



ADDIS ABABA UNIVERSITY
SCHOOL OF GRADUATE STUDIES
FACULTY OF TECHNOLOGY
ELECTRICAL AND COMPUTER ENGINEERING
DEPARTMENT

A WHITELIST BASED IMPLEMENTATION OF ANTIVIRUS- DEFINITION FILE
TO DETECT UNKNOWN MALICIOUS ACTIVITY

By
Yishak Ibrahim Omer

A thesis submitted to the school of Graduate studies of Addis Ababa
University in partial fulfillment of the requirements for the degree of
Masters of Science in Electrical and Computer Engineering
(Computer Engineering)

January, 2009
Addis Ababa, Ethiopia

ADDIS ABABA UNIVERSITY
SCHOOL OF GRADUATE STUDIES
FACULTY OF TECHNOLOGY
DEPARTMENT OF ELECTRICAL AND COMPUTER
ENGINEERING

**A WHITELIST BASED IMPLEMENTATION OF ANTIVIRUS-
DEFINITION FILE TO DETECT UNKNOWN MALICIOUS
ACTIVITY**

By

Yishak Ibrahim Omer

Advisor

Dr. Manoj V.N.V

ADDIS ABABA UNIVERSITY
SCHOOL OF GRADUATE STUDIES

Declaration

I, the undersigned student, declare that this thesis work is my original work, has not been presented for a degree in this or any other universities, and all sources of materials used for the thesis work have been fully acknowledged.

Name: Yishak Ibrahim Omer

Signature: _____

Place: Addis Ababa

Date of submission: January 21, 2009

This thesis has been submitted for examination with my approval as a university advisor.

Dr. Manoj V.N.V

Signature: _____

Advisor's Name

ADDIS ABABA UNIVERSITY
SCHOOL OF GRADUATE STUDIES

**A WHITELIST BASED IMPLEMENTATION OF ANTIVIRUS-
DEFINITION FILE TO DETECT UNKNOWN MALICIOUS
ACTIVITY**

By

Yishak Ibrahim Omer

FACULTY OF TECHNOLOGY

APPROVAL BY BOARD OF EXAMINERS

Chairman Dept. of
Graduate Committee

Signature

Dr. Manoj V.N.V
Advisor

Signature

Examiner

Signature

TABLE OF CONTENTS

List of Tables	viii
List of Equations	viii
List of Figures	ix
Acknowledgment	xi
Acronyms	x
Abstract	xii
CHAPTER I	1
Introduction	1
1.2 Background of the problem	2
1.3 Objective:	3
1.3.1 General objective	3
1.3.2 Specific objective	3
1.4 Summary of Activities Performed for the thesis	4
1.5 Organization of the Thesis	5
CHAPTER II	6
Types of Malwares and Methods of combating them	6
2.1 Types of Malware	6
2.1.1 Virus's	6
2.1.1.1 Boot Virus	6
2.1.1.2 Parasitic viruses/ File Infectors	7
2.1.1.3 Date viruses/ Logic Bombs/ Time Bomb	7
2.1.1.4 Macro Virus	7
2.1.1.5 Encrypted Virus	7
2.1.1.6 Polymorphic virus	8
2.1.1.7 Stealth Virus	8
2.1.2 Other Malware	8
2.1.2.1 Trojan Horse	8
2.1.2.2 Worms	8
2.1.2.3 Rootkits	9
2.1.2.4 Botnets	9
2.1.2.5 Spyware	9
2.1.2.6 Keyloggers	10
2.1.2.7 Adware	10
2.1.2.8 Clickbots	10
2.2 Types of Attacks	10
2.2.1 Social Engineering	10
2.2.2 Mass E-Mailers	11
2.2.3 Exploit on Software Vulnerabilities	11
2.2.4 Phishing	11
2.2.5 Pharming	12
2.3 Methods to Combat Malware	12
2.3.1 Traditional Blacklist Solutions	12

2.3.2 Advanced Blacklist Solutions	13
2.3.3 Intrusion Detection	14
2.3.4 The Whitelist Solution.....	15
2.4 Anti-Virus	16
2.4.1 Signature detection or Pattern Matching	16
2.4.2 X – Raying.....	16
2.4.3. Emulation	16
2.4.4 Frequency Analysis	17
2.4.5 Heuristics.....	17
2.5 Use of Machine Learning Techniques	17
2.5.1 Data Mining Approach.....	17
2.5.2 Neural Networks.....	18
2.5.3 Hidden Markov Models.....	19
Chapter III.....	21
Chronology and Case Studies of Selected Malwares	21
3.1 Milestones in Evolution of Malwares	21
3.2 Selected Case Studies of Notorious Malwares	25
3.2.2 The Code Red Worm.....	26
3.2.3 The Nimda Worm.....	27
3.2.4 The Blaster and SQL Slammer Worms	28
CHAPTER IV	31
Related Works.....	31
4.1 Computer Immunology	31
4.1.1 Researches at University of New Mexico	31
4.1.2 Researches at Technical University of Denmark	31
4.2 Computer Security	33
4.3 Malware Detection and Antivirus.....	33
CHAPTER V	36
WhiteList based System Proposal.....	36
5.1 Whitelist based definition file Proposal	36
5.2 Malwares Survival and Reproduction Interception.....	38
5.3 The registry and its use as a Whitelist parameter	39
5.3.1 The Registry.....	39
5.3.1.1 Structure of the Registry	39
5.3.1.2 Registry Storage Space	40
5.3.1.3 Predefined Keys.....	40
5.3.2 How the registry and File System can be used as a WhiteList	41
5.4 Experimental Setting and Tools.....	42
5.5 Sample Collection.....	43
CHAPTER VI.....	45
EXPERIMENTAL RESULTS, WHITELISTING ALGORITHM AND DISCUSSION.....	45
6.1 RESULTS	45
6.1.1 Registry Monitoring	45

6.1.2 Tracing application Activity Using the KrView Program.....	45
6.1.3 API TRACE	50
6.2 Algorithms for Whitelisting	52
6.2.1 WhiteListing Algorithm	53
6.2.2 Scanning Algorithm	54
6.2.3 I/O Algorithm	55
6.2.4 API Tracing	56
6.2.5 Decision Algorithm	57
6.3 Implemented WhiteList Definition File Detection comparisons	58
CHAPTER VII.....	59
CONCLUSION, RECOMMENDATION AND FUTURE WORK.....	59
Recommendation and Future Works	59
SUMMARY	60
APPENDIX.....	61
APPENDIX I: Top 10 Viruses in History.....	61
APPENDIX II : Registry and File Auto Start Locations considered in the thesis.....	66
APPENIX III : Ten steps to safer computing	70
APPENIX IV: Malwares and Benign Programs Used.....	72
Reference:	75

List of Tables

Table 5.1 Command Prompt Trace at 30s Snapshot Interval	49
Table 5.2 Solitaire Trace at 30s Snapshot Interval	49
Table 5.3 Virus_08's Trace at 30s Snapshot Interval	50
Table 5.4 File I/O API Tracing for Benign and Malware Programs.	52

List of Equations

Equation 5.1 Coefficient of Variance	46
Equation 5.2 File I/O API Call Running Frequency calculation	50

List of Figures

Figure 1.1 Summary of Thesis Activity.....	4
Figure 3.1The System Shutdown dialog box caused by the Blaster worm	29
Figure 6.1 I/O READ Operation Performed by Benign and Malware programs	47
Figure 6.2 I/O Write Operation Performed by Benign and Malware programs	48
Figure 6.3 I/O Other Operation Performed by Benign and Malware programs	48
Figure 6.4 File I/O API Trace for Benign and Malware Programs at 30s Snapshot	51
Figure 6.5 WhiteListing Algorithm	53
Figure 6.6 Scanning Algorithm.....	54
Figure 6.7 I/O Tracing Algorithm.....	55
Figure 6.8 API Tracing Algorithm.....	56
Figure 6.9 Decision Making Algorithm.....	57
Figure 6.10 Unknown Malicious Malware Detection among Different products	58

Acronyms

AIS	Artificial Immune System
ANSI	American National Standards Institute
API	Application Programming Interface
AV	Anti Virus
BIOS	Basic Input Output System
CD	Compact Disk
CPU	Central Processing Unit
CV	Coefficient of Variance
DCOM	Distributed Component Object Model
DDOS	Distributed Denial of Service
DLL	Dynamic Link Library
DNS	Domain Name Server
DOS	Disk Operating System
HMM	Hidden Markov Model
HTTP	Hyper Text Transfer Protocol
I/O	Input Output
IDS	Intrusion Detection System
IEEE	Institute of Electrical and Electronics Engineers
MBR	Master Boot Record
NGVCK	Next Generation Virus Creation Kit
OS	Operating System
PC	Personal Computer
PE	Portable Executable
PIN	Personal Identification Number
UDP	User Datagram Protocol
URL	Uniform Resource Locator
USB	Universal Serial Bus

Acknowledgment

The last two years of Postgraduate education culminating in this thesis would not have been possible without the help, support and advice of family, friends and colleagues. Most of all I like to thank my parents for supporting me in all aspects through the bad and good times. My friend's companionship was a lot more fun, joy and pleasing specially in times of stress and hard work where a break was desperately needed. Finally but not the least I would like to thank my advisor Dr. Manoj VNV for agreeing to work with me in this project where the topic was an area I have a lot of interest and where no work in the area has been done in the department before.

Abstract

The battle between malware and antivirus makers has been going on since the late 1980's. The malware makers are inventing and using different methods to infect, replicate and propagate to machines and the Malwares have been given classifications like Viruses, worms ,Trojans to name a few. In response the antivirus makes have come up with different detection mechanisms. Today blacklisting and Heuristics are the dominant technologies used by Anti-Virus (AV) Scanner engines and Databases. However, the sheer number of computer viruses and the shift in the internal design of computer Malwares in the last 6 years alone indicate that a new trend in dealing with malicious activities needs to be addressed.

In this thesis a whitelist based approach to detect unknown malicious activities is addressed. In this approach a window's Operating System registry settings and entries are used to build a whitelist profile on programs existing on the Personal Computer (PC). Latter on this information is used to identify new entries in the registry and this will be processed to identify them as malicious or benign using a statistical based scan engine. The engine uses suspected programs Input/output (I/O) Read, Write and Other Operation together with Application Programming Interface (API) Trace to classify it as malware or benign.

A user level scan engine was written and the engine was tested on 40 code generated malicious programs that include virus's, worms and Trojans and 30 benign programs resulting in high true positive detection rate and No false positive detection. The same sample was processed with commercial Antivirus Software's including Symantec endpoint, Avast, AVG and Kaspersky. The thesis detected 95 % of the malwares while the next nearest match was Symantec endpoint with 67 % detection rate followed by Symantec 10.0 with 38 %. The other product AVG has 11 % detection rate. Kaspersky and Avast Antivirus were not able to detect any of the malwares. The high detection rate of the thesis scan engine shows that the methods used can be integrated into a heuristics scan engine to achieve a high true positive detection rate of unknown malicious activities.

CHAPTER I

Introduction

Malicious software that infects a computer without the knowledge of the user is called a Malware. Malwares can be classified into Viruses, Worms, Spyware, Rootkit, Trojan horse, etc. Malicious programs infect a computer and then travel through the network connections, the Internet, or by carrying it on a removable medium such as a floppy disk, CD, or USB drive to infect other computers. The Internet serves to increase the vulnerability as the infection spreads not only across a local network, but to any computer across the world.

Motivations behind Malware attacks have been changing constantly over time. The first few malwares were simple machine language programs [1] classified as Viruses. The viruses had the ability to infect other executable code. When a user executes such code, the virus would take control of the system and infect more files. Such viruses were known as parasites as they would not totally destroy the running machine. These viruses were easy to detect and remove, as administrators only had to search for patterns of instructions to find instructions that matched an infection. These patterns were known as virus signatures. This method worked well till the virus writers started writing polymorphic viruses. Writing a complex routine that could change portions of it significantly and also retain virus like properties was a challenge, hence the writers chose for a compromise and wrote virus programs that were encrypted, with the encryption engine being different on every infection. The flaw with this method was that any encrypted code has to be decrypted in memory to execute.

The antivirus researchers used this and created CPU emulators that would run the code. This allowed detection of encrypted polymorphic viruses. After this came a generation of malware that would install a root kit, and save the virus programs as hidden and system files and remain elusive to most of the anti malware programs.

Today, malicious programs attempt to shut off the security processes such as Anti Virus, Firewall etc. running on the system they infect, hence making the host computer a sitting

target for all sorts of exploits. This marked a significant change in the tug of war between the viruses and the antivirus software. Till now any infection could be contained and cleaned by the antivirus after the arrival of the update. Once the malicious programs started targeting the antivirus products themselves, there would not be any further updates.

1.2 Background of the problem

Software such as anti-virus solutions and Firewalls offer some protection to users against attacks, however, they are not completely effective. Researchers have shown that there is no algorithm that can perfectly detect the presence of malicious code. The reason for this is that anti-virus relies on virus definitions and known behavioral patterns to identify malicious code. A code that is completely new in design is bound to effectively use the zero day exploit. That is the virus writers take advantage of the fact that all the machines around the world will be vulnerable to a security threat on the day the threat is created. This is because the anti-virus Software will have no definitions describing the code as malicious in nature [1].

Anti-virus and other security software also suffer from other vulnerabilities. The anti-virus process can be killed by any process in the system with administrator privileges, or it can also be infected by viruses, due to which the virus detection engine is rendered useless. A malicious program that obtains administrator privilege and manipulates other processes in the system is known as a Rootkit. [3]

The methodologies used by antivirus companies have undergone changes; however the latest trend of killing the antivirus process threatens to make their presence inconsequential. In addition malicious programs have started to patch definition files. There is an urgent need to address this issue.

1.3 Objective:

1.3.1 General objective

The general objective of this thesis is to study and evaluate algorithms and methods implemented in existing anti-virus software's to detect unknown malicious programs and enhance the signature based detection mechanism to include features other than the byte sequence of a malicious program for example WhiteLists. All analysis and tests will be for Windows systems.

1.3.2 Specific objective

The specific objective of this thesis includes:

- Studying and evaluating existing antivirus systems methods of detecting unknown malicious Malware against live computer Malwares.
- Studying different methods and algorithms in the area of Intrusion detection systems in both UNIX and Windows Platform and Analyze their relevance for whitelist in the windows environment.
- Studying different methods of Malware survival, replication, transmission and Infection methods and Analyze if a parameter that can be integrated in whitelist exists.
- Design an algorithm to implement whitelist and Identify Unknown malicious activity based on whitelist.
- Test the Implemented algorithm and methods against Live Malwares.

1.4 Summary of Activities Performed for the thesis

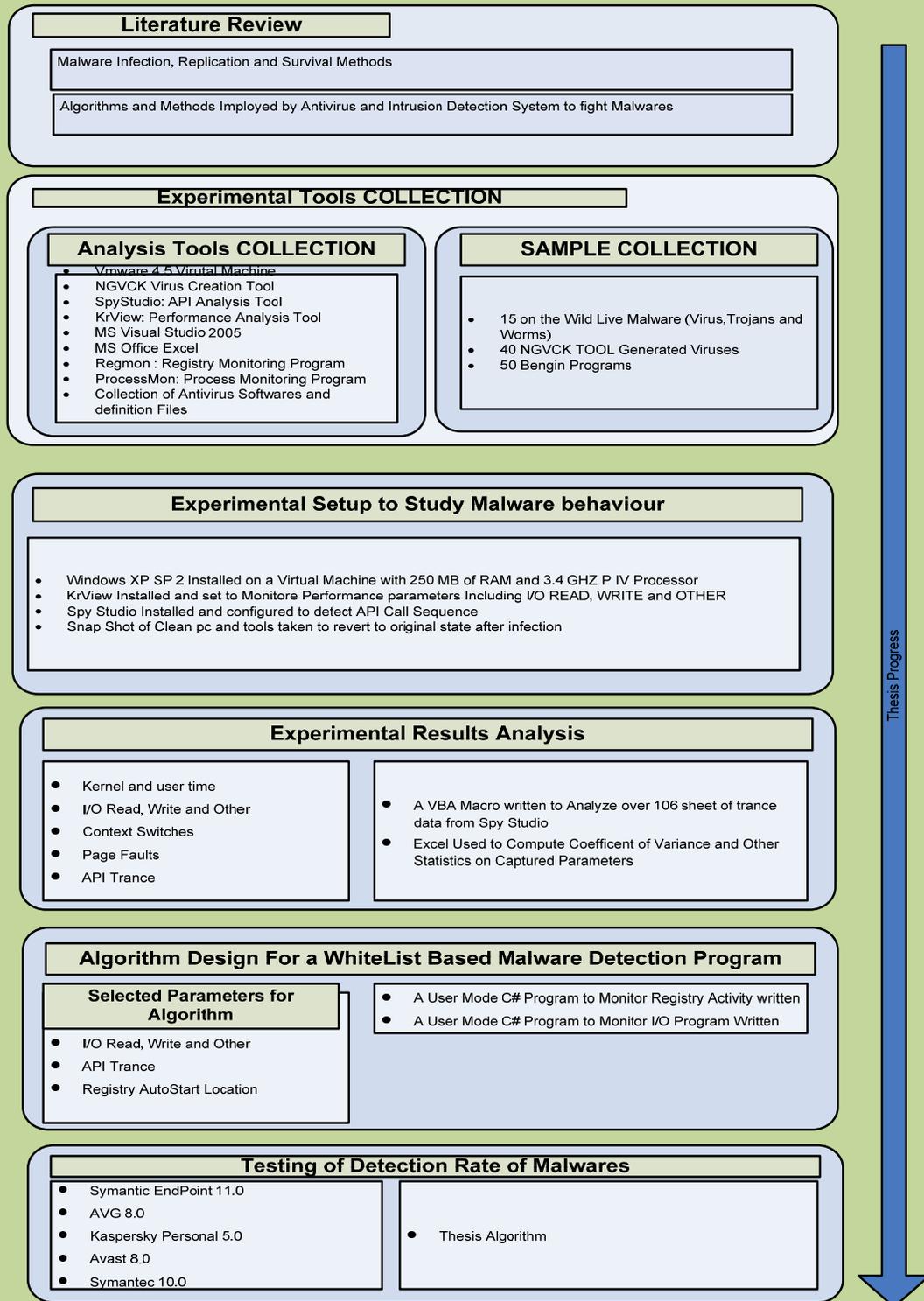


Figure 1.1 Summary of Thesis Activity

1.5 Organization of the Thesis

The thesis is organized into seven chapters. Chapter II Deals with historical background about Malwares, their classification and methods used by antivirus software makers to combat them. Chapter III gives an insight into the chronology of computer Malwares and discussion on a selected case study of malwares done by other researchers that caused havoc. Chapter IV discusses related works by other researchers in the area. Chapter V puts forward a proposal for a WhiteList based antivirus solution, experimental settings and Sample selection. In Chapter VI the results of experiments performed and algorithm proposal based on experimental results will be presented. Chapter VII closes the thesis by presenting conclusion, recommendation and future works.

CHAPTER II

Types of Malwares and Methods of combating them

Computer viruses are programs that are deliberately designed to interfere with computer operation, record, corrupt, or delete data, or spread themselves to other computers and throughout the Internet [1]. One of the first well known malware was the Internet worm which was released in 1988 [1, 11]. It broke into the system by means of flaws in the system software. It was not designed to do any damage, but was simply made to estimate the size of the internet. However, each computer started getting infected multiple times; the consequence was that the machines became very slow to the point that they could not be used.

Personal Computer Viruses first started appearing in the 1980's. In 1990 it was estimated that there were 500 virus programs. By 1996 the estimate rose to around 10,000. It was estimated that by 2002 there were 70,000 viruses [12]. Today the number of known computer virus is estimated to be around 103,000 [1]

The first few viruses were simple machine language programs. The viruses had the ability to infect other executable code. When a user would execute such code, the virus would take control of the system and infect more files. These viruses are known as parasites as they would not totally destroy the running machine. These viruses are easy to detect and remove, as administrators only had to search for patterns called virus signatures to identify infections.

2.1 Types of Malware

2.1.1 Virus's

2.1.1.1 Boot Virus

Boot sector viruses account for about 5 percent of known PC viruses [1]. These types of viruses operate by infecting the Master Boot Record (MBR) of a PC [26]. The MBR is a program that resides on the first sector of a hard disk; it runs every time a PC starts up and is responsible for loading the rest of the Operating System. This makes a Boot sector virus

extremely dangerous, although they are very few in number compared to viruses that infect files. Once loaded a boot sector virus resides in memory and is capable of infecting any drive placed on the system. They are very difficult to remove and it requires a bootable anti-virus disk to properly remove the virus.

2.1.1.2 Parasitic viruses/ File Infectors

This type of virus attaches itself onto files or executables [26], leaving the contents of the file unchanged. About 85 percent of all known viruses are of this type [1]. When a user runs the infected application or file, the virus code executes and copies itself into the memory. The code then attempts to spread itself onto other applications, and also any removable disks attached to the machine.

2.1.1.3 Date viruses/ Logic Bombs/ Time Bomb

These are types of viruses that reside in a machine and get triggered by some event such as a particular date or a day of the week [20].

2.1.1.4 Macro Virus

These are programs that take advantage of the macro utilities that are built into programs like Excel and Word. They are fairly easy to write and target documents that save macro code within the body of the document. Macro viruses are no longer widespread after people have become more cautious about using the Office macro feature [9] and today this feature is disabled by default.

2.1.1.5 Encrypted Virus

This is a type of virus whose body is encrypted. The virus itself contains the key for decryption and a decryption engine within itself. The encryption key varies from infection to infection causing the encrypted body to appear differently in every instance [7, 12]. This methodology was used by virus writers to hide the virus from signature scanning techniques.

2.1.1.6 Polymorphic virus

A polymorphic virus contains an encrypted body and a decryption engine like the encrypted virus. In addition to this, it also has a mutation engine that creates new encryption schemes for every infection. If a user runs a program that contains the virus, the decryption engine first executes and places the virus body in memory. The virus then starts the mutation engine which generates an encryption - decryption routine for the next infection. The virus finally encrypts a copy of itself and the mutation engine using the new encryption engine and places itself in a new file. This creates a virus that does not have a fixed signature to scan for as no infections look the same [7].

2.1.1.7 Stealth Virus

This type of virus attempts to hide its presence by hooking onto some system calls. The first stealth virus was 'Brain' which attempted to hook Interrupt 13 and certain other system calls that detect viruses in DOS. This is a trend that is prevalent even today.

2.1.2 Other Malware

2.1.2.1 Trojan Horse

Trojan Horse is a program that enters a machine disguised or embedded inside legitimate software. The Trojan looks harmless or something interesting to a user, but is actually harmful when executed. Each Trojan has its own characteristic that is dependent on what the designer intended it to do. The Trojan depends on successful implementation of social engineering concepts as it has to fool a user into installing the code and does not depend on security flaws or loopholes present in the system. Once inside a system, it can exploit any resource or use the machine as a zombie, or use the infected system as a launch pad for further attacks.[1]

2.1.2.2 Worms

A worm is a self replicating program. Unlike a virus, it does not attach itself to any existing program. It uses the network resources to infect other machines in the network. Worms

always harm the network whereas viruses always infect or corrupt files on a targeted computer. [26]

2.1.2.3 Rootkits

A Rootkit is a set of impostor operating system tools (tools that list the set of active processes, allow users to change passwords, etc.) that are meant to replace the standard version of those tools such that the activities of an attacker that has compromised the system can be hidden [3]. Once a rootkit is successfully installed, the impostor version of the operating system tools becomes the default version. A system administrator may inadvertently use the impostor version of the tools and may be unable to see processes that the attacker is running, files or log entries that result from the attacker's activity and even network connections to other machines created by the attacker.

2.1.2.4 Botnets

Once an attacker compromises a machine, the attacker can add that machine to a larger network of compromised machines. A botnet is a network of software tools that attackers use to control large numbers of machines at once [3]. A botnet of machines can be used, for example, to launch a Distributed Denial of Service (DDoS) attack in which each of the machines assimilated into the botnet is instructed to flood a particular victim with IP packets. If an attacker installs a rootkit on each machine in a botnet, the existence of the botnet could remain quite hidden until the time at which a significant attack is launched.[3]

2.1.2.5 Spyware

Spyware is software that monitors the activity of a system and some or all of its users without their consent [3]. For example, spyware may collect information about what web pages a user visits, what search queries a user enters, and what electronic commerce Transactions a user conducts. Spyware may report such activity to an unauthorized party for marketing purposes or other financial gain.[3]

A Whitelist based implementation of Anti-Virus definition file to detect unknown malicious activity

2.1.2.6 Keyloggers

A keylogger is a type of spyware that monitors user keyboard or mouse input and reports some or all such activity to an adversary [3]. Keyloggers are often used to steal usernames, passwords, credit card numbers, bank account numbers, and Personal Identification Numbers (PINs). [1]

2.1.2.7 Adware

Adware is software that shows advertisements to users potentially (but not necessarily) without their consent [3]. In some cases, adware provides the user with the option of paying for software in exchange for not having to see ads.

2.1.2.8 Clickbots

A clickbot is a software tool that clicks on ads (issues Hyper Text Transfer protocol (HTTP) requests for advertiser web pages) to help an attacker conduct click fraud [3]. They can receive instructions from a botnet master server as to what ads to click, and how often and when to click them.

2.2 Types of Attacks

A malware may choose one or a combination of several attack methods to compromise a system. Some of these techniques are discussed below.

2.2.1 Social Engineering

Social Engineering is a common method of attack by viruses and worms, especially those ones that spread by means of e-mail and messenger. It is used to manipulate people into performing certain actions or divulging information [1]. In most cases, the malware arrives in the e-mail of a user as part of a picture or an executable, which when viewed launches a back-door program or a Trojan. The best way to prevent social engineering attacks is not to trust e-mails or messenger texts coming from un-known sources. It is possible that a malware may use the e-mail id of known person to spread the attachment, however, these mails

usually follow a set template in terms of the content, hence if a user learns how to recognize the template these attacks can be averted.[1,3]

2.2.2 Mass E-Mailers

Mass E-Mailers are malwares that arrive by e-mail on a machine. If the user executes the attachment, the Trojan executes in the background and obtains control over the machine. It then looks up at the address book of the user and sends itself out to everyone on the contacts list. Most mass mailers use social engineering tricks and concepts to trick users into opening the attachment. [1]

2.2.3 Exploit on Software Vulnerabilities

Operating systems and system software contain many bugs or vulnerabilities that can be exploited to gain control of a machine. Exploit refers to a small section of code that can take advantage of the flaw present in the software. This code is generally reused in numerous Trojans and viruses before the vulnerability is fixed by the software developers by means of a patch. Many exploits are designed to provide root access to a machine. It can be done by just one exploit, or by means of multiple exploit, each providing escalated level of privileges to the attacker. A single exploit takes advantage of specific software vulnerability. [1,3]

Exploits are categorized by the vulnerability that they exploit. A few types of exploits are [1]:

- Buffer Overflow
- Heap Overflow
- Integer Overflow
- Code Injection
- SQL Injection
- Cross-site Scripting

2.2.4 Phishing

Phishing is an activity used to steal information from users using social engineering techniques [3]. This is usually done by disguising as some trusted entity during electronic

communication. Phishing is typically carried out by e-mail. It directs users to websites that look identical to that of the trusted entities, where an attempt is made to trick the user into revealing the password or some other secret. It is very difficult to prevent phishing attacks by means of monitoring software; the user has to be careful to avoid getting tricked. [1]

2.2.5 Pharming

Pharming is an attack aimed at redirecting a website's traffic to another fake website [3]. It can be achieved by either changing the Hosts file on a computer or by exploiting some vulnerability in the Domain Name Server (DNS) server software. Compromised servers are known as having been poisoned. Pharming is used to steal secret information. Countering Pharming is a difficult problem. Anti-Virus and similar software cannot provide protection against this threat.[1,3]

2.3 Methods to Combat Malware

One of the largest categories of traditional responses to malware detection are blacklist solutions, including traditional blacklist solutions such as antivirus and anti-spyware software, and advanced blacklist solutions such as heuristic additions.

2.3.1 Traditional Blacklist Solutions

A blacklist is a list of a particular entity, whether domain names, email addresses, or malware programs, that are considered dangerous or damage causing, and are denied entry to the infrastructure they are trying to penetrate [22]. For example, a web site can be placed on a blacklist because it is known to be fraudulent, or because it exploits browser vulnerabilities to send spyware or other unwanted software to a user.

Common examples of traditional blacklist solutions are antivirus and anti-spyware software. Blacklist software works by blocking known threats. Antivirus software companies have a list of known malwares that they provide to their subscribers. When a new malware becomes

known, the antivirus companies create a defense against it and provide that update to their users.

Blacklist software can also be used to prevent email spam. Users can create a rule in a spam filter program that prevents email from a particular destination (or matching other specified criteria) from being delivered, even though the spam filter program would have ordinarily allowed it.

Some of the Blacklist cons and pros are:

Blacklist solution benefits:

- Updates to malware lists are automatic and do not require time consuming maintenance
- allows malware to be identified and eliminated
- Updates can be done on the fly by an update services server
- offers complete security and protection against all currently known threats

Blacklist solution drawbacks:

- Users are essentially giving control of their networks to a third-party vendor, and Need to continually update malware and spyware definitions, which increases the load on hardware and network bandwidth
- The modular design is expensive, and difficult to set up and maintain
- the solution requires viruses or spyware to be identified and added to the blacklist, leaving workstations and networks vulnerable to a day-zero attack
- The scanning of all incoming and outgoing IP traffic results in slower workstations
- Remote users must obey strict rules to update all definition files on a regular basis to ensure security

2.3.2 Advanced Blacklist Solutions

Heuristics is the application of experience-derived knowledge to a problem [22]. It is sometimes used to describe software that screens and filters out messages likely to contain a

computer virus or other malware. Heuristic software looks for known sources, commonly-used text phrases, and transmission or content patterns that company history has shown to be associated with email containing viruses.

Heuristics is a term coined by antivirus researchers to describe an antivirus program that detects malwares by analyzing the program's structure, its behavior, and other attributes, instead of looking for signatures.

Heuristic solution benefits:

- do not need definition file updates
- may potentially intercept day-zero attacks
- provide another layer of protection because they do not rely completely on definition files
- can sometimes find a threat not listed in a blacklist

Heuristic solution drawbacks:

- makes assumptions about the problem it is trying to solve, and can yield less than optimum results
- Legitimate emails in large volume of mail may also fall into the pattern, resulting in many “false positives” and delaying the delivery of valid email
- Technology is relatively new; time will be needed to develop and improve it

2.3.3 Intrusion Detection

With the increase of attacks on computers and networks in recent years [1], improved and essentially automated surveillance has become a necessary addition to Information Technology (IT) security. Intrusion detection is the process of monitoring the events occurring in a computer system or network and analyzing them for signs of intrusions. Intrusions are defined as attempts to compromise the confidentiality, integrity or availability of a computer or network or to bypass its security mechanisms. They are caused by attackers accessing a system from the Internet, by authorized users of the systems who attempt to gain additional privileges for which they are not authorized and by authorized users who misuse

the privileges given to them. An Intrusion-Detection System (IDS) is the Software and hardware implementation that automates this monitoring and analysis process.

2.3.4 The Whitelist Solution

Whitelist technology is the opposite of blacklist technology; the list of entities, whether domain names, email addresses, or executables, is a list of what is allowed to penetrate a system [22]. For example, a whitelist of domain names is a list of URLs that are authorized to display, despite any rules of an email spam blocker program. The most common examples of whitelist solutions are email based, with users creating a list of authorized addresses that they can receive mail from, again despite the rules of an anti-spam program.

A widely used whitelist security system is a Firewall. In a firewall we can provide Internet Protocol (IP) address of known machines or Port Numbers that are allowed. This list is used to allow traffic to and from known destinations. This solution is effective in securing a corporate network and even our personal computers while navigating the World Wide Web.

Whitelist solution benefits:

- No virus or spyware definition updates are needed; therefore, systems are always protected from day-zero virus attacks
- Constant scanning of incoming and outgoing IP traffic is not necessary; therefore, there is no decrease in performance
- No unauthorized executable files, such as a chat program, spyware, or Trojans will ever install or run; therefore, staff productivity increases and downtime decreases
- No illegal or unlicensed software will ever be installed on system workstations; therefore, an organization is not at risk for fines from the Business Software Alliance
- Hardware and support budget costs are reduced from a decrease in re-imaging PC's on a regular basis; therefore, organizations can re-channel resources to other activities

2.4 Anti-Virus

The appearance of computer virus in the 1980's caused the emergence of the anti-virus. Anti-virus is software that scans the system either continuously, or at specified times for the presence of malicious entities. The anti-virus software has constantly evolved with the virus. The methodologies used by anti-virus software areas below:

2.4.1 Signature detection or Pattern Matching

Signature detection involves the anti-virus application scanning the computer for files that contain a code that it recognizes as malware. The initial anti-virus programs would scan the entire executable file to find the presence of the virus code. Later the programmers found out that most of the virus infections involve virus placing its code on the entry point of the program. Hence the scanners started checking the entry point of the program; this methodology became obsolete when encrypted and polymorphic viruses emerged.[1]

2.4.2 X – Raying

As encrypted viruses arrived, anti-virus software started using brute force decryption of the code. This was possible since they had to do known plaintext attack on the encrypted code; the plaintext was the known virus definitions. The virus writers used random encryption algorithms; hence the protocols were easy to crack. This type of scanning was known as X-raying [1].

2.4.3. Emulation

Another technique that emerged due to the appearance of polymorphic viruses was Emulation. This involved starting a Central Processing Unit (CPU) emulator and loading the executable file on it. The virus would not realize that it was being run on an emulation environment. This allowed the software to observe the behavior of the program in a closed environment where no damage would be done to the user machine. [1]

2.4.4 Frequency Analysis

Frequency analysis involved scanning a code for the presence or absence of particular opcodes. Programmers found that most legitimate programs would use Disk Operating System (DOS) interrupts like 21h in their code. This interrupt was hardly visible in malicious codes. Hence they would examine frequency of such opcodes, and programs having them would not be scanned. [1]

2.4.5 Heuristics

Malware writers slowly started using techniques such as entry point obfuscations, by which they would not keep the virus code at the entry point of the executable. The emergence of viruses that could change their properties required a change in virus detection technology. Programmers came up with decision support systems, where a scan would have weight system. Points were allotted to behavior such as memory access, file access, page faults. After scanning, the number of points accumulated determines if a file would be flagged for a virus from a known set of behavioral trend. [1, 26]

2.5 Use of Machine Learning Techniques

Various researchers have attempted to use machine learning techniques to perform heuristic analysis on metamorphic viruses. The main machine learning techniques researched include:

- 1) Data mining methods
- 2) Neural networks
- 3) Hidden Markov models.

2.5.1 Data Mining Approach

Data mining is a technique used for the efficient discovery of valuable yet non-obvious information from any large collection of data [5]. Data mining methods are often used to detect patterns in a large set of data. These patterns are then used to identify future instances in a similar type of data. Amanda [5] experimented with a number of data mining techniques to identify new malicious binaries. They used three learning algorithms to train a set of classifiers on some publicly available malicious and benign executables. They compared

their algorithms to a traditional signature-based method and reported a higher detection rate for each of their algorithms. However, their algorithms also resulted in higher false positive rates when compared to signature-based method.

The key to any data mining framework is the extraction of features, which are properties extracted from examples in the dataset. Amanda [5] extracted some static properties of the binaries as features. These include system resource information (the list of Dynamic Link Libraries (DLLs), the list of DLL function calls, and the number of different function calls within each DLL) obtained from the program header, and consecutive printable characters found in the files. The most informative feature they used was byte sequences, which were short sequences of machine code instructions generated by the hexdump tool.

The features would be used in different training algorithms. There was an inductive rule-based learner that generated Boolean rules to learn what a malicious executable was; a probabilistic method that applied Bayes rule to compute the likelihood of a particular program being malicious, given its set of features; and a multi-classifier system that combined the output of other classifiers to give the most likely prediction.

2.5.2 Neural Networks

Researchers at IBM implemented a neural network for heuristic detection of boot sector viruses [2]. The features they used were short byte strings, called trigrams, which appear frequently in viral boot sectors but not in clean boot sectors. They extracted about 50 features from a corpus of training data, which consisted of both viral and legitimate boot sectors. Each sample in the dataset was then represented by a Boolean vector indicating the presence or absence of these features.

The network was single-layered with no hidden units. It was trained using classic back propagation technique. One common problem with neural network is overfitting, which

occurs when a network is trained to identify the training set but fails to generalize to unseen instances. To eliminate this problem, multiple networks were trained using different features and a voting scheme was used to determine the final prediction.

The neural network was able to identify 80-85% of viral boot sectors in the validation set with a false positive rate of less than 1%. The neural network classifier has been incorporated into the IBM Antivirus software which has identified about 75% of new boot sector viruses since it was released [21]. It can be said that neural networks are very effective in detecting viruses closely related to those in the training set. They can also identify new families of viruses containing similar features as the training samples. [2]

2.5.3 Hidden Markov Models

Hidden Markov models (HMMs) are well suited for statistical pattern analysis. Since their initial application to speech recognition problems in the early 1970's [15], HMMs have been applied to many other areas including biological sequence analysis [7].

An HMM is a state machine where the transitions between states have fixed probabilities. Each state in an HMM is associated with a probability distribution for observing a set of observation symbols. HMM can be trained to represent a set of data, which is usually in the form of observation sequences. The states in the trained HMM then represent the features of the input data, while the transition and the observation probabilities represent the statistical properties of these features. Given any observation sequence, we can match it against a trained HMM to determine the probability of seeing such a sequence. The probability will be high if the sequence is similar to the training sequences. [2]

Metamorphic viruses form families of viruses. Even though members in the same family mutate and change their appearances, some similarities must exist for the variants to maintain the same functionality. Detecting virus variants thus reduces to finding ways to detect these

similarities. Hidden Markov models provide a means to describe sequence variations statistically. HMMs similar to those used in protein sequence analysis can be used to model virus families. In virus modeling, the states correspond to the features of the virus code, while the observations are instructions or opcodes making up the program. A trained model is able to assign high probabilities to and thus identify viruses belonging to the same family as the viruses in the training set. [2]

Chapter III

Chronology and Case Studies of Selected Malwares

This chapter starts by presenting the major milestones in the history of computer malwares. Then it discusses selected case studies performed by other researchers on notorious malwares in computing history.

3.1 Milestones in Evolution of Malwares

1982

The possibility of a computer virus – a program that could reproduce itself – was first suggested in 1949 by John von Neumann; however, it wasn't until 1982 that the first such program, called Elk Cloner, was written [22].

1984

From 1984 onwards, Dr. Fred Cohen produced several academic papers that explored and defined the concept of a computer virus. Cohen defined a computer virus as “a program that can ‘infect’ other programs by modifying them to include a possibly evolved version of itself”. The word “virus” itself was invented by Cohen's faculty advisor, Leonard Adleman, who created a virus theorem, proving mathematically that it is not possible to determine for sure whether a virus is or is not present on a computer.[26]

1986

In 1986, first PC virus called Brain was created by two brothers in Pakistan in an attempt to deter the pirating of a software product they had written. This virus escaped into the wild and spread across the globe, turning up on PCs in various US universities. The Brain's source code was subsequently used as a basis to build other viruses. [26]

1987

From 1986 onwards, new viruses began to appear every few months, and consequently software developers created anti-virus programs in an attempt to deal with the problem. The first such product, called Vaccine, was written in 1987 and others quickly followed.[26]

1988

The Morris worm, written by Robert Morris [3], a student at Cornell University, was the first worm, which was released onto the Internet from a computer in Massachusetts Institute of Technology in November 1988. Until then viruses had been passed from one computer to another by virus files on a floppy disk. Worms introduced a new, more direct and automatic means of infection, copying itself from one machine to another over a network. The Morris worm was originally intended to count the number of computers connected to the Internet, however it ended up bringing many of the computers it infected to a grinding halt. [26]

1990

In 1989 there were about 30 known viruses, a mere handful compared with today's figures of more than 100,000 [22]. This changed in 1990 with the advent of virus exchange bulletin boards, which had large numbers of viruses available for download along with the source code. In order to use the bulletin board, you had to upload a virus too, ensuring that the population of viruses grew more quickly. [26]

1991-1993

By 1991, the proliferation of virus authors had begun to have its impact and viruses became increasingly sophisticated with new ways of working and concealing activities. Virus kits appeared to aid virus authors. Polymorphic viruses appeared – viruses that cannot be recognized by their signature because the virus file mutates as they proliferate. In 1993, a German virus appeared which disabled the Microsoft anti-virus product which runs on MS-DOS 6.0. From this point onwards, the AV industry bloomed trying to understand, recognize and neutralize new viruses after they appeared.

Virus outbreaks became big news in 1992 because of the hysteria that bubbled up around the Michelangelo virus. This virus was a “sleeper” that was timed to activate on all infected PCs on March 6th (Michelangelo's birthday). By then virus infection was commonplace and the prospect of a general meltdown of PCs in offices across the world engaged the minds of the

public. It never actually happened, because the virus was less infectious than some commentators had suggested.

1993-2006

The commercial AV companies prospered both because the number of new viruses emerging was growing at a remarkable rate, and because viruses had become news. By 1992, most regular PC users had read the virus stories and many had experienced a virus infection. As AV vendors never actually solved the virus problem, virus infections continued to occur. This continually demonstrated the need for protection, boosting the sale of AV products because they were the “only solution available.” [22]

The Internet connected all the world’s computers together, providing a far more fertile environment for virus infection. In 1995 macro viruses emerged using Word and Excel files to spread themselves. Email viruses also became prevalent. An important virus-spreading technique was introduced by the I Love You virus, in 2000. As soon as an email carrying the virus with the “I Love You” title was opened, a virus executed, demonstrating the power of social engineering.

Malwares began to include Trojan payloads that allowed hackers to take control of infected PCs. Malwares appeared (for example, CodeRed and SQL Slammer) that were based on a specific software vulnerability. Such malwares could achieve mass infection in hours or, in the case of SQL Slammer, in minutes. Remarkably, it took a mere 10 minutes for SQL Slammer to infect 90 percent of the computers on the Internet that had the vulnerability it exploited - long before any AV vendors were even aware of its existence.

In 2002 a new phenomenon, AdWare and SpyWare, emerged. Such software was introduced by fooling the user into downloading the software and allowing it to run. Once installed, the user would be plagued with pop-up advertisements. AV technology was not able to stop this, so AV vendors created and sold new anti-spyware products.

By 2004, the age of the amateur malware writers was over and malwares had become a highly useful tool in the hands of the cyber-criminal, who herded together huge networks in order to distribute spam, carry out denial of service attacks and spread malwares. By 2005, malware writers were targeting malwares at specific organizations, in order to carry out some specific criminal act, such as data theft or fraud. The hobbyist hackers had been superseded by professional criminals.

Ineffectiveness of Blacklist based AV Technology

The weakness of AV technology becomes apparent if one constructs a timeline of the evolution of malwares. AV technology never became a prosperous business until around 1993. By then the population of PCs was growing at a dramatic rate, viruses spread by floppy disk had become common and viruses were in the news from time to time. [22]

If AV technology had been effective, then the problem of viruses and other associated malware would have gradually faded away. But quite the opposite happened. The Internet provided an exceptional breeding ground for Malwares and helped malware authors from across the world to share their accumulated knowledge. The malware problem escalated.

By 1999 the worst possible outcome was occurring. Mass virus infections were happening regularly and the costs to businesses and computer users were extreme. The costs of mass viruses as calculated by Computer Economics, (<http://www.computereconomics.com>) are: [25]

- 1999 - Melissa (\$1.5 bn)
- 2000 - I Love You (\$8.75 bn)
- 2001 - Code Red et al (\$5.5 bn)
- 2002 - Klez et al (\$1.65 bn)
- 2003 - Slammer et al (\$4 bn)
- 2004 - MyDoom (\$4 bn)

3.2 Selected Case Studies of Notorious Malwares

This section discusses five of the notorious malwares Morris Worm, Code Red, Nimda, Blaster and SQL Slammer. These malwares caused havoc and introduced new ways of infiltrating systems. A lot of research and papers have been done on these malwares and the section summarizes the works of Neil Daswani et al [3] .

3.2.1. The Morris Worm

The Morris worm was named after its creator, Robert Morris. Morris was a graduate student at Cornell University when he wrote the worm. When he first deployed the worm, it was able to infect over 6,000 computers in just a few hours. [3]

The Morris worm used the Internet to propagate from one machine to the other, and it did not need any human assistance to spread. The Morris worm made copies of itself as it moved from one computer to the other. The act of copying itself caused substantial damage on its own. The amount of network traffic that was generated by the worm scanning for other computers to infect was extensive. The effort required by system administrators to even determine if a particular computer was infected, let alone remove it, was also significant.

The Morris worm took advantage of the UNIX program fingerd, SendMail and Unix commands rexec and rsh. The fingerd program is a server process that answers queries from a client program called finger. The finger client program allows users to find out if other users are logged onto a particular system. The Morris worm took advantage of the fact that fingerd was homogeneously deployed on all UNIX systems. To propagate from one machine to another, the Morris worm exploited a buffer overflow vulnerability in the fingerd server. [3]

The sendmail program, deployed on UNIX by default, is used to route e-mails from one UNIX server to another. It allows mails to be routed to processes in addition to mailbox files. It has a “debug mode” built into it that allows a remote user to execute a command and send a mail to it, instead of just sending the mail to an already running process. The Morris worm

took advantage of the debug mode to have the mails that it sends execute “arbitrary” code. In particular, Morris had the servers execute code that copies the worm from one machine to another. The debug mode feature should have been disabled on all of the production UNIX systems that it was installed on, but it was not.[3]

A third vulnerability that the Morris worm took advantage of was the use of two additional UNIX commands called rexec and rsh, both of which allow a user to remotely execute a command on another computer. The rexec command required a password. The Morris worm had a list of 432 common passwords hard-coded in it, and attempted to log into other machines using that list of passwords. Once it successfully logged into a machine with a given username and one of the passwords, it would attempt to log into additional machines to which the compromised machine was connected. In some cases, the rexec command was used to remotely execute a command on the additional machines with the guessed password. In other cases, due to the way that the rsh command works, the additional machine would allow a login without a username and password because the machine was whitelisted by the user. [3]

3.2.2 The Code Red Worm

In 2001, the Code Red worm surfaced. It exploited a buffer overflow vulnerability in the Microsoft IIS web server. The web server had an “indexing server” feature turned on by default. Code Red took advantage of the buffer overflow vulnerability in IIS to propagate. Once Code Red infected a particular machine, it started randomly scanning other IP addresses to try to connect to other IIS web servers at those IP addresses. It spread from one web server to another quickly (over 2,000 hosts per minute)[3].

Code Red was interesting because it was able to spread at speeds that humans simply could not keep up with. In order for the worm to spread from one web server to another, it only had to construct an IP address, connect to the web server at that IP address, and exploit the buffer overflow vulnerability at the other web server. The entire process took milliseconds. Human

response takes minutes or hours. Since the worm was able to spread to thousands of machines within minutes, there was little that anyone could do to react to the attack quickly enough to curtail it.

Another characteristic of the Code Red worm is that it spread rampantly, even though there was virus scanning software running on some of the machines it infected. Virus scanning utilities often scan for infected files—they look for particular bit patterns (signatures) in files that may be infected with viruses. However, to prevent being detected, Code Red would just stay resident in the web server's memory. Code Red did not write any files to disk, and, as a result, was able to evade automated detection by some typical virus scanners. At the same time, a user could check if her machine was infected simply by viewing the home page returned by her web server. Anyone visiting an infected web server was alerted, since Code Red defaced the front page of the web server. Unlike most worms, Code Red was much more easily detectable by humans than some virus scanners.[3]

Because Code Red was resident only in memory, it could be eliminated from a particular web server just by rebooting the machine. Yet, even if you rebooted an infected web server, it would typically get reinfected very quickly. So many other infected web servers were continuously scanning for victims that it wasn't long before one of them happened to construct the IP address for your server and re infect it. As such, firewalls were used to block traffic from being sent to web servers to prevent reinfection.[3]

3.2.3 The Nimda Worm

The Nimda worm was very interesting since it took some of what Code Red did and made it a lot worse. Nimda not only spread from web server to web server, but it employed multiple propagation vectors. A propagation vector, in the context of worms, is a method by which the worm spreads to another machine. Code Red, by comparison, only used one propagation vector. [3]

Like Code Red, Nimda spread from web server to web server. In addition, Nimda spread from web servers to web clients by infecting files on the web server. Whenever a web browser connected to that web server and downloaded an infected file, it also became infected. Nimda used the infected client to continue to spread the worm. Nimda sent out e-mails from the infected client to other machines containing the worm's code as a payload. (A payload is the data that the worm carries when it travels from one machine to another) Therefore, Nimda took all of what Code Red did, packaged in a couple of other different propagation vectors, and thereby increased its ability to spread aggressively.[3]

The Code Red and Nimda worms spread so quickly that it caught the attention of many academics and researchers. There are now entire workshops and conferences studying the speed at which worms spread and the potential defenses that we might be able to use as counter measures (e.g., the Workshop on Rapid Malcode, held in association with the ACM Conference on Computer and Communications Security). Some projects presented at such conferences explore the commonalities between some of the mathematical models that can be used to understand both biological spread of viruses and the technological spread of worms.

3.2.4 The Blaster and SQL Slammer Worms

In 2003, the Blaster and SQL Slammer worms surfaced. Blaster, like Code Red, took advantage of a buffer overflow vulnerability in Microsoft's operating system. Instead of attacking a web server, however, Blaster attacked a Distributed Component Object Model (DCOM) service that was running as part of the operating system. Microsoft deployed a patch for the vulnerability at <http://windowsupdate.microsoft.com> on July 16, 2003. While users could have downloaded the patch and inoculated their systems against an attack that took advantage of such a buffer overflow vulnerability, many unfortunately did not. Patching a system is inconvenient and gets in the way of the "real work" that users are interested in doing. On August 11, 2003, even though the DCOM vulnerability was announced and a patch was deployed, the Blaster worm was still able to take advantage of the vulnerability to launch its attack.[3]

The Blaster worm used an exploit that would cause the user's system to start shutting down. The dialog box shown in Figure 3-1 would pop up on a user's screen once their host was infected.



Figure 3.1 The System Shutdown dialog box caused by the Blaster worm

The Blaster worm attacked hosts running versions of Windows NT, 2000, and XP. It did not need to be running a web server. Most users were surprised by the dialog box that popped up as their system shut down. Some users thought that the dialog box might be due to some operating system bug, which they assumed could be corrected by simply by letting their system reboot.

Once the worm caused the system to shut down and reboot, the worm issued a DDoS attack against the Windows Update site (<http://windowsupdate.microsoft.com>). So even when users realized that their PCs may have been infected with a worm, when they tried to go to the Windows Update site to patch their systems, the deluge of DoS traffic sent to the site from their own computers prevented them from doing so.[3]

The Blaster worm coupled some characteristics of the previous worms (exploiting a buffer overflow and randomly scanning for new hosts to which to propagate) with a DDoS attack against a web site that had the patch to fix the problem.

SQL Slammer was another worm that appeared the same year as Blaster. SQL Slammer, like Blaster and some of the other previous worms, took advantage of a buffer overflow vulnerability. However, instead of attacking an operating system service (as in the case of Blaster) or the web server application (as in the case of Code Red), SQL Slammer attacked the Microsoft SQL Server database application.

There are many “mission critical” types of applications that depend upon databases such as Microsoft SQL Server. Once SQL Slammer hit a particular database server, it disabled that server and continued scanning random (Internet Protocol) IP addresses for other SQL Server machines that it could infect. The excessive traffic generated by the SQL Slammer worm as it scanned for other SQL servers to infect caused outages in approximately 13,000 Bank of America ATMs, which prevented users from withdrawing money. In addition, Continental Airlines’ systems were affected—some regional flights were canceled and others were delayed [22]. The SQL Slammer worm was serious enough that the White House was notified. [3]

Another characteristic of SQL Slammer is that it took a single User Datagram Protocol (UDP) packet of only 376 bytes to exploit the buffer overflow vulnerability to propagate the worm. Since UDP is connectionless, the worm was able to spread extremely quickly. The worm infected at least 75,000 hosts, and 90 percent of them were infected within the first 10 minutes of the worm’s release [3].

While the SQL Slammer worm caused most of its outages primarily because of the traffic that it generated by scanning other SQL servers, the worm could have been much worse if, for example, it had been designed to delete data, change data, or publish data on the Internet.

CHAPTER IV

Related Works

In this section related works will be presented in chronological order. The works presented deal with intrusion and anomaly detection systems, Security vulnerability studies and works related to malware detection and Antivirus Protection.

4.1 Computer Immunology

4.1.1 Researches at University of New Mexico

Researchers at the University of New Mexico were the pioneers [4] in investigating relationships between human immunology system and implementing it as a model to detect intrusion. In their work they showed that a model based on the working of human immunology can be used to construct anomaly and intrusion detection system. To achieve this source of data was based on System Calls (Application Programming Interface (API) in Windows). They came up with different methods to analyze system call sequence including sliding windows and n contiguous calls to identify anomalous behaviors. Their pioneering work in the UNIX environment was adapted by latter researchers to extend their work.

4.1.2 Researches at Technical University of Denmark

In 2002 Rune Schmidt Jensen [2] used components and techniques from the biological immune system and already published papers on computer immune systems to design a system for virus detection. Like the biological immune system it is able to distinguish between self and non-self, their system was able to distinguish between normal behavior and abnormal behavior in programs.

They have found that the normal behavior of a program could be represented by Hidden Markov Models (HMMs) and used two different approaches to train the HMMs for the

normal behavior: a static approach and a dynamic approach. In the static approach he defined the normal behavior from the binary code of a program, whereas in the dynamic approach he used traces of system calls generated by a program to define the normal behavior of a program. Furthermore, he discussed and showed how to train the HMMs to represent the normal behavior of the programs.

To test all the different ways of using the HMMs he implemented software in Java. A framework for representing and training the HMMs were developed. He experimented with representing the behavior of programs by training HMMs on binary code from the programs and on traces of system calls generated by the programs. He observed how well the HMMs detected randomly made changes in programs, and how well it detected that the programs had been infected with a virus.

The experiments indicated that the HMMs were quite good at detecting changed behavior. HMMs trained on static code from one or several non-infected programs were able to detect if any of the programs were infected with a virus. The same positive results were found when training a HMM on traces of system calls generated by a non-infected program – again the HMMs were able to detect that the program were infected with a virus. He concluded that HMMs can be used to detect changed behavior in programs due to virus infections.

From the experiments he showed that training HMMs on traces of system calls were much faster than training HMMs on binary code. This together with the fact that the dynamic approach can detect the virus while it is trying to execute itself convinced them that the dynamic approach is the ideal approach to use. Furthermore, the dynamic approach enabled them to train in a real environment tracking how the user normally uses the programs. In this way they were able to get a more true profile of how the programs normally behave in the user's environment.

The only drawback of their research is that they tested it with a single virus. In addition they infected a 3 kb file with 5 kb Virus. The Infection can easily be detected since the change is

large. However, their approach has shown alternative ways to detect viruses and they concluded that a further investigation is required.

4.2 Computer Security

In 2004 a paper by Fausi Qattan & Fredrik Thernelius at Stockholm University - Royal Institute of Technology [6] worked to find and isolate deficiencies in current protection mechanisms. They concluded that neither personal firewalls nor antivirus programs should work on their own. They are necessary components for implementing security today but they need to be complemented or replaced by other security measures.

They indicated that some services are critical to the operating system and cannot be switched off or restarted, this should also be the case for the protection software that is running. It should not be possible to switch them off, at least not without providing some sort of extra authentication process. When testing the Trojans they discovered that they were able to imitate other processes using a technique called “memory injection”. The possibility to do this is a flaw in the operating system.

Finally they proposed digital immune defense system as an interesting solution for the future. If the research within this area can produce a defense system for the computer that acts like the immune defense system then we would get an autonomous system that could protect itself against both known and unknown malware. This would be an ideal situation.

4.3 Malware Detection and Antivirus

In 2007 Raghunathan Srinivasan of Arizona State University [1] presented an approach to improve the reliability of the anti-virus process by hiding its presence from other processes on the machine. The reason that the anti-virus program has to be hidden from all other processes is that a malware may infect any process on the system; in such a situation no component of a consumer computer can be trusted.

The first step in solving this problem was changing the names of the files and changing the registry entries by installing the process under a different name. This step helps in working around attacks that scan the registry entries and the file system to identify the anti-virus program.

Next, the program was started by means of a different process, by this; the starting point of the anti-virus program was obscured. After this, the process was continuously migrated to different address spaces to avoid detection by any malware. This particular component enables the anti-virus process to overcome attacks that take a system snapshot, and identify the anti-virus process by finding the files used by each process. By moving the code at regular intervals of time, such a snapshot would not be very useful in killing the anti-virus process as it would have migrated to another process space while the results of the snapshot are calculated. After this, multiple watch processes were installed to detect if the anti-virus program is shut down at any point of time. The design also included checking of library files at system shutdown to detect any unauthorized changes. Finally, the implementation of whitelists and scattering of definition files was left as future work.

On the September 2007 IEEE International Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications , Sergei Bezobrazov and Vladimir Golovko of Brest State Technical University [21], presented a paper on Neural Networks for Artificial Immune Systems (AIS): LVQ for Detectors Construction. In the paper they examined the AIS approach for malicious code detection. The AIS is able to discern between non-infected file of operation system and malicious code. The feature of the AIS consists in capability for unknown malicious malware detection. Application of the ANN for detectors generation allowed them to create the powerful detectors. Undesirable detectors are destroyed during the selection process which allows avoiding false detection appearance. Uniqueness of detectors consists in capability to detect several malicious viruses. That is detector can detect viruses analogous with that malicious code on which training are realized. In that way they significantly increased probability of unknown malicious code detection.

Their experiments show it is necessary to enlarge population of detectors creation. Presence of random probability by detectors generation enables to create different detectors. However it is significant that detectors ability depends on files on which they are trained. It is desirable for training process a various non-infected files and all types of malicious code to have. If your computer system with outdated antivirus bases can be unprotected in the face of new malicious code attack then the AIS gives you a high probability to detect it. Application of the AIS for malicious code detection will expand the potentialities of existing antivirus software and will increase level of computer systems security.

CHAPTER V

WhiteList based System Proposal

This chapter begins elaboration of activities performed in the thesis and puts forward a whitelist based definition file proposal. In the previous chapters different issues related to malwares were discussed. This includes classification of malwares, selected case studies on notorious malwares since the era of Personal Computers and related works in the area of malwares was presented. In addition the technology used to fight malwares by Anti-Virus and Intrusion Detection Systems (IDS) were covered. The general trend in the war against malware is the use of blacklisting anti-virus definition file.

Whenever a new malware is discovered the malwares sample must be entered in the database. For Polymorphic and Metamorphic malwares, for each malware, hundreds if not thousands of samples need to be entered. The major drawback of this well established system is that our system is open to Zero Day Attack – Our PC can be infected by a new malware, the protection is only for known malwares samples. Furthermore, as the database grows the scanning becomes slow as matching for specific patterns needs more time.

This Thesis's main goal is researching a new approach to building an Anti-Virus definition File that enable us more protection even from unknown malwares at the time the database was built. The approach will be building a WhiteList based database instead of blacklist. The main research will focus on identifying what parameters must be part of a WhiteList definition file.

5.1 Whitelist based definition file Proposal

The increasing number and technique of malicious systems needs a different approach to detect and eliminate these threats. As discussed in the literature review, many researchers have tried to implement a solution to the problem in a different dimension other than the traditional black list technology. These approaches implement methods as varied as neural

networks, hidden Markov models, and data mining as a tool to analyze the available data and reach at a result.

In most of the researches the source of the data is primarily system calls in UNIX and API Traces in Windows. Most of these approaches have practical application in Intrusion detection systems. However, their application in Commercial and Day to Day Anti-Virus Systems is almost nonexistent, IBM Corporation being the first one to implement a neural networks based antivirus to detect unknown boot Virus [2,5]

Apart from System Calls security systems setup a whitelist of allowed executables and only those files can execute. Some security protocols ask if the user has initiated an action and whether the system will allow it to continue (Windows Vista Feature). The drawback of this system is the average user has to make a decision whether to allow or deny an application to run. Furthermore social engineering methods can fool the user, let alone being tired of message boxes and simply pressing ok.

In this research, a proposal to whitelist based solution to detect unknown malicious activities will be made. Although this approach is not new, in this thesis a different implementation will be proposed and it will be demonstrated that the approach works. All researches done and covered in the related works try to deal with the problem from machine learning techniques in the area of neural networks, Hidden Markov Models and Data mining approaches. None of the systems were economical nor commercially viable. This paper tries to approach the problems by identifying parameters that can be used as well as trying to identify new ones. From what has been done in other papers, monitoring of Microsoft Windows Registry, Application programming Interfaces (API's) and opcode monitoring are ones used as parameters for their algorithms.

For the research and experiments for WhiteList database the following parameter will be considered

- Windows Registry

- Application programming Interfaces
- Opcodes
- Kernel and user time
- Input/Output Read, Write and Other
- Context Switches
- Page Faults

The experiments to be conducted look into whether each one or a combination of them can be used to classify a given executable as malware or benign. To identify peculiar behavior of malwares sample of malwares (Live) as well as code generated Malwares will be selected. In addition documentation on the behaviors of known malwares from Symantec, MacAfee, Kaspersky and Sophos Anti-Virus Company's website were used to extract parameters that most malwares make use of.

5.2 Malwares Survival and Reproduction Interception

What makes any malwares a malware is its ability to survive and reproduce by infecting others and running itself whenever the operating system boots. The scenarios that a malware executable can start automatically when the Operating System boots are

1. It puts keys in Auto Start locations in the registry [13]
2. Infects Auto starting benign programs reference or their configuration settings in the Registry , so that the malware also starts when the program starts
3. Infects Windows Core Processes and Uses them to start as they are always guaranteed to run (for example WinLogon process)
4. Provides its own Gina.dll Library that performs third party authentication mechanism to windows and uses it to perform or load itself. The Gina

Library can be used to provide custom authentication mechanism such as logging to a machine using finger printing [16].

5. By putting a file 'AutoRun.inf' on a root of a hard Disk or Removable Device with a reference to the virus.
6. Putting a copy of itself in the Windows Startup folders
7. Adding a reference to AutoExec.bat file

The only exception to this rules are boot Viruses, which were very common in the 1990's. A boot Virus can load itself in memory even before the Basic Input Output System(BIOS) transfers control to the operating system. For a complete list of registry and file auto start locations see Appendix II.

A WhiteList definition file needs to take these cases into consideration in order to avoid a malware from taking advantage of the above scenarios. By doing so a malwares survival, reproduction and ability to infect others will decrease dramatically.

5.3 The registry and its use as a Whitelist parameter

In this section, important structures of the windows operating system will be presented which will be used to implement the whitlelist definition file database. One of the most vital parts of the Windows Operating System is the Registry.

5.3.1 The Registry

The registry is a system-defined database that applications and Microsoft Windows system components use to store and retrieve configuration data [16]. The registry stores data in binary files. To manipulate registry data, an application must use the registry functions. The data stored in the registry varies according to the Windows platform that is used.

5.3.1.1 Structure of the Registry

The registry stores data in a hierarchically structured tree. Each node in the tree is called a key. Each key can contain both sub keys and data entries called values. Sometimes, the

presence of a key is all the data that an application requires; other times, an application opens a key and uses the values associated with the key. A key can have any number of values, and the values can be in any form [23].

Each key has a name consisting of one or more printable (American National Standards Institute) ANSI characters that is, characters ranging from values 32 through 127. Key names cannot include a space, a backslash (\), or a wildcard character (* or ?). Key names beginning with a period (.) are reserved. The name of each subkey is unique with respect to the key that is immediately above it in the hierarchy. Key names are not localized into other languages, although values may be [23].

5.3.1.2 Registry Storage Space

Although there are few technical limits to the type and size of data an application can store in the registry, certain practical guidelines exist to promote system efficiency. An application should store configuration and initialization data in the registry, but other kinds of data should be stored elsewhere.

Generally, data consisting of more than one or two kilobytes should be stored as a file and referred to by using a key in the registry rather than being stored as a value. Instead of duplicating large pieces of data in the registry, an application should save the data as a file and refer to the file. Executable binary code should never be stored in the registry. A value entry uses much less registry space than a key. To save space, an application should group similar data together as a structure and store the structure as a value rather than storing each of the structure members as a separate key. (Storing the data in binary form allows an application to store data in one value that would otherwise be made up of several incompatible types.)

5.3.1.3 Predefined Keys

Windows provides two predefined keys at the root of the registry: HKEY_LOCAL_MACHINE and HKEY_USERS. In addition, two sub keys are defined:

HKEY_CLASSES_ROOT (a sub key of HKEY_LOCAL_MACHINE) and HKEY_CURRENT_USER (a sub key of HKEY_USERS). These registry handles are valid for all Win32 implementations of the registry, although the use of the handles may vary from platform to platform [23].

Predefined keys help an application navigate in the registry and make it possible to develop tools that allow a system administrator to manipulate categories of data. Applications that add data to the registry should always work within the framework of predefined keys, so administrative tools can find and use the new data.

HKEY_CLASSES_ROOT: Registry entries subordinate to this key define types (or classes) of documents and the properties associated with those types. Data stored under this key is used by Windows shell applications and by object linking and embedding (OLE) applications.

HKEY_CURRENT_USER: Registry entries subordinate to this key define the preferences of the current user. These preferences include the settings of environment variables, data about program groups, colors, printers, network connections, and application preferences.

HKEY_LOCAL_MACHINE: Registry entries subordinate to this key define the physical state of the computer, including data about the bus type, system memory, and installed hardware and software.

HKEY_USERS: Registry entries subordinate to this key define the default user configuration for new users on the local computer and the user configuration for the current user.

5.3.2 How the registry and File System can be used as a WhiteList

Malware can take a form from a computer virus to a worm, Trojan or other or even a combination of the indicated forms, which have different propagation vector characteristics. However one fact is true to all Malwares – Reproduction and the ability to infect others. In order to do this a malware has to put a reference to itself in the windows registry so that it

can auto start when Windows operating System boots. For a complete list of registry auto start locations considered in the thesis see appendix II.

Normal Programs that Auto Start also put a reference to themselves in registry. The challenge is to identify the existing keys as Malware or Not. In this thesis a whitelist will be built from existing auto start locations. The assumption made is that when the whitelist is built the system is assumed to be clean of Malwares. Then the system will handle the entry of new programs into the whitelist after verifying them as malware or benign.

5.4 Experimental Setting and Tools

A Virtual-PC environment was used to study, observe and record each and every activity a malware performs when it first infects a PC and what it does subsequently. For this a Virtual-PC Software as well as system activity monitoring tools recommended by Microsoft Corporation for Microsoft Windows Operating System were used. The following tools and experimental settings where used

1. **VMware[10]:** Windows XP SP2 was installed on this virtual machine software. All test related to Malware infection was observed on this environment. Settings for it are
 - a. RAM :256 MB
 - b. Hard Disk Space : 4GB
 - c. Processor: Pentium IV 3 GHZ
2. **Spy Studio:** This software is used to track API Sequence for a particular program. In this work the API trace for both Malwares and benign programs was conducted
3. **KrView[14]:** This software is used to monitor system performance or monitor a specific or group of software's characteristics such as time they passed in privileged or user mode, number of I/O performed etc. The technique was applied for malwares as well as benign programs.
4. **Microsoft Visual Studio 2005 :** C# Language was used to write the whitelisting program and Verification code.

5. **Microsoft Excel 2003:** Used to write a macro program to analyze data output from spystudio and krview programs
6. **Next Generation Virus Creation Kit (NGVCK) :** Used to Generate a Malwares with different characteristics.

5.5 Sample Collection

For experiments to be conducted windows Portable executable (PE) files are needed. The sample executables need to be in one of the following types

- **Benign program:** These are legitimate programs such as Microsoft Word, Excel etc...
- **Live malwares:** These are malwares in the wild that may or may not have Payloads which are detectable by latest antivirus black list definition files.
- **Code Generated Malwares:** These are malware programs generated from Virus Creation software.

For the experiments to be conducted 118 sample of benign and Malwares were collected

1. 50 Benign programs were selected
2. Malwares
 - a. 44 Malwares were Generated using Next Generation Virus Creation tool (NGVCK).In [7, 24] it was shown that the Virus generated by this tool has a similarity less than 2%.
 - b. 24 On the Wild Viruses were collected from Internet cafes and Infected PC in the University, Outside Office Settings and the Internet. The method of collection was using an empty USB Flash disk and inserting it into an infected or suspect PC so that it becomes infected. Then the USB Flash disk was processed on a Virtual PC environment to extract the malwares sample.

In Total 68 Malwares were collected. From the Malware and Benign Samples there was a need to further divide the samples for experiment and control (Testing). For the experiment the samples are used to analyze the behavior of the executables. The controls (test) samples

are used for testing the implemented algorithms. To further classify the samples, each sample was labeled and a statistical random number table was used to select the samples. Twelve percent of Malwares and twenty four percent of benign programs were used for experiments and the rest used for control (Test) samples.

Selecting what sample size must be used was a great challenge as the sample must be representative of the total population. In malware researches done before there in no rule or data that shows what the total population is or how the selection is made. The general trend followed by works described in the related works was experimenting with few viruses in a controlled environment and testing the result in a live environment. On this thesis a decision was made to make analysis on selected malware and see if the result can apply to a larger population and repeat these steps again and again.

Furthermore, the whitelisting concept emphasizes that behavioral analysis of a small sample must be applicable to a larger population of malware as the basics of replication of different kinds of malware are similar. Otherwise for new malware like blacklisting the details of it must be in the database. In [27] it is shown that by using replication and File I/O API alone it is possible to identify malwares. Therefore, 12% of malwares were selected to study their replication patterns. Since benign programs behavior varies much more than malwares 24 % (twice that of the malwares study sample) was selected for study.

CHAPTER VI

EXPERIMENTAL RESULTS, WHITELISTING ALGORITHM AND DISCUSSION

6.1 RESULTS

6.1.1 Registry Monitoring

The windows registry is extensively used to store application configuration information by both benign and malware programs. In the thesis more than 35 registry settings that enable application to auto start at boot time were considered. The finding is that benign program put only a single key in the registry in documented areas that include

- HKLM/SOFTWARE/MICOSOFT/WINDOWS/CURRENTVERSION/RUN
- HKCU/SOFTWARE/MICOSOFT/WINDOWS/CURRENTVERSION/RUN

Most of the Malwares put a key in these areas. However, what makes Malwares especial is that more than 95% of Malwares considered make more than one reference in the registry at auto start location. These behaviors alone in collaboration with check summing enable us to isolate malware since no benign program makes multiple copies of itself run during start up. When this behavior is coupled with I/O and API Tracing a Highly accurate detection of Malwares is possible.

6.1.2 Tracing application Activity Using the KrView Program

Benign and Malware programs activity was traced using the krview program. The trace was done in 30 second snapshots for 3 Minutes. The output was a parameter reading for both the system and specified program that include

- Kernel and user time
- I/O Read, Write and Other
- Context Switches
- Page Faults

Context Switches and Page Faults traces for malware and benign programs was discarded as the reading for this does not provide any distinctive patterns that enable us to differentiate between the two. Furthermore, the value for these readings varies for a given application depending on number of programs running and available free space.

Kernel and User time data are characteristic of a program that has little dependence on external parameters. However the traces do not provide any distinctive margins to differentiate between malware and benign programs. Upon closer investigation the reason for this was that most Windows applications depend on the functionality provided by the windows API. Most functions in the API Execute in Kernel Space. Therefore the Time a program spends in user or kernel time do not provide clear cut margins as it is used by both kinds of programs.

The I/O Traces considered are I/O Read, I/O write and I/O other. I/O read is the number read operations a process performs within a given period of time. I/O write is the number of write operations a process does within a given period of time. The I/O other is a measure of the number of control messages exchanged issued or received by a process within a given period of time.

I/O Traces were very effective in depicting clear cut margins between malware and benign programs. The raw data trace does not provide clear cut margins. However, statistical analysis on the data provides clear cut margins. The statistical analysis Performed was Coefficient of Variance (CV).

The coefficient of variation (CV) is a normalized measure of dispersion of a probability distribution. It is defined as the ratio of the Standard deviation (σ) to the mean (μ) [19]

$$CV = \frac{\sigma}{\mu} \quad \mu \neq 0$$

Equation 6. 1 Coefficient of Variance

The Coefficient of Variance was calculated for both benign and malware programs for their I/O Traces. Figures 6.1, 6.2 and 6.3 show the I/O read, write and other traces. Most benign programs have CV value greater than 50. In contrast all Malwares have a CV value less than 20.

Based on experimental data the Cut off Margin (I/O Threshold) to decide if a program is benign or malware is taken to be CV=25. The value 25 is taken as an initialization value for the whitelisting algorithm. This value may go up or down in unit step values based on the PC configuration the algorithm is running and scanning results.

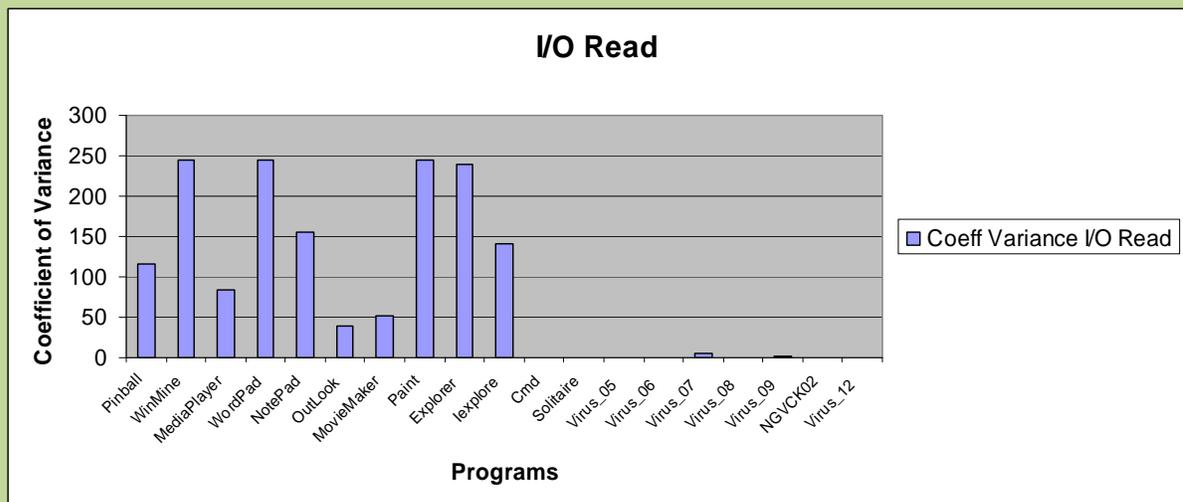


Figure 6.1 I/O READ Operation Performed by Benign and Malware programs

The reason for selecting to move the CV Threshold by unit step up or down is that the margins vary by large margins not in the range of decimal numbers. Considering the classification between a malware and benign program based on margins separated by decimal difference is very close. The experimental data of I/O Traces for malwares and benign programs shows that the difference between their CV Values is more than 40. Therefore, while designing the algorithm taking a unit step was logical incase in real time environment the difference become closer. However, there is not limit in making the difference margin lower or higher.

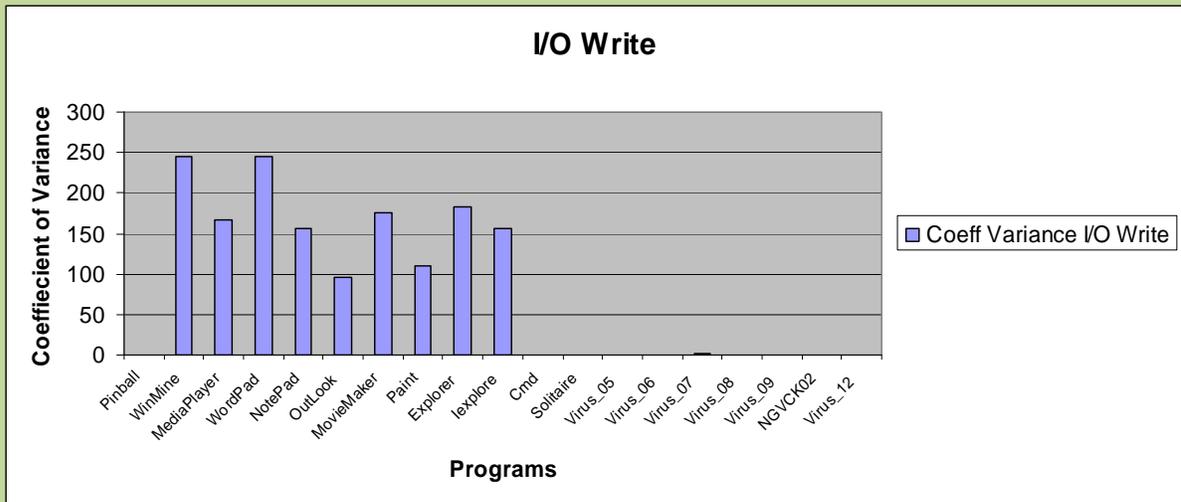


Figure 6.2 I/O Write Operation Performed by Benign and Malware programs

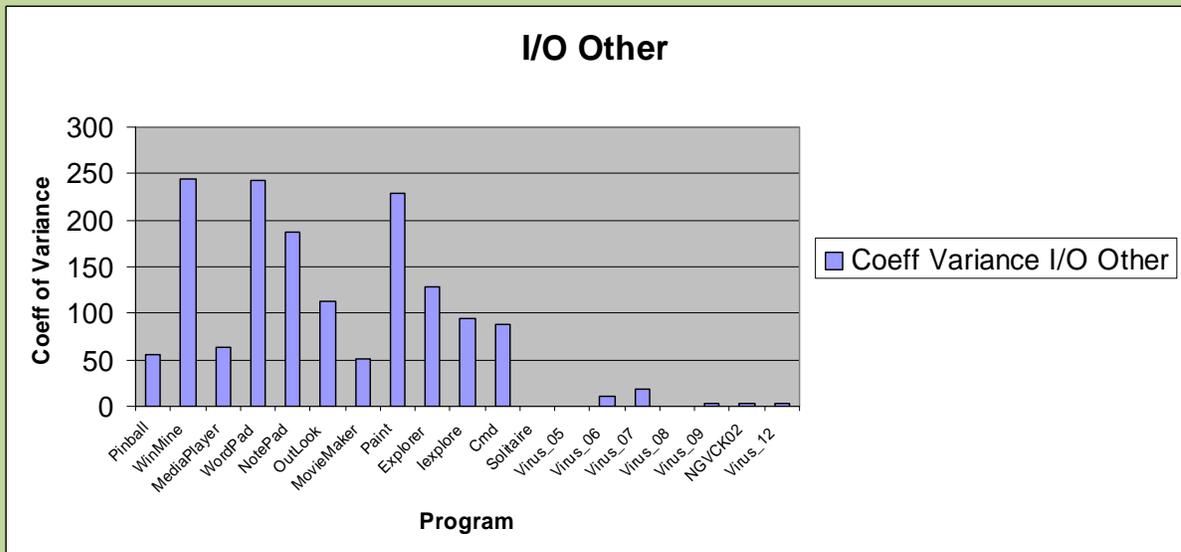


Figure 6.3 I/O Other Operation Performed by Benign and Malware programs

Some benign programs like command prompt and the windows game solitaire have $CV < 25$. On first approach Benign Programs having a $CV < 25$ are classified as Malware. However this confusion can be solved by closely looking at the raw I/O trace and the behavior and definition of CV. The CV is defined only for μ not equal to Zero.

A Whitelist based implementation of Anti-Virus definition file to detect unknown malicious activity

As can be seen in Table 6.1, 6.2 and 6.3 we need to differentiate between CV value equal to zero and CV value undefined. CV value can be zero if all our readings in a given time window are all equal. This will make the standard deviation equal to zero. CV value will be undefined if all our reading are all zero.

Therefore readings with CV undefined are exception to the rule $CV < 25$. In such cases, the readings from I/O read, write and other must be considered collectively. When this rule is adopted, solitaire and command program will not be classified as malware. This rule must be applied to any program under scrutiny. Table 6.1, 6.2 and 6.3 also show snapshots taken in 30 second intervals for 3 Minutes and how CV is calculated.

Element	30(S)	60(S)	90(S)	120(S)	150(S)	180(S)	STD DEV	AVG	CV
I/O Read Operations	0	0	0	0	0	0	0	0	#DIV/0!
I/O Write Operations	0	0	0	0	0	0	0	0	#DIV/0!
I/O Other Operations	170	4	0	210	70	156	89.70544	101.6667	88.23486

Table 6.1 Command Prompt Trace at 30s Snapshot Interval

Element	30(S)	60(S)	90(S)	120(S)	150(S)	180(S)	STD DEV	AVG	CV
I/O Read Operations	0	0	0	0	0	0	0	0	#DIV/0!
I/O Write Operations	0	0	0	0	0	0	0	0	#DIV/0!
I/O Other Operations	0	0	0	0	0	0	0	0	#DIV/0!

Table 6.2 Solitaire Trace at 30s Snapshot Interval

Element	30(S)	60(S)	90(S)	120(S)	150(S)	180(S)	STD DEV	AVG	CV
I/O Read Operations	0	0	0	0	0	0	0	0	#DIV/0!
I/O Write Operations	30	30	30	30	30	30	0	30	0
I/O Other Operations	1210	1210	1210	1210	1210	1210	0	1210	0

Table 6.3 Virus_08's Trace at 30s Snapshot Interval

6.1.3 API TRACE

API Trace (System Calls in UNIX) has undergone extensive research since 1996. Experimental results from different researchers as described in literature review have shown that API Traces can indicate malicious activity. Generally, most researches focused on Analyzing sequence of system calls using Machine learning algorithms including Neural Networks and Hidden Markov Model. In addition Data Mining methods have been extensively been experimented with. The drawback of this method was training the network or analyzing the data took a longer time that render them less attracting to real time scanning engines. In this paper the API traces are analyzed statistically rather than going through machine learning algorithm to find a clear cut margin between benign and malware programs.

API Traces can be categorized into groups based on their functionality – drawing, multimedia, File I/O etc. In this paper we have made a focus on the FILE I/O API Functions as they are the ones that contribute to the survival, transmission and replication of a malware [27].

Spy Studio software was used to trace API I/O functions activity. The trace was taken in 30second interval (1 Snapshot) for 3 Minutes.

To calculate the overall Frequency of An Api Trace (OFT) for n Snapshots.

$$\text{OFT} = \frac{SS_n + SS_{n-1} + SS_{n-2} + \dots + SS_1}{n}$$

Equation 6.2 File I/O API Call Running Frequency calculation

SS_n is the sum for snapshot 'n'. It is a sum of the number of times I/O API functions are called during the observation period (30 seconds). The average of all snapshots taken is called 'Overall Frequency Trace' (OFT). OFT is simple the average of SS_n 's in the observation period. From experimental data any value greater than 150 (API Threshold) is considered as malware. This value will be taken as a starting value for the algorithms. Based on the PC configuration this value may go up or down by the whitelist building algorithm. Table 6.4 Show a Single Snapshot of some applications and I/O APIs. It also shows how SS_n is calculated.

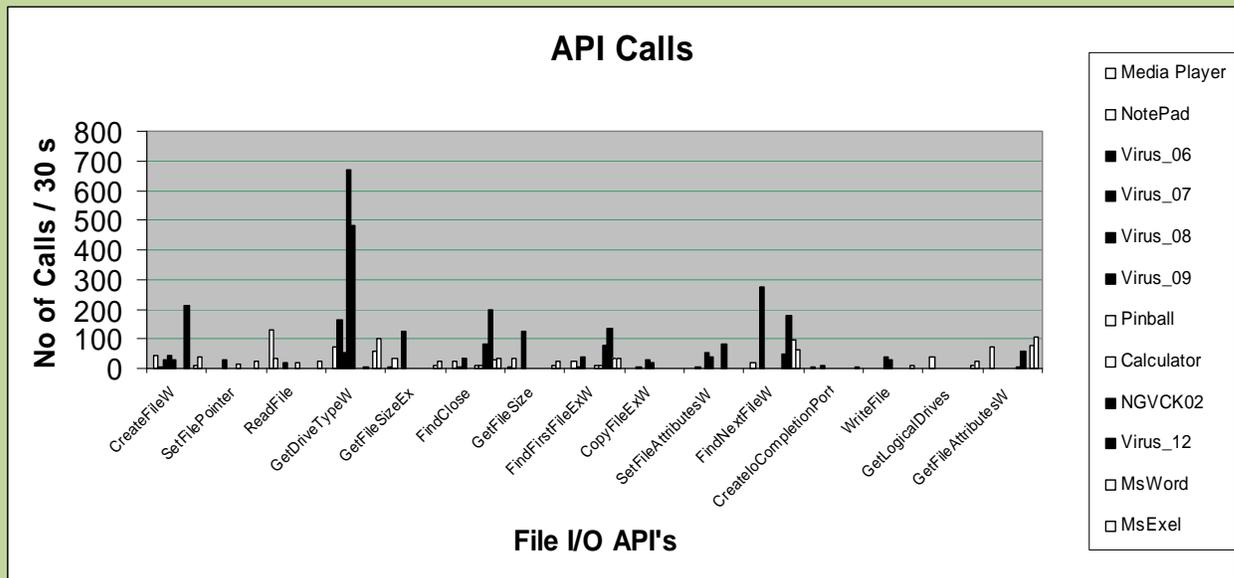


Figure 6.4 File I/O API Trace for Benign and Malware Programs at 30s Snapshot

In table 6.4 the data for a single snapshot is shown. From the data It can be seen that benign programs can have SS_n value greater than 150, for example excel with SS_n 447. This happens because during the snapshot time of 30 seconds the user was saving the document. This brings us to the conclusion that a decision on a processes whether it is a malware or not cannot be made on a single snapshot alone. Rather, consecutive snapshots must be taken and the average calculated and the decision must be based on that average as shown by equation 6.2.

	Media Player	NotePad	PinBall	Calculator	Virus_06	Virus_07	Virus_08	Virus_09	NGVCK02	Virus_12	MSWord	MSExcel
CreateFileW	0	45	0	0	7	28	41	30	210	0	12	38
SetFilePointer	0	0	14	0	0	31	0	0	0	0	26	0
ReadFile	130	32	20	0	0	17	0	0	0	0	0	25
GetDriveTypeW	0	70	0	0	164	52	668	480	6	0	59	101
GetFileSizeEx	4	32	0	0	0	127	0	0	1	0	12	25
FindClose	0	23	11	9	7	34	0	0	81	198	28	35
GetFileSize	4	32	0	0	0	127	0	0	0	0	12	25
FindFirstFileExW	0	23	11	10	7	37	0	0	76	134	34	35
CopyFileExW	0	0	0	0	7	0	28	20	0	0	0	0
SetFileAttributesW	0	0	0	0	6	0	52	40	80	0	2	0
FindNextFileW	0	20	0	0	0	276	0	0	50	179	97	62
CreateIoCompletionPort	0	5	0	0	0	9	0	0	0	0	2	3
WriteFile	0	0	0	0	0	0	39	30	0	0	11	0
GetLogicalDrives	0	38	0	0	0	0	0	0	2	0	8	24
GetFileAttributesW	0	71	0	3	0	0	0	0	60	0	76	104
SUM (SnapShot SSn)	138	391	56	22	198	738	828	600	566	511	379	477

Table 6.4 File I/O API Tracing for Benign and Malware Programs for a one snapshot

6.2 Algorithms for Whitelisting

In this section algorithms that enable us to build a whitelist database as well as scanning algorithm will be presented. The algorithms were implemented based on the results described in this chapter earlier. The whitelist algorithm will be presented first then the scanning algorithm will be put forward followed by the decision sub system. I/O and API Tracing Parts of the Scanning algorithm details will be shown separately. It is assumed that when the whitelist algorithm runs for the first time, the machine it is running on will be free from malware.

6.2.1 WhiteListing Algorithm

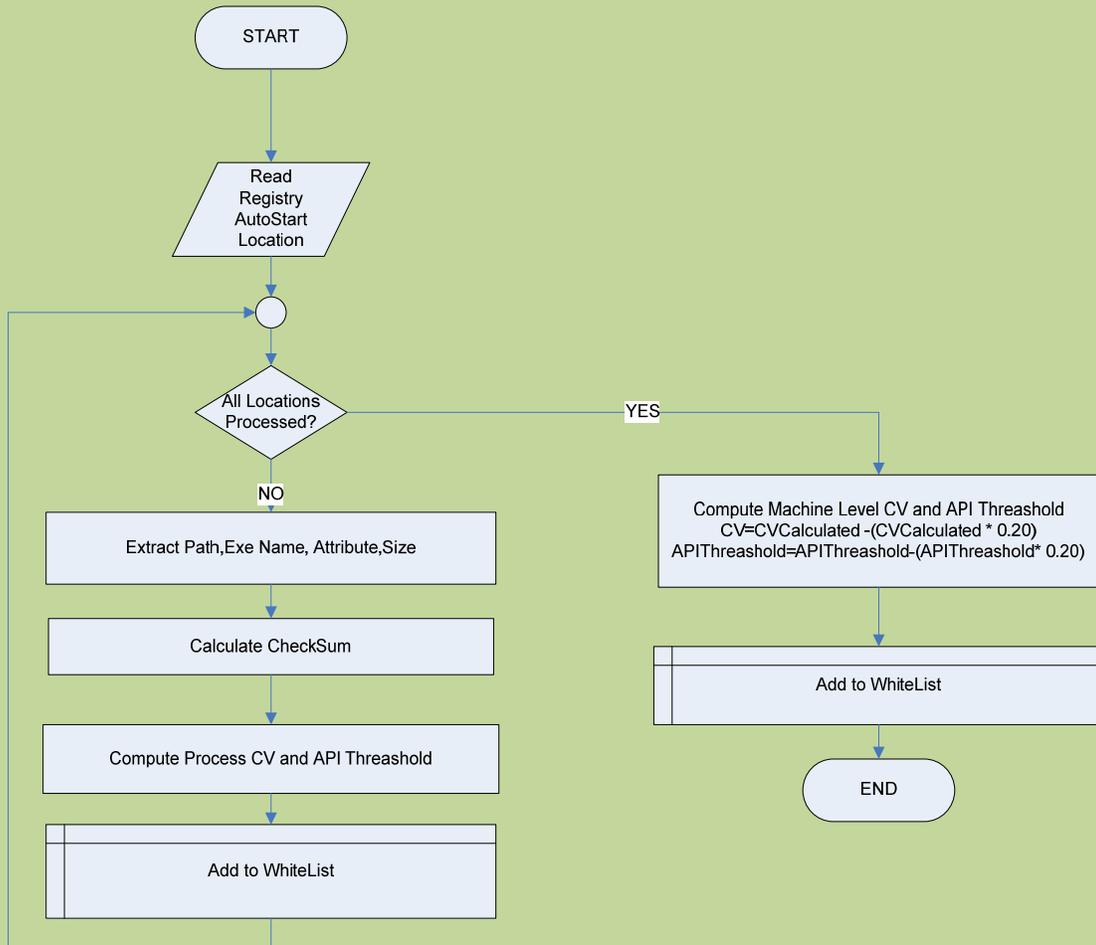


Figure 6.5 WhiteListing Algorithm

6.2.2 Scanning Algorithm

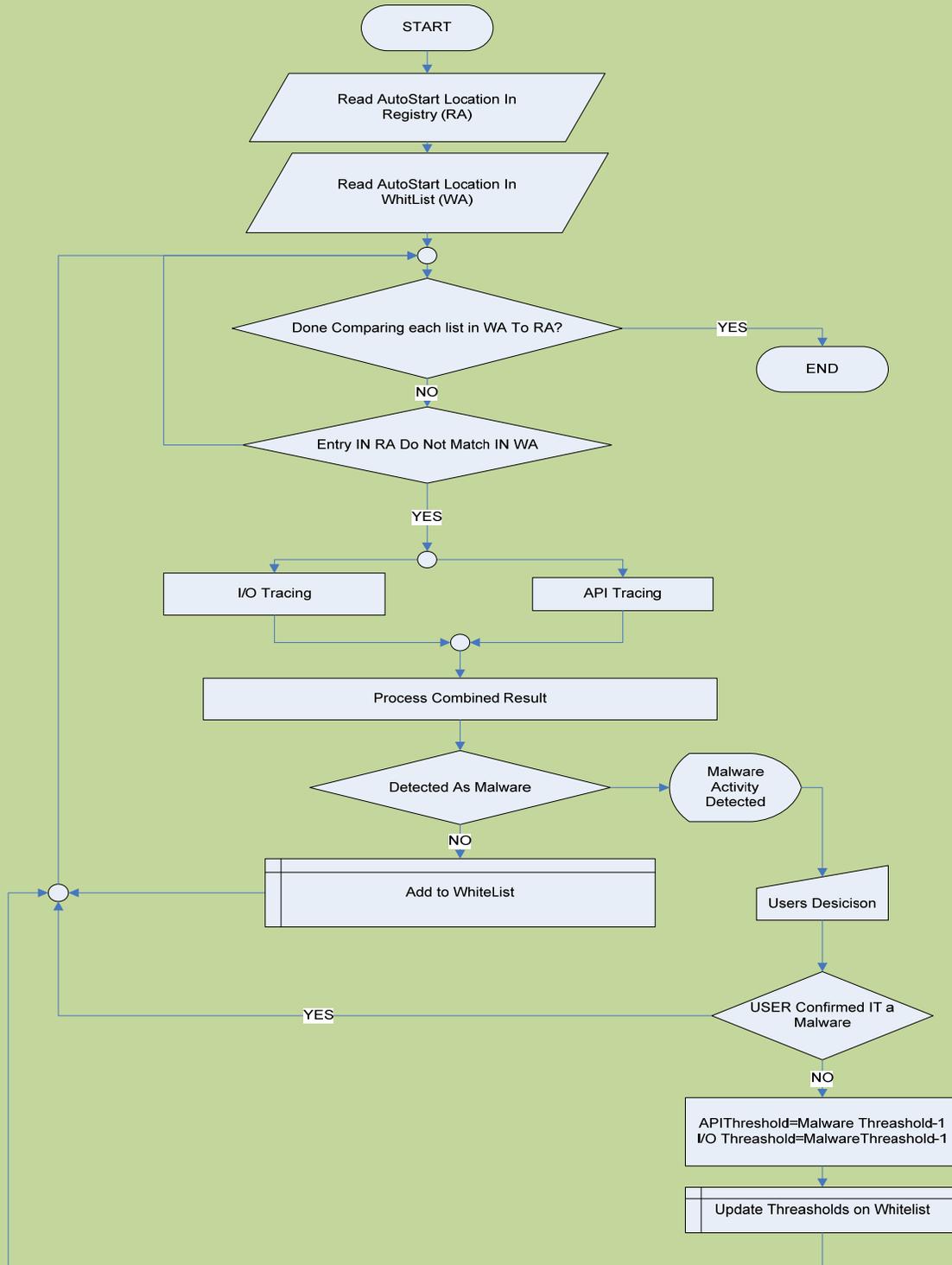


Figure 6.6 Scanning Algorithm

6.2.3 I/O Algorithm

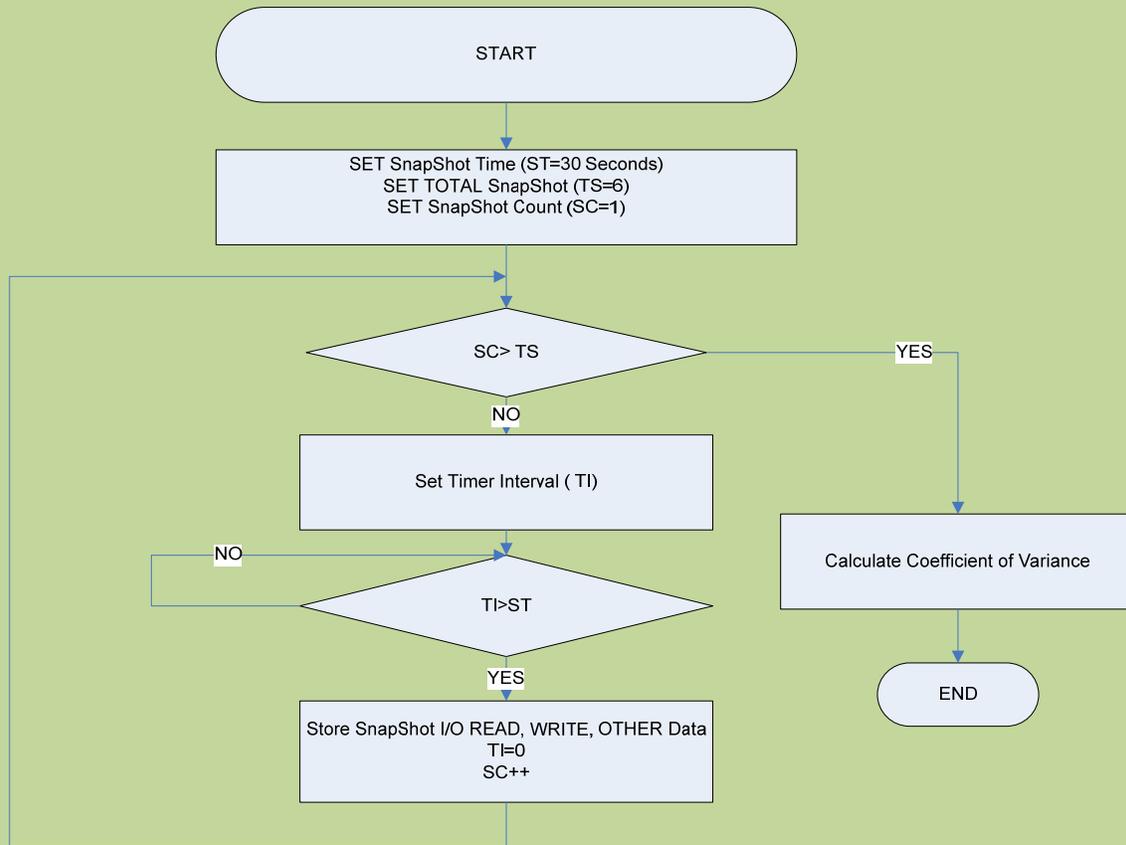


Figure 6.7 I/O Tracing Algorithm

6.2.4 API Tracing

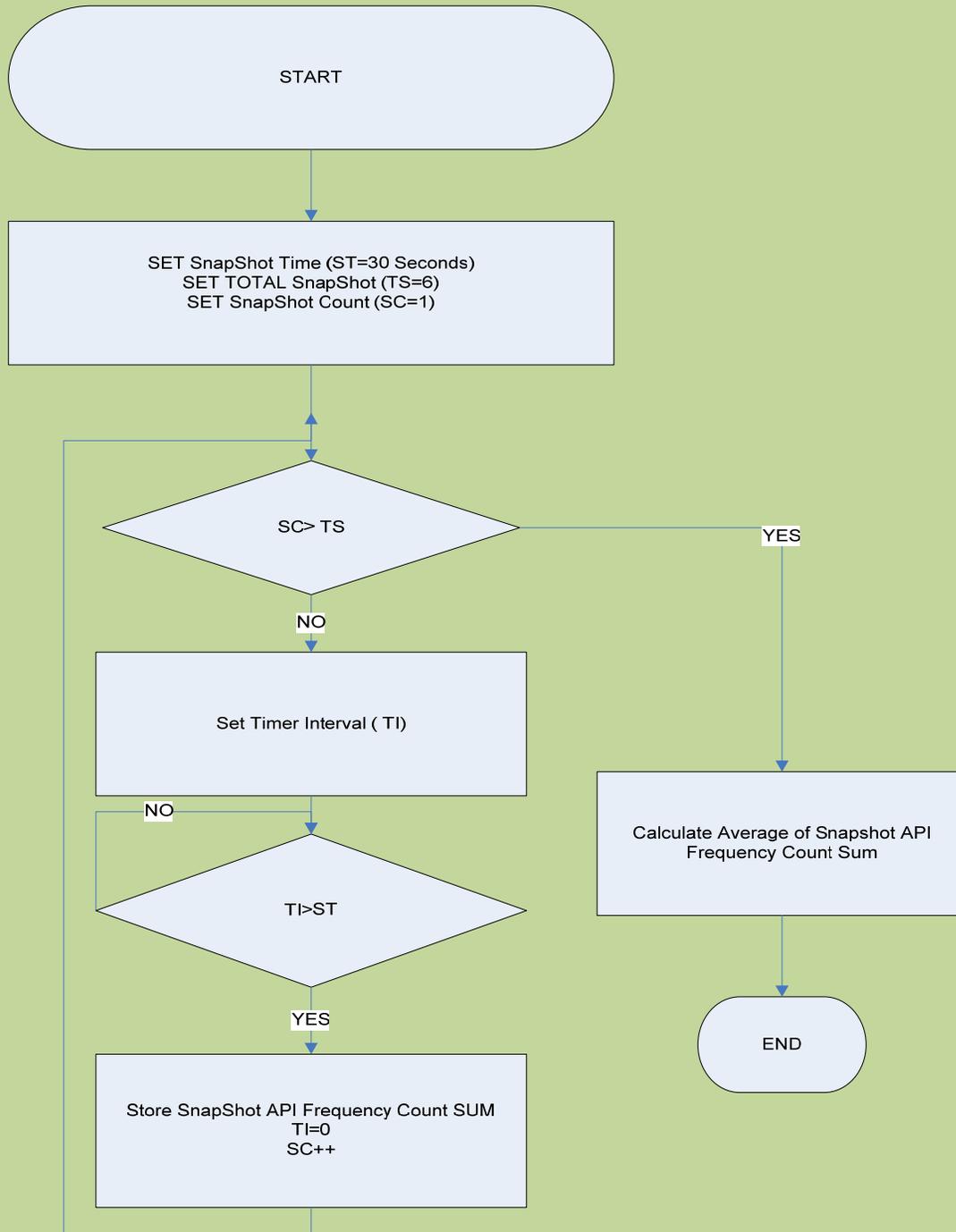


Figure 6.8 API Tracing Algorithms

6.2.5 Decision Algorithm

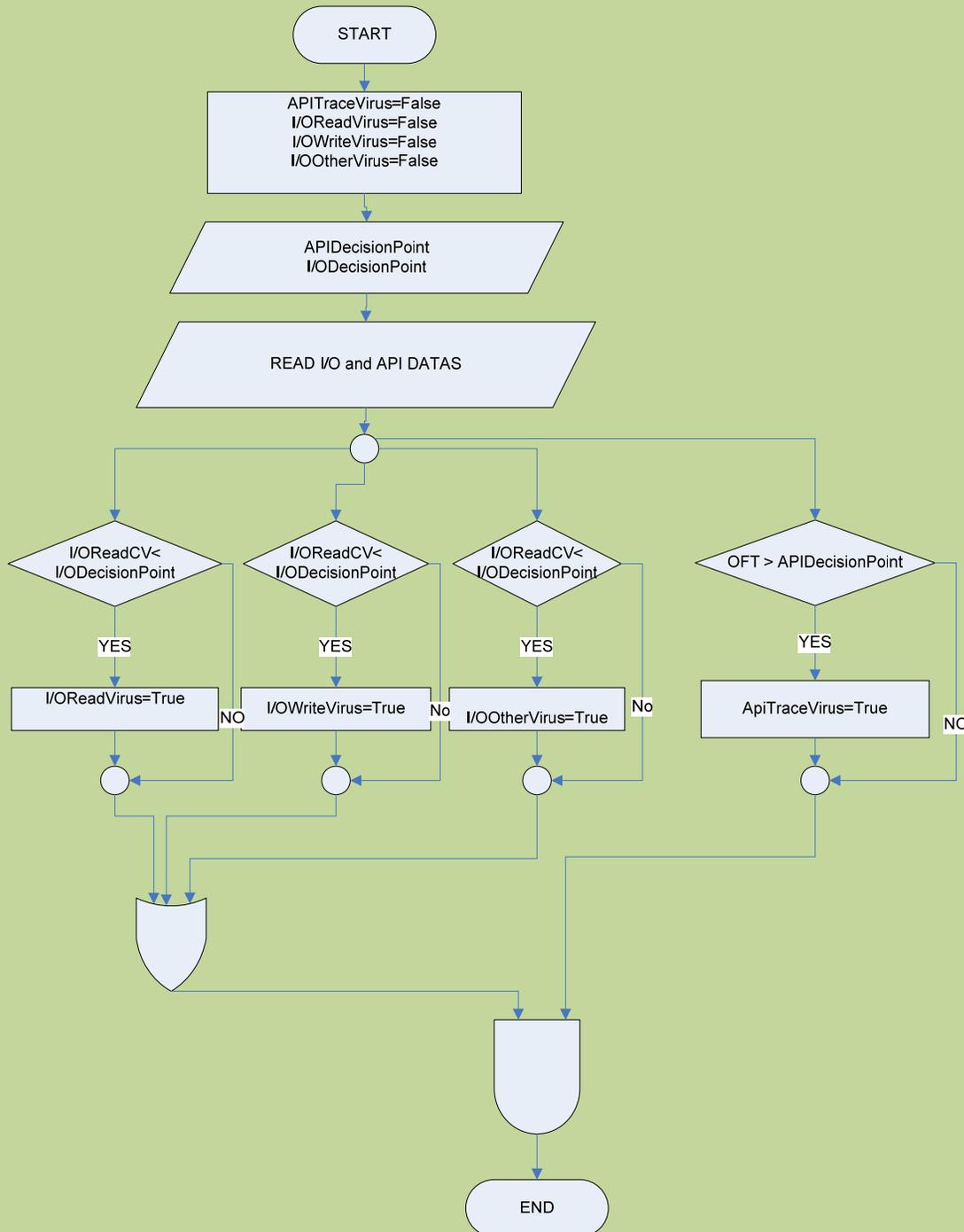


Figure 6.9 Decision Making Algorithm

6.3 Implemented WhiteList Definition File Detection comparisons

To compare the detection rate of unknown malware on the thesis whitelist database with commercial software, latest commercial antivirus were collected. Only malware samples from NGVCK Kit were considered because we are sure that these Malwares from this KIT do not exist in the blacklist (signature) database. However, the 24 Live Malwares were not tested because there is no way of knowing whether the signature or heuristics engine caught the malware. The Live Malwares were used to test the thesis scanning engine and they were all detected by it.

To start with comparison a user level non-real time program was written that implements a whitelist definition file based on the research results above. The thesis testing program builds a WhiteList database for the VIRTUAL-PC it is running on. Then the VIRTUAL-PC is infected with viruses and benign programs were loaded and the thesis test program and commercial antivirus software's were run to observe their detection rate.

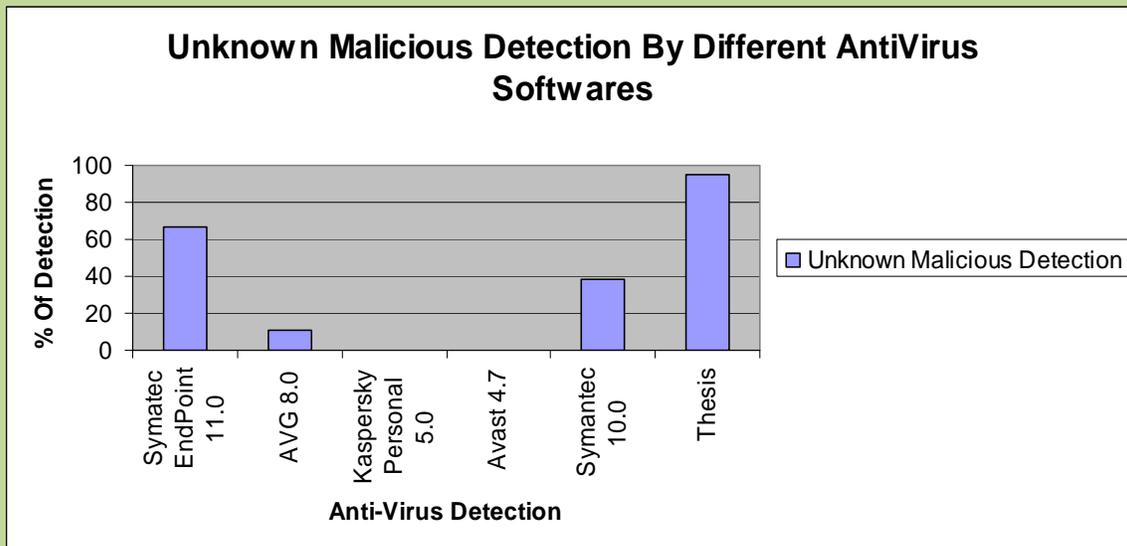


Figure 6.10 Unknown Malicious Malware Detection among Different products

CHAPTER VII

CONCLUSION, RECOMMENDATION AND FUTURE WORK

In this thesis work we have considered a whitelist approach of malware detection. In the research we have tried to test different parameters that enable us to profile processes. Among these registry, File System monitoring, I/O and API Tracing were effective. Although each of the methods enables us to detect malware individually, when combined and working together the method can detect all Malwares, detecting 90 % of malwares in the sample.

In comparison to the heuristics technologies in commercial antivirus systems, the approach has outperformed them all for the test samples. Although whitelist method of detecting of malware is not new up to this point, Neural networks, Hidden Markov Models or Data mining approaches are the existing theoretical methods. These approaches worked at laboratory settings but never made it to the commercial world- the time and complexity required making them harder to manage. The thesis has shown that this whitelist approach can be implemented using a different approach to the problem, that is encouraging. In addition the whitelist database is dynamically built on the target PC.

In the research we have tested 68 Malwares that include Viruses, Worms and Trojans. The results show that this method can be effective in the real world if integrated in Heuristics engines that use CPU emulation. The thesis approach was able to detect 90% of the sample while the next higher detection was 68%.

Recommendation and Future Works

The promising and encouraging results of this thesis have shown that the approach is viable. To facilitate the integration and stability of the approach more testing on a large number of Malwares and on a working environment needs to be conducted. The thesis considered Viruses, Worms and Trojans. However, its suitability to other classes of Malwares such as Adwares, rootkits needs to be investigated. In addition the approach will not be able to detect

very slowly moving malware that infect particular files (AutoCAD or MAYA Non-Executable Files). Apart from this the effect of varying the snapshot time of 30 seconds and Observation period of 180 seconds needs further investigation.

SUMMARY

- Combination of Registry Monitoring with API and I/O Tracing Enables us to detect Unknown Malwares
- This Methodology can best be Integrated in Heuristics Scanners
- The Algorithm Needs Practical Implementation and Testing on More malware and Real Environment
- The Algorithm needs further Work in areas of Slow Moving Viruses like those attacking only Auto CAD or Maya Files (Non Executable) Only.
- 180 Seconds for monitoring processes may need to be reconsidered or be made variable. To make a fast detection engine it needs to be lowered. However, to detect time bombs it needs to be extended more. A compromise between these extremities needs to be made.

APPENDIX

APPENDIX I: Top 10 Viruses in History

Top 10 viruses

Which viruses are the most successful ever? Here is a Selection of those that traveled furthest, infected most computers ... or survived the longest.

Love Bug

(VBS/LoveLet-A)

The Love Bug is probably the best-known virus. By pretending to be a love letter, it played on users' curiosity, spreading around the world in hours.

First seen: May 2000

Origin: The Philippines

AKA: Love Letter

Type: Visual Basic Script worm

Trigger: On initial infection

Effects: The original version sends an email with the subject line 'I LOVE YOU' and the text 'kindly check the attached love letter coming from me'. Opening the attachment allows the virus to run. If Microsoft Outlook is installed, the virus tries to forward itself to all addresses in the Outlook address book. It can also distribute itself to other newsgroup users, steal user information and overwrite certain files.

Form

Form featured in the top ten viruses for eight years and is still widespread. On DOS and early versions of Windows, it behaved inconspicuously, so it spread widely.

First seen: 1991

Origin: Switzerland

Type: Boot sector virus

Trigger: 18th of the month

Effects: Produces a click every time you press a key; can prevent Windows NT computers from working.

Kakworm

(VBS/Kakworm)

Kakworm made it possible for users to become infected just by viewing infected email.

First seen: 1999

Type: Visual Basic

Script worm

Trigger: On initial infection (for most effects) or 1st of any month (for Windows shutdown side-effect)

Effects: The worm arrives embedded in an email message. If you are using Outlook or Outlook Express with Internet Explorer 5, the machine can be infected when you open or preview the infected email. The virus changes the Outlook Express settings so that the virus code is automatically included with all outgoing mail. On the 1st of any month after 5 pm, it displays the message ‘Kagou-Anti_Kro\$oft says not today’ and shuts down Windows.

Anticmos

Anticmos is a typical boot sector virus. It was widespread in the mid-1990s and frequently appeared in the top ten viruses.

First seen: January 1994

Origin: First detected in Hong Kong, but believed to originate in China.

Type: Boot sector virus

Trigger: Random

Effects: Tries to erase information about the type of floppy and hard disk drives installed.

Melissa

(WM97/Melissa)

Melissa is an email virus that uses psychological subtlety to spread rapidly. It appears to come from someone you know and to include a document you would definitely want to read. As a result, Melissa spread worldwide within a single day.

A Whitelist based implementation of Anti-Virus definition file to detect unknown malicious activity

First seen: March 1999

Origin: A 31-year-old US programmer, David L Smith, posted an infected document on an alt.sex usenet newsgroup

Type: Word 97 macro virus; also Word 2000 aware

Trigger: On initial infection

Effects: Sends a message to the first fifty addresses in all the address books accessible by Microsoft Outlook, using the current user's name in the subject line. There is an attachment containing a copy of the infected document. If the minute and day are the same when the document is opened (e.g. 10.05 am on the 5th), the virus adds text about the game Scrabble to the document.

New Zealand

New Zealand was easily the commonest virus in the early 1990s.

First seen: Late 1980s

Origin: New Zealand

AKA: Stoned

Type: Boot sector virus

Trigger: Once in 8 times, if booted from a floppy

Effects: Displays the message 'Your PC is now Stoned!', Puts a copy of the original boot sector in a sector that is last in the root directory of a 360K disk. This can damage larger disks.

Concept

(WM/Concept)

Concept achieved instant success by being shipped accidentally on official Microsoft software. It was the first macro virus found in the wild and one of the commonest viruses in 1996-1998. The virus takes control with its AutoOpen macro, which Word runs automatically, and carries out infection with its FileSaveAs macro, which runs when Word saves a document. Many variants exist.

First seen: August 1995

Type: Macro virus

Trigger: None

A Whitelist based implementation of Anti-Virus definition file to detect unknown malicious activity

Effects: When you open an infected document, a dialog box titled 'Microsoft Word' and containing the figure 1 appears. The virus includes the text 'That's enough to prove my point' but this is never displayed.

CIH (Chernobyl)

(W95/CIH-10xx)

CIH was the first virus to damage computer hardware. Once it overwrites the BIOS, the computer cannot be used until the BIOS chip is replaced.

First seen: June 1998

Origin: Written by Chen Ing-Hau of Taiwan

Type: Parasitic virus that runs on Windows 95 computers

Trigger: April 26th, with variants which trigger on June 26th or the 26th of any month

Effects: Tries to overwrite the BIOS and then overwrites the hard disk.

Parity Boot

Parity Boot spreads on the boot sectors of floppy disks. Its success shows that boot sector viruses, which were commonest in the 1980s and early 1990s, can still thrive. This virus was still among the most commonly reported as recently as 1998. It was particularly common in Germany, where it was distributed on a magazine cover-disk in 1994.

First seen: March 1993

Origin: Possibly Germany

Type: Boot sector virus

Trigger: Random

Effects: Displays the message 'PARITY CHECK' and freezes the computer. This mimics a genuine memory error. As a result, users often think that there is a problem with their computer's RAM (Random Access Memory).

Happy99

(W32/Ska-Happy99)

Happy99 was the first well-known virus to spread itself rapidly by email.

First seen: January 1999

Origin: Posted to a newsgroup by French virus writer 'Spanska'

A Whitelist based implementation of Anti-Virus definition file to detect unknown malicious activity

Type: File virus that runs on Windows 95/98/Me/NT/2000 computers

Trigger: None

Effects: Displays a fireworks effect and the message 'Happy New Year 1999'. The virus also modifies the file wsock32.dll in the Windows system directory so that whenever an email is sent, a second message including the virus is sent too.

APPENDIX II : Registry and File Auto Start Locations considered in the thesis

Registry Autostart Locations

1. KEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run\ All values in this key are executed.
- 2.HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunOnce\ All values in this key are executed, and then their autostart reference is deleted.
- 3.HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunServices\ All values in this key are executed as services.
- 4.HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunServicesOnce\ All values in this key are executed as services, and then their autostart reference is deleted.
5. KEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run\ All values in this key are executed.
- 6.HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunOnce\ All values in this key are executed, and then their autostart reference is deleted.
- 7.HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunOnce\Setup\ Used only by Setup. Displays a progress dialog box as the keys are run one at a time.
8. HKEY_USERS\.Default\Software\Microsoft\Windows\CurrentVersion\Run\ Similar to the Run key from HKEY_CURRENT_USER.
- 9.HKEY_USERS\.Default\Software\Microsoft\Windows\CurrentVersion\RunOnce\ Similar to the RunOnce key from HKEY_CURRENT_USER.
- 10.HKEY_LOCAL_MACHINE\Software\Microsoft\WindowsNT\CurrentVersion\Winlogon The "Shell" value is monitored. This value is executed after you log in.
11. HKEY_LOCAL_MACHINE\Software\Microsoft\Active Setup\Installed Components\ All subkeys are monitored, with special attention paid to the "StubPath" value in each subkey.

12. HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\VxD\ All subkeys are monitored, with special attention paid to the "StaticVXD" value in each subkey.
13. HKEY_CURRENT_USER\Control Panel\Desktop The "SCRNSAVE.EXE" value is monitored. This value is launched when your screen saver activates.
14. HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Session Manager The "BootExecute" value is monitored. Files listed here are Native Applications that are executed before Windows starts.
15. HKEY_CLASSES_ROOT\vbsfile\shell\open\command\ Executed whenever a .VBS file (Visual Basic Script) is run.
16. HKEY_CLASSES_ROOT\vbfile\shell\open\command\ Executed whenever a .VBE file (Encoded Visual Basic Script) is run.
17. HKEY_CLASSES_ROOT\jsfile\shell\open\command\ Executed whenever a .JS file (Javascript) is run.
18. HKEY_CLASSES_ROOT\jsefile\shell\open\command\ Executed whenever a .JSE file (Encoded Javascript) is run.
19. HKEY_CLASSES_ROOT\wshfile\shell\open\command\ Executed whenever a .WSH file (Windows Scripting Host) is run.
20. HKEY_CLASSES_ROOT\wsffile\shell\open\command\ Executed whenever a .WSF file (Windows Scripting File) is run.
21. HKEY_CLASSES_ROOT\exefile\shell\open\command\ Executed whenever a .EXE file (Executable) is run.
22. HKEY_CLASSES_ROOT\comfile\shell\open\command\ Executed whenever a .COM file (Command) is run.
23. HKEY_CLASSES_ROOT\batfile\shell\open\command\ Executed whenever a .BAT file (Batch Command) is run.
24. HKEY_CLASSES_ROOT\scrfile\shell\open\command\ Executed whenever a .SCR file (Screen Saver) is run.
25. HKEY_CLASSES_ROOT\piffile\shell\open\command\ Executed whenever a .PIF file (Portable Interchange Format) is run.

26. HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\ Services marked to startup automatically are executed before user login.
- 27.HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\Winsock2\Parameters\Protocol_Catalog\Catalog_Entries\ Layered Service Providers, executed before user login.
28. HKEY_LOCAL_MACHINE\System\Control\WOW\cmdline Executed when a 16-bit Windows executable is executed.
- 29.HKEY_LOCAL_MACHINE\System\Control\WOW\wowcmdline Executed when a 16-bit DOS application is executed.
- 30.HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Winlogon\Userinit Executed when a user logs in.
- 31.KEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\ShellServiceObjectDelayLoad\ Executed by explorer.exe as soon as it has loaded.
- 32.HKEY_CURRENT_USER\Software\Microsoft\Windows NT\CurrentVersion\Windows\run Executed when the user logs in.
- 33.HKEY_CURRENT_USER\Software\Microsoft\Windows NT\CurrentVersion\Windows\load Executed when the user logs in.
- 34.HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer\run\ Subvalues are executed when Explorer initialises.
- 35.HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer\run\ Subvalues are executed when Explorer initialises.

Folder Autostart Locations

1. *windir*\Start Menu\Programs\Startup\
2. User\Startup\
3. All Users\Startup\
4. *windir*\system\iosubsys\
5. *windir*\system\mmm32\

6. *windir*\Tasks\

File Autostart Locations

1. c:\explorer.exe
2. c:\autoexec.bat
3. c:\config.sys
4. *windir*\wininit.ini
5. *windir*\winstart.bat
6. *windir*\win.ini - [windows] "load"
7. *windir*\win.ini - [windows] "run"
8. *windir*\system.ini - [boot] "shell"
9. *windir*\system.ini - [boot] "scrnsave.exe"
10. *windir*\dosstart.bat
11. *windir*\system\autoexec.nt
12. *windir*\system\config.nt

APPENIX III : Ten steps to safer computing

Apart from using anti-virus software, there are plenty of simple measures you can take to help protect yourself and your company from viruses. Here are our ten top tips for trouble-free computing.

1. Don't use documents in .doc and .xls format or Disable Macro feature

Disable the macro feature in Microsoft Office. Otherwise Save your Word documents in RTF (Rich Text Format) and your Excel spreadsheets as CSV (Comma Separated Values) files. These formats don't support macros, so they cannot spread macro viruses, which are by far the commonest virus threat. Tell other people to supply you with RTF and CSV files. Beware, though! Some macro viruses intercept FileSaveAs RTF and save the file with an RTF extension but DOC format. To be absolutely safe, use text-only files.

2. Don't launch unsolicited programs or documents

If you don't know that something is virus-free, assume it isn't. Tell people in your organization that they should not download unauthorized programs and documents, including screensavers or 'joke' programs, from the internet. Have a policy that all programs must be authorized by an IT manager and virus-checked before they are used.

3. Forward warnings to one authorized person only

Hoaxes are as big a problem as viruses themselves. Tell users not to forward virus warnings to their friends, colleagues or everyone in their address book. Have a company policy that all warnings go to one named person or department only.

4. Block files with double extensions at the gateway

Some viruses disguise the fact that they are programs by using a 'double extension', such as .TXT.VBS, after their filename. At first glance a file like LOVE-LETTER-FORYOU.TXT.VBS or ANNAKOURNIKOVA.JPG.VBS may seem to be a harmless text file or a graphic. You should block any file with double extensions at the email gateway.

5. Block unwanted file types at the email gateway

Many viruses now use VBS (Visual Basic Script) and Windows scrap object (SHS) file types to spread. It is unlikely that your organisation needs to receive these file types from outside, so block them at the email gateway.

6. Change your computer's bootup sequence

Most computers try to boot from floppy disk (the A: drive) first. Your IT staff should change the CMOS settings so that the computer boots from the hard disk by default. Then, even if an infected floppy is left in the computer, it cannot be infected by a boot sector virus. If you need to boot from floppy at any time, you can have the settings changed back.

7. Write-protect floppies/USB (if there is any) before giving to other users

A Whitelist based implementation of Anti-Virus definition file to detect unknown malicious activity

A write-protected floppy cannot be infected.

8. Follow software companies' security bulletins

Watch out for security news and download patches to protect against new virus threats.

9. Subscribe to an email alert service

An alert service can warn you about new viruses and offer virus identities that will enable your anti-virus software to detect them.

10. Make regular backups of all programs and data

If you are infected with a virus, you will be able to restore any lost programs and data.

APPENIX IV: Malwares and Benign Programs Used

Real World Malwares Used (Kaspersky Classification)

1. Torjan-Downloader.Win32.Agent.pdl
2. Worm.Win32.Autorun.rht
3. Worm.Win32.Autorun.dsf
4. Trojan.Win32.QQPass.wk
5. Trojan.Win32.Agent.dys
6. Trojan.win32.startpage.ajh
7. Worm.win32.fujack.ass
8. AdobeRd9
9. Worm.win32.Autorun.djt
10. Trojan-psw.win32.onlinegames.acgo
11. Torjan-psw.win32.onlinegames.xlg
12. Worm.win32.Autorun.dve
13. Torjan-psw.win32.onlinegames.zzl
14. Costructor.Dos.G2
15. Virus.Vbs.Autorun.ad
16. Email-worm.win32.vb.cb
17. Virus.win32.Agent.cb
18. Worm.win32.Autorun.dve
19. Worm.win32.Autorun.dic
20. Worm.win32.Autorun.lub
21. Worm.win32.Autorun.ryf
22. Trojan program Trojan-GameThief.Win32.Magania.ahhn
23. Trojan program Trojan-GameThief.Win32.OnlineGames.tmkf
24. Virus.msword.Twno

NB. This real world malwares are classified as VIRUS_XXX in the thesis.

Benign Programs

1. Notepad 5.1
2. Wordpad 5.1
3. pinball 5.1
4. Media Player 9.0
5. command prompt 5.1
6. Explorer 5.1
7. RegSnap 1.0
8. Microsoft Management Console 5.1
9. Hash Calculator 1.0
10. Mines Weeper 5.1

11. Solitaire 5.1
12. Internet Explorer 6.0
13. Fire Fox 2.0
14. SQL Server Management Studio 2005
15. Windows Movie Maker 5.1
16. Windows Messenger 4.7
17. Outlook Express 6.0
18. Folder Healer 1.5
19. Tune Up Utilities 2007
20. Net Beans 5.5
21. Visual Studio 2005
22. Microsoft Paint 5.1
23. Clean Manager 5.1
24. Visual Studio 2005
25. Registry Editor 5.1
26. Microsoft Word 2003
27. Microsoft Excel 2003
28. Microsoft Access 2003
29. Microsoft PowerPoint 2003
30. Microsoft Project 2003
31. Microsoft FrontPage 2003
32. Microsoft Visio 2003
33. Microsoft Outlook 2003
34. Microsoft InfoPath 2003
35. Profidel V 1.5
36. Power Ge'ez 2005
37. Visual Ge'ez 2006
38. Macromedia Dream weaver 8
39. Microsoft SQL Server 2000
40. iTunes 7.5

41. Robo Help 5.0
42. Advanced Port Scanner 1.1
43. Installshield Professional 6.0
44. Nero Burner 7.0
45. Microsoft Flight Simulator 2003
46. Partition Magic 8.0
47. Download Accelerator 8.0
48. Need For Speed
49. Adobe Professional 7.0
50. Adobe Photoshop CS2

Next Generation Virus Creation Tool Generated Malwares (NGVCK)

All Malwares generated with this tool are names NGVCK_XXX

Where XXX is a serial Number

Eg :NGVCK_001,NGVCK_002 etc...

Malware Behaviour	The Malware ID			
Virus	NGVCK_001,	NGVCK_004,	NGVCK_007,	NGVCK_008,
	NGVCK_009,	NGVCK_010,	NGVCK_011,	NGVCK_012,
	NGVCK_013,	NGVCK_014,	NGVCK_015,	NGVCK_016,
	NGVCK_044			
Worm	NGVCK_002,	NGVCK_005,	NGVCK_017,	NGVCK_018,
	NGVCK_019,	NGVCK_020,	NGVCK_021,	NGVCK_022,
	NGVCK_023,	NGVCK_024,	NGVCK_025,	NGVCK_026,
	NGVCK_027,	NGVCK_028,	NGVCK_029,	NGVCK_030
Trojan horse	NGVCK_031,	NGVCK_032,	NGVCK_033,	NGVCK_034,
	NGVCK_035,	NGVCK_036,	NGVCK_037,	NGVCK_038,
	NGVCK_039,	NGVCK_040,	NGVCK_041,	NGVCK_042,
	NGVCK_043			

Reference:

- [1]. Raghunathan Srinivasan, "Protecting Anti-Virus software under Viral attack", August 2007, Arizona State University
- [2] Rune Schmidt Jensen, "Immune System for Virus detection and Elimination", 2002, Technical University of Denmark
- [3] Neil Daswani, Christoph Kern, and Anita Kesavan, " Foundations of Security: What Every Programmer Needs to Know ",2007 Apress.
- [4] Steven A. Hofmeyr, Stephanie Forrest and Anil Somayaji," Intrusion Detection using Sequences of System Calls", August 1998,University of New Mexico
- [5] Amanda Delamer,"Intrusion detection with data Mining", 2002, Donou University
- [6] Fausi Qattan & Fredrik Thernelius , "Deficiencies in Current Software Protection Mechanisms and Alternatives for Securing Computer Integrity", June 2004,Stockholm University
- [7] Wing Wong, "Analysis and Detection of Metamorphic computer viruses", May 2006, San Jose State University 2003
- [8] Mark Ludwig, "The Little black book of computer viruses",1996
- [9] Thomas M. Chen, "Trends in Viruses and Worms", 2004, Southern Methodist University
- [10] Microsoft Virtual PC: <http://www.microsoft.com>, January 2007.
- [11] VX Heavens: <http://vx.netlux.org>, April 2007
- [12] Moinuddin Mohammed,"Zeroing in on Metamorphic Computer Viruses",2003 University of Louisiana at Lafayette
- [13] Ankit Faida, "Ethical Hacking", 2006, Macmillan Indian ltd
- [14] Code Project : www.codeproject.com , 2003-2009
- [15] William Arnold & Gerald Tesaro," AUTOMATICALLY GENERATED WIN32 HEURISTIC VIRUS DETECTION", *VIRUS BULLETIN CONFERENCE* ,2000 Virus Bulletin Ltd,
- [16]. Mark E. Russinoich,David A Solomon, "Microsoft Windows Internals", 2005,Microsoft Press
- [17] B.Yegnanarayana, "Artificial Neural Networks", 2003, Pretence Hall of India

- [18] Jeff Duntemann, "Assembly Language Step By Step ", *Second Edition* ,2000 , Wiley Computer publishing
- [19] Richard A. Johnson, "Probability and Statistics For Engineers", Seventh Edition, 2006, Prentice Hall of India
- [20] Ankit Faida,"Email Hacking", First Edition,2006, Macmillian India Limited
- [21] Sergei Bezobrazov, Vladimir Golovko, "Neural Networks for Artificial Immune Systems: LVQ for Detectors Construction", 2007, IEEE
- [22] Robin Bloor, The Extraordinary Failure of Anti-Virus Technology, 2007 , Hurwitz & Associates
- [23] Inprise Corporation, Win32 Programmers Reference , 1998, Inprise Corporation
- [24] Wing Wong and Mark Stamp, Hunting for Metamorphic Engines, 2007, San Jos e State University
- [25] Tom Schen, Research in Computer Viruses and Worms, 2004 , SMU
- [26] Paul Oldfield, Computer viruses demystified, 2003, Sophos Plc
- [27] Jose Andre Morales, Peter J. Clarke, Yi Deng, Characterizing and Detecting Virus Replication, 2008, IEEE