

**AUTOMATED CLASSIFICATION OF ANDROID APPS USING  
MACHINE LEARNING CLASSIFIERS**

**A PROJECT REPORT**

*Submitted by*

**DIVAKAR RAJU C 412711205016**

**NARESH BALAJI A 412711205043**

**NAVEEN KUMAR S 412711205044**

*in partial fulfillment for the award of the degree*

*of*

**BACHELOR OF TECHNOLOGY**

**IN**

**INFORMATION TECHNOLOGY**



**APRIL 2015**

ANNA UNIVERSITY: CHENNAI 600 025

**BONAFIDE CERTIFICATE**

Certified that this project report “**Automated Classification of Android Apps using Machine Learning Classifiers**” is the bonafide of **Divakar Raju C (412711204016)**, **Naresh Balaji A(412711205043)**, **Naveen Kumar S (412711205044)** who carried out the project work under my supervision.

**SIGNATURE**

Dr.S.Surendran M.E., Ph.D

**HEAD OF THE DEPARTMENT**

DEPT OF INFORMATION TECHNOLOGY  
TAGORE ENGINEERING COLLEGE,  
RATHINAMANGALAM,  
VANDALUR,  
CHENNAI-600 127

Submitted for viva-voice on .....

**Internal Examiner**

**SIGNATURE**

Mr.A.Chinnasamy M.E.,MBA.,(Ph.D)

**SUPERVISOR**

DEPT OF INFORMATION TECHNOLOGY  
TAGORE ENGINEERING COLLEGE,  
RATHINAMANGALAM,  
VANDALUR,  
CHENNAI-600 127

**External Examiner**

## ACKNOWLEDGEMENT

We would like to thank the management and Chairperson **Prof.Ms.M.Mala,M.A.,M.Phil.**, for extending their support in providing all the facilities for completing our project.

We would like to express our sincere thanks to our Principal, **Dr.P.Kasinatha Pandiyan, B.E., M.S., Ph.D., F.I.E., M.I.S.T.E.**, for having been a Source of inspiration to us and providing the facilities required for the project Completion.

We would like to thank the Head of the Department, Information Technology, **Dr.S.Surendran, M.E., Ph.D.**, for offering all the support and encouragement that was instrumental in the successful completion of the project.

We would like to thank **Mr.A.Chinnasamy M.E.,MBA.,(Ph.D).**, our project guide, for his constant guidance and encouragement in bringing out this project successfully.

We would like to extend our thanks to **Mrs.G.Bhuvaneshwari M.E.**, our project coordinator, for supporting in the development of this project.

We would like to express our gratitude to **Mr.N.Devarajan.**, Application Developer at ThoughtWorks Inc for his external guidance in shaping up this project.

We would also like to thank **Mr.M.Vasanthan** and **Mr.P.S.Omprakash** our lab technicians for their technical support in developing this project.

## **ABSTRACT**

Android applications are becoming increasingly because android phones are widespread and steadily gaining popularity. Unfortunately, such phenomenon draws attention to malicious application developers so that malicious applications are increasing rapidly. These malicious applications are capable to stealing sensitive and private user information and to damage the phone hardware as well. Therefore it is highly essential to identify and detect these malicious applications which is present in both official Android Market and alternative application markets.

We extract two features for the Android applications- Permissions extracted from AndroidManifest.xml file and the Strings extracted from the complied source code file(dex file). Permissions are the kernel access given to each Android application to access core functionalities of the phone hardware. We classify these permissions according to their usage by both genuine and malicious applications. We also extract theconst-string labelled strings from the source code of applications. Since, source code analysis is used in detection of desktop malware, we apply the same mechanism to identify and detect Android malware.

Machine Learning algorithms are used to train and test our classifier framework. We use 3 supervised learning algorithms for our project- NaiveBayes, J48(Weka implementation of C4.5 algorithm) and NaiveBayesUpdateable. We have collected 182 genuine applications and 49 malware applications to train and test our framework. The classifier framework is trained using a combination of all the genuine and malware applications and tested by providing individual samples of the malware applications.

## **TABLE OF CONTENTS**

<b>CHAP NO.</b>	<b>TITLE</b>	<b>PAGE NO.</b>
	<b>ABSTRACT</b>	<b>i</b>
	<b>TABLE OF CONTENTS</b>	<b>ii</b>
	<b>LIST OF FIGURES</b>	<b>vi</b>
<b>1.</b>	<b>INTRODUCTION</b>	<b>1</b>
	1.1 SCOPE OF PROJECT	3
	1.2 INTRODUCTION TO ARTIFICIAL INTELLIGENCE	3
	1.3 INTRODUCTION TO MACHINE LEARNING	4
	1.4 TYPES OF MACHINE LEARNING	4
	1.5 INTRODUCTION TO SUPERIVSED MACHINE LEARNING	5
<b>2.</b>	<b>LITERATURE SURVEY</b>	<b>7</b>
	2.1 INTRODUCTION	7
	2.2 LITERATURE SURVEY	8
<b>3.</b>	<b>SYSTEM REQUIREMENT SPECIFICATION</b>	<b>14</b>
	2.3 INTRODUCTION	14
	2.4 SYSTEM REQUIREMENTS	14
	2.4.1 HARDWARE SPECIFICATION	14
	2.4.2 SOFTWARE SPECIFICATION	14

2.5	TECHNOLOGY USED	15
2.5.1	SOFTWARE DESCRIPTION	15
2.5.2	INTRODUCTION ON J2EE	15
2.5.3	COMPONENTS OF J2EE	16
2.5.4	J2EE CLIENTS	16
2.5.5	WEB CLIENTS	17
2.5.6	APPLETS	17
2.5.7	APPLICATION CLIENTS	17
2.6	WEKA (MACHINE LEARNING)	18
2.7	AXMLPRINTER2 TOOL.JAR	19
2.8	DEDEXER.JAR	20
2.9	WINDOWS POWERSHELL	20
<b>3.</b>	<b>SYSTEM ANALYSIS</b>	<b>21</b>
3.1	INTRODUCTION	21
3.2	EXISTING SYSTEM	21
3.2.1	DISADVANTAGES	22
3.3	PROPOSED SYSTEM	22
3.3.1	ADVANTAGES OF PROPOSED SYSTEM	23
3.4	WORKING METHODOLOGY OF PROPOSED SYSTEM	23
<b>4.</b>	<b>SYSTEM ARCHITECTURE</b>	<b>24</b>
4.1	INTRODUCTION	24
4.2	OVERALL ARCHITECTURE	25
4.3	UML DIAGRAM	27

4.3.1	USE CASE DIAGRAM	27
4.3.1.1	FEATURE EXTRACTION USE CASE DIAGRAM	28
4.3.1.2	FRAMEWORK TRAINING AND TESTING USE CASE DIAGRAM	29
4.3.2	SEQUENCE DIAGRAM	30
4.3.2.1	APP CLASSIFIER SEQUENCE DIAGRAM	30
4.3.3	COLLABORATION DIAGRAM	31
4.3.3.1	ANDROID APP CLASSIFIER COLLABORATIONDIAGRAM	32
<b>6</b>	<b>MODULES</b>	<b>33</b>
6.1	INTRODUCTION	33
6.2	MODULE	33
6.3	MODULE DESCRIPTION	34
6.3.1	DATA COLLECTION(APK FILES)	34
6.3.2	FEATURE EXTRACTION	35
6.3.2.1	XML DECRYPTION	36
6.3.2.2	DEX FILE DECRYPTION	37
6.3.3	DATASET CREATION	38
6.3.3.1	ARFF FILES ON PERMISSIONS	39
6.3.3.2	ARFF FILES ON STRINGS	40
6.3.4	TRAINING THE CLASSIFIER (USING TRAINING DATASET)	40

	6.3.4.1 J48 (OR) C4.5 MACHINE LERINING	
	CLASSIFIER	41
	6.3.4.2NAIVEBAYES MACHINE LEARNING	
	CLASSIFIER	42
	6.3.5 FRAMEWORK TESTING(USING TEST DATA)	43
<b>7</b>	<b>IMPLEMENTATION AND TESTING</b>	<b>44</b>
	6.4 INTRODUCTION	44
	6.5 TYPES OF TESTING	44
	6.6 SDLC	45
<b>8</b>	<b>CONCLUSION</b>	<b>47</b>
<b>9</b>	<b>FUTURE ENHANCEMENTS</b>	<b>48</b>
	<b>APPENDIX-1</b>	<b>49</b>
	<b>APPENDIX-2</b>	<b>76</b>
	<b>APPENDIX-3</b>	<b>80</b>
	<b>REFERENCES</b>	<b>91</b>



## LIST OF FIGURES

<b>FIGURE NO.</b>	<b>DESCRIPTION</b>	<b>PAGE NO.</b>
Fig 1.5	SUPERVISED MACHINE LEARNING	6
Fig 5.2	OVERALL ARCHITECTURE	25
Fig 5.3.1.1	FEATURE EXTRACTION USE CASE DIAGRAM	28
Fig 5.3.1.2	FRAMEWORK TRAINING AND TESTING USE CASE DIAGRAM	29
Fig 5.3.2.1	APP CLASSIFIER SEQUENCE DIAGRAM	31
Fig 5.3.3.1	ANDROID APP CLASSIFIER COLLABORATION DIAGRAM	32
Fig 6.3.2.1	XML DECRYPTION	36
Fig 6.3.2.2	DEX FILE DECRYPTION	38
Fig 7.3	SDLC	46
Fig (A)	ENCRYPTED ANDROID MANIFEST.XML	76
Fig (B)	DECRYPTED ANDROID MANIFEST.XML	77
Fig (C)	COMPILED SOURCE CODE(DEX FILE)	78
Fig (D)	DECOMPILED DEX FILE	79
Fig (E)	GENERATED ARFF FILE ON GENUINE PERMISSION	80
Fig (F)	GENERATED ARFF FILE ON MALWARE PERMISSIONS	81
Fig (G)	GENERATED ARFF FILE ON GENUINE STRINGS	82
Fig (H)	GENERATED ARFF FILE ON MALWARE STRINGS	83
Fig (I)	CROSS VALIDATION OUTPUT ON PERMISSIONS	84
Fig (J)	CROSS VALIDATION OUTPUT ON STRINGS	85
Fig (K)	INDIVIDUAL TESTING RESULTS	86

# **CHAPTER 1**

## **INTRODUCTION**

Since Google developed an android platform, android phones have evolved over the years. Smartphones are becoming increasingly popular. They have evolved into small and portable personal computers that allow users to browse the Internet, send e-mails or connect to a remote desktop. These small computers are able to perform complex computing tasks and communicate through different methods (e.g., GPRS, WiFi or Bluetooth). In the last decade, users of these devices had problems installing mobile applications. They had to download an application from a website and then, install it on the device. In order to protect the device and avoid piracy, several operating systems (e.g., Symbian) employ an authentication system based on certificates that causes several inconveniences for the users (e.g., they cannot install applications despite having bought them). Nowadays, thanks to the deployment of Internet connections in mobile devices, there are new methods to distribute applications. Users can install any application without even connecting the mobile device to the computer. They only need an account of an application store in order to buy and install new applications. Apple's AppStore was the first store to implement this new model and it turned to be successful. Other manufacturers such as Google, RIM and Microsoft have followed the same business model developing application stores accessible from the device. These facts have increased the number of developers for mobile platforms and the number of mobile applications. According to Apple<sup>1</sup>, the number of available applications on the App Store is over 350,000, while Android Market<sup>2</sup> has over 200,000 applications. Regarding the application stores, while for Apple devices the AppStore is the single official way to obtain applications, Android allows users to install applications that have been downloaded from alternative markets or directly from Internet. According to their response to the US Federal

Communication Commission's July 20093, Apple applies a rigorous review process made by at least two reviewers. In contrast, Android relies on its security permission system and on the user's sound judgment. Unfortunately, users usually have no security consciousness and they do not read required permissions before installing an application. Although both AppStore and Android Market include clauses in the terms of services that urge developers not to submit malicious software, both have hosted malware in their stores. To solve this problem, they have developed tools for removing remotely these malicious applications. Therefore both models are insufficient to ensure user's safety and new models should be included in order to improve the security of the devices. Machine learning techniques have been applied for classifying applications mainly focused on malware detection. Their goal is to classify applications on 2 main categories: malware or genuine. There are other works that try to classify applications specifying the malware class, e.g., trojan, worms, virus. With regards to Android applications, there is a lack of malware samples. Anyway, the number of samples is increasing exponentially and several approaches have been proposed to detect Android Malware. Shabtai et al. trained machine learning models using other features, e.g., parsing apk contained xml and counting xml elements, attributes or namespaces. To evaluate their models, they selected features using three selection methods: Information Gain, Fisher Score and Chi-Square. They obtained 89% of accuracy classifying applications into 2 categories: tools or games. In light of this background, we propose here a new method for classifying Android applications into several categories (e.g., entertainment or productivity) using features extracted both from the Android Market and the application itself.

Summarizing, our main findings in this paper are:

- We describe the process of extracting features from the Android .apk files.
- We propose a new representation for Android applications in order to develop an automatic categorization approach.
- We perform an empirical validation of our approach and show that it can achieve high accuracy rates.

## **1.1 SCOPE OF THE PROJECT**

The scope of the project is to classify the android apps whether it is genuine or malware by validating kernel permissions and strings.

## **1.2 INTRODUCTION TO ARTIFICIAL INTELLIGENCE**

Artificial intelligence (AI) is the intelligence exhibited by machines or software. It is an academic field of study which studies the goal of creating intelligence. Major AI researchers and textbooks define this field as "the study and design of intelligent agents", where an intelligent agent is a system that perceives its environment and takes actions that maximize its chances of success. John McCarthy, who coined the term in 1955, defines it as "the science and engineering of making intelligent machines".

AI research is highly technical and specialized, and is deeply divided into subfields that often fail to communicate with each other. Some of the division is due to social and cultural factors: subfields have grown up around particular institutions and the work of individual researchers. AI research is also divided by several technical issues. Some subfields focus on the solution of specific problems. Others focus on one of several possible approaches or on the use of a particular tool or towards the accomplishment of particular applications.

## 1.3 INTRODUCTION TO MACHINE LEARNING

Machine learning is a scientific discipline that explores the construction and study of algorithms that can learn from data. Such algorithms operate by building a model from example inputs and using that to make predictions or decisions, rather than following strictly static program instructions. Machine learning is closely related to and often overlaps with computational statistics; a discipline which also specializes in prediction-making.

Machine learning is a subfield of computer science stemming from research into artificial intelligence. It has strong ties to statistics and mathematical optimization, which deliver methods, theory and application domains to the field. Machine learning is employed in a range of computing tasks where designing and programming explicit, rule-based algorithms is infeasible. Example applications include spam filtering, optical character recognition (OCR), search engines and computer vision. Machine learning is sometimes conflated with data mining, although that focuses more on exploratory data analysis. Machine learning and pattern recognition "can be viewed as two facets of the same field.

## 1.4 TYPES OF MACHINE LEARNING

Machine learning tasks are typically classified into three broad categories, depending on the nature of the learning "signal" or "feedback" available to a learning system. These are:

- **Supervised learning:** The computer is presented with example inputs and their desired outputs, given by a "teacher", and the goal is to learn a general rule that maps inputs to outputs.
- **Unsupervised learning:** no labels are given to the learning algorithm, leaving it on its own to find structure in its input. Unsupervised learning

can be a goal in itself (discovering hidden patterns in data) or a means towards an end.

- In **reinforcement learning**, a computer program interacts with a dynamic environment in which it must perform a certain goal (such as driving a vehicle), without a teacher explicitly telling it whether it has come close to its goal or not. Another example is learning to play a game by playing against an opponent.

Between supervised and unsupervised learning is semi-supervised learning, where the teacher gives an incomplete training signal: a training set with some (often many) of the target outputs missing. Transduction is a special case of this principle where the entire set of problem instances is known at learning time, except that part of the targets are missing.

## **1.5 INTRODUCTION TO SUPERVISED MACHINE LEARNING**

Supervised learning is the machine learning task of inferring a function from labeled training data. The training data consist of a set of training examples. In supervised learning, each example is a pair consisting of an input object (typically a vector) and a desired output value (also called the supervisory signal). A supervised learning algorithm analyzes the training data and produces an inferred function, which can be used for mapping new examples. An optimal scenario will allow for the algorithm to correctly determine the class labels for unseen instances. This requires the learning algorithm to generalize from the training data to unseen situations in a "reasonable" way.

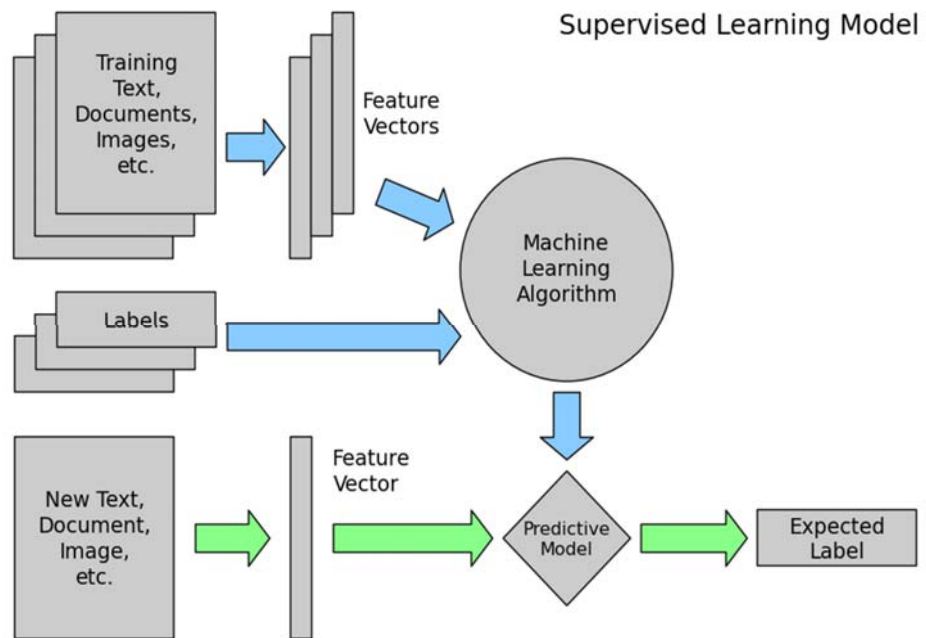


Figure 1.5 SUPERVISED MACHINE LEARNING

## **CHAPTER 2**

### **LITERATURE REVIEW**

#### **2.1 INTRODUCTION**

The purpose of literature survey is to give the complete information about the reference papers. The goal of the literature review is to specify the technical related papers which form the foundation of this project. Literature survey is the documentation of a comprehensive review of the published and unpublished work from secondary sources data in the areas of specific interest to the researcher. The library is a rich storage base for secondary data and researchers used to spend several weeks and sometimes months going through books, journals, newspapers, magazines, conference proceedings, doctoral dissertations, master's theses, government publications and financial reports to find information on their research topic. With computerized databases now readily available and accessible the literature search is much speedier and easier.

The researcher could start the literature survey even as the information from the unstructured and structured interviews is being gathered. Reviewing the literature on the topic area at this time helps the researcher to focus further interviews more meaningfully on certain aspects found to be important in the published studies even if these had not surfaced during the earlier questioning. So the literature survey is important for gathering the secondary data for the research which might be proved very helpful in the research. The literature survey can be conducted for several reasons. The literature review can be in any area of the business.



## 2.2 LITERATURE SURVEY

**[1] Wei Wang, Xing Wang, Dawei Feng, Jiqiang Liu, Zhen Han, Xiangliang Zhang, “Exploring Permissions-induced Risk in Android Applications for Malicious Application Detection”**

Android permissions requested by an app depict the app’s behavioral patterns. In order to help understanding Android permissions, in this paper, the authors have explored the permission-induced risk in Android apps on three levels in a systematic manner. First, they thoroughly analyze the risk of individual permissions and the risk of group of collaborative permissions. They employ three feature ranking methods, namely mutual information, Correlation Coefficient and T-test to rank Android individual permissions with respect to their risk. They then use Sequential Forward Selection(SFS) as well as Principal Component Analysis(PCA) to identify risky permissions subsets. Secondly, they evaluate the usefulness of risky permissions for malware detection with Support Vector Machine (SVM), decision trees as well as random forest. Thirdly, they have performed in depth analysis of the detection results and discuss the feasibility as well as the limitation of malware detection based on permissions request

They have evaluated their methods on a very large official app set consisting of 310,926 benign apps and 4868 real world malware apps and on a third party app sets. Their empirical results show that their malware detectors built on risky permissions give satisfied performance with a detection rate of 94.62% with a false positive rate of 0.6%, catch the malware apps essential patterns on violating permissions access regulations, and are universally applicable to unknown malware apps(detection rate of 74.03%)

**[2]Dong-uk Kim, Jeongtae Kim, Sehun Kim, "Malicious Application Detection Framework using Feature Extraction Tool on Android Market".**

In Machine Learning methods, there are two ways to classify instances- dynamic detection and static detection to detect these malicious applications. The dynamic detection has a high detection rate but uses a lot of resources and takes long time to detect malicious behaviors. In addition, it is possible to infect malicious code after implementing detection method. On the other hand, the static detection has a small amount of resources and quick detection time but has a low detection rate.

To these weaknesses, they have proposed a malicious application detection framework on Android market and an automatic extraction tool of malicious features for static detection. This framework is able to perform both detection methods using Machine Learning and is expected to perform a thorough analysis than previous framework. Also they have introduced automatic extraction tool is expected to help a code analysis for static detection.

This framework is able to perform both detection methods using Machine Learning in Android market so that the problem of resource limitation can be solved. In addition, this framework performs static detection based on methods of APIs, which is used to manage Android services and is related to sending SMS, calling and information spill, so that the problem of static detection based on permission can be solved. The features based of APIs can be extracted by automatic extraction tool, which is made by us, for static detection.

**[3]BorjaSanz, Igor Santos, Carlos Laorden, XabierUgarte-Pedrero and Pablo Garcia Bringas, “On the Automatic Categorization of Android Applications”**

They have proposed an approach that can automatically characterize the different types of applications can be helpful for both organizing the Android Market and detecting fraudulent or malicious applications. In this paper, they propose a new method for categorizing Android applications through machine-learning techniques. To represent each application, their method extracts different feature sets: (i) the frequency of occurrence of the printable strings, (ii) the different permissions from the application itself and (iii) the permissions of the application extracted from the Android Market. They have evaluated this approach of automatically categorization of Android applications and show that the method achieves a high performance.

They use Android Market features to add another kind of information to the model. The Android Market contains information about the behavior of an application and, also, information users provide, such as the application rating or the number of ratings, which according to the results of the application of the Information Gain method, are important features.

They obtain the permissions of the application using both the Android Market and the configuration file “AndroidManifest.xml”. Although they may seem redundant information, in some cases, there are several changes between permissions from the application and the Market (e.g., the difference between app version and hosted version). In these cases, Android Market provides the permissions of the last version of the application.

**[4]BorjaSanz, Igor Santos, Carlos Laorden, XabierUgarte-Pedrero and Pablo Garcia Bringas, “MADS: Malicious Android Applications Detection through String Analysis”**

The authors of this paper presents MADS (Malicious Android applications Detection through String analysis), a novel approach for detection of malware in Android. This method employs the strings contained in the disassembled Android applications, constructing a bag of words model in order to train machine-learning algorithms to provide detection of malicious applications.

In summary, their main contributions are:

- They present a new technique for the representation of Android applications, based on the bag of words model formed by the strings contained in the disassembled application.
- They adapt well-known machine learning classifiers to provide detection of malicious applications in Android.
- They have found out that machine-learning algorithms can provide detection of malicious applications in Android using the strings contained in the disassembled application as features.

One of the most widely-used techniques for classic malware detection is the usage of strings contained in the files. This technique extracts every character strings within an executable file. The information that may be found in these strings can be, for example, options in the menus of the application or malicious URLs to connect to. In this way, by means of an analysis of these data, it is possible to extract valuable information in order to determine whether an application is malicious or not.

**[5]Matthew G. Schultz, ErezZadok, Salvatore J. Stolfo, “Data Mining Methods for Detection of New Malicious Executables”**

Data mining methods detect patterns in large amounts of data, such as byte code, and use these patterns to detect future instances in similar data. Our framework uses classifiers to detect new malicious executable. A classifier is a rule set, or detection model, generated by the data mining algorithm that was trained over a given set of training data.

They designed a framework that used data mining algorithms to train multiple classifiers on a set of malicious and benign executable to detect new examples. The binaries were first statically analyzed to extract properties of the binary, and then the classifiers trained over a subset of the data.

**[6]Yajin Zhou, Zhi Wang, Wu Zhou, Xuxian Jiang, “Hey, You, Get Off My Market: Detecing Malicious Apps in Official and Alternative Android Markets”**

In this paper, the authors have presented a systematic study for the detection of malicious applications (or apps) on popular Android Markets. To this end, they first propose a permission based behavioral footprinting scheme to detect new samples of known Android malware families. Then they apply a Heuristics-based filtering scheme to identify certain inherent behaviors of unknown malicious families. They implemented both schemes in a system called DroidRanger.

The experiments with 204, 040 apps collected from five different Android Markets in May-June 2011 reveal 211 malicious ones: 32 from the official Android Market (0.02% infection rate) and 179 from alternative marketplaces (infection rates ranging from 0.20% to 0.47%). Among those malicious apps, their system also uncovered two zero-day malware (in 40 apps): one from the official Android Market and the other from alternative marketplaces.

**[7]Adam P. Fuchs, AvikChaudhuri, Jeffrey S. Foster , “SCanDroid: Automated Security Certification of Android Applications”**

In this paper the authors have presented SCANDROID, a tool for reasoning automatically about the security of Android applications. SCANDROID’s analysis is modular to allow incremental checking of applications as they are installed on an Android device. It extracts security specifications from manifests that accompany such applications, and checks whether data flows through those applications are consistent with those specifications. To our knowledge, SCANDROID is the first program analysis tool for Android, and we expect it to be useful for automated security certification of Android applications

**[8]Steven Artz, Siegfried Rasthofer, Eric Bodden, “SuSi: A Tool for the Fullt Automated Classification and Categorization of Android Sources and Sinks”**

Authors have propose SuSi, a novel and fully automated machine-learning approach for identifying sources and sinks directly from the Android source code. On the training set, SuSi achieves a recall and precision of more than 92%. To provide more fine-grained information, SuSi further categorizes the sources (e.g., unique identifier, location information, etc.) and sinks (e.g., network, file, etc.), with an average precision and recall of about 89%.

## **CHAPTER 3**

### **SYSTEM REQUIREMENT SPECIFICATION**

#### **3.1 INTRODUCTION**

The requirements specification is a technical specification of requirements for the software products. It is the first step in the requirements analysis process it lists the requirements of a particular software system including functional, performance and security requirements. The requirements also provide usage scenarios from a user, an operational and an administrative perspective. The purpose of software requirements specifications is to provide a detailed overview of the software project, its parameters and goals. This describes the project target audience and its user interface, hardware and software requirements. It defines how the client, team and audience see the project and its functionality.

#### **3.2 SYSTEM REQUIREMENTS:**

##### **3.2.1 HARDWARE SPECIFICATION**

System	: Dual Core 2.4 GHz
Speed	: 2.40 GHz
RAM Capacity	: 4 GB
Hard Disk Capacity	: 40 GB
Keyboard	: Standard 102 Key

##### **3.2.2 SOFTWARE SPECIFICATION**

Operating System	: Windows 7 Ultimate/Service Pack 1
Front-end used	: Eclipse Luna, Weka
Front-end Version	: Release 4.4.0
Tools Used	:WEKA 3.6.12 AXMLPrinter2.jar,Dedexer.jar

Coding Platform : JDK 1.8

Coding languages used : Java, Windows PowerShell

### **3.3 TECHNOLOGY USED**

#### **3.3.1 SOFTWARE DESCRIPTION:**

##### **ECLIPSE LUNA**

Eclipse is an integrated development environment (IDE). It contains a base workspace and an extensible plug-in system for customizing the environment. Written mostly in Java, Eclipse can be used to develop applications. By means of various plug-ins, Eclipse may also be used to develop applications in other programming languages: Ada, ABAP, C, C++, COBOL, Fortran, Haskell, JavaScript, Lasso, Lua, Natural, Perl, PHP, Prolog, Python, R, Ruby (including Ruby on Rails framework), Scala, Clojure, Groovy, Scheme, and Erlang. It can also be used to develop packages for the software Mathematica. Development environments include the Eclipse Java development tools (JDT) for Java and Scala, Eclipse CDT for C/C++ and Eclipse PDT for PHP, among others.

The initial codebase originated from IBM VisualAge. The Eclipse software development kit (SDK), which includes the Java development tools, is meant for Java developers. Users can extend its abilities by installing plug-ins written for the Eclipse Platform, such as development toolkits for other programming languages, and can write and contribute their own plug-in modules.

#### **3.3.2 INTRODUCTION ON J2EE**

Java Platform, Enterprise Edition or Java EE is Oracle's enterprise Java computing platform. The platform provides an API and runtime environment for developing and running enterprise software, including network and web services, and other large-scale, multi-tiered, scalable, reliable, and secure



network applications. Java EE extends the Java Platform, Standard Edition (Java SE), providing an API for object-relational mapping, distributed and multi-tier architectures, and web services. The platform incorporates a design based largely on modular components running on an application server. Software for Java EE is primarily developed in the Java programming language. The platform emphasizes convention over configuration and annotations for configuration. Optionally XML can be used to override annotations or to deviate from the platform defaults.

### **3.3.3 COMPONENTS OF J2EE**

J2EE applications are made up of components. A J2EE component is a self-contained functional software unit that is assembled into a J2EE application with its related classes and files and that communicates with other components.

Application clients and applets are components that run on the client.

Java Servlet and JavaServer Pages (JSP) technology components are Web components that run on the server. Enterprise JavaBeans (EJB) components (enterprise beans) are business components that run on the server.

J2EE components are written in the Java programming language and are compiled in the same way as any program in the language. The difference between J2EE components and "standard" Java classes is that J2EE components are assembled into a J2EE application, verified to be well formed and in compliance with the J2EE specification, and deployed to production, where they are run and managed by the J2EE server.

The J2EE specification defines the following J2EE components:

### **3.3.4 J2EE CLIENTS**

A J2EE client can be a Web client or an application client.

### **3.3.5 WEB CLIENTS**

A Web client consists of two parts: dynamic Web pages containing various types of markup language (HTML, XML, and so on), which are generated by Web components running in the Web tier, and a Web browser, which renders the pages received from the server.

A Web client is sometimes called a thin client. Thin clients usually do not do things like query databases, execute complex business rules, or connect to legacy applications. When you use a thin client, heavyweight operations like these are off-loaded to enterprise beans executing on the J2EE server where they can leverage the security, speed, services, and reliability of J2EE server-side technologies.

### **3.3.6 APPLETS**

A Web page received from the Web tier can include an embedded applet. An applet is a small client application written in the Java programming language that executes in the Java virtual machine installed in the Web browser. However, client systems will likely need the Java Plug-in and possibly a security policy file in order for the applet to successfully execute in the Web browser.

Web components are the preferred API for creating a Web client program because no plug-ins or security policy files are needed on the client systems. Also, Web components enable cleaner and more modular application design because they provide a way to separate applications programming from Web page design. Personnel involved in Web page design thus do not need to understand Java programming language syntax to do their jobs.

### **3.3.7 APPLICATIONS CLIENTS**

A J2EE application client runs on a client machine and provides a way for users to handle tasks that require a richer user interface than can be provided

by a markup language. It typically has a graphical user interface (GUI) created from Swing or Abstract Window Toolkit (AWT) APIs, but a command-line interface is certainly possible.

Application clients directly access enterprise beans running in the business tier. However, if application requirements warrant it, a J2EE application client can open an HTTP connection to establish communication with a servlet running in the Web tier.

### **3.4 WEKA (MACHINE LEARNING)**

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. Weka is free software available under the GNU General Public License.

Weka (pronounced to rhyme with Mecca) is a workbench that 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. The original non-Java version of Weka was a TCL/TK front-end to (mostly third-party) modeling algorithms implemented in other programming languages, plus data preprocessing utilities in C, and a Make file-based system for running machine learning experiments. This original version was primarily designed as a tool for analyzing data from agricultural domains, but the more recent fully Java-based version (Weka 3), for which development started in 1997, is now used in many different application areas, in particular for educational purposes and research.

Advantages of Weka include:

- free availability under the GNU General Public License
- portability, since it is fully implemented in the Java programming language and thus runs on almost any modern computing platform

- a comprehensive collection of data preprocessing and modeling techniques
- ease of use due to its graphical user interfaces.

Weka supports several standard data mining tasks, more specifically, data preprocessing, clustering, classification, regression, visualization, and feature selection. All of Weka's techniques 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). Weka provides access to SQL databases using Java Database Connectivity and can process the result returned by a database query. It is not capable of multi-relational data mining, but there is separate software for converting a collection of linked database tables into a single table that is suitable for processing using Weka. Another important area that is currently not covered by the algorithms included in the Weka distribution is sequence modeling.

### **3.5 AXMLPRINTER2.JAR**

AXMLPrinter2 is a tool for decrypting AndoirManifest.xml files. This file contains the permissions used by the Android app. To view these permissions, first we need to extract the AndroidManifest.xml file from the apk package. But, this file will be in encrypted form. In order to decrypt the file, we use AXMLPrinter2 tool to decrypt. This tool is run from either CommandPrompt or Windows PowerShell.

#### **Syntax:**

**java -jar AXMLPrinter2.jar AndroidManifest.xml >> filename.txt**

### **3.6 DEDEXER.JAR**

Dedexer is a disassembler tool for DEX files. DEX is a format introduced by the creators of the Android platform . The format and the associated opcode set is in distant relationship with the Java class file format and Java bytecodes. Dedexer is able to read the DEX format and turn into an "assembly-like format". This format was largely influenced by the Jasmin syntax but contains Dalvikopcodes . For this reason, Jasmin is not able to compile the generated files.

#### **Syntax:**

**java -jar ddx.jar -d <directory><dex file>**

### **3.7 WINDOWS POWERSHELL**

Windows PowerShell is a task automation and configuration management framework from Microsoft, consisting of a command-line shell and associated scripting language built on the .NET Framework. PowerShell provides full access to COM and WMI, enabling administrators to perform administrative tasks on both local and remote Windows systems as well as WS-Management and CIM enabling management of remote Linux systems and network devices.

In PowerShell, administrative tasks are generally performed by cmdlets (pronounced command-lets), which are specialized .NET classes implementing a particular operation. Sets of cmdlets may be combined into scripts, executables (which are standalone applications), or by instantiating regular .NET classes (or WMI/COM Objects). These work by accessing data in different data stores, like the file system or registry, which are made available to the PowerShell runtime via Windows PowerShell providers.

Windows PowerShell also provides a hosting API with which the Windows PowerShell runtime can be embedded inside other applications. These applications can then use Windows PowerShell functionality to implement certain operations, including those exposed via the graphical interface.

## **CHAPTER 4**

### **SYSTEM ANALYSIS**

#### **4.1 INTRODUCTION**

System analysis is the study of how the systems entity interacts each other. This field is closely related to requirements analysis of operations research. It also helps one to provide decision to identify the best course of action that has been taken when developing this project. This field is closely related to requirements analysis or operations research. It is also "an explicit formal inquiry carried out to help someone (referred to as the decision maker) identify a better course of action and make a better decision than he might otherwise have made.

The terms analysis and synthesis come from Greek where they mean respectively "to take apart" and "to put together". These terms are used in scientific disciplines from mathematics and logic to economics and psychology to denote similar investigative procedures. Analysis is defined as the procedure by which we break down an intellectual or substantial whole into parts. Synthesis is defined as the procedure by which we combine separate elements or components in order to form a coherent whole. Systems analysis researchers apply methodology to the analysis of systems involved to form an overall picture. System analysis is used in every field where there is a work of developing something. Analysis can also be defined as a series of components that perform organic function together.

#### **4.2 EXISTING SYSTEM**

The existing system uses a framework that validates the permissions required by the applications based on the permissions it classifies whether the app is genuine or malware. Every android application requires Kernel permissions like "READ\_SMS","WRITE\_SMS" , "GET\_WIFI\_STATE" , "WRITE\_EXTERNAL\_STORAGE","ACCESS\_COARSE\_LOCATION","A

CCESS\_FINE\_LOCATION”,”ACTIVITY\_RECOGNITION”,”GET\_ACCOUNTS”,”MANAGE\_ACCOUNTS”,”USE\_CREDENTIALS”,”DISABLE\_KEYGUARD”,ACCESS\_WIFI\_STATE”,”ACCESS\_NETWORK\_STATE”-which are essential for applications to function. The existing system scans for more vulnerable kernel permissions which may steal or damage user information. The existing system modelled a framework to classify permissions as normal permissions and vulnerable permissions. Based on the usage of these vulnerable permission, the system classifies the applications as genuine or malware.

#### **4.2.1 DISADVANTAGES**

Classification of android apps based on only permissions is not sufficient. Many applications can be still infective by using normal permissions. Therefore detection using app permissions is not very effective. The existing system does not perform source code analysis to understand the behavior of the application. Each Android apk consists of a dex file which comprises the compiled source code. Since the source code reveals the working of the application, It is necessary to build a system that includes both the source code and kernel permissions.

#### **4.3 PROPOSED SYSTEM**

Our proposed system builds a model that uses both kernel permissions and source code of applications. We analyze “AndroidManifest.xml” file to obtain the kernel permissions that the application uses. We also decompile the “dex” file which is the compiled version of the source code to extract the strings used in the application. We decrypt the Manifest xml and decompile the dex file to obtain the required data for building the classifier framework. Since string analysis is used in detection of desktop malware, we use this mechanism to detect and identify android malware.

#### **4.3.1 ADVANTAGES OF PROPOSED SYSTEM**

Our proposed system classifies the android apps by analyzing both permissions and strings in the apps. Many android apps gets infected by classifying the apps only on the permission because many android apps are still infective with the genuine permission, so classifying the android apps with both strings and permission increase the efficiency of the system. Since string analysis is used to detect desktop malware we used this mechanism in our system.

#### **4.4 WORKING METHODOLOGY OF PROPOSED SYSTEM**

We first collect the android apps form the AppMarket in an app repository and futher we extract it using winrar tool .The future extraction consists of Encrypted AndroidManiFest.xml files and dex files. Android ManiFest.xml files are decrypted using AXMLPrinter2 tool and decompilation of dex files is done by dedexer tool. We automate this process for several files using windows powershell.

Using J48 and NaiveBayes Machine learning classifier algorithm we create training set of both genuine and malware. And then we use the collection of our test data to identify the nature of android app whether it is genuine or malware.



## **CHAPTER 5**

### **SYSTEM ARCHITECTURE**

#### **5.1 INTRODUCTION**

A system architecture or systems architecture is the conceptual model that defines the structure, behavior, and more views of a system. An architecture description is a formal description and representation of a system, organized in a way that supports reasoning about the structures of the system.

A system architecture can comprise system components, the externally visible properties of those components, the relationships (e.g. the behavior) between them. It can provide a plan from which products can be procured, and systems developed, that will work together to implement the overall system. There have been efforts to formalize languages to describe system architecture, collectively these are called architecture description languages (ADLs).system generally refers to a programmable hardware machine and its included program. And a systems engineer is defined as one concerned with the complete device, both hardware and software and, more particularly, all of the interfaces of the device, including that between hardware and software, and especially between the complete device and its user (the CHI). The hardware engineer deals (more or less) exclusively with the hardware device; the software engineer deals (more or less) exclusively with the software program; and the systems engineer is responsible for seeing that the software program is capable of properly running within the hardware device, and that the system composed of the two entities is capable of properly interacting with its external environment, especially the user, and performing its intended function.

## 5.2 OVERALL ARCHITECTURE

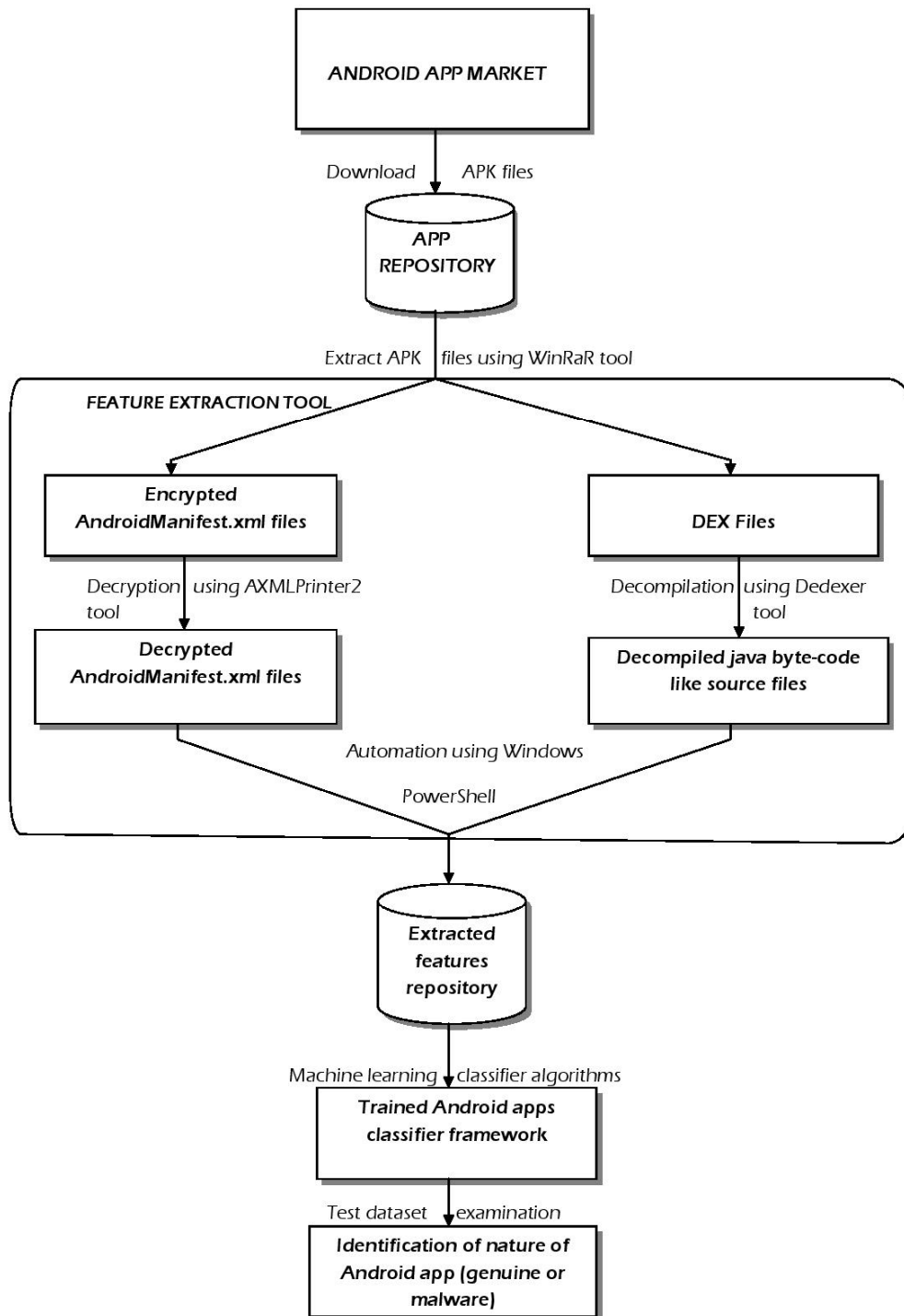


figure 5.2 Overall Architecture Diagram

The overall architecture diagram is given above. Our system begins with the phase of downloading Android applications. Both genuine and malware applications need to be downloaded to train and test our system. Malware Android applications cannot be easily downloaded from the internet. We became a member in virusshare.com- a malware repository used for research and analysis purposes. After we have created a repository of Android applications, we need to extract the AndroidManifest.xml and classes.dex files from each of the .apk package.

Both the files will be in encrypted form. In order to decrypt the manifest xml file, we use AXMLPrinter2.jar tool. To decompile the dex file (compiled java source code), we use dex2jar.jar tool. Both the tools are automated and executed using Windows PowerShell. Since we need to perform the decrypting of xml files and decompiling of dex files for all the .apk packages we have downloaded, it is literally impossible to separately use the tool for every application.

Therefore, we have written PowerShell scripts to automate the above process for all the Android applications (182 genuine and 49 malware applications). After extracting the files, we need to extract the features from both the files. From the manifest file, we need to extract system permissions and from the decompiled dex file we need to search and extract strings under the label “const-string”.

Both the feature sets will be the input to our classifier framework. We will train the framework using the feature set. After training phase comes the testing phase where individual applications are tested for their nature based on their system permissions and strings used in the source code. Testing of the applications will be based on the feature set given as input for the training phase of the framework

## 5.3 UML DIAGRAM

Unified Modeling Language (UML) is a standard language for creating blueprints that depicts the structure and design of the software system. The scope UML is a language for specifying artifacts, visualizing artifacts, constructing artifacts and documenting artifacts. UML provides the following diagrams to represent the software process.

- Use case diagram
- Sequence diagram
- Collaboration diagram

### 5.3.1 USE CASE DIAGRAM:

A use case diagram depicts the various operations that a system performs. It contains use cases, actors and their relationships. Use cases are the sequences of actions that form a single unit of work for an actor. An actor represents a user who is external to the system and interacts with the use case. Lines are relationship between two cases.

In systems engineering, use cases are used at a higher level than within software engineering, often representing missions or stakeholder goals. The detailed requirements may then be captured in Systems Modeling Language (SysML) or as contractual statements. As an important requirement technique, use cases have been widely used in modern software engineering over the last two decades.

The use case diagram given below depicts the process of downloading Android applications and extracting features from it. The process of feature extraction includes extracting Android permissions from the manifest xml file and extracting const-string labelled strings from the compiled java source code.

### 5.3.1.1 FEATURE EXTRACTION USE CASE DIAGRAM:

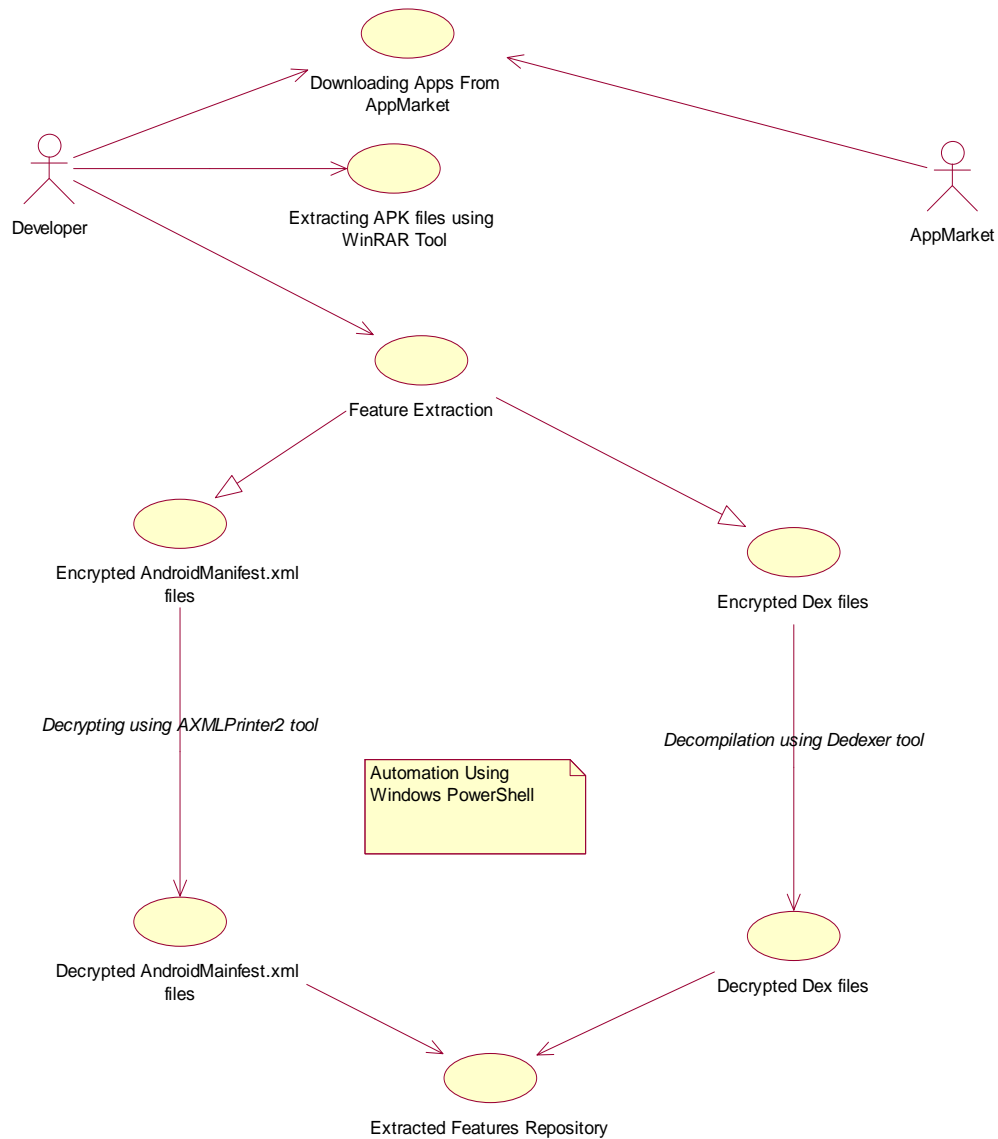


Figure 5.3.1.1 Feature Extraction Use Case diagram

Our system begins with the phase of downloading Android applications. Both genuine and malware applications need to be downloaded to train and test our system. After we have created a repository of Android applications, we need

to extract the AndroidManifest.xml and classes.dex files from each of the .apk package.

Both the files will be in encrypted form. In order to decrypt the manifest xml file, we use AXMLPrinter2.jar tool. To decompile the dex file (compiled java source code), we use dexdexer.jar tool. Both the tools are automated and executed using Windows PowerShell.

### 5.3.1.2 FRAMEWORK TRAINING AND TESTING USE CASE DIAGRAM

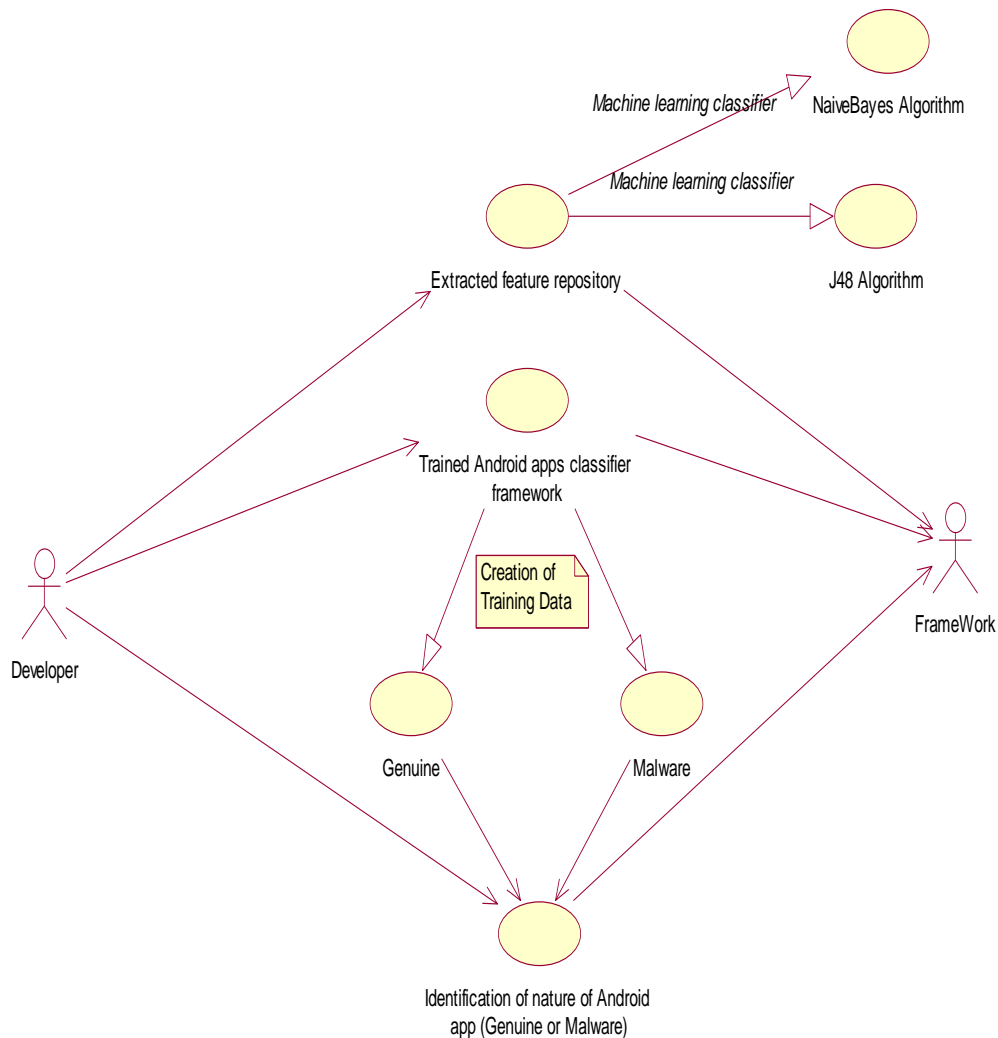


Figure 5.3.1.2 Framework Training and Testing Use Case diagram

Both the feature sets will be the input to our classifier framework. We will train the framework using the feature set. After training phase comes the testing phase where individual applications are tested for their nature based on their system permissions and strings used in the source code. Testing of the applications will be based on the feature set given as input for the training phase of the framework

### **5.3.2 SEQUENCE DIAGRAM**

Sequence diagrams represent and interactions between objects in the form of messages ordered in a sequence by time. The difference between the sequence and communication diagrams is that communications diagrams emphasis on the structural organization of objects as opposed to sequence diagrams they show the messages exchanged between objects ordered in sequence by time.

#### **5.3.2.1 APP CLASSIFIER SEQUENCE DIAGRAM**

The admin sequence portrays the sequence of activities carried out in the process of Android app classification. Both genuine and malware applications need to be downloaded to train and test our system. After we have created a repository of Android applications, we need to extract the AndroidManifest.xml and classes.dex files from each of the .apk package.

Both the files will be in encrypted form. In order to decrypt the manifest xml file, we use AXMLPrinter2.jar tool. To decompile the dex file (compiled java source code), we use dexdexer.jar tool. Both the tools are automated and executed using Windows PowerShell.

Both the feature sets will be the input to our classifier framework. After training phase comes the testing phase where individual applications are tested for their nature based on their system permissions and strings used in the source

code. Testing of the applications will be based on the feature set given as input for the training phase of the framework

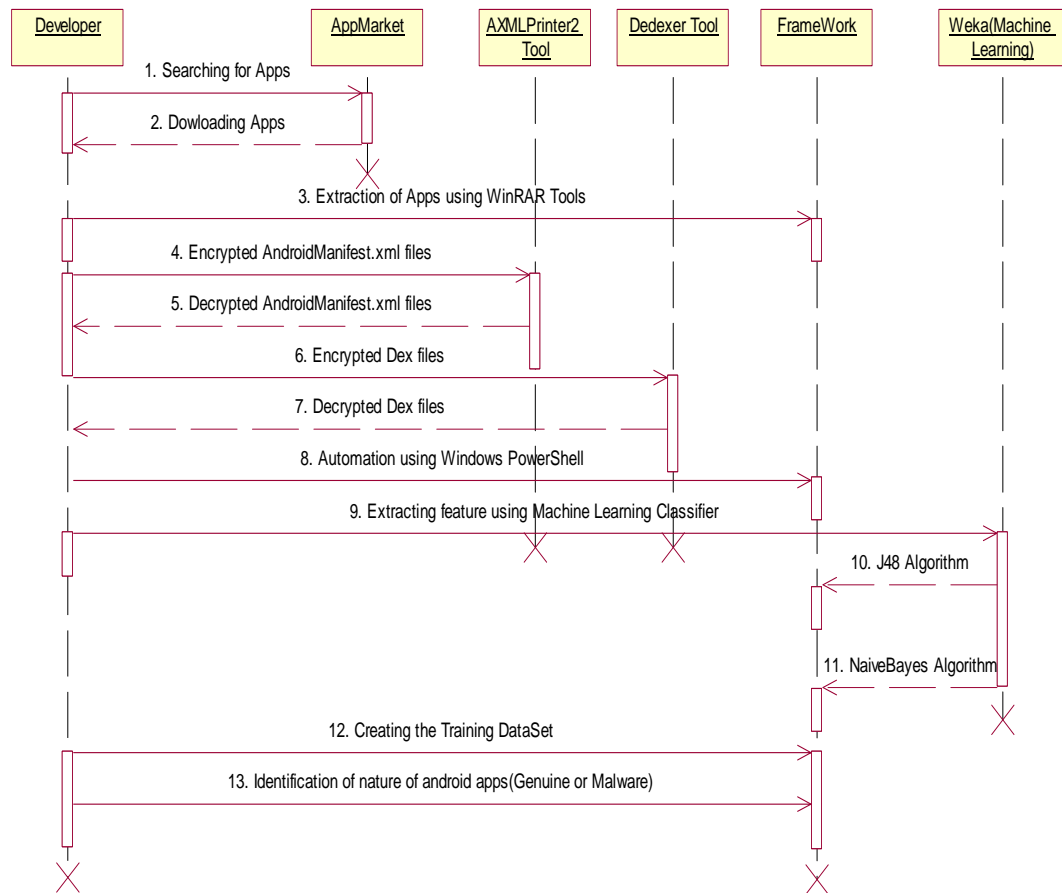


Figure 5.3.2.1 App Classifier Sequence diagram

### 5.3.3 COLLABORATION DIAGRAM:

A collaboration diagram describe interactions among objects in terms of sequenced in terms of sequenced messages. Collaboration diagrams represent a combination of information taken from class, sequence and use case diagram describing both static and dynamic behavior of the system.



### 5.3.3.1 ANDROID APP CLASSIFIER COLLABORATION DIAGRAM

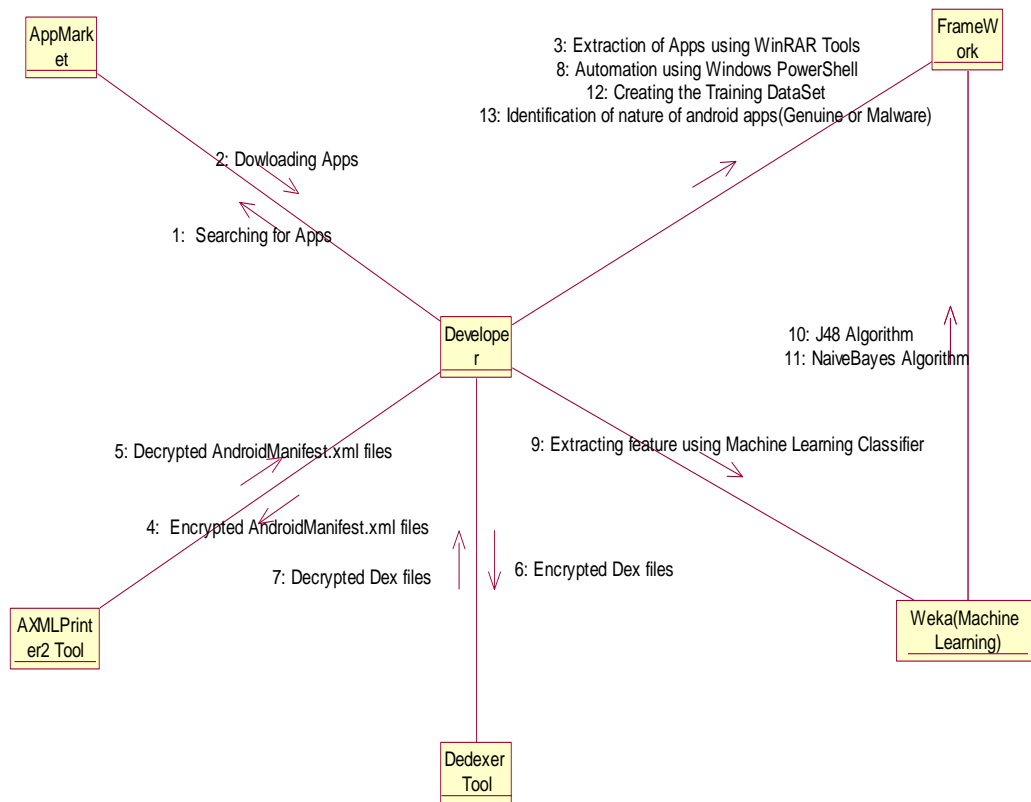


Figure 5.3.3.1 Android App Classifier Collaboration diagram

## **CHAPTER 6**

### **MODULES**

#### **6.1 INTRODUCTION**

In software, a module is a part of a program. Programs are composed of one or more independently developed modules that are not combined until the program is linked. A single module can contain one or several routines.

In hardware, a module is a self-contained component. Here, the software team makes the final decisions on all the features of the system. Group the system's features according to the types of visible operations they will perform. Identify modules, eg: by grouping features and/or tasks together. Identify the tasks that will implement the software features. Clearly define the role of each task in its module. Keep in mind that a feature may be implemented using several tasks, and conversely a single task may implement several features.

#### **6.2 MODULES**

Our proposed system is divided into five distinct modules described as follows:

- Data collection(apk files).
- Feature extraction.
  - Xml decryption.
  - Dex file decryption.
- Dataset creation.
- Training the Classifier Framework(using Training Data).
- Framework Testing(using Test Data)

## **6.3 MODULE DESCRIPTION**

### **6.3.1 DATA COLLECTION (APK FILES)**

Data collection is the process of gathering and measuring information on variables of interest, in an established systematic fashion that enables one to answer stated research questions, test hypotheses, and evaluate outcomes. The data collection component of research is common to all fields of study including physical and social sciences, humanities, business, etc. While methods vary by discipline, the emphasis on ensuring accurate and honest collection remains the same. The goal for all data collection is to capture quality evidence that then translates to rich data analysis and allows the building of a convincing and credible answer to questions that have been posed.

Regardless of the field of study or preference for defining data (quantitative, qualitative), accurate data collection is essential to maintaining the integrity of research. Both the selectionyhu7 of appropriate data collection instruments (existing, modified, or newly developed) and clearly delineated instructions for their correct use reduce the likelihood of errors occurring.

A formal data collection process is necessary as it ensures that data gathered are both defined and accurate and that subsequent decisions based on arguments embodied in the findings are valid. The process provides both a baseline from which to measure and in certain cases a target on what to improve.

Malware, short for malicious software, is any software used to disrupt computer operation, gather sensitive information, or gain access to private computer systems. Malware is defined by its malicious intent, acting against the requirements of the computer user, and does not include software that causes unintentional harm due to some deficiency. The term badware is sometimes used, and applied to both true (malicious) malware and unintentionally harmful software.

Malware may be stealthy, intended to steal information or spy on computer users for an extended period without their knowledge, as for example Regin, or it may be designed to cause harm, often as sabotage (e.g., Stuxnet), or to extort payment (CryptoLocker). 'Malware' is an umbrella term used to refer to a variety of forms of hostile or intrusive software, including computer viruses, worms, trojan horses, ransomware, spyware, adware, scareware, and other malicious programs. It can take the form of executable code, scripts, active content, and other software. Malware is often disguised as, or embedded in, non-malicious files. As of 2011 the majority of active malware threats were worms or trojans rather than viruses.

Here in this module we collected genuine apps from the Android AppMarket and the malware apps will not be available in the app store, so we downloaded it from a particular blog where we had given our usage of malware apps and also our base paper for their reference. They had given a particular login id and password where we had downloaded the malware apps.

we collected an average of about 182 genuine apps and 49 malware apps for this project.

### **6.3.2 FEATURE EXTRACTION**

In machine learning, pattern recognition and in image processing, feature extraction starts from an initial set of measured data and builds derived values (features) intended to be informative, non-redundant, facilitating the subsequent learning and generalization steps, in some cases leading to better human interpretations. Feature extraction is related to dimensionality reduction.

When the input data to an algorithm is too large to be processed and it is suspected to be redundant (e.g. the same measurement in both feet and meters, or the repetitiveness of images presented as pixels), then it can be transformed into a reduced set of features (also named features vector). This process is called feature extraction. The extracted features are expected to contain the relevant

information from the input data, so that the desired task can be performed by using this reduced representation instead of the complete initial data.

### 6.3.2.1 XML DECRYPTION

The extracted android Manifest.xml file from the apps would be in encrypted format in order to extract information from that we need to decrypt it using AXMLPrinter2 tool.

Encryption is the process of encoding messages or information in such a way that only authorized parties can read it. Encryption does not of itself prevent interception, but denies the message content to the interceptor. In an encryption scheme, the message or information, referred to as plaintext, is encrypted using an encryption algorithm, generating ciphertext that can only be read if decrypted. For technical reasons, an encryption scheme usually uses a pseudo-random encryption key generated by an algorithm. It is in principle possible to decrypt the message without possessing the key, but, for a well-designed encryption scheme, large computational resources and skill are required.

Every apk files consists of encrypted AndroidManifest.xml files which contains the app permissions. In order to extract those permissions from Android Manifest.xml files we need to decrypt it using AxmlPrinter2 tool.

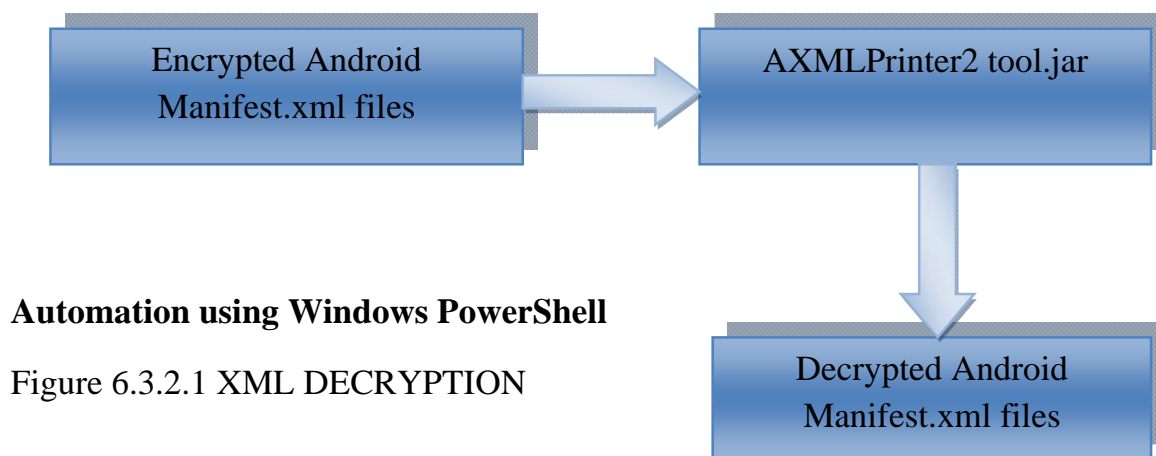


Figure 6.3.2.1 XML DECRYPTION

Windows PowerShell is a task automation and configuration management framework from Microsoft, consisting of a command-line shell and associated scripting language built on the .NET Framework. PowerShell provides full access to COM and WMI, enabling administrators to perform administrative tasks on both local and remote Windows systems as well as WS-Management and CIM enabling management of remote Linux systems and network devices.

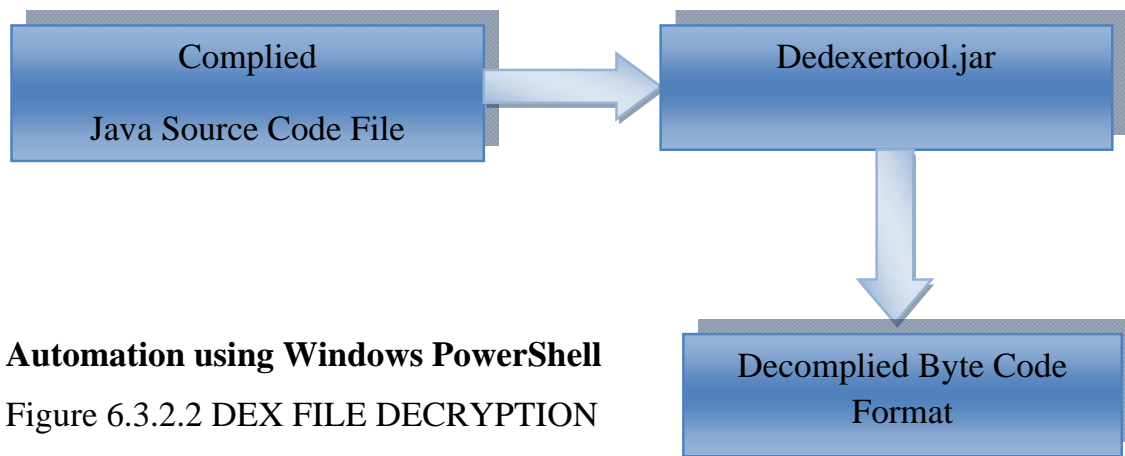
Windows PowerShell also provides a hosting API with which the Windows PowerShell runtime can be embedded inside other applications. These applications can then use Windows PowerShell functionality to implement certain operations, including those exposed via the graphical interface. This capability has been used by Microsoft Exchange Server 2007 to expose its management functionality as PowerShell cmdlets and providers and implement the graphical management tools as PowerShell hosts which invoke the necessary cmdlets. Other Microsoft applications including Microsoft SQL Server 2008 also expose their management interface via PowerShell cmdlets. With PowerShell, graphical interface-based management applications on Windows are layered on top of Windows PowerShell. A PowerShell scripting interface for Windows products is mandated by Microsoft's Common Engineering Criteria.

#### **6.3.2.2 DEX FILE DECRYPTION**

All Android application consists of its compiled java source code as 'dex' files To extract strings from the source code, we need to decompile it using Dedexer tool

Encryption is the process of encoding messages or information in such a way that only authorized parties can read it. Encryption does not of itself prevent interception, but denies the message content to the interceptor. In an encryption scheme, the message or information, referred to as plaintext, is encrypted using an encryption algorithm, generating ciphertext that can only be read if decrypted. For technical reasons, an encryption scheme usually uses a pseudo-random encryption key generated by an algorithm. It is in principle possible to decrypt the message without possessing the key, but, for a well-

designed encryption scheme, large computational resources and skill are required.



### Automation using Windows PowerShell

Figure 6.3.2.2 DEX FILE DECRYPTION

## 6.3.3 DATASET CREATION

### ARFF FILES

An ARFF (Attribute-Relation File Format) file is an ASCII text file that describes a list of instances sharing a set of attributes. ARFF files were developed by the Machine Learning Project at the Department of Computer Science of The University of Waikato for use with the Weka machine learning software. This document describes the version of ARFF used with Weka versions 3.2 to 3.3; this is an extension of the ARFF format as described in the data mining book written by Ian H. Witten and Eibe Frank (the new additions are string attributes, date attributes, and sparse instances).

The relation name is defined as the first line in the ARFF file. The format is:

**@relation <relation-name>**

where<relation-name> is a string. The string must be quoted if the name includes spaces.

Attribute declarations take the form of an ordered sequence of @attribute statements. Each attribute in the data set has its own @attribute statement which uniquely defines the name of that attribute and its data type. The order the attributes are declared indicates the column position in the data section of the file. For example, if an attribute is the third one declared then Weka expects that all that attribute's values will be found in the third comma delimited column.

ARFF file is standard input / output file format used by WEKA API. Both the training data and test data must be given as an ARFF file whose file structure is given below

### Syntax

**@ relation relation name**

**@ attribute attribute name**

**@ data**

.....

.....

.....

### 6.3.3.1 ARFF FILES ON PERMISSIONS

ARFF file on app permissions consists of kernel permission extracted from AndroidManifest.xml

In order to obtain them, we have implemented DOM XML Parser to parse each xml file. The data section of the ARFF will look like:

**@data**

**android.permission.WAKE\_LOCK,genuine**

**android.permission.INTERNET,genuine**

In the data segment given above, first segment consists of permission value and the second segment consists of class value whether it is genuine or malware



In case of test data class value will be given as '?', so that the classifier framework will automatically predict its nature.

### **6.3.3.2 ARFF FILES ON STRINGS**

ARFF file on strings consists of all the string values extracted from the source code

To obtain the strings, we have developed a text parser which iterates through every decompiled dex file and extracts the “const-string” values

The data segment of the ARFF file will look like:

**@data**

**'v0,\"BackStackEntry\\\",malware'**

**'v2,\"Commit: \\\",malware'**

**'v6,\"Bump nesting of \\\",malware**

Similar to ARFF on permissions, the first section of the data portion consists of the string value followed by its class value.

### **6.3.4 TRAINING THE CLASSIFIER FRAMEWORK(USING TRAINING DATASET)**

A training set is a set of data used in various areas of information science to discover potentially predictive relationships. Training sets are used in artificial intelligence, machine learning, genetic programming, intelligent systems, and statistics. In all these fields, a training set has much the same role and is often used in conjunction with a test set.

In artificial intelligence or machine learning, a training set consists of an input vector and an answer vector, and is used together with a supervised learning method to train a knowledge database (e.g. a neural net or a naive Bayes classifier) used by an AI machine. In statistical modeling, a training set is used to fit a model that can be used to predict a "response value" from one or more "predictors." The fitting can include both variable selection and parameter

estimation. Statistical models used for prediction are often called regression models, of which linear regression and logistic regression are two examples. In these fields, a major emphasis is placed on avoiding overfitting, so as to achieve the best possible performance on an independent test set that follows the same probability distribution as the training set.

#### **6.3.4.1 J48 (OR) C4.5 MACHINE LEARNING CLASSIFIER**

C4.5 is an algorithm used to generate a decision tree developed by Ross Quinlan. C4.5 is an extension of Quinlan's earlier ID3 algorithm. The decision trees generated by C4.5 can be used for classification, and for this reason, C4.5 is often referred to as a statistical classifier.

C4.5 builds decision trees from a set of training data in the same way as ID3, using the concept of information entropy. The training data is a set  $S = \{s_1, s_2, \dots\}$  of already classified samples. Each sample  $s_i$  consists of a  $p$ -dimensional vector  $(x_{1,i}, x_{2,i}, \dots, x_{p,i})$ , where the  $x_j$  represent attributes or features of the sample, as well as the class in which  $s_i$  falls.

At each node of the tree, C4.5 chooses the attribute of the data that most effectively splits its set of samples into subsets enriched in one class or the other. The splitting criterion is the normalized information gain (difference in entropy). The attribute with the highest normalized information gain is chosen to make the decision. The C4.5 algorithm then recurs on the smaller sublists.

All the samples in the list belong to the same class. When this happens, it simply creates a leaf node for the decision tree saying to choose that class.

None of the features provide any information gain. In this case, C4.5 creates a decision node higher up the tree using the expected value of the class. Instance of previously-unseen class encountered. Again, C4.5 creates a decision node higher up the tree using the expected value.

#### **6.3.4.2 NAIVEBAYES MACHINE LEARNING CLASSIFIER**

In machine learning, naive Bayes classifiers are a family of simple probabilistic classifiers based on applying Bayes' theorem with strong (naive) independence assumptions between the features.

Naive Bayes has been studied extensively since the 1950s. It was introduced under a different name into the text retrieval community in the early 1960s, and remains a popular (baseline) method for text categorization, the problem of judging documents as belonging to one category or the other (such as spam or legitimate, sports or politics, etc.) with word frequencies as the features. With appropriate preprocessing, it is competitive in this domain with more advanced methods including support vector machines. It also finds application in automatic medical diagnosis.

Naive Bayes classifiers are highly scalable, requiring a number of parameters linear in the number of variables (features/predictors) in a learning problem. Maximum-likelihood training can be done by evaluating a closed-form expression, which takes linear time, rather than by expensive iterative approximation as used for many other types of classifiers.

In the statistics and computer science literature, Naive Bayes models are known under a variety of names, including simple Bayes and independence Bayes. All these names reference the use of Bayes' theorem in the classifier's decision rule, but naive Bayes is not (necessarily) a Bayesian method; Russell and Norvig note that "[naive Bayes] is sometimes called a Bayesian classifier, a somewhat careless usage that has prompted true Bayesians to call it the idiot Bayes model."

### **6.3.5 FRAMEWORK TESTING (USING TEST DATA)**

Test data is data which has been specifically identified for use in tests, typically of a computer program.

Some data may be used in a confirmatory way, typically to verify that a given set of input to a given function produces some expected result. Other data may be used in order to challenge the ability of the program to respond to unusual, extreme, exceptional, or unexpected input.

Test data may be produced in a focused or systematic way (as is typically the case in domain testing), or by using other, less-focused approaches (as is typically the case in high-volume randomized automated tests). Test data may be produced by the tester, or by a program or function that aids the tester. Test data may be recorded for re-use, or used once and then forgotten.

## **CHAPTER 7**

### **IMPLEMENTATION AND TESTING**

#### **7.1 INTRODUCTION**

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub assemblies, assemblies and/or a finished product. It is the process of exercising software with the intent of ensuring that the software system meets its requirements and user expectation and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

#### **7.2 TYPES OF TEST**

**Unit testing** – Testing of individual software components or modules. Typically done by the programmer and not by testers, as it requires detailed knowledge of the internal program design and code. may require developing test driver modules or test harnesses.

- **Incremental integration testing** – Bottom up approach for testing i.e continuous testing of an application as new functionality is added; Application functionality and modules should be independent enough to test separately. done by programmers or by testers.
- **Integration testing** – Testing of integrated modules to verify combined functionality after integration. Modules are typically code modules, individual applications, client and server applications on a network, etc. This type of testing is especially relevant to client/server and distributed systems.
- **Functional testing** – This type of testing ignores the internal parts and focus on the output is as per requirement or not. Black-box type testing geared to functional requirements of an application.

- **System testing** – Entire system is tested as per the requirements. Black-box type testing that is based on overall requirements specifications, covers all combined parts of a system.
- **End-to-end testing** – Similar to system testing, involves testing of a complete application environment in a situation that mimics real-world use, such as interacting with a database, using network communications, or interacting with other hardware, applications, or systems if appropriate.
- **Acceptance testing**

Acceptance testing can mean one of two things:

1. A smoke test is used as an acceptance test prior to introducing a new build to the main testing process, i.e. before integration or regression
  2. Acceptance testing performed by the customer, often in their lab environment on their own HW, is known as user acceptance testing (UAT).
- Acceptance testing may be performed as part of the hand-off process between any two phases of development.

### **7.3 SDLC:**

SDLC is overall process of developing information on system through a multi-step process from investigation of initial requirements through analysis, design, implementation and maintenance.

It to describe a process for planning, creating, testing, and deploying an information system. The systems development life-cycle concept applies to a range of hardware and software configurations, as a system can be composed of hardware only, software only, or a combination

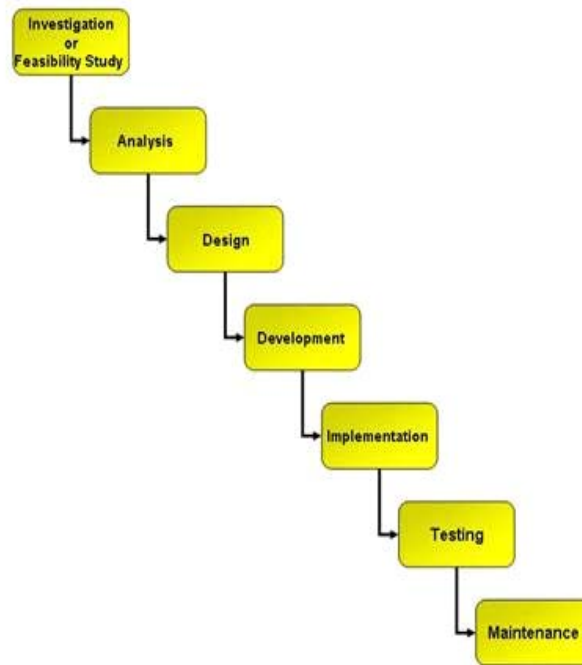


FIGURE 7.3 SOFTWARE DEVELOPMENT LIFE CYCLE

## **CHAPTER 8**

### **CONCLUSION**

In this system we propose a new method for detecting Android malware using string features to train machine-learning techniques. In order to validate our method, we collected several malware samples of Android applications. Then, we extracted the aforementioned features for each application and trained the models, evaluating each configuration. J48 and NaiveBayes was the best classifier obtaining very high accuracy levels. Nevertheless, there are several considerations regarding the viability of our approach.



## **CHAPTER 9**

### **FUTURE ENHANCEMENTS**

Our project is developed basically on android platform. We have kept in mind about the growing technologies and this project can be implement in Windows Phone as well as on IOS application. With the emergent of various Smartphone platforms, we can create and implement this application on any platforms since it is based on Java which emphasizes portability of codes and platform independency.

## APPENDIX-I

### SOURCE CODE

#### WINDOWS POWERSHELL SCRIPTS:

##### **ExtractingXMLFiles.ps1**

```
#to set the parent folder which contains the apks
$parent='D:\project\apks\malicious'
#to set the destination folder to save the encrypted xml files
$destination='D:\project\apks\malicious\exml'
#to recurse through the parent folder for files with extension .xml
$files= @()
Get-ChildItem $parent -Recurse -Filter "*.rar" | %{
$files=$files+$_.FullName
}
foreach($f in $files){
#to run the winrar command for unzipping
&'C:\Program Files\WinRaR\UnRaR.exe' x -or $f AndroidManifest.xml
$destination
}
```

##### **ExtractingDEXFiles.ps1**

```
#to set the parent folder which contains the apks\edex
$parent='D:\project\apks\rar'
#to set the destination folder to save the compiled dex files
$destination='D:\project\apks\edex'
#to recurse through the parent folder for files with extension .rar
$files= @()
Get-ChildItem $parent -Recurse -Filter "*.rar" | %{
$files=$files+$_.FullName
}
```

```

foreach($f in $files){
#to run the winrar command for unzipping
&'C:\Program Files\WinRaR\UnRaR.exe' x -or $f *.dex $destination
}

```

### **DecryptingXMLFiles.ps1**

```

#to set the parent folder which contains the encrypted xml files
$parent='D:\project\apks\exml'
#to set the destination folder to save the decrypted xml files
$destination='D:\project\apks\dexml\xml'
#To recurse through the parent folder for files which extension .xml
$files= @()
Get-ChildItem $parent -Recurse -Filter "*.xml" | %{
$files=$files+$_.FullName
}
foreach($f in $files){
#to run the AXMLPrinter2.jar
cmd.exe /c "java -jar AXMLPrinter2.jar $f >> $f.xml"
}

```

### **DecompilingDEXFiles.ps1**

```

#Setting the source folder which contains the dex files and to set the destination
folder to save the decompiled dex files
$parent='D:\project\apks\edex'
$destination='D:\project\apks\dedex'
#To loop through the parent folder for all the files which has the extension .dex
$files=Get-ChildItem $parent -Recurse -Filter "*.dex"
foreach($f in $files){
#To get the file name to create folders for each apk's source code

```

```

$fname=$f.Name.SubString(0);
if(-not (Test-Path $fname)){mkdir $fname};
$fullpath=$destination+"\\"+$fname
#To run the dedexer.jar file
cmd.exe /c "java -jar ddx.jar -o -D -r -d $fullpath $f"
}

```

## **JAVA SOURCE CODE:**

### **ClassifierFramework.java**

```

package androidappclassifier;

import featuresparser.txtparser;
import featuresparser.xmlparser;
import classifierbuilder.j48classifier;
import classifierbuilder.naivebayesclassifier;
import classifierbuilder.naivebayesupdateableclassifier;

public class classifierframework {
    public static void main(string[] args) throws exception {
        try {
            //to invoke the xml parser class
            xmlparserxp=new xmlparser();
            xp.permissionsparsing();
            //to invoke the text parser class
            txtparsertp=new txtparser();
            tp.stringparsing();
            //naive bayes algorithm invocation
            naivebayesclassifiernbc=new naivebayesclassifier();
            //j48 algorithm invocation

```

```

j48classifier j48c=new j48classifier();
//naivebayesupdateable algorithm invlocation
naivebayesupdateableclassifiernbs=new
naivebayesupdateableclassifier();
//cross validation results on permissions
nbc.naivebayesonpermissionswithcrossvalidation();
j48c.j48onpermissionswithcrossvalidation();
//individual testing results on permissions
nbc.naivebayesonenuinepermissionswithoutcrossvalidation();
nbc.naivebayesonmalwarepermissionswithoutcrossvalidation();
//cross validation results on strings
nbs.naivebayesupdateableonstringswithcrossvalidation();
nbc.naivebayesonstringswithcrossvalidation();
}
catch (exception e) {
e.printStackTrace();
}
}
}

```

### **XmlParser.java**

```

packagefeaturesparser;

importjava.io.File;
importjava.io.IOException;
//packages needed to parse the xml file using DOM parser
importjavax.xml.parsers.DocumentBuilder;
importjavax.xml.parsers.DocumentBuilderFactory;
importjavax.xml.parsers.ParserConfigurationException;

```

```

org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;
import arffgenerator.ARFFOnPermissions;

public class XmlParser
{
    public void permissionsParsing() throws Exception
    {
        DocumentBuilderFactory factory=DocumentBuilderFactory.newInstance();
        try
        {
            DocumentBuilder document=factory.newDocumentBuilder();
            for(File xmlFile: new File("Genuine/dexml").listFiles())
                //to parse the malicious app's xml files use the below for loop
                //for(File xmlFile: new File("Malware/dexml").listFiles())
            {
                //to get the file's path
                Document doc=document.parse(xmlFile.getAbsolutePath());
                //to search the tag name "uses-permission"
                NodeList permissionsList=doc.getElementsByTagName("uses-permission");
                //to set the string array's length to the number of permissions in the parsing xml
                //file
                String[] permissionValue=new String[permissionsList.getLength()];
                for(int i=0;i<permissionsList.getLength();i++)
                {
                    Node n=permissionsList.item(i);

```

```

//first to verify the node type
if(n.getNodeType()==Node.ELEMENT_NODE)
{
    //to cast the node type to element type since getAttribute() method accepts only
    element types
    Element permission=(Element) n;
    permissionValue[i]=permission.getAttribute("android:name");
}
}

//creating object for the class which contains the ARFF file generating method
ARFFOnPermissionsobj=new ARFFOnPermissions();
obj.arffBuilder(permissionValue);
}
}

catch (ParserConfigurationException e)
{
    e.printStackTrace();
}

catch (SAXException e)
{
    e.printStackTrace();
}

catch (IOException e)
{
    e.printStackTrace();
}

catch(Exception e)
{
    e.printStackTrace();
}

```

```
}  
}  
}
```

### **TxtParser.java**

```
package featuresparser;
```

```
//IO packages needed to read the dex files
```

```
import java.io.BufferedReader;
```

```
import java.io.File;
```

```
import java.io.FileNotFoundException;
```

```
import java.io.FileReader;
```

```
import java.io.IOException;
```

```
//packages needed for Regular Expression forming and matching with file  
contents
```

```
import java.util.regex.Matcher;
```

```
import java.util.regex.Pattern;
```

```
import arffgenerator.ARFFOnStrings;
```

```
public class TxtParser
```

```
{
```

```
public void stringParsing() throws IOException {
```

```
//Two Pattern and Matcher class objects are declared.
```

```
//One is for finding the string array length to store the values and other one is to  
parse the file
```

```
Pattern regExpPattern, regExpPattern1;
```

```
Matcher expMatcher, expMatcher1;
```

```
//Two BufferedReader objects to parse the file twice
```

```
BufferedReader dexFileReader, dexFileReader1;
```



```

//two string arrays-one for unfiltered and one for filtered string values
String[] stringValue,filteredStringValue;
for(File dexFile: new File("D:/project/apks/dedex").listFiles())
//to parse the malicious apps dex files use the below for loop
//for(File dexFile: new File("D:/project/apks/malicious/dedex").listFiles())
{
//variable needed to set the length of the string array
intstringInstancesCount=0,filteredInstancesCount=0;
inti=0,l=0;
try
{
dexFileReader=new BufferedReader(new FileReader(dexFile));
//two string variables to store the current line which is being read
String currentLine,currentLine1;
while((currentLine=dexFileReader.readLine())!=null)
{
//First to match the pattern "const-string" in order to find the no of occurrences
in the file
regExpPattern=Pattern.compile("const-string");
expMatcher=regExpPattern.matcher("");
expMatcher.reset(currentLine);
while(expMatcher.find())
{
//if the regex if matched, increment the counter variable to set the array size
++stringInstancesCount;
}
}
//string array size of being set to the value of counter variable
stringValue=new String[stringInstancesCount];

```

```

//dex file is being parsed for the second time to extract the const-string values
dexFileReader1=new BufferedReader(new FileReader(dexFile));
regExpPattern1=Pattern.compile("(?<=const-string).*");
expMatcher1=regExpPattern1.matcher("");
while((currentLine1=dexFileReader1.readLine())!=null)
{
expMatcher1.reset(currentLine1);
while(expMatcher1.find())
{
stringValue[i]=expMatcher1.group();
//to increment the i variable to move to the next index position of array
++i;
}
}
//to filter the unwanted and empty string values(for eg:string length less than 15)
//to set the length for filtered string array
for(int j=0;j<stringValue.length;j++)
{
if(stringValue[j].length()>15)
{
++filteredInstancesCount;
}
}
//to set the filtered string count to the string array
filteredStringValue=new String[filteredInstancesCount];
//to store the const-string values in the array
for(int k=0;k<stringValue.length;k++)
{
if(stringValue[k].length()>15)

```

```

{
    filteredStringValue[l]=stringValue[k];
    ++l;
}
}

//string array is passed to the method that generates the ARFF file.
ARFFOnStringsobj=new ARFFOnStrings();
obj.arffBuilder(filteredStringValue);
}
catch (FileNotFoundException e)
{
    e.printStackTrace();
}
catch (IOException e) {
    e.printStackTrace();
}
}
}
}
}
}
}

```

### **ARFFOnPermissions.java**

```

package arffgenerator;

```

```

//below are the WEKA API's packages needed to build ARFF file

```

```

import weka.core.Attribute;
import weka.core.FastVector;
import weka.core.Instance;
import weka.core.Instances;

```

```

//java.io packages needed to write output to a file
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

public class ARFFOnPermissions
{
    //declaration of variables and types needed for creating ARFF file using WEKA
    API
    FastVector attributes;
    FastVector classAttribute;
    Instances data;
    double[] dataValue;
    BufferedWriter arffWriter;
    static int count=1;
    public ARFFOnPermissions()
    {
        // To set up the attributes
        attributes=new FastVector();
        // To define the permissions attribute
        attributes.addElement(new Attribute("permissionvalue",(FastVector)null));
        // To define the class attribute
        classAttribute=new FastVector(2);
        // Two class attribute value-genuine and malware
        classAttribute.addElement("genuine");
        classAttribute.addElement("malware");
        // adding the class attribute to dataset
        attributes.addElement(new Attribute("class",classAttribute));
        // To create the instances object

```

```

data=new Instances("AndroidAppClassifier",attributes,0);
}

public void arffBuilder(String[] permissions) throws IOException {
for(int i=0;i<permissions.length;i++)
{
//it is essential to initialize double array everytime when the method is called
dataValue=new double[data.numAttributes()];
dataValue[0]=data.attribute(0).addStringValue(permissions[i]);
dataValue[1]=classAttribute.indexOf("genuine");
//use the below line to set class option as malware for malicious apps
//dataValue[1]=classAttribute.indexOf("malware");
data.add(new Instance(1.0,dataValue));
}
arffWriter=new BufferedWriter(new FileWriter("genuine"+count+".arff"));
//use the below line to name the files for malicious apps
//arffWriter=new BufferedWriter(new FileWriter("malicious"+count+".arff"));
arffWriter.write(data.toString());
arffWriter.flush();
++count;
}
}

```

### **ARFFOnStrings.java**

```

package arffgenerator;

```

```

//below are the WEKA API's packages needed to build ARFF file
import weka.core.Attribute;
import weka.core.FastVector;

```

```

import weka.core.Instance;
import weka.core.Instances;

//java.io packages needed to write output to a file
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

public class ARFFOnStrings
{
    //declaration of variables and types needed for creating ARFF file using WEKA
    FastVector attributes;
    FastVector classAttribute;
    Instances data;
    double[] dataValue;
    BufferedWriter arffWriter;
    static int count=1;
    public ARFFOnStrings()
    {
        // To set up the attributes
        attributes=new FastVector();
        // To define the permissions attribute
        attributes.addElement(new Attribute("conststringValue",(FastVector)null));
        // To define the class attribute
        classAttribute=new FastVector(2);
        // Two class attribute value-genuine and malware
        classAttribute.addElement("genuine");
        classAttribute.addElement("malware");
        // adding the class attribute to dataset
    }
}

```

```

attributes.addElement(new Attribute("class",classAttribute));
// To create the instances object
data=new Instances("AndroidAppClassifier",attributes,0);
}
public void arffBuilder(String[] stringVal) throws IOException
{
for(inti=0;i<stringVal.length;i++)
{
//it is essential to initialize double array everytime when the method is called
dataValue=new double[data.numAttributes()];
dataValue[0]=data.attribute(0).addStringValue(stringVal[i]);
dataValue[1]=classAttribute.indexOf("genuine");
//use the below line to set class option as malware for malicious apps
//dataValue[1]=classAttribute.indexOf("malware");
data.add(new Instance(1.0,dataValue));
}
arffWriter=new BufferedWriter(new FileWriter("genuine"+count+".arff"));
//use the below line to name the files for malicious apps
//arffWriter=new BufferedWriter(new FileWriter("malicious"+count+".arff"));
arffWriter.write(data.toString());
arffWriter.flush();
++count;
}
}

```

### **NaiveBayesClassifier.java**

```
packageclassifierbuilder;
```

```
importjava.io.BufferedReader;
```

```

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Random;

import weka.classifiers.Evaluation;
import weka.classifiers.bayes.NaiveBayes;
import weka.core.Instances;
import weka.filters.*;
import weka.filters.unsupervised.attribute.*;

public class NaiveBayesClassifier {
    static int count=1, count2=1;;

    public void naiveBayesOnPermissionsWithCrossValidation() throws
    Exception, FileNotFoundException, IOException
    {
        BufferedReader datasetReader=null;
        try {
            datasetReader=new BufferedReader(new
            FileReader("Datasets/CrossEvaluationDatasetForPermissions.arff"));
            //to load the dataset to the trainer
            Instances classifierTrainer=new Instances(datasetReader);
            //to convert string attribute to numeric(naive bayes cannot accept string
            attributes)
            StringToWordVector filter=new StringToWordVector();

```



```

filter.setInputFormat(classifierTrainer);
Instances filteredTrainer=Filter.useFilter(classifierTrainer, filter);
//to set the class index-class attribute will be put first by the
StringToWordVector class
filteredTrainer.setClassIndex(0);
datasetReader.close();
//to write the numeric type arff file
//BufferedWriternominalTypeWriter=new BufferedWriter(new
FileWriter("nominal.arff"));
//nominalTypeWriter.write(filteredTrainer.toString());
//to instantiate the NaiveBayesweka class
NaiveBayes classifier=new NaiveBayes();
//to build the classifier with the dataset
classifier.buildClassifier(filteredTrainer);
//to define the evaluation model
Evaluation evaluate=new Evaluation(filteredTrainer);
//to set the cross validation characteristics
evaluate.crossValidateModel(classifier, filteredTrainer, 15, new Random(1));
//System.out.println(evaluate.toSummaryString("\n The classification
results\n",true));
//to write the output to a log file
BufferedWriteroutputWriter=new BufferedWriter(new
FileWriter("OutputStatistics/Demo/CrossValidationResults/NaiveBayesCrossV
alidationResultsOnPermissions.log"));
outputWriter.write(evaluate.toSummaryString());
outputWriter.close();
} catch (Exception e) {
e.printStackTrace();
}

```

```

}

public void naiveBayesOnGenuinePermissionsWithoutCrossValidation()
throws Exception,FileNotFoundException,IOException
{
    BufferedReaderdataReader=null;
    //training data
    dataReader=new BufferedReader(new
    FileReader("Datasets/TrainingDataset2.arff"));
    Instances classifierTrainer=new Instances(dataReader);
    //to convert string attribute to numeric(naive bayes cannot accept string
    attributes)
    StringToWordVector filter=new StringToWordVector();
    filter.setInputFormat(classifierTrainer);
    Instances filteredTrainer=Filter.useFilter(classifierTrainer, filter);
    //to set the class index-class attribute will be put first by the
    StringToWordVector class
    filteredTrainer.setClassIndex(0);
    NaiveBayes classifier=new NaiveBayes();
    classifier.buildClassifier(filteredTrainer);
    //test data
    for(File testFile: new File("Genuine/ARFFOnPermissionsGenuine").listFiles())
    {
        dataReader=new BufferedReader(new FileReader(testFile));
        Instances classifierTester=new Instances(dataReader);
        //to convert string attribute to numeric(naive bayes cannot accept string
        attributes)
        StringToWordVectorfilterTester=new StringToWordVector();
        filterTester.setInputFormat(classifierTester);
        Instances filteredTester=Filter.useFilter(classifierTester, filterTester);
    }
}

```

```

//to set the class index-class attribute will be put first by the
StringToWordVector class
filteredTester.setClassIndex(0);
//evaluate the model
Evaluation evaluate=new Evaluation(filteredTrainer);
evaluate.evaluateModel(classifier,filteredTester);
//System.out.println(evaluate.toSummaryString("\n The test results\n",false));
BufferedWriteroutputWriter=new BufferedWriter(new
FileWriter("OutputStatistics/Demo/IndividualTesting/GenuineTesting/"+"Genui
neApp"+count+".log"));
outputWriter.write(evaluate.toSummaryString("\n The Test results\n",false));
outputWriter.flush();
++count;
}
dataReader.close();
}
public void naiveBayesOnMalwarePermissionsWithoutCrossValidation()
throws Exception,FileNotFoundException,IOException
{
BufferedReaderdataReader=null;
//training data
dataReader=new BufferedReader(new
FileReader("Datasets/TrainingDataset.arff"));
Instances classifierTrainer=new Instances(dataReader);
//to convert string attribute to numeric(naive bayes cannot accept string
attributes)
StringToWordVector filter=new StringToWordVector();
filter.setInputFormat(classifierTrainer);
Instances filteredTrainer=Filter.useFilter(classifierTrainer, filter);

```

```

//to set the class index-class attribute will be put first by the
StringToWordVector class
filteredTrainer.setClassIndex(0);
NaiveBayes classifier=new NaiveBayes();
classifier.buildClassifier(filteredTrainer);
//test data
for(File testFile: new File("Malware/ARFFOnPermissionsMalware").listFiles())
{
dataReader=new BufferedReader(new FileReader(testFile));
Instances classifierTester=new Instances(dataReader);
//to convert string attribute to numeric(naive bayes cannot accept string
attributes)
StringToWordVectorfilterTester=new StringToWordVector();
filterTester.setInputFormat(classifierTester);
Instances filteredTester=Filter.useFilter(classifierTester, filterTester);
//to set the class index-class attribute will be put first by the
StringToWordVector class
filteredTester.setClassIndex(0);
//evaluate the model
Evaluation evaluate=new Evaluation(filteredTrainer);
evaluate.evaluateModel(classifier,filteredTester);
//System.out.println(evaluate.toSummaryString("\n The test results\n",false));
BufferedWriteroutputWriter=new BufferedWriter(new
FileWriter("OutputStatistics/Demo/IndividualTesting/MalwareTesting"+"Malw
areApp"+count2+".log"));
outputWriter.write(evaluate.toSummaryString("\n The Test results\n",false));
outputWriter.flush();
++count2;
}

```

```

dataReader.close();
}
public void naiveBayesOnStringsWithCrossValidation() throws
Exception,FileNotFoundException,IOException
{
BufferedReaderdatasetReader=null;
try {
datasetReader=new BufferedReader(new
FileReader("Datasets/CrossEvaluationDatasetForStrings.arff"));
//to load the dataset to the trainer
Instances classifierTrainer=new Instances(datasetReader);
//to convert string attribute to numeric(naive bayes cannot accept string
attributes)
StringToWordVector filter=new StringToWordVector();
filter.setInputFormat(classifierTrainer);
Instances filteredTrainer=Filter.useFilter(classifierTrainer, filter);
//to set the class index-class attribute will be put first by the
StringToWordVector class
filteredTrainer.setClassIndex(0);
datasetReader.close();
//to write the numeric type arff file
//BufferedWriternominalTypeWriter=new BufferedWriter(new
FileWriter("nominal.arff"));
//nominalTypeWriter.write(filteredTrainer.toString());
//to instantiate the NaiveBayesweka class
NaiveBayes classifier=new NaiveBayes();
//to build the classifier with the dataset
classifier.buildClassifier(filteredTrainer);
//to define the evaluation model

```

```

Evaluation evaluate=new Evaluation(filteredTrainer);
//to set the cross validation characteristics
evaluate.crossValidateModel(classifier, filteredTrainer, 15, new Random(1));
//System.out.println(evaluate.toSummaryString("\n The classification
results\n",true));
//to write the output to a log file
BufferedWriteroutputWriter=new BufferedWriter(new
FileWriter("OutputStatistics/CrossValidationResults/NaiveBayesCrossValidatio
nResultsOnStrings.log"));
outputWriter.write(evaluate.toSummaryString());
outputWriter.close();
} catch (Exception e) {
e.printStackTrace();
}
}
}

```

### **J48Classifier.java**

```

packageclassifierbuilder;

importjava.io.BufferedReader;
importjava.io.BufferedWriter;
importjava.io.File;
importjava.io.FileNotFoundException;
importjava.io.FileReader;
importjava.io.FileWriter;
importjava.io.IOException;
importjava.util.Random;

```

```

import weka.classifiers.Evaluation;
import weka.classifiers.bayes.NaiveBayes;
import weka.classifiers.trees.J48;
import weka.core.Instances;
import weka.filters.Filter;
import weka.filters.unsupervised.attribute.StringToWordVector;

public class J48Classifier {
    static int count=1;

    public void j48OnPermissionsWithCrossValidation() throws
    Exception,FileNotFoundException,IOException
    {
        try
        {
            BufferedReader inputReader=null;
            inputReader=new BufferedReader(new
            FileReader("Datasets/CrossEvaluationDatasetForPermissions.arff"));
            //to load the dataset to the trainer
            Instances classifierTrainer=new Instances(inputReader);
            //to convert string attribute to numeric(naive bayes cannot accept string
            attributes)
            StringToWordVector filter=new StringToWordVector();
            filter.setInputFormat(classifierTrainer);
            Instances filteredTrainer=Filter.useFilter(classifierTrainer, filter);
            //to set the class index-class attribute will be put first by the
            StringToWordVector class
            filteredTrainer.setClassIndex(0);
            inputReader.close();

```

```

J48 classifier=new J48();
//build the J48 classifier
classifier.buildClassifier(filteredTrainer);
//to define the evaluation model
Evaluation evaluate=new Evaluation(filteredTrainer);
//to set the cross validation characteristics
evaluate.crossValidateModel(classifier, filteredTrainer, 15, new Random(1));
//System.out.println(evaluate.toSummaryString("\n The classification
results\n",true));
BufferedWriteroutputWriter=new BufferedWriter(new
FileWriter("OutputStatistics/Demo/CrossValidationResults/J48CrossValidation
ResultsOnPermissions.log"));
outputWriter.write(evaluate.toSummaryString());
outputWriter.close();
}
catch (Exception e) {
e.printStackTrace();
}
}

public void j48OnPermissionsWithoutCrossValidation() throws
Exception,FileNotFoundException,IOException
{
BufferedReaderdataReader=null;
//training data
dataReader=new BufferedReader(new
FileReader("Datasets/TrainingDataset2.arff"));
Instances classifierTrainer=new Instances(dataReader);
//to convert string attribute to numeric(naive bayes cannot accept strings)

```



```

StringToWordVector filter=new StringToWordVector();
filter.setInputFormat(classifierTrainer);
Instances filteredTrainer=Filter.useFilter(classifierTrainer, filter);
//to set the class index-class attribute will be put first by the
StringToWordVector class
filteredTrainer.setClassIndex(0);
J48 classifier=new J48();
classifier.buildClassifier(filteredTrainer);
//test data
for(File testFile: new File("Genuine/ARFFOnPermissionsGenuine").listFiles())
{
dataReader=new BufferedReader(new FileReader(testFile));
Instances classifierTester=new Instances(dataReader);
//to convert string attribute to numeric(naive bayes cannot accept string
attributes)
StringToWordVectorfilterTester=new StringToWordVector();
filterTester.setInputFormat(classifierTester);
Instances filteredTester=Filter.useFilter(classifierTester, filterTester);
//to set the class index-class attribute will be put first by the
StringToWordVector class
filteredTester.setClassIndex(0);
//evaluate the model
Evaluation evaluate=new Evaluation(filteredTrainer);
evaluate.evaluateModel(classifier,filteredTester);
//System.out.println(evaluate.toSummaryString("\n The test results\n",false));
BufferedWriteroutputWriter=new BufferedWriter(new
FileWriter("OutputStatistics/IndividualTesting/"+ "app"+count+".log"));
outputWriter.write(evaluate.toSummaryString("\n The Test results\n",false));
outputWriter.flush();

```

```

++count;
}
dataReader.close();
}
}

```

### **NaiveBayesUpdateableClassifier.java**

```

package classifierbuilder;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Random;

import weka.classifiers.Evaluation;
import weka.classifiers.bayes.NaiveBayesUpdateable;
import weka.core.Instances;
import weka.filters.*;
import weka.filters.unsupervised.attribute.*;

public class NaiveBayesUpdateableClassifier {
    public void naiveBayesUpdateableOnStringsWithCrossValidation() throws
    Exception, FileNotFoundException, IOException
    {
        BufferedReader datasetReader = null;

```

```

try {
datasetReader=new BufferedReader(new
FileReader("Datasets/CrossEvaluationDatasetForStrings.arff"));
//to load the dataset to the trainer
Instances classifierTrainer=new Instances(datasetReader);
//to convert string attribute to numeric(naive bayes cannot accept string
attributes)
StringToWordVector filter=new StringToWordVector();
filter.setInputFormat(classifierTrainer);
Instances filteredTrainer=Filter.useFilter(classifierTrainer, filter);
//to set the class index-class attribute will be put first by the
StringToWordVector class
filteredTrainer.setClassIndex(0);
datasetReader.close();
//to write the numeric type arff file
//BufferedWriternominalTypeWriter=new BufferedWriter(new
FileWriter("nominal.arff"));
//nominalTypeWriter.write(filteredTrainer.toString());
//to instantiate the NaiveBayesweka class
NaiveBayesUpdateable classifier=new NaiveBayesUpdateable();
//to build the classifier with the dataset
classifier.buildClassifier(filteredTrainer);
//to define the evaluation model
Evaluation evaluate=new Evaluation(filteredTrainer);
//to set the cross validation characteristics
evaluate.crossValidateModel(classifier, filteredTrainer, 15, new Random(1));
//System.out.println(evaluate.toSummaryString("\n The classification
results\n",true));
//to write the output to a log file

```

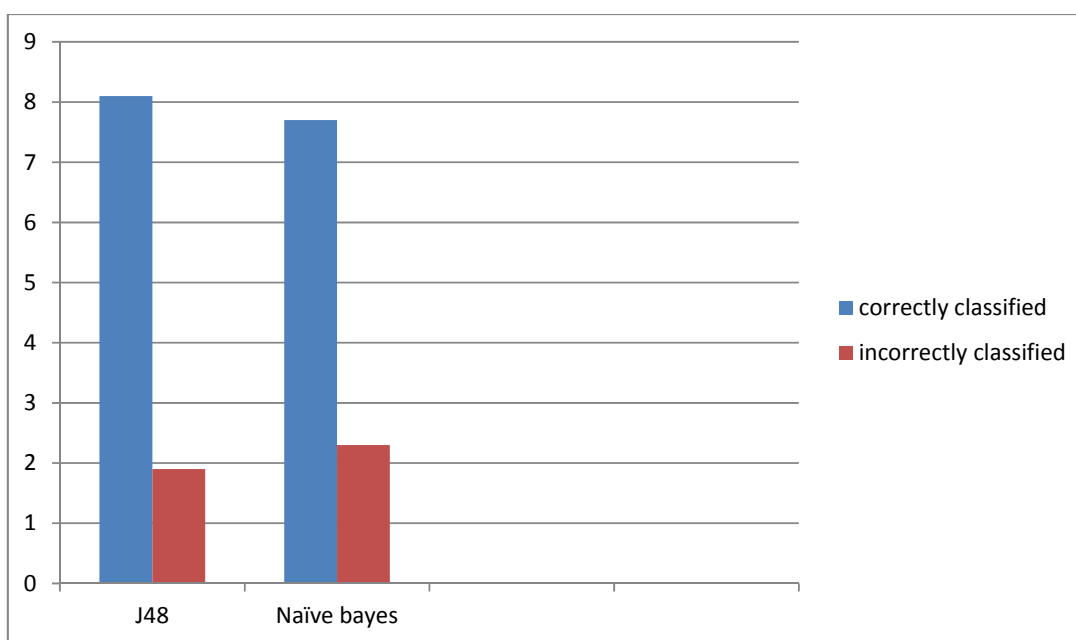
```
BufferedWriteroutputWriter=new BufferedWriter(new
FileWriter("OutputStatistics/CrossValidationResults/NaiveBayesSimpleCrossV
alidationResultsOnStrings.log"));
outputWriter.write(evaluate.toSummaryString());
outputWriter.close();
} catch (Exception e) {
e.printStackTrace();
}
}
}
```

## APPENDIX-II

### OUTPUT AND RESULTS

#### CROSS VALIDATION RESULTS ON PERMISSIONS

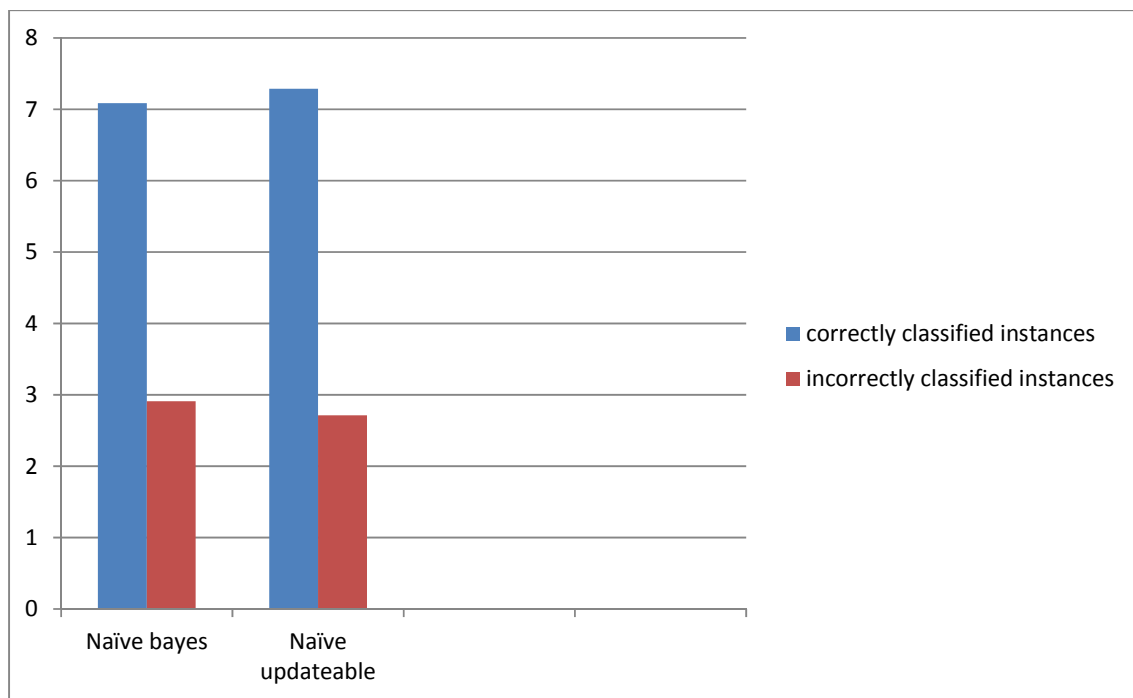
Algorithm	Correctly classified instances	Correctly classified instances (%)	Incorrectly classified instances	Incorrectly classified instances (%)
J48	2384	80.6768	571	19.3232
Naïve bayes	2298	77.7665	657	22.2335



#### CROSS VALIDATION RESULTS ON PERMISSIONS

## CROSS VALIDATION RESULTS FOR STRINGS

Algorithm	Correctly classified instances	Correctly classified instances (%)	Incorrectly classified instances	Incorrectly classified instances (%)
Naïve Bayes	268473	70.8757	110321	29.1243
NaïveUpdateable	270473	72.8757	108321	27.1243



## CROSS VALIDATION RESULTS ON STRINGS

## **INDIVIDUAL MALWARE APPLICATION TESTING RESULTS**

Algorithm	Correctly classified instances	Correctly classified instances (%)	Incorrectly classified instances	Incorrectly classified instances (%)
Naïve bayes	1040	86.4986	224	13.5014

## **CLASSIFIED MALWARE SAMPLE RESULTS**

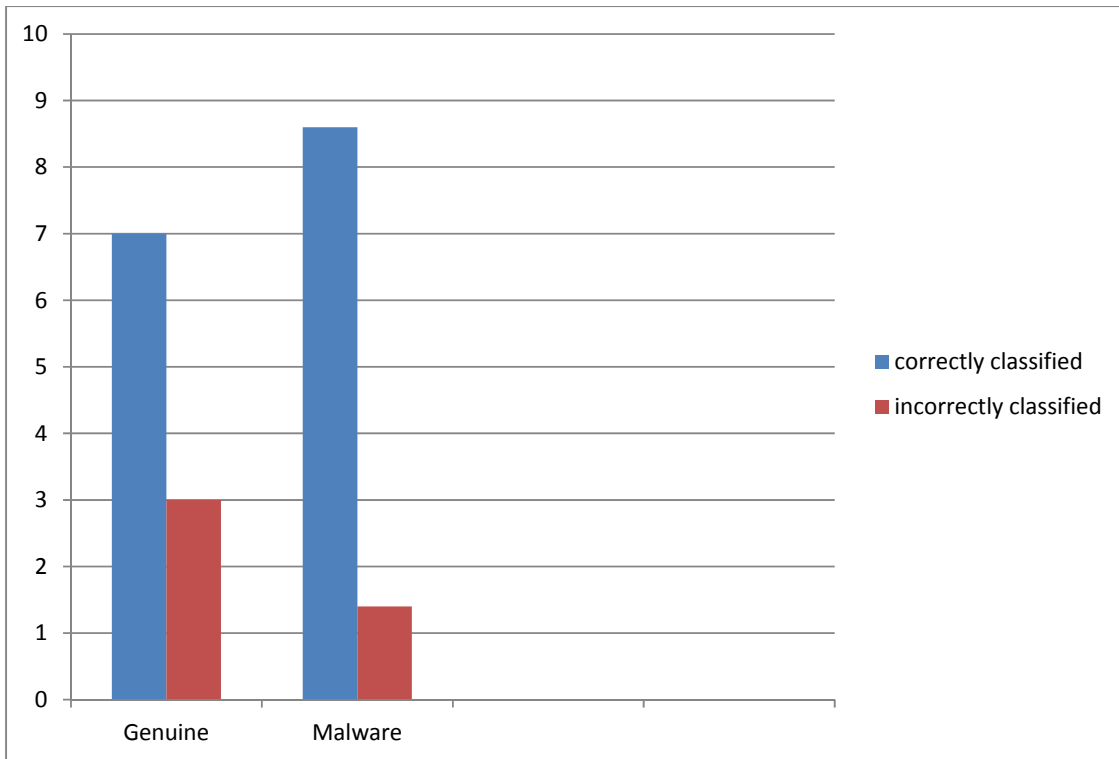
Algorithm	Total samples	Correctly classified samples	Incorrectly classified samples
Naïve bayes	49	42	7

## **INDIVIDUAL GENUINE APPLICATION TESTING RESULTS**

Algorithm	Correctly classified instances	Correctly classified instances (%)	Incorrectly classified instances	Incorrectly classified instances (%)
Naïve bayes	1552	70.0183	805	29.9817

## **CLASSIFIED GENUINE SAMPLES**

Algorithm	Total samples	Correctly classified samples	Incorrectly classified samples
Naïve bayes	105	77	28



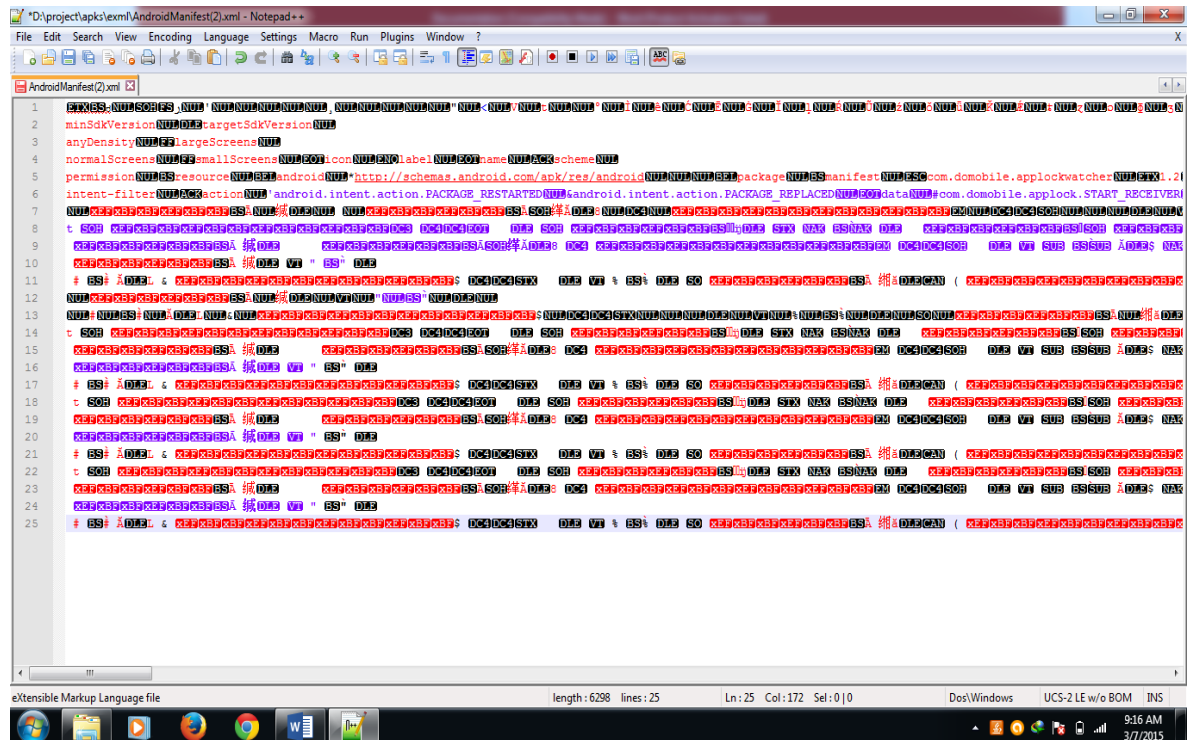
## INDIVIDUAL APP TESTING RESULTS



## APPENDIX-III

### SCREENSHOTS

#### ENCRYPTED ANDROIDMANIFEST.XML



The screenshot shows a Notepad++ window titled "D:\project\apk\exam\AndroidManifest(2).xml - Notepad++". The file content is an AndroidManifest.xml file where the XML tags and attributes have been replaced with a series of redacted characters, primarily 'S' and 't', making the file unreadable. The visible text includes:

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3         package="com.domobile.applclockwatcher"
4         android:versionCode="1"
5         android:versionName="1.0"
6         android:targetSdkVersion="11"
7         android:installLocation="auto">
8     <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED">
9     </uses-permission>
10    <application
11        android:label="@string/app_name"
12        android:icon="@mipmap/ic_launcher"
13        android:allowBackup="true"
14        android:fullBackupContent="true"
15        android:usesCleartextTraffic="true">
16        <activity
17            android:name=".MainActivity"
18            android:label="@string/app_name"
19            android:launchMode="singleTop"
20            android:theme="@style/AppTheme"
21            android:exported="true">
22            <intent-filter>
23                <action android:name="android.intent.action.MAIN">
24                </action>
25                <category android:name="android.intent.category.LAUNCHER">
26                </category>
27            </intent-filter>
28        </activity>
29    </application>
30 </manifest>
```

The status bar at the bottom indicates the file is an "eXtensible Markup Language file" with a length of 6298, 25 lines, and 172 columns. The system clock shows 9:16 AM on 3/7/2015.

FIGURE A

#### EXPLANATION:

The screenshot given above shows the AndroidManifest.xml file in encrypted format. This file contains the permissions used by the Android application.

## DECRYPTED ANDROIDMANIFEST.XML:

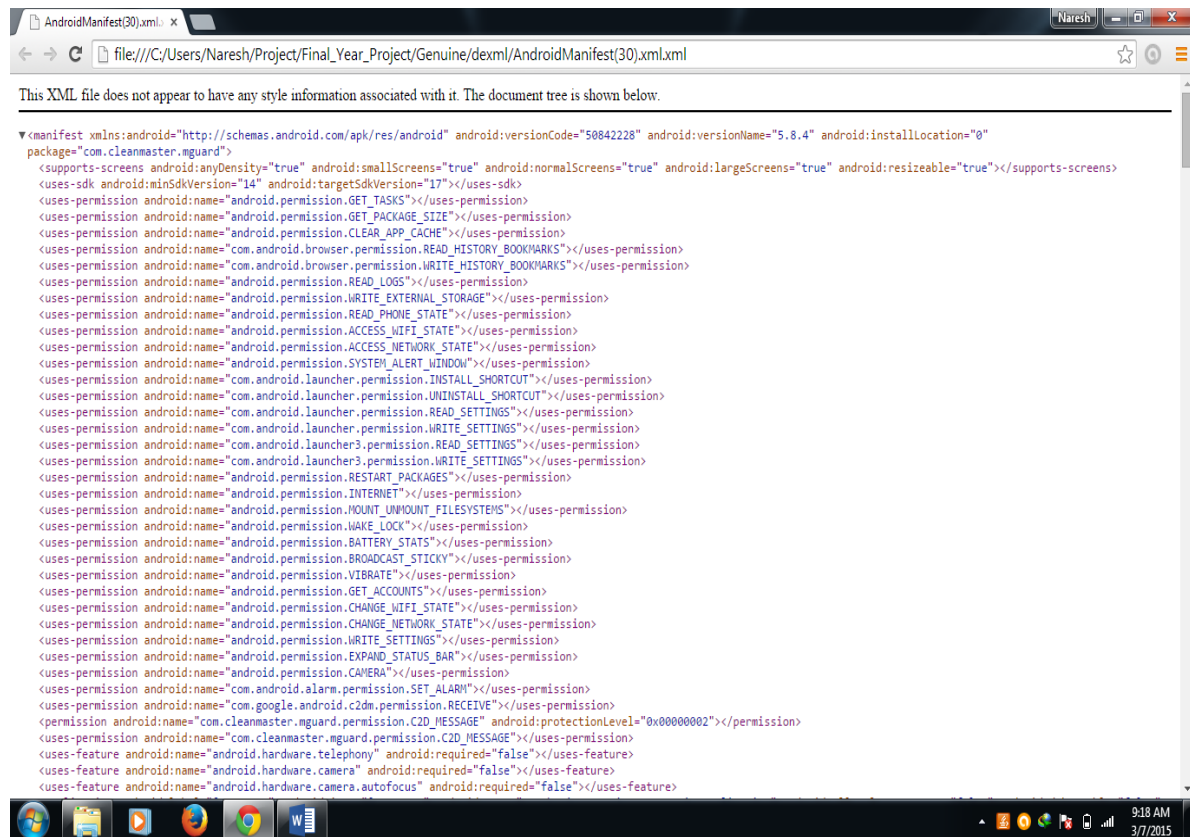


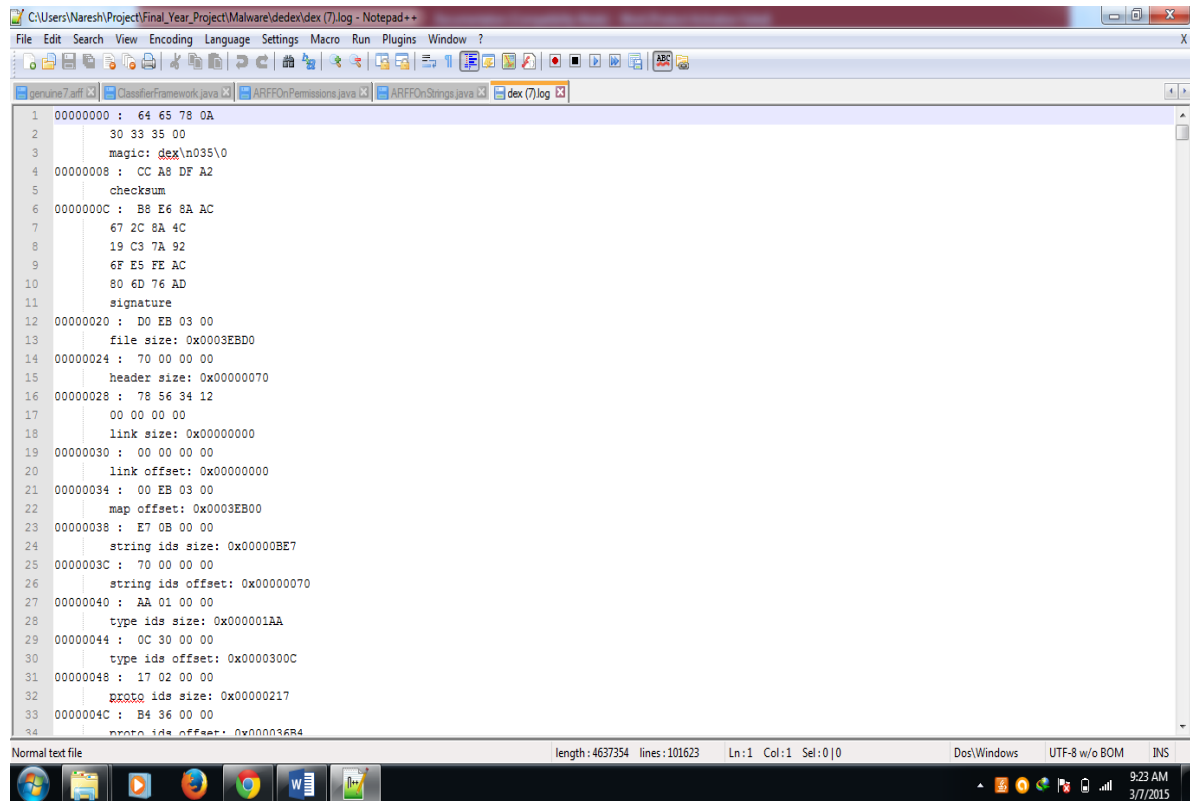
FIGURE B

This screenshot is the decrypted form of AndroidManifest.xml file after processing the file using AXMLPrinter2.jar

**EXPLANATION:**

82

## DECOMPILED DEX FILE:



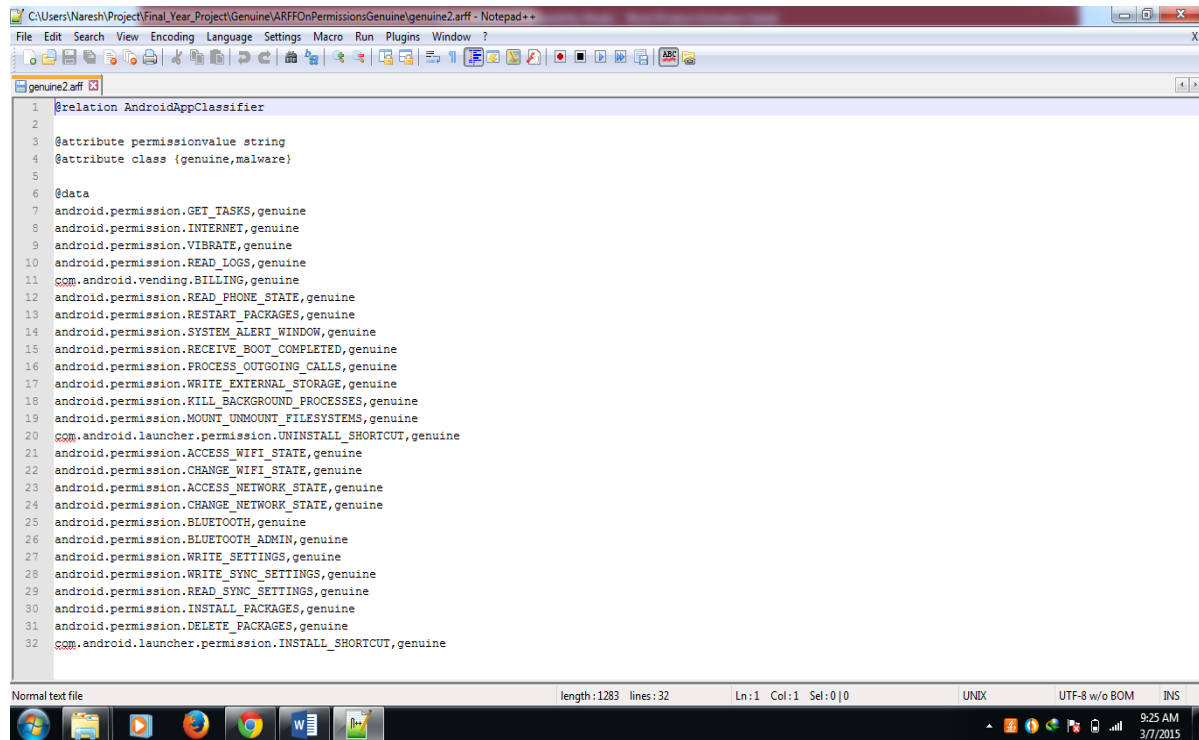
```
1 00000000 : 64 65 78 0A
2      30 33 35 00
3      magic: dex\n035\0
4 00000008 : CC A8 DF A2
5      checksum
6 0000000C : B8 E6 8A AC
7      67 2C 8A 4C
8      19 C3 7A 92
9      6F E5 FE AC
10     80 6D 76 AD
11     signature
12 00000020 : D0 EB 03 00
13     file size: 0x0003EBD0
14 00000024 : 70 00 00 00
15     header size: 0x00000070
16 00000028 : 78 56 34 12
17     00 00 00 00
18     link size: 0x00000000
19 00000030 : 00 00 00 00
20     link offset: 0x00000000
21 00000034 : 00 EB 03 00
22     map offset: 0x0003EB00
23 00000038 : E7 0B 00 00
24     string ids size: 0x00000BE7
25 0000003C : 70 00 00 00
26     string ids offset: 0x00000070
27 00000040 : AA 01 00 00
28     type ids size: 0x000001AA
29 00000044 : 0C 30 00 00
30     type ids offset: 0x0000300C
31 00000048 : 17 02 00 00
32     string ids size: 0x00000217
33 0000004C : B4 36 00 00
34     type ids offset: 0x000006R4
```

FIGURE D

## EXPLANATION:

The above screenshot shows the decompiled dex file using Dedexer.jar

## GENERATED ARFF FILE ON GENUINE PERMISSIONS:



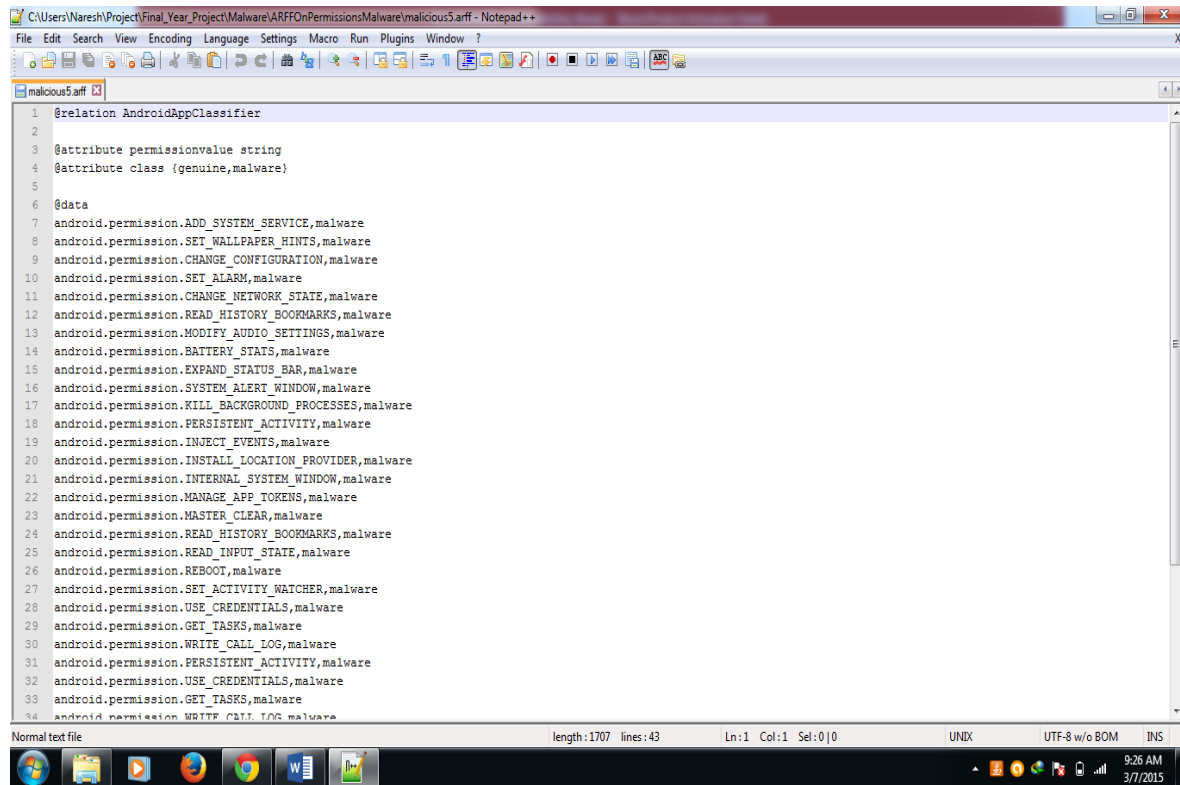
```
1 @relation AndroidAppClassifier
2
3 @attribute permissionvalue string
4 @attribute class {genuine,malware}
5
6 @data
7 android.permission.GET_TASKS,genuine
8 android.permission.INTERNET,genuine
9 android.permission.VIBRATE,genuine
10 android.permission.READ_LOGS,genuine
11 com.android.vending.BILLING,genuine
12 android.permission.READ_PHONE_STATE,genuine
13 android.permission.RESTART_PACKAGES,genuine
14 android.permission.SYSTEM_ALERT_WINDOW,genuine
15 android.permission.RECEIVE_BOOT_COMPLETED,genuine
16 android.permission.PROCESS_OUTGOING_CALLS,genuine
17 android.permission.WRITE_EXTERNAL_STORAGE,genuine
18 android.permission.KILL_BACKGROUND_PROCESSES,genuine
19 android.permission.MOUNT_UNMOUNT_FILESYSTEMS,genuine
20 com.android.launcher.permission.UNINSTALL_SHORTCUT,genuine
21 android.permission.ACCESS_WIFI_STATE,genuine
22 android.permission.CHANGE_WIFI_STATE,genuine
23 android.permission.ACCESS_NETWORK_STATE,genuine
24 android.permission.CHANGE_NETWORK_STATE,genuine
25 android.permission.BLUETOOTH,genuine
26 android.permission.BLUETOOTH_ADMIN,genuine
27 android.permission.WRITE_SETTINGS,genuine
28 android.permission.WRITE_SYNC_SETTINGS,genuine
29 android.permission.READ_SYNC_SETTINGS,genuine
30 android.permission.INSTALL_PACKAGES,genuine
31 android.permission.DELETE_PACKAGES,genuine
32 com.android.launcher.permission.INSTALL_SHORTCUT,genuine
```

FIGURE E

## EXPLANATION:

The above screenshot shows the ARFF file generated using Weka packages for permissions used by the genuine Android applications

## GENERATED ARFF FILE ON MALWARE PERMISSIONS:



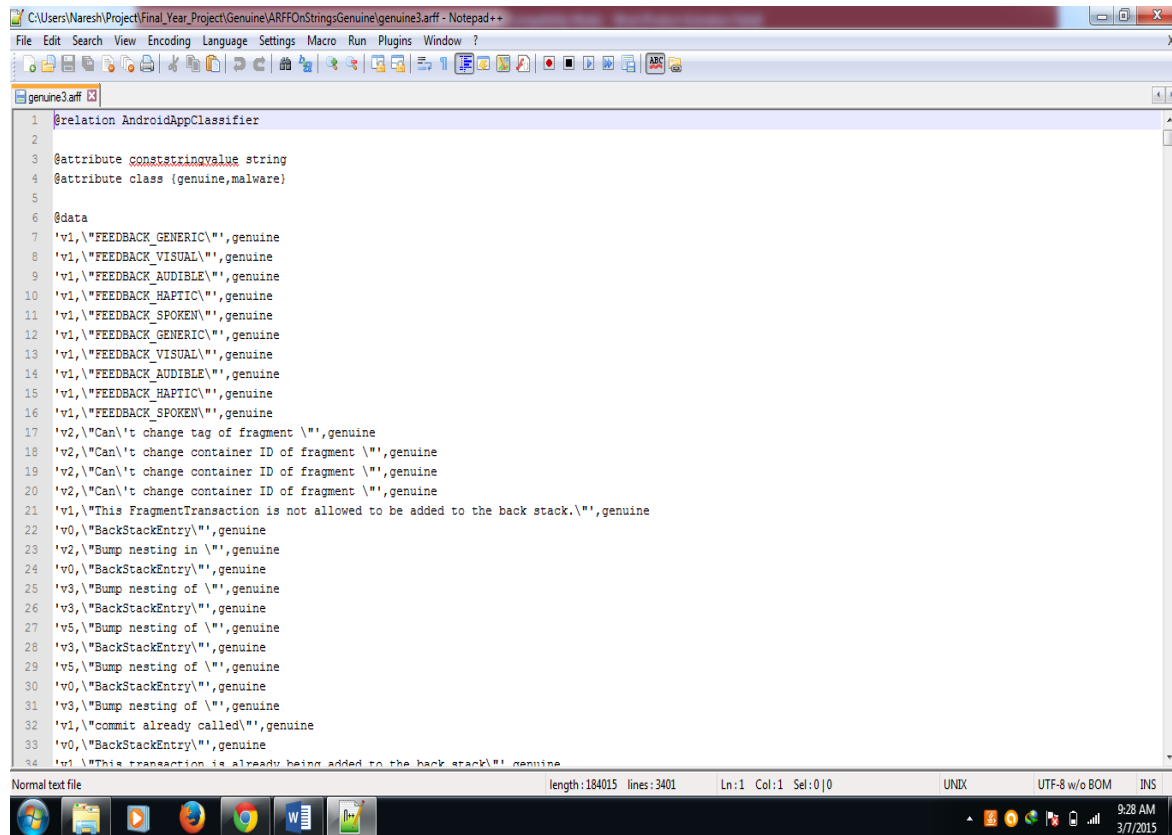
```
1 @relation AndroidAppClassifier
2
3 @attribute permissionvalue string
4 @attribute class {genuine,malware}
5
6 @data
7 android.permission.ADD_SYSTEM_SERVICE,malware
8 android.permission.SET_WALLPAPER_HINTS,malware
9 android.permission.CHANGE_CONFIGURATION,malware
10 android.permission.SET_ALARM,malware
11 android.permission.CHANGE_NETWORK_STATE,malware
12 android.permission.READ_HISTORY_BOOKMARKS,malware
13 android.permission.MODIFY_AUDIO_SETTINGS,malware
14 android.permission.BATTERY_STATS,malware
15 android.permission.EXPAND_STATUS_BAR,malware
16 android.permission.SYSTEM_ALERT_WINDOW,malware
17 android.permission.KILL_BACKGROUND_PROCESSES,malware
18 android.permission.PERSISTENT_ACTIVITY,malware
19 android.permission.INJECT_EVENTS,malware
20 android.permission.INSTALL_LOCATION_PROVIDER,malware
21 android.permission.INTERNAL_SYSTEM_WINDOW,malware
22 android.permission.MANAGE_APP_TOKENS,malware
23 android.permission.MASTER_CLEAR,malware
24 android.permission.READ_HISTORY_BOOKMARKS,malware
25 android.permission.READ_INPUT_STATE,malware
26 android.permission.REBOOT,malware
27 android.permission.SET_ACTIVITY_WATCHER,malware
28 android.permission.USE_CREDENTIALS,malware
29 android.permission.GET_TASKS,malware
30 android.permission.WRITE_CALL_LOG,malware
31 android.permission.PERSISTENT_ACTIVITY,malware
32 android.permission.USE_CREDENTIALS,malware
33 android.permission.GET_TASKS,malware
34 android.permission.WRITE_CALL_LOG,malware
```

FIGURE F

### EXPLANATION:

The above screenshot shows the ARFF file generated using Weka packages for permissions used by the malicious Android applications

## GENERATED ARFF FILE ON GENUINE STRINGS:



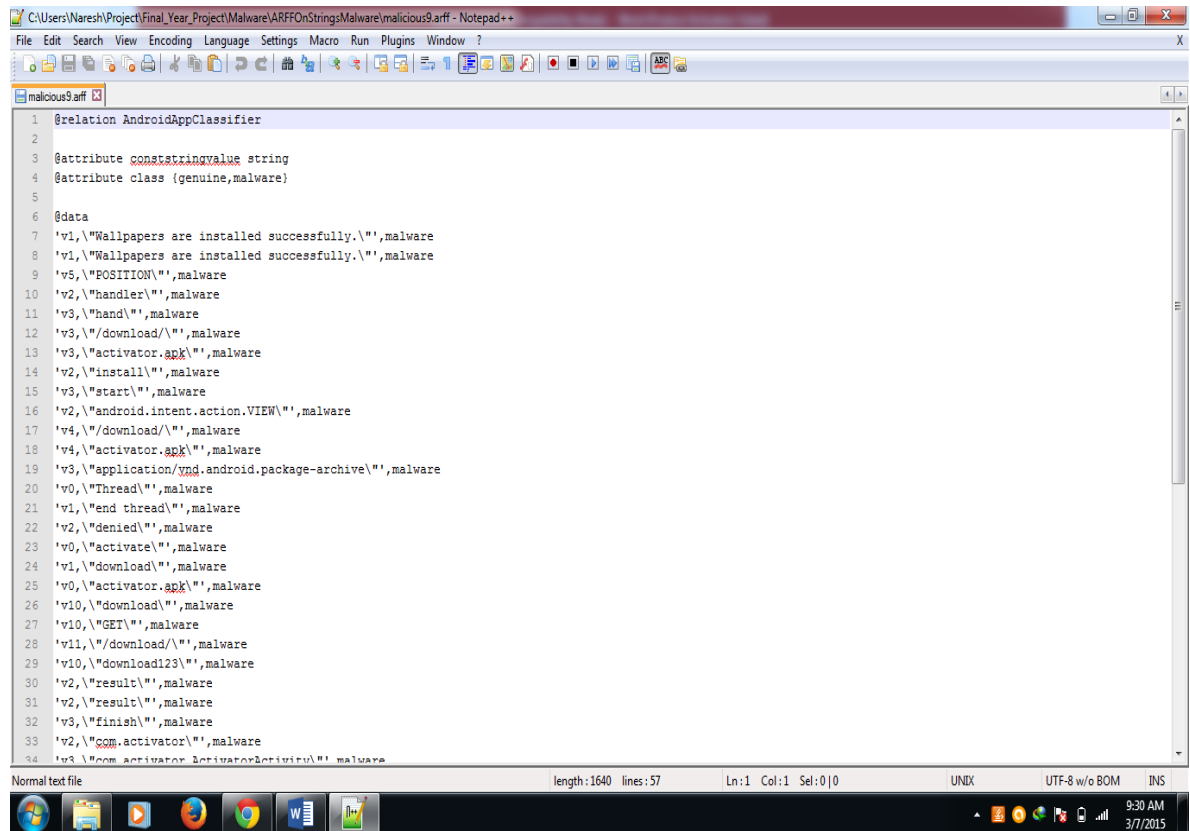
```
1 @relation AndroidAppClassifier
2
3 @attribute conststringValue string
4 @attribute class {genuine,malware}
5
6 @data
7 'v1, \\\"FEEDBACK_GENERIC\\\"',genuine
8 'v1, \\\"FEEDBACK_VISUAL\\\"',genuine
9 'v1, \\\"FEEDBACK_AUDIBLE\\\"',genuine
10 'v1, \\\"FEEDBACK_HAPTIC\\\"',genuine
11 'v1, \\\"FEEDBACK_SPOKEN\\\"',genuine
12 'v1, \\\"FEEDBACK_GENERIC\\\"',genuine
13 'v1, \\\"FEEDBACK_VISUAL\\\"',genuine
14 'v1, \\\"FEEDBACK_AUDIBLE\\\"',genuine
15 'v1, \\\"FEEDBACK_HAPTIC\\\"',genuine
16 'v1, \\\"FEEDBACK_SPOKEN\\\"',genuine
17 'v2, \\\"Can\\'t change tag of fragment \\\"',genuine
18 'v2, \\\"Can\\'t change container ID of fragment \\\"',genuine
19 'v2, \\\"Can\\'t change container ID of fragment \\\"',genuine
20 'v2, \\\"Can\\'t change container ID of fragment \\\"',genuine
21 'v1, \\\"This FragmentTransaction is not allowed to be added to the back stack.\\\"',genuine
22 'v0, \\\"BackStackEntry\\\"',genuine
23 'v2, \\\"Bump nesting in \\\"',genuine
24 'v0, \\\"BackStackEntry\\\"',genuine
25 'v3, \\\"Bump nesting of \\\"',genuine
26 'v3, \\\"BackStackEntry\\\"',genuine
27 'v5, \\\"Bump nesting of \\\"',genuine
28 'v3, \\\"BackStackEntry\\\"',genuine
29 'v5, \\\"Bump nesting of \\\"',genuine
30 'v0, \\\"BackStackEntry\\\"',genuine
31 'v3, \\\"Bump nesting of \\\"',genuine
32 'v1, \\\"commit already called\\\"',genuine
33 'v0, \\\"BackStackEntry\\\"',genuine
34 'v1, \\\"This transaction is already being added to the back stack\\\"',genuine
```

FIGURE G

### EXPLANATION:

The above screenshot shows the ARFF file generated using Weka packages for strings present in genuine Android applications

## GENERATED ARFF FILE ON MALWARE STRINGS:



```
1 @relation AndroidAppClassifier
2
3 @attribute consttringvalue string
4 @attribute class {genuine,malware}
5
6 @data
7 'v1,\"Wallpapers are installed successfully.\",malware
8 'v1,\"Wallpapers are installed successfully.\",malware
9 'v5,\"POSITION\",malware
10 'v2,\"handler\",malware
11 'v3,\"hand\",malware
12 'v3,\"/download/\",malware
13 'v3,\"activator.apk\",malware
14 'v2,\"install\",malware
15 'v3,\"start\",malware
16 'v2,\"android.intent.action.VIEW\",malware
17 'v4,\"/download/\",malware
18 'v4,\"activator.apk\",malware
19 'v3,\"application/vnd.android.package-archive\",malware
20 'v0,\"Thread\",malware
21 'v1,\"end thread\",malware
22 'v2,\"denied\",malware
23 'v0,\"activate\",malware
24 'v1,\"download\",malware
25 'v0,\"activator.apk\",malware
26 'v10,\"download\",malware
27 'v10,\"GET\",malware
28 'v1,\"/download/\",malware
29 'v10,\"download123\",malware
30 'v2,\"result\",malware
31 'v2,\"result\",malware
32 'v3,\"finish\",malware
33 'v2,\"com.activator\",malware
34 'v3,\"com.activator.activator\",malware
```

FIGURE H

### EXPLANATION:

The above screenshot shows the ARFF file generated using Weka packages for strings present in malicious Android applications



**CROSS VALIDATION OUTPUT ON PERMISSIONS:**

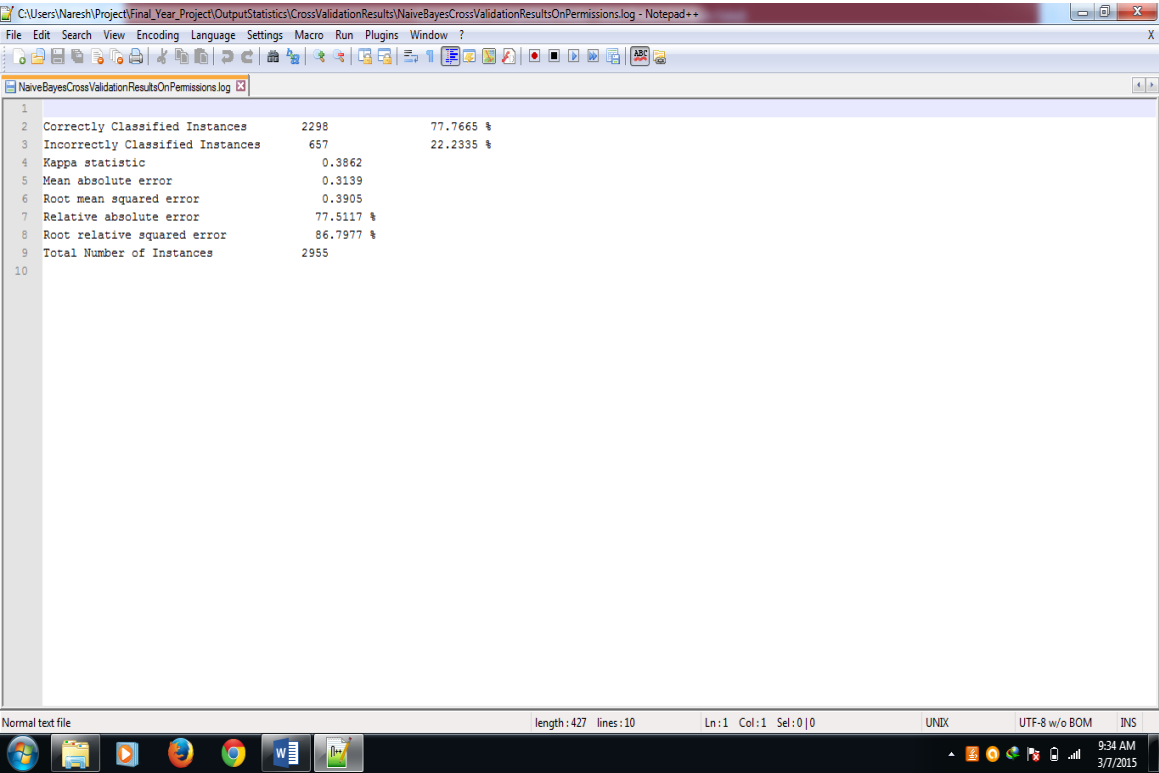


FIGURE I

**EXPLANATION:**

The above screenshot shows the result of the classifier framework using Cross-Validation scheme on permissions.

**CROSS VALIDATION OUTPUT ON STRINGS:**

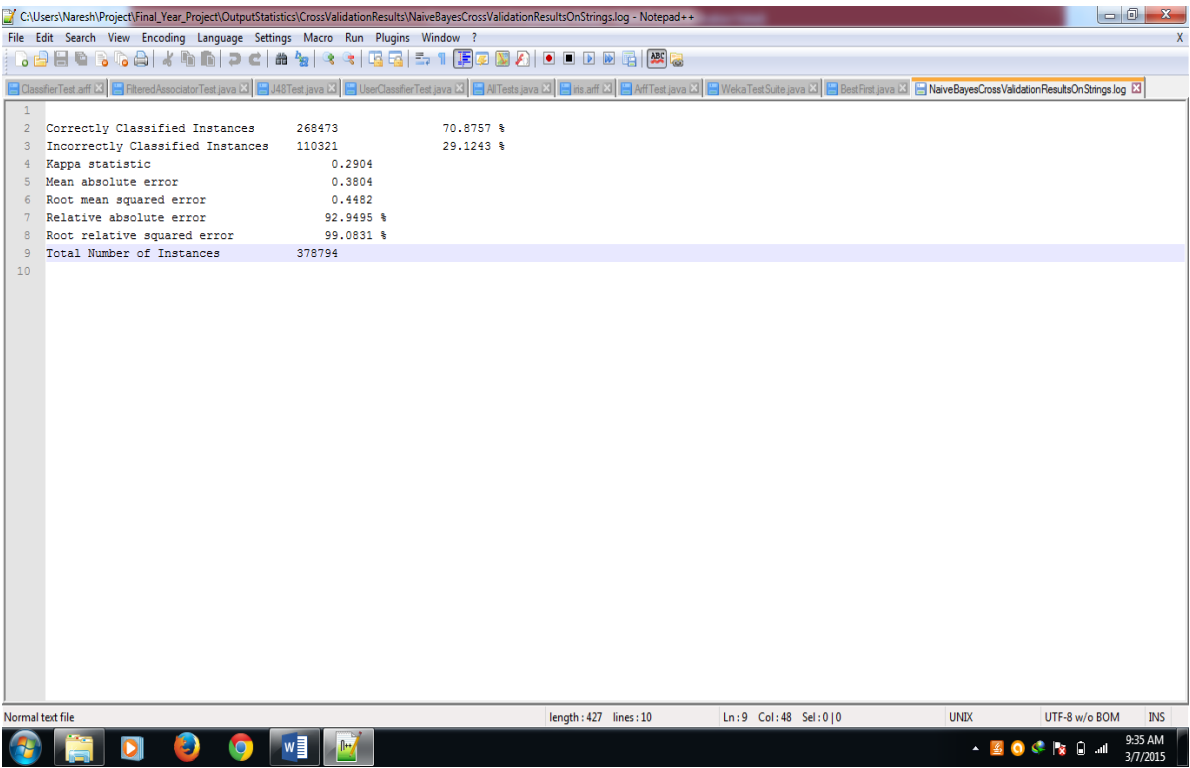
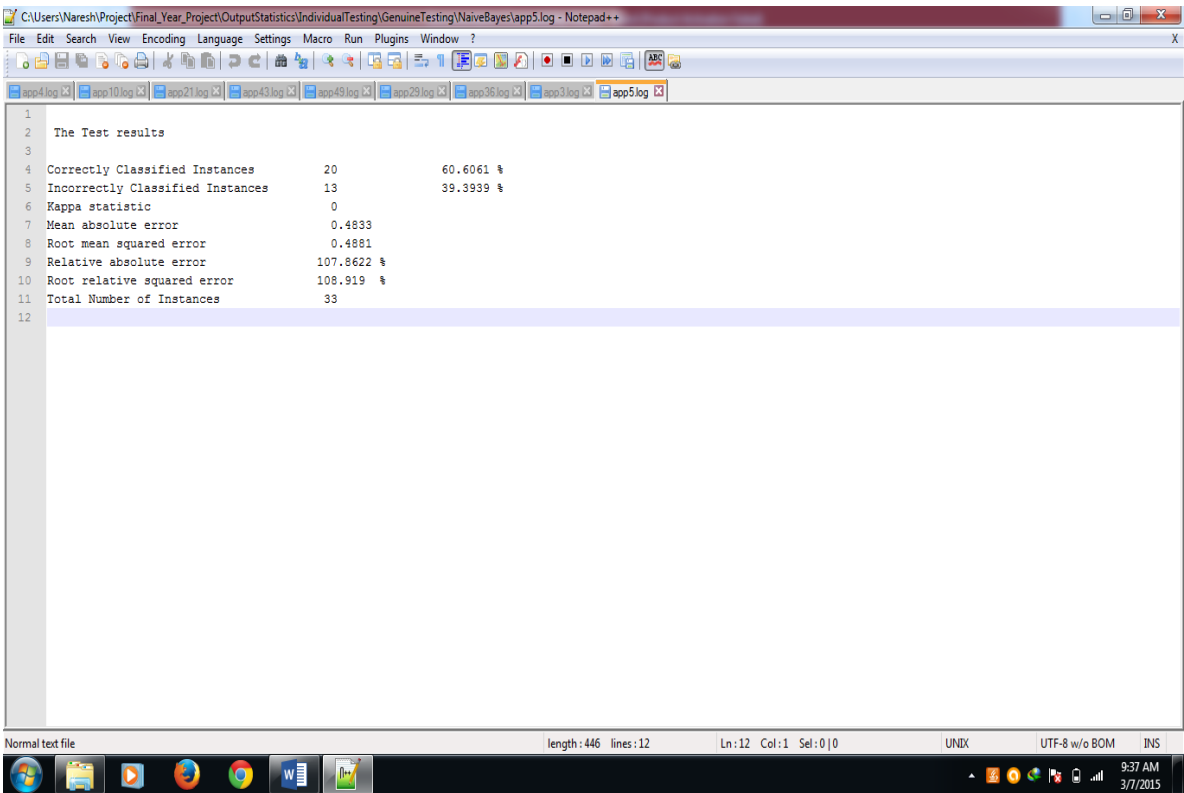


FIGURE J

**EXPLANATION:**

The above screenshot shows the result of the classifier framework using Cross-Validation scheme on strings.

**INDIVIDUAL TESTING RESULTS:**



**FIGURE K**

**EXPLANATION:**

The above screenshot shows the result of the classifier framework on Individual Testing on applications

## REFERENCES

1. Wei Wang, Xing Wang, Dawei Feng, Jiqiang Liu, Zhen Han, Xiangliang Zhang, “Exploring Permissions-induced Risk in Android Applications for Malicious Application Detection”
2. Dong-uk Kim, Jeongtae Kim, Sehun Kim, "Malicious Application Detection Framework using Feature Extraction Tool on Android Market".
3. BorjaSanz, Igor Santos, Carlos Laorden, XabierUgarte-Pedrero and Pablo Garcia Bringas, “On the Automatic Categorization of Android Applications”
4. BorjaSanz, Igor Santos, Carlos Laorden, XabierUgarte-Pedrero and Pablo Garcia Bringas, “MADS: Malicious Android Applications Detection through String Analysis”
5. Matthew G. Schultz, ErezZadok, Salvatore J. Stolfo, “Data Mining Methods for Detection of New Malicious Executables”
6. Yajin Zhou, Zhi Wang, Wu Zhou, Xuxian Jiang, “Hey, You, Get Off My Market: Detecing Malicious Apps in Official and Alternative Android Markets”
7. Adam P. Fuchs, AvikChaudhuri, Jeffrey S. Foster , “SCanDroid: Automated Security Certification of Android Applications”
8. Steven Artz, Siegfried Rasthofer, Eric Bodden, “SuSi: A Tool for the Fullt Automated Classification and Categorization of Android Sources and Sinks”
9. AsafShabtai, YuvaiFledel, Yuval Elovici, “Automated Static Code Anaylsis for Classifying Android Applications Using Machine Learning”
10. Deepak Koundel, SurajIthape, VishakhaKhobaragade, Rajat Jain, “Malware Classification using Naïve Bayes Classifier for Android OS”
11. Franklin Tchakounte, “Permission-based Malware Detection Mechanisms: Analysis and Perspectives”

12. Dedexer, <http://dedexer.sourceforge.net/>
13. AXMLPrinter2, <https://code.google.com/p/android4me/downloads/detail?name=AXMLPrinter2.jar&>
14. Weka (Machine Learning tool), <http://sourceforge.net/projects/weka/>
15. Artificial Intelligence, [http://en.wikipedia.org/wiki/Artificial\\_intelligence](http://en.wikipedia.org/wiki/Artificial_intelligence)
16. Machine Learning, [http://en.wikipedia.org/wiki/Machine\\_learning](http://en.wikipedia.org/wiki/Machine_learning)
17. Supervised Machine Learning,  
[http://en.wikipedia.org/wiki/Supervised\\_learning](http://en.wikipedia.org/wiki/Supervised_learning),  
[http://www.astroml.org/sklearn\\_tutorial/images/plot\\_ML\\_flow\\_chart\\_1.png](http://www.astroml.org/sklearn_tutorial/images/plot_ML_flow_chart_1.png)
18. Cross-Validation Statistics, [http://en.wikipedia.org/wiki/Cross-validation\\_\(statistics\)](http://en.wikipedia.org/wiki/Cross-validation_(statistics))
19. C4.5 Algorithm, [http://en.wikipedia.org/wiki/C4.5\\_algorithm](http://en.wikipedia.org/wiki/C4.5_algorithm)
20. Naïve Bayes Classifier Algorithm,  
[http://en.wikipedia.org/wiki/Naive\\_Bayes\\_classifier](http://en.wikipedia.org/wiki/Naive_Bayes_classifier)