Summer 8-13-2014

Feature selection and clustering for malicious and benign software characterization

Dalbir Kaur R. Chhabra
University of New Orleans, dchhabra@uno.edu

Feature selection and clustering for malicious and benign software characterization

A Thesis

Submitted to the Graduate Faculty of the
University of New Orleans
in partial fulfillment of the
requirements for the degree of

Master of Science
in
Computer Science
Information Assurance

by

Dalbir Kaur Chhabra

B.E. Gujarat University, 2010

August 2014

ACKNOWLEDGEMENT

I am very thankful to Dr. Mahdi Abdelguerfi and Dr. Shengru Tu for offering me wonderful opportunity of pursuing Master's degree at University of New Orleans. I would like to express my deepest appreciation to my advisor, Dr. Irfan Ahmed, who provided me with a learning opportunity in his laboratory and helped me to grow as a student. I would also like to express my sincere gratitude to my thesis committee members, Dr. Golden Richard III and Dr. Adlai Depano for providing me guidance and criticism that helped me in completion of my thesis successfully.

I would like to extend my sincere appreciation to my family for their guidance, encouragement, and support. I would like to thank my colleagues and friends for their contributions during my course of study. John Finigan, helped me to setup system and provided me with huge dataset of malware and Aisha Ibrahim Ali-Gombe guided me through unsupervised learning algorithm and Matlab. I take this opportunity to thank the faculty and administrative members of the department of Computer Science for their availability, whenever needed, without making any appointments.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

AV	Antivirus
PE	Portable Executable
SOM	Self Organizing Map
COFF	Common Object File Format
.EXE	Executable Files
DLL	Dynamic Link Library
.OBJ	Object Code
KDD	Knowledge Discovery in Database
SVM	Support Vector Machines
ANN	Artificial Neural Network
VS	Visual Studio
IDE	Integrated Development Environment
VC++	Visual C++
VM	Virtual Machines
NFD	Number of Files in dataset
HFC	Hierarchical Falsely Clustered
KFC	K-mean Falsely Clustered
SFC	SOM Falsely Clustered

ABSTRACT

Malware or malicious code is design to gather sensitive information without knowledge or permission of the users or damage files in the computer system. As the use of computer systems and Internet is increasing, the threat of malware is also growing. Moreover, the increase in data is raising difficulties to identify if the executables are malicious or benign. Hence, we have devised a method that collects features from portable executable file format using static malware analysis technique. We have also optimized the important or useful features by either normalizing or giving weightage to the feature. Furthermore, we have compared accuracy of various unsupervised learning algorithms for clustering huge dataset of samples. So once the clusters are created we can use antivirus (AV) to identify one or two file and if they are detected by AV then all the files in cluster are malicious even if the files contain novel or unknown malware; otherwise all are benign.

Keywords:

Static malware analysis, Portable Executable, unsupervised learning algorithm, malicious or benign samples, feature selection, clustering, software characterization

CHAPTER 1: PROJECT OVERVIEW

1.1 Introduction

As the use of computer systems and Internet is increasing, the need for network security is also growing. The lack of sophisticated protection on network has attracted skilled and motivated cyber criminals to introduce wide range of security attacks. This security attacks are the 'malicious' codes or software that are designed for performing illegal or unethical task, which are commonly known as malware.

Malware or malicious codes are designed to gather sensitive information without knowledge or permission of the users, gain unauthorized access to the system resources, or damage files in the computer system. Malware does not damage the computer system or any network equipment physically but it can harm the data or available resources by using all the available RAM, CPU usage, network bandwidth or storage spaces. Malware should not be confused with the defective software that is intended for legitimate work but has some errors. A majority of malware attacks computer system or network through the Internet.

Earlier the anti-malware vendors and researchers used to detect malware on its signature but the enormous increase in the malware has made their detection very difficult. Thus, it has now become important to develop new technique(s) using malware analysis.

1.2 Statement of problem

Identifying malware from executables has become a huge problem. As the data are increasing, number of executables is also growing at a high pace. Thus, by clustering the executables and

then by just analyzing one executable, will help in generalizing if the executables in cluster are benign or malicious. This method will increase the detection rate of malware.

1.3 Project Objective

The enormous increase in the malware has made it difficult for the researcher and anti-malware vendors to detect the malicious executables. Moreover, there is an increase in the data and it is difficult to distinguish between malicious and benign executables without running them. Our objective was to understand the PE file format and extract the important and useful features that will help in clustering them in benign and malicious groups.

The goal of our research was to explore a number of standard data mining techniques or algorithms in order to cluster the executables using static anomaly detection with highest accuracy.

1.4 Project Scope

The proposed method will help to identify the malicious executables from the collected executables with highest accuracy. When there are large numbers of executables and identifying malware is complex, then proposed method can be used to create clusters of benign and malicious executables.

Analyzing the executables without running them saves lot of time and work. The static analysis approach is less dirty and painful compared to dynamic analysis. The data mining techniques

makes the method more effective, efficient and quick. Various clustering algorithms are compared and one with the highest accuracy is used.

1.5 Limitation

This research aimed to enhance the malware detection by using data mining classification techniques. The proposed method has following limitations:

- The method is limited to the analysis of executables that can run on the various version of Microsoft Windows Operating System; Win-32 and Win-64 bit Portable Executable (PE) files. All datasets (benign and malware) used were in Win-32 or Win-64 PE format.
- The research was limited to static malware detection technique.
- The research was based on unsupervised learning algorithm to identify malware.

1.6 Thesis Structure

The thesis is organized as mentioned below.

Background:

In chapter 2, a broad background is given regarding the project. It introduces malware and its types in brief and followed by that is Malware analysis and its techniques. This explains the types of malware that we were using in our dataset and an important technique of our thesis project called the static malware analysis techniques. In section 2.4 the Portable Executable file format is introduced from which we will be collecting the features, which will be important for clustering.

Related Work:

In chapter 3, we have presented the related work for our thesis project. This section helps in understanding what and how much is done related to techniques involved in this project. In section 3.1, we have discussed few papers on malware detection and in section 3.2 we are discussing work done using data mining techniques. Lastly in section 3.3 we discuss our project and the work, which is already done in this field. We then summarized how our work is different from other work and how it will be useful.

Proposed Methodology:

Chapter 4 proposes the method of our thesis work. It then explains each step of our method like data collection, data preprocessing, feature extraction, and feature selection. At last in section 4.5, all the clustering algorithm such as Hierarchical, K-mean and Self Organizing Map (SOM), which we implemented in our research work is explained.

Experiment and outcome of project:

Experimental tools and environment and the experiment results with regards to unsupervised leaning algorithm are discussed in chapter 5. All 9 important experiments, their results and detailed analysis are covered in this chapter. Then in section 5.3, discussion on the experiment and results is done.

Conclusion and Future study:

Chapter 6 gives the conclusion on thesis project. Furthermore, the future studies are also discussed in this chapter. At last, thesis study is summarized in section 6.3 of the chapter.

CHAPTER 2: BACKGROUND

2.1 Malware and its types

Malware are the malicious codes that are written to perform unethical tasks for example accessing system without authorization from administrator, collecting sensitive information, damaging data or system resources. Malware can be categorized into, among others, Virus, Worm, Spyware, Trojan, Backdoor, Rootkit. The most common types of malware are Virus, Worm and Trojan.

Types of malware [1]:

Virus:

It is a form of malware that copy itself and spread to other computers. It spread to other computer by attaching themselves to the various programs or executing code when user executes the infected program. Virus replicates itself and leaves its infections as it travels from one system to another. Generally, virus is not released unless user executes the infected program but once it is released it starts replicating itself.

Worm:

Worms are standalone programs that exploit operating system vulnerabilities to spread over computer networks. Worms do not travel by linking itself to an existing program like Virus does. Worms can also contain payloads, a piece of code designed to perform illegal or unethical task that damage host computer or network.

Spyware:

Spyware, as the name suggest is the malware type that spies on the user activity. The spying includes gathering information, among others like the websites visited, browser history, system or account information, financial data, and banking details. These gathered information is then transmitted to malware owners. Spyware does not infect host system but once it enters the system; it installs itself and collects information in background so that it remains undetected.

Trojan:

A Trojan horse or commonly known as Trojan is a type of malware, which is non-self replicating. It misrepresent itself as a normal executable and trick user to download and install the malware. A Trojan can give remote access of infected system to hackers or attackers, which allow them to steal data, install more malware or modify files.

Backdoor:

Backdoor malware allows the unauthorized access of the computer to the hackers while remaining undetected. The threat of this kind of malware was initiated when multi-user and network operating system were started using widely. Backdoors can be created by rewriting compiler and not changing the code before or after the compilation. The complier can be written in such a way that a piece of code, compiles the code normally but also triggers backdoor.

Rootkit:

Rootkit malware type is same as Trojan or backdoor in a way that it tries to gain access of the computer system without permission or knowledge of the user and tries to remain undetected.

The rootkit is different from Trojan as it is installed by the hacker after gaining the access or automatically. Unlike other malware rootkit gains full access of the system, it can modify or install any software. It is a standalone program that tries to hide processes, registry data, network connections or files. It is nearly impossible or difficult to detect and remove a rootkit malware from the infected computer.

2.2 Introduction to Malware Analysis

The intention of creating malicious codes has been changing from more widespread to more sophisticated by targeting sensitive information such as passwords, credit card information and bank details that has made malware analysis very crucial.

Since the malware causes significant loss of critical data and sometimes damage the network, the malware detection has become one of the most critical issues in the field of computer security. Thus, in order to detect malware efficiently, malware analysis plays an important role.

2.3 Malware Analysis Techniques

Malware analysis can be broadly classified into static and dynamic technique [2].

Dynamic Malware Analysis Technique:

Dynamic malware analysis technique is also commonly known as behavioral analysis as it examines the behavior of malware by executing the binary code in the controlled environment. Behavioral analysis is quicker but tricky way of malware analysis as while doing analysis if

malware is not provided the suitable environment then there are more chances that analyst will miss the characteristics of malware.

Behavior or dynamic analysis can be further divided into two categories as basic and advanced. Basic malware analysis provides suitable environment to malware, monitor their execution and gather all the information related to their runtime behavior. These information can be related to API or system calls, files added, removed and/or modified, new services installed, changes in processes or registry files or any modifications in system settings.

Advanced behavior analysis requires knowledge of windows internals and specific programming. Analysts can load binary code into debugger tools such as ollydbg or windbg and run malware code line by line and monitor its activity.

Static Malware Analysis Technique:

Static malware analysis technique is performed by analyzing code statically without running the sample using tools such as PE Viewer, CFF explorer and more. Thus it is also known as code analysis. This technique is safer than dynamic malware analysis technique as malware are not executed. If malware is packed then analysis cannot be performed on it without unpacking the malware.

Code analysis technique can also be further classified into basic and advanced categories. Basic code analysis technique is not very efficient but it is easy and very quick. The goal of static analysis is to classify the sample into malicious and benign executables without understanding the capabilities of samples. It does not require checking the actual instructions of sample. Basic

code analysis includes identifying if antivirus detects any sample, sample is packed or unpacked, its version information, any suspicious imports by executables or if any PE field format is malformed.

Advanced code analysis, like advanced behavioral analysis, requires knowledge of windows internals, Assemble language and compiler code. In this type of analysis analysts are required to load the binary code into disassembler such as IDAPro to perform reverse engineering and completely analyze the executables. After performing reverse engineering analysts will understand how the code works or how malware infects the system, which in turn will help to reduce infection or help to create better security and defense softwares. This is the most effective technique.

2.4 Introduction to Portable Executable Format

The PE file format is derived from the earlier common object file format (COFF) found on VAX/VMS. The primary goal behind designing PE file format was to standardize the executable format of all the versions for windows operating system on all supported processors. The secondary goal was to provide the smart and easy way for the windows operating system to run program and also store essential information which is required to execute a piece of code [3].

The PE format was initially designed to support Win32-based systems or 32-bit operating system of Microsoft. Later, few modifications were done in PE format to support Win64-based systems or 64-bit operating system of Microsoft, also known as PE32+. The PE format can be used on versions of operating systems such as Windows NT, Windows 95, Windows XP, and Windows

7, 32-bit or 64-bit. Executable files (.exe), Dynamic Link Library (DLL) (.dll), Object code (.obj) are kind of PE file formats. The difference between .exe and .dll files is only of a single bit, which indicates if the file should be treated as an exe or as a dll.

PE File Structure:

PE file consists of number of headers and sections, organized as a linear stream of data as shown in Figure 2.1 and for better understanding of each field in headers we have created Table2.1. PE file begins with an MS-DOS header, a real mode program stub, and a PE file signature. Immediately following are all headers and sections [4].

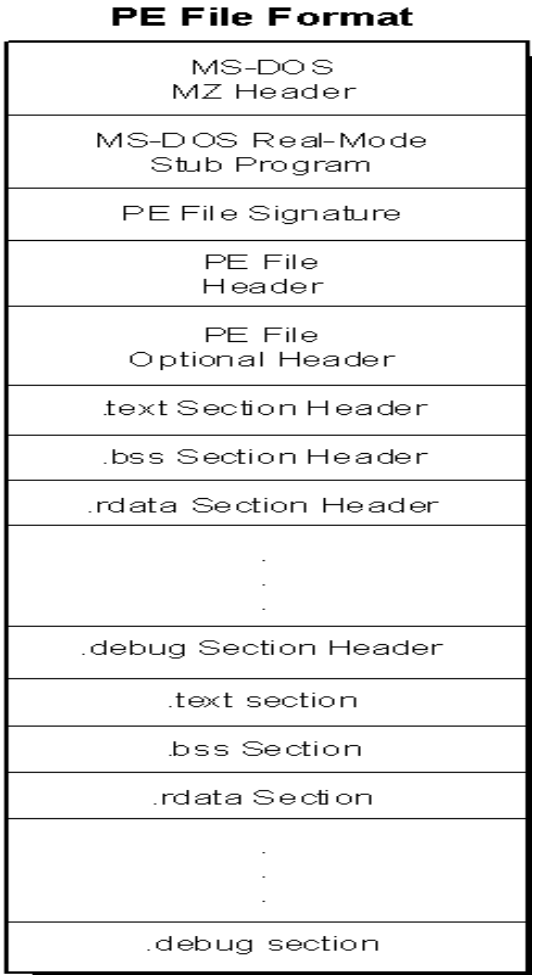

Figure 2.1 PE File Format [4]

(Table 2.1 Cont.)

Size	Field	Description
IMAGE_DOS_HEAD		
uint16_t	e_magic	Magic number
uint16_t	e_cblp	Bytes on last page of file
uint16_t	e_cp	Pages in file
uint16_t	e_crlc	Relocations
uint16_t	e_cparhdr	Size of header in paragraphs
uint16_t	e_minalloc	Minimum extra paragraphs needed
uint16_t	e_maxalloc	Maximum extra paragraphs needed
uint16_t	e_ss	Initial (relative) SS value
uint16_t	e_sp	Initial SP value
uint16_t	e_csum	Checksum
uint16_t	e_ip	Initial IP value
uint16_t	e_cs	Initial (relative) CS value
uint16_t	e_lfarlc	File address of relocation table
uint16_t	e_ovno	Overlay number
uint16_t	e_res1[4]	Reserved words
uint16_t	e_oemid	OEM identifier (for e_oeminfo)
uint16_t	e_oeminfo	OEM information; e_oemid specific
uint16_t	e_res2[10]	Reserved words
uint32_t	e_lfanew	Address of image file header
IMAGE_FILE_HEAD		
uint16_t	Machine	machine version
uint16_t	NumberOfSections	No. of Sections
uint32_t	TimeDateStamp	Date time stamp
uint32_t	PointerToSymbolTable	Pointer to symbol table
uint32_t	NumberOfSymbols	No. of Symbols
uint16_t	SizeOfOptionalHeader	Size of Optional header
uint16_t	Characteristics	characteristics
IMAGE_DATA_DIRECT		
uint32_t	VirtualAddress	Virtual address of data directory
uint32_t	Size	Size of data directory
IMAGE_OPTIONAL_HEAD		
uint16_t	Magic	

unsigned char	MajorLinkerVersion	Version of Major Linker
unsigned char	MinorLinkerVersion	Version of Minor Linker
uint32_t	SizeOfCode	Size of Code
uint32_t	SizeOfInitializedData	Size of Initialized Data
uint32_t	SizeOfUninitializedData	Size of Uninitialized Data
uint32_t	AddressOfEntryPoint	Address of Entry Point
uint32_t	BaseOfCode	Base of Code
uint32_t	BaseOfData	Base of Data
uint64_t	ImageBase	Image Base
uint32_t	SectionAlignment	Section Alignment
uint32_t	FileAlignment	File Alignment
uint16_t	MajorOperatingSystemVersion	Major Operating System Version
uint16_t	MinorOperatingSystemVersion	Minor Operating System Version
uint16_t	MajorImageVersion	Major Image Version
uint16_t	MinorImageVersion	Minor Image Version
uint16_t	MajorSubsystemVersion	Major Subsystem Version
uint16_t	MinorSubsystemVersion	Minor Subsystem Version
uint32_t	Reserved1	Reserved1
uint32_t	SizeOfImage	Size of Image
uint32_t	SizeOfHeaders	Size of Headers
uint32_t	CheckSum	Check Sum
uint16_t	Subsystem	Subsystem
uint16_t	DllCharacteristics	Dll Characteristics
uint64_t	SizeOfStackReserve	Size of Stack Reserve
uint64_t	SizeOfStackCommit	Size of Stack Commit
uint64_t	SizeOfHeapReserve	Size of Heap Reserve
uint64_t	SizeOfHeapCommit	Size of Heap Commit
uint32_t	LoaderFlags	Loader Flags
uint32_t	NumberOfRvaAndSizes	Number of Rva and Sizes
IMAGE_DATA_DIRECT	DataDirectory[16]	Array of Data Directory
IMAGE_NT_HEAD		
uint64_t	Signature	Signature
IMAGE_FILE_HEAD	FileHeader	File Header
IMAGE_OPTIONAL_HEAD	OptionalHeader	Optional Header
IMAGE_SECTION_HEAD		
unsigned char ;	Name[8]	Array for name of section
uint32_t	PhysicalAddress	Physical Address
uint32_t	VirtualSize	Virtual Size
uint32_t	VirtualAddress	Virtual Address
uint32_t	SizeOfRawData	Size of Raw Data

uint32_t	PointerToRawData	Pointer to Raw Data
uint32_t	PointerToRelocations	Pointer to Relocations
uint32_t	PointerToLinenumbers	Pointer to Line numbers
uint16_t	NumberOfRelocations	Number of Relocations
uint16_t	NumberOfLinenumbers	Number of Line numbers
uint32_t	Characteristics	Characteristics

Table 2.1 PE Header Fields [5]

All sections can be easily understood by looking at Table 2.2. In general PE file will have at least two sections, one for code and the other for data.

Section Name	Section Description
.text	Code Section Contains the executable code
CODE	Code Section of file linked by Borland Delphi or Borland Pascal Contains the executable code
.data	Data Section Stores global data accessed throughout the program
DATA	Data Section of file linked by Borland Delphi or Borland Pascal Stores global data accessed throughout the program
.rdata	Section for Constant Data Holds read-only data that is globally accessible within the program
.idata	Import Table 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	Export Table 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	Exception section Present only in 64-bit executable and stores exception-handling information
.tls	TLS Table Contains information of thread local variables
.reloc	Relocation Information Contains information for relocation of library files
.rsrc	Resource Information Stores resources needed by the executable

Table 2.2 PE Section [6]

CHAPTER 3: RELATED WORK

Significant amount of research has been done in past to detect malware using windows PE file format. Moreover, many anti-malware vendors have adopted different methods to identify the malicious executables. Researcher and malware analysts have applied different approaches in determining the process to detect malware by using static and/or dynamic malware analysis technique. Additionally various data mining technique have also been used. The following sections discuss the related work.

3.1 Malware Detection

Much of the research done in malware detection falls into realm of static and/or dynamic malware analysis. Liao [7] has developed a method that extract features from the PE headers and finds top 5 distinguishing features of malware to identify them. He has also developed an Icon extractor to extract icons from PE file and find top 3 icons and eight misleading icons from malware. The dataset consist of 5598 malware and 1237 benign samples. Treadwell [8] presented an obfuscated malware detection technique that scans for suspicious patterns in PE format on packed malware by comparing to signature available by antivirus products. Moreover, Christodorescu [9] scans for malicious pattern in the code.

Comar [10] has developed a novel framework, which detects known and novel malware with high precision. Features were collected from the layer 3 and layer 4 network flow characteristics. The framework then transforms the features using tree, decreasing the imperfections in the data that results in useful features. Alzab [11] has developed an automated system to extract features

from API call and have used n-gram to detect benign or malicious executable. The dataset used for the experiment contains 242 malware and 72 benign files.

3.2 Data Mining

Iwamoto [12] has proposed a method of classifying samples into malware and benign by using API call sequences. Aljamea [13] has proposed a static analysis approach using text based search techniques, control flow graph, hashing, and decision tree algorithm to cluster samples. Similarly, Mutant X-S [14] groups the malware according to similarity in their code instruction whereas, OPEM [15] uses features such as counting the existence of operational codes extracted using static analysis technique and the execution information using dynamic technique.

In 2008, McBoost [16] framework was proposed that is primarily a malware collection tool and its utility as an online real-time malware detection tool is limited due to high processing overheads and relatively low detection rates [17]. However, the scope of their work is limited to the detection of packed executables only. In 2009, Shafiq [17] PE-Miner, a framework, which collects structural features, performs feature reduction and does clustering using data mining was proposed. PE-Miner extracts around 189 features, which were then reduced by RFR, PCA or HWT. Dataset used by PE-Miner have 1447 benign, 10,339 malware from VX heaven and 5,526 malware from Malfease.

3.3 Discussion and Summary

Several methods have been proposed by extracting different features from the PE file format such as DLL's, API function calls, frequency of instruction code, API call sequences, text based

search technique, and hashing. Some methods collect wide range of feature and uses feature reduction technique to reduce the dimensionality of the features. Moreover, many supervised and unsupervised data mining techniques such as decision tree, IBk, J48, NB, SMO, RIPPER, hierarchical, K-mean and many more has been used to cluster or classify the samples.

We have proposed a method, which extracts fewer features from the PE header and compares various data mining methods to increase the accuracy of the clusters, which in turn improves the accuracy of detecting benign or malicious executables. The features that we used were file size, number of sections, number of unknown sections, number of dll called, Size of Code, Size of initialized data, size of uninitialized data, Size of Image, Check sum, DLL Characteristics, Size of Stack Reserve, Size of Stack Commit and no. of directories. We have compared three unsupervised data mining techniques hierarchical, K-mean and SOM [18].

CHAPTER 4: PROPOSED METHODOLOGY

4.1 Proposed Approach

Malware has become serious problem, since the use of computers and high-speed network is increasing in our day-to-day life. Thus there is an urgent need for effective methods to categorize the malware automatically. Malware categorization is an important problem in malware analysis and recently many clustering techniques have been created. However, such techniques have limited effectiveness and few were used commercially.

The main objective of this research is to develop a method that improves the efficiency of distinguishing benign and malicious executables by using data mining, feature reduction and classification algorithms based on static malware analysis. The method can be understood easily from the Figure 4.1.

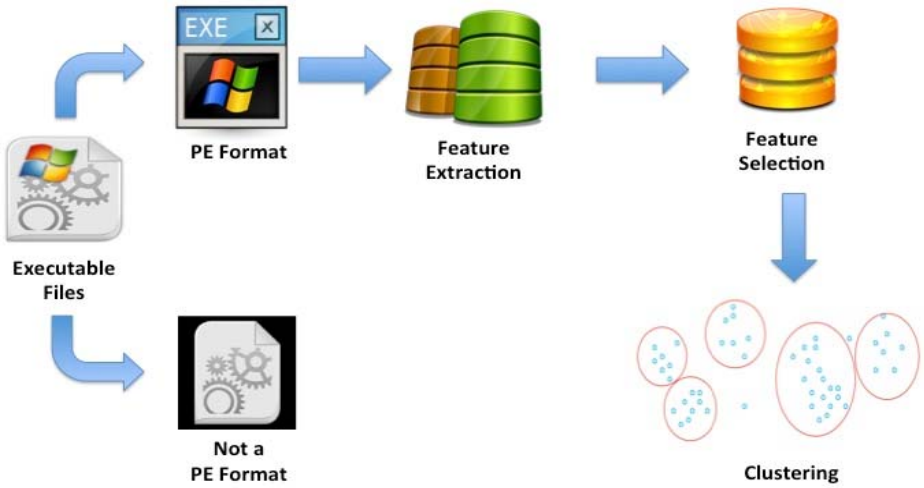

Figure 4. 1 Proposed Methodology

The method can be divided into specific objectives such as:

- Collecting datasets of benign and malicious executables to perform experiments
- Writing code to parse PE format and extract important and useful features
- Selecting the most related features and removing or eliminating the outliers
- Finding appropriate data mining classification techniques such as Hierarchical clustering, K-mean algorithm, and SOM neural network to create benign and malicious sample clusters
- Optimizing the approach by comparing various data mining techniques and increasing efficiency of cluster

We have written code, which reads the entire directory passed and checks if the executables is a PE file. Once it confirms, it will start extracting useful features and write them to an excel file. Later an algorithm is implemented which selects the most related features and deletes the outliers from the excel file. Then clustering techniques were used to cluster the samples effectively and efficiently. Results of the clustering techniques were compared and optimized.

4.2 Introduction to Data Mining

Data Mining is a computational process of describing or finding similarity or patterns in the large amount of data. It helps in the analysis of data by discovering new trends or correlation in data already present in database in novel ways that it becomes useful to the data analysis. Data mining is considered to be an application of machine learning which uses both pattern recognition technologies as well as statistical and mathematical techniques.

Data mining is also an integral part of knowledge discovery in database (KDDs). The steps involved in KDD process are mentioned below [19].

- Selection – Reducing dataset by maintaining the integrity of original dataset
- Pre-processing – To remove noise or irrelevant data
- Transformation – Transform data into required formats
- Data Mining – To obtain or discover useful patterns in data
- Interpretation/ Evaluation – To identify useful patterns by applying validation and evaluating the results

Below mentioned are primary six tasks involved in data mining process [19].

- Clustering – Identify similarity in data and form groups or clusters
- Classification – Categories new data in predefined classes by discovering predictive learning function.
- Regression – Discovering predictive learning function, which models data with the least error.
- Summarization – Finding a compact representation for a set or subset of data
- Anomaly detection – Identify the most significant changes or errors in data set
- Dependency Modeling – Finding relationship between features or their values in a dataset

Data mining process uses machine learning algorithms depending on whether the class labels are provided for learning, these algorithms can be divided into two categories supervised learning or unsupervised learning.

Supervised Learning uses labeled training data, which consists of both input and desired output values. Supervised learning algorithm is trained using the training dataset and new data or test data is categorized accordingly. Supervised learning algorithms include but are not limited to decision tree, support vector machines (SVM), naïve bayes, random forest, regression models, and nearest neighbor [20].

Unsupervised Learning algorithm tries to find hidden structure by using unlabeled data. The model generating the output much either is stochastic to avoid producing same output each time. Unsupervised algorithms include but are not limited to hierarchical, K-mean, SOM, DBSCAN and OPTICS [21].

4.3 Data Collection

We have created three datasets of malware and benign binaries to obtain optimized results. Our malware datasets were collected from [22] and our benign files were gathered from various versions of Microsoft windows operating system files and various windows application of 32 bit and 64 bit. Our first dataset contains 22,172 binaries, second contains 14,467 binaries and third comprises of 11,960 binaries. The binaries contain both dlls and executables. The overview of dataset is mentioned in below Table 4.1.

(Table 4.1 Cont.)

Dataset	Benign		Malware	
1	Windows 7 Ent SP0 dll	1958	Backdoor	5046
	Windows 7 Ent SP0 exe	442	Worm	5713
	Windows 7 Ent SP1 dll	1941	Flooder	190
	Windows 7 Ent SP1 exe	444		
	Windows 8 Ent SP0 dll	2593		
	Windows 8 Ent SP0 exe	483		
	Windows 8 Ent SP1 dll	2858		
	Windows 8 Ent SP1 exe	504		
	Total No. of Files	**11223**		**10949**
		22172		
	Maximum File Size	**67967488**		**9507381**
	Minimum File size	**718**		**1476**
2	Windows Vista Ent SP0 dll	1566	Trojan	7242
	Windows Vista Ent SP0 exe	387	Packed	366
	Windows Vista Ent SP1 dll	1579	Flooder	381
	Windows Vista Ent SP1 exe	392		
	Windows Vista Ent SP2 dll	1594		
	Windows Vista Ent SP2 exe	394		
	Applications	34		
	System and others	532		
	Total	**6478**		**7989**
		14467		
	Maximum File Size	**30595068**		**34748928**
	Minimum File size	**718**		**2097**
3	Windows XP SP1 dll	1110	Constructor	616
	Windows XP SP1 exe	305	Exploit	649
	Windows XP SP2 dll	1128	Hacktool	705
	Windows XP SP2 exe	311	Rootkit	3180
	Windows XP SP3 dll	1242	Hoax	1125
	Windows XP SP3 exe	323		
	Windows Server 2000 dll	959		
	Windows Server 2000 exe	307		

	Total	**5685**		**6275**
		11960		
	Maximum File Size	**11286096**		**7089664**
	Minimum File size	**817**		**672**

Table 4.1 Data Collection

4.4 Data Pre-processing

The goal of our research was to gather useful features and explore various data mining techniques, which helps to create clusters with high accuracy. Our focus is to extract useful features from PE headers that distinguish between benign and malicious files.

Some binaries collected were not in PE, win 32 bit or win 64 bit format, so we have written code in such a way that it checks if the given binary is in the required format or not before trying to extract features.

Some of the malware collected were packed as hacker tries to hide the content of malware by packing it. Similarly, some of the benign files were packed so that the application can be protected from cracking. In order to extract useful features by disassembling, the packed or compressed binaries were unpacked. Then each binary is disassembled and all the features from the PE headers were extracted.

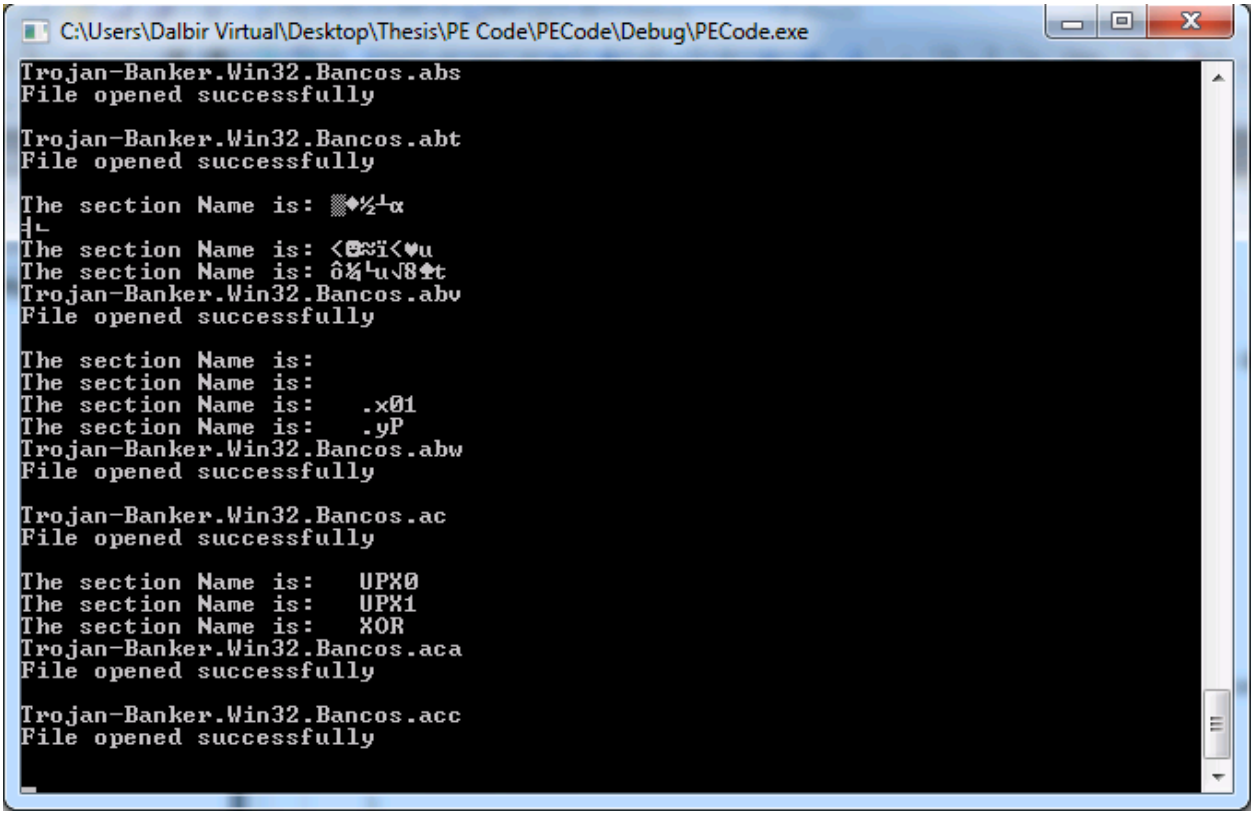

Figure 4.2 PE Headers Extraction

Figure 4.2 shows the output in command prompt window while running the code for feature extraction. Only the unknown section names are printed on the output window.

4.4.1 Feature Extraction

This section explains and describes the features extracted from PE files. We have written code to disassemble the PE file. We have collected 53 features from all the dataset of benign and malicious files. All 53 features are mentioned and described in Table 4.2.

(Table 4.2 Cont.)

SR No.	Feature Name	Header	Description
1	FSize		File Size
2	DBpp	DOS	Bytes on last page of file

3	DPf	DOS	Pages in file
4	DR	DOS	Relocations
5	DShp	DOS	Size of header in paragraphs
6	Dmaxp	DOS	Minimum extra paragraphs needed
7	Dminp	DOS	Maximum extra paragraphs needed
8	Diss	DOS	Initial (relative) SS value
9	DSPv	DOS	Initial SP value
10	Dcsum	DOS	Checksum
11	DIPv	DOS	Initial IP value
12	Dicsv	DOS	Initial (relative) CS value
13	Dreltab	DOS	File address of relocation table
14	DONum	DOS	Overlay number
15	NSec	File	No. of Sections
16	DateTime	File	Date Time
17	FPointerToSymbolTable	File	Pointer To Symbol Table
18	FNumberOfSymbols	File	Number Of Symbols
19	FSizeOfOptionalHeader	File	Size Of Optional Header
20	FCharacteristics	File	Characteristics
21	USec	File	No. of Unknown Sections
22	IDll	File	No. of DLL Function call
23	OMajorLinkerVersion	OPT	Major Linker Version
24	OMinorLinkerVersion	OPT	Minor Linker Version
25	SCode	OPT	Size of Code
26	SIData	OPT	Size of Initialized Data
27	SUData	OPT	Size of Uninitialized Data
28	OAddressOfEntryPoint	OPT	Address Of Entry Point
29	OBaseOfCode	OPT	Base Of Code
30	OBaseOfData	OPT	Base Of Data
31	OImageBase	OPT	Image Base
32	OSectionAlignment	OPT	Section Alignment
33	OFileAlignment	OPT	File Alignment
34	OMajorOperatingSystemVersion	OPT	Major Operating System Version
35	OMinorOperatingSystemVersion	OPT	Minor Operating System Version
36	MImageVer	OPT	Major Image Version
37	OMinorImageVersion	OPT	Minor Image Version
38	OMajorSubsystemVersion	OPT	Major Subsystem Version
39	OMinorSubsystemVersion	OPT	Minor Subsystem Version
40	OReserved1	OPT	Reserved1

41	OSizeOfImage	OPT	Size Of Image
42	OSizeOfHeaders	OPT	Size Of Headers
43	CSum	OPT	CheckSum
44	OSubsystem	OPT	Subsystem
45	DLLChar	OPT	Dll Characteristics
46	OSizeOfStackReserve	OPT	Size Of Stack Reserve
47	OSizeOfStackCommit	OPT	Size Of Stack Commit
48	OSizeOfHeapReserve	OPT	Size Of Heap Reserve
49	OSizeOfHeapCommit	OPT	Size Of Heap Commit
50	OLoaderFlags	OPT	Loader Flags
51	ONumberOfRvaAndSizes	OPT	Number Of Rva and Sizes
52	NDirectories	OPT	No. of Directories
53	DModified		Date Modified

Table 4.2 PE Features Extraction

Features like USec, IDll and DModified are the features, which were created using the data obtained from PE headers. USec feature is for calculating number of unknown section name in PE file. A counter representing USec feature is initialized and incremented if a section name obtain from PE file is other than common name like .text, .data, .pdata, .rdata, .rsrc, .reloc, .tls, .idata, .edata, CODE, DATA or BSS. Similarly, IDLL is for counting number of dlls called in Import directory. DModified is the modification date of PE file. This feature is collected from the properties of the PE file.

4.5 Feature Selection

All 53 extracted features were considered as initial set of features. Several experiments were carried out using this set of features but the accuracy obtained after applying data mining techniques was average. So, to improve the performance and accuracy of clustering with minimum measurements and low overheads, we removed the features that were outliers or not

giving useful information. Thus we have selected 13 features, which are mentioned and described in Table 4.3.

SR No.	Feature Name	Header	Description
1	FSize		File Size
2	NSec	File	No. of Sections
3	USec	File	No. of Unknown Sections
4	IDll	File	No. of DLL Function call
5	SCode	OPT	Size of Code
6	SIData	OPT	Size of Initialized Data
7	SUData	OPT	Size of Uninitialized Data
8	OSizeOfImage	OPT	Size Of Image
9	CSum	OPT	Check Sum
10	DLLChar	OPT	Dll Characteristics
11	OSizeOfStackReserve	OPT	Size Of Stack Reserve
12	OSizeOfStackCommit	OPT	Size Of Stack Commit
13	NDirectories	OPT	No. of Directories

Table 4.3 PE Features Selection

SR No.	Feature Name	Header	Description
1	NSec	File	Normalized No. of Sections
2	USec	File	Weighted No. of Unknown Sections
3	IDll	File	Normalized No. of DLL Function call
4	SCode	OPT	Normalized Size of Code
5	SIData	OPT	Normalized Size of Initialized Data
6	SUData	OPT	Normalized Size of Uninitialized Data
7	NDirectories	OPT	Normalized No. of Directories

Table 4.4 PE Features

Now for better understanding of the data, various features were normalized and given weightage. Normalization helps in comparing the values for different data sets in a way that it eliminates the effects of certain gross influences. There were some features in our list of selected features whose data may vary according to the size of files. Thus for better comparison we have

normalized the features by dividing the data with its file size. We have also given more weightage to a feature by multiplying its data with constant integer value hundred. The below mentioned Table 4.4 contains normalized and weighted features.

4.6 Clustering Algorithm

The clustering algorithm forms group or clusters such that member inside a cluster are similar, and distinct from the objects in other clusters. For statistical analysis of data, clustering method is used frequently in the field of data mining. The cluster or groups can be formed, by using various algorithms that differ in the criteria. The criteria used for forming clusters may include groups with small distances among the data, dense areas, intervals or particular statistical distributions [23].

We have used hierarchical, K-mean and self-organizing map clustering algorithm for our data analysis. The notion of each of the clustering algorithm is different and varies significantly in their properties.

4.6.1 Hierarchical Learning Algorithm

Hierarchical clustering is an example of connectivity models, which uses distance connectivity to create groups or clusters. The hierarchy is a tree of clusters, which contains a node as child clusters while sibling cluster share a common parent node [24].

We have used agglomerative hierarchical cluster analysis that is already implemented in Matlab. The hierarchical algorithm distinguishes every pair of data in dataset. Then it links the pair of data in close space using the distance criteria that is generated by distinguishing every pair of data in dataset. In next step a binary tree is formed using data paired into binary clusters. At last, the clusters were created either by finding natural clusters in the binary tree or by using cutting off criteria [25].

The function *pdist* calculates the Euclidean distance between each data pair to distinguish every pair of data in dataset. The *linkage* function clusters data into binary tree and *dendogram* function helps to view the binary cluster tree graphically for better understanding. The function *cluster* is applied to find cut in tree to form clusters either naturally or by giving arbitrary cutoff value in function itself. Figure 4.3 shows the code written in Matlab for hierarchical clustering.

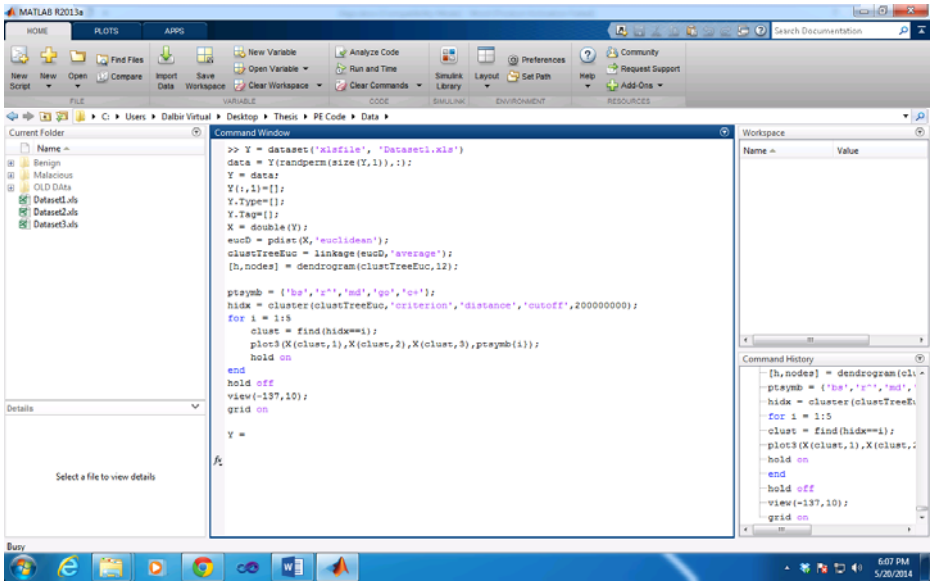

Figure 4.3 Hierarchical Matlab Code

As mentioned in the above Hierarchical Matlab code, we used distance cutoff to create clusters of the dataset.

4.6.2 K-mean Learning Algorithm

K-mean is an example of partition based clustering algorithm. The k-mean algorithm forms clusters from the data by allocating an index number to each observation according to their mean value. Each cluster is created or given same index number with nearest mean value. Unlike hierarchical clustering, k-mean creates cluster on single level and using each observation rather than similarities or dissimilarities between data pair. Thus k-mean is more suitable and efficient for cluster analysis of large datasets [26].

K-mean treats each observation in the data as unique. It creates partition in which objects within each cluster are similar, and distinct from the objects in other clusters [27]. The centroid of the cluster is calculated by the sum of distances from all data and it helps to increase the efficiency of the cluster. K-mean iteratively improves by minimizing the sum of distances of each data from its clusters centroid. The algorithm iterates until the sum of distances of data from centroid cannot be reduced further. Thus the resultant clusters were compact and well separated as possible.

The k-mean algorithm is already implemented in Matlab. As explained earlier, k-mean algorithm returns index of the clusters. Figure 4.4 represents the code written in Matlab to use k-mean algorithm.

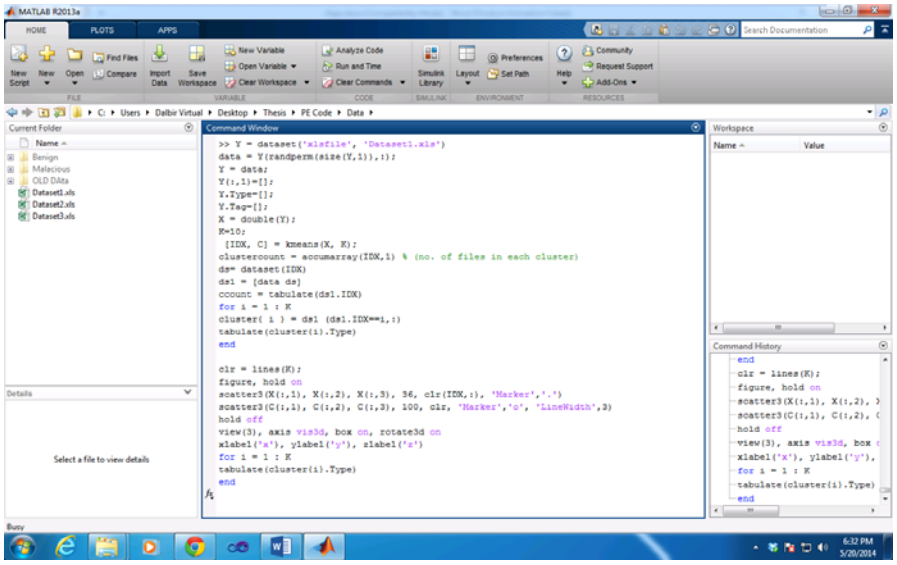

Figure 4.4 K-mean Matlab Code

As mentioned in Figure 4.4 we asked the K-mean algorithm to create 10 clusters at maximum. We came up with cluster number 10 by trial and error. When number of clusters was increased, empty clusters were created and when the number of clusters was decreased, there was significant drop in the accuracy.

4.6.3 Self Organizing Mapping Algorithm

The SOM is an excellent tool in exploratory phase of data mining [18]. SOM is a type of artificial neural network (ANN) that produces discrete clusters by training the network using unsupervised learning. SOM uses neighborhood function to preserve the topological properties of input [28]. SOM groups the data based on similarity between objects. Thus it can be used to create clusters of benign and malicious objects.

The other version of SOM training, which we have used is called batch algorithm. It presents whole dataset to the network then algorithm determines winning neuron and updates each weight vector by moving it to the average position of all input vector [29].

Matlab has a tool, which can be used to run SOM algorithm. As shown in Figure 4.5 we can first create grid and train data using batch algorithm in Figure 4.6.

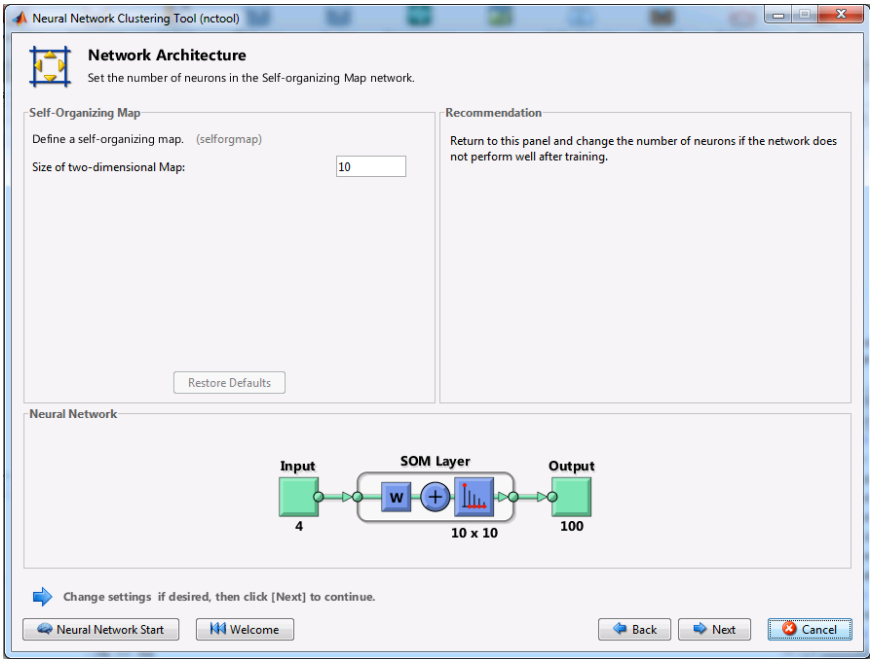

Figure 4.5 Neural Network Architecture

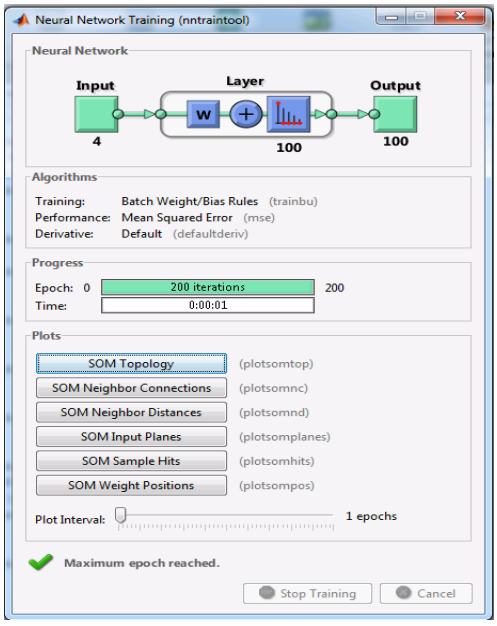

Figure 4.6 Neural Network Training

SOM creates different type of plots for the better understanding of analysis results. Figure 4.7 gives view of SOM topology.

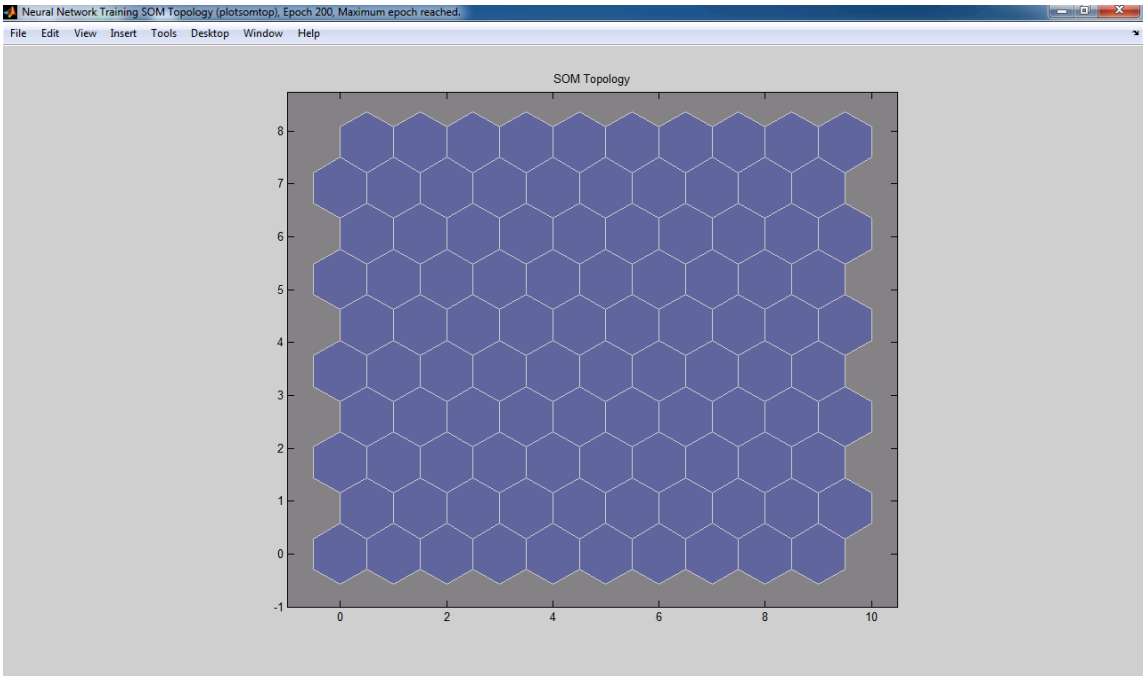

Figure 4.7 SOM Topology

Figure 4.8 shows SOM neighbor connections

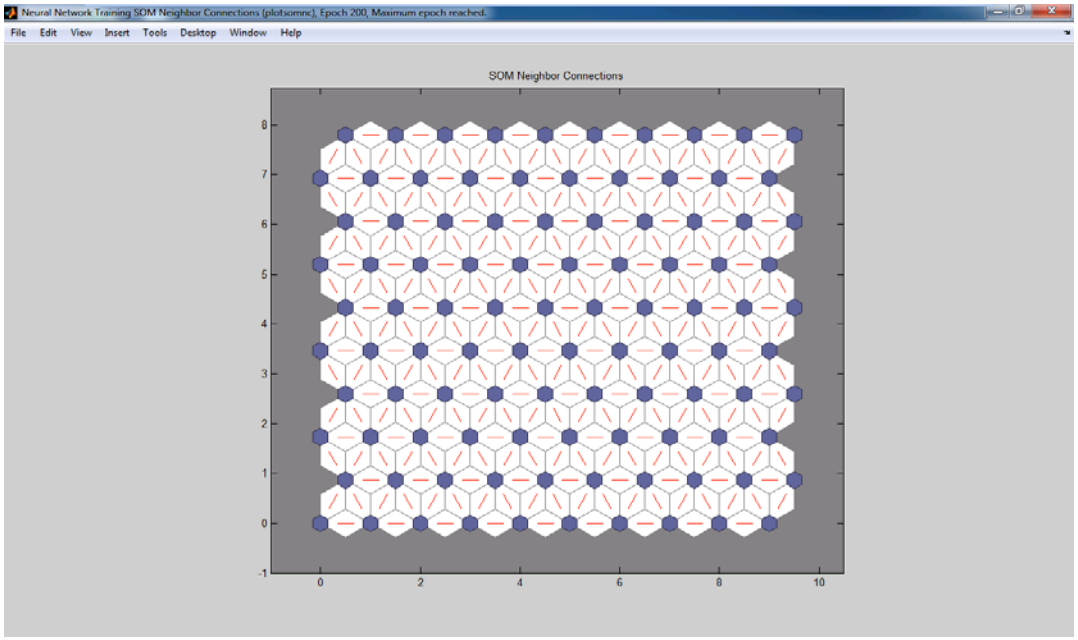

Figure 4.8 SOM Neighbor Connections

Figure 4.9 depicts SOM neighbor weight distances.

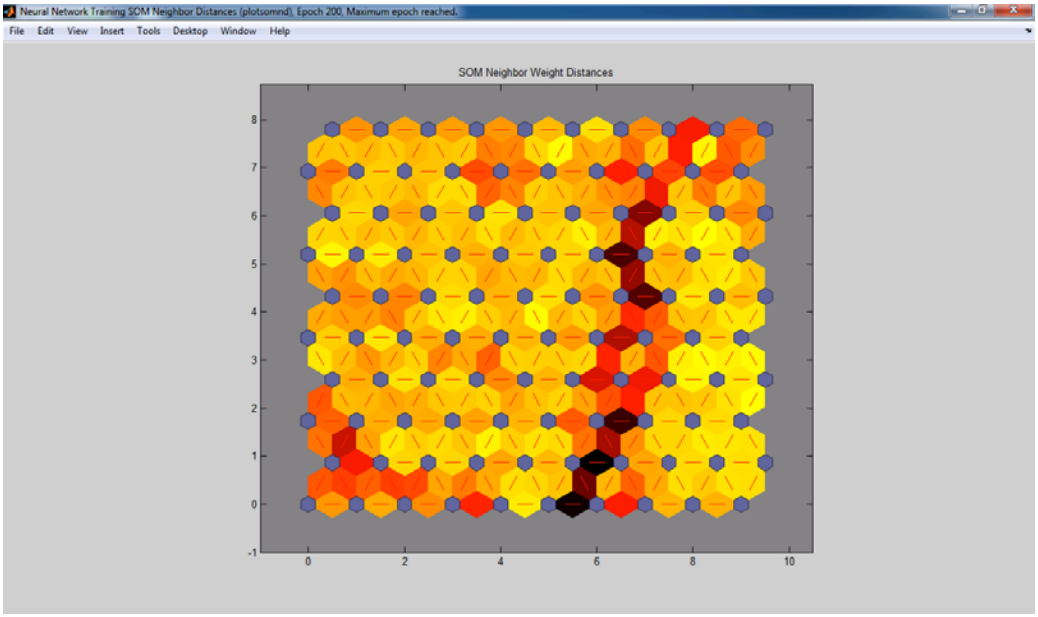

Figure 4.9 SOM Neighbor Weight Distances

Figure 4.10 gives overview of SOM weight planes.

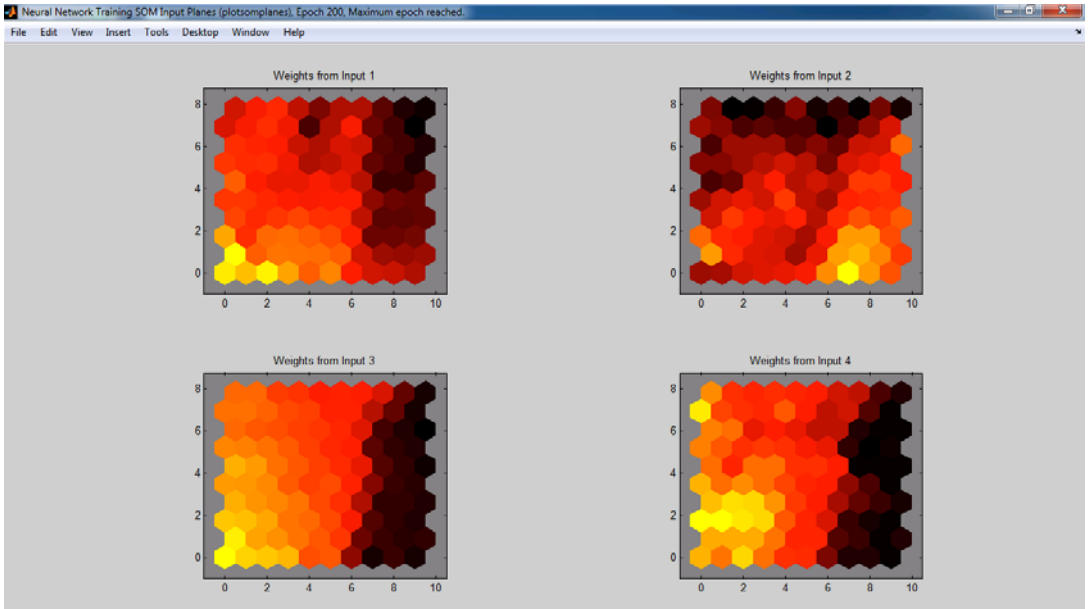

Figure 4.10 SOM Weight Planes

Figure 4.11 shows SOM weight positions.

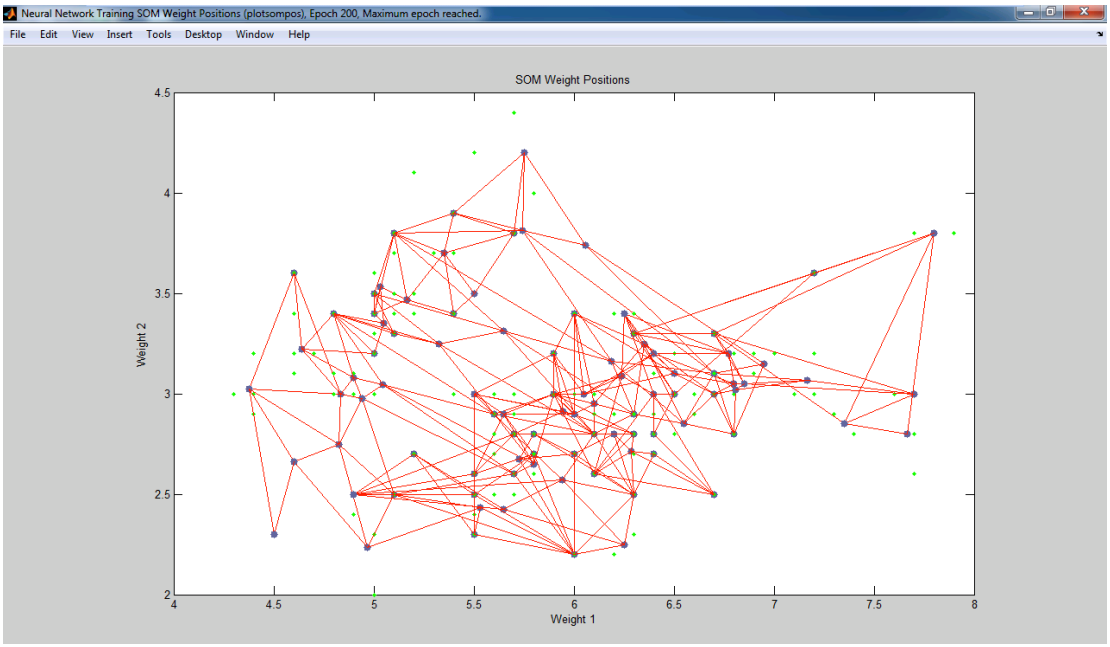

Figure 4.11 SOM Weight Positions

Figure 4.12 gives gist on SOM sample hits.

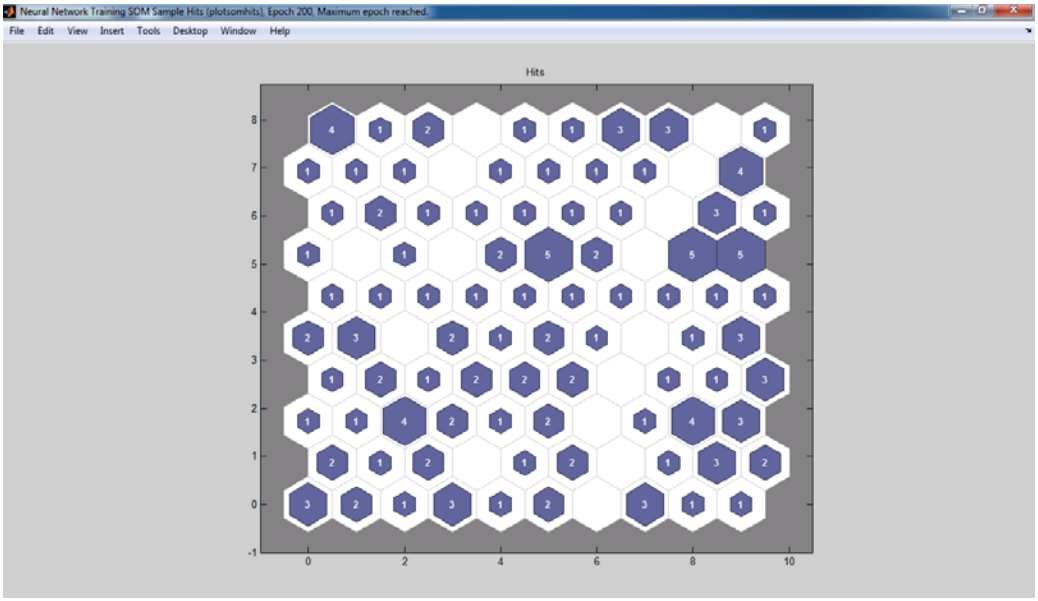

Figure 4.12 SOM Sample Hits

In SOM weight distance and planes, the weights were closer if it is indicated by light colors and the distances were larger if it is indicated by darker band. The color difference distinguishes between the data points weight and distance.

Figure 4.13 represents the code written in Matlab to use SOM algorithm.

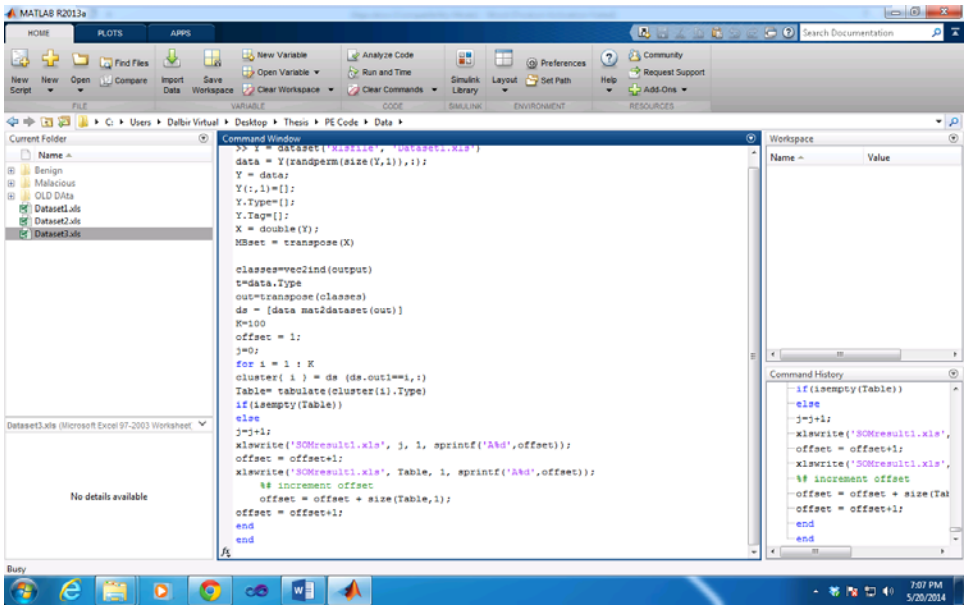

Figure 4.13 SOM Matlab Code

As mentioned in the code, SOM algorithm is first trained and output is saved in Matlab. Then using output, clusters were created. The algorithm is asked to create maximum of 100 clusters.

CHAPTER 5: OUTCOME OF PROJECT

5.1 Experimental Tools and Environment

This section gives an overview on the tools and environment used for the thesis project.

5.1.1 Microsoft Visual Studio

Visual Studio (VS) is a comprehensive collection of tools and services that helps to create a wide variety of applications [30]. VS is an integrated development environment (IDE) from Microsoft that supports different programming languages that allows to write, edit or debug code. We have created Visual C++ (VC++) project using Microsoft VS to disintegrate or collect PE headers values from executables and dynamic link library files. Microsoft VC++ implements both C and C++ compiler and specific tools for integration with VS IDE. We have added several packages and in-built libraries to write code.

5.1.2 Matlab

Matlab is a language with strong abstraction and easy to use visual interface that can be used for data analysis, developing algorithm, creating models and applications, for programming and more. The built-in tools, mathematical functions and algorithms help to explore multiple approaches and get the result faster. Matlab can be used in various fields but not limited to computational biology, computational finance, simulation, image and video processing, signal processing, data analysis, control systems [31].

5.1.3 VMware Workstation

VMware workstation is a hypervisor, which helps users to create one or more virtual machines (VM) on a single system. Each VM can run different operating systems such as Linux, Ubuntu, and different version of Windows. VMware workstation supports host network, share hard drives, and USB devices with VM [32].

VM plays an integral part in malware analysis. VM enables us to create a virtual environment with all the necessary tools and applications required according to our need. We have used VMware workstation version 10.0.1 build-1379776 with windows 7 operating system and 5.7 GB RAM.

VM has a very useful and important feature called snapshot, which saves the state of machine. We can create snapshots of our VM that will save all the data, application installed and settings of the operating system at that very moment. Once the snapshot is created we can restore the saved state whenever it is necessary. For example, if a malware affects the system we can restore the machine to its stored state by restoring a snapshot. The snapshot feature saves lot of time and work while doing malware analysis.

5.2 Experimental Results

Experiment 1:

Input: Dataset1

Algorithm: Hierarchical algorithm

Output: Figure 5.1 represent the binary tree of hierarchical algorithm on dataset1 in dendogram form with top twelve nodes. We have created dendogram with cutoff as the dataset is very large and dendogram representing the complete dataset was visually not clear. Now, to go one step further we have created clusters and represented in Figure 5.2. Similarly, for clearer view we have represented few cluster and data points in the graph.

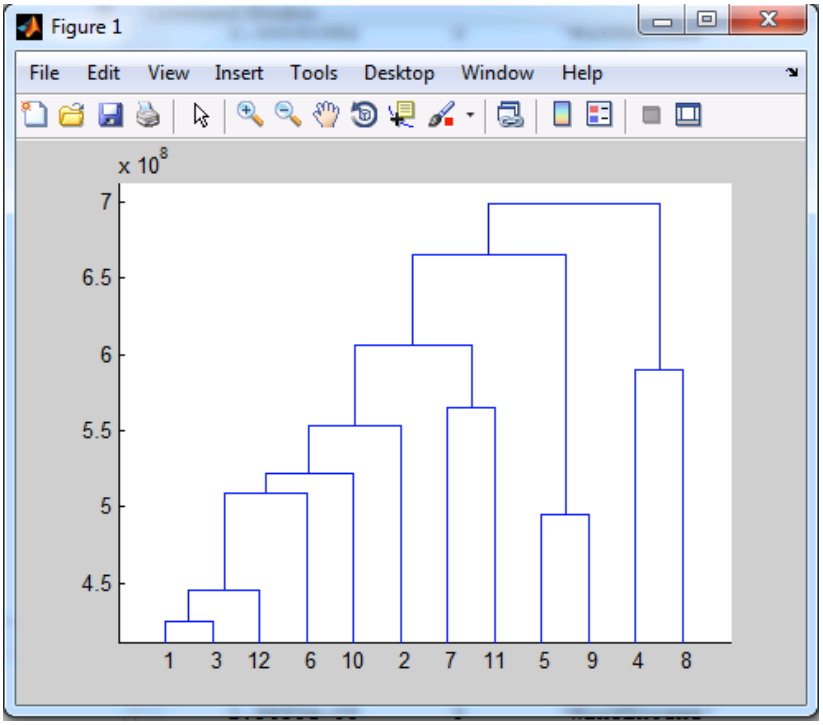

Figure 5.1 Experiment 1 binary tree

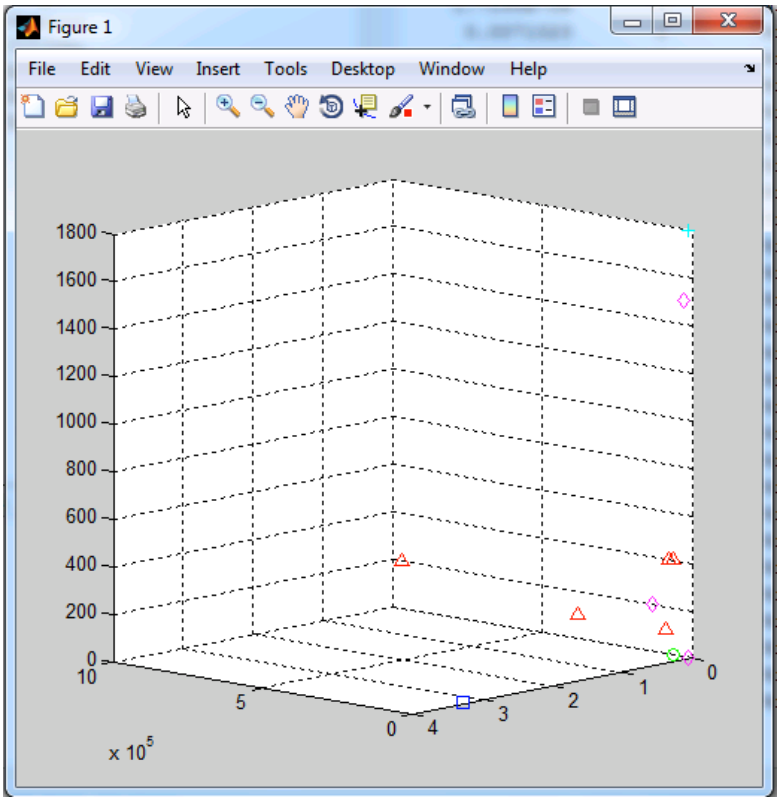

Figure 5.2 Experiment 1 Plot

Clusters were created using cutoff on distance criteria. Table 5.1 represents the 28 clusters created with useful details. Column "Value" represents the number of executables with type benign or malicious and "Percentage" gives the idea on percent of executables type in cluster.

(Table 5.1 Cont.)

Hierarchical on Datset 1											
Cluster	**Executable**	**Value**	**Percentage**	**Cluster**	**Executable**	**Value**	**Percentage**	**Cluster**	**Executable**	**Value**	**Percentage**
1	Benign	0	0.00%	**11**	Benign	8	50.00%	**20**	Benign	11127	51.01%
	Malacious	1	100.00%		Malacious	8	50.00%		Malacious	10686	48.99%
2	Benign	0	0.00%	**12**	Benign	0	0.00%	**21**	Benign	7	77.78%
	Malacious	59	100.00%		Malacious	1	100.00%		Malacious	2	22.22%
3	Benign	6	85.71%	**13**	Benign	3	100.00%	**22**	Benign	0	0.00%
	Malacious	1	14.29%		Malacious	0	0.00%		Malacious	56	100.00%
4	Benign	4	100.00%	**14**	Benign	0	0.00%	**23**	Benign	3	100.00%
	Malacious	0	0.00%		Malacious	1	100.00%		Malacious	0	0.00%
5	Benign	3	100.00%	**15**	Benign	0	0.00%	**24**	Benign	0	0.00%
	Malacious	0	0.00%		Malacious	1	100.00%		Malacious	91	100.00%

6	Benign	0	0.00%	**16**	Benign	0	0.00%	**25**	Benign	4	100.00%
	Malacious	30	100.00%		Malacious	1	100.00%		Malacious	0	0.00%
7	Benign	6	100.00%	**17**	Benign	0	0.00%	**26**	Benign	7	100.00%
	Malacious	0	0.00%		Malacious	3	100.00%		Malacious	0	0.00%
8	Benign	3	50.00%	**18**	Benign	0	0.00%	**27**	Benign	4	100.00%
	Malacious	3	50.00%		Malacious	1	100.00%		Malacious	0	0.00%
9	Benign	3	75.00%	**19**	Benign	0	0.00%	**28**	Benign	3	100.00%
	Malacious	1	25.00%		Malacious	1	100.00%		Malacious	0	0.00%
10	Benign	32	94.12%								
	Malacious	2	5.88%								

Table 5.1 Experiment 1 Clusters

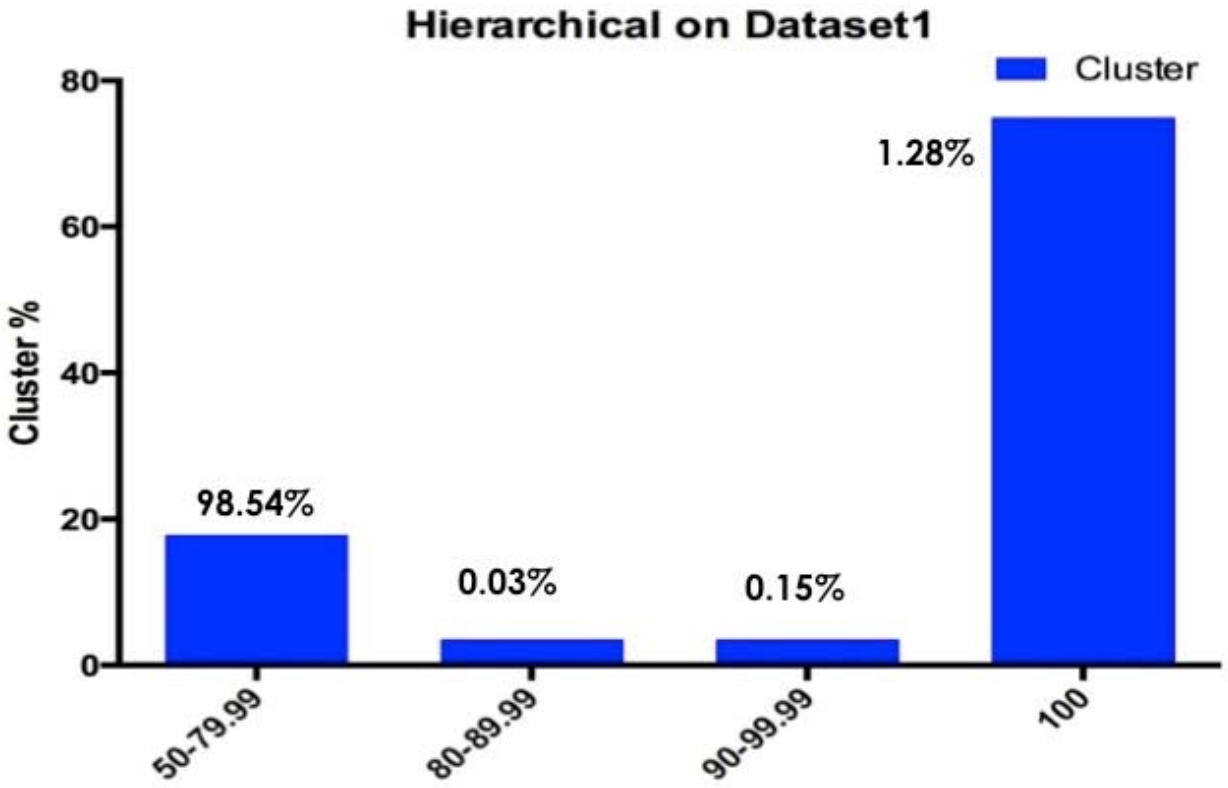

Figure 5.3 Experiment 1 Graph

Analysis: There were around 22,172 executables in dataset. After applying hierarchical algorithm, 28 clusters were created using distance cutoff. Hereafter analyzing the cluster data we found that around 10,702 executables were clustered wrongly, which results in accuracy of 51.73%.

Now we further analyze the results or clusters obtained after applying Hierarchical algorithm on Dataset1. We have created a graph in Figure 5.3 for better understanding of our results. The X-axis represents the cluster with 50% to 100% accuracy rate divided in four groups. The Y- axis represents the percentage of the clusters that belongs to the groups mentioned in X-axis. Each of the bar has a percentage label which represents the percentage of executables from Datset1 that belongs to the group. Moreover, from the figure 5.3 we observe that 98.54% of executables from Dataset1 have 50-79.99% accuracy and less than 20% of clusters belong to this group. Thus, we know that most of the executables from Dataset1 belong to less than 20% of clusters and has efficiency between 50% to 79.99% and rest of the clusters have only 1.46% of executables. Hence, the efficiency obtained in this experiment is very low.

Experiment 2:

Input: Dataset1

Algorithm: K-means algorithm

Output: Figure 5.3 represent centroid of the clusters created using kmean algorithm. Table 5.2 shows the percentage of Benign and Malicious executables in each cluster.

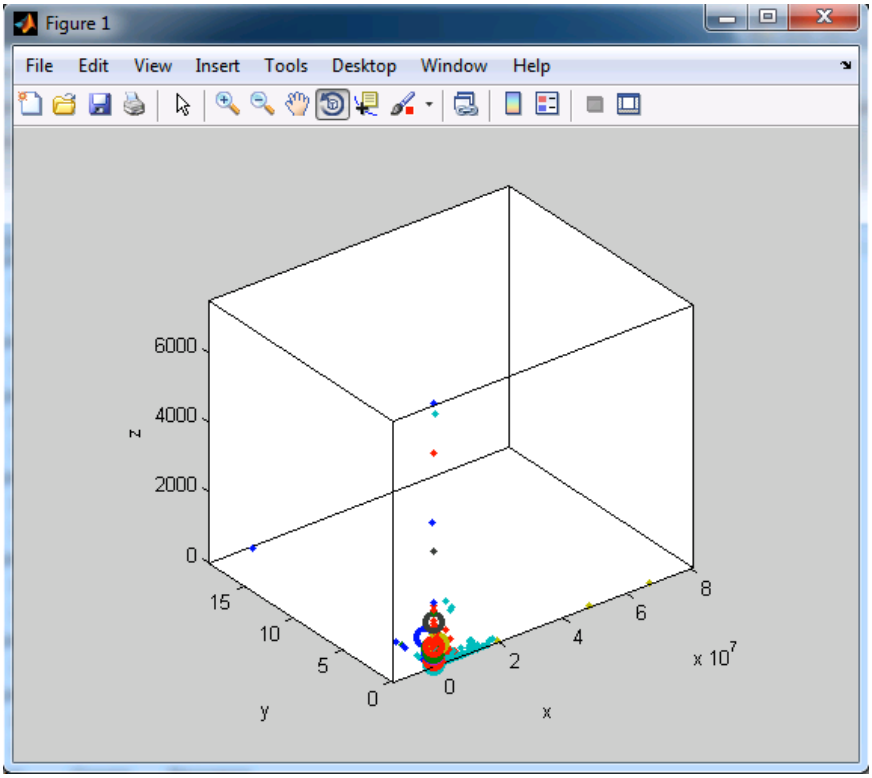

Figure 5.4 Experiment 2 Plot

Kmean on Datset 1							
Cluster	**Executable**	**Value**	**Percentage**	**Cluster**	**Executable**	**Value**	**Percentage**
1	Benign	0	0.00%	**6**	Benign	64	73.56%
	Malacious	73	100.00%		Malacious	23	26.44%
2	Benign	19	79.17%	**7**	Benign	7	63.64%
	Malacious	5	20.83%		Malacious	4	36.36%
3	Benign	333	3.06%	**8**	Benign	50	79.37%
	Malacious	10541	96.94%		Malacious	13	20.63%
4	Benign	10733	98.89%	**9**	Benign	0	0.00%
	Malacious	120	1.11%		Malacious	41	100.00%
5	Benign	0	0.00%	**10**	Benign	17	23.29%
	Malacious	73	100.00%		Malacious	56	76.71%

Table 5.2 Experiment 2 Clusters

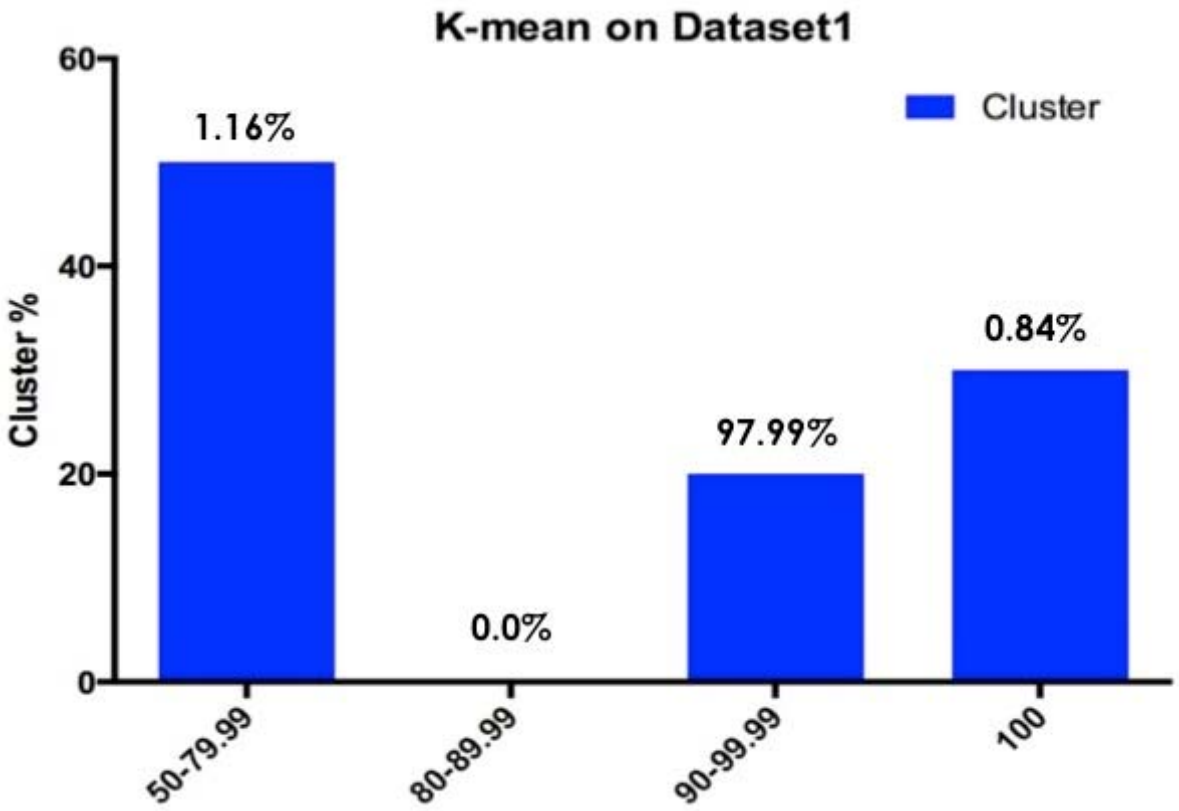

Figure 5.5 Experiment 2 Graph

Analysis: There were around 22,172 executables in dataset that is used for clustering. After applying K-mean algorithm 10 clusters are created. The number of clusters were decided by trial and error and given to the algorithm. If we create more than 10 clusters than empty clusters were created and if we give less than accuracy is reduced significantly. After analyzing the details of each cluster given in Table 5.2, we found that out of 22,172 around 515 executables were wrongly clustered, which gives accuracy of 97.68%. This accuracy is significantly more than obtained in experiment1.

Similar to Experiment 1 we have created a graph in Figure 5.5 for better understanding and analysis of the results or clustered obtained by applying K-mean algorithm on Dataset1. As shown in Figure 97.99% executables from Dataset1 belongs to the clusters having accuracy

of 90-99.99% but out of the clusters created approximately 20% of clusters have this kind of efficiency.

Experiment 3:

Input: Dataset1

Algorithm: SOM algorithm

Output: A grid of 10 X 10 is created according to the input. Figure 5.4 represents the sample hits, which means it shows the number of elements in each cluster. Table 5.3 gives the details of each cluster.

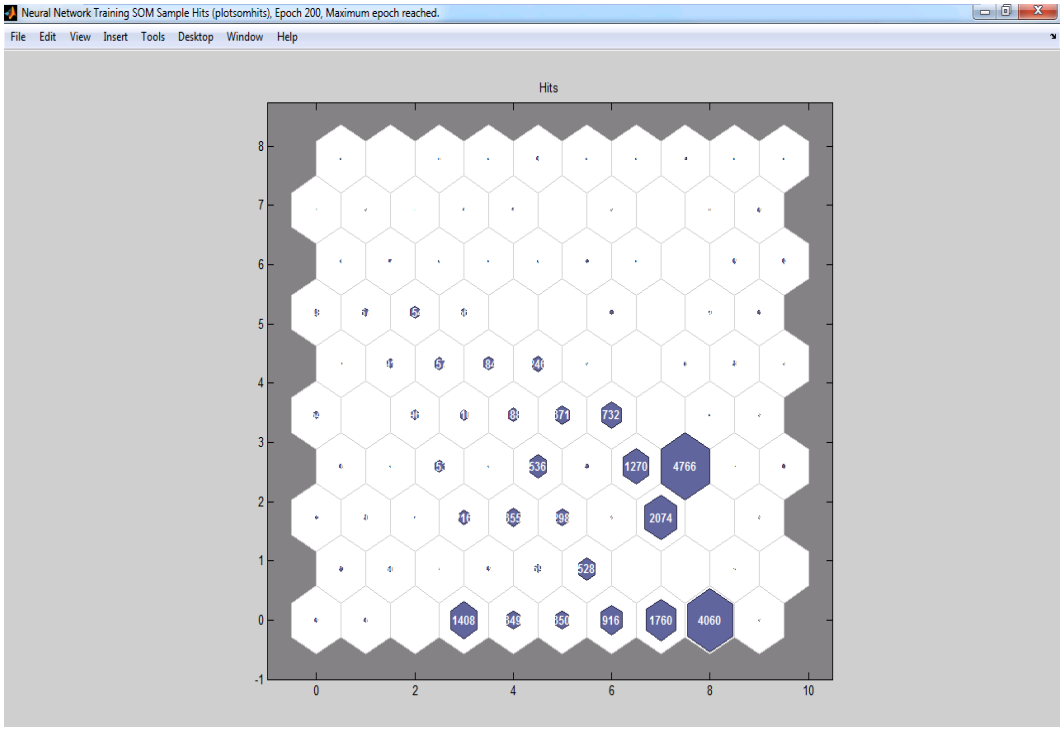

Figure 5.6 Experiment 3 SOM Hits

(Table 5.3 Cont.)

SOM on Datset 1											
Cluster	**Executable**	**Value**	**Percentage**	**Cluster**	**Executable**	**Value**	**Percentage**	**Cluster**	**Executable**	**Value**	**Percentage**
1	Benign	0	0.00%	**30**	Benign	536	100.00%	**58**	Benign	0	0.00%
	Malacious	19	100.00%		Malacious	0	0.00%		Malacious	9	100.00%
2	Benign	0	0.00%	**31**	Benign	22	95.65%	**59**	Benign	0	0.00%
	Malacious	16	100.00%		Malacious	1	4.35%		Malacious	11	100.00%
3	Benign	68	4.83%	**32**	Benign	1270	100.00%	**60**	Benign	10	100.00%
	Malacious	1340	95.17%		Malacious	0	0.00%		Malacious	0	0.00%
4	Benign	67	19.20%	**33**	Benign	4718	98.99%	**61**	Benign	3	21.43%
	Malacious	282	80.80%		Malacious	48	1.01%		Malacious	11	78.57%
5	Benign	0	0.00%	**34**	Benign	0	0.00%	**62**	Benign	6	100.00%
	Malacious	350	100.00%		Malacious	1	100.00%		Malacious	0	0.00%
6	Benign	0	0.00%	**35**	Benign	0	0.00%	**63**	Benign	3	100.00%
	Malacious	916	100.00%		Malacious	13	100.00%		Malacious	0	0.00%
7	Benign	16	0.91%	**36**	Benign	51	94.44%	**64**	Benign	6	100.00%
	Malacious	1744	99.09%		Malacious	3	5.56%		Malacious	0	0.00%
8	Benign	118	2.91%	**37**	Benign	93	96.88%	**65**	Benign	11	91.67%
	Malacious	3942	97.09%		Malacious	3	5.56%		Malacious	1	8.33%
9	Benign	4	80.00%	**38**	Benign	103	93.64%	**66**	Benign	3	100.00%
	Malacious	1	20.00%		Malacious	7	6.36%		Malacious	0	0.00%
10	Benign	0	0.00%	**39**	Benign	188	100.00%	**67**	Benign	0	0.00%
	Malacious	21	100.00%		Malacious	0	0.00%		Malacious	19	100.00%
11	Benign	0	0.00%	**40**	Benign	371	100.00%	**68**	Benign	0	0.00%
	Malacious	30	100.00%		Malacious	0	0.00%		Malacious	18	100.00%
12	Benign	0	0.00%	**41**	Benign	732	100.00%	**69**	Benign	2	100.00%
	Malacious	3	100.00%		Malacious	0	0.00%		Malacious	0	0.00%
13	Benign	0	0.00%	**42**	Benign	0	0.00%	**70**	Benign	3	100.00%
	Malacious	19	100.00%		Malacious	5	100.00%		Malacious	0	0.00%
14	Benign	0	0.00%	**43**	Benign	0	0.00%	**71**	Benign	0	0.00%
	Malacious	69	100.00%		Malacious	8	100.00%		Malacious	1	100.00%
15	Benign	12	2.27%	**44**	Benign	0	0.00%	**72**	Benign	6	66.67%
	Malacious	516	97.73%		Malacious	1	100.00%		Malacious	3	33.33%
16	Benign	0	0.00%	**45**	Benign	88	96.70%	**73**	Benign	7	77.78%
	Malacious	2	100.00%		Malacious	3	3.30%		Malacious	2	22.22%
17	Benign	0	0.00%	**46**	Benign	0	0.00%	**74**	Benign	0	0.00%

	Malacious	11	100.00%		Malacious	157	100.00%		Malacious	3	100.00%
18	Benign	0	0.00%	**47**	Benign	168	91.30%	**75**	Benign	4	100.00%
	Malacious	20	100.00%		Malacious	16	8.70%		Malacious	0	0.00%
19	Benign	0	0.00%	**48**	Benign	246	100.00%	**76**	Benign	0	0.00%
	Malacious	2	100.00%		Malacious	0	0.00%		Malacious	19	100.00%
20	Benign	1	0.46%	**49**	Benign	3	100.00%	**77**	Benign	0	0.00%
	Malacious	215	99.54%		Malacious	0	0.00%		Malacious	3	100.00%
21	Benign	27	7.61%	**50**	Benign	0	0.00%	**78**	Benign	4	100.00%
	Malacious	328	92.39%		Malacious	18	100.00%		Malacious	0	0.00%
22	Benign	3	1.01%	**51**	Benign	0	0.00%	**79**	Benign	6	100.00%
	Malacious	295	98.99%		Malacious	28	100.00%		Malacious	0	0.00%
23	Benign	8	100.00%	**52**	Benign	0	0.00%	**80**	Benign	8	50.00%
	Malacious	0	0.00%		Malacious	5	100.00%		Malacious	8	50.00%
24	Benign	2060	99.32%	**53**	Benign	47	97.92%	**81**	Benign	3	100.00%
	Malacious	14	0.68%		Malacious	1	2.08%		Malacious	0	0.00%
25	Benign	3	60.00%	**54**	Benign	66	98.51%	**82**	Benign	3	100.00%
	Malacious	2	40.00%		Malacious	1	1.49%		Malacious	0	0.00%
26	Benign	0	0.00%	**55**	Benign	0	0.00%	**83**	Benign	7	100.00%
	Malacious	16	100.00%		Malacious	154	100.00%		Malacious	0	0.00%
27	Benign	0	0.00%	**56**	Benign	0	0.00%	**84**	Benign	3	100.00%
	Malacious	3	100.00%		Malacious	56	100.00%		Malacious	0	0.00%
28	Benign	21	13.73%	**57**	Benign	12	92.31%	**85**	Benign	3	100.00%
	Malacious	132	86.27%		Malacious	1	7.69%		Malacious	0	0.00%
29	Benign	0	0.00%								
	Malacious	3	100.00%								

Table 5.3 Experiment 3 Clusters

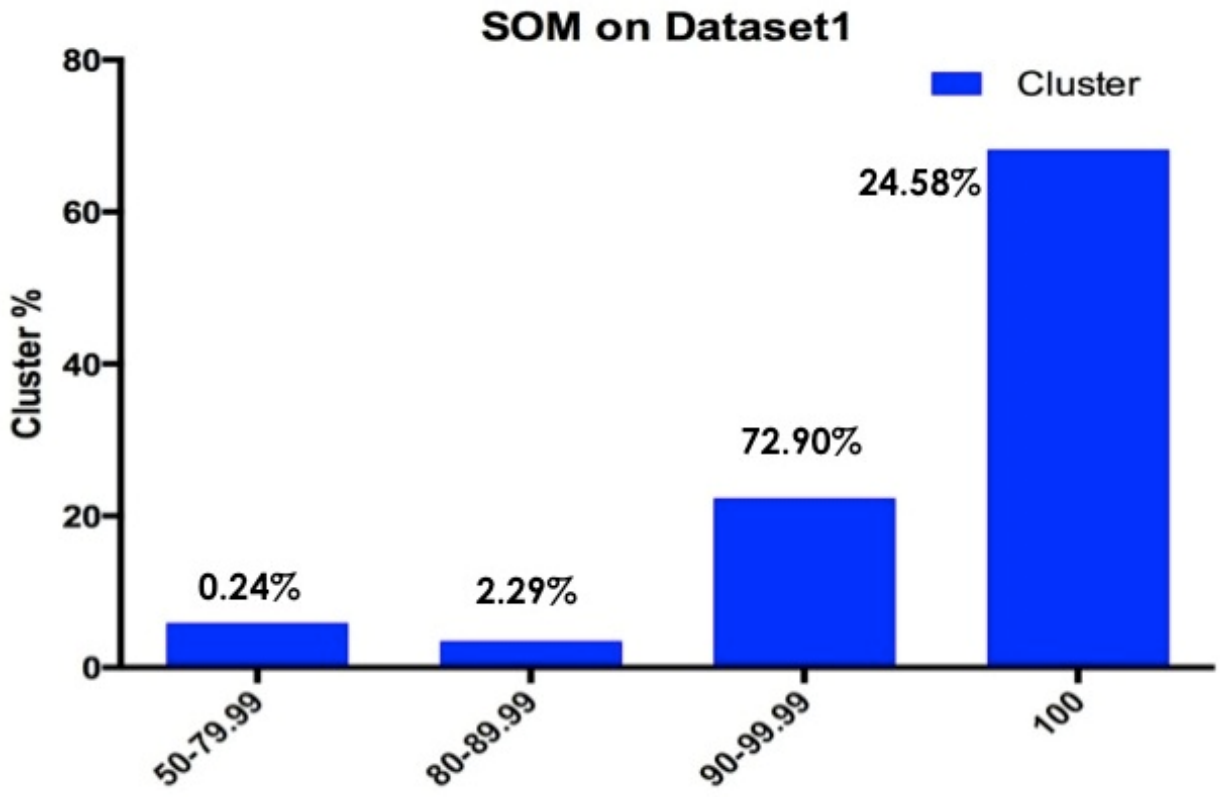

Figure 5.7 Experiment 3 Graph

Analysis: There were around 22,172 executables that were clustered using SOM algorithm. We have asked SOM algorithm to create maximum 100 clusters from the given dataset. SOM creates 85 clusters with elements or data and 15 of them were empty. After analysis of clusters mentioned in Table 5.3 we found that around 451 out of 22,172 were wrongly clustered, which gives accuracy of 97.97%. This accuracy is best so far, obtained for dataset1.

We have analyzed the results or clusters obtained from Dataset1 after applying SOM algorithm and created a graph in Figure 5.7. This graph helps to understand the analysis easily. As we can see that maximum clusters have 90-99.99% or 100% accuracy and around 97% of executables from Dataset belongs to this groups. Thus, this algorithm has the highest accuracy for Dataset1.

Experiment 4:

Input: Dataset2

Algorithm: Hierarchical algorithm

Output: Experiment 4 is same as experiment 1 just the input dataset is different. Same as experiment1 Figure 5.8 represent dendogram and Figure 5.9 represent cluster plot.

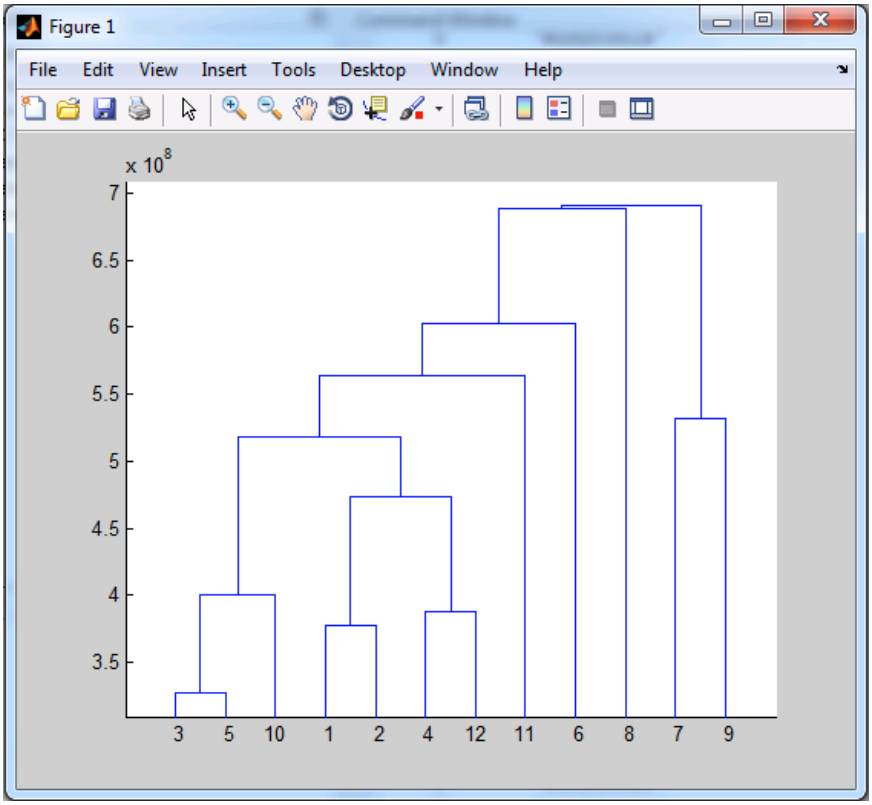

Figure 5.8 Experiment 4 binary tree

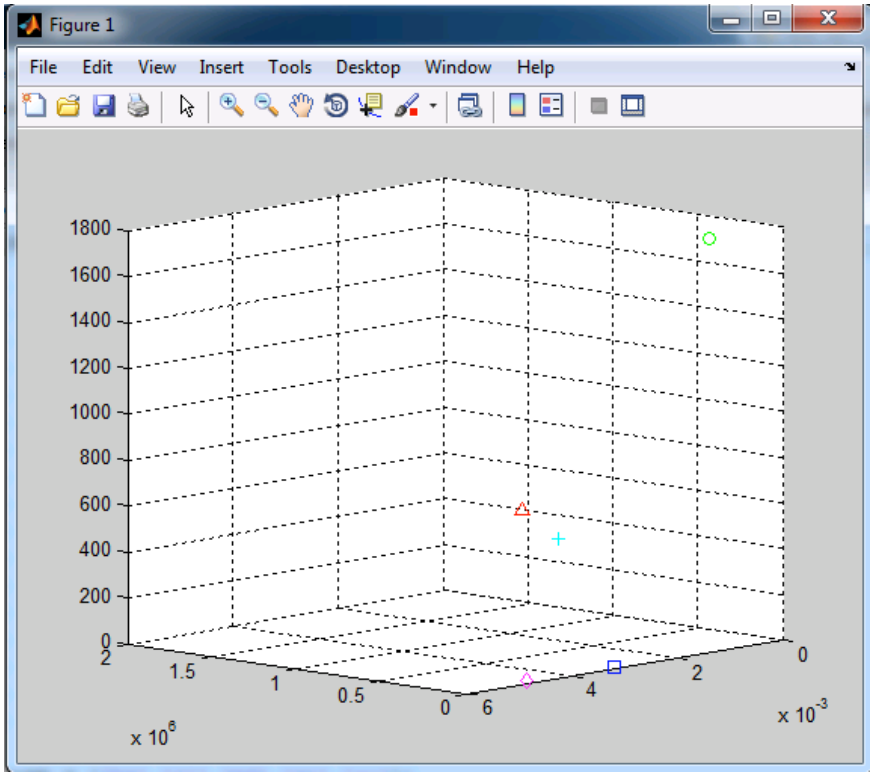

Figure 5.9 Experiment 4 Plot

(Table 5.4 Cont.)

Hierarchical on Datset 2											
Cluster	**Executable**	**Value**	**Percentage**	**Cluster**	**Executable**	**Value**	**Percentage**	**Cluster**	**Executable**	**Value**	**Percentage**
1	Benign	3	100.00%	**10**	Benign	34	100.00%	**19**	Benign	0	0.00%
	Malacious	0	0.00%		Malacious	0	0.00%		Malacious	2	100.00%
2	Benign	0	0.00%	**11**	Benign	0	0.00%	**20**	Benign	4	100.00%
	Malacious	1	100.00%		Malacious	1	100.00%		Malacious	0	0.00%
3	Benign	3	100.00%	**12**	Benign	3	100.00%	**21**	Benign	4	100.00%
	Malacious	0	0.00%		Malacious	0	0.00%		Malacious	0	0.00%
4	Benign	3	100.00%	**13**	Benign	0	0.00%	**22**	Benign	7	87.50%
	Malacious	0	0.00%		Malacious	1	100.00%		Malacious	1	12.50%
5	Benign	0	0.00%	**14**	Benign	6	100.00%	**23**	Benign	11	100.00%
	Malacious	1	100.00%		Malacious	0	0.00%		Malacious	1	0.00%
6	Benign	6	100.00%	**15**	Benign	1	100.00%	**24**	Benign	0	0.00%
	Malacious	0	0.00%		Malacious	0	0.00%		Malacious	8	100.00%
7	Benign	3	100.00%	**16**	Benign	3	100.00%	**25**	Benign	0	0.00%
	Malacious	0	0.00%		Malacious	0	0.00%		Malacious	2	100.00%
8	Benign	0	0.00%	**17**	Benign	6377	44.45%	**26**	Benign	3	100.00%

	Malacious	1	100.00%		Malacious	7969	55.55%		Malacious	0	0.00%
9	Benign	0	0.00%	**18**	Benign	7	100.00%				
	Malacious	1	100.00%		Malacious	0	0.00%				

Table 5.4 Experiment 4 Clusters

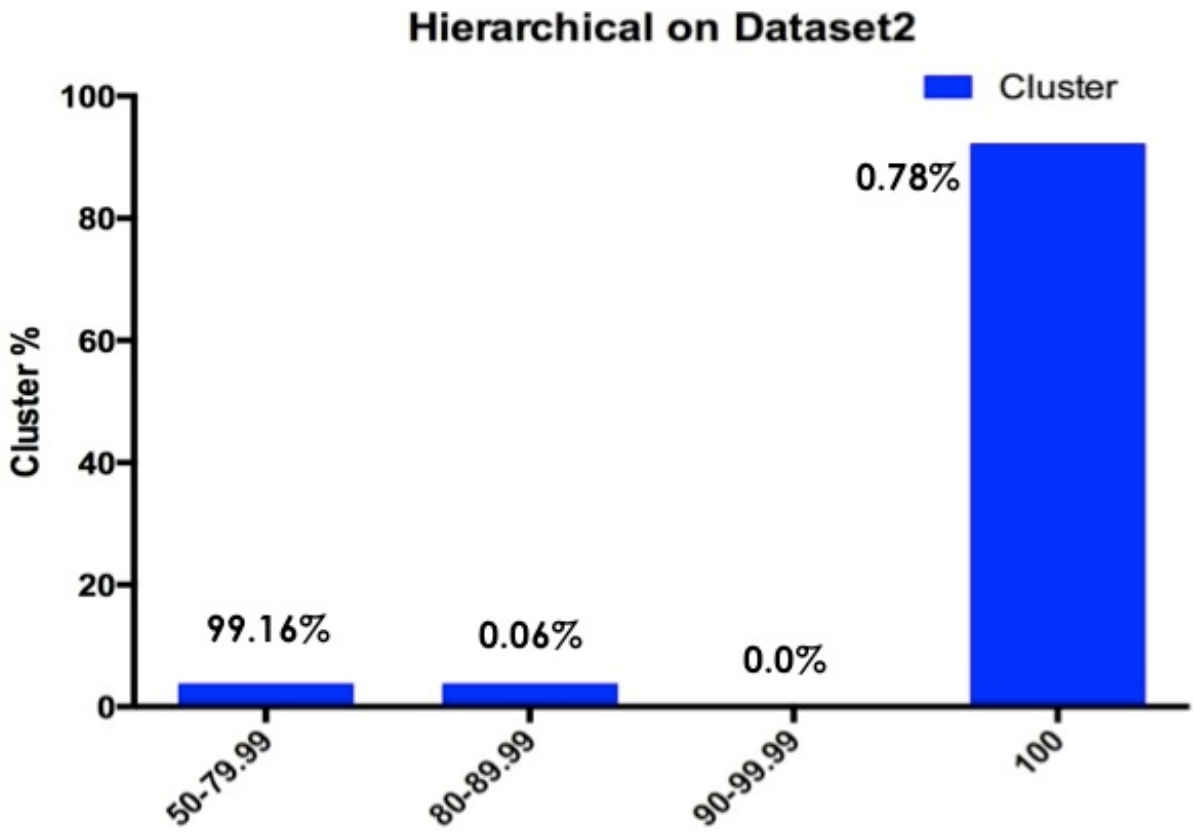

Figure 5.10 Experiment 4 Graph

Analysis: There were around 14,467 executables in dataset, which is used for clustering. Then applying hierarchical algorithm that uses distance cutoff for creating cluster, around 26 clusters were created. After analyzing the cluster details in Table 5.4, we found that out of 14,467 around 6379 were wrongly clustered. This gives accuracy of 55.91% to the experiment, which is low.

The graph in Figure 5.10 represents the analysis results in an efficient way. The graph shows that there were maximum clusters with 100% accuracy but that contains only 0.78% of executables from Dataset2. About 99.16% of executables belongs to the lowest accuracy rates with few clusters only. Thus, Experiment 4 does not serve our purpose.

Experiment 5:

Input: Dataset2

Algorithm: K-mean algorithm

Output: This experiment is same as Experiment 2 just input dataset is different. Figure 5.11 represent centroid of clusters and Table 5.5 represent cluster data.

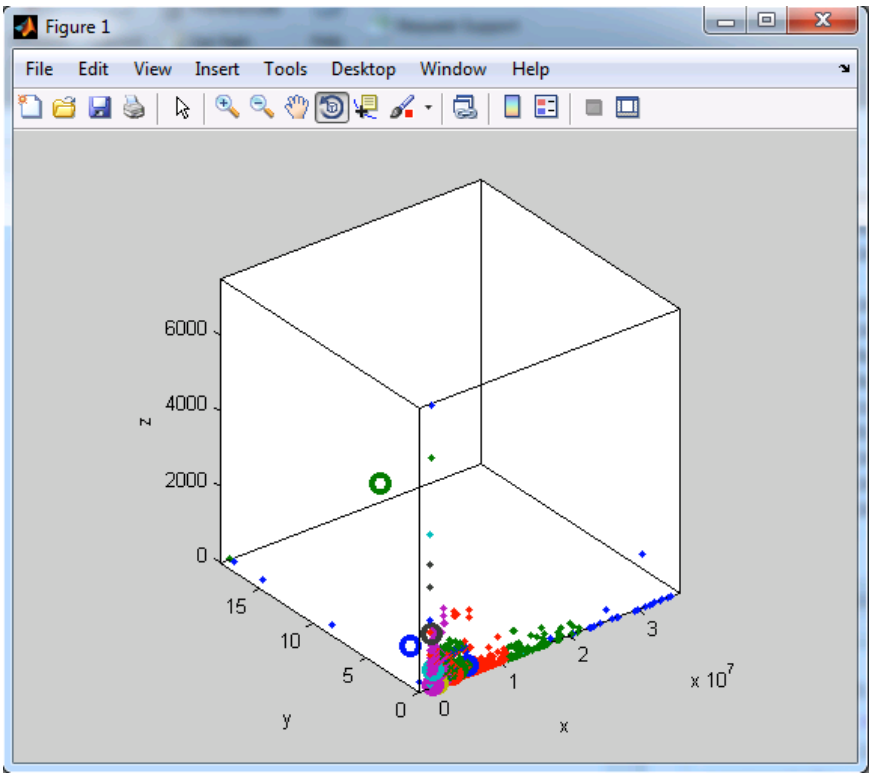

Figure 5.11 Experiment 5 Plot

Kmean on Datset 2							
Cluster	**Executable**	**Value**	**Percentage**	**Cluster**	**Executable**	**Value**	**Percentage**
1	Benign	41	91.11%	**6**	Benign	11	47.83%
	Malacious	4	8.89%		Malacious	12	52.17%
2	Benign	16	1.05%	**7**	Benign	7	77.78%
	Malacious	1505	98.95%		Malacious	2	22.22%
3	Benign	207	11.47%	**8**	Benign	14	4.56%
	Malacious	1598	88.53%		Malacious	293	95.44%
4	Benign	35	94.59%	**9**	Benign	4	100%
	Malacious	2	5.41%		Malacious	0	0.00%
5	Benign	6143	57.33%	**10**	Benign	0	0.00%
	Malacious	4573	42.67%		Malacious	0	0.00%

Table 5.5 Experiment 5 Clusters

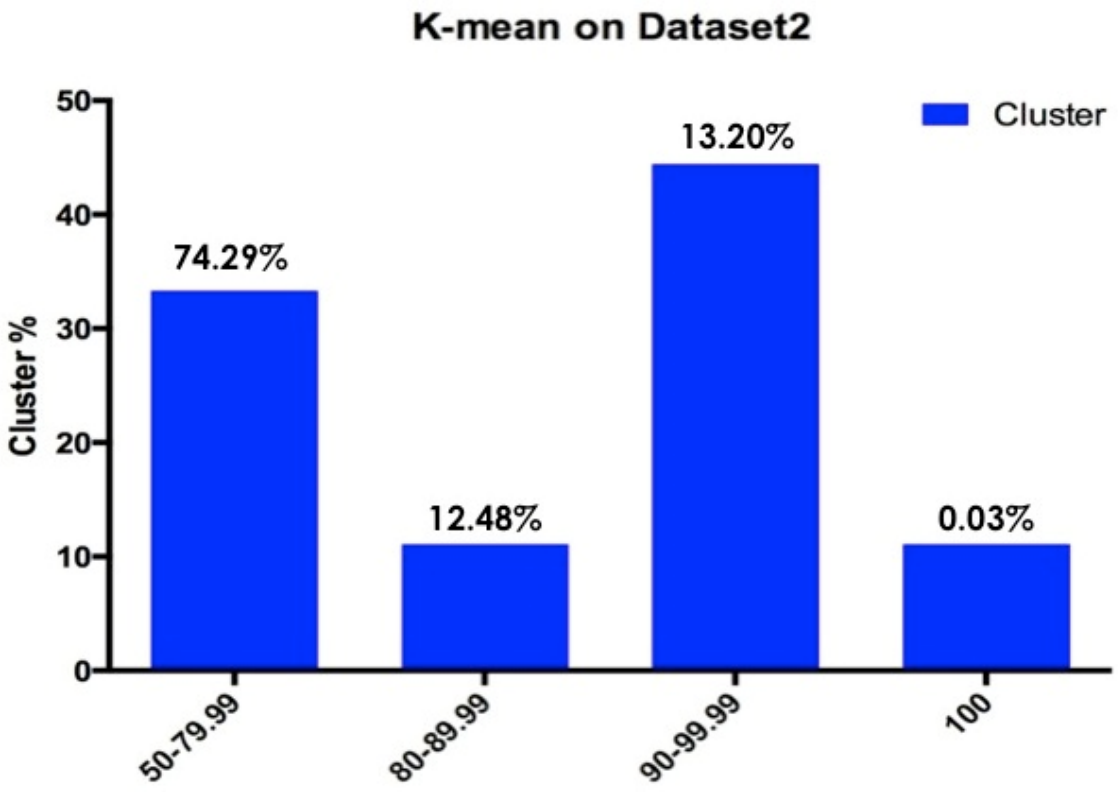

Figure 5.12 Experiment 5 Graph

Analysis: There were around 14,467 executables in this dataset. We apply K-mean algorithm and try to create 10 clusters out of which 9 have data and one empty cluster is created. After analyzing the cluster data mentioned in Table 5.5, we found that around 4829 executables were clustered wrongly. This gives accuracy of 66.62%, which is more than hierarchical clustering but not best.

For better understanding of analysis results we have created a graph shown in Figure 5.12, which gives gist of the clusters with their accuracy and number of executables belonging to each group. Around 45% of clusters have accuracy of 90-99.99% and only 13.20 % of executables from Dataset2 belongs to this group. Maximum of executables belongs to the lowest accuracy group and hence the K-mean algorithm has low accuracy.

Experiment 6:

Input: Dataset2

Algorithm: SOM algorithm

Output: This experiment is same as experiment 3 but the input dataset is different. Figure 5.13 gives the sample hits of the clusters and Table 5.6 gives cluster details.

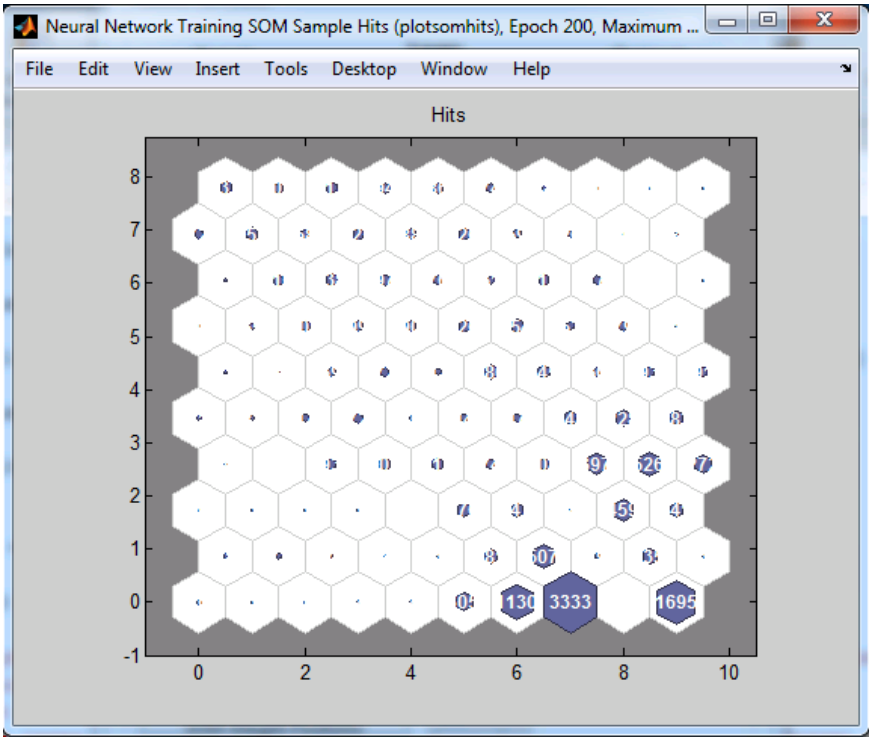

Figure 5.13 Experiment 6 SOM Hits

(Table 5.6 Cont.)

SOM on Datset 2											
Cluster	**Executable**	**Value**	**Percentage**	**Cluster**	**Executable**	**Value**	**Percentage**	**Cluster**	**Executable**	**Value**	**Percentage**
1	Benign	15	93.75%	**33**	Benign	40	54.05%	**65**	Benign	0	0.00%
	Malacious	1	6.25%		Malacious	34	45.95%		Malacious	53	100.00%
2	Benign	3	100.00%	**34**	Benign	26	24.07%	**66**	Benign	0	0.00%
	Malacious	0	0.00%		Malacious	82	75.93%		Malacious	79	100.00%
3	Benign	4	100.00%	**35**	Benign	0	0.00%	**67**	Benign	3	100.00%
	Malacious	0	0.00%		Malacious	397	100.00%		Malacious	0	0.00%
4	Benign	0	0.00%	**36**	Benign	12	2.28%	**68**	Benign	0	0.00%
	Malacious	9	100.00%		Malacious	514	97.72%		Malacious	12	100.00%
5	Benign	7	77.78%	**37**	Benign	0	0.00%	**69**	Benign	0	0.00%
	Malacious	2	22.22%		Malacious	272	100.00%		Malacious	110	100.00%
6	Benign	305	100.00%	**38**	Benign	1	9.09%	**70**	Benign	0	0.00%
	Malacious	0	0.00%		Malacious	10	90.91%		Malacious	138	100.00%
7	Benign	1129	99.91%	**39**	Benign	3	13.64%	**71**	Benign	0	0.00%
	Malacious	1	0.09%		Malacious	19	86.36%		Malacious	97	100.00%
8	Benign	3319	99.58%	**40**	Benign	27	57.45%	**72**	Benign	0	0.00%
	Malacious	14	0.42%		Malacious	20	42.55%		Malacious	76	100.00%

9	Benign	137	8.08%	**41**	Benign	0	0.00%	**73**	Benign	0	0.00%
	Malacious	1558	91.92%		Malacious	72	100.00%		Malacious	51	100.00%
10	Benign	16	88.89%	**42**	Benign	0	0.00%	**74**	Benign	0	0.00%
	Malacious	2	11.11%		Malacious	10	100.00%		Malacious	113	100.00%
11	Benign	23	100.00%	**43**	Benign	0	0.00%	**75**	Benign	1	1.30%
	Malacious	0	0.00%		Malacious	44	100.00%		Malacious	76	98.70%
12	Benign	7	100.00%	**44**	Benign	0	0.00%	**76**	Benign	6	100.00%
	Malacious	0	0.00%		Malacious	47	100.00%		Malacious	0	0.00%
13	Benign	0	0.00%	**45**	Benign	2	1.35%	**77**	Benign	3	4.17%
	Malacious	2	100.00%		Malacious	146	98.65%		Malacious	69	95.83%
14	Benign	3	100.00%	**46**	Benign	1	0.44%	**78**	Benign	0	0.00%
	Malacious	0	0.00%		Malacious	227	99.56%		Malacious	153	100.00%
15	Benign	181	100.00%	**47**	Benign	1	0.55%	**79**	Benign	0	0.00%
	Malacious	0	0.00%		Malacious	181	99.45%		Malacious	158	100.00%
16	Benign	507	100.00%	**48**	Benign	0	0.00%	**80**	Benign	0	0.00%
	Malacious	0	0.00%		Malacious	11	100.00%		Malacious	125	100.00%
17	Benign	19	100.00%	**49**	Benign	0	0.00%	**81**	Benign	0	0.00%
	Malacious	0	0.00%		Malacious	1	100.00%		Malacious	88	100.00%
18	Benign	116	49.36%	**50**	Benign	0	0.00%	**82**	Benign	0	0.00%
	Malacious	119	50.64%		Malacious	59	100.00%		Malacious	126	100.00%
19	Benign	3	100.00%	**51**	Benign	0	0.00%	**83**	Benign	0	0.00%
	Malacious	0	0.00%		Malacious	72	100.00%		Malacious	59	100.00%
20	Benign	3	100.00%	**52**	Benign	0	0.00%	**84**	Benign	1	3.85%
	Malacious	0	0.00%		Malacious	42	100.00%		Malacious	25	96.15%
21	Benign	4	80.00%	**53**	Benign	0	0.00%	**85**	Benign	1	100.00%
	Malacious	1	20.00%		Malacious	180	100.00%		Malacious	0	0.00%
22	Benign	3	100.00%	**54**	Benign	0	0.00%	**86**	Benign	6	75.00%
	Malacious	0	0.00%		Malacious	145	100.00%		Malacious	2	25.00%
23	Benign	0	0.00%	**55**	Benign	1	1.79%	**87**	Benign	0	0.00%
	Malacious	3	100.00%		Malacious	55	98.21%		Malacious	132	100.00%
24	Benign	151	85.80%	**56**	Benign	0	0.00%	**88**	Benign	0	0.00%
	Malacious	25	14.20%		Malacious	93	100.00%		Malacious	102	100.00%
25	Benign	146	99.32%	**57**	Benign	1	1.10%	**89**	Benign	0	0.00%
	Malacious	1	0.68%		Malacious	90	98.90%		Malacious	117	100.00%
26	Benign	1	50.00%	**58**	Benign	3	75.00%	**90**	Benign	0	0.00%
	Malacious	1	50.00%		Malacious	1	25.00%		Malacious	94	100.00%

27	Benign	1	0.22%	**59**	Benign	0	0.00%	**91**	Benign	0	0.00%
	Malacious	458	99.78%		Malacious	35	100.00%		Malacious	86	100.00%
28	Benign	44	31.21%	**60**	Benign	0	100.00%	**92**	Benign	0	0.00%
	Malacious	97	68.79%		Malacious	101	0.00%		Malacious	74	100.00%
29	Benign	4	100.00%	**61**	Benign	0	0.00%	**93**	Benign	0	0.00%
	Malacious	0	0.00%		Malacious	99	100.00%		Malacious	18	100.00%
30	Benign	44	47.31%	**62**	Benign	0	0.00%	**94**	Benign	3	75.00%
	Malacious	49	52.69%		Malacious	90	100.00%		Malacious	1	25.00%
31	Benign	68	66.02%	**63**	Benign	0	0.00%	**95**	Benign	4	80.00%
	Malacious	35	33.98%		Malacious	126	100.00%		Malacious	1	20.00%
32	Benign	66	58.41%	**64**	Benign	0	0.00%	**96**	Benign	3	100.00%
	Malacious	47	41.59%		Malacious	154	100.00%		Malacious	0	0.00%

Table 5.6 Experiment 6 Clusters

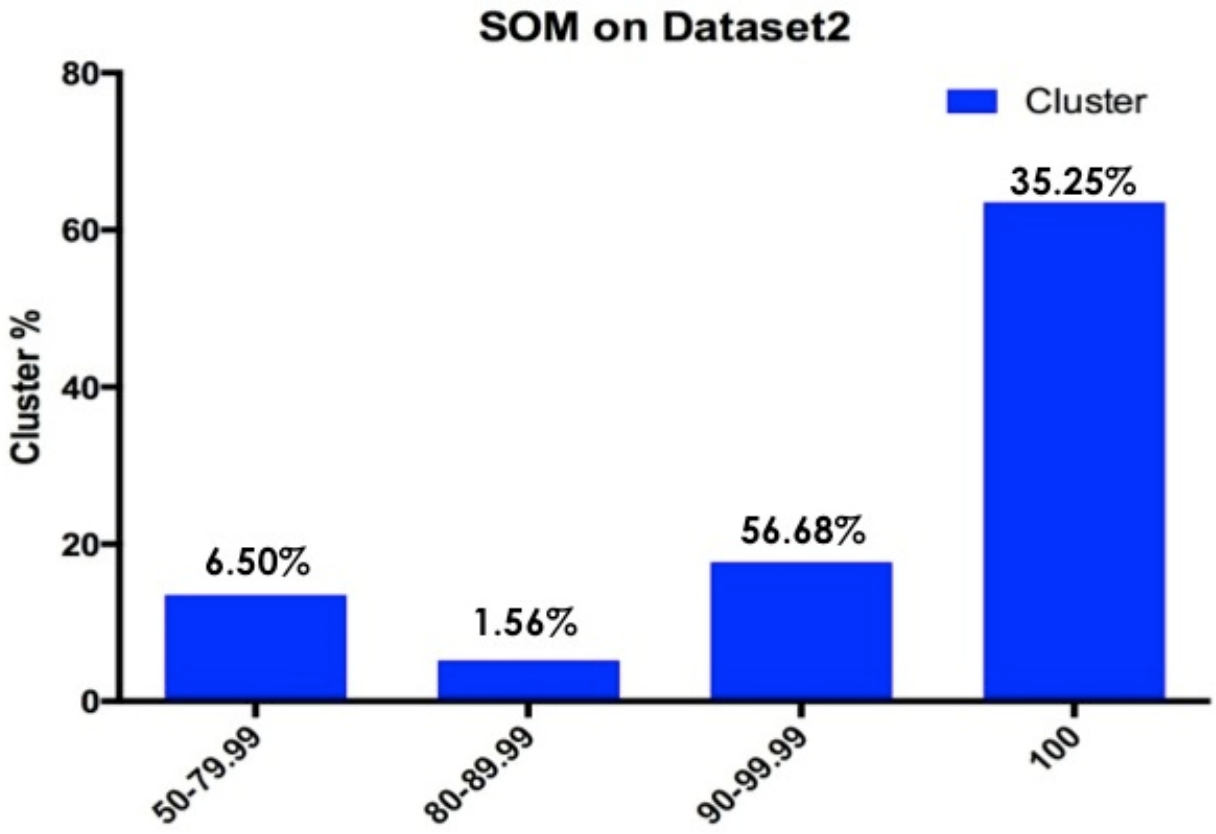

Figure 5.14 Experiment 6 Graph

Analysis: There were around 14,467 executables that is clustered using SOM algorithm. We asked algorithm to create maximum of 100 clusters but 96 clusters were created with data and 4 empty clusters were created as mentioned in Table 5.6. In this algorithm around 584 executables were wrongly classified. This results in accuracy of 95.96%, which is best among three algorithms on dataset2.

Figure 5.14 shows a graph with analysis result for our better understanding. Similar to experiment 3, this experiment also has around 92% of files from Dataset2 that belongs to either 90-99.99% or 100% group, which in turn has maximum number of clusters. Hence, like experiment 3, experiment 6 has highest accuracy for Dataset2.

Experiment 7:

Input: Dataset3

Algorithm: Hierarchical algorithm

Output: This experiment is same as experiment 1 and 4 just the input dataset is different. Figure 5.15 is dendogram of binary tree and Figure 5.16 represents the clusters.

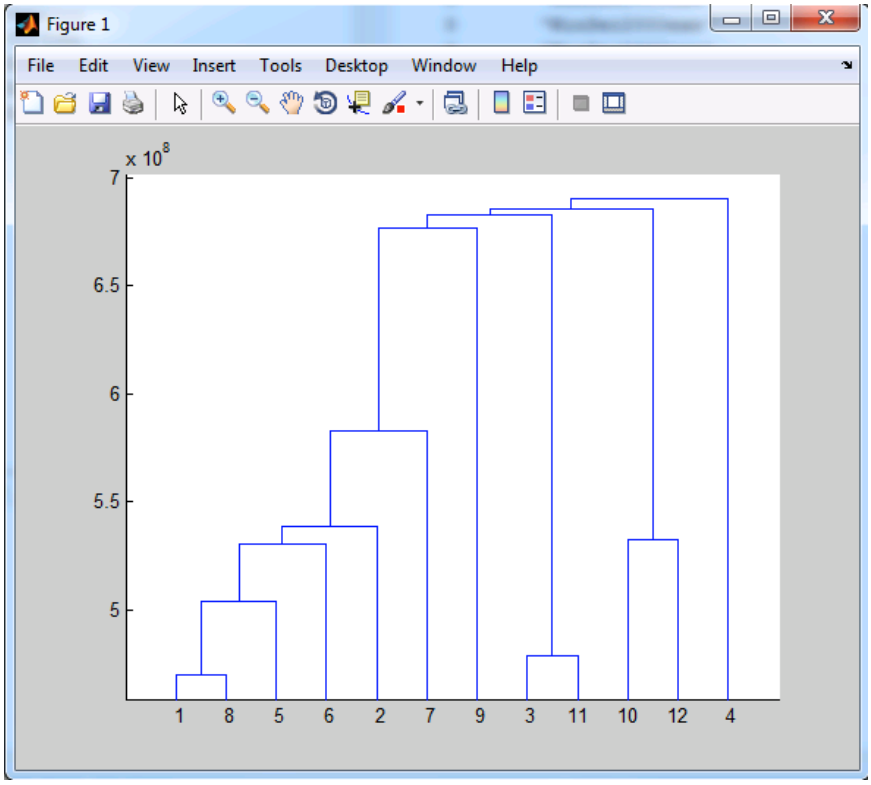

Figure 5.15 Experiment 7 Binary Tree

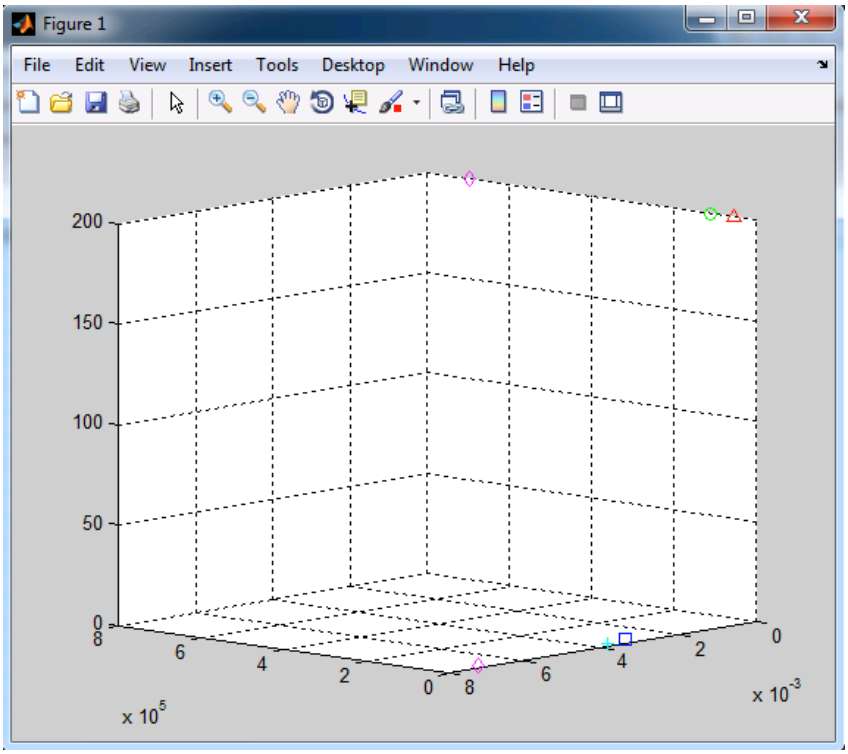

Figure 5.16 Experiment 7 Plot

Hierarchical on Datset 3											
Cluster	**Executable**	**Value**	**Percentage**	**Cluster**	**Executable**	**Value**	**Percentage**	**Cluster**	**Executable**	**Value**	**Percentage**
1	Benign	3	100.00%	**13**	Benign	0	0.00%	**25**	Benign	11	78.57%
	Malacious	0	0.00%		Malacious	1	100.00%		Malacious	3	21.43%
2	Benign	0	0.00%	**14**	Benign	0	0.00%	**26**	Benign	4	100.00%
	Malacious	1	100.00%		Malacious	1	100.00%		Malacious	0	0.00%
3	Benign	4	80.00%	**15**	Benign	0	0.00%	**27**	Benign	4	100.00%
	Malacious	1	20.00%		Malacious	2	100.00%		Malacious	0	0.00%
4	Benign	0	0.00%	**16**	Benign	2	100.00%	**28**	Benign	4	100.00%
	Malacious	1	100.00%		Malacious	0	0.00%		Malacious	0	0.00%
5	Benign	1	100.00%	**17**	Benign	0	0.00%	**29**	Benign	12	100.00%
	Malacious	0	0.00%		Malacious	1	100.00%		Malacious	0	0.00%
6	Benign	0	0.00%	**18**	Benign	4	100.00%	**30**	Benign	0	0.00%
	Malacious	4	100.00%		Malacious	0	0.00%		Malacious	1	100.00%
7	Benign	11	100.00%	**19**	Benign	1	100.00%	**31**	Benign	0	0.00%
	Malacious	0	0.00%		Malacious	0	0.00%		Malacious	2	100.00%
8	Benign	4	100.00%	**20**	Benign	1	100.00%	**32**	Benign	4	100.00%
	Malacious	0	0.00%		Malacious	0	0.00%		Malacious	0	0.00%
9	Benign	47	97.92%	**21**	Benign	0	0.00%	**33**	Benign	0	0.00%
	Malacious	1	2.08%		Malacious	4	100.00%		Malacious	4	100.00%
10	Benign	4	100.00%	**22**	Benign	4	100.00%	**34**	Benign	7	100.00%
	Malacious	0	0.00%		Malacious	0	0.00%		Malacious	0	0.00%
11	Benign	0	0.00%	**23**	Benign	5555	47.10%	**35**	Benign	0	0.00%
	Malacious	1	100.00%		Malacious	6239	52.90%		Malacious	1	100.00%
12	Benign	0	0.00%	**24**	Benign	0	0.00%				
	Malacious	2	100.00%		Malacious	3	100.00%				

Table 5.7 Experiment 7 Clusters

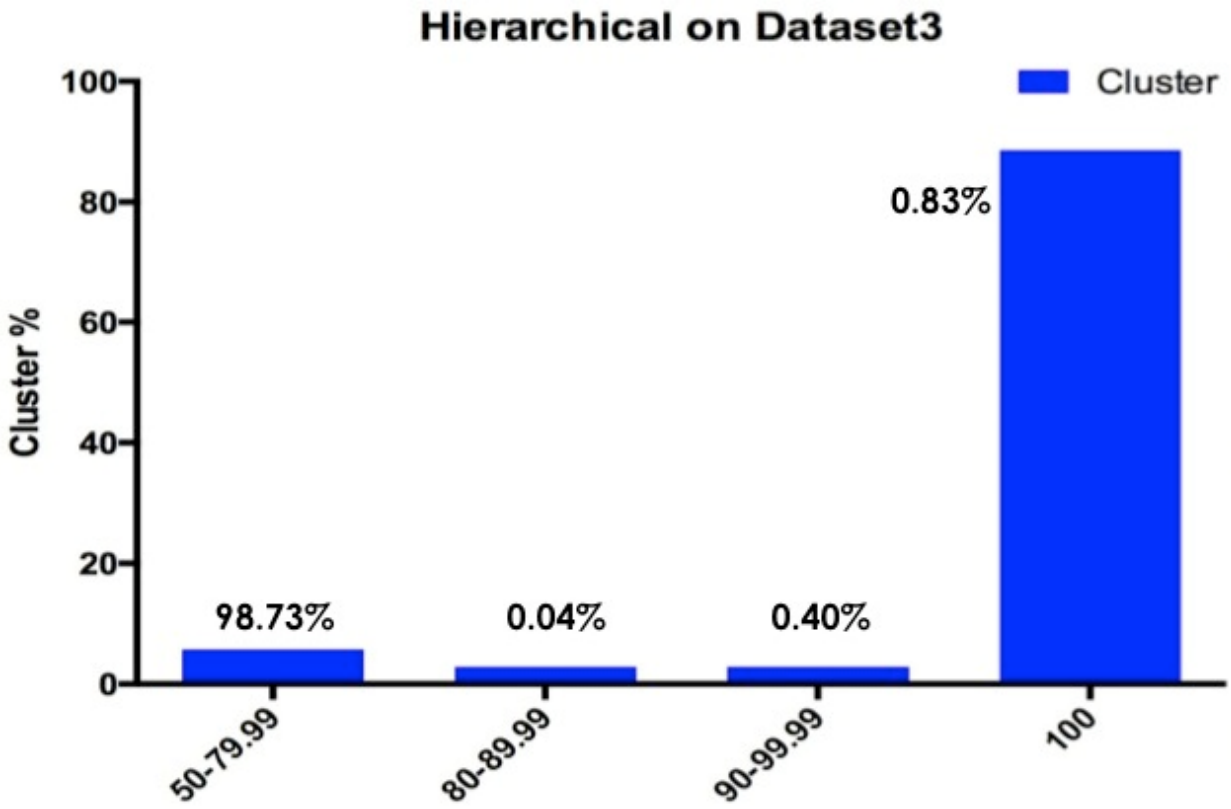

Figure 5.17 Experiment 7 Graph

Analysis: There were around 11,960 executables in this dataset, which were clustered using Hierarchical algorithm with distance cutoff. After applying algorithm around 35 clusters were created as mentioned in Table 5.7. From this table we can infer that out of 11,960 around 5,560 executables were wrongly clustered. Thus accuracy of only 53.51% is received for this dataset.

Analysis of clusters created by applying Hierarchical algorithm on Dataset3 is shown in Figure 5.17, in the form of a graph. This graph shows groups of clusters with the accuracy as mentioned on X-axis and has a label with the percentage of files belonging to each group. As we can see from the graph more clusters were created with 100% accuracy but it covers only 0.83% of the executables from the Dataset3. Most of the files from the Dataset3 have lowest

accuracy and only few numbers of clusters were created. Thus from this result we can infer that Hierarchical algorithm will not help us to create clusters with highest accuracy.

Experiment 8:

Input: Dataset3

Algorithm: K-mean algorithm

Output: Experiment 8 is same as experiment 2 & 5 just the input dataset is different. Figure 5.18 represent centroid of clusters and Table 5.8 gives the number of malware and benign in each cluster.

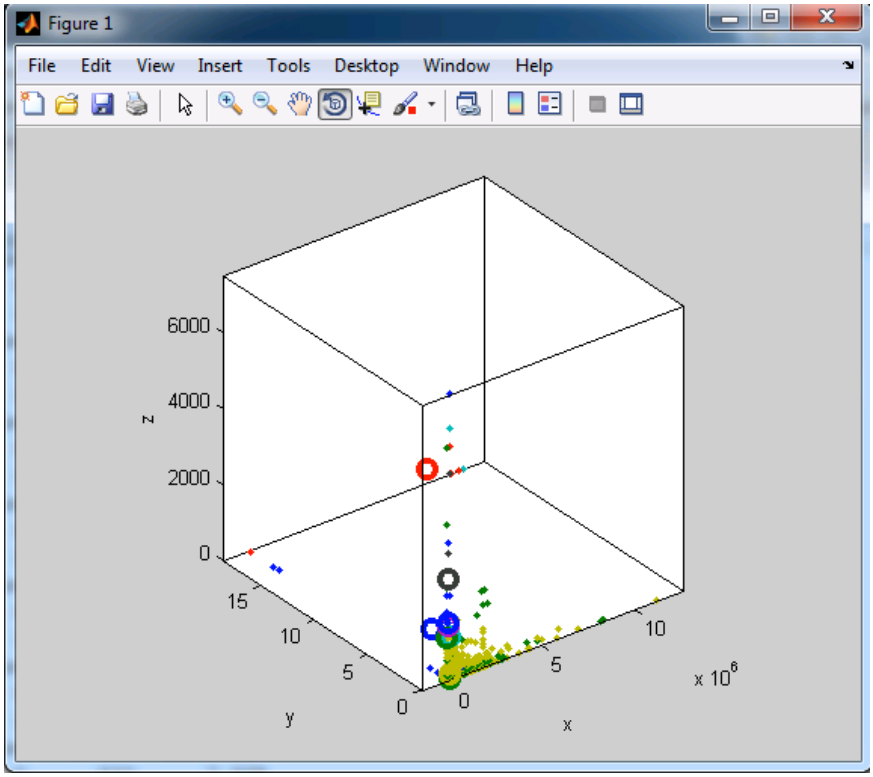

Figure 5.18 Experiment 8 Plot

Kmean on Datset 3							
Cluster	**Executable**	**Value**	**Percentage**	**Cluster**	**Executable**	**Value**	**Percentage**
1	Benign	31	86.11%	**6**	Benign	522	7.83%
	Malacious	5	13.89%		Malacious	6143	92.17%
2	Benign	5045	98.08%	**7**	Benign	5	71.43%
	Malacious	99	1.92%		Malacious	2	28.57%
3	Benign	5	62.50%	**8**	Benign	4	36.36%
	Malacious	3	37.50%		Malacious	7	63.64%
4	Benign	15	65.22%	**9**	Benign	43	93.48%
	Malacious	8	34.78%		Malacious	3	6.52%
5	Benign	15	75.00%	**10**	Benign	0	0.00%
	Malacious	5	25.00%		Malacious	0	0.00%

Table 5.8 Experiment 8 Clusters

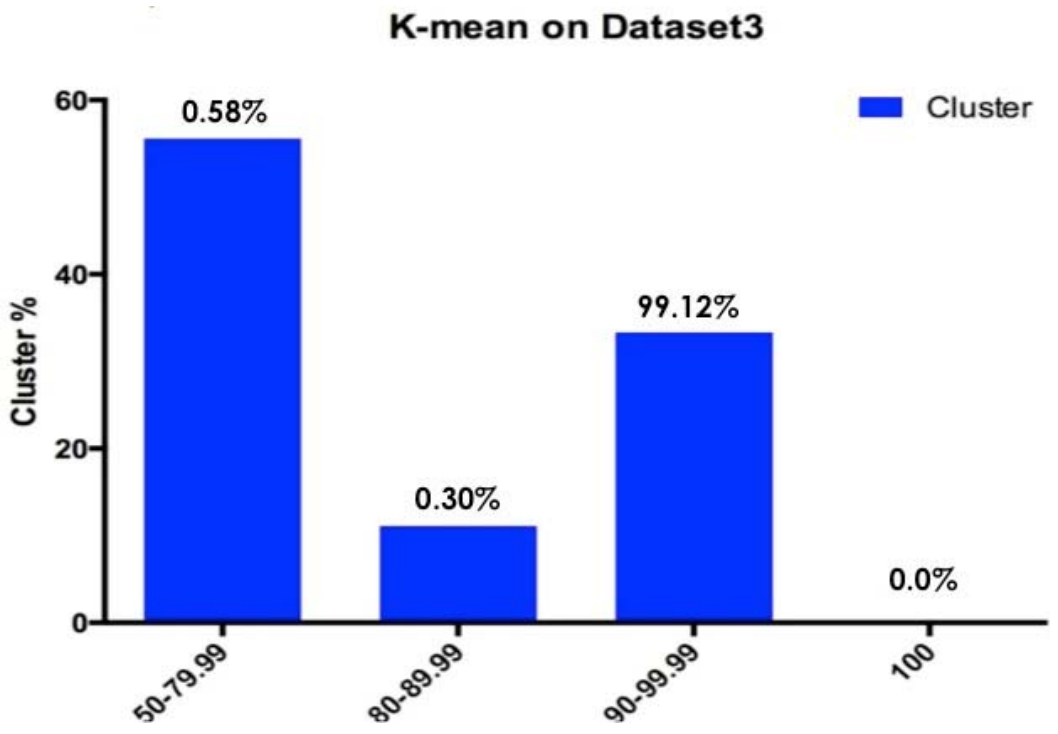

Figure 5.19 Experiment 8 Graph

Analysis: There were 11,960 executables that were clustered using K-mean algorithm. The algorithm is asked to create 10 clusters but 9 clusters were created with data and 1 empty cluster is created. We can infer from Table 5.8 that around 651 executables were clustered incorrectly. As a result we get accuracy of 94.56%, which is significantly more than Hierarchical algorithm applied on same dataset.

From Figure 5.19 we observed that less than 40% of clusters have accuracy of 90-99.99% and this group contains about 99.12% of executables from Dataset3. The highest bar in the graph is for lowest accuracy, which means most of the clusters have lowest accuracy but that covers only 0.58% of executables from Dataset3. This experiment has good accuracy compared to Experiment 7.

Experiment 9:

Input: Dataset3

Algorithm: SOM algorithm

Output: Experiment 9 is same as experiment 3 and 6 just the input dataset is different. Figure 5.20 gives the sample hits that represent number of elements or data in each cluster.

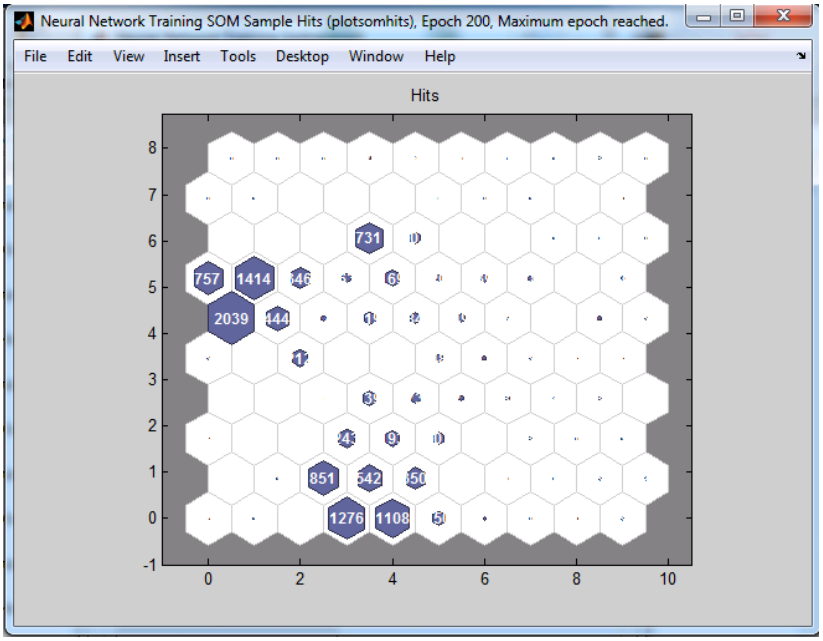

Figure 5.20 Experiment 9 SOM Hits

(Table 5.9 Cont.)

SOM on Datset 3											
Cluster	**Executable**	**Value**	**Percentage**	**Cluster**	**Executable**	**Value**	**Percentage**	**Cluster**	**Executable**	**Value**	**Percentage**
1	Benign	0	0.00%	**27**	Benign	73	100.00%	**53**	Benign	0	0.00%
	Malacious	2	100.00%		Malacious	0	0.00%		Malacious	20	100.00%
2	Benign	5	100.00%	**28**	Benign	19	86.36%	**54**	Benign	0	0.00%
	Malacious	0	0.00%		Malacious	3	13.64%		Malacious	39	100.00%
3	Benign	1276	100.00%	**29**	Benign	7	87.50%	**55**	Benign	0	0.00%
	Malacious	0	0.00%		Malacious	1	12.50%		Malacious	18	100.00%
4	Benign	1108	100.00%	**30**	Benign	0	0.00%	**56**	Benign	16	100.00%
	Malacious	0	0.00%		Malacious	2	100.00%		Malacious	0	0.00%
5	Benign	66	44.00%	**31**	Benign	8	100.00%	**57**	Benign	1	0.14%
	Malacious	84	56.00%		Malacious	0	0.00%		Malacious	730	99.86%
6	Benign	8	72.73%	**32**	Benign	4	80.00%	**58**	Benign	3	2.91%
	Malacious	3	27.27%		Malacious	1	20.00%		Malacious	100	97.09%
7	Benign	0	0.00%	**33**	Benign	0	0.00%	**59**	Benign	0	0.00%
	Malacious	4	100.00%		Malacious	212	100.00%		Malacious	3	100.00%
8	Benign	0	0.00%	**34**	Benign	48	100.00%	**60**	Benign	4	100.00%
	Malacious	1	100.00%		Malacious	0	0.00%		Malacious	0	0.00%
9	Benign	8	100.00%	**35**	Benign	10	83.33%	**61**	Benign	4	21.43%
	Malacious	0	0.00%		Malacious	2	16.67%		Malacious	0	78.57%

10	Benign	3	100.00%	**36**	Benign	5	100.00%	**62**	Benign	4	100.00%
	Malacious	0	0.00%		Malacious	0	0.00%		Malacious	0	0.00%
11	Benign	851	100.00%	**37**	Benign	0	0.00%	**63**	Benign	4	66.67%
	Malacious	0	0.00%		Malacious	1	100.00%		Malacious	2	33.33%
12	Benign	542	100.00%	**38**	Benign	0	0.00%	**64**	Benign	1	100.00%
	Malacious	0	0.00%		Malacious	1	100.00%		Malacious	0	0.00%
13	Benign	349	99.71%	**39**	Benign	42	2.06%	**65**	Benign	4	100.00%
	Malacious	1	0.29%		Malacious	1997	97.94%		Malacious	0	0.00%
14	Benign	4	100.00%	**40**	Benign	80	18.02%	**66**	Benign	5	100.00%
	Malacious	0	0.00%		Malacious	364	81.98%		Malacious	0	0.00%
15	Benign	4	100.00%	**41**	Benign	6	35.29%	**67**	Benign	1	100.00%
	Malacious	0	0.00%		Malacious	11	64.71%		Malacious	0	0.00%
16	Benign	8	100.00%	**42**	Benign	44	36.97%	**68**	Benign	0	0.00%
	Malacious	0	0.00%		Malacious	75	63.03%		Malacious	4	100.00%
17	Benign	7	0.00%	**43**	Benign	41	48.81%	**69**	Benign	0	0.00%
	Malacious	2	100.00%		Malacious	43	51.19%		Malacious	4	100.00%
18	Benign	1	100.00%	**44**	Benign	16	32.65%	**70**	Benign	0	0.00%
	Malacious	0	0.00%		Malacious	33	67.35%		Malacious	4	100.00%
19	Benign	242	99.59%	**45**	Benign	0	0.00%	**71**	Benign	7	100.00%
	Malacious	1	0.41%		Malacious	3	100.00%		Malacious	0	0.00%
20	Benign	191	98.96%	**46**	Benign	11	91.67%	**72**	Benign	0	0.00%
	Malacious	2	1.04%		Malacious	1	8.33%		Malacious	2	100.00%
21	Benign	107	100.00%	**47**	Benign	4	80.00%	**73**	Benign	0	0.00%
	Malacious	0	0.00%		Malacious	1	20.00%		Malacious	4	100.00%
22	Benign	4	50.00%	**48**	Benign	3	0.40%	**74**	Benign	5	100.00%
	Malacious	4	50.00%		Malacious	754	99.60%		Malacious	0	0.00%
23	Benign	4	100.00%	**49**	Benign	172	12.16%	**75**	Benign	3	100.00%
	Malacious	0	0.00%		Malacious	1242	87.84%		Malacious	0	0.00%
24	Benign	4	66.67%	**50**	Benign	81	23.41%	**76**	Benign	8	100.00%
	Malacious	2	33.33%		Malacious	265	76.59%		Malacious	0	0.00%
25	Benign	1	100.00%	**51**	Benign	0	0.00%	**77**	Benign	4	100.00%
	Malacious	0	0.00%		Malacious	63	100.00%		Malacious	0	0.00%
26	Benign	139	100.00%	**52**	Benign	5	2.96%				
	Malacious	0	0.00%		Malacious	164	97.04%				

Table 5.9 Experiment 9 Clusters

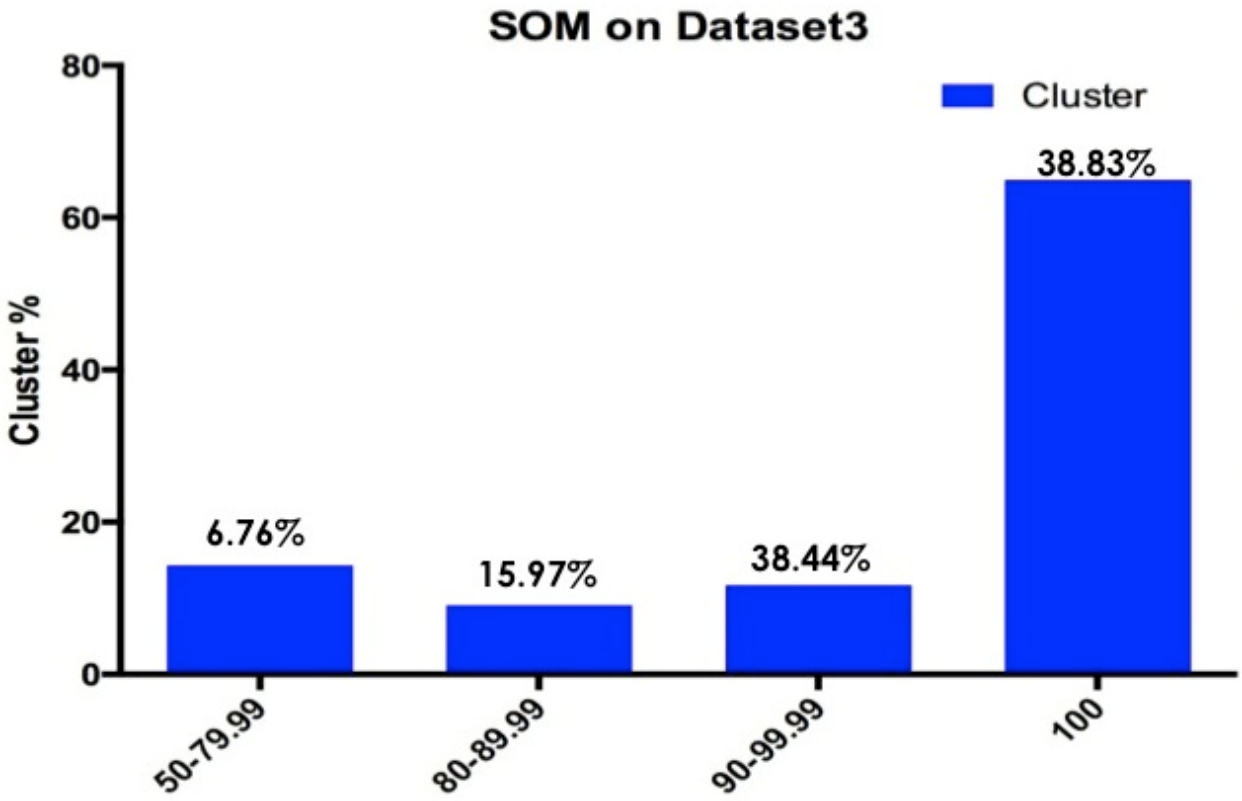

Figure 5.21 Experiment 9 Graph

Analysis: There were 11,960 executables in dataset that were clustered using SOM algorithm. The algorithm is asked to create maximum of 100 clusters out of which 77 clusters with data and 33 empty clusters were created. After analyzing the data mentioned in Table 5.9, we found that 586 executables were clustered incorrectly which leads to accuracy of 95.10%. Thus, SOM has highest accuracy among three algorithms.

Moreover, to understand the analysis mentioned in Table 5.9 in an efficient way, we have created graph in Figure 5.21. Around 77% of executables from Dataset3 has 90 to 100% accuracy and

more than 60% of clusters have 100% of accuracy. Thus SOM algorithm has highest accuracy on Dataset3.

5.3 Discussion

The following Table 5.10, Figure 5.22 and Figure 5.23 show the summary of experiments results.

	Dataset1	**Dataset2**	**Dataset3**
Number of Files in dataset (NFD)	**22172**	**14467**	**11960**
Hierarchical Falsely Clustered (HFC)	10702	6379	5560
Kmean Falsely Clustered (KFC)	515	4829	651
SOM Falsely Clustered (SFC)	451	584	586

Table 5.10 Results Summary

Dataset	**Algorithm**	**50-79.99**		**80-89.99**		**90-99.99**		**100**	
		Cluster %	Files %	Cluster %	Files %	Cluster %	Files %	Cluster %	Files %
Dataset1	**Hierarchical**	17.86	98.54	3.57	0.03	3.57	0.15	75.00	1.28
	K-mean	50.00	1.16	0.00	0.00	20.00	97.99	30.00	0.84
	SOM	5.88	0.24	3.53	2.29	**22.35**	**72.90**	**68.24**	**24.58**
Dataset2	**Hierarchical**	3.85	99.16	3.85	0.06	0.00	0.00	92.31	0.78
	K-mean	33.33	74.29	11.11	12.48	44.44	13.20	11.11	0.03
	SOM	13.54	6.50	5.21	1.56	**17.71**	**56.68**	**63.54**	**35.25**
Dataset3	**Hierarchical**	5.71	98.73	2.86	0.04	2.86	0.40	88.57	0.83
	K-mean	55.56	0.58	11.11	0.30	33.33	99.12	0.00	0.00
	SOM	15.58	6.76	9.09	15.97	**11.69**	**38.44**	**63.64**	**38.83**

Table 5.11 Results Analysis

Table 5.10 is represented in graphical form in Figure 5.22 for better understanding and comparison. First set of three bars compares the number of executables in each datasets, second set of bars compares number of executables that were incorrectly clustered using Hierarchical

algorithm, third set of bars compares number of executables falsely clustered using K-mean algorithm on three datasets and fourth set of bars compares falsely clustered executables using SOM algorithm on all three datasets.

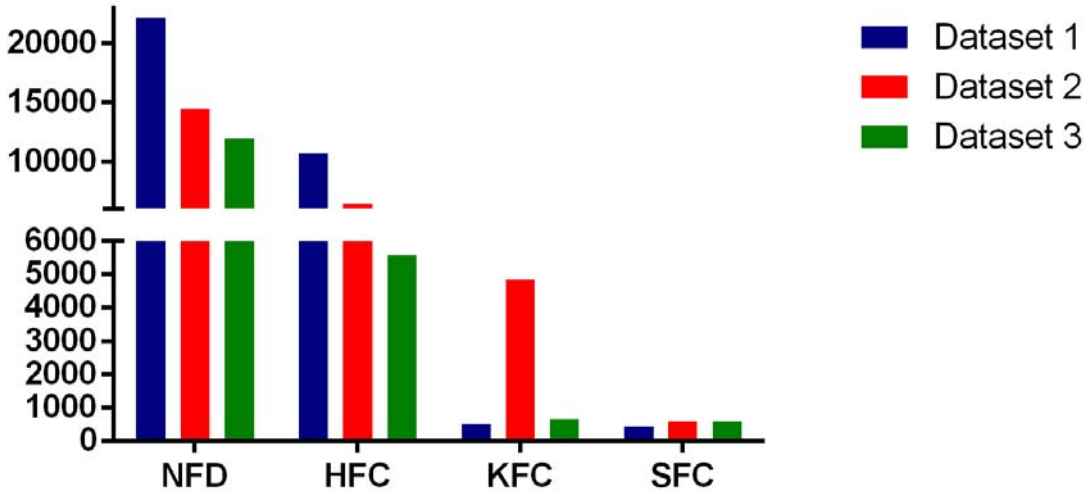

Figure 5.22

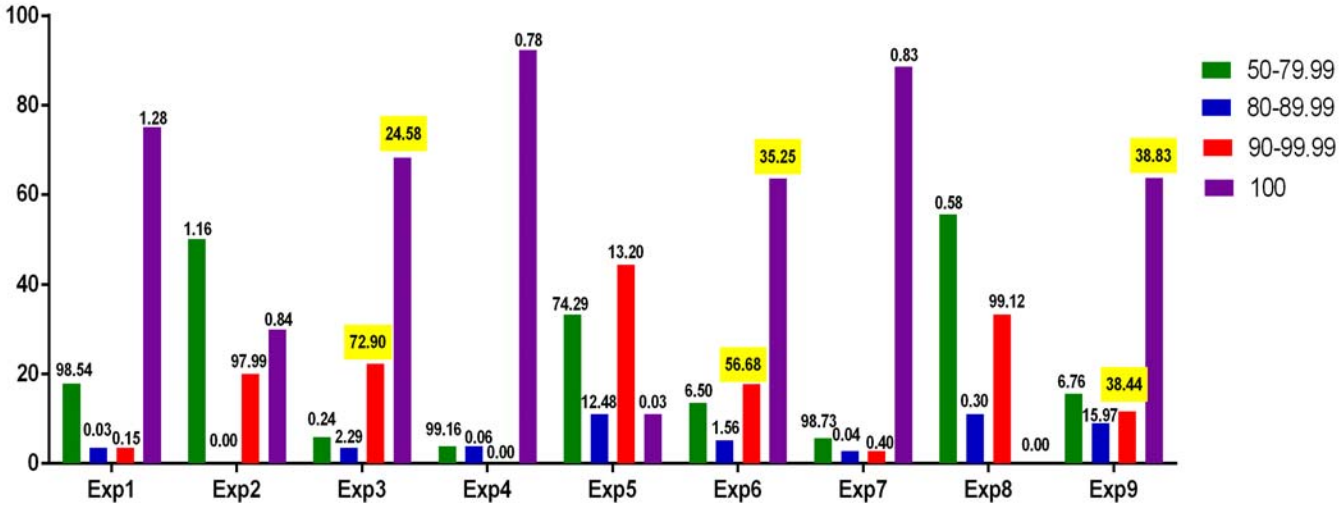

Figure 5.23

The comparison of results obtained from Hierarchical, K-mean and SOM algorithm on each dataset is represented in Figure 5.23. The X-axis represents the results obtained from all nine

experiments. Each experiment has four bars, each representing percentage of cluster having accuracy in the ranges mentioned in graph. Each range is labeled with the percentage of files belonging to that range from the dataset. For example, the first bar from experiment 1 with color green represents the percentage of cluster belonging to 50-79.99% accuracy and the label 98.54% says the percentage of executables that belongs to this clusters from the complete dataset. The highlighted text covers most of the executables from the dataset, which belongs in 90 to 100 range. Thus from the graph we know that SOM algorithm has best accuracy.

CHAPTER 6: CONCLUSION AND FUTURE STUDY

6.1 Conclusion

Several experiments have been conducted on various datasets out of which some are mentioned in experiment results section. Different features have been collected and various clustering algorithm were used to cluster the executables. We finally found some interesting features among others like number of unknown sections, number of dll function calls, number of directories, size of executables, and an algorithm with highest accuracy, efficiency and lowest falsely clustered rate.

After all the analysis and experiments among unsupervised learning algorithm like hierarchical, K-mean and SOM, we found 13 interesting features from PE headers and SOM clustering algorithm as best as it gives highest accuracy, efficiency and low falsely clustered rate while clustering any dataset. Thus after clustering we need to check one or two executables and say that the cluster either contains all the benign or all malicious executables.

6.2 Future Study

Future studies will address the four aspects in which the accuracy of clustering can be increased. The first aspect is to collect more static features, second aspect is to conduct dynamic analysis and collect some more useful feature, thirdly to implement some more unsupervised machine learning algorithm and lastly try further clustering malicious clusters into their types.

More static features can be collected from code section and other sections of PE file format,

which will be useful in clustering. Opcodes as predictor of malware [33] can be implemented that will collect frequency of opcode. The frequency of string or any other string or Text related features could be used as mentioned in Text based search [34].

As per [35], static analysis alone will not help in successful clustering or detection of malware. Thus adding some important and useful features from dynamic analysis can increase accuracy and efficiency of clustering. AMAL [36] like using API call sequence if combined with this project can also increase efficiency.

Different unsupervised machine learning algorithm can be implemented that helps in increasing accuracy of clustering. Clustering with Gaussian Mixture Models [37] and Hidden markov models [38] can be easily used as they are already implemented in Matlab. Results from this algorithm can be compared with already implement algorithms.

Further, clustering executables first in benign and malicious and then further create clusters of malicious executables according to their types like Trojan, spyware, backdoor, rootkit, virus, worm and more. Targeting the features that makes them different from one another can help to cluster malware according to their types.

6.3 Summary

This chapter concludes the thesis with discussion, analysis of experiments, and future extension of the project. Although some addition and improvements will help in increasing accuracy but this project has successfully gathered useful static features, which helps in clustering executables, efficiently and accurately using unsupervised learning algorithms.

REFERENCES

[1] http://en.wikipedia.org/wiki/Malware

[2] http://www.sans.org/course/reverse-engineering-malware-malware-analysis-tools-techniques

[3] http://msdn.microsoft.com/en-us/magazine/cc301805.aspx

[4] http://www.csn.ul.ie/~caolan/publink/winresdump/winresdump/doc/pefile2.html

[5] http://msdn.microsoft.com/en-us/library/ms809762.aspx

[6] http://www.ntcore.com/files/inject2exe.htm#TheWindowsNTdata2_2

[7] Yibin Liao. PE-Header-Based Malware Study and Detection

[8] Scott Treadwell, Mian Zhou. A Heuristic Approach for Detection of Obfuscated Malware

[9] M Christodorescu, S Jha. Static Analysis of Executables to Detect Malicious Patterns

[10] P Comar, L Liu, S Saha, P Tan, A Nucci. Combining Supervised and Unsupervised Learning for Zero-Day Malware Detection

[11] M Alazab, R Layton, S Venkataraman, P Watters. Malware Detection Based on Structural and Behavioural Features of API Calls

[12] Kazuki Iwamoto, K atsumi Wasaki . Malware Classification based on Extracted API Sequences using Static Analysis

[13] M Aljamea, V Ghanaei, C Iliopoulos, R Overill. Static Analysis and Clustering of Malware Applying Text Based Search

[14] X Hu, S Bhatkar, K Griffin, K Shin. MutantX-S: Scalable Malware Clustering Based on Static Features

[15] I Santos, J Devesa, F Brezo, J Nieves, and P Bringas. OPEM: A Static-Dynamic Approach for Machine-learning-based Malware Detection

[16] R Perdisci, A Lanzi, W Lee. McBoost: Boosting Scalability in Malware Collection and Analysis Using Statistical Classification of Executables

[17] M Shafiq, S Tabish, F Mirza, M Farooq. A Framework for Efficient Mining of Structural Information to Detect Zero-Day Malicious Portable Executables

[18] Juha Vesanto and Esa Alhoniemi, Student Member, IEEE. Clustering of the Self-Organizing Map

[19] http://en.wikipedia.org/wiki/Data_mining

[20] http://en.wikipedia.org/wiki/Supervised_learning

[21] http://en.wikipedia.org/wiki/Unsupervised_learning

[22] http://vxheaven.org/vl.php

[23] http://en.wikipedia.org/wiki/Cluster_analysis

[24] Pavel Berkhin. Survey of clustering data mining techniques. Technical report, Accrue Software, San Jose, CA, 2002.

[25] http://www.mathworks.com/help/stats/hierarchical-clustering.html

[26] http://en.wikipedia.org/wiki/K-means_clustering

[27] http://www.mathworks.com/help/stats/k-means-clustering.html#bq_679x-19

[28] http://en.wikipedia.org/wiki/Self-organizing_map

[29] http://www.mathworks.com/help/nnet/ug/cluster-with-self-organizing-map-neural-network.html

[30] http://www.visualstudio.com

[31] http://www.mathworks.com/products/matlab/

[32] http://www.vmware.com/products/workstation/

[33] Daniel Bilar. Opcodes as predictor for malware

[34] M Aljamea, V Ghanaei, C Iliopoulos, R Overill. Static Analysis and Clustering of Malware Applying Text Based Search

[35] A Moser, C Kruegel, and E Kirda. Limits of Static Analysis for Malware Detection

[36] A Mohaisen, O Alrawi. AMAL: High-Fidelity, Behavior-based Automated Malware Analysis and Classification

[37] http://www.mathworks.com/help/stats/gaussian-mixture-models.html

[38] http://www.mathworks.com/help/stats/hidden-markov-models-hmm.html

VITA

The author was born in Ahmedabad, Gujarat, India. She obtained her Bachelor's degree in Computer Engineering from Gujarat University in 2010. She then gained 2 years of field experience in India where she was part of research software development and network engineering team, with a goal to produce high quality projects and virtual private network (VPN) with security for the company.

To utilize her education and work experience towards earning Masters and in order to advance her skills and knowledge in the field of Computer Science, she joined University of New Orleans (UNO) in Fall 2012. Throughout the Masters program she had graduate assistantship where she worked on various web applications. Courses such as computer forensics and network security kindled her interest for the field of Information Assurance (IA). To pursue her interest and perform her thesis research in the field of IA, she joined Dr. Irfan Ahmed's (her thesis advisor) research team that works on malware, computer security and digital forensics at UNO.