified into four major steps which are as follows –

1. Dataset Creation
2. Feature Extraction
3. Feature Selection
4. Model Creation

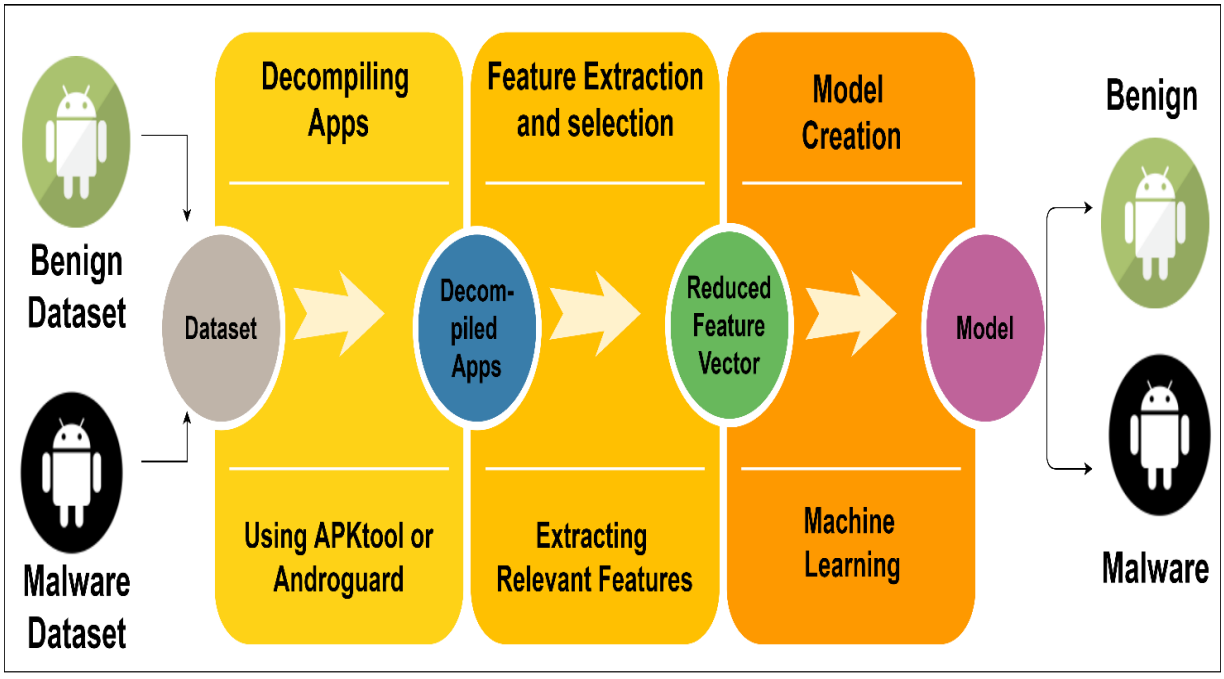

Fig. 3.1: General Framework for malware detection

3.1 DATASET CREATION

In the initial phase, malware dataset and benign dataset is created. Malware dataset can be constructed by collecting samples manually by accessing malicious sources or from online malware repositories like VirusShare [7]. We have used VirusShare repository for malware samples. The benign set can be constructed by collecting samples from the PlayStore or third-party app stores. We have used AndroZoo Repository for benign samples [6]. The Dataset is summarized in the below table: -

TABLE 3.1: DATASET

Data Source	QUANTITY
AndroZoo (Benign)	2000
VirusShare (Malware)	2000

3.2 FEATURE EXTRACTION

Once samples are collected, the feature extraction process is applied to the dataset which extracts the relevant features from the dataset and produces a binary representation or a graph representation of the features. We have used binary representation for generating a feature vector. Let us consider a one-dimensional feature vector *f*(x) where X is a set of finite features defined as below.

$$X = < X_1, X_2, X_3 \ldots\ldots\ldots X_n \mid X_i = feature > \quad (1)$$

Here a feature can be permission, a function call, system calls, hardware detail, System event or an Intent. These features are extracted by using static or dynamic analysis techniques. Once features are extracted, the feature vector is constructed. We can use frequency representation and binary representation for constructing a feature vector. We have used binary representation in our research. In binary representation, 1 denotes feature is present and 0 represent feature is not present as shown in equation (2). In frequency representation, n denotes no of times that particular feature is present in the application as shown in equation (3).

$$f(\mathrm{x}) = \begin{cases} 1 & feature\ present \\ 0 & otherwise \end{cases} \quad (2)$$

$$f(x) = \{ n_i : count\ of\ feature\ i \} \quad (3)$$

Features can be extracted statically or dynamically as discussed in the previous section. In static analysis, we analyze the source code of the APK while in dynamic analysis, we execute the APK and extract features by monitoring its behavior. A feature vector is generated for each application denoting the behavior of the application. We have stored feature vector of each APK into a common CSV file. The CSV file looks like the following as shown in fig. 3.2. Here, under the classification attribute, we have used 1 as malware and 0 as benign.

classificat	android.p	android.p	android.p	android.p	com.goog	com.goog	android.p	android.p
1	1	0	0	1	0	0	0	1
1	1	1	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0
1	1	1	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0

Fig. 3.2: Sample Feature Vector

3.2.1 Extracting Static Features

For extracting static features, we have used the Androguard framework for reverse engineering Android applications [18]. We have extracted API Calls, Permissions, System events (Components, Intents, Broadcast Receivers, and Services) and Opcodes from a large set of 4000 applications.

3.2.2 Extracting Dynamic Features

For extracting dynamic features, we have used DroidBox framework for behavior analysis of android applications [23]. We have extracted Data leaks, File Accesses, Enforced permissions, and crypto calls from a large set of 4000 applications.

In our research, we have extracted a total of 118333 features from the dataset each categorized into eight broad categories. The features are summarized into the below table: -

TABLE 3.2: FEATURE SET SUMMARY

CATEGORIES	NO. OF FEATURES EXTRACTED
S_1 - API Calls	30105
S_2 - Crypto Calls	51
S_3 - Data Leaks	3
S_4 - Enforced Permissions	1000
S_5 - System Events	63112
S_6 - Files	22392
S_7 - Opcode	226
S_8 - Permissions	1444

3.3 FEATURE SELECTION

Usually, the feature set is very large which results in a large number of computations and requires a lot of resources. Therefore, feature selection techniques like Information Gain are used to minimize the dimensionality of the vector. Feature selection technique ranks the features on the basis of their contribution to malware detection. Best features are selected and rest features are removed thus reducing the size of the feature vector. Reducing Feature vector size has various advantages which are as follows –

- Reduces Overfitting of data
- Reduces the time required to build the model.
- Reduce training time.
- Avoids the curse of dimensionality.

We have used Information Gain as Attribute Selection algorithm to reduce the feature vector dimensionality. We have reduced the size of the feature vector by considering top features based on their score. We have filtered the top features that have Info Gain score of greater than 0.1.

Let ***A*** be the set of all attributes and ***E*** set of all training examples. *V*(*x*, *a*) be the value of specific example *x* for attribute *a*. H defines Entropy. Values(a) represents a set of all values of a ϵ ***A.***

Information gain is defined by the following relation –

$$IG(E,a) = H(E) - \sum_{v \epsilon values(a)} \left(\frac{|\{x \epsilon E | V(x,a) = v\}|}{|E|} . H(\{x \epsilon E | V(x,a) = v\}) \right) \quad (4)$$

3.4 MODEL CREATION

In this Phase, machine learning techniques are applied to the reduced feature vector to learn a model which is used for malware classification. Both supervised and unsupervised techniques can be used. Many states of the art techniques that are widely used to detect malware are Naïve Bayes, SVM, DT and RF.

Initially, the dataset consisting of malware and benign apps are split into two parts with 80% samples for training and 20% for testing. There are many factors that are used for determining the quality of the model like accuracy, F1, Recall, precision etc. These metrics help to justify the performance of the algorithm being used for model creation.

Experimental Studies shows that static analysis is widely used for detecting mobile malware as it is efficient and requires less computation. However, sophisticated malware can bypass static malware detectors. On the other hand, the Dynamic analysis is computationally expensive and hence, the on-device dynamic analysis is rarely used. Combination of both techniques is preferred and yields a better result.

We have used TensorFlow and keras (python-based framework) for implementing a deep neural network. We have implemented multi-model neural networks by varying number of neurons present in hidden layer and number of hidden layers in the network. Our results show the promising result with good detection accuracy. We have compared our approach with other state of the art

techniques. A brief overview of the techniques that have been used in this research has been given in the next section.

3.3.1 Neural Network

Before understanding the DNN, we should first understand the difference between DNN and shallow neural network. A neural network is a heuristic technique that aims to mimic the human brain to solve the problem of regression or classification. It helps in solving the problems faster than traditional systems and with good result. Here, we have covered the basic concept and terminologies involved in the ANN. We have also discussed the architecture of a simple perceptron as shown in figure 3.3. For the general model of the ANN, the net input can be calculated as follows −

$$Y_{in} = X_1.W_1 + X_2.W_2 + X_3.W_3\ldots\ldots\ldots + X_m.W_m \quad (5)$$

$$Y_{in} = \sum_i^m Xi.Wi \quad (6)$$

The output is computed by using the activation function on the input.

$$Y=F(Y_{in}) \quad (7)$$

Where X_i is input, W_i is weight, F is activation function.

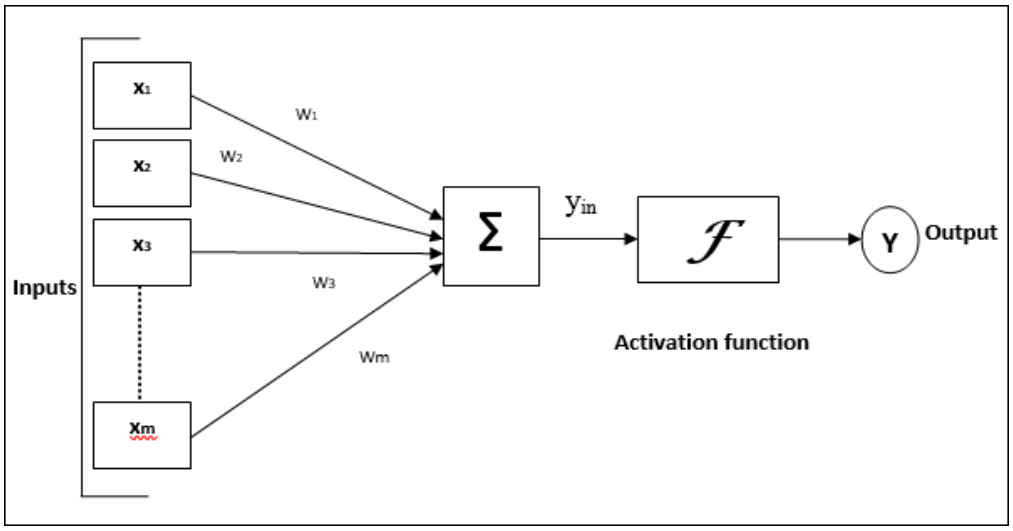

Fig. 3.3: General Model of ANN [30]

3.3.2 Deep Neural Network

Deep Learning is a type heuristic approach that is based on ANN. It has been applied to many areas like Computer Vision, Speech Recognition, Social Network Analysis, etc. A DNN is a type of artificial neural network with more than two layers. A neural network is a machine learning model to mimic the processing of the biological brain. It is inspired by the distributed communication nodes as in a biological system. A DNN can also be defined as a network that has at least one hidden layer between input and output layer. Each layer performs some kind of processing in order to get the desired result in the end.

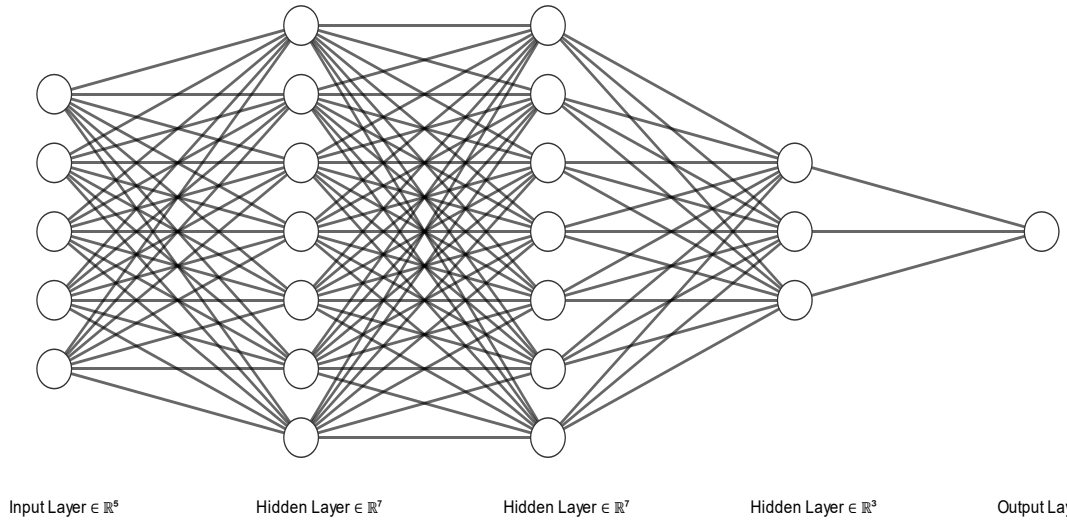

Fig 3.4: Deep Neural Network

3.3.3 Multi-Modal Neural Network

Multi-Modal learning is an approach to learning a good model by using the joint representation of different modalities. Modality refers to the way in which something can be experienced. A problem can be labeled as a multimodal problem when it includes multiple such modalities. For example, consider a case of an image. An image is usually associated with a description (A long text describing the image), tags and pixel intensities. These all constitute the different modalities for the problem. Multi-Modal learning helps to learn such models that can solve these problems efficiently and effectively. Multi-Modal learning models are also capable of learning missing modalities given the observed ones. In a multi-modal learning approach, multiple models are built based on different modalities.

All the models are then merged by using a merging layer and then final modal is used to predict the result. The process is shown in figure 3.5 as shown below.

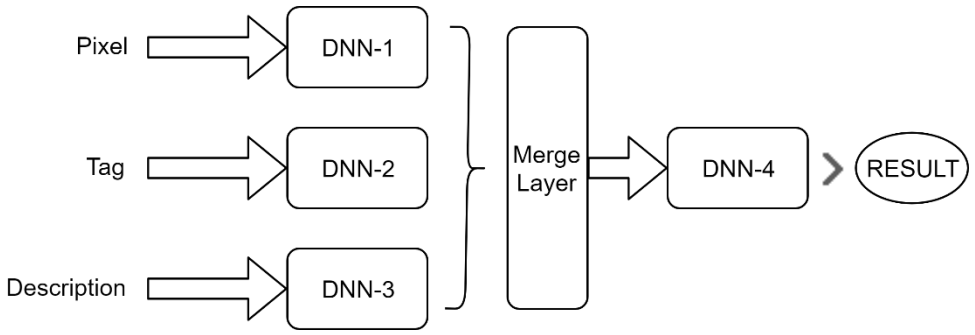

Fig 3.5: Multi-Modal Deep Neural Network (DNN)

3.3.5 Rotation Forest

Rotation Forest model is also an ensemble learning technique that is used to solve classification and regression problems. Unlike Random forest, Rotation forest is used when number of ensembles are small. Rotation forest model requires some parameters that are defined by the user.

Let **t** be the number of trees required to be built. We start the iteration from 1 to **t**. We perform the following steps for each tree –

1. Split the attributes in the training set into K nonoverlapping subsets of equal size.
2. We have K datasets, each with K attributes. For each of the K datasets, we proceed to do the following.
3. Bootstrap 75% of the data from each K dataset and use the bootstrapped sample for further steps.
4. Run a principal component analysis on the ith subset in K. Retain all the principal components. For every feature j in the Kth subset, we have a principal component, a. Let's denote it as a_{ij}, where it's the principal component for the jth attribute in the ith subset.
5. Store the principal components for the subset.

6. Create a rotation matrix of size, n X n, where n is the total number of attributes. Arrange the principal component in the matrix such that the components match the position of the feature in the original training dataset.
7. Project the training dataset on the rotation matrix using the matrix multiplication.
8. Build a decision tree with the projected data set.
9. Store the tree and rotation matrix.

Chapter 4

PROPOSED WORK

This chapter discusses the proposed architecture in depth. The proposed architecture is illustrated in Fig. 4.1. The chapter is split into two sections –

Section 4.1 gives a brief overview of the proposed architecture

Section 4.2 discusses model training and prediction.

4.1 PROPOSED FRAMEWORK

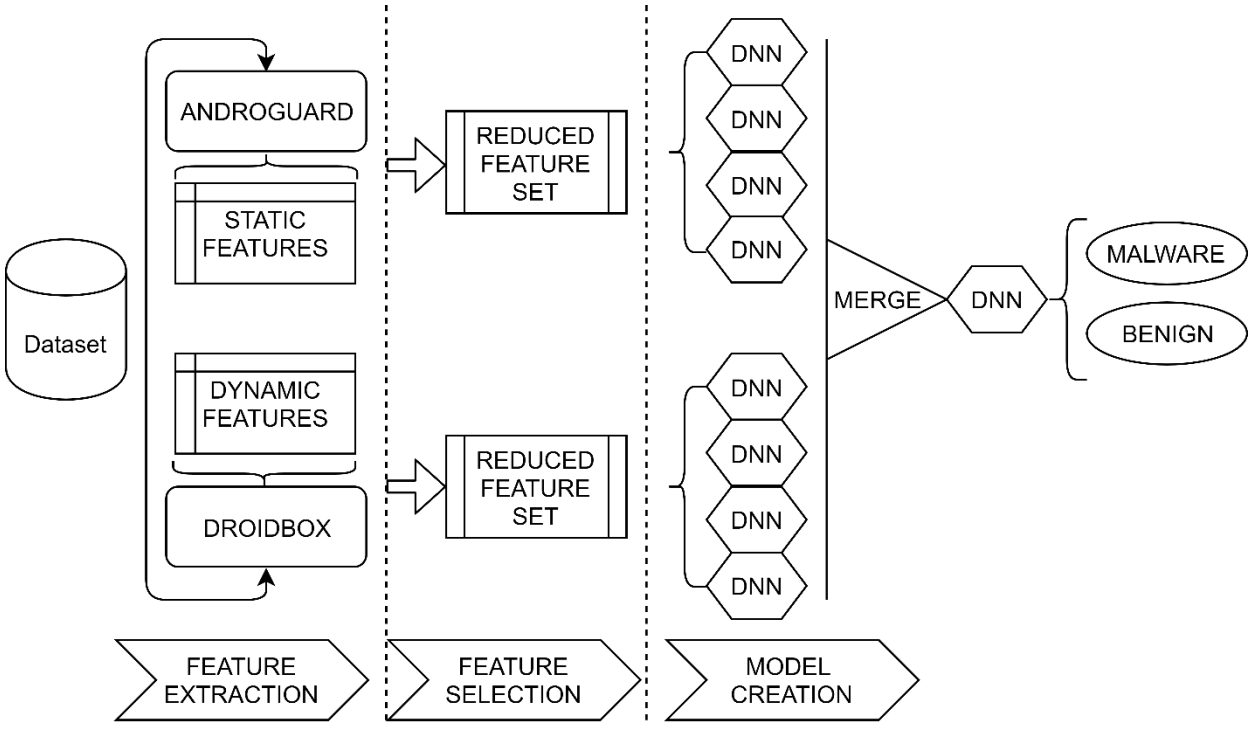

Fig. 4.1: Proposed Architecture

In this research, we have implemented a hybrid modal based on multi-modal learning. Our approach is categorized into four broad steps – dataset creation, feature extraction, feature

selection, and model creation. These steps in detail are summarized below –

1. In the first step, the dataset is downloaded from repositories. Malware apps are collected from VirusShare repository [7] and benign apps are collected from AndroZoo repository [6]. We have collected a total of 4000 applications.

2. In the next step, the features are extracted from the collected dataset. We have used two approaches for feature extraction – static analysis and Dynamic analysis. We have used AndroGuard for collecting static features and Droidbox for collecting dynamic features. We have collected a total of 118333 features. All the data is persisted in a CSV file.

3. In the next step, we have used Info Gain feature selection algorithm to reduce the feature vector dimensionality. Not only it helped in reducing the feature vector size but also helped in improving the computational time and accuracy of the classifiers.

4. In the final step, we have implemented our proposed approach as shown in the fig. 4.1. We have categorized the features into eight broad categories. We have built a classifier for each feature category using a deep neural network. We then have used a Merge layer to merge all the models. Finally, we have used a deep neural network to predict the result.

4.2 MODEL TRAINING AND PREDICTIONS

A multi-modal neural network contains a number of models that together provides an end result. It consists of an initial DNN and final DNN. The initial DNN consists of one input layer and two hidden layers with each neuron having ReLU as the activation function. The number of neurons is varied in the network in order to tune it for better predictions. The final DNN consists of one merging layer, two hidden layers, and one final output layer. Each layer defined in this network has used ReLU function to be used as activation function except the final output layer. The final output layer has used Sigmoid as the activation function. We have used two activation functions which are defined below.

4.2.1 Activation Function

Activation functions are very significant in a ANN. They help by introducing non-linear properties to our network. Its main purpose is to convert an input signal of a node to an output signal based on some computation. There are many types of activation function. In this research, we have focused only on ReLU and Sigmoid function.

1. **Sigmoid Activation Function –** It is an activation function of the form as shown in equation 8. It is an S-shaped curve with value range 0 to 1.

$$F(x) = 1/(1 + e^{-x}) \quad (8)$$

2. **ReLU Activation Function –** It stands for Rectified Linear units. It is defined by the equation number 9. It is a very simple and efficient activation function. Its value range is [0, inf].

$$F(x) = MAX(0, x) \quad (9)$$

4.2.2 Performance Metrics

In this research, we have compared our technique with other state of the art techniques. We have compared different approaches based on key performance metrics which are considered as standard while comparing machine learning algorithms. The following metrics are used to compare different algorithms –

1. Precision
2. Recall
3. Accuracy
4. F-Score

Chapter 5

EXPERIMENTAL RESULTS

In this section, we have briefly reported the performance of our implemented approach and compared it with other state of the art approaches. Our main motive is to demonstrate that multi-modal learning is an efficient and effective technique for detecting android malwares. We are able to achieve the best result that is 97.25% accuracy using this technique. The following system configuration has been used while conducting the experiments:

- Processor: Intel Core i5 2.7 Ghz
- Main Memory: 8 GB
- Software Used: Jupyter Notebook

The chapter is divided into following sections –

Section 5.1 discusses the evaluation method.
Sections 5.2 gives a brief about our analysis and result.

5.1 MODEL EVALUATION

We have used precision, F1, Recall, and accuracy as performance metrics. It is found that these measurements are very effective to assess the quality of classification in case of Android Malware detection.

i. **Precision**

It is defined as the probability that app is classified correctly as malware.

$$Precision = \frac{TP}{TP+FP} \quad (10)$$

ii. **Recall**

It is defined as a fraction of total malware samples that are labelled as malware.

$$Recall = \frac{TP}{TP+FN} \quad (11)$$

iii. **Accuracy**

It is defined as the ratio of correctly classified samples by total number of samples.

$$Accuracy = \frac{TP+TN}{TP+FP+TN+FN} \quad (12)$$

iv. F-Score

It is also called the F1 score or F measure. It is defined by the below equation.

$$FScore = 2 * (\frac{Precision * Recall}{Precision + Recall}) \quad (13)$$

5.2 MODEL PREDICTION

We have divided the large dataset of 4000 applications with 80% samples for training and rest 20% samples for testing. Training Dataset is used to train the classifier and the Testing dataset is used to evaluate the model. We have split the dataset with 80% samples for training and 20% samples for testing. We have analyzed a few approaches based on how they have constructed the feature vector. In comparison, we have focused on and compared different models based on a set of features used in the approach. All the feature set is broadly categorized into eight categories. Let us consider S be the set of all features. The eight disjoint sets are defined below.

S1: Permissions
S2: API Calls
S3: Opcode
S4: System Events
S5: Enforced Permissions
S6: Data Leaks
S7: File

S8: Crypto Calls

In our approach, we have followed a notation to represent multi-modal architecture. The multi-modal network consists of initial and final DNN. The initial DNN has one input layer and 2 hidden layers each having n number of neurons. The final DNN has one merging layer, 2 hidden layers and 1 output layer with predefined neurons. Syntax to describe the multi-modal architecture is defined in equation 14.

Representation: $[\ [X_1, X_2, X_3 \ldots]\ [X_1, X_2, X_3 \ldots]]$ (14)

Here the first part represents the configuration of initial deep neural network and the second part represents the configuration of a final deep neural network. The X_i represents the number of neurons present in the hidden layer. The shape of the 1-D array represents the total number of hidden layers. We have summarized our analysis in table 5.1.

TABLE 5.1: MULTI-MODAL NEURAL NETWORK ANALYSIS

MULTI-MODAL NETWORK CONFIGURATION	ACCURACY	PRECISION	RECALL	F-SCORE
[[5,5,5] [5,5,5]]	96.12%	99.23%	93.25%	96.14%
[[5,5] [5,5]]	96.5%	98.98%	94.21%	96.54%
[[5][5]]	96.87%	99.24%	94.69%	96.91%
[[10,10,10] [10,10,10]]	96.87%	98.75%	95.81%	96.93%
[[10,10] [10,10]]	97%	99.24%	94.93%	97.04%
[[10][10]]	96.87%	99.24%	94.69%	96.91%
[[20,20,20] [20,20,20]]	96.75%	98.74%	94.93%	96.80%
[[20,20] [20,20]]	97.25%	99.24%	95.42%	97.29%
[[20][20]]	96.75%	98.74%	94.93%	96.80%
[[30,30,30] [30,30,30]]	96.62%	98.98%	94.45%	96.67%
[[30,30] [30,30]]	97.12%	99.24%	95.18%	97.17%
[[30][30]]	96.87%	98.99%	94.93%	96.92%

Based on the above analysis, we found that the best result is achieved when we use two hidden layers in both initial and final DNN with each hidden layer having 20 neurons. The best result

achieved is 97.25% accuracy. We have compared our approach with two other approaches. Approach-1 is similar to DREBIN [5] and Approach-2 is similar to DroidDet [4]. Both these techniques are based on static analysis only. We have extended both the techniques for dynamic and hybrid analysis. We have provided a comparative analysis which is given in the below table 5.2.

TABLE 5.2: COMPARATIVE ANALYSIS

APPROACH	MODEL	FEATURE SET	ACCURACY	PRECISION	RECALL	F-SCORE
APPROACH-1	SVM	STATIC	87.12%	88%	87%	87%
APPROACH-2	ROTATION FOREST MODEL	STATIC	87.37%	88%	87%	87%
EXTENDED APPROACH-1	SVM	DYNAMIC	72%	68.74%	53.25%	65.53%
		HYBRID	89.35%	85.88%	88.45%	89.44%
EXTENDED APPROACH-2	ROTATION FOREST MODEL	DYNAMIC	71.62%	66.40%	49.47%	62.60%
		HYBRID	91.5%	87.79%	88.59%	90.76%
OUR APPROACH	DEEP NEURAL NETWORK	STATIC	95.87%	98.64%	92.8%	95.67%
		DYNAMIC	72%	91.22%	50.48%	65%
		HYBRID	96.12%	94.61%	98.05%	96.30%
OUR APPROACH	MULTI-MODAL NEURAL NETWORK	HYBRID	97.25%	99.24%	95.42%	97.29%

Chapter 6

CONCLUSION

In this research, we have discussed malware, malware types and their source of distribution. We have also discussed different malware detection techniques and highlighted the issues and limitations of those techniques. Keeping those limitations in mind, we have implemented a hybrid android malware detection technique using multi-modal learning to detect malwares in android devices. This model helped to overcome the problems of both static and dynamic analysis when used individually. We have also compared our approach with two other approaches and found that multi-modal learning yields a better result. We have also extended the approaches with dynamic and hybrid analysis. We have also implemented a deep neural network and compared it with our approach. The result indicates that multi-modal learning is an effective technique and yield a better result when compared with other techniques. Our analysis also suggests that the model yields a better result when features are provided to it individually rather than combining all features into one long feature vector. In this research, we have used both static and dynamic feature extracted from a large dataset of 4000 applications. The applications were collected from VirusShare and AndroZoo repository. We have achieved a good result with 97.25% accuracy.

REFERENCES

[1] Statista, “Mobile OS Market Share,” *www.statista.com*, 2018. [Online]. Available: https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/.

[2] K. Labs, “Kaspersky_Securelist Threat Report,” 2019. [Online]. Available: https://securelist.com/threat-category/mobile-threats/.

[3] Q. Heal, “Quick Heal Annual Threat Report,” 2019.

[4] H. J. Zhu, Z. H. You, Z. X. Zhu, W. L. Shi, X. Chen, and L. Cheng, “DroidDet: Effective and robust detection of android malware using static analysis along with rotation forest model,” *Neurocomputing*, vol. 272, pp. 638–646, 2018.

[5] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, “Drebin: Effective and Explainable Detection of Android Malware in Your Pocket,” *Proc. 2014 Netw. Distrib. Syst. Secur. Symp.*, 2014.

[6] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, “AndroZoo : Collecting Millions of Android Apps for the Research Community,” pp. 468–471, 2016.

[7] “VirusShare,” 2019. [Online]. Available: https://virusshare.com/.

[8] “First Kotlin-Developed Malicious App Signs Users Up for Premium SMS Services - TrendLabs Security Intelligence Blog.” [Online]. Available: http://blog.trendmicro.com/trendlabs-security-intelligence/first-kotlin-developed-malicious-app-signs-users-premium-sms-services/.

[9] Wikipedia, “WannaCry ransomware attack,” *Wikipedia Foundation*, 2017. [Online]. Available: https://en.wikipedia.org/wiki/WannaCry_ransomware_attack.

[10] “Android.Samsapo - Mobile Worm.” [Online]. Available: https://www.symantec.com/security_response/writeup.jsp?docid=2014-050111-1908-99.

[11] “Android/Trojan.AsiaHitGroup - Malwarebytes Labs _ Malwarebytes Labs.” [Online]. Available: https://blog.malwarebytes.com/cybercrime/2017/11/new-trojan-malware-discovered-google-play/.

[12] “Mobile Spyware- Go Keyboard app.” [Online]. Available: https://www.bleepingcomputer.com/news/security/popular-android-keyboard-app-caught-collecting-user-data-running-external-code/.

[13] R. Katarya and P. Mamgain, “A Survey on Android Malware Detection Techniques,” *Int. J. Comput. Eng. Appl.*, vol. XII, 2018.

[14] J. Li, L. Sun, Q. Yan, Z. Li, W. Srisa-an, and H. Ye, “Significant Permission Identification for Machine Learning Based Android Malware Detection,” vol. 3203, no. c, 2018.

[15] A. Firdaus, N. B. Anuar, A. Karim, M. Faizal, and A. Razak, “Discovering optimal features using static analysis and a genetic search based method for Android malware detection *,” vol. 19, no. 6, pp. 712–736, 2018.

[16] E. Billah, M. Debbabi, A. Derhab, and D. Mouheb, “MalDozer : Automatic framework for

android malware detection using deep learning," *Digit. Investig.*, vol. 24, pp. S48–S59, 2018.

[17] C. Tumbleson and R. Wiśniewski, "Apktool - A tool for reverse engineering 3rd party, closed, binary Android apps." [Online]. Available: https://ibotpeaches.github.io/Apktool/.

[18] "Androguard - Reverse engineering Tool." [Online]. Available: https://github.com/androguard/androguard.

[19] S. Wang, Z. Chen, Q. Yan, B. Yang, L. Peng, and Z. Jia, "A mobile malware detection method using behavior features in network traffic," *J. Netw. Comput. Appl.*, 2019.

[20] P. Feng, J. Ma, C. Sun, and Y. Ma, "A Novel Dynamic Android Malware Detection System With Ensemble Learning," *IEEE Access*, vol. 6, pp. 30996–31011, 2018.

[21] W. You, B. Liang, W. Shi, P. Wang, and X. Zhang, "TaintMan: an ART-Compatible Dynamic Taint Analysis Framework on Unmodified and Non-Rooted Android Devices," *IEEE Trans. Dependable Secur. Comput.*, vol. 5971, no. c, 2017.

[22] L. Wei, W. Luo, J. Weng, Y. Zhong, X. Zhang, and Z. Yan, "Machine Learning-based Malicious Application Detection of Android," *IEEE Access*, vol. PP, no. 99, p. 1, 2017.

[23] "GitHub - pjlantz_droidbox_ Dynamic analysis of Android apps." [Online]. Available: https://github.com/pjlantz/droidbox.

[24] S. Arshad *et al.*, "SAMADroid : A Novel 3-Level Hybrid Malware Detection Model for Android Operating System," *IEEE Access*, vol. 6, pp. 4321–4339, 2018.

[25] Z. Yuan, Y. Lu, and Y. Xue, "Droiddetector: android malware characterization and detection using deep learning," *Tsinghua Sci. Technol.*, vol. 21, no. 1, pp. 114–123, 2016.

[26] M. Sun, X. Li, J. C. S. Lui, R. T. B. Ma, and Z. Liang, "Monet: A User-Oriented Behavior-Based Malware Variants Detection System for Android," *IEEE Trans. Inf. Forensics Secur.*, vol. 12, no. 5, pp. 1103–1112, 2017.

[27] X. Su, D. Zhang, W. Li, and K. Zhao, "A Deep Learning Approach to Android Malware Feature Learning and Detection," *2016 IEEE Trust.*, pp. 244–251, 2016.

[28] D. Li, Z. Wang, and Y. Xue, "Fine-grained Android Malware Detection based on Deep Learning," *2018 IEEE Conf. Commun. Netw. Secur.*, vol. 1, no. L, pp. 1–2, 2018.

[29] M. A. Akcayol, "Permission Based Android Malware Detection With Multilayer Perceptron," pp. 0–3, 2018.

[30] "Neural Network," 2018. [Online]. Available: https://www.tutorialspoint.com/artificial_intelligence/artificial_intelligence_neural_networks.htm.

LIST OF PUBLICATIONS OF THE CANDIDATE'S WORK

[1] Published a research paper titled "*A Survey on Android Malware Detection Techniques* " in *International Journal of Computer Engineering and Applications*, Volume XII, Special Issue, August 18, www.ijcea.com ISSN 2321-3469. Presented the paper in IEEE Conference ICRTCST-2018.