CAPITAL UNIVERSITY OF SCIENCE AND TECHNOLOGY, ISLAMABAD

Android Malware Detection Using the Combination of Different Static Features

by

Maryam Khadam

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

in the

Faculty of Computing
Department of Computer Science

2018

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

CAPITAL UNIVERSITY OF SCIENCE & TECHNOLOGY

ISLAMABAD

CERTIFICATE OF APPROVAL

Android Malware Detection Using the Combination of Different Static Features

by

Maryam Khadam

MCS153012

THESIS EXAMINING COMMITTEE

S. No.	Examiner	Name	Organization
(a)	External Examiner	Dr. Mansoor Ahmad	COMSATS University Islamabad.
(b)	Internal Examiner	Dr. Aamir Nadeem	CUST, Islamabad.
(c)	Supervisor	Dr. Muhammad Arshad Islam	CUST, Islamabad.

Dr. Muhammad Arshad Islam
Thesis Supervisor
November, 2018

Dr. Nayyer Masood
Head
Dept. of Computer Science
November, 2018

Dr. Muhammad Abdul Qadir
Dean
Faculty of Computing
November, 2018

Author's Declaration

I, **Maryam Khadam** hereby state that my MS thesis titled "**Android Malware Detection Using the Combination of Different Static Features**" is my own work and has not been submitted previously by me for taking any degree from Capital University of Science and 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.

(Maryam Khadam)

Registration No: MCS153012

Plagiarism Undertaking

I solemnly declare that research work presented in this thesis titled "**Android Malware Detection Using the Combination of Different Static Features**" 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 Capital University of Science and Technology 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 my name on the HEC/University website on which names of students are placed who submitted plagiarized work.

(Maryam Khadam)

Registration No: MCS153012

Acknowledgements

All praise and exaltation are due to ALLAH (S.W.T) The creator and Sustainer of all seen and unseen worlds. 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 Arshad Islam for his directions, assistance, and guidance. I sincerely thanked for his support, encouragement and technical advice in the research area. I am 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 words can ever be sufficient for the gratitude I have for my mother 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.

Aameen

Maryam Kadam

Abstract

Recent smartphones offer a large number of services and with the increase of services security threat also increases. Use of personal data is not secure due to malware. Malware hacker can steal user personal information like messages, contacts, phone number, email accounts etc., and use this information for different criminal activities. Malware is basically a piece of code that is built with harmful intention. Malware attack the smartphone by using Android applications. Nowa- days large number of Android applications has been developed but a user does not sure about applications whether these applications are secure or not. For the detection of malware many different approaches used some author use static and some use dynamic malware detections techniques. Both static and dynamic malware detection approaches have some advantages and some limitations. In our research, we used static malware detection approach for the detection of malware. We aim to compare the performance of static analysis techniques by using three categories of features (permission, intents, and API call) for the detection of malware. Although static malware detection has already been used but there is lack of research done that analyses all features of apk files on the single dataset. This approach examines the permissions tags and intent filter within the manifest file. Moreover, we analyze the source code of the Android application; this approach examines the Java code for Android applications. Our experiment results show remarkable results when we used info gain feature selection algorithm. We have obtained 97% accuracy rate in case of intents by using Decision tree. We have observed that overall performance of Decision Tree is good in case of info gain feature selection algorithm. Decision tree shows the accuracy value of 97% in case of intents and 93% in case of permission and 95% for the combined feature set and 85% for API Calls by using info gain algorithm. The results showed that the individual feature set of API Calls are not good for Android malware detection.

Contents

List of Figures

List of Tables

List of Abbreviations

Waikato Environment for Knowledge Analysis (WEKA)

Random Forest (RF)

Naïve Bays (NB)

Gradient boosting (GB)

Decision Tree (DT)

False Positive Rate (FPR)

True Positive Rate (TPR)

Android package Kit (APK)

Principal ComponentsAnalysis (PCA)

Application Program Interface (API)

Chapter 1

Introduction

Malware is malicious code that is expressly designed with unsafe aims, corresponding to viruses' Trojan horses or worms. Malware is growing at an alarming rate with approximately one million Android applications are actually malware (Internet security threat report 2015). The worldwide damages from malware exceed 400 billion per year (cybercrime report 2014). Android has grabbed up to 87.5% of the global mobile phone market (Arjun Kharpal 2016). Consequently, an increasing number of security threats emerged. Now-a-days people store all their private information, financial information and personal information like photographs, videos bank details in their cell phones, so these cell phones turn out to be more helpless and an assailant can attack their devices to steal their information. Due to the openness of the Android system, it becomes simple for an assailant to create malware which can hurt any Android gadget (Barsiya, et al. 2016). These situations motivated us to research in malware analysis. There are three malware detection approaches:

1.1 Approaches

1.1.1 Static Approach

In the static technique, application code is analyzed without actually running it. The static approach is very fast and it does not require any effort to execute the malicious code for identification. Application code is analyzed without actually executing it. The advantage of the static approach is quickly detected malware, consume fewer resources low cost and less time consuming because does not involve execution of the application. Disadvantage is that only detect known malware types (Vidhya Rao. et al. 2017). The most commonly used static features are the Permission and API calls.

1.1.2 Dynamic Approach

It is a detection technique which aims at evaluating malware by executing the application and the main advantage of this technique is that determines the application behavior during runtime and provides deep analysis and high detection rate with unknown malware detection. Dynamic technique checks the application code as well as system calls during application execution and monitors its interaction with the system. During this analysis, system calls are observed that are more likely to occur in malware than in normal applications. These behavior similarities among malicious apps could be used to detect new malware. A disadvantage is that this technique consumes large storage space and requires high power consumption. More time required to observe the appearance of the malicious activity (Vidhya Rao. et al. 2017).

1.1.3 Hybrid Approach

Hybrid approach is a combination of the static and dynamic approach. It is an innovation or technique that can incorporate run-time information extricated from

the dynamic approach into a static examination calculation to distinguish conduct or malignant usefulness in the applications. The hybrid analysis method involves combining static features obtained while analyzing the application and dynamic features and information extracted while the application is executed. Though it could increase the accuracy of the detection rate, it makes the analysis process is time-consuming and increases cost (Lifan Xu, et al. 2016).

1.2 Motivation

This research focuses on static techniques because the static approach is quickly detected malware, consumes fewer resources low cost and less time consuming because does not involve execution of the application. We aim to compare the performance of static analysis techniques by using three categories of features (permission, intents, and API call) for the detection of malware. Although some other researcher also uses static analysis in their research there is no research done that analyses all features of apk files on the same dataset. We start our static analysis by examining the manifest file. This approach examines the permissions tags and intent filter within the manifest file. Moreover, we analyze the source code of the Android application; this approach examines the java code for Android applications.

Following Features Sets are Available for Static Malware Detection

- Permission-based analysis
- Intent-based analysis
- Function call-based analysis
- API call analysis
- Opcode based analysis
- Network-based analysis

This work focuses on three types of features set for static analysis permission, intents, and API call analysis.

1.2.1 Permissions

In the case of smartphones applications, researchers concentrate on permissions mechanisms to determine what is allowed to do in applications. A client is displayed with a list of permissions at the time of app installation and can choose to grant all the permissions or reject it. If a user doesn't grant these permissions, it will result in stopping the installation. This could be a great security measure because the operations run by one application cannot affect or interfere with other applications. The permissions requested for the resources are in the Android Manifest.XML file(Saba Arshad, et al. 2016).

1.2.2 Intent

The intent is a complex messaging system on the Android platform (Ali Feizollah, et al. 2017). This is a method for controlling what applications can do once they are introduced on Android. Android Intents are defined in the AndroidManifest file. Inter-process communication is one of the most notable features of the Android framework as it allows the reuse of components across process boundaries. It is used as the gateway to access different sensitive services in the Android framework. In the Android platform, this communication system is usually driven by a late runtime binding messaging object known as Intent. Intent objects provide an abstract definition of the operations an application intends to perform (Ali Feizollah, et al. 2017). Intents have three components – action, category, and data.

The action component describes what kind of action is to be executed by the Intent such as MAIN, CALL, BATTERY LOW, SCREEN ON, and EDIT. Intents specify the category they belong to, such as LAUNCHER, BROWSABLE, and

GADGET. The data components provide the necessary data to the active component. For instance, CALL action requires the phone number, and EDIT action needs document or HTTP URL to complete the action (Ali Feizollah, et al. 2017).

1.2.3 Application Program Interface (API Calls)

API calls are basically a piece of code act as an intermediate between program and system. Variables of different programs requests to the system for the use of different resources by using API calls. By using API calls program requests to the system for different resources.

1.3 Purpose

The purpose of this research is to evaluate the Android Malware detection capability by using three categories of features (permission, Intents, and API call) for Android applications. It will also help the Android application developers to write software that cannot be exploited easily.

Following is the list of categories of features for static analysis

- Permission-based analysis
- Intent-based analysis
- API call analysis

1.4 Scope

In this research, we use static analysis on the combination of features and apply machine learning algorithms to detect the malware. Research has shown the effectiveness of machine learning systems in Android Malware detection.

In a machine learning based malware detection system, selection of the feature set from the dataset is one of the most critical steps. Feature selection itself depends upon the analysis method through which they are extracted. It is the analysis technique that determines the compatibility of features with the classification algorithm. Due to the lack of time we have selected only three categories of the features and used these features in our research.

1.5 Problem Statement

A lot of solutions (R. Sato, et al. 2013) (Daniel Arp, et al. 2014) (Idrees F, et al. 2014) (Almin, et al. 2015) (Saidah Mastura A. Ghani, et al. 2015) have been developed for malware detection that significantly improved the user's security. But a unified approach by using these three features set (permission, intent and API calls) and comparison of these three categories of features for the detection of Android malware is missing.

1.6 Research Question

1. Which category (permission, intent and API call) of the static features are most effective for malware detection?
2. Can accuracy be improved using a combination of these features?
3. Which classifier (Gradient Boosting, Decision tree, Random forest. Naive Bayes) performs well for the detection of malware.

Our research will be focused on these research questions with reference to the machine learning algorithms.

1.7 Proposed Solution

1.7.1 Permission and Intent Analysis

We start our static analysis by examining the manifest file. This approach examines the permissions tags and intent filter within the manifest file to check the malicious behavior of applications.

List of the most recently used feature list of permission available in Table 1.1 (De Capitani di Vimercati, et al. 2017), (Zhenxiang Chen, et al. 2015)

Table 1.1: Feature List of Permissions

INTERNET
ACCESS_NETWORK_STATE
WRITE_EXTERNAL_STORAGE
WAKE_LOCK
READ_PHONE_STATE
ACCESS_WIFI_STATE
GET_ACCOUNTS
VIBRATE
BILLING ACCESS_COARSE_LOCATION
SEND_SMS
RECEIVE_SMS
WAKE_LOCK
READ_SMS
ACCESS_COARSE_LOCATION
ACCESS_FINE_LOCATION
READ_CONTACTS

List of the most recently used feature list of intents is available in Table 2. (Ali Feizollah, et al.2017).

Table 1.2: Feature List of Intents

SEND_MULTIPLE
SCREEN_OFF
USER_PRESENT
SEARCH
PICK
DIAL
GET_CONTENT
EDIT
MEDIA_MOUNTED
BOOT_COMPLETED
SENDTO
SCREEN_OFF
TEXT
SEND
PACKAGE_ADDED
SCREEN_ON
CALL

1.7.2 Static Code Analysis

We analyze the source code of the Android application; this approach examines the java code of Android applications on the same dataset that has not been considered by previous researchers. Java source code is analyzed in the dex file. For source code analysis we use the API call of the Android application by using dex file of the android application.

1.7.3 Comparison

After analyzing all features of malware detection techniques on the same data set we compare our results to check which techniques can provide better accuracy for malware detection.

1.8 The Significance of the Solution

In android malware detection, a prominent research gap is the absence of a study of all features of the Android application on the same data set. The significance of this solution is to provide a comparison of malware detection accuracy using the following categories of feature set:

1. Permission
2. Intents
3. API call

All of these features belong to static analysis. Identify the attribute after merging the attributes of all these features.

1.9 Tools and Techniques

Following tools and techniques will be used in this research.

1. OS: Ubuntu (version LTS 16.04) 64 bit
2. Weka tool
3. Machine Learning Classifiers (Gradient Boosting, Decision Tree, Random forest. Naïve Bayes)
4. APK Tool
5. Python script for feature extraction
6. Python script for the feature vector
7. Microsoft Excel

Chapter 2

Related Work

Android security against rising malware has been an interesting issue among the analysts for a couple of years. Numerous static and dynamic techniques have been suggested for the identification of the vulnerability of the mobile applications. Both the static and dynamic analysis methods have their own benefits and limitations. This chapter presents an extensive overview of the research work of static analysis for Android malware detection.

Static analysis is a procedure to examine the behavior of an Android application without actually executing it (Vidhya Rao, et al. 2017). The static analysis is very helpful in recognizing malicious performance that may not trigger before the occurrence of a particular condition.

2.1 Permission-Based Static Analysis

Almin, et al. (2015) laid their research upon analysis of permissions used by an application during installation. The authors proposed a system called Android Application Analyzer, that perform an analysis of the installed application to check whether the application is harmful or not. In the proposed system, the k-means clustering algorithm is used during permission authorization process to determine that the application contains the malicious program or not. The results will use

naïve Bayesian classification algorithm to check accurately either the application is malicious or benign. They compare their result with other antivirus solutions like McAfee, Kaspersky, and Avast. They found that their application detects that malware that is undetected by the antivirus software. The drawback of their system is that if an unknown malware family is detected, then it's a new cluster has to be created considering the permissions used by the new family.

2.2 Permission and Intent-Based Static Analysis

R. Sato, et al. (2013) analyze just Android Manifest file. For the malware detection, they proposed a lightweight approach, and for the experimentation and analyzing of their approach they used genuine samples of Android malware. They demonstrate that the new technique can successfully distinguish Android malware, notwithstanding when the example is obscure. In the first step, decompile manifest file for extraction of the specific data. Further, they contrast the extracted data with the keyword records that are given in this new strategy. At that point, compute the malignancy score of the example by contrasting the data in the extricated list. in the event that the threat score surpasses the limit esteem, the example is judged as malware. The results of their trial demonstrated that the right proportion of distinguishing clean application is 91.4%, recognizing malware tests are 87.5%, and it is 90.0% altogether. A few restrictions were seen amid experimentation that demonstrates that some malware tests were not identified by the proposed strategy. Along these lines, the genuine test is to join this strategy with different strategies to understand a considerably more exact identification technique.

Idrees F, et al. (2014) proposed the permissions and intent filters patterns-based identification of malware and benign applications. Authors perform classification by applying different machine learning algorithms. In the first phase of their proposed method extract permission and intents from Android Manifest file. The next

phase is for processing the collected data into a suitable format to input the classifiers. In the last stage, data is provided to the classifiers that assess the refined data and classify the apps into malicious and benign applications. Authors use the dataset consisting of of45 malware and 300 benign applications. They found that 35 permissions are used by both malware and benign applications. The machine learning techniques K*, Naïve Bayes, and Prism were used for classification. The strength of their proposed approach is the combination of intents and permissions in an efficient manner. However, they ignored other characteristics of malware that are triggered by different events. Sometimes, malware has the least dangerous permissions and normal intents. However, malware often performs certain malicious activities during execution.

Ali Feizollah, et al. (2017) done their research on Android Intents as semantically rich features for malware identification. Authors show that Intents are powerful features for the detection of malware in Android applications. Authors use machine learning algorithms to classify intents and related malware. They named their technique AndroDialysis. The main module of the proposed technique called a detector. Detector use four other sub-modules (decompiler, extractor, intelligent learner and decision maker) to performs the task of detection. The first submodule is responsible for decompiling the Android applications. The second submodule performs extraction and extract Android intents from the manifest file of the Android application. The third module named intelligent learner sub-module uses the features database to extract data. Bayesian network machine learning algorithm uses to learn the pattern of the data. The fourth module makes decisions about application whether it is clean or malicious. They achieve malware detection rate of 91% using Android Intent as a major feature and got a detection rate of 83% using Android permissions.

2.3 Permission and API Call Based Static Analysis

Daniel Arp, et al. (2014) proposed DREBIN, a lightweight methodology for the revelation of Android malware that engages perceiving malicious applications directly on the phone. DREBIN plays out an expansive static examination, assembling however many features of an application as would be prudent. In the initial step, DREBIN statically examines a given Android application and concentrates diverse feature sets from the application's show and dex code installing in vector space. The extricated feature sets are then mapped to a joint vector space, where patterns and feature combination can be examined. The implanting of the feature sets empowers us to distinguish malware utilizing proficient strategies of machine learning, for example, linear Support Vector Machines. In an assessment of 123,453 applications and 5,560 malware tests, DREBIN beats a few related methodologies and recognizes 94% of the malware with a couple of false alerts.

Mrs. Gunjan Kapse, et al. (2015) proposed a more extensive static investigation of utilization covering more features with permission. This expands the malware recognition quality. The diverse features which would be examined are the permission and dangerous API calls. Doing this would identify the malware with more exactness. The application would be delegated clean or malware effectively. The first step is the extraction of permission and API calls. permissions are extracted from the manifest file and API calls are extracted from dex files. Weights are doled out to permission and API calls in view of their pernicious nature. In the event that the aggregate weight of permission and API calls of an application surpasses a predefined edge, at that point, the application is sorted as malware. Permission-based detection method has given 81.3% accuracy rate with 0.87 TPR and 0.25 FPR. Permission and API call detection-based method have given 85.0% accuracy rate with 0.88 TPR and 0.20 FPR. The proposed technique for identification of malware on Android application by using permission and API Calls detect malware more accurately than only permission-based malware detection.

2.4 API Call Based Static Analysis

Saidah Mastura A. Ghani, et al. (2015) played out a static examination in both malware and clean Android application. The source code of malware and clean application have compared. This examination looks at the malware and clean applications which indistinguishable from each other. This implies both malware and clean applications in this examination are indistinguishable however the malware applications were at that point infused by noxious code. This strategy has two stages which include extraction stage and feature examination stage. In include extraction stage both malware and clean applications have extracted by utilizing Androguard, a reverse engineering tool. The antivirus has infused into the clean applications previously they are extracted. This is done to ensure the clean applications are extremely perfect from the malware. In the feature correlation stage, the source code amongst malware and clean applications have analyzed. At that point, the separation of source code amongst malware and clean applications have distinguished. At that point, the parameter is ordered into two classes. The classifications are APIs and manager classes. There are 238 APIs in Android from level 1 API to level 22 API. The primary classification has identified the majority of the APIs engaged with both malware and clean applications. From that point onward, the APIs classes which required with manager classes have recognized. The example of API with chief classes is Telephony Manager, SmsManager, Power Manager, Connectivity Manager, and Notification Manager. The after effects of malware and clean applications from the two classifications have drawn utilizing charts. The recurrence of APIs and manager classes classifications have contrasted with clean and malware applications to recognize the most as often as possible utilized API and manager classes by malware. The outcome from this examination is exceptionally valuable particularly for Android applications engineers to take some mindfulness when utilizing the Android APIs and manager classes.

ElMouatez Billah Karbab, et al. (2018) propose MalDozer, a straightforward, viable and productive system for Android malware recognition in view of machine learning strategies and raw sequences of API strategy brings with a specific end

goal to recognize Android malware. MalDozer consequently extracts and learns in the malware and the clean sample from the real sample to recognize Android malware. Amid the preparation, MalDozer can naturally perceive suspicious behavior utilizing just the sequence of raw technique brings in the assembly code. MalDozer accomplishes a high precision in malware discovery under different datasets, including Malgenome and Drebin. The outcomes demonstrate that MalDozer can effectively identify malware and credit them to their genuine families with a F1-Score of 96% - 99% and a false positive rate of 0.06% - 2%, under all tried datasets and settings. In addition, MalDozer can productively keep running under numerous arrangement models, extending from servers to little IoT gadgets. The burden of this system is that could oppose certain confusion procedures on the grounds that exclusive consider the API technique calls.

2.5 Hybrid Based Static Analysis

Kim, et al. (2012) introduced the astatic analyzer called SCANDAL that examines the applications at the Dalvik byte-code level and identifies the privacy spillage. The proposed technique searches for the flow of data from the application to any remote server. The Dalvik byte-code contains method invocation and other instruction that modifies the order of execution of code and jumbles the code. The byte-code analysis detects the possible execution path of an application. Authors have inspected 90 applications extracted from Google play and 8 malwares from the third-party app store. They discovered that 11 applications from the Google app store and 8 malwares involve data leakage. The applications which utilize reflections for information leakage is not supported in the SCANDAL. The authors did not automate reflection semantics to recognize the security leakage in malignant applications taken from the third-party app store.

Faruki P, et al. (2013) proposed a signature based statistical analyzing technique AndroSimilar to detect the newly emerging variants of known malware created using code obfuscation and re-packaging. The proposed technique compares a

malware's signatures; with the signature produced by AndroSimilar for the application under analysis. Authors use the dataset based on 6779 apps collected from the Google Play store and other third-party application stores. Authors apply different code obfuscation techniques to alter the signature of an application under analysis. They test 456applications for method renaming, insertion of spam codes encryption of different strings and altering the control flow. They succeed in identifying up to 60% samples correctly. The AndroSimilar faces some limitations too. Their technique uses the comparison of signatures for identifying malware and benign apps. However, the employed signature database is limited to handle malware from different families. Moreover, new families of malware remain undetected.

Zheng M, et al. (2013) presented a signature-based malware detection system called Droid analytics. Authors extract op-codes from the application and analyze. The proposed system generates signatures and pairs them with the known malware after extracting the malicious contents. The proposed technique produces signatures at 3 levels. First level signatures are generated at the method level. These signatures are generated on the basis of API calls. The second-level signatures are the class level. The third-level signatures are generated by combining the signatures of all the classes. The Droid analytics correctly identifies 2,494 malware samples that belong to 102 different malware families. Further, the proposed system correctly identifies 342 repackaged malwares from 06 different malware families. The Droid analytics suffers from some limitations too. He detects malware on the basis of signature produced through classes mostly find in malware families. However, during experimentation, it was observed that some signatures are common in both the benign and malware. For the detection of repackaged malware, the similarity score is used that does not suffice to produce a 100% accurate solution and may produce false positives.

Singla, et al. (2015) considered different features from the header file and opcode frequency to distinguish the malware from benign apps. They collected the number of function calls and distinguished them in malware and clean samples. They collected the exact numbers of function calls by creating a common list of all

known function calls and then used the string-matching function to gain the result. Operational Code was the second parameter they used for malware identification. They extracted a global list of opcodes and matched the strings; they extracted from under observation apps and identify the same no of calls made by opcode by these apps. Further, they identify that occurrence of specific opcode frequency was higher in malware as compared to that in clean files. They proposed on the need for some dynamic technique which could classify some advanced malware. Such techniques execute malware and trace its behavior.

Mehadi Hassan, et al. (2017) proposed a straight time function call graph (FCG) vector representation based on clustering for improved accuracy. FCG based features safeguard the basic data in malware code as functions and the caller-callee connection between them. The author proposed a malware characterization strategy that gatherings malware tests into malware families. The strategy in light of FCGs, yet not at all like past works, defeat the execution overhead connected with the FCG based approach by utilizing a novel method to change over FCG representation into a vector representation. The proposed system has the accompanying preferences. It is quicker contrasted with past works for FCG based malware order. It has a higher precision contrasted with past works. The chart feature vector, extracted from FCGs, can be effectively joined with other non-diagram features.

2.6 Critical Analysis

After a comprehensive study of state-of-the-art techniques for Android malware analysis, we summaries the strengths and weaknesses of the current approaches shown in Table 2.1.

TABLE 2.1: Critical analysis of literature review

References	Methodology	Strength	Weakness
Kim, et al. (2012)	Static Bytecode	Identify privacy leakage Low cost	Expensive in terms of time and memory.
R. Sato, et al. (2013)	Permissions and intents	Detect unknown malware sample	Produces result just based on the Manifest file.
Faruki P, et al (2013)	Static analysis for repackaged software	Applied different code obfuscation techniques to alter the signature of an application under analysis.	High false positives ratio.The signature database was limited
Zheng M, et al. (2013)	Static, Signature-based, opcode	Three level signatures are used efficient to identify repackaged software	The false positive ratio is high because some signatures are common in both malware and non-malware applications.
Daniel Arp, et al. (2014)	API calls, Permissions, Network address	Try to cover these features combine	Intents not used
Idrees F, et al. (2014)	Permissions and intents	Help to improve the efficiency of the anti-malware programs	Generates results only on the basis of the manifest file.
Almin, et al. (2015)	Permission-based static analysis, machine learning for classification	Lightweight and quick approach	Failed to detect new malware families
Singla, et al. (2015)	Static, opcode, function calls	Identify that occurrence of specific opcode frequency was high in malware	Unable to classify malware that uses code obfuscation
Saidah Mastura A. Ghani, et al. (2015)	API calls	The outcome from this exploration is extremely helpful particularly for Android applications engineers to take some mindfulness when utilizing the Android APIs	Machine learning tools not used
Gunjan Kapse, et al. (2015)	Permissions and suspicious API calls.	It yields better TPR and accuracy than Permission based Malware detection.	API calls are missing
Feizollah, et al. (2017)	Static analysis, intents, and permission	Covering all the aspects that intents can play to identify malware	His study alone is not sufficient to detect the wide range of malware
Mehadi Hassan, et al. (2017)	Function call	It has a 93% classification accuracy	Only examine function call
ElMouatez Billah Karbab, et al. (2018)	API call	Accomplishes exceedingly exact malware identification and family attribution.	Only consider the API method Calls.

Table 2.1 describes the strengths and weaknesses of existing approaches. This table indicates that earlier researchers concentrated on one aspect and ignores another basic aspect of an application vulnerability and also ignore combine features aspect of an Android application.

In previous work, some authors use intents and permission some use API calls but there is a lack of research that uses the combination of intents, permissions and API calls on same data set to analyze the malware applications.

Chapter 3

Research Methodology

This chapter covers the methodology of our research. It explains the detail of our methodology steps one by one. It explains the detail of the dataset, feature extraction process and training of machine learning based classifiers.

Figure 3.1 depicts the overall working of the research methodology. The process starts working with the input in the form of APK files. Although some other researcher also uses static analysis in their research there is no research done that analyses all types of features of apk files on the same dataset. Each Android application is the part of the Android Package Kit (APK). Each APK is a compressed or zip file that consists of application classes (i.e., .dex files), resource files, and a manifest (.xml) file. The class files contain the source code of the application functionality. Resource files hold information related to graphical or audio components (e.g., images, clips, etc.). The manifest file describes the intents and permission details of the android application.

1. The first phase is the disassembling phase apk tool is used for disassembling the apk files. Android apk file is given as input to the apk tool that extracts XML file and java files from it (Winsniewski R. 2012).

2. In the next step extracts the application's permissions and intents data from the manifest file and API calls from the java file.

3. After extraction of permissions, intents and API calls these extracted features to act as feature vectors (for the machine learning-based classifier).

4. These feature vectors are used to train machine learning classifier. Once the classifier is trained, it will be able to identify or classify the given android apps into malware or benign (based on similar static features).

3.1 APK Tool

APK tool is the tool for reverse engineering. It can decode resources to nearly original form and rebuild them after making some modifications (Winsniewski R. 2012).

3.1.1 APK Tool Features

- Disassembling resources to nearly original form (including resources, classes. Dex and XML).
- Rebuilding decoded resources back to binary APK/JAR
- Organizing and handling APKs that depend on framework resources
- Smali Debugging
- Helping with repetitive tasks

3.2 Research Methodology

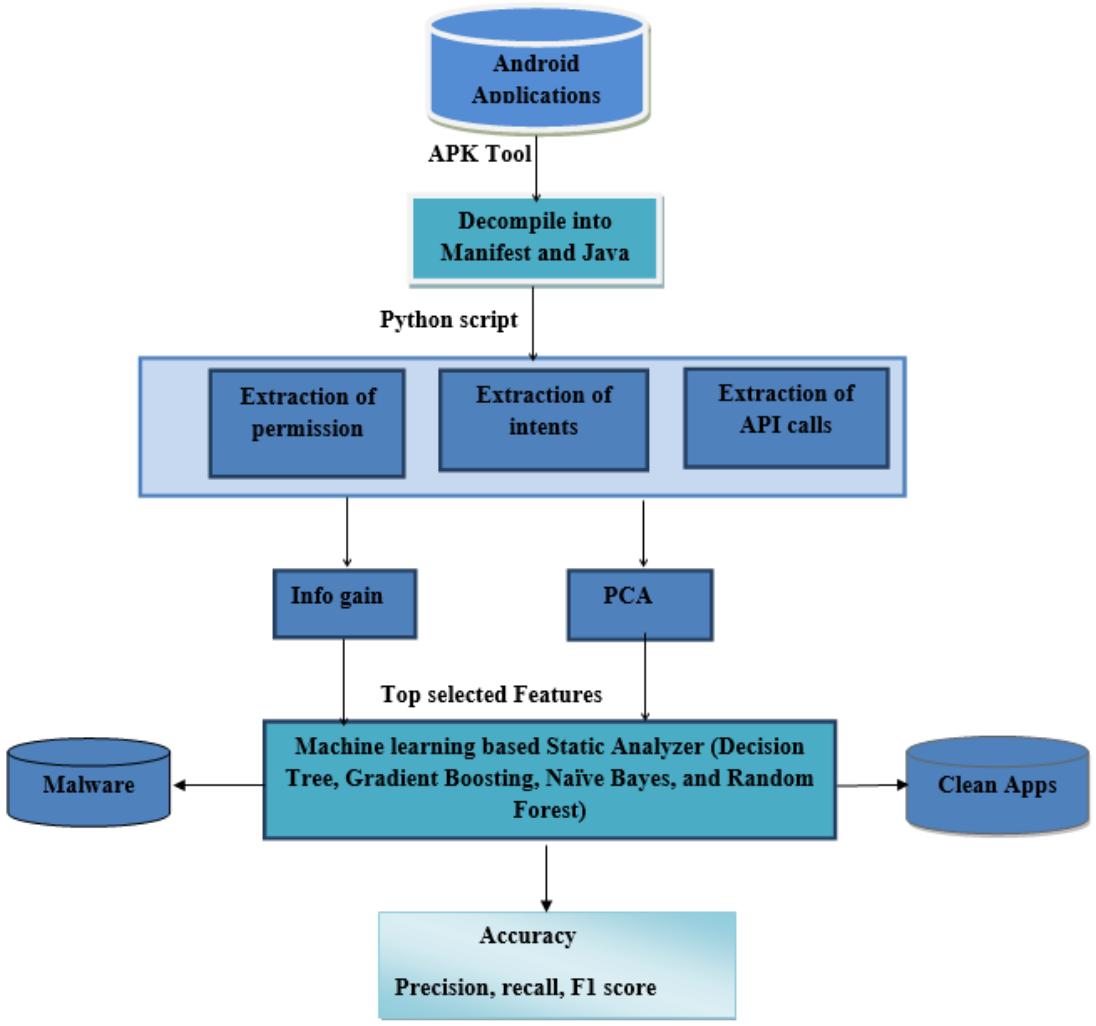


FIGURE 3.1: Architecture of Overall Methodology

3.3 Data Collection

Our dataset consists of applications that are in the form of .apk files. The benign dataset is collected randomly from the Google play store. The benign dataset consists of 500 applications that belong to 28 different categories. Different categories of applications are available on Google play store. Table 3.1 shows the categories of benign applications. For malware, we acquired Drebin dataset1 that consists of thousands of malwares based on several families. Drebin dataset is one of the

most used malware datasets. For research purpose, Drebin dataset free available. The dataset contains 5,560 applications from 179 different malware families. Table 3.2 shows the different families of malware applications. The samples have been collected in the period from August 2010 to October 2012 (Arp et al., 2014). We used 500 malware and 500 clean applications. For the training of our dataset, we used 70% of our total data that consist of 350 applications and 30% dataset used for testing that consist of 150 applications.

TABLE 3.1: Categories of Applications in the Benign Dataset

S.NO.	**Applications categories**	**Count**	**S.NO.**	**Applications categories**	**Count**
1	Health & Fitness	20	15	Travel and local	20
2	Art & Design	15	16	Food & Drink	15
3	Beauty	10	17	Lifestyle	10
4	Business	20	18	Video Players & Editors	20
5	Communication	15	19	Weather	15
6	Education	20	20	Social	30
7	Event	15	21	Shopping	15
8	House & Home	20	22	Tools	20
9	Sports	30	23	Parenting	10
10	Productivity	15	24	News & Magazines	15
11	Photography	30	25	Music & Audio	30
12	Camera	15	26	Medical	15
13	Finance	20	27	Entertainment	20
14	Auto & vehicles	10	28	Music & Audio	10

[1](https://www.sec.cs.tu-bs.de/ danarp/drebin/)

TABLE 3.2: Families of Application in the Malware Dataset

S.NO.	**Malware family**	**Count**	**S.NO.**	**Malware family**	**Count**
1	Plankton	20	15	SmsWatcher	20
2	DroidKungFu	15	16	UpdtKiller	15
3	GinMaster	10	17	Gappusin	10
4	FakeDoc	20	18	Proreso	20
5	FakeInstaller	15	19	Mobsquz	15
6	Opfake	20	20	Cosha	30
7	BaseBridge	15	21	SpyMob	15
8	Nisev	20	22	Coogos	20
9	Adrd	30	23	Updtbot	10
10	Kmin	15	24	Ackposts	15
11	Geinimi	30	25	Fatakr	30
12	DroidDream	15	26	Vidro	15
13	FakeRun	20	27	Booster	20
14	Iconosys	10	28	EWalls	10

3.3.1 Training of Machine Learning Based Analysis

For training of machine learning based analyzer we extract two important features permissions and intents from the manifest files of an android application and API calls from dex files of an android application.

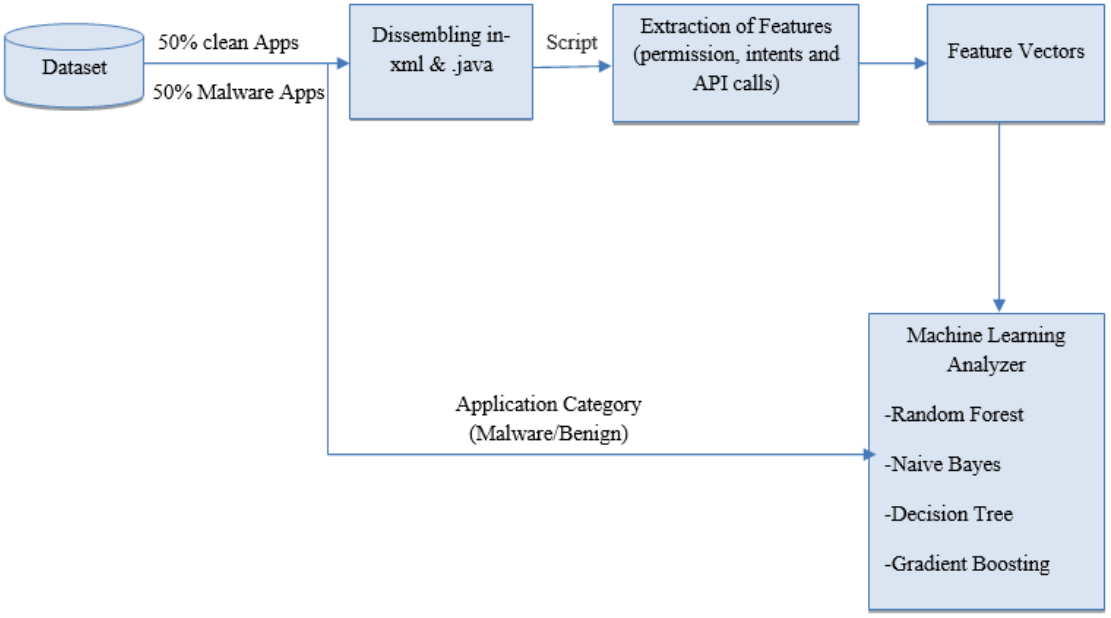


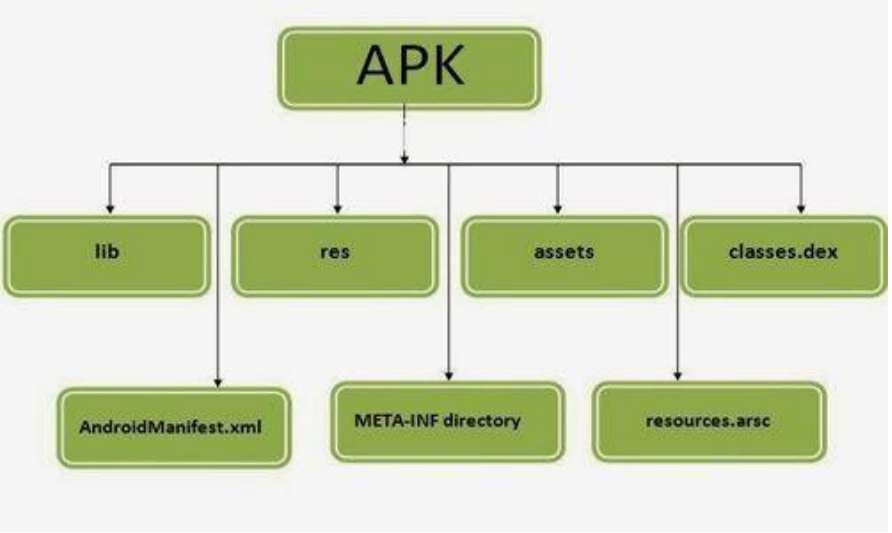
FIGURE 3.2: Training of Machine Learning Based Static Analysis

Figure 3.2 depicts the complete training process of the machine learning based static analysis.

For training the classifier, 70% of the dataset (380 applications) is used for training purposes. Remaining 30% (120 applications) is used for testing. This 70% of the dataset that is used for training purpose consist of 50% clean and 50% malware applications. Applications are disassembled into .xml and .java files using APK Tool. The next step is feature extraction step by using feature extraction script permissions and intents are extracted from the manifest file and API calls are extracted from dex files. After extraction of all these features converts these features into feature vectors. These feature vectors along with application category labels (i.e., malware or benign) are used to train a machine learning based static analyzer (Random Forest, Naive Bayes, Decision Tree, Gradient Boosting).

3.3.2 Extraction of Static Features

Each Android application is the part of the Android Package Kit (APK). Each APK is a compressed or zip file that consists of application classes (i.e., .dex files), resource files, and a manifest (.xml) file. The class files contain the source code of the application functionality. Resource files hold graphical or audio components (e.g., images, clips, etc.). The manifest file describes the intents and permission details of the android application. We start our feature extraction process by acquiring the APK file using the feature extraction script. The feature extraction script extracts all the permissions and intents from the manifest file and API calls from the dex files.



To extract permissions and intents from android manifest (.xml) file, we have written a python script which reads the tags of permissions and intents from manifest file and export these features into a .csv. By using the .csv file we have extracted the feature vectors of required features.

```
from xml.dom.minidom import parseString
import os
import sys

import glob
for a in glob.glob('*.apk'):
        print "apktool working"
        try:
                x="apktool d -f "+str(a)
                os.system(x)
                b=a[0:len(a)-4]


                if os.path.isfile(str(b)+"_intent.csv"):
                        os.remove(str(b)+"_intent.csv")|
                        os.remove(str(b)+"_permission.csv")

                with open(str(b)+'/AndroidManifest.xml','r') as f:
                        data = f.read()

                dom = parseString(data)
                nodes = dom.getElementsByTagName('intent-filter')
                intents=open("i/"+str(b)+"_intent.txt","a")
                permission=open("p/"+str(b)+"_permission.txt","a")
                intents.write("intent_name,Vector,type")
                intents.write("\n")
                print "Extracting Intents"
                for node in nodes:
                        y1=str(node.toxml()).find('''"''')
                        y2=str(node.toxml()).find('''"''',y1+1)
                        intents.write(str(node.toxml())[y1+1:y2])
                        intents.write(",1")
                        channels =node.getElementsByTagName("data")
                        if channels ==[]:
                                intents.write(",explicit")

                        else:
                                intents.write(",implicit")
                        intents.write("\n")
                intents.close()
                print "Extracting permission"
                permission.write("permission_name,Vector")
                permission.write("\n")
                nodes = dom.getElementsByTagName('uses-permission')
                for node in nodes:
                        y1=str(node.toxml()).find('''"''')
                        y2=str(node.toxml()).find('''"''',y1+1)
                        permission.write(str(node.toxml())[y1+1:y2])
                        permission.write(",1")
                        permission.write("\n")
                permission.close()

                os.rename("i/"+str(b)+"_intent.txt","i/"+str(b)+"_intent.csv")
                os.rename("p/"+str(b)+"_permission.txt","p/"+str(b)+"_permission.csv")
        except:
                print " \n \n***** Error in File *********", a ,"\n \n "
```

FIGURE 3.3: Feature Extraction Script

Detail of the feature extraction script that takes an APK file and dissembling them into dex and XML files:

1. In step one script takes apk files as an input and by using Apk tool decompiles apk files.

2. In the second step after de-compilation, we get dex files, resource files, and XML files.

3. In the third step, the script read the android manifest.xml file and extracts all intents and permission from manifest file and stores these intents and permission in.CSV files.

The same process is repeated for all apk files.

Figure 3.3 shows the python script step by step working

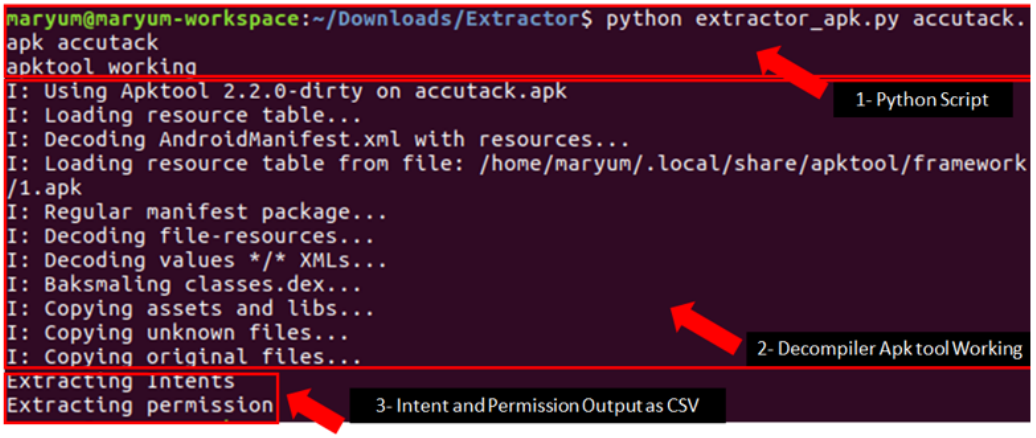


FIGURE 3.4: Python Script Working

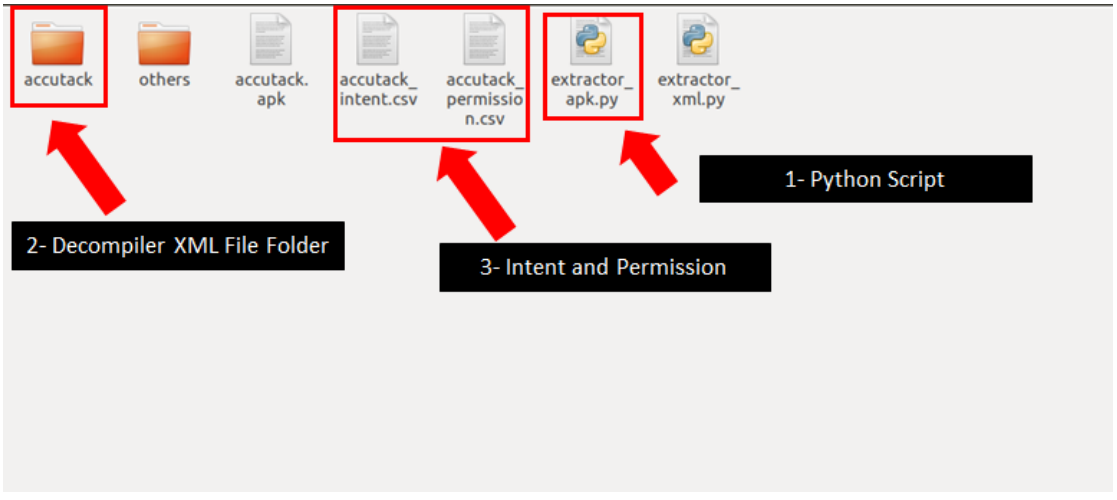


FIGURE 3.5: CSV Files

Same process repeat for non-malware applications and CSV files of both malware and non-malware are used for feature vector.

Detail of the feature extraction script that takes an APK file as an input and dissembling them into dex files for extraction of API Calls:

1. In step one script takes apk files as an input and by using Apk tool decompiles apk files.

2. In the second step after de-compilation, we get dex files, dex files contain classes.dex and every class use some methods we can call methods by using classes.

3. In the third step, we created a call graph by using classes. In our call graph every method is a node and when one method calls to other methods it creates an edge and every node in our call graph is a feature of API call.

3.3.3 Feature Vector

Intents and permission CSV files are used for feature vector. Feature vector script read CSV files of both malware and non-malware application and print all features of malware and non-malware 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 comprising a set of 1029 android permissions extracted from apk files. For every application at its location in the dataset, we produce a binary sequence:1, if permission exists in the data set 0, otherwise.

The recognized permissions are arranged as a sequence of 0 and 1. The presence of a specific permission is denoted by the value 1 and the absence is denoted by the value 0 in the list.

Following sequence represents an example of the permission vector for a clean application

3.3.4 Permission Vector for non- Malware

1 0 0 1 1 0 1 1 1 0 0

1 1 0 1 0 0 1 1 1 0 1

0 0 0 0 0 0 0 0 0 0 1

0 0 0 0 0 0 0 0 0 0 1

0 0 0 0 0 0 0 0 0 0 1

0 0 0 0 0 0 0 0 1

1 at the end of this feature vector represents the permission vector for a clean application.

Following sample represents the permission vector for a malware application:

3.3.5 Permission Vector for Malware

1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1

0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 1 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 at the end of this feature vector represents the permission vector for a clean application.

Here, we remove redundant (unnecessary and repeated) permissions from the dataset. We observed during our experimentation that the redundancy can create adverse effects on the classification process. After removing the redundant permission, we obtained 88 unique permissions.

Detail of the python script for the feature vector

```
import pandas as pd

import os

import numpy as np
if os.path.exists("feature_name.csv"):
    os.remove("feature_name.csv")
if os.path.exists("feature_set.csv"):
    os.remove("feature_set.csv")

import glob
for name in glob.glob('Combined/*.csv'):

        print "File in progress:", name

        datafile=pd.read_csv(name)
        a1=datafile.iloc[:, 0]
        with open("feature_name.csv", 'a') as f:
                a1.to_csv(f, header=False,index=False)


datafile=pd.read_csv("feature_name.csv")
a1_d=datafile.iloc[:, 0]


data=a1_d.unique()
import pandas as pd
df_1 = pd.DataFrame(data)
df_1.to_csv("feature_name_unique.csv")


line=""

for name in glob.glob('Malware/*.csv'):
        myf=open("feature_set.csv","a")

        print "File in progress:", name

        datafile=pd.read_csv(name)
        a1=datafile.iloc[:, 0]
        myf.write("1")
        myf.write("\n")
        for d in range(0,len(data)):
                -
                datalog=0
                for d1 in range(0,len(a1)):
                        if data[d] == a1[d1]:
                                datalog=1
                line+=str(datalog)+","
                myf.write(line)


                line=""
        myf.close()


for name in glob.glob('Non_Malware/*.csv'):
        myf=open("feature_set.csv","a")

        print "File in progress:", name

        datafile=pd.read_csv(name)
        a1=datafile.iloc[:, 0]
        myf.write("0")
        myf.write("\n")
        for d in range(0,len(data)):

                datalog=0
                for d1 in range(0,len(a1)):
                        if data[d] == a1[d1]:
                                datalog=1
                line+=str(datalog)+","
                myf.write(line)


                line=""
        myf.close()

print "All Features:",len(a1_d)
print "Unique Features:",len(a1_d.unique())
#if os.path.exists("Output.csv"):
 #   os.remove("Output.csv")
```

FIGURE 3.6: Python script for the feature vector

The same procedure used for the feature vector of intents we obtained 1745 intents vector after removing redundancy we obtained 105 unique intents.

3.3.6 Feature Selection by Using Infogain

After applying feature selection method, we selected top 10, 20, 30 and 40 permissions, Intents and API Calls that play the key role in malware identification. The focus of the feature selection method is to select those attributes of the dataset which are most appropriate and helpful in the employed machine learning model. For this purpose, we used the Information gain (IG) is a feature ranking method based on decision trees that exhibit good classification performance (M. T. Martin-Valdivia 2008). Information gain used in feature selection constitutes a filter approach. The thought behind IG is to choose features that uncover the most data about the classes. Ideally, such features are highly discriminative and occur in a single class (R. Mukras 2007).

Information gain is a measure based on entropy; it indicates to what extent the whole entropy is reduced if we know the value of a specific attribute. Therefore, the IG value indicates how much information this attribute contributes to the data set (M. T. Martin-Valdivia 2008). Each feature has its own IG value which determines whether this feature is to be selected or not. A threshold value is used for checking the features; if a feature has a greater IG value than the threshold, the feature is chosen; otherwise, it is not selected (Cheng-Huei Yang et al 2009). In our research information gain finds the certain patterns of the permissions and intents appearing in the Android application and assign weights to the information (to emphasize the effectiveness of the features).

List of top 10 permissions and their score that play the vital role for identification of malware Table 3.3

Table 3.3: List of Top Ten Permissions and their Score

Permissions	**Score**
WRITE_CONTACTS	0.173103
READ_PHONE_STATE	0.128233
CHANGE_WIFI_STATE	0.058602
RECEIVE_SMS	0.058242
CALL_PHONE	0.047947
ACCESS_COARSE_LOCATION	0.042571
ACCESS_NETWORK_STATE	0.042478
WRITE_EXTERNAL_STORAGE	0.036509
RECEIVE_BOOT_COMPLETED	0.034439
SET_WALLPAPER	0.03169

List of top 10 intents and their score that play the vital role for identification of malware Table 3.4

Table 3.4: List of Top Ten Intents and their Score

Intents	Score
BATTERY_CHANGED_ACTION	0.253591
BOOT_COMPLETED	0.097431
SIG_STR	0.094266
PHONE_STATE	0.08986
SEARCH	0.078222
APPWIDGET_UPDATE	0.061465
NEW_OUTGOING_CALL	0.05858
SHORTCUT_TOGGLE	0.055685
Default	0.042682
CREATE_SHORTCUT	0.024502

List of top 10 API calls and their score that play the vital role for identification of malware Table 3.5

Table 3.5: List of Top 10 API Calls and their Score

API Calls	Score
Landroid/os/Handler;->sendMessage (Landroid/os/Message;)Z	0.064279
Landroid/os/Handler;->obtainMessage () Landroid/os/Message;	0.064004
Lcom/keji/util/pf;-><init> () V [access_flags=public constructor] @ 0x201c8	0.050851
Lcom/keji/util/pf;->a([B)Ljava/lang/String; [access_flags=public static] @ 0x120fc	0.041647
MAIN	0.039235
Lcom/keji/util/pc;->a (Ljava/lang/String;)Ljava/lang/String; [access_flags=public static] @ 0x11c9c	0.038521
Lcom/energysource/szj/embeded/PermissionJudge$1;-><init> (Lcom/energysource/szj/embeded/PermissionJudge;)V [access_flags=constructor] @ 0xd958	0.032044
Landroid/app/Activity;->getPackageName () Ljava/lang/String;	0.027106
Landroid/widget/ImageView;->setImageResource(I)V	0.026679
Landroid/widget/TextView;->setText (Ljava/lang/CharSequence;)V	0.026489

3.3.7 Feature Selection by using PCA (Principal Component Analysis)

PCA is the dimensionality reduction algorithm. The main use of the PCA is to reduce the size of the feature space while retaining as much of the information as possible. PCA combines similar attributes and creates new ones. We, for the most part, have information with a substantial number of variables, some of which may be corresponded. This relationship between factors achieves an excess in the data that can be accumulated by the informational collection. In this way, keeping in mind the end goal to decrease the computational and cost complexities, we use PCA to transform the original variables to the linear combination of these variables which are independent. PCA is the statistical technique that linearly transforms an original set of variables into a substantially smaller set of uncorrelated variables that represent most of the information in the original set of variables. Its goal is to reduce the dimensionality of the original data set. A small set of uncorrelated variables is much easier to understand and use in further analysis than a large set of correlated variables (George h. Dunteman 1989).

3.4 Selection of Classifier for Training Phases

I have used Weka tool for training of the classifiers. Weka (Waikato Environment for Knowledge Analysis) is a popular suite of machine learning software written in Java, developed at the University of Waikato, New Zealand. The Weka suite contains a collection of visualization tools and algorithms for data analysis and predictive modeling, together with graphical user interfaces for easy access to this functionality. There are various advantages of Weka:

- It is freely available under the GNU General Public License
- It is portable since it is fully implemented in the Java programming language and thus runs on almost any architecture

- It is a huge collection of data preprocessing and modeling techniques
- It is easy to use due to its graphical user interface

All techniques of Weka's software are predicated on the assumption that the data is available as a single flat file or relation, where each data point is described by a fixed number of attributes (normally, numeric or nominal attributes, but some other attribute types are also supported) (Rohit Arora et al 2012).

All features (permission, intents, and API calls) that we are selected are used to train machine learning classifiers by using Weka tool. Selection of the correct classifier for the training phase is the most critical step of our research. In our research we have used supervised learning. Supervised learning is based on labeled data. In this case, we have an initial dataset, where data samples are mapped to the correct outcome. Examples of supervised learning are regression and classification problems. Usually we can say that if the output is a real number and continuous, then it is regression problem. We can say that if the output is categorical variables, then it is classification problem. For classification problems we can use following classification algorithms. we select four classifiers, Naïve bays (Jones, 1997), Random Forest (Breiman, 2001), Gradient boosting (Jason Brownlee, 2016) and Decision tree (Dobra A, 2009) for training phase.

3.4.1 Naive Bayes

Naive Bayes(Jones, 1997)is a simple classifier that uses the Bayes formula to match the labels with their corresponding dataset labels. The supposition behind this calculation states that the features do not depend on each other statistically for their targeted labels. Several experiments have confirmed that naive bays can be trained on a small amount of data. The adjustment of its parameter is simple, and the runtime performance is better compared to the other classifiers.

$$P(A/B) = P(B/A)\ P(A)/P(B) \qquad .... \qquad [1]$$

In equation [1] A represent malware class and B represent non-malware class.

3.4.2 Random Forest

Random Forest (Breiman, 2001) is an ensemble learning algorithm that was developed to overcome the over-fitting issue of the decision tree. For the training purpose, this technique is used to train several trees instead of training a single decision tree. As input, this algorithm takes the random subsets of the features.

info gain = -(pos/total) *log2(pos/5) - (neg/total) *log2(neg/total) .... [2]

In equation [2] positive value is malware and negative value is non-malware.

3.4.3 Gradient Boosting

Gradient boosting (Jason Brownlee, 2016) is a champion among the most intense systems for building order models. The idea is to combine weak leaner in such a way that overall model accuracy is optimal. The model is said to be weak if it performs better than slightly better than random chance. The output of different weak learner is combined in such a way that its loss function should be optimized. The main objective is that each model loss should be minimized by adding weak learner using procedures like gradient descent. When one weak leaner is added in the model, the other left unchanged. Gradient boosting involves three elements:

1. A loss function to be optimized.
2. A weak learner to make predictions.
3. An additive model to add weak learners to minimize the loss function.

3.4.4 Decision Tree

Decision tree (Dobra A, 2009) classifiers used binary tree for classification. As any other classifier, the decision tree classifiers use values of attributes/features of the data to make a class label (discrete) prediction. Structurally, decision tree classifiers are organized like a decision tree in which simple conditions on (usually single) attributes label the edge between an intermediate node and its children. Leaves are labeled by class label predictions. Decision tree also uses entropy formula.

3.5 Evaluation Measurements

Accuracy is the fraction of the total number of correctly classified apps as malware or benign.

Accuracy= TP+TN/Total

Precision The fraction of the actual correctly classified apps to the total predicted observations in all apps.

Precision=TP/TP+FP

The recall is the fraction of the correctly predicted positive applications to the total number of the apps that are in actual class.

Recall=TP/TP+FN

F Measure The harmonic means of precision and recall. F measure represents the value that tells how much the model is capable of making fine distinctions.

$$FMeasure = 2 \times Recall \times Precision/Recall + Precision$$

Confusion Matrix indicates the number of correctly and incorrectly classified apps. For our dataset, there were two types of data i.e.; malware and benign. The confusion matrix shows the results in the following terms:

True Positive (TP) The count of the malware applications that are correctly classified as malware.

False Negative (FN) The count of the malware applications that are incorrectly classified as benign.

True Negative (TN) The count of the benign applications that are correctly classified as non-malware (benign).

False Positive (FP) The count of the benign applications that are incorrectly classified as malware.

Chapter 4

Results and Discussion

In this chapter, we evaluate static feature using machine learning techniques for the purpose of Android malware detection. On the basis of experiments, we have collected best features which are most effective for a better classification. We have analyzed the performance of different features by using feature selection algorithms (info gain and PCA) and concluded the best set of features based on the attained results against each classifier. Results present the performance of four different machine learning classifiers (Random Forest, Naive Bayes, Decision Tree, Gradient Boosting) of Android malware detection. We have applied all four classifiers (discussed in the previous chapter) to our dataset of malware and non-malware applications and evaluate the performance of all these classifiers on the intents, permissions and API calls separately. Finally, these classifiers are applied on the combination of all these feature sets to analyze the best performing features by using feature selection algorithms also evaluated the effectiveness of the feature selection algorithms and best performing combination of classification and feature selection algorithms. The following sections describe the detail of our experiments.

4.1 Experimental Setup

The experimental setup is shown in table 4.1

TABLE 4.1: Experimental Setup

1	CPU	Intel® Core (TM) i7-6500 CPU, 2.50 GHz
2	Installed Memory	12 GB
3	O.S	Ubuntu 16.04
4	APK Tool	2.2.1
5	Python	2.2.13
6	Machine Learning Classifier	-Random Forest -Naive Bayes -Decision Tree -Gradient Boosting

4.2 Dataset

For this study, our dataset consists of applications that are in the form of .apk files. The benign dataset is collected randomly from the Google play store. The benign dataset consists of 500 applications that belong to 28 different categories. We acquired Drebin dataset that consists of thousands of Malwares based on several families. To do research on Android malware and to enable a comparison of different detection approaches, Drebin dataset is free available. The dataset contains 5,560 applications from 179 different malware families. The samples have been collected in the period from August 2010 to October 2012 (Arp et al., 2014). We have 500 malware and 500 clean applications. For the training of our dataset, we use 70% of our total data that consist of 350 applications and 30% dataset used for testing that consist of 150 applications.

4.3 Feature Selection

The aim of this step is to reduce the high-dimension of feature instances in our collected dataset by introducing subsets of features. The most suitable features (obtained using info gain and PCA method) are then selected as suitable features which are most helpful in identifying application class (malware or benign). Feature selection minimizes the time required for training/ testing and increases the

accuracy to generate simple interpreted models. Before performing feature selection, we have cleaned our datasets by omitting redundant features. Usually, the decision of filtering out or keeping a certain set of features depends on the platform which provides that feature. Therefore, during the feature selection process, we have paid more attention to the features provided by the Android platform. To serve this purpose, we have used info gain (discussed in Section 3.2.4) to get the most appropriate feature's list having a significant role in the classification. Total 189 android application features (based on different permissions and intents) were selected out of total 1700 produced by the static analysis. These features consist of permissions and intents collectively. Similarly, 92 android API calls were selected out of total 700 produced by static analysis Figure 4.1 describes the feature selection of top ten features of permission (ranked by the info gain method) which play a key role in malware identification using static analysis. Features obtained as a result of info gain method consists of permissions. These features play a vital role in the classification process, as they retain the maximum information for the classification.

Permission Features WRITE_CONTACTS Permission has the highest rank among all other features. READ_PHONE_STATE permission is second highest. SET_WALLPAPER have the lowest rank among the top ten ranked features.

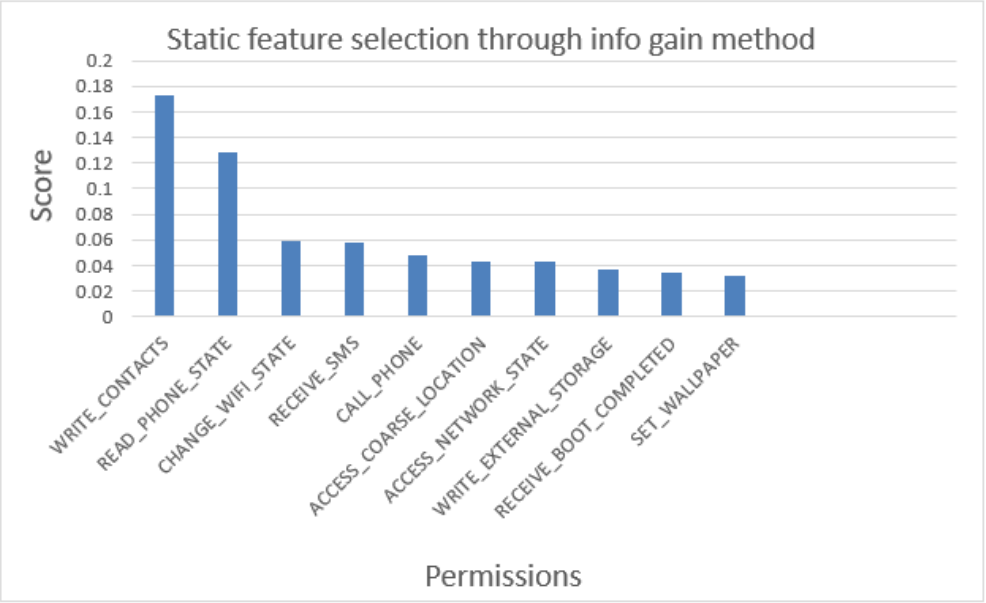


FIGURE 4.1: Rank Wise Feature Selection of Permission Using info Gain Method

Table 4.2 shows the brief description of the features obtained through the feature selection method. Each description tells about the basic functionality of the top-ranked feature.

TABLE 4.2: Feature Description of the Top-Ranked Static Feature of Permission

Rank	**Feature Name**	**Description**
1	WRITE_CONTACTS	Allows an application to write the user"s contacts data.
2	READ_PHONE_STATE	Allows read-only access to phone state, including the phone number of the device, current cellular network information, the status of any ongoing calls, and a list of any phone accounts registered on the device.
3	CHANGE_WIFI_STATE	Allows an application to change WI-FI connectivity state.
4	RECEIVE_SMS	Allows an application to receive an SMS.
5	CALL_PHONE	Allows an application to initiate a phone call without going through the dialer user interface for the user to confirm the call.

Rank	Feature Name	Description
6	ACCESS_COARSE_LOCATION	Allows an application to access approximate location.
7	ACCESS_NETWORK_STATE	Allows an application to access information about networks.
8	WRITE_EXTERNAL_STORAGE	Allows an application to write from external storage.
9	RECEIVE_BOOT_COMPLETED	Allows an application to receive the intent.action BOOT COMPLETED that is broadcast after the system finishes booting.
10	SET_WALLPAPER	Allows an application to set the wallpaper.

Intent Features Figure 4.2 indicates the role of top ten features of intents (ranked by the info gain method) which play a key role in malware identification using static analysis.BATTERY_CHANGED_ACTION intent has the highest rank among all other features.BOOT_COMPLETED is second highest. CREATE_SHORTCUT have the lowest rank among the top ten ranked features.

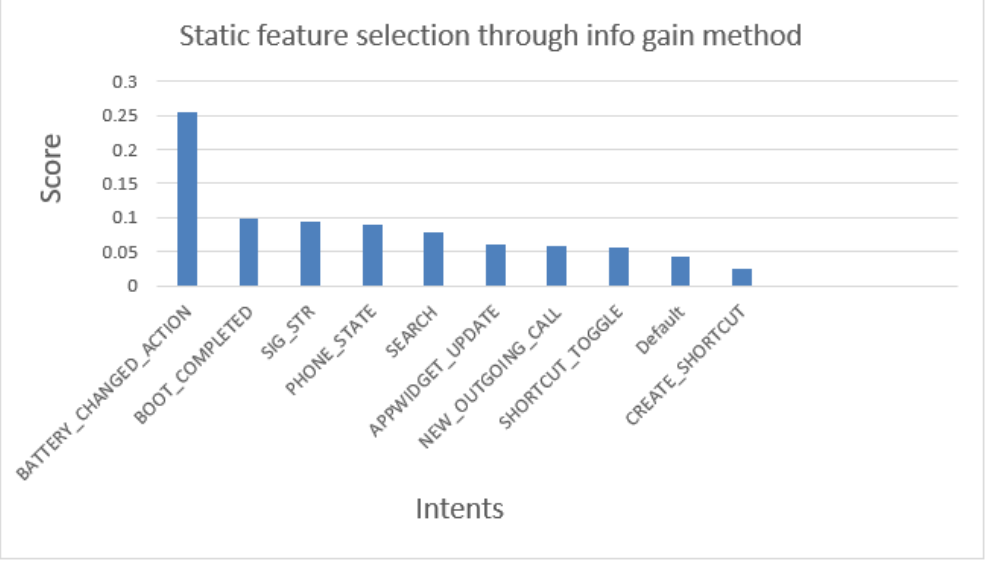


FIGURE 4.2: Rank Wise Feature Selection of Intents Using Info Gain Method

Table 4.3 shows the brief description of the features obtained through the feature selection method (info gain). Each description tells about the basic functionality of the top-ranked feature.

TABLE 4.3: Feature Description of Top ranked Static Feature of Intents

Rank	Feature Name	Description
1	BATTERY_CHANGED_ACTION	This is a sticky broadcast containing the charging state, level, and other information about the battery.
2	BOOT_COMPLETED	Broadcast Action: This is broadcast once after the user has finished booting. It can be used to perform application-specific initialization, such as installing alarms.
3	SIG_STR	Represents strings that can be assigned to other objects
4	PHONE_STATE	Allows read-only access to phone state, including the phone number of the device, current cellular network information, the status of any ongoing calls, and a list of any phone accounts registered on the device.
5	SEARCH	Activity Action: Perform a search.
6	APPWIDGET_UPDATE	Sent when it is time to update your AppWidget.
7	NEW_OUTGOING_CALL	When the user initiates a call, the system notifies interested apps by sending an ordered broadcast of the New outgoing call Intent, attaching the original phone number, URI, and other information as extras.
8	SHORTCUT_TOGGLE	Create shortcut
9	Default	This is the most common action performed on data – it is the generic action you can use on a piece of data to get the most reasonable thing to occur
10	CREATE_SHORTCUT	App creates shortcut

API Call Features Figure 4.3 indicates the role of the top ten features of API calls (ranked by the info gain method) which play a key role in malware identification using static analysis.sendMessage (Landroid/os/Message;API calls has the highest rank among all other features.obtainMessage () Landroid/os/Message;is second highest. setText (Ljava/lang/CharSequence;) have the lowest rank among the top ten ranked features.

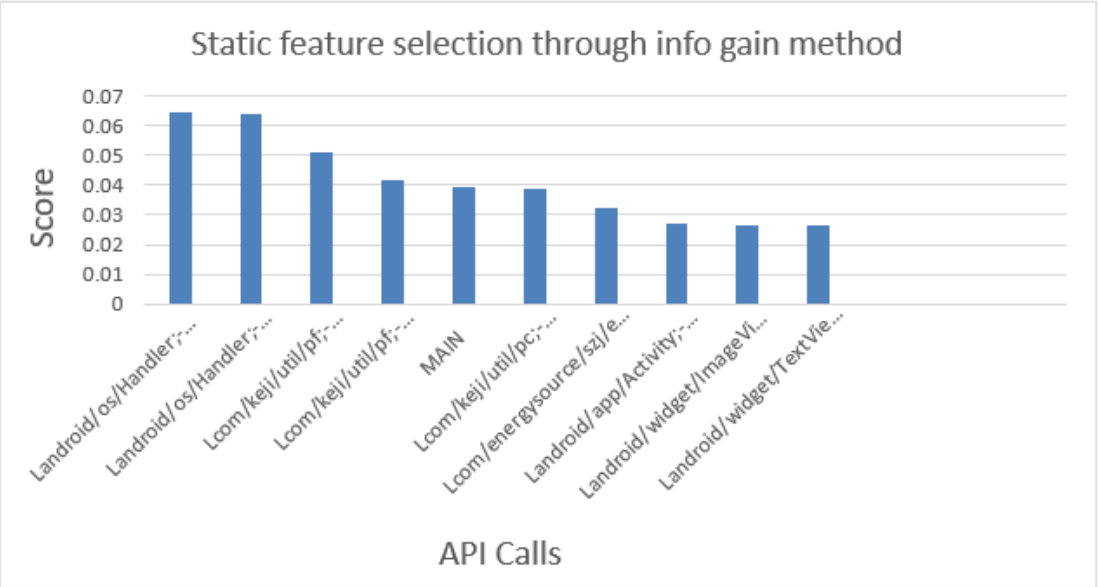


FIGURE 4.3: Rank Wise Feature Selection of API calls Using Info Gain Method

4.4 Classification Results with Feature Selection

4.4.1 Malware Detection of Top Ranked Feature (Info Gain)

R.Q 1 Which category of the static features are most effective for malware detection?

The variations in the f-measure achieved against used classifiers can be seen in Figure 4.4.

Figure 4.4 shows that Decision tree has the highest value of f-measure that is 93% Random Forest and Gradient Boosting has the second highest value off - the measure that is 92%. The Naive Bayes show the lower value of f-measure that is 88%.

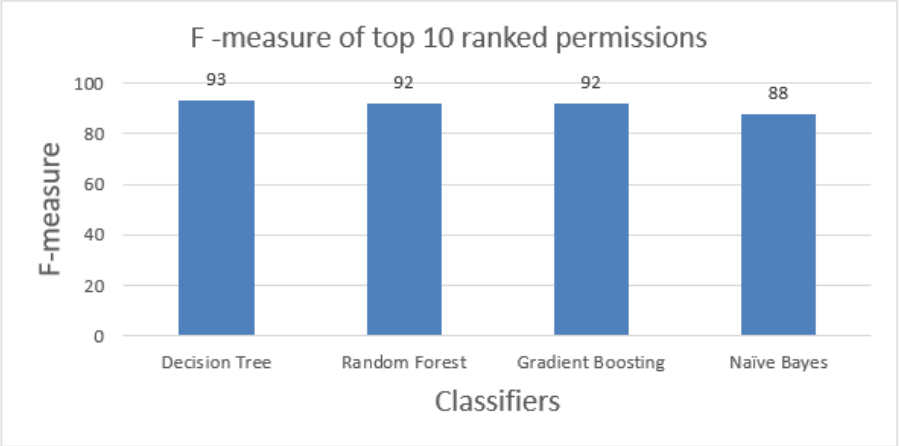


FIGURE 4.4: F- measure of Top 10 ranked permissions features by using four classifiers

Figure 4.5 shows that Decision tree and Gradient Boosting and Random Forest has the highest value of f measure that is 97%. The Naive Bayes show the lower value of f-measure that is 93%.

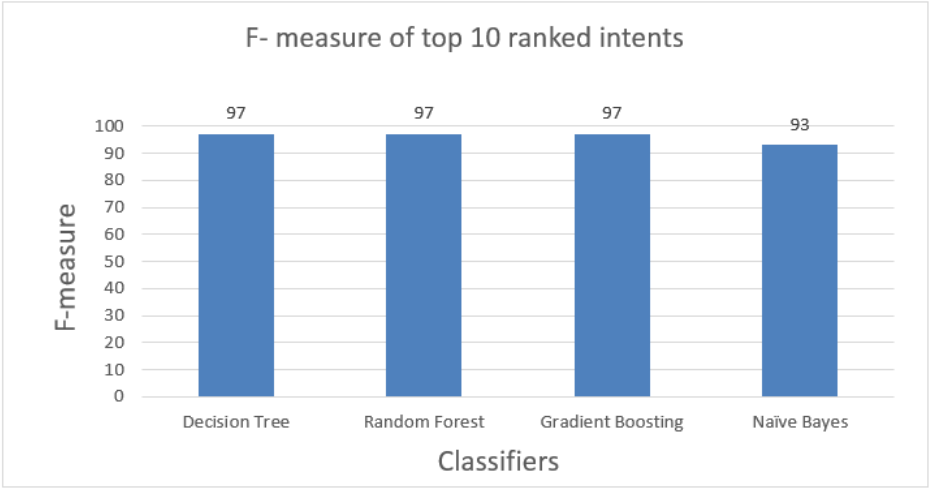


FIGURE 4.5: F-measure of Top 10 Ranked Intents Features by Using Four Classifiers

Figure 4.6 shows that the F-measure of the four classifiers using API calls are the same that is 84%.

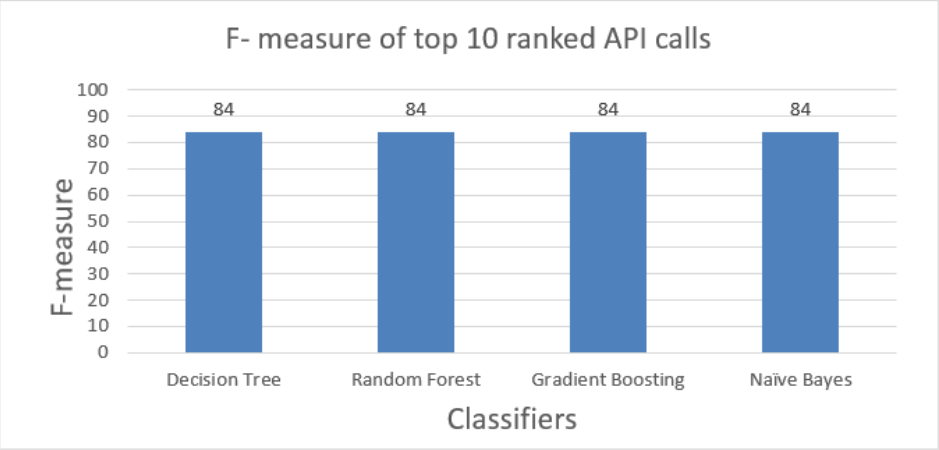


FIGURE 4.6: F measure of Top 10 ranked API calls features by using four classifiers

The results of the precision and recall of classification using different classifiers and info gain feature selection algorithm are given in Figure 4.7, figure 4.8, and Figure 4.9.

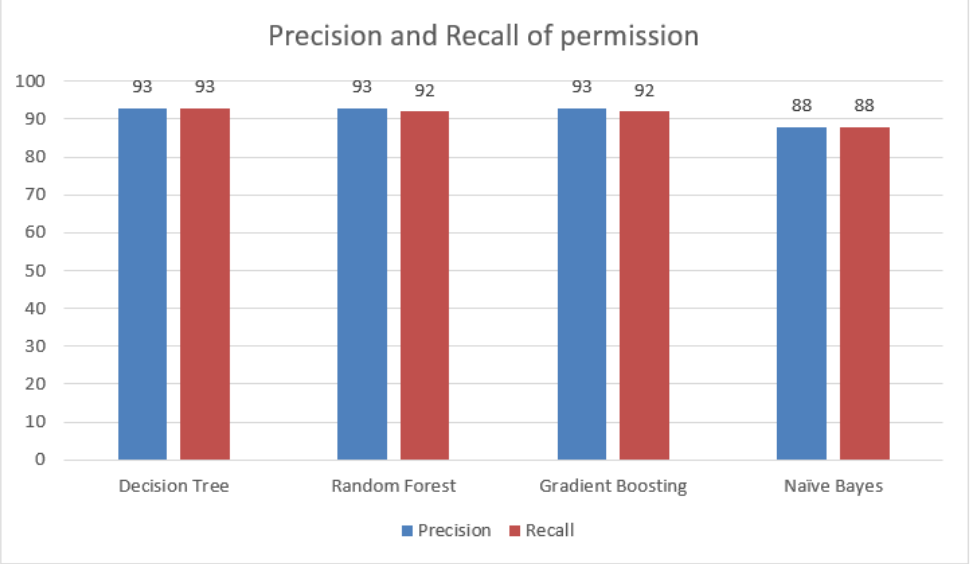


FIGURE 4.7: Precision and Recall for permission (info gain)

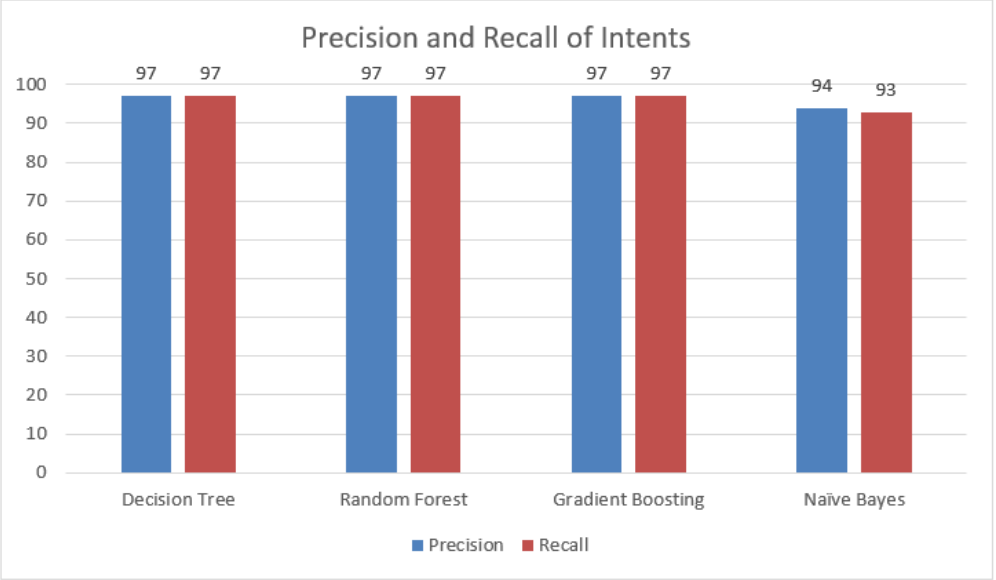


FIGURE 4.8: Precision and Recall for intents (info gain)

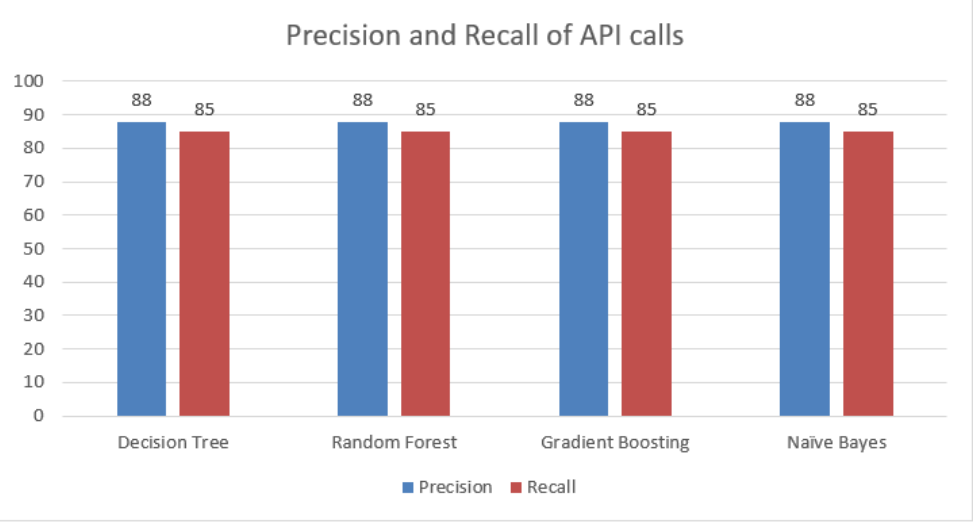


FIGURE 4.9: Precision and Recall for API calls (info gain)

(As shown in Figure 4.10 it is quite evident that the highest achieved accuracy is 97% when the classification is done for the intents analysis. Decision tree Random forest Gradient boosting classifiers behave same for the intent analysis Similarly, for the permission analysis, the classifier which achieved best results was the Decision Tree. Random forest and Gradient Boosting achieved second highest accuracy in case of permission. For the API calls analysis, all four classifiers performance

is the same but the malware detection rate is very low as compare to intents and permission.

We have concluded that overall performance of the Decision Tree classifier is good in case info gain feature selection algorithm. We have also concluded that intents are the great static feature for the detection of malware in case of info gain feature selection algorithm.

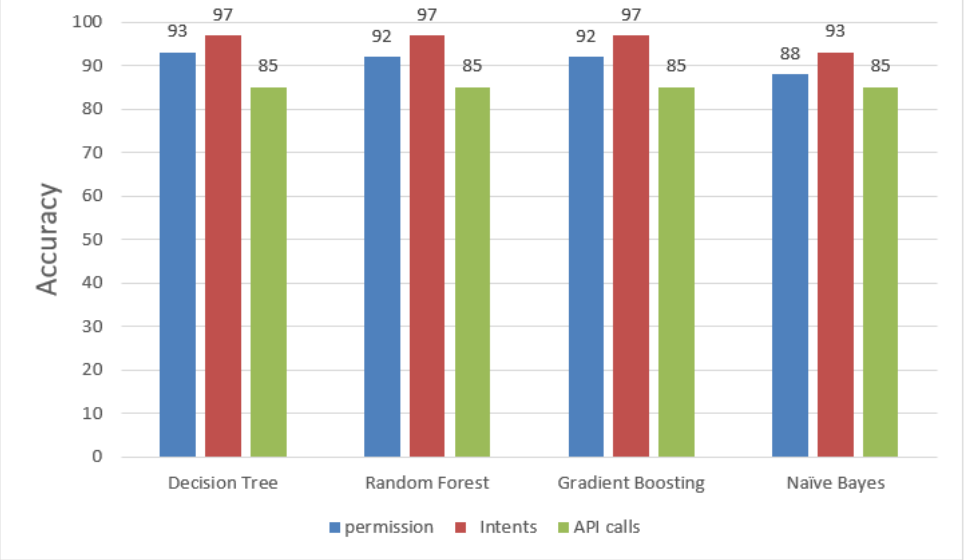


FIGURE 4.10: Accuracy of Permission, Intents and API Calls by Using Info Gain

4.4.2 Malware Detection of Top Ranked Feature (PCA)

In this section, we have discussed the results of selected feature sets (using PCA) which play a key role in malware identification.

Figure 4.11 shows that Random Forest has the highest value of f-measure that is 96% Decision Tree has the second highest value of fa measure that is 95%. The Naive Bayes and Gradient Boosting show the lower value of f-measure that is 94%.

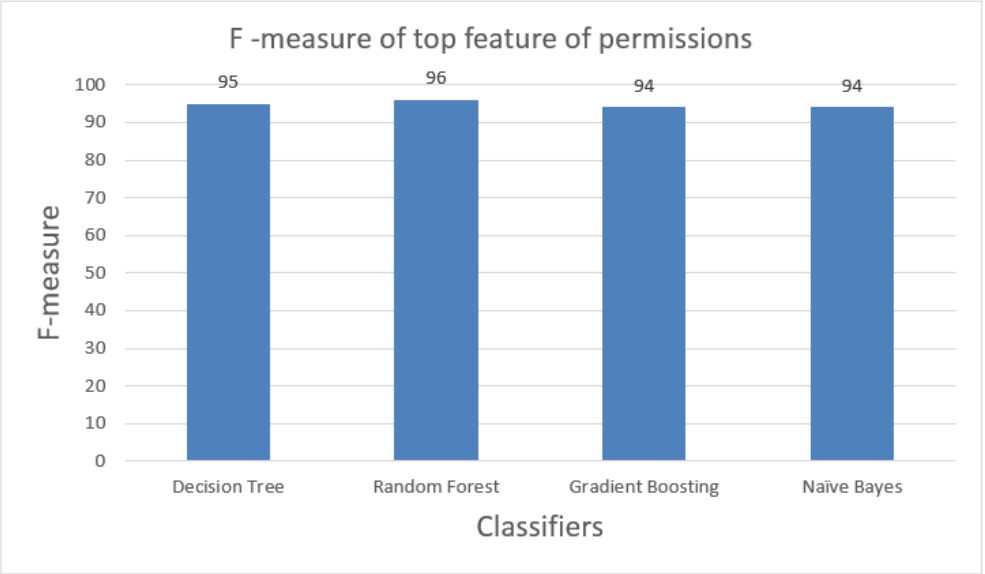


FIGURE 4.11: F- measure of Top Permissions Features by Using Four Classifiers

Figure 4.12 shows that Random forest and Gradient Boosting has the highest value off- the measure that is 93%. Decision Tree has the second highest value that is 92% The Naive Bayes show the lower value of f-measure that is 83%.

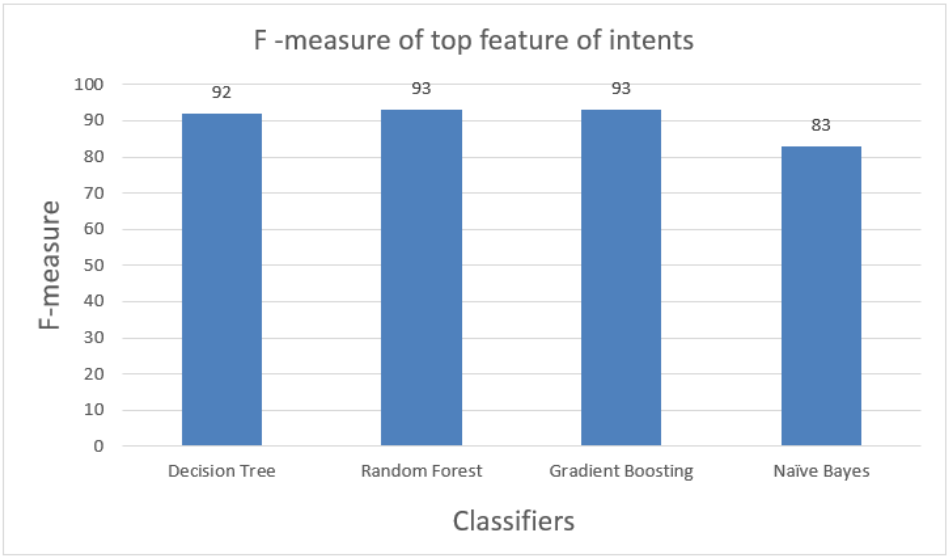


FIGURE 4.12: F-measure of Top Intents Features by Using Four Classifiers

Figure 4.13 shows that the F-measure of the four classifiers using API calls are the same that is 80%.

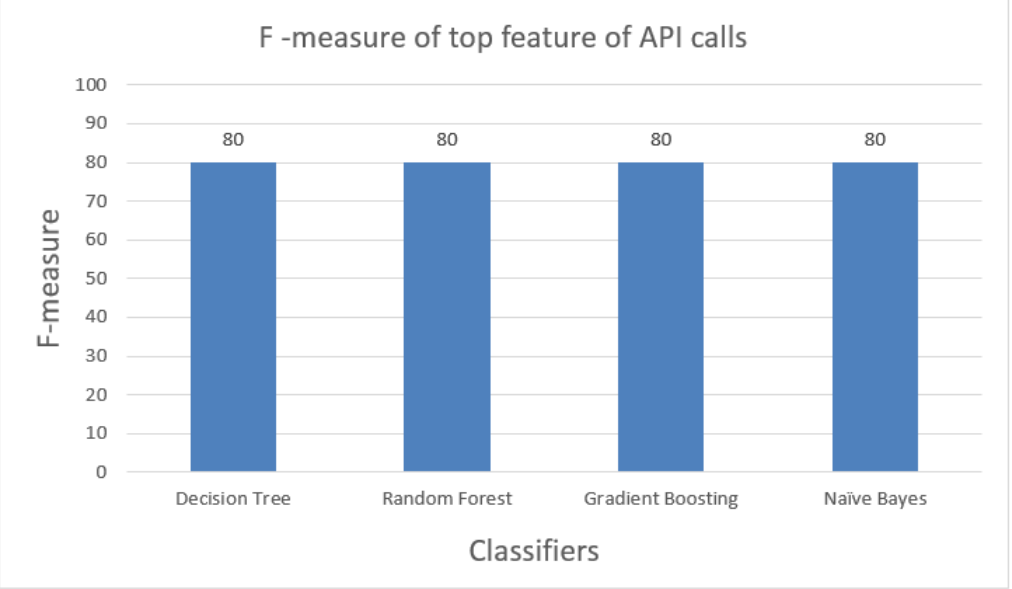


FIGURE 4.13: F- measure of Top API Calls Features by Using Four Classifiers

The results of the precision and recall of classification using different classifiers and PCA feature selection algorithm are given in Figure 4.14, figure 4.15, and Figure 4.16.

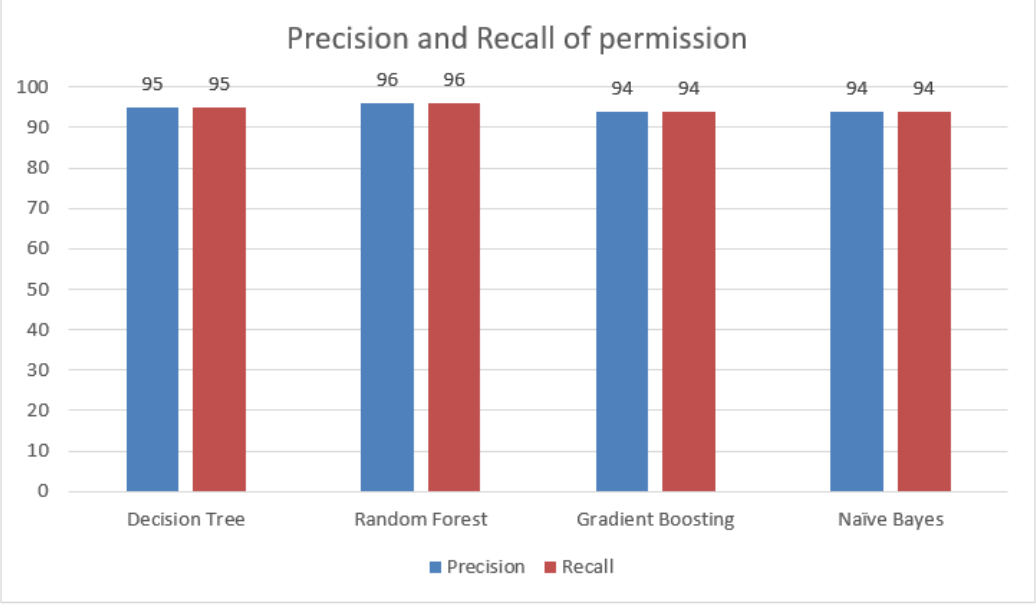


FIGURE 4.14: Precision and Recall for permission (PCA)

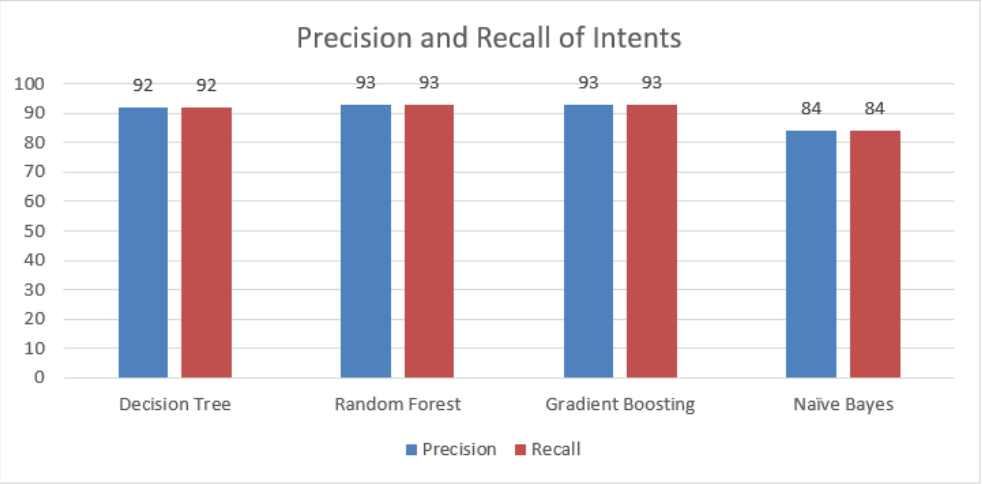


FIGURE 4.15: Precision and Recall for Intents (PCA)

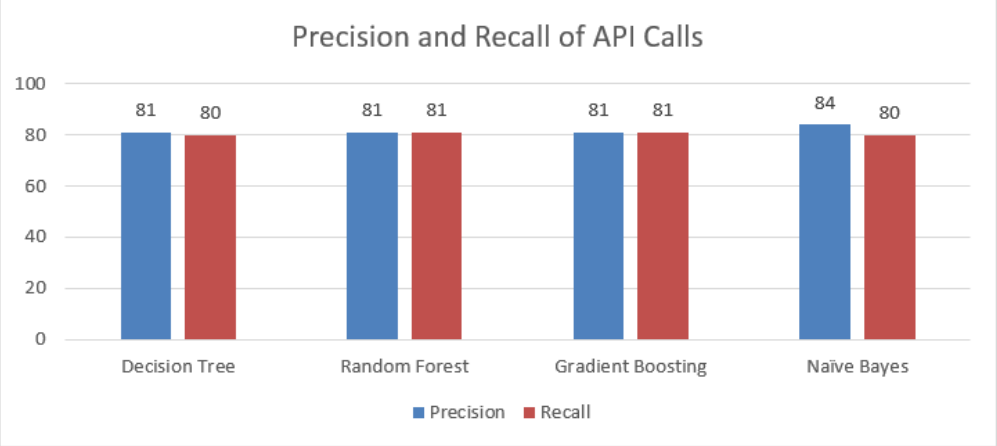


FIGURE 4.16: Precision and Recall for API Calls (PCA)

(As shown in Figure 4.17 it is quite evident that the highest achieved accuracy is 96% when the classification is done for the permission analysis by using Random Forest Classifier. We have concluded that overall performance of the Random Forest classifier is good in case of PCA feature selection algorithm. We have also concluded that permissions are the great static feature for the detection of malware in case of PCA feature selection algorithm.

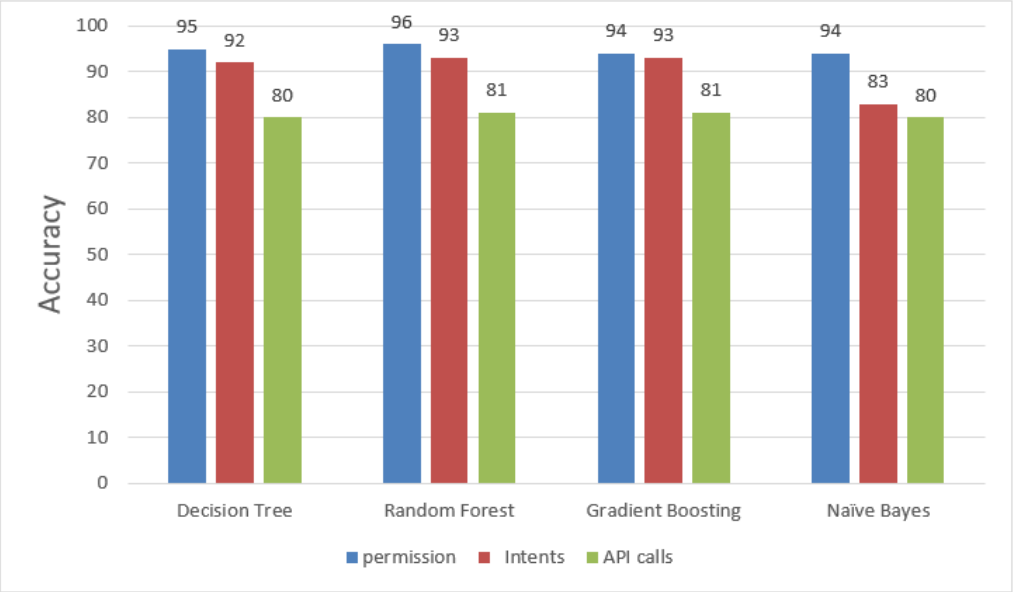


FIGURE 4.17: Accuracy of permission, intents and API calls by using PCA

4.4.3 Malware Detection using Combined Features (Info Gain)

R. Q2 Can accuracy be improved using a combination of these features (permission, intents, API calls)?

We trained the classifiers using intents, permission, and API calls features separately by using info gain and PCA feature reduction algorithm and compares which algorithm gives us best accuracy in case of all these (permission, intents, API calls) feature set. Now to assess classification accuracy, we have combined all these features in a single Boolean vector and compared the malware detection results by using info gain and PCA feature reduction algorithms. Furthermore, this feature vector is used to train a pure machine learning classifier that is trained on high-dimensional features to select a wide range malware in the Android applications.

Figure 4.18 shows that the f-measure and accuracy obtained using four classifiers (RF, DT, GB, and NB). The Decision Tree and Gradient Boosting have the

same accuracy i.e., 95%, while the Random Forest shows the 94% accuracy and f-measure. The Naïve Bayes accuracy is below 93%.

Figure 4.18 indicates that the Decision Tree and Gradient Boosting has a high accuracy rate and can be employed for combined features. The figure also indicates that the Decision tree classifier performs well with the info gain feature selection algorithm as it performs well with individual features.

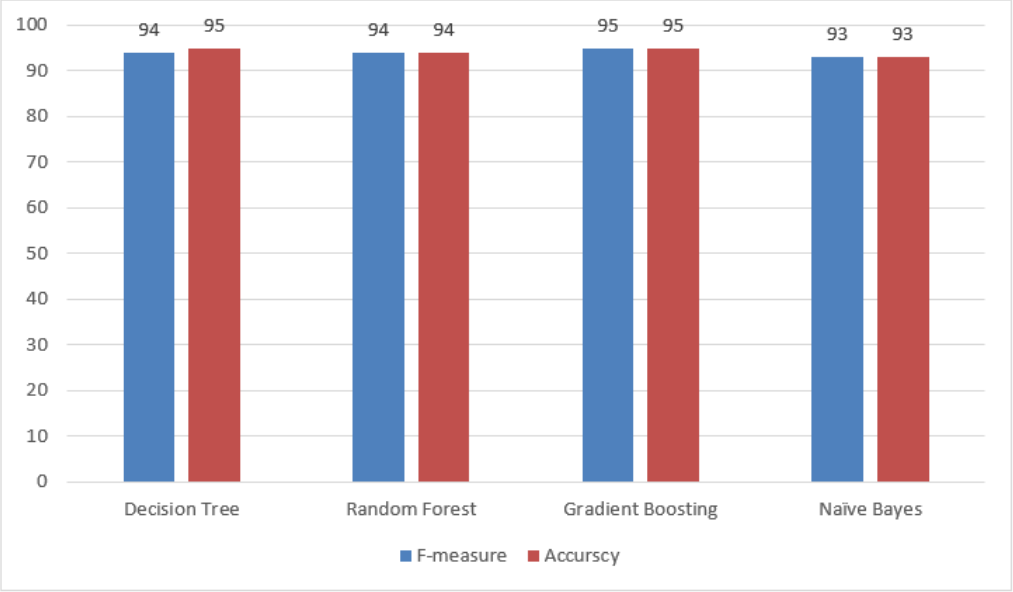


FIGURE 4.18: F-Measure and accuracy of combined features using info gain

Figure 4.19 shows that every increase in true-positive rate is accompanied by the decrease in false-positive rate. The Decision tree has the lowest false positive rate while the Naive Bayes shows the highest false positive rate. High false positive rate indicates that a classifier incorrectly classifies malware applications as benign applications.

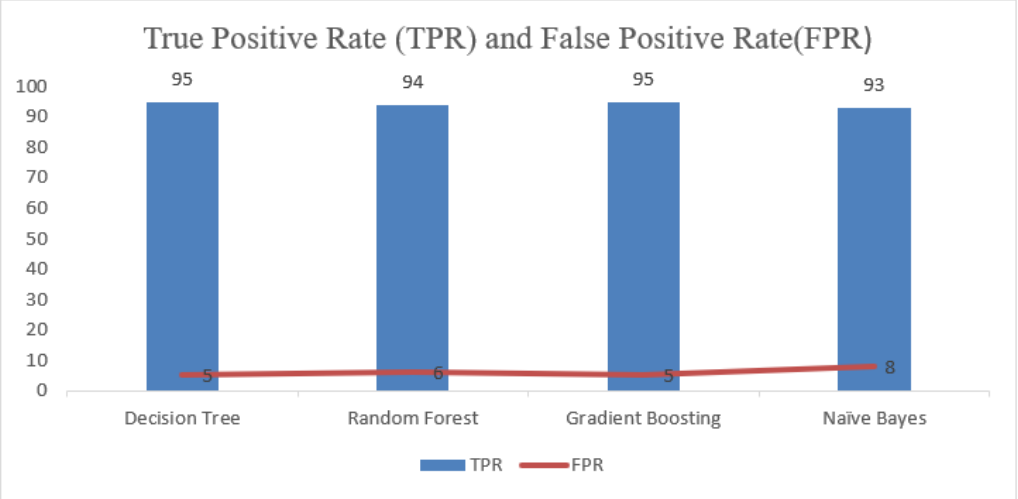


FIGURE 4.19: TPR and FPR of Combined Features Using Info Gain

4.4.4 Malware Detection using Combined Features (PCA)

Figure 4.20 shows that the f-measure and accuracy obtained using four classifiers (RF, DT, GB, and NB). The Gradient Boosting has the high accuracy i.e., 88%. The Naïve Bayes accuracy is low 75%.

Figure 4.20 indicates that the Decision Tree and Gradient Boosting has a high accuracy rate and can be employed for combined features.

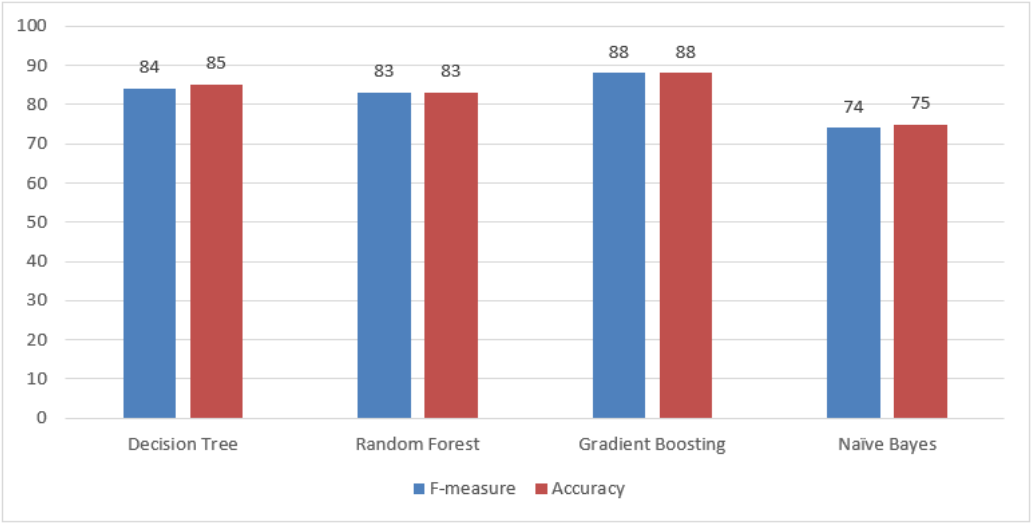


FIGURE 4.20: F-Measure and Accuracy of Combined Features Using PCA

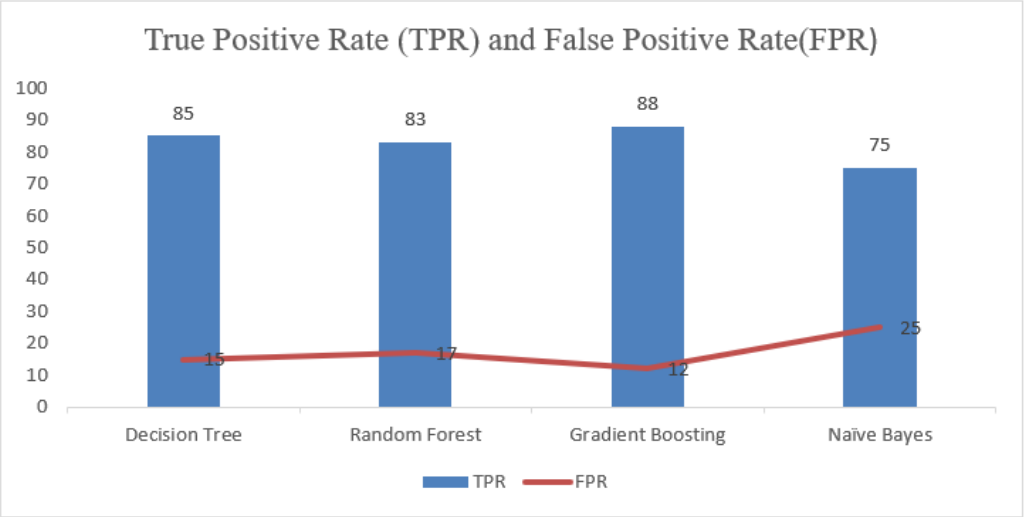


FIGURE 4.21: TPR and FPR of Combined Features Using PCA

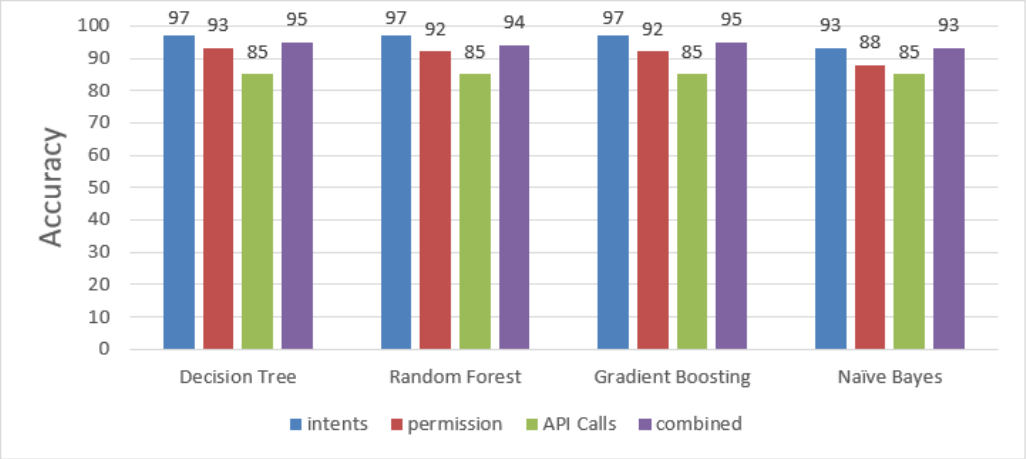


FIGURE 4.22: TPR and FPR of Combined Features Using PCA

Figure 4.22 shows that intents are the best feature set for detection of android malware with the highest accuracy rate of 97%. Combined feature set gives us the second highest performance of 95% by using the Decision tree.

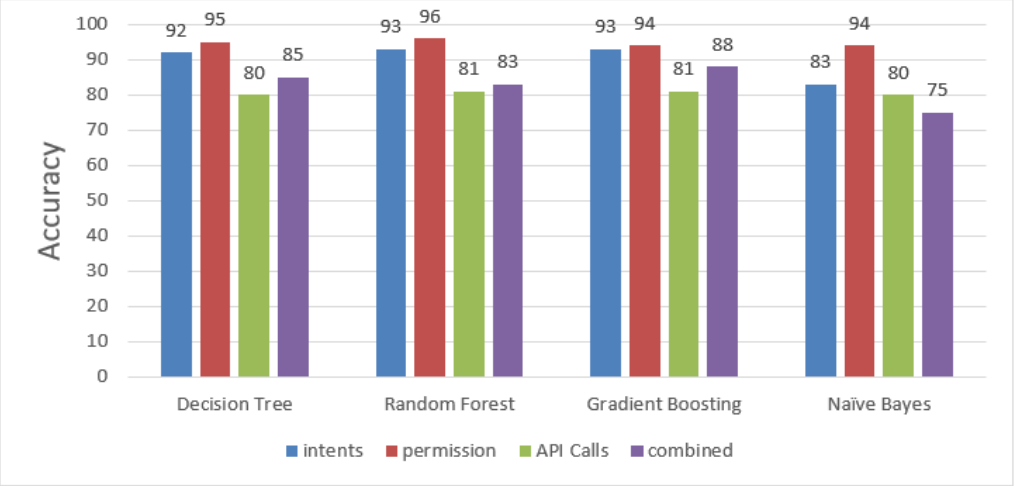


FIGURE 4.23: TPR and FPR of Combined Features Using PCA

Figure 4.23 shows that PCA feature reduction algorithm gives us best performance with the feature set permission by using the Random Forest classifier. Intent feature set gives us the second highest performance by using PCA. Combine feature set gives us third highest performance by using PCA. API Calls performance is very low for malware detection.

4.5 Classification Results without Feature Selection

The aim of this phase is to sort out an accurate classifier and feature selection algorithm that can discriminate between the malicious and the benign applications using feature patterns. We have considered four classification algorithms i.e. Naïve Bayes (Jones, 1997), Random Forest (Breiman, 2001), Gradient boosting (Jason Brownlee, 2016) and decision tree (Dobra A, 2009) and two feature selection algorithms info gain and PCA to be evaluated. First, the classification algorithms, Naïve Bayes, Decision tree, random forest, gradient boosting, were run on top 10 separate feature sets of intents, permission and API calls by using info gain feature

selection algorithm and then feature selection algorithms were run in combination with classification algorithms. Secondly, the classification algorithms were run on top selected separate feature sets of intents, permission and API calls by using PCA feature selection algorithm and then feature selection algorithms were run in combination with classification algorithms.

First, we have discussed results of all feature sets and then discussed results of selected feature sets (using info gain and PCA) which play a key role in malware identification using static analysis. The aim of using feature selection algorithm is to identify the best algorithm that takes minimum attributes and gives us best accuracy. We have compared both feature selection algorithms to identify the best algorithm.

4.5.1 Performance Evaluation of the Classifiers

R.Q 3 Which classifier (Gradient Boosting, Decision tree, Random forest. Naïve Bayes) performs well for the detection of malware?

Figure 4.24 shows the results of the performance of different classification algorithms in WEKA using the Accuracy values. Accuracy is the fraction of the total number of correctly classified applications as malware or benign.

The goal of this test is to compare the results of different classifiers with one other to check the selection of most appropriate (with higher accuracy) machine learning algorithms. From the results of this experiment, we can see that the Gradient Boosting classifier gives an Accuracy value 95% closer to the optimum value 1. The x-axis shows four classifiers and the Y-axis shows the accuracy value.

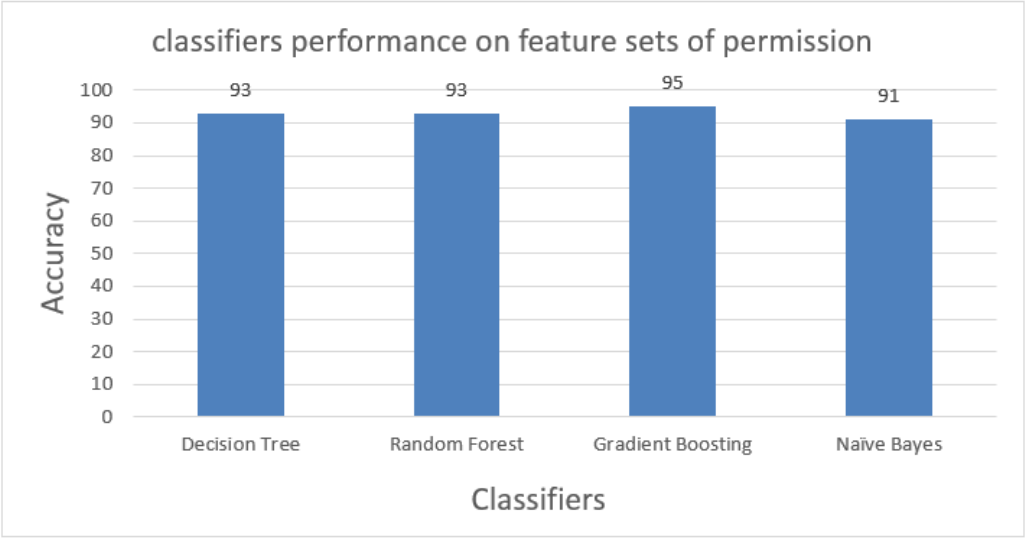


FIGURE 4.24: Performance of Four Classifiers on the Feature Set of Permission

Figure 4.25 shows the results of the performance of different classification algorithms by using static feature set of intents to check the performance of different classifiers on intents. From the results of this experiment, we can see that Decision Tree, Random Forest and Gradients Boosting algorithms gives an Accuracy value 97% closer to the optimum value 1.

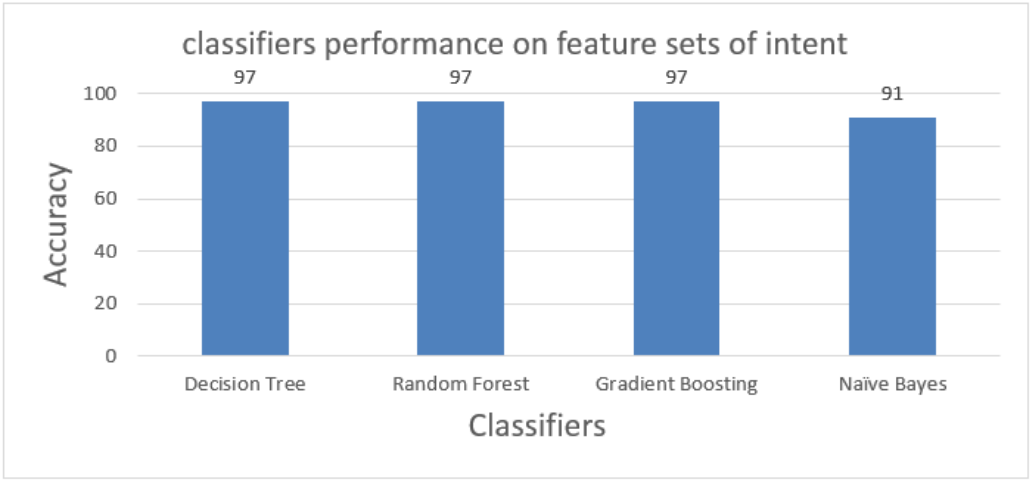


FIGURE 4.25: Performance of Four Classifiers on the Feature Set of Intents

Figure 4.26 shows the results of the performance of different classification algorithms by using static feature set of API calls to check the performance of different

classifiers on API calls. From the results of this experiment, we can see that Gradient Boosting algorithm gives an Accuracy value 90% closer to the optimum value 1. Decision Tree and Random Forest gives us the second highest value of 80% and the Naive Bayes classifier give us the lowest value that is 76%

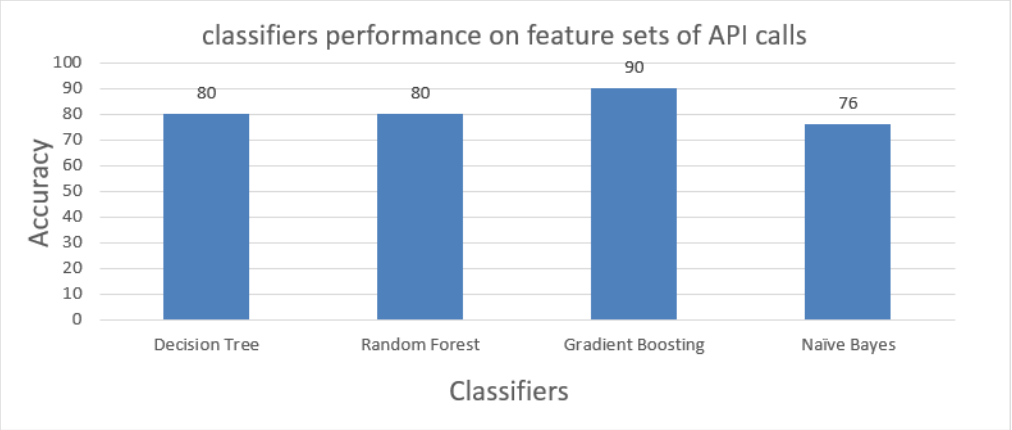


FIGURE 4.26: Performance of Four classifiers on the Feature set of API Calls

Figure 4.24, 425 and Figure 4.26 show the comparison of four classifiers for three types of the feature set of static data. The comparison results show that Gradient Boosting gives us best performance as compared to other classifiers Random Forest and decision tree performance is almost same in all types of feature sets and Naive Bayes give us lowest performance in all type of feature sets.

4.6 Evaluation

Ali Feizollah, et al. (2017) done their research on Android intents and permissions for malware identification. Authors show that intents are powerful features for the detection of malware in Android applications. They achieve malware detection rate of 91% using Android intent as a major feature and got a detection rate of 83% using Android permissions. We compared our results to the previous paper Ali Feizollah, et al. (2017) and achieved high accuracy rate for both Android intents and permissions as compare to pervious paper. Table 4.4 shows the top 10 features of permission used in previous paper and also shows top 10 feature which we have obtained in our research.

TABLE 4.4: Top 10 Feature List of Permission

Feature list of previous paper	**Feature list of our research**
INTERNET	WRITE_CONTACTS
WRITE_EXTERNAL_STORAGE	READ_PHONE_STATE
READ_PHONE_STATE	CHANGE_WIFI_STATE
SEND_SMS	RECEIVE_SMS
RECEIVE_SMS	CALL_PHONE
WAKE_LOCK	ACCESS_COARSE_LOCATION
READ_SMS	ACCESS_NETWORK_STATE
ACCESS_COARSE_LOCATION	WRITE_EXTERNAL_STORAGE
ACCESS_FINE_LOCATION	RECEIVE_BOOT_COMPLETED
READ_CONTACTS	SET_WALLPAPER

Table 4.5 shows the top 10 features of intents used in previous paper and also shows top 10 features which we have obtained in our research.

TABLE 4.5: Top 10 Feature List of Intents

Feature list of previous paper	**Feature list of our research**
BATTERY_CHANGED	BATTERY_CHANGED_ACTION
Edit_intent	BOOT_COMPLETED
BOOT_COMPLETED	SIG_STR
DIAL	PHONE_STATE
SCREEN_OFF	SEARCH
USER_PRESENT	APPWIDGET_UPDATE
PICK_FILE	NEW_OUTGOING_CALL
GET_CONTENT	SHORTCUT_TOGGLE
SEND	Default
MEDIA_MOUNTED	CREATE_SHORTCUT

4.6.1 Comparison

Table 4.6 shows the result comparison of permission with similar work (Ali Feizollah, et al. (2017).

TABLE 4.6: Result Comparison with Similar Work by Using Permission

classifiers	**(Ali Feizollah, et al. (2017).**	**Current Study**		
		Precision	**Recall**	**F-measure**
Decision Tree	90%	93%	93%	93%
Random Forest	89%	93%	92%	92%
Gradient Boosting	89%	92%	93%	92%
Naïve Bayes	85%	88%	88%	88%

Table 4.7 shows the result comparison of intents with similar work (Ali Feizollah, et al. (2017).

TABLE 4.7: Result Comparison with Similar Work by Using Intents

classifiers	(Ali Feizollah, et al. (2017).	Current Study		
		Precision	Recall	F-measure
Decision Tree	90%	97%	97%	97%
Random Forest	89%	97%	97%	97%
Gradient Boosting	89%	97%	97%	97%
Naïve Bayes	85%	94%	93%	93%

Table 4.6 and 4.7 shows the comparison of our work with previous work. The comparison results show that our results achieved high accuracy value that is 97% as compared to previous work (Ali Feizollah, et al. (2017). These results also show that our selected classifiers perform good as compare to the previous work.

The results presented in this chapter show that intents are the best feature set for malware detection as compare to permission and API calls. Android intents achieved high accuracy rate by using three classifiers (Decision tree, Random forest, Gradient Boosting) and info gain feature selection algorithm. We have seen that the Naïve Bayes classifier gives the very low performance as compared to other classifiers. Combined feature set gives us the second highest performance by using the Decision tree. Permission and API call feature set give us low performance. Overall performance of Decision Tree is good for android malware detection. Info gain is the best feature selection algorithm as compare to PCA.

Chapter 5

Conclusions

In Android malware detection, a prominent research gap is the absence of a study of machine learning based classifiers, which utilize all the features (Intents, Permissions and API Calls) to classify an android application as benign or malware.

In previous work, some author uses intents and permission some use API calls but there is a lack of research that uses the combination of intents, permissions and API Calls on same data set to analyze the malware applications. This research work presents the comparative performance of four different machine learning classifiers (Random Forest, Decision Tree, Gradient Boosting, Naïve Bayes) and two feature selection algorithms (info gain, PCA) of Android malware detection and compare them. Moreover, these classifiers are trained on the features which have not been coupled in the literature.

We performed Static analysis on manifest file to extract permissions and intents, generated the very informative feature vector. Further, this feature vector used to train classifiers for efficient malware detection. Similarly, we performed static analysis using API calls.

We identified the features that play important role in malware identification. We have done another experiment by combining the features set (permission, intent, API calls) obtained from the static analysis.

We draw the following conclusion from our experimentation and results:

- In a machine learning based malware detection system, selection of the feature set from the dataset is one of the most critical steps.
- Feature selection itself depends upon the analysis method through which they are extracted. It is the analysis technique that determines the compatibility of features with the classification algorithm.
- Feature selection algorithms also another important step in our research.

Our conclusion consists of two feature selection algorithms info gain and PCA. Experimental results explain the conclusions of this research. We get remarkable results regarding classifier training for malware identification. Trained classifier (Random Forest based) shows the accuracy value of 96% in case of permission and 93% in case of intents and 83% for the combined feature set and 81% for API Calls by using PCA algorithm. The overall performance of Random Forest is best as compare to other classifiers when we have used PCA and permission is the best feature set for malware detection. The TPR for this feature set is high up to 96% and the FPR is low as 4% that indicates the accurate performance of this classifier and feature set for malware detection identification.

We get remarkable results when we used info gain feature selection algorithm we get 97% accuracy rate in case of intents by using Decision tree we have absorbed that overall performance of Decision Tree is good in case of info gain feature selection algorithm. Decision tree shows the accuracy value of 97% in case of intents and 93% in case of permission and 95% for the combined feature set and 85% for API Calls by using info gain algorithm. We have concluded that info gain feature selection algorithm is best as compare to PCA. We have also concluded that intents are the best feature set for detection of android malware with the highest accuracy rate of 97% with the combination of info gain feature selection algorithm that we have not seen in literature. The individual feature set of API Calls are not good for Android malware detection.

The result of this experiment opens the door to another aspect of malware identification. Trained classifier on combined feature set identified the 95% malware accurately. The accuracy value of the best classifier is 97% that indicates the classifiers learn the malware patterns successfully.

5.1 Future Work

As a future work, we plan to validate our framework on the different data set (Malgenome). We can even train a classifier in the future which can detect the malware application and classifies them into families. Combination of machine learning algorithms can also be trained on the combined feature vector. we can also check the performance of other machine learning algorithm also use other feature selection algorithm to evaluate the performance of android malware detection. We can also use other feature selection algorithms to validate our results.

Bibliography

Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K. and Siemens, C.E.R.T., 2014, February. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. vol. 14, pp. 23-26.

Almin, S.B. and Chatterjee, M., 2015. A novel approach to detect android malware. Procedia Computer Science, vol. 45, pp.407-417.

Breiman, L., 2001. Random forests. Machine learning, vol. 45, pp.5-32.

Cheng-Huei Yang, Li-Yeh Chuang and Cheng-Hong Yang. (2009). A Hybrid Filter/Wrapper Method for Feature Selection of Microarray Data" Journal of Medical and Biological Engineering, vol.30, pp. 23-28.

Dobra A. (2009) Decision Tree Classification. springer pp. 532-538.

De Capitani di Vimercati and Fabio Martinelli. (2017) "ICT Systems Security and Privacy Protection". Published in 32nd IFIP TC 11 International Conference, pp.29-31,

ElMouatez Billah Karbab, Mourad Debbabi, Derhab and Mouhab. (2018) "MalDozer: The Automatic framework for Android malware detection using deep learning". DFRWS 2018 Europe d Proceedings of the Fifth Annual DFRWS Europe, vol 24, pp.23-26.

Faruki, P., Ganmoor, V., Laxmi, V., Gaur, M.S. and Bharmal, A., 2013, November. AndroSimilar: robust statistical feature signature for Android malware

detection. In Proceedings of the 6th International Conference on Security of Information and Networks pp. 152-159.

Feizollah, A., Anuar, N.B., Salleh, R., Suarez-Tangil, G. and Furnell, S., 2017. Androdialysis: Analysis of android intent effectiveness in malware detection. computers & security, vol. 65, pp.121-134.

George h. Dunteman. (1989) "Principal Components Analysis" published by SAGE Publications Ltd. pp.1009-1021.

Gunjan Kapse and Aruna Gupta. (2015) "Detection of Malware on Android based on Application Features". International Journal of Computer Science and Information Technologies, vol. 6, pp.3561-3564.

Idrees F and Rajarajan M. (2014) "Investigating the android intents and permissions for malware detection". Wireless and Mobile Computing, Networking and Communications (WiMob), IEEE 10th International Conference on. IEEE, pp. 354-358.

Jones KS. (1997) Readings in information retrieval: Morgan Kaufmann, pp.456-459.

Jason Brownlee, (2016) "A Gentle Introduction to the Gradient Boosting Algorithm for Machine Learning". vol. 9, pp.556-560.

Jung, H.M., Kim, K.B. and Cho, H.J., 2016. A study of android malware detection techniques in virtual environment. Cluster Computing, volume number 19, pp.2295-2304.

Kim J, Yoon Y and Kwangkeun Yi. (2012) "ScanDal: The Static analyzer for detecting privacy leaks in android applications". vol. 12 pp.1-110.

Losses, N., 2014. Estimating the global cost of cybercrime. McAfee, Centre for Strategic & International Studies.

Lifan Xu, Dongping Zhang, Nuwan Jayasena and John cavazos. (2016) "HADM Hybrid Analysis for Detection of Malware". Proceedings of SAI intelligent system conference. pp. 702-724.

Mehadi Hassan and Philip K. Chan. (2017) "Scalable Function Call Graph-based Malware Classification". Proceedings of the Seventh ACM on Conference. pp. 239-248.

Martín-Valdivia, M.T., Díaz-Galiano, M.C., Montejo-Raez, A. 2008. Using information gain to improve multi-modal information retrieval systems. Information Processing & Management, vol. 44, pp.1146-1158.

R. Mukras, R., Wiratunga, N., Lothian, R., Chakraborti, S. and Harper, D., 2007. Information gain feature selection for ordinal text classification using probability re-distribution. In Proceedings of the Textlink workshop at IJCAI vol. 7, p. 1-16.

Rohit Arora and Suman. (2012) "Comparative Analysis of Classification Algorithms on Different Datasets using WEKA" International Journal of Computer Applications vol.54, pp. 1-13.

Sato R, Chiba D and Goto S. (2013) "Detecting Android malware by analyzing manifest files". Proceedings of the Asia-Pacific Advanced Network vol. 36, p. 23-31.

Singla S, Gandotra E, Bansal D, and Sanjeev Sofat. (2015) "A Novel Approach to Malware Detection using Static Classification". International Journal of Computer Science and Information Security vol. 13, pp. 1-3.

Saidah Mastura, Mohd Faizal Abdollah, Robiah Yusof and Mohid Zaki. (2015) " Recognizing API Features for Malware Detection Using Static Analysis". Journal of Wireless Networking and Communications vol 5, pp. 6-12.

Saba Arshad, Abid Khan, Munam Ali Shah, Mansoor Ahmad. (2016) "Android Malware Detection & Protection: A Survey" IJACSA) International Journal of Advanced Computer Science and Applications, vol. 7, pp. 1-2.

Tarang Kumar Barsiya, Manasi and Rajesh Wadhwani. (2016) "Android malware analysis: A survey paper". International Journal of Control, Automation, Communication and Systems (IJCACS), vol.1, pp.1-1.

Vidhya Rao, K Hande and J. Eng.Dev. (2017) "A comparative study of static, dynamic and hybrid analysis techniques for Android malware detection. vol.,5, pp.1433-1436.

Weka: http://www.cs.waikato.ac.nz/ml/weka/ 4 April, 2018

Zheng M, Sun M and Lui JC. (2013) "Droid analytics: a signature based analytic system to collect, extract, analyze and associate android malware". Trust, Security and Privacy in Computing and Communications (TrustCom), 12th IEEE International Conference IEEE, pp. 163-171.

Zhenxiang Chen, Hongbo Han, Qiben Yan, B. Yang, L. Pang and L. Zang. (2015) "A First Look at Android Malware Traffic in First Few Minutes". Published in conference Trustcom/BigDataSE/ISPA, vol. 1, pp. 206-213.

Appendix A

Feature extraction code for APK files

```
from XML.dom.minidom import parseString
import os
import sys

import glob
for a in glob.glob("*.apk"):
print "apktool working"
try:
    x="apktool d -f "+str(a)
    os.system(x)
    b=a[0:len(a)-4]


    if os.path.isfile(str(b)+"_intent.csv"):
        os.remove(str(b)+"_intent.csv")
        os.remove(str(b)+"_permission.csv")

    with open(str(b)+"/AndroidManifest.xml","r") as f:
        data = f.read()

    dom = parseString(data)
    nodes = dom.getElementsByTagName("intent-filter")
    intents=open("i/"+str(b)+"_intent.txt","a")
    permission=open("p/"+str(b)+"_permission.txt","a")
    intents.write("intent_name,Vector,type")
    intents.write("\n")
    print "Extracting Intents"
    for node in nodes:
        y1=str(node.toxml()).find("""""")
        y2=str(node.toxml()).find("""""",y1+1)
        intents.write(str(node.toxml())[y1+1:y2])
        intents.write(",1")
        channels =node.getElementsByTagName("data")
        if channels ==[]:
            intents.write(",explicit")
        else:
            intents.write(",implicit")
```

```
            intents.write("\n")
        intents.close()
        print "Extracting permission"
        permission.write("permission_name,Vector")
        permission.write("\n")
        nodes = dom.getElementsByTagName("uses-permission")
        for node in nodes:
            y1=str(node.toxml()).find("""""")
            y2=str(node.toxml()).find("""""",y1+1)
            permission.write(str(node.toxml())[y1+1:y2])
            permission.write(",1")
            permission.write("\n")
        permission.close()
        os.rename("i/"+str(b)+"_intent.txt","i/"+str(b)+"_intent.csv")
        os.rename("p/"+str(b)+"_permission.txt","p/"+str(b)+"_permission.csv")
    except:
        print " \n \n***** Error in File *********", a ,"\n \n "
    from xml.dom.minidom import parseString
    import os
    import sys

    import glob
    for a in glob.glob("*.apk"):
    print "apktool working"
    try:
        x="apktool d -f "+str(a)
        os.system(x)
        b=a[0:len(a)-4]

        if os.path.isfile(str(b)+"_intent.csv"):
            os.remove(str(b)+"_intent.csv")
            os.remove(str(b)+"_permission.csv")

        with open(str(b)+"/AndroidManifest.xml","r") as f:
            data = f.read()

        dom = parseString(data)
```

```
    nodes = dom.getElementsByTagName("intent-filter")
    intents=open("i/"+str(b)+"_intent.txt","a")
    permission=open("p/"+str(b)+"_permission.txt","a")
    intents.write("intent_name,Vector,type")
    intents.write("\n")
    print "Extracting Intents"
    for node in nodes:
        y1=str(node.toxml()).find("""""")
        y2=str(node.toxml()).find("""""",y1+1)
        intents.write(str(node.toxml())[y1+1:y2])
        intents.write(",1")
        channels =node.getElementsByTagName("data")
        if channels ==[]:
            intents.write(",explicit")

        else:
            intents.write(",implicit")
        intents.write("\n")
    intents.close()
    print "Extracting permission"
    permission.write("permission_name,Vector")
    permission.write("\n")
    nodes = dom.getElementsByTagName("uses-permission")
    for node in nodes:
        y1=str(node.toxml()).find("""""")
        y2=str(node.toxml()).find("""""",y1+1)
        permission.write(str(node.toxml())[y1+1:y2])
        permission.write(",1")
        permission.write("\n")
    permission.close()
    os.rename("i/"+str(b)+"_intent.txt","i/"+str(b)+"_intent.csv")
    os.rename("p/"+str(b)+"_permission.txt","p/"+str(b)+"_permission.csv")
except:
    print " \n \n***** Error in File *********", a ,"\n \n "
import pandas as pd

import os
```

```
import numpy as np
if os.path.exists("feature_name.csv"):
    os.remove("feature_name.csv")
if os.path.exists("feature_set.csv"):
    os.remove("feature_set.csv")

import glob
for name in glob.glob("Combined/*.csv"):

    print "File in progress:", name

    datafile=pd.read_csv(name)
    a1=datafile.iloc[:, 0]
    with open("feature_name.csv", "a") as f:
        a1.to_csv(f, header=False,index=False)


datafile=pd.read_csv("feature_name.csv")
a1_d=datafile.iloc[:, 0]



data=a1_d.unique()
import pandas as pd
df_1 = pd.DataFrame(data)
df_1.to_csv("feature_name_unique.csv")



line=""

for name in glob.glob("Malware/*.csv"):
    myf=open("feature_set.csv","a")

    print "File in progress:", name

    datafile=pd.read_csv(name)
    a1=datafile.iloc[:, 0]
```

```
myf.write("1")
myf.write("\n")
for d in range(0,len(data)):

    datalog=0
    for d1 in range(0,len(a1)):
        if data[d] == a1[d1]:
            datalog=1
    line+=str(datalog)+","
    myf.write(line)


    line=""
myf.close()


for name in glob.glob("Non_Malware/*.csv"):
myf=open("feature_set.csv","a")

print "File in progress:", name

datafile=pd.read_csv(name)
a1=datafile.iloc[:, 0]
myf.write("0")
myf.write("\n")
for d in range(0,len(data)):

    datalog=0
    for d1 in range(0,len(a1)):
        if data[d] == a1[d1]:
            datalog=1
    line+=str(datalog)+","
    myf.write(line)


    line=""
myf.close()
```

```
print "All Features:",len(a1_d)
print "Unique Features:",len(a1_d.unique())
#if os.path.exists("Output.csv"):
 #   os.remove("Output.csv")
import pandas as pd

import os

import numpy as np
if os.path.exists("feature_name.csv"):
    os.remove("feature_name.csv")
if os.path.exists("feature_set.csv"):
    os.remove("feature_set.csv")

import glob
for name in glob.glob("Combined/*.csv"):

    print "File in progress:", name

    datafile=pd.read_csv(name)
    a1=datafile.iloc[:, 0]
    with open("feature_name.csv", "a") as f:
        a1.to_csv(f, header=False,index=False)


datafile=pd.read_csv("feature_name.csv")
a1_d=datafile.iloc[:, 0]



data=a1_d.unique()
import pandas as pd
df_1 = pd.DataFrame(data)
df_1.to_csv("feature_name_unique.csv")
line=""
```

```
for name in glob.glob("Malware/*.csv"):
myf=open("feature_set.csv","a")

print "File in progress:", name

datafile=pd.read_csv(name)
a1=datafile.iloc[:, 0]
myf.write("1")
myf.write("\n")
for d in range(0,len(data)):

    datalog=0
    for d1 in range(0,len(a1)):
        if data[d] == a1[d1]:
            datalog=1
    line+=str(datalog)+","
    myf.write(line)


    line=""
myf.close()


for name in glob.glob("Non_Malware/*.csv"):
myf=open("feature_set.csv","a")

print "File in progress:", name

datafile=pd.read_csv(name)
a1=datafile.iloc[:, 0]
myf.write("0")
myf.write("\n")
for d in range(0,len(data)):

    datalog=0
    for d1 in range(0,len(a1)):
        if data[d] == a1[d1]:
            datalog=1
```

```
        datalog=0
        for d1 in range(0,len(a1)):
            if data[d] == a1[d1]:
                datalog=1
        line+=str(datalog)+","
        myf.write(line)


        line=""
myf.close()

print "All Features:",len(a1_d)
print "Unique Features:",len(a1_d.unique())
#if os.path.exists("Output.csv"):
 #   os.remove("Output.csv")
```

Appendix B

Standard Configurations of Random Forest in WEKA 3.6

Property	Configurations
Maxdepth	None
Max feature	Auto
Min samplesleaf	1
Minweight fractionleaf	0
Warmstart	FALSE

Standard Configurations of Decision Tree in WEKA 3.6

Property	Configurations
Maxfeatures	None
Maxdepth	None
Minsamplesleaf	1
Minweightfractionleaf	0
Presort	FALSE

Standard Configurations of Gradient Boosting in WEKA 3.6

Property	**Configurations**
Maxfeatures	None
Maxdepth	3
Minsamplesleaf	1
Minweightfractionleaf	0
Warmstart	FALSE