CAPITAL UNIVERSITY OF SCIENCE AND TECHNOLOGY, ISLAMABAD

Android Ransomware Detection using Machine Learning Techniques to Mitigate Adversarial Evasion Attacks

by

Madiha Ameer

A thesis submitted in partial fulfillment for the degree of Master of Science

in the

Faculty of Computing
Department of Computer Science

2019

I dedicated my dissertation work to my parents and my teachers. A special feelings of gratitude to my loving father and brother for their love, endless support and encouragement.

CERTIFICATE OF APPROVAL

Android Ransomware Detection using Machine Learning Techniques to Mitigate Adversarial Evasion Attacks

by
Madiha Ameer
(MCS173042)

EXAMINING COMMITTEE

S. No.	Examiner	Name	Organization
(a)	External	Dr. Ayyaz Hussain	IIUI, Islamabad
(b)	Internal	Dr. M. Shahid Iqbal Malik	CUST, Islamabad
(c)	Supervisor	Dr. Muhammad Aleem	CUST, Islamabad

Dr. Muhammad Aleem
Thesis Supervisor
November, 2019

Dr. Nayyer Masood
Head
Dept. of Computer Science
November, 2019

Dr. Muhammad Abdul Qadir
Dean
Faculty of Computing
November, 2019

Author's Declaration

I, Madiha Ameer hereby state that my MS thesis titled "**Android Ransomware Detection using Machine Learning Techniques to Mitigate Adversarial Evasion Attacks**" is my own work and has not been submitted previously by me for taking any degree from the Capital University of Science & Technology Islamabad or anywhere else in the country/abroad.

At any time if my statement is found to be incorrect even after my graduation, the University has the right to withdraw my MS Degree.

Madiha Ameer
(MCS173042)

Plagiarism Undertaking

I solemnly declare that research work presented in this thesis titled "*Android Ransomware Detection using Machine Learning Techniques to Mitigate Adversarial Evasion Attacks*" is solely my research work with no significant contribution from any other person. Small contribution/help wherever taken has been dully acknowledged and that complete thesis has been written by me.

I understand the zero tolerance policy of the HEC and the Capital university of science & Technology Islamabad towards plagiarism. Therefore, I as an author of the above titled thesis declare that no portion of my thesis has been plagiarized and any material used as reference is properly referred/cited.

I undertake that if I am found guilty of any formal plagiarism in the above titled thesis even after award of MS Degree, the University reserves the right to withdraw/revoke my MS Degree and that HEC and the University have the right to publish out name on the HEC/University website on which names of students are placed who submitted plagiarized work.

Madiha Ameer
(MCS173042)

List of Publications

It is certified that following publication(s) have been made out of the research work that has been carried out for this thesis:-

1. **Ameer, M.**, Murtaza, S., & Aleem, M. (2018). A Study of Android-based Ransomware: Discovery, Methods, and Impacts. , 13(3).," *Journal of Information Assurance & Security*, vol. 13(3), pp. 109-117, 2018.

Madiha Ameer
(MCS173042)

Acknowledgements

All praise and exaltation to **ALLAH (S.W.T)** the Creator and sustainer of all the seen and unseen word. First and foremost I would like to express my gratitude and thanks to Him for providing me the boundaries and blessings to complete this work. Secondly, I would like to express my sincerest appreciation to my supervisor **Dr. Muhammad Aleem** for his directions, assistance, and guidance. I sincerely thanked for his support, encouragement and technical advice in the research area. I an heartily thankful to him from the final level, as he enabled me to develop an understanding of the subject. He has taught me, both consciously and unconsciously, how good experimental work is carried out, Sir, you will always be remembered in my prayers. I am highly indebted to my parents and my family, for their expectations, assistance, support and encouragement throughout the completion of this Master of Science degree. They form the most important part of my life. After **ALLAH (S.W.T)** they are the sole source of my being in this world. No word can ever be sufficient for the gratitude I have for my father and for my family. A special thanks to my brother for his support and encouragement to complete this Master of Science degree. I pray to ALLAH (S.W.T) that may He bestows me with true success in all fields in both worlds and shower His blessed knowledge upon me for the betterment of all Muslims and whole Mankind.

Madiha Ameer
(MCS173042)

Abstract

Ransomware is a specific kind of malware designed to extort money from someone by locking up their device and personal data files. The device owner is then required to pay a ransom to possibly get the device unlocked and restore access to the personal data files. From the last few couples of years due to excessive use of Android cell phones and their technically advanced capabilities turn them into information storage devices, which attract ransomware writers to attack these devices for more financial benefits. Android ransomware is the most threatening one among all other kinds of Android malware. Developments of new Android malware make it difficult to differentiate between ransomware and other Android malware. Therefore, it becomes mandatory to perform the behavioral analysis of Android ransomware samples to recognize their malicious nature that differs from other malware. Generally, there are two types of analysis mechanisms. First one is the static analysis which is used to analyze the code structure and information flow anomalies without executing the ransomware. Second is dynamic analysis, the dynamic analysis examines run-time behavior by executing ransomware in a controlled environment. Static analysis is unable to detect all possible attack patterns whereas dynamic analysis provides additional protection and capable to detect known and unknown ransomware. Therefore we proposed a hybrid approach which is the combination of both static and dynamic analysis. We examine the permissions, text, and network-based features statically and dynamically monitors the memory usage, system call logs, and CPU usage in order to detect and classify the Android ransomware from other malware using machine learning techniques. We used a machine learning-based ensemble analysis approach to mitigate the adversarial evasion attacks. These ensemble machine learning classifiers are trained on the aforementioned static and dynamic features extracted from Android malware(ransomware and non-ransomware) applications. Our experimental results depict that the proposed ensemble machine learning-based Android ransomware classification and detection technique is capable to classify unknown ransomware that exhibits the same static and dynamic behavior with 100% accuracy. Moreover, It is capable of mitigating adversarial evasion attacks.

Contents

List of Figures

List of Tables

Abbreviations

ADB	Android Debug Bridge
API	Application Program Interface
APK	Android Package Kit
AB	AdaBoost
CPU	Central Processing Unit
C&C	Command and Control
FPR	False Positive Rate
GB	Gradient Boosting
IG	Information Gain
IP	Internet Protocol
LR	Logistic Regression
NB	Naive Bayes
NRW	Non-ransomware
RW	Ransomware
RF	Random Forest
SMS	Short Message Service
SMO	Sequential Minimal Optimization
SVC	Support Vector Classification
SVM	Support Vector Machine
TPR	True Positive Rate
URL	Uniform Resource Locator
WEKA	Waikato Environment for Knowledge Analysis

Chapter 1

Introduction

Ransomware attacks have turned out to be a top security threat confronting individuals and corporations. It is a wicked kind of malware that differs from other malware which may be simply annoying, delete or corrupt files, change system configuration or try to capture information from passwords or keystrokes afterword send it to control server. Whereas, the ransomware (RW) notify the victim after infecting the system. Notification usually demands payment in an untraceable mode in order to restore the system to its prior state [1]. Ransomware comes in different forms. Two general categories of ransomware are Crypto and Locker ransomware [2]. Crypto ransomware finds and encrypts the files on the device using a strong cipher to make them inaccessible by the user. The locker ransomware locks the device itself, mostly by locking user interface or using the pop-up overlay, so that the user cannot even get into it [3]. Ransomware is not only a Windows operating system's phenomena it also attacks the Android devices. By the end of 2018 Android have over 86.8% of the total market share in mobile phones [4].With the growing use of cell phones and Android being the most widely recognized operating system for mobile devices [5], Android phones have become a popular and profitable target for hackers because users mostly keep their personal and valuable data on these devices [1]. In September 2018, McAfee lab declared that the aggregate of ransomware attacks had reached 17 million and Android ransomware would be one of the more prominent security risks in the near future [6].

Ransomware are distributed from the malicious server, google play store or third-party app store. It may be distributed in the form of any legitimate application or by using social engineering tactics that deceive the user to download malicious content such as software updates, fake apps from third-party app stores [7] or by clicking on the spam link sent by SMS [3]. However, modern Android ransomware usually spread through compromised applications that are freely available to the user through third party app stores. Generally, ransomware attackers select the popular application to mimic or infect, to improve the probability that victim will download their version. Depending upon the complexity of attack, the attackers retaining the original functionality of the application may add malicious code to it, or the application may only portray the icon and name of the original application. This is done to silently install ransomware on the device without raising doubts of user [8]. After successful installation, the ransomware gathers data of the victim's device, search for the targeted assets for example files, resources, etc. on the other hand, it communicates with C&C server for obtaining the encryption key if the key was not already included in its payload. After that ransomware hijack (lock/encrypt) the targeted resources according to its type and display a message to the victim asking for ransom payment along with the payment instructions [3]. Screenshot of Android ransomware is shown in Figure 1.1.

At present, ransomware particularly made for Android devices are on the rise [3]. Due to the alarming increase in Android ransomware applications, the analysis and detection of Android ransomware have become an important research area. Till date, a few Android ransomware detection and classification techniques have been proposed [1, 5, 7, 9–13]. Ransomware detection techniques can be classified into two categories: Static and Dynamic analysis. The Static analysis uses syntax or structural properties of the application to determine its maliciousness [7]. Static analysis relies on features extraction (without execution) generally from resources files, Android Manifest files, Java Bytecode, etc. Android Manifest file contains all required Permissions that are the central design point of the Android security model [14]. By default, no application has permission to access sensitive data (such as contacts, SMS) and certain system features (such as camera, internet,

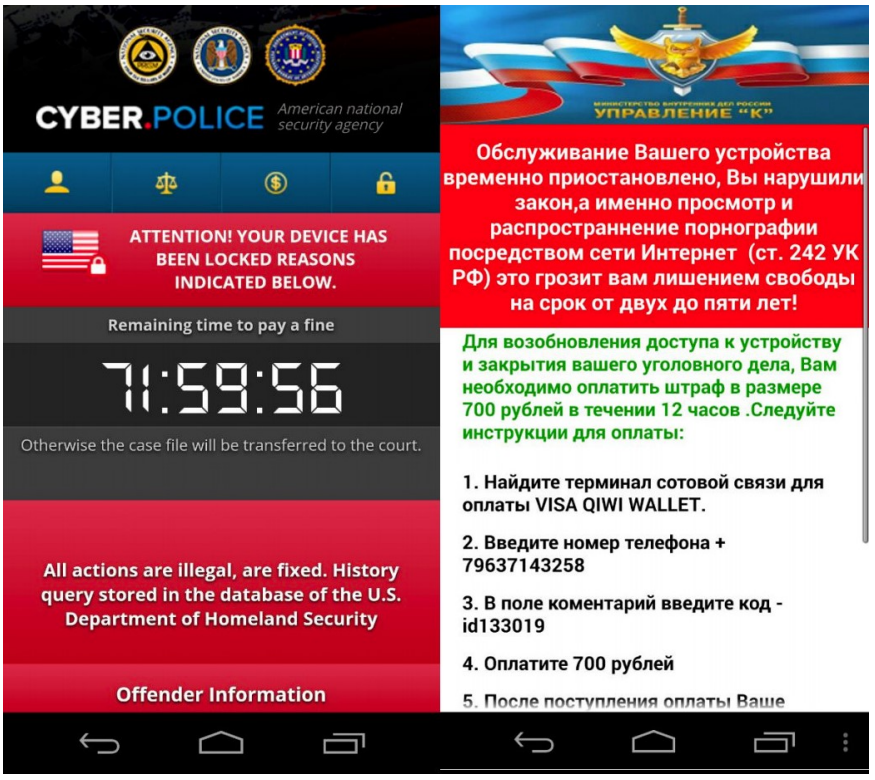


FIGURE 1.1: Screenshot of Android ransomware Attack. [15]

etc). Ransomware developers mostly exploit permissions for privilege escalation and gain access to the sensitive data stored in the device.

The dynamic analysis aims to detect malicious behavior during program execution. Dynamic analysis can deal with the features such as dynamic code loading, the sequence of system calls collected during application execution, network activity, CPU usage, and memory usage [3]. Behavioral similarities of ransomware applications can be helpful in detecting new ransomware applications.

State-of-the-art Android ransomware detection techniques [5, 10–13] usually do not consider the structural features specific to the ransomware deportment [1] e.g., text inside the source code. Ransomware can contain some specific threatening strings inside its code e.g., to lock, encrypt, porn, etc.

Another important feature that is ignored by many researchers [1, 5, 10–13] is permissions. Most of the Android ransomware require particular permissions [14] (such as BIND_DEVICE_ADMIN, KILL_BACKGROUND_PROCESS, and RECEIVE_BOOT_COMPLETED, etc.,) which can be helpful for the detection of

those ransomware [7]. Besides that, the Android ransomware regularly establish connections to the network in order to fetch commands or send back the data gathered from the device [8]. Hence, network addresses (email address, IP address, URLs) might be present in the code of several ransomware samples, which can be helpful in ransomware detection. These network-based features are never been analyzed before statically for the detection of Android ransomware [1, 5, 7, 9–13, 16].

Behavioral analysis using hardware features such as CPU usage, memory usage, and system call logs could be helpful for classification [17] of Android ransomware, since these are more resilient to modification as compared to the static features [3] that ransomware can evade through code obfuscation and encryption, etc., [18]. However, in current behavior-based detection solutions [1, 7, 9, 11] there is a lack of utilization of these features (such as System call logs, CPU and memory usage). Furthermore, Detection and classification of Android ransomware is never done before by employing a combination of all these aforementioned static and dynamic features and machine learning approaches.

In recent decades, due to the advancements in machine learning techniques a significant amount of research work done for Android malware detection utilize machine learning techniques [13, 14, 19]. In spite of the fact that machine learning techniques have manifested their adequacy in detection of malware programs [3], machine learning classifiers are not much resilient to adversarial attacks due to the property of machine learning theory that "In the learning phase data set used for training remain representative of the problem domain assuming that no intentionally harmful modification of data happens" [20]. In machine learning, adversarial attacks are the techniques that are employed to fool the machine learning model through malicious inputs. Adversarial attacks in machine learning can be categorized into two types [21] (1) Evasion attacks (2) Poisoning attacks. In evasion attacks, attackers intentionally fabricate the malicious input in such a way that the classification model misclassifies it as benign [20]. While in poisoning attacks, attackers poison the training data in order to compromise the whole learning process.

The focus of this study is to mitigate evasion attacks in Android ransomware detection such as code obfuscation, its usage to evade malware/ransomware detection. Most of the existing ransomware techniques fail due to the change in input feature vector for analysis (because if one aspect is used for obfuscation it changes whole feature vector as result the trained classifier misclassify the ransomware). Therefore, this research proposes an ensemble-based analysis mechanism to mitigate evasion attacks. With the aim that, if ransomware bypasses any of the base classifiers in the ensemble, the other base classifiers can detect it and make the correct classification. Moreover, the Android features that are considered here, are not an easy target to be modified for evasion. On the basis of this motivation, we propose a technique that combines the effectiveness of both static (Permissions, Text, Network-based features) and dynamic features (system call logs, CPU and memory usage) to detect extensive data set of Android ransomware using ensemble machine learning model thus the ransomware cannot evade the detection.

The major contributions of this research work are as follows:

1. Extraction of network features and Text from code and permissions from manifest file.

2. Extraction of CPU usage, memory usage, and system call logs by executing the ransomware and malware (non-ransomware) application samples using genymotion.

3. Trained two distinct ensemble machine learning models comprising of multiple machine learning algorithms for both static and dynamic feature sets to detect Android ransomware applications.

4. Performed evaluation of the effectiveness of hybrid distinct ensemble analysis approach to mitigate the adversarial evasion attacks on a large data set of fabricated feature vectors of Android ransomware samples.

1.1 Purpose

The purpose of this study is to explore Android features and provide a new way to detect Android ransomware that can reduce adversarial evasion attacks. This research work helps in providing a framework for effective Android ransomware detection. This framework will be a stepping stone for Antivirus vendors and malware experts to take these features and ensemble model into account to create and improve new Android ransomware detection tools and anti-viruses.

We propose a hybrid distinct ensemble learning technique that uses various static and dynamic features to mitigate evasion attacks and increase ransomware detection rate on android platform.

1.2 Problem Statement

There is a need to develop an approach that employs a combination of both effective static application features (such as permissions, text and networking features) and dynamic features (such as CPU usage, memory usage, and system call logs) to detect both *crypto* and *locker* Android ransomware and mitigate adversarial evasion attacks. The classification of Android malware into ransomware and non-ransomware is never done before using a combination of multiple static and dynamic features such as permissions, network-based features (email addresses, IP addresses, and URL), Text, system call logs and hardware features(CPU & memory usage). Moreover, machine learning based ensemble technique is not used previously for the classification of Android ransomware.

1.3 Research Questions

The problems discussed above in Section 1.2 have led us to the following research questions:

1. How to classify Android malware into RW and NRW using both static and dynamic features?
2. How to mitigate adversarial evasion attacks using ensemble machine learning?
3. How to detect zero-day ransomware?
4. Which features play a significant role in the detection of Android ransomware?
5. Which ensemble is more useful for the detection of Android ransomware?

1.4 Proposed Solution

To answer the aforementioned research questions, we propose an Android ransomware detection technique that integrates the robust features from both static and dynamic approaches in order to detect the wide range of Android ransomware from malware. Our proposed security mechanism analyze permission tags in the manifest file, static text by parsing the dissembled code and network-based features (such as URL, IP addresses and email addresses, etc.) are examined to check whether the application will communicate to any malicious server after installation. CPU usage, memory usage, and system calls are observed during dynamic testing to observe the run time behavior of the application, which is used to detect zero-day attacks. This approach covers the static as well as the dynamic behavior of applications by examining the combination of Permissions, Text, Network addressed, System calls, CPU and memory usage features of Android application that have not been used in this combination before. In order to mitigate the adversarial evasion attacks, distinct ensemble analysis mechanism is used. For both static and dynamic extracted feature sets an independent ensemble model is trained. each ensemble model classifies application based on its input feature vector and generates a label (RW/NRW). Classification results of both ensemble

models are combined by a meta classifier and a final label for application is generated. The application will be classified as ransomware if any of the ensemble analyzers assigns it RW label.

1.5 Significance of The Solution

Currently, the most extreme line of defense against ransomware is the antivirus products that recognize attacks using signature-based techniques. However, Android ransomware writers actively develop new techniques to evade current solutions. Ransomware is the most malignant among all malware. Therefore it is essential to develop a new security mechanism that is resistant and capable of reducing evasion attacks. Considering this the proposed technique employs the machine learning based ensemble analyzer that helps to mitigate the adversarial evasion attacks. The proposed mechanism perceives vulnerabilities that help to effectively classify and detect Android ransomware from other malware that are non-ransomware and less harmful. It might help to expose more security indicates that requires to be improved by antivirus solution providers. The proposed technique provides deeper insight into the Android application features to provide a framework against Android ransomware attacks. The combination of these features helps to form future prevention strategies for Android ransomware and strengthen security solutions.

1.6 Tools & Techniques

Following tools and techniques are used in this research:

1. Operating System: Ubuntu (Version LTS 16.04) 64 bit, Windows 8 64bit.
2. Machine Learning Classifiers: Naive Bayes [22], Decision Tree, Random Forest [23], Random Tree [24], Support Vector Classifier [25], Logistic Regression

[26], Adaptive Boosting [27], Gradient Boosting [28], SVM with SMO [29], Jrip [30].

3. WEKA Tool: For training and testing machine Learning classifiers [31].

4. APK Tool: To reverse engineer the .apk files [32].

5. Python script for static feature extraction: extract Permission request features from AndroidManifest.xml, Network features and Text from code.

6. Python script for feature vectors.

7. Genymotion: to run Android virtual device [33].

8. Oracle VM Virtual Box: to setup virtual machine. Prerequisite for Genymotion.

9. Microsoft Excel 2013.

Chapter 2

Background and Literature Review

2.1 Android Ransomware

Since the origination of Android operating system, its popularity has increased and keeps on doing so. Overtime, attackers considered Android as a profitable target and developed malware for Android. Various type of Android malware are there that threatens the victim differently. Android malware are classified into different categories according to their behavior and characteristics. Ransomware is one of the subcategory of Android malware.

Android ransomware is the most challenging type of malware whose ultimate goal is to achieve an economic benefit. Ransomware threaten victims to block access to the device or data unless ransom is paid. It is commonly carried out using a Trojan that tricks the user into thinking that it is a legitimate application. After running the Trojan, it invokes the ransomware to encrypt the files stored on the device for making them inaccessible to the user. At that point, it demands payment normally through cryptocurrency to unlock the device. The victim cannot open their files without paying the requested ransom, for the reason that private key required for decryption of files is normally stored on C&C server [8, 16].

2.1.1 Crypto-Ransomware & Locker-Ransomware

Generally, two types of Android ransomware are there [8]. First one is *locker ransomware* that denies access to the mobile device using a locking mechanism or a pop-up overlay on the user interface. Locker ransomware does not encrypt the data files however the user cannot use the device. It is the most common type of Android ransomware. The second type is *crypto-ransomware* that encrypts the stored files and takes over the device. It is much more common in computers however it also attacks the Android devices. Crypto ransomware demands ransom to decrypt the encrypted files, and locker ransomware demand ransom for unlocking or allowing access to the device [3, 14].

2.1.2 Families of Android Ransomware

Android ransomware started distributing in 2013. The first Android ransomware appeared in mid-2013 in Russia, UK, India, Switzerland, and Germany. It was locker ransomware and distributed through drive-by download [34]. 900,000 mobile devices were infected in the first 30 days [35]. After that Android ransomware continues to spread widely using different social engineering tactics and distribution methods such as fake app updates, third-party app stores, SMS Messages, etc. The first crypto-ransomware "Simplocker" was seen in early 2014 in USA. Simplocker encrypted the files having .jpg, .jpeg, .txt, .doc, .mp4, extensions and asked victims to pay 200-500USD through Money pack voucher. By the end of 2016, it affected 150,000 Android users [36].

In 2015 another locker ransomware named "Lockerpin" was discovered that used to gain administrative privileges and locks the device by setting a pin. According to ESET's LiveGrid, it infected android users mostly in the USA with a percentage share of 72% [37]. The rate of Android ransomware infection continues to increase by the end of 2017. The number of new emerging Android ransomware threats suggests that a large number of attackers are trying to jump in the ransomware trend by developing their own threats. Most widely recognized Android

TABLE 2.1: Android ransomware inventing year and category

Name	**Year**	**Category**
Xbot	2015	*Crypto – Locker*
Svpeng	2013	*Locker*
Slocker	2016	*Crypto – Locker*
ScarePackage	2014	*Locker*
Simplocker	2014	*Crypto*
Reveton/Police Ransomware	2014	*Locker*
Koler	2014	*Locker*
Jisut	2014	*Locker*
Fusob	2016	*Locker*
LokiBot	2017	*Crypto – Locker*
Double Locker	2017	*Crypto – Locker*
AndroidDefender	2013	*Locker*
AndroidCharger	2017	*Locker*
AndroidFlocker	2015	*Crypto – Locker*
Leaker Locker	2017	*Locker*
FakeAvast	2013	*Locker*
LockerPin	2015	*Locker*
LockDroid	2014	*Locker*

ransomware families along with their inventing year and category are listed in Table 2.1 [8].

2.1.3 Anatomy of Android Ransomware Attack

Generally, all Android ransomware attacks take place in five phases: Deployment, Installation, C&C server, Destruction and extortion as shown in Figure 2.1. In the first phase, ransomware tries to get into the target device by means of different methods such as drive-by download, phishing email, third-party app stores, and fake app updates [8].

FIGURE 2.1: Anatomy of Android ransomware attacks.

After the malicious payload is delivered to the targeted device the ransomware application tries to install itself and acquire administrative privileges. The device gets affected, then it tries to communicate with its C&C server looking for instructions. Some ransomware variants report back a significant amount of device information. In the case of crypto-ransomware, it communicates with the C&C server and obtains the encryption key, if the encryption key is not added before in its payload [14]. At this point, ransomware search for the particular type of files and folders (such as jpg, .doc, .xlsx, .pdf, etc.) and starts encrypting those files using a strong cipher. Android crypto-ransomware such as simplocker uses AES encryption scheme in order to encrypt the data present in SD card and removes all backup points [14]. Locker ransomware locks the device and extorts victim to pay a ransom in the form of untraceable payment method such as Bitcoins, money pack, Ukash cards, etc. by displaying non-detachable Ransom message, fake law, and enforcement agencies messages or continuous pop-ups. As shown in the Figure 2.2

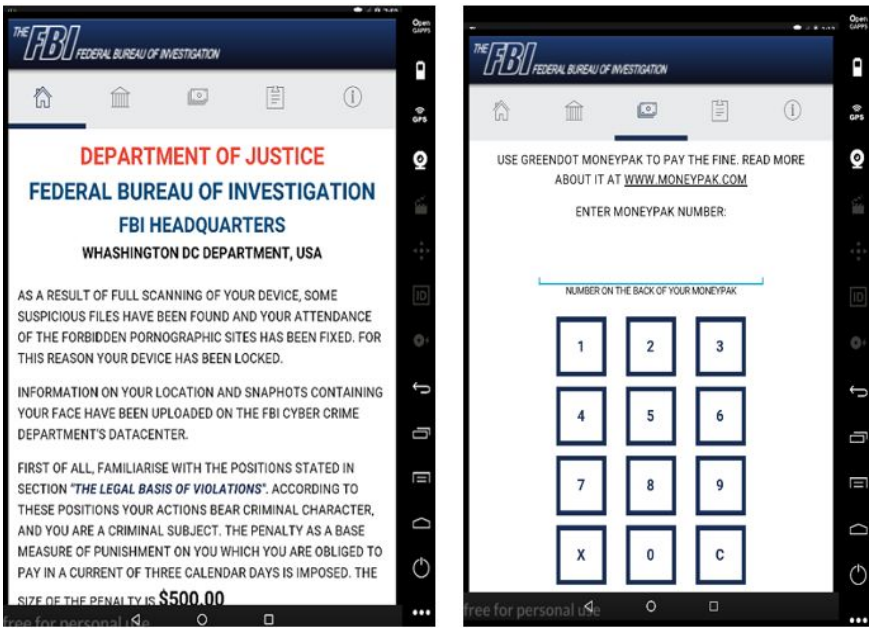


FIGURE 2.2: Screen shot of Android ransomware payment.

2.2 Literature Review

Till date, various Android ransomware detection techniques have been developed. The effectiveness and efficiency of a technique depend upon the analysis approach used to mine the unique and distinctive features that correctly represents ransomware. The purpose of analysis is to extract those explanatory features that denote maliciousness and helps in forming consequent detection strategies [13]. Analysis techniques are mainly classified into two types' static analysis and dynamic analysis. While, hybrid analysis involves both static and dynamic analyses. Both types of analyses have their own benefits and limitations. This chapter represents an extensive overview of the state-of-the-art research work done for the detection of Android ransomware

2.2.1 Static Analysis

Static analysis is a system to inspect the functionality and malevolence of an application by dissembling and exploring its source code without execution of the application. Static analysis is useful for finding malicious behavior that may

not operate until the specific conditions happen. For Android ransomware, static analysis techniques have been used to extract various representative features that help in identifying these malicious applications.

In [12], the authors described a model checking technique to identify malicious payload in Android ransomware. This technique is organized into three sub-processes (Formal Model Construction, Temporal Logic Properties construction and Ransomware family detection). In Formal Model Construction Byte code of the application is parsed and suitable formal models of the system are produced, Temporal Logic Properties construction defines the characteristics behavior of ransomware that is written as a set of properties. Ransomware family detection invokes formal verification environment including model checker to detect ransomware family. Distinguishing features of this approach are the use of formal methods and recognition of ransomware from Java Byte Code.

In [1], authors proposed HelDroid, an Android ransomware detection approach. HelDroid proposed three generic indicators: Threatening Text Detector, Encryption Detector, and Locking Detector. These independent detectors are able to be executed in parallel and each detector finds the specific indicator of compromise. Threatening Text Detector recognizes threatening phrases in statically and dynamically allocated strings. Encryption Detector checks whether the disassembled code of the sample under analysis contains traces of unsolicited file-encryption operations. Static taint-analysis track flows starting to the functions which contact the storage and ending to the functions which compose encrypted content and erase the first record. Locking Detector investigates if the application under analysis is capable of locking the device. It utilizes a firm smali emulation to discover locking capabilities and recognize the occurrence of encryption by utilizing taint analysis which is computation extensive. Their solution to for distinguish encryption was restricted to the Android well-characterized API and a malware author could easily evade the framework by using native code.

2.2.2 Dynamic Analysis

Dynamic analysis examines the application during execution. It can absolutely recognize the malicious behaviors which are not detected by static analysis technique. In dynamic analysis, the application is executed in a controlled environment to observe the real behavior of the application and the way it interacts with the underlying operating system. Dynamic analysis is more effective because it observes the actual determination of the application and resilient to evasion. Moreover, dynamic analysis can detect the unknown variants of ransomware families based on their general behavior signatures.

In [9], authors proposed DNA-Droid which was basically an instant hybrid detection framework. This framework rapidly performed static analysis on application, if it was labeled as suspicious, then the application was continuously monitored and its run-time behavior was profiled. When the profile becomes analogous to a group of malicious profiles, this framework terminates the program; otherwise, it continues to monitor the application for another five minutes. Results described that the DNA-Droid can effectively detect ransomware samples in the beginning time of their malevolent activity. The structural design of their proposed framework contains the three main components: static analysis component, the dynamic analysis component, and a detection component. The static component includes three sub-components (Text Classification Module (TCM), Image Classification Module (ICM) and API calls and permissions Module (APM)) for evaluating multiple characteristics of an APK file. Dynamic Module uses to profile malware families dependent on the API call sequences and creates a DNA for every family. In the identification stage, the run-time behavior of a sceptical sample is consistently contrasted to groups of DNA.

In [5], authors presented a ransomware prevention technique that diligently monitors and specifies processes and particular file directories using processors and memory usage, and I/O rate statistics so that the process with uncommon conduct can be distinguished. If a process is detected as doubtful then system stops the process and deletes it contingent upon the user input. After confirmation

by the user, the data of suspected and uncommon procedures was put away in the database. On the bases of data gathered from recognized ransomware, later on, damages caused by such ransomware could be prevented. The proposed technique was implemented using three components (Configuration, Monitoring, and processing). Configuration component creates observing list table, Monitoring component monitors memory, processor, and storage I/O usage of each process, Processing component decide the handling of processes that are declared suspicious by the monitoring component and makes an exemption or separation of the procedure. The detection speed of this technique was fast because unlike other techniques, rather than a mobile application it can be employed in Android source code. Without getting data about the ransomware it can lessen harms brought about by unknown ransomware.

In [11], the authors described RansomProber, a real-time detection technique for crypto-ransomware. RansomProber dissects UI (User Interface) gadgets of related actions and coordinates of the user's finger movements and recognize whether the record encryption activity is started by user or ransomware. RansomProber comprises of three steps: Encryption analysis, Foreground analysis, and Layout analysis. The encryption analysis component is used to identify whether some files are encrypted. Foreground analysis component chooses whether the encryption procedure has its place with the application that the user is associating with and UI gadgets of related actions and task directions of the user are broke down in the Layout analysis component. RansomProber can identify repackaging ransomware that attacks an application without encryption. However, fails to detect in case of repackaging with encryption or compression operations.

2.2.3 Hybrid Analysis

Although static analysis techniques are faster to Android ransomware detection, these techniques come up short against code obfuscation and encrypted or packed families (the families that use packers to compress their payloads). Likewise, dynamic analysis can miss a portion of the code sections that are not executed, this

makes dynamic analysis techniques vulnerable to evasion. Therefore employing both static and dynamic analysis techniques can be more helpful for Android ransomware detection.

In [10], authors introduced RanDroid, an automated approach that measures the structural match between the set of gathered data from the assessed application and the compromising data gathered from known ransomware variations to categorize the application as ransomware or a goodware. In static analysis RanDroid extract app's information such as images and text from XML layout files, resources, and classes.dex files. Dynamic analysis captures the extortion activities and examines the existence of threatening notes or locking screen. Image Similarity Measurement (ISM) and String Similarity Measurement (SSM) are used to find the similarity between extracted information and previously collected information of known ransomware. Based on the similarity scores the inspected application is considered as suspicious, good ware or ransomware.

In [13], authors proposed a hybrid approach for Android ransomware classification & detection that first examine the application to be used on a device prior to its installation by static analysis based on the frequency of opcode. After that dynamic analysis recognizes if the system is under attack by monitoring usage of CPU, memory, network, and system call statistics. Both static and dynamic detection involves two phases: Preprocessing and Learning. In Static preprocessing phase, numeric estimations of frequencies of opcode are acquired after that in learning phase classifiers are trained using a labeled dataset that can detect and classify ransomware application afterward. While in dynamic preprocessing phase features are extracted from execution logs of applications. Furthermore, in learning stage, classifiers are prepared to perceive execution records related with malevolent conduct. Their results revealed that the hybrid method can detect ransomware with 100% precision and have less than 4% false-positive rate.

In [38] authors presented a hybrid approach for classification and detection of ransomware from malware. At first they statically examined strings, PEview and list of DLLs functions to classify the malware as ransomware or non-ransomware

using machine learning classifiers. If the classifier classified it as non-ransomware then further dynamic analysis is performed on it. In dynamic analysis they monitored the file operations, API Calls, Registry keys and hardware performance counters and used machine learning classifier to detect ransomware. Hardware performance counters were helpful to detect and classify ransomware and non-ransomware. Their technique was Applicable for Windows platform only. Their used machine learning classifiers are vulnerable to adversarial evasion attack.

2.2.4 Critical Analysis

After a comprehensive study of state-of-the-art techniques of Android ransomware detection, we summaries the strengths and weaknesses of the current approaches in Table 2.2.

From the literature analysis it has been observed that despite the fact that Android ransomware attacks are on the ascent, we have encountered a small number of related works in literature that are addressing the issue of Android ransomware detection. Most of the previously proposed techniques either perform only static detection. [1, 10, 12] or dynamic detection [5, 11, 13].

TABLE 2.2: Critical analysis of literature review

Ref	Methodology	Strengths	Weaknesses
[1]	The approach comprises text classifier based on NLP features. The application of taint tracking for detecting file-encrypting flows. A lightweight Smali emulation technique to detect locking strategies.	Exhibit both generalization and detection capabilities, being able to efficiently detect new variants and families.	Only used threatening text detection. Ignores the other behavioral detection features such as System call statistics, which could be helpful for detection.

[9]	DNA Droid is a hybrid approach. The static approach is based on text and image classification as well as on API calls and application permissions. The dynamic analysis consists of sequences of API calls.	DNA-Droid instant detection module demonstrated high capability of detecting ransomware activity in beginnings before the infection occurs.	Dynamic detection is only applied on applications marked suspicious by the static detection. In dynamic detection, only sequence of API calls is used.
[10]	RanDroid measures the structural similarity between collected images and texts of application and predefines threatening one collected from known ransomware.	Can extract threatening messages from variants that use evasion techniques.	Misclassify the samples due to mismatch of images with different resolution, misspelled and different font styles of extracted text. Samples with foreign language text also cannot be classified by it.
[5]	This technique dynamically monitors read and write accesses to the file systems. It can detect and stop ransomware having abnormal CPU and I/O usage.	It can detect altered and new patterns of ransomware without attaining information about particular ransomware families.	It cannot detect ransomware with threatening text and locking ability. Static detection is not performed which can be more efficient.

[38]	Statically examined strings, PEview and list of DLLs functions. Dynamically monitors file operations, API Calls, Registry keys and hardware performance counters.	Hardware performance counters are helpful to detect and classify RW from NRW.	Applicable for Windows platform only. vulnerable to adversarial evasion attack.
[12]	Use model checking approach to detect Android ransomware. Analyze Java Byte Code to construct Formal models and develop logic rules to detect ransomware families.	Recognition of ransomware family without decompilation. Independence from obfuscation. Ease of parsing low-level code.	Dynamic behaviors of applications are not examined.
[11]	Perform Encryption analysis to detect file encrypted. Foreground analysis checks whether the encryption process is consistent with foreground application. Layout analysis analyze UI Widgets of related activities and operation coordinates of the user.	RansomProber can detect encrypting ransomware in real-time. Abnormal encryption operations can be detected before ransomware causes irretrievable damages.	Detect only crypto-ransomware. Sophisticated code obfuscation techniques bypass the RansomProber.

[13]	Employ a hybrid approach for Android ransomware detection. Static analysis uses the frequency of opcode. Dynamic analysis monitors CPU usage, memory usage, network usage, and system call statistics.	Sequences of opcodes and system call statistics can effectively detect Android malware.	Dataset of Android ransomware was small, which may not support different families of Android ransomware. The used Machine learning model can be vulnerable to adversarial evasion attack.

Even though static analysis is quick, secure and precise in recognizing known ransomware [3] but exclusively employing static detection could be vulnerable to ransomware attacks with code obfuscation to change their structure [12] and unable to deal with samples that encrypt or compress their payloads. While dynamic analysis is resilient to evasion [13] and can detect unknown ransomware based on the general behavioral signatures [5]. Besides that, the dynamic analysis also has some flaws such as some actions only triggers under specific conditions which may not be present in a testing environment such as emulator [3]. Therefore, utilizing behavioral detection features that are resilient to the evasion [5, 13] in combination with the effective static analysis features [5] could be worthwhile for Android ransomware detection.

Few other techniques [11] that employed hybrid approach are either type specific that deals with only one type such as crypto-ransomware [11] or only tackle a specific ransomware family. Family specific detection lacks the ability of generalizing the solution for applying it to any type of ransomware. Furthermore, previously proposed Android ransomware detection approaches [9, 13] that employed machine learning techniques can be vulnerable to adversarial evasion attacks. The whole

detection model can be compromised by the attackers merely applying code obfuscation or other evasion techniques on one aspect because tempering in one aspect can change the whole feature vector as result ransomware can remain undetected by the employed single machine learning classifier.

Chapter 3

Research Methodology

This chapter covers the methodology of our proposed research work. We discuss the details of data set, extraction of both static and dynamic analysis features and training of machine learning classifiers. We have performed our experimental analysis by using hybrid distinct ensemble analyzers to enhance the performance of learning and mitigate the evasion attacks. Section 3.1 explains the hybrid distinct ensemble analysis followed by the training phase of ensemble analyzers in Section 3.1.1, data set description in Section 3.1.2, extraction of static features in Section 3.1.3, extraction of dynamic features in Section 3.1.4.

3.1 Proposed Hybrid Distinct Ensemble Analysis

The general approach used for the proposed hybrid analyzer is the hybrid distinct ensemble analyzers. Ensemble learning uses several classification algorithms to acquire effective performance than could be achieved from any of the constituent classifiers alone. The concluding prediction is made as to the label which is predicted by the majority of classifier [39]. The proposed hybrid distinct ensemble analysis consists of two separate machine learning based ensembles, trained on static and dynamic feature vectors separately. Each ensemble model consists of

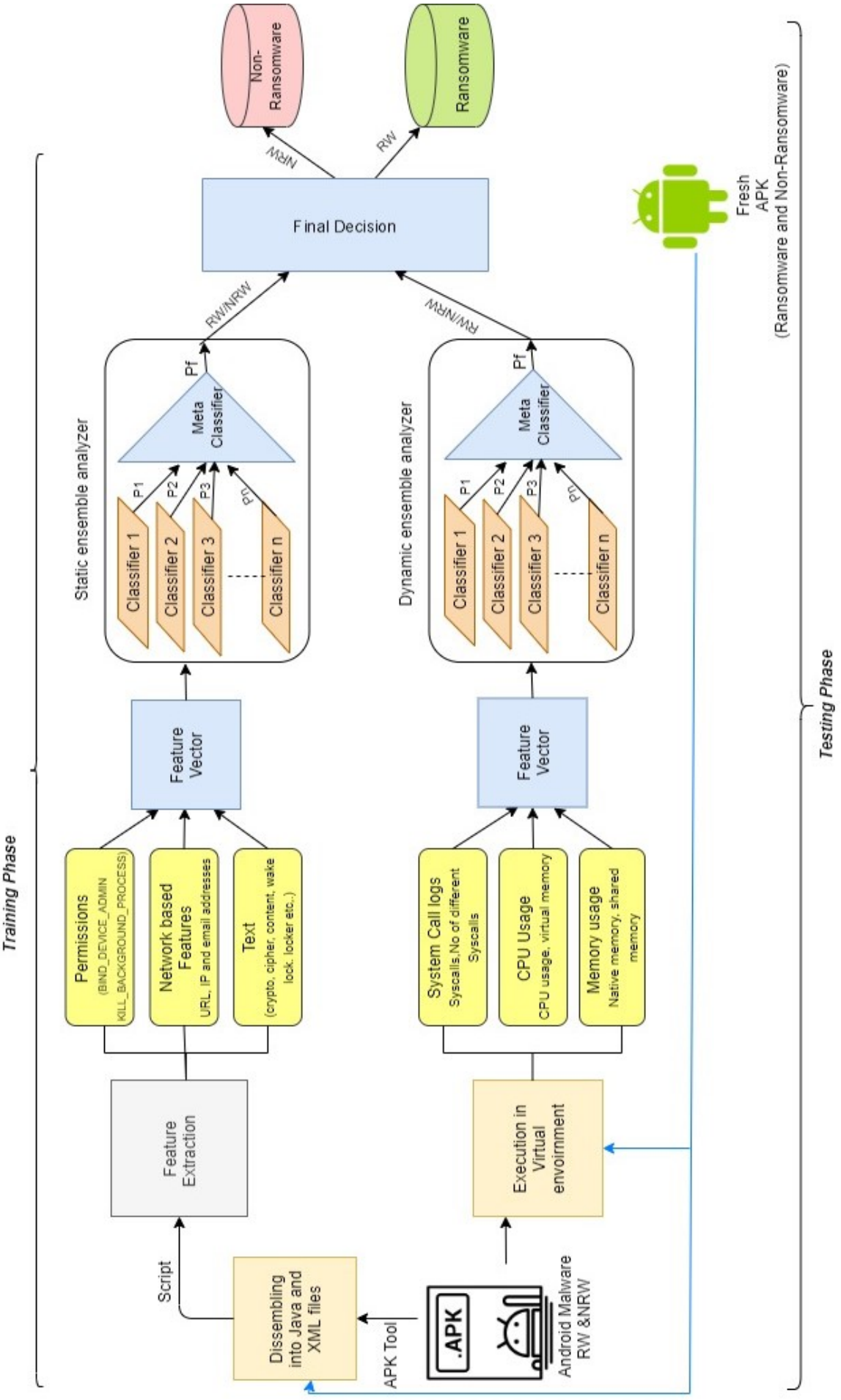


FIGURE 3.1: Architecture of the proposed methodology

an odd number of machine learning classification algorithms such as three or five. Stacking ensemble method is applied for training and testing of these classification algorithms, where each algorithm is trained on the entire feature vector. The classification results of classifiers are combined and then the final label is predicted [40]. Application labeled as RW by any one of the ensemble models is considered as ransomware. We call it hybrid distinct ensemble analyzer due to the separate ensemble machine learning models used for static and dynamic analyses data sets as explained in Figure 3.1. Both static and dynamic analysis are performed in parallel for all the applications.

The proposed hybrid Android ransomware classification and detection approach based on the ensemble learning is depicted in the Figure 3.1. We started with the set of malware APK files consisting of both Android ransomware and non-ransomware as input. Each Android application is packaged into .apk file which is, in fact, a compressed file comprising of several files and folders such as classes.dex files, assets, resources, META-INF and AndroidManifest.xml files, etc. classes.dex files contain source code of application functionality. Assets folder comprises of non-compiled resources and its directory structure is retained. Resource files hold information related to the graphical or audio components (such as images, clips, etc.). For authentication and verification of the application, its digital signature and developer's certificate are stored in the META-INF folder. AndroidManifest.xml file particularly holds metadata of an application, for example package name, desired permissions, and definition of different components such as broadcast receivers, services, activities, and minimum/maximum platform supported libraries to be connected against and so on.

In first phase, features of static and dynamic analysis are extracted from the APK file. Static and dynamic analyses are initiated at the same time. For static features extraction, the APK file is dissembled into java and XML files and for this purpose, we used Apk tool [32]. Apk tool is a freely available open source utility that decompose an APK file into its integrant resources [41]. The received Java and XML files are further scanned to extract features. Android has a particular

permission strategy; permissions are allowed by the user upon application installation [9]. These permissions are extracted from AndroidManifest.xml files. While text and network based features such as email addresses, IP addresses, and URL from java files. These network features describe to whom the application communicate after installation. Since activities of Android ransomware are mostly network-based [8] therefore these network features helps in detection of Android ransomware applications. The Figure 3.1 shows these static features extraction process.

In the second phase, these features are converted into a combine feature vector and supplies to the proposed static ensemble learning model. Static ensemble machine learning model is trained on these feature vectors. Once this static ensemble is trained it can classify the application and assigns a label RW/NRW based on the identical static features.

Likewise, for dynamic analysis, each APK file is executed in an emulation environment to record its dynamic features. The extracted dynamic features of an APK file are converted into a feature vector. These feature vectors are further used to train dynamic ensemble model. Afterword the dynamic ensemble classifies the application and assign label RW/NRW. The final decision is made on the classification results of both static and dynamic ensembles in such a way that if anyone of these two ensembles classifies application as ransomware by assigning it RW label. Then it will be classified as ransomware. The application is classified as non-ransomware only if both static and dynamic ensembles assign it the similar NRW label. Malware applications classified as ransomware are added to the Android ransomware data set while the all other applications declared as non-ransomware are added to the Non-ransomware data set.

Algorithm 1 describes the features extraction process of both static and dynamic analysis from APK files, the conversion of extracted features into feature vectors and the classification mechanism employed for the detection of Android ransomware.

Algorithm 1 Feature Extraction and Classification

INPUT: APK_{File}
OUTPUT: Malware or Ransomware

for *all* $f \in F$ **do** ▷ F is APK folder
  $APK_{File} \leftarrow Open(file)$
  $manifest_{File}$, $java_{File} \leftarrow APK\_Tool(APK_{File})$
  if $manifest_{File} == androidmanifest.xml$ **then**
    permission$\leftarrow Get_Permission(androidmanifest.xml)$
    for *all* $permission_(i) \in permission$ **do**
      if $Permission_{(list)}[i] == permission_{(i)}$ **then**
        $Vector_{(Permission)}[\,] \leftarrow 1$
      end if
      $Vector_{(Permission)}[\,] \leftarrow 0$
    end for
  end if
  $network_{vector} \leftarrow TF\_IDF(manifest_{File}, java_{File}, network_{File})$
  $Text_{vector} \leftarrow TF\_IDF(manifest_{File}, java_{File}, Text_{File})$
  $Dynamic_{vector} \leftarrow Virtual\_Environment(APK_{File})$
  $Output_1 \leftarrow Classify(network_{vector} + Text_{vector} + Vector_{Permission})$
  $Output_2 \leftarrow Classify(Dynamic_{vector})$
  $Output \leftarrow OR(Output_1, Output_2)$
end for
Return $Output$

3.1.1 Training of Hybrid Distinct Ensembles

The training phase of hybrid analyzer starts with the analysis of 50% Android ransomware and 50% non-ransomware applications in a controlled environment. The first phase is of feature extraction. Static features are extracted using a feature extraction script that takes APK file as input, disassemble it into Java and XML files and extract permissions from AndroidManifest.xml file, network-based features and text from java files. All these features are transformed into a feature vector for training machine learning base static ensemble model. Dynamic features are extracted by executing applications in Android emulator called Genymotion. CPU usage, system calls statistics, and memory usage are recorded by executing applications one by one on Android emulator.

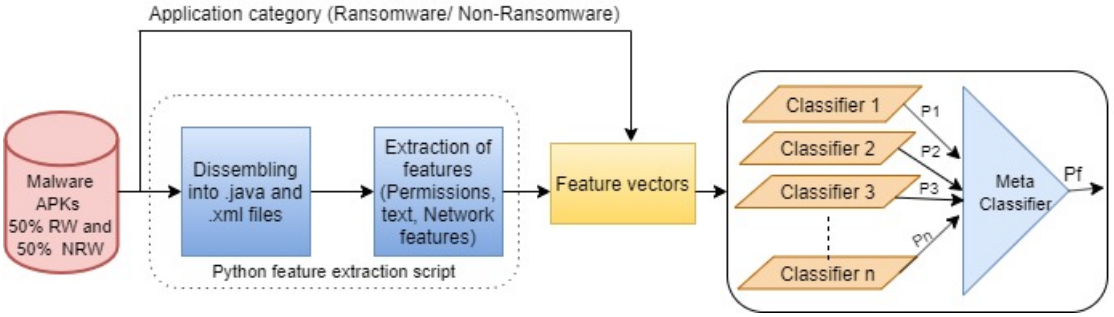


FIGURE 3.2: Training of ML based static ensemble analyzer

These features are utilized to produce a feature vector for the machine learning base dynamic ensemble model. Two separate ensemble machine learning models are used to classify applications based on their static and dynamic features as shown in Figures3.2 and Figure 3.3 Each ensemble is trained in such a way that the feature vectors along with their category (ransomware labeled as 1 and non-ransomware labeled as 0) are supplied to the ensemble. Each member classifier (e.g., Naïve Bayes, Random forest, AdaBoost, Decision Tree etc.,) in the ensemble is trained on each entire feature vector shown in the Figure3.2 and Figure 3.3. Once all these ensemble members are trained they can classify the applications and assign class labels RW/NRW. Outputs of all member classifiers are provided to Meta classifier that combines these outputs via combination rule (majority voting) for assigning the one final label. Since we are dealing with a two-class problem, therefore, we use majority voting scheme to decide the final label. Based on the outputs of both ensemble models final label is assigned to the applications. For assigning the final label "OR" operation is performed on the outputs of both ensembles. Thus, if anyone of the two ensembles assigns RW label to the application, it will be classified as ransomware.

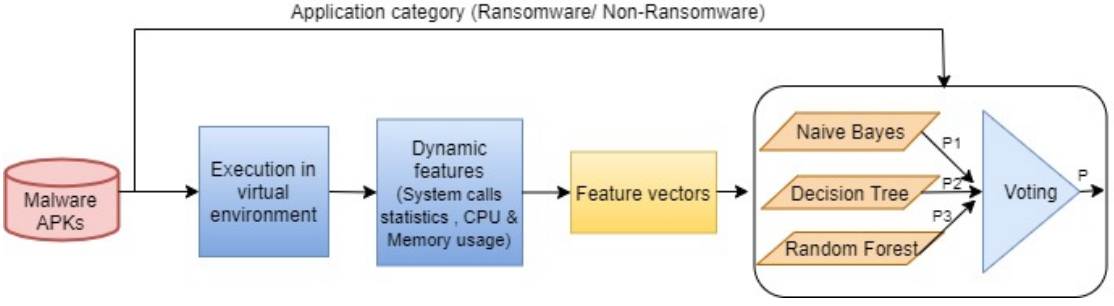


FIGURE 3.3: Training of ML based dynamic ensemble analyzer

After training, we will validate our framework on fabricated data set of Android ransomware to examine its capability of mitigating adversarial evasion attacks. To achieve the objective of training machine learning algorithms a few important steps are there. These steps are discussed in the upcoming sections.

3.1.2 Data Collection

The whole dataset used in this study comprises of two datasets: Android malware (Drebin [42]) and Android ransomware (RansomProber [43]) in the form of .apk files. For malware data set of Drebin [42] is collected. Drebin is a benchmark repository that contains more than five thousand malware applications from 179 different malware families. While the data set of Android ransomware is collected from [43] which was used for experiments in RansomProber. RansomProber dataset consists of more than two thousand samples obtained from related security announcements, threat reports from existing antivirus companies and security blogs [11]. Their data set demonstrates a good coverage of existing Android ransomware families.

TABLE 3.1: Detail of Experimented Dataset

Application Type	**Total Number of Applications**	**Applications used in experiments**
Ransomware	5500	275
Non-Ramsomware	2280	275

Table 3.1 depicts the overview of dataset used for experiments in the proposed technique. The employed classifiers were trained on the behavioral features of both randomly selected ransomware and non-ransomware applications along with the explicit labeling (Ransomware/Non-Ransomware). Disjoint sets of data are used for training and testing purpose. We have used a large number of applications to ensure that our data set is unbiased. Our used data set is not limited to the applications with certain attributes that may help in created results.

3.1.3 Extraction of Static Features

Our experimental data set consists of .apk files. Android Package Kit (APK) is a file format that Android uses to distribute and install applications. It contains all elements such as classes (.dex files), resource files and manifest files that an application needs to install correctly on the device. The manifest file contains permissions and other configuration details of the application. Our feature extraction process starts by acquiring APK files using a feature extraction script (shown in Appendix A). We have written a python script to extract permissions from manifest.xml file, text, and network-based features (IP addresses, email addresses and URL) from dex files. The script decompiles the APK files, extract these features and store them into .txt files. We further used these .txt files to extract feature vectors of required features. Detail of feature extraction script:

1. The script takes APK file as input and uses Apk Tool to disintegrate the APK file.

2. After disintegration we get .xml files, .dex files and resource files.

3. The script reads 'uses-permission' tags and extract all permissions from AndroidManifest.xml file and store these permissions in a .txt file.

4. Similarly, it uses .dex files to extract text from source code. URL, email addresses and IP addresses are extracted from dex files using regular expressions. After extraction, these features are stored in .txt files (as shown in Figure 3.5).

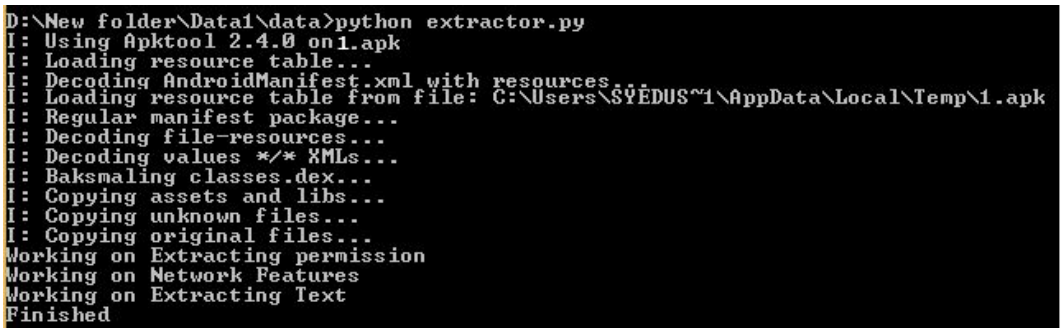


FIGURE 3.4: Python Feature Extraction script working

Working of feature extraction script is displayed in Figure 3.4. Similar process is repeated for all APK files. .txt files of both ransomware and non-ransomware applications are used for feature vector creation. We have seen that a certain number of permissions are common in ransomware applications likewise, there are some permissions that are similar in non-ransomware applications.

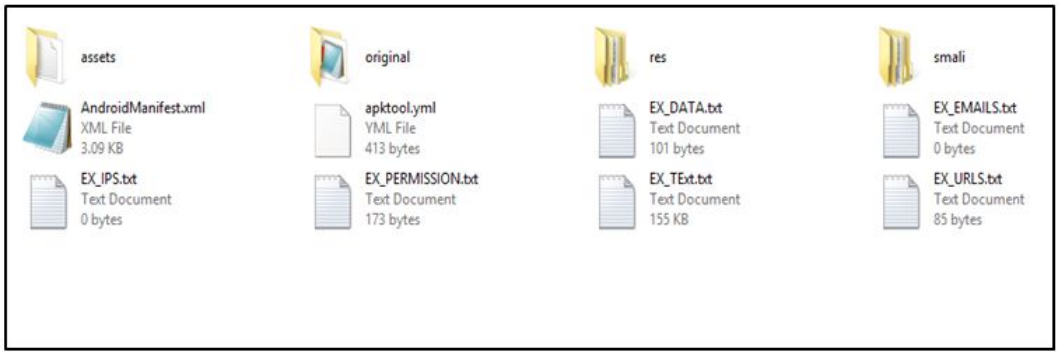


FIGURE 3.5: .txt Files of features extracted by the script

Figure 3.6 represents all those permissions that are mostly used by non-ransomware malware applications. From Figure 3.6 it can be seen that INTERNET and READ_SETTINGS have a percentage of 14, these are the most used permissions in non-ransomware applications among all other 8 permissions. Through INTERNET permission application can access the internet.

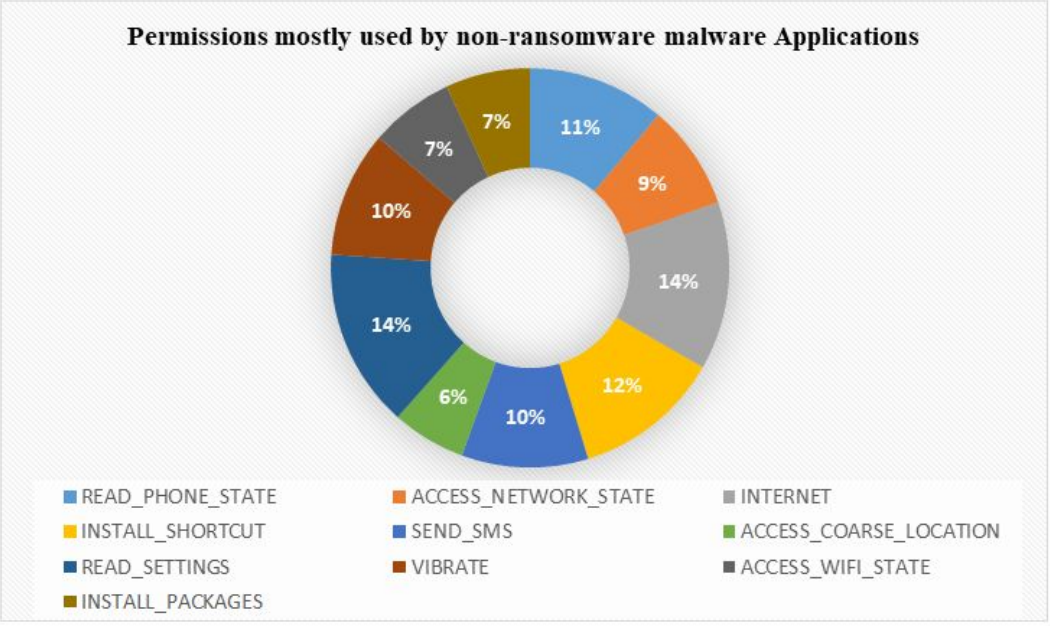


FIGURE 3.6: Ten most used permissions by non-ransomware applications

It allows the application to open network sockets. READ_SETTINGS permission allows an application to read the settings and shortcuts in the home. INSTALL_SHORTCUTS is the second most requested permission having a percentage of 12. It allows an application to install a shortcut in Launcher. READ_ PHONE _STATE provides an application requesting it with read-only access to a huge amount of data such as number of the device, IMEI, information of the SIM card like IMSI number. SEND_SMS permissions enable the application to catch any response to the initial text message and hide the exchange of text messages from user. Application asks ACCESS_NETWORK_STATE permission in order to access all information about networks. The application utilizing instant messages needs SEND_SMS permissions to send messages from device.

ACCESS_WIFI_ STATE allows the application to access information about Wi-Fi networks. INSTALL_ PACKAGES permission, if granted, allows the application to install new applications on the device. Application uses ACCESS_COARSE_ LOCATION permission to access approximate location. Most frequent permissions in ransomware applications are depicted in Figure 3.7. RECEIVE_BOOT_COMPLETED and WAKE_LOCK are the most requested permissions in ransomware applications. The percentage of these permissions is 15, highest among all.

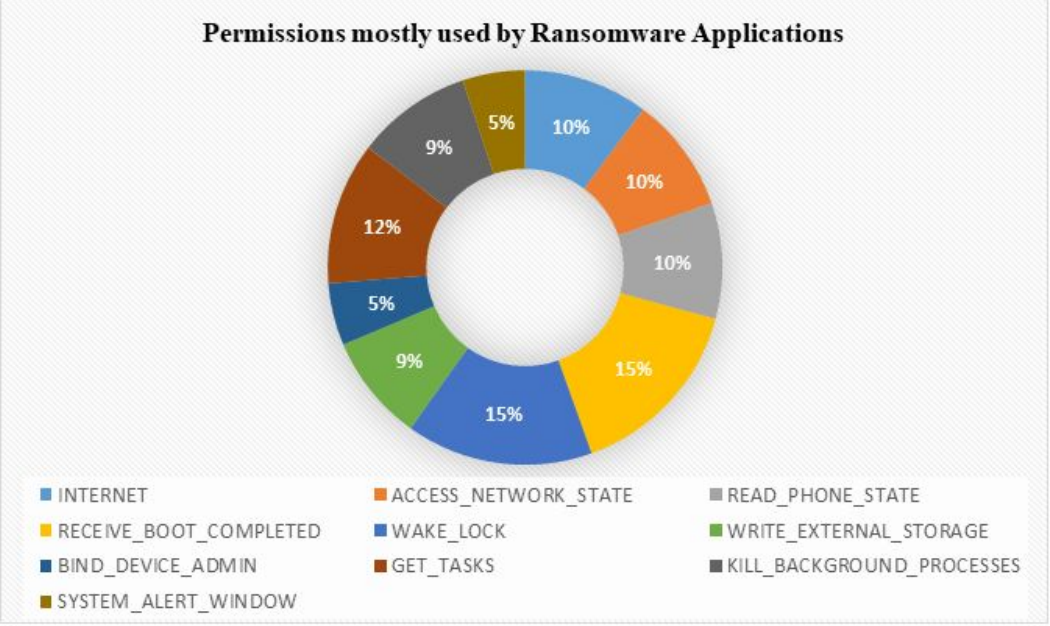


FIGURE 3.7: Ten most used permissions by ransomware applications

RECEIVE_BOOT_COMPLETED allows the application to receive the ACTION_ BOOT_COMPLETED which is broadcast after the system finishes booting. WAKE _LOCK allows using power manager wake locks to keep screen and processor from sleeping. The second most used permission is GET-TASK, through GET_TASK permission application get information about recently running tasks. Ransomware uses permissions such as KILL_BACKGROUND_ PROCESS in order to stop the running antivirus processes to prevent detection.

WRITE _EXTERNAL_STORAGE allows application to write to the external storage. BIND_ DEVICE-ADMIN ensure that the system will only interact with one application. Removal of ransomware granted with these permissions becomes much difficult. Application with SYSTEM _ALERT_WINDOW permission can create pop-up windows with advertisements while the user browses the web or write a text message. All these extracted permissions are converted into a binary feature vector. the feature vector of each application created from permissions and other static features (network addresses and text) along with application label (RW/NRW) is used to train machine learning based static ensemble model. The static feature extraction process can be seen in Figure 3.2. The training data set consists of 50% ransomware and 50% non- ransomware applications.

3.1.4 Extraction of Dynamic Features

Dynamic analysis involves the execution of Android application in the virtual environment to examine its run time behavior. It is effective to detect the malicious behavior of applications that remain undetected during static analysis. In previous research work, a lot of attention has been paid to study data leakage and sequence of API calls in dynamic analysis [7, 9]. A promising approach toward efficient dynamic Android ransomware detection is the identification of a limited set of features that provide the ability to discriminate between ransomware and non-ransomware behavior. In our proposed framework we explore such kind of dynamic features (CPU usage, System calls statistics, Memory usage) that are less

expensive and more indicative for detecting the presence of Android ransomware [13].

Hence execution traces containing this data needed to be collected by running the application in a controlled environment. These traces have been recorded by manually executing applications one at a time for the interval of 10 minutes in Android emulator. Though, some of the traces are shorter, due to the emulator minor setbacks. However, longer execution period provides us more significant results. The Android emulator Genymotion release 3.0.2 is selected for dynamic analysis of Android malware applications because it's an open-source software and supports Android studio.

The reason behind using Android emulator instead of the original device is that the emulation environment gives the more capacity to execute a large number of malicious samples, within a reasonable time, so that the obtained results have statistical significance [44]. The virtual device is reinitialized every time before running each new application to avoid the possible interventions from previously running application such as changed setting, running processes and modifications of operating system files, etc. ADB (Android Debug Bridge) is used to monitor memory usage and CPU usage of the applications. ADB is a command-line tool that allows PC to communicate with an emulator instance or Android device. Strace (a tool for tracing system calls) is used to collect system calls of applications. For dynamic features extraction the following steps are performed for each application:

1. Starting of Android Virtual Device from genymotion.
2. Installation of application on Android virtual device.
3. Obtaining Package name of the APK.
4. Making different events (e.g., swipes, presses, touch screens) during the execution of application.
5. Memory monitoring by using *'adb shell dumpsys meminfo package name'* command (shown in Figure 3.8).

6. Recording CPU usage by means of *'adb shell dumpsys cpuinfo'* command (as represented in Figure 3.9).

7. Identify and retrieve process id (pid) using the package name.

8. Collecting system calls statistics through *'strace –p pid'*. An entry point to trace system calls of the process as displayed in Figure 3.10.

9. Terminating application after 10 minutes.

10. Saving these features in a .xlsx file along with the APK name (As illustrated in Figure 3.11).

11. Uninstall of application.

```
C:\Program Files\Genymobile\Genymotion\tools>adb shell dumpsys meminfo com.soft.
android.appinstaller -d
Applications Memory Usage (kB):
Uptime: 299823 Realtime: 299824

** MEMINFO in pid 2042 [com.soft.android.appinstaller] **
                   Pss  Private  Private  Swapped     Heap     Heap     Heap
                 Total    Dirty    Clean    Dirty     Size    Alloc     Free
                ------   ------   ------   ------   ------   ------   ------
  Native Heap     6961     6428        0        0    29184    27867     1316
  Dalvik Heap     1876     1840        0        0     2256     1968      288
 Dalvik Other      373      372        0        0
        Stack      112      112        0        0
       Ashmem    35136    35136        0        0
    Other dev         4        0        4        0
     .so mmap     1189      100        0        0
    .apk mmap      356        0       16        0
    .ttf mmap      183        0      100        0
    .dex mmap      140        4      136        0
    .oat mmap     2201        0       64        0
    .art mmap      857      460        0        0
   Other mmap       38        8        0        0
      Unknown      101      100        0        0
        TOTAL    49527    44560      320        0    31440    29835     1604

App Summary
                       Pss(KB)
                        ------
           Java Heap:     2300
         Native Heap:     6428
                Code:      420
               Stack:      112
            Graphics:        0
       Private Other:    35620
              System:     4647

               TOTAL:    49527       TOTAL SWAP (KB):        0

 Objects
               Views:       24         ViewRootImpl:        2
         AppContexts:        4           Activities:        2
              Assets:        2        AssetManagers:        2
       Local Binders:        8        Proxy Binders:       18
       Parcel memory:        4         Parcel count:       18
    Death Recipients:        0      OpenSSL Sockets:        0

 SQL
         MEMORY_USED:        0
  PAGECACHE_OVERFLOW:        0          MALLOC_SIZE:        0

 Asset Allocations
    zip:/data/app/com.soft.android.appinstaller-1/base.apk:/resources.arsc: 5K
```

FIGURE 3.8: Memory usage of application

We have taken into account all the features related to system calls, memory and CPU usage that can be accessed in Android. In total there are 73 features for each running application. These features are listed in Table 3.2. There are 5 features related to CPU: three related to CPU usage and two related to the virtual memory exceptions (major and minor faults). 63 features are related to the different aspects of memory usage and 5 represents statistics of system calls. These features are further converted into a feature vector for the purpose of classification.

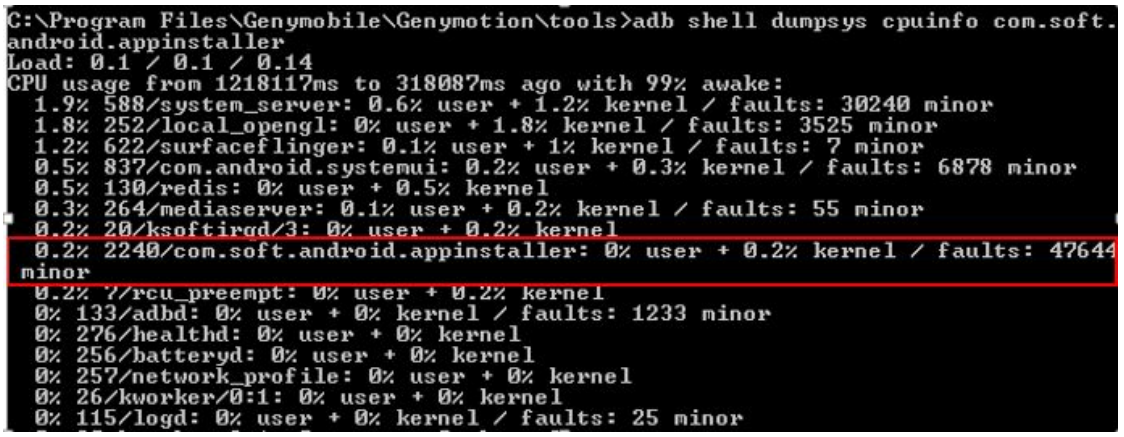


FIGURE 3.9: CPU usage during application execution

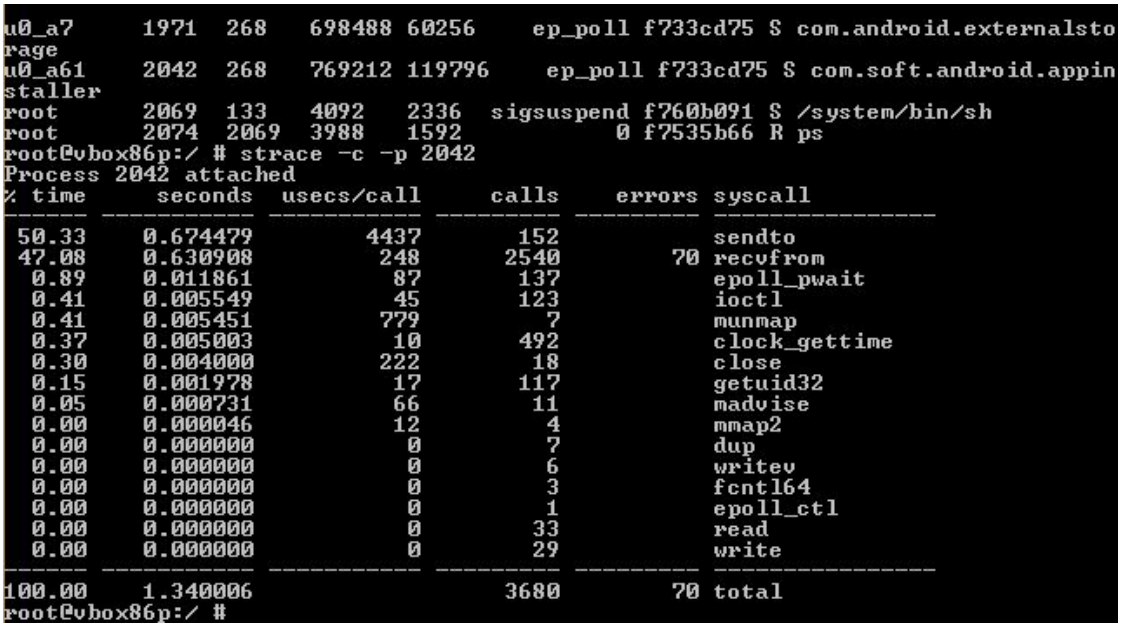


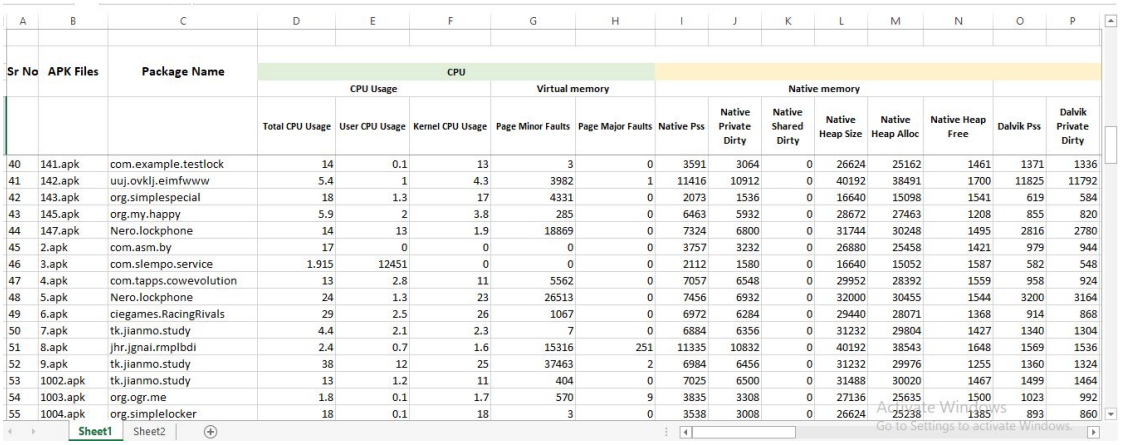
FIGURE 3.10: System calls log

TABLE 3.2: List of collected dynamic features

Category	**Feature Names**	
CPU	CPU usage	Total CPU Usage
		User CPU Usage
		Kernel CPU Usage
	Virtual Memory	Page Minor Faults
		Page Major Faults
Memory	Native memory	Native Pss
		Native Private Dirty
		Native Shared Dirty
		Native Heap Size
		Native Heap Alloc
		Native Heap Free
	Dalvik memory	Dalvik Pss
		Dalvik Private Dirty
		Dalvik Shared Dirty
		Dalvik Heap Size
		Dalvik Heap Alloc
		Dalvik Heap Free
	Android shared memory	Ashmem Pss
		Ashmem Private Dirty
		Ashmem Shared Dirty
	Memory-mapped native Code	.so mmap Pss
		.so mmap Private Dirty
		.so mmap Shared Dirty
	Memory mapped Dalvik	.dex mmap Pss
		.dex mmap Private Dirty
		.dex mmap Shared Dirty
	Memory-mapped fonts	.ttf mmap Pss
		.ttf mmap Private Dirty
		.ttf mmap Shared Dirty

	Other memory-mapped files	.oat mmap Pss
		.oat mmap Private Dirty
		.oat mmap Shared Dirty
		.art mmap Pss
		.art mmap Private Dirty
		.art mmap Shared Dirty
		.apk mmap Pss
		.apk mmap Private Dirty
		.apk mmap Shared Dirty
		Other mmap Pss
		Other mmap Private Dirty
		Other mmap shared Dirty
	Non-classified memory	Unknown Pss
		Unknown Private Dirty
		Unknown Shared Dirty
		Other dev Pss
		Other dev private Dirty
		Other dev Shared Dirty
	Memory Totals	TOTAL Pss
		TOTAL Private Dirty
		TOTAL Shared Dirty
		TOTAL Heap Size
		TOTAL Heap Alloc
		TOTAL Heap Free
	Objects	Views
		ViewRoot Impl
		App Contexts
		Activities
		Assets
		AssetManagers
		Local Binders

		Proxy Binders
		Parcel Memory
		Parcel Count
		Death Recipient
		OpenSSL Sockets
	SQL	MEMORY-USED
		PAGECACHE-OVERFLOW
		MALLOC-SIZE
System calls logs		Total number of Syscalls occuring
		Name of all syscalls occuring
		No. of times each system call occurring (once or multiple times)



Sr No	APK Files	Package Name	CPU					Native memory							
			CPU Usage			Virtual memory									
			Total CPU Usage	User CPU Usage	Kernel CPU Usage	Page Minor Faults	Page Major Faults	Native Pss	Native Private Dirty	Native Shared Dirty	Native Heap Size	Native Heap Alloc	Native Heap Free	Dalvik Pss	Dalvik Private Dirty
40	141.apk	com.example.testlock	14	0.1	13	3	0	3591	3064	0	26624	25162	1461	1371	1336
41	142.apk	uuj.ovklj.eimfwww	5.4	1	4.3	3982	1	11416	10912	0	40192	38491	1700	11825	11792
42	143.apk	org.simplespecial	18	1.3	17	4331	0	2073	1536	0	16640	15098	1541	619	584
43	145.apk	org.my.happy	5.9	2	3.8	285	0	6463	5932	0	28672	27463	1208	855	820
44	147.apk	Nero.lockphone	14	13	1.9	18869	0	7324	6800	0	31744	30248	1495	2816	2780
45	2.apk	com.asm.by	17	0	0	0	0	3757	3232	0	26880	25458	1421	979	944
46	3.apk	com.slempo.service	1.915	12451	0	0	0	2112	1580	0	16640	15052	1587	582	548
47	4.apk	com.tapps.cowevolution	13	2.8	11	5562	0	7057	6548	0	29952	28392	1559	958	924
48	5.apk	Nero.lockphone	24	1.3	23	26513	0	7456	6932	0	32000	30455	1544	3200	3164
49	6.apk	ciegames.RacingRivals	29	2.5	26	1067	0	6972	6284	0	29440	28071	1368	914	868
50	7.apk	tk.jianmo.study	4.4	2.1	2.3	7	0	6884	6356	0	31232	29804	1427	1340	1304
51	8.apk	jhr.jgnai.rmplbdi	2.4	0.7	1.6	15316	251	11335	10832	0	40192	38543	1648	1569	1536
52	9.apk	tk.jianmo.study	38	12	25	37463	2	6984	6456	0	31232	29976	1255	1360	1324
53	1002.apk	tk.jianmo.study	13	1.2	11	404	0	7025	6500	0	31488	30020	1467	1499	1464
54	1003.apk	org.ogr.me	1.8	0.1	1.7	570	9	3835	3308	0	27136	25635	1500	1023	992
55	1004.apk	org.simplelocker	18	0.1	18	3	0	3538	3008	0	26624	25238	1385	893	860

Sheet1 Sheet2

FIGURE 3.11: Dynamic features sheet

3.1.5 Static Feature Vector

.txt files of Permissions, Text, IP, URL, and email are used for feature vector creation. Feature vector script read .txt files of both ransomware and non-ransomware applications and print feature vectors of each application after getting a record of all features to identify unique features of each application and put the record in the output .csv file. Let V be a vector consisting of a set of 400 Android permissions. V= S1, S2, ..., Si

Si= 1 if *ith* permission exists

0 otherwise

For every application at its location in the data set, we produce a binary sequence. All the recognized unique permissions are then arranged as a sequence of 0 and 1. The presence of a specific permission is denoted by 1 and the absence is denoted by 0 in the list. Following sequence represents an example of the permission vector for Android ransomware application.

1 1 0 1 0 1 1 1 0 1 1 0 1 0 1 0 1 0 1 1 1 0 1 1 1 0 1 0 1 1 0 1 0 1 0 1 1 0 0 1 1 0 1 1
0 1 1 1 1 0 1 0 1 0 1 1 1 1 0 0 1 0 1 1 0 0 1 0 1 1 0 1 0 0 0 0 0 0 1 0 0 0 0 1 1 0 1 1
1 1 0 1 0 1 0 0 0 0 0 1 0 0 0 1 0 1 0 0 1 0 0 0 1 0 0 0 0 0 1 0 0 1 0 1 1 1 0 0 1 0 0 1
0 0 1 0 0 1 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 1 0 0 0 1 0 0 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 1
0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 0 0 0 1 0 0 1 0 0 0 0 1 0 0 0 1 0 1 0 1

Last bit of vector represent the category of application. in the above example, 1 at the end of the feature vector represents that the permission vector is of a ransomware application. In case of non-ransomware last bit will be 0. We removed all the redundant permissions from data set because redundancy can create adverse effects in classification. After removing redundant permissions, we obtained 166 unique permissions.

We created feature vectors of text and network-based features by mean of TF-IDF vectorizer as both contain strings [45]. TF-IDF vectorizer transforms textual features into feature vectors that can be used as input to the classification algorithm. The TFIDF is more extravagant and successful representation for textual data classification purpose [46]. The static feature vectors along with their application category (1/0) are used to train machine learning based static ensemble analyzer.

3.1.6 Dynamic Feature Vector

Multiple features of each application shown in Table 3.2 obtained during dynamic analysis are transformed into numeric feature vectors. The presence of a feature is denoted by its value, and 0 in case of absence of feature. These feature vectors

along with application category are used to train machine learning based dynamic ensemble analyzer that classifies Android malware applications into ransomware and non-ransomware in the given data set.

3.1.7 Feature Selection using InfoGain

There are huge numbers of features in the data set, and it can happen that irrelevant and redundant features confuse classifiers and decrease detection performance [47]. Therefore it is needed to diminish those features in order to find the most relevant among them that helps to solve predictive modeling problems. For this purpose, we performed features selection based on the information gain criterion [48] to find the most appropriate features by assigning weights to the information, to emphasize the effectiveness of the feature. We evaluated feature usefulness by Information Gain Attribute Evaluator a method from WEKA machine learning tool. We selected 72 out of 2911 after applying feature selection algorithm based on the static results and selected 45 out of 130 based on the dynamic results. The information gain of an attribute C on sample data S can be calculated as below:

$$InfoGain(C, rj) = entropy(C) - entropy\ (C—rj)$$

3.1.8 Feature Selection using PCA

In certain applications, it is desired to pick a subset of unique features rather than finding a mapping that utilizes all of the features. The advantages of using subset of features could be the reduction in computing cost and elimination of noisy features while keeping their information by utilizing clean features. The other feature selection algorithms are either computationally expensive or select subset of redundant features. Consequently, we performed feature selection using Principal Component Analysis (PCA). PCA is a dimensionality reduction algorithm that allows us to recognize the correlations and patterns in the dataset [49]. So the dataset can be transformed into low dimensional dataset by removing these

correlations and without any loss of important information. PCA is a mathematical procedure that converts a number of correlated variables into small number of uncorrelated variables known as principal components. This small subset of uncorrelated variables is a lot more feasible to recognize and practice in analysis as compared to the large set of correlated features [50].

3.2 Classifiers used for Training

The obtained static and dynamic features (as shown in the Figure 3.2 and Figure 3.3) are used to train static and dynamic ensemble machine learning analyzers respectively. Selection of correct member classifiers in the ensemble for training is the most critical phase of our research. Previous studies [9, 13, 17] suggested us different classifiers based on their accomplished results. Therefore we selected multiple classifiers including Naive Bayes [22], Random forest [23], Decision Tree (J48) [51], Support Vector Machine (SVM), Logistic Regression, AdaBoost [27], etc., for the proposed Android ransomware detection framework. In our research work we have used supervised learning. Supervised learning is based on label data set. Thus we have an initial data set where data samples are mapped to the correct outcomes.

We used data set of 550 Android malware applications (50% ransomware and 50% non-ransomware) and performed 10-fold cross-validation utilizing the whole data set. In the matter of 10-fold cross-validation, the data set is distributed into 10 parts, and in each cycle one part comprising of nine folds, is taken into consideration as a training set and the rest of the part is utilized as a test set. This technique is reiterated ten times, each time a different training and test set is used. After performing ten iterations the average detection performance is declared. Cross-validation provides the greatest accuracy of implementation [52]. We have used WEKA (Waikato Environment for Knowledge Analysis) tool for the training of classifiers. WEKA is a collection of machine learning algorithms for data mining tasks. It is an open-source software written in java, issued under the

GNU General Public License. It contains tools for data preparation, classification, regression, and visualization, etc., [31].

3.2.1 Naive Bayes

Naive Bayes is probabilistic classifier. It applies probability theory and Bayes theorem for making assumptions that features are independent [22]. It is particularly suited where the dimensionality of the input is high. Despite its simplicity, it has proven to be fast, accurate and reliable. It often outperforms more sophisticated classification methods [53]. Naive Bayes is incorporated in our research work because of its success being mentioned in malware classification studies [13] from several years.

3.2.2 Decision Tree (J48)

We have used another machine learning algorithm Decision Tree to find out the way the attributes-vector behaves for a number of samples. Also on the basis of the attribute value, the values for the newly generated samples are being found. The value of attribute may lie in the data set of dependent and independent variables. Dependent variables indicate the attributes to be predicted while independent variables indicate the attributes that help in the prediction of dependent variables [51]. Decision Tree generates rules for the prediction of the target variable. With the help of a tree classification algorithm, the critical distribution of data is easily understandable. J48 (i.e., an open-source java implementation of C4.5 release in WEKA data mining tool) performs accounting of missing values, decision Tree pruning, continuous attribute value ranges, derivation of rules, etc.,[54]

3.2.3 Random Tree

Random tree is a tree that is built randomly from a lot of potential trees that have k number of random features at every node. Here in this situation “At random”

implies that in the set of trees every individual tree has an equivalent stroke to be sampled. Then it can be said that each tree has "Uniform" dispersion. Random trees can be produced proficiently and the integration of huge collections of random trees prompts exact models [24].

3.2.4 Random Forest

A random forest consists of multiple random decision trees. Two types of randomness are built in the trees. Firstly, each tree is built on a random sample from original data. Secondly at each node a subset of features are randomly selected to generate the best split [23]. From the previous studies it has been observed that Random Forest has outstanding accuracy appeared to be so far among the currently used classification algorithms due to its effective execution on large data set and eventually estimation of important variables in classification [55].

3.2.5 Support Vector Classifier

Support vector classifier (SVC) is a supervised machine learning algorithm that can be employed for both classification and regression purposes. The objective of SVC is to fit the provided data, running the best fit hyper plane that categorizes the provided data. After getting the hyper plane features can be fed to the classifier to see what the predicted class is. For multi-classes SVC uses one verses one strategy [25].

3.2.6 Logistic Regression

Logistic regression (LR) is a linear classifier that computes the restrictive probabilities of possible results and selects the one with the greatest probability. This approach is commonly used in various fields. Logistic regression converts its output using the logistic sigmoid function to restore a probability value which would then be able to be mapped to at least two discrete classes [26].

3.2.7 Adaptive Boosting (AdaBoosting)

Boosting is a common ensemble method that generates a strong classifier from various weak classifiers. This is done by constructing a model from training data, then making a model to rectify the errors from the principal model. Models are added until the training set is perfectly predicted without any error or extreme numbers of models are added. AdaBoost works by weighting the observations, it puts more weight on problematic or hard to classify samples and less on those that are effectively handled before.

The addition of new weak learners is made sequentially that focus their training on the most difficult patterns [27]. AdaBoost is a Meta classification algorithm that can be utilized with numerous different algorithms to boost their performance by consolidating their results into a weighted entirety that represents the concluding output. Here we used AdaBoost M1 with SVM base for ensemble evaluation since it achieves better performance than the AdaBoost with another kind of weak learners [56].

3.2.8 Gradient Boosting

Gradient boosting (GB) used to train numerous models in a progressive, additive and sequential manner. AdaBoost and gradient boosting differs only from shortcoming identifications of weak learners. Adaboost identify the shortcoming by using high weight data points while gradient boosting perform same by using gradients in the loss function. The output of different weak learners is combined in such a way that its loss function can be optimized [57]. Loss function is a measurement of how good the predictive model is at classifying the underlying data. Gradient boosting allows optimizing the loss function by adding weak learners in gradient decent procedure. GB works great with numeric values, as it requires no data pre-processing. It is capable of handling missing data and often provide predictive accuracy that cannot be beat [28].

3.2.9 Support Vector Machine with Sequential Minimal Optimization

Support Vector Machine (SVM) examines identifies and matches patterns of data for classification purpose. It uses hyperplane to divide data into regions of n-dimensional space. The hyperplane keeps the values of a margin between the regions to the maximum. SVM uses a kernel function that leads to a non-linear classification surface instead of a linear hyperplane [58]. Sequential Minimal Optimization (SMO) is an iterative algorithm utilized for resolving optimization problems that arise in training phase Support Vector Machine (SVM). SMO perform fragmentation of the problem into a sequence of smallest possible sub-problems, which are then resolved analytically [29].

3.2.10 JRip

JRip is a rule-based classification algorithm. It devices a proportional rule learner called as "Repeated Incremental Pruning to Produce Error Reduction (RIPPER)" to extract the rule directly from data and utilizes consecutive covering algorithms to make requested rule lists. The algorithm experiences four stages. (1) Growing a rule (2) Pruning (3) Optimization and (4)Selection [30].

3.3 Evaluation Measurement

For performance evaluation of different ensemble analyzers we use the following metrics.

Accuracy:

We have used accuracy for accessing results. It is the fraction of total number of applications correctly classified as ransomware or non-ransomware. The accuracy of a detection mechanism can be calculated as Equation (3.1) [10] [19].

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.1}$$

Precision: Precision is the fraction of the predicted correctly classified applications to the total of all applications that are correctly real positive. It can be calculated from Equation (3.2) [12].

$$Precision = \frac{TP}{TP + FP} \tag{3.2}$$

Recall: The recall is a fraction of the predicted correctly classified applications to the total number of applications classified correctly or incorrectly. Recall can be determined from Equation (3.3) [12].

$$Recall = \frac{TP}{TP + FN} \tag{3.3}$$

F-Measure: F-Measure is the harmonic mean of precision and recall. It symbolizes the capability of the model for making fine distinctions. F-Measure of a detection model can be computed by means of Equation (3.4) [12].

$$F\text{-}Measure = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{3.4}$$

Chapter 4

Results and Discussion

4.1 Introduction

In this chapter, we will evaluate our proposed framework of Android ransomware detection based on the ensemble machine learning techniques to mitigate the evasion attacks. We have used improved hybrid analysis to arrange the data set for the training of machine learning based hybrid distinct ensemble analyzer. We will talk about the data set preparation through hybrid analysis, data set, feature selection, experimental setup and performance of the proposed framework by evaluating the results obtained.

4.2 Experimental Setup

Our proposed framework is based on a hybrid analysis that requires good computational resources for both static and dynamic analysis. Both static and dynamic analyses have particular requirements for computational resources. The dynamic analysis require more resources when contrasted with the static analysis. The Machine used for experiments and its specification is shown in Table 4.1.

TABLE 4.1: System configuration

CPU	Intel® Core™ i5-5200 CPU @ 2.20GHz
Memory	4 GB
OS	Window 8
APK Decompilation Tool	Apk Tool
Data mining Tool	Weka 3.8.3
Android Emulator Configuration	
Platform	Genymotion 2.12.2
Device	Custom Tablet
Android version	6.0.0
API Level	21
CPU	1.5 GHz
Memory size	2048 MB
Data Disk Capacity	16384

4.3 Dataset

In this research work the data set used for experiments, comprised of two categories of Android malware applications: ransomware and non-ransomware. Android ransomware applications were collected from RansomProber data set [43]. The collected ransomware data set consists of 2280 samples that demonstrate a good coverage of existing Android ransomware families. Android malware (non-ransomware) applications were collected from Drebin data set [42]. Which consists of 5500 malware applications based on the 178 Android malware families. The data set contains multiple variants of each family. We performed experiments on the data set of randomly selected 550 Android malware applications comprising of 50% ransomware and 50% non-ransomware applications. We set up two types of experimental environments for static and dynamic analyzers considered here. Detail of both experimental setups can be seen in Chapter 3. The features extracted from both analysis are recorded in the form of Boolean vectors. For the training and

testing purpose, we used 10-fold cross-validation methods. Feature count obtained as a result of static analysis is 2911 however dynamic feature count is 130.

4.4 Feature Selection

Better classification results can be achieved if we add more features to the data set. Yet, some times keeping a large number of redundant features not only increases the learning time but also affect the reliability and accuracy of the classification rate obtained. Irrelevant and redundant features can confuse classifiers and decrease the detection rate. Therefore the reduction in high-dimension of feature instances by removing irrelevant features is an essential requirement. We performed a separate feature selection phase to select those attributes of the data set which are most appropriate and helpful in identifying application class (RW/NRW). Before performing feature selection we have cleaned our data set by removing redundant features. Normally the decision of keeping or removing a specific set of features relies upon the platform which provides that features. In this way, while performing feature selection we have given more consideration to the features provided by the Android platform.

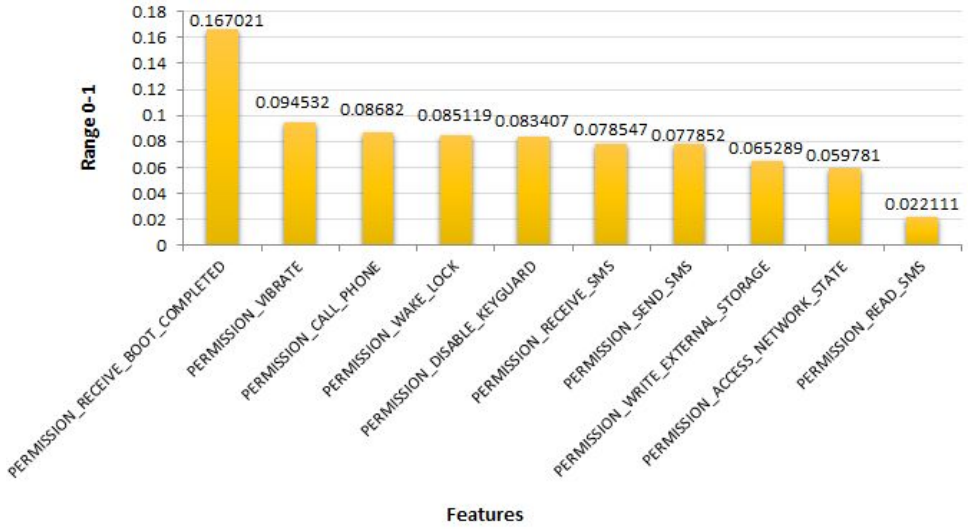


FIGURE 4.1: Top-ranked static features by InfoGain method

For this purpose we used Information gain (IG) method and Principal Component Analysis (PCA) (as discussed in Section 3.1.7 and Section 3.1.8) to get the list of most appropriate features who have significant role in classification. Information gain is a feature ranking technique which depends on decision trees that manifest a great classification performance. InfoGain selected total 72 features out of 2911 features produced by the static analysis. These features consist of permissions, network addresses, and text. Figure 4.1 describes the top ten ranked features obtained through the *InfoGain* method from the whole static data set of. All these features play an important role in the detection of Android ransomware as they retain the maximum information for classification. From Figure 4.1, it can be depicted that the PEMISSION_RECIEVE_BOOT_COMPLETEED has the highest rank among all other features, similarly, PERMISSIOM_VIBRATE has the second-highest rank and PERMISSION_READ_SMS has the lowest rank among all other nine features. The maximum range of ranked static features is 0.17 which is not much significant. The brief description of top-ranked static features obtained through *InfoGain* method is provided in Table 4.2.

TABLE 4.2: Description of top ranked static features

Feature Name	**Description**
PERMISSION_RECEIVE_BOOT _COMPLETED	Allows an application to receive the ACTION_BOOT_COMPLETED which is broadcast after the system finishes booting.
PERMISSION_VIBRATE	Allows access to vibrator.
PERMISSION_CALL_PHONE	Allows application to initiate a phone call without going through the dialer user interface for the user to confirm the call.
PERMISSION_WAKE_LOCK	Allows using Power Manager Wake Locks to keep processor from sleeping or screen from dimming.

PERMISSION_DISABLE_ KEYGUARD	Allows an application to disable the key guard if not secure.
PERMISSION_RECEIVE_ SMS	Allows application to intercept any responses to the initial text message.
PERMISSION_SEND_SMS	Applications leveraging text messages need this permission to send them from the device.
PERMISSION_WRITE_ EXTERNAL_STORAGE	allows application to write to the external storage
PERMISSION_ACCESS_ NETWORK_STATE	Allows application to access all information about network.
PERMISSION_READ_SMS	Allows application to intercept and send text messages and hide the text message exchange from the user.

Top-ranked dynamic features with their corresponding scores obtained from *info-Gain* method are represented in Figure 4.2.

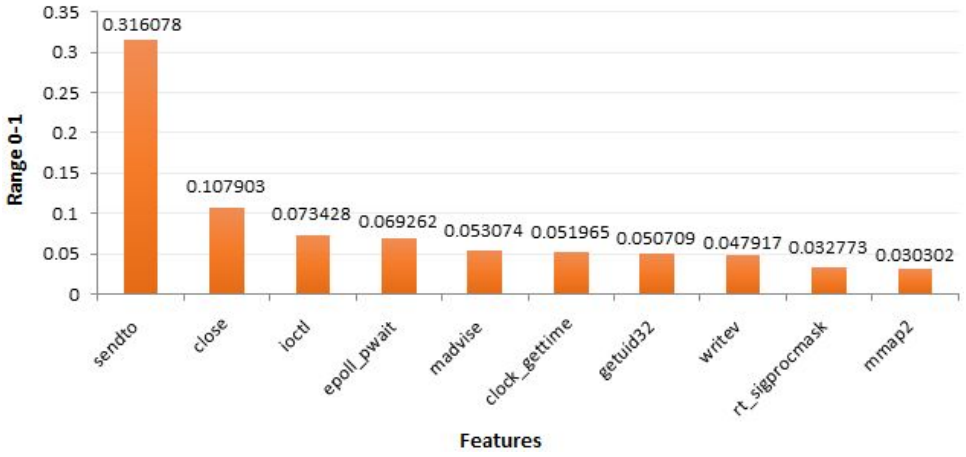


FIGURE 4.2: Top-ranked dynamic features by InfoGain method

Out of 130 dynamic features 45 are selected by *InfoGain* method. From the figure, it can be clearly seen that sendto obtained the highest rank which is 0.35 through

InfoGain method.It is system calls based feature obtained through dynamic analysis. Table 4.3 indicates the role of each selected dynamic features in Android application. It can be seen that system calls play a vital role in the classification and detection of Android ransomware.

TABLE 4.3: Description of top ranked dynamic features

Feature Name	Description
sendto	Used to send a message to another socket.
close	Closes a file descriptor, so that it no longer refers to any file and may be reused. Any record locks held on the file, it was associated with, and own by the process, are remove.
ioctl	Manipulates the underlying device parameters of special file. Particularly, many operating characteristics of character special files may be controlled with this request.
epoll_pwait	Wait for an I/O event on an epoll file referred to by the file descriptor.
madvise	Give advice or directions to the kernel about the address range beginning at address addr and with size length bytes. The goal of advice is to improve application performance.
clock_gettime	Retrieve the time of specified clock.
getuid32	Return the real user ID of the calling process. Supporting 32-bit IDs.
writev	Write data into multiple buffers.
rt_sigprocmask	Fetch or change the signal mask of the calling thread. Signal mask is the set of signals whose delivery is currently blocked by the caller.
mmap2	Map files or devices into memory.

4.5 Classification

In this stage, we examined the classifier used over the data set of Android malware (both ransomware and non-ransomware). The purpose behind doing this is to find out an appropriate classifier ensemble that can distinguish between ransomware and non-ransomware applications using feature patterns and will be helpful in mitigating evasion attacks. We have considered multiple classification algorithms including Naïve Bayes [22], Random forest [23], Decision tree (J48) [51], Support Vector Classifier (SVC) [25], Logistic Regression [26], Gradient Boosting [27] and Ada boosting [57], etc., to be used in different combinations in ensemble for evaluation of this approach. We assess optimal parameters for each classifier by experiments and using both training and testing results. These optimal parameters produce great classification accuracy on the bases of 10-fold cross-validation.

Cross-validation is a technique to evaluate machine learning algorithms by partitioning the original data set into two sets: training set is to train the algorithm and test set is to evaluate the algorithm. In k-fold cross-validation, the original data set is randomly partitioned into k equal size subsets. From the k subsets, a single subset is held out as a validation set for testing the algorithm. The remaining k-1 subsets are used as training set. The cross-validation process is then repeated exactly k times (the number of folds). Hence each of the k subsets is used exactly one time as validation set [52]. The benefit of using this strategy is that all observations are used for both training and testing, and each observation is used for validation exactly once.

After the k experiments, the weighted average of the classification accuracy is calculated, which indicates the suitability of the parameters of classifier. At last, we select the algorithm that produces the best results in the form of the optimal values of Precision, Recall and F-Measure (as discussed in Section 3.3).

4.6 Classification Results with Feature Selection

Based on the values of True positives (TP), False positives (FP) and False Negatives (FN) we have calculated precision, recall and F-measure for the static, dynamic and hybrid data respectively. These measurements were made by evaluating each single classification algorithm as well as different combinations of ensemble learning.

4.6.1 Classification Results of Top Ranked Features from InfoGain

Figure 4.3 and Figure 4.4 shows effects of data set shuffling on the F-Measure during testing and training, and the results of precision and recall of classification using single classifier and different ensembles on selected static data set through *InfoGain* method. Precision and recall values show the same range due to the nature of the employed data set. The Figure 4.3 displays that single classifiers J48,

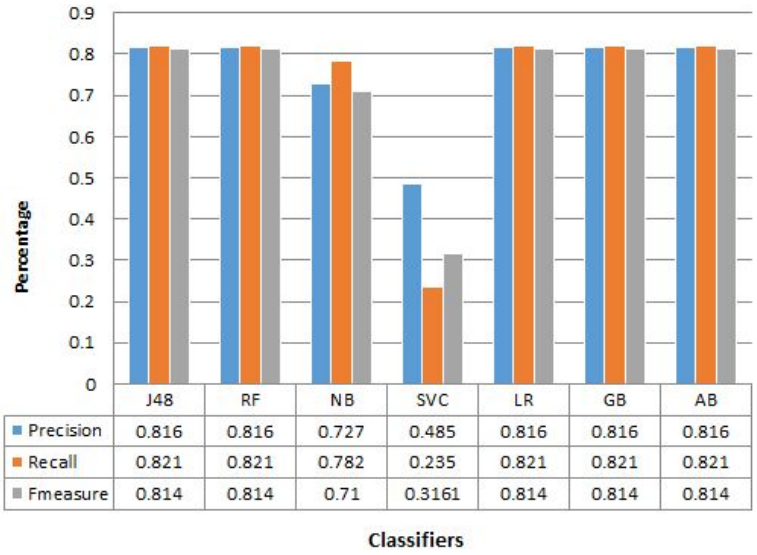


FIGURE 4.3: Evaluation results of ranked static data (InfoGain) Using a single machine learning algorithm

RF, LR, GB, and AB gain the equally highest precision value of 0.816 that shows their equivalent improvement of 9% and 33% against NB and SVC respectively. J48, RF, LR, GB, and AB also have equal recall and F-measures, which depicts

their better performance as compared to the NB and SVC on the selected static data. Comparatively the range of precision, recall, and F-measure is similar for

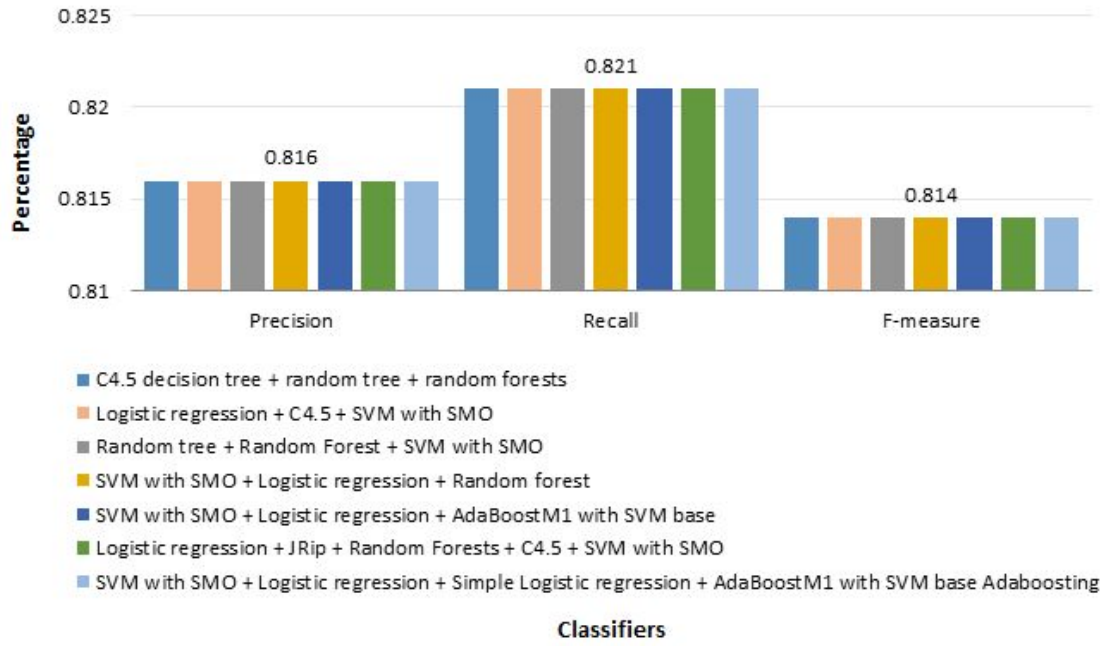


FIGURE 4.4: Evaluation results of ranked static data (InfoGain) using ensemble learning

this data in case of both single algorithm and ensemble algorithms. Figure 4.4 displays the results of precision, recall and F-measure on static data using ensemble learning. in order to find the best ensemble for both static and dynamic analysis, we performed experiments using odd (three or five) number of base algorithms in different combinations and applied majority voting for results. Different ensemble algorithms along with their performance results on the static ranked data set are shown in Figure 4.4. It can be seen from Figure that ensemble learning in all combination of algorithms has similar precision, recall, and F-measure which are 0.816, 0.821 and 0.814 respectively. Figure 4.5 and Figure 4.5 depicts the precision, recall, and F-Measure of single classifier and different ensemble models for dynamic data.

Figure 4.5 shows that AB algorithm generated the highest 0.967, 0.967 and 0.966 values of precision, recall, and F-measure respectively, among all other six algorithms. RF is the second-best performing algorithm with 0.959 precision, recall,

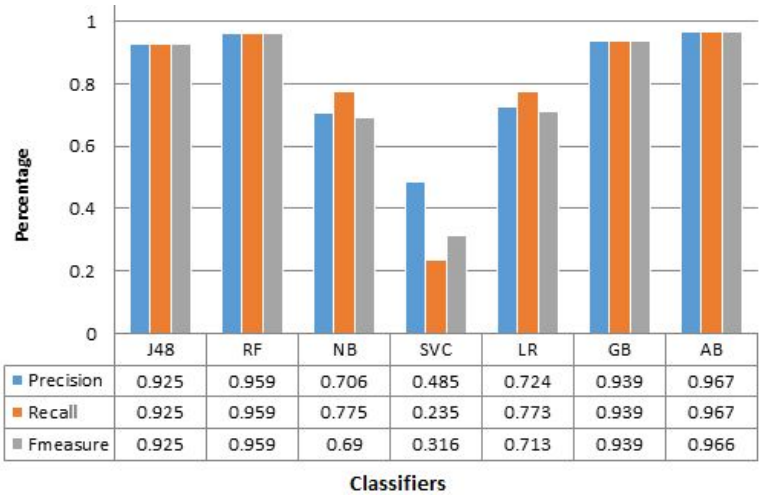


FIGURE 4.5: Evaluation results of ranked dynamic data (InfoGain) Using a single machine learning algorithm

and F-measure. However, SVC presented the lowest performance with 0.485 precision, 0.235 recall and 0.316 F-measure. Figure 4.6 describes the performance of ensembles on dynamic ranked data. For the selected dynamic data two ensembles, one with the base classifiers C4.5 decision tree + random tree + random forests and

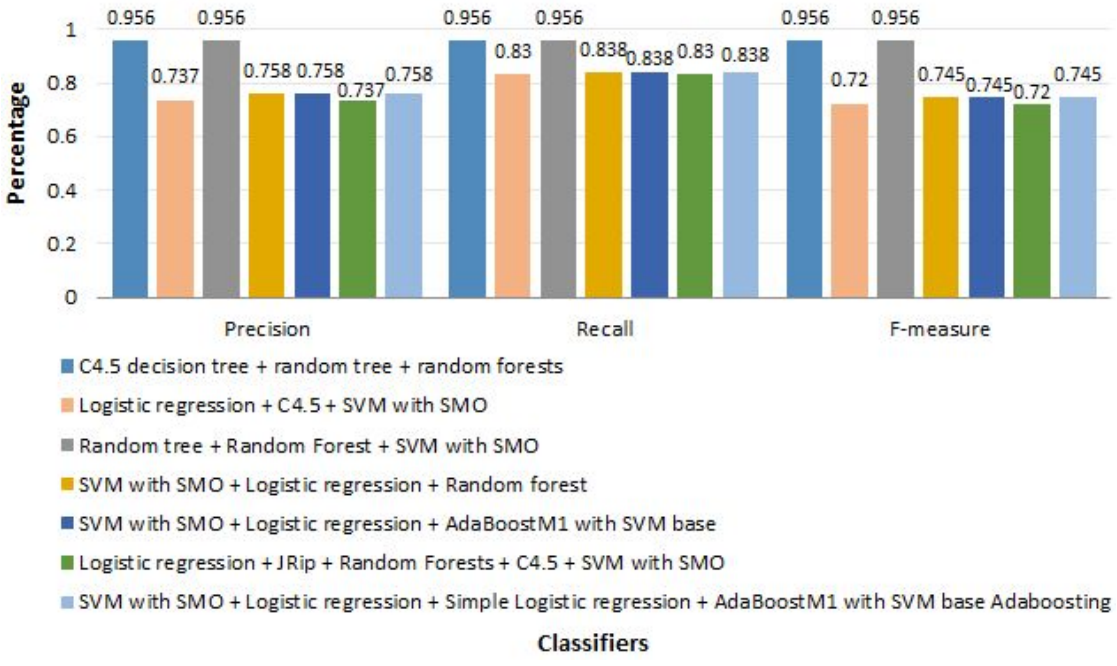
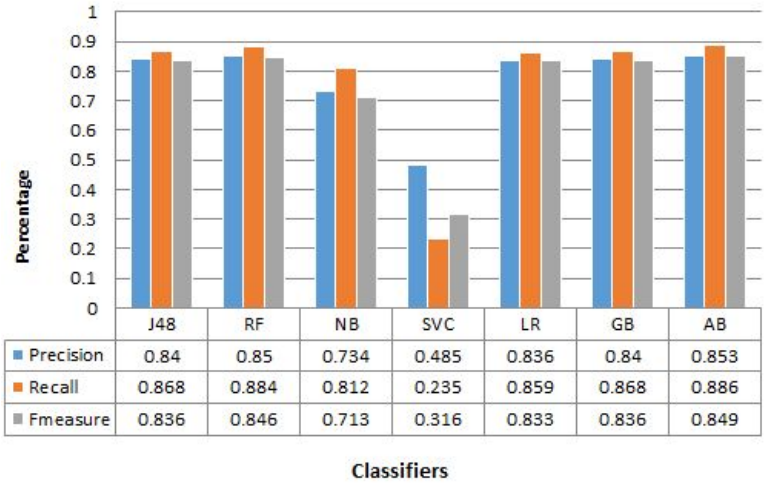


FIGURE 4.6: Evaluation results of ranked dynamic data (InfoGain) using ensemble learning

the second with Random tree + Random Forest + SVM with SMO base algorithms have similar highest precision, recall and F-measures value, which is 0.956. Ensemble with SVM with SMO + Logistic regression + Random forest and SVM with SMO + Logistic regression + AdaBoostM1 with SVM base have the second-highest values of measures. For the ranked dynamic data, the performance of single algorithms slightly differs from ensemble algorithms by showing improvement of 1%.



	J48	RF	NB	SVC	LR	GB	AB
Precision	0.84	0.85	0.734	0.485	0.836	0.84	0.853
Recall	0.868	0.884	0.812	0.235	0.859	0.868	0.886
Fmeasure	0.836	0.846	0.713	0.316	0.833	0.836	0.849

FIGURE 4.7: Evaluation results of ranked hybrid data (InfoGain) Using single machine learning algorithm

Figure 4.7 and Figure 4.8 represent the performance measures of hybrid distinct analysis (as described in Section 3.1) using both single machine learning algorithms and ensemble algorithms. with values of 0.853, 0.886, and 0.849 for precision, recall and F-measure respectively. However, the performance of ensemble having Random tree + Random Forest + SVM with SMO as member classifiers is significant among all others with 0.863 precision, 0.892 recall and 0.86 F-measure values.

From the figures, it can be seen that the for the ranked hybrid data ensemble algorithms perform well as compare to single algorithms. From single algorithms performance measures of AB are highest

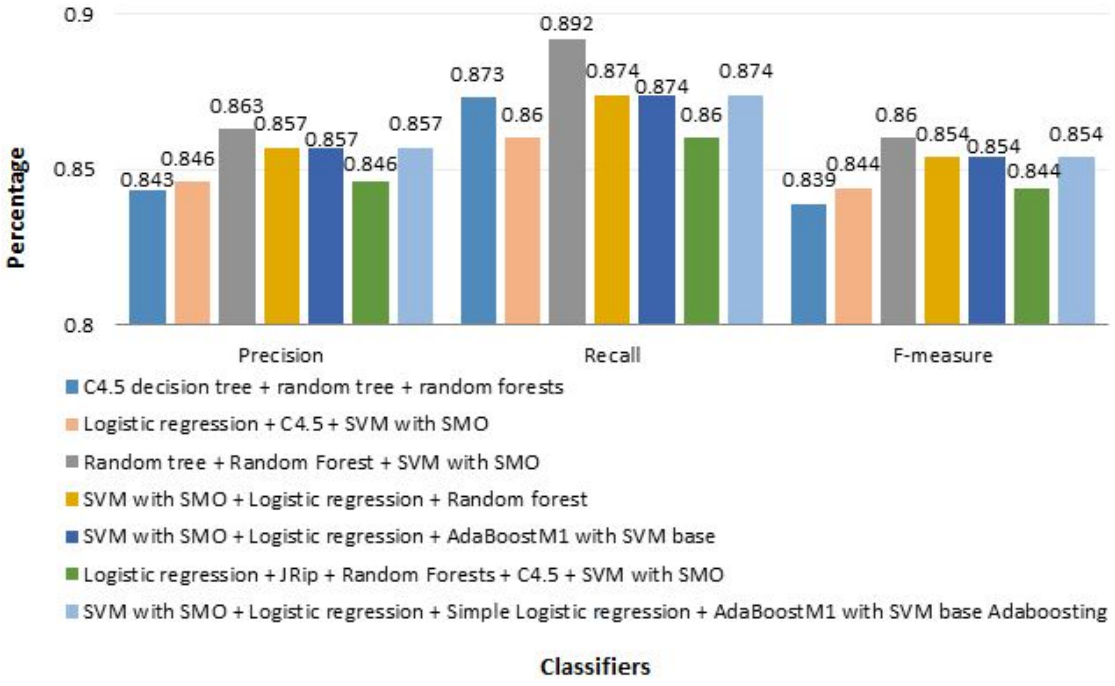


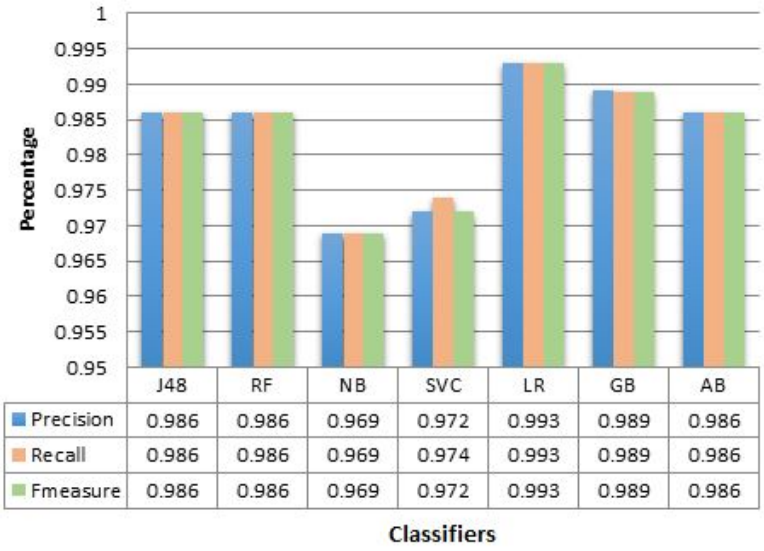
FIGURE 4.8: Evaluation results of ranked hybrid data (InfoGain) Using ensemble learning

This ensemble algorithm generated 11% improvement over F-measure, 6% over precision and 10% over F-measure of highly performing single machine learning algorithm AB.

4.6.2 Classification Results of Top Ranked Features from PCA

Figure 4.9 represents the performance results of single classifiers for the static data selected through PCA. Results show that LR is one step ahead with 0.993 precision, recall and F-measure in case of selected static data of PCA. AB, GB, RF and J48 exhibits the similar values of precision, recall and F-measure. The performance of other two classifiers SVC and NB is also significant wih. NB is the least performing classifier with 0.969 precision, recall and F-measure on PCA selected static data. Figure 4.10 represents the classification results of different ensembles on selected static data of PCA. The results described that the performance of the ensemble "Logistic regression + C4.5 + SVM with SMO" is highest among

all with 0.993 value of each performance measure. The performance results of all other ensembles are similar in terms of 0.989 precision, recall and F-measure, except the one ensemble "Random tree + Random Forest + SVM with SMO" whose performance is one percent reduced for the selected static data.



	J48	RF	NB	SVC	LR	GB	AB
Precision	0.986	0.986	0.969	0.972	0.993	0.989	0.986
Recall	0.986	0.986	0.969	0.974	0.993	0.989	0.986
Fmeasure	0.986	0.986	0.969	0.972	0.993	0.989	0.986

FIGURE 4.9: Evaluation results of ranked static data (PCA) Using single machine learning algorithm

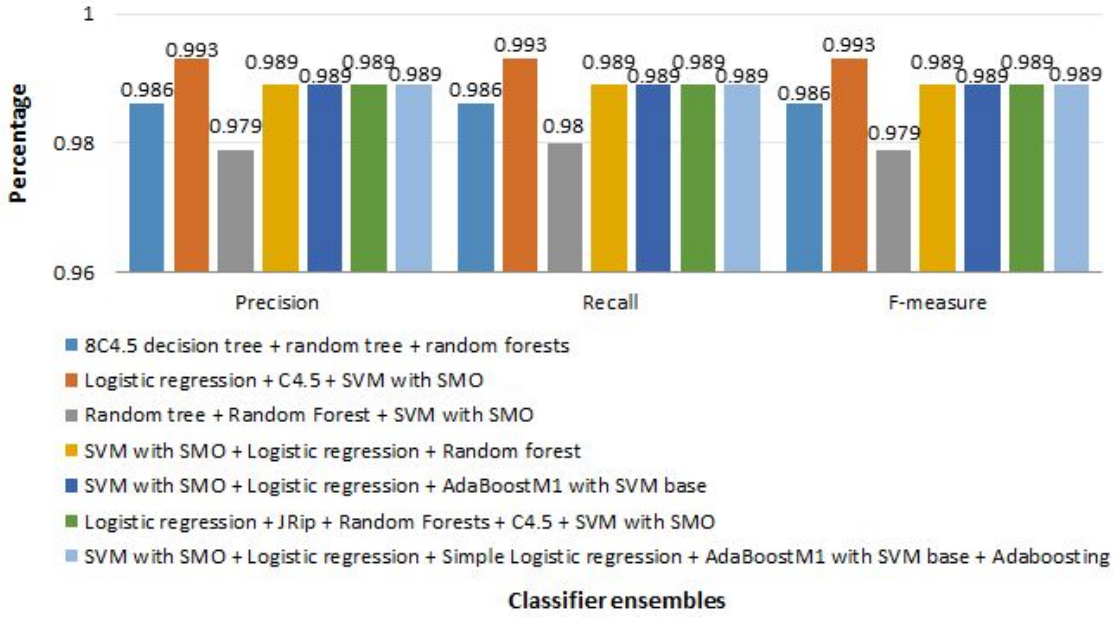
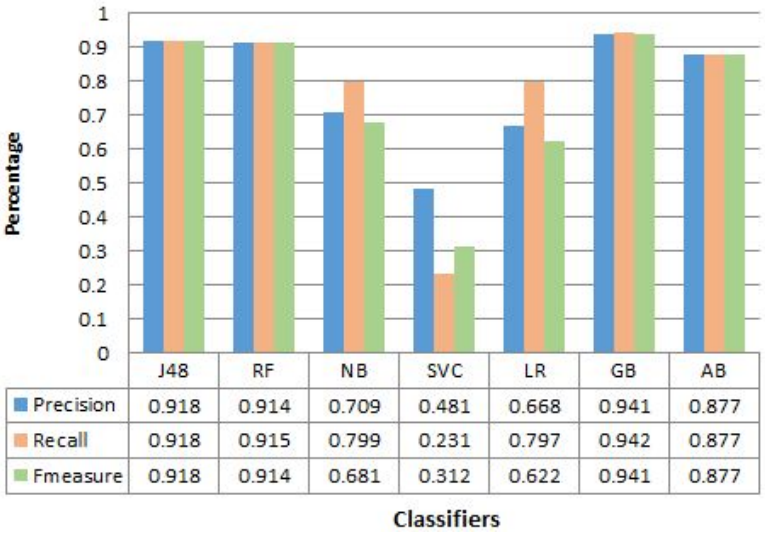


FIGURE 4.10: Evaluation results of ranked static data (PCA) Using ensemble learning

The performance results of every single classifier and multiple different ensembles for the ranked dynamic data gained through PCA are represented in Figure 4.11 and Figure 4.12 From Figure 4.11 it is quite evident that the GB outperformed all other classifiers with 0.942 recall and 0.94 precision and F-measure. The performance of RF and J48 is equally well. However, SVC performed below average for the selected dynamic data from PCA.



	J48	RF	NB	SVC	LR	GB	AB
Precision	0.918	0.914	0.709	0.481	0.668	0.941	0.877
Recall	0.918	0.915	0.799	0.231	0.797	0.942	0.877
Fmeasure	0.918	0.914	0.681	0.312	0.622	0.941	0.877

FIGURE 4.11: Evaluation results of ranked dynamic data (PCA) Using single machine learning algorithm

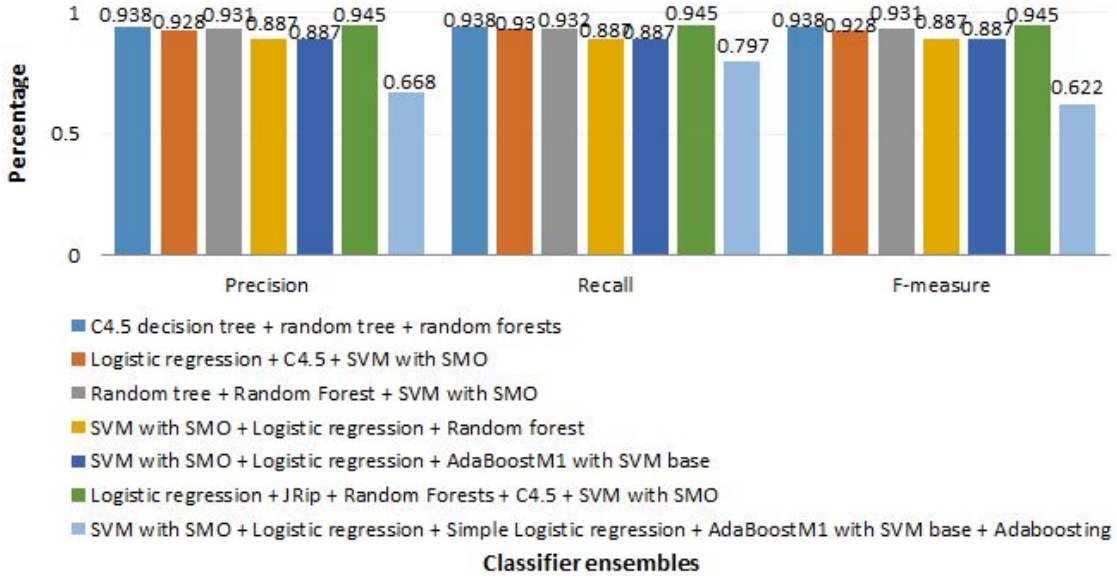
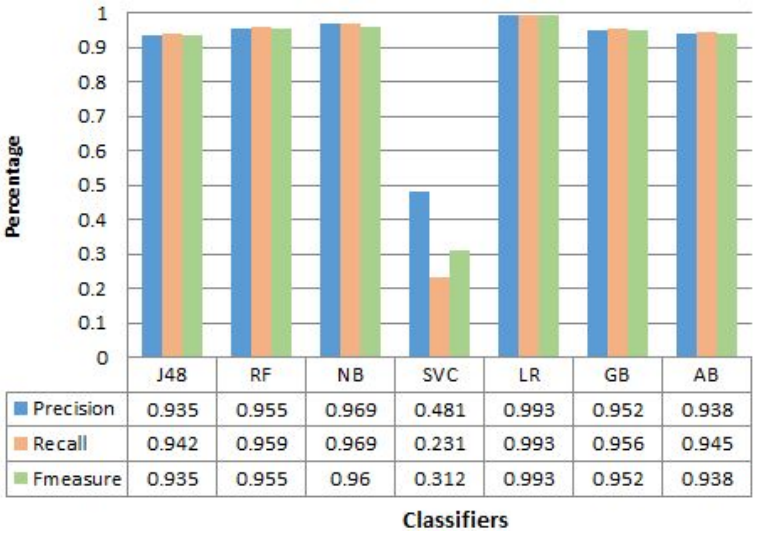


FIGURE 4.12: Evaluation results of ranked dynamic data (PCA) Using ensemble learning

In Figure 4.12 it can be seen that the performance of ensemble with base classifiers" Logistic regression + JRip + Random Forests + C4.5 + SVM with SMO" is effective among all with 0.945 precision, recall and F-measure. The performance of ensemble "SVM with SMO + Logistic regression + Simple Logistic regression + AdaBoostM1 with SVM base + Adaboosting" is lowest with 0.668 precision, 0.797 recall and 0.622 F-measure.

Figure 4.13 and Figure 4.14 describes the performance results of single classifiers and ensembles on selected hybrid data of PCA. Figure 4.13 shows that the performance of LR is highest among all with 0.993 precision, recall, and F-measure. The performance of SVC remained lowest with 0.481 precision, 0.231 recall and 0.312 F-measure values which are not up to the mark. All other single classifiers apart from SVC performed significantly well for hybrid selected data.



	J48	RF	NB	SVC	LR	GB	AB
Precision	0.935	0.955	0.969	0.481	0.993	0.952	0.938
Recall	0.942	0.959	0.969	0.231	0.993	0.956	0.945
Fmeasure	0.935	0.955	0.96	0.312	0.993	0.952	0.938

FIGURE 4.13: Evaluation results of ranked hybrid data (PCA) Using single machine learning algorithm

Figure 4.14 shows that the ensemble with base classifiers "SVM with SMO + Logistic regression + Simple Logistic regression + AdaBoostM1 with SVM base + Adaboosting" outperformed all other ensembles on selected hybrid data through PCA by obtaining 0.989 precision, recall, and F-measures. The Performance of other ensembles is also significant. From Figure 4.13 and Figure 4.14 we can see each single, as well as ensemble classifier, performed much better on selected data of PCA as compared to the selected data of info gain. Removal of redundant

features increases accuracy of classification as well as reduces the computational cost.

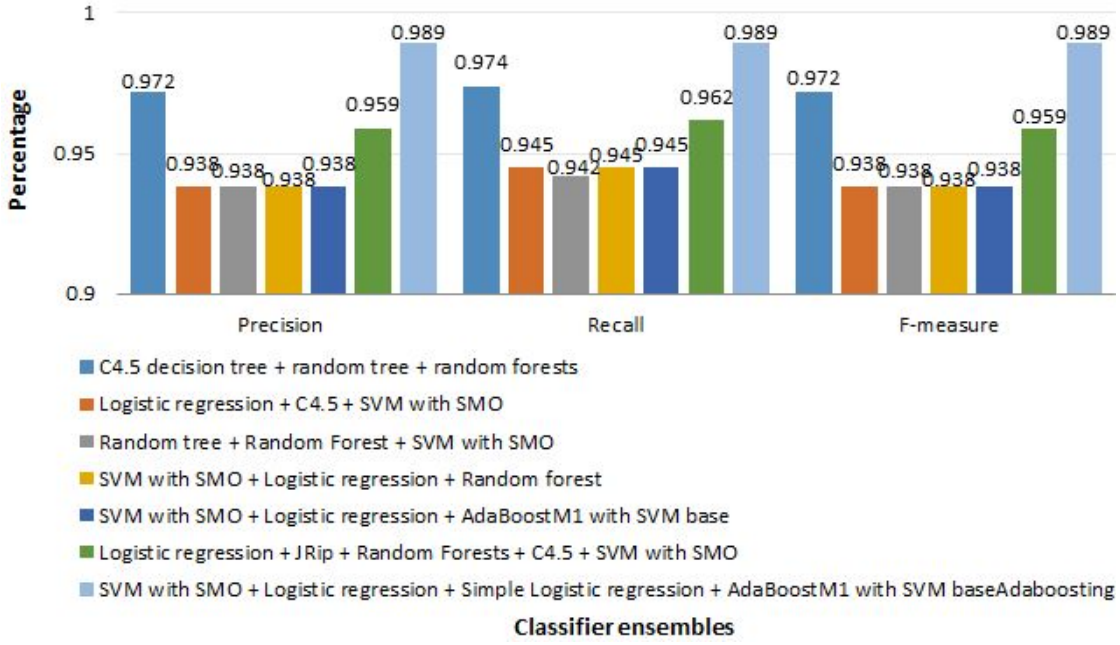
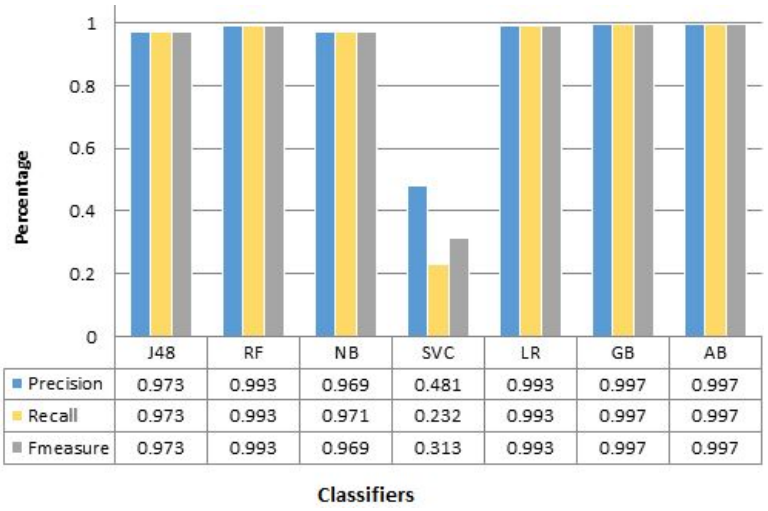


FIGURE 4.14: Evaluation results of ranked dynamic data (PCA) Using ensemble learning

4.7 Classification Results without Feature Selection

We have seen the performance of single and ensemble algorithms on ranked data set obtained through *InfoGain* method. The reduction in the dimension of feature vectors may have affected the performance of a few classifiers. Therefore we evaluated the performance of all single and ensemble classifiers on whole features data set without performing feature selection. Figure 4.15 and Figure 4.16 represent the performance measure of single and ensemble algorithms on static data set.

Figure 4.15 shows that AB and GB algorithms have best 0.997 Precision, recalls and F-measures. RF and LR both have the second best measures that are 0.993. J48 have 0.973, NB has 0.969, 0.971 and 0.969 precision, recall and F-measure respectively.



	J48	RF	NB	SVC	LR	GB	AB
Precision	0.973	0.993	0.969	0.481	0.993	0.997	0.997
Recall	0.973	0.993	0.971	0.232	0.993	0.997	0.997
Fmeasure	0.973	0.993	0.969	0.313	0.993	0.997	0.997

FIGURE 4.15: Evaluation results of static data using single machine learning algorithm

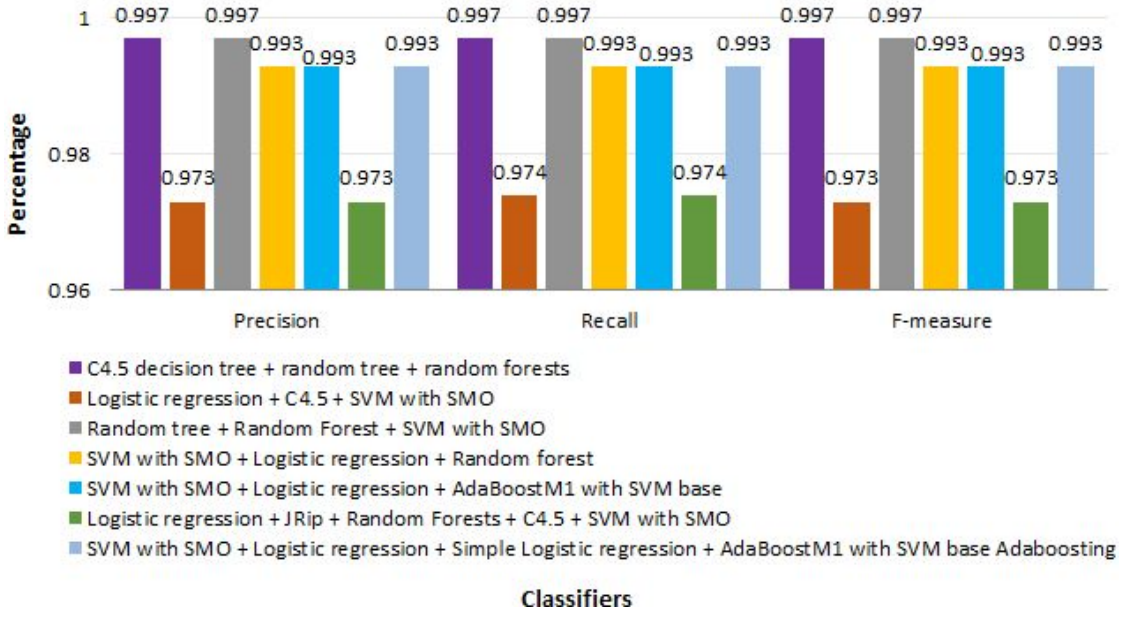
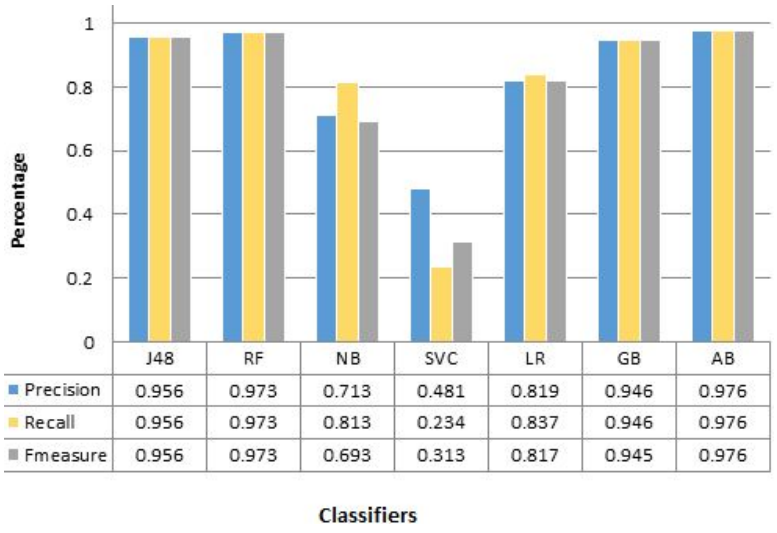


FIGURE 4.16: Evaluation results of static data using ensemble learning

The performance of SVC remains the lowest among all. Results in Figure 4.16 shows similar behavior as seen in Figure 4.15 the highest measures are 0.997 generated by ensembles of C4.5 decision tree + random tree + random forests base algorithms. And Random tree + Random Forest + SVM with SMO base algorithms. The ensemble with lowest performance measures among all other ensembles also has 0.973 precision, recall, and F-measures which is much significant. The precision, recall, and F-measure for dynamic data using both single and ensemble



	J48	RF	NB	SVC	LR	GB	AB
Precision	0.956	0.973	0.713	0.481	0.819	0.946	0.976
Recall	0.956	0.973	0.813	0.234	0.837	0.946	0.976
Fmeasure	0.956	0.973	0.693	0.313	0.817	0.945	0.976

FIGURE 4.17: Evaluation results for dynamic data using single classifier

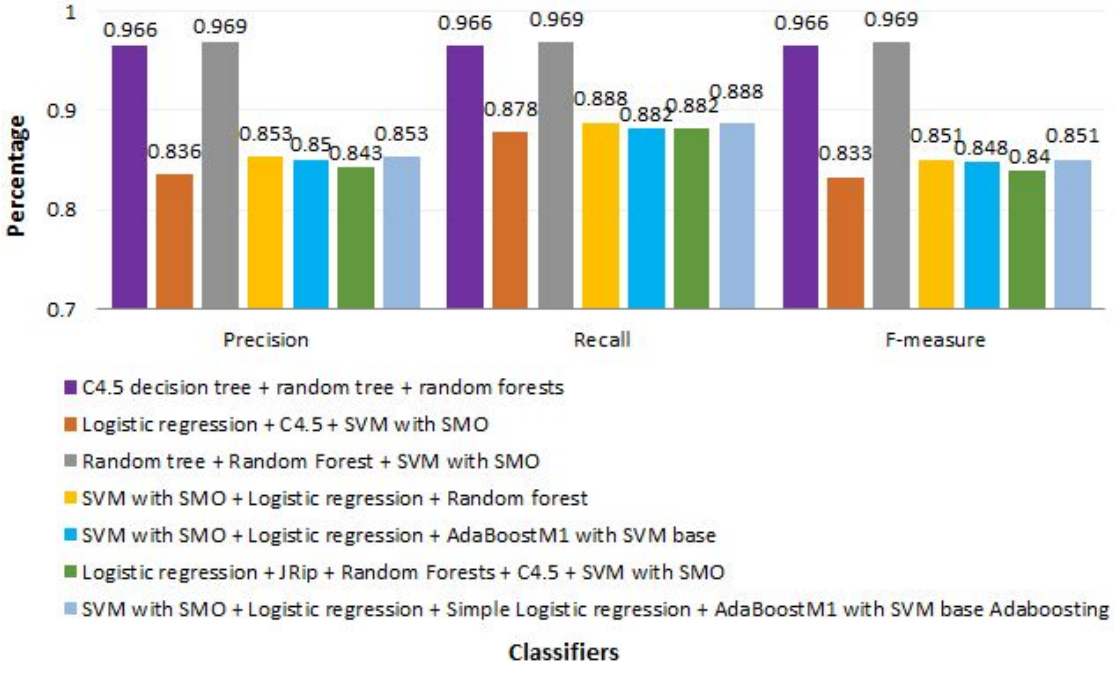


FIGURE 4.18: Evaluation results for dynamic data using ensemble learning

learning are shown in Figures 4.17 and Figure 4.18. The single classifier AB performs equally well for dynamic data too, with 0.976 precision, recall, and F-measure. RF generated second highest 0.973, J48 third with 0.956 and GB fourth with 0.946 precision, recall, and F-measure. In Figure 4.18 Random tree + Random Forest + SVM with SMO ensemble achieved high performance with 0.969 precision, recall and F-measure, followed by C4.5 decision tree + random tree + random forests with 0.966 performance measures.

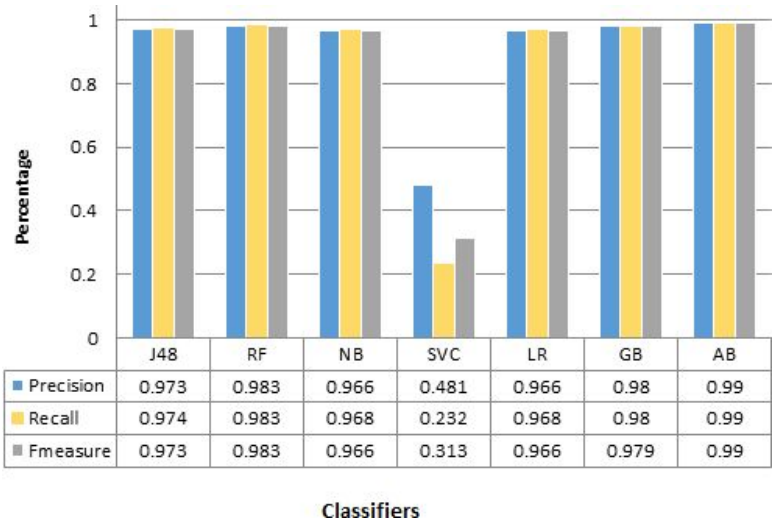


FIGURE 4.19: Evaluation results for Hybrid distinct data using single classifier

Figure 4.19 shows the performance of all single classifiers on hybrid distinct analysis, which consists of both static and dynamic analysis, a detailed explanation is given in Section 3.1 (using single classifier instead of the ensemble). The results of single classifiers for hybrid analysis are given in Figure 4.19 which indicates that the performance of AB is significantly high among all other single classifiers. All single algorithms performed well for the provided hybrid distinct data. Except for SVC whose performance remains same, lowest across all experiments.

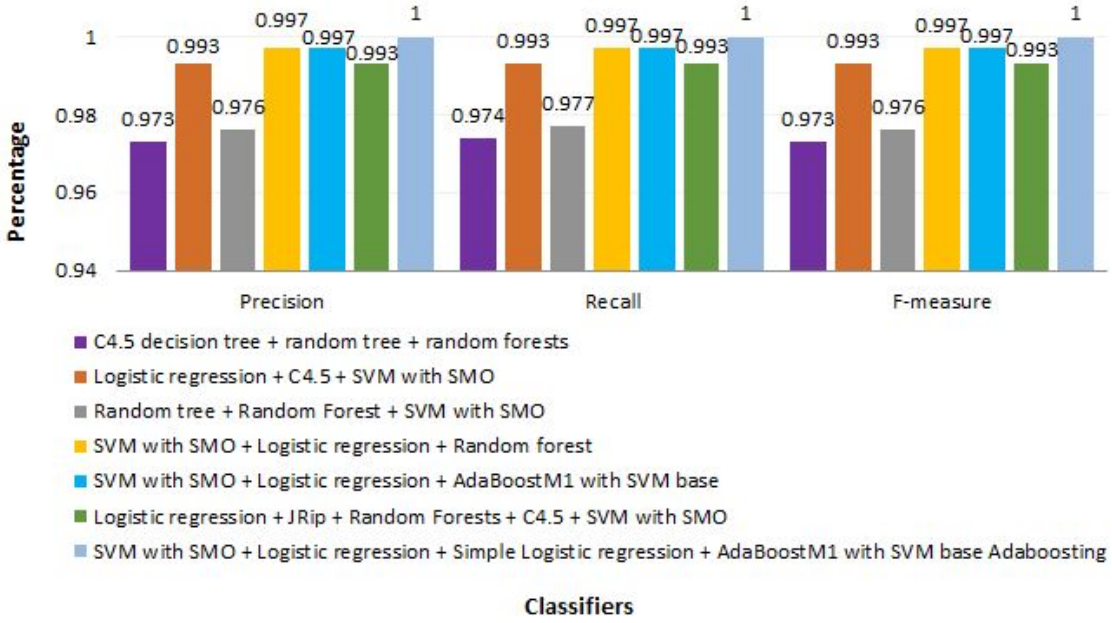


FIGURE 4.20: Evaluation results for Hybrid distinct data using ensemble Learning

Figure 4.20 shows the performance of our proposed approach that is hybrid distinct ensemble analysis explained in Section 3.1 it comprises of two distinct machine learning ensemble models for static and dynamic features data sets. The results of all different ensembles are shown in Figure 4.20 which depict that ensemble with SVM with SMO + Logistic regression + Simple Logistic regression + AdaBoostM1 with SVM base+Adaboosting member classifiers outperformed all other ensembles as well as single machine learning algorithms by obtaining 100% precision, recall, and F-measures. The precision of ensemble with SVM with SMO + Logistic regression + AdaBoostM1 with SVM base member's algorithms and ensemble with SVM with SMO + Logistic regression + Random forest base algorithms are the second high among all other ensembles having 0.997 precision, recall and F-measure.

From Figure 4.19 and Figure 4.20 it is quite evident that the highest performance is achieved when the classification is performed using ensemble with base classifiers:

1. SVM with SMO + Logistic regression + Simple Logistic regression + AdaBoostM1 with SVM base + Adaboosting
2. SVM with SMO + Logistic regression + AdaBoostM1 with SVM base.
3. SVM with SMO + Logistic regression + Random forest

The first ensemble achieved the highest performance in classification over the hybrid distinct data set, which is improved only 0.3% over the other two ensembles. This indicates that these ensembles produce similar results in Android ransomware classification and detection. Logistic regression + C4.5 + SVM with SMO also displayed good performance with up to 0.993 precision, which specifies that it is the second-best ensemble algorithm that identifies ransomware correctly from malware.

4.8 Test Results of Hybrid Distinct Ensemble Analysis Approach Against Fabricated Inputs

In order to validate the resilience of the proposed hybrid distinct ensemble model against adversarial evasion attacks, we tested the model using different fabricated inputs. These fabricated inputs are produced by making slight changes in feature vectors of known Android ransomware. We evaluated the performance of the proposed model for mitigating evasion attacks by making 1 bit, 10-bits, and 20-bits changes in the input feature vector of known ransomware.

4.8.1 Results Against 1-bit Fabricated Input

the adversarial evasion attacks because the one-bit change in input feature vector results in evasion from the underlying classification model. In order to evaluate the proposed model against such kind of fabrication in the input feature vector, we randomly selected 100 feature vectors of known ransomware and made a one-bit change in permissions of each vector.

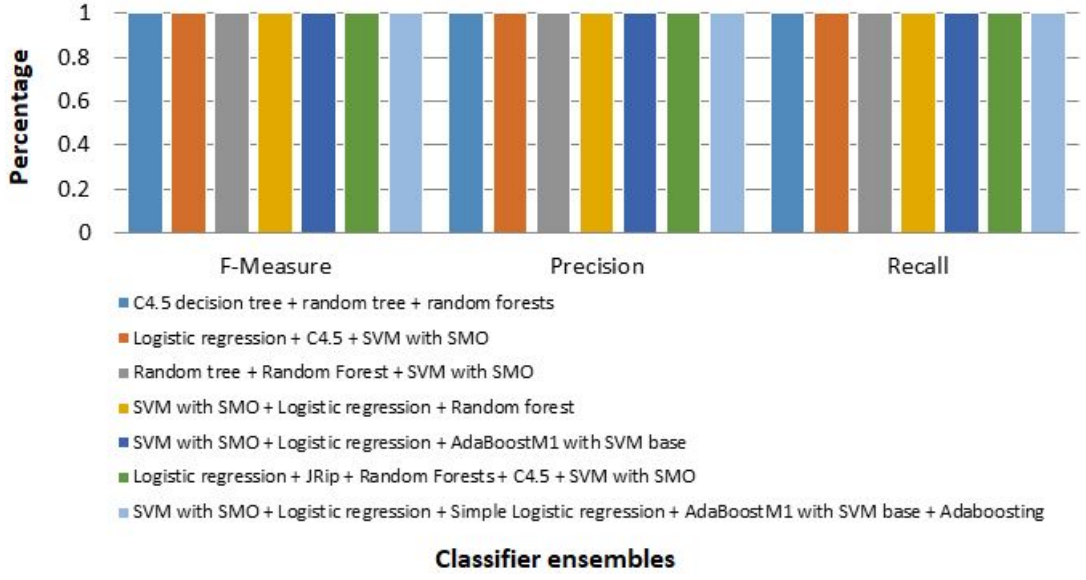


FIGURE 4.21: Precision, Recall & F-Measure of Hybrid distinct ensemble analysis approach Using one-bit fabricated data.

Hence the permissions are the most susceptible feature for encryption or renaming obfuscation. We tested the hybrid distinct ensemble analyzer (explained in detail in Section 3.1) on these fabricated feature vectors, by employing different ensemble algorithms. Test results of different ensembles in the form of precision, recall, and F-Measure are represented in Figure 4.21.

From the figure, it can be clearly seen that each ensemble generated 100% measures of precision, recall, and F-measure for one-bit fabricated data. Figure 4.22 indicates the accuracy of different ensembles on hybrid distinct ensemble analysis approach. From the figure, it can be depicted the accuracy of each model is 100% against one-bit fabricated data.

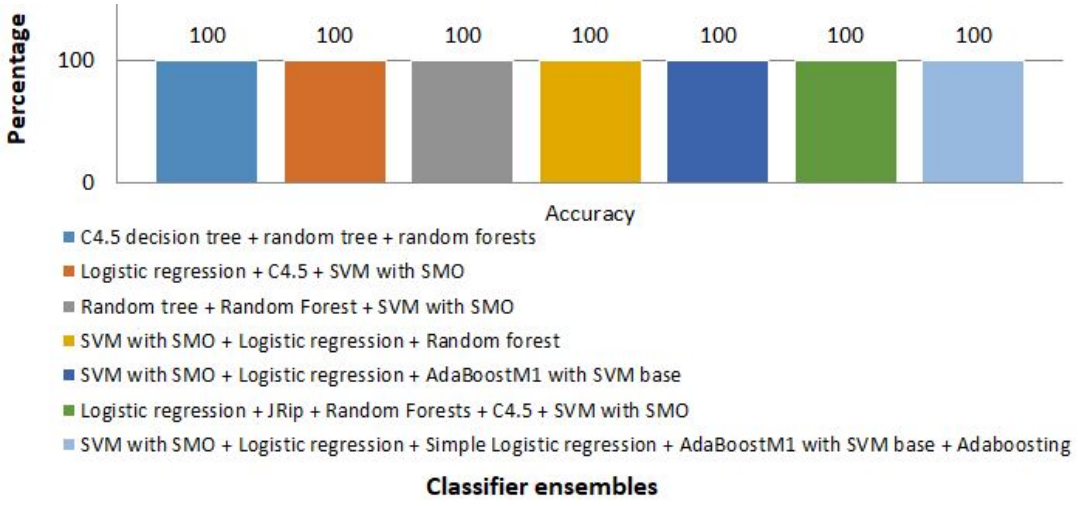


FIGURE 4.22: Accuracy of Hybrid distinct ensemble analysis approach using one-bit fabricated data.

4.8.2 Results Against 10-bit Fabricated Input

For more precise resilience validation test of the model, we fabricated the data of randomly selected 100 feature vectors of Android ransomware, in such a way that in each feature vector 10-bits related to the permissions features are modified. Again the hybrid distinct ensemble model is tested on these modified feature vectors. Results of each ensemble are displayed in Figure 4.23. It is evident from the figure that ensemble with member classifier "SVM with SMO + Logistic regression + Simple Logistic regression + AdaBoostM1 with SVM base + Adaboosting"

achieved 97%, 98%, and 100%precision, recall and F-measure values respectively which are lowest among all. All other ensembles achieved 100% precision, recall, and F-measures. Figure 4.24 represents the accuracy of ensembles for 10-bit fabricated data.

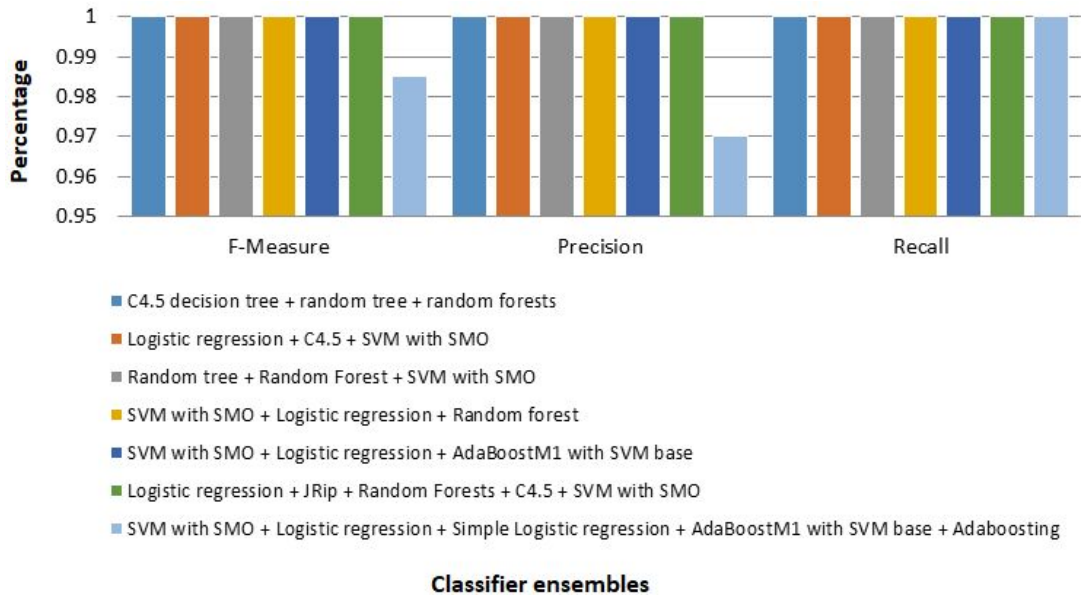


FIGURE 4.23: Precision, Recall & F-Measure of Hybrid distinct ensemble analyzer Using 10-bit fabricated data

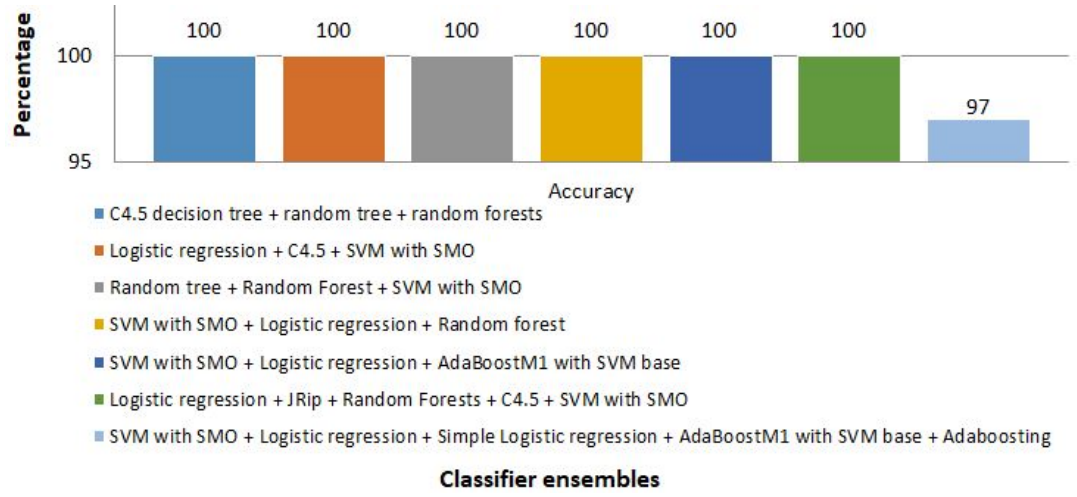


FIGURE 4.24: Precision, Recall & F-Measure of Hybrid distinct ensemble analyzer Using 10-bit fabricated data

Figure 4.23 and Figure 4.24 displays the same picture regarding performance of different ensembles.

4.8.3 Results Against 20-bit Fabricated Input

After evaluating the model on 1-bit and 10-bit fabricated inputs and achieving 100% accuracy on all ensembles except one. we tried to test the model against 20-bit fabricated inputs data. The fabricated input feature vectors are obtained through modifying twenty random bits belonging to the permissions, in each feature vector of ransomware. This fabrication in each feature vector almost changes the whole aspect related to the permissions. Since from the extracted data of permissions, it was assessed that barely any ransomware application can ask more than twenty unique permissions. On these fabricated feature vectors We tested our proposed hybrid distinct ensemble analysis model. Figure 4.25 and Figure 4.26 depicts the obtained test results of different ensembles.

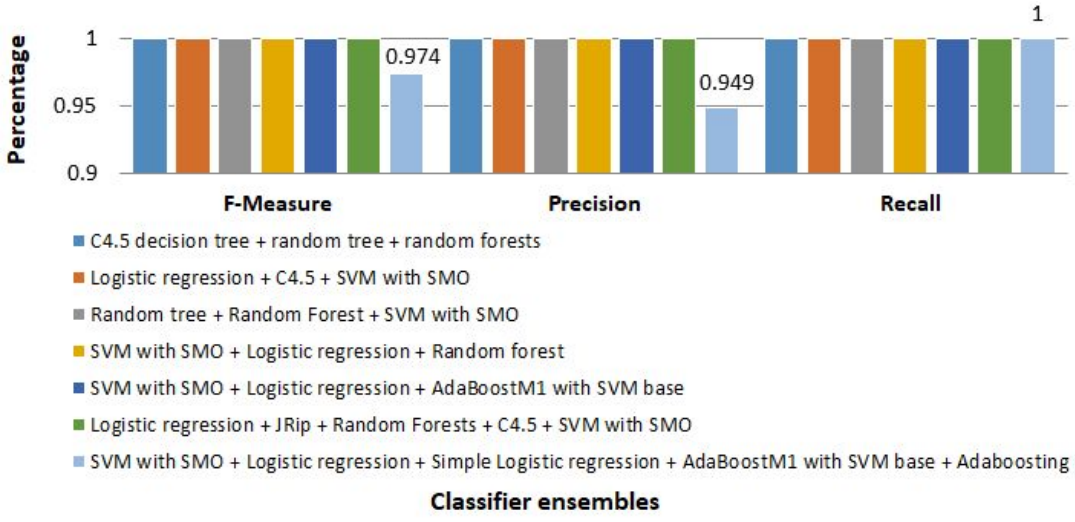


FIGURE 4.25: Precision, Recall & F-Measure Using 10-bit fabricated data

From Figure 4.25 it is quite evident that this substantial fabrication in permissions does not affect the performance of the hybrid distinct ensemble model. Except for the one ensemble (SVM with SMO + Logistic Regression + Simple Logistic regression + AdaBoostM1 with SVM base +Adaboosting), whose precision and F-Measure are 0.949 and 0.974 respectively, which are lowest among all. Figure 4.26 represents the accuracy of the model using different ensembles on 20-bits fabricated input feature vectors.

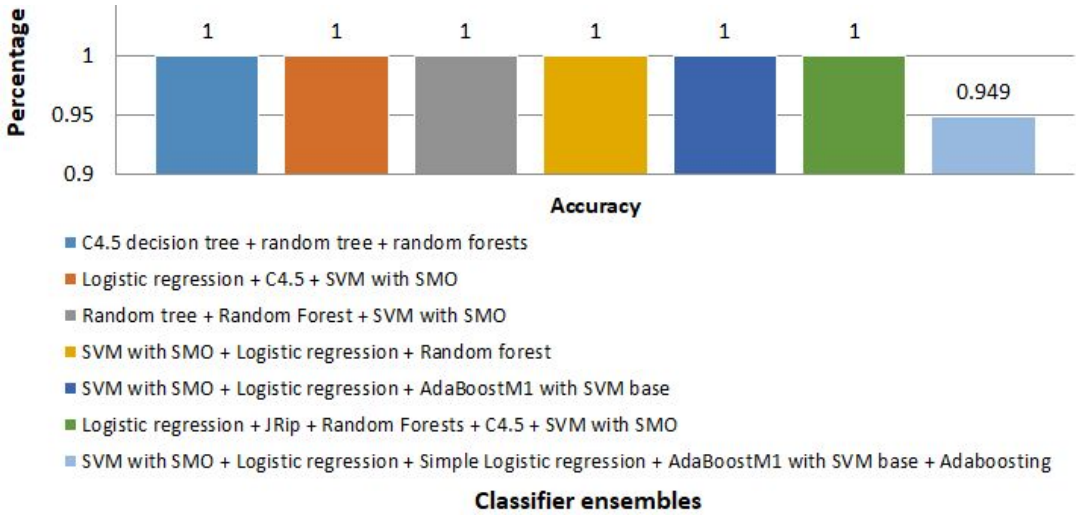


FIGURE 4.26: Accuracy of hybrid distinct ensemble against 20-bit fabricated data

The figure shows the same behavior as seen in Figure 4.25. All ensembles achieved 100% accuracy except the one (SVM with SMO + Logistic Regression + Simple Logistic regression + AdaBoostM1 with SVM base +Adaboosting). The accuracy of this ensemble is 94.9%.

The results obtained from these three tests validate this fact that hybrid distinct ensemble analysis approach is resilient to adversarial evasion attacks. It is capable of detecting Android ransomware as well as their their fabricated samples with high accuracy

In spite of the fact that test results of both single and ensemble classifiers have shown their encouraging performance, the performance of ensemble learning was somewhat better than single classifiers. This ensemble learning can be used in anti-ransomware systems. It would be able to detect new and unknown Android ransomware samples since it works on learned malicious patterns of ransomware rather than relying on signatures. The reason behind employing hybrid distinct ensemble learning is to decline the adversarial evasion attacks. The obtained promising test results of the proposed ensemble-based model against fabricated inputs validated this fact that it is capable of mitigating evasion attacks.

Ensemble classifiers are much slower as they required more computational time to apply multiple machine learning algorithms (three or five) and their results processing instead of only one classifier. However, taking into consideration, the rise of Android ransomware attacks, their irreversible effects, and severity of nature, it is essential to design such kind of Android ransomware detection technique that can effectively detect the Android ransomware and mitigate evasion attacks. To achieve this objective we have to trade off between overhead in terms of computational cost due to the ensemble learning and the achieved security from ransomware attacks.

Chapter 5

Conclusion and Future Work

In this chapter, we summarize the study by pointing out its achievements. It reviews the important findings as well as highlight the potential areas for future improvement.

5.1 Conclusion

With Android overwhelmingly the most popular operating system for mobile devices, ransomware specially made for Android devices (phone, tablets, etc.,) are on the rise. The computer security firm Symantec conservatively estimates that Android ransomware extorts hundreds of millions of dollars from victims annually. Due to a large amount of money to be made, new versions of Android ransomware appears frequently. This allows evasion from anti-viruses and intrusion detection methods. Android malware share some common instructions to the ransomware, with the exception that ransomware have some other more threatening issues. Ransomware locks the device or completely encrypt the files stored on device and demand a ransom for their release. Paying the ransom is no guarantee that the device will be released, in many cases, it is not. Therefore, the classification of Android ransomware from other malware (non-ransomware) becomes essential. This

research work presents an ensemble machine learning-based approach that considers a variety of features to classify the Android ransomware from non-ransomware and mitigate the adversarial evasion attacks. We used a combination of multiple static and dynamic features such as Permissions, Network-based features (IP addresses, URL and email addresses), Text, System calls statistics, CPU and memory usage. In this study, we evaluated the performance of different machine learning ensembles in Android ransomware detection and classification, on both static and dynamic feature sets. Moreover, we found the features that play a vital role in the classification of Android ransomware from other malware. It has been seen that permissions and system call logs are the two most relevant features for the detection and classification of Android ransomware from non-ransomware.

We presented a hybrid distinct ensemble analysis approach for the detection and classification of Android ransomware. As the name suggests it involves the static and dynamic analysis and two separate ensemble machine learning analyzers for both static and dynamic sets of features. Our detection mechanism starts with the extraction of both static and dynamic features from APK files. After features extraction, these features are provided to the static and dynamic machine learning-based ensembles which predict the class of malware application as ransomware or non-ransomware. Both ensemble analyzers consist of a group of an odd number of member classifiers such as C4.5, Random Forest, JRip, Logistic regression, etc., these classifiers make a class prediction for the input feature vector and final prediction is made by the meta classifier on the basis of majority voting. Malware application is classified as ransomware if any of the static or dynamic ensemble analyzers predict it to be ransomware. Moreover, we examine the capability of the proposed model for mitigating adversarial evasion attacks by testing it on fabricated inputs. Our experimented data set consists of 550 Android malware applications consisting of 275 ransomware and 275 non-ransomware applications.

The obtained results support our decisions regarding the training of ensemble analyzers for both ransomware detection and mitigation of evasion attacks, by accomplishing good results. Our proposed distinct ensemble analysis mechanism shows promising results by achieving 100% values of precision, recall, and F-measure in

case of Android ransomware detection. Moreover, the proposed hybrid distinct ensemble analysis approach validates itself as the resilient model against adversarial evasion attacks by achieving 100% accuracy against fabricated inputs.

5.2 Future Work

Extremely promising classification results from the utilization of ensemble machine learning analysis encouraged more advancement in this work. We can also prove our framework on huge size dataset. In the future, we can train such kind of ensemble machine learning analyzer that can detect the ransomware application and classify them into families. Moreover, we can employ different other combinations of machine learning algorithms in ensemble learning. The other feature selection algorithms can also be utilized to validate our results. Beside that, we can explore more dynamic and static properties of applications.

Bibliography

[1] Nicoló Andronio, Stefano Zanero, and Federico Maggi. Heldroid: Dissecting and detecting mobile ransomware. In *International Symposium on Recent Advances in Intrusion Detection*, volume 9404, pages 382–404. Springer, 2015.

[2] Daniele Sgandurra, Luis Muñoz-González, Rabih Mohsen, and Emil C. Lupu. Automated dynamic analysis of ransomware: Benefits, limitations and use for detection. *CoRR*, abs/1609.03020, 2016. URL http://arxiv.org/abs/1609.03020.

[3] Bander Ali Saleh Al-rimy, Mohd Aizaini Maarof, and Syed Zainudeen Mohd Shaid. Ransomware threat success factors, taxonomy, and countermeasures: A survey and research directions. *Computers & Security*, 74:144–166, 2018.

[4] statista. Share of Android OS of global smartphone shipments from 1st quarter 2011 to 2nd quarter 2018*. https://www.statista.com/statistics/236027/global-smartphone-os-market-share-of-Android/, 2019. [Online; accessed March-2019].

[5] Sanggeun Song, Bongjoon Kim, and Sangjun Lee. The effective ransomware prevention technique using process monitoring on android platform. *Mobile Information Systems*, 2016, 2016.

[6] McAfee Labs. McAfee Labs Threats Report September 2018. [25]https://www.mcafee.com/enterprise/en-us/assets/reports/rp-quarterly-threats-sep-2018.pdf, 2019. [Online; accessed March-2019].

[7] Tianda Yang, Yu Yang, Kai Qian, Dan Chia-Tien Lo, Ying Qian, and Lixin Tao. Automated detection and analysis for android ransomware. In *2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems*, pages 1338–1343. IEEE, 2015.

[8] Madiha Ameer, Sumera Murtaza, and Muhammad Aleem. A study of android-based ransomware: Discovery, methods, and impacts. *Journal of Information Assurance & Security*, 13(3):109–117, 2018.

[9] Amirhossein Gharib and Ali Ghorbani. Dna-droid: A real-time android ransomware detection framework. In *International Conference on Network and System Security*, volume 10394, pages 184–198. Springer, 2017.

[10] Abdulrahman Alzahrani, Ali Alshehri, Hani Alshahrani, Raed Alharthi, Huirong Fu, Anyi Liu, and Ye Zhu. Randroid: Structural similarity approach for detecting ransomware applications in android platform. In *2018 IEEE International Conference on Electro/Information Technology (EIT)*, pages 0892–0897. IEEE, 2018.

[11] Jing Chen, Chiheng Wang, Ziming Zhao, Kai Chen, Ruiying Du, and Gail-Joon Ahn. Uncovering the face of android ransomware: Characterization and real-time detection. *IEEE Transactions on Information Forensics and Security*, 13(5):1286–1300, 2017.

[12] Francesco Mercaldo, Vittoria Nardone, Antonella Santone, and Corrado Aaron Visaggio. Ransomware steals your phone. formal methods rescue it. In *International Conference on Formal Techniques for Distributed Objects, Components, and Systems*, volume 9688, pages 212–221. Springer, 2016.

[13] Alberto Ferrante, Miroslaw Malek, Fabio Martinelli, Francesco Mercaldo, and Jelena Milosevic. Extinguishing ransomware-a hybrid approach to android ransomware detection. In *International Symposium on Foundations and Practice of Security*, volume 10723, pages 242–258. Springer, 2017.

[14] Pavol Zavarsky, Dale Lindskog, et al. Experimental analysis of ransomware on windows and android platforms: Evolution and characterization. *Procedia Computer Science*, 94:465–472, 2016.

[15] kaspersky. kaspersky daily. [60]https://www.kaspersky.com/blog/ransomware-faq/13387/, 2019. [Online; accessed july-2019].

[16] Sana Aurangzeb, Muhammad Aleem, Muhammad Azhar Iqbal, and Muhammad Arshad Islam. Ransomware: A survey and trends. *Journal of Information Assurance & Security*, 6(2):48–58, 2017.

[17] Daniel Nieuwenhuizen. A behavioural-based approach to ransomware detection. *Whitepaper. MWR Labs Whitepaper*, 2017.

[18] Sergii Banin and Geir Olav Dyrkolbotn. Multinomial malware classification via low-level features. *Digital Investigation*, 26:S107–S117, 2018.

[19] Asaf Shabtai, Uri Kanonov, Yuval Elovici, Chanan Glezer, and Yael Weiss. "andromaly": a behavioral malware detection framework for android devices. *Journal of Intelligent Information Systems*, 38(1):161–190, 2012.

[20] Pavel Laskov and Richard Lippmann. Machine learning in adversarial environments, 2010.

[21] Battista Biggio and Fabio Roli. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition*, 84:317–331, 2018.

[22] K.S. Jones, P. Willett, and Jones. *Readings in Information Retrieval*. Morgan Kaufmann series in multimedia information and systems. Morgan Kaufman, 1997. ISBN 9781558604544. URL https://books.google.com.pk/books?id=Nt5nDTYQ0okC.

[23] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[24] Yongheng Zhao and Yanxia Zhang. Comparison of decision tree methods for finding active objects. *Advances in Space Research*, 41(12):1955–1959, 2008.

[25] Asa Ben-Hur, David Horn, Hava T Siegelmann, and Vladimir Vapnik. Support vector clustering. *Journal of machine learning research*, 2(Dec):125–137, 2001.

[26] Stephan Dreiseitl and Lucila Ohno-Machado. Logistic regression and artificial neural network classification models: a methodology review. *Journal of biomedical informatics*, 35(5-6):352–359, 2002.

[27] Yoav Freund, Robert E Schapire, et al. Experiments with a new boosting algorithm. -, 96:148–156, 1996.

[28] Jason Brownlee. A gentle introduction to the gradient boosting algorithm for machine learning. *Machine Learning Mastery,[En línea][Citado el: 3 de Junio de 2018.] URL: https://machinelearningmastery. com/gentleintroduction-gradient-boosting-algorithm-machine-learning*, 2016.

[29] Gary William Flake and Steve Lawrence. Efficient svm regression training with smo. In *Machine Learning*, volume 46, pages 271–290. Citeseer, 2000.

[30] V Veeralakshmi and D Ramyachitra. Ripple down rule learner (ridor) classifier for iris dataset. *Issues*, 1(1):79–85, 2015.

[31] Wikipedia. Weka (machine learning). [34][34]https://en.wikipedia.org/wiki/Weka_(machine_learning), 2019. [Online; accessed july-2019].

[32] APK Tool. APK Tool Documentstion. [32]https://ibotpeaches.github.io/Apktool/documentation/, 2019. [Online; accessed july-2019].

[33] genymotion. genymotion release. [59]https://www.genymotion.com/product-release-note/desktop/, 2019. [Online; accessed -April-2019].

[34] KnowBe4. Ransomware Knowledgebase. [28]https://www.knowbe4.com/svpeng-mobile-ransomware, 2019. [Online; accessed june-2019].

[35] TechBeacon. Ransomware on the rise: The evolution of a cyberattack. [29]https://techbeacon.com/security/ransomware-rise-evolution-cyberattack, 2019. [Online; accessed july-2019].

[36] Ibrar Yaqoob, Ejaz Ahmed, Muhammad Habib ur Rehman, Abdelmuttlib Ibrahim Abdalla Ahmed, Mohammed Ali Al-garadi, Muhammad Imran, and Mohsen Guizani. The rise of ransomware and emerging security challenges in the internet of things. *Computer Networks*, 129:444–458, 2017.

[37] ESE Static. The 6 biggest ransomware attacks of the last 5 years. [31]https://cdn3prodint.esetstatic.com/ESET/SG/whitepapers/2016_Rise_of_Android_Ransomware.pdf, 2019. [Online; accessed july-2019].

[38] Sana Aurangzeb. A machine learning based hybrid approach to classify and detect windows ransomware, 2018.

[39] Wei Wang, Yuanyuan Li, Xing Wang, Jiqiang Liu, and Xiangliang Zhang. Detecting android malicious apps and categorizing benign apps with ensemble of classifiers. *Future Generation Computer Systems*, 78:987–994, 2018.

[40] Nikola Milosevic, Ali Dehghantanha, and Kim-Kwang Raymond Choo. Machine learning aided android malware classification. *Computers & Electrical Engineering*, 61:266–274, 2017.

[41] AZMI AMINORDIN, FAIZAL MA, and ROBIAH YUSOF. Android malware classification base on application category using static code analysis. *Journal of Theoretical and Applied Information Technology*, 96(20):6854–6863, 2018.

[42] Drebin. The Drebin Dataset. [22]https://www.sec.cs.tu-bs.de/~danarp/drebin/, 2019. [Online; accessed March-2019].

[43] Ransomprober. Share of Android OS of global smartphone shipments from 1st quarter 2011 to 2nd quarter 2018*. [21]http://csp.whu.edu.cn/RansomProber/download.html/, 2019. [Online; accessed March-2019].

[44] Sanya Chaba, Rahul Kumar, Rohan Pant, and Mayank Dave. Malware detection approach for android systems using system call logs. *CoRR*, abs/1709.08805, 2017. URL http://arxiv.org/abs/1709.08805.

[45] Abdurrahman Pektaş and Tankut Acarman. Ensemble machine learning approach for android malware classification using hybrid features. In Marek Kurzynski, Michal Wozniak, and Robert Burduk, editors, *Proceedings of the 10th International Conference on Computer Recognition Systems CORES 2017*, pages 191–200, Cham, 2018. Springer International Publishing. ISBN 978-3-319-59162-9.

[46] Asaf Shabtai, Robert Moskovitch, Clint Feher, Shlomi Dolev, and Yuval Elovici. Detecting unknown malicious code by applying classification techniques on opcode patterns. *Security Informatics*, 1(1):1, 2012.

[47] Rahman Mukras, Nirmalie Wiratunga, Robert Lothian, Sutanu Chakraborti, and David Harper. Information gain feature selection for ordinal text classification using probability re-distribution. In *Proceedings of the Textlink workshop at IJCAI*, volume 7, page 16, 2007.

[48] María Teresa Martín-Valdivia, Manuel Carlos Díaz-Galiano, Arturo Montejo-Raez, and LA Ureña-López. Using information gain to improve multi-modal information retrieval systems. *Information Processing & Management*, 44(3): 1146–1158, 2008.

[49] Yijuan Lu, Ira Cohen, Xiang Sean Zhou, and Qi Tian. Feature selection using principal feature analysis. In *Proceedings of the 15th ACM International Conference on Multimedia*, MM '07, pages 301–304, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-702-5. doi: 10.1145/1291233.1291297. URL http://doi.acm.org/10.1145/1291233.1291297.

[50] Fengxi Song, Zhongwei Guo, and Dayong Mei. Feature selection using principal component analysis. In *2010 international conference on system science, engineering design and manufacturing informatization*, volume 1, pages 27–30. IEEE, 2010.

[51] J.R. Quinlan. *C4.5: Programs for Machine Learning*. Ebrary online. Elsevier Science, 2014. ISBN 9780080500584. URL https://books.google.com.pk/books?id=b3ujBQAAQBAJ.

[52] Ron Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, volume 14, pages 1137–1145. Montreal, Canada, 1995.

[53] K.S. Jones, P. Willett, and Jones. *Readings in Information Retrieval*. Morgan Kaufmann series in multimedia information and systems. Morgan Kaufman, 1997. ISBN 9781558604544. URL https://books.google.com.pk/books?id=Nt5nDTYQ0okC.

[54] Fairuz Amalina Narudin, Ali Feizollah, Nor Badrul Anuar, and Abdullah Gani. Evaluation of machine learning classifiers for mobile malware detection. *Soft Computing*, 20(1):343–357, 2016.

[55] M. Zubair Shafiq, S. Momina Tabish, and Muddassar Farooq. Are evolutionary rule learning algorithms appropriate for malware detection? In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, GECCO '09, pages 1915–1916, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-325-9. doi: 10.1145/1569901.1570233. URL http://doi.acm.org/10.1145/1569901.1570233.

[56] Xuchun Li, Lei Wang, and Eric Sung. A study of adaboost with svm based weak learners. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 1, pages 196–201. IEEE, 2005.

[57] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, 29(5):1189–1232, 2001.

[58] Olivier Chapelle and Vladimir Vapnik. Model selection for support vector machines. In *Advances in neural information processing systems*, pages 230–236, 2000.

Appendix A

Feature Extraction Code For APK Files

```
import os,sys,re
from nltk.corpus import stopwords
from bs4 import BeautifulSoup
from nltk.corpus import stopwords
from nltk import tokenize
from xml.dom.minidom import parseString
from nltk.stem import PorterStemmer

extension = ``.smali"
sourcesList = []
def getUniqueWords(allWords) :
    uniqueWords = []
    for i in allWords:
        if not i in uniqueWords:
            uniqueWords.append(i)
    return uniqueWords

def apk_ex():
    with open(path+`/AndroidManifest.xml',`r') as f:
        data = f.read()
```

```
    dom = parseString(data)
    nodes = dom.getElementsByTagName('uses-permission')
    for node in nodes:
        y1=str(node.toxml()).find('"')
        y2=str(node.toxml()).find('"',y1+1)
        d_red=str(node.toxml()[y1+1:y2]).split(".")
        permission = open(path+"/EX_PERMISSION.txt","a")
        if len(d_red)>0:
            permission.write(d_red[len(d_red)-1])
            permission.write("\n")

data=['abstract','boolean','break','byte','case','catch',
'char','class','const','continue','default','do','double',
'else','extends','final','finally','float','for','goto',
'if','implements','import','instanceof','int','interface',
'long','native','new','null','package','private',
'protected','public','return','short','static','super',
'switch','synchronized','this','throw','throws','transient'
,'try','void','volatile','while','assert','enum','strictfp']
def review_to_words( raw_review ):
    review_text = BeautifulSoup(raw_review, "lxml").get_text()
    letters_only = re.sub("[^a-zA-Z]", " ", review_text)
    words = letters_only.lower().split()
    stops = set(stopwords.words("english"))
    meaningful_words = [w for w in words if not w in stops]
    meaningful_words=[w for w in meaningful_words if not w
                      in data]
    #ps = PorterStemmer()
    #meaningful_words = [ps.stem(x) for x in meaningful_words]
    #meaningful_words=getUniqueWords(meaningful_words)
    return( " ".join( meaningful_words ))
```

```
def fileReader(file):
    f = open(file,``rb").read()
    return f

def Lister(path):
    for fpath, dirs,files in os.walk(path):
        for file in files:
            if extension in file:
                sourcesList.append(os.path.join(fpath,file))
class Extractor:
    def __init__(self,file):
        self.file = file

    def emailEX(self):
        emails = open(path+``/EX_EMAILS.txt",``a")
        data = fileReader(self.file)
        data=data.decode(`utf-8')
        ex_emails= list(set(re.findall(r`[a-z0-9\.\-+_]+@
                [a-z0-9\.\-+_]+\.[a-z]+', data)))

        for email in ex_emails:
                    if len(email) < 2:
                        pass
                    else:
                        emails.write(email.strip()+``\n")

    def urlsEX(self):
        urls = open(path+``/EX_URLS.txt",``a")
        data = fileReader(self.file)
        data=data.decode(`utf-8')
```

```
        ex_urls = list(set(re.findall(r'(?:https?|ftp):
                \/\/[\w/\-?=%.]+\.[\w/\-?=%.]+', data)))

        for url in ex_urls:
            if len(url) < 2:
                pass
            else:
                urls.write(url.strip()+''\n")

    def ipsEX(self):
        ips = open(path+''/EX_IPS.txt",''a")
        data = fileReader(self.file)
        data=data.decode('utf-8')
        ex_ips = list(set(re.findall(r'[0-9]+(?:\.[0-9]+)
                 {3}', data)))

        for ip in ex_ips:
            ips.write(ip.strip()+''\n")

    def apk_ex(self):
        apkf = APK(path+''.apk")
        ex_ips=apkf.get_permissions()

        for ip in ex_ips:
            ips.write(ip+''\n")

    def text_ex(self):

        ips = open(path+''/EX_TExt.txt",''a")
        data = fileReader(self.file)
        data=data.decode('utf-8')
```

```
        ex_ips = review_to_words(data)
        ex_ips=tokenize.sent_tokenize(ex_ips)
        #print (ex_ips)
        for ip in ex_ips:
            ips.write(ip.strip()+''\n")
        #ips.write(ex_ips)

    def interes_files(self):
        int_files = open(path+''/EX_DATA.txt",''a")
        words = ['base_url','ftp_",''db_",''pass",''user_pass",
        ''user_name",''smtp_",''passwd",''mysql://",''ftp://"]
        data = fileReader(self.file)
        data=data.decode('utf-8')
        for word in words:
            if word.upper() in data or word.lower() in data:
                int_files.write(''{} \t: {}\n".format(word,
                self.file))
import glob
import os
if __name__ == '__main__':
        for path in glob.glob('*.apk'):
        try:
                path=path[0:len(path)-4]
                print (path)
                x=''apktool d -f "+path+''.apk"
                os.system(x)
                Lister(path)
                print (''Working on Extracting permission")
                apk_ex()
                print (''Working on Network Features")
                for sl in sourcesList:
```

```
                    EX = Extractor(sl)
                    EX.urlsEX()
                    EX.emailEX()
                    EX.ipsEX()
                    EX.interes_files()
                except Exception as e:
                    print (``Error in apk :" ,e)
        print (``Finished")
```

Appendix B

Standard Configuration of AdaBoost in WEKA 3.8.3

Property	Configuration
base_estimator	None
learning_rate	1
n_estimators	50
random_state	None

Standard Configuration of Decision Tree in WEKA 3.8.3

Property	Configuration
max_features	0
max_depth	0
min_samples_leaf	1
min_samples_split	2
min_weight_fraction_leaf	0
presort	FALSE

Standard Configuration of Gradient Boosting in WEKA 3.8.3

Property	Configuration
learning_rate	0.1
loss	deviance
max_depth	3

max_features	None
max_leaf_nodes	None
min_samples_leaf	1
min_weight_fraction_leaf	0
n_estimators	100
presort	auto
subsample	1
tol	0.0001
validation_fraction	0.1
warm_start	FALSE

Standard Configuration of Logistic Regression in WEKA 3.8.3

Property	**Configuration**
intercept_scaling	1
max_iter	100
multi_class	multinomial
penalty	12
solver	Ibfgs
tol	0.0001
warm_start	FALSE

Standard Configuration of Random Forest in WEKA 3.8.3

Property	**Configuration**
bootstrap	TRUE
max_depth	None
max_features	auto
max_leaf_nodes	None
min_samples_leaf	1
min_samples_split	2

n_estimators	warn
warm_start	FALSE

Standard Configuration of Naive Bayes in WEKA 3.8.3

Property	**Configuration**
Priors	None
Var_smoothing	1.00E-09

Standard Configuration of Support Vector Classifier in WEKA 3.8.3

Property	**Configuration**
C	1
cache_size	200
decision_function_shape	one-vs-rest
degree	3
kernel	Redial basis function
max_iter	-1
probability	FALSE
tol	0.001
verbose	FALSE

Standard Configuration of Voting Classifier in WEKA 3.8.3

Property	**Configuration**
flatten_transform	TRUE
n_jobs	None
voting	Hard
weights	None