RANSOMSHIELD: MITIGATING ANDROID RANSOMWARE ATTACKS THROUGH PERMISSION-BASED ANALYSIS USING MACHINE LEARNING

By

Tatheer Ayesha

Fall 2020 – MS (IS) – 00000328873

Supervisor

Dr. Mehdi Hussain

Department of Computing

A thesis submitted in partial fulfillment of the requirements for the degree of Masters of Science in Information Security (MS IS)

In

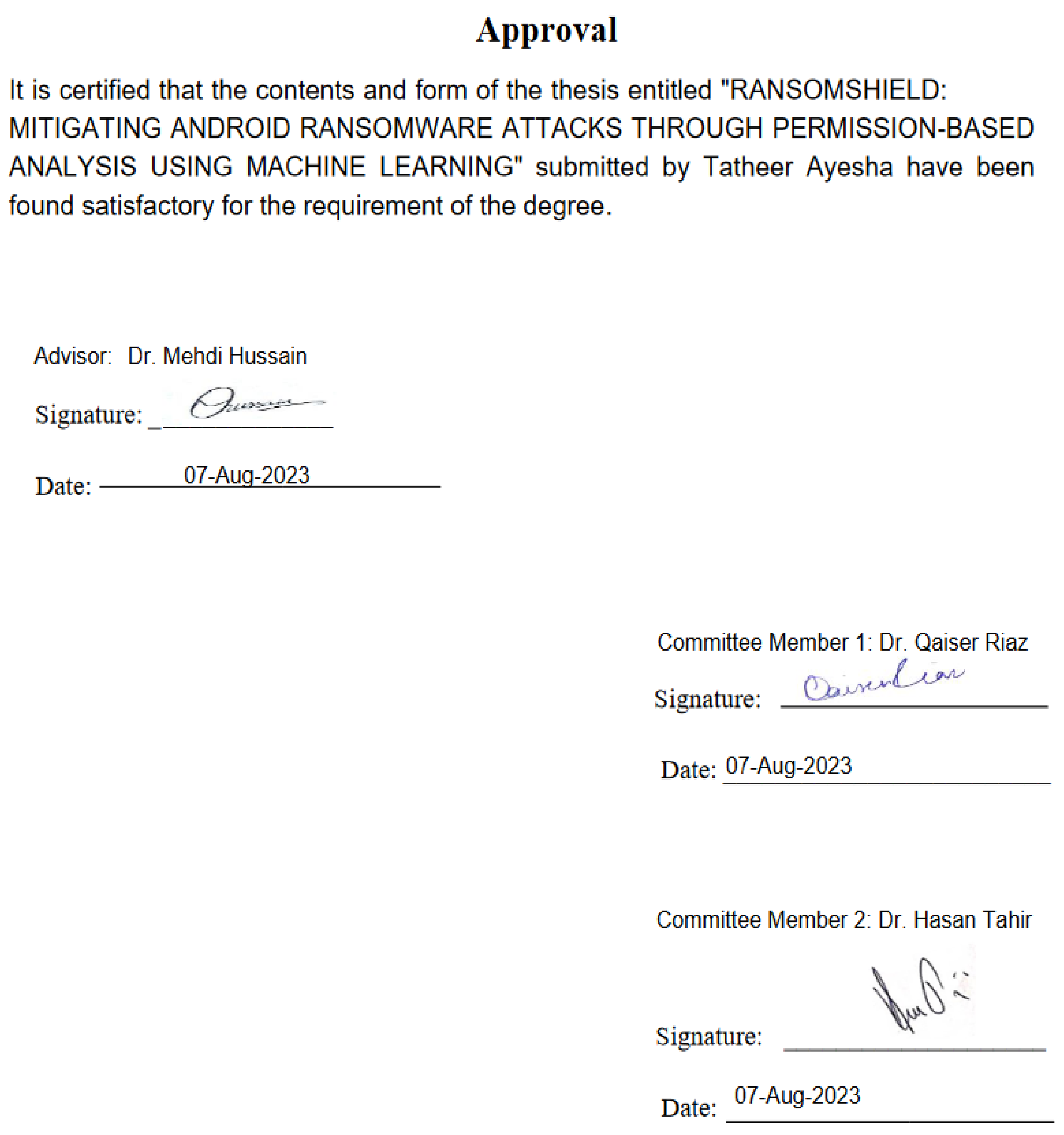
School of Electrical Engineering and Computer Science,

National University of Sciences and Technology (NUST),

Islamabad, Pakistan.

(August 2023)

Approval



Approval

It is certified that the contents and form of the thesis entitled "RANSOMSHIELD: MITIGATING ANDROID RANSOMWARE ATTACKS THROUGH PERMISSION-BASED ANALYSIS USING MACHINE LEARNING" submitted by Tatheer Ayesha have been found satisfactory for the requirement of the degree.

Advisor: Dr. Mehdi Hussain

Signature: [illegible]

Date: 07-Aug-2023

Committee Member 1: Dr. Qaiser Riaz

Signature: [illegible]

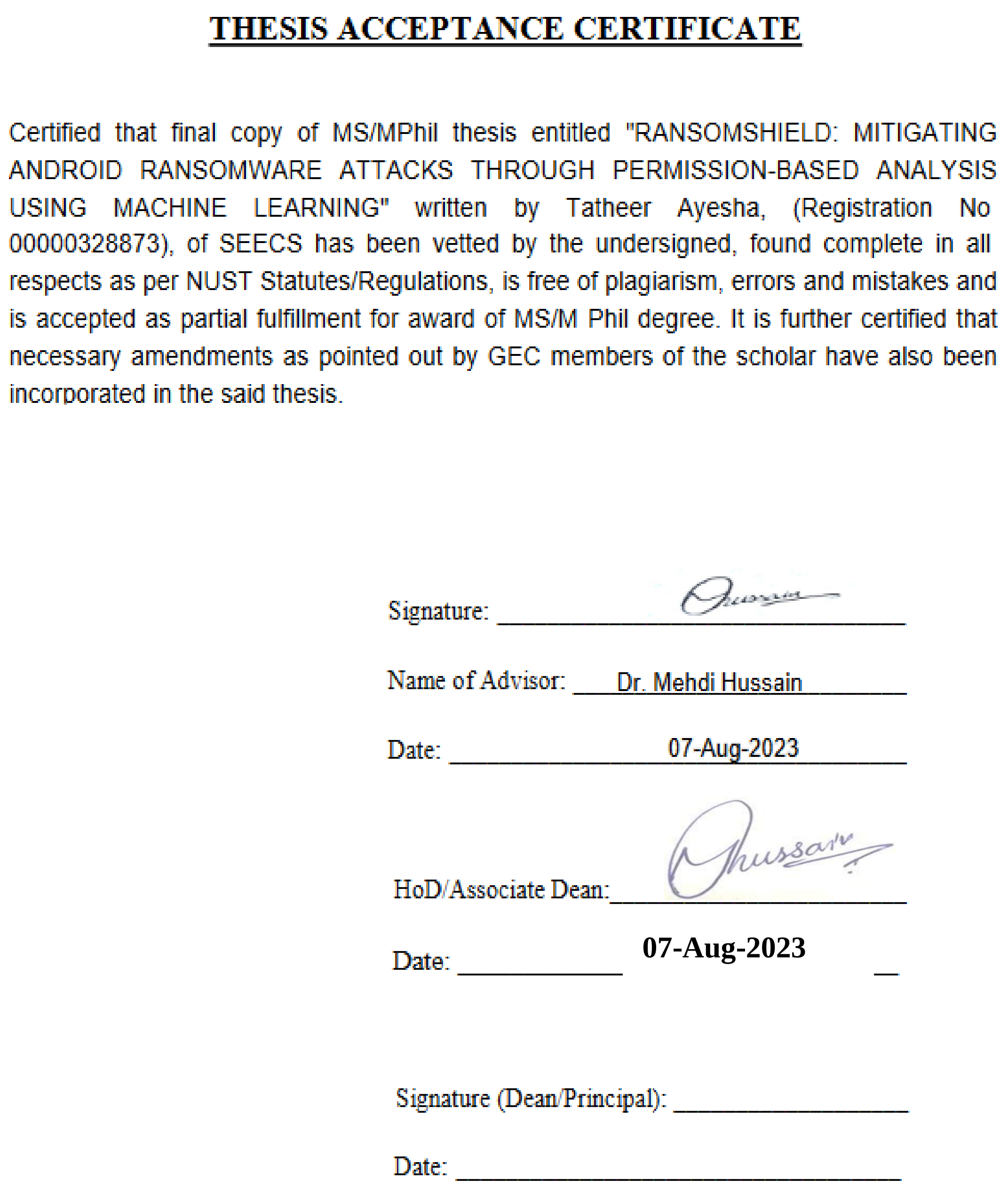
Date: 07-Aug-2023

Committee Member 2: Dr. Hasan Tahir

Signature: [illegible]

Date: 07-Aug-2023

Thesis Acceptance Certificate



THESIS ACCEPTANCE CERTIFICATE

Certified that final copy of MS/MPhil thesis entitled "RANSOMSHIELD: MITIGATING ANDROID RANSOMWARE ATTACKS THROUGH PERMISSION-BASED ANALYSIS USING MACHINE LEARNING" written by Tatheer Ayesha, (Registration No 00000328873), of SEECS has been vetted by the undersigned, found complete in all respects as per NUST Statutes/Regulations, is free of plagiarism, errors and mistakes and is accepted as partial fulfillment for award of MS/M Phil degree. It is further certified that necessary amendments as pointed out by GEC members of the scholar have also been incorporated in the said thesis.

Signature: [illegible]

Name of Advisor: Dr. Mehdi Hussain

Date: 07-Aug-2023

HoD/Associate Dean: [illegible]

Date: **07-Aug-2023**

Signature (Dean/Principal): ____________

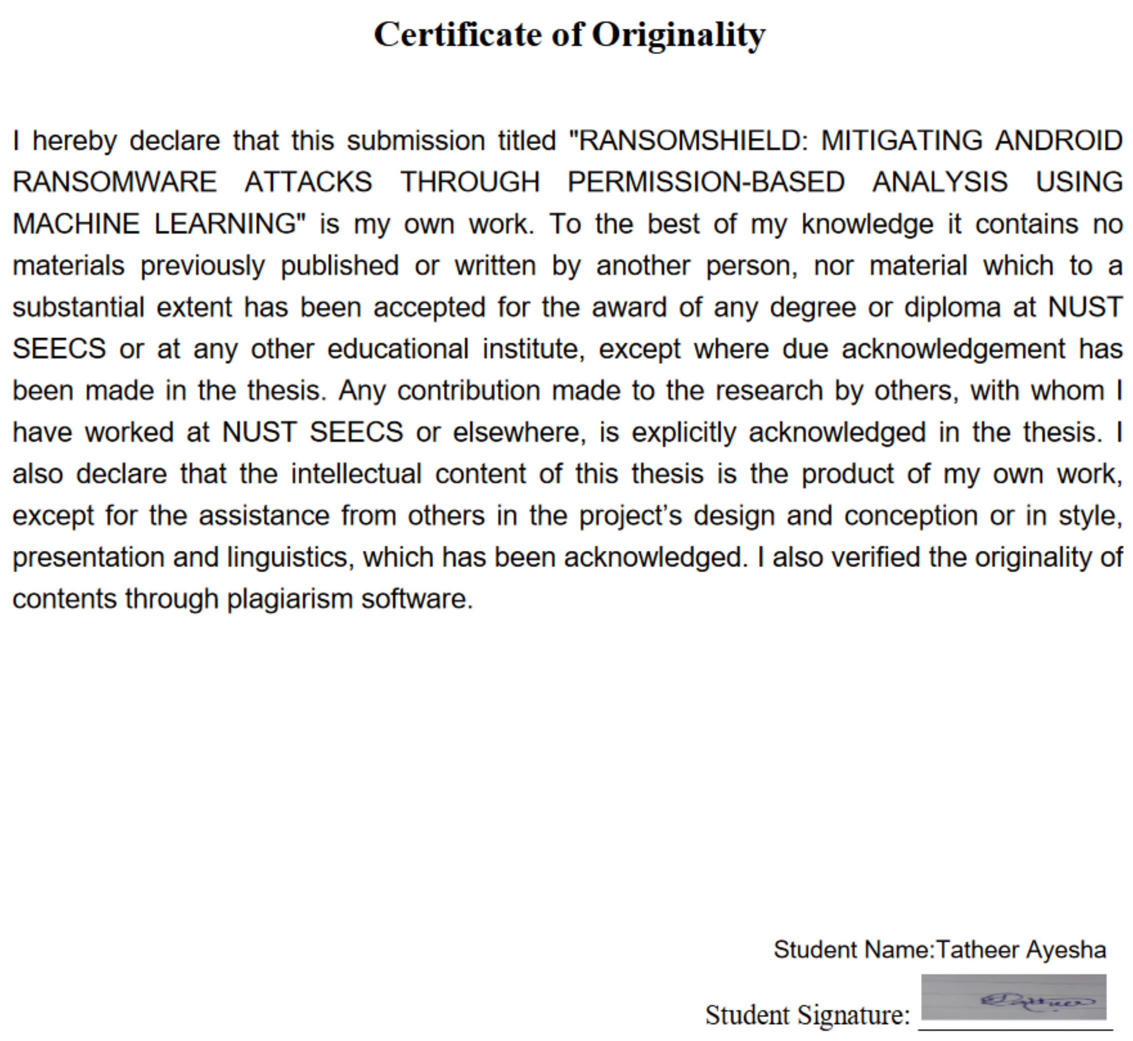
Date: ____________

Dedication

I dedicate this thesis to my Teachers, Parents and Siblings for their endless prayers, guidance, and encouragement.

.

Certificate of Originality



Certificate of Originality

I hereby declare that this submission titled "RANSOMSHIELD: MITIGATING ANDROID RANSOMWARE ATTACKS THROUGH PERMISSION-BASED ANALYSIS USING MACHINE LEARNING" is my own work. To the best of my knowledge it contains no materials previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any degree or diploma at NUST SEECS or at any other educational institute, except where due acknowledgement has been made in the thesis. Any contribution made to the research by others, with whom I have worked at NUST SEECS or elsewhere, is explicitly acknowledged in the thesis. I also declare that the intellectual content of this thesis is the product of my own work, except for the assistance from others in the project's design and conception or in style, presentation and linguistics, which has been acknowledged. I also verified the originality of contents through plagiarism software.

Student Name:Tatheer Ayesha

Student Signature: [illegible]

Acknowledgment

First of all, I would like to thank Allah, the Almighty for giving me the ability and strength to carry out this research.

My deepest gratitude to my supervisor Dr. Mehdi Hussain for his continuous support and guidance during my thesis. I could not have imagined having a better supervisor and mentor for my master's degree. I am also thankful to my teachers for providing me with an academic base, which enabled me to complete this thesis.

I am thankful to all my fellows and friends for their support and motivation. Last but not the least, I would like to thank my parents for their endless prayers and support throughout.

Table of Contents

List of Figures

List of Tables

Abstract

Ransomware is one of the threatening malwares for security systems, targeting both Windows and mobile platforms. It has the ability of encrypting sensitive user data and command a deliverance of data in return. The extensive growth of ransomware attacks is due to the spread of mobile malware with irrelevant permissions and malware codes in mobile applications. In the literature, there are several proposed models for detecting ransomware. These models typically utilize various attributes, such as API calls, system calls, intents, permissions, and other dynamic features of an application. However, the extensive utilization of the aforementioned attributes can lead to the increased complexity of the detection system. Therefore, a deep investigation of Android Permissions to identify the significant set of permissions that can be used to detect ransomware applications prior to their initiation is focused in this study. The proposed RansomShield technique first identifies the significant permissions to be used and then employed machine learning algorithms to classify. Through our implementation, the proposed model successfully identified 16 significant permissions to predict ransomware applications with 97% detection accuracy. The classifiers we used for this model are supervised for ransomware detection for accomplishing high accuracies i.e., 97% with Random Forest, 95% with Decision Tree, 97% with SVM, 95% with Logistic Regression, 73% with Naïve Bayes, 94 % with Bagging, 100% with Gradient boosting and 97% with KNN models. The proposed model outperformed the existing model regarding a limited no. of permissions while achieving high accuracy. Further, a new permission-based dataset is created that is online and available for future researchers.

Keywords: *Android, Ransomware, Machine learning, Accuracy, Permissions*

1. Introduction

The overview of fundamental concepts, their importance, and the background studies are elaborated in this chapter. It outlines the thesis's overall anatomy and briefly focuses its additional composition. The purpose of conducting the research is explained as well. This chapter discusses the important contributions, notable advantages, the scope of the work, and the main goals.

1.1 Overview

Android mobile is common among the mobile platforms. This made Android users a well-known target for many security attacks. The Mobile Operating system market shared 71.94% of Android[1]. However, from the official app stores, approximately 2.56 million applications[2] can be downloaded. Many users choose to use Android mobile phones due to their affordability, portability, and the ability to access useful applications without spending any money. Meanwhile, Google play store has created open-source policy of App availability which enable the accessibility for the users and developers.

The issued policy offered great lenience for App authentication to maintain the recognition of the platform at the time of issuance. Nevertheless, app availability, usage, and the deployment of malware is also linked with the interest of malicious on android devices.

Mobile malware statistics show that variants of malware are developing every few seconds[3]. These malicious apps have a hidden capability of performing offensive activities to target individuals or organizations, malicious applications are released in different variants which

1: "gs.statcounter", https://gs.statcounter.com/android-version-market-share/mobile-tablet/worldwide. [Accessed 2 January 2023].
2: "Businessofapps", https://www.businessofapps.com/data/app-statistics/, available. [Accessed 2 January 2023].
3: "securelist", https://securelist.com/it-threat-evolution-in-q2-2022-mobile-statistics/107123/. [Accessed 2 January 2023].
4:"welivesecurity",https://www.welivesecurity.com/wpcontent/uploads/2016/02/Rise_of_Android_Ransomware.pdf. [Accessed 12 December 2022]

makes them undetectable[4]. The installation of malicious applications becomes common as users download the applications without verifying the play stores, app name, and app permissions during installation. This unawareness of users gives chance the malicious actors to easily intrude into mobile phones through apps. To overcome the spread of malicious applications, researchers developed various detection models for mobile malware. They are also discussed with its pros and cons in literature. One of the malware types known as ransomware, has been found common among almost all platforms. Different tactics are used by ransomware to encrypt or lock the device. It serves the purpose of demanding money to releases the data.

1.2 Motivation

This research is focused on the permissions for ransomware detection. In literature, existing machine learning models are used for various malicious and specifically ransomware APKs detection approaches with consideration of the permissions and API calls [20-34]. Although, identifying effective and significant permissions can always be difficult. In this research, we have aimed at detection of ransomware residing in the Android platform using permissions-based approach. The research is inspired by Fahad Akbar et al. [2], Chen et al. [9], Alzahrani et al. [10] and Alsoghyer et al. [15] which detects the malwares in APKs based on permissions.

The precision detection of ransomware by minimizing the number of permissions required for classification is objective of this study. The research conducted by Chen et al. [9] considered 21 permissions for ransomware detection. The proposed approach has two primary aims. First, to develop an efficient dataset that is originally used for evaluation purposes. Second, to demonstrate that reducing the number of permissions that can minimize the processing load and improve the overall system accuracy.

1.3 Objectives

Research involves scientific methods to thoroughly examine specific issue or problem. In this investigation, we conducted a comprehensive analysis of hazardous behaviors in Android applications through static analysis to gain new insights. The main points of this study are described as:

- Exploring the potential of static analysis and discovering new possibilities by utilizing permissions as the primary static feature.
- Applying machine learning models to find out the effectiveness of different permissions in the analysis.
- Identifying the minimum set of permissions required for conducting the study.
- Obtained results should be better or comparable to the existing approaches.
- Incorporating other existing classifiers such as Naïve Bayes, Random Forest, Gradient boosting, bagging and KNN to improve the performance of the current method.
- Lastly, developing a lightweight Android Ransomware detection mechanism.

1.4 Research Questions

This section emphasizes the research questions that are framed to guide this study, as follows:

- **What is the motivation behind conducting this research?**

The Android ecosystem is continuously under threat from malwares, particularly ransomwares, which can compromise the user's sensitive data. Since such data is important to users, they require protection shield against such threats.

While there exist many Android ransomware detection methods, each with their own benefits, there is a need to explore if these methods can be made more efficient to offer lightweight detection approaches in terms of computational costs.

- **What are the research steps involved and its significance accordingly?**

This investigation aims to analyze the importance of static analysis for Android ransomware, with the goal of classifying ransomware mobile applications using the fewest possible static features. This will enable malware analysts and the research community to quickly identify ransomware applications. The study will employ both qualitative and quantitative static analysis of Android Ransomwares. To conduct the study, we have divided our approach into the following four steps:

- Samples collection and environmental setup
- Dataset Creation
- Feature Extraction
- Classification of generated results
- Report Writing

- **What are the aims of this study?**

The primary focus of this study is on the following aspects:

a) Determining the significant set of static features required to accurately classify a ransomware Android application.
b) Identifying the most effective classification model among those considered in this investigation, namely SVM, Decision Tree, Gradient Boosting, KNN, Bagging, Random Forest, and Naïve Bayes.

1.5 Problem Statement

To protect the users' data and block unauthorized access, the Android platform has implemented a security model based on permissions for the apps. Permissions are commonly used security evaluation metric in the Android platform. Apps require permission to be enabled for providing their functionality to users. Hybrid permissions used together may result in opposing behaviors, such as the app accessing and spreading users' sensitive information. Therefore, permission probing is a necessary step in Android malware detection. By closely examining the process of permission analysis, we can improve malware detection.

Thus, the problem statement for our study is to determine the minimum set of features, using permission-based static analysis of Android ransomware apps, to develop a lightweight ransomware detection system.

1.6 Solution Description

This study proposes an effective method for detecting android ransomware by utilizing permission-based static analysis of Android applications. The researchers first decompiled the apps using APK tool and obtained the APKs from CICAndMal2017 Dataset [17], AndroZoo [18] and the Official Play Store for the purpose of extracting permissions. Further a filtration process applied to extracted permissions using the Random Forest’s feature importance technique. In next steps, feature set is generated and different machine learning models are used for detecting ransomware Android apps utilization.

1.7 Thesis Contribution

Our research strategy “RANSOMSHIELD: Mitigating Android Ransomware Attacks Through Machine Learning-Powered Permissions-Based Analysis" By incorporating frequently used classifiers such as Bagging, Decision Tree, Logistic Regression, Gradient Boosting, Random

Forest, Naïve bayes, Support Vector Machine and KNN algorithms, our research methodology significantly enhanced the precision of ransomware detection. The proposed approach offers the following contributions:

The contributions of proposed plan are as follows:

- It achieved a detection accuracy of about 97% while reducing the number of required permission features.
- It introduced a lightweight framework for detecting ransomware APKs.
- It is compatible with other existing machine learning classifiers.
- The source code for the RansomShield is fully functional and available on GitHub.

1.8 Thesis Structure

The research continues as follows: Chapter 2 layout an overview of previous work on static analysis and detecting android ransomware applications. Chapter 3 provides the research approach employed in the study. Chapter 4 outlines the experimental setup, while Chapter 5 presents the experiment's results and the activities involved in data collection. Chapter 6 analyzes and discusses the experimental findings. Finally, the thesis is summed up and future research recommendation are made in Chapter 7.

1.9 Summary

The chapter aims to introduce crucial concepts related to malware analysis, with a focus on static and dynamic analysis techniques used to detect ransomware applications. It offers an outline of the scope and aims of the thesis, as well as a run-through of the research work's general structure. The subsequent chapter will delve into the literature review that was conducted for this thesis in more detail.

2. Literature Review

This chapter demonstrates the related work and techniques carried out by different researchers over the years. The work done is contributed towards the android ransomware detection.

2.1 Overview

The widespread use of Android as a smartphone operating system has prompted numerous studies to be conducted on Android apps from various perspectives. When investigating any types of malware, the technical aspect can be categorized into multiple branches, But main are static analysis, dynamic analysis, and a combination of both known as hybrid analysis (as Figure 2.1 represents). Hybrid malware analysis combines various techniques, including static and dynamic analysis, behavioral observation, network monitoring, memory inspection, reverse engineering, automated tools, and threat intelligence, to fully comprehend the behavior and characteristics of the malware. This means hybrid malware analysis includes all the artifacts (opcodes, intents, permissions, strings ip addresses, URLs etc) extracted by the various analysis techniques. The similar types of techniques are also performed for Android ransomware detection. This section aims to explore the current literature on the detection of ransomware Android apps using these analytical approaches.

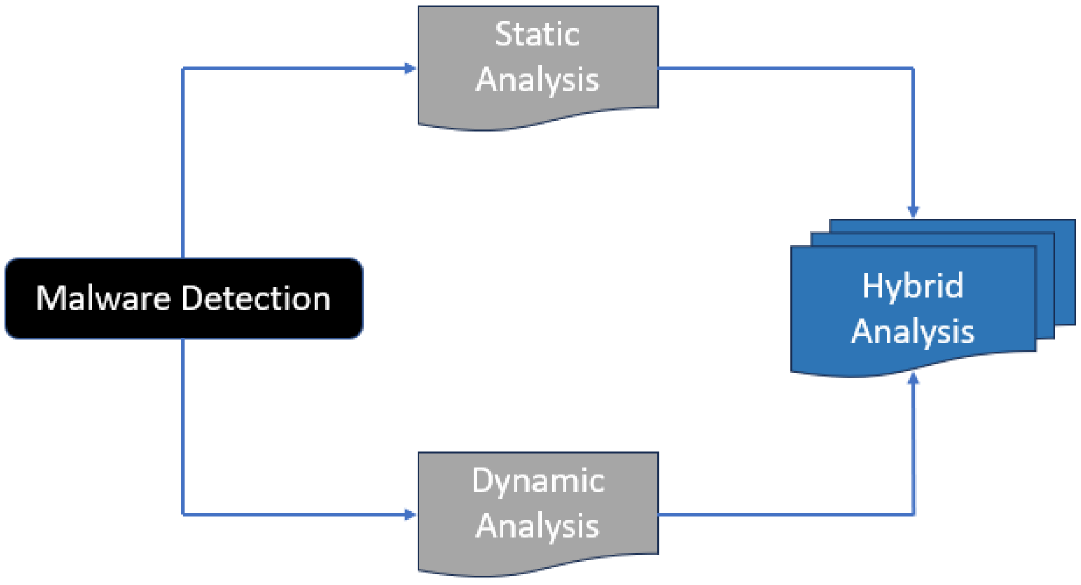


Figure 2.1 Malware Analysis Techniques

2.1.1 Malware Injection Process

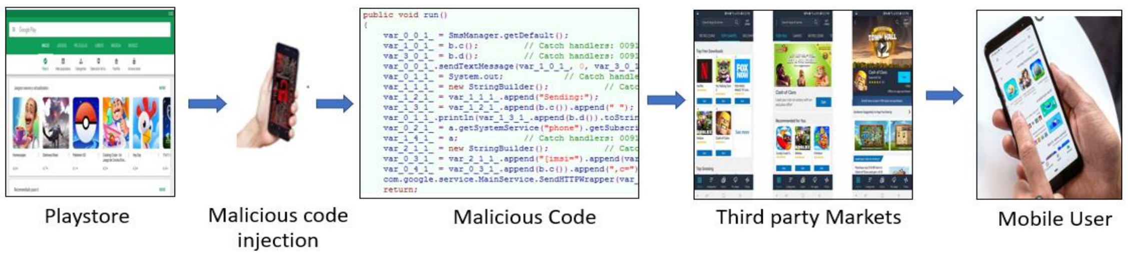
Malware injection is a process of creating malicious applications by carrying out three phases:

(i) Collecting either benign or malicious applications

(ii) Injecting malicious codes

(iii) Releasing these malicious applications to the third-party stores [36].

This process of malware injection is shown in the **Error! Reference source not found.**.2 as:



```
public void run()
{
    var_0_0_1_ = SmsManager.getDefault();
    var_1_0_1_ = b.c();          // Catch handlers: 0091
    var_3_0_1_ = b.d();          // Catch handlers: 0091
    var_0_0_1_.sendTextMessage(var_1_0_1_, 0, var_3_0_1
    var_0_1_1_ = System.out;              // Catch handle
    var_1_1_1_ = new StringBuilder();              // Catc
    var_1_2_1_ = var_1_1_1_.append("Sending:");
    var_1_3_1_ = var_1_2_1_.append(b.c()).append(" ");
    var_0_1_1_.println(var_1_3_1_.append(b.d()).toStrin
    var_0_2_1_ = a.getSystemService("phone").getSubscri
    var_1_4_1_ = a;              // Catch handlers: 0091
    var_2_1_1_ = new StringBuilder();              // Catc
    var_0_3_1_ = var_2_1_1_.append("[imsi=").append(var
    var_0_4_1_ = var_0_3_1_.append(b.c()).append(",c=")
    com.google.service.MainService.SendHTTPWrapper(var_
    return;
```

Figure 2.2-Malware Injection Process [36]

2.1.2 Malware Detection Mechanisms

To detect the malicious APKs on the Android phone, three strategies can be used. These Strategies are Static, Dynamic and Hybrid Analysis. The malware detection mechanisms for Android is shown in the figure 2.3.

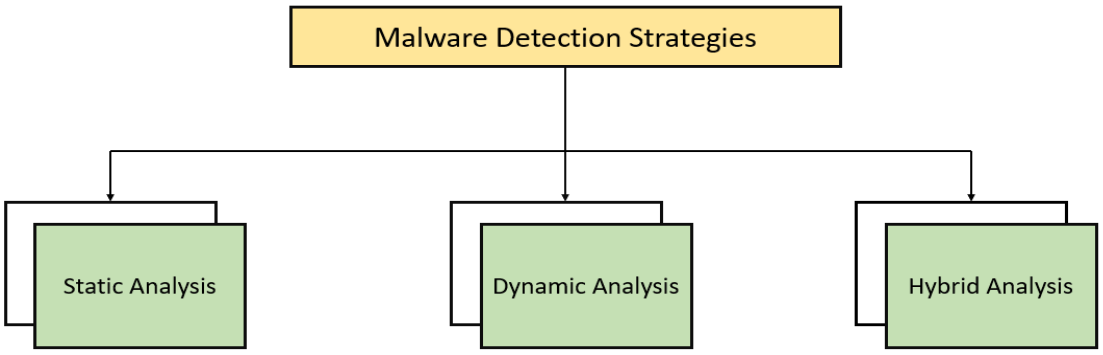


Figure 2.3-Malware Detection Strategies

a) Static Analysis

Static analysis is a security testing technique that evaluates the source code, byte code or application binaries in a non-runtime environment. It gives indications of security vulnerabilities to analyzes Meta and supporting details [37]. A broad range of strategies and techniques are used in static analysis to predict the runtime behavior of software before it is executed. This approach is particularly useful in identifying malicious or repackaged apps before they are installed or executed.

In the case of Android apps, which are distributed as application packages (APKs), the first step in static analysis is typically to decompile the app using tools such as AndroGuard [38] or apktool [39]. This process extracts Smali files containing essential information about the app, which are then examined for indications of malicious behavior. Based on an estimate of its potential properties, static analysis identifies whether an app can be considered as malicious or not. These approximations are usually based on factors such as permissions, code analysis or

API calls, Intents, App components, file properties, native code, and more [40]. Static analysis is often the preferred approach for detecting malware. It is considered an efficient method for protecting the market, as it consumes fewer resources and time. This faster detection method is particularly beneficial for resource-constrained Android devices [41].

b) Dynamic Analysis

Dynamic analysis focuses on the behavior of applications during runtime. It serves the purpose to recognize malicious behavior on real or emulated devices which is carried by deploying and running the application. Dynamic analysis can uncover hidden objectives of malware applications that may not be evident through static analysis. This type of analysis typically involves some level of interaction with the application, manually or automated. It gathers information on network activity, processor execution, system calls, SMS messages sent and received, phone calls, and other relevant data. By analyzing this information, dynamic analysis can help to separate between legitimate and malicious applications. Unlike static analysis, dynamic analysis provides a more accurate reflection of an app's true intentions. However, it can be resource-intensive and time-consuming to execute, making it less practical in certain situations [41].

c) Hybrid Analysis

Both static and dynamic analysis techniques are used in Hybrid Analysis to enhance detection accuracy. By combining the strengths and weaknesses of both approaches, it provides a more thorough examination of both application behaviors and installation files, making it a highly comprehensive analysis approach. Nevertheless, like dynamic analysis, hybrid analysis can be resource intensive.

2.1.3 Comparison of Analysis Techniques

Static analysis examines application characteristics in a safe environment where the likelihood of malicious behavior execution is minimal. This approach is particularly useful in environments with limited time and resources. In contrast, dynamic analysis provides a more detailed examination of applications and is considered a highly accurate detection method, albeit at a higher computational cost. Following the execution of different APKs from static analysis, dynamic analysis is performed.

For this reason, static analysis is generally faster and useful for developing an initial understanding of the expected behavior of APKs.

2.1.4 Ransomware

Ransomware is categorized intovarious types that were known, such as WannaCry, Petya/NotPetya, Locky, Cerber, CryptoLocker, Maze, Sodinokibi/REvil, GrandCab and Ryuk. These malicious programs lock or encrypt files and demand payment to restore accessibility. However, it's crucial to understand that new variants may have emerged since then. Wannna cry ransomware targets the systems having eternal blue vulnerability in Microsoft Windows systems. Petya and its variant NotPetya proved to be highly destructive ransomware. They focused on Windows systems and employed diverse propagation techniques. Locky propagated through malicious email attachments, encrypting files on infected systems, and demanding a ransom for decryption. Cerber ransomware distinguished itself by using voice synthesis to deliver ransom messages. Crypto ransomware targets files as a hijacked resource. Maze ransomware gained attention for its practice of exfiltrating sensitive data before encrypting files, posing a threat to publish the data unless the ransom was paid. Revil ransomware used advanced encryption techniques to encrypt victims' data, making it inaccessible until a ransom is paid. Grand Crab ransomware was distributed through various vectors, including exploit kits, spam emails, and compromised websites. Its RaaS model allowed a wider range of cybercriminals to deploy the ransomware. That's why the pre-detection of Android ransomware is necessary to protect users from it. The most common infection vectors for Android ransomware involve hiding it under legitimate applications by adding malicious code to them while retaining the original functionality of the legitimate application. Recently, malware authors developed ransomware apps by the name of popular applications and icons to keep them undetectable for malicious activities[4]. However, a general taxonomy of ransomware is shown in Figure 2.4:

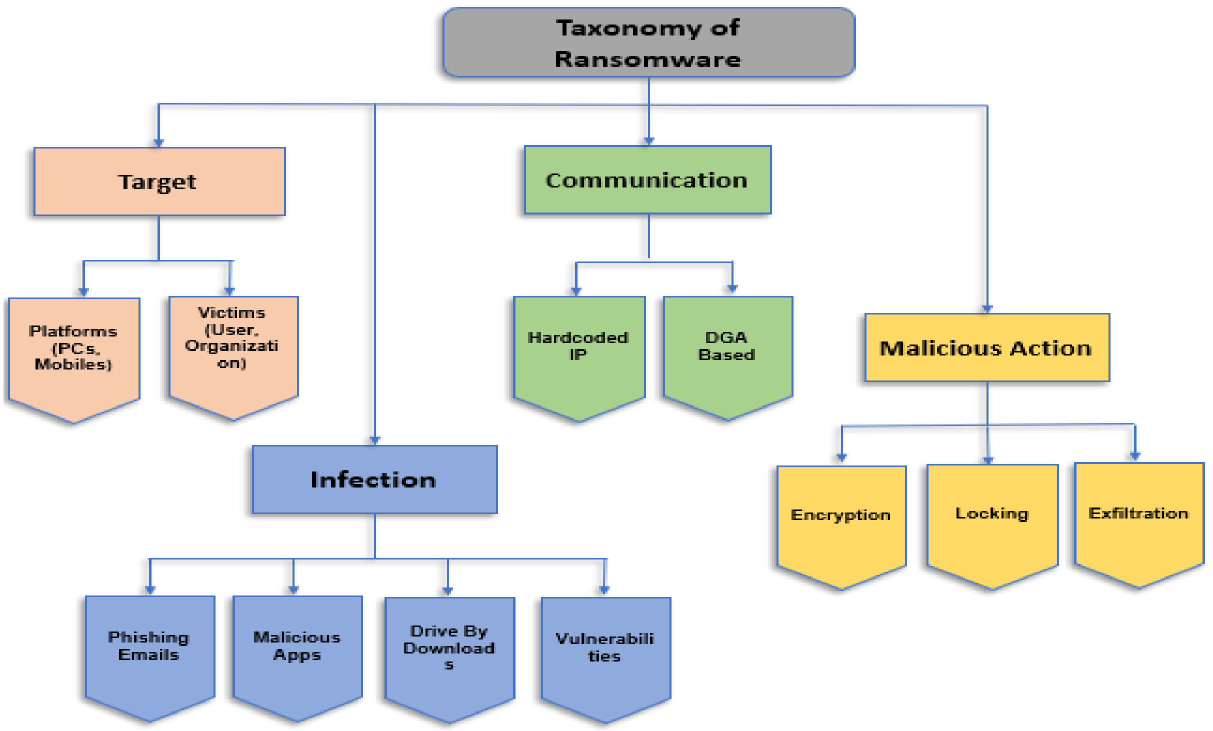


Figure 2.4-Taxonomy of Ransomware [1]

Some of the common commands used by ransomware on the Android platform are[4]:

- The phone's browser is redirected to an arbitrary URL
- You can send an SMS to either one or all of the contacts saved on the phone.
- Locking or unlocking of the device

- Snip received SMS and contacts [1]
- Update to a new version
- Displays a different ransom note [3]
- Enables or disables the internet connectivity.
- Tracks the target’s location.

The variants of Android ransomware which are used in our dataset include simplelocker, jisut, lockerpin, charger, koler, pletor, RansomBO, porndroid, svpeng, and wannalocker [3].

2.1.5 Android OS Architecture

In this study, we are targeting Android APKs ransomware. It is important to understand the Android OS architecture. In the Android OS architecture (as shown in Figure 2.5), when a user downloads an app for installation, the app requires permissions from the user before it can be installed on the device. Android permissions are well-known and creates interesting vulnerability possibility that can be exploited by malicious actors. Generally, the Android platform implements a permission-based protective model against malicious APKs to protect users. However, unnecessary, and irrelevant permissions can be grants and user are exploited for malicious activities. In order to distinguish between harmful and benign permissions, these permissions must be handled properly.

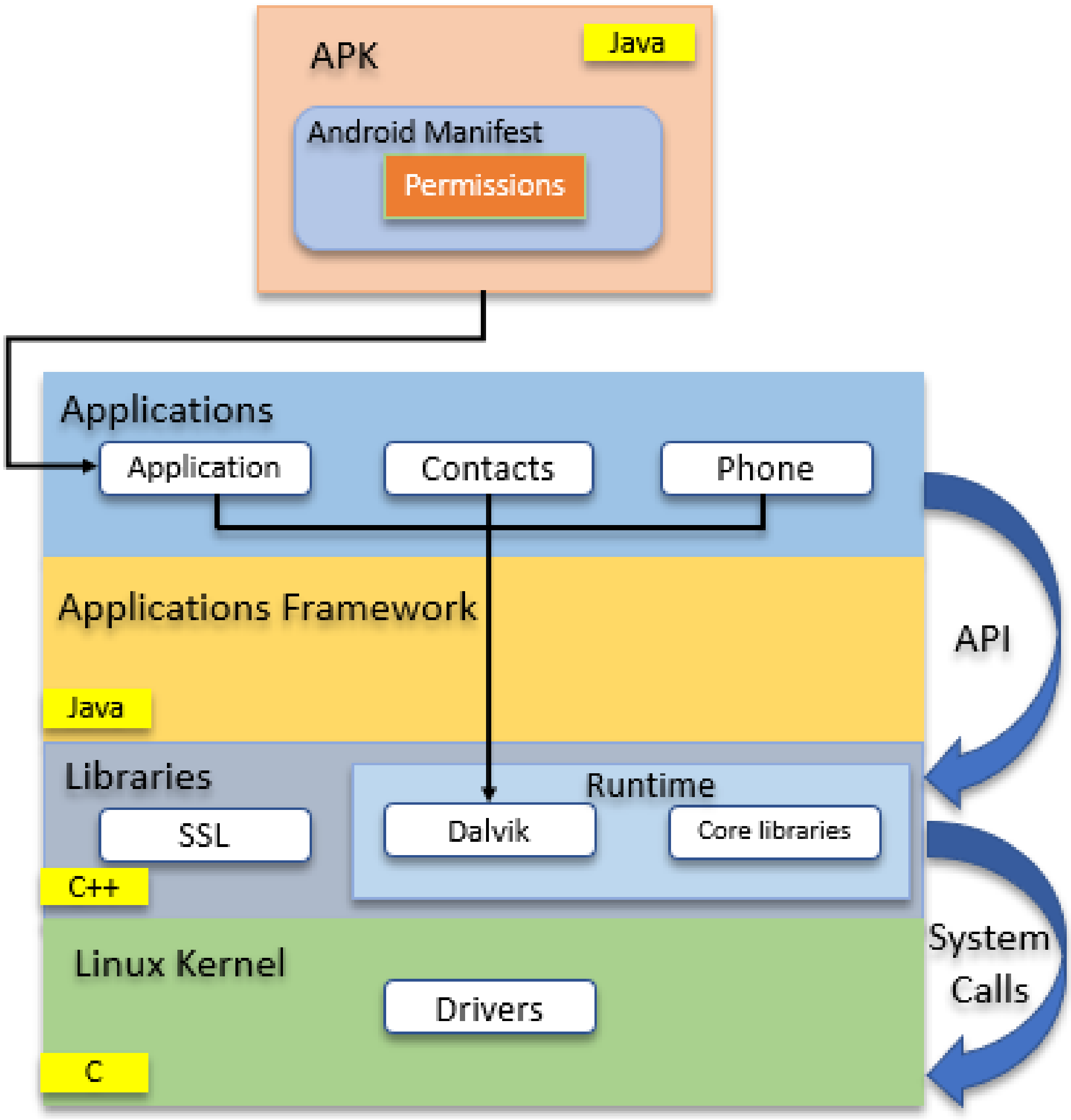


Figure 2.5-Android OS Architecture

In Android, to perform the different activities multiple methods in the code of APK needs to be called. Below is the flowchart that how the activity performed by an android APK is launched as shown in Figure 2.6. The figure 2.6 describes that when the activity is launched on android app, different methods accessed by the app to perform the functions, the user wants to be carried out. First method is onCreate () called, then onStart () method is accessed to begin the activity. But if any other activity comes in action the other activity gets resumed using onResume () method. When the user wants to end the activity onDestroy () method is called.

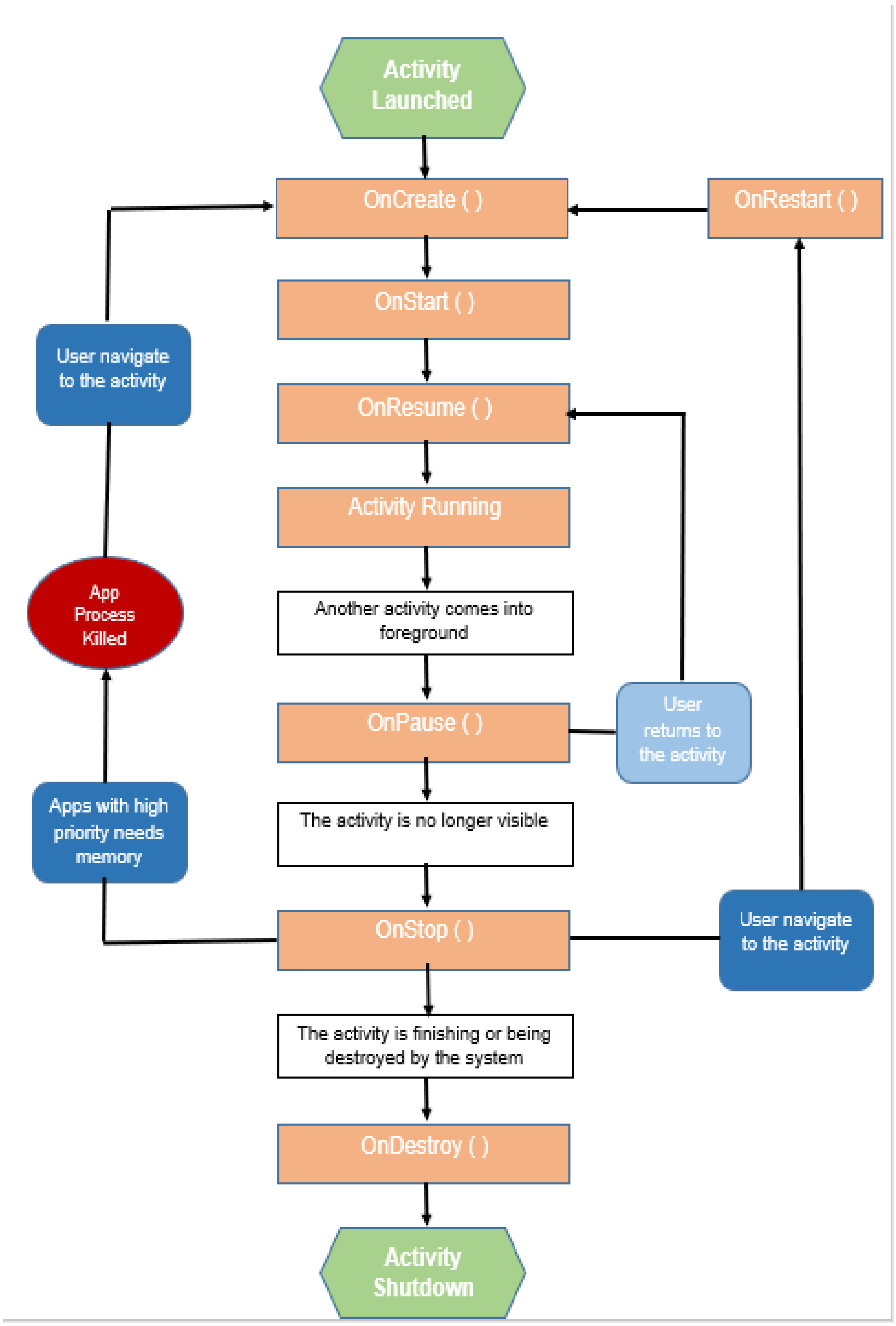

Activity Launched
OnCreate ( )
OnRestart ( )
OnStart ( )
User navigate to the activity
OnResume ( )
Activity Running
App Process Killed
Another activity comes into foreground
OnPause ( )
User returns to the activity
Apps with high priority needs memory
The activity is no longer visible
OnStop ( )
User navigate to the activity
The activity is finishing or being destroyed by the system
OnDestroy ( )
Activity Shutdown

Figure 2.6-Android App Activity Flowchart

2.1.6 Information extracted by using Static Analysis

APK (Android Package format) is the standard format for Android applications, which can be coded in various languages such as Java, Kotlin, C++, among others, although Java remains the official language. The Android SDK is used to compile all the application data and resource files, which are then combined into the APK file. This file contain all the components of an Android application and it is a zip-compressed file with an extension of .apk [42]. For installation of an Android App, it is used by the Android platform. By examining the APK files, one can observe that they have a directory structure similar to the one demonstrated in Table 2.1.

Table 2.1-APK File Structure - important locations for the analysis [42]

File	**Description**
assets/	Static files in APK
lib/	Native Library files which are program dependent
res/	Resources of the application
META-INF/	App’s signing certificates
AndroidManifest.xml	App’s global configuration file
classes.dex	DEX executable code files
resources.arsc	Resource configuration file in Binary Format

a) Assets

Static files are stored in the assets folder. Resource files can be organized into subdirectories of any depth, and users are free to arrange the file structure in any way they prefer. Generally, the Asset Manager class is used to access these assets.

b) lib

The folder designated for storing an application's dependent native libraries contains libraries that are specific to the Android platform and typically developed using C/C++. These libraries are sorted according to various CPU models like ARM, x86, MIPS, and so on [42].

c) res

This directory is designated for storing the resources related to the applications. Unlike the assets folder, the resources stored here are assigned a unique ID that is linked to the Android project's.R file, allowing them to be accessed directly using R.id.filename. The res directory includes various subdirectories, which are detailed in Table 2.2 along with their purposes.

Table 2.2-Directories inside lib folder

Sub Directory	Purpose
Anim	Keeping animation files
Drawable	Keeping Image resources
Layout	For layout files
Values	Keeping application values in separate files; i.e., colors.xml, dimens.xml, string.xml, styles.xml etc.
Xml	Used for keeping related xml files

d) META-INF

The META-INF folder stores the application's signature information, which can be used to verify its integrity at any time [42]. Android SDK generates this signing information during the packaging process and saves it in the META-INF folder.

e) AndroidManifest.xml

The AndroidManifest.xml file is a crucial configuration file for applications which provides comprehensive information, enabling the Android platform to comprehend the applications. It is a mandatory file for each application, and its name is fixed and cannot be changed. The AndroidManifest.xml file details an application's activities, services, providers, and receivers, as well as its name, package name, SDK version, and permissions [43].

f) classes.dex

The Android platform is modeled after Java's virtual machine and employs a similar strategy utilizing the Dalvik virtual machine, which is a streamlined version of the Java virtual machine. Typically, program code is transformed into bytecode, which is saved in a class file. The Dalvik virtual machine executes this Dalvik bytecode. The Android SDK relies on the dx tool to translate Java bytecodes into Dalvik bytecodes [42]. This tool has the ability to optimize code by consolidating, restructuring, and enhancing class files, resulting in smaller file sizes and faster runtimes when packaging Android applications.

g) resources.arsc

To locate resources based on their resource IDs, the Android platform uses the information stored in the resources.arsc file. This file maintains the association between the resource files and their corresponding resource IDs. As a result, when a resource file is required in the source code, it can be effortlessly retrieved using the findViewById() function.

2.1.7 Machine Learning

Machine learning is a type of technology programmed rather it makes computer systems capable to enhance their performance on a specific task by learning form data approach is used. It basically develops algorithms and models that are able to analyze and interpret large sets of data for patterns and relationships identification and further use this information to make predictions or take actions. Machine learning is useful in a variety of applications for example different images could be given as input data and then a random image could be recognized, speech could be recognized, recommendation systems, and predictive analytics [44].

2.1.7.1 Machine Learning Algorithms

This part describes the different ML algorithms such as Gradient Boosting, Decision Tree, Logistic Regression, Naïve Bayes, Support Vector Machines (SVM), Bagging Classifier, Random Forest and K-Nearest Neighbors in detail:

a) Decision Tree

A decision tree is a machine learning algorithm which serves for classification and regression. It is a tree-like model where each end denotes a feature or attribute, and each branch denotes a decision rule or condition based on that feature. The tree structure is built using a training dataset, where the algorithm selects the most informative feature at each node and splits the data into smaller subsets. Decision trees are popular due to their interpretability, ease of use, and ability to handle both categorical and continuous variables [45].

b) Random Forest

Random Forests utilize an amalgamation of classification/regression trees to generate models that can effectively represent nonlinear and multi-modal functions, resulting in a comparatively efficient learning process [44]. This collection of Random Forests comprises randomized decision trees, with decision trees serving as the root classifier. By using a bootstrap sample of data, each tree is created and overall features has random subset with respect to the features set for each division. This approach is renowned for creating a collective with low bias and variance [46].

c) Support Vector Machine (SVM)

SVM is a known as one of supervised learning algorithm used for classification and regression. The optimal hyperplane is used in a high-dimensional space by SVM to separate two data classes of data and the margin between the classes is maximized to achieve the results [44].

d) Logistic Regression

It is a supervised learning algorithm utilized to analyze the relatedness among an independent and dependent variable. This method is typically used for classification problems, with the aim of predicting the probability of a binary outcome, for example whether a consumer will purchase a product, based on the input variable values. The primary purpose of LR is to approximate the dependent variable probability using a statistical function known as a logistic function [47].

e) Naïve Bayes

This algorithm is designed for classification purposes. It is a type of Bayesian network which makes the assumptions that the class variable is available and feature variables are not dependent. This assumption enables it to effectively classify data in high-dimensional spaces while calculating the joint probability of a full variable set is complex. It is not preferable for directly estimating the class posterior because the unrealistic independence hypothesis can result in inaccurate probability [44].

f) Gradient Boosting

Gradient boosting is a technique that uses an iterative approach to construct an ensemble of weak predictive models, typically decision trees, so the errors generated by the previous model is assessed by each trained model. The algorithm determines the gradient of the loss function with respect to the current model's predictions at each iteration and then further uses it to train the next model. The final prediction is made by adjoining the results of all the models in the ensemble. Gradient boosting is well-suited for classification and regression tasks, particularly those with complex, non-linear connection between the features and the target variable. It is known for its high accuracy and flexibility in handling different types of data [47].

g) Bagging Classifier

The bagging classifier is a machine learning technique used for classification tasks that involves the grouping of multiple models qualified on multiple subsets of the training data. Bagging stands for Bootstrap Aggregating, and it works by randomly testing the training data with replacement to develop multiple subsets of the data. The bagging classifier is known for its ability to reduce overfitting and enhances the stability of the model by reducing the variance in the predictions. It is also able to handle high-dimensional data and noisy input [48].

h) KNN

K-nearest neighbors (KNN) is a type of non-parametric algorithm and in these algorithms the underlying distribution of the data is not used for making any expectations. In KNN, the algorithm classifies new data points by comparing them to the K closest training examples (i.e., the "neighbors") in the feature space. A majority vote of the classes of the K nearest neighbors is then actually responsible for class of new data point. KNN is familiar and used for its and ease of implementation and simplicity [49].

2.2 Related work under Static Analysis

This section will elaborate the existing malware and ransomware detection techniques. Generally, Android app files are compiled using the SDK, and Java language is used to compile these APKs. These APKs contain different files including Android manifest file, Java folder, and a resource folder. Malicious actors modify these files to add ransomware functionality. The malicious actors decompile the apps and ask for several permissions to perform the malicious actions [19]. Felt et al. [20] considered only two permissions for their detection model i.e., SMS and READ_PHONE_STATE, but these permissions are also accessed by benign applications. The Android operating system solely verifies whether developers have declared permissions in the Android manifest file and applies no security checks to the permissions requested by the apps. To address this, a framework (kirin) is developed that defined rules and requirements. The rules stated by Kirin framework are that the application must not contain these permissions related labels like SMS, ACCESS FINE LOCATION, INTERNET, PHONE_STATE, SHORTCUT, and AUDIO. An app will be identified as a malicious app if it tries to access to the mentioned six techniques [21].

Similarly, the DroidMat framework, proposed by Wu et. al. [22], extracts the permissions to prove that the malicious APKs requested more permissions than the benign ones. A DREBIN method, proposed by Arp et al. [23], detects Android malicious applications and identified that SEND_SMS permission is mostly used by major malicious applications for posting SMS to high-rate numbers. On the other hand, Sanz et al. [24] and Qiao et al. [25] worked on the occurrence of permissions asked by mobile applications. They concluded that INTERNET permission is the most commonly requested permission among both by the benign and malicious APKs. The majority of malicious APKs used SMS-related permissions to carry out malicious activities. Li et al. [26] also identified the set of frequently and rarely used permissions for the categorization of benign and malicious applications. Diamantaris et. al. [27] created a set of permissions that are necessary to be used by the main functions of Android applications and further discussed how these permissions are integrated with third-party libraries. Additionally, they identified a set of 30 permissions utilized by third-party and core libraries in Android applications. For Android malware detection a model is proposed by McDonald et al. [6], it uses machine learning and manifest permissions. In their study, 27 permissions were identified for generic Android malware identification. Although several detection models (Han et al.; Hasanabadi et al.; Zhang et al.; Alzaylaee et al.; McDonald et al.)

have been designed to detect generic Android malware, limited detection schemes are available for detecting Android ransomware.

Researchers have shifted their focus toward Android ransomware since 2015. An Android ransomware dataset was created by Andronio et al. [28] and analyzed the Android ransomware threatening text displayed on a mobile screen in English and Russian language. It is possible to exhibit the cautionary message in several languages, each with its own unique structural arrangement. Saracino et al. [13] inspected permissions and system calls for detection, while Mercaldo et al. [29] analyzed Java Bytecode to observe the behavior of ransomware on Android phones. Mairco et al. [30] detected Android ransomware based on Dalvik bytecode to observe invoke-type instructions. Gharib and Ghorbani et al. [7] considered analyzed text, permissions, logos, system, and API call sequences. While Ferrante et al. [8] emphasized the n-grams opcodes (n ¼ 2), memory, system calls, network traffic logs, and CPU usage by the Android ransomware for the detection. Chen et al. [9] developed a dataset that recorded the user's layout click based on widgets and movements, which was then used to detect Android ransomware.

Similarly, Alzahrani et al. [10] developed a hybrid model by comparing the app structure with known ransomware, identifying warning text, and analyzing images used for the research. Su et al. [11] detected locker ransomware by considering the text, permissions, window properties, and system commands. However, ransomware can evade detection by displaying text via images. Scalas et al. [12] used system API packages, classes, and methods as features. Lachtar et al. [31] extracted opcodes from instructions to use as a feature to detect Android ransomware. Alzahrani et al. [14] analyzed permissions, intents, and API calls for mobile ransomware detection. Bibi et al. [32] suggested a dynamic mechanism which is comprised of 19 important features related to network headers and packets to identify Android ransomware. Alsoghyer et al. [15] suggested a permission-based detection model and identified 115 malicious permissions for Android ransomware detection. Sharma et al. [5] proposed a detection model by using analyzed text, permissions, images, and Java files. Abdullah et al. [33] described a dynamic detection system that detects Android ransomware by using system calls. From the aforementioned techniques of ransomware detection using different feature sets and detection accuracies are shown in Table 2.3-A.

Table 2.3-A Comparison of Ransomware Detection Techniques

Related Work	**Framework**	**Analysis Technique**	**Feature Set**	**Machine Learning Technique**	**Permissions used**	**Accuracy**
Andronio et al. [28], 2015	HelDroid	Static	Text, Locking, Encryption	Natural Language Processing	No	94%
Saracino et al. [13], 2016	Madam	Static, Dynamic	Permissions, System calls	KNN	Yes	96.9%
Mercaldo et al. [29], 2016	R-inside out	Static	Java Bytecode	CWNC model	No	Not stated
Mairco et al. [30], 2017	R-PackDroid	Static	Invoke-type instructions	RF	No	Not stated
Gharib and Ghorbani et al. [7], 2017	DNA-Droid	Static, Dynamic	Text, Images, API, Instructions	NB, RF, SVM, Adaboost, DNN	No	Not stated
Ferrante et al. [8], 2017	Extinguish Ransom	Static, Dynamic	Memory, System calls, Logs	DT, NB, LR	No	100%
Chen et al. [9], 2017	RansomProber	Static, Dynamic	Widgets, Activities	User interface, User Clicks	No	99%
Alzahrani et al. [10], 2018	RAndroid	Static, Dynamic	Text, Images	Not stated	No	91%

[11], 2019	Locker	Static	Text, Commands, Background operations (permissions, Priority, admin, window, system)	LR, RF, DT, SVM	Yes	99.98%
Scalas et al. [12], 2019		static	API packages, Classes, Methods	RF	No	97%
Lachtar et al. [31], 2019	Native instructions based	Static	A dictionary contains unique opcodes present	RF, SVM, KNN, ANN	No	99.8%
Alzahrani et al. [14], 2019	API-Based	Static	34 API, 48 Permissions, 4 Intents	KNN, LR, SVM, RF	Yes	97.62%
Bibi et al. [32], 2019	Mutifactor feature filtration and recurrent neural network	Dynamic	Network packets, Headers	LSTM	No	97.08%
Alsoghyer et al. [15], 2020	Permissions based	Static	115 Permissions are used	RF, J48, SMO, NB	Yes	96.9%
Abdullah et al. [33], 2021	System calls based	Dynamic	52 System calls are used	RF,J48,NB	No	98.31%
Sharma et al. [5], 2021	GPU-Based	Static	Permissions, Text, Images, JAVA files	LR, ANN, SVM	Yes	98.04%
Sifat et al. [34], 2023	Traffic analysis based	Dynamic	Traffic Analysis	Ensemble Learning	No	74.09%

*KNN: K-Nearest Neighbors, CWNC: Concurrency Workbench of new century RF: Random Forest, NB: Naïve Bayes; SVM: Support Vector Machine, DNN: Deep Neural Network, DT: Decision Tree, LR: Logistic

Regression, ANN: Artificial Neural Network, LSTM: Long Short-Term Memory, J48: Decision tree, SMO: Sequential minimal Optimization

As shown in Table 2.3-A, most of the existing malware detection models utilized permissions along with other features. While permission-based detection is efficient for the pre-detection of malware, there is a need for Android-based models that employ an effective set of permissions and achieve high detection accuracy through machine learning algorithms. To best of our knowledge, there is lack to detect the android based ransomware APKs using permission. To address this gap, this study proposes a ransomware Android detection model that utilizes a significant set of permissions to enhance detection accuracy.

2.3 Summary

The chapter discussed an outline of the background and related work of the thesis, including related literature and a critical analysis of the existing studies. By examining previous research and methods used in the literature, it aids in the formulation of a solution to the identified problem. The succeeding chapter will delve into the research methodology utilized throughout the thesis.

3. Research Methodology

In this chapter, we will explain the methodology which is followed to carry out this thesis research. A brief description of the methods that are used in our research methodology along with the phases followed in the research process, i.e., samples collection, feature set construction, and malware detection using machine learning classifiers are given in this chapter.

3.1 Introduction

Research refers to the systematic gathering and analysis of data related to a particular field of study. Its primary objective is to uncover new knowledge and facts through a scientific investigation [50].. In the words of Clifford Woody, the research process involves identifying and redefining the problem, developing a hypothesis and potential solutions, collecting, and evaluating data, making assumptions, drawing conclusions, and testing the hypothesis to confirm its validity [52].

3.1.1 Overview of Research Methodology

In this part, we will elaborate the proposed work that aims to create a minimal permissions-based Android Ransomware detection model. To create a minimal permissions-based Android ransomware detection model, we collected ransomware APKs from the Canadian Institute of Cyber Security [17] and benign APKs from Androzoo [18]. We decompiled these files using the APK tool and Android Studio to extract permissions, which were filtered using the feature importance technique to create a dataset for machine learning models. Multiple machine learning models were utilized to determine the most important permissions that can differentiate ransomware from benign applications. These permissions were compared with those used to detect ransomware and generic Android malware. For training and testing, we employed Decision Tree, Random Forest, SVM, Logistic Regression, Naïve Bayes, Gradient Boosting, Bagging and KNN classifiers.

The proposed RansomShield model consists of the following components:

- Collect malicious and benign samples and evaluate their effectiveness using VirusTotal.
- Creating a dataset by considering the permissions as features and identifying the most relevant features.
- Employed machine-learning algorithms to construct predictive models.
- Analyzing the effect of permission on the detection of ransomware and benign apps.
- Applying supervised ML algorithms for classification and selecting the best model.

3.2 Thesis Research Methodology

The proposed RansomShield model is shown in Figure 3.1. The model shows the stages of collecting, decompilation, data cleansing, dataset creation, selecting significant permissions, and then classifying carried out for the identification of Android ransomware. The key components are discussed in the below topics.

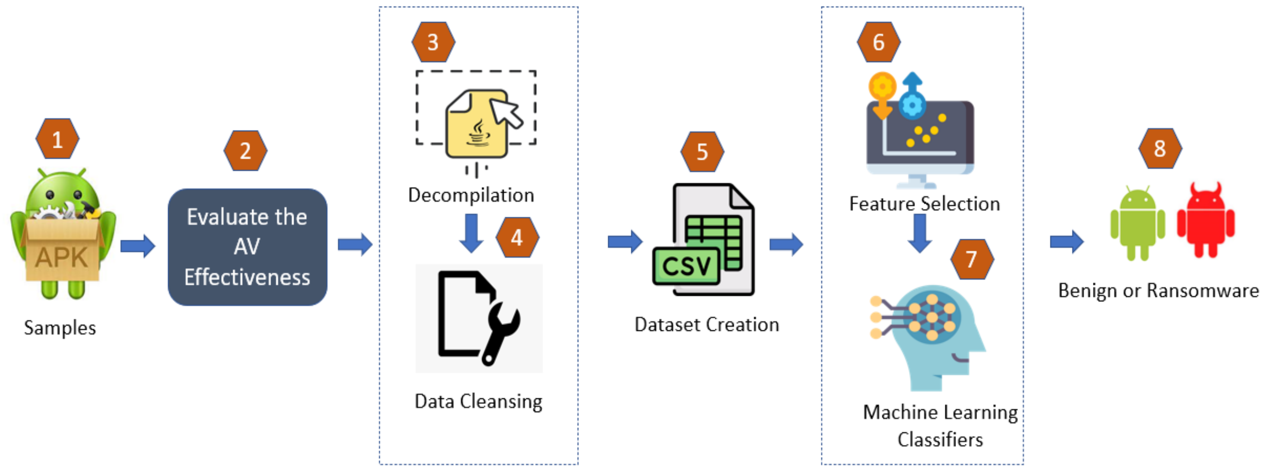
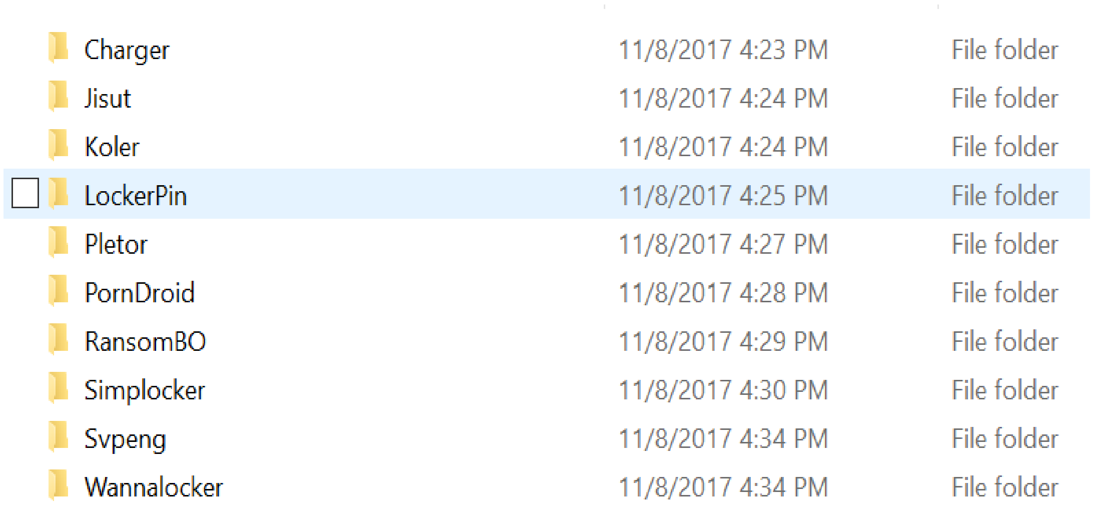


Figure 3.1-Proposed Model Flow Diagram: RansomShield

Firstly, for the collection of Android samples, we collected benign and ransomware sample applications. The legitimate applications gathered from Google Play Store and Androzoo dataset [18]. For AndroZoo dataset, a script written in Python is used along with the API key provided by AndroZoo. However, for Android ransomware samples CICAndMal2017 dataset [17] is employed. To appraise the usefulness of the proposed methodology, and for orderly data training; cross-validation experimentations are performed for these samples. The ransomware samples used for the research belonged to 11 different variants of ransomware as shown below in Figure 3.2:



Charger	11/8/2017 4:23 PM	File folder
Jisut	11/8/2017 4:24 PM	File folder
Koler	11/8/2017 4:24 PM	File folder
LockerPin	11/8/2017 4:25 PM	File folder
Pletor	11/8/2017 4:27 PM	File folder
PornDroid	11/8/2017 4:28 PM	File folder
RansomBO	11/8/2017 4:29 PM	File folder
Simplocker	11/8/2017 4:30 PM	File folder
Svpeng	11/8/2017 4:34 PM	File folder
Wannalocker	11/8/2017 4:34 PM	File folder

Figure 3.2-Android Ransomware APKs

3.2.1 Constructing the Features Set

To construct and classify ML models, the initial stage involves gathering essential characteristics from a dataset. These features are typically listed in the AndroidManifest.xml file of an Android application package (APK) including requested permissions from an application. Proposed approach extracts features by utilizing the APK tool to decompile ransomware applications from the CICAndMal2017 dataset [17]. Our primarily emphasis is on permissions to compile a list of feature sets for conducting static analysis and to gain a thorough understanding of each app's behavior.

3.2.2 Parameter Feature Set Construction

To create the dataset, we have used various tools for the static analysis of applications. Firstly, we employed Android Studio to obtain the permissions from the Android Manifest.xml file. Additionally, we employed the APK tool for the decompilation of Android applications, as it is a command line tool also known for reverse engineering applications. We created the dataset based on the permissions requested by applications. We also installed MobSF (static analysis) and analyzed the applications in it. To study the other features, grasp the behavior of each app and request permissions description as well to select the appropriate applications for the dataset. We extracted and considered each permission requested by the applications. Further reduced the permissions systematically considering the influence of permissions among them. Figure 3.2 described the strategy that we implemented for our proposed research. The figure 3.3 below summarized the proposed research methodology into five steps that are data collection, dataset construction, deep analysis, constructing the model and evaluate that model by applying different machine learning algorithms.

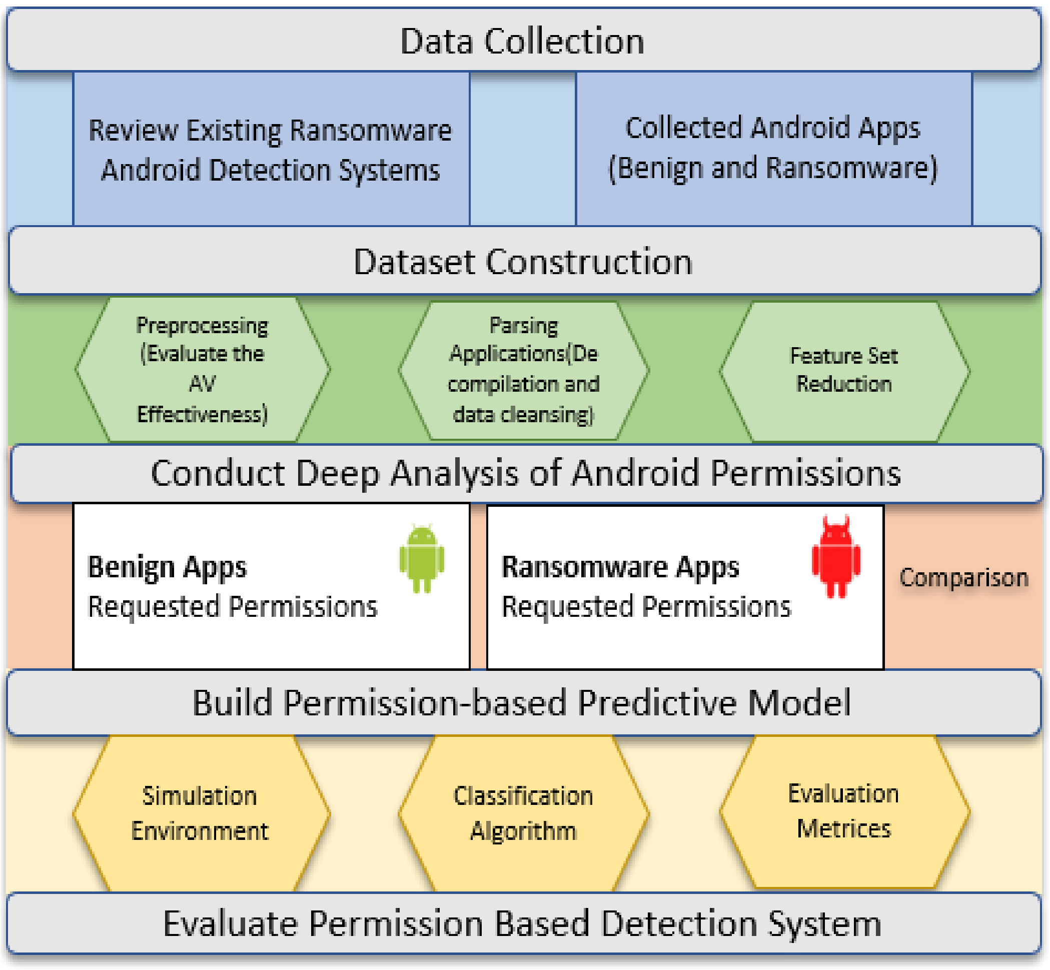

Data Collection
Review Existing Ransomware Android Detection Systems
Collected Android Apps (Benign and Ransomware)
Dataset Construction
Preprocessing (Evaluate the AV Effectiveness)
Parsing Applications(De compilation and data cleansing)
Feature Set Reduction
Conduct Deep Analysis of Android Permissions
Benign Apps Requested Permissions
Ransomware Apps Requested Permissions
Comparison
Build Permission-based Predictive Model
Simulation Environment
Classification Algorithm
Evaluation Metrices
Evaluate Permission Based Detection System

Figure 3.3-Proposed Research Methodology

3.2.3 Extraction of Core Features

After constructing the dataset, the next step was to extract the core features required to differentiate the ransomware apps from the benign apps. For choosing the effective permissions, we have explored and evaluated the risky permissions by Google, Zhu et al. [35], RansomProber [9], and RansomAnalysis [7] for Android generic malware and ransomware detection as shown in Table 3.1. It shows the dangerous permissions identified by the above-mentioned models:

Table 3.1-Comparison of Permissions Identified by different models

Permissions	**Google**	**Sun et al. [26] 2016**	**Chen et al. [9] 2017**	**Zhu et al. [16] 2018**	**Chen et al. [9] 2017**	**Sharma et al. [5] 2021**
	General Malware Permissions				Ransomware Permissions	
READ_PHONE_STATE	✓	✓	✓		✓	✓
WRITE_EXTERNAL_STORAGE	✓		✓	✓	✓	✓
ACCESS_COARSE_LOCATION	✓		✓			
ACCESS_FINE_LOCATION	✓		✓			
RECORD_AUDIO	✓					
READ_EXTERNAL_STORAGE	✓	✓			✓	
SEND_SMS	✓	✓	✓	✓	✓	
CAMERA	✓	✓			✓	
RECEIVE_SMS	✓		✓	✓	✓	
GET_ACCOUNTS	✓				✓	
READ_SMS	✓	✓	✓		✓	
READ_CONTACTS	✓	✓	✓		✓	
ACCESS_NOTIFICATION_POLICY	✓					
WRITE_CONTACTS	✓	✓				
READ_CALENDAR	✓					
READ_CALL_LOG	✓	✓				
WRITE_CALENDAR	✓					
INSTALL_PACKAGES	✓	✓		✓		
SET_ALARM	✓			✓		
BODY_SENSORS	✓					
WRITE_SECURE_SETTINGS	✓			✓		
WRITE_CALL_LOG	✓					
UPDATE_DEVICE_STATS	✓			✓		
READ_HISTORY_BOOKMARKS	✓	✓	✓	✓		
WRITE_HISTORY_BOOKMARKS	✓		✓	✓		

RECEIVE_BOOT_COMPLETED		✓	✓		✓	✓
INTERNET			✓		✓	✓
ACCESS_NETWORK_STATE			✓		✓	✓
ACCESS_WIFI_STATE		✓	✓		✓	
VIBRATE			✓		✓	
WAKE_LOCK			✓		✓	✓
INSTALL_SHORTCUT			✓			
WIRTE_SMS			✓			
CALL_PHONE			✓			
READ_SETTINGS			✓			
GET_TASKS		✓	✓		✓	✓
KILL_BACKGROUND_PROCESSES					✓	✓
SYSTEM_ALERT_WINDOW		✓			✓	✓
DISABLE_KEYGUARD		✓			✓	✓
CHANGE_WIFI_STATE		✓			✓	
WRITE_SETTINGS		✓			✓	
CHANGE_NETWORK_STATE		✓				
READ_LOGS		✓				
RESTART_PACKAGES		✓				
SET_WALLPAPER		✓				
WRITE_APN_SETTINGS		✓				
TOTAL PERMISSIONS	26	22	21	9	21	10

Table 3.1 presents the permissions that have been filtered out by current reverse engineering and machine learning techniques for detecting generic Android malware and ransomware. Some of these permissions are commonly in detecting Malwares. We have chosen a significant set of features by disregarding the permissions that are less effective for detection. To create the dataset, we initially considered 158 permissions that have been requested by both benign and Android ransomware applications. Alzahrani et al. [14] employed 48 permissions for

identifying Android ransomware. However, it did not specify which permissions are used. On the other hand, in the proposed strategy, we have reduced permissions up to 16 and improved accuracy. Further, we employed the Random Forest feature importance characteristic to evaluate the importance of permissions. The permissions we have selected for our proposed model are listed in Table 3.2 along with their importance. The proposed permissions set is as follows:

Table 3.2-Description of proposed Permissions

S.no:	**Proposed Permissions**	**Feature Importance**	**Description**
1.	SEND_SMS	0.07024	To send messages and cost the money of those messages without confirmation
2.	READ_PHONE_STATE	0.03367	To access the phone features of the device.
3.	READ_SMS	0.03100	To read the SMS from the phone and SIM card.
4.	READ_CONTACTS	0.02655	To read all the contact data saved on the device.
5.	RECEIVE_BOOT_COMPLETED	0.16460	To check when the device boots up
6.	INTERNET	0.06132	To open the network sockets
7.	ACCESS_WIFI_STATE	0.02021	To view the information about status of Wi-Fi networks.
8.	WAKE_LOCK	0.05255	To keep the device screen on and stop it from sleeping.
9.	GET_TASKS	0.08072	The currently and recently running tasks information can be accessed through this permission.
10.	SYSTEM_ALERT_WINDOW	0.27866	To take over the entire screen window
11.	ACCESS_NETWORKS_STATE	0.01802	To get information about status of networks.

12.	READ_EXTERNAL_STORAGE	0.04895	It allows to read from external storage
13.	READ_SETTINGS	0.03116	To view the settings
14.	MODIFY_AUDIO_SETTINGS	0.02286	To change global audio settings like routing and volume.
15.	RECEIVE	0.02771	View the receiving settings
16.	BILLING	0.03178	View the billing related information

The feature importance graph generated for our proposed 16 permissions by Random Forest is shown in Figure 3.4. The graph below shows the importance percentage of each identified permission for ransomware detection.

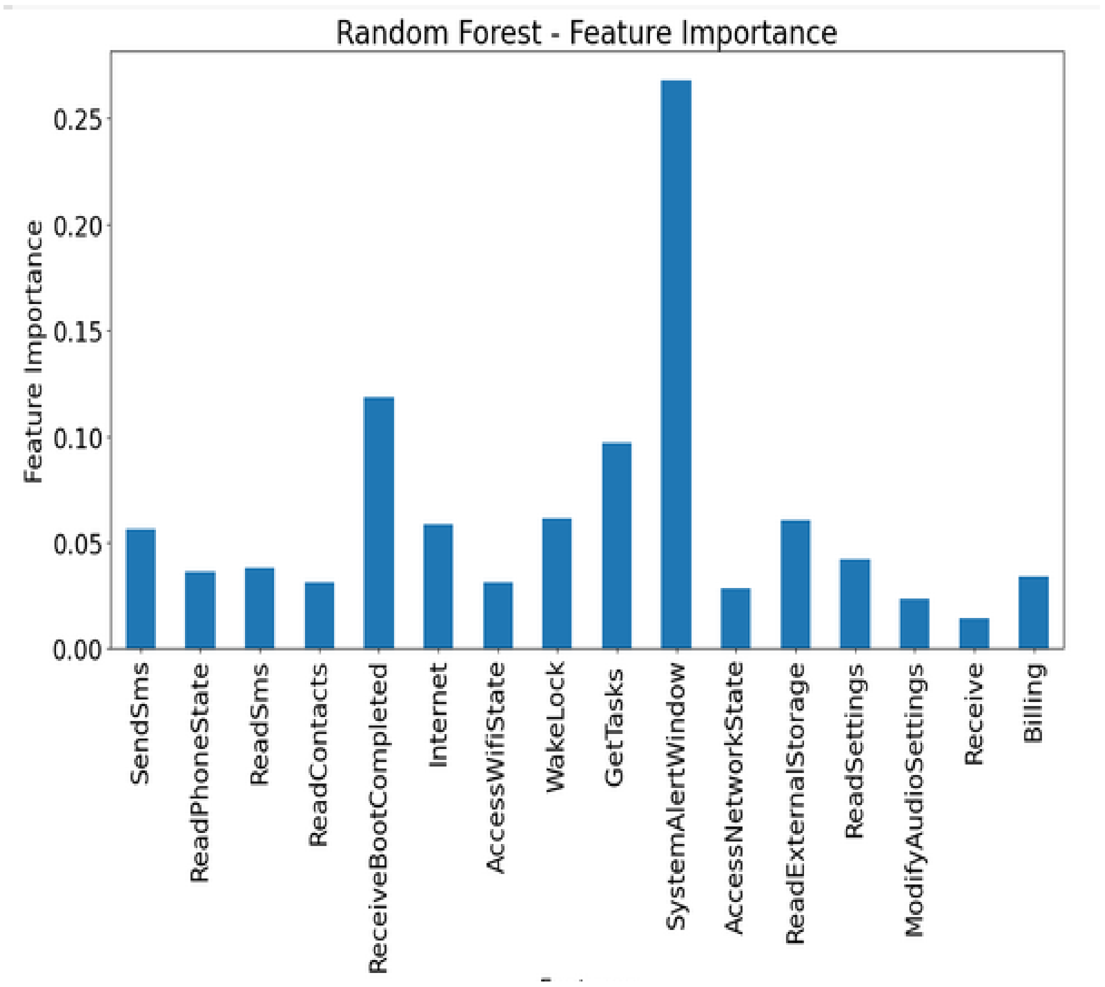


Figure 3.4-Permission Importance Graph

3.2.4 Dataset Creation

The permission information is interpreted into the 0's and 1's format. The 0's shows the app denying permission and 1's indicate the requesting permission. The permission set selected from benign and ransomware APKs represented in 0's and 1's are merged to create a single dataset for analysis. Many detection models use extensive number of permissions i.e., 115 permissions are proposed in [15]. Our proposed model dataset filtered the minimum number of features for ransomware APKs detection. The comparison of selected features of our proposed research are shown in Table 3.3:

Table 3.3-Comparison of the proposed features with Sharma et al. [5] and Chen et. al. [9] methods' features.

Permissions/Features	**Chen et al. [9], 2017**	**Alzahrani et al. [14], 2019**	**Alsoghyer et al. [15], 2020**	**Sharma et al. [5], 2021**	**Proposed Method**
READ_PHONE_STATE	✓		✓	✓	✓
WRITE _EXTERNAL_STORAGE	✓		✓	✓	
READ_EXTERNAL_STORAGE	✓				✓
SEND_SMS	✓		✓		✓
CAMERA	✓				
RECEIVE_SMS	✓				
GET_ACCOUNTS	✓				
READ_SMS	✓				✓
READ_CONTACTS	✓				✓
RECEIVE_BOOT_COMPLETED	✓		✓	✓	✓
INTERNET	✓		✓	✓	✓
ACCESS_NETWORK_STATE	✓		✓	✓	✓
ACCESS_WIFI_STATE	✓				✓
VIBRATE	✓				
WAKE_LOCK	✓		✓	✓	✓
GET_TASKS	✓			✓	✓
KILL_BACKGROUND_PROCESSES	✓		✓	✓	
SYSTEM_ALERT_WINDOW	✓		✓	✓	✓
DISABLE_KEYGUARD	✓			✓	
CHANGE_WIFI_STATE	✓				
WRITE_SETTINGS	✓				

READ_SETTINGS					✓
MODIFY_AUDIO_SETTINGS					✓
RECEIVE					✓
BILLING					✓
BIND_DEVICE_ADMIN			✓		
Other (…)		✓	✓		
Total Permissions	**21**	**48**	**115**	**10**	**16**
Other properties					
Intents		✓		✓	
Images				✓	
Text				✓	
Lock				✓	
Encrypt				✓	
Encode				✓	
API Calls		✓			
Accuracy	99%	96.9%	97.62%	98%	97%

Table 3.3 shows the comparison of the permissions identified by different models. Our model identified 16 permissions with high detection accuracy. However, Sharma et al. [5] identified 10 permissions but also relied on other features such as intents, images, text, lock, encrypt, and encode. Similarly, Alzahrani et al. [14] identified 48 permissions, 4 intents and 34 API calls for ransomware detection. However, permissions are not mentioned in their study. While Alsoghyer et al. [15], 2020 identified 115 permissions for detection. Chen et al. [9] identified 21 permissions while achieving the highest detection accuracies. However, the proposed model employed 28% fewer permissions as compared to Chen et al. [9] while compromising the detection accuracy of 2%. Our proposed model main aim is to reduce the permissions for ransomware detection.

3.2.5 Ransomware Detection by Employing Machine Learning Models

Within this section, we utilize classifiers to accurately detect ransomware applications while minimizing false positive results. Our dataset comprises ransomware samples extracted from the CICAndMal2017 dataset. For training purposes, we allocate 80% of the samples, employing eight distinct classifiers: Decision Tree, Random Forest, SVM, Logistic Regression, Naive Bayes, Gradient Boosting, Bagging, and KNN classifiers. Subsequently, we employ the remaining 20% of the samples to evaluate the models' performance and compare the obtained outcomes.

This section is comprised of applying supervised machine learning classifiers. The proposed model is consisting of two parts: the first part is the creation of the dataset and selection of the appropriate features (permissions) and the second part comprises the preparation and authentication of the supervised learners with various ML algorithms. The proposed dataset contains an equal ratio of benign and ransomware. Android APKs. Google Collab platform in which Python scripts and split methods are used for training and dataset testing.

3.3 Summary

Within this chapter, various methodologies have been explored which have been utilized in prior research and can be replicated to obtain comparable outcomes. The overarching approach entails acquiring datasets from reliable sources, culling and pinpointing the relevant features, and ultimately employing various Machine Learning techniques for classification purposes. Moving forward, the subsequent chapter will delve into the experimental configurations, and which is formulated for conducting this specific analysis.

4. Experimental Setup

The present chapter outlines the experimental arrangement which has been devised for establishing an appropriate research environment. It also rationalizes the choice of certain employed methodologies. Furthermore, this chapter furnishes the specifications of the system configurations.

4.1 Overview

To undertake the experimental analysis, a setup akin to the one employed by H. J. Zhu et. al. [16] has been adopted. The experimental setup encompasses an android dataset that comprises both ransomware and benign applications. Additionally, a PC is utilized to execute a Python-based codebase, responsible for generating the feature set and conducting the evaluation process.

4.2 Setting up Environment

To facilitate the experimentation process, a machine operating on the Windows platform is used. The system specifications are presented in the table provided below, denoted as Table 4.1.

Table 4.1-System specifications

Property	**Description**
Manufacturer	DELL
Model	Dell
Architecture	x64 based
Operating System	Windows 10 Pro
Processor	Intel(R) Core(TM) i7-8650U CPU @ 1.90GHz 2.11 GHz

RAM	16 GB
Storage	500 GB

4.3 Constructing Android Ransomware Dataset

To construct the dataset, we gathered various samples of Android applications from two distinct sets: ransomware and benign. The benign samples are selected from the Androzoo dataset [18], encompassing applications from diverse categories such as business, entertainment, finance, and games. This approach aimed to ensure a wide range of diversity within the samples.

Regarding the ransomware samples, multiple options were available. However, we opted to utilize the CICAndMal2017 [17] dataset due to the Canadian Institute of Cybersecurity's reputation for providing up-to-date collections of malware samples. Accessing these datasets required signing up on the CIC website [https://www.unb.ca/cic/datasets/index.html] and providing consent to acknowledge the associated risks of downloading and utilizing the dataset.

4.4 Environmental Setup

Once the samples are collected, we set up an environment to analyze the Android ransomware samples. For setup, experiments are executed on Windows 10 host machine running VBOX having a Windows 10 machine installed. Inside the VBOX Windows 10, we installed Android Studio, APK tool, and MobSF for static analysis of applications. The collected ransomware samples are taken from CICAndMal2017 dataset. The description of tools used are described in Table 4.2 as:

Table 4.2- Tools Description

S.No:	**Tool Name**	**Description**
1.	Android Studio	It provides the integrated environment for development of applications. It also analyzes the app statically. For dynamic analyzation, genny motion must be integrated with it.
2.	APK Tool	It is the best tool for reverse engineering of applications. it decodes the app almost near to original form. Different static features can be extracted out through it.
3.	MobSF	It is an automated tool for pen testing, malware analysis and security assessment of mobile applications.

4.5 Evaluation of APKs through VirusTotal

After the collection of malware samples, we employed the VirusTotal [35] to distinguish the regular and ransomware malware by adding its signature. The VirusTotal API key can also be used for uploading the APK files to it. All the applications are submitted to the VirusTotal to verify the calculated score by different anti-virus engines.

4.6 Downloading the Python Codebase

The code base has been developed entirely from scratch utilizing the Python programming language. The code for this research can be seamlessly incorporated into the experimentation environment.

4.7 Installing Pre-requisite Software

Before proceeding with the experimentation process, it is necessary to install certain prerequisite software. The following software are required:

1. An archiving tool like WinRAR; it is used for extracting application samples. You can download it from the following link: WinRAR Download
2. Python interpreter, preferably version 3.8 or higher. This software is required for tasks such as extracting applications, generating feature sets, and performing classifications. You can download the Python interpreter from the following link: Python 3.8+ Download
3. If there is a need to modify the codebase or customize the feature set, any Python Integrated Development Environment (IDE) can be utilized, such as IntelliJ IDEA. You can download IntelliJ IDEA from the following link: IntelliJ IDEA Download

4.8 Summary

Within this chapter, we have presented the proposed experimental setup designed for conducting the analysis. The chapter encompasses discussions on the collection of the necessary dataset, establishment of the essential environment, and the accompanying codebase. Additionally, comprehensive information regarding the installation of prerequisite software for the analysis process, including their respective sources, have been provided in this section.

5.Experimental Results

In this chapter, the obtained results and their analysis is presented in the form of classification outcomes. Additionally, the accomplishments of the research are extensively discussed within this chapter.

5.1 Overview

Permissions are utilized by Android applications to offer various functionalities to users, but unfortunately, malware developers exploit these permissions for malicious purposes. This study conducts a comprehensive analysis on an Android dataset containing both benign and ransomware applications. The subsequent discussion focuses on the utilization of different evaluation metrics during the analysis to gauge the efficacy of the approach.

5.2 Evaluation Measures

To assess the performance, the evaluation utilizes the metrics of Sensitivity, Precision, Accuracy, Area under Curve (AUC), and the Receiver Operating Characteristic (ROC). Accordingly, the following formulas depict their respective information.

1) $Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$

2) $Sensitivity = \frac{TP}{TP + FN}$

3) $Precision = \frac{TP}{TP + FP}$

4) $F1 - Score = \frac{TP}{TP + \frac{1}{2}(FP + FN)}$

In the context of these metrics, the true positive (TP) signifies the samples that are correctly anticipated as positive based on count of positive testing. Conversely, false positive (FP) refers to samples that are incorrectly anticipated as positive based on the count of negative testing. Similarly, true negative (TN) denotes the count of negative testing samples that are accurately anticipated as negative, while false negative (FN) represents the count of positive testing samples that are falsely anticipated as negative.

5.3 Evaluating Research Effectiveness

To evaluate the efficacy of RansomShield, we conducted over 20 iterations on the feature dataset. After reducing the requested permissions by the samples, we identified the 16 most significant permissions out of the initial 158. Additionally, we calculated the performance metrics values for each classifier of the machine-learning algorithm to compare their results. To calculate the performance of Chen et. al. method on our proposed dataset with its 21 features, we have observed a similar detection accuracy as compared to our proposed model with 16 permissions. Chen et. al. detection accuracies were around 98% for decision tree, 97 % for random forest, 93% for logistic regression, 97% for SVM, 74% with Naïve Bayes, 97% with KNN, 94% with Bagging, and 100% with gradient boosting classifiers. Table 5.1 shows the performance metrics values of each classifier for the proposed Ransom Shield.

Table 5.1-The performance comparison of the proposed RansomShield and existing methods

S. No:	**ML Classifiers**	**Accuracy (proposed model) Permissions=16**	**Precision**	**Recall**	**F1 Score**	**K-Fold Validation**	**Accuracy (Reference Approach) Permissions=21**
1.	Decision Tree	95%	95%	96%	95%	93.5%	98%
2.	Random Forest	97%	98%	98%	98%	97%	97%
3.	Logistic Regression	95%	95%	96%	95%	88.5%	93%

4.	Bagging Classifier	94%	98%	98%	985	92.9%	94%
5.	Support Vector Machine	97%	98%	98%	98%	93%	97%
6.	K-Nearest Neighbor	97% (knn=1) 97% (knn=5)	98% 98%	98% 98%	98% 98%	95% (knn=1) 92% (knn=5)	97%
7.	Gradient Boosting	100%	100%	100%	100%	94.6%	100%
8.	Naïve Bayes	73.8%	74%	77%	73%	75%	74%

Our experimental results showed the highest detection accuracy is 100% with gradient boosting. As Gradient boosting is known for its effectiveness and flexibility due to the requirement of weak subtrees in the ensemble. This allows gradient descent to efficiently steer towards a favorable solution. Consequently, gradient boosting often delivers impressive results without the need for extensive tuning. That’s why the gradient boosting showed highest accuracy for ransomware detection. After gradient boosting, Random Forest, SVM and KNN showed 97% detection accuracy. The accomplishment of these algorithms is influenced by components such as the dataset's quality and characteristics, feature engineering techniques, hyperparameter tuning, and various other factors. The choice of the most appropriate algorithm is based on its own advantages and limitations, and it may vary for a specific problem. While Naïve bayes showed 73.8% detection because in reality, the assumption of feature independence, which is inherent in the naive Bayes algorithm, is often invalid. In return, the accuracy of naive Bayes tends to be lower compared to more sophisticated algorithms.

In order to understand how well a classification model, accomplish across all classification thresholds, ROC (Receiver Operating Characteristics) curve is used. ROC curve illustrates how well a classification model performs across all classification thresholds. One can create a graph by varying classification thresholds and plotting the false positive rate (FPR) against the true positive rate (TPR), the ROC curve provides a visual representation of the model's performance. If the threshold of classification is decreased, items will be increased in category of positive, leading to rise in both false positives and true positives. In contrast, AUC curve is the area under the curve. It is an estimate of performance across all the classification thresholds. The ROC and AUC graphs for machine learning classifiers can be seen in Figure 11 for the proposed model.

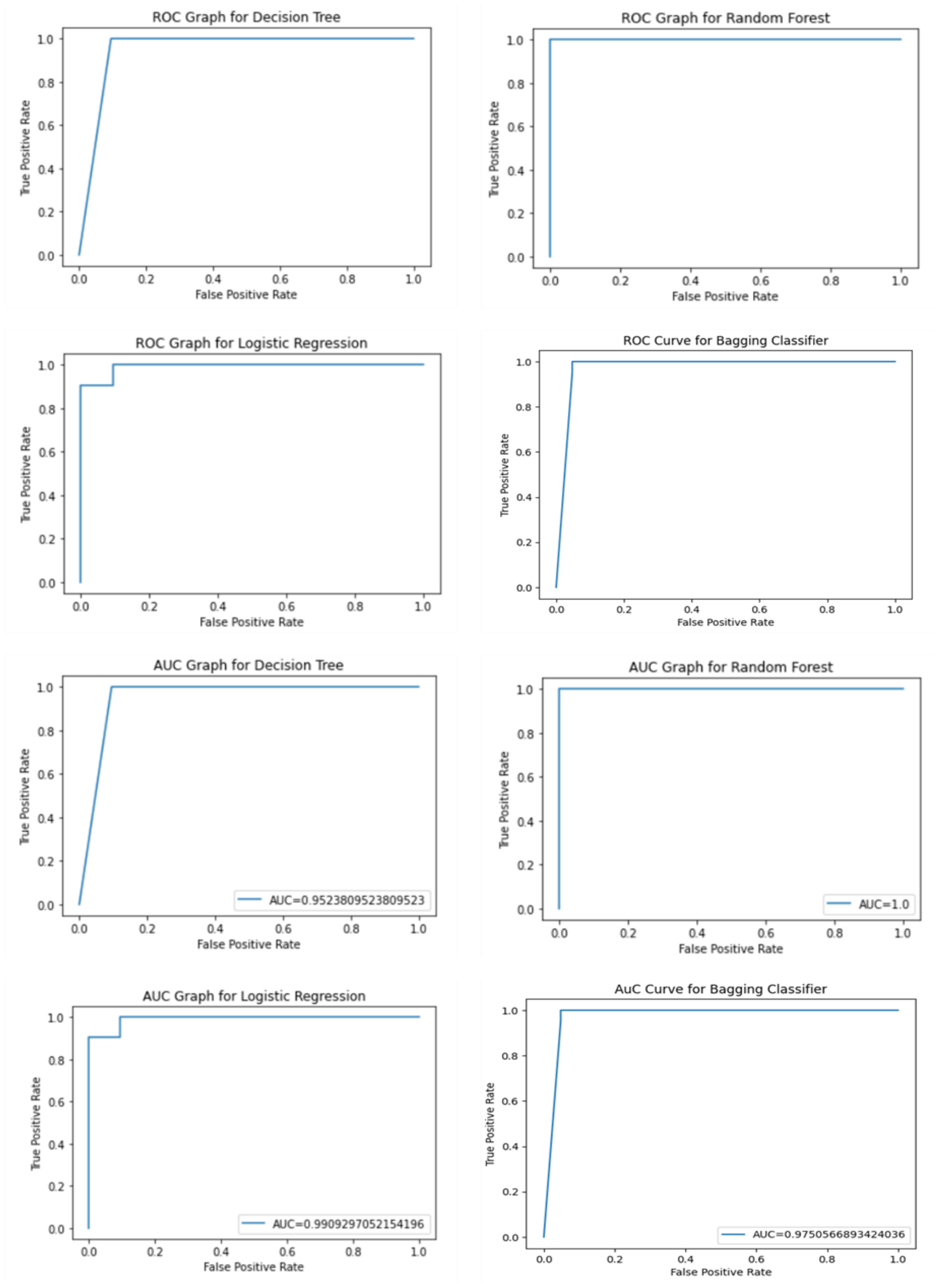


Figure 5.1-ROC and AUC Graphs

The above figure 5.1 shows the graphs of RoC and AuC of all the applied machine learning classifiers. Random forest proved to be best for ROC and AUC as at 0

5.4 Summary

This chapter delves into the analysis and outcomes obtained during the research. Subsequently, the subsequent chapter focuses on the validation and verification of the achieved results.

6. Discussion and Analysis

This chapter describes validation and verification of the results obtained from the experimentation. They are examined in relation to the proposed feature set. Throughout the study, the results gathered from various classifiers together with Decision Tree, Random Forest, Logistic Regression, Naïve Bayes, Bagging, Gradient Boosting, and KNN classifiers (discussed in chapter 5) are compared with the method suggested by Chen et al. [9]. Additionally, the observed results are further validated against the Chen et al. [9] approach within this chapter.

6.1 Overview

In this part, we assess the efficacy of the investigation conducted on Android ransomware detection. We have chosen several traditional machine-learning algorithms, inclusive of Decision Tree, Random Forest, SVM, Logistic Regression, Naive Bayes, Gradient Boosting, Bagging, and KNN classifier models. These models were evaluated using permissions as a metric to gauge the effectiveness of the suggested approach. Instead of using the 21 different permissions set employed in the Chen approach [9], our research utilized a reduced feature set consisting of only 16 permissions deemed most significant. Table 13 displays the detection accuracies achieved by the Chen approach [9], which ranged around 99%. However, our suggested method surpassed the Chen approach [9] in the course of performance, utilizing a minimal feature set while achieving similar detection accuracies.

6.2 Comparison with Reference Approach

Table 6.1 illustrates the comprehensive performance contrast between the proposed and Chen et al. [9] approaches. It is evident from the results that the Decision Tree classifier demonstrates the highest detection performance in the Chen approach. Importantly, our proposed scheme achieves comparable accuracies with Chen et al. [9] for the KNN and Random Forest models, despite employing a reduced number of features. Thus, we can conclude that the Random Forest and SVM-based ensemble classifier, utilizing the proposed feature set, achieve the highest accuracy ratings of 97%. Additionally, Table 6.1 highlights the successful reduction of five features in our proposed scheme by preserving homogeneous detection accuracies. Based on the aforementioned contrast, it is evident that our proposed method surpassed the Chen approach [9] in the course of accuracy, with the Random Forest and SVM classifier emerging as the best-performing models.

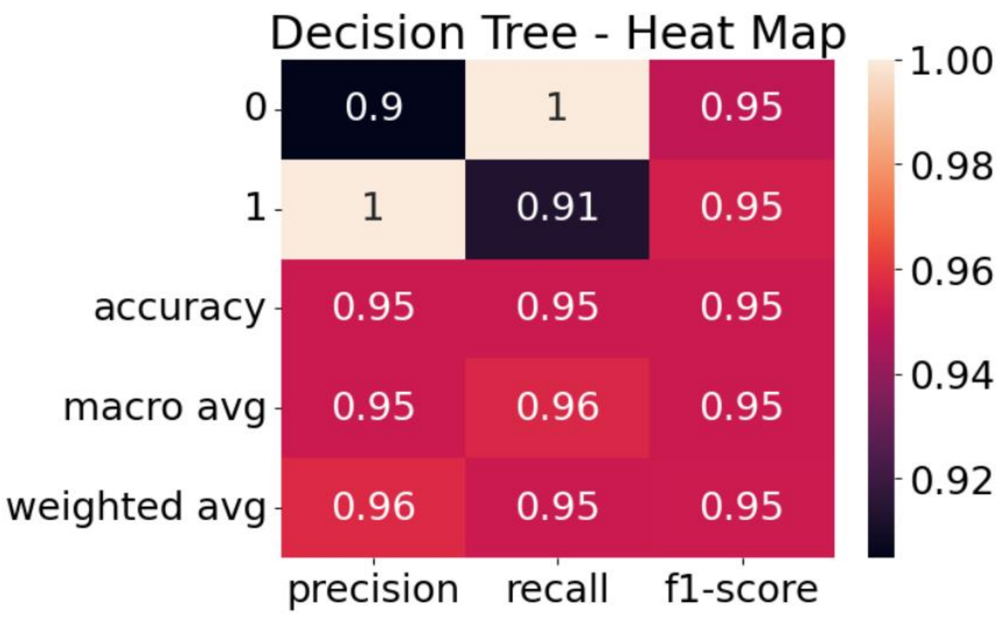
To calculate the performance of Chen et. al. method on a proposed dataset with its 21 features, we have observed a similar detection accuracy as compared to our proposed model with 16 permissions.

Table 6.1(a) -Comparison of Accuracies of Chen and Proposed Approaches

S. No:	**ML Classifiers**	**Accuracy (Reference Approach) Features: 21**	**Accuracy (Proposed Approach) Features: 16**
1.	Decision Tree	98%	95%
2.	Random Forest	97%	97%
3.	Logistic Regression	93%	95%
4.	Bagging Classifier	94%	94%
5.	Support Vector Machine	93%	97%
6.	K-Nearest Neighbor	97%	97% (knn=1) 97% (knn=5)
7.	Gradient Boosting	100%	100%

8.	Naïve Bayes	74%	73.8%

6.3 Heat Map of Applied Machine Learning algorithms



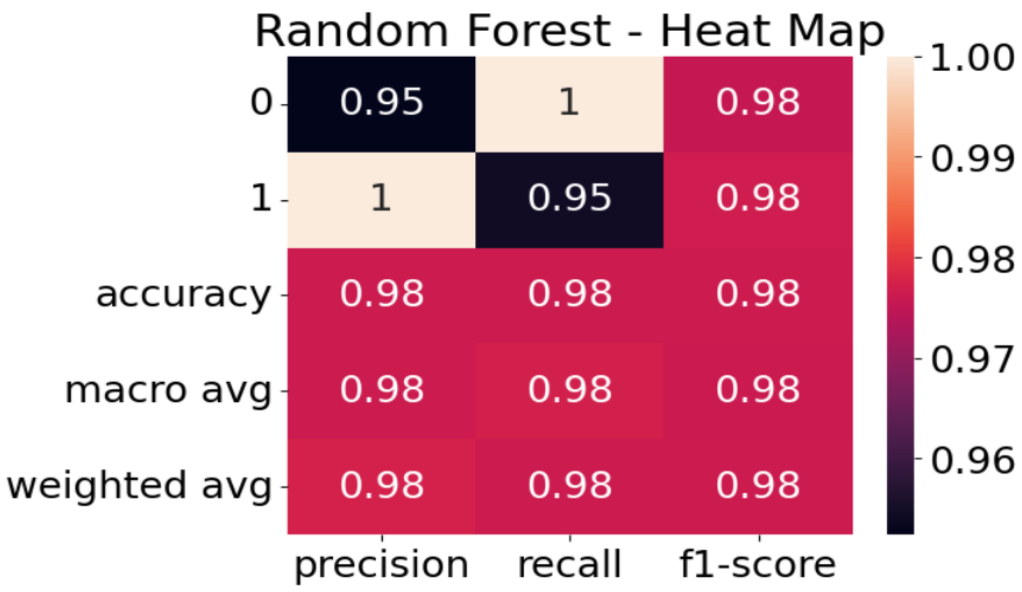


Figure 6.1-Decision tree and Random Forest heat maps

The above figure 6.1(a) represents the heat maps of Decision Tree and Random Forest algorithms. Heat map is a graphical representation that uses color coding to visualize the strength of correlations between variables. It aids in identifying the most effective features for building Machine Learning models by displaying the coefficients. The heat map converts the correlation matrix into a visual display of colors. Heat map of decision tree showed 95% accuracy. It represents all the results in visual form. Also, it can be clearly seen from the heatmap of random forest that the accuracy is 98% as mentioned in above chapters.

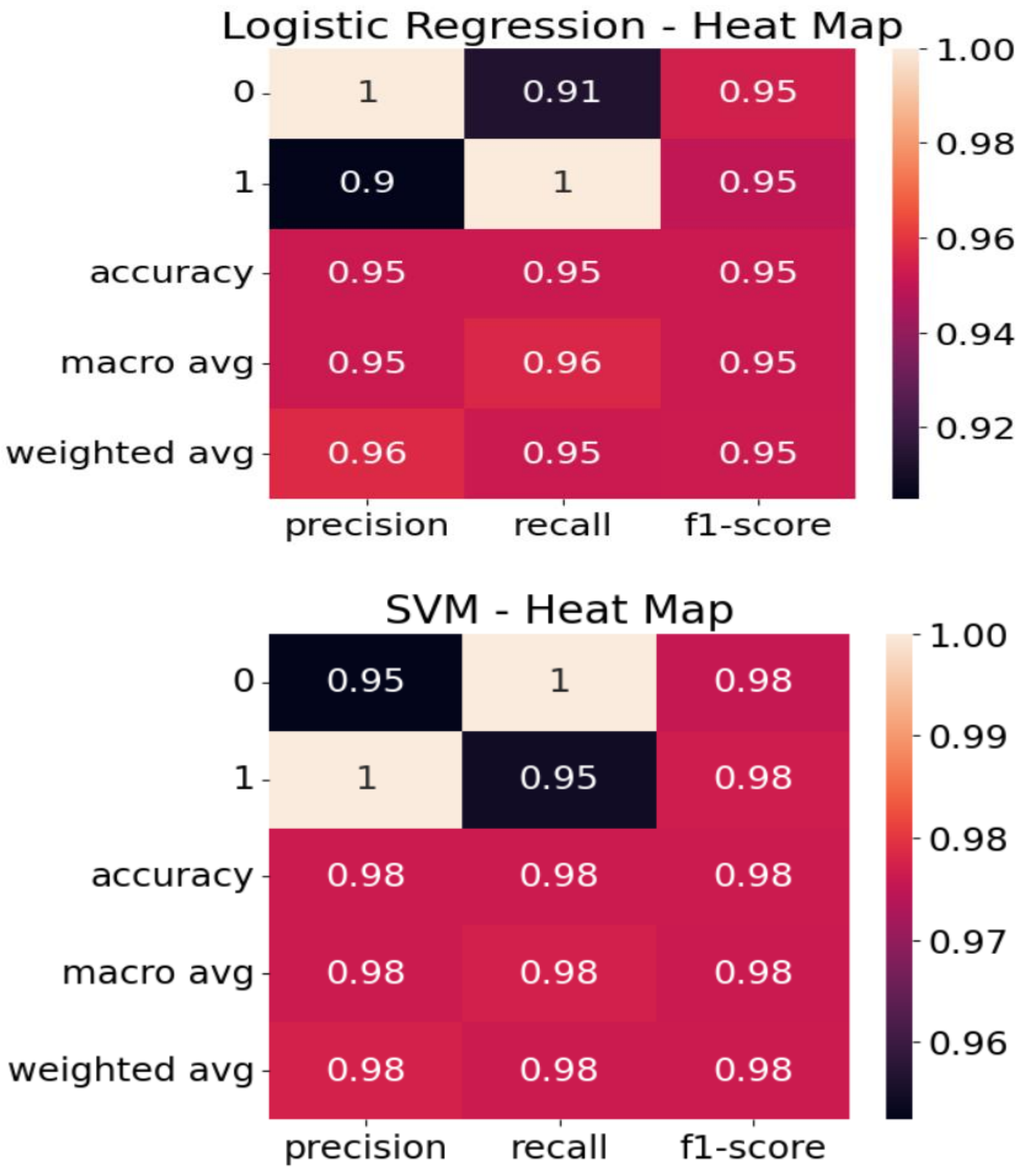


Figure 6.2-Logistic Regression and SVM heat maps

Figure 6.2 illustrates the heatmaps for logistic regression and SVM algorithms. The results shown in heatmap are the same as described. The accuracies for logistic regression and SVM are 95% and 98% respectively.

The Table 6.1 (b) below shows the comparison of proposed framework with the existing reference approaches.

Table 6.2(b) -Comparison of Proposed Approaches with existing Reference Approaches

Related Work	**Framework**	**Analysis Technique**	**Feature Set**	**Machine Learning Techniques**	**Permissions**	**Accuracy**
Chen et al. [9], 2018	RansomProber	Static, Dynamic	Widgets, Activities, 21 Permissions	User interface, User Clicks	No	99%
Alzahrani et al. [10], 2018	RAndroid	Static, Dynamic	Text, Images	Not stated	s	91%
Scalas et al. [12], 2019		Static	API packages, Classes, Methods	RF	No	97%
Lachtar et al. [31], 2019	Native instructions based	Static	A dictionary contains unique opcodes present	RF, SVM, KNN, ANN	No	99.8%
Alzahrani et al. [14], 2019	API-Based	Static	34 API, 48 Permissions, 4 Intents	KNN, LR, SVM, RF	Yes	97.62%
Alsoghyer et al. [15], 2020	Permissions based	Static	115 Permissions are used	RF, J48, SMO, NB	Yes	96.9%

Proposed Approach, 2023	RansomShield	Static	16 Permissions	SVM, DT, RF, LR, NB, KNN, GB, Bagging Classifier	Yes	97%

6.4 Applicability of the Proposed Approach

The following use cases are provided to enhance our understanding of the application of the proposed RansomShield. Early detection patterns can greatly assist in minimizing and ideally preventing damages in ransomware detection analysis. Therefore, drawing from the aforementioned analysis, we can apply the research's effectiveness to various scenarios, including the following examples:

A. The ability to distinguish between Android ransomware applications and benign ones can be facilitated by analyzing the permissions requested during the installation process.

B. The module can be integrated into end devices as a lightweight anti-ransomware application. For each new installation, it extracts permission features from the APK file and transfers the data to the trained classifier. The classifier then utilizes this data for classification purposes. Depending on the results, the installation is either permitted or the detection is reported to the end-users.

C. It is possible to implement it on application stores, where it can be utilized for categorizing applications that are being uploaded, prior to their availability to the general public.

D. By implementing both host-based and market-based approaches, it is possible to leverage its capabilities for additional security enhancements. This implementation can offer extended verification measures, even for applications available outside the official app store.

6.5 Summary

In this chapter, we have consolidated significant findings and conducted validations to verify our results. A comparative analysis has been performed, contrasting the outcomes derived from the Chen et al. [9] approach with our own findings. Additionally, we have explored potential applications of our proposed approach. The subsequent chapter focuses on the conclusion and future prospects of our work.

7. Conclusion & Future Work

In this chapter, we will summarized the thesis and gives future research directions for researchers. It outlines various avenues for research and highlights unresolved research issues that require attention from the academic community.

7.1 Conclusion

Android ransomware is a significant danger to the mobile market, and machine learning algorithms can be utilized to detect security risks and achieve high accuracy through feature selection and algorithm performance. This study focuses on static analysis of applications to create an effective model for android ransomware detection. We examine various approaches used for detecting android ransomware and compare our dataset's permissions with other models' permissions to demonstrate the efficacy of our model, which produced high-accuracy results with all four applied machine classifiers. Our evaluation indicates that by selecting a significant set of permissions and creating our own dataset, RansomShield achieves high-accuracy results.

Using the feature dataset at hand, we conducted classification experiments employing different classifiers, including Gradient Boosting, Decision Tree, SVM, Logistic regression, Naïve Bayes, Bagging, Random Forest and KNN. When comparing our results with the existing method, we observed that our proposed strategy achieved comparable levels of accuracy on the classifiers utilized by Chen et al. [9], and it also demonstrated the potential for improved results when employing other classifiers. The comparison outcomes clearly indicate that Random Forest, SVM, and KNN outperform the remaining five selected classifiers.

The results of our experiments demonstrate that when we employed the proposed feature dataset, we can attain a baseline accuracy of 74% using the Naïve Bayes classifier.

Furthermore, by employing other classifiers, we can achieve accuracy rates above 90%. Based on these findings, we draw the conclusion that Random Forest, SVM, and KNN classifiers exhibit superior performance, detecting ransomware APKs with suggested features set at approximately 97%.

7.2 Limitation & Future Work

The proposed work has a significant drawback in that it relies solely on Android permissions for detecting ransomware and has a limited dataset. While using permissions can help identify apps that excessively request permissions, there may be instances where an app declares dangerous permissions but doesn't actually use them. To ensure comprehensive detection, it is important to address such edge cases through malware analysis. Therefore, future efforts will aim to examine both the declaration and usage of permissions for improved detection.

To extend our proposed research, the dataset can be enhanced, and larger datasets may explore to minimize the permissions. Additionally, dynamic features can be incorporated into the detection features by considering the permissions identified in our study as a starting point. The best sandbox for analyzing the dynamic behavior of Android applications is MobSF with Genny motion, which can be utilized in future research to improve the detection accuracies.

8.Bibliography

[1] Oz, H., Aris, A., Levi, A., & Uluagac, A. S. (2022). A survey on ransomware: Evolution, taxonomy, and defense solutions. ACM Computing Surveys (CSUR), 54(11s), 1-37.

[2] Akbar, F., Hussain, M., Mumtaz, R., Riaz, Q., Wahab, A. W. A., & Jung, K. H. (2022). Permissions-based detection of android malware using machine learning. Symmetry, 14(4), 718.

[3] Sharma, S., Kumar, R., & Krishna, C. R. (2020). RansomAnalysis: The evolution and investigation of Android ransomware. In Proceedings of International Conference on IoT Inclusive Life (ICIIL 2019), NITTTR Chandigarh, India (pp. 33-41). Springer Singapore.

[4] Wu, D. J., Mao, C. H., Wei, T. E., Lee, H. M., & Wu, K. P. (2012, August). DroidMat: Android malware detection through manifest and API calls tracing. In 2012 Seventh Asia joint conference on information security (pp. 62-69). IEEE.

[5] Sharma, S., Krishna, C. R., & Kumar, R. (2021). RansomDroid: Forensic analysis and detection of Android Ransomware using unsupervised machine learning technique. Forensic Science International: Digital Investigation, 37, 301168.

[6] Herron, N., Glisson, W. B., McDonald, J. T., & Benton, R. K. (2021, January). Machine learning-based android malware detection using manifest permissions. Proceedings of the 54th Hawaii International Conference on System Sciences.

[7] Gharib, A., & Ghorbani, A. (2017). Dna-droid: A real-time android ransomware detection framework. In Network and System Security: 11th International Conference, NSS 2017, Helsinki, Finland, August 21–23, 2017, Proceedings 11 (pp. 184-198). Springer International Publishing.

[8] Ferrante, A., Malek, M., Martinelli, F., Mercaldo, F., & Milosevic, J. (2018). Extinguishing ransomware-a hybrid approach to android ransomware detection. In Foundations and Practice of Security: 10th International Symposium, FPS 2017, Nancy, France, October 23-25, 2017, Revised Selected Papers 10 (pp. 242-258). Springer International Publishing.

[9] Chen, J., Wang, C., Zhao, Z., Chen, K., Du, R., & Ahn, G. J. (2017). Uncovering the face of android ransomware: Characterization and real-time detection. IEEE Transactions on Information Forensics and Security, 13(5), 1286-1300.

[10] Alzahrani, A., Alshehri, A., Alshahrani, H., Alharthi, R., Fu, H., Liu, A., & Zhu, Y. (2018, May). Randroid: Structural similarity approach for detecting ransomware applications in android platform. In 2018 IEEE International Conference on Electro/Information Technology (EIT) (pp. 0892-0897). IEEE.

[11] Su, D., Liu, J., Wang, X., & Wang, W. (2018). Detecting Android locker-ransomware on chinese social networks. IEEE Access, 7, 20381-20393.

[12] Scalas, M., Maiorca, D., Mercaldo, F., Visaggio, C. A., Martinelli, F., & Giacinto, G. (2019). On the effectiveness of system API-related information for Android ransomware detection. Computers & Security, 86, 168-182.

[13] Saracino, A., Sgandurra, D., Dini, G., & Martinelli, F. (2016). Madam: Effective and efficient behavior-based android malware detection and prevention. IEEE Transactions on Dependable and Secure Computing, 15(1), 83-97.

[14] Alzahrani, A., Alshahrani, H., Alshehri, A., & Fu, H. (2019, December). An intelligent behavior-based ransomware detection system for android platform. In 2019 First IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA) (pp. 28-35). IEEE.

[15] Alsoghyer, S., & Almomani, I. (2020, March). On the effectiveness of application permissions for Android ransomware detection. In 2020 6th conference on data science and machine learning applications (CDMA) (pp. 94-99). IEEE.

[16] Zhu, H. J., You, Z. H., Zhu, Z. X., Shi, W. L., Chen, X., & Cheng, L. (2018). DroidDet: effective and robust detection of android malware using static analysis along with rotation forest model. Neurocomputing, 272, 638-646.

[17] CIC. To download the dataset for android ransomware. Accessed: Sep, 2022. [Online].Available: https://www.unb.ca/cic/datasets/index.html

[18] Androzoo. Download Benign apk files for analyzing. Accessed: Aug, 2022. [Online]. Available: https://androzoo.uni.lu/

[19] Faruki, P., Bharmal, A., Laxmi, V., Ganmoor, V., Gaur, M. S., Conti, M., & Rajarajan, M. (2014). Android security: a survey of issues, malware penetration, and defenses. IEEE communications surveys & tutorials, 17(2), 998-1022.

[20] Felt, A. P., Finifter, M., Chin, E., Hanna, S., & Wagner, D. (2011, October). A survey of mobile malware in the wild. In Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices (pp. 3-14).

[21] Enck, W., Ongtang, M., & McDaniel, P. (2009, November). On lightweight mobile phone application certification. In Proceedings of the 16th ACM conference on Computer and communications security (pp. 235-245).

[22] Wu, D. J., Mao, C. H., Wei, T. E., Lee, H. M., & Wu, K. P. (2012, August). Droidmat: Android malware detection through manifest and api calls tracing. In 2012 Seventh Asia joint conference on information security (pp. 62-69). IEEE.

[23] Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K., & Siemens, C. E. R. T. (2014, February). Drebin: Effective and explainable detection of android malware in your pocket. In Ndss (Vol. 14, pp. 23-26).

[24] Sanz, B., Santos, I., Laorden, C., Ugarte-Pedrero, X., Nieves, J., Bringas, P. G., & Álvarez Marañón, G. (2013). MAMA: manifest analysis for malware detection in android. Cybernetics and Systems, 44(6-7), 469-488.

[25] Qiao, M., Sung, A. H., & Liu, Q. (2016, July). Merging permission and api features for android malware detection. In 2016 5th IIAI international congress on advanced applied informatics (IIAI-AAI) (pp. 566-571). IEEE.

[26] Li, J., Sun, L., Yan, Q., Li, Z., Srisa-An, W., & Ye, H. (2018). Significant permission identification for machine-learning-based android malware detection. IEEE Transactions on Industrial Informatics, 14(7), 3216-3225.

[27] Diamantaris, M., Papadopoulos, E. P., Markatos, E. P., Ioannidis, S., & Polakis, J. (2019, March). Reaper: real-time app analysis for augmenting the android permission system. In Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy (pp. 37-48).

[28] Andronio, N., Zanero, S., & Maggi, F. (2015). Heldroid: Dissecting and detecting mobile ransomware. In Research in Attacks, Intrusions, and Defenses: 18th International Symposium, RAID 2015, Kyoto, Japan, November 2-4, 2015. Proceedings 18 (pp. 382-404). Springer International Publishing.

[29] Mercaldo, F., Nardone, V., & Santone, A. (2016, August). Ransomware inside out. In 2016 11th International Conference on Availability, Reliability and Security (ARES) (pp. 628-637). IEEE.

[30] Maiorca, D., Mercaldo, F., Giacinto, G., Visaggio, C. A., & Martinelli, F. (2017, April). R-PackDroid: API package-based characterization and detection of mobile ransomware. In Proceedings of the symposium on applied computing (pp. 1718-1723).

[31] Lachtar, N., Ibdah, D., & Bacha, A. (2019). The case for native instructions in the detection of mobile ransomware. IEEE Letters of the Computer Society, 2(2), 16-19.

[32] Bibi, I., Akhunzada, A., Malik, J., Ahmed, G., & Raza, M. (2019, August). An effective Android ransomware detection through multi-factor feature filtration and recurrent neural network. In 2019 UK/China Emerging Technologies (UCET) (pp. 1-4). IEEE.

[33] Abdullah, Z., Muhadi, F. W., Saudi, M. M., Hamid, I. R. A., & Foozy, C. F. M. (2020). Android ransomware detection based on dynamic obtained features. In Recent Advances on Soft Computing and Data Mining: Proceedings of the Fourth International Conference on Soft Computing and Data Mining (SCDM 2020), Melaka, Malaysia, January 22– 23, 2020 (pp. 121-129). Springer International Publishing.

[34] Sifat, S., Hossain, M. S., Tonny, S. A., Majumder, B., Mahajabin, R., & Shakhawat, H. M. (2023). Android Ransomware Attacks Detection with Optimized Ensemble Learning. In Advances in Cybersecurity, Cybercrimes, and Smart Emerging Technologies (pp. 41-53). Cham: Springer International Publishing.

[35] VirusTotal. Analyze Suspicious Files and URLs to Detect Types of Malware, automatically. Accessed: Oct, 2022. [Online]. Available: https://www.virustotal.com.

[36] Alazab, M., Alazab, M., Shalaginov, A., Mesleh, A., & Awajan, A. (2020). Intelligent mobile malware detection using permission requests and API calls. Future Generation Computer Systems. doi:10.1016/j.future.2020.02.002

[37] Asma Razgallah, Raphaël Khoury, Sylvain Hallé, Kobra Khanmohammadi, "A survey of malware detection in Android apps: Recommendations and perspectives for future research", Computer Science Review 39 (2021) 100358

[38] Androguard: Reverse engineering, Malware analysis of Android applications. [online] (Accessed September 20, 2023) https://github.com/androguard/androguard

[39] Apktool; A tool for reverse engineering Android apk files [online] (Accessed September 20, 2023), https://ibotpeaches.github.io/Apktool/

[40] Wang, Wei, Meichen Zhao, Zhenzhen Gao, Guangquan Xu, Hequn Xian, Yuanyuan Li, and Xiangliang Zhang. "Constructing features for detecting android malicious applications: issues, taxonomy and directions." IEEE access 7 (2019): 67602-67631.

[41] Wang, W., Zhao, M., Gao, Z., Xu, G., Xian, H., Li, Y., & Zhang, X. (2019). Constructing Features for Detecting Android Malicious Applications: Issues, Taxonomy and Directions. *IEEE Access*, *7*, 67602–67631. https://doi.org/10.1109/ACCESS.2019.2918139

[42] Wang, Z., Liu, Q., & Chi, Y. (2020). Review of android malware detection based on deep learning. In *IEEE Access* (Vol. 8, pp. 181102–181126). Institute of

Electrical and Electronics Engineers Inc. https://doi.org/10.1109/ACCESS.2020.3028370

[43] App Manifest Overview [online] (Accessed September 20, 2023), https://developer.android.com/guide/topics/ manifest/manifest-intro

[44] El, I., Li, N. R., & Murphy, M. J. (n.d.). Theory and Applications Machine Learning in Radiation Oncology.

[45] "What is decision tree?," IBM, [Online]. Available: https://www.ibm.com/topics/decision-trees. [Accessed 5 2023].

[46] Ellis, K., Kerr, J., Godbole, S., Lanckriet, G., Wing, D., & Marshall, S. (2014). A random forest classifier for the prediction of energy expenditure and type of physical activity from wrist and hip accelerometers. *Physiological Measurement*, *35*(11). https://doi.org/10.1088/0967-3334/35/11/2191

[47] "Logistic Regression — Detailed Overview," Towardsdatascience, [Online]. Available: https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc. [Accessed 23 May 2023].

[48] "ML|Bagging Classifier," geeksforgeeks, [Online]. Available: https://www.geeksforgeeks.org/ml-bagging-classifier/. [Accessed 16 May 2023].

[49] "k-nearest-neighbor-algorithm-for-machine-learning," JavaTpoint, [Online]. Available: https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning. [Accessed 16 May 2023].

[50] S. Rajasekar, P. Philominathan, and V. Chinnathambi, “Research Methodology”, 2013. Available: http://arxiv.org/pdf/physics/ 0601009.pdf .

[51] W. C. Booth, G. G. Colomb, and J. M. Williams, “The Craft of Research.”[Online].Available:http://sir.spbu.ru/en/programs/master/master_program_in _international_relations/digital_library/Book Research seminar by Booth.pdf

[52] C. Woody, “Chapter 3: Research Methodology,” 2001. Available: https://shodhganga.inflibnet.ac.in/bitstream/10603/2026/16/16_chapter 3.pdf

9. Appendices

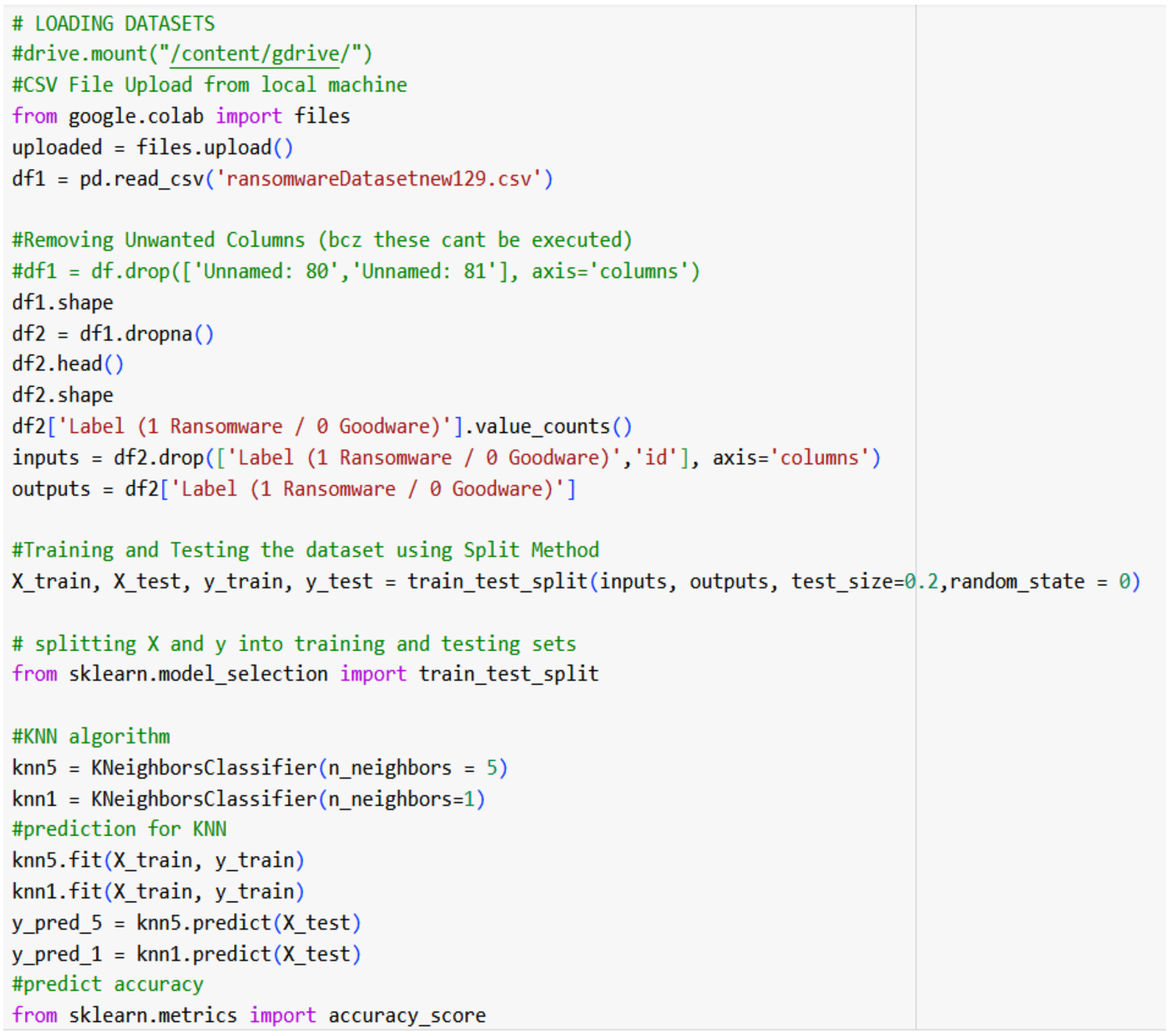
Appendix-A

Permission letter

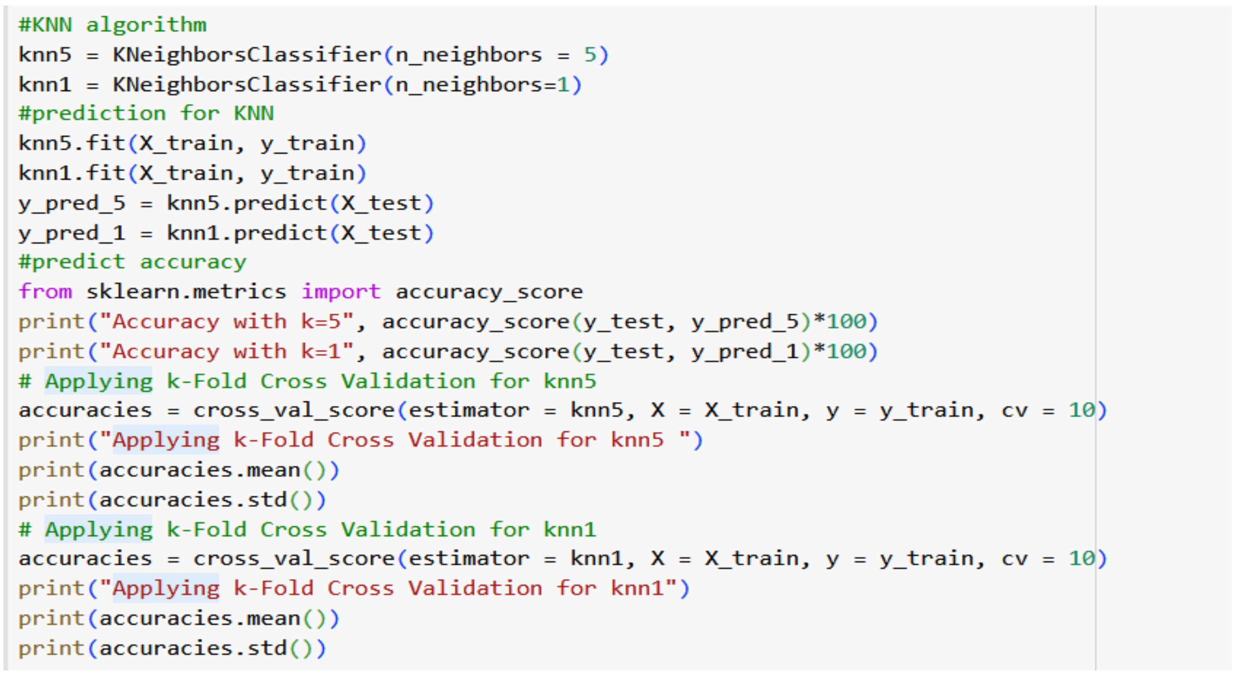
Appendix-B

Code

```
from sklearn.metrics import classification_report
import pandas as pd
from google.colab import drive

from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import plot_tree
from sklearn import metrics

import numpy as np
import seaborn as sns
from sklearn.datasets import make_classification
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import RandomForestRegressor
from sklearn.inspection import permutation_importance
from matplotlib import pyplot
from matplotlib import pyplot as plt
from sklearn.model_selection import cross_val_score

from sklearn.datasets import make_blobs
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report
from sklearn.model_selection import KFold
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import BaggingClassifier
from numpy import mean
from numpy import std
```



```python
# LOADING DATASETS
#drive.mount("/content/gdrive/")
#CSV File Upload from local machine
from google.colab import files
uploaded = files.upload()
df1 = pd.read_csv('ransomwareDatasetnew129.csv')

#Removing Unwanted Columns (bcz these cant be executed)
#df1 = df.drop(['Unnamed: 80','Unnamed: 81'], axis='columns')
df1.shape
df2 = df1.dropna()
df2.head()
df2.shape
df2['Label (1 Ransomware / 0 Goodware)'].value_counts()
inputs = df2.drop(['Label (1 Ransomware / 0 Goodware)','id'], axis='columns')
outputs = df2['Label (1 Ransomware / 0 Goodware)']

#Training and Testing the dataset using Split Method
X_train, X_test, y_train, y_test = train_test_split(inputs, outputs, test_size=0.2,random_state = 0)

# splitting X and y into training and testing sets
from sklearn.model_selection import train_test_split

#KNN algorithm
knn5 = KNeighborsClassifier(n_neighbors = 5)
knn1 = KNeighborsClassifier(n_neighbors=1)
#prediction for KNN
knn5.fit(X_train, y_train)
knn1.fit(X_train, y_train)
y_pred_5 = knn5.predict(X_test)
y_pred_1 = knn1.predict(X_test)
#predict accuracy
from sklearn.metrics import accuracy_score
```



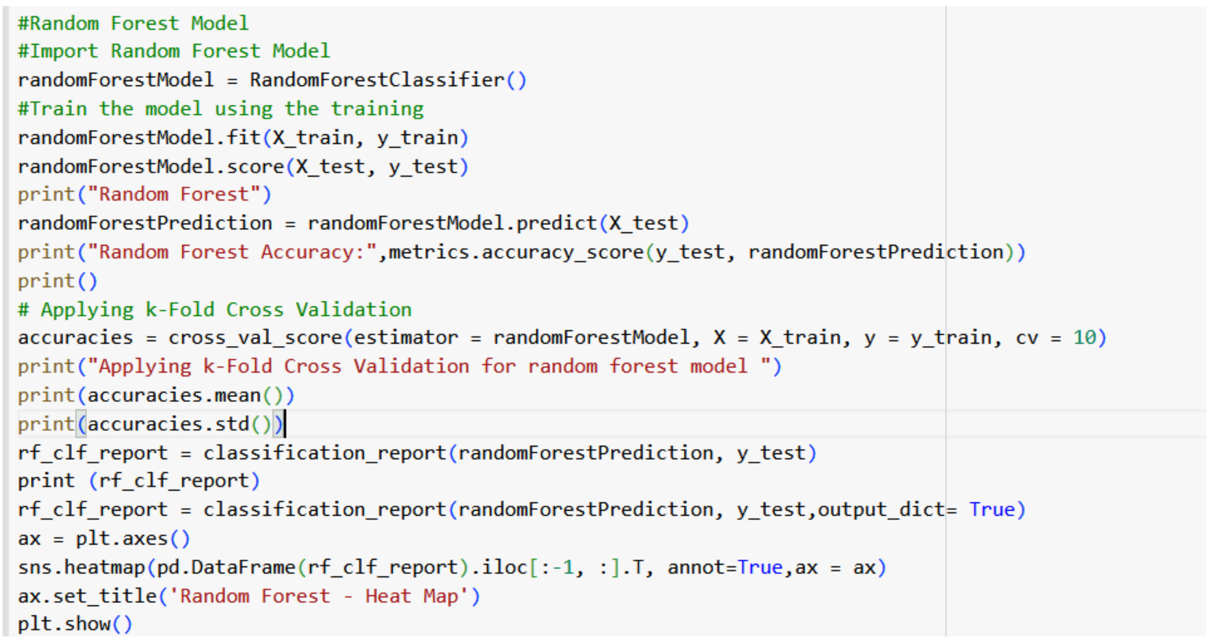
```python
#KNN algorithm
knn5 = KNeighborsClassifier(n_neighbors = 5)
knn1 = KNeighborsClassifier(n_neighbors=1)
#prediction for KNN
knn5.fit(X_train, y_train)
knn1.fit(X_train, y_train)
y_pred_5 = knn5.predict(X_test)
y_pred_1 = knn1.predict(X_test)
#predict accuracy
from sklearn.metrics import accuracy_score
print("Accuracy with k=5", accuracy_score(y_test, y_pred_5)*100)
print("Accuracy with k=1", accuracy_score(y_test, y_pred_1)*100)
# Applying k-Fold Cross Validation for knn5
accuracies = cross_val_score(estimator = knn5, X = X_train, y = y_train, cv = 10)
print("Applying k-Fold Cross Validation for knn5 ")
print(accuracies.mean())
print(accuracies.std())
# Applying k-Fold Cross Validation for knn1
accuracies = cross_val_score(estimator = knn1, X = X_train, y = y_train, cv = 10)
print("Applying k-Fold Cross Validation for knn1")
print(accuracies.mean())
print(accuracies.std())
```

```
#gradient boosting algorithm
gradient_booster = GradientBoostingClassifier(learning_rate=0.1)
gradient_booster.get_params()
gradient_booster.fit(X_train,y_train)
print("GBA Acuracy:",classification_report(y_train,gradient_booster.predict(X_train)))
# Applying k-Fold Cross Validation for gradient booster
accuracies = cross_val_score(estimator = gradient_booster, X = X_train, y = y_train, cv = 10)
print("Applying k-Fold Cross Validation for gradient boosting ")
print(accuracies.mean())
print(accuracies.std())

# Naive Bayes Model
# guassian NB training the model on training set
gnb = GaussianNB()
gnb.fit(X_train, y_train)
# making predictions on the testing set
y_pred = gnb.predict(X_test)
# comparing actual response values (y_test) with predicted response values (y_pred)
from sklearn import metrics
print("Gaussian Naive Bayes model accuracy(in %):", metrics.accuracy_score(y_test, y_pred)*100)
# Applying k-Fold Cross Validation
accuracies = cross_val_score(estimator = gnb, X = X_train, y = y_train, cv = 10)
print("Applying k-Fold Cross Validation for naive beyes ")
print(accuracies.mean())
print(accuracies.std())
```

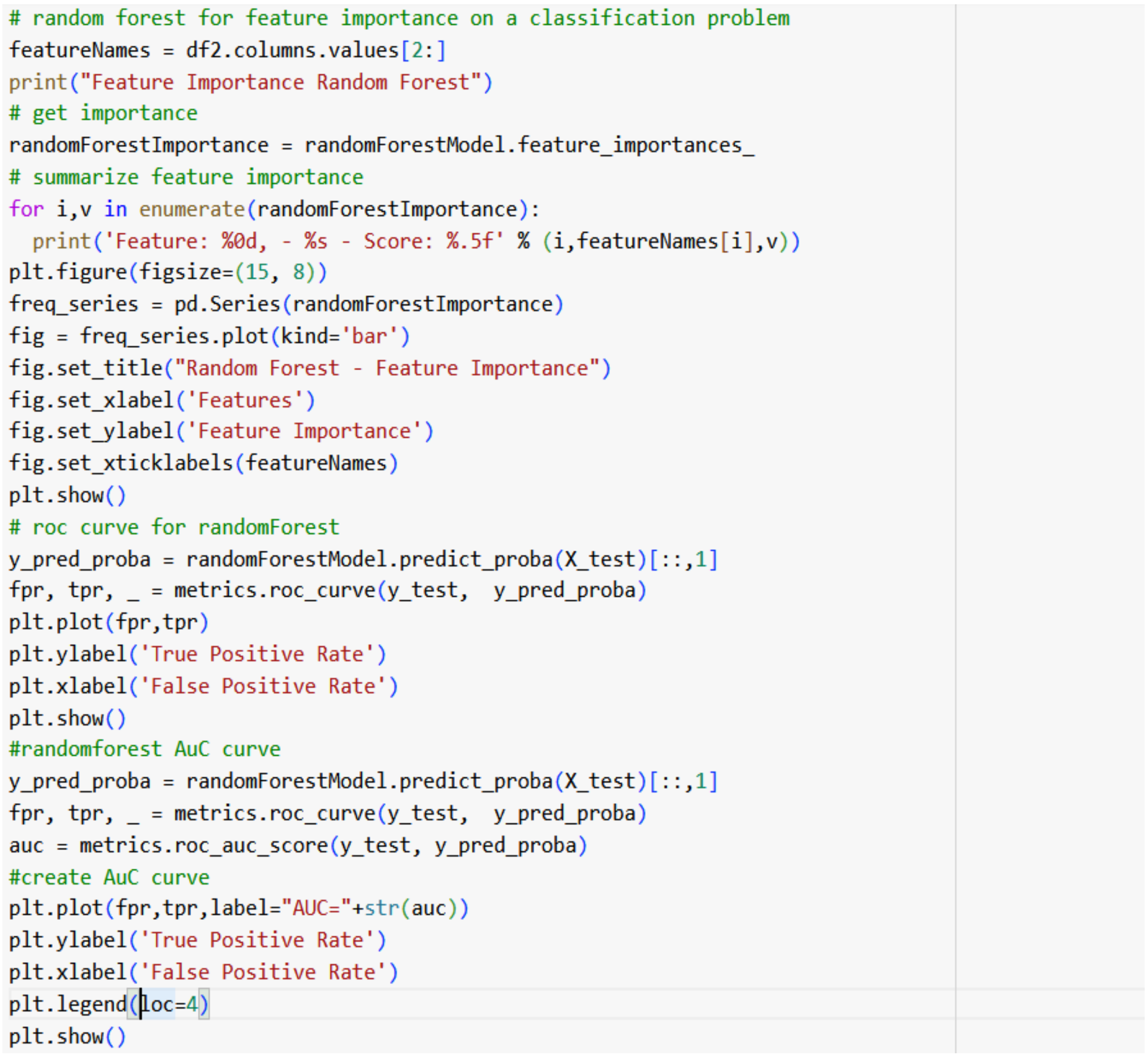
```
#Bagging Classifier
# Append the model and score to their respective list
oob_model = BaggingClassifier(n_estimators = 12, oob_score = True,random_state = 22)
oob_model.fit(X_train, y_train)
print(oob_model.oob_score_)
# Applying k-Fold Cross Validation
accuracies = cross_val_score(estimator = oob_model, X = X_train, y = y_train, cv = 10)
print("Applying k-Fold Cross Validation for bagging classifier ")
print(accuracies.mean())
print(accuracies.std())

# roc curve for bagging classifier
#define metrics
y_pred_proba = oob_model.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
#create ROC curve
plt.plot(fpr,tpr)
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
#bagging classifier AuC curve
#define metrics
y_pred_proba = oob_model.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)
#create AUC curve
plt.plot(fpr,tpr,label="AUC="+str(auc))
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.show()
```

```
# Create Decision Tree classifer object
decisionTreeModel  = DecisionTreeClassifier()
decisionTreeModel .fit(X_train, y_train)
decisionTreeModel .score(X_test, y_test)
print("Decision Tree")
decisionTreePrediction = decisionTreeModel .predict(X_test)
print("Decision Tree Accuarcy:",metrics.accuracy_score(y_test, decisionTreePrediction))
print()
# Applying k-Fold Cross Validation
accuracies = cross_val_score(estimator = decisionTreeModel, X = X_train, y = y_train, cv = 10)
print("Applying k-Fold Cross Validation for decision tree ")
print(accuracies.mean())
print(accuracies.std())
dt_clf_report=classification_report(decisionTreePrediction, y_test )
print(dt_clf_report)
dt_clf_report=classification_report(decisionTreePrediction, y_test,output_dict = True )
ax = plt.axes()
sns.heatmap(pd.DataFrame(dt_clf_report).iloc[:-1, :].T, annot=True,ax = ax)
# plot_tree=(decisionTreeModel )
ax.set_title('Decision Tree - Heat Map')
plt.show()
```

```
# roc curve for decision tree
y_pred_proba = decisionTreeModel.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
#create ROC curve
plt.plot(fpr,tpr)
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.title('ROC Graph Decision Tree')
plt.show()
#decision tree AuC curve
y_pred_proba = decisionTreeModel.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)
#create AuC curve
plt.plot(fpr,tpr,label="AUC="+str(auc))
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.show()
```



```python
#Random Forest Model
#Import Random Forest Model
randomForestModel = RandomForestClassifier()
#Train the model using the training
randomForestModel.fit(X_train, y_train)
randomForestModel.score(X_test, y_test)
print("Random Forest")
randomForestPrediction = randomForestModel.predict(X_test)
print("Random Forest Accuracy:",metrics.accuracy_score(y_test, randomForestPrediction))
print()
# Applying k-Fold Cross Validation
accuracies = cross_val_score(estimator = randomForestModel, X = X_train, y = y_train, cv = 10)
print("Applying k-Fold Cross Validation for random forest model ")
print(accuracies.mean())
print(accuracies.std())
rf_clf_report = classification_report(randomForestPrediction, y_test)
print (rf_clf_report)
rf_clf_report = classification_report(randomForestPrediction, y_test,output_dict= True)
ax = plt.axes()
sns.heatmap(pd.DataFrame(rf_clf_report).iloc[:-1, :].T, annot=True,ax = ax)
ax.set_title('Random Forest - Heat Map')
plt.show()
```



```python
# random forest for feature importance on a classification problem
featureNames = df2.columns.values[2:]
print("Feature Importance Random Forest")
# get importance
randomForestImportance = randomForestModel.feature_importances_
# summarize feature importance
for i,v in enumerate(randomForestImportance):
  print('Feature: %0d, - %s - Score: %.5f' % (i,featureNames[i],v))
plt.figure(figsize=(15, 8))
freq_series = pd.Series(randomForestImportance)
fig = freq_series.plot(kind='bar')
fig.set_title("Random Forest - Feature Importance")
fig.set_xlabel('Features')
fig.set_ylabel('Feature Importance')
fig.set_xticklabels(featureNames)
plt.show()
# roc curve for randomForest
y_pred_proba = randomForestModel.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
plt.plot(fpr,tpr)
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
#randomforest AuC curve
y_pred_proba = randomForestModel.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)
#create AuC curve
plt.plot(fpr,tpr,label="AUC="+str(auc))
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.show()
```

```
#Logistic regression
logisticRegressionModel = LogisticRegression()
logisticRegressionModel.fit(X_train, y_train)
logisticRegressionModel.score(X_test, y_test)
Logistic_YPrediction = logisticRegressionModel.predict(X_test)
print("Logistic Regression")
print("Logistic Regression Accuarcy:",metrics.accuracy_score(y_test, Logistic_YPrediction))
print()
prediction = logisticRegressionModel.predict(X_test)
# Applying k-Fold Cross Validation
accuracies = cross_val_score(estimator = logisticRegressionModel, X = X_train, y = y_train, cv = 10)
print("Applying k-Fold Cross Validation for logistic regression ")
print(accuracies.mean())
print(accuracies.std())
lr_clf_report = classification_report(prediction, y_test)
print(lr_clf_report)
lr_clf_report = classification_report(prediction, y_test,output_dict=True)
ax = plt.axes()
sns.heatmap(pd.DataFrame(lr_clf_report).iloc[:-1, :].T, annot=True,ax = ax)
ax.set_title('Logistic Regression - Heat Map')
```

```
# roc curve for logistic regression
#define metrics
y_pred_proba = logisticRegressionModel.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
#create ROC curve
plt.plot(fpr,tpr)
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
#logistic regression AuC curve
#define metrics
y_pred_proba = logisticRegressionModel.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)
#create ROC curve
plt.plot(fpr,tpr,label="AUC="+str(auc))
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.show()
```

```
#SVM Model
SVMModel = SVC()
SVMModel.fit(X_train, y_train)
SVMModel.score(X_test, y_test)
SVM_YPrediction = SVMModel.predict(X_test)
print("SVM MODEL")
print("SVM_Accuracy:",metrics.accuracy_score(y_test, SVM_YPrediction))
print()
prediction = SVMModel.predict(X_test)
# Applying k-Fold Cross Validation
accuracies = cross_val_score(estimator = SVMModel, X = X_train, y = y_train, cv = 10)
print("Applying k-Fold Cross Validation for SVM ")
print(accuracies.mean())
print(accuracies.std())
svm_clf_report = classification_report(prediction, y_test)
print(svm_clf_report)
svm_clf_report = classification_report(prediction, y_test,output_dict=True)
ax = plt.axes()
sns.heatmap(pd.DataFrame(svm_clf_report).iloc[:-1, :].T, annot=True)
ax.set_title('SVM - Heat Map')
plt.show()
```

```
#SVM ROC curve
#define metrics
y_pred_proba = SVMModel.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)

#create ROC curve
plt.plot(fpr,tpr)
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()

#SVM AuC curve
#define metrics
y_pred_proba = SVMModel.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)

#create AuC curve
plt.plot(fpr,tpr,label="AUC="+str(auc))
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.show()
```