Islamic University of Gaza
Deanery of Higher Studies
Information Technology program

Spyware Detection Using Data Mining for Windows Portable Executable Files

By:

Fadel Omar Shaban

120091437

Supervised by:

Dr. Tawfiq S. Barhoom

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Science In Information Technology

2013-1434H

بسم الله الرحمن الرحيم

﴿قُلْ إِنَّ صَلاتِي وَنُسُكِي وَمَحْيَايَ وَمَمَاتِي لِلَّهِ رَبِّ الْعَالَمِينَ لا
شَرِيكَ لَهُ وَبِذَلِكَ أُمِرْتُ وَأَنَا أَوَّلُ الْمُسْلِمِينَ﴾(الأنعام - 162، 163.)

ACKNOWLEDGMENTS

First and Foremost, I am very grateful to almighty ALLAH whose blessings have always been source of encouragement for me and who gave me the ability to complete this task.

This thesis would not exist without the help, advice, support, guidance, and encouragement of many people.

In particular, I wish to express my sincere appreciation to my supervisor Dr. Tawfiq S. Barhoom, without his help, guidance, and continuous follow-up; this research would never have been.

Also I would like to extend my thanks to the academic staff of the Faculty of Information Technology who taught me different courses and helped me during my Master’s study.

Special greetings to my family, especially my parents, who have always kept me in their prayers, who have suffered a lot to make me happy.

Last but not least, I wish to express my sincere thanks to all those who have one way or another helped me in making this study a success.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ANN	Artificial Neural Networks
API	Application Programming Interface
AUC	Area Under the Curve
BDT	Boosted Decision Tree
BNB	Boosted Naïve Bayes
CA	Computer Associates
CFBE	Common Feature-based Extraction
DCFS	Frequency feature selection measure
DLL	Dynamic link library
DR	Detection Rate
DT	Decision Tree
FPR	False Positive Rate
FS	Fisher Score
FSG	Fast Small Good
GR	Gain Ratio
IAT	Import Address Table
IDA Pro	Interactive Disassembler Pro
IG	Information Gain
IGR	Information Gain Ratio
KDDs	knowledge discovery in databases
kNN	K−Nearest Neighbor
NB	Naïve Bayes
OA	Overall Accuracy
OEP	Original Entry Point
OS	Operating Systems
PE	Portable Executable
PEiD	Portable Executable iDentifier
RF	Random Forest
RFSs	Reduced Feature Sets (RFSs)
ROC	Receiver Operating Characteristic
SVMs	Support Vector Machines
TPR	True Positive Rate
UPack	Ultimate PE Packer
UPX	Ultimate Packer for Executables

Abstract

Malware represents a significant problem that threatens the security of computer systems. Spyware is one of the recent types of malware that represents a serious threat to confidentiality. The traditional approaches using signature-based to detect malicious programs fails in detecting new and unknown spyware. Many of the malware detection techniques which have been developed on malware are not investigated in terms of spyware detection. In this research, we investigate the spyware detection by using data mining techniques based on mining Application Programming Interface (API) calls. 2084 spyware and 1065 benign windows Portable Executable (PE) file samples were collected from the Internet in order to create binary data set. API call statically extracted from binary file, then generate a set of features and features selection was performed, these features are then used to train a classifier. We evaluated a variety of classification algorithms, including Random forest, Naïve Bayes (NB), K−Nearest Neighbor (kNN), JRip, J48 decision trees, and support vector machines (SVMs). The accuracy and the area under ROC curve are used for the evaluation of classifier performance. The results show that we achieved an overall accuracy of 98.09% with an area under the ROC curve of 0.995.

Keywords: Spyware, Data Mining, Classification, Portable Executable (PE), Packers, API Call.

الكشف عن برامج التجسس في ملفات ويندوز التنفيذية باستخدام تنقيب البيانات

ملخص

تشكل البرمجيات الخبيثة مشكلة كبيرة تهدد أمن أنظمة الحاسوب. تعتبر برامج التجسس إحدى أنواع البرامج الخبيثة التي تمثل تهديدا خطيراً للخصوصية. إن الأساليب التقليدية التي تستخدم التوقيع للكشف عن البرامج الخبيثة تفشل في الكشف عن البرامج الخبيثة الجديدة و غير المعروفة.

العديد من تقنيات الكشف عن البرمجيات الخبيثة التي تم تطويرها للبرمجيات الخبيثة لم يتم التحقق منها من حيث الكشف عن برامج التجسس.

في هذا البحث، نتحقق من إمكانية الكشف عن برامج التجسس باستخدام تقنيات التنقيب عن البيانات في استدعاءات واجهة برمجة التطبيقات (API). تم جمع 2084 ملف تجسس و 1065 ملف سليم كعينات من نوع ملفات ويندوز قابل للتنفيذ من الإنترنت و ذلك من أجل أنشاء مجموعة البيانات الثنائية. تم استخراج استدعاء API بشكل ثابت من الملف الثنائي ، ثم تكوين مجموعة من الميزات و من ثم تم الاختيار من هذه الميزات ، هذه الميزات استخدمت لتدريب المصنف. قمنا بتقييم مجموعة متنوعة من خوارزميات التصنيف (Random forest, naive Bayes, IBk, JRip, J48, decision trees, and support vector machines.). حيث تم استخدام الدقة والمنطقة تحت منحنى ROC لتقييم أداء المصنف. وأظهرت النتائج التي حققناها أن الدقة الاجمالية للمصنف 98.09٪ مع منطقة تحت منحنى ROC 0.995.

كلمات مفتاحية :التجسس ، تنقيب البيانات، التصنيف ، ملف محمول قابل للتنفيذ ، Packers، استدعاءAPI.

CHAPTER 1: Introduction

1.1 Introduction

Malicious software, commonly termed as malwares, is one of the major threats on the privacy of sensitive data and the availability of critical services. Vasudevan and Yerraballi in [58], describes malware as "a generic term that encompasses viruses, Trojans, spywares and other intrusive code". Malwares can be classified into viruses, worms, Trojans, spywares, adwares and a variety of other classes and subclasses, that sometimes overlap and often represent closely related subclasses of each other [52].

Many researches have been carried out in viruses, which represented the only major malicious threats to computer users in order to successfully detect and remove them from computer systems [45]. However, spyware represent recent type of malicious threat, which has not been extensively studied [45]. Spyware can be defined as "Any software that monitors user behavior, or gathers information about the user without adequate notice, consent, or control from the user" [8]. Spyware can be divided into two categories, advertising spyware and surveillance spyware [63].

Advertising spyware may display some advertisements on user's screen or log some information about the user for business.

Surveillance spyware, which is more dangerous and used by hackers to stealthily access your computer or even launch an attack from your computer. They are usually running in the background to collect and transmit critical information such as keystrokes, bank accounts, credit card numbers and passwords, etc. [8]. Spyware detection different from viruses or worms detecting as vendors of spyware-hosting applications usually include them in bundles with popular free software [45].

Malware detection becomes one of the most critical issues in the field of computer security since malicious executables induce significant loss and damages [3]. Analysis techniques can broadly be classified into two types static and dynamic. In the static analysis the code of the program is examined without actually running the program while in dynamic analysis the program is executed in a real or virtual environment [46]. In static anomaly-based detection, characteristics about the file structure of the program under inspection are used to detect malicious code [20].

According to Panda Security's [39] malware detection and analysis laboratory malware report for the first quarter of 2009, Spyware category has increased almost eleven percent, which rose from 2.5% in the previous quarter to 13.15% in the first three months of the year, placing itself as the second most detected malware category in first quarter of 2009 [39]. In 2007, 56 percent of the total malware seen by CA Global Security Advisor and CA customers were malicious spyware [11]. In first half of 2010, the most prevalent Trojan families 47% of the total Trojan are information stealers, and attackers use various file formats in infection and propagation the amount of Windows i386 (32-bit) portable executables used was 92% in first half of 2009 and 87% in 2010 [12]. Microsoft Windows dominates the world's desktop operating system market share with almost 91.71% of the market share while Mac and Linux share the remaining 8.29% with a lot of other available operating systems [31]. According to Security firm, BitDefender test on Nov. 9, 2012 [6], newly launched Window 8 is prone to infection by some 15 percent of the 100 malware families, even with Windows Defender activated. When tested with Windows Defender disabled 60.78 percent ran successfully.

Consequently, this gives a clear indication that Spyware is one of the biggest threats to the users.

We recognize the need for an efficient method for spyware detection that able to detect new (unknown) spyware.

1.2 Statement of the problem

Spyware becomes a serious problem in these days with the integration of computers and high-speed networks into our daily lives. One way of spreading the spyware is Windows Portable Executable (PE) file that hosts spyware which is widely used. There is little research on Spyware detection, and these techniques are unable to achieve an acceptable and sufficient accuracy and can be improved.

1.3 Objectives

The enormous increasing in spyware made it difficult for researchers and anti-malware vendors to detect such unknown spyware; this causes an increasing signature lag. Our objective of this research is to explore a number of standards data mining techniques in order to distinguish between the benign and spyware executables.

The goal of our research is to explore a number of standards data mining techniques in order to create accurate detectors able to detect the new spyware file.

1.3.1 Main objective

The main objective of this research is to have spyware detection method that can improve more effective detect the spyware by using data mining classification algorithms based on static anomaly detection that can be detect known and new, previously unseen spyware from windows Portable Executable (PE) file.

1.3.2 Specific objectives

There are many specific objectives we can extracts from the main objective such as:

I. Collecting data set to perform experiments in the absence of publicly available spyware dataset.

II. Finding and selecting the most related features and feature type that appropriate for the mining process.

III. Finding the appropriate data mining classification techniques such as (Support Vector Machines, Naive Bayes, and Decision Tree) to detect spyware.

IV. Trying to reduce the false positive and false negative rate and increase the detection rate of the detector.

V. Design and Build Spyware detection model based on static anomaly detection using data mining that able to detect known and unknown spyware.

1.4 Scope and Limitation:

This research aims to enhance spyware detection by using data mining classification algorithms. The work is applied with some limitation and assumption such as:

1. The research is limited to the analysis of executable that can run on the Microsoft Windows operating system; Win32 Portable Executable (PE) files all datasets binary (benign or spyware) are in Windows PE format.
2. The research based on supervised learning algorithms to detect spyware.
3. The research will be limited to one approach of malware detection techniques based on static anomaly detection.

1.5 Importance of the research

The importance of the selected topic comes from:

I. While there are a lot of traditional techniques to detect malware and spyware but most of which are signature-based, which fail to detect new previously unseen malicious. So we try to help in enhancing spyware detection.

II. Many of the research conducted on malware detection in general but a little research exists in Spyware detection, so this research participates in enhancing and to enriching this research area.

III. Microsoft's Windows operating system dominates the world's desktop operating system; this gives an indicator those windows platforms, and its Portable Executable (PE) files will be the target of attacks by attackers.

1.6 Thesis Organization

This dissertation has been divided into five major chapters, which are structured around the objectives of the research.

Chapter 2 Provides Literature Review of Malware detection techniques, Portable Executable File (PE), Packers and Unpacking Also; this chapter presents details about Data Mining techniques, Classification algorithms, Classification Performance and cross-validation.

Chapter 3 provides the some related work of malware and spyware detection.

Chapter 4 provides description about data collection, the methodology steps used in spyware detection. Also this chapter describes feature extraction and feature selection.

Chapter 5 Give the details about the sets of experiments, and analyze the experimental results. Finally, Results Comparison and the summary and the future will be drawn in Chapter 6.

CHAPTER 2: Literature Review

This chapter provides a brief introduction about malware Detection Techniques and describes various approaches of detection; the Portable Executable File (PE) and its headers and sections. Finally, we will explain the use of data mining, especially classification techniques, and clarify their performance measures.

2.1 Malware Detection Techniques

A malware detector is a system that attempts to identify malware. Techniques used for detecting malware can be categorized broadly into two categories: anomaly-based detection and signature-based detection [20].

2.1.1 Anomaly-based detection

Usually occurs in two phases a training phase in which the detector attempts to learn the normal behavior and a detection phase. The advantage of an anomaly-based detection is its ability to detect zero-day attacks. The success of these techniques depends on what features should be learnt in training phase to distinguish malware and benign accurately [59].

2.1.1.1 Specification-based detection

Specification-based detection is a special type of anomaly-based detection, which leverage some specification or rule, set of what is valid behavior in order to decide the maliciousness of a program under inspection.

2.1.2 Signature-based detection

Sometimes referred to as misuse detection uses its characterization of what is known to be malicious to decide the maliciousness of a program under inspection.

As in Figure 2.1 each detection techniques can employ one of three different approaches (static, dynamic, or hybrid). Each technique specific approach or analysis of an anomaly-based or signature-based is determined by how the technique gathers information to detect malware. Static approach attempts to detect malware before the program executes. However, dynamic approach attempts to detect malicious behavior during program execution or after program execution. In hybrid techniques a combination of two approaches is used [20].

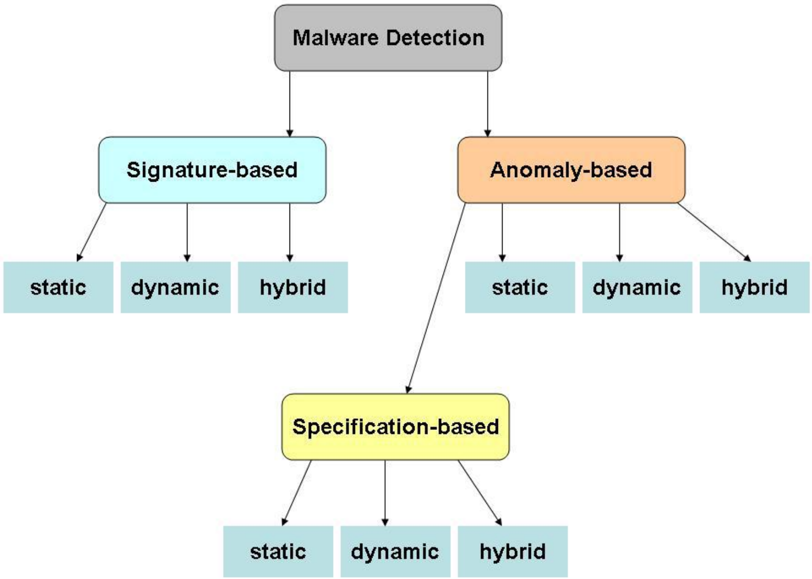


Figure 2.1 A classification of malware detection techniques [20].

Both approaches dynamic and static analysis techniques have their unique advantages and disadvantages. Dynamic analysis provides only a partial "effects-oriented" profile of the full potential of a given malware binary. Dynamic analysis cannot reveal the effects of programming logic that fails to execute during the runtime analysis [44].

Static program analysis offers the potential for a more comprehensive assessment of the entire code and data of the program. For example, by analyzing the sequence of invoked system calls and APIs, performing control flow analysis, and tracking data segment references, it is possible to infer logical code bombs, temporal triggers, and other malicious system interactions, and from these form higher level semantics about malicious behavior. Features such as the presence of network communication logic, registry and OS manipulations, object creations can be detected, whether these capabilities are exercised at runtime or not [44].

2.2 Portable Executable File

The Portable Executable File Format (PE) was designed to be a standard executable format for use on all versions of the Windows operating systems on all supported processors [52] [37] [38]. The PE file format is derived from COFF (common object file format) found on VAX/VMS, but it is an updated version. It is called portable because the same file format is used for all flavors of Windows, on all supported CPUs.

The PE file format was defined to provide the best way for the Windows Operating System to execute code and also to store the essential data which is needed to run a program.

2.2.1 The PE File Headers and Sections

PE file consists of number of headers and sections with no size constraints Figure 2.2 shows the overall structure of a PE file and Table 2.1 provides a good understanding of PE sections. The PE file format contains a header followed by a series of sections. At a minimum PE file will have two sections one for code and the other for data. These sections contain either code or data (and occasionally a mixture of both). Some sections contain code or data declared by the actual application, whereas other data sections contain important information for the operating system. The PE header starts with the signature of the file and contains the file properties, such as number of sections and timestamp. The PE header describes vital pieces of the portable executable it instructs the operating system on how to map the executable in memory.

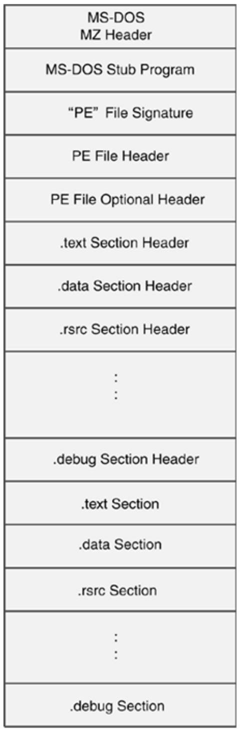


Figure 2.2 A high-level view of the PE file image [50].

2.2.1.1.1

The following are the most common and interesting sections in a PE file [19]:

- **.text:** This section is the default code section that contains the instructions that the CPU executes. All other sections store data and supporting information. Generally, this is the only section that can execute, and it should be the only section that includes code.
- **.rdata**: This section contains the import and export information. Also can store other read-only data used by the program. Sometimes a file will contain an .idata and .edata section, which stores the import and export information.
- **.data:** The program's global data stores in this section, which is accessible from anywhere in the program. Local data is not stored in this section, or anywhere else in the PE file. Also contains the file's Original Entry Point (OEP) which refers to the execution entry point (where the file execution begins) of a portable executable file.
- **.rsrc**: The resources used by the executable including this section that are not considered part of the executable, such as icons, images, menus, and strings. Strings can be stored

either in the .rsrc section or in the main program, but they are often stored in the .rsrc section for Multilanguage support.

Section names are actually irrelevant as they are ignored by the OS since it uses other information in the PE header to determine how a section is used and can vary across different compilers. For example, Visual Studio uses .text for executable code, but Borland Delphi uses CODE. Table 2.1 lists the most common PE File Sections.

Table 2.1 Sections of a PE File for a Windows Executable [19].

Executable Section	**Section Description**
.text	Contains the executable code
.rdata	Holds read-only data that is globally accessible within the program
.data	Stores global data accessed throughout the prog
.idata	Sometimes present and stores the import function information; if this section is not present, the import function information is stored in the .rdata section
.edata	Sometimes present and stores the export function information; if this section is not present, the export function information is stored in the .rdata section
.pdata	Present only in 64-bit executables and stores exception-handling information
.rsrc	Stores resources needed by the executable
.reloc	Contains information for relocation of library files

The most important thing about PE files is that the data structures on disk are the same data structures used in memory, so once a module is loaded, the Import Address Table (IAT) contains the address that is invoked when calling imported APIs.

2.2.2 Importing Functions

When any PE file loads, Windows loader is responsible to locate all the imported functions and data and make those addresses available to the file being loaded. The import section (.idata) contains information about all the functions imported by the executable from dynamic link libraries DLLs. When the binary is loaded this information is stored in several data structures. The most important of these are the Import Directory and the Import Address Table. Figure 2.3 illustrates the high-level view of the mechanism. The required DLLs are mapped into the memory address space of the application, and the export table in the DLL is used to determine the virtual addresses of the functions that need to be linked. A table called the Import Address Table (IAT) is filled in by the loader and linker with the virtual addresses of each imported function. This table is referred to by indirect control flow instructions in the program to call the functions in the linked DLL [44].

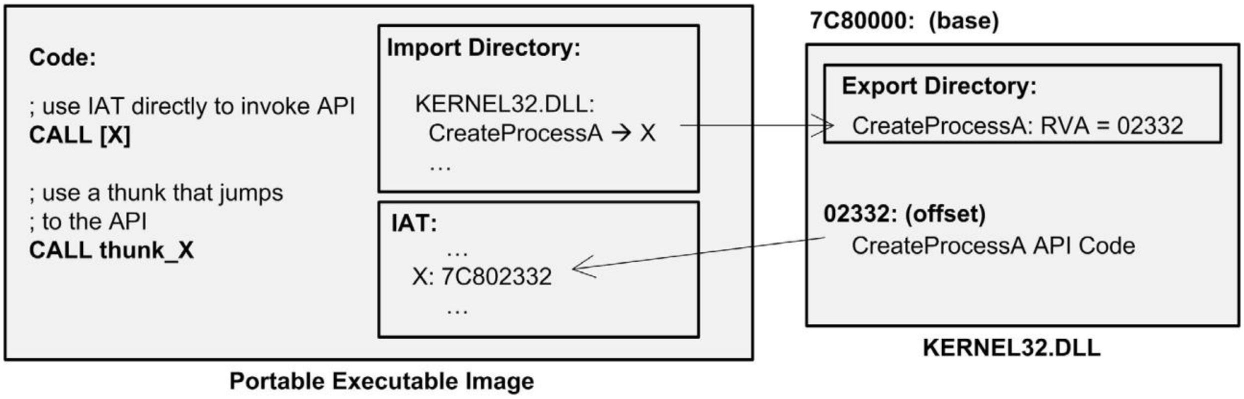


Figure 2.3 Example of the standard linking mechanism of PE executables in Windows [44].

2.3 Packers and Unpacking

Malware authors often use various techniques such as packing or obfuscation to hide the content of their files and making it difficult to detect or analyze [19]. Code Obfuscation means modifying the program code in a way to preserve it’s functionally with the aim to making the code difficult to disassemble or decompile [29].

Packers are programs that compress and encrypt other executable files in a disk and associating it with a restoration routine which restore the original executable images when the packed files are loaded into memories [67] [23]. A packed file is a type of archive file; thereby not all packed files are bad. Packers can help in protect applications against cracking and protecting codes from decompiling also saving storage space since packers compress executable file [67].

Using packers is becoming increasingly popular among malware authors. Packers help malware writers to hide actual program logic and keeping the malware undetected for long time by evading detection from anti-malware programs as well as spreading faster due to its smaller size. Also malware writer can easily generate a new malware by packing an existing one to bypass signature based detector, according to [51]more than 50% of new malware are produced simply by repacking existing malware, using different packers.

Presently, there are many different packing utilities tools available, most of which can be used by anyone with minimal computer skills. Each of these tools may have several variants or versions. The Source code for a number of packing tools, such as UPX, FSG, Yodacrypt and Morphine is available on the Internet. These can easily be modified by any malware writer, by modification of the compression algorithms or by adding one or even several encryption layers [33].

Packers can be classified based on their purposes and behaviors into four categories: [67]

- Compressors: Simply shrink file sizes through compression with little or no anti-unpacking tricks [67]. Popular Examples of compressors include the Ultimate PE Packer UPack, Ultimate Packer for Executables (UPX) [55] and ASPack [47].
- Crypters: the Packer encrypts and obfuscates the original file contents and prevents the files from being unpacked without any compression [67]. Examples of crypters widely used by malware developers: Yoda's Crypter [56] and PolyCrypt PE [22].
- Protectors: A hybrid packer that combines features from both compressors and crypters [67]. Examples of commercial protectors, such as Armadillo [54] and Themida [53].
- Bundlers pack: The Packer package of multiple executable and data files into a single bundled executable file, which unpacks and accesses files within the package without extracting them to disk [67]. Examples of typical PE bundlers include PEBundle [7] and MoleBox [50].

Packer works by parses PE internal structures. Then, it reorganizes PE headers, sections, import tables, and export tables into new structures; the packed executable is compressed, encrypted, or otherwise transformed and attaches a code segment that the malware will invoke before the OEP. Packers can pack the entire executable, including all data and the resource section, or pack only the code and data sections. The outer layer works to unpack the inner executable in memory and transfers execution to it. Figure 2.4 shows its packed counterpart. This code is called the stub, and it decompresses the original data and locates the OEP. With packed programs, the unpacking stub is loaded by the OS, and then the unpacking stub loads the original program [19] [67] [43]. The unpacking stub performs three steps [19] :

1. Unpacks the original executable into memory
2. Resolves all of the imports of the original executable
3. Transfers execution to the original entry point (OEP)

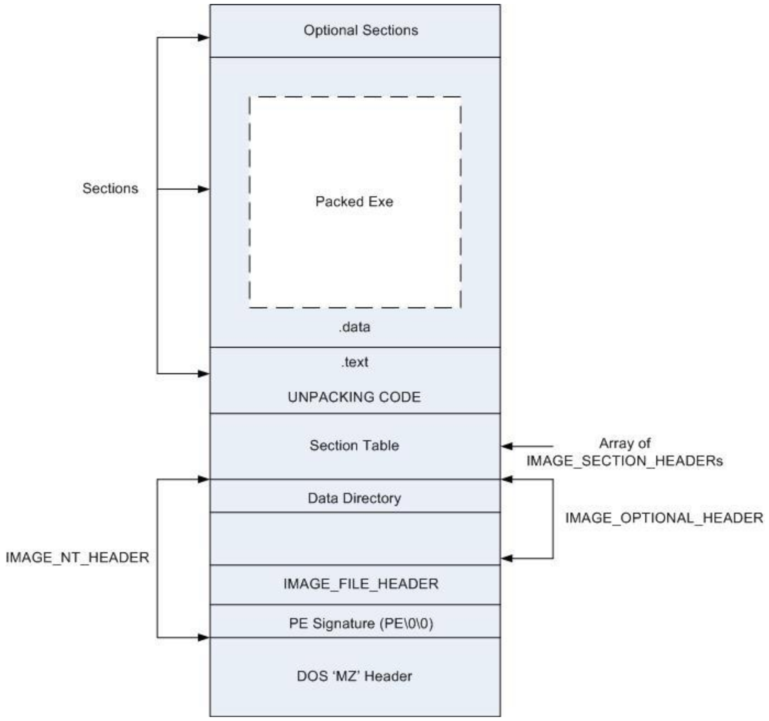


Figure 2.4 PE structure of a packed executable [43].

Detection of packer is very important for the analysis of the malware. If we know the packer, we can unpack the code and then analyze the malware. Packer detection approach can be categorized into static or dynamic, based on where the detection happens, either before the code gets loaded into the memory or after [43]. There are many heuristic indicators that the file is packed, such as the following [28]:

- Small number of functions: The file only few built-in functions, whereas normal, unpacked programs will have many more.
- Small number of imports: The file imports fewer than 10 API functions from libraries supplied by the OS. This indicates that either the file is very limited in functionality or a packer has “hidden” the API functions.
- Encoding instructions inside a loop: The series of IMUL, ADD, SHR, XOR, and AND instructions with hard-coded numbers inside a loop indicates that the program performs some type of obfuscation or de-obfuscation.

Packed and obfuscated code will often include at least two functions which are used to load and gain access to additional functions LoadLibrary and GetProcAddress [19].

PEiD [48] tool which is a free tool for the detection of packers is most commonly used with signature-based packers, cryptors and compilers for PE file detection, and it can detect more than 600 different signatures in PE files [66].

2.4 Data Mining

This section introduces an overview of the basic concepts of Data Mining process and classification algorithms.

Data mining is considered as an application of machine learning. Data mining can be defined as the process of discovering patterns in data [25]. It’s refers to the process of discovering new meaningful correlation, patterns and trends from large amounts of data stored in repositories, using pattern recognition technologies as well as statistical and mathematical techniques [27].

Data mining have two primary goals prediction and description [17].

Prediction: It involves predicting unknown or future values using some variables or fields in the data set.

Description: It involves finding patterns describing the data that can be interpreted by humans.

Data mining is also an integral part of knowledge discovery in databases (KDDs) which includes several steps [14].

1. Data cleaning, which is also known as data cleansing, it is a phase in which noise and irrelevant data are removed from the collection.
2. Data integration combines data from multiple and heterogeneous sources into one database.
3. Data selection, which allows the user to obtain a reduced representation of the data set to keep the integrity of the original data set in a reduced volume.
4. Data transformation, the selected data is transformed into suitable formats.
5. Data mining, analysis tools are applied to discover potentially useful patterns.
6. Pattern evaluation, identifies interesting and useful patterns using given validation measures.
7. Knowledge representation, the final phase of the knowledge-discovery process, discovered knowledge is presented to the users in visual forms.

The primary data-mining tasks described as the following [17]

1. Classification: discovery of a predictive learning function that classifies a data item into one of more predefined classes.
2. Regression: discovery of a predictive learning function, which maps a data item to a real value prediction variable.
3. Clustering: identify a finite set of categories or clusters to describe the data.
4. Summarization: descriptive task that involves methods for finding a compact description for a set (or subset) of data.
5. Dependency Modeling: finding a local model that describes significant dependencies between variables or between the values of a feature in a data set.
6. Change and Deviation Detection: discovering the most significant changes in the data set.

2.4.1 Data Reduction

The most important issue that determines the quality of a data mining technique is the choice of data representation, and selection, reduction or transformation of features [24]. Data reduction techniques can be applied to obtain a reduced representation of the data set that is much smaller in volume and closely maintains the integrity of the original data [17].

The data reduction should be performed prior to applying data mining; the basic operation in data reduction are delete a column, delete a row, and reduce the number of values in a column (smooth a feature) [24].

2.4.1.1 Attribute Subset Selection

Feature selection (also known as subset selection) is a process commonly used in machine learning, wherein a subset of the features available from the data is selected and irrelevant, weakly relevant or redundant attributes or dimensions may be detected and removed. The best subset contains the least number of dimensions that most contribute to accuracy. Mining on a reduced set of attributes has an additional benefit. It reduces the number of attributes appearing in the discovered patterns, helping to make the patterns easier to understand [17].

Basically, choosing the most relevant features to achieve maximum performance with the minimum measurement and processing effort [24].

A feature-reduction process should result in

- less data so that the data mining algorithm can learn faster;
- higher accuracy of a data mining process so that the model can generalize better from data;
- simple results of a data mining process so that they are easier to understand and use;
- Fewer feature so that in the next round of data collection, a saving can be made by removing redundant or irrelevant features.

Information gain

Information gain is defined as the difference between the original information requirement and the new requirement [17].

The entropy of variable X is defined as [4]:

$$H(X) = -\sum_{i} P(X_i)\log_2(P(X_i)) \tag{2.1}$$

And the entropy of X after observing values of another variable Y is defined as:

$$H(X|Y) = -\sum_{i} P(y_i) \sum_{i} P(x_i|y_i) \log_2(P(x_i|y_i)) \tag{2.2}$$

Where $P(X_i)$ are the prior probabilities for the values of X, and $P(x_i|y_i)$ is the posterior probabilities of xi given the values of yi. The amount by which the entropy of X decreases reflects additional information about X provided by Y and is called information gain. The information gain is given by:

$$IG(X|Y) = H(X) - H(X|Y) \tag{2.3}$$

2.4.2 Classification

Classification is a supervised machine learning technique which is the process of finding a model that describes and distinguishes data classes for the purpose of being able to use the model to predict the class of objects whose class label is unknown [17].

Data classification is a two-step process.

- Step 1: A classifier is built describing a predetermined set of data classes or concepts.
  In this step, learning, training data are analyzed by a classification algorithm. The goal of learning is to create a classification model, known as a classifier.

- Step 2: Here, the model is used for classification. First, the predictive accuracy of the classifier is estimated.
  In this step, test data are used to estimate the accuracy of the classification rules. If the accuracy is considered acceptable, the rules can be applied to the classification of new data tuples.

2.4.3 Classification algorithms

This section will cover brief overview about the classification algorithms used in our research.

2.4.3.1 Random Forest

Random forest is an ensemble classifier that consists of a collection of tree structured classifiers as in Figure 2.5. The output of random forest is decided by the votes given by all individual trees. For the given set of inputs $\{h(x,\Theta k\ ), k=1, ...\}$ where x represents the observed input and $\{\Theta k\}$ are independent identically distributed random vectors and each tree casts a unit vote for the most popular class at input x [10].

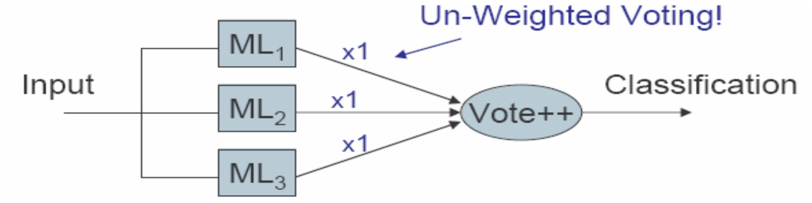


Figure 2.5 Random Forest [30].

In random forests, there is no need for cross validation or a test set to get an unbiased estimate of the test error. Since each tree is constructed using the bootstrap sample.

2.4.3.2 Decision Tree (DT)

Decision tree algorithm is a data mining induction techniques that recursively partitions a data set of records. Decision trees can handle high dimensional data, the learning and classification steps are simple and fast [17]. Decision tree is a flow-chart-like tree structure, where each node denotes a test on an attribute value, each branch represents an outcome of the test, and tree leaves represent classes [17].

A decision tree is created in two phases:

1) Tree Building Phase

   Tree building is done in top-down manner, this phase can repeatedly partition the training data until all the examples in each partition belong to one class or the partition is sufficiently small.

2) Tree Pruning Phase

   This phase should remove dependency on statistical noise or variation that may be particular only to the training set.

```
Algorithm: Generate decision tree. Generate a decision tree from the training tuples of data
partition D.
Input:
Data partition, D, which is a set of training tuples and their associated class labels;
attribute list, the set of candidate attributes;
Attribute selection method, a procedure to determine the splitting criterion that “best” partitions
the data tuples into individual classes. This criterion consists of a splitting attribute
and, possibly, either a split point or splitting subset.
Output: A decision tree.
Method:
(1) create a node N;
(2) if tuples in D are all of the same class, C then
(3) return N as a leaf node labeled with the class C;
(4) if attribute list is empty then
(5) return N as a leaf node labeled with the majority class in D; // majority voting
(6) apply Attribute selection method(D, attribute list) to find the “best” splitting criterion;
(7) label node N with splitting criterion;
(8) if splitting attribute is discrete-valued and
multiway splits allowed then // not restricted to binary trees
(9) attribute list attribute list □ splitting attribute; // remove splitting attribute
(10) for each outcome j of splitting criterion
// partition the tuples and grow subtrees for each partition
(11) let Dj be the set of data tuples in D satisfying outcome j; // a partition
(12) if Dj is empty then
(13) attach a leaf labeled with the majority class in D to node N;
(14) else attach the node returned by Generate decision tree(Dj, attribute list) to node N;
endfor
(15) return N;
```

Figure 2.6 Basic algorithm for inducing a decision tree from training tuples [17].

J48 is decision-tree-based learning algorithm, it’s an enhanced version of C4.5 based on the ID3 algorithm.

2.4.3.3 Support vector machines

The Support vector machines (SVMs) proposed by Vapnik [57] is a kind of machine learning algorithm with high generalization ability and substantial theory. SVMs provide a state-of-the-art learning method that has been highly successful in a variety of applications [26]. The learning algorithms of SVM can be used either a classification or regression [36].

The basic form of SVM takes a set of input data and predicts, for each given input, which of two possible classes forms the output. For classification, SVM performs classification by finding the separating hyperplane between two classes in a high dimension feature space that maximize the separation margin between the two classes [63] the vectors that define the hyperplane are the support vectors. As shown in figure 2.7 there are many linear classifiers (hyperplanes) that able to separate data into multiple classes. However, only one hyperplane achieves maximum separation. If such a hyperplane exists it's known as the maximum-margin hyperplane and such a linear classifier is known as a maximum margin classifier [36].

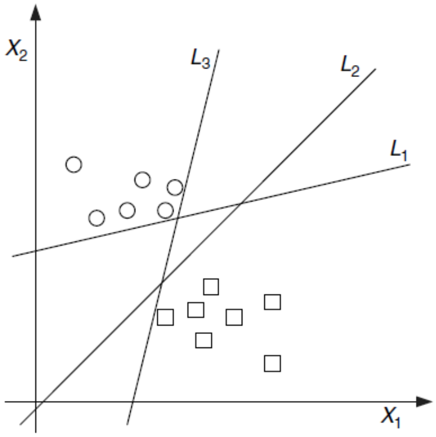


Figure 2.7 Many linear classifiers (hyper planes) may separate the data [36].

Given a training dataset (x1, y1), (x2, y2)… (x_n, y_n) where each x_i is a vector of real numbers and y_i is the corresponding class either 1 or -1. Then the training algorithm attempts to place a hyperplane between points y_i = 1 and points where y_i = -1. Any hyperplane Eq. 2.4 [36] can be written as the set of points X satisfying

$$w.x - b = 0 \qquad (2.5)$$

The vector *w* points perpendicular to the separating hyperplane. Adding the offset parameter *b* allows us to increase the margin.

We are interested in the maximum margin; we are interested in the support vectors and the parallel hyperplanes closest to these support vectors in either class. These hyperplanes can be described by the equations.

$$w.x - b = \ \ 1 \qquad (2.6)$$
$$w.x - b = -1 \qquad (2.7\ )$$

By using geometry, we find the distance between the hyperplanes is 2/|*w*|, so we want to minimize |*w*|. To exclude data points, we need to ensure that for all *i* either

$$w.xi - b \geq 1 \ \text{Or} \tag{2.8}$$

$$w.xi - b \leq -1 \tag{2.9}$$

This can be rewritten as:

$$Yi(w.x - b) \geq 1, 1 \leq i \leq n \tag{2.10}$$

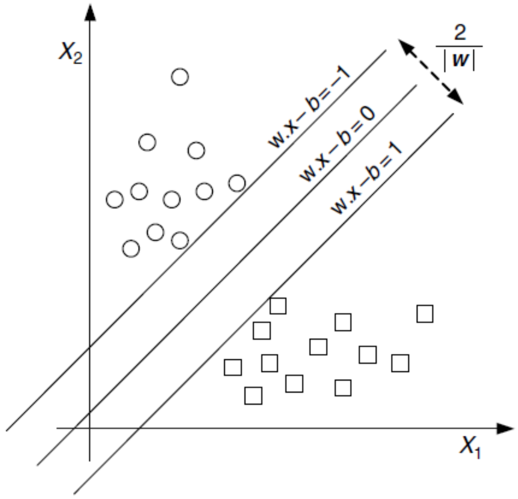


Figure 2.8 Maximum separation hyperplans [36].

2.4.3.4 K-Nearest-Neighbor Classifiers

K-nearest neighbor (KNN) [1] has been widely used as an effective classification model. K-NN is an example of instance-based learning, or lazy learning where the function is only approximated locally and all computation is deferred until classification [13]. The k-nearest neighbor algorithms classify an instance according to a majority vote of its *k* most similar instances [1]. Given a test instance *x*, its *k* closest neighborsy_1,…., y_k are found and a vote is conducted to assign the most common class to x. That is, the class of x, denoted by c(x), is determined by (2.11) [21].

$$C(x) = \arg\max_{c \in C} \sum_{i=1}^{k} \delta(c, c(y_i)) \qquad (2.11)$$

Where c(yi) is the class of yi, and δ is a function that δ (u; v) = 1 if u = v.

Usually "Closeness" is defined in terms of a distance metric, such as Euclidean distance where The Euclidean distance between two points , say, X1 = (x11, x12, : : : , x1n) and X2 = (x21, x22, : : : , x2n), is determined by (2.12) [17].

$$\mathrm{dist}(X1, X2) = \sqrt{\sum_{i=1}^{n} (x_{1i} - X_{2i})^2} \qquad (2.12)$$

2.4.3.5 The Naïve Bayes (NB) Algorithm

Bayesian classifiers are statistical classifiers. They can predict class membership probabilities. The Naive Bayes Classifier technique is based on Bayesian theorem. They assume that the effect of an attribute value on a given class is independent of the values of the other attributes [17].

The idea behind a Naive Bayes algorithm is the Bayes' Theorem and the maximum posteriori hypothesis. Bayes Theorem finds the probability of an event occurring given the probability of another event that has occurred already [2]. Given X=x1,x2,…,Xn be a feature vector x with n attributes values, and a class variable Cj, C=c1,c2,… cj Bayesian classifiers can predict class membership Cj with probabilities $P(X|Cj)$ for the feature vector X whose distribution depends on the class Cj . The class Cj for which the probability is given by$P(Cj|X)$ is called the maximum posteriori probability that feature vector x belongs, and can be computed from $P(X|Cj)$by Bayes rule:

$$P(Cj|x) = \frac{P(x|Cj)\,P(Cj)}{P(x)} \qquad (2.13)$$

It applies naïve conditional independence assumptions which states that all n features X=X1, X2, . . .Xn of the feature vector x are all conditionally independent of one another, given P(Cj), and Naïve Bayes assumption is calculated as follows:

$$P(x|C_j) = P(x1, x2, \ldots xn|C_j) = \prod_{i=1}^{n} P(x_i|C_j) \qquad (2.14)$$

$$P\left(C_j \middle| x\right) = \frac{P\left(C_j\right) \prod_{i=1}^{n} P\left(x_i \middle| C_j\right)}{P(x)} \tag{2.15}$$

The most probable hypothesis given the training data “Maximum a posteriori” hypothesis results in the following:

$$C_{max} = \arg\max_{C_m} P\left(C_j\right) \prod_{i=1}^{n} P\left(x_i \middle| C_j\right) \tag{2.16}$$

2.4.3.6 RIPPER

Repeated Incremental Pruning to Produce Error Reduction (RIPPER), proposed by William W. Cohen as an optimized version of IREP [23]. RIPPER is a rule-based learner that builds a set of rules that identify the classes while minimizing the amount of error.

2.4.4 Classification Performance

Estimating the accuracy of a classifier is important for future prediction accuracy and for comparing and choosing classifiers.

2.4.4.1 Confusion matrix

The confusion matrix is a useful tool for analyzing how classifier can recognize tuples of different classes [17]. A confusion matrix is an array which visualizes the performance of an algorithm. A confusion matrix for two classes is shown in Table 2.2 each row of the confusion matrix represents the instances in a predicted class, while each rows represent the actual (true) classifications.

Table 2.2 confusion matrix of a Two-Class Prediction [65].

Confusion matrix of a Two-Class Prediction			
		Actual Class	
		yes	no
Predicted Class	yes	True Positive (TP)	False Positive (FP)
	no	False Negative (FN)	True Negative (TN)

In the two-class case with classes yes and no, there are four different possible outcomes [65] [17].

- **True positives (TP)** True positives refer to the positive instances that were correctly labeled by the classifier.
- **True negatives (TN)** are the negative instances that were correctly labeled by the classifier.
- **False positive (FP)** is when the outcome is incorrectly predicted as yes (or positive) when it is actually no (negative).
- **False negative (FN)** is when the outcome is incorrectly predicted as negative when it is actually positive.

The performance of the classifier evaluates using the true positive rate, false positive rate and overall accuracy (OA) which are defined as follows:

True positive rate (TPR): also called hit rate or recall of a classifier is estimated by dividing the correctly classified positives (the true positive count) by the total positive count.

$$True\ positive\ rate = \frac{\mathbf{TP}}{\mathbf{TP + FN}} \tag{2.17}$$

False positive rate (FPR): also called false alarm rate of the classifier is estimated by dividing the incorrectly classified negatives (the false negative count) by the total negatives.

$$False\ Positive\ rate = \frac{\mathbf{FP}}{\mathbf{TN + FP}} \tag{2.18}$$

Overall accuracy (OA) of a classifier is estimated by dividing the total correctly classified positives and negatives by the total number of samples [44].

$$Accuracy = \frac{\mathbf{TP + TN}}{\mathbf{TP + TN + FP + FN}} \tag{2.19}$$

True Negative Rate (TNR): refer to the proportion of negative instances that are correctly identified [17].

$$\text{True Negative Rate} = \frac{\mathbf{TN}}{\mathbf{TN + FP}} \tag{2.20}$$

Precision: refer to the fraction of correctly retrieved instances [68].

$$\text{Precision} = \frac{\mathbf{TP}}{\mathbf{TP + FP}} \tag{2.21}$$

Recall: refer to the fraction of correctly retrieved instances out of all matching instances in the database [68].

$$\text{Recall} = \frac{\mathbf{TP}}{\mathbf{TP + FN}} \tag{2.22}$$

A classifier can achieve a very high accuracy by simply saying that all data are negative. It is thus useful to measure the classification performance by ignoring correctly predicted negative data. F measure is often used to measure the performance of a system when a single number is preferred [68].

F-measure: defined as the harmonic mean of precision and recall [68].

$$\mathbf{F = 2 \ * \frac{Precision * Recall}{Precision + Recall}} \tag{2.23}$$

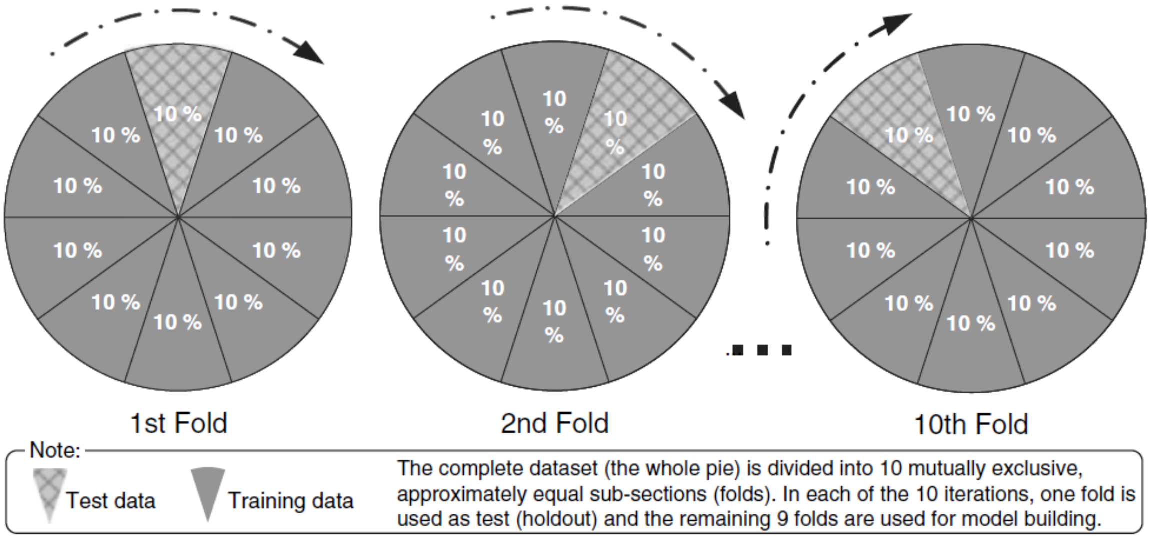
2.4.4.2 CROSS-VALIDATION

In order to obtain a reliable estimate of classifier accuracy cross validation is one of the common techniques for assessing accuracy based on randomly sampled partitions of the given data [17]. In practical terms, it is common to hold out one-third of the data for testing and use the remaining two-thirds for training [65]

In k-fold cross validation, the complete data set is randomly divided into k mutually exclusive subsets of approximately equal size. The classification model is trained and tested k times. Each time it is trained on all but one folds and tested on the remaining single fold. The overall accuracy of a model is calculated by simply averaging the k individual accuracy measures ((2.24) [35].

$$CVA \ = \frac{1}{K} \sum_{i=1}^{k} Ai \tag{2.24}$$

Where CVA stands for cross validation accuracy, k is the number of folds used, and A is the accuracy measure of each folds.



The complete dataset (the whole pie) is divided into 10 mutually exclusive, approximately equal sub-sections (folds). In each of the 10 iterations, one fold is used as test (holdout) and the remaining 9 folds are used for model building.

Figure 2.9 Pictorial representation of 10-fold cross validation [36]

In stratified k-fold cross validation, the folds are created in a way that class is represented in approximately the same proportions as in the full dataset. The k-fold cross validation is also called 10-fold cross validation, because the k taking the value of 10 has been the most common practice.

2.4.4.3 ROC

A receiver operating characteristics (ROC) curve is a technique for visualizing, organizing and selecting classifiers based on their performance [16] [15]. Figure 2.10 shows an example ROC curve. ROC curves depict the performance of a classifier without regard to class distribution or error costs [65]. The ROC curve is the plot of the true positive rate on the vertical axis against the true negative rate on the horizontal axis [65].

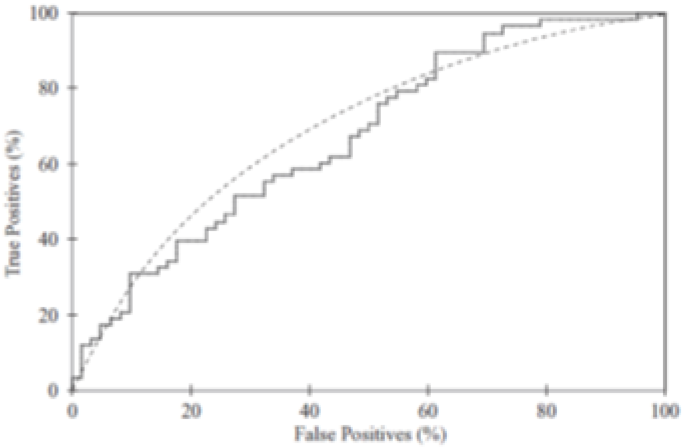


Figure 2.10 a sample ROC curve [65].

The diagonal line y = x represents the strategy of randomly guessing a class. For example, if a classifier randomly guesses the positive class half the time, it can be expected to get half the positives and half the negatives correct; this yields the point (0.5, 0.5) in ROC space. Any classifier that appears in the lower right triangle performs worse than random guessing. This triangle is therefore usually empty in ROC graphs [15]. Any classifier that produces a point in the lower right triangle can be negated to produce a point in the upper left triangle. Any classifier on the diagonal may be said to have no information about the class.

An ROC curve is a two-dimensional depiction of classifier performance. To summarize ROC curves in a single quantity and to compare classifiers we may want to reduce ROC performance to a single scalar value representing expected performance, a common method is to calculate the area under the ROC curve, abbreviated AUC roughly speaking, the larger the area the better the model [15] [65].

CHAPTER 3: Related Work

This chapter provides an overview of the related work of the thesis. There has been a significant amount of research in recent years to detect malicious; we focus on spyware detection in Windows portable execution (PE) file. Researchers apply different approaches in detection process based on data mining. Literature show that different types of features extracted such as Byte sequences N-gram based, API call and strings in binaries. In the following, we discuss these approaches. First, we discuss malware detection approaches and then we discuss spyware detection.

3.1 Malware detection

Schultz et al. [42] present a data mining framework which uses classifiers to detect new previously unseen malicious executables.

The goal of the work was to explore a number of standard data mining techniques to compute accurate detectors for new (unseen) binaries. The dataset consisted of a total of 4266 programs contained 3265 malicious and 1001 clean programs were no duplicate programs in the data set and every example in the set is labeled either malicious or benign. Three different types of features were statically extracted from the binary DLL usage information, Byte sequences, and strings extracted from the binaries.

The DLL usage information used three different features, the list of DLLs used by the binary, the list of DLL function calls made by the binary and the number of different function calls within each DLL to understand how resources affected a binary's behavior. RIPPER (a rule based system) applied to the DLL usage information features and provides an overall accuracy of detection 83.62%, 88.36% and 89.07% respectively.

Naive Bayes applied to strings features and give an overall detection accuracy of 97.11%. While Multi-Classifier system applied to byte sequences (n-grams) features. This scheme provides an overall detection accuracy of 96.88%. A signature-based method used for comparison, the result show that Signature Method with strings had the lowest False Positive Rate. Highest overall accuracy was the Naive Bayes algorithm with strings. And detection rate for the signature-based methods are lower than the data mining methods.

However the researcher explained that the string features are not robust enough and can be easily defeated by encoding or encrypting the strings in a file. Furthermore, research has not distinguished between packed and non-packed executables. Also this paper was published in 2001 and there was no spyware in their training or test dataset.

A similar work is done by Kolter and Maloof [25] present the use of machine learning and data mining to detect and classify malicious executables, their dataset consisted of 1971 benign and 1651 malicious executables, all were in the Windows PE format. They used the hexdump utility to convert each executable to hexadecimal codes in an ASCII format. Then they produced n-grams, by combining each four byte sequence into a single term. This processing resulted in 255904403 distinct n-grams then Information gain was used to choose top 500 n-grams as features, they evaluated different classifiers including Instance based Learner, naive Bayes, Support Vector Machines, decision trees, boost SVMs, J48, and naive Bayes. Best detector was reported using the boosted decision tree J48 algorithm with an area under the Receiver Operating Characteristic (ROC) curve of 0.996. Also they classify malicious executables based on the function of their payload they applying the classifiers only on 525 of the 1651 malicious and ROC curve for detecting payload function were in the neighborhood of 0.9.

However the researcher not taking into account the possibility effect of the executables they collected are packed or not, neglected the possible effect of packing on executable may produce over-optimistic results.

Wang et al. [64] use static analysis method to exploit the information in PE headers for the detection of malware. This work was based on the assumption that there would be difference in the characteristics of PE headers for malware and benign software as they were developed for different purposes. Their detection model consists of four stages, which include attribute extraction, attribute binarization, attribute elimination, and feature selection and classifier training. They have done experiments on a dataset that consists of 1908 benign and 7863 malicious executables. The malware samples contained viruses, email worms, Trojans and Backdoors. They collected most of the benign executables from Windows 2000 and XP operating system and several common user applications acquired from the web site PChome. PE headers were dumped using a program called DUMPBIN of all the files. Every header in the PE was considered as a potential attribute. Every field in the dataset was converted to binary value in the attribute binarization process. In elimination stage unimportant and redundant attributes were eliminated. After this elimination stage, all executables were converted into Boolean vectors according to the remaining attributes. The Support Vector Machines was used for classification and the detection accuracy calculated by using five-fold cross-validation.

The results of experiment when evaluated the performance without performing feature selection as an overall accuracy, 98.19%, 93.96%, 84.11% and 89.54% were obtained for virus, email worm, Trojans and backdoors respectively. However the result after removing useless and redundant attributes were 98.23%, 94.07%, 84.20%, 89.93% for Virus, email worm, Trojans and backdoors respectively.

However almost malware use packer and/or obfuscation techniques the research has not addressed the effect of packing on the executable; also the researcher explained that the detection performances of a Trojan and backdoor detections were not sufficient and need more improvement.

Moskovitch et al. [34] proposed the idea of using (Operation Code) OpCodes, generated by disassembling the executables for detection of unknown malicious code. They collected a dataset of malicious and benign executables for the Windows operating system; the dataset contains 7688 malicious files and 22,735 benign files. They presented a full methodology for the detection of unknown malicious code, based on text categorization concepts. A sequence of OpCodes extracted from each file representing execution flow of machine operations and several n-gram lengths were considered where each n-gram was composed of n sequential OpCodes which they term OpCode-n-grams. Three feature selection measures used as a baseline, the document frequency measure DF (the amount of files in which the term appeared in), Gain Ratio (GR) and Fisher Score (FS). Artificial Neural Networks (ANN), Decision Trees, Naïve Bayes, and their boosted versions, BDT and BNB, respectively were used for classification. They reported that greater than 99% accuracy can be achieved through the use of a training set that has a malicious file percentage lower than 15%.

Veeramani and Nitin [59] propose an automated framework for analyzing and classifying executables based on extracting relevant application programming interface (API) calls from sub categories of malware. They have done experiments on a dataset that consists of 210 malicious executables and 300 benign executables. They collected the benign executables from system32 folder in Windows XP system. The freely available unpackers tools are used to unpack the malware executables before disassembly then The Interactive Disassembler Pro (IDA Pro) has used to statically extract the list of API calls and a reference from the Microsoft Developer Network (MSDN) is used for matching and in identifying the windows API's. For feature selection the Document Class wise Frequency feature selection measure (DCFS) have used for selecting the relevant API calls for each malware category separately. The SVM classifier is used for classification, experiments were performed on various size of n-gram (1,2,3,4); These various values provide a detection accuracy of 97.23 %,94.47%,93.96 % ,and 91.70% respectively.

However the researcher used unpacking tools to unpack the malware executables only and do not consider the possibility of that benign executables may be packing. Furthermore, the experiment conducted on a relatively small set of malicious and benign PE files which may be quite unrepresentative of the malware as a whole.

3.2 Spyware detection

Bozagac [9] takes Schultz [42]'s work as a candidate and applies its techniques against a new spyware dataset collected in 2005. The purpose was specifically to see whether their techniques suitable of spyware detection. The dataset consists of 312 benign (non-malicious) executables and 614 spyware executables. Only the Naive Bayes algorithm was evaluated, and run the algorithm for a window size of 2 and 4 separately using 5-fold cross validation technique. He concludes that data mining based heuristic scheme has the potential to be used for detecting new spyware. These schemes provide an overall accuracy of 91.28% when using window size of 4.

However the researcher explained that the overall accuracy was insufficient and relatively high false positive rate. Furthermore, the researcher has not addressed packing effect on the executable.

T. Wang et al. [63] presented a surveillance spyware detection system by combining static and dynamic analyses; the system used APIs and DLL usage information as a static features and changes in registry, system files/folders and network states extracted by monitoring the running program in the virtual machine to detect spywares. They have done experiments on a dataset that consists of 407 spyware programs and 740 benign programs. These executables samples collected over a period of time. Using information gain method for feature selection, ranked total 790 features and finally selected 81 features, consisting of 67 static and 14 dynamic. An SVM was used to classify spywares from benign programs. They reported that the Overall Accuracy value does not exceed 85% when static analysis is used only and 95.20% when using dynamic analysis while they achieve an Overall Accuracy 97.9% when both static and dynamic analysis used together with a 0.68% false positive rate.

Researcher explained that both static analysis and dynamic analysis used in system; Experiments conducted on 81 features were 67 of these features was obtained from static analysis of executable. However researcher not taking into accounts the possibility effect of packing on executable. Furthermore the static analysis of the files is poor and has a low overall accuracy 81.69%.

Shazhad et al. [45] Present Spyware detection approach by using data mining technologies. Their objective is to determine whether spyware could be detected by classifiers generated from n-gram based data sets, they evaluate three different n-gram sizes (namely: 4, 5, and 6) for the experiments. The dataset consisted of a total of 137 binaries out of which 119 are benign and 18 are spyware binaries which focus on executable PE files for the Windows platform. Two different approaches of features extraction: the Common Feature-based Extraction (CFBE) and the Frequency-based Feature Extraction (FBFE). Two decision tree based algorithms J48 and Random Forest, one rule base algorithm JRip, one Bayesian network algorithm Naive Bayes, and one support vector machine algorithm SMO are applied. The ZeroR algorithm is used as a base line for all the experiments. The experiments results indicate that the approach is successful, achieving a 90.5 % overall accuracy with the J48 decision tree algorithm when using common n-gram feature selection method where n-gram size was 6.

However researcher explained the drawback of their research the overall accuracy 90.5 % is not sufficient and the small number of spyware files in the training or test dataset, play an important role to evaluate the result also the existence of high false positive rate.

Bahraminikoo et al. [5] present a Utilization of Data Mining to Detect Spyware. The goal of the work is to establish a method in Spyware detection research using data mining techniques. The research focus on analysis of executable files for the Windows platform. The Common Feature-based Extraction (CFBE) is used to extract features from the binary files. The sequences of bytes extracted from the hexadecimal dump of the binary files have been represented by n-grams. CFBE method is used to obtain Reduced Feature Sets (RFSs) which are then used to generate the ARFF files and includes instances from the frequency range of 50-80. A number of learning methods, namely BF Tree, Naïve Bayes, Random Forest and SMO are used for classification. The authors evaluate each learning algorithm by performing cross-validation tests to ensure that the generated classifiers are not tested on the training data. They report their result as an overall accuracy of 90.5% is achieved with the BF-tree algorithms.

The researcher explained their experiments have achieved 90.5% overall accuracy which is not sufficient also result shows that a high false positive rate. The details about the data sets (training, and testing data sets) are unclear, since it plays an important role to evaluate results.

3.3 Discussion and summary

In some of the above mentioned related works ([9], [25], [42], [45], and [5]), byte sequences (n-grams) analysis has been widely investigated and used as a static features, but n-grams would be ineffective feature for detecting encrypted and obfuscated malware, because of its sensitivity to order. Using strings extracted from the binaries as features as in [42] would be ineffective and not roust enough because the string and can be easily defeated by encoding or encrypting the strings in a file.

In [64] and [63] researcher do not consider the effect of packer and/or obfuscation techniques on the executable. Furthermore, researcher unpacks the malware executables only [59] and do not consider the possibility of that benign executables may be packing. The classifier may be biased in classification to distinguish between packed and non-packed instead of malware and benign executables as a result of when the training dataset contain packed malware file and non-packed benign.

In [45] and [5] the researcher explained their experiments have achieved relatively low overall accuracy which is not sufficient also result show that a high False Positive rate.

In this chapter we presented an overview about some of related non-signature based malware and spyware detection techniques. These techniques based on data mining classification techniques. We explained the strength and drawbacks of these methods used in relevant researches, which were many problems related to accuracy, detection rate and dataset.

CHAPTER 4: Data Collection and Preprocessing

This chapter provides a description about data collection, the Preprocessing steps used in spyware detection. Furthermore, this chapter describes feature extraction and feature selection.

4.1 Data Collection

In this section, we present an overview of the binary data set used in our study.

Due to no data has been published from previous studies; we had to build spyware data set by using two sources of spyware collections. The first is downloading from VX Heavens source [61], while the second source is downloading from vx-archiv website [62]. The benign files were gathered from various versions of Microsoft Windows family, including application software such as Image editing, Spreadsheet, Word processing and Windows system files and a wide range of portable benign tools. The entire data set consist of 3149 PE files where 2084 files were spyware and 1065 files were benign windows PE files.

4.2 Data Preprocessing

This section describes the overall methodology adopted in the research. The goal of our research was to explore a number of standards data mining techniques in order to create accurate detectors able to detect the new spyware file. We focus on static extraction of API call as features to distinguish between benign and spyware file. Figure 4.1 System Overview of Spyware detection methodology. gives an overview of the Spyware detection methodology.

Due to that, hackers can use packer to hide content of their malware also benign executables may use packing techniques in order to protect applications against cracking, therefore some of the files in the data set were either compressed or packed. These files could not be disassembled and therefore, these files must be unpacked before being disassemble.

Then each file is disassembled, and API call information extracted using as a feature and each file is represented as a vector of API call information.

In order to reduce the redundant attributes or dimensions of data, feature reduction was performed to enhance compactness of the feature vector.

Classifications algorithm applies to the representative vectors of the files after reduction (such as SVM, Random forest and j48). This step is divided into two subsequent phases: training and testing in the training phase the classifier will learn to classify the file as either benign or spyware, While in the testing phase, the performance is evaluated.

The detection methodology steps as the following:

Step 1: Unpack the packed PE file.

Step 2: Disassemble the binary executable and extract API calls.

Step 3: Selection of relevant API Calls.

Step 4: Classification algorithms apply to data set and training, validation and testing of the classifiers.

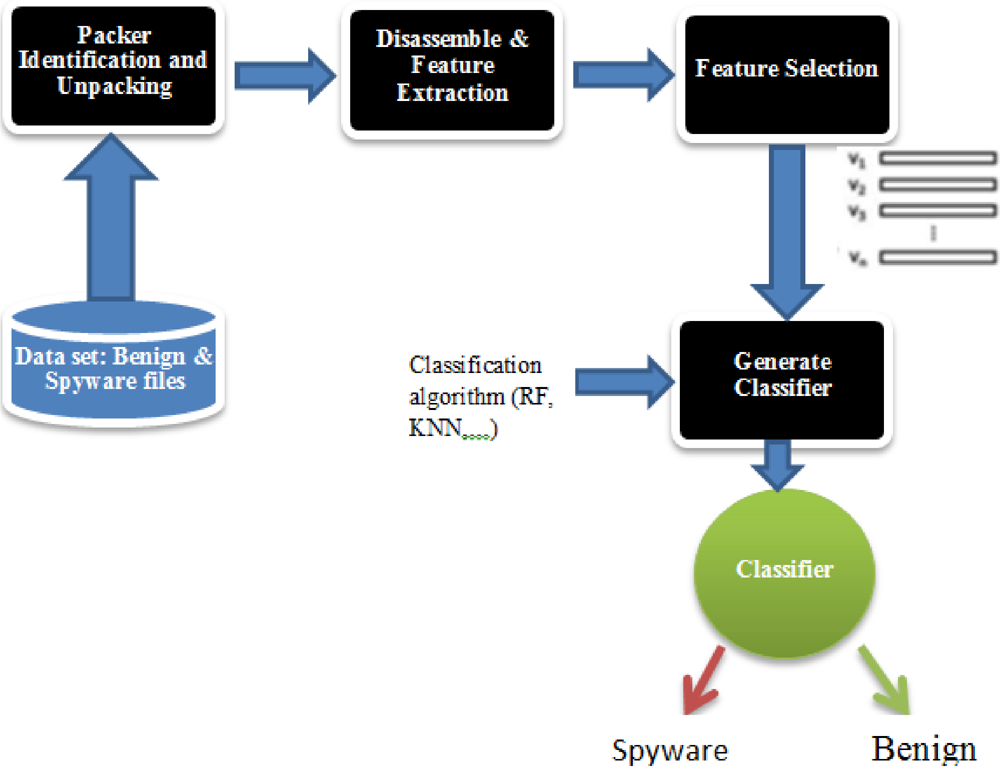

Packer Identification and Unpacking
Disassemble & Feature Extraction
Feature Selection
v_1
v_2
v_3
v_n
Data set: Benign & Spyware files
Classification algorithm (RF, KNN,....)
Generate Classifier
Classifier
Spyware
Benign

Figure 4.1 System Overview of Spyware detection methodology.

4.3 Step 1: Unpack the spyware.

Packers and code obfuscation often used by malware writers to make their code difficult to detect or analyze. Identification of right packer and using proper unpacking tools is essential to perform accurate analysis.

In our experiment conducted on 2084 samples of spyware. Identification of right packer is the core of this step. To identify the packer used in PE file this is achieved by using the packer identification tool. We used two tools to detect the PE packer PEiD [48]and ExEinfo PE [49], PEiD is a detector used for most common packers, cryptors, compilers and even signature-based packer detection in PE files. Table 4.1 illustrates the distribution of various obfuscation packers on the spyware collection.

Table 4.1 Packers/Compilers Analysis of Spyware.

Packer/ compilers Name	Number of Spyware	Percent
Microsoft (Visual Basic + Visual C++)	777	37.28%
UPX	391	18.76%
Borland Delphi	315	15.12%
Nothing found	323	15.50%
Upack	82	3.93%
FSG	55	2.64%
PECompact	53	2.54%
ASPACK	51	2.45%
NSPack	19	0.91%
Asprotect	18	0.86%

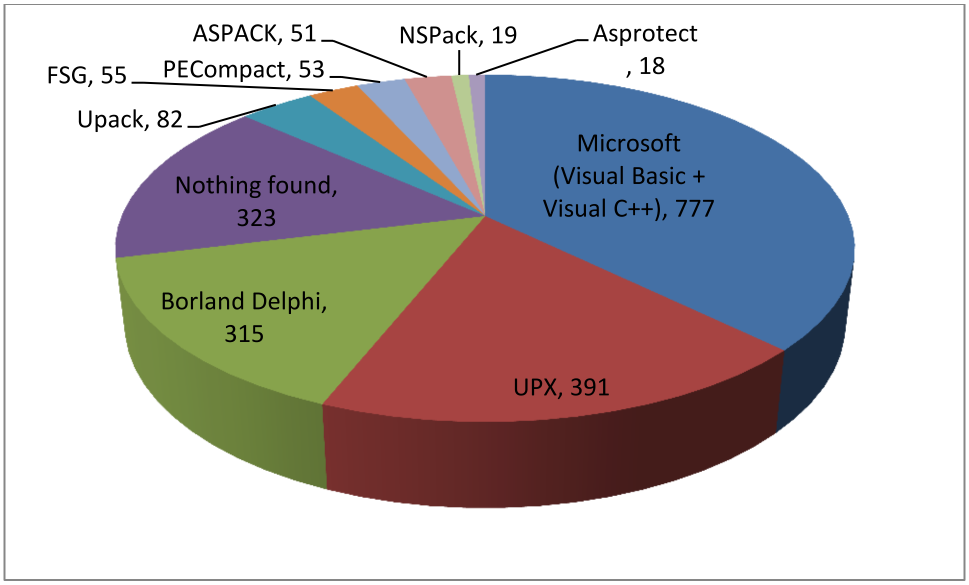


Figure 4.2 Spyware Packer/ compilers distribution.

The result in Table 4.1 and Figure 4.2 show that about 47% of the spyware files packed, we can determine that the majority of spyware change their signature by applying packing techniques to evade detection by signature scanners. However, approximately 15% spyware PE files cannot be classified as either packed or non-packed.

Since a packed file is a type of archive file and not all packed files are bad. Therefor the benign files may be packed; we use PEiD to identify the packer in the benign files.

After Identification of right packer we used a number of most popular packing/unpacking tools such as UPX, ASPack, PECompact, etc. to unpack the PE files to make the PEs recognizable by the PE disassembler.

4.4 Step 2: Disassemble the binary executable and feature extraction

In this step, we statically extracted the features that represented different information contained within each binary file.

User-level malware programs require the invocation of system calls to interact with the OS in order to perform malicious actions. Therefore, analyzing and extracting malicious behaviors from these programs requires the identification of system calls invoked within the code [44]. We employ the most reliable disassembly tool for static analysis, namely, Interactive Disassembler Pro (IDA Pro) [18]. It automatically recognizes API calls for various compilers. IDA Pro loads the selected file into memory to analyze the relevant portion of the program. After disassembling the PE file the information extracted from it.

The total of 21390 distinct API calls was extracted from the 3149 PE file called from 373 distinct DLL. After this, data set is available for feature generation and feature selection.

4.5 Feature Extraction

This section explains the feature extraction and provides a description of the feature sets used in this research.

In order to understand how resources affected a PE file behavior, we statically extracted different features, these features used to construct five sets of features that represented different information contained within each binary. These sets of features were then used by different algorithms to produce detection models.

1. The list of DLLs used by the PE file.

2. The number of different API calls the PE file has imported from the corresponding DLL.

3. The number of different API function calls the PE file has used from API call categories.

4. The list of selected API function calls used by the PE file.

5. A combination of Selected API calls categories and a list of select API function calls.

All sets of features contain the same number of records 3149 records, but different in the number and type of features. Since the task is to detect spyware executables, we refer to the spyware class as the positive class and refer to the benign class as the negative class.

In the first features set list of DLLs used by the PE file as features. Every PE files benign and spyware represented as a binary vector of DLL used by a file, based on the DLL used or not by the PE file. For example, if a PE file imports a DLL, then the corresponding entry in the vector that represents the executable file set to one. For vector namely V represent a PE file X, where Vi = 1 if and only if the PE file has used the ith DLL and Vi = 0 if corresponding PE file has not used the DLL; (4.1 presents as features representation for the PE Files.

$$Vi_{\textbf{File x}} = \begin{cases} 1, & \text{if DLLi is used by File x} \\ 0, & \text{otherwise} \end{cases} \tag{4.1}$$

This processing resulted in feature vector of 373 binary features representing each file.

While in the second features set, the number of different API calls the PE file has imported from the corresponding DLL. Accumulative of features are used, every PE file is represented as a vector of the count of different functions called from each DLL not the number of times API functions might have been called. So that the value of each element in the vector is equal to the count of API calls the PE file has imported from the corresponding DLL. For example if the PE file import two distinct functions from DLL library (e.g. WSOCK32.DLL) and three distinct functions from DLL library (e.g. Kernel32.dll) the value of the element corresponds to the DLL in vector V that represent the file would be as V= {2,3,……..}.This processing resulted in 373 features.

In the third features set, API call categories are used as a feature, the categorization made by Microsoft [35] is used to categorize the API call which used by [41]. Every PE file is represented as a vector of API calls categories and the value of each element in this vector is equal to the number of API calls that PE file has used from the corresponding category. This processing resulted in 95 features.

In features set 4: the list of selected API function calls in the PE file. In this, features set the vector is constructed from function call name and their DLL name which the function came from used as features. Because this method can generate large number of features which are not discriminative, we decide to use the functions that more relevant which are come from the top discriminative DLL results from features set 2 after selections. This processing resulted in a binary vector with 5096 features.

In features set 5: the feature vector is constructed from a combination of discriminative API calls features result from features set 4 and a discriminative API call categories result from features set 3. This processing resulted in a vector with 839 features.

4.6 Feature Selection

The five sets of features created are considered as the initial set of features. In order to achieve maximum performance and improve the detection accuracy of classifiers with the minimum measurement and reduce the processing overheads in training and testing of classifiers, it makes sense to remove features that might not convey useful information.

We examine three different feature ranking algorithms such as: Information Gain; Information Gain Ratio and Support Vector Machine algorithms in order to find suitable features ranking, in our study, the overall accuracy is used to select the best feature ranking algorithm. We do our experimental on features Set 1 “list of DLLs used by the PE file” and features Set 2 “The number of different API calls the PE file has imported from the corresponding DLL”. Experimental results on features Set 1 with Random forest and J48 classifiers are shown in Table 4.2 and Figure 4.3.

Table 4.2 Experiments results on features Set 1 using different ranking algorithms.

Ranking algorithm	Classifier	Accuracy after evaluation
Weight by Information Gain	Random forest	86.41%
	J48	81.90%
Weight by Information Gain Ratio	Random forest	85.65%
	J48	81.90%
Weight by SVM	Random forest	86.88%
	J48	84.03%

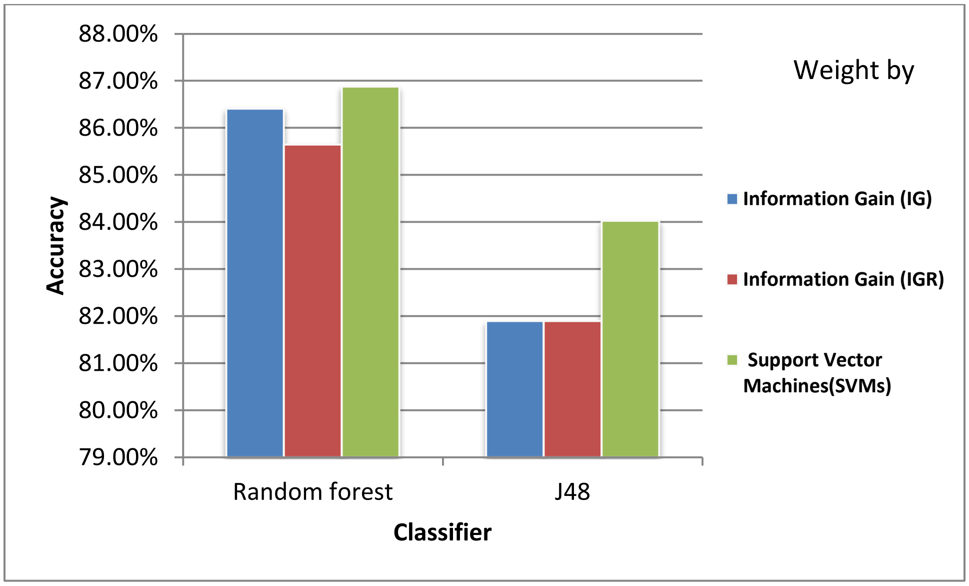


Figure 4.3 Experiments results on features Set 1 using different ranking algorithms.

The experimental results on features Set 2 “The number of different API calls the PE file has imported from the corresponding DLL” with Random forest and J48 classifiers are shown in Table 4.3 and Figure 4.4.

Table 4.3 Experiments results on features Set 2 using different ranking algorithms.

Ranking algorithm	Classifier	Accuracy after evaluation
Weight by Information Gain	Random forest	92.03%
	J48	89.71%
Weight by Information Gain Ratio	Random forest	92.25%
	J48	89.62%
Weight by SVM	Random forest	92.98%
	J48	89.65%

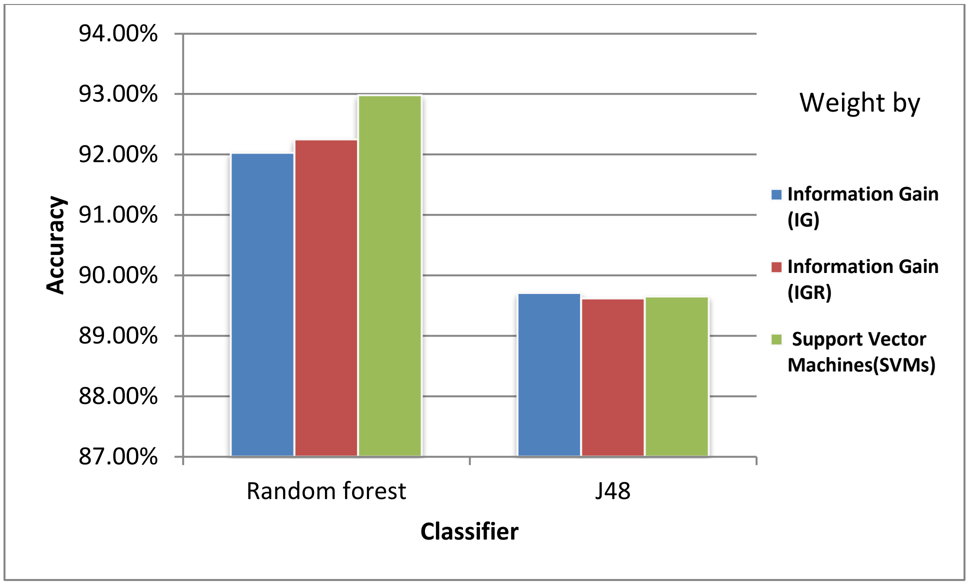


Figure 4.4 Experiments results on features Set 2 using different ranking algorithms.

After evaluating of the three types of feature ranking algorithms, results are shown in Table 4.2 and Table 4.3 suggest that, Weight by SVM achieved the best accuracy. We adopt the Weight by SVM Ranking algorithm in our experimental on all features sets.

The decision of which features to be removed was based on experimental using SVM, Table 4.4 summarize the result of feature selection.

Table 4.4 The result of feature selection

Features Set	weight relation	weight	Total attributes	Number of Selected attributes	Selected attributes Percentage
Features Set 1	greater	0.01	373	268	71.8%
Features Set 2	greater	0.013	373	236	63.2%
Features Set 3	greater	0.01	95	67	70.5%
Features Set 4	greater	0.13	5096	581	11.40%
Features Set 5	Top K	240	648	240	37.0%

In the first features set “list of DLLs used by the PE file”; out of 373 attributes, 286 important attributes were selected by selecting the feature that weights greater than 0.01. While in the second features set “The number of different API calls the PE file has imported from the corresponding DLL” in total of 373 attributes 236 attributes were selected which having weights greater than 0.013. In features set 3 “the number of different API function calls the PE file has used from API call categories” from 95 attributes, the 67 attributes were selected, which having weights greater than 0.01. In total of 5096 attributes were obtained in features set 4 “the list of selected API function calls used by the PE file” the 581 important attributes were selected by selecting the feature that weights greater than 0.013. While in features set 5 “A combination of selected API calls categories and a list of select API function calls” from 648 attributes the attributes the top 240 attributes were selected.

CHAPTER 5: Experiments and Results analysis

This chapter describes experiments performed in this research. The primary objective of our research is to distinguish between benign and spyware PE files, to achieve this goal, we have done the experiments on the five features sets.

Different supervised learning classification algorithms used such as (Random forest, KNN, Naive Bayes, Support Vector Machine, Decision Tree, JRip) on the features sets.

To check the generalizing ability of the trained classifiers, stratified tenfold cross validation is used, in this technique, the dataset is randomly divided into 10 sets of size n/10, train on nine datasets and test on one then repeat 10 times and take a mean accuracy, where each class is represented in approximately the same proportion as in the full data set [65]. We have compared our result with technique proposed by Shazhad et al. [45] and Wang et al [63]. The overall accuracy and AUC, F-measure are used to determine the detection accuracy of each approach.

5.1 Experimental Environment and Tools

All of our experiments done on an Intel(R) Core i3 2.10GHz processor with 4 GB RAM. The Microsoft Windows XP SP2 is installed on VMware Workstation Ver. 7.0.0 build-203739 environment.

Rapid Miner 5.2.008 with Weka Extension [40] is used to conduct our experiments practical and extracting the required results.

VMware [60]: that provides virtualization software for operating system.

5.2 Performance Evaluation Metrics

To evaluate our approach, we use the following estimates to evaluate the performance of the proposed approach:

- True Positive (TP): Number of correctly detected spyware files.
- False Positive (FP): Number of wrongly detected benign files.
- True Negative (TN): Number of correctly detected benign files.

- False Negative (FN): Number of wrongly detected spyware files.
- Detection Rate (recall) (DR): Percentage of correctly detected spyware files.

In addition to Overall Accuracy, which measures the number of absolutely correctly classified instances, either positive or negative, divided by the entire number of instances, The AUC (Area Under ROC Curve) is used to determine the detection accuracy of each approach. ROC curves are insensitive to changes in class distribution. If the proportion of positive to negative instances changes in a test set, the ROC curves will not change [15]. Also ROC graphs are enable visualizing and organizing classifier performance without regard to class distributions or error costs [16].

True Positive Rate (TPR): Percentage of correctly identified spyware files.

False Positive Rate (FPR): Percentage of wrongly identified benign files.

5.3 Algorithm Configuration

In this sub section, we provide a list of algorithms used in our study and their parameter, were as the parameter not listed used with their default values.

K-Nearest-Neighbor (k-NN)

We use k-nearest neighbor algorithm implemented in Rapidminer. The value of k (number of nearest neighbors) is set to be 5, and the weighted vote have not used. The measure type is set to be numerical with Euclidean distance measures.

Decision Tree (J48)

We use Weka: W-J48 that is weka implementation of C4.5 decision tree classifier in Rapidminer with Weka Extension. The default parameters for J48 have been used. The confidence factor for pruning is set to 0.25, where smaller values incur more pruning. The minimum number of instances per leaf equals 2 the default value. Since all of our selected features are numeric except the class labels we do not utilize binary splits on nominal attributes for building trees.

Decision tree

We use decision tree classifier that implemented in Rapidminer this decision tree learner works similar to Quinlan's C4.5 or CART. The tree induction algorithm works as follows.

Whenever a new node is created at a certain stage, an attribute is picked to maximize the discriminative power of that node with respect to the examples assigned to the particular subtree. This discriminative power is measured by a criterion, e.g. using (information gain, gain ratio, gini index, etc.) [40].

In our study, the information gain is used as a criterion for selecting attributes and numerical splits where the minimal gain which must be achieved in order to produce a split is set to 0.01. The minimal size of a node in order to allow a split is set at 4. The minimal size of all leaves is set at 2. The maximum tree depth is set to 20 and the confidence level used for the pessimistic error calculation of pruning is set at 0.25. The number of alternative nodes tried when prepruning would prevent a split is set to 3. The pre pruning and pruning are enabling.

Naïve Bayes

We use default parameters for Naïve Bayes classifier that implemented in Rapidminer. The Laplace correction to prevent high influence of zero probabilities is set to be true. While on Naive Bayes (Kernel) The Laplace correction to prevent high influence of zero probabilities is to be true. The kernel density estimation mode (estimation mode) is set to be greedy the default value. Where Minimum kernel bandwidth equals 0.1 and number of kernels equals 10.

Random Forest

We use Weka: W-Random Forest classifier in Rapidminer with Weka Extension. We investigate the accuracy of the classifier at two different value of number of tree to build parameter 10 and 100, where the maximum depth of the trees is set to be 0 allowing unlimited depth.

Support Vector Machine (LibSVM)

We use the default parameters for Support Vector Machine (LibSVM) that implemented in Rapidminer with a linear kernel type. The cost parameter is set at 0. The tolerance of termination criterion is set at 0.0010. The weights are set to 1 for all classes where the weight was not defined. Using shrinking heuristics and confidence for multiclass parameters are selected.

JRip (RIPPER)

We use default parameters for Weka: W-JRip WEKA implementation of RIPPER algorithm using Rapidminer with Weka Extension tool. The training set is divided into 3 types, one is used for pruning and the rest are used for growing the rules. Where the minimum weight of instances within a split is set to 2 and the number of runs of optimization is set to be 2.

Table 5.1 Summarizes algorithms configuration.

Table 5.1 Algorithm Configuration

	Classification Algorithm	Parameters
1	Weka:W-RandomForest	Number of trees to build: 10 (Default), The maximum depth of the trees: 0 (default)
		Number of trees to build: 100, The maximum depth of the trees:0 . (default)
2	K-Nearest-Neighbor (k-NN)	k: number of nearest neighbors:5 measure types: numerical measure numerical measure: Euclidean distance Weighted vote: unselected.
3	Naive Bayes	laplace correction: Use Laplace correction to prevent high influence of zero probabilities: true (default)
4	Naive Bayes (Kernel)	laplace correction: true (default) estimation mode: greedy (default) bandwidth: 0.1 number of kernels: 10
5	Support Vector Machine (LibSVM)	svm type: C-SVC. kernel type: linear. cost parameter C: 0.0 cache size: 80 (default) epsilon: 0.0010 (default) shrinking: true (default) calculate confidences: false (default) confidence for multiclass: true (default)
6	Weka:W-J48	Use unpruned tree: false (default) confidence threshold for pruning: 0.25 (default) minimum number of instances per leaf:2 (default)
7	Weka:W-JRip	Number of folds for REP One fold is used as pruning set. Default value: 3.0 Minimal weights of instances within a split. Default value: 2.0 Number of runs of optimizations. Default value: 2.0 The seed of randomization Default value: 1.0 Whether NOT check the error rate>=0.5 in stopping criteria Default value:

		false Whether NOT use pruning. Default value: false
8	Decision Tree	criterion: information gain minimal size for split: 4 (default) minimal leaf size: 2 (default) minimal gain: 0.01 maximal depth: 20 (default) confidence: 0.25 (default) number of prepruning alternatives: 3 (default)

5.4 Experimental Results

In this section, five different groups of experiments have been constructed. These experiments are grouped according to features sets presented in chapter 4; in each group, nine classification algorithms apply in order to determine the best set of features and the most appropriate classification algorithm. The details of these experiments are explained as follows:

5.4.1 Experiment on features set 1 “list of DLLs used by the PE file”.

We performed our first set of experiments on the features set 1 in order to measure the efficiency of features set 1; nine experiments have been done. This features set had 268 attributes. Results are shown in Table 5.2, Figure 5.1 and Figure 5.2. Feature set produced as results of the list of DLLs used by the PE file have shown that Random Forest with 100 trees has yielded the highest overall accuracy 87.30%. It’s slightly better than Random Forest with 10 trees 86.88%. KNN has lowest accuracy level 79.52%. The difference between higher and lowest accuracy is 7.78.

Table 5.2 Experiments results on features Set 1.

Algorithm	TPR (Recall)	FPR	Precision	Overall Accuracy	F-measure	AUC
Random forest 10	95.01	13.50	86.50	86.88	90.56	0.931
Random forest 100	95.25	13.17	86.83	87.30	90.85	0.938
KNN	99.14	23.28	76.72	79.52	86.50	0.591
Naive Bayes	99.42	23.23	76.77	79.71	86.64	0.411
Naive Bayes(kernal)	98.08	20.31	79.69	82.18	87.93	0.893
Support Vector Machine (LibSVM)	97.17	19.13	80.87	82.92	88.27	0.889
W-J48	92.37	15.16	84.84	84.03	88.44	0.894
JRip	91.27	17.63	82.37	81.30	86.59	0.779
Decision tree	93.91	15.13	84.87	84.88	89.16	0.908

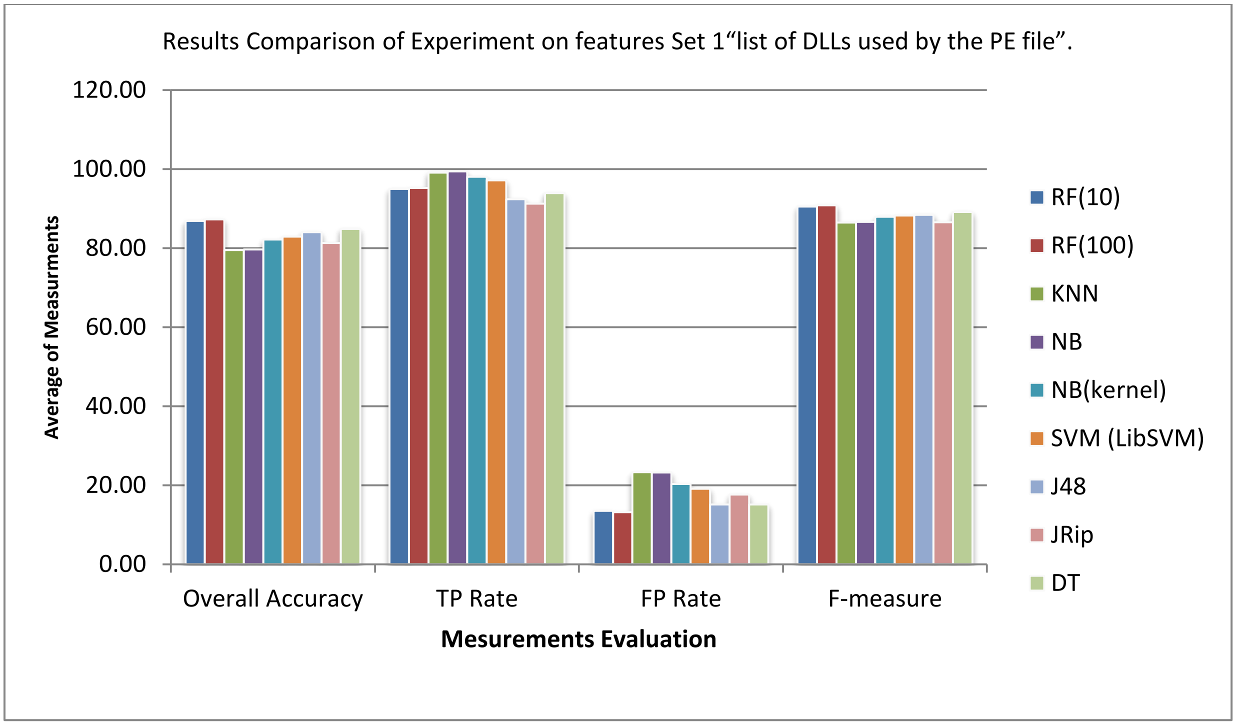


Figure 5.1 Experiments results on features Set 1.

For the Area under ROC Curve, Random Forest with 100 trees 0.938 is best, it slightly better than Random Forest with 10 trees 0.931; the lowest AUC is given by Naive Bayes 0.411. It is clear from table and graphs that results of Random Forest with 100 trees had the highest overall accuracy 87.30% and AUC 0.938, True Positive Rate 95.25% with the lowest false positive rate at 13.17% and F-measure 90.85 are better than other classifiers.

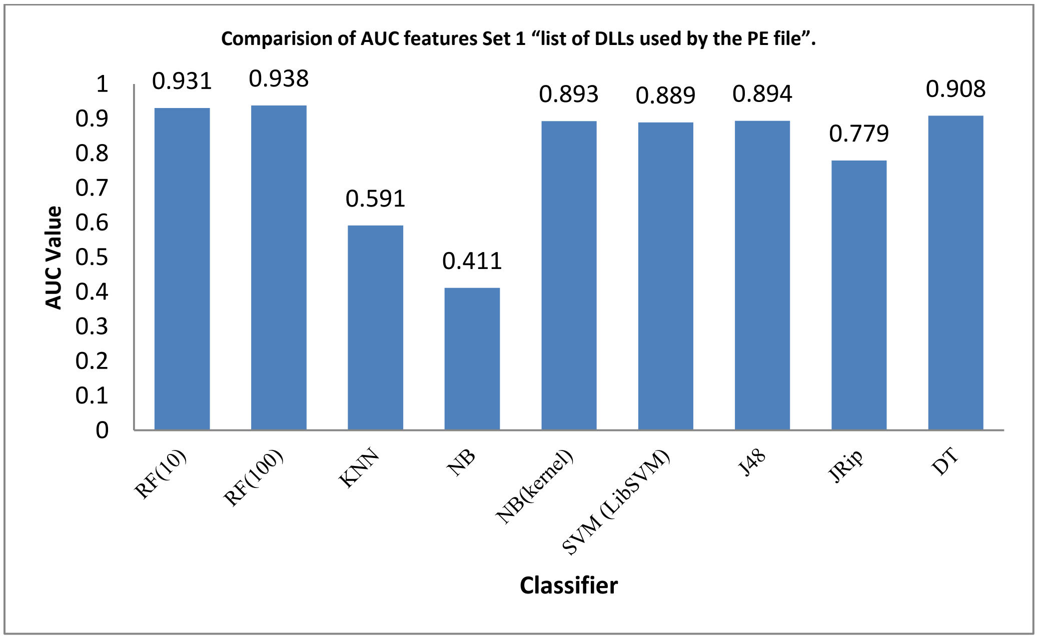


Figure 5.2 Comparison of AUC on features Set 1.

5.4.2 Experiment on features set 2 "number of different API calls the PE file has imported from the corresponding DLL".

To measure the efficiency of features set 2, feature set produced as results of the number of different API calls the PE file has imported from the corresponding DLL, This features set had 236 attributes. Nine experiments have been done. Table 5.3, Figure 5.3 and Figure 5.4, illustrates experiment results. The Random forest algorithm with 10 Trees had the highest overall accuracy 92.98% and highest F-measure 94.73 with low false positive rate at 5.87%. Whereas Support Vector Machine (LibSVM) had the highest True Positive Rate 98.51% with lowest overall accuracy 80.47% and high false positive rate 22.15%.

Table 5.3 Experiments results on features Set 2.

Algorithm	TPR (Recall)	FPR	Precision	Overall Accuracy	F-measure	AUC
Random forest 10	95.35	5.87	94.13	92.98	94.73	0.968
Random forest 100	96.02	6.54	93.46	92.92	90.85	0.977
KNN	94.48	13.49	86.51	86.60	90.32	0.867
Naive Bayes	99.42	22.51	77.49	80.50	86.64	0.439
Naive Bayes(kernal)	94.81	16.62	83.38	83.53	87.93	0.799
Support Vector Machine (LibSVM)	98.51	22.15	77.85	80.47	88.27	0.873
W-J48	93.71	9.08	90.92	89.65	88.44	0.902
JRip	93.71	10.90	89.10	88.25	86.59	0.867
Decision tree	93.81	9.57	90.43	89.33	89.16	0.874

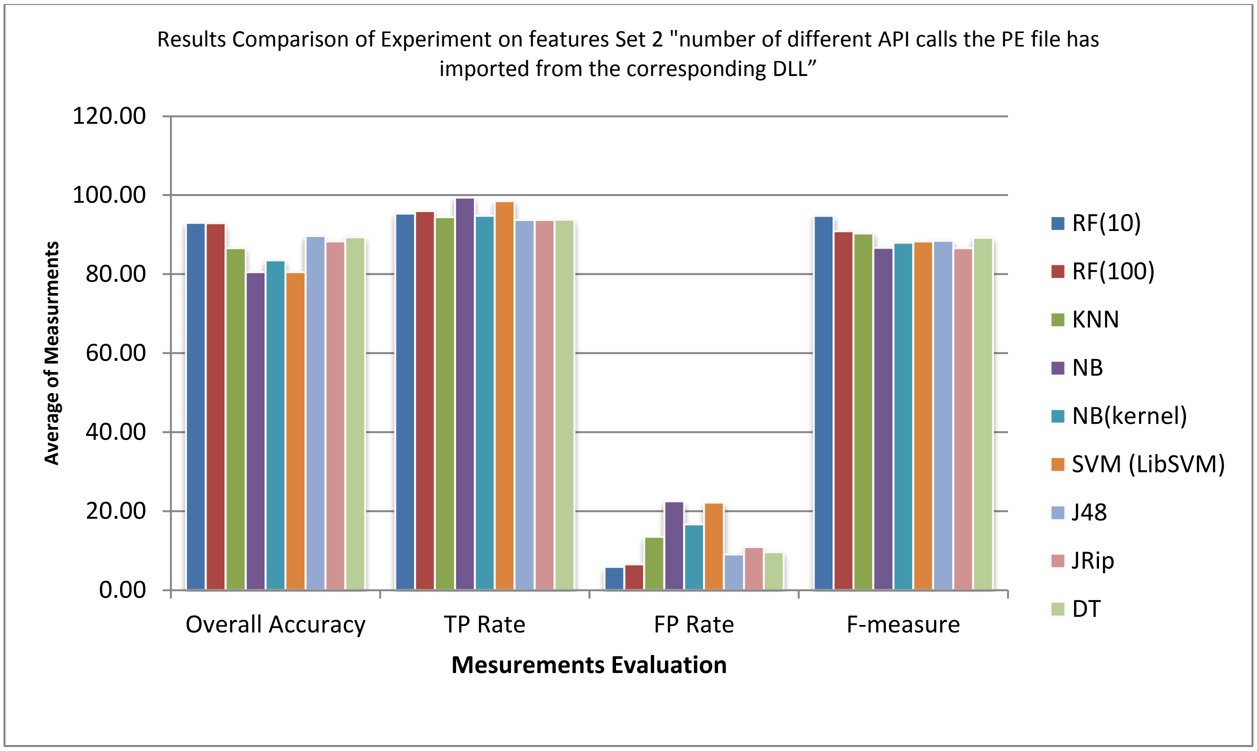


Figure 5.3 Experiments results on features Set 2.

For the Area under ROC Curve, Random Forest with 100 trees 0.977 is best, it slightly better than Random Forest with 10 trees 0.968; the lowest AUC is given by Naive Bayes 0.439.

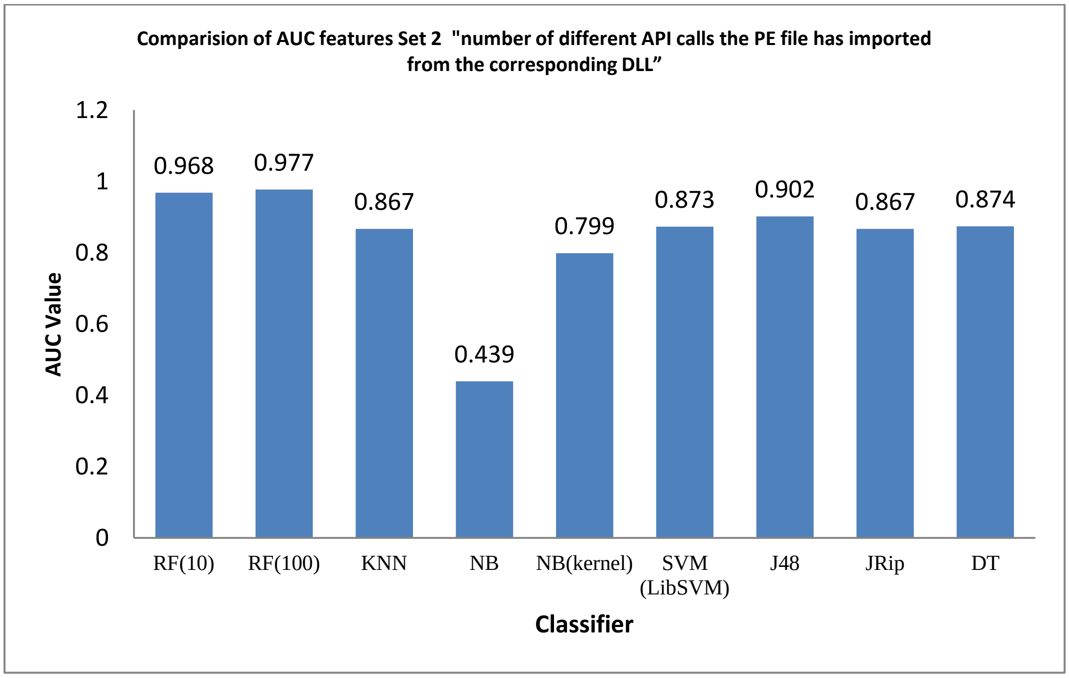


Figure 5.4 Comparison of AUC on features Set 2.

5.4.3 Experiment on features set 3 “The number of different API function calls the PE file has used from API call categories”.

We conducted our third experiments on features set 3. This features set had 67 attributes. Results for experimental on features set 3 are shown in Table 5.4 and Figure 5.5; Figure 5.6 show a comparison of AUC on features Set. Feature set produced as results of the number of different API function calls the PE file has used from API call categories. The experimental showed that Random forest algorithm with 100 trees has highest overall accuracy 95.49%, among all. Naive Bayes have achieved lowest level 75.83%.

The highest TPR is given by Random forest algorithm with 100 trees 98.70%. High TPR with Low FPR is given by Random Forest with 100 trees. Highest FPR is given by Support Vector Machine (LibSVM) 13.73%.

Table 5.4 Experiments results on features Set 3.

Algorithm	TPR (Recall)	FPR	Precision	Overall Accuracy	F-measure	AUC
Random forest 10	98.13	5.24	94.76	95.17	96.42	0.982
Random forest 100	98.70	5.29	94.71	95.49	96.66	0.985
KNN	97.22	8.08	91.92	92.51	94.50	0.884
Naive Bayes	63.92	0.67	99.33	75.83	77.78	0.943
Naive Bayes(kernel)	95.49	8.01	91.99	91.49	93.71	0.956
Support Vector Machine (LibSVM)	98.56	13.73	86.27	88.66	92.00	0.956
W-J48	98.08	6.54	93.46	94.19	95.72	0.965
JRip	98.32	7.87	92.13	93.33	95.13	0.914
Decision tree	98.18	7.21	92.79	93.74	95.41	0.959

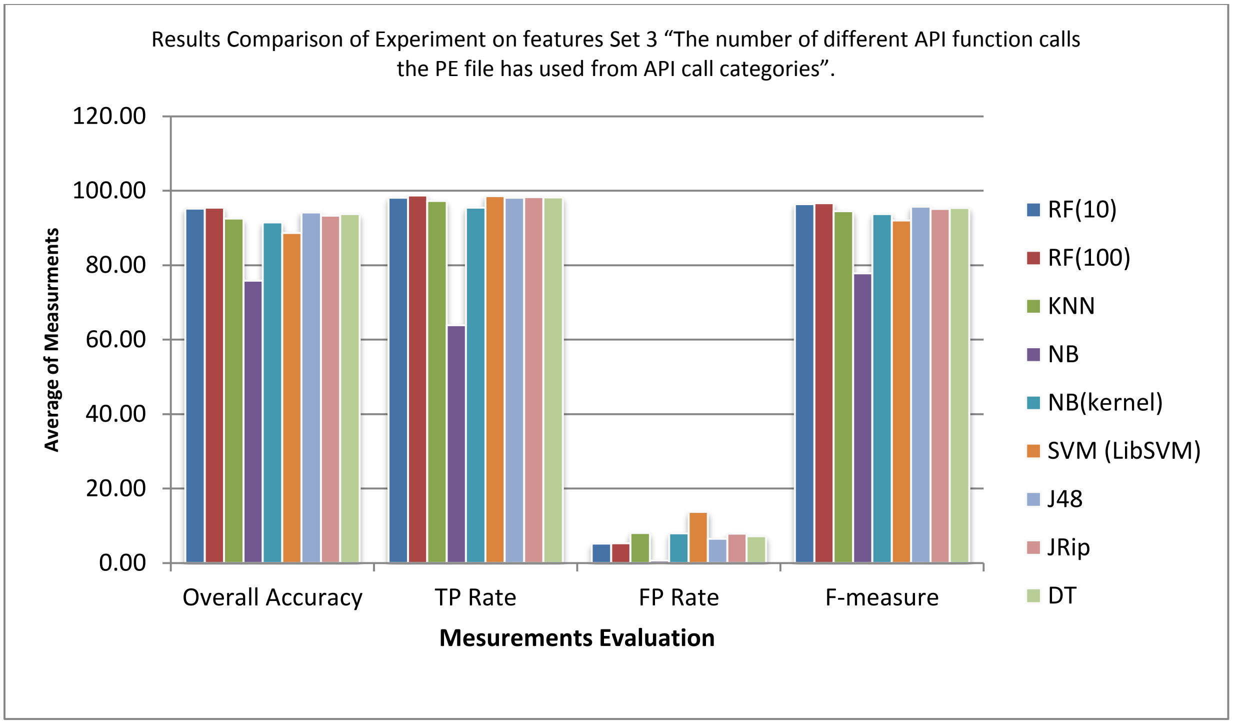


Figure 5.5 Experiments results on features Set 3.

For the Area under ROC Curve, Random Forest with 100 trees performed is the highest 0.985 while the KNN have the lowest results 0.884.

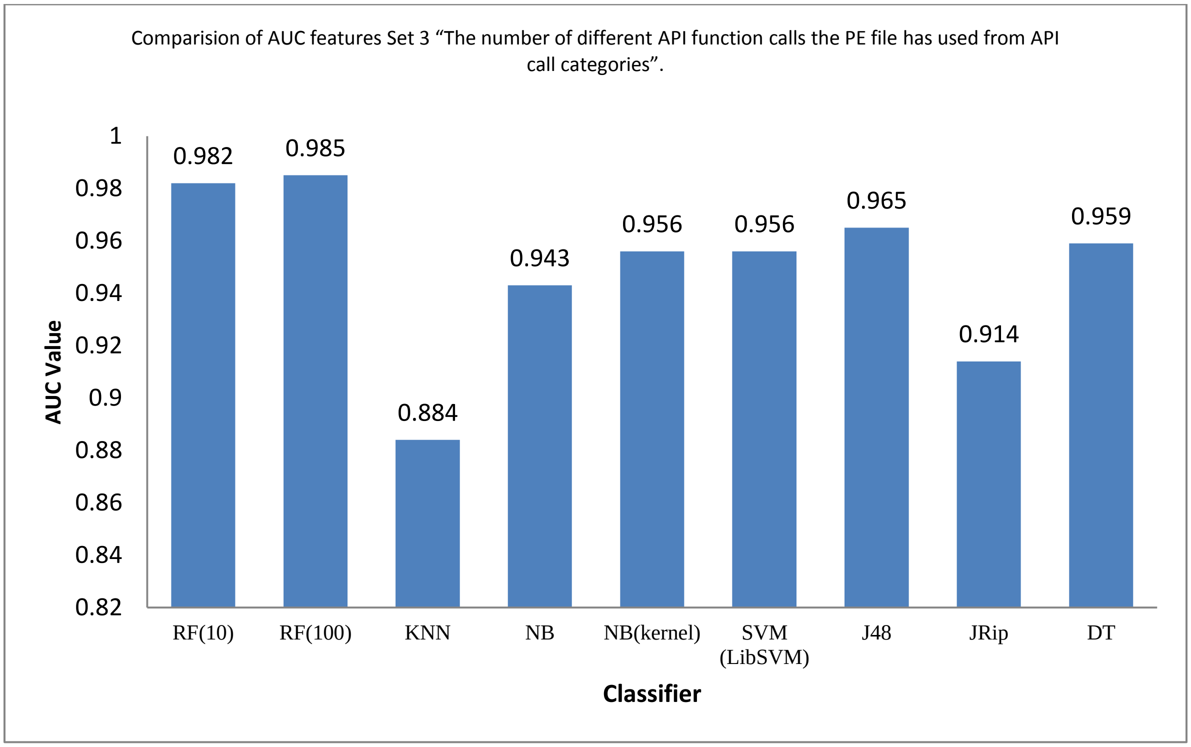


Figure 5.6 Comparison of AUC on features Set 3.

5.4.4 Experiment on features set 4 “The list of selected API function calls used by the PE file”.

Results for experimental on features set 4 are shown in Table 5.5 and Figure 5.7 and Figure 5.8 feature set produced as results of the list of selected API function calls used by the PE file. This features set had 581 attributes. The results showed that Random forest algorithm with 100 trees has highest overall accuracy 96.06% among all. It’s slightly better than Random Forest with 10 trees 95.81%. Naive Bayes (kernel) have achieved lowest level 86.82%. The difference between higher and lowest performance is 9.24%.

The highest TPR is given by Random Forest with 100 trees 97.60%. It’s slightly better than Random Forest with 10 trees. Lowest FPR is given by Random Forest with 10 trees 3.39%. Lowest TPR with highest FPR is given by Naive Bayes (kernel).

Table 5.5 Experiments results on features Set 4.

Algorithm	TPR (Recall)	FPR	Precision	Overall Accuracy	F-measure	AUC
Random forest 10	97.07	3.39	96.61	95.81	96.84	0.981
Random forest 100	97.60	3.51	96.49	96.06	97.04	0.986
KNN	97.02	7.33	92.67	92.95	94.80	0.905
Naive Bayes	92.37	7.72	92.28	89.84	92.33	0.85
Naive Bayes(kernel)	89.68	9.67	90.33	86.82	90.01	0.877
Support Vector Machine (LibSVM)	97.36	7.56	92.44	92.98	94.84	0.975
W-J48	96.79	5.57	94.43	94.09	95.59	0.937
JRip	95.20	6.28	93.72	92.60	94.45	0.924
Decision Tree	96.45	5.72	94.28	93.78	95.35	0.943

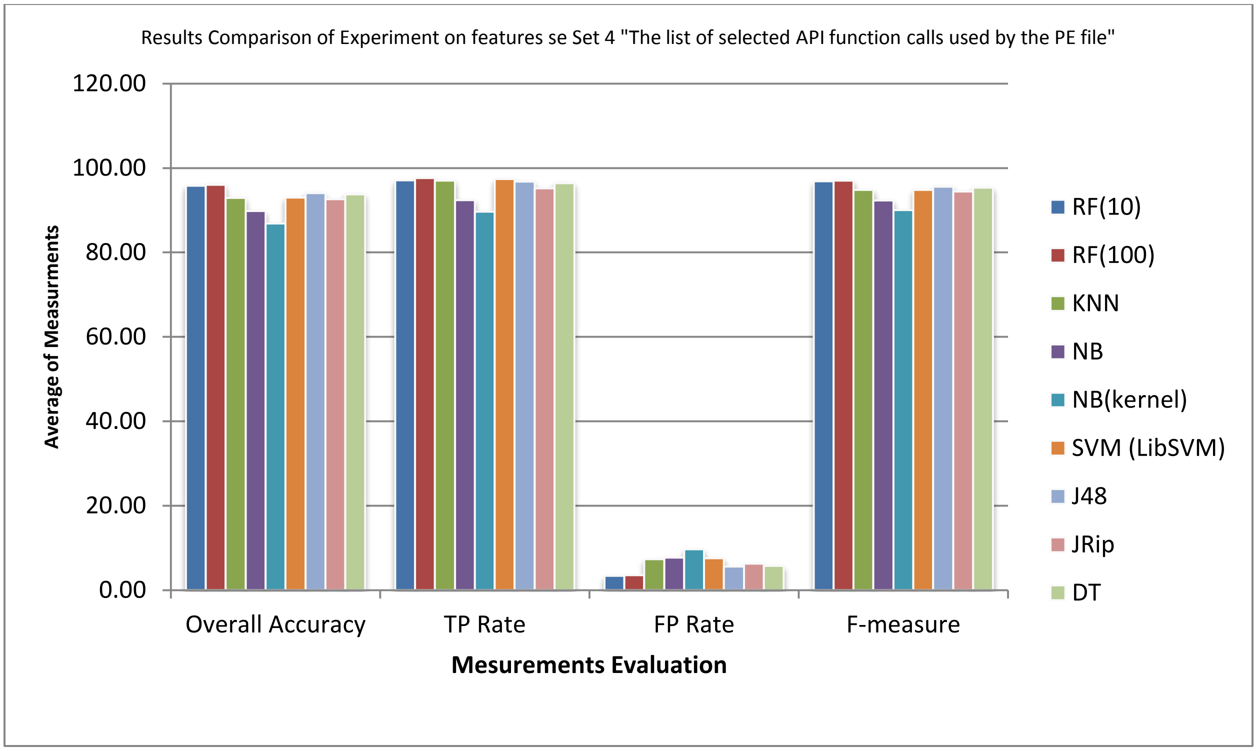


Figure 5.7 Experiments results on features Set 4.

For the Area under ROC Curve, Random Forest with 100 trees performed is the highest 0.986 while the Naive Bayes have the lowest results 0.85.

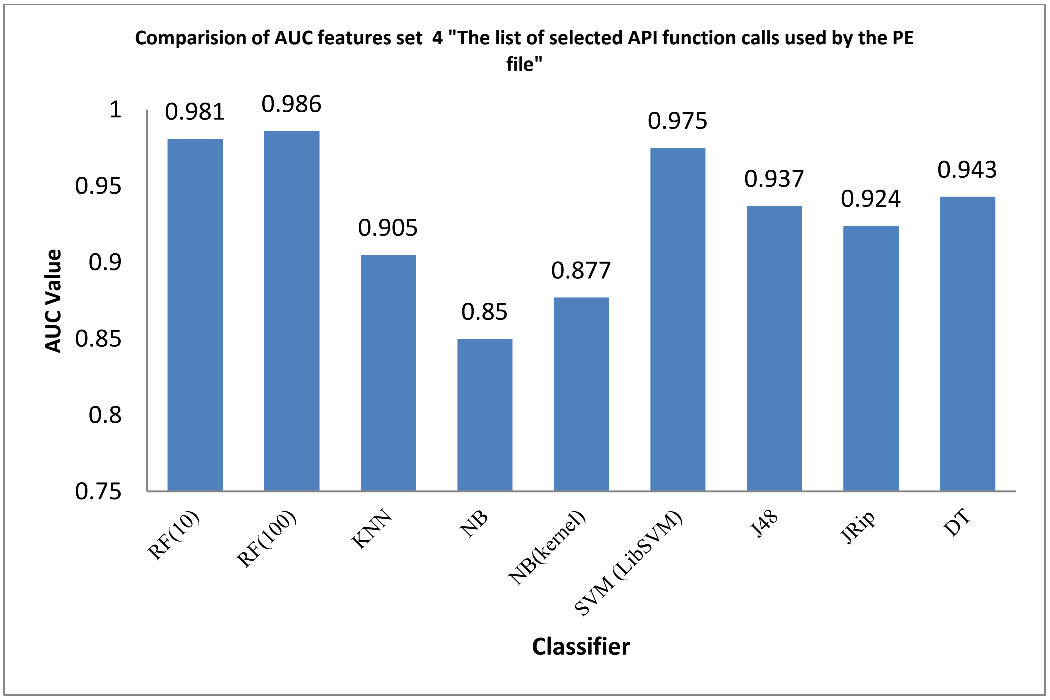


Figure 5.8 Comparison of AUC on features set 4.

5.4.5 Experiment on features set 5 “A combination of Selected API calls categories and a list of selected API function calls”

To measure the efficiency of features set 5, results are shown in Table 5.6 and Figure 5.9 and Figure 5.10. Feature set produced as results of combination of selected API calls categories and list of selected API function calls. This features set had 240 attributes. The result showed that Random Forest with 100 trees has yielded the highest overall accuracy 98.09% which is the best result in overall accuracy among all. It’s slightly better than Random Forest with 10 trees 97.59%. Naive Bayes have the lowest accuracy level 94.82%. The difference between higher and lowest accuracy is 3.27.

The highest TPR is given by Random Forest with 100 trees 98.85%. High TPR with Low FPR is given by Random Forest with 100 trees 98.85% and 1.72%. Lowest TPR is given by Naive Bayes 96.26% while the highest FPR is given by Support Vector Machine (LibSVM) 5.44%.

Table 5.6 Experiments results on features set 5.

Algorithm	TPR (Recall)	FPR	Precision	Overall Accuracy	F-measure	AUC
Random forest 10	98.22	1.87	98.13	97.59	98.18	0.993
Random forest 100	98.85	1.72	98.28	98.09	98.56	0.995
KNN	97.65	4.95	95.05	95.08	96.33	0.968
Naive Bayes	96.26	4.07	95.93	94.82	96.10	0.975
Naive Bayes(kernel)	97.74	4.68	95.32	95.32	96.52	0.982
Support Vector Machine (LibSVM)	98.51	5.44	94.56	95.27	96.50	0.992
W-J48	97.41	3.70	96.30	95.81	96.85	0.972
JRip	96.83	3.49	96.51	95.59	96.67	0.957
Decision Tree	97.65	3.60	96.40	96.03	97.02	0.962

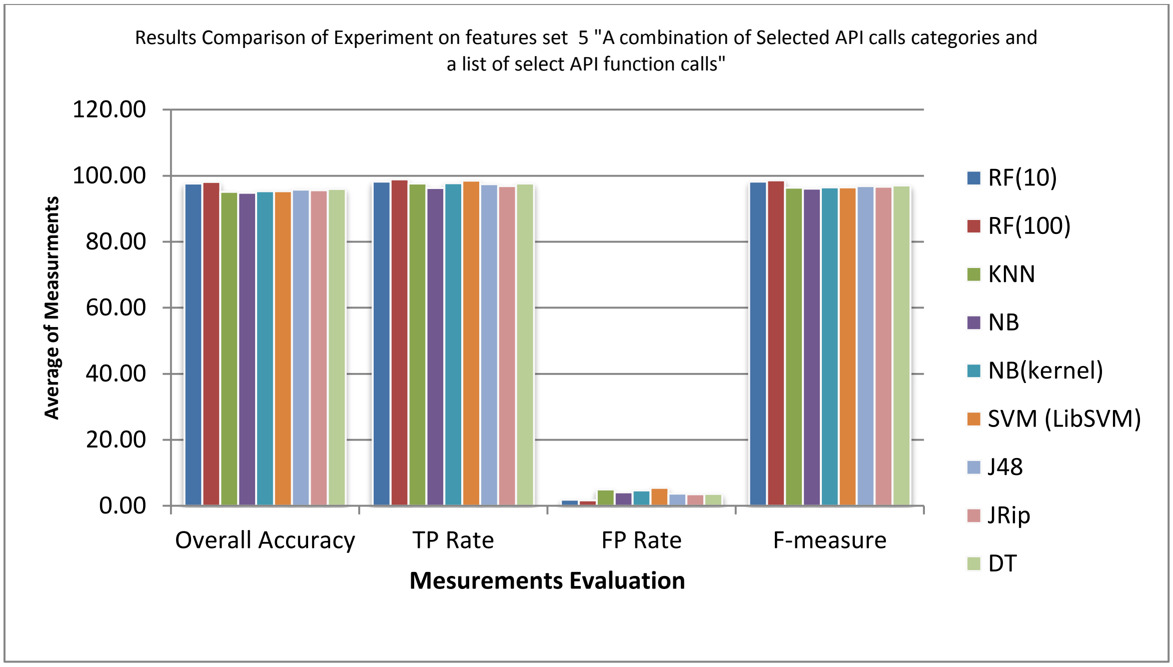


Figure 5.9 Experiments results on features set 5.

For the Area under ROC Curve, Random Forest with 100 trees 0.995 is the best, it slightly better than Random Forest with 10 trees 0.993 and Support Vector Machine (LibSVM) 0.992 results are very close to each other. It is clear from table and graphs that results of Random Forest with 100 trees had the highest overall accuracy 98.09% and AUC 0.995, True Positive Rate 98.85% with the lowest false positive rate at 1.72% are better than other classifiers.

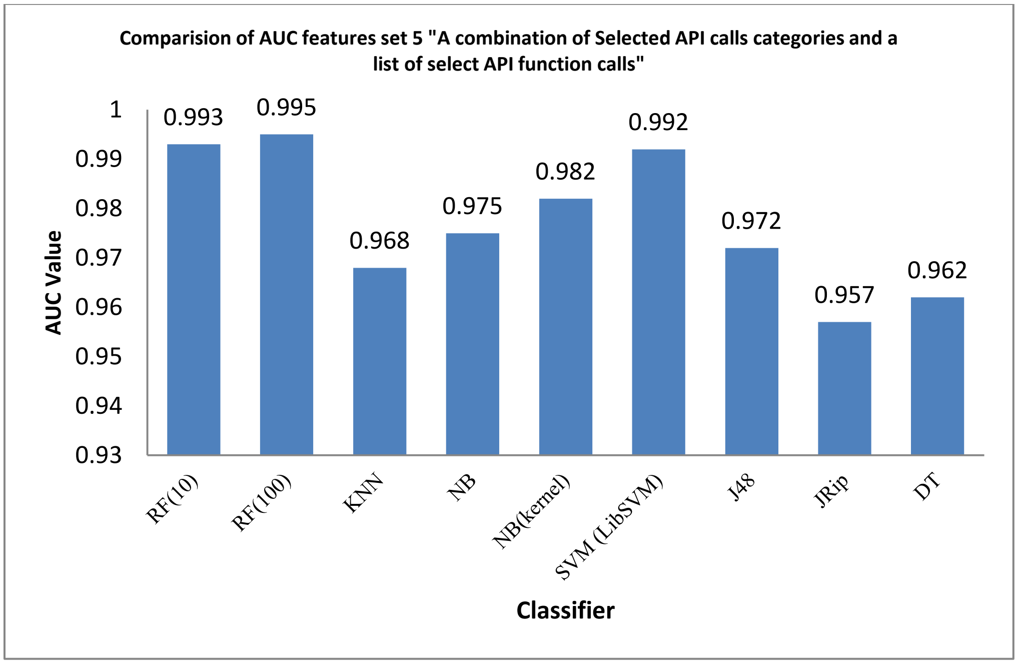


Figure 5.10 Comparison of AUC on features set 5.

5.5 Discussion and summary

The following Table 5.7, Figure 5.11 and Figure 5.12 show the summary of all experiments results.

Table 5.7 Experiments results of RF (100) on ALL features sets.

Features set	TPR (Recall)	FPR	Precision	Overall Accuracy	F-measure	AUC
Features Set 1	95.25	13.17	86.83	87.30	90.85	0.938
Features Set 2	96.02	6.54	93.46	92.92	90.85	0.977
Features Set 3	98.70	5.29	94.71	95.49	96.66	0.985
Features Set 4	97.60	3.51	96.49	96.06	97.04	0.986
Features Set 5	98.85	1.72	98.28	98.09	98.56	0.995

In the first experiment, the features set had 268 attributes, this attributes based on the list of DLLs used by the PE file; The Random Forest with 100 trees has provides the highest overall accuracy 87.30% with, AUC 0.938. True Positive Rate 95.25% with the lowest false positive rate at 13.17% and F-measure 90.85 are better than other classifiers.

In the second experiment, feature vectors produced as results of the number of different API calls the PE file has imported from the corresponding DLL. In this experiment, we found that the Random forest algorithm with 10 Trees had the highest overall accuracy 92.98% and highest F-measure 94.73 with low false positive rate at 5.87%. While the Random forest algorithm with 100 Trees had the highest AUC 0.977, it slightly better than Random Forest with 10 trees 0.968;

In the third experiment, Feature vectors produced as results of the number of different API function calls the PE file has used from API call categories. The experimental showed that Random forest algorithm with 100 trees has highest overall accuracy 95.49%, and highest TPR 98.70% with low FPR at 5.29% and AUC 0.985.

In the fourth experiment, the results showed that Random forest algorithm with 100 trees has highest overall accuracy 96.06% among all and 0.986 AUC.

In the fifth experiment, features set produced as results of combination of selected API calls categories and list of select API function calls. In this experiment, we found that Random Forest with 100 trees has yielded the highest overall accuracy 98.09% with highest TPR 98.85% and lowest FPR 1.72% while keeping AUC as high as 0.995.

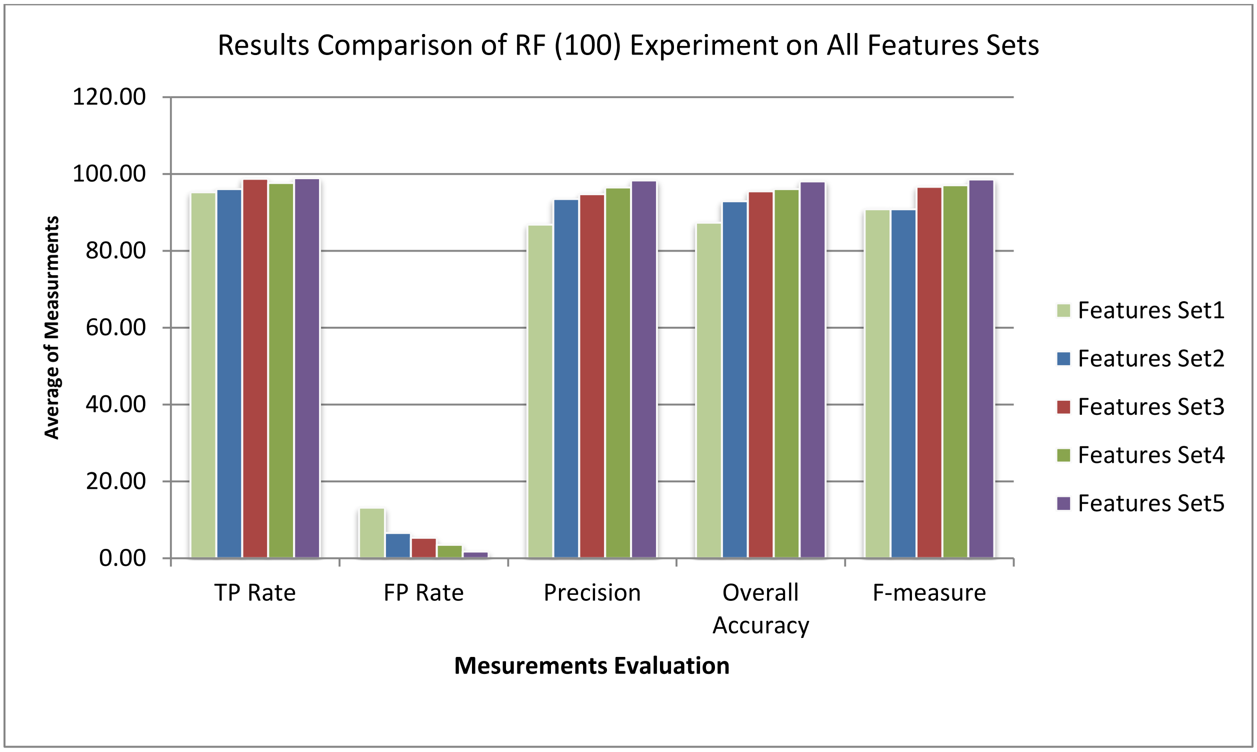


Figure 5.11 Results Comparison of RF (100) Experiment on All Features Sets.

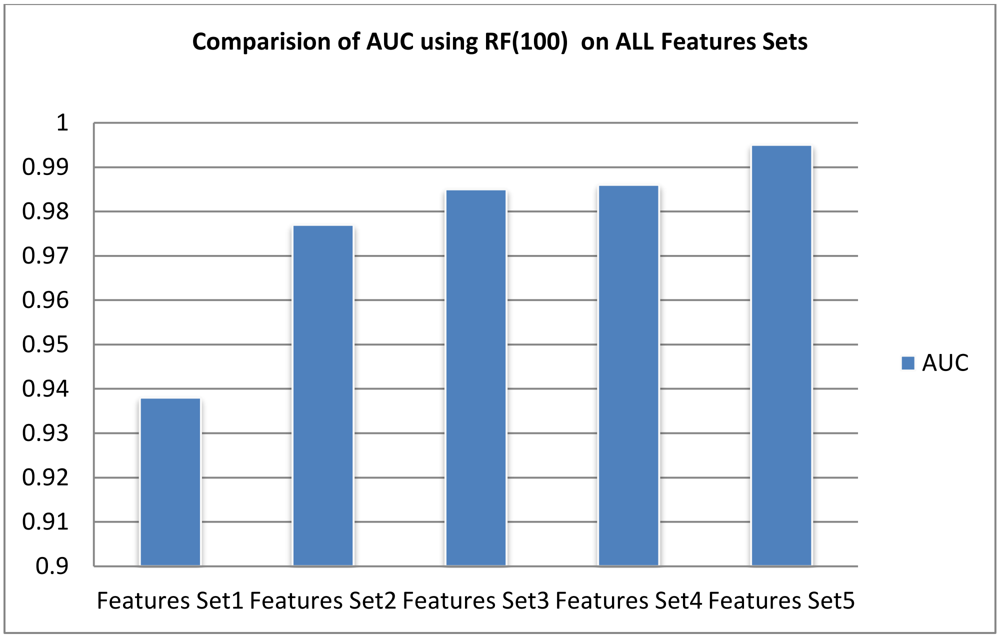


Figure 5.12 Comparison of AUC using RF (100) on ALL Features Set.

The experimental results show that the Random Forest has the best performance in classifying all features sets in respect to Overall Accuracy and F-measure. In all the experiments using Random Forest, the value of ROC area is above 0.93 which shows the better quality of classification.

To choose a methodology among these feature extraction, selection and classification methods as shown in Table 5.8, Figure 5.11 and Figure 5.12, the features set 5 which produced as results of combination of Selected API calls categories and a list of selected API function calls with Random Forest with 100 trees, achieved the highest overall accuracy 98.09% and AUC 0.995 and lowest false positive rate at 1.72% with F-measure at 98.56.

As a summary of this chapter, five different groups of experiments have been done. In each group, nine classification algorithms applied in order to determine the best set of features and the most appropriate classification algorithm. The accuracy and the area under ROC curve are used for the evaluation of classifier performance. The experiments show that Random Forest has the best performance in classifying all features sets. We achieved an overall accuracy of 98.09% with an area under the ROC curve of 0.995.

CHAPTER 6: Results Comparison and Summary

This chapter provides a comparison of the proposed method with recent work. Furthermore, summarizing the study and suggests possible future work.

6.1 Results Comparison

We have compared our result with other promising spyware detection schemes proposed by Shazhad et al. [45] and Wang et al [63]. The first comparisons with Shazhad et al. [45] use some variation of n-gram analysis for spyware detection. While Figure 6.1 compares our result versus Shazhad et al [45], Table 6.1 shows improvements we have achieved. We have achieved an overall accuracy of 98.09% while Shazhad et al. [45] achieved an overall accuracy of 90.5 % with an n-gram size of 6 from the J48 classifier. Also we used area under ROC curves as a performance metric proposed method archived an AUC of 0.995 compared to 0.833. Proposed method improved in several aspects: accuracy by 7.59% and AUC by 0.162.

Table 6.1 proposed method's Improvements to Shazhad et al.

Measure	Shazhad et al.	Proposed method	Improvement
Accuracy	90.5%	98.09%	7.59%
AUC	0.833	0.995	0.162

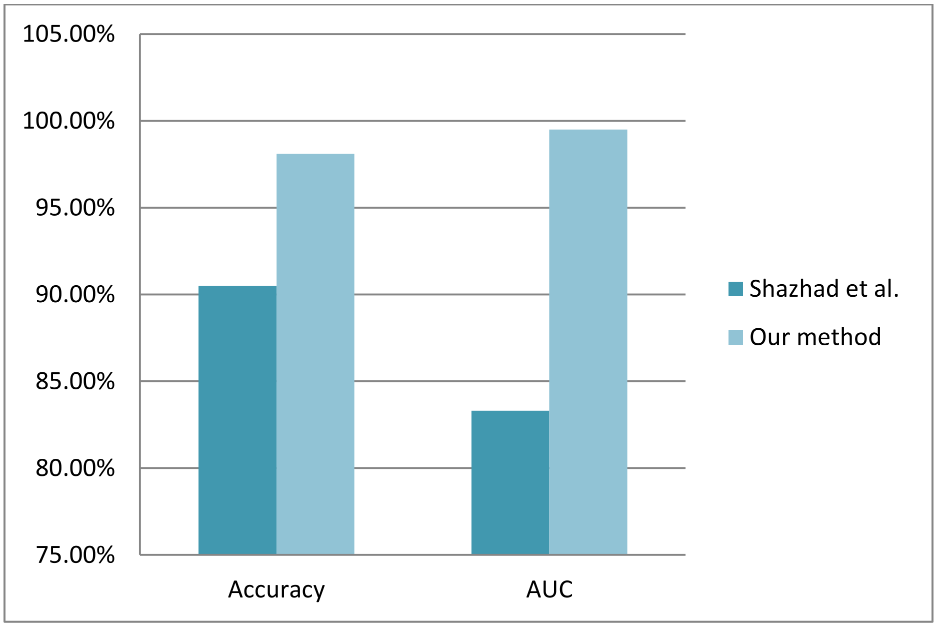


Figure 6.1 Proposed method's results versus Shazhad et al.

In the second comparison, A Surveillance Spyware Detection System Based on Data Mining Methods [63], uses both static and dynamic analysis. Table 6.2 shows the performance comparisons, they achieve an overall accuracy of 81.69 % when static analysis is used only, and 95.20% when using dynamic analysis while they achieve an overall accuracy of 97.9% when both static and dynamic analysis used together, as training and testing using the same datasets denoted as OA. When 10-fold cross validation is considered, the overall accuracy is 96.43% denoted as CV. We have achieved an overall accuracy of 98.09% by using static analysis only. Proposed method improved in accuracy by 13.09% over their static analysis, 2.89% over dynamic analysis and slightly better than both static and dynamic analysis by 0.19 %. Also the True Positive Rate improved by 3.52%, the false positive rate of proposed method is less than static only and dynamic only analysis by 19.09%, 0.17% respectively.

Table 6.2 Proposed method's results versus A Surveillance Spyware Detection System Based on Data Mining Methods.

Measure	A Surveillance Spyware Detection System Based on Data Mining Methods [63]						Proposed method
	Static		Dynamic		Both (Static and Dynamic)		
	OA	CV	OA	CV	OA	CV	
Accuracy	81.69%	80.99%	95.20%	93.12%	97.91%	96.43%	98.09%
TPR	86.49%		89.99%		95.33%		98.85%
FPR	20.81%		1.89%		0.68%		1.72%

6.2 Summary

Spyware is the greatest threat of user privacy violations. Signature-based anti-spyware tools can easily be evaded through obfuscation transformations.

In this work, the problem of detecting new and unknown spyware is addressed. The main contributions provided by this research, comes from building of spyware dataset to perform our experiments in the lack of a publicly available spyware dataset. And the static extraction of features and applying different data mining algorithm as complete process.

In this research, API Calls are used as features during the classification process with the aim of identifying spyware files. We performed an extensive evaluation using data set comprising 3149 PE files; Where 2084 files were spyware, and 1065 files were benign.

After extracting Windows API calls information from PE file, five features sets are constructed, which constructed. For features selection we compare the performance of three different feature ranking algorithms: Information Gain; Information Gain Ratio and Support Vector Machine, from these algorithms the weight by SVM Ranking algorithm adopted to use as feature selection on all features sets in order to select the most relevant features.

The evaluation consisted of five different groups of experiments. In each group, nine classification algorithms were applied in order to determine the best set of features and the most appropriate classification algorithm. For evaluation purposes in addition to Total Accuracy, which measures the number of absolutely correctly classified instances, either positive or negative, divided by the entire number of instances. The AUC is used to determine the detection accuracy of each approach; where ROC curves are insensitive to changes in class distribution.

After evaluating a variety of classification methods, results suggest that, for the task of detecting spyware executable, Random Forest classifier achieved the best detector with an , overall accuracy 98.09% and area under the ROC curve of 0.995. From analyzing the experimental results, we can conclude that, finding static features of each spyware and applying data mining techniques to the data helps in detecting the new spyware.

6.3 Future Work

In future work, with the thousands of new spyware generated each day manually detection is time consuming; we hope to automate the whole spyware detection process. And we are planning to evaluate our method on a larger dataset of spyware and benign executables.

Obfuscation and packing can affect the performance of the detection; we hope to do more about de-obfuscation and unpacking the binary files. We consider these tasks as the future works that aid in detecting new spyware more efficiently.

References

[1] D. W. Aha, D. Kibler, and M. K. Albert, "Instance-based learning algorithm," *Journal of Machine Learning*, vol. 6, pp. 37-66, 1991.

[2] Mamoun Alazab, S. Venkatraman, P. Watters, and Moutaz Alazab, "Zero-day malware detection based on supervised learning algorithms of API call signatures," in *Australasian Data Mining Conference (9th : 2011 : Ballarat, Vic.)*, Ballarat, Australia , 2011 , pp. 171 -182.

[3] A. Altaher, S. Ramadass, and A. Ali, "Computer Virus Detection Using Features Ranking and Machine Learning," *Australian Journal of Basic and Applied Sciences*, vol. 5, no. 9, pp. 1482-1486, 2011.

[4] A. Altaher, Supriyanto, A. ALmomani, M. Anbar, and S. Ramadass, "Malware detection based on evolving clustering method for classification," *Academic Journals*, vol. 7, no. 22, pp. 2031-2036, June 2012.

[5] P. Bahraminikoo, M. Yeganeh, and G. Babu, "Utilization Data Mining to Detect Spyware," *IOSR Journal of Computer Engineering (IOSRJCE)*, vol. 4, no. 3, pp. 01-04, Oct. 2012.

[6] (2012, November) bitdefender. [Online]. http://www.bitdefender.com/news/windows-8-prone-to-infection-by-leading-malware-threats-controlled-test-shows-2646.html; [Accessed 20.012.2012].

[7] Bitsum. Pebundle. [Online]. http://www.bitsum.com/pebundle.asp; [Accessed 12.04.2012].

[8] M. Boldt and B. Carlsson, "Privacy invasive software and preventive mechanisms," in *2nd International Conference on Systems and Networks Communications*, 2006.

[9] C. D. Bozagac, "Application of Data Mining based Malicious Code Detection Techniques for Detecting new Spyware," Bilkent University, White paper 2005.

[10] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[11] CA. (2008) Internet Security Outlook. [Online]. http://ca.com/files/SecurityAdvisorNews/ca_security_2008_white_paper_final.pdf; [Accessed 20/12/2011].

[12] CA. (2010) State of the Internet 2010: A Report on the Ever-Changing Threat Landscape. [Online]. http://www.ca.com/~/media/Files/SecurityAdvisorNews/h12010threatreport_244199.pdf; [Accessed 20/12/2011].

[13] R. Chang, Z. Pei, and C. Zhang, "A Modified Editing k-nearest Neighbor Rule," *JOURNAL OF COMPUTERS*, vol. 6, no. 7, 2011.

[14] S. Dua and X. Du, *Data Mining and Machine Learning in Cybersecurity*.: The Auerbach, 2011.

[15] T. Fawcett, "An introduction to ROC analysis," *Elsevier*, 2005.

[16] T. Fawcett, "ROC Graphs: Notes and Practical Considerations for Researchers. ," HP Laboratories, Palo Alto, USA, Technical report 2004.

[17] J. Han and M. Kamber, *Data Mining: Concepts and Techniques*.: The Morgan Kaufmann, 2006.

[18] Hex-Rays. [Online]. http://www.hex-rays.com/idapro/; [Accessed 1.03.2012].

[19] A. Honig and M. Sikorski, *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*.: No Starch Press San Francisco, CA , 2012.

[20] N. Idika and A. Mathur, "A Survey of Malware Detection Techniques.," Software Engineering Research Center, Technical report 2007.

[21] L. Jiang, H. Zhang, and J. Su, "Learning k-Nearest Neighbor Naive Bayes For Ranking," *Proceedings of First International Conference on Advanced Data Mining and Applications (ADMA2005)*, pp. 175-185, 2005.

[22] JlabSoftware. Polycrypt pe. [Online]. http://download.cnet.com/PolyCrypt-PE/; [Accessed 12.04.2012].

[23] M. Kang, P. Poosankam, and H. Yin, "Renovo: A hidden code extractor for packed executables," *In Proc. Fifth ACM Workshop on Recurring Malcode (WORM 2007)*, 2007.

[24] M. Kantardzic. (2003) Data Mining: Concepts, Models, Methods, and Algorithms. ebooked.

[25] J. Kolter and M. Maloof, "Learning to Detect Malicious Executables in the Wild," in *In Proceedings of the 2004 ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2004.

[26] S. KULKARNI and G. HARMAN, *An Elementary Introduction to statistical learning*.: WILEY, 2011.

[27] D. LAROSE, *Discovering Knowledge in Data an Introduction to Data Mining*.: John Wiley & Sons, 2005.

[28] M. Ligh, S. Adair, M. Richard, and B. Hartstein, *Malware Analyst's Cookbook and DVD: Tools and Techniques for Fighting Malicious Code*.: John Wiley & Sons, 2010.

[29] C. Linn and S. Debray, "Obfuscation of Executable Code to Improve Resistance to Static Disassembly," in *in 10th ACM conference on Computer and communications security*, vol. 9, Washington, DC, USA, 2003, p. 290.

[30] White Mark. (Fall 2005) ECE591Q - Machine Learning – Lecture slides.

[31] Market Share Statistics for Internet Technologies. [Online]. http://www.netmarketshare.com/; [Accessed 5.02.2013].

[32] M. E. Maron and J. L. Kuhns, "On relevance, probabilistic indexing and information retrieval," *Journal of the Association of Computing Machinery*, vol. 7, no. 3, pp. 216-244, 1960.

[33] L. Martignoni, M. Christodorescu, and S. Jha, "Omniunpack: Fast, generic, and safe unpacking of malware," in *in Proceedings of the AnnualComputer Security Applications Conference (ACSAC)*, 2007, pp. 431-441.

[34] R. Moskovitch et al., "Unknown malcode detection using OPCODE representation," in *1st European Conference on Intelligence and Security Informatics, (EuroISI 2008)*, Berlin, Germany, 2008, pp. 204-215.

[35] MSDN. Library. Functions by category. [Online]. http://msdn.microsoft.com/en-us/library/aa383686/; [Accessed 5.10.2012].

[36] D. Olson and D. Delen, *Advanced data mining techniques*.: Springer-Verlag Berlin Heidelberg, 2008.

[37] M. Pietrek. An in-depth look into the Win32 Portable Executable file. [Online]. http://msdn.microsoft.com/en-s/magazine/cc301805.aspx; [Accessed 25/12/2011].

[38] M. Pietrek. An in-depth look into the Win32 Portable Executable file part 2. [Online]. http://msdn.microsoft.com/en-s/magazine/cc301808.aspx; [Accessed 1.03.2012].

[39] Quarterly Report PandaLabs. [Online]. ttp://www.pandasecurity.com/img/enc/Quarterly_Report_PandaLabs_Q1_2009.pdf;[Accessed 10/12/2011].

[40] Rapid Miner 5.2.008. [Online]. http://www.rapidminer.com; [Accessed 5.10.2012].

[41] A. Sami, H. Rahimi, B. Yadegari, and S. Hashemi, "Malware Detection Based on Mining API Calls," *ACM Symposium on Applied Computing*, April 2010.

[42] M. Schultz, E. Eskin, E. Zadok, and S. Stolfo, "Data Mining Methods for Detection of New Malicious Executables.," *In Proceedings of the IEEE Symposium on Security and Privacy*, pp. 38–49, 2001.

[43] T. Scott and Z. Mian, "A heuristic approach for detection of obfuscated malware," 2009.

[44] M. Sharif, V. Yegneswaran, H. Saidi, P. Porras, and W. Lee, "Eureka: A framework for enabling static malware analysis," *Computer Security - ESORICS, Lecture Notes in Computer Science (LNCS)*, pp. 481-

500, 2008.

[45] R. Shazhad, S. Haider, and N. Lavesson, "Detection of Spyware by Mining Executable Files," in *International Conference on Availability, Reliability and Security*, 2010.

[46] M. Siddiqui, M. C. Wang, and J. Lee, "Detecting Internet worms Using Data Mining Techniques," *Journal of Systemics, Cybernetics and Informatics*, vol. 6, no. 6, pp. 48-53, 2009.

[47] Softidentity. Aspack. [Online]. www.aspack.com; [Accessed 14.04.2012].

[48] softpedia. [Online]. http://www.softpedia.com/get/Programming/Packers-Crypters-Protectors/PEiD-updated.shtml; [Accessed 12.04.2012].

[49] Softpedia. [Online]. http://www.softpedia.com/get/Programming/Packers-Crypters-Protectors/ExEinfo-PE.shtml; [Accessed 12.04.2012].

[50] Software, T. Molebox. [Online]. http://www.molebox.com/; [Accessed 12.04.2012].

[51] A. Stepan, "Improving proactive detection of packed malware," *Virus Bulletin*, pp. 11–13, Mar. 2006.

[52] P. Szor, *The Art of Computer Virus Research and Defense.* New Jersey: Addison Wesley for Symantec Press, 2005.

[53] Technologies, O. Themida. [Online]. http://www.oreans.com/themida.php; [Accessed 12.04.2012].

[54] Toolworks, T. S. R. Armadillo. [Online]. http://www.siliconrealms.com/; [Accessed 12.04.2012].

[55] Ultimate packer for executables. [Online]. http://upx.sourceforge.net/; [Accessed 12.04.2012].

[56] Unknown. Yoda's crypter. [Online]. http://yodas-crypter.softpedia.com; [Accessed 12.04.2012].

[57] V. N. Vapnik, *Statistical Learning Theory*. New York: John Wiley and Sons, 1998.

[58] A. Vasudevan and R. Yerraballi, "Spike: Engineering malware analysis tools using unobtrusive binary-instrumentation," in *In Proceedings of the 29th Australasian Computer Science Conference*, 2006, pp. 311–320.

[59] R. Veeramani and Rai Nitin, "Windows API based Malware Detection and Framework Analysis," *International Journal of Scientific & Engineering Research*, vol. 3, no. 3, March 2012.

[60] VMware. [Online]. http://www.vmware.com/; [Accessed 14.06.2012].

[61] VX Heavens Virus Collection, VX Heavens website. [Online]. http://vx.netlux.org. ; [Accessed 09.01.2012].

[62] vx-archiv. [Online]. http://vx-archiv.at; [Accessed 12.03.2012].

[63] T. Wang et al., "A Surveillance Spyware Detection System Based on Data Mining Methods," 2006.

[64] T. Wang, C. Wu, and C. Hsieh, "Detecting Unknown Malicious Executables Using Portable Executable Headers," in *Fifth International Joint Conference on INC, IMS and IDC*, 2009, pp. 278-284.

[65] L. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, 3rd ed.: the Morgan Kaufmann Series in Data Management Systems, 2011.

[66] C. Yang-Sec , K. Ik-kyun, O. Jin-Tae, and R. Jae-Cheol, "PE File Header Analysis-Based Packed PE File Detection Technique (PHAD)," in *Computer Science and its Applications, 2008. CSA '08. International Symposium on*, 2008, pp. 28 - 31.

[67] W. Yan, Z. Zhang, and N. Ansari, "Revealing packed malware," *Security Privacy, IEEE*, vol. 6, no. 5, pp. 65 –69, sept.- oct. 2008.

[68] N. Ye, *The Handbook of Data Mining*.: Lawrence Erlbaum Associates, Inc, 2003.