AUTOMATED MALWARE ANALYSIS TO IDENTIFY DATA ESPIONAGE AND BACKDOOR CREATION

MCS

by

Azhar Javed

A thesis submitted to the faculty of Information Security Department Military College of Signals, National University of Sciences and Technology, Rawalpindi in partial fulfillment of the requirements for the degree of MS in Information Security

November 2014

ABSTRACT

In the recent past, malwares have become a serious cyber security threat which has not only targeted individuals and organizations but has also threatened the cyber space of countries around the world. Amongst malware variants, Trojans designed for data espionage and backdoor creation dominates the threat landscape. This necessitate in depth study of these malwares with the scope of extracting static features like APIs, strings, IP Addresses, URLs, email addresses etc. by and large found in such mal codes.

In this dissertation, an endeavored has been made to firstly establish a set of patterns, tagged as APIs and Strings persistently existent in these malwares by articulating an analysis framework. Presence of features in malware and benign dataset was checked and after assigning the weight to each feature, score is calculated. Later on using the percentile approach, threshold value for both (API and Mal String) feature set is determined. Secondly, keeping the feature set and threshold value as parameters, a methodology is proposed to automatically analyse the malwares designed for data espionage and backdoor creation.

The proposed methodology was tested by using a separate dataset of malware and benign application and based on common performance attributes; it was compared with previous work in the relevant field.

ACKNOWLEDGEMENTS

All praises to Allah Almighty, the most gracious and the most merciful, who blessed me with the opportunity, courage and determination to accomplish this humble contribution towards knowledge.

My sincere and heartfelt thanks to Lt Col Dr. Baber Aslam, Head of Information Security Department, for helping me in every aspect to undertake this research. My special thanks to Lt Col Dr. Monis Akhlaq, whom guidance and assistance made it possible for me to accomplish this desertion. I am also grateful to my respected committee members Lt Col Dr. Imran Rashid and Lecturer Waleed Bin Shahid for their unconditional support and directions.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
CLI	Command Line Interface
CVE	Common Vulnerabilities Exposures
DDoS	Distributed Denial of Service
DLL	Dynamic Link Library
DNS	Domain Name Service
DoS	Denial of Service
FN	False Negative
FP	False Positive
GUI	Graphical User Interface
IAT	Import Address Table
Mal	Malicious
OEP	Original Entry Point
OS	Operating System
PE	Portable Executable
ROL	Rotate Left
ROT	Rotate
TN	True Negative
TP	True Positive
UPX	Ultimate Packer for Executables
URL	Universal Resource Locator
XOR	Exclusive OR

Chapter 1

INTRODUCTION

1.1 Chapter Overview

This introductory chapter unfolds the most researched domain of malware analysis, starting from malware taxonomy to their analysis technologies. It will also explore various tools used in malware analysis. In the later portion of this introductory chapter, statement of research problem, research fundamental objectives and author's contribution are endorsed. How this dissertation is being organized is reflected as last part of this chapter.

1.2 Background

The rapid advancements in the field of information technology has revolutionized our day to day life and now it has become a vital component of every organization, ranging from public to private, small to large and profit to non-profit organizations. Most of the perceptible and challenging threats ever faced by these organizations are caused by malware (a mal intended piece of software code) characterized by numerous facets of threats like DOS, key logging, backdoors, botnets, phishing and pharming attacks. These malwares are not only limited to business organization but has also emerged as a cyber-threat for countries around the world.

The massive threat caused by these malware necessitates a concrete counter measure. For this purpose we need to analyze the capabilities of the understudy malware by employing a thorough malware analysis. By and large malware analysis involves a two-step approach, namely behavior analysis and code analysis (reverse engineering). Lot of commercial and open source tools are available for analyzing malware in an arbitrary manner each working independently and some are even having common elements of functionality as well. While processing through these tools analyzing a piece of mal intended code is often difficult and time-consuming as it involves recording the behavioral aspects as well as it implicates sifting through assembly code to search for valuable strings and / or understanding the code to uncover the complete picture of mal-intends.

This research will make an endeavor to conduct reverse engineering of malware designed for data espionage and backdoor creation purpose. Based on the important artifacts / patterns established during detailed study of such types of malwares a framework will be proposed which will not only be time efficient but will also be consistent in terms of extraction of artifacts in the shape of report it generates.

1.3 Malware Taxonomy

Malware “a mal intended piece of software code” or malicious software is a collective term used for all kinds of threats including viruses worms and trojans. Out of these, trojans dominates the threat backdrop. There is a general mis-conception about malwares and often it is considered that Trojans, worms and viruses are interchangeable terms but in fact, each one refers to a separate class of malware[1].

1.3.1 Virus

is a buzz word, known to all since the inception of computers in our lives. They are transmitted from files to files and needs human intervention for their propagation. It hides itself within a code and runs as and when the actual file is executed. Viruses are delivered in a variety of ways for example via email attachment or thumb drive etc.

1.3.2 Worm

Internet Worms are a kind of malware that copies itself and relies on a network infrastructure for its further transmission and delivery. Few worms like zombie create a backdoor for the attacker. Email attachments and free download sites are considered as the best sources of infection for this malware variant.

1.3.3 Trojans

Trojans are a kind of malwares which always presents itself as benign software, most essentially required by the intended victim. But in fact they have hidden objectives to achieve in the garb of this apparent useful functionality. They are often circulated with cracked software utilities either downloaded by victim intentionally or unintentionally through drive-by download. Often Trojan payload is equipped with backdoor creation, data espionage, key logging and data stealing mal intended functionalities. Common types are discussed is the succeeding paragraphs.

1.3.3.1 Backdoors

Backdoor permits an unauthorized entity to take full control of a victim's system without his/her consent. Backdoor Trojan always presents itself as a legitimate software tool very essentially required by the user. Other delivery option includes hitting a malicious website or clicking a link in spam email. Upon execution, it adds itself into a startup routine of system and looks for an internet connection. Once system goes online it connects the system with its author who then takes over the system to perform different task includes but not limited to download / upload of files, key logging, sending spam emails or stealing passwords etc.

1.3.3.2 Data Espionage

It involves a deliberate collection of data from a computer without the permission of the owner of the information. Trojans are often used to access the system in order to undertake data espionage. They come as a payload of Trojans in shape of sniffers, password hash grabbers and key loggers to perform data espionage operations.

1.3.3.3 Botnet

Botnet is set of infected systems which are under control of a remote hacker over the internet. The individual systems within the botnet are termed as zombies. All zombies within the same botnet receive the same instructions from a single command-and-control server. Attacks like DDoS, Backdoor creation, sending bulk spam emails etc are usually launched using Botnet.

1.3.3.4 Launcher

Malicious program employed as pad to launch other malicious codes. Normally, they use unconventional techniques for launching other malicious programs in order to ensure covertness. They aim at getting greater access to the system.

1.3.3.5 Downloader

Downloaders are fabricated for the sole purpose to download other malicious codes. Whenever an attacker gains access to the system for the first time, he/she installs the downloader just to maintain access for longer duration.

1.3.3.6 Rootkit

It is a malicious piece of software code which is designed to conceal the existence of programs and processes on a particular system. They are used by malwares to hide their malicious activity so that they remain undetected. In this way the attacker gains access of the computer without knowledge of its owner and keeps on launching other attacks in a stealth mode.

1.4 Malware - A Cyber Threat

Massive threats posed by malwares are not only confined to private institutions but has also evolved as a major security concern in the survival of cyber space for many countries around the world. Over a period of time, Trojan has emerged as the most favorite choice in this regard by acquiring a share of 71.85% amongst all newly developed malicious codes [2] as shown in Figure 1 .

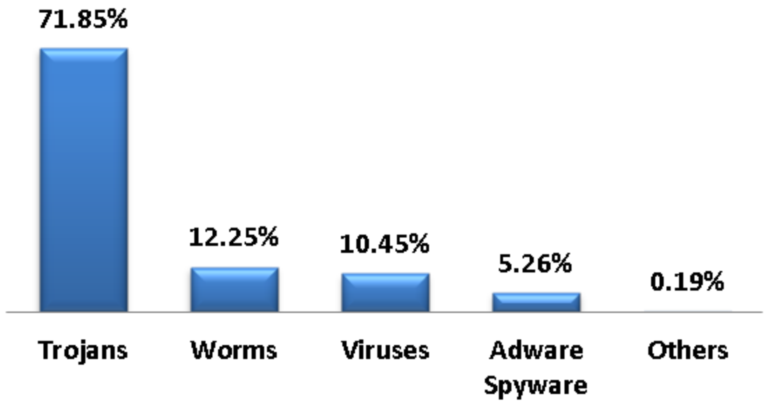

Figure 1: By Type New Malware Strains in 1st Quarter 2014

While analyzing the share of infections type wise, trojan dominated the threat landscape as per Panda Security Labs Quarterly Report for the 1st Quarter 2014 [2] which is shown in Figure 2.

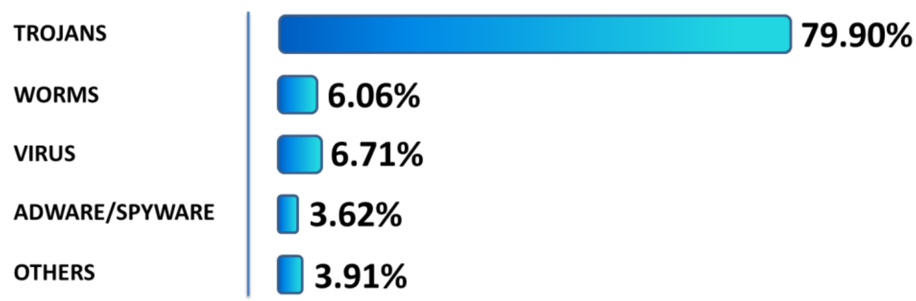

Figure 2: By Type Malware Infections in 1st Quarter 2014

1.5 Why Malware Analysis?

The massive threat caused by malware necessitates a concrete counter measure, but prior to that we need to analyze the capabilities of the understudy malware. Following are the key benefits or outcomes which are normally aimed at are as follows[3]:-

1. In depth malware analysis is often required by incident response team in any organization to investigate the incident in terms of its intended goals, nature of severity and its implications on business continuity plan. By analyzing a malware, actual intends of the malware author could be revealed.
2. What if the forensic expert is encountered with evidence that contains lot of valuable information in shape of malicious softwares, which is quite common in

such cases? Sound and accurate malware analysis is the only path that leads to a correct finding.

3. Malware analysis also helps in extracting a wide variety of vital pieces of information that reveals the attacker intentions and overall design of security breach.
4. Extracting features commonly found in a particular type of malware can serve as a hint or trace to detect such kinds of unknown malware in future.

1.6 Malware Analysis Techniques

Dynamic and static analysis are the two major categories which are not only interrelated but are often considered as a two-way approach, in which one supplements the other [4].

1.6.1 Dynamic Analysis

In this approach the malware sample is always observed while it is made to run in a controlled environment. Behavioral aspects like file system changes, windows registry modifications and network related activities are carefully recorded and then co-related to build a malicious code hypothesis.

1.6.2 Static Analysis

As the name dictates, static analysis is limited to code analysis in a static manner i.e without actually executing the malicious code, which is often performed by disassemblers and debuggers. Going through important portions of code line by line is a laborious and time taking activity but it often gives a complete picture of the malicious code in terms of its intentions, likely targets and estimated effects upon execution. For in-depth investigations malware analysts always consider static analysis as the appropriate option.

1.7 Malware Analysis Tools

There is a wide variety of tools available to malware analysts [4], [5]. Choice of correct and appropriate tool is the foremost step before undertaking a malware analysis effort. Confining to the scope of this research thesis, only tools most commonly used in static malware analysis are manifested in the succeeding paragraphs.

1.7.1 Virus Total

An anti-virus search engine used for the purpose of malware detection based on the signatures [6].

1.7.2 Strings

A wonderful utility developed by SysInternals to extract ASCII strings embedded in the code [7].

1.7.3 BenText

It is similar to ‘Strings’ in terms of functionality. A GUI based utility to search and extract embedded strings [8].

1.7.4 MD5SUM

A CLI utility to calculate MD5 hash in order to record sample fingerprints [9].

1.7.5 PE Explorer

It is a free GUI based tool which scans the portable executable (PE) file and explore the resource tree. Import and export tables can be viewed in detailed using this utility[10].

1.7.6 PEiD

GUI based light weight tool used to scan a PE file. It determines the original entry point (OEP) and packer identification (if any) by making use of entropy [11].

1.7.7 UPX

It stands for ‘Ultimate Packer for eXecutables’. By using switch ‘–d’ it can unpack the executables packed with UPX and its variants [12].

1.7.8 XOR Search

A CLI based utility which attempts to search for a specified string encoded with XOR, ROT and ROL [13].

1.8 Portable Executable (PE)

All windows based executable follow a standard file format known as PE file format. For example .exe, .dll (dynamically link libraries) and object codes [14]. As reflected in yearly Internet Security Threat Report 2014 [15] published by Symantec Corporation, alone in year 2013, 50% or more email had attachments in .exe format which were used for launching phishing attacks. In most of the cases the windows executable designed for malicious purpose are kept as light as possible. Therefore, they are designed to call for API functions at run time instead of including the .dll files itself at compile time. Information on functions to be loaded or called at run time is statically available in import table. Layout of PE file is like a data structure as shown in Table 1.

DOS (MZ) Header
DOS Stub
PE Header
Section 1
Section 2
…….
Section n

Table 1: PE File Format

1.8.1 PE File Header

PE file header contains meta data about executable file, information about the code, application type, library functions needed during execution and memory requirements [14]. These vital informations are very essential for a malware analyst as shown in Table 2 .

Field	**Details of Information**
Imports	Library functions to be used by the malware
Exports	Local Functions to be called by other libraries or programs
Time Date Stamp	Time and Date when compiled
Sections	Section names and sizes

Subsystem	Depicts about GUI or Command-line application type
Resources	Strings (ASCII), program icon, menus etc.

Table 2: PE Header Information

1.8.2 PE File Sections

PE header is followed by sections like .text, .rdata, .data and .rsrc etc, which also contains various information as illustrated in Table 3.

Section	**Details of Information**
.text	Executable Code
.rdata	Global data (Read Only),to be accessed within a program
.data	Global data universally accessible anywhere in the program
.idata	Import Functions (Optional Section)
.edata	Export Functions (Optional Section)
.rsrc	Resource needed

Table 3: Sections of a PE File

1.9 Research Investigations

Research investigations shall cover followings:-

1. Conducting reverse engineering of malware designed for data espionage and backdoor creation.
2. Identification of the important artifacts / patterns commonly found in such type of malwares.
3. Defining mechanism to automate the static analysis process based on established artifacts.

1.10 Motivation

Statically analyzing a program or software code affords an opportunity to carry out an in-depth assessment by establishing a correlation between code and data it contains. By prudently inspecting the series of system calls, APIs and valuable strings it is likely to deduce logical code bombs and time or event based triggers[16].

Features such as the presence of network communication logic, registry, object creations and operating system manipulations can be detected, irrespective of execution on runtime or else. Static analysis, when performed on a de-obfuscated code can match and even inform dynamic program analysis with a more comprehensive insight of the program logic [16]. This approach is also free of run time overheads [17].

Manually analyzing the binary code is time consuming and prone to inconsistencies and inaccuracies due to human errors. In today's era, majority of the malwares are designed to achieve specific goals and targeted at vital organizations. In order to safeguard from this cyber threat, there is dire need to evolve a framework which uses the advantages of static analysis in order to perform automatic malware analysis of executable designed for backdoor creation / data espionage.

1.11 Research Objectives

The main objectives of thesis are:-

1. Malware analysis / identifying specific features of known malwares in PE format designed for data espionage / backdoor creation encompassing signatures / fingerprints, correct file type identifications, extraction of mal-intended code (PE) from benign file, embedded strings and API Calls.
2. Propose a framework to analyze malware encompassing malware identification, packer's identification, unpacking, extraction of embedded API calls and strings. Based on features decision is made to declare any piece of software code as malware or benign. Vital artifacts like URL, IP Address, email etc. are also shown in analysis report.

1.12 Author's Contributions

1. Proposed a framework for undertaking static malware analysis in a systematic and logical order.
2. Feature set (patterns) for API calls and Mal Strings commonly found has been established by analyzing 200 malware samples of known backdoor / data espionage.

3. Proposed a logical flow which is capable of extracting artifacts and features from a suspicious .exe file. Based on the occurrence of features it can decide whether or not the sample is a malware (backdoor / data espionage) or benign.
4. As a proof of concept a prototype tool / utility is developed in python which takes an .exe file as input and after processing generates a report stating whether or not the sample is a malware along with other individual artifacts.
5. The developed utility has been tested on random samples and the results are quite encouraging.

1.13 Thesis Organization

Thesis has been organized in chapters, in which Chapter 2 illustrates a concise and quick literature review of existing solutions in the domain of automated malware analysis along with their advantages and disadvantages. Chapter 3 is all about the malware and benign datasets, details on their acquisition, family it belongs to and further usage in methodology development and testing phase. Proposed solution / framework for automated malware analysis to identify data espionage and backdoor creation are expounded in Chapter 4. Testing of proposed solution and their results along with performance metrics are endorsed in Chapter 5. Chapter 6 is presented as a concluding chapter for this research along with limitations and opportunities for future work explorations.

LITERATURE REVIEW

2.1 Chapter Overview

Literature review chapter exhibits synopsis of research work already conducted in the relevant field. While elucidating their ideas, methodology and results an in-depth analysis on their performance along with pros and cons is also discussed in the subsequent sections of this chapter.

2.2 Automation for Malware Analysis Architecture

It presents an infrastructure for malware analysis which works on network segmentation of traffic, processing of malware samples either on multiple virtual machines or on physical machines as per requirement. Major components of this setup are scheduler, dissector, packet sniffer, virtual machine and a pool of physical machines. Scheduler deals with the availability, reversion and allocation of machines. Capturing of network traffic while sample is made to run is performed by packet sniffer. These packets are fed to dissection for analysis. Dynamic analysis is performed in virtual environment by taking memory dump. Based on the sequence of library calls malware samples are grouped. Although it implements a fast analysis, performs antivirus engines comparisons but all is done by executing the sample and observing its behavior. Approach has limitation of analyzing the malware with “Timeout” and “Event Based” triggers [18].

2.3 Machine Learning based Malware Analysis

A machine learning based framework for automatic malware behavior analysis. It processes large data sets of malware samples and observes their running behavior within a sandbox environment. Behavior analysis report is then incorporated in a vector space. Each segment in vector space is associated with pattern (behavioral aspect). Methods of clustering and classification are applied on this dataset to determine known and new malware categories [19]. Short fall of this methodology are that firstly it requires a large data set and secondly it ignored the potentials of static.

2.4 Timeline Methodology for Reverse Engineering Malwares

Another attempt to formalize a structured method for malware analysis reverse engineering has been proposed by a team of malware analysts at Purdue Malware Laboratory. As compared with the attempt to undertake malware analysis using a technique at random, by adopting this logical flow time efficient, accurate and consistent results can be achieved [20]. This methodology can only serve as a generalized guideline for a malware analyst in order to undertake malware analysis in an organized way with tools information like availability and precedence order in which they are to be applied.

2.5 Dynamic Analysis using TTAnalyze

To dynamically analyze a malicious executable a tool named “TTAnalyze” was developed as an emulated environment to test binaries. Upon execution binaries actions are monitored in the shape of API calls and functions it invokes. Due to emulated environment, methodology remains invisible to the malware code [21]. It generates a comprehensive and concise report. Tool delivers a quick analysis of an unknown malware but can only work on a single execution path.

2.6 Taiwan Malware Analysis Net (TWMAN)

A client-server architecture based tool TWMAN was proposed to undertake malware behavioral analysis. It works on real machines with the belief that malware are always designed to run and infect real system and unlikely to reveal its full functionality on virtual environment. After creating a clean restore image of client, it retrieves malware sample from the repository (server) and runs on client. Vital information related to network activity, files / registers changes is collected and saved in server for formulating an analysis report. Client is then restored back to clean state. This technique performs behavioral analysis and involves huge computing resources [22].

2.7 Cuckoo Sandbox

Cuckoo sandbox is a python based open source malware sandbox application. It is able to automate the whole analysis process with high influx of malware samples. Actual code is open to all and can be customized as per needs and requirement. Concurrent analysis can be run with effective trace of processes in a recursive manner. Behavioral signature along with report is produced as an outcome of this framework. However the full capability of the malware might remain hidden due to the fact that in sandbox environment some portion of code is never triggered, firstly due to absence of a specific event and secondly due to detection of environment by code itself [23].

2.8 Malware Detection using API Calls

Data mining methodologies are being employed to build a framework for analysis of PE files. The fundamental concept behind this structure is that it is possible to determine the behavioral aspect of any malware if its API calls are taken into account during analysis. There are three core components namely an analyzer, feature generator and selector and a classifier. Analyzer simply reads the PE header and extracts all imported API calls. While selecting the feature two aspects are considered; first one is the expected behavior of the sample and the second one is dependent on its distinctive behavior. Expected behavior is based on a combination of feature set, whereas distinctive behavior relied on a single instance of a particular feature. Last component, the classifier performs classification of samples based on selected features. When this methodology is applied on a dataset, it is able to categories samples as positive (malware) or negative (benign) based on features of individual PE sample. The discussed setup has a fairly high accuracy (98.31%) and considerably low false alarm rate (1.51%) when seen in comparison with previous effort in this domain [24].

2.9 Online Malware Analysis Services

Several free online malware analysis services are publically available as listed in Table 4. Malware analyst can make use of these services in order to save time and efforts as compared to manual analysis. But there is a downside about these solutions i.e. Malware sample is required to be shared with them which is serious security concern for any organization [4]. So, before uploading the targeted sample, the analyst must consider following:-

1. Organizational policy on usage of such online malware analysis service.
2. Sharing the samples tend amounts disclosing the detection of targeted attack to malware writer (attacker).

Service	Brief Description
Eureka	It implements a binary unpacking strategy and incorporates API de-obfuscation capabilities to enable the structural analysis of the malware logic. Users can upload their suspicious binaries along with details of the IP Address from where this binary came from. It produces a call graph, summary of found strings and a list of embedded DNS entries [16].
Malwr	Backend functionality is built on the top of Cuckoo Sandbox. It also incorporates other open source services like Virus Total etc. A non-profit service whose services are based on open source and non-commercial technologies. Upon submitting the sample it responds with a complete analysis report [25].
Anubis	Malware analysis service where windows executables, Android Mal Apps and suspicious URL can be submitted for analysis report[26].
Threat Expert	Fully automated advanced threat analysis solution which produces a technically sound analysis report [27].

Table 4: Summary of Few Online Malware Analysis Services

2.10 Chapter Summary

Previous research efforts in the field of automated malware analysis were discussed in this chapter. Out of these Cuckoo sandbox and Detection using API calls stands out as best solutions in the domain of dynamic and static malware analysis techniques respectively.

Both frameworks provide a sound baseline for further exploration and future research efforts.

Chapter 3

DATASETS

3.1 Chapter Overview

The chapter gives an insight of malware samples acquired for the purpose of static analysis. During analysis of individual malware sample, the peculiar features and patterns were recorded for each malware. These samples are used both for the experimentation and later on for the validation of proposed methodology in order to detect such features. Collected samples belong to different malware families of known backdoors / data espionage and more so they are not only collected from a single source in order to avoid monotony of peculiar repetitions of features / patterns.

3.2 Malware Data Set Sources

Malware dataset was mainly acquired from two web resources i.e. Open Malware Project operated by Georgia Tech Information Security Center and Contagio Malware Dump.

3.2.1 Open Malware

Open Malware which was formerly known as “Offensive Computing” is a repository of known malware samples free for analysis purpose to the computer security community [28]. The main purpose of these resources was to provide a platform for the malware analyst in order to improve their skill in protecting their organization information asset and network infrastructures. To accomplish this task malware samples as well as their analysis were made available to public for research purpose. It is the largest malware repository which is available to the public free of cost which is being hosted by Georgia Tech Information Security Center.

3.2.2 Contagio Malware Dump

Contagio is also another repository of latest malware samples, emerging malware threats, comments and analysis by malware analysts all over the world [29]. Malware samples are available in zip files which is password protected in order to avoid accidental self-infection. Password scheme is communicated via email upon request. Contagio claims that anyone who downloads the samples implicitly agrees to wave off their claim for any damage caused by these malwares.

3.2.3 Acquisition of Dataset

The dataset from "Open Malware" was acquired by authenticating through a valid google email account. Thereafter required samples were downloaded in zip format, protected with a common password "infected". Similarly the dataset from "Contagio Malware Dump" was downloaded but their password scheme was acquired by contacting them via email. Samples downloaded from this repository are named in CVE format. A total of 260 known backdoor / data espionage malware samples belonging to 50 variants were analyzed during methodology development and validation phase. Details of known backdoor / data espionage samples along with variant names and no of samples used in each of these categories are appended in Table 5.

Trojan Name	No of Samples
Afcore	11
BackOrifice	5
Beta	1
Bifrose	4
DeepThroat	2
Delf	4
DonaldDick	13
Gift	2
Girlfriend	18
Hupigon	1
InCommander	7
IRCBot	6
Netbus	10
Netdevil	13
Optix	7
OptixPro	10
Server	6
Sub7	8
Subseven	9

Agent	2
APT	1
CoreFlood	3
Danton	13
Doly	3
ECC	1
Graybird	4
Kel	1
Krippled	1
Nerte	5
Netcontrol	3
Prorat	3
Ptakks	6
Remote Control	2
Servu	6
Tofsee	3
Turkojan	1
Win32	4
Bebloh	3
Stuxnet	5
CyberSpy	8
Disttrack_shamoon	1
Espionage	2
Flamer	9
Gauss	4
Hydraq	7
Shamoon	2
SpyEye	4
Thief	6
TrojanSpy	8
Zeus	2

Table 5: List of Trojan Variants along with the Number of Samples Used as Dataset

3.2.4 Data Set for Methodology Development

Out of the 260 acquired malware samples dataset, 200 samples were analyzed / used during the process of methodology development. The details of samples used in this process along with variant names are appended in Table 6.

Backdoor / Data Espionage Trojan Variants (Methodology Development)	Afcore	Kel
	Beta	Krippled
	Bifrose	Nerte
	Delf	Netcontrol
	Gift	Prorat
	Girlfriend	Remote Control
	Hupigon	Servu
	InCommander	Tofsee
	IRCBot	Turkojan
	Netbus	Bebloh
	Netdevil	Stuxnet
	OptixPro	CyberSpy
	Sub7	Disttrack_shamoon
	Subseven	Espionage
	Agent	Flamer
	APT	Hydraq
	CoreFlood	Shamoon
	Danton	TrojanSpy
	ECC	Zeus
	Graybird	

Table 6: Trojan Variants used for Methodology Development

3.2.5 Data Set for Methodology Testing

In order to validate / test the purposed methodology, malware samples belonging to families other then used in development phase were used. A total of 60 backdoor / data espionage malware samples were used during testing phase. The details are mentioned in Table 7.

Backdoor / Data Espionage Trojan Variants (Methodology Testing)	BackOrifice	Ptakks
	DeepThroat	Win32
	DonaldDick	Gauss
	Optix	SpyEye
	Server	Thief
	Doly	

Table 7:Trojan Variants used for Methodology Testing

3.3 Acquisition of Benign Applications

200 Benign applications were obtained from "Contagio Malware Dump". The developed methodology was made to run on these benign applications in order to determine the unique features present in backdoor and/or data espionage samples. 200 benign applications were used during methodology development while rest of the 60 applications were used during testing / validation phase.

3.4 Chapter Summary

Chapter on datasets gave an insight of malware (backdoor and /or data espionage) and benign samples. How they were acquired from different resources and later on how they were used during methodology development and testing phase. A total of 260 malware samples from 50 variants of Trojans were acquired from "Open Malware" and "Contagio Malware Dump", while 260 benign samples were obtained from "Contagio Malware Dump" as per procedure stated in the chapter.

C h a p t e r 4

PROPOSED METHODOLOGY

4.1 Chapter Overview

This chapter elaborates a systematic approach which leads to a proposed methodology to identify malware designed for data espionage and backdoor creation. At first, it deliberates at a framework to extract features such as API calls and strings, followed by occurrence of malicious features and strings commonly found in known backdoors and data espionage malwares. Using the acceptance rule based on weightage of features in contrast with benign executables, features are selected or rejected. Finalized feature set and their respective weightages are then used to calculate scores of individual malware and benign samples. Percentile approach is then used to set the threshold score for deciding whether the executable sample is a malware or benign. At the end, using the artifacts extraction techniques integrated with feature set and threshold value, a methodology is proposed to meet the objectives set by this research effort.

4.2 Static Malware Analysis Framework

Malware analysis of 200 known backdoors and data espionage samples was conducted to determine a set of features and patterns commonly found in such kind of mal codes. The whole process was split in five distinct phases as depicted in Figure 3. In order to remain focused on the research objectives following research parameters were set right from outset:-

1. Malware samples were limited to PE file formats only.
2. De-Obfuscation was confined to deal with malwares packed with UPX [12] only.
3. Scope of analysis was restricted to extraction of API Calls using IAT, strings and network related strings while dissecting a PE file structure.
4. API Calls and Strings (Mal code) are considered as a candidate for common feature set.

5. Unique patterns like URLs, emails, Registry entries, IP Address and passwords etc. are taken as individual artifacts for each sample.

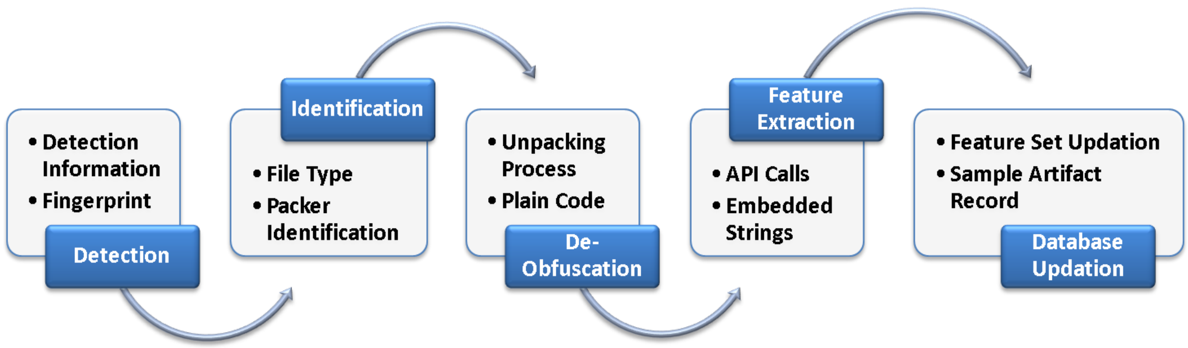

Figure 3: Static Malware Analysis Framework

4.2.1 Detection of Malware Samples

Acquired samples of known backdoors and data espionage were put at trial across several online malware detection engines to validate their tagged identification. The entire process is depicted in Figure 4. In that hash of a single malware sample is searched at each malware detection facility and return results are compared to confirm their correct identification. The same process is repeated for entire dataset.

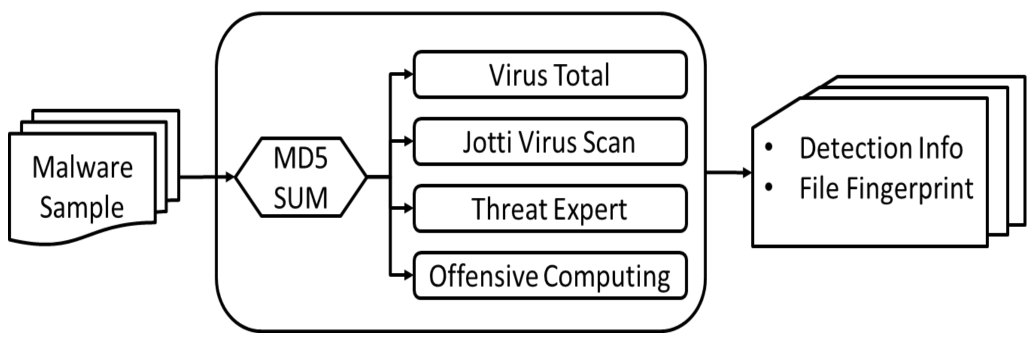

Figure 4: Malware Detection Phase

4.2.2 Identification of Malware

Sample is analyzed for correct file type using TrID utility [30]. Usage of this command line utility is shown in Figure 5.

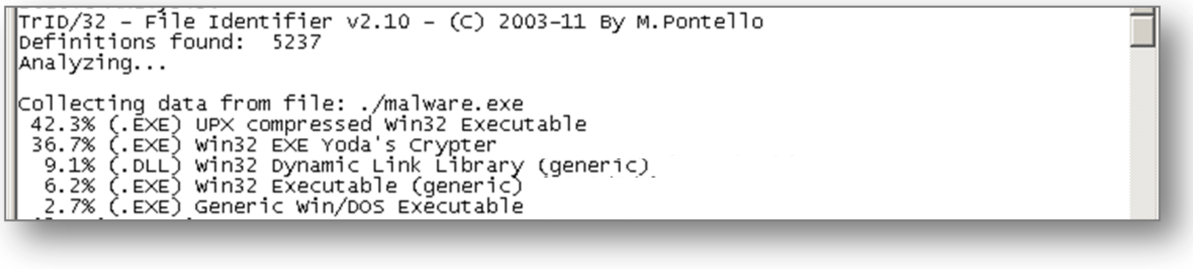

Figure 5: TrID Usage in Identification Phase

After having confirmed the sample as .exe file, in order to determine whether or not the malware code is packed and if packed what is the packer identification. A GUI based utility named PEiD [11] is used for this purpose as illustrated in Figure 6.

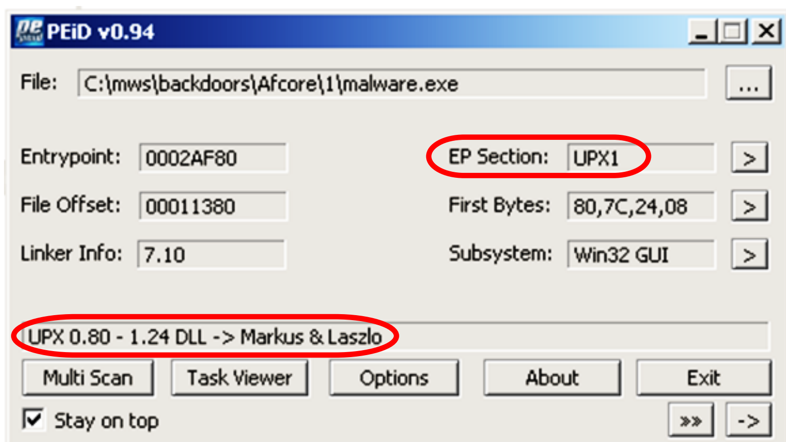

Figure 6: Packet Identification using PEiD

4.2.3 De-Obfuscation

An attempt is made to unpack the sample using UPX with switch "-d". If the sample was packed with UPX and its variants it will be unpacked successfully as shown in Figure 7. It is to be noted that the unpacked file differs in hash as well as in size from the original file.

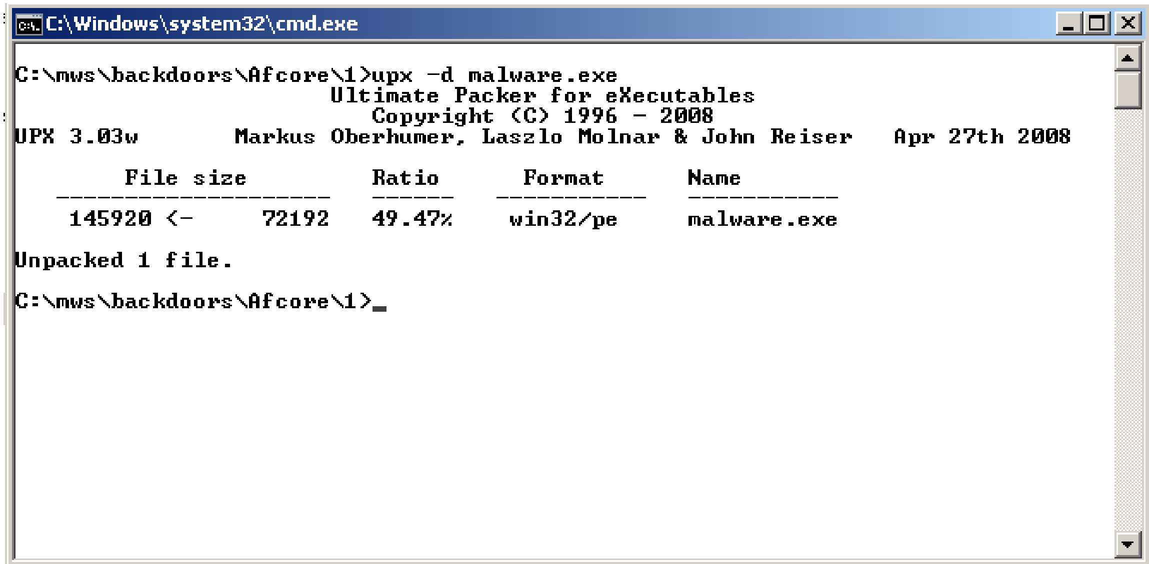

Figure 7: Unpacking of Packed Malware Sample using UPX

4.2.4 Extraction of Features

Each PE file contains an import tables which carries very useful information such as DLLs and API function calls. On dissecting the PE using PE explorer, API Calls can be extracted without actually executing the code. Lot of useful and important strings are embedded in the code at compile time. 'Strings' developed by Windows SysInternals is a wonderful utility for extracting these ASCII strings from the malicious code [7].The whole extraction process is depicted in Figure 8.

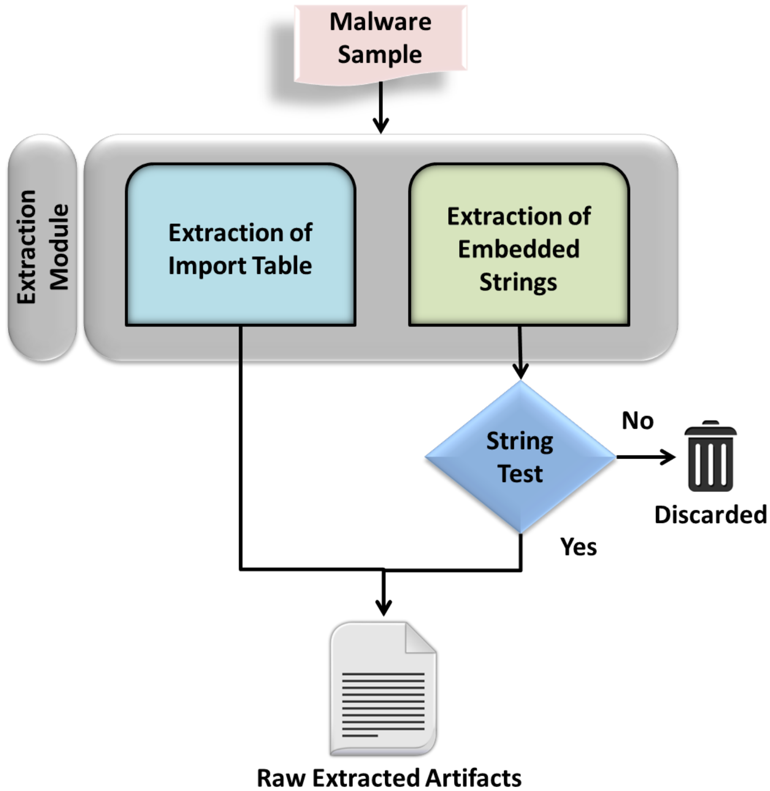

Figure 8: Feature Extraction Phase

4.2.5 Update of Database

Extracted artifacts are then compared with the existing data base, in case of new artifacts found in the known sample of Trojan Backdoor and /or Data Espionage, data base files are updated by adding new features and this process is repeated till complete analysis of all malware samples. API calls and meaningful strings are manually searched in raw artifacts file. In addition to common artifacts (API and strings), unique patterns in the shape of network related strings like URLs, emails, IP Address etc. , registry entries and passwords etc. are also looked for and recorded as individual artifacts for each sample and stored in sample artifacts found file. Updation phase is illustrated in Figure 9.

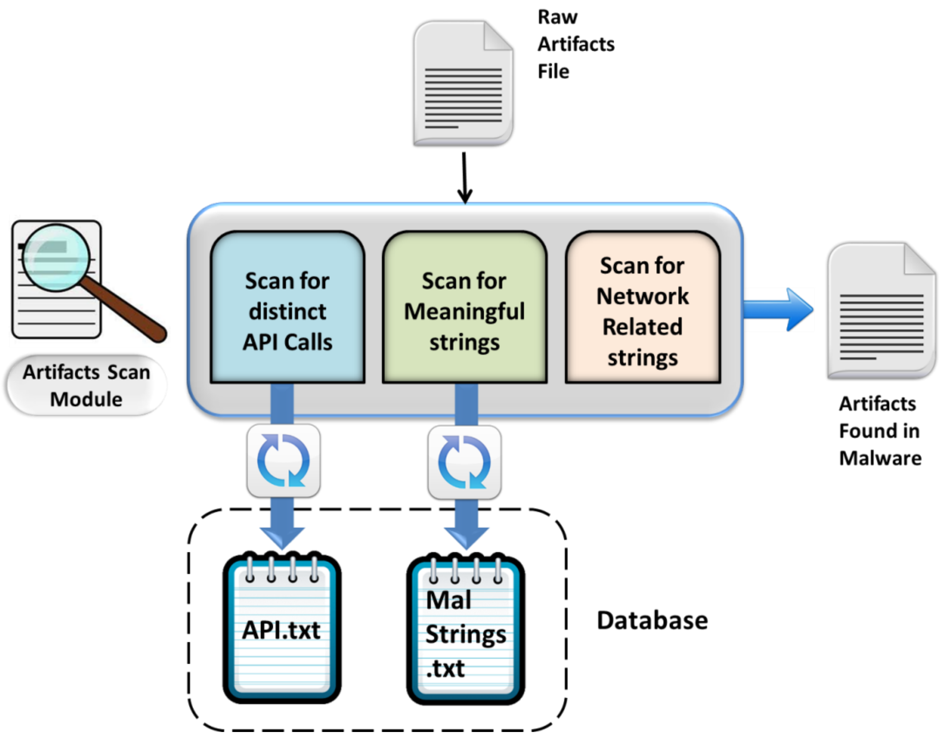

Figure 9: Database Updation Phase

On termination of update phase, a feature set for both API and Mal Strings is stored as database which will act as a baseline for checking / searching for the presence of such patterns in a malware sample. Feature set for API and Mal strings so obtained is placed at Appendix A and B.

4.3 Selection of Features in a Malware Designed for Data Espionage and Backdoor Creation

200 malware samples of known backdoor / data espionage and 200 benign applications were checked / searched for the presence of feature set as per flow diagram shown in Figure 10.

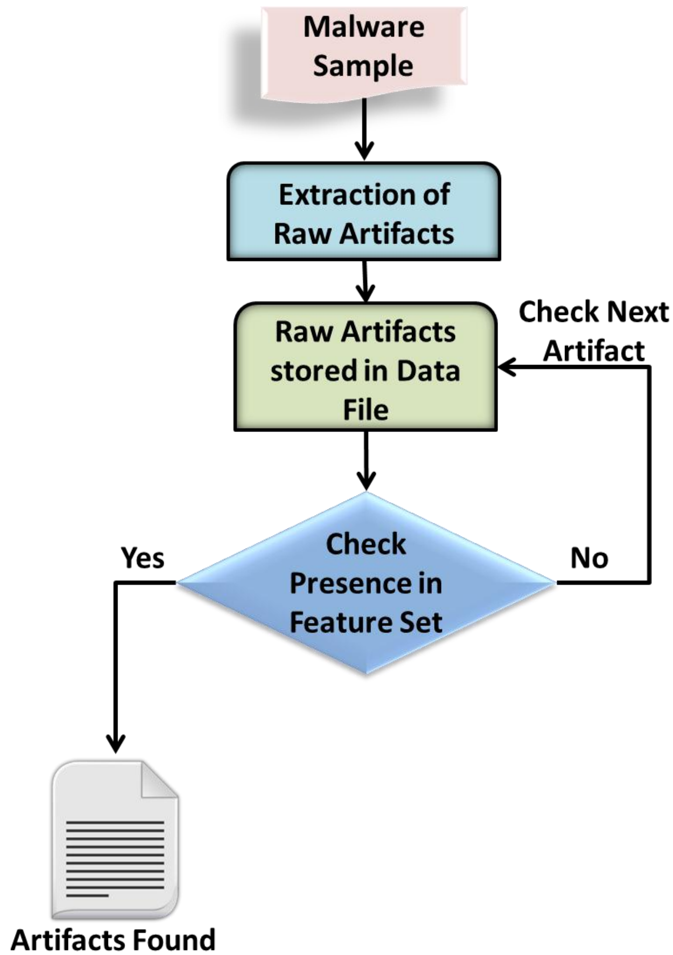

Figure 10: Checking Presence of Features in Malware Sample

4.3.1 Features Presence in Malware Dataset

Using the flow chart depicted in Figure 10, all malware samples are put at trial for the presence of features. If a particular feature is present in a sample it is recorded as '1' against that feature else '0' is recorded. The occurrences of feature sets in malware dataset are shown in Appendix C and D.

4.3.2 Features Presence in Benign Dataset

Using the flow chart depicted in Figure 10, all benign samples are put at trial for the presence of features. If a particular feature is present in a sample it is recorded as '1' against that feature else ‘0’ is recorded. The occurrences of feature sets in benign dataset are shown in Appendix E and F.

4.3.3 Calculating Difference of Features Presence

Alone presence of a particular feature at a high occurrence in malware dataset does warrant it to be an indicative of a malware, because the same feature may also be present at a comparable ratio in benign dataset as well. So, taking the arithmetic difference of a feature occurrence in malware and benign data will indicate true malignant as shown in Appendix G and H.

4.3.4 Feature Set Selection

After sorting the features in ascending order as per their arithmetic difference as depicted in Appendix G and H. Based on difference in feature occurrence, weightage is assigned to each feature entry. Feature “Selection” or “Rejection” is governed by the rule that if weightage is greater than zero, the feature is “Selected” else “Rejected”. Finally, feature set for both API and Mal Strings is defined along with their respective score as shown in Appendix I and J.

4.4 Score Calculation based on Selected Feature Set

Based on presence of a particular feature individual scores are added up to calculate the overall score of a particular sample. Let F_i is a feature which is present in a particular sample which is assigned a value‘1’ if present else ‘0’and W_i be its respective weightage, then the total malicious score is defined as:

$$\text{Total Score} = \sum(F_i * W_i) \qquad (4.1)$$

Where, i ranges from 1 to n, and n is the total no of features in a feature set. Following equation 4.1, score is calculated for both feature sets and for malware and benign datasets are illustrated in Appendix K and L.

4.5 Setting Threshold

After calculating score for malware and benign dataset, there is need to establish a threshold value such that if total score of any sample is greater than the set threshold value, it is declared as malware else benign. In order to set the threshold for both the feature sets, percentile approach is adopted [31]. For setting the threshold, malware samples dataset score is sorted in ascending order. Keeping the strict criteria, percentile value of 5 is selected. For 200 malware samples, percentile value is calculated as under:-

$$P_{05} = \frac{5}{100} . (200+1) \qquad (4.2)$$

$P_{05} = 10$

It implies that 10th value has percentile value of 5. So, referencing Appendix K, 10th value has a score of 1983 in case of API and 240 in case of Mal Strings. These values depicts that 5% values of score in malware sample dataset are less than these as shown in Figure 11 and Figure 12 and established as threshold for respective feature set.

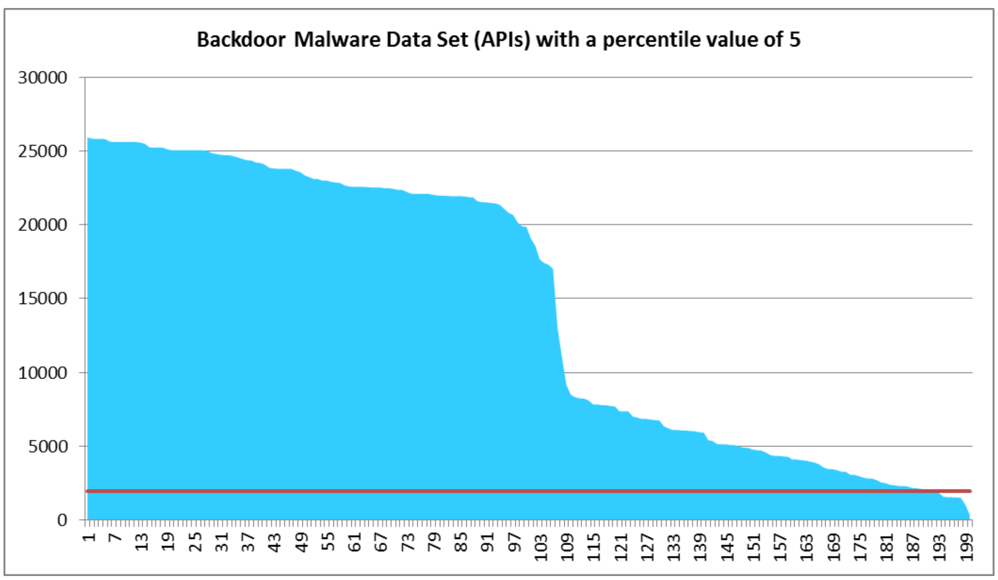

Figure 11: Graph showing Threshold for APIs using percentile value of 5

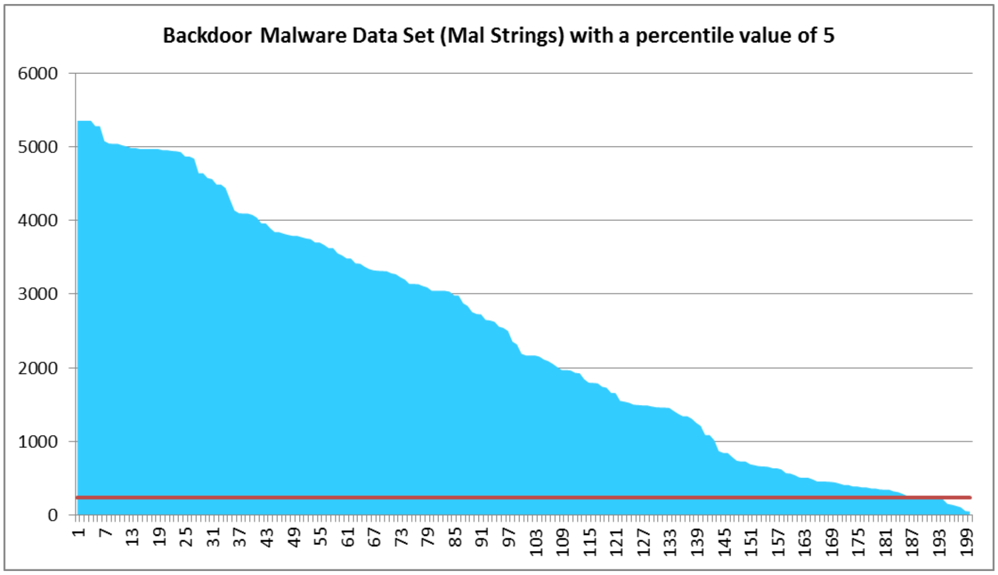

Figure 12: Graph showing Threshold for Mal Strings using percentile value of 5

4.6 Finalization of Proposed Methodology

After incorporating the feature sets and threshold value, the proposed automated malware analysis framework is refined as a proposed methodology for automated malware analysis to identify malware designed for data espionage and backdoor creation. Any suspicious executable sample is given as an input for quick static analysis. The automated analysis process as elucidated in Figure 13 is completed in following eight strides:-

1. PE file is checked for its detection (if any) using anti-malware detection engines.

2. Sample is then scanned for its correct file type identification.

3. Packer identification and unpacking attempt.

4. Raw artifacts (APIs, Strings and Network related Strings) extracted from unpacked PE file.

5. Feature Set presence is checked in raw artifacts file.

6. Separate score in both feature set (API and Mal Strings) is calculated.

7. Both scores are compared with its respective threshold values. If any value is above the threshold it is marked as “Malware” else “Benign”.

8. Final Report along with individual artifacts is compiled for further deep analysis.

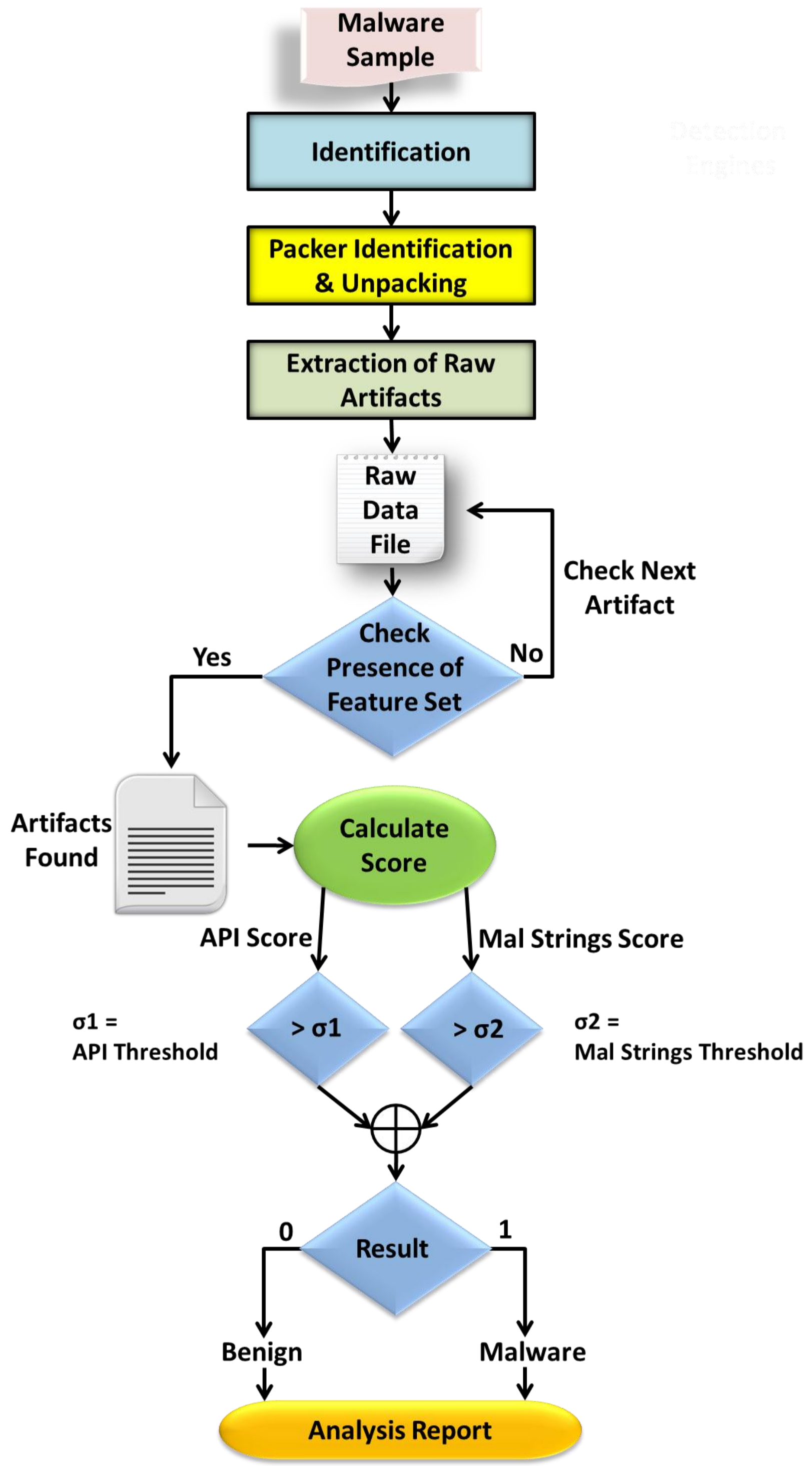

Figure 13: Proposed Methodology for Automated Malware Analysis

4.7 Chapter Summary

In this chapter the proposed methodology was discoursed adopting a systematic and an elaborative approach. At first, a framework for malware analysis was developed and based on that malware analysis of 200 each of backdoor / data espionage and benign samples were undertaken. Based on the artifacts / patterns found, feature set for API calls and Mal Strings were established. Using the percentile methodology, threshold for the feature set is determined. Lastly, a proposed methodology for automated malware analysis to identify data espionage and backdoor creation was developed and conferred.

EVALUATION OF PROPOSED METHODOLOGY

5.1 Chapter Overview

This chapter encompasses discussion on testing and validation of proposed methodology. Random samples of known backdoor / data espionage and benign softwares are at trial during this phase. After determining the TN and FP, performance metrics like accuracy, precision, sensitivity and false alarm rate are calculated.

5.2 Data Set for Testing

Data set for testing comprises of 60 malwares (backdoor/data espionage) and 60 benign softwares which were acquired from Open Malware [28] and Contagio Malware Dump [29] respectively. Malware and benign samples were used at random to test the performance of proposed methodology.

5.3 Testing Results

The methodology calculates API and Mal String score for each specimen and then decides whether or not the specimen is malware or benign. The decision is based on the comparison of score value with the threshold value. Testing results for malware and benign applications are shown in Table 8 and Table 9 respectively.

Malware Sample	Score		Decision	Result
	API (A)	Mal String (M)	If (A>σ1 'OR' M> σ2) "Malware" Else "Benign"	
1	20196	2774	Malware	TP
2	9986	1663	Malware	TP
3	20196	2939	Malware	TP
4	5797	1491	Malware	TP
5	7480	1535	Malware	TP
6	4031	499	Malware	TP
7	22164	3714	Malware	TP
8	19216	3317	Malware	TP

9	7481	917	Malware	TP
10	19179	2939	Malware	TP
11	6672	1117	Malware	TP
12	15227	781	Malware	TP
13	14442	1138	Malware	TP
14	19235	3104	Malware	TP
15	6672	1117	Malware	TP
16	6026	1374	Malware	TP
17	19216	3084	Malware	TP
18	8132	917	Malware	TP
19	19179	3023	Malware	TP
20	14456	1244	Malware	TP
21	6443	963	Malware	TP
22	5972	547	Malware	TP
23	11887	1392	Malware	TP
24	22449	1921	Malware	TP
25	23336	4772	Malware	TP
26	22678	3526	Malware	TP
27	25260	4602	Malware	TP
28	7461	1535	Malware	TP
29	22327	2801	Malware	TP
30	22101	2796	Malware	TP
31	6642	1162	Malware	TP
32	25486	4441	Malware	TP
33	22908	4345	Malware	TP
34	6256	1218	Malware	TP
35	6256	1437	Malware	TP
36	2889	60	Malware	TP
37	13380	265	Malware	TP
38	13169	526	Malware	TP
39	14850	905	Malware	TP
40	13711	615	Malware	TP
41	13681	840	Malware	TP
42	13299	615	Malware	TP
43	21416	3145	Malware	TP
44	17260	2243	Malware	TP

45	25080	3548	Malware	TP
46	22508	2915	Malware	TP
47	1790	274	Malware	TP
48	4152	263	Malware	TP
49	4157	263	Malware	TP
50	4152	263	Malware	TP
51	3891	242	Malware	TP
52	3859	776	Malware	TP
53	6542	493	Malware	TP
54	5927	1041	Malware	TP
55	23851	5332	Malware	TP
56	25436	4994	Malware	TP
57	25436	4994	Malware	TP
58	23851	5332	Malware	TP
59	20921	2998	Malware	TP
60	25436	4994	Malware	TP

Table 8: Testing Results of Malware Samples

Benign Sample	**Score**		**Decision**	**Result**
	API (A)	**Mal String (M)**	**If (A>σ1 'OR' M> σ2) "Malware" Else "Benign"**	
1	297	102	Benign	TN
2	22	89	Benign	TN
3	22	56	Benign	TN
4	632	74	Benign	TN
5	250	56	Benign	TN
6	146	208	Benign	TN
7	30	113	Benign	TN
8	22	35	Benign	TN
9	1427	108	Benign	TN
10	431	82	Benign	TN
11	122	49	Benign	TN
12	140	66	Benign	TN
13	22	35	Benign	TN
14	1427	59	Benign	TN
15	1427	108	Benign	TN
16	1427	108	Benign	TN

17	22	35	Benign	TN
18	1427	59	Benign	TN
19	154	157	Benign	TN
20	22	35	Benign	TN
21	34	100	Benign	TN
22	431	77	Benign	TN
23	363	56	Benign	TN
24	294	145	Benign	TN
25	1427	59	Benign	TN
26	1427	113	Benign	TN
27	755	129	Benign	TN
28	1427	108	Benign	TN
29	1427	108	Benign	TN
30	1427	59	Benign	TN
31	364	110	Benign	TN
32	867	89	Benign	TN
33	387	526	Malware	FP
34	34	157	Benign	TN
35	756	213	Benign	TN
36	133	135	Benign	TN
37	1427	113	Benign	TN
38	774	76	Benign	TN
39	1442	232	Benign	TN
40	73	89	Benign	TN
41	238	35	Benign	TN
42	210	238	Benign	TN
43	1427	108	Benign	TN
44	610	201	Benign	TN
45	54	135	Benign	TN
46	1002	54	Benign	TN
47	79	77	Benign	TN
48	22	35	Benign	TN
49	22	136	Benign	TN
50	1960	59	Benign	TN
51	84	0	Benign	TN
52	79	213	Benign	TN

53	599	226	Benign	TN
54	188	82	Benign	TN
55	1861	59	Benign	TN
56	44	95	Benign	TN
57	30	113	Benign	TN
58	1160	167	Benign	TN
59	22	89	Benign	TN
60	169	63	Benign	TN

Table 9:Testing Results of Benign Samples

5.4 Performance Metrics

Performance of proposed methodology is measured in terms of accuracy, precision and false alarm rate[32]. The summary of performance metrics on results is illustrated in Table 10.

5.4.1 True Positive (TP)

Methodology correctly identifies given sample as "Malware".

5.4.2 False Negative (FN)

Methodology wrongly identifies given sample as "Benign".

5.4.3 True Negative (TN)

Methodology correctly identifies given sample as "Benign".

5.4.4 False Positive (FP)

Methodology wrongly identifies given sample as "Malware".

5.4.5 Accuracy

Portion of all correct decision are measured in terms of Accuracy and are calculated as:

$$Accuracy = \frac{(TP + TN)}{(TP + FP + TN + FN)}$$

5.4.6 Precision

Portion of predicted positive cases that were correctly identified as positive is known as precision.

$$Precision = \frac{TP}{(TP + FP)}$$

5.4.7 Recall (Sensitivity)

Portion of correct categories that were assigned by the methodology is defined as “Recall” or “Sensitivity”. It is phenomenon to measure the sensitivity of the methodology.

$$Recall = \frac{TP}{(FP + FN)}$$

5.4.8 False Alarm Rate (FAR)

Number of misclassification on a given set of samples is termed as false alarm rate. It is calculated as:

$$FAR = \frac{(FP + FN)}{(TP + FP + TN + FN)}$$

Total No of Malware Samples	M	60
True Positive	TP	60
False Negative	FN	-
Total No of Benign Samples	N	60
True Negative	TN	59
False Positive	FP	1
Accuracy	ACC	99.17%
Precision	P	98.3%
Recall (Sensitivity)	R	100%
False Alarm Rate	FAR	0.83%

Table 10: Summary of Performance Metrics

5.5 Results Validation

If we compare our results with “Detection of Malware Based on Mining API” formulated by Sami et al. [24] and “Intelligence Malware Detection System (IMDS)” proposed by Yangfang Ye et al. [33] which are shown in Table 11, we see that our results are quite promising.

Performance Metric	IMDS	Malware Detection using API Mining	Our Proposed Method
Accuracy	93.07%	98.31%	99.17%
Precision	80.13%	98.5%	98.3%
Recall (Sensitivity)	97.19%	99.7%	100%
False Alarm Rate	19.86%	1.51%	0.83%

Table 11 : Comparison of Results with Existing Methods

Chapter 6

CONCLUSION AND FUTURE DIRECTIONS

6.1 Chapter Overview

The chapter concludes this dissertation by summing up the research objectives accomplished and also provides candid directions for future work in the field of automated malware analysis.

6.2 Research Goals Attained

During the process of this research starting from the literature review and ending at the validation and testing of proposed methodology, following objectives have been attained:-

1. Review of malware categories / types, malware analysis techniques including tools and architectures.
2. In-depth analysis of existing solutions on the subject.
3. Developed a framework for static analysis of malware. Employing this framework for analysis of known backdoor / data espionage malwares in order to build a feature set (APIs and Mal strings) commonly found in such type of mal codes.
4. Finalization of these features / patterns in contrast with benign application to refine the feature set that warrants an application to be categorized as malware or otherwise.
5. Computing malware score for each sample based on weightage of their found artifacts amongst both feature set separately and setting their respective threshold for acceptance and rejection of a candidate application as malware.
6. Keeping proposed framework for static malware analysis as baseline and taking feature sets / patterns and threshold score as parameters developed a methodology to automatically analyze malware designed for data espionage and backdoor creation.
7. As a proof of concept, a python script has been implemented to verify the utility of proposed methodology.

8. Validation of proposed methodology by putting a reasonable number of malware and benign samples at trial in random sequence. In that, one by one test samples are given as input to the python script and their results along with report on artifacts found are recorded.
9. Finally with the help of performance metrics, results are compared with previous work in the relevant field.

6.3 Future Directions

1. Research was confined to either un-obfuscated malwares or packed with UPX and its variants. Modern / sophisticated cyber-attacks relies on malware that are designed by employing a number of other obfuscation techniques which are hard to crack. Countering these methods is a challenging avenue in this field.

2. This dissertation focused on malwares of PE format designed for data espionage and malware creations only. Expanding the scope vertically towards exploring the rootkits and horizontally towards other file types like .pdf, .doc, .xls and web formats would be a remarkable effort in the field of automated malware analysis.

3. If this utility is incorporated with existing anti-malware techniques, it could pay dividends in combating against new attacks in a more befitting manner.

PUBLICATIONS

1. "On the Approach of Static Feature Extraction in Trojans to Combat against Zero-day Threats", IEEE International Conference on IT Convergence and Security 2014 (ICITCS-2014), 28-30 October 2014.

2. "Patterns in Malware Designed for Data Espionage and Backdoor Creation", IEEE International Bhurban Conference on Applied Sciences and Technology 2015 (IBCAST-2015), 13-17 January 2015.

REFERENCES

[1] Sophos, “Thesaurus: The A-Z of computer and data security threats.” .

[2] Panda Security, “PandaLabs Quarterly Report January-March 2014,” 2014.

[3] L. Zeltser, “Introduction to Malware Analysis,” *CS2107-Semester IV, 2012-2013*. SANS Institute, pp. 1–36.

[4] L. Zeltser, “Analyzing Malicious Software,” pp. 59–83, 2010.

[5] A. Verma, W. Jeberson, and V. Singh, “A LITERATURE REVIEW ON MALWARE AND ITS ANALYSIS,” 2013, vol. 05, no. 16, pp. 71–82.

[6] “Virus Total.” [Online]. Available: https://www.virustotal.com/.

[7] Windows Sysinternals, “Strings Utility.” [Online]. Available: http://technet.microsoft.com/en-us/sysinternals/bb897439.

[8] “Ben Text.” [Online]. Available: http://www.mcafee.com/us/downloads/free-tools/bintext.aspx.

[9] “MD5sums for Windows.” [Online]. Available: http://www.pc-tools.net/win32/md5sums/.

[10] Heaven Tools, “PE Explorer.” [Online]. Available: http://www.heaventools.com/overview.htm.

[11] “PEiD Description.” [Online]. Available: http://www.aldeid.com/wiki/PEiD#PEiD.

[12] “UPX - Ultimate Packers for Executables.” [Online]. Available: http://upx.sourceforge.net/.

[13] Didier Stevens, “XORsearch.” [Online]. Available: http://blog.didierstevens.com/programs/xorsearch/.

[14] M. Sikorski, A. Honig, and S. Lawler, *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*. 2012.

[15] Symantec Corporation, “Internet Security Threat Report 2014,” 2014.

[16] M. Sharif, V. Yegneswaran, H. Saidi, P. Porras, and W. Lee, “Eureka : A Framework for Enabling Static Malware Analysis,” pp. 481–500, 2008.

[17] J. Bergeron, M. Debbabi, J. Desharnais, M. M. Erhioui, Y. Lavoie, N. Tawbi, J. Bergeron, M. Debbabi, J. Desharnais, M. Erhioui, Y. Lavoie, and N. Tawbi, "Static Detection of Malicious Code in Executable Programs ∗."

[18] R. R. Branco and U. Shamir, "Architecture for automation of malware analysis," *Malicious Unwanted Softw. (MALWARE), 2010 5th Int. Conf.*, 2010.

[19] K. Rieck, P. Trinius, C. Willems, and T. Holz, "Automatic analysis of malware behavior using machine learning," *J. Comput. Secur.*, vol. 19, pp. 639–668, 2011.

[20] C. Q. Nguyen, N. G. Street, and J. E. Goldman, "Malware analysis reverse engineering (MARE) methodology & malware defense (MD) timeline," *2010 Inf. Secur. Curric. ...*, 2010.

[21] U. Bayer, C. Kruegel, and E. Kirda, "TTAnalyze : A Tool for Analyzing Malware."

[22] H.-D. Huang, C.-S. Lee, H.-Y. Kao, Y.-L. Tsai, and J.-G. Chang, "Malware behavioral analysis system: TWMAN," *2011 IEEE Symp. Intell. Agent*, pp. 1–8, 2011.

[23] Cuckoo Sandbox, "Automated Malware Analysis." [Online]. Available: http://www.cuckoosandbox.org/about.html#about.

[24] A. Sami, B. Yadegari, and H. Rahimi, "Malware detection based on mining API calls," *Proc. ...*, pp. 1020–1025, 2010.

[25] Cuckoo Sandbox, "Malwr - Malware Analysis." [Online]. Available: https://malwr.com/.

[26] International Secure Systems Lab, "Anubis - Malware Analysis for Unknown Binaries." [Online]. Available: http://anubis.iseclab.org/.

[27] "Threat Expert." [Online]. Available: http://www.threatexpert.com/submit.aspx.

[28] Georgia Tech Information Security Center, "Open Malware." [Online]. Available: http://oc.gtisc.gatech.edu:8080/.

[29] Milaparkour, "Contagio Malware Dump." [Online]. Available: http://contagiodump.blogspot.com/.

[30] Marco Pontello, "TrID - File Identifier." [Online]. Available: http://mark0.net/soft-trid-e.html.

[31] NIST, *Engineering Statistics Handbook*. http://www.itl.nist.gov/div898/handbook/prc/section2/prc262.htm.

[32] R. Tian, L. Batten, R. Islam, and S. Versteeg, "An automated classification system based on the strings of trojan and virus families," *2009 4th Int. Conf. Malicious Unwanted Softw.*, pp. 23–30, Oct. 2009.

[33] D. Wang and T. Li, "IMDS : Intelligent Malware Detection System," 2007.

Appendix A

Initial API Calls Feature Set

Ser	API	Ser	API
1.	VirtualAlloc	35.	ShowWindow
2.	VirtualFree	36.	WaitForSingleObject
3.	CreateFileA	37.	EnumWindows
4.	ReadFile	38.	GetModuleFileNameA
5.	SetFilePointer	39.	PeekMessageA
6.	RtlUnwind	40.	RegCloseKey
7.	GetModuleHandleA	41.	RegisterClassA
8.	CreateThread	42.	WideCharToMultiByte
9.	SetEndOfFile	43.	GetCommandLineA
10.	FindFirstFileA	44.	LocalAlloc
11.	GetFileSize	45.	lstrcpyA
12.	GetVersionExA	46.	MessageBoxA
13.	FindClose	47.	SystemParametersInfoA
14.	GetVersion	48.	CallNextHookEx
15.	RegOpenKeyExA	49.	FindWindowA
16.	TlsGetValue	50.	FreeLibrary
17.	TlsSetValue	51.	RegSetValueExA
18.	ExitProcess	52.	SetWindowLongA
19.	RegQueryValueExA	53.	UnhookWindowsHookEx
20.	DestroyWindow	54.	WSAStartup
21.	GetFileType	55.	CloseHandle
22.	lstrlenA	56.	GetStdHandle
23.	RaiseException	57.	SetWindowsHookExA
24.	SetErrorMode	58.	socket
25.	WriteFile	59.	TranslateMessage
26.	PostMessageA	60.	WSACleanup
27.	SendMessageA	61.	CompareStringA
28.	DefWindowProcA	62.	CreateEventA
29.	DispatchMessageA	63.	GetActiveWindow
30.	InitializeCriticalSection	64.	GetCPInfo
31.	CreateWindowExA	65.	GetParent
32.	GetDiskFreeSpaceA	66.	GetSystemMetrics
33.	GetLocaleInfoA	67.	GetWindowThreadProcessId
34.	LoadLibraryA	68.	SetTimer

Ser	API	Ser	API
69.	closesocket	86.	GlobalLock
70.	GetWindow	87.	InterlockedDecrement
71.	GetWindowLongA	88.	InterlockedIncrement
72.	GetWindowTextA	89.	MultiByteToWideChar
73.	IsWindowVisible	90.	SelectObject
74.	KillTimer	91.	SetEvent
75.	LoadCursorA	92.	SizeofResource
76.	PostQuitMessage	93	UnregisterClassA
77.	ShellExecuteA	94.	DeleteDC
78.	VirtualQuery	95.	FindResourceA
79.	BitBlt	96.	GetForegroundWindow
80.	DeleteFileA	97.	GetStartupInfoA
81.	FileTimeToLocalFileTime	98.	GetThreadLocale
82.	FormatMessageA	99.	GlobalUnlock
83.	GetClientRect	100.	LoadLibraryExA
84.	GetDiskFreeSpaceExA	101.	SysFreeString
85.	GetKeyboardType	102.	VariantClear
1536.	DialogBoxIndirectParamA	1554.	RegOpenKeyExW
1537.	EndDialog	1555.	DllCanUnloadNow
1538.	FillPath	1556.	DllGetClassObject
1539.	LoadLibraryExW	1557.	FlsAlloc
1540.	lstrcmpW	1558.	FlsFree
1541.	lstrlenW	1559.	FlsGetValue
1542.	RegCreateKeyW	1560.	FlsSetValue
1543.	RegEnumKeyA	1561.	GetVersionExW
1544.	ShellExecuteExA	1562.	InterlockedCompareExchange
1545.	CoFreeUnusedLibraries	1563.	RegSetValueExW
1546.	CoTaskMemAlloc	1564.	DllRegisterServer
1547.	GetProcessWindowStation	1565.	GetCurrentProcess
1548.	LoadStringW	1566.	InitializeCriticalSectionAndSpinCount
1549.	TerminateProcess	1567.	GetModuleHandleW
1550.	CoTaskMemFree	1568.	IsDebuggerPresent
1551.	GetModuleFileNameW	1569.	SetUnhandledExceptionFilter
1552.	IsValidCodePage	1570.	QueryPerformanceCounter
1553.	LoadLibraryW	1571.	GetSystemTimeAsFileTime

Appendix B

Initial Mal Strings Feature Set

Ser	Mal Strings	Ser	Mal Strings
1.	OWNER	35.	Username
2.	Password	36.	Deleting all files of current folder
3.	HOST	37.	On connect
4.	DISABLED	38.	REGISTER
5.	File not found	39.	Locked
6.	Stack overflow	40.	Passwords
7.	WinSock	41.	ScktComp
8.	Not Found	42.	UseDockManager
9.	Too many open files	43.	HideSelection
10.	Division by zero	44.	SOCKS
11.	SysUtils	45.	No address specified
12.	Remote	46.	UrlMon
13.	shared	47.	FocusControl
14.	SHUTDOWN	48.	PasswordChar
15.	TIMER	49.	CreateKey
16.	File access denied	50.	Send message
17.	Floating point division by zero	51.	INSTALL
18.	Floating point overflow	52.	not connected
19.	Floating point underflow	53.	Port:
20.	No argument for format '%s'	54.	RedrawNow
21.	Privileged instruction	55.	Sending
22.	Read beyond end of file	56.	Connection timed out
23.	Disconnect	57.	Download
24.	SERVER	58.	Delete file
25.	Stream	59.	ServerSock
26.	Stream read error	60.	Address already in use
27.	Stream write error	61.	Already connected
28.	Class %s not found	62.	Connected to
29.	Commctrl	63.	Connection reset by peer
30.	RegisterAutomation	64.	Delete *.*
31.	Resource %s not found	65.	Host is down
32.	WindowState	66.	Listening
33.	WSocket	67.	Network is down
34.	Sender	68.	Network is unreachable

Ser	Mal Strings	Ser	Mal Strings
69.	No route to Host	86.	System directory
70.	RESTART	87.	Windows directory
71.	Socket is already connected	88.	Create Directory
72.	Socket is not connected	89.	Host unreachable
73.	Host not found	90.	LocalPort
74.	Upload	91.	Network unreachable
75.	Reboot	92.	SocksAuthentication
76.	ReBOOT	93	SocksServer
77.	Too many users	94.	BROWSER
78.	TRANSFER	95.	winspool.drv
79.	Resolve Host	96.	Connecting to
80.	Too many processes	97.	Disconnected
81.	From:	98.	ServerSocket
82.	login	99.	ServerType
83.	Logoff	100.	Directory not empty
84.	Disconnecting	101.	Enable window
85.	Exit Windows	102.	File deleted
670	retrieving user name	688	Shared device mapped successfully
671	Run at startup	689	SMTP_relay
672	Runtime pathname	690	SMTP_relay = %s
673	Save Downloaded File to	691	TCP_ports
674	Save Shot to File	692	TCP_ports = %s
675	ScanList	693	There is no screensaver password
676	Screen contents of console '%s'	694	Total files
677	ScreenSaver password: '%s'	695	Transfer finished
678	Searching for file '%s' from root '%s':	696	Value '%s' deleted
679	Sending %s to %d.%d.%d.%d:%d	697	Value '%s' set
680	ServerMain: %d ports, server is up	698	Value renamed
681	ServerName = %s	699	Value retrieved
682	Set CMOS Password	700	webserver.exe
683	Set Current Date and Time	701	Administrators
684	Set DateTime	702	Create SubKey
685	Set File DateTime	703	ThreadingModel
686	Set Screen Saver`s Password	704	Value set
687	Share added successfully	705	RELEASE

Appendix C

Presence ofAPIin Malware Dataset

Ser	API	%age of Occurrence in Malwares (P)
1.	VirtualAlloc	88.50
2.	VirtualFree	86.50
3.	CreateFileA	87.50
4.	ReadFile	89.50
5.	SetFilePointer	89.00
6.	RtlUnwind	81.50
7.	GetModuleHandleA	90.50
8.	CreateThread	82.00
9.	SetEndOfFile	78.00
10.	FindFirstFileA	76.50
11.	GetFileSize	81.50
12.	GetVersionExA	82.50
13.	FindClose	77.50
14.	GetVersion	78.00
15.	RegOpenKeyExA	82.00
16.	TlsGetValue	77.00
17.	TlsSetValue	77.00
18.	ExitProcess	87.00
19.	RegQueryValueExA	81.50
20.	DestroyWindow	72.00
21.	GetFileType	83.50
22.	lstrlenA	78.00
23.	RaiseException	72.50
24.	SetErrorMode	70.00
25.	WriteFile	93.50
26.	PostMessageA	70.50
27.	SendMessageA	73.50
28.	DefWindowProcA	71.00
29.	DispatchMessageA	72.00
30.	InitializeCriticalSection	77.00
31.	CreateWindowExA	69.50
32.	GetDiskFreeSpaceA	67.00
33.	GetLocaleInfoA	79.00
34.	LoadLibraryA	90.50

Ser	API	%age of Occurrence in Malwares (P)
35.	ShowWindow	70.00
36.	WaitForSingleObject	85.50
37.	EnumWindows	66.00
38.	GetModuleFileNameA	91.00
39.	PeekMessageA	69.50
40.	RegCloseKey	90.50
41.	RegisterClassA	68.00
42.	WideCharToMultiByte	85.50
43.	GetCommandLineA	82.00
44.	LocalAlloc	70.00
45.	lstrcpyA	67.00
46.	MessageBoxA	83.50
47.	SystemParametersInfoA	65.00
48.	CallNextHookEx	64.00
49.	FindWindowA	63.50
50.	FreeLibrary	85.00
1554.	RegOpenKeyExW	8.50
1555.	DllCanUnloadNow	7.00
1556.	DllGetClassObject	7.50
1557.	FlsAlloc	7.50
1558.	FlsFree	7.50
1559.	FlsGetValue	7.50
1560.	FlsSetValue	7.50
1561.	GetVersionExW	6.00
1562.	InterlockedCompareExchange	2.00
1563.	RegSetValueExW	6.50
1564.	DllRegisterServer	4.50
1565.	GetCurrentProcess	55.50
1566.	InitializeCriticalSectionAndSpinCount	8.50
1567.	GetModuleHandleW	10.00
1568.	IsDebuggerPresent	7.00
1569.	SetUnhandledExceptionFilter	12.00
1570.	QueryPerformanceCounter	20.50
1571.	GetSystemTimeAsFileTime	14.50

Appendix D

Presence of Mal Strings in Malware Dataset

Ser	Mal Strings	%age of Occurrence in Malwares (P)
1.	OWNER	71.00
2.	Password	66.50
3.	HOST	75.50
4.	DISABLED	64.00
5.	File not found	62.50
6.	Stack overflow	60.50
7.	WinSock	61.50
8.	Not Found	67.50
9.	Too many open files	60.50
10.	Division by zero	57.00
11.	SysUtils	57.00
12.	Remote	63.00
13.	shared	60.00
14.	SHUTDOWN	62.50
15.	TIMER	69.00
16.	File access denied	55.00
17.	Floating point division by zero	55.00
18.	Floating point overflow	55.00
19.	Floating point underflow	55.00
20.	No argument for format '%s'	55.00
21.	Privileged instruction	55.00
22.	Read beyond end of file	55.00
23.	Disconnect	59.50
24.	SERVER	79.50
25.	Stream	74.50
26.	Stream read error	53.00
27.	Stream write error	53.00
28.	Class %s not found	52.00
29.	Commctrl	52.00
30.	RegisterAutomation	51.50
31.	Resource %s not found	51.50
32.	WindowState	52.50
33.	WSocket	50.50
34.	Sender	54.00

Ser	Mal Strings	%age of Occurrence in Malwares (P)
35.	Username	51.50
36.	Deleting all files of current folder	49.00
37.	On connect	50.00
38.	REGISTER	81.50
39.	Locked	58.50
40.	Passwords	47.50
41.	ScktComp	47.00
42.	UseDockManager	46.00
43.	HideSelection	44.50
44.	SOCKS	46.00
45.	No address specified	44.00
46.	UrlMon	45.00
47.	FocusControl	40.50
48.	PasswordChar	40.50
49.	CreateKey	50.00
50.	Send message	47.00
688	Shared device mapped successfully	0.00
689	SMTP_relay	0.00
690	SMTP_relay = %s	0.00
691	TCP_ports	0.00
692	TCP_ports = %s	0.00
693	There is no screensaver password	0.00
694	Total files	0.00
695	Transfer finished	0.00
696	Value '%s' deleted	0.00
697	Value '%s' set	0.00
698	Value renamed	0.00
699	Value retrieved	0.00
700	webserver.exe	0.00
701	Administrators	0.00
702	Create SubKey	0.00
703	ThreadingModel	6.00
704	Value set	0.50
705	RELEASE	52.50

Appendix E

Presence ofAPIin Benign Dataset

Ser	API	%age of Occurrence in Benign Dataset (Q)
1.	VirtualAlloc	3.50
2.	VirtualFree	2.50
3.	CreateFileA	5.50
4.	ReadFile	8.50
5.	SetFilePointer	8.50
6.	RtlUnwind	1.50
7.	GetModuleHandleA	11.50
8.	CreateThread	5.50
9.	SetEndOfFile	1.00
10.	FindFirstFileA	1.00
11.	GetFileSize	5.50
12.	GetVersionExA	6.50
13.	FindClose	3.00
14.	GetVersion	4.00
15.	RegOpenKeyExA	9.50
16.	TlsGetValue	4.50
17.	TlsSetValue	4.50
18.	ExitProcess	15.00
19.	RegQueryValueExA	11.00
20.	DestroyWindow	2.00
21.	GetFileType	13.50
22.	lstrlenA	8.50
23.	RaiseException	3.00
24.	SetErrorMode	0.50
25.	WriteFile	24.00
26.	PostMessageA	2.00
27.	SendMessageA	4.50
28.	DefWindowProcA	3.50
29.	DispatchMessageA	4.00
30.	InitializeCriticalSection	9.00
31.	CreateWindowExA	2.50
32.	GetDiskFreeSpaceA	0.00
33.	GetLocaleInfoA	12.50
34.	LoadLibraryA	23.50

Ser	API	%age of Occurrence in Benign Dataset (Q)
35.	ShowWindow	3.50
36.	WaitForSingleObject	19.00
37.	EnumWindows	0.50
38.	GetModuleFileNameA	25.00
39.	PeekMessageA	3.50
40.	RegCloseKey	24.50
41.	RegisterClassA	2.00
42.	WideCharToMultiByte	19.50
43.	GetCommandLineA	17.00
44.	LocalAlloc	5.00
45.	lstrcpyA	2.00
46.	MessageBoxA	19.00
47.	SystemParametersInfoA	0.00
48.	CallNextHookEx	0.50
49.	FindWindowA	0.00
50.	FreeLibrary	21.00
1554.	RegOpenKeyExW	12.50
1555.	DllCanUnloadNow	12.00
1556.	DllGetClassObject	12.00
1557.	FlsAlloc	12.00
1558.	FlsFree	12.00
1559.	FlsGetValue	12.00
1560.	FlsSetValue	12.00
1561.	GetVersionExW	11.00
1562.	InterlockedCompareExchange	6.50
1563.	RegSetValueExW	11.00
1564.	DllRegisterServer	10.00
1565.	GetCurrentProcess	63.50
1566.	InitializeCriticalSectionAndSpinCount	16.00
1567.	GetModuleHandleW	23.00
1568.	IsDebuggerPresent	34.00
1569.	SetUnhandledExceptionFilter	58.50
1570.	QueryPerformanceCounter	80.00
1571.	GetSystemTimeAsFileTime	80.00

Appendix F

Presence of Mal Strings in Benign Dataset

Ser	Mal Strings	%age of Occurrence in Benign Dataset (Q)
1.	OWNER	5.00
2.	Password	2.00
3.	HOST	12.50
4.	DISABLED	2.00
5.	File not found	1.50
6.	Stack overflow	0.00
7.	WinSock	1.50
8.	Not Found	9.00
9.	Too many open files	1.50
10.	Division by zero	0.00
11.	SysUtils	0.00
12.	Remote	7.00
13.	shared	4.50
14.	SHUTDOWN	7.00
15.	TIMER	13.00
16.	File access denied	0.00
17.	Floating point division by zero	0.00
18.	Floating point overflow	0.00
19.	Floating point underflow	0.00
20.	No argument for format '%s'	0.00
21.	Privileged instruction	0.00
22.	Read beyond end of file	0.00
23.	Disconnect	5.50
24.	SERVER	25.50
25.	Stream	20.50
26.	Stream read error	0.00
27.	Stream write error	0.00
28.	Class %s not found	0.00
29.	Commctrl	0.00
30.	RegisterAutomation	0.00
31.	Resource %s not found	0.00
32.	WindowState	0.50
33.	WSocket	0.00
34.	Sender	4.00

Ser	Mal Strings	%age of Occurrence in Benign Dataset (Q)
35.	Username	2.00
36.	Deleting all files of current folder	0.00
37.	On connect	1.00
38.	REGISTER	33.00
39.	Locked	12.00
40.	Passwords	0.50
41.	ScktComp	0.00
42.	UseDockManager	0.00
43.	HideSelection	0.00
44.	SOCKS	1.00
45.	No address specified	0.00
46.	UrlMon	1.00
47.	FocusControl	0.00
48.	PasswordChar	0.00
49.	CreateKey	10.00
50.	Send message	7.00
688	Shared device mapped successfully	0.00
689	SMTP_relay	0.00
690	SMTP_relay = %s	0.00
691	TCP_ports	0.00
692	TCP_ports = %s	0.00
693	There is no screensaver password	0.00
694	Total files	0.00
695	Transfer finished	0.00
696	Value '%s' deleted	0.00
697	Value '%s' set	0.00
698	Value renamed	0.00
699	Value retrieved	0.00
700	webserver.exe	0.00
701	Administrators	0.50
702	Create SubKey	0.50
703	ThreadingModel	6.50
704	Value set	4.50
705	RELEASE	59.00

Appendix G

Difference Between Presence of API in Malware & Benign Dataset

Ser	API	P	Q	Δ = P-Q
1.	VirtualAlloc	88.50	3.50	85.00
2.	VirtualFree	86.50	2.50	84.00
3.	CreateFileA	87.50	5.50	82.00
4.	ReadFile	89.50	8.50	81.00
5.	SetFilePointer	89.00	8.50	80.50
6.	RtlUnwind	81.50	1.50	80.00
7.	GetModuleHandleA	90.50	11.50	79.00
8.	CreateThread	82.00	5.50	76.50
9.	SetEndOfFile	78.00	1.00	77.00
10.	FindFirstFileA	76.50	1.00	75.50
11.	GetFileSize	81.50	5.50	76.00
12.	GetVersionExA	82.50	6.50	76.00
13.	FindClose	77.50	3.00	74.50
14.	GetVersion	78.00	4.00	74.00
15.	RegOpenKeyExA	82.00	9.50	72.50
16.	TlsGetValue	77.00	4.50	72.50
17.	TlsSetValue	77.00	4.50	72.50
18.	ExitProcess	87.00	15.00	72.00
19.	RegQueryValueExA	81.50	11.00	70.50
20.	DestroyWindow	72.00	2.00	70.00
21.	GetFileType	83.50	13.50	70.00
22.	lstrlenA	78.00	8.50	69.50
23.	RaiseException	72.50	3.00	69.50
24.	SetErrorMode	70.00	0.50	69.50
25.	WriteFile	93.50	24.00	69.50
26.	PostMessageA	70.50	2.00	68.50
27.	SendMessageA	73.50	4.50	69.00
28.	DefWindowProcA	71.00	3.50	67.50
29.	DispatchMessageA	72.00	4.00	68.00
30.	InitializeCriticalSection	77.00	9.00	68.00
31.	CreateWindowExA	69.50	2.50	67.00
32.	GetDiskFreeSpaceA	67.00	0.00	67.00
33.	GetLocaleInfoA	79.00	12.50	66.50
34.	LoadLibraryA	90.50	23.50	67.00

Ser	API	P	Q	Δ = P-Q
35.	ShowWindow	70.00	3.50	66.50
36.	WaitForSingleObject	85.50	19.00	66.50
37.	EnumWindows	66.00	0.50	65.50
38.	GetModuleFileNameA	91.00	25.00	66.00
39.	PeekMessageA	69.50	3.50	66.00
40.	RegCloseKey	90.50	24.50	66.00
41.	RegisterClassA	68.00	2.00	66.00
42.	WideCharToMultiByte	85.50	19.50	66.00
43.	GetCommandLineA	82.00	17.00	65.00
44.	LocalAlloc	70.00	5.00	65.00
45.	lstrcpyA	67.00	2.00	65.00
46.	MessageBoxA	83.50	19.00	64.50
47.	SystemParametersInfoA	65.00	0.00	65.00
48.	CallNextHookEx	64.00	0.50	63.50
49.	FindWindowA	63.50	0.00	63.50
50.	FreeLibrary	85.00	21.00	64.00
1554.	RegOpenKeyExW	8.50	12.50	-4.00
1555.	DllCanUnloadNow	7.00	12.00	-5.00
1556.	DllGetClassObject	7.50	12.00	-4.50
1557.	FlsAlloc	7.50	12.00	-4.50
1558.	FlsFree	7.50	12.00	-4.50
1559.	FlsGetValue	7.50	12.00	-4.50
1560.	FlsSetValue	7.50	12.00	-4.50
1561.	GetVersionExW	6.00	11.00	-5.00
1562.	InterlockedCompareExchange	2.00	6.50	-4.50
1563.	RegSetValueExW	6.50	11.00	-4.50
1564.	DllRegisterServer	4.50	10.00	-5.50
1565.	GetCurrentProcess	55.50	63.50	-8.00
1566.	InitializeCriticalSectionAndSpinCount	8.50	16.00	-7.50
1567.	GetModuleHandleW	10.00	23.00	-13.00
1568.	IsDebuggerPresent	7.00	34.00	-27.00
1569.	SetUnhandledExceptionFilter	12.00	58.50	-46.50
1570.	QueryPerformanceCounter	20.50	80.00	-59.50
1571.	GetSystemTimeAsFileTime	14.50	80.00	-65.50

Appendix H

Difference Between Presence of Mal Strings in Malware & Benign Dataset

Ser	Mal Strings	P	Q	Δ = P-Q
1.	OWNER	71.00	5.00	66.00
2.	Password	66.50	2.00	64.50
3.	HOST	75.50	12.50	63.00
4.	DISABLED	64.00	2.00	62.00
5.	File not found	62.50	1.50	61.00
6.	Stack overflow	60.50	0.00	60.50
7.	WinSock	61.50	1.50	60.00
8.	Not Found	67.50	9.00	58.50
9.	Too many open files	60.50	1.50	59.00
10.	Division by zero	57.00	0.00	57.00
11.	SysUtils	57.00	0.00	57.00
12.	Remote	63.00	7.00	56.00
13.	shared	60.00	4.50	55.50
14.	SHUTDOWN	62.50	7.00	55.50
15.	TIMER	69.00	13.00	56.00
16.	File access denied	55.00	0.00	55.00
17.	Floating point division by zero	55.00	0.00	55.00
18.	Floating point overflow	55.00	0.00	55.00
19.	Floating point underflow	55.00	0.00	55.00
20.	No argument for format '%s'	55.00	0.00	55.00
21.	Privileged instruction	55.00	0.00	55.00
22.	Read beyond end of file	55.00	0.00	55.00
23.	Disconnect	59.50	5.50	54.00
24.	SERVER	79.50	25.50	54.00
25.	Stream	74.50	20.50	54.00
26.	Stream read error	53.00	0.00	53.00
27.	Stream write error	53.00	0.00	53.00
28.	Class %s not found	52.00	0.00	52.00
29.	Commctrl	52.00	0.00	52.00
30.	RegisterAutomation	51.50	0.00	51.50
31.	Resource %s not found	51.50	0.00	51.50
32.	WindowState	52.50	0.50	52.00
33.	WSocket	50.50	0.00	50.50
34.	Sender	54.00	4.00	50.00

Ser	Mal Strings	P	Q	Δ = P-Q
35.	Username	51.50	2.00	49.50
36.	Deleting all files of current folder	49.00	0.00	49.00
37.	On connect	50.00	1.00	49.00
38.	REGISTER	81.50	33.00	48.50
39.	Locked	58.50	12.00	46.50
40.	Passwords	47.50	0.50	47.00
41.	ScktComp	47.00	0.00	47.00
42.	UseDockManager	46.00	0.00	46.00
43.	HideSelection	44.50	0.00	44.50
44.	SOCKS	46.00	1.00	45.00
45.	No address specified	44.00	0.00	44.00
46.	UrlMon	45.00	1.00	44.00
47.	FocusControl	40.50	0.00	40.50
48.	PasswordChar	40.50	0.00	40.50
49.	CreateKey	50.00	10.00	40.00
50.	Send message	47.00	7.00	40.00
688	Shared device mapped successfully	0.00	0.00	0.00
689	SMTP_relay	0.00	0.00	0.00
690	SMTP_relay = %s	0.00	0.00	0.00
691	TCP_ports	0.00	0.00	0.00
692	TCP_ports = %s	0.00	0.00	0.00
693	There is no screensaver password	0.00	0.00	0.00
694	Total files	0.00	0.00	0.00
695	Transfer finished	0.00	0.00	0.00
696	Value '%s' deleted	0.00	0.00	0.00
697	Value '%s' set	0.00	0.00	0.00
698	Value renamed	0.00	0.00	0.00
699	Value retrieved	0.00	0.00	0.00
700	webserver.exe	0.00	0.00	0.00
701	Administrators	0.00	0.50	-0.50
702	Create SubKey	0.00	0.50	-0.50
703	ThreadingModel	6.00	6.50	-0.50
704	Value set	0.50	4.50	-4.00
705	RELEASE	52.50	59.00	-6.50

Appendix I

Selection of Features in API Feature Set

Ser	API	Δ = P-Q	Weight	Result
1.	VirtualAlloc	85.00	85	Selected
2.	VirtualFree	84.00	84	Selected
3.	CreateFileA	82.00	82	Selected
4.	ReadFile	81.00	81	Selected
5.	SetFilePointer	80.50	81	Selected
6.	RtlUnwind	80.00	80	Selected
7.	GetModuleHandleA	79.00	79	Selected
8.	CreateThread	76.50	77	Selected
9.	SetEndOfFile	77.00	77	Selected
10.	FindFirstFileA	75.50	76	Selected
11.	GetFileSize	76.00	76	Selected
12.	GetVersionExA	76.00	76	Selected
13.	FindClose	74.50	75	Selected
14.	GetVersion	74.00	74	Selected
15.	RegOpenKeyExA	72.50	73	Selected
16.	TlsGetValue	72.50	73	Selected
17.	TlsSetValue	72.50	73	Selected
18.	ExitProcess	72.00	72	Selected
19.	RegQueryValueExA	70.50	71	Selected
20.	DestroyWindow	70.00	70	Selected
21.	GetFileType	70.00	70	Selected
22.	lstrlenA	69.50	70	Selected
23.	RaiseException	69.50	70	Selected
24.	SetErrorMode	69.50	70	Selected
25.	WriteFile	69.50	70	Selected
26.	PostMessageA	68.50	69	Selected
27.	SendMessageA	69.00	69	Selected
28.	DefWindowProcA	67.50	68	Selected
29.	DispatchMessageA	68.00	68	Selected
30.	InitializeCriticalSection	68.00	68	Selected
31.	CreateWindowExA	67.00	67	Selected
32.	GetDiskFreeSpaceA	67.00	67	Selected
33.	GetLocaleInfoA	66.50	67	Selected
34.	LoadLibraryA	67.00	67	Selected

Ser	API	Δ = P-Q	Weight	Result
35.	ShowWindow	66.50	67	Selected
36.	WaitForSingleObject	66.50	67	Selected
37.	EnumWindows	65.50	66	Selected
38.	GetModuleFileNameA	66.00	66	Selected
39.	PeekMessageA	66.00	66	Selected
40.	RegCloseKey	66.00	66	Selected
41.	RegisterClassA	66.00	66	Selected
42.	WideCharToMultiByte	66.00	66	Selected
43.	GetCommandLineA	65.00	65	Selected
44.	LocalAlloc	65.00	65	Selected
45.	lstrcpyA	65.00	65	Selected
46.	MessageBoxA	64.50	65	Selected
47.	SystemParametersInfoA	65.00	65	Selected
48.	CallNextHookEx	63.50	64	Selected
49.	FindWindowA	63.50	64	Selected
50.	FreeLibrary	64.00	64	Selected
1554.	RegOpenKeyExW	-4.00	-4	Rejected
1555.	DllCanUnloadNow	-5.00	-5	Rejected
1556.	DllGetClassObject	-4.50	-5	Rejected
1557.	FlsAlloc	-4.50	-5	Rejected
1558.	FlsFree	-4.50	-5	Rejected
1559.	FlsGetValue	-4.50	-5	Rejected
1560.	FlsSetValue	-4.50	-5	Rejected
1561.	GetVersionExW	-5.00	-5	Rejected
1562.	InterlockedCompareExchange	-4.50	-5	Rejected
1563.	RegSetValueExW	-4.50	-5	Rejected
1564.	DllRegisterServer	-5.50	-6	Rejected
1565.	GetCurrentProcess	-8.00	-8	Rejected
1566.	InitializeCriticalSectionAndSpinCount	-7.50	-8	Rejected
1567.	GetModuleHandleW	-13.00	-13	Rejected
1568.	IsDebuggerPresent	-27.00	-27	Rejected
1569.	SetUnhandledExceptionFilter	-46.50	-47	Rejected
1570.	QueryPerformanceCounter	-59.50	-60	Rejected
1571.	GetSystemTimeAsFileTime	-65.50	-66	Rejected

Appendix J

Selection of Features in Mal Strings Feature Set

Ser	Mal Strings	Δ = P-Q	Weight	Result
1.	OWNER	66.00	66	Selected
2.	Password	64.50	65	Selected
3.	HOST	63.00	63	Selected
4.	DISABLED	62.00	62	Selected
5.	File not found	61.00	61	Selected
6.	Stack overflow	60.50	61	Selected
7.	WinSock	60.00	60	Selected
8.	Not Found	58.50	59	Selected
9.	Too many open files	59.00	59	Selected
10.	Division by zero	57.00	57	Selected
11.	SysUtils	57.00	57	Selected
12.	Remote	56.00	56	Selected
13.	shared	55.50	56	Selected
14.	SHUTDOWN	55.50	56	Selected
15.	TIMER	56.00	56	Selected
16.	File access denied	55.00	55	Selected
17.	Floating point division by zero	55.00	55	Selected
18.	Floating point overflow	55.00	55	Selected
19.	Floating point underflow	55.00	55	Selected
20.	No argument for format '%s'	55.00	55	Selected
21.	Privileged instruction	55.00	55	Selected
22.	Read beyond end of file	55.00	55	Selected
23.	Disconnect	54.00	54	Selected
24.	SERVER	54.00	54	Selected
25.	Stream	54.00	54	Selected
26.	Stream read error	53.00	53	Selected
27.	Stream write error	53.00	53	Selected
28.	Class %s not found	52.00	52	Selected
29.	Commctrl	52.00	52	Selected
30.	RegisterAutomation	51.50	52	Selected
31.	Resource %s not found	51.50	52	Selected
32.	WindowState	52.00	52	Selected
33.	WSocket	50.50	51	Selected
34.	Sender	50.00	50	Selected

Ser	Mal Strings	Δ = P-Q	Weight	Result
35.	Username	49.50	50	Selected
36.	Deleting all files of current folder	49.00	49	Selected
37.	On connect	49.00	49	Selected
38.	REGISTER	48.50	49	Selected
39.	Locked	46.50	47	Selected
40.	Passwords	47.00	47	Selected
41.	ScktComp	47.00	47	Selected
42.	UseDockManager	46.00	46	Selected
43.	HideSelection	44.50	45	Selected
44.	SOCKS	45.00	45	Selected
45.	No address specified	44.00	44	Selected
46.	UrlMon	44.00	44	Selected
47.	FocusControl	40.50	41	Selected
48.	PasswordChar	40.50	41	Selected
49.	CreateKey	40.00	40	Selected
50.	Send message	40.00	40	Selected
688	Shared device mapped successfully	0.00	0	Rejected
689	SMTP_relay	0.00	0	Rejected
690	SMTP_relay = %s	0.00	0	Rejected
691	TCP_ports	0.00	0	Rejected
692	TCP_ports = %s	0.00	0	Rejected
693	There is no screensaver password	0.00	0	Rejected
694	Total files	0.00	0	Rejected
695	Transfer finished	0.00	0	Rejected
696	Value '%s' deleted	0.00	0	Rejected
697	Value '%s' set	0.00	0	Rejected
698	Value renamed	0.00	0	Rejected
699	Value retrieved	0.00	0	Rejected
700	webserver.exe	0.00	0	Rejected
701	Administrators	-0.50	-1	Rejected
702	Create SubKey	-0.50	-1	Rejected
703	ThreadingModel	-0.50	-1	Rejected
704	Value set	-4.00	-4	Rejected
705	RELEASE	-6.50	-7	Rejected

Appendix K

Malware Data Set Score for API and Mal Strings (Ascending Order)

Malware Sample	API Score	Mal String Score
1	336	49
2	1024	55
3	1499	106
4	1535	124
5	1538	141
6	1538	153
7	1586	214
8	1836	227
9	1960	239
10	**1983**	**240**
11	1998	240
12	2110	247
13	2150	251
14	2169	251
15	2270	262
16	2304	289
17	2308	312
18	2356	322
19	2390	345
20	2499	345
21	2542	348
22	2703	361
23	2803	362
24	2809	375
25	2869	376
26	2972	388
27	3059	388
28	3076	407
29	3276	408
30	3276	426
31	3385	443
32	3447	451
33	3453	455
34	3569	456

Malware Sample	API Score	Mal String Score
35	3773	456
36	3893	484
37	3956	506
38	4023	507
39	4050	510
40	4103	541
41	4103	564
42	4283	569
43	4310	619
44	4348	636
45	4348	637
46	4403	656
47	4574	661
48	4705	664
49	4729	677
50	4777	691
51	4888	727
52	4901	727
53	4969	739
54	5085	789
55	5085	844
56	5128	844
57	5128	869
58	5151	1016
59	5354	1085
60	5412	1085
61	5924	1210
62	5947	1252
63	6014	1308
64	6030	1342
65	6065	1342
66	6073	1374
67	6099	1413
68	6101	1455

Malware Sample	API Score	Mal String Score
69	6213	1460
70	6348	1461
71	6744	1465
72	6772	1476
73	6804	1490
74	6859	1490
75	6859	1495
76	6942	1500
77	7010	1525
78	7358	1540
79	7358	1552
80	7358	1656
81	7689	1660
82	7728	1729
83	7788	1743
84	7788	1790
85	7833	1795
86	7838	1799
87	8103	1845
88	8229	1926
89	8248	1931
90	8321	1963
91	8495	1970
92	9163	1970
93	11069	2010
94	13037	2053
95	17017	2090
96	17298	2113
97	17420	2152
98	17663	2168
99	18577	2168
100	19080	2168
101	19863	2191
102	19908	2317

Malware Sample	API Score	Mal String Score
103	20178	2356
104	20678	2500
105	20816	2541
106	21084	2559
107	21362	2625
108	21458	2643
109	21482	2651
110	21532	2725
111	21544	2731
112	21600	2757
113	21858	2840
114	21878	2876
115	21933	2981
116	21944	2984
117	21944	3034
118	21946	3045
119	21972	3045
120	21981	3045
121	21989	3046
122	22033	3092
123	22101	3110
124	22101	3136
125	22101	3138
126	22101	3139
127	22117	3198
128	22238	3230
129	22370	3270
130	22370	3283
131	22440	3310
132	22485	3314
133	22494	3317
134	22528	3324
135	22528	3342
136	22535	3374

Malware Sample	API Score	Mal String Score
137	22566	3415
138	22576	3417
139	22576	3483
140	22576	3485
141	22606	3526
142	22678	3554
143	22838	3626
144	22876	3626
145	22913	3668
146	23002	3701
147	23005	3701
148	23104	3747
149	23111	3756
150	23241	3773
151	23359	3791
152	23579	3791
153	23659	3804
154	23796	3820
155	23806	3841
156	23806	3842
157	23806	3894
158	23823	3960
159	23872	3960
160	24087	4038
161	24202	4076
162	24223	4094
163	24363	4094
164	24388	4100
165	24463	4136
166	24571	4285
167	24651	4441
168	24730	4488
169	24730	4489
170	24754	4563

Malware Sample	API Score	Mal String Score
171	24806	4576
172	24868	4643
173	25023	4643
174	25065	4841
175	25065	4868
176	25065	4870
177	25065	4930
178	25065	4941
179	25065	4946
180	25065	4955
181	25065	4955
182	25122	4970
183	25234	4970
184	25234	4970
185	25234	4970
186	25257	4970
187	25486	4983
188	25580	4983
189	25615	5006
190	25615	5021
191	25615	5041
192	25615	5041
193	25615	5046
194	25615	5078
195	25650	5280
196	25821	5280
197	25826	5354
198	25826	5354
199	25850	5354
200	25935	5354

Total Score	2906159	484255
Average	14531	2421

Appendix L

Benign Data Set Score for API and Mal Strings (Descending Order)

Benign Sample	API Score	Mal String Score
1	5415	698
2	5415	689
3	5415	643
4	3416	620
5	3394	572
6	2822	571
7	2627	478
8	2580	423
9	2178	404
10	2140	384
11	2060	377
12	1806	367
13	1760	352
14	1677	344
15	1627	333
16	1597	325
17	1594	316
18	1548	315
19	1548	310
20	1497	305
21	1446	304
22	1427	304
23	1427	300
24	1427	300
25	1427	293
26	1427	293
27	1427	293
28	1427	287
29	1427	285
30	1427	281
31	1427	281
32	1427	281
33	1427	277
34	1427	274

Benign Sample	API Score	Mal String Score
35	1427	270
36	1427	269
37	1427	268
38	1427	268
39	1427	268
40	1427	268
41	1427	268
42	1427	268
43	1427	268
44	1394	268
45	1394	262
46	1272	257
47	1207	248
48	1181	245
49	1160	241
50	1143	235
51	1102	233
52	1093	226
53	1009	219
54	1009	218
55	1009	217
56	1009	216
57	1009	213
58	1009	213
59	1009	211
60	1009	203
61	999	203
62	980	199
63	977	198
64	974	195
65	893	195
66	870	195
67	869	194
68	862	192

Benign Sample	API Score	Mal String Score
69	858	187
70	858	175
71	854	174
72	813	166
73	774	163
74	756	160
75	755	158
76	755	157
77	751	157
78	744	157
79	724	157
80	701	157
81	699	155
82	697	155
83	691	155
84	691	155
85	689	155
86	689	153
87	667	153
88	639	150
89	632	147
90	599	145
91	487	135
92	472	135
93	443	135
94	443	131
95	433	130
96	431	129
97	431	129
98	428	123
99	390	120
100	363	116
101	363	113
102	359	113

Benign Sample	API Score	Mal String Score
103	347	113
104	347	113
105	315	113
106	312	113
107	307	113
108	294	113
109	266	113
110	260	112
111	252	112
112	244	110
113	238	108
114	205	108
115	201	108
116	192	108
117	166	108
118	154	108
119	154	108
120	154	103
121	150	100
122	150	99
123	149	99
124	146	98
125	140	95
126	133	91
127	133	91
128	129	91
129	129	89
130	129	89
131	125	89
132	125	89
133	125	89
134	122	89
135	118	89
136	103	89

Benign Sample	API Score	Mal String Score
137	93	89
138	91	89
139	90	89
140	88	88
141	88	82
142	79	82
143	79	82
144	77	80
145	73	72
146	73	66
147	73	66
148	72	63
149	72	63
150	71	63
151	62	63
152	57	59
153	57	59
154	54	59
155	49	59
156	47	59
157	47	59
158	47	59
159	44	59
160	35	59
161	34	56
162	34	54
163	34	54
164	34	49
165	34	49
166	34	49
167	34	47
168	24	47
169	24	39
170	24	39

Benign Sample	API Score	Mal String Score
171	22	39
172	22	39
173	22	35
174	22	35
175	22	35
176	22	35
177	22	35
178	22	35
179	22	35
180	22	35
181	22	35
182	22	35
183	22	24
184	22	23
185	22	12
186	22	0
187	22	0
188	22	0
189	22	0
190	22	0
191	22	0
192	22	0
193	22	0
194	22	0
195	22	0
196	22	0
197	22	0
198	5	0
199	2	0
200	0	0

Total Score	141520	31888
Average	708	159