# MALWARE ANALYSIS AND PREVENTION

**A Thesis**

***Submitted***

*in partial fulfillment of the Requirements for*

*the award of the degree of*

**Master of Science** (By Research)

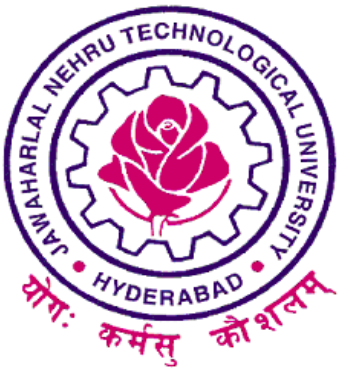
in

**Faculty of Computer Science and Engineering**

by

**Himanshu Pareek**

[Reg No. 0903MS0612]




**Directorate of Research and Development**

**Jawaharlal Nehru Technological University, Hyderabad**

**Kukatpally, Hyderabad - 500085**

**India**

**July 2015**

# DECLARATION

I hereby declare that the work described in this thesis entitled **MALWARE ANALYSIS AND PREVENTION** which is being submitted by me in partial fulfillment for the award of M.S. By Research in the Dept. of **Computer Science And Engineering** to the Jawaharlal Nehru Technological University, Hyderabad, Kukatpally, Hyderabad (Telangana) - 500085, is a result of investigations carried out by me under the guidance of Dr. N. Sarat Chandra Babu and Mrs. P. R. L. Eswari.

The work is original and has not been submitted for any degree or diploma in this or any other university.

Place: Hyderabad

Date: 31.07.2015

Signature:
Name of Candidate: Himanshu Pareek
Roll No: 0903MS0612

# CERTIFICATE

This is to certify that the thesis / dissertation entitled **MALWARE ANALYSIS AND PREVENTION** that is being submitted by **Himanshu Pareek** in partial fulfillment for the award of M.S. By Research in **Computer Science And Engineering** to the Jawaharlal Nehru Technological University, Hyderabad is a record of bonafide work carried out by him under our guidance and supervision.

The results embodied in this thesis have not been submitted to any other University or Institute for the award of any degree or diploma.

| (Signature of Co-Supervisor) | (Signature of Supervisor) |
|---|---|
| P R L Eswari<br>Joint Director,<br>C-DAC Hyderabad,<br>Hardware Park, Hyderabad | Dr. N Sarat Chandra Babu<br>Executive Director,<br>C-DAC Bengaluru,<br>Electronics City, Bengaluru |
| (Name and Designation) | (Name and Designation) |

# CERTIFICATE

This is to certify that the thesis entitled **MALWARE ANALYSIS AND PREVENTION** that is being submitted by Sri **Himanshu Pareek** in partial fulfillment for the award of M.S. By Research in **Computer Science And Engineering** to the Jawaharlal Nehru Technological University, Hyderabad is a record of bonafide work carried out by him at our organization.

(Signature of Director of Organization)

(Name and Designation)

E. Magesh
Director, C-DAC Hyderabad,
Hardware Park, Hyderabad.

# Acknowledgments

I would like to thank my supervisor Dr. N Sarat Chandra Babu for his support and advice. He has encouraged me to do research in computer security. He motivated me to start with something flexible rather than a rigid topic. It helped me to understand variety of colors and wings of computer security and helped me to innovate in my capability. He has been very helpful in my growth as a researcher and gave the right perspective to identify and view the problems.

I would like to thank my co-supervisor Mrs. P R L Eswari for all her support and motivation to carry out the research work. I shall always hold the lessons I learned from her about keeping continuity of research work. I shall always remain obliged and grateful to my supervisor and co-supervisor for all their kindness and guidance. I consider myself very privileged to be part of the C-DAC Hyderabad and MS by Research Program.

I would like to thank Mr. Mahesh U. Patil who has always helped me by discussing some interesting issues despite of his busy schedule. He guided me by reviewing research papers as a professional reviewer before sending them to review committees.

I would like to thank Mr. E. Magesh, the director of C-DAC Hyderabad for his continued support for pursuing higher studies. Next I would like to thank the former Directors of C-DAC Hyderabad, Mr. D. K Jain and Mr. Muralidharan for their support and encouragement. The resources and facilities in C-DAC Labs were always available for the employees doing research.

I also warm heartedly thank my colleagues and friends in the labs, Sandeep Romana and G. Jyostna, Anishka Singh, Rohit Arora for their unwavering support and help. And I thank all my colleagues at C-DAC

Hyderabad for making it happen.

I should thank to all malware sharing websites for generously hosting malicious binaries, PDFs and PCAP files for the research. I would like to thank everyone including researchers, software providers and latex community.

To my parents and sister

To my wife Bhakti and my little daughter

# Publications Based on this Thesis

1. Pareek Himanshu, P. Eswari, N. Sarat Chandra Babu. Entropy and n-gram Analysis of Malicious PDF Documents. In International Journal of Engineering Research and Technology, vol. 2, no. 2 (February-2013). ESRSA Publications, 2013.

2. Pareek Himanshu, P. R. L. Eswari, and N. Sarat Chandra Babu. APPBACS: AN APPLICATION BEHAVIOR ANALYSIS AND CLASSIFICATION SYSTEM. International Journal of Computer Science and Information Technology 5, no. 2 (2013). pp.53-61.

3. Pareek Himanshu, and N. Sarat Chandra Babu. Complementing static and dynamic analysis approaches for better network defense. In Dependable Systems and Networks (DSN), 2013 43rd Annual IEEE/IFIP International Conference on, pp. 1-2. IEEE, 2013.

4. Pareek Himanshu. Malicious PDF Document Detection Based On Feature Extraction And Entropy. International Journal of Security, Privacy and Trust Management ( IJSPTM) Vol 2, No 5, October 2013. pp:31-35

# Abstract

Malware is a continuously evolving problem for computer users and has shown rapid growth in last decade in terms of its complexity and scope of attacks. Though a standard antivirus and firewall combination provides fair malware prevention capabilities, research is required to mitigate the growing malware threat and improve current analysis and prevention mechanisms. Signature based techniques are most popular and it is difficult to imagine an antimalware solution without signatures. It slowly became a standard because of low false positives and less CPU time required. But these mechanisms often fall short to prevent malware which come in a new avatar every day. Moreover malware number is now so large that signature based scanning consumes too much disk space and takes longer time to scan against malware.

Static heuristic techniques suffer from both high false positive rate and false negative rate while dynamic heuristic techniques are a bit difficult to deploy at host level and gateway level. As part of this work, we explored various malware analysis techniques and prevention techniques. We explored to figure out "what will make a good approach for malware prevention in the network".

We realized that gateway level malware detection is not taking advantage of either static heuristics based detection and dynamic analysis based detection in an effective manner. We designed and

implemented a unique gateway level approach to prevent a malicious file reaching to host in the network. The approach complements static and dynamic analysis.

To support our approach we designed and implemented static analysis and dynamic analysis based malware detection approaches. We focus on Windows executable and malicious PDF documents as these are major attack vectors. Hence we worked for dynamic analysis and static analysis of these threats.

We propose Automated Dynamic Analysis and Malware Identification System which captures the behavior of a program by executing it and then analyze the traces and is capable of giving a verdict whether program of file under test is malicious. Every behavior captured is searched in behavior database and if found, a severity is assigned and finally a power distance is calculated. If power distance score is more than threshold level then program is termed as malicious. For dynamic analysis of PDF files, we used Adobe Reader.

We also design, implement and evaluate the static analysis based detection for PE (Portable Executable) files and PDF files. Both of these static analysis based detectors were designed on principle of extracting various features and then using classification algorithms which helps in classifying the file as benign or malicious. Feature set and definition will defer for two file types. For example, for PE file it is more of header flags, execution flags etc. and for PDF files it is like whether file contains obfuscated JavaScript, suspicious JavaScript, low entropy values etc.

The advantage of complementing approach is discussed and potential of such a solution in this direction along with conclusions is presented in the report.

# Contents

**Abstract** **ix**

**Notations and Abbreviations** **xvii**

**1 Introduction** **1**

1.1 Types of Malware . . . . . 3

1.2 Anatomy of modern day malware . . . . . 7

1.3 Malware Economy . . . . . 9

1.3.1 Spam Business Model . . . . . 9

1.3.2 Advertisement Business Model . . . . . 9

1.4 Obfuscated Malware . . . . . 10

1.5 Malware Analysis and Prevention . . . . . 11

1.6 Application Whitelisting . . . . . 13

1.7 Motivation and Problem Statement . . . . . 14

1.8 Organization of the thesis . . . . . 17

**2 Literature Survey** **18**

**3 Complementing Static And Dynamic Analysis for Network Defense** **22**

3.1 Introduction . . . . . 22

3.2 Approach . . . . . 25

3.3 Results . . . . . 28

**4 Automated Dynamic Analysis and Malware Identification 32**

4.1 Introduction . . . . . 32

4.2 Related Work . . . . . 33

4.3 Overview of ADAMIS . . . . . 34

4.4 Dataset . . . . . 35

4.5 Methodology . . . . . 36

4.6 Implementation . . . . . 42

4.7 Evaluation . . . . . 43

**5 Detecting the Malicious Files By Static Analysis 47**

5.1 Malicious PDF Detection using Static Analysis . . . . . 47

5.1.1 Portable Document Format . . . . . 48

5.1.2 Datasets . . . . . 50

5.1.3 Experiment: Entropy Analysis of PDF Documents . 50

5.1.4 Experiment: n-gram Analysis of PDF Documents . . 52

5.1.5 Malicious PDF Identification using Feature Extraction 54

5.2 Malicious Portable Executable (PE) Detection using Static Analysis . . . . . 59

**6 Conclusion and Future work 62**

**References 63**

**Appendix A Calculating Entropy 72**

**Appendix B Reconstructing PDF File 74**

**Appendix C Sample Analysis Report 76**

# List of Figures

1.1 Increase in malware count for last ten years . . . . . 2
1.2 Increase in new malware for last ten years . . . . . 2
1.3 A Botnet master building its network . . . . . 5
1.4 Anatomy of a Modern Malware . . . . . 7
1.5 Spam Business Model . . . . . 10
1.6 Advertiser Business Model . . . . . 10
1.7 PE32 Image file in packed form . . . . . 11
1.8 A possible improvement for deployment of Whitelisting . . 14
1.9 Trojan Attacks, Source: Microsoft Inc. Security Report 2010 14
1.10 User negligence more harmful, Source: Microsoft Inc. Security Report 2010 . . . . . 15

3.1 Timeline of remedies available for a threat . . . . . 23
3.2 Comparing Signature based scanning with heuristic techniques . . . . . 24
3.3 Complementing Static and Dynamic Analysis . . . . . 25
3.4 Drop Last Packet Approach . . . . . 26
3.5 System Design . . . . . 27
3.6 State Diagram for client . . . . . 28
3.7 Portion of Analysis report of mal.pdf . . . . . 29

4.1 Three Steps in ADAMIS . . . . . 34
4.2 ADAMIS Behavior Capture System . . . . . 43

4.3 Comparison of PD Score of malicious and benign applications . . . . . 44
4.4 Comparing results from euclidean and power distance . . . 45

5.1 Minimal PDF file to understand the format . . . . . 49
5.2 Entropy Analysis of PDF Documents . . . . . 52
5.3 Comparison of True Positive Rate with various classifiers . 57
5.4 Comparision of True Positive Rate with various classifiers . 58
5.5 Time taken to scan the PDF file . . . . . 58

C.1 Sample Report from ADAMIS, Win32.IRCBot.nw . . . . . 76

# List of Tables

3.1 Malicious traits detected in the mal.pdf . . . . . 30
3.2 Comparing Results of Static and Dynamic analysis . . . . . 31

4.1 Severity Number and Weights assigned to each behavior . 36
4.2 List of Behaviors . . . . . 39
4.3 Win32 APIs intercepted using Detour Library . . . . . 42
4.4 True Positive and False Positive Results . . . . . 44
4.5 Power Distance calculated on sample malware . . . . . 46

5.1 Results of Entropy Calculation . . . . . 51
5.2 Results with J48 Decision Tree . . . . . 54
5.3 Features Extracted from PDF File . . . . . 56
5.4 List of Characteristics extracted from PE file . . . . . 60
5.5 Results by static analysis of PE files . . . . . 61

# Keywords

Malware Analysis, Malware Detection and Prevention, Static Analysis, Dynamic Analysis, Gateway Level Malware Prevention, Malicious Windows Binary, Malicious PDF Document, Entropy, n–gram, Feature Selection, Classification, Similarity Scores.

# Notations and Abbreviations

**ADAMIS** Automated Dynamic Analysis and Malware Identification System. xvi, 16, 24, 33, 61

**DDI** Device Driver Interface. xvi, 42

**ED** Euclidean Distance. xvi, 38

**NSRL** National Software Reference Library. xvi, 13

**PD** Power Distance. xvi, 35

**PDF** Portable Document Format. xvi, 3

**PE** Portable Executable. xvi, 3

**PUP** Potentially Unwanted Program. xvi, 7, 15

**Spam** An unsolicited email. xvi

# Chapter 1

# Introduction

Malware[1] (**mal**icious soft**ware**) is defined as software designed to infiltrate a computer system without the owner's informed consent. However a Trojan program infiltrates the computer system with the owner's informed consent and it might perform some tasks which are without the owner's knowledge. Initially malware attacks were observed in late 80s. In initial days of malware, the primary goal of malware authors was to show supremacy in programming skills, or disrupt the other people's work and rate of growth of malware was considerably slow. However as of today number of new malware is growing at very high rate and existing tools are falling short of providing the security. Figure 1.1 shows the growth of malware over last ten years while figure 1.2 shows the growth of new malware for last ten years [1].

Malware attacks have become sophisticated and authors are now money oriented. They want to steal banking, and other personal data from the victim's computer. Moreover malware is also used by governments to spy and disrupt the other state's functioning.

[1]Webroot Computer Security Glossary

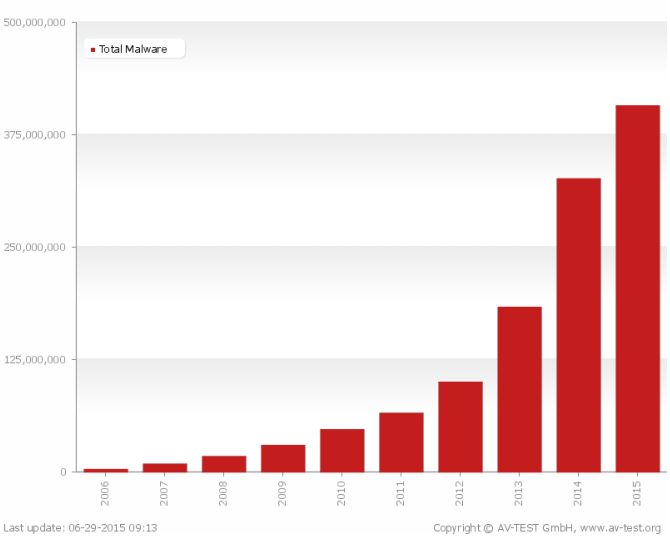



Figure 1.1: Increase in malware count for last ten years

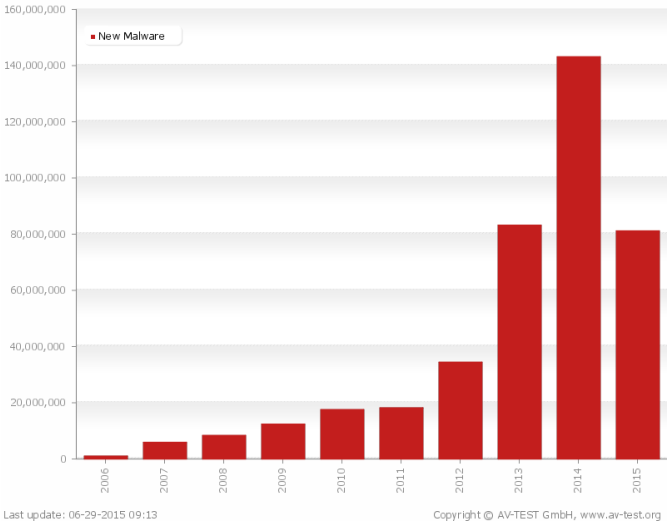



Figure 1.2: Increase in new malware for last ten years

## 1.1 Types of Malware

Malware infiltrates a computer using various attack vectors and executes code albeit large amount of code on the victim's computer. The attack vector may be complicated but still it can be categorized in two broad categories. First, victim downloads a program thinking that it is useful and it turns out to bad. Second, attacker or malware exploits some weakness in the system configuration or code and is able to inject malicious code into computer. The attackers use Windows Portable Executable (PE), Portable Document Format (PDF) and many other type of files to infect the potential victims. Malware is best categorized using their functionality and attack mechanism. Following is a list of these types.

**Virus** Virus is the most primitive form of malware. This is a malicious program which attaches itself to a file preferably binary. When infected binary file is executed, virus also executes. Virus has taken more complicated forms of boot sector viruses, multi-partite viruses and macro viruses. Most recent form of virus is EPO virus (Entry Point Obfuscation). Virus spreads by sharing infected files, sending infected files as attachments in e-mail.

**Worm** A worm is identical to virus but is more dangerous as it has got the capability of spreading at its own. Worm unlike virus, does not require the host file to propagate from one computer to another computer.

**Trojan** Of course, this type of malware gets its name from mythological Trojan horse and it does as much as trickery also. The Trojan horse malware will look as useful software and will try to persuade the computer user to install it. Once it is installed or

executed, it starts malicious activities. These types of malware are infamous for downloading more malicious software on the computer and acting as backdoor.

**Rootkit** A rootkit is a software system that consists of one or more programs designed to hide the fact that system is compromised. For example, one binary which carries motivational logic and one binary which itself is hidden and hides the first binary. Root kits are of different types based on at what layer the hiding is done. The various layers at which hiding is done are:

1. Hardware level
2. Hypervisor level
3. User mode and
4. Kernel mode

One of recent advanced form of root kit is the one which infects master boot record (MBR) of the computer.

**Adware** Adware is one of the most common forms of malware. It secretly embeds itself into the computer program (often browsers) and analyzes web browsing habits and then related pop-up windows and advertisements are shown. Adware often relies on web browser cookies for their working. A user can delete them by using web browser specific instructions. A more harmful and debatable version of cookies are Flash cookies. These cookies can not be deleted using web browsers. They can be deleted using web interface provided by flash player.

**keylogger** The keylogger refers to a program which monitors all the keystrokes by computer user, records them and sends them to another user. Keylogger software is classified as malware when

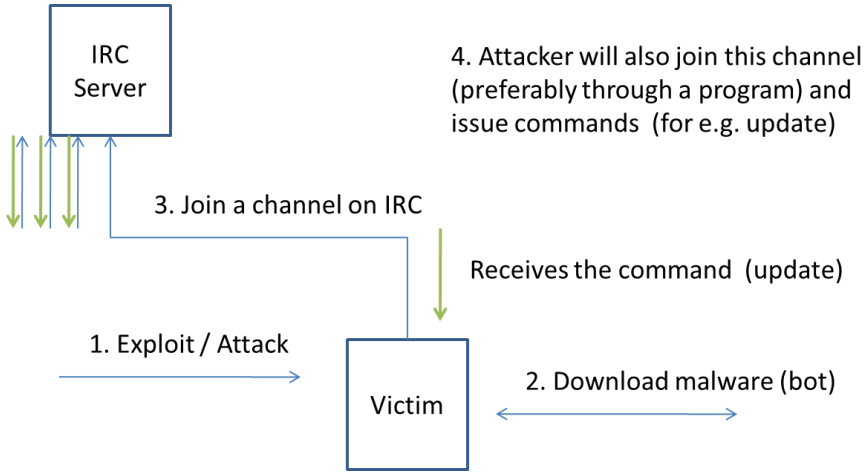



Figure 1.3: A Botnet master building its network

keystroke logging is done without the consent of user and primarily done for stealing the sensitive data such as credit card numbers, passwords etc.

**Browser hijackers** Browser hijackers will redirect to a different homepage. Browser hijackers which redirect to adult sites can leave some bookmarks, which have been known to cost people their jobs. Some hijackers redirect user to a page that has an advertisement telling him that his computer is infected and that he should purchase and run their virus checker which is actually a fake antivirus software. Other browser hijackers are written purely for increasing page views to a particular website in order to get more advertising revenue.

**Botnet** Botnets are another deadly form of malware which make a network at their own and coordinate to carry out the attacks. A malicious bot binary propagates by itself and when infects a PC turns that into a zombie. It takes command from its master and executes them on system and user has no idea about it. A simple scenario for understanding botnet is shown in Figure 1.3.

In the first step, victim system is exploited and malicious bot is

downloaded. Botnet master is already prepared with a automated IRC channel. Bot connects to this channel and receives the commands from its master or server. This server can be called Command and Control (C&C) Server. Attackers also use HTTP websites and twitter handles[2] to create channels. Another kind of botnet is Peer to peer (P2P) where bot binaries also act as C&C server for other binaries. In this case Bot master connects to only one infected system and then command propagates through bot binaries also. Botnets are used by hackers for various purposes viz. distributed denial of service attacks, spam and using victim for other sensitive attacks.

**Drive By Download** Though not a fundamental type of malware, this list will be incomplete with out mentioning drive by download. Web is the biggest source for infecting end systems and Driver-by-Download is the major attack vector used by attackers for the purpose. Attacker first gain access to malicious site by exploiting a vulnerability in the web application and post their malicious code there. When user visits this legitimate website, a malicious page is also loaded silently on user's computer (e.g. hidden iframe) and this malicious page is redirected to an exploit server. Once exploit is executed successfully malware is downloaded from the server to the victim's computer. These malware are often zero day to zero hour attacks and they change their signature before they make entry another time in the same network. There exists separate research work which tries to prevent drive-by-download attacks by analyzing web requests, but in our proposed approach even if drive by download attack is successful, incoming binary is analyzed to prevent the attack.

[2] www.twitter.com

## 1.2 Anatomy of modern day malware

Modern day malware is a complete software and hence most of the times it is difficult to classify it as one of the categories described above. Malware researchers added some more categories like Potentially Unwanted Program (PUP), exploit, malicious page etc. Nicolas Falliere explains the anotamy of malware by means of reverse engineering and its assembly code [2].

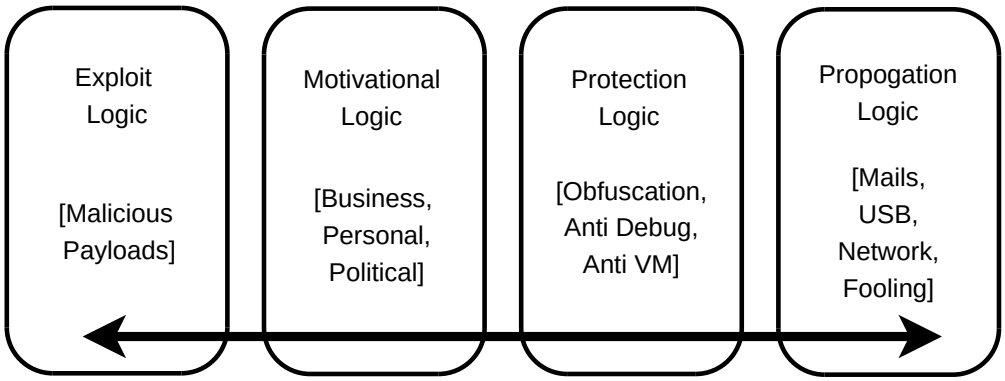



Figure 1.4: Anatomy of a Modern Malware

We explain four components of a modern day malware and suggest if this can be used as better nomenclature for malware, figure 1.4.

**Exploit Logic** This logic is required by malware to infiltrate in the system. This logic can be some code which exploits the vulnerability in an application or Operating system. This logic also refers to a method which exploits the vulnerability in security practices. For example, attacker can exploit computer user's lack of awareness to install malicious software on his system.

**Motivational Logic** This is the logic which is directly visible to a computer user. A malware might be written for destroying the data or stealing the data. This logic is basically motivation behind writing the malware. Ransom ware is primarily a malware named on basis of its motivational logic. A ransom-ware is locks (or

encrypts) victim's data and asking ransom for unlocking (or decrypting) it.

**Protection Logic** Malware also has protection logic to protect itself from being analyzed and removed. For example:

- Malware can carry a kernel mode component which will hide malware process.
- Malware binaries will be packed and code will be unpacked at the time of execution. This helps malware to bypass file system scanning in antivirus.
- Malware can carry logic to halt its execution if any debugger process is attached to it. It makes the job of analyzing the malware more difficult.
- Malware can also carry the logic to halt its execution if it detect that it is being run in virtualized environment or sandbox.
- Malware can kill processes of popular malware analysis software such as process monitor, process explorer etc. or will kill the antivirus process.

**Propagation Logic** Finally, malware would also like to infect as many computers as possible. For this malware uses the technique of infecting the portable storage media and devices, emails and infecting the websites. Targeted attacks may not carry propagation logic. Stuxnet presented another generation of targeted attacks where it infected a lot other machines as well to keep confusion about the attacker and its targeted victim.

## 1.3 Malware Economy

There exists many business models in underground malware economy. In a famous incident of hacking [3], Russian Business Network (an ISP in Russia) was behind the attack against Bank of India. Bank of India website was infected with Trojans to steal passwords and ISP was aiding hackers in building botnet.

Here we describe two in-practice business models.

### 1.3.1 Spam Business Model

A Phisher first develops the duplicate website of a bank to steal the customer password, credit card number and other details. Now he wants many people to come to this website so that he can cheat more people and make more money. For inviting people to his fraud website, the preferred method is to email them a phishing message. Now to send more phishing messages, he requires someone who can draft emails which can bypass spam scanners. Spammer for doing his task will require some mechanism to send millions of spam emails. A person who owns and controls a botnet can do this by directing zombies to send a given email. The botnet owner will require the services of a guy who can exploit the OS vulnerabilities, write a malware and pack it. Victims come to fraud website and all cyber criminals get their share. This is depicted in Figure 1.5.

### 1.3.2 Advertisement Business Model

A botnet owner need not always work on contract. He can have a legitimate web site. His website may have lot of advertisements and when any user clicks on the advertisements, he earns money. Now, utilizing services from malware authors and other specialists, he can

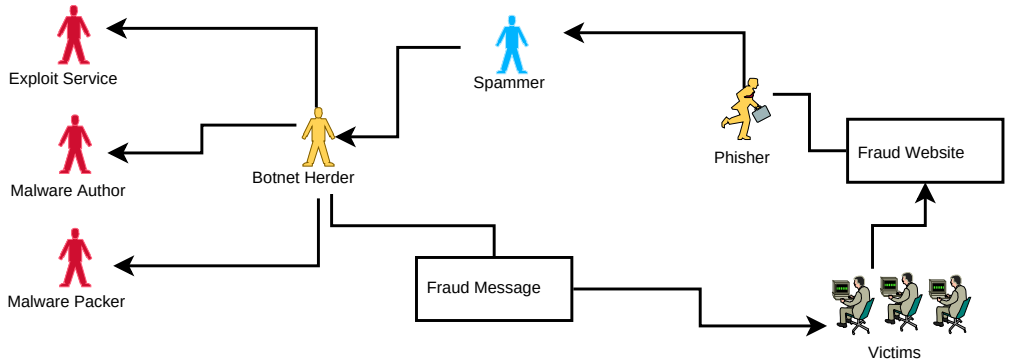



Figure 1.5: Spam Business Model

build and control a botnet which can do clicks on his website. By the result, he can earn a lot of money.

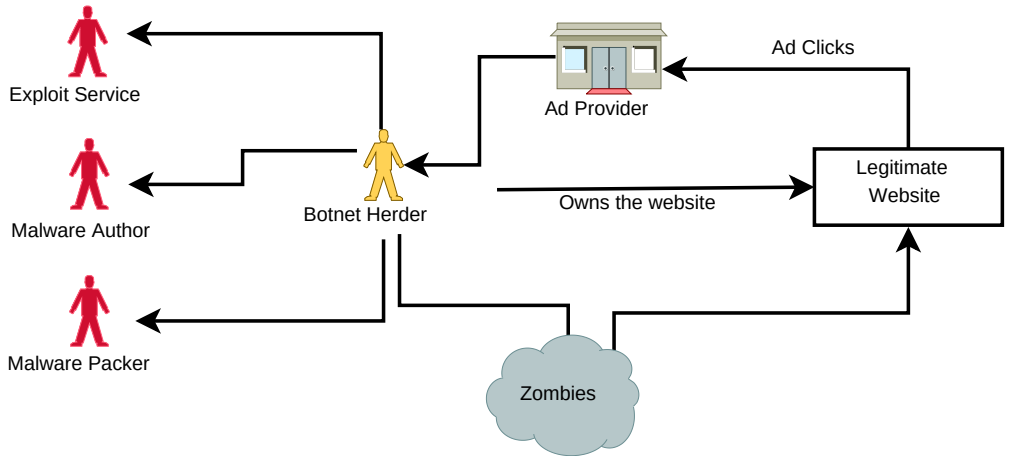



Figure 1.6: Advertiser Business Model

This possible business model is depicted in Figure 1.6.

## 1.4 Obfuscated Malware

In order to hinder analysis of the malware binary, attackers make use of obfuscation techniques. A typical obfuscated malware consists of decryptor code and original binary in encrypted form as data. When binary is loaded and executed, decryptor runs and decrypts the original binary and brings it in memory. This thwarts the signature matching when same original binary is encrypted with some other encryption. To make things further complicated polymorphic malware

keeps changing the decryptor code using techniques like dead code insertion, register reassignment, code transposition, subroutine reordering and instruction substitution [4].

Another form, metamorphic malware does not use encryption or packing techniques and keep changing semantics of its body whenever it propagates to another system.

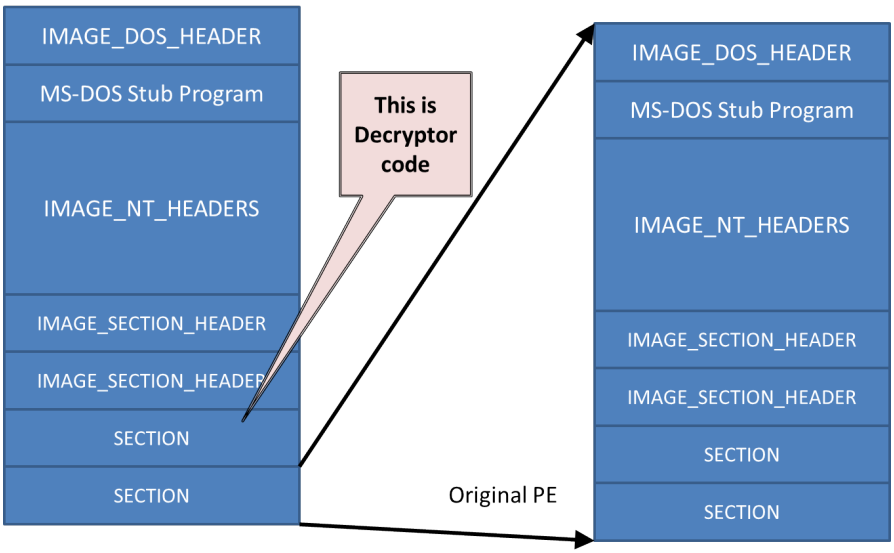



Figure 1.7: PE32 Image file in packed form

Figure 1.7 shows the concept of packed executable. Original malware is embedded into another PE built for the purpose. When packed PE runs, its decryptor code brings original PE into memory and executes it. There are thousands of packers and obfuscators available. Some examples are UPX, Themida, ASPack, VMProtect and Petite. Only UPX is a legitimate packer with native unpacker also available.

## 1.5 Malware Analysis and Prevention

Malware Analysis is a critical preliminary exercise for malware prevention. Malware analysis provides insight into malware and helps researchers to create malware prevention technologies. Malware analysis can be done in two ways.

**Static analysis** In this approach, suspicious file is analyzed without executing it. This includes header and structure analysis, disassembly and control flow graph construction, API analysis and entropy analysis. Static analysis becomes necessary when dynamic analysis does not give any further clues and analyst need its code and embedded data.

**Dynamic analysis** In this approach, suspicious file is analyzed by executing it and observing its actions.This includes executing the malware either on a real system, or virtual OS or emulator.

Static signature based scanning is a preferred approach to prevent malware as it prevents malware as soon as it enters into system. However to protect from zero day malware run time and emulation based detection is also proposed. They have the advantage of being more accurate but are burdened with disadvantage of giving a chance to malware to infect the system. Over the time static heuristic techniques to prevent malware has picked up the pace and much work has been done in this direction.

Signature based scanning is the preferred approach for malware prevention for computer systems but suffers from the fact that it is not efficient to detect new malware. Moreover static analysis is alone not sufficient to detect all kinds of malware [5].

To prevent zero day malware heuristic[3] techniques are preferred. Heuristic technique can be either static analysis based or dynamic analysis based. Heuristics refer to experience based problem solving. When we learn with a past experience and apply that learning to solve similar type of problem. For example, my three year old daughter has observed that mobile phone are hand held devices and regular need battery charging. Once she saw a camera for the first time it was not

[3] https://en.wikipedia.org/wiki/Heuristic

powering on, she was quick to conclude that it needs charging. This method of applying experience on new problems is referred as heuristics. We make use of both static and dynamic heuristic techniques. Heuristics is not as simple as signature based scanning or whitelisting and needs to be applied in new innovative ways to get better results [6].

## 1.6 Application Whitelisting

While antivirus software traditionally uses blacklisting based approach to detect malware, the approach of application whitelisting identifies known applications and does not allow any other applications. There are many solutions available to implement the application whitelisting in the network viz. Bit9 Parity, McAfee Application Control, Symantec Application Control, and Faronics Application Whitelisting.

Moreover, some organizations provide prebuilt application whitelists for use with application control clients. Following are the popular ones:

- Mandiant Corporation
- National Software Reference Library (NSRL)[4]

However, this approach suffers again from the fundamental problem that how an administrator or user will determine whether an application is good or bad if he has to select an application outside the pre-built whitelists. With our research we claim that this situation can also be improved.

An example network architecture with application whitelisting is shown in figure 1.8. The boxes in green indicate that static and dynamic analysis can be used in complementing manner with the Application whitelisting approach and thus can be more reliable.

[4] http://www.nsrl.nist.gov/

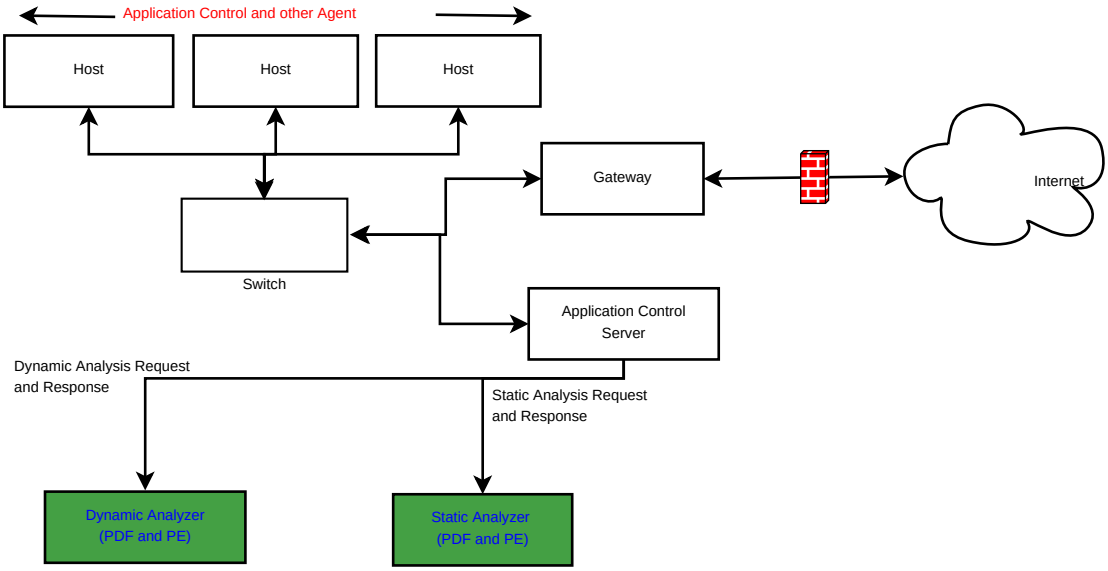



Figure 1.8: A possible improvement for deployment of Whitelisting

## 1.7 Motivation and Problem Statement

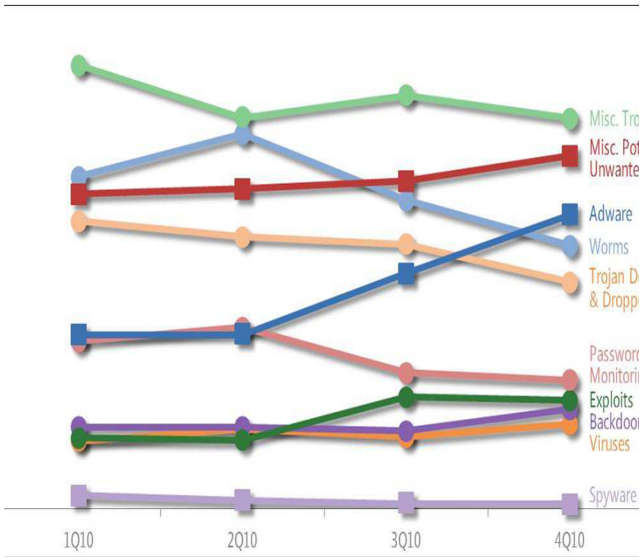



Figure 1.9: Trojan Attacks, Source: Microsoft Inc. Security Report 2010

We understand that a complete arsenal is required to secure a network from malware and not just one solution (or research or product) can do all. We choose our methodology as to first study malware attacks, carry out malware analysis and then explore if better strategy for preventing malware attacks in network could be designed

and implemented. We relied on current trends in malware attacks as outlined by vendors of anti malware software.

Security reports of Cisco, Microsoft and McAfee all claim that Trojan attacks are more as compared to other attacks.

Cisco Security Report (2008) states that "..of the malware distributed via the web, a large portion consisted of Trojans, designed to seem innocuous, invisible, or attractive before being installed" [7].

McAfee Threat Report 2010 [8] mentions that "email attachments have delivered malware for years, yet the increasing number of attacks targeted at corporations, journalists, and individual users often fool them into downloading Trojans and other malware."

Microsoft Security report 2011 [9] says that "Trojans account for around 20 to 25% of malware attacks." It is shown in figure 1.9. It also states that Trojan downloads are around 15% of total malware seen. It also states that Trojan attacks are due to user negligence. Moreover, PUP are on rise.

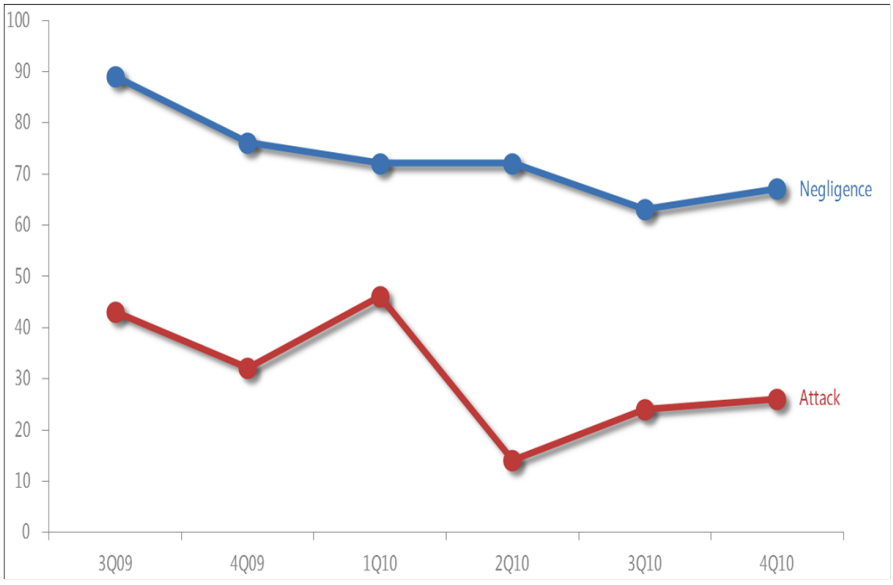



Figure 1.10: User negligence more harmful, Source: Microsoft Inc. Security Report 2010

We understand that behavior analyzers which can tell a user if a particular program is good or bad are unavailable which actually

inspires the user for negligence", figure 1.10.

Other important tap in preventing malware is gateway and study of popular Gateway Level Anti-malware solutions reveals that signature based scanning is prominent in these products as this is more efficient in terms of time consumption [10] [11] [12].

Hence we define our problem statement as the following two main points and hence set our objectives.

1. Limitation is unavailability of tools which can tell a user that this program is malicious.

2. Malware should be prevented at network gateway level but scanning should be effective and should take less time.

We strongly observe that to mitigate malware threat in the network it should be countered at both levels i.e. the end point system and gateway level. Host Systems should be protected from malicious binaries and documents. Biggest weak link is negligence of users. This can be solved by using application whitelisting approach with our proposed Automated Dynamic Analysis and Malware Identification System (ADAMIS) explained in chapter 4.

As much as possible, threats should be prevented at the network gateway level. We discuss problems with gateway level scanning. Our goal is to utilize the static and dynamic heuristic techniques for malware prevention at gateway level. We also explore static analysis for detecting malicious PE files and PDF files.

We define our goals as the following

1. Dynamic Behavioral Analysis To carry out the dynamic behavior analysis of malware. We carry out the analysis for Windows PE and PDF files. For dynamically analyzing the PDF Files, we analyze the adobe reader program.

2. Static Feature Analysis To carry out the static analysis of malware based on file features. Here also we decide to study and carry out the analysis on PE and PDF Files.

3. Network Level Malware Prevention Approach Our main objective is to improve the state of network level malware prevention approach by utilizing the static and dynamic heuristic techniques explored in previous objectives.

## 1.8 Organization of the thesis

The rest of the document is structured as following. Chapter 2 explains a brief about literature survey, related works in dynamic analysis and gateway level malware prevention. Chapter 3 explains our approach for gateway level prevention of malicious files which complements static and dynamic analysis approaches. This chapter utilizes material from reference [67] which is authored by Himanshu Pareek and Dr. N. Sarat Chandra Babu. Chapter 4 explains the developed dynamic analysis approach. This chapter utilizes material from reference [68] which is authored by Himanshu Pareek, Mrs. P R L Eswari and Dr. N. Sarat Chandra Babu. Chapter 5 is about static analysis based detection of malicious PE files and PDF documents. This chapter utilizes material from reference [69], reference [70], both authored by Himanshu Pareek, Mrs. P R L Eswari and Dr. N. Sarat Chandra Babu.

# Chapter 2

# Literature Survey

Nwokedi Idika, Aditya Kapoor [13] presented a state of art survey of malware detection techniques. Fundamentally malware detection or analysis can be done either by using static analysis or dynamic analysis. Authors use this fact and divide the malware detection techniques into three broad categories viz. anomaly based detection, specification based detection and signature based detection. Each category of detection is further divided into static, dynamic and hybrid.

Anomaly based detection techniques operate in two phases. First phase is referred as training phase or learning phase. In this phase detector makes the best attempt to learn the program's normal behavior. Second phase is called detection or enforcement phase. In dynamic anomaly based detection malware detection system learns the program behavior by executing the program in a sandbox and monitor the program behavior when it is actually running and if program deviates from earlier recorded normal behavior it flags the deviation as malware detection. In static analysis based anomaly detection, file structure characteristics are used to detect malware. Malware is not executed either at the time of learning or time of detection. Many hybrid approach can also exist. For example, program behavior can be

learned by static analysis and constructing call flow graphs but enforcement can be done at the time of program execution.

Specification based detection is a variant (or can be called sub–category) of anomaly based detectors. This type of detectors deliver or construct behavior rule set as the output of training phase. This type of detection can again be classified into dynamic, static and hybrid. Similarly Signature based detection schemes construct a model viz. regular expression, byte pattern etc. to represent the unique characteristics of malware.

Martin Stecher [12] discusses about the challenges when it comes to detect malware at the gateway level. This paper encouraged us to think of an innovative approach at the network level to detect malware.

John V. Harrison [14] explains that a threat to network security is user initiated malware execution. This paper states that users assists the hackers by downloading the malware from websites.

T. Holz et.al [15], presents CWSandbox where they hook as many system calls as possible on Windows OS. U. Bayer et al. [16] presents emulation based approach for dynamic analysis of malicious code.

Manuel Egele et al. [17] presents a survey on dynamic analysis techniques of malicious code. Bradley J. N abholz [18] presents the design of an automated malware analysis system. Yongtao Hu et.al [19] present method to detect unknown malicious executable based on run-time Behavior. It dynamically analyzes an executable on an assigned system and is able to give verdict that whether executable is malicious or benign. Our approach presented in the thesis combines malware analysis approach with run time malware detection approach.

Robert Lyda et.al [20] uses entropy analysis to find Encrypted and Packed Malware. Yang-seo Choi et.al. [21] presents an approach for detecting encoded executable files via executable file header analysis.

Asaf Shabtai et.al proposes to classify the malware using opcode n–grams [22]. It was difficult to detect obfuscated malware using this approach and takes too much time to be deployed at gateway level. But we were inspired by this and used similar approach for detecting malicious PDF documents. Igor Santos et.al. [23] presents "Idea" which is used for Opcode sequence based Malware Detection. Scott Treadwell and Mian Zhou [24] proposes a heuristic approach for detection of obfuscated malware. These papers present various approaches to detect packed PE malware based on header analysis. We also implemented a similar approach to carry out the static analysis of executable to detect whether it is benign or malicious.

Charles Smutz and Angelos Stavrou [25] presents Ruminate: A Scalable Architecture for Deep Network Analysis. Work done by Charles Smutz and Angelos Stavrou is remarkable in the direction of deep network analysis to detect malware but is short of steps to prevent malware files from entering in the network. Phani Vadrevu et.al presents AMICO which differentiates between malware and benign file downloads based on learning of download behavior of network users [26].

Laskov Pavel and Nedim Šrndić [27] presents an approach to detect malicious PDF documents via static analysis. First, presence of JavaScript in the document is found and then analysis on these scripts is performed. Li Wei-Jen et.al in their paper analyzes structure of malcode bearing documents and uses entropy and n-grams to identify the malicious documents [60]. Blonce Alexandre explains PDF document structure, analyzes malicious PDF documents and explains how PDF features are used to construct malicious documents [59]. Tzermias Zacharias et.al combines static and dynamic analysis techniques to detect a malicious PDF documents [61]. Using static

analysis, javascript is extracted and is analyzed using dynamic analysis.

We designed and implemented a approach inspired from these works but also utilized other parameters to detect malicious PDF documents.

# Chapter 3

# Complementing Static And Dynamic Analysis for Network Defense

## 3.1 Introduction

The prevention of malicious code at gateway level remains a challenge [6], [12]. Available malware prevention at gateway level scans for malware signatures at per packet level and provides only detection at file level which requires re-assembly of packet stream [10], [11]. Emmanuel Hooper draws the attention towards large number of false positive alerts generated by Intrusion detection systems [29]. The paper proposes statistical analysis for verification of connections' identity and reduce the false positives. Te–Shun Chou proposes a ensemble architecture for more effective threat detection at gateway level [30].

This chapter utilizes material from reference which is authored by Himanshu Pareek and Dr. N. Sarat Chandra Babu [67].

Signature based approaches are excellent in detecting previously known threats (we refer malicious files as threat onwards) but cannot detect new threats. When it comes to heuristics, static analysis based approaches are preferred to be deployed at network level rather than dynamic approaches. Even though dynamic approach is better in detecting threat, its deployment either at host level or network level will be awful for a network administrator. Static approaches based on heuristics suffer from false positive rate and sometime threat goes undetected due to use of sophisticated obfuscation [5].

First, we assume that signature based solutions are quick to formulate once a threat has been recognized. With some time taken dynamic analysis based approaches with low false negative rates are available and static analysis approaches with low false negative rates are little late to conceive.

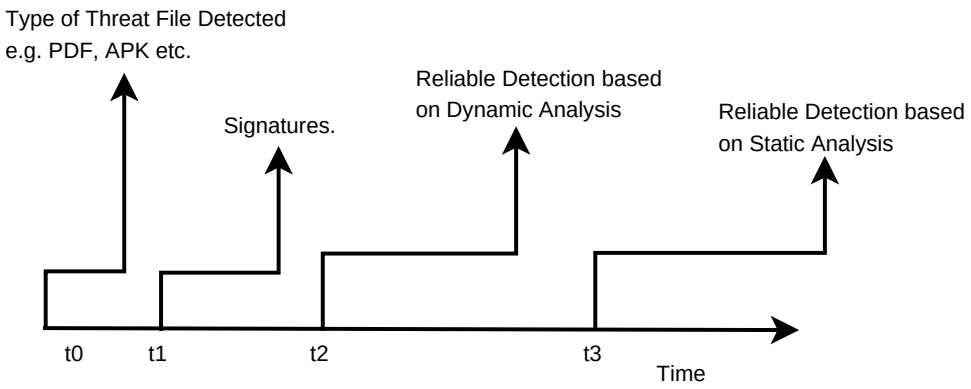



Figure 3.1: Timeline of remedies available for a threat

This is shown in figure 3.1. Signature based detection approaches are good for malware prevention at gateway level as scanner program don't even need the complete file. They depict extremely low false negative rate. Dynamic and static heuristics require a file to be constructed at gateway level, their comparison is shown in figure 3.2. Both of them can predict malware with low false negative rates but it takes time to mature these technologies. We also observe and assume

that dynamic analysis heuristics will be easier to achieve low False negative rate in a shorter time span than static analysis techniques.

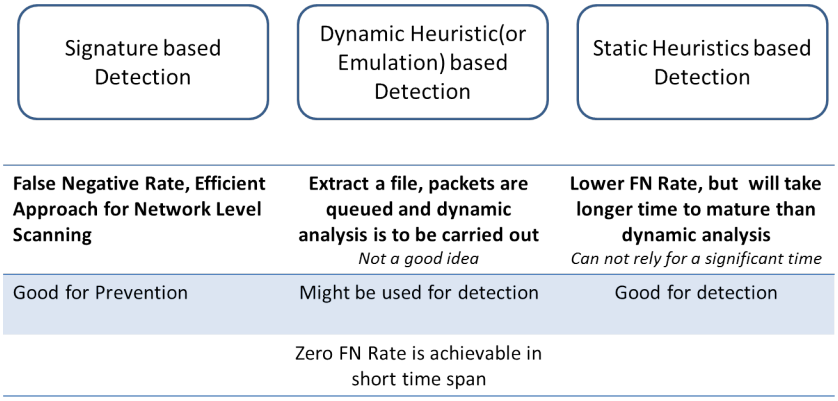



Figure 3.2: Comparing Signature based scanning with heuristic techniques

We observe that static heuristic approach essentially classify the files into three categories: benign, malicious and suspicious. Suspicious category states that file may be malicious but not compulsorily.

With our work, we intend to achieve a system which can effectively take a decision on suspicious files during time duration (t3-t2) without deploying dynamic analysis based detection software at host level or submission control with the user.

We focus on Windows Portable Executable (PE) and PDF files. For static analysis of PE and PDF files we used approaches inspired from Scott Treadwell [24] and Pavel Laskov [27] respectively. However, the approach explained by Pavel Laskov takes 20ms to scan a file and required some modifications to decrease time taken per file. For dynamic analysis of the suspicious PE files, we have designed and implemented ADAMIS (explained in chapter 4) which monitors the program behavior, collects traces and performs analysis to give a verdict. Cuckoo sandbox can also be used [28] for collecting behavior traces.

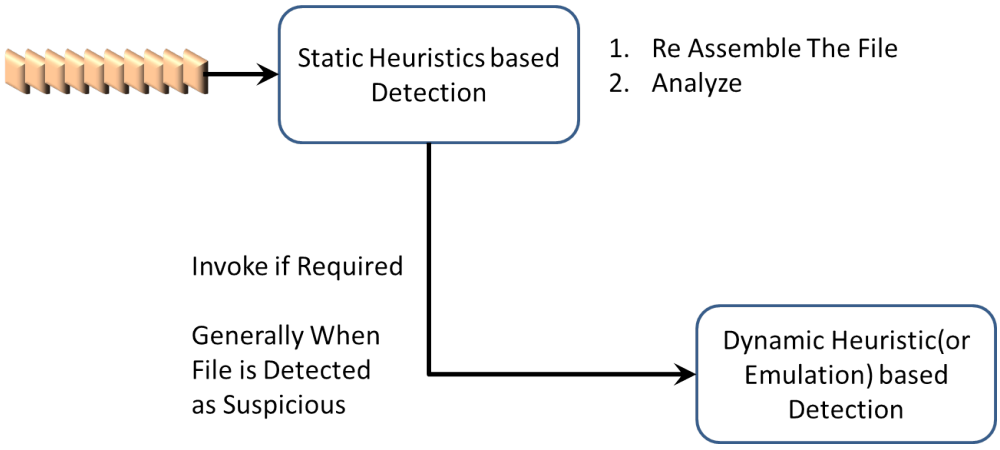



Figure 3.3: Complementing Static and Dynamic Analysis

## 3.2 Approach

System has four main components which are defined below.

- Host Agent: We install an Agent program at host which monitors file requests and file responses (file arriving from internet location) and is responsible for holding the file in quarantine state and taking appropriate action when final decision comes in.
- Gateway Controller: A controller program at gateway level coordinates the actions of detecting file type, sending file to static and dynamic analyzers.
- Static Analyzer: Component which does the heuristic based static analysis.
- Dynamic Analyzer: Component which does the dynamic analysis and gives a verdict whether the file is benign.

Figure 3.3 shows the approach which makes use of both static analysis and dynamic analysis. An incoming file is reassembled and statically analyzed. If the analysis is not able to give the verdict or file's complicity is out of scope of static analysis it can be given to dynamic analysis based detector.

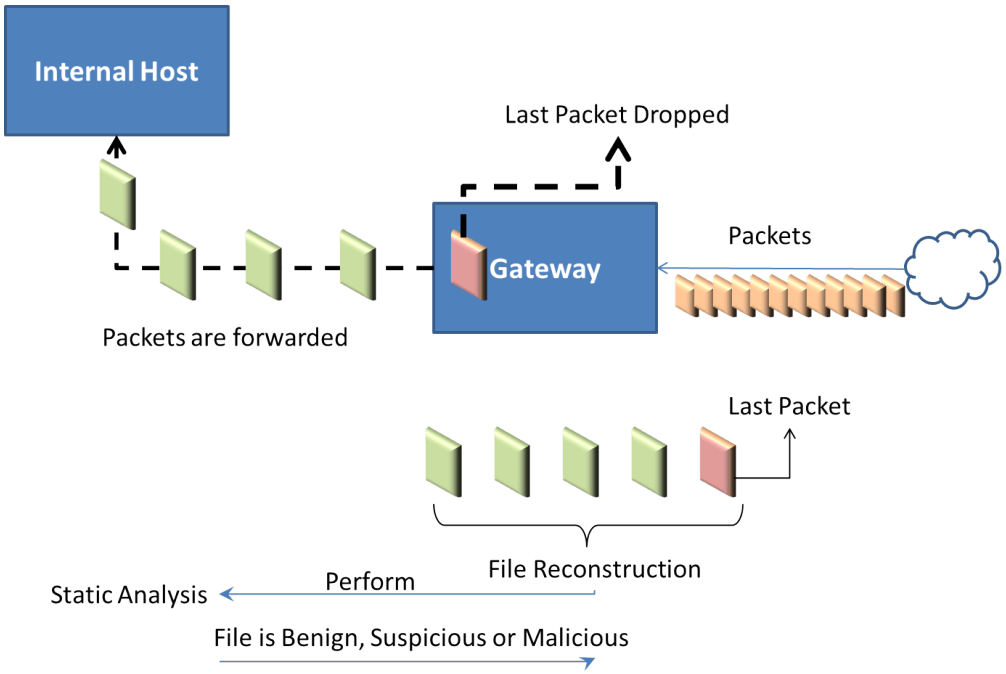



Figure 3.4: Drop Last Packet Approach

It is must to have complete file before static heuristics or dynamic heuristics can be invoked. Figure 3.4 shows the approach used for reassembling the file. It is easy to reconstruct the file while in detection mode. But challenge arises when malicious file has to be prevented from reaching the host. We assume that file contents if changed by even a few bytes will leave file useless. While file is being reconstructed we allow the packets to go from gateway to the host and at the same time a copy of packet is retained at the gateway controller to construct the file. When the last packet arrives, it is not sent to the host and retained at controller only to reconstruct the file. If file is found to be malicious in later step of analysis, last packet is dropped and never sent to the host.

With the involved techniques and concepts explained, we now can explain the overall approach followed with the help of Figure 3.5. When ever a file request is sent, host agent logs this file request. Later when gateway receives the file, controller program sends the file for static analysis. For this mechanism, we continue to forward the packets to

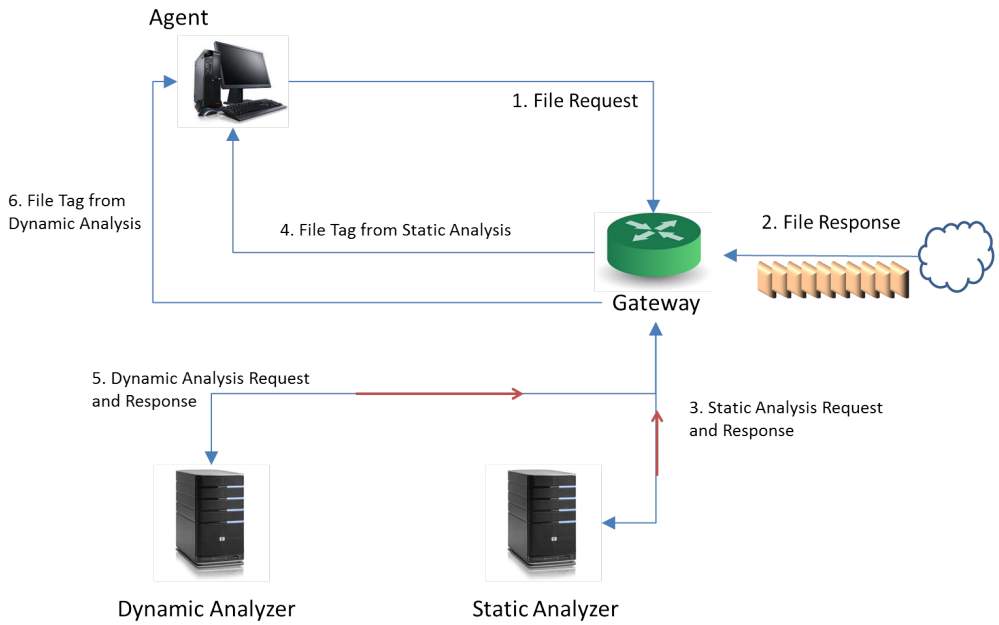



Figure 3.5: System Design

host and keep on rebuilding the file. With the help of the host agent this file is kept in quarantined state. Once the last packet is received, static analysis can be performed. Code fragment for reconstructing a PDF file from network stream is given in Appendix B. For static analysis we achieve an implementation which takes 5̃ms per file for analysis. If static analyzer tags the file as malicious then final packet is dropped. If file is called as benign or suspicious then final packet is allowed. With the decision taken on final packet, controller program sends a message to host agent telling the tag of the file. If tag is malicious, file would not have arrived at host completely as last packet was dropped and host agent will delete the file (if a part present on system) and terminate the requester program. If tag is benign, nothing is to be done other than removing the file from quarantine list. If tag is suspicious, then controller program will send it to the dynamic analyzer and host agent will keep the file in quarantined state. Controller program will now get the response from dynamic analyzer as either benign or malicious. This new tag will be sent to host agent which on receiving the tag will take

appropriate action. This is shown in Figure 3.6. State captions are shown in red color while actions received are shown in green color.

We have implemented our own simple protocol to transfer these messages but to have a better implementation OPES callout protocol (OCP) [31] can be used. We have designed and implemented system with focus on PE and PDF files on Windows OS. Proposed approach and implementation for PDF files can be applied to other OS but this has been out of scope of this work. The host agent is implemented on Windows OS. In host agent implementation, NDIS (Network Device Interface Specification) intermediate filter driver is used for logging file requests and mini filter driver is used to block access to suspicious files.

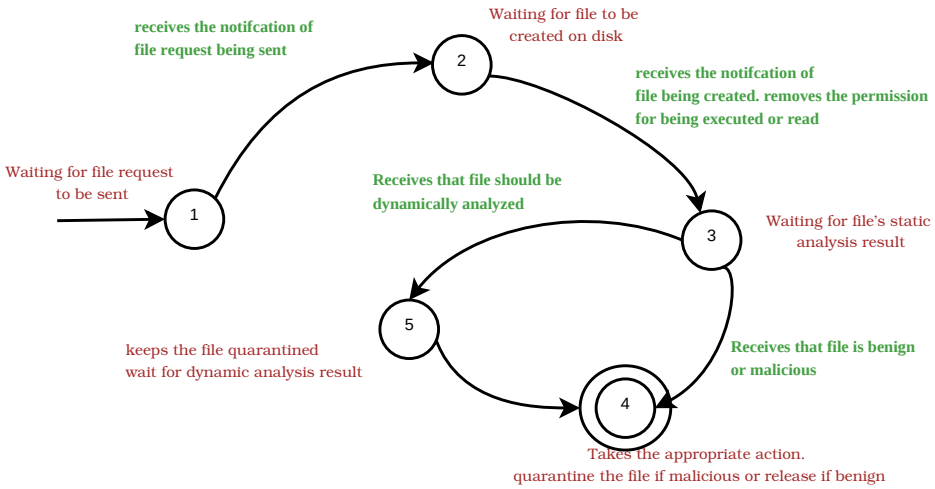



Figure 3.6: State Diagram for client

## 3.3 Results

A summary of results is presented in this section. We could identify some malware which were undetected by one approach but detected by other technique.

We take the case of malicious PDF file[1] mal.pdf. On date of writing,

[1] 34d8b44ccdb7c1a6885c7d11c1d87be3, Exploit:Win32/Pdfjsc.AD

```
 Analysis for C:\bad\mal.pdf
( 34d8b44ccdb7c1a6885c7d11c1d87be3 )
new files created
...\Administrator\Local Settings\Temp\AcrAD04.tmp
..\Application Data\Adobe\Acrobat\8.0\JavaScripts\glob.js
...\Application Data\Adobe\Acrobat\8.0\JavaScripts\glob.settings.js
writing files
..\Administrator\Local Settings\Temp\AcrAD04.tmp
Automated Analysis
Adding new mount points.
Changes the internet settings
creating and writing another pdf file
changes the time stamps of too many files.
the final score = 7.2
```

Figure 3.7: Portion of Analysis report of mal.pdf

this file was not identified malicious by 23 anti virus software out of 56 anti virus software listed on virustotal.com[2].

However Gallus[3] was not able to scan the document and crashed. We tried multiple times and the result was same. With features included in our dynamic analyzer, PDF file was detected. The file when opened with Adobe Reader 8.0 changed internet settings using registry which is quite unacceptable when a pdf file is opened. The more interesting fact with this PDF file was that it created another PDF file which is also indication towards the fact that file was malicious. The portion of dynamic analysis report for this file is shown in figure 3.7. Generated dummy PDF file is a blank page and is actually shown to user.

Static analysis of this file (table 3.1, approach explained in chapter 5) found 5 javascript in the file and none of them was obfuscated. It found entropy well above 2 and hence within limits. It also found '/OpenAction' Tag but none was clubbed with javascript. The file was not identified as malicious.

[2]virustotal.com is a popular multiple AV scan service

[3]https://gallus.honeynet.org.my/

Table 3.1: Malicious traits detected in the mal.pdf

| Feature or Tag | Frequency of appearance |
|---|---|
| /JS | 2 |
| /Javascript | 3 |
| /OpenAction | 1 |
| /Launch | 0 |
| Javascript after deflating | 0 |
| Obfuscated Javascript | 0 |
| Stream Entropy | Greater than 2 |
| AsciiHexDecode | 1 |
| chars after Last EOF | No |

Around 200 out of 2600 PE malware were not detected by static analysis component but detected by dynamic analysis component. We also took 33 latest samples and observed that 3 samples were not detected by static analysis but were detected by dynamic analysis. It is shown in table 3.2.

Static analysis based detection of PDF file produced better results as 695 out of 700 malicious PDF files were detected. Dynamic analysis based detection could detect 650 out of 700 files. False positive rate in case of dynamic analysis based detection was close to nil whereas false positive rate achieved with static analysis was 4%.

Table 3.2: Comparing Results of Static and Dynamic analysis

| File Hash | Detection using Static Analysis | Detection using Dynamic Analysis |
|---|---|---|
| 0e9eb771552fd27f0c6d6abccd0ff22e | yes | yes |
| 8be6d6663eebd2f9e034fdff4c7653f5 | yes | yes |
| d6a4efd06e58d2929713ce462868634c | yes | yes |
| 0f05fbf00965494bf0ddcda40f79d496 | yes | yes |
| e17855233209d664d7b55581c55c1c7f | yes | yes |
| 594122aac5891d9e330d5cc1034c248f | yes | yes |
| 00632896197e64afab73efd4c020df89 | yes | yes |
| c66828fab9495c5e7b3ea98d05dd1d95 | yes | yes |
| 40865ec020c33d0a9e0598b3fe179dbe | yes | yes |
| e6984a03519f4c303401096503b2d9ad | yes | yes |
| 3b9caa512a9154698e80aa4789c7befe | yes | yes |
| 0aad632e77ffef2e174b193987ad110d | yes | yes |
| 08d16815b94a11b0941a99dafdfb3060 | yes | yes |
| b78477bd8e0250365f1e369e17de3677 | yes | yes |
| fb2fc8178fbee78260f6427a527aca77 | No | yes |
| 62d05683e4939fe4ec14728c3258a5db | No | yes |
| 4fe6627e9ecb72caf3f64fc46cbaac0d | yes | No |
| 0d94e9ad068be0ed1e3c0ab80d4aa677 | yes | yes |
| 06c3828a7f16b8faee70fb1ac08eb209 | yes | yes |
| 06558a4d1877265a24fb4a36314c0c3c | yes | yes |
| 0d38fddd0cdae09097dfc96982966aca | yes | yes |
| 098dbfe7107cefedef119b7d278df3cb | yes | yes |
| 028f575446cff5859525898ed13d9077 | yes | yes |
| 0670ad6291a36d8d508d6678be5ac0f7 | yes | yes |
| 0790f9a7cab21919639e051edf179f46 | yes | yes |
| 0ad084ef420e65249548c5f03f605f5b | yes | yes |
| 00eca922b49a7584bff9ea922411c8e6 | yes | yes |
| 13ea0bd03e835d931c33f0dba72ce23a | yes | yes |
| 058e92a7ecd59700e1518418e4ec72aa | yes | yes |
| 105cafb7857c92f2df643834656e6152 | yes | yes |
| 00e0a7548f4f09f68a5d819b9c12f7cc | yes | yes |
| 0ac4d333d55ae17bfc38fabb6d359641 | No | yes |
| 08265a6d47522f43d8d2e69ff9d015cd | yes | yes |

# Chapter 4

# Automated Dynamic Analysis and Malware Identification

## 4.1 Introduction

Malware authors use many attack vectors like infecting legitimate websites, sending spam and maligning search engine results to tempt users to download malicious applications. Quick heal Technologies mentions that malware is the largest threat to PCs and Trojan is most occurring of them [32]. Though users install security software like firewall and antivirus and take other security measures like continuous updates and browser security etc. to secure them, it may fall short against increasing sophistication and use of social engineering traps in malware attacks. Users fall into trap and install the software which was actually a Trojan and puts the computer in compromised state. These kind of attacks are generally zero day malware and a behavior analysis can be very useful to have an indication about software's capabilities [33].

Malware analysts make use of manual and automated analysis tools to determine how malware performs its activities. Zerowine,

cuckoo, Anubis and Cwsandbox automated malware analysis tools [34], [35] [36] [15]. We design and implement a system to analyze the behavior of an application and identify it as either benign or malicious. We call the system ADAMIS. ADAMIS performs the analysis using kernel level hooking in Windows OS and automates using VIX API of VMware. A set of predefined rules of malicious behavior patterns is matched against the application behavior trace to find out if analyzed application is malicious.

This chapter utilizes material from reference [68] which is authored by Himanshu Pareek, Mrs. P R L Eswari and Dr. N. Sarat Chandra Babu.

## 4.2 Related Work

CWsanbox proposes automated dynamic malware analysis using API Hooking, DLL Injection etc [15]. U. Bayer proposes TTAnalyze which pitches for the use of emulator for dynamic analysis purposes [16]. These tools gives a malware behavior analysis report but do not present any score to help a nontechnical user to understand the analysis results. Arne Vidstrom at ntsecurity.nu demonstrates that malware can bypass emulator based analysis [37].

Mandiant Red Curtain uses static parameters of an executable like presence of packer signatures, entropy and PE anomalies and presents a score on given binary to indicate whether it is malicious [38]. Scott Treadwell demonstrates the use of weighted features to detect packed executable [24]. We use the similar approach to assign severity levels to malicious behaviors.

Konrad Rieck proposes the machine learning based approach for clustering the malware binaries as per their families [39]. Christian

Seifert, et al. presents Capture-BAT which analyzes malware and advocates the use of kernel level filtering, file system filter driver and registry callbacks for analysis purposes [40]. Wu Naiqi et.al proposes a novel approach to detect Trojan horses using program tracing [41].

## 4.3 Overview of ADAMIS

ADAMIS is designed, implemented and evaluated on Windows XP SP3 and onwards. ADAMIS also provides the feature of reverting back the changes on system which allows even a professional analyst to quickly analyze suspicious programs.

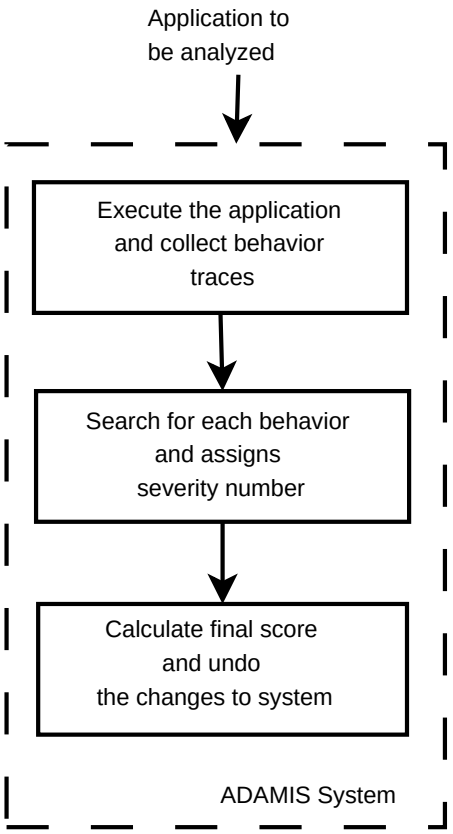



Figure 4.1: Three Steps in ADAMIS

The tool works in three phases as shown in figure 4.1.

**Step 1** User first runs the ADAMIS tool with path of untrusted application as an input. This automatically starts virtual OS,

copies the the application inside and ADAMIS executes the application and captures the behavior of application. It then automatically reverts back to clean state and copies the report back to host OS. Behavior traces can be collected from other tools like zerowine and cuckoo and ADAMIS analysis module can be used. Advantage with ADAMIS is that if executed on host OS directly, all changes are wiped off after analysis is over.

**Step 2** ADAMIS scans these behaviors traces against predefined malicious behavior patterns. Every detected malicious pattern is assigned a severity number. Assignment of severity and predefined behaviors are explained in next section.

**Step 3** ADAMIS calculates the score based on Power Distance (PD) using the severity numbers and identifies the application under three categories: Benign, Potentially unsafe, and Malicious [42]. User should not run a malicious application at all and potentially unsafe application with administrator privileges.

## 4.4 Dataset

We did a honeypot setup using Dionaea tool [43] at BSNL broadband in our lab. Dionaea tool lures attacker by posing a vulnerable system and then collects the malware. We collected malware from malware sharing websites as well. We used these malware for learning and framing malicious behavior patterns and collected malware from www.virussign.com for testing. Testing was performed on a set of 2500 malware.

## 4.5 Methodology

We first implemented our own dynamic analysis tool to collect malware behavior trace and then automated using VMware VIX API [44]. This helped us in analyzing a large number of malware and also making the tool more usable. With these reports we framed a list of malicious behavior patterns which can be matched against.

Table 4.1: Severity Number and Weights assigned to each behavior

| Behavior (or Category) | Severity | Weight |
|---|---|---|
| NonAcceptableFileSystemActivity | 7 | count |
| NonAcceptableRegistryActivity | 7 | count |
| DeleteSelf | 7 | 2 |
| CopySelf | 7 | 2 |
| WriteProcessMemory | 5 | 2 |
| PackedExecutable | 5 | 2 |
| AbnormalURL | 5 | 2 |
| ListenActivity | 5 | 2 |
| EnumerateProcesses | 3 | 2 |
| ReadProcessMemory | 3 | 2 |
| AvoidableRegistryActivity | 2 | count |
| AvoidableFileSystemActivity | 2 | count |
| PackagedInstaller | 0.3 | 2 |

**How We Assign these severity and weight numbers** Table 4.1 lists the behaviors monitored along with severity numbers and weights assigned to them. First four are categories and rest are plain behavior monitored. For example, category AvoidableFileSystemActivity contains behavior like creating an executable but creating a copy of itself is kept in other category. We assign a severity to each malicious characteristic indicating how risky is this behavior. This is based on our learning and experience with analyzing malware and are kept relative. For example, in our behavior rule set we observed that a file if copies itself to some other location or deletes itself after having copied

new files: this is the easy indicator that this application is malicious. Similarly, some unacceptable file system and registry activity are also equally severe. To these most severe activities we assign a severity of 7. Then to relatively less powerful indicators like packed executable, writing other process memory and listen activity are assigned a severity of 5.

Initially in this list, the weight didn't existed and we calculated final score with the help of Euclidean distance (explained later in this section) which did not give any observable result 4.4. Then we observed that frequency of avoidable and non-avoidable file system and registry activity can be used. So power of this behaviors was changed to 'count' and others were kept same at 2.

Following notations are used onwards to explain the analysis process.

B is the set of behaviors depicted by a process under consideration.

$$b = \{b_1, b_2, b_3, ....., b_i\},$$

Behaviors are grouped under various categories.

$$C = \{C_1, C_2, ....., C_N\}$$

where $N = 14$ and

$$B_i \in C_k$$

Vector V represents the behaviors along with their category.

$$V = \{v_1, v_2, v_3, .....v_n\}$$

where

$$v_i = \{B_i, C_i\}$$

Z is a vector of categories along with their severity number and Weight.

$$Z = \{z_i, z_2, z_3, ....., z_N\}$$

where

$$z_i = \{C_i, S, W\}$$

If behavior exhibited is $B_i$ then it is searched in V to find which category $C_i$ it belongs. Then $C_i$ is searched in Z to find the severity of the exhibited behavior. All the processes created by original program are analyzed in similar manner recursively.

We first executed our experiments without any weight and calculated Euclidean Distance (ED). Euclidean distance is used for measuring distance between two points in a plane. If coordinates of $P_1$ are $x_1$, $y_1$ and of $P_2$ are $x_2$, $y_2$, the euclidean distance between $P_2$ and $P_1$ will be

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

With this distance measure we estimate closeness or similarity between two positions. Here $x_1$, $y_1$ etc. are our points for benign points and assumed to be zero. Hence, a higher number shows higher distance from benign and indicates that this is more malicious. The severity numbers given in table 4.1 the positions ($y_n$) of malware applications.

A incomplete list of behavior rules $B_i$ of suspicious file system activity and registry activity is shown in 4.2. There are 12 file system activities and 32 registry activities divided into avoidable and non-acceptable.

As mentioned in the table 4.2, our dynamic analysis system considers that whether files created are executable or non-executable.

Table 4.2: List of Behaviors

| Category: AvoidableFileSystemActivity |
|---|
| Write to Windows Directory |
| Create a new PE32 file or script |
| Writing to Event Logs |
| Edit services file and add services (port number) to TCPIP Services |
| **Category: NonAcceptableFileSystemActivity** |
| Writing to Hosts file |
| Writing to lhosts.sam file |
| Changing Active Directory Settings using dssec.dat |
| **Category: AvoidableRegistryActivity** |
| Adds a program which will run automatically on next login |
| Enumerating Registry Keys |
| Installing Services and Drivers |
| **Category: NonAcceptableRegistryActivity** |
| Adds a program to safeboot |
| Install hidden services in svchost.exe |
| Disables registry editing tool |
| Disables Task Manager |
| Adding URL Search hooks in IE |
| Antivirus notification settings are changed |

Non-executable files like *.txt are not given any severity. It compares the hash of the original executable against created executable files and deleted files. If they are same, proposed system is able to report whether application is creating or deleting itself.

We use fakeDNS [45] which sets the DNS to a host configured by us and INETSIM [46] which simulates all the internet services like FTP, HTTP, telnet etc. Using this setup we monitor DNS and other internet queries made by the application. We also create a dedicated thread to capture the packets flowing during application execution. This is implemented using PCAP Library [47]. ADAMIS then analyzes the PCAP dump to search for NETBIOS queries also made by the application [48].

ADAMIS also scan executable against file magic signatures to detect whether it is an installer package [50]. If executable is a

packaged installer then, chances of getting classified as benign application increases. We keep this feature with severity number of 0.3.

If executable is packed using a known packer then chances of getting classified as malware increases and hence we put this feature in rule categories with a severity of 5. We make use of PEiD signatures for the purpose [49].

We use n-gram approach for determining an abnormal URL proposed by Mike Geide [51]. The author by applying 3–grams and 4–grams, identifies words which are more common in malicious domains like free, evil, virus, anti, best, adul, music, funpic etc. Some readers can argue to assign a higher severity number to behavior 'AbnormalURL'.

The gap between two applications was not clearly visible with the euclidean distance. We observed that no difference is created between two applications even if one application does a single non acceptable file system activity and the other application which does 5 such activities. So we decided to put a power which is dynamically calculated count and indicator of their frequency of occurrence. For other behaviors, we keep the power 2 same as in euclidean distance.

Power distance is same as euclidean distance but the power is not 2 and remains variable. Following is the formula:

$$PD = \sqrt{\sum_{i=1}^{K=14} (S_i)^W}$$

Where K is total number of behaviors depicted by the application and $S_i$ is a severity number of a particular behavior. If power distance is below 2, the application is classified as safe. Values more than 4 are considered malicious. These threshold values are determined by

analyzing many applications using tool and calculating PD.

**Dynamic Analysis of PDF files** For analyzing a given PDF files, we invoke the Virtual machine which is pre-installed with Adobe Reader. A malicious PDF is supposed to first create a binary on system either by downloading or extracting from itself and then executing the new binary. So all the behaviors discussed above are monitored here as well. But here three behaviors of Reader program are considered important viz.

(i) If reader program write a executable file on storage,

(ii) If it sends a web request to download an executable file type.

(iii) If it creates another PDF file.

Both of these behaviors are given a severity of 5 so that a power distance automatically comes to a range which is malicious even if they only occur. PDF exploits may affect either single or multiple versions of reader program. For handling such cases we need to maintain different versions of reader program either installed in a single virtual machine or maintain multiple virtual machines and run them all. Static analysis based detection of PDF file produced better results as 695 out of 700 malicious PDF files were detected. Dynamic analysis based detection could detect 650 out of 700 files. This was due to requirement that malicious PDF files require a specific version of PDF reader to execute the true behavior. We used Adobe reader 8.1 and 9.0 while some PDF files required Adobe Reader 7.x, 8.0 and Foxit PDF Reader as well.

## 4.6 Implementation

The major contribution of this component is to collect and analyze the behavior patterns of a given executable and giving a severity score to determine whether it is malicious. Though there are other tools available like Cuckoo, ZeroWine and Capture-BAT, they were inadequate for collecting the proposed behavior trace, analysis and easy deployment at gateway. ADAMIS uses file system mini filter driver to capture file system activity [52] and registry callbacks driver to monitor registry operations [53]. ADAMIS also registers a callback function using Device Driver Interface (DDI) PsSetCreateProcessNotifyRoutine which notifies in case when a process starts and process exits [54].

We use Microsoft detour library [56], [55] to hook Win32 APIs. Detour library uses a unique trampoline design to intercept Win32 functions. Table 4.3 shows Win32 APIs hooked using detour library and their importance.

Table 4.3: Win32 APIs intercepted using Detour Library

| Win32 API Hooked | Behaviour Monitored |
|---|---|
| EnumerateProcesses | Application enumerates all running processes in system |
| WriteProcessMemory | Application writes to other process's memory |
| ReadProcessMemory | Application reads other process's memory |

Components of ADAMIS which capture different behaviors are depicted in Figure 4.2.

ADAMIS controller module reads the input program file path from supplied command line argument and executes the program as a child of browser program to evade the bypass technique of the malware. If

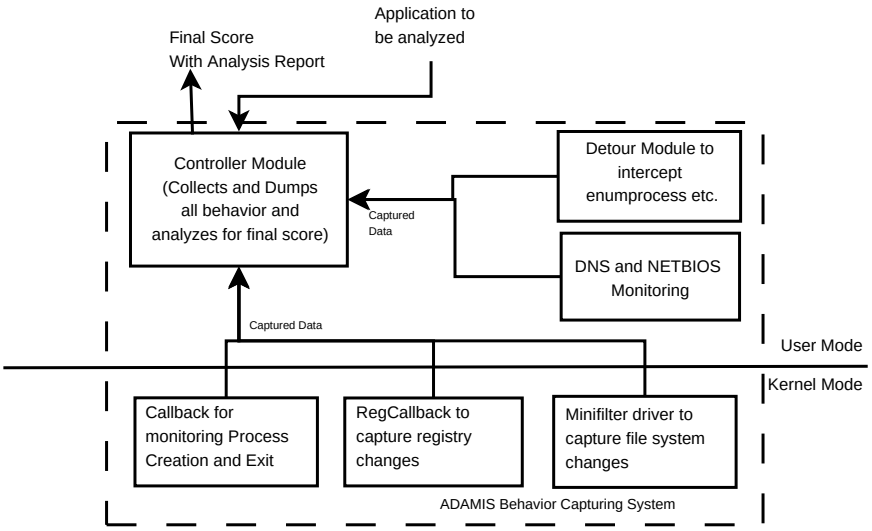



Figure 4.2: ADAMIS Behavior Capture System

malware process is started is suspended mode, malware detect that they are being executed in an analysis environment. If supplied file is a PDF, then it starts the pre–configured PDF reader program. Process Id is identified and passed to registry callback driver (which monitor registry operations) and mini-filter driver (which monitor file system operation) via communication ports. Controller Module starts all the monitoring modules, resumes the process and then starts collecting the data from all monitoring modules. After all malware processes exit controller module does the analysis of collected behavior patterns. If they don't at their own controller program terminates all after 3 minutes.

## 4.7 Evaluation

We evaluated ADAMIS on collected malware data set. Results showed that tool is capable of finding out if an application is malware or benign in an efficient way.

True Positive Rate (TPR) is calculated as

$$True\ Positive\ Rate = \frac{Number\ of\ Malware\ Samples\ Detected\ as\ Malware}{Total\ Number\ of\ Malware\ Sample}$$

False Positive Rate (FPR) is calculated as

$$False\ Positive\ Rate = \frac{Number\ of\ Benign\ Samples\ Detected\ as\ Malware}{Total\ Number\ of\ Benign\ Sample}$$

The true positive rate of this tool on malware applications is 98.46% while false positive rate is 2%. This is shown in table 4.4.

We compare the power distance scores of malicious and benign applications in figure 4.3. Malicious applications score higher as compared to benign applications.

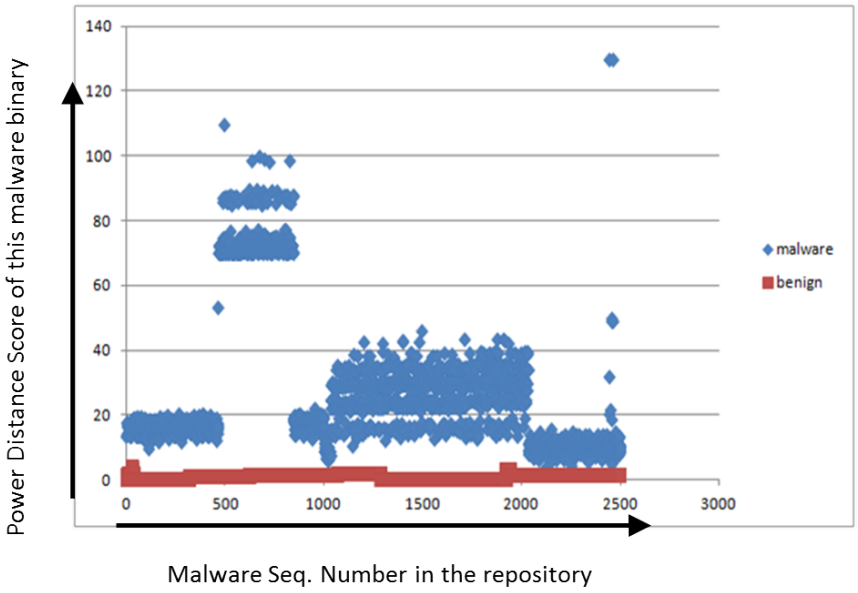



Figure 4.3: Comparison of PD Score of malicious and benign applications

Table 4.4: True Positive and False Positive Results

| | Number | Identified Correctly | Identified Wrong |
|---|---|---|---|
| Malware Applications | 2600 | 2560 | 40 |
| Benign Applications | 2600 | 2548 | 52 |

We actually first calculated the Euclidean distance (as mentioned earlier) and also compared it with results of power distance, figure 4.4.

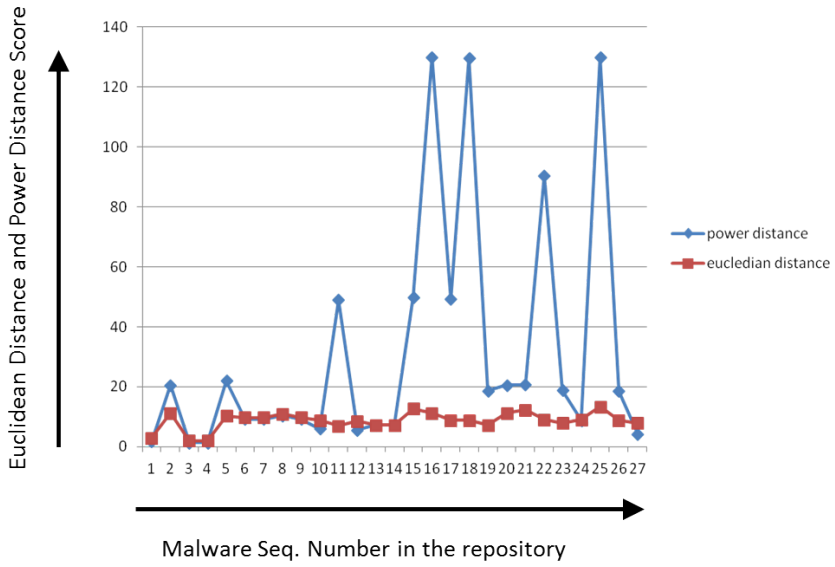



Figure 4.4: Comparing results from euclidean and power distance

Power distance shows observable variation and helps in classifying benign and malicious binaries.

We tested on virus, fake AV, Trojans, bots, keyloggers and their score was never less than 4. Table 4.5 lists scores of a few malware from our dataset. Malware PWS: Win32Coced.2_25 is classified as malicious even though it only generate software cracks. Sysinternals' DbgView application scored 1.732 and Free Download Manager Lite version setup scored 1.41. Malware analysis report for Win32.IRCBot.nw is shown in Appendix C.

Table 4.5: Power Distance calculated on sample malware

| Malware Name | MD5 Hash | PD Score |
|---|---|---|
| Kelihos.F | 06212688b30e84f461b11f3447db0288 | 6.08 |
| Fesber.F | 09345046b32116776cd2ca7a3258e6bd | 18.68 |
| Kelihos.F | 0a3cbb2d62569da6e66918026e960320 | 5.83 |
| Tibs.N | 0a99c3c553857f37779188960a433dfa | 49.06 |
| Lmir.YS | 0aec2295c9d4041ff448d9234630e438 | 5.56 |
| PolyCrypt | 2cbd501e9ee3d2642563c8f678ba85f6 | 50.53 |
| DelfInject | 20270d9efbd030c513f92e319b857dc0 | 343 |
| Spybot | 003b5e16b0055cd7c7f96a3bf87c94ac | 158.12 |
| Fynloski.A | 1a9954b4cb98693c763b8dda6bf9ca9c | 129.90 |
| Delfsnif | 0d36488b44d139f8be8566863757326e | 49.34 |
| Zegost.AD | 0858278e6edbed50e111d744740c568d | 4.12 |
| Centim.gen | 03398554de5502383c0f1da1793e3af0 | 18.62 |
| Sality.AM | 0ac8bc130a84fb9aa33aeef7d1cf4dd8 | 7 |
| Ramnit.X | 0e7d448c4dfe52e807bb28b3377632fd | 10.72 |
| Autorun.UE | 01d090d4e61ec3dfda77fe81f1e38917 | 129.65 |
| Kilfno.C | 1c7c7ef0a2d5d7e069a5fdc8dbbb347f | 18.62 |
| Artemis | 1c502992e64870fe16472dfe4ec7c8b6 | 24.63 |
| Rbot | 03da32043abfbe9f454ce0d65090e5b7 | 129.95 |
| Autorun.gen | 04d5562e93ab212a8780b82e96b5f7f0 | 7.87 |
| Fynloski.A | 0652666a1e6f71786fe3a0ac3280a593 | 49.04 |
| Hostil.gen!A | 18682b3841caa137dc71c9d534e9c315 | 129.64 |
| Virut.AE | 1753c6bc0fd253d4a0c44da90e7fcca6 | 8.66 |
| Allaple.A | 02951b39b4b936a89973e2fa1849c9f8 | 3.16 |
| Kelihos.F | 1074cb7ea87df7349fadc0a2ae496157 | 5.83 |
| Emerleox.K | 0116740c0dd21930e2de9e87432dae72 | 7.81 |
| Small.16.AJ | 0825b9b20271f87490aaaa8f92bfd686 | 7.28 |

# Chapter 5

# Detecting the Malicious Files By Static Analysis

## 5.1 Malicious PDF Detection using Static Analysis

Malicious documents attacks are preferred method for attackers to invade into a computer system. The users can easily be lured with carefully crafted emails with titles like important information about your credit card, bank account, offer letter, new flat details etc. Malware attackers keep discovering vulnerabilities and attack computer users for stealing secret data, financial credentials etc [32]. Attackers by sending malicious documents can invade the security of a computer user.

In this research component, we designed and developed a static analysis based approach for detection of malicious PDF documents. We analyzed a corpus of malicious documents and evaluated the use of entropy, n–gram and feature selection based methods to detect malicious PDF documents.

Robert Lyda explained that high degree of randomness in bytes of

an executable indicates towards encryption and obfuscation and hence used entropy to classify packed executable [20]. Brandon Dixon[1] on his blog maintains corpus of malicious PDF files and their entropy values.

In this work, we assumed that degree of randomness in an exploit file should be less than a regular benign file. The work in this chapter uses material from reference [69], reference [70], both authored by Himanshu Pareek, Mrs. P R L Eswari and Dr. N. Sarat Chandra Babu.

### 5.1.1 Portable Document Format

Portable Document format, commonly know as PDF is certainly the most preferred file format for a range of applications. In this section, we understand its file format and see malware authors use the weaknesses in this format to do reliable malware delivery.

A developer if asked to design or parse a document format, it is not uncommon to start it from beginning and unearthing relevant information by reading further bytes. But this does not happen with PDF. First 8 bytes are header in form of a string %PDPF-x.y. After reading these 8 bytes a parser has to reach trailer where further information about parsing is present. Figure 5.1 shows a PDF document in text form. We have used this minimal pdf file.[2]

PDF file is composed of a header, trailer, cross reference table and objects. Header can be parsed to confirm if the file opened is a PDF file and its version can also be read. After this parsing should be done from bottom of the file. First "%%EOF" should be located and going backwards, the keyword 'startxref' should be found out. The 'startxref' gives the address of the cross reference table which contains the list of the objects in the document and their addresses in the file. Cross reference table starts with keyword xref and shows the first object in

[1] blog.9bplus.com

[2] http://brendanzagaeski.appspot.com/minimal.pdf

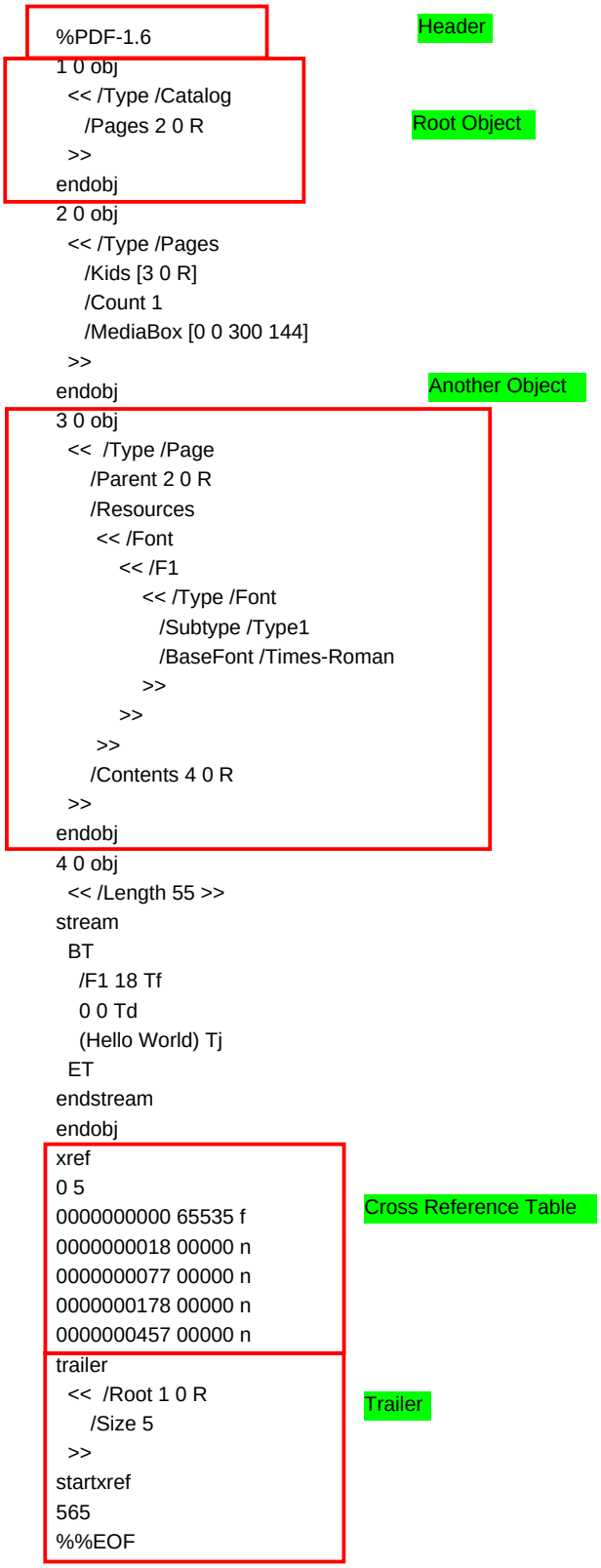



Figure 5.1: Minimal PDF file to understand the format

table and number of objects in the table. It also shows if object is in use or free.

Every piece of information is embedded in the objects scattered all over the PDF file. But what makes its logical view. Logical view is also constructed from the trailer. Trailer has one line "Root 1 0 R" which means that the object number 1 is the root of the document or this is the first object which has to be processed. Object 1 is a catalog type object and links to object 2 which is Page type object. Object 2 in turn links to Object 3 which in turn links to object 4 and also indicates that object 4 contains data via '/Contents'. Object contains keyword 'stream' and followed by data.

### 5.1.2 Datasets

We utilized malicious PDF files from malware hosting websites[3] for analysis and learning purposes. We also got PDF malware set from Brandon Dixon to which we used as test set.

### 5.1.3 Experiment: Entropy Analysis of PDF Documents

Entropy is a method of measuring randomness or uncertainty in a given set of data. For calculating the entropy of file, our data set is sequence of bytes in the file [20]. Entropy can be calculated by using following formula:s

$$H(x) = -\sum_{i=1}^{N} p(i) \log_2 p(i)$$

Where p(i) is the probability of $i^{th}$ unit of information and specifically in our case it is $i^{th}$ byte of file. The value of H(x) will be high

[3] http://contagiodump.blogspot.com

if probabilities of occurrence of bytes are low i.e. can not be predicted easily and will be low if probability of occurrence of bytes are high or is predictable. For example take three words: 'thesis', 'abcd' and 'qytlmjg'. If see their randomness using pattern of occurrence of the characters, entropy (or randomness) is low in first word. Entropy of second word will be high and of third word will be even higher. Encrypted Files can be classified using such entropy analysis with understanding that probabilities of occurrence of bytes will be low. In our experiment, we assumed that malicious documents might be carrying false information and padding and hence entropy value should be low. In fact to identify whether a source file belongs to C or Java, entropy of file can be calculated by using word as the unit.

We calculated the entropy of files based on bytes. Our thought process was that if a PDF is exploit it is containing lot of dummy strings. We made an assumption that entropy of such exploit documents should be quite lower as compared to a benign document containing meaningful text. Appendix A contains the code for calculating entropy of file which we used in our experiments. We automated this by enumerating all the files in the directory and calculating entropy of all the files using single program execution. Following results were obtained.

The entropy values cannot be a direct indication towards a benign or malicous file but if used with other features it can be useful. Based on our experiments, we arrive the confidence interval of entropy values where we can conclude if a PDF document is suspicious, table 5.1.

Table 5.1: Results of Entropy Calculation

| Dataset | Average | Minimum | Maximum | CI |
|---|---|---|---|---|
| Benign | 7.70 | 2.95 | 8.0 | NA |
| Malicious | 4.80 | 0.99 | 8.0 | 0.0–2.7 |

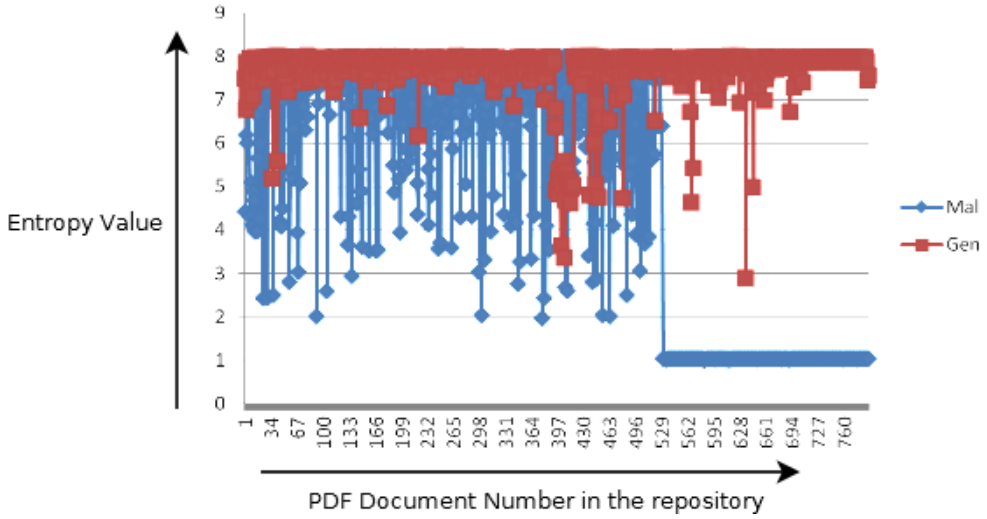



Figure 5.2: Entropy Analysis of PDF Documents

Minimum entropy calculated on a corpus of 10000 good PDF documents came out to be 2.95 while minimum entropy calculated on a same number of bad PDF documents came out to be 0.99. We used this difference as if entropy of a input PDF document is too low that is 0 to 2.7 it can be suspected. We use this learning in next section.

### 5.1.4 Experiment: n-gram Analysis of PDF Documents

N–grams are sub–strings of a large string of length N. They are understood as shifting windows over a large string. Suppose there is a string WORLD, 1-grams of this string will be W, O, R, L and D. If we take 2–grams of this string then they will be: WO, OR, RL and LD. Similarly consider the following sentence: I am writing thesis. Now if word is taken as unit, then 2–grams will be "I am", "am writing" and "writing thesis". N–gram are capable of giving syntactic and semantic view of a text.

We first produce hex dumps of PDF files. From hex dumps, we then generate 2–grams from these files with word as a unit. To analyze the n–gram output, term frequency and TFIDF can be used. Both the terms are defined below.

TF, Term frequency is the numerical measurement of how often a term occurs in respect to other terms in the document. Term frequency of a particular term can be obtained by dividing its frequency by the frequency of maximum occurring term in the document. This can be obtained by following formula.

$$tf = \frac{f(t,d)}{\max\{f(w,d) : w \in d\}}$$

TFIDF, term frequency inverse document frequency is a numerical measurement which reflects how important a word is to a document in a collection. This is obtained by multiplying TF by IDF.

$$tfidf(t,d,D) = tf \times idf$$

Where IDF is inverse document frequency

$$idf = \log_2 \frac{N}{DF}$$

Where N is the number of documents and DF is number of files in which the term appears.

For the particular task, we collected 792 malicious documents 792 benign documents. With this we did the experiment multiple times but program to generate hexdumps and n–grams used to continue for a long time and output file size used to grow very large to be handled by analyzer programs. So we kept a threshold term frequency of 0.05 and stopped entertaining terms with term frequency of less than a threshold value. The threshold value was chosen on the basis of failed experiments and is just big enough. A bit increase from here used to disrupt the analysis process. We analyzed this data using standard machine learning algorithms. We combined both the data sets and

final data set contained following set as following:

$$tuple = tf, tfidf, mal0gen$$

Where mal0gen is term which can take two values, either 0 or 1. This is kept 0 for benign documents and 1 for malicious document. We used J48 algorithm which is open source implementation of C4.5 algorithm in WEKA Tool. The obtained results are shown in Table 5.2.

Table 5.2: Results with J48 Decision Tree

| Dataset Type | Identified Correctly | Identified Wrong |
|---|---|---|
| Benign | 46422 | 511 |
| Malicious | 65247 | 289 |

These results indicate that a model built with TF and TFIDF of PDF documents with the help of J48 algorithm is very efficient with 0.01 percent false positives and 0.0044 percent false negative rate but the time it takes to analyze a document was huge and hence this approach was discarded.

### 5.1.5 Malicious PDF Identification using Feature Extraction

While doing malicious PDF analysis, we observed how a malicious PDF document is able to carry out the attack. We analyzed PDF documents in training set for the following features.

- /JS - The number of javascript launched.
- /JavaScript - The number of embedded javascript.
- /OpenAction - The presence of an open action
- /AsciiHexDecode - The use of the asciihexdecode filter.

- /Launch - The use of the /Launch statement.
- /FlateDecode

These feature set was decided on the work done by Paul Baccas [57] to indicate what are prominent features used in malicious PDF documents as compared to benign documents. Importance of these feature sets can be understood from Adobe PDF Specification [58].

OpenAction specifies an action or destination page to be displayed when the document is opened. Malicious PDF use Openaction to execute a javascript as soon as document is opened [64]. 'Launch' specifies to launch an application which is definitely very useful to craft a malicious PDF. '/JS' and '/JavaScript' indicates that a JavaScript is present inside the object. PDF Specification supports many data encoding techniques like 'FlateDecode' to compress data which may help to keep the document size less. In our approach, we pick up such filters and uncompress the data to discover any java script code. If java script is found after deflating the compressed data then we assign a severity of two. We rely on the approach given by Young Han Choi [66] to detect obfuscated java script and if found that is given a severity of 3. We also took in account that a PDF document with the intent of exploiting may contain some garbage strings and hence reduce the document entropy. This is not a reliable method to detect malicious document and we just used this as a parameter with very strict restriction. If entropy of a document goes below than two then flag is raised and a severity of two is assigned. Following feature vector is calculated for every PDF file for further calculation.

After determining number of features contained by a PDF document, we calculate Power Distance Score as per following formula.

Table 5.3: Features Extracted from PDF File

| Feature | Severity Number |
|---|---|
| The number of JavaScript launched. | 1 |
| The number of embedded JavaScript. | 1 |
| The presence of an open action | 1 |
| The use of the Launch statement. | 1 |
| The use of the asciihexdecode filter. | 1 |
| Chars After Last EOF | 1 |
| (/JS OR /JavaScript) AND (/OpenAction OR /Launch) | 3 |
| If Stream Entropy less than 2 | 2 |
| Javascript After deflating | 2 |
| Obfuscated Javascript | 3 |

$$PD = \sqrt{\sum_{i=1}^{K} (S_i)^W}$$

Where K is total number of features contained by the PDF document and $S_i$ is a severity number of a particular feature. W is constant and equals to 2. By doing further research W can be assigned different numbers and a better score can be calulated. For benign PDF documents the score is less than 1 while any document with score more than 1 can be categorized as suspicious. As score reaches around 4 there is a high probability the document is malicious. This method based on Power distance calculation showed high false positive rate. Then, we turned to specialized machine learning algorithms to classify the PDF document based on our feature set.

To identify features from the PDF file, we used PDFiD.py from Didier Stevens [62] and for some features we wrote our own implementation. We used Weka [63] implemenetation of various algorithms.

We utilized malicious PDF files from malware hosting website [4]. We decided to keep 3000 benign files and 3000 malicious files in training

[4] http://contagiodump.blogspot.com

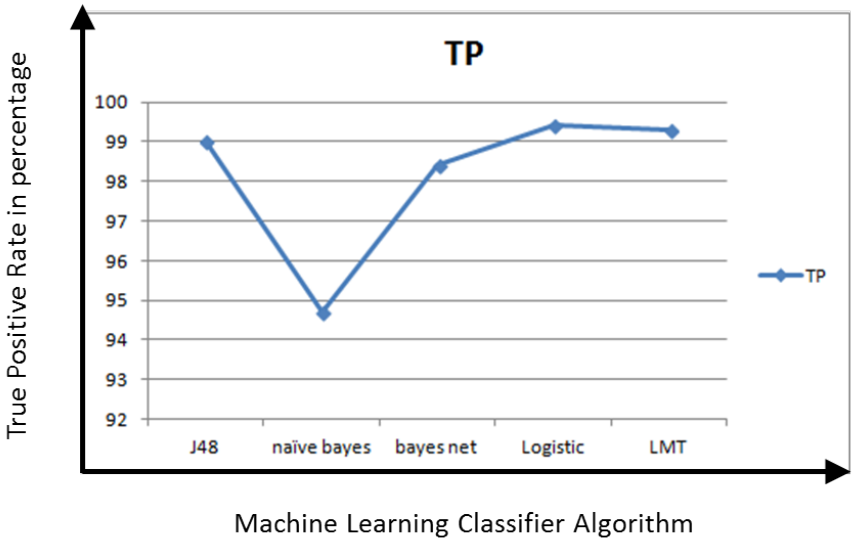



Figure 5.3: Comparison of True Positive Rate with various classifiers

set and keep on collecting new malware from various users[5] and keep them in test set. We kept 700 PDF documents in the test set.

True positive rate of various classifiers applied on test dataset are depicted in Figure 5.3 and false positive rates are depicted in 5.4. Logistic and LMT algorithms show high true positive rates slightly better that J48 (or C4.5). Logistic algorithm showed higher false positive rate as compared to LMT and J48. J48 (or C4.5) was the most accurate algorithm in our case. True positive rate of various classifiers applied on test dataset are depicted in Figure 5.3 and false positive rates are depicted in 5.4. Logistic and LMT algorithms show high true positive rates slightly better that J48 (or C4.5). Logistic algorithm showed higher false positive rate as compared to LMT and J48. J48 (or C4.5) was the most accurate algorithm in our case.

Figure 5.5 shows how time taken to scan a file increases with the size of the file.

[5] www.offensivecomputing.net

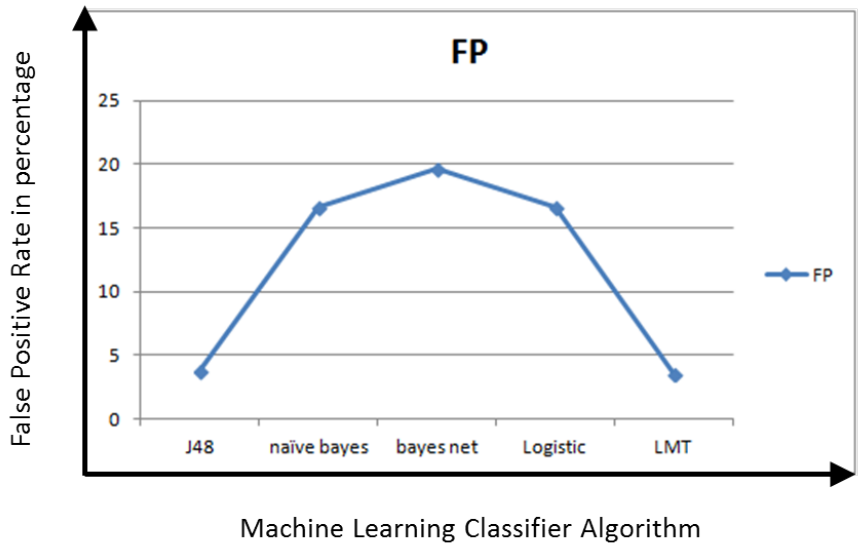



Figure 5.4: Comparision of True Positive Rate with various classifiers

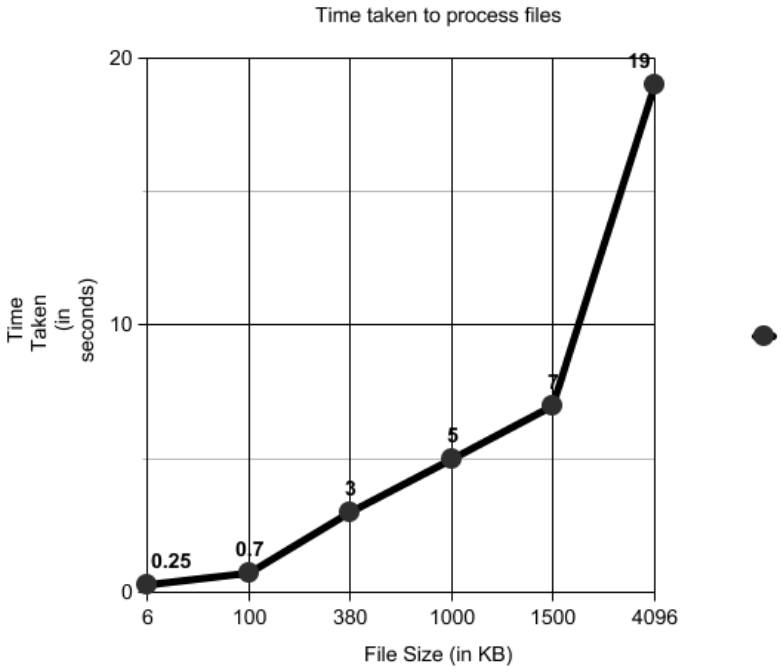



Figure 5.5: Time taken to scan the PDF file

## 5.2 Malicious Portable Executable (PE) Detection using Static Analysis

Portable Executable (standard executable format on Windows OS) has always been the preferred form of malware attacks. Even malicious documents, java-scripts etc. are only the first stage of overall attack strategy of attackers. In next stages a executable file is downloaded and executed directly or injected in another process.

Robert Lyda [20] has proposed a Packed PE Detection technique based on entropy analysis while Yang-seo Choi [21] and Scott Treadwell [24] have proposed approaches based PE header analysis. We implement a malicious detection approach based on anomalies present in malicious PE when compared to benign files. We implement a module to detect malicious PE files based on static analysis. The approach is based on papers of Scott Treadwell [24] and Rohit Arora et.al [65]. Table 5.4 shows the list of characteristics extracted from the file under test and scores given to each characteristic. This detection technique is not the contribution of this research work but was required to be implemented to support the original novel approach presented in chapter 3.

Entropy is calculated using following formula.

$$H(x) = -\sum_{i=1}^{N} p(i) \log_2 p(i)$$

Where p(i) is the probability of $i^{th}$ unit of information and specifically in our case it is $i^{th}$ byte of file. We use the confidence intervals given by Robert Lyda in this paper [20]. Based on this if entropy of a section is higher than 7.3 we flag it as encrypted section and assign a score of 3.

Moreover, every characteristic extracted is quantified and a

Table 5.4: List of Characteristics extracted from PE file

| Characteristic Description | Score |
|---|---|
| Presence of fake entry point | 5 |
| Sections which execute and have write permissions both | 5 |
| No code section present | 5 |
| Very less number of imports | 3 |
| Remarkable high section entropy | 3 |
| Recognized names used by packer | 2 |
| Non printable chars | 2 |
| Sections with no name or unknown names | 2 |
| Sections which have Data and execute permissions both | 5 |
| Sections if mismatch in Execute and contain code flags | 5 |
| Sections which have Data and Code permissions both | 5 |

euclidean distance score is calculated as per following formula.

$$Score = \sqrt{\sum_{i=1}^{K} (S_i)^2}$$

Where S is severity assigned to each characteristic to quantify it.

**How We Assign these severity numbers** This is again based on our learning and experience with analyzing malware and are kept relative. For example, if a section name is not present or unknown or contains non printable characters does not strongly indicate that file is malicious and hence kept at lowest severity values. Similarly for severity number of 'less imports', when we observe executable of even a hello world program it contains many import functions but many malware contains less than even 5. Such programs were assigned a severity level of 3. Similarly, other behaviors in the list are strongest indicators of a malicious file and hence assigned the highest severity number of 5.

**Results** We tested this approach on 2600 malware samples (same data set is used) and following results were obtained with a threshold

value of 4. That is if power distance was more than 4 we classified as malware otherwise benign.

Table 5.5: Results by static analysis of PE files

| | Number | Identified Correctly | Identified Wrong |
|---|---|---|---|
| Malware files | 2600 | 2300 | 300 |
| Benign files | 2600 | 2228 | 372 |

Table 5.5 shows that detection rate of 88.46% was achieved and a false positive rate of 14.3% was recorded. Our dynamic analysis system ADAMIS which is also implemented on similar concepts showed better results than this.

# Chapter 6

# Conclusion and Future work

With this research work, we proposed a new malware prevention technique at gateway which overcomes challenges of file reconstruction and not being able to use the dynamic analysis techniques. Though there are many run time detection and dynamic analysis techniques available, we proposed a mechanism to automatically act on behavior traces to identify whether a binary is malicious. For PDF type of malware, we carried out the static analysis to detect them which can be deployed at gateway level.

The research assumes that dynamic analysis is more accurately predictive. Static analysis can become more accurate but with those improvements as well the challenge remains same of controlling the packets and file at gateway. For this our proposed mechanisms of file tagging will still be useful. Our dynamic analyzer showed 98% detection rate and static analysis based detection of PDF documents showed 99% detection rate by using standard machine learning classifiers. Around 200 out of 2600 PE malware were not detected by static analysis component but detected by dynamic analysis component. We also took 33 latest samples and observed that 3

samples were not detected by static analysis but were detected by dynamic analysis. Also there existed a sample which was detected by static analysis but not dynamic analysis. We can also conclude that it is better to deploy static analysis based detection for files like PDF, DOC etc which require another software to execute than dynamic analysis based detection because of requirement of multiple software version installations. Dynamic analysis based detection of malicious PDF files missed 40 samples which were detected by static analysis. There were few samples which were not detected by static analysis but dynamic analysis. One of such case was presented in chapter 3.

Hence to improve the proposed mechanism, we suggest dynamic analysis using multiple virtualization software or emulators, and operating system versions. Work can be extended to research real time algorithms which can suggest whether submitting binary first to static or dynamic will be more beneficial for network defense. The malware obfuscated using virtualized packers like VMProtect, Themida and Molebox need specific attention.

# References

[1] Malware Statistics, AV-Test, An Independent Security Institute, Germany. Available: www.av-test.org

[2] Nicolas Falliere. 2006, Anatomy of a Malware. Available: www.packetstormsecurity.net

[3] Liam Tung, September 2007, Bank Of India Attack, Available: www.zdnet.com/article/infamous-russian-isp-behind-bank-of-india-hack

[4] You I., Yim K. *Malware obfuscation techniques: A brief survey*. In Proceedings of the 2010 International Conference on Broadband, Wireless Computing, Communication and Applications (Washington, DC, USA, 2010), IEEE Computer Society, pp. 297–300.

[5] Microsoft Inc., 2007, Understanding Anti-Malware Technologies, Available: http://download.microsoft.com/download/a/b/e/abefdf1c-96bd-40d6-a138-e320b6b25bd3/understandingantimalwaretechnologies.pdf

[6] Nick Del Grosso. It's time to rethink your corporate malware strategy. SANS, 2002.

[7] Cisco Inc. Cisco Security Report 2008, Available: www.cisco.com/go/securityreport

[8] McAfee Inc. McAfee Security Report 2010, Available: www.mcafee.com/in/resources/reports

[9] Microsoft Inc. Microsoft Security Report 2010, Available: www.microsoft.com/security/sir

[10] Sonicwall Inc. *Sonicwall Gateway Antivirus Overview, Gateway Antivirus, Anti-Spyware, Intrusion Prevention, and Application Intelligence and Control Service*, 2011.

[11] McAfee Inc. *McAfee Gateway Anti Malware Engine*. 2012.

[12] Martin Stecher. *Stopping malware at the gateway âĂŞ challenges and solutions*. Virus Bulletin Conference September 2007

[13] Nwokedi Idika, Aditya Kapoor. Survey of Malware Detection Techniques. Technical Report, Purdue University, 2007.

[14] John V. Harrison. Enhancing Network Security, By Preventing User-Initiated Malware Execution. ITCC, 2005

[15] T. Holz, F. Freiling and C. Willems. Toward Automated Dynamic Malware Analysis Using CWSandbox. Proc. IEEE Symp. Security and Privacy 2007, pp. 32-39.

[16] U. Bayer, A. Moser, C. Krgel and E. Kirda. Dynamic Analysis of Malicious Code. J. Computer Virology, vol. 2, no. 1, pp. 67-77, 2006.

[17] Manuel Egele, Theodoor Scholte, Engin Kirda, Chritstopher Kruegel. A Survey on Automated Dynamic Malware Analysis Techniques and Tools. ACM Computing Surveys, Vol. 44, No. 4, 2012.

[18] Bradley J. N abholz. Design of an automated malware analysis system. Technical Report, Purdue University, 2010

[19] Yongtao Hu, Liang Chen, Ming Xu, Ning Zheng, Yanhua Guo. Unknown Malicious Executable Detection Based on Run-time Behavior. International Conference on Fuzzy Systems and Knowledge Discovery, 2008

[20] Robert Lyda, J Hamrock, Using Entropy Analysis to find Encrypted and Packed Malware. IEEE Security and Privacy, 2007

[21] Choi, Yang-seo, Ik-kyun Kim, Jin-tae Oh, and Jae-cheol Ryou. "Encoded Executable File Detection Technique via Executable File Header Analysis." International Journal of Hybrid Information Technology Vol.2, no. 2 (2009): 25-35.

[22] Asaf Shabtai, Robert Moskovitch1, Clint Feher, Shlomi Dolev and Yuval Elovici1. Detecting unknown malicious code by applying classification techniques on OpCode patterns. Security Informatics, Springer Open Access Journal, 2012.

[23] Igor Santos, Felix Brezo, Javier Nieves, Yoseba K. Penya, Borja Sanz, Carlos Laorden, and Pablo G. Bringas . Idea: Opcode sequence based Malware Detection. ESSoS 2010

[24] Treadwell, Scott, and Mian Zhou. A heuristic approach for detection of obfuscated malware. IEEE International Conference on Intelligence and Security Informatics, 2009.

[25] Charles Smutz, Angelos Stavrou. Ruminate: A Scalable Architecture for Deep Network Analysis. Technical Report. George Mason University, 2010.

[26] Vadrevu Phani, Babak Rahbarinia, Roberto Perdisci, Kang Li, and Manos Antonakakis. "Measuring and detecting malware downloads in live network traffic." In Computer

SecurityâĂŞESORICS. pp. 556-573. Springer Berlin Heidelberg, 2013.

[27] Laskov, Pavel, and Nedim Åărndiǎ. Static detection of malicious JavaScript-bearing PDF documents. Proceedings of the 27th Annual Computer Security Applications Conference. ACM, 2011.

[28] Claudio Guarnieri. cuckoo sandbox, Available: www.cuckoosandbox.org.

[29] Emmanuel Hooper. An Efficient and Intelligent Intrusion Detection and Response System using Virtual Private Networks. International Journal of Security and Its Applications, Vol. 5 No. 2, July 2007.

[30] Te-Shun Chou. Cyber Security Threats Detection Using Ensemble Architecture. East Carolina University, USA. International Journal of Security and Its Applications Vol. 5 No. 2, April, 2011.

[31] Alex Rousskov, RFC 4037: Open Pluggable Edge Services (OPES) Callout Protocol (OCP) Core. IETF, March (2005).

[32] QuickHeal Inc. January 2013, Quick Heal Technologies Report, Available: www.quickheal.com

[33] Lenny Zeltser. October 2010, Three phases of malware analysis, Available: http://computer-forensics.sans.org

[34] Lenny Zeltser. October 2010, Free Toolkits for Automating Malware Analysis, Available: http://blog.zeltser.com.

[35] Lenny Zeltser. August 2010, Free Automated Malware Analysis Services, Available: http://zeltser.com.

[36] Joxean Koret. ZeroWine Malware Analysis tool, Available: http://zerowine.sourceforge.net.

[37] Arne Vidstrom, Evading the Norman Sandbox Analyzer, Available: http://ntsecurity.nu

[38] Mandiant, Red Curtain. Available: www.mandiant.com

[39] K. Rieck, P. Trinius, C. Willems, and T. Holz. Automatic Analysis of Malware Behavior using Machine Learning. Journal of Computer Security 2011.

[40] Seifert C., Steenson R., Welch I., Komisarczuk P., Endicott-Popovsky B., Capture - A behavioral analysis tool for applications and documents. digital investigation 4 (2007): 23-30.

[41] Wu Naiqi, Yanming Qian, and Guiqing Chen. "A novel approach to Trojan horse detection by process tracing." In Proceedings of the IEEE International Conference on Networking, Sensing and Control, 2006, pp. 721-726.

[42] Statsoft Inc., Power Distance, Available: www.statsoft.com/textbook/cluster-analysis

[43] Dionaea, Malware Collection Tool. Available: dionaea.carnivore.it

[44] VMWare Inc., VIX API Documentation. Available: www.vmware.com/support/developer/vix-api/

[45] Lenny Zeltser, FakeDNS, Available: https://zeltser.com/fake-dns-tools-for-malware-analysis

[46] INETSIM, Internet Services Simulation Suite. Available: www.inetsim.org

[47] Riverbed Technology, PCAP Library, Available: www.winpcap.org

[48] Himanshu Pareek. Scanning a PCAP dump to find DNS and NETBIOS queries, Available: www.codeproject.com/Tips/465850/Scanning-a-PCAP-dump-to-find-DNS-and-NETBIOS-queri

[49] Panda Security Inc. Packer Signatures. Available: http://research.pandasecurity.com

[50] TrID, File Identifier, Available: http://mark0.net/soft-trid-e.html

[51] Mike Geide, Zscaler Inc. N-gram Character Sequence Analysis of Benign against Malicious Domains and URLs. Available: http://analysis-manifold.com/ngram_whitepaper.pdf

[52] Microsoft Inc. File System Mini filter driver. Available: http://msdn.microsoft.com/enus/library/windows/hardware/ff540402\%28v=vs.85\%29.aspx

[53] Microsoft Inc, Filtering Registry Calls. Available: http://msdn.microsoft.com/enus/library/windows/hardware/ff545879\%28v=vs.85\%29.aspx

[54] Microsoft Inc., PsSetCreateProcessNotifyRoutine Routine, Available: http://msdn.microsoft.com/enus/library/windows/hardware/ff559951\%28v=vs.85\%29.aspx

[55] Microsoft Inc., Detour Library. Software Package for re-routing Win32 APIs underneath applications, Available: http://research.microsoft.com/en-us/projects/detours

[56] Galen Hunt and Doug Brubacher, Detours: Binary Interception of Win32 Functions, In Third USENIX Windows NT Symposium, USENIX, July 1999

[57] Paul Baccas, Finding Rules For Heuristic Detection Of Malicious PDFs: With Analysis Of Embedded Exploit Code, Virus Bulletin Conference, September 2010

[58] Adobe Systems Inc., Adobe Portable Document Format Version 1.7, Available: http://wwwimages.adobe.com/content/dam/Adobe/en/devnet/pdf/pdfs/pdf_reference_1-7.pdf

[59] Blonce Alexandre, Eric Filiol, and Laurent Frayssignes. "Portable document format (pdf) security analysis and malware threats." In Presentations of Europe BlackHat 2008 Conference. 2008.

[60] Li Wei-Jen, Salvatore Stolfo, Angelos Stavrou, Elli Androulaki, and Angelos D. Keromytis. "A study of malcode-bearing documents." In Detection of Intrusions and Malware, and Vulnerability Assessment, pp. 231-250. Springer Berlin Heidelberg, 2007.

[61] Tzermias Zacharias, Giorgos Sykiotakis, Michalis Polychronakis, and Evangelos P. Markatos. "Combining static and dynamic analysis for the detection of malicious documents." In Proceedings of the Fourth European Workshop on System Security, p. 4. ACM, 2011.

[62] Didier Stevens, PDFiD Tool. Available: http://blog.didierstevens.com/category/pdf

[63] Hall Mark, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA data mining software: an update. ACM SIGKDD explorations newsletter 11, no. 1 (2009): 10-18.

[64] Tim Xia, Exploit Action with PDF OpenAction, Available: http://securitylabs.websense.com/content/Blogs/3202.aspx

[65] Arora Rohit, Anishka Singh, Himanshu Pareek, and Usha Rani Edara. "A Heuristics-based Static Analysis Approach for Detecting Packed PE Binaries." International Journal of Security and Its Applications Vol. 7, no. 5 (2013): 257-268.

[66] YoungHan Choi, TaeGhyoon Kim, SeokJin Choi. Automatic Detection for JavaScript Obfuscation Attacks in Web Pages through String Pattern Analysis. The Attached Institute of ETRI, Korea International Journal of Security and Its Applications Vol. 5 No. 2, April, 2010

[67] Pareek Himanshu, and N. Sarat Chandra Babu. Complementing static and dynamic analysis approaches for better network defense. In Dependable Systems and Networks (DSN), 2013 43rd Annual IEEE/IFIP International Conference on, pp. 1-2. IEEE, 2013.

[68] Pareek Himanshu, P. R. L. Eswari, and N. Sarat Chandra Babu. APPBACS: An Application Behavior Analysis and Classification System. International Journal of Computer Science and Information Technology 5, no. 2 (2013). pp.53-61.

[69] Pareek Himanshu, P. Eswari, N. Sarat Chandra Babu. Entropy and n-gram Analysis of Malicious PDF Documents. In International Journal of Engineering Research and Technology, vol. 2, no. 2 (February-2013). ESRSA Publications, 2013.

[70] Pareek Himanshu. Malicious PDF Document Detection Based On Feature Extraction And Entropy. International Journal of Security, Privacy and Trust Management ( IJSPTM) Vol 2, No 5, October 2013. pp:31-35

# Appendix A

## Calculating Entropy

```
// structure for storing file info-
// along with its entropy
typedef struct _Entropy_{
FileInfo FI;
double entropy;
double entropy_2;
}EntropyFI, *LPEntropyFI;
.
.

unsigned int CalculateEntropy
(LPEntropyFI pFileForCalcEntropy)
{
unsigned int iCnt = 0;
unsigned int jCnt = 0;
unsigned int iValue = 0;
unsigned int retVal = SUCCESS;
unsigned char* FilePtr;
double p, lp, sum=0;
unsigned int counts[256] = {};
```

```
FilePtr =
(unsigned char *)pFileForCalcEntropy->FI.lpFileBase;

while(iCnt < pFileForCalcEntropy->FI.FileSize)
{
iValue = *FilePtr++;
counts[iValue]++;
++iCnt;
}

for(jCnt = 0; jCnt < 256; jCnt++) {
if (counts[jCnt] == 0)
continue;

p = 1.0*counts[jCnt]/iCnt;
lp = log(p)/log(2.0);
sum -= p*lp;
} // for

pFileForCalcEntropy->entropy = sum;
return retVal;
}
```

# Appendix B

## Reconstructing PDF File

```
void ALDataParser::ParseData(u_char *pkt_data)
{
unsigned int i, j, http_hdr_len;
unsigned int app_data_len, expected_len = 0;
const char *length = "Content-Length: ";
char *pch;
static unsigned int content_len = 0;
Parse(pkt_data); // parse the packet
switch (PacketLevel)  {
case HTTPPACKET:
if(datalen) {
if(!session_started) {
aDataFound=aTree.SearchAhoCorasik((char *)app_data);
if(aDataFound.sDataFound != ""){
cout<<PacketLevel<<".."<<iphdrlen<<"..";
cout<<total_len<<".."<<tcphdrlen<<"..";
cout<<datalen<<endl;
if(aDataFound.sDataFound ==
"Content-Type: application/pdf"){
session_started = 1;
```

```
cout<<"seqnum:"<<ntohl(th->seq_num)<<endl;
nexSeqNum = ntohl(th->seq_num) + datalen;
pch = strstr((char *)app_data, length);
pch = pch + strlen(length);
content_len = atoi(pch);
fprintf(stdout, "\n-->>%d<<--\n", content_len);
}
while(i<datalen)
fprintf(stdout, "%c", app_data[i++]);
}  }
else if(session_started){
if(ntohl(th->seq_num) == nexSeqNum){
FILE *fp;
fp = fopen(gTempFileNameString, "ab");
fwrite(app_data, 1, datalen, fp);
fclose(fp);
nexSeqNum = ntohl(th->seq_num) + datalen;
content_len = content_len - datalen;
if(content_len <= 0) {
session_started = 0;
}
else {
fprintf(stdout, "len=%d\n", content_len);
}  }  }  } // data is present - if condition ends
break;
default:
break;
} // switch ends
} // Function Parse Data Ends
```

# Appendix C

# Sample Analysis Report

```
Analysis for C:\Samples\virusign(487).vir
( 01728028b1894963c527ccbf70d041ea )
--> new files created
C:\WINDOWS\system\smss.exe
C:\WINDOWS\system32\nvsvcd.exe
--> opened files
\Device\HarddiskVolume1\WINDOWS\system32\psapi.dll
\Device\HarddiskVolume1\WINDOWS\system32\dnsapi.dll
\Device\HarddiskVolume1\WINDOWS\system32\ws2_32.dll
\Device\HarddiskVolume1\WINDOWS\system32\ws2help.dll
\Device\HarddiskVolume1\WINDOWS\system32\wininet.dll
\Device\HarddiskVolume1\Samples\virusign(487).vir
\Device\HarddiskVolume1\WINDOWS\system32\nvsvcd.exe
\Device\HarddiskVolume1\WINDOWS\system32\apphelp.dll
\Device\HarddiskVolume1\WINDOWS\AppPatch\sysmain.sdb
\Device\HarddiskVolume1\WINDOWS\system32\shell32.dll
\Device\HarddiskVolume1\WINDOWS\system32\comctl32.dll
... many more files
\Device\HarddiskVolume1\AUTOEXEC.BAT
--> these files are read
 Device\HarddiskVolume1\AUTOEXEC.BAT
--> these files are written into
\Device\HarddiskVolume1\WINDOWS\system\smss.exe
\Device\HarddiskVolume1\WINDOWS\system32\nvsvcd.exe
--> these files' timestamps changed
\Device\HarddiskVolume1\WINDOWS\system\smss.exe
--> these registry keys are created
\REGISTRY\MACHINE\SYSTEM\ControlSet001\Services\Tcpip\Parameters
\REGISTRY\MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
\REGISTRY\USER\.\Software\Microsoft\Windows\CurrentVersion\Explorer\User Shell Folders
\REGISTRY\MACHINE\SOFTWARE\Microsoft\Tracing
\REGISTRY\USER\..\Software\Microsoft\Windows NT\CurrentVersion\Winlogon
\REGISTRY\MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders
\REGISTRY\MACHINE\SYSTEM\ControlSet001\Services\SharedAccess\Epoch
--> these registry keys values set
\REGISTRY\MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\.nvsvc
... Many oter registry keys
\Software\Microsoft\Windows\CurrentVersion\Internet Settings\MigrateProxy
\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ProxyEnable
\Software\Microsoft\windows\CurrentVersion\Internet Settings\ProxyEnable
-->these processes are created
creates a process (1276): (\Device\HarddiskVolume1\WINDOWS\system32\nvsvcd.exe) with options
(C:\WINDOWS\system32\nvsvcd.exe
-install0&?\REGISTRY\MACHINE\SYSTEM\ControlSet001\Control\SafeBoot\Option)
-->These are the connections made by the process
process 1428 found in listen state on TCP: 0.0.0.0:1881
Automated Analysis
1. Adds a program which will run automatically on next login
2. Changes the internet settings
3. Creates to 2 executables
4. Program listens on a non standard port
5. Adding itself to safeboot
the final score = 49.3153
```

Figure C.1: Sample Report from ADAMIS, Win32.IRCBot.nw